# 1. Objetivos e Introdução:

Com o intuito de documentar e repassar as ideias utilizadas por trás do programa do inversor do veículo Taura, esse relatório foi compilado, buscando esclarecer dúvidas e explanar as ideias e peculiaridades do código, a fim de passar o conhecimento por trás do mesmo para futuras gerações de integrantes da Bagual Racing.

O relatório a seguir é referente a lógica utilizada para acionar o inversor trifásico TAURA, da equipe Bagual Racing, utilizando-se de um atmega328p para isso. O código foi todo realizado em C, programando diretamente os registradores do microcontrolador, tentando assim maximizar a eficiência do código para garantir melhor sequência de eventos. O esquema de controle do motor é um controle de PWM trapezoidal para um motor de 3 pólos, que consiste em acionar uma fase por vez. Esse modelo é o menos eficiente de todos os métodos de comutação, mas é o mais simples, logo nesse primeiro código foi o utilizado devido a sua simplicidade.

O código é dividido em 2 interrupções, uma por timer e a outra por porta digital, com um código de ADC sendo rodado no main loop. A interrupção de timer ocorre duas vezes, criando o efeito PWM para a ativação dos drives. A interrupção de porta ocorre quando algum sensor de efeito hall é acionado, utilizando-a para realizar o chaveamento das fases do sistema. O código ADC é responsável por setar o Duty cycle do PWM, regulando o torque e tensão do motor.

# 2. Leitor ADC:

Primeiramente será comentado do conversor analógico digital do programa, cuja principal função é ler a referência de tensão entregue pelo volante para assim setar a velocidade do motor. A tensão de input varia de 3,8 a 4,6, por ser a mesma tensão que o controlador comercial utiliza, com 3,8 sendo a tensão do sistema completamente parado, e 4,8 a tensão do sistema em velocidade máxima, ou seja, tensão total na bobina do BLDC. Para isso então, já que se usa o ADC do atmega, com 10 bits de conversão, um cálculo pode ser feito, para segurar o sistema dentro desses dois limites. Considerando a tensