UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIA EXATAS E DE TECNOLOGIA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

RELATÓRIO DE PROJETO FINAL

MICROCONTROLADORES E APLICAÇÕES

Hygor Leonardo Baffero Marques 386073
Igor Guilherme Bianchi 558400
João Pasqualini Costa 558338

Professor: Emerson Carlos Pedrino

São Carlos
12/2017

Sumário

1. Introdução

O trabalho tem como objetivo elucidar dúvidas surgidas durante o andamento da disciplina de Microcontroladores e Aplicações, além de ampliar o conhecimento que não fora possível ser absorvido durante as aulas teóricas. O microcontrolador que será utilizado em todas as implementações deste relatório é o AT89C51, da família 8051 da Atmel. O clock das simulações é fixado em 12 MHz.

As ferramentas utilizadas para os projetos foram a software Proteus para simulação dos códigos e dos circuitos eletrônicos, também o software Proteus para compilar o código Assembly da **seção 2** e o software Keil C para compilar os códigos em C e gerar o arquivo .HEX das **seções 3, 4 e 5.**

A primeira parte deste relatório, **seção 2,** irá desenvolver um controle simples de motor de passo em linguagem Assembly e simulado na plataforma Proteus.

A partir da segunda parte, **seção 3,** serão desenvolvidas ferramentas para o controle PID implementado na **seção 5** deste relatório. Serão apresentadas suas simulações na ferramenta Proteus e códigos em linguagem C.

Por fim, será apresentada a conclusão do grupo em relação ao projeto desenvolvido e bibliografia utilizada.

2. Implementação do motor de passo

O motor de passo é muito utilizado em situações em que a precisão de faz necessária, como em aplicações robóticas e impressora. Seu funcionamento se dá por campos eletromagnéticos que são ativados e desativados eletronicamente [1]. Este controle eletrônico é o que será trabalhado nesta seção.

Para que seja possível o controle do motor, foi utilizado o circuito integrado ULN2003A, **Figura 1.** Este CI contém sete transistores Darlington e funciona como se houvesse sete transistores TIP120 tudo em um pacote, suportando até 500 mA por canal e com uma queda de tensão interna de cerca de 1V quando ligado.

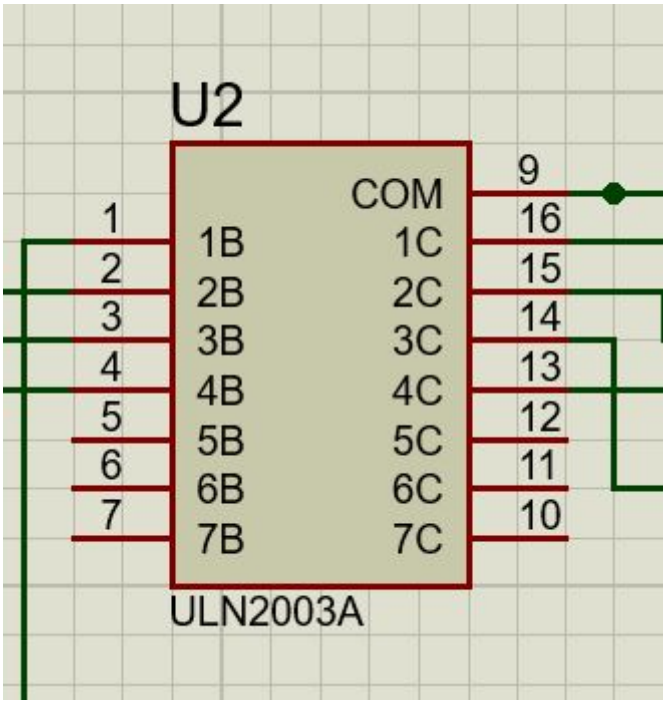

Figura 1. Circuito Integrado ULN2003A

O motor utilizado na simulação é do tipo unipolar de 6 fios. São 2 fios para o Vcc, 1 fio de força e 4 fios para as bobinas que controlam a sua movimentação, **Figura 2**. Para que seja controlado, a sequência enviada para o motor deve estar de acordo com a sua configuração [2].

No controle desenvolvido o motor anda 90º a cada passo e quando completa 360º inverte seu sentido de rotação. Para que seja possível isto e como dito anteriormente, devemos enviar o acionamento das bobinas na sequência correta e ele se dá na forma descrita na **Tabela 1.**

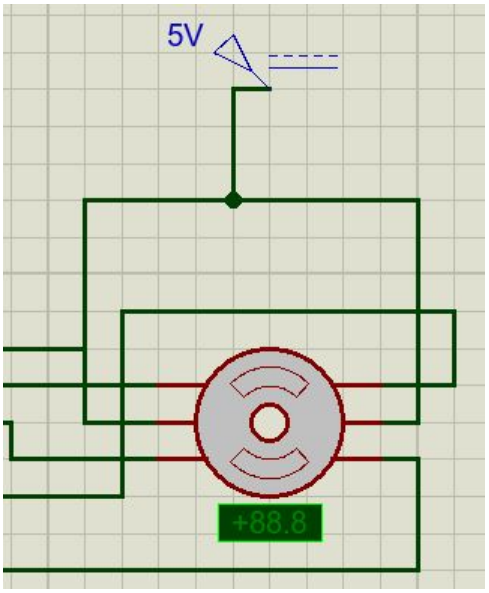

Figura 2. Motor de passo utilizado na simulação.

Valor do sinal	Posição
1100	90º
0110	180º
0011	270º
1001	360º

Tabela 1. Sinais enviados para a movimentação do motor.

O código é Assembly implementado no microcontrolador AT89C51 é demonstrado a seguir:

```
org 0H
stepper equ P1

main:
mov ACC, #4

; move o motor no sentido horário de 90 em 90º
horario:
    mov stepper, #0CH
    acall delay
    mov stepper, #06H
    acall delay
    mov stepper, #03H
    acall delay
    mov stepper, #09H
    acall delay
```

```
; move no sentido anti-horario de 90 em 90º voltando para a posição original
antihorario:
        mov stepper, #03H
        acall delay
        mov stepper, #06H
        acall delay
        mov stepper, #0CH
        acall delay
        mov stepper, #09H
        acall delay
        djnz  ACC, horario
        sjmp main

; delay que varia conforme o tempo e com isso aumenta a velocidade
delay:
        mov r7, ACC
wait2:
        mov r6,#0FFH
wait1:
        mov r5,#0FFH
wait:
        djnz r5,wait
        djnz r6,wait1
        djnz r7,wait2
        ret
        end
```

O motor a cada volta completa aumenta sua velocidade através da diminuição do delay entre as trocas de posição. As portas P1.0 ~ P1.4 são as responsáveis por enviar o sinal ao CI ULN2003A. O circuito eletrônico final implementado no Proteus pode ser visualizado na **Figura 3.**

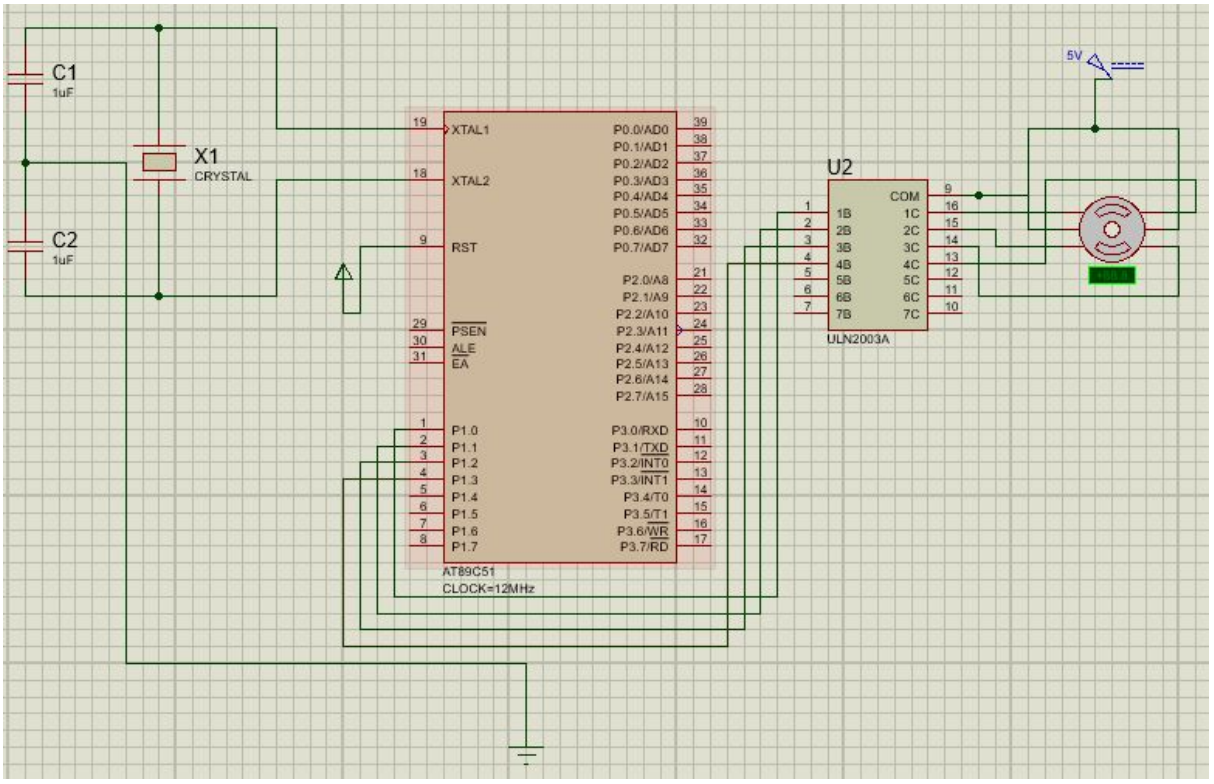

Figura 3. Circuito para controle do motor de passo

3. Implementação do PWM

Esta implementação tem como objetivo a criação de uma ferramenta que irá auxiliar na construção do controle PID. Como o próprio nome já diz, o PWM (Pulse Width Modulation) atua como modulador por largura de pulso. Em vez de serem enviados sinais contínuos de tensão para um motor ou outra aplicação, são enviados pulsos que podem variar sua largura. Consequentemente, pode-se determinar a tensão de saída através destes pulsos digitais. Quanto mais largo for o pulso, mais próximo da tensão fornecida estará. O contrário é verdadeiro também.

A **Figura 4** ilustra como é feita a modulação. A partir de um sinal de entrada de 12V, para se obter 3V de saída, deve-se modular a largura em 25% do tempo em nível alto e 75% do tempo em nível baixo, por exemplo. Dessa forma é possível controlar a tensão somente com sinais digitais, o que torna a vida de um programador de microcontrolador extremamente mais fácil.

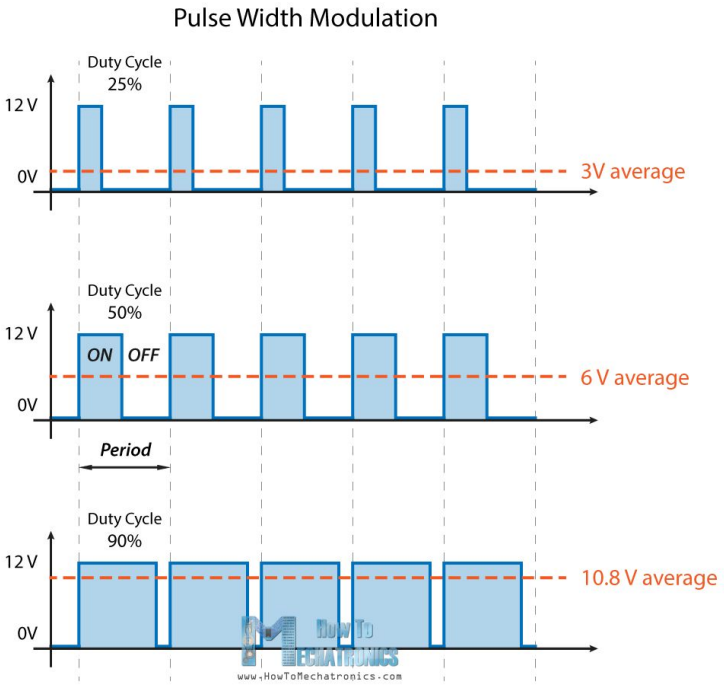

Figura 4. Funcionamento do PWM.[3]

A família de microcontroladores 8051 não conta com PWM nativo, é necessário sua implementação desde o início. São utilizados 3 pinos do MC:

- dois pinos (P1.0 e P1.1) são de *push buttons* de controle, que incrementam e "decrementam" a largura do pulso, alterando a velocidade do motor;
- um pino para a saída do PWM (P3.7), responsável por variar a largura do pulso;

Para que seja possível a variação do nível lógico comitantemente ativando ou desativando a tensão de alimentação do motor, é utilizado um transistor do tipo NPN. A base é ligada no PWM, o coletor na saída do motor e o emissor é aterrado. Assim, a corrente só passa se a base estiver ativada.

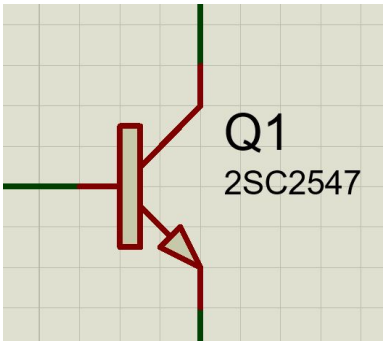

Figura 4. Transistor de alimentação do motor com o PWM

O código foi desenvolvido em linguagem C e compilado na ferramenta de suporte da família 8051 Keil C. Pode ser visualizado a seguir:

```
// definicao dos pinos
sbit Inc = 0x90; //P1.0
sbit Dec = 0x91; //P1.1
sbit PWM = 0x0B7; //P3.7

void delay(int);

void main(void){
        int speed;
        while(1){
                if(!Dec)
                // diminui a velocidade diminuindo a largura do pulso
                        speed = speed > 0 ? speed - 1 : 0;
                if(!Inc)
                // aumenta a velocidade aumentando a largura do pulso
                        speed = speed < 500 ? speed + 1 : 500;

                // a largura do pulso é feita através de um delay
                // quanto menor a velocidade, maior o delay
                PWM=1;
                delay(speed);
                PWM=0;
                delay(500-speed);
   }
 }

void delay(int j)
 { for(; j>0; j--)
   {
   }
 }
```

O delay no código funciona como a largura do pulso desejada, quanto maior o delay maior será a largura e, consequentemente, a velocidade do motor DC utilizado na simulação. O circuito eletrônico final simulado no Proteus pode ser visualizado na **Figura 5.**

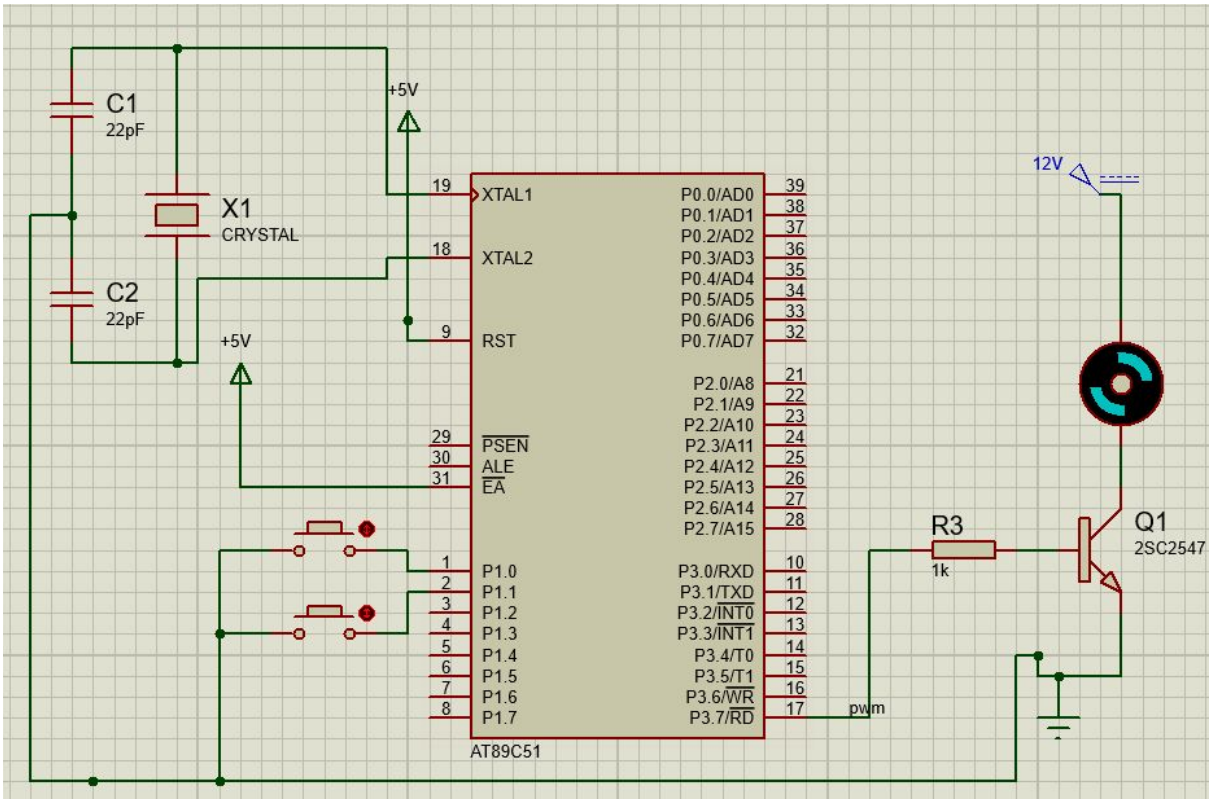

Figura 5. Circuito eletrônico da implementação de um motor DC com controle PWM

4. Implementação do contador de giro

Nesta seção serão feitas duas implementações: a primeira simplesmente contará o número de voltas do motor, já a segunda contará as rotações por minutos RPM do motor DC. A segunda é um melhoramento da primeira implementação, pois insere outra variável na conta, o tempo e fará uso de interrupção do timer. Os dois utilizam interrupção externa sensível a borda para detectar as rotações do motor com sensor de posição.

As portas de saída utilizadas para o PWM que comandará o motor são P3.0 e P3.1 e as portas de saída para os leds de indicação do sentido de rotação são as P0.1 e P0.2. A porta de entrada para o sensor de posição é a P3.2 e a porta de entrada para a chave que determina o sentido de rotação é a P0.0.

Cada display de 7 segmentos utilizado para contar as voltas na primeira implementação e para contar RPM na segunda, é acionado por 4 portas que enviam um sinal BCD de 4 bits e o próprio display decodifica para o valor correspondente. As portas são: P1 e P2.0 à P2.3.

É utilizado também o circuito integrado L293D que conta com duas pontes H, só será utilizado uma ponte H nos dois casos. Com este circuito podemos ter um configuração que permite o envio de sinal PWM para as duas extremidades do motor enquanto o L293D é alimentado com uma tensão fixa e modula-se a partir dela. A **Figura 6** pode ilustrar melhor como fica o CI quando ligado ao motor DC. IN1 e IN2 são as entradas dos PWM's vindo do MC, variando apenas uma entrada e mantendo a outra em nível baixo, o motor gira em uma direção. Invertendo o sinal oscilante e o fixo baixo, o motor gira no sentido inverso. OUT1 e OUT2 são as

saídas que conectam às extremidades do motor. Vss é o sinal de referência e Vs é a tensão de alimentação do motor.

O contador de giro tem um limite de 999 que é o máximo que 3 displays de 7 segmentos configurados na forma decimal podem mostrar. A tensão de alimentação do motor pode ser regulada por um potênciometro atrelado ao Vs.

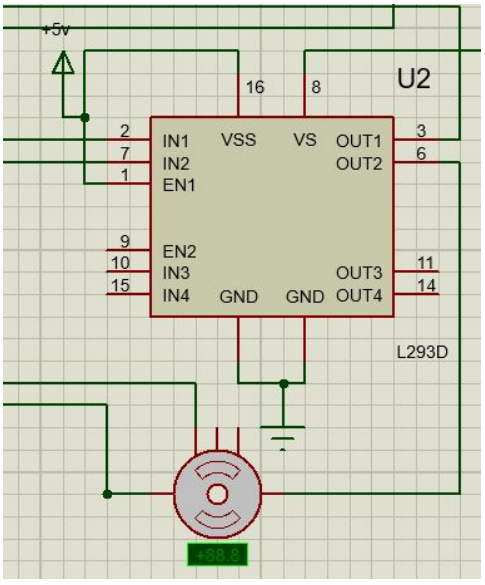

Figura 6. Configuração da ponte H com o motor DC.

O maior trabalho esteve em encontrar uma forma de construir os displays de 7 segmentos dado o número limitado de portas da família 8051. Por isso foi escolhido displays que possuem decodificador BCD. O PWM não foi utilizado nesta implementação mas suas portas estão configurados no circuito eletrônico para apenas mudar o sentido de rotação do motor. O código do contador de giro pode ser visto a seguir.

```
//definicao das portas
sbit P1_0 = 0x90;

sbit P0_0 = 0x80;
sbit LED0 = 0x81;
sbit LED1 = 0x82;

sbit P3_0 = 0x0B0;
sbit P3_1 = 0x0B1;

#include <reg51.h>

int contador =0, aux;
int nums[10]={0,1,2,3,4,5,6,7,8,9};  //vetor de 9 digitos para o 7 segmentos
int digito[3] = {0,0,0}; // vetor para armazenar os 3 digitos do display da simulacao

void delay(void);
void ex0_isr (void) interrupt 0{
```

```
        contador++;     // conta quantas vezes a interupção é chamada, ou seja, quantas rotações o motor está fazendo

        //procedimentos para pegar os digitos do contador e passar para os displays
        if(contador>=100){
          aux = contador;
                digito[0] = aux/100;
                aux %= 100;
                digito[1] = aux/10;
                aux %= 10;
                digito[2] = aux;
        }else if(contador >= 10){
                aux = contador;
                digito[0] = 0;
                digito[1] = aux/10;
                aux %= 10;
                digito[2] = aux;
        }else{
                digito[0] = 0;
                digito[1] = 0;
                digito[2] = contador;
        }
                // valores em BCD
                aux = digito[0]<<4;   //deloca 4 bits para esquerda o valor
                P1 = aux + digito[1]; // passa o valor para a porta desejada junto com o digito do meio
                P2 = digito[2];
        if(contador >= 999){
                 contador = 0;
        }
}

void main(void) {
        IT0 = 1; // ativa interrupção sensível a borda
        EX0 = 1; // ativa interrupção externa
        EA = 1; //ativa interupções
        while (1){

        //seleciona o sentido de rotação do motor
                if(P0_0 == 1){
                        LED0 = 1;
                        LED1 = 0;
                        P3_0 = 1;
                        P3_1 = 0;
                }
                else{
                       LED0 = 0;
                        LED1 = 1;
```

```
                    P3_0 = 0;
                    P3_1 = 1;
                }
        }
}
```

A variação deste código vem com o desenvolvimento para que ele conte as rotações por minuto e não somente o número total de voltas. Para isso é necessário a criação de um timer de 50 ms e combinado com 10 repetições deste timer, para dar uma amostra da velocidade a cada 500 ms ou 0,5 s.O motor desta enviará 30 pulsos por cada rotação completa, isto deixa a contagem mais precisa porém aumenta o número de interrupções. Um pulso é uma interrupção externa pois é sensível a borda.

A contagem é feita da seguinte forma: cada rotação tem 30 pulsos, a cada meio segundo é calculada a velocidade; se para 1 pulso por rotação em amostras de 1 s multiplica-se o contador por 60 para obter o RPM, proporcionalmente 30 pulsos por rotação com amostras de 0,5 s multiplica-se por 4 para obter o RPM. O código em linguagem C pode ser visualizado a seguir.

```
//definicao das portas
sbit P1_0 = 0x90;
sbit P0_0 = 0x80;
sbit P3_0 = 0x0B0;
sbit P3_1 = 0x0B1;

#include <reg51.h>

int contador,timerCounter;
int nums[10]={0,1,2,3,4,5,6,7,8,9};  //vetor de 9 digitos para o 7 segmentos
int digito[3] = {0,0,0}; // vetor para armazenar os 3 digitos do display da simulacao


void passaDigito(void);
void ex0_isr (void) interrupt 0
{
    contador++;
// conta quantas vezes a interupção é chamada, ou seja, quantas rotações o motor
//está fazendo
}

void timer0(void) interrupt 1{ //interrupçao do timer 0, a cada 50ms ela é chamada
        TR0 = 0;
        TH0 = 0x3C;
        TL0 = 0x0AF; // carrega o timer 0 com 50ms
        TF0 = 0;
        if(timerCounter < 9){
//são contadas amostras das rotações do motor a cada 0.5s
                    timerCounter++;
```

```
        }
        else{
// quando dá os 500ms, a função de passagem de valor para os 7-seg é chamada
                timerCounter = 0;
                EX0 = 0;
                passaDigito();
                //PID();
                contador = 0;
                EX0 = 1;
        }
        TR0 = 1;
}

void main(void)
 {
        TMOD = 0x01; // ativa timer 0 no modo 1
        TH0 = 0x3C;
        TL0 = 0x0AF; // carrega o timer 0 com 50ms
        TR0 = 1; // liga o timer 0
        IT0 = 1; // ativa interrupção sensível a borda
        EX0 = 1; // ativa interrupção externa
        EA = 1; //ativa interupções
        ET0 = 1; //ativa interrupção para o timer 0

   while (1){
                                        //seleciona o sentido de rotação do motor
                if(P0_0 == 1){
                        P3_0 = 1;
                        P3_1 = 0;
                }
                else{
                        P3_0 = 0;
                        P3_1 = 1;
                }
   }
}

 void passaDigito(){
        int aux;
// cada rotação completa do motor envia 30 pulsos de interrupção
// contador tem armazenado a quantidade de pulsos em 500ms
// se para 1 pulso por rotação em 1s fazemos *60 o contador
// para 30 pulsos por rotação em 0,5s temos *4 o contador
// o .35 é fator de atraso das instruções
        contador = contador * 4;
// pega os valores de cada digito do contador
        if(contador>=100){
                digito[0] = contador/100;
```

```
            contador %= 100;
            digito[1] = contador/10;
            contador %= 10;
            digito[2] = contador;
        }else if(contador >= 10){
            digito[0] = 0;
            digito[1] = contador/10;
            contador %= 10;
            digito[2] = contador;
        }else{
            digito[0] = 0;
            digito[1] = 0;
            digito[2] = contador;
        }
            // valores em BCD
            aux = digito[0]<<4;   //deloca 4 bits para esquerda o valor
            P1 = aux + digito[1]; // passa o valor para a porta desejada junto com o
digito do meio
            P2 = digito[2];
 }
```

O circuito eletrônico resultante das duas implementações é igual e pode ser visualizado na **Figura 7.**

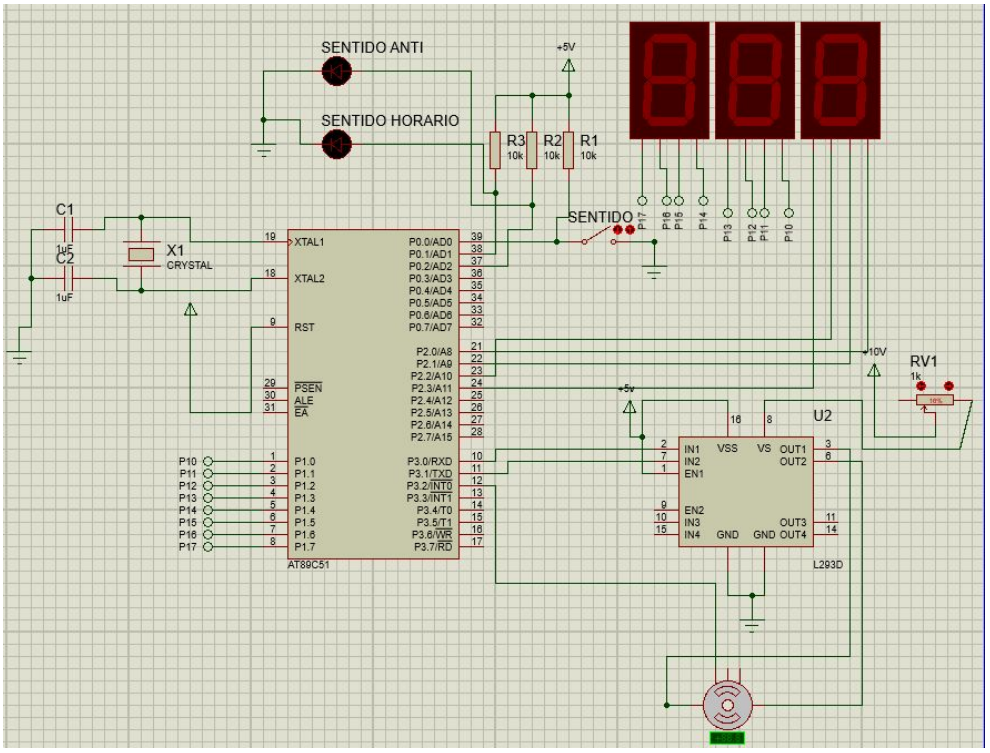

Figura 7. Circuito eletrônico do contador de giro.

5. Implementação do PID

A idéia básica por trás de um controlador PID é ler um sensor, calcular a resposta de saída do atuador através do cálculo proporcional, integral e derivativo e então somar os três componentes para calcular a saída [4]. A última implementação deste projeto reúne as etapas anteriores desenvolvidas com o motor DC. Para o funcionamento do PID é necessário o contador de giro (sensor) e a partir dele pode-se definir qual a modulação de saída necessária (PWM, resposta de saída do atuador) para chegar ao ponto de velocidade esperado.

A componente proporcional depende apenas da diferença entre o ponto de ajuste e a variável de processo. Esta diferença é referida como o termo de erro. Em geral, aumentando o ganho proporcional irá aumentar a velocidade da resposta do sistema de controle [4].

A componente integral soma o termo de erro ao longo do tempo. O resultado é que mesmo um pequeno erro fará com que a componente integral aumente lentamente. A resposta integral irá aumentando ao longo do tempo a menos que o erro seja zero, portanto, o efeito é o de conduzir o erro de estado estacionário para zero [4].

A componente derivada faz com que a saída diminua se a variável de processo está aumentando rapidamente. A derivada de resposta é proporcional à taxa de variação da variável de processo. Aumentar o parâmetro do tempo derivativo (Kd) fará com que o sistema de controle reaja mais fortemente à mudanças no parâmetro de erro aumentando a velocidade da resposta global de controle do sistema [4].

Encontrar os parâmetros ideais não é uma tarefa fácil e depende também dos parâmetros elétricos do motor utilizado. Nesta implementação foca-se mais no funcionamento do que a análise de overshooting, tempo de resposta e erro em regime.

A simulação da condição de carga no motor é feita através dos *push buttons* inseridos anteriormente que regulam a velocidade do motor. Como agora será utilizado um método de controle, escolhe-se um valor para a velocidade e este é definido como setpoint do sistema. Variando a velocidade nos botões, o sistema tentará sempre voltar para o setpoint. Os parâmetros Kp, Kd e Ki determinam se o operador quer que o sistema chegue rápido neste setpoint, se oscile muito e se o erro entre o setpoint e o valor atual do motor é grande ou pequeno. A resposta em relação a entrada pode ser visualizada tanto no conta-giro feito com os displays de 7 segmentos quando num osciloscópio simulado, **Figura 8.**

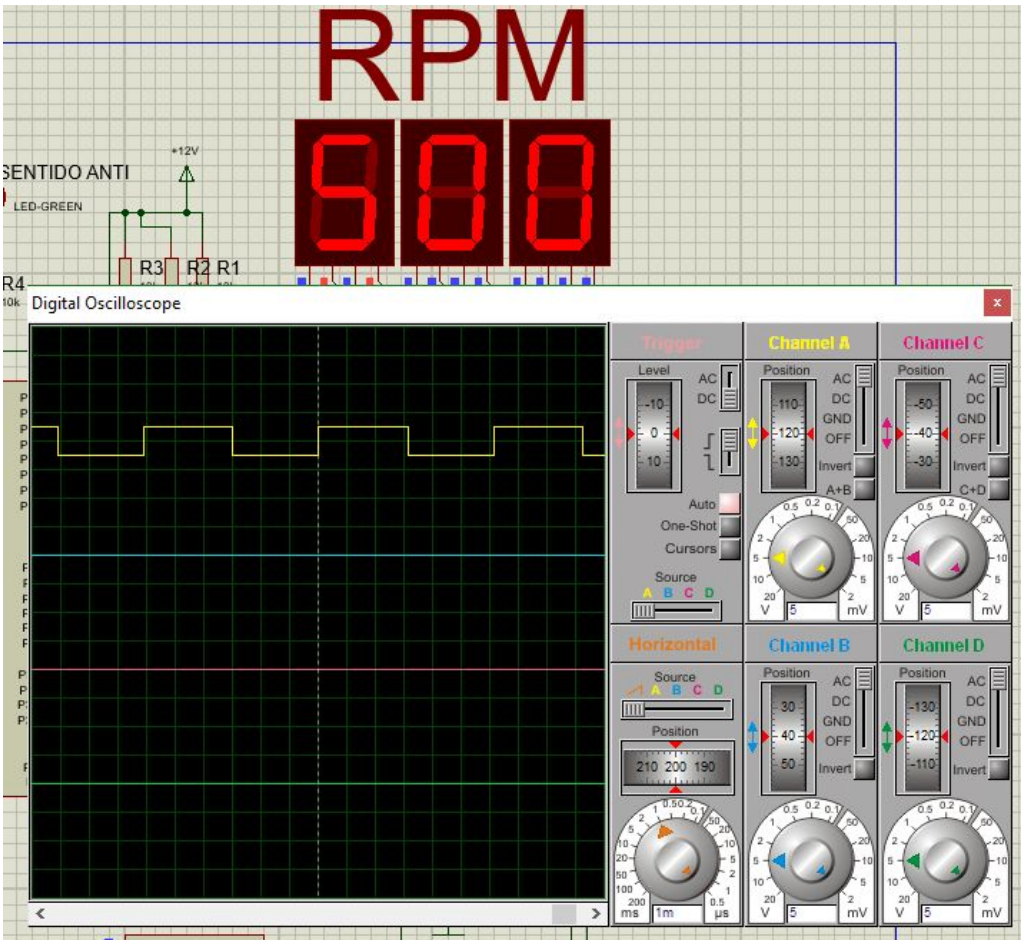

Figura 8. Saída do sistema no osciloscópio e nos displays de 7 segmentos.

O código em linguagem, que pode ser visualizado a seguir, reuniu os códigos desenvolvidas anteriormente e ainda conta com o cálculo do PID [5] a cada 500 ms, junto com o cálculo do RPM.

```
sbit P1_0 = 0x90; // o endereço do pino 1_0 passa a se chamar P1_0
sbit P2_4 = 0x0A4;

//bits da chave, incremento e decremento definidos nas portas P0.0, P0.1 E P0.2
sbit chave = 0x80;
sbit Inc = 0x81;
sbit Dec = 0X82;
sbit LED0 = 0x83; // leds de sentido definidos nas portas p0.3 e p0.4
sbit LED1 = 0x84;

// bits do PWM definidos nas portas P3.0 e P3.1
sbit PWM0 = 0x0B0;
sbit PWM1 = 0x0B1;

#include <reg51.h>
#include <math.h>

// parametros do pid
```

```
float Kp = 0.5, Kd = 0.1, Ki = 0.2;
int contador, direcao, timerCounter, speed, carga, erro = 0, erroAnterior = 0, integral = 0, derivativo;
int nums[10]={0,1,2,3,4,5,6,7,8,9};  //vetor de 9 digitos para o 7 segmentos
int digito[3] = {0,0,0}; // vetor para armazenar os 3 digitos do display da simulacao

// setpoint do motor 200RPM
int setPoint = 500;


void delay(int);
void passaDigito(void);
void acionaPWM(int,int);
void PID(void);

void ex0_isr (void) interrupt 0{ // interrupção externa que corresponde aos pulsos do motor
    contador++;   // conta quantas vezes a interupção é chamada, ou seja, quantas rotações o motor está fazendo
}

void timer0(void) interrupt 1{ //interrupçao do timer 0, a cada 50ms ela é chamada
                TR0 = 0;
                TH0 = 0x3C;
                TL0 = 0x0AF; // carrega o timer 0 com 50ms
                TF0 = 0;
                if(timerCounter < 9){//são contadas amostras das rotações do motor a cada 0.5s
                                timerCounter++;
                }
                else{ // quando dá os 500ms, a função de passagem de valor para os 7-seg é chamada
                        timerCounter = 0;
                        EX0 = 0;
                        passaDigito();
                        //PID();
                        contador = 0;
                        EX0 = 1;
                }
                TR0 = 1;
}



void main(void) {
        TMOD = 0x01; // ativa timer 0 no modo 1
        TH0 = 0x3C;
        TL0 = 0x0AF; // carrega o timer 0 com 50ms
```

```
        TR0 = 1; // liga o timer 0
        IT0 = 1; // ativa interrupção sensível a borda
        EX0 = 1; // ativa interrupção externa
        EA = 1; //ativa interupções
        ET0 = 1; //ativa interrupção para o timer 0

        while (1){
                // o sentido de rotação é definido dependendo da posição da chave
                if(chave == 0){
                        direcao = 0;
                        LED0 = 0;
                        LED1 = 1;
                }
                else{
                        direcao =1;
                        LED0 = 1;
                        LED1 = 0;
                }
                //botoes para simular carga no motor e mostrar o PID funcionando
                if(!Inc)
                // incrementa velocidade
                        speed = speed > 0 ? speed - 4 : 0;
                if(!Dec)
                // decrementa velocidade
                        speed = speed < 1000 ? speed + 4 : 1000;

                //envia a direção definida e a velocidade
                acionaPWM(direcao, speed);
  }
}

 void delay(int h){
   for(;h>0;h--);

 }

 // calcula a velocidade de acordo com PID
 void PID(){
        erro = setPoint - contador;
        integral = integral + erro;
        derivativo = erro - erroAnterior;
        erroAnterior = erro;
        speed = Kp*erro + Ki*integral + Kd*derivativo;
}

 // função que oscila a saída dos pinos
  // aumentando o delay, criam-se pulsos de menor largura e consequentemente
velocidade menor
```

```
 // para máxima velocidade a saída deve permanecer sempre em 1
void acionaPWM(int direcao, int velocidade){
        if (direcao == 1){
                PWM1=0;
                PWM0=1;
                delay(velocidade);
                PWM0=0;
                delay(setPoint-velocidade);
        }
        else{
                PWM0 = 0;
                PWM1=1;
                delay(velocidade);
                PWM1=0;
                delay(setPoint-velocidade);
        }
}

 void passaDigito(){
        int aux;
        // cada rotação completa do motor envia 30 pulsos de interrupção
        // contador tem armazenado a quantidade de pulsos em 500ms
        // se para 1 pulso por rotação em 1s fazemos *60 o contador
        // para 30 pulsos por rotação em 0,5s temos *4 o contador
        contador = contador * 4;
        PID();
        // pega os valores de cada digito do contador
        if(contador>=100){
                digito[0] = contador/100;
                contador %= 100;
                digito[1] = contador/10;
                contador %= 10;
                digito[2] = contador;
        }else if(contador >= 10){
                digito[0] = 0;
                digito[1] = contador/10;
                contador %= 10;
                digito[2] = contador;
        }else{
                digito[0] = 0;
                digito[1] = 0;
                digito[2] = contador;
        }
        // valores em BCD
        aux = digito[0]<<4;   //deloca 4 bits para esquerda o valor
        P1 = aux + digito[1];
// passa o valor para a porta desejada junto com o digito do meio
        P2 = digito[2];
```

}

O circuito eletrônico final implementado no Proteus com PWM, displays de 7 segmentos, *push buttons*, leds de sentido de rotação e PID pode ser visualizado na **Figura 9.**

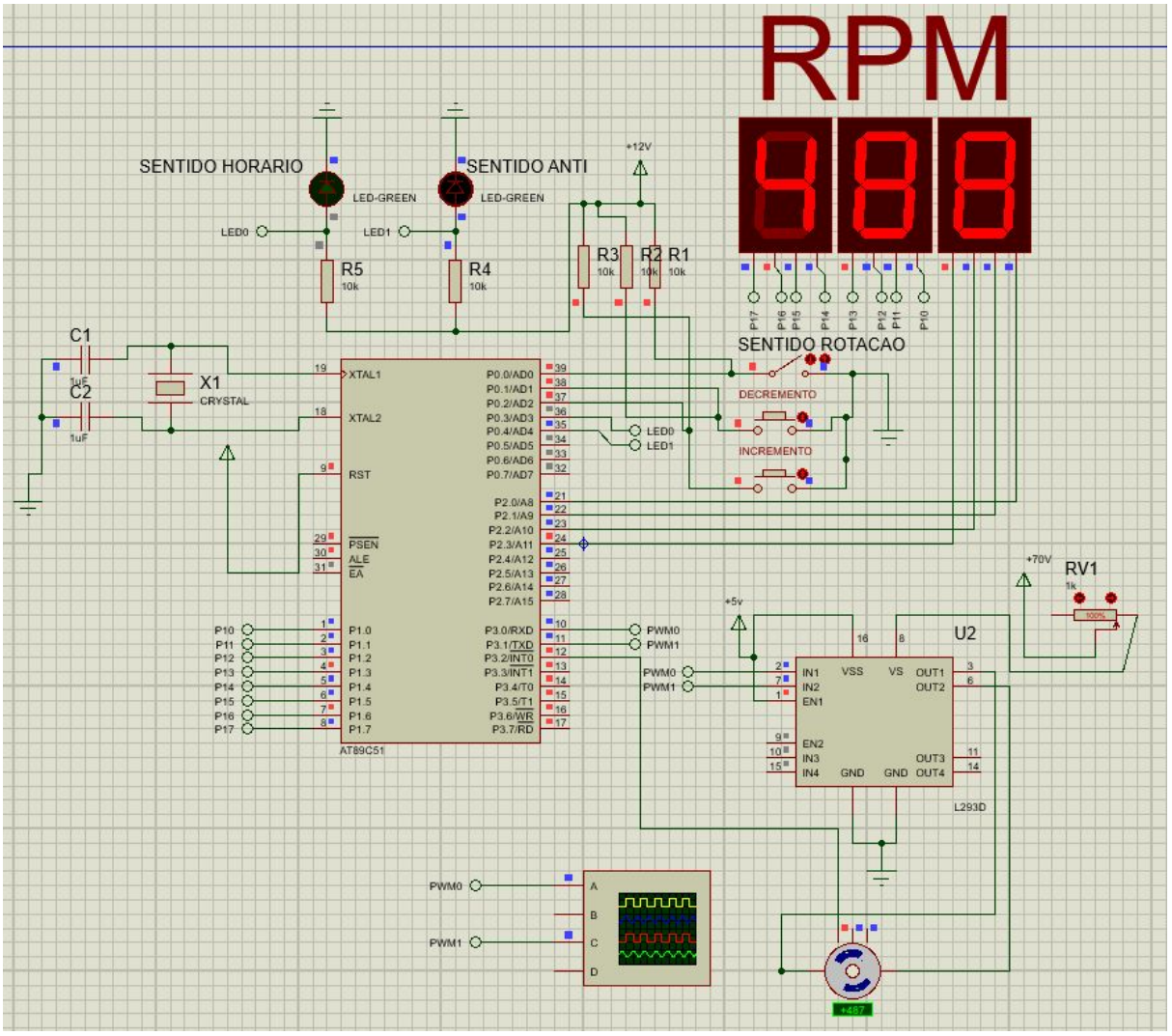

Figura 9. Circuito eletrônico final.

6. Conclusão

Através destas implementações, o grupo pode colocar em prática tudo aquilo que aprendeu em sala de aula bem como se aprofundar em temas que não tiveram muito espaço na disciplina como a utilização do software de simulação Proteus e a criação de códigos programação para a família 8051 via linguagem C.

O microcontrolador utilizado se mostrou adequado para a prática, com o PID implementado custou apenas 1707 bytes de memória de programa, sobrando ainda cerca de 2300 bytes a serem utilizados por outras funções.

Cabe ressaltar que é possível chegar em um resultado ainda mais satisfatório do controlador PID fazendo um estudo aprofundado do motor DC e utilizando métodos para ajuste dos parâmetros Kp, Kd e Ki. Um exemplo é o método Ziegler-Nichols.

7. Bibliografia

1. WIKIPEDIA. **Motor de Passo**. Disponível em: <https://pt.wikipedia.org/wiki/Motor_de_passo>. Acesso em: Dezembro de 2017.
2. VIVALDINI, K. C. T. **Motores de Passo.** Disponível em: <http://www.mecatronica.eesc.usp.br/wiki/upload/0/0a/Motor_passo.pdf>. Acesso em: Dezembro de 2017.
3. HOW TO MECHATRONICS. **Arduino DC Motor Control Tutorial – L298N | PWM | H-Bridge.** Disponível em: <http://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/>. Acesso em: Dezembro de 2017.
4. NATIONAL INSTRUMENTS. **Explicando a Teoria PID.** Disponível em: <http://www.ni.com/white-paper/3782/pt/>. Acesso em: Dezembro de 2017.
5. GILLARD, G. **An introduction and tutorial for PID controllers.** Disponível em: <https://www.madison-schools.com/site/handlers/filedownload.ashx?moduleinstanceid=19089&dataid=70925&FileName=PID%20Programming%20Guide.pdf>. Acesso em: Dezembro de 2017.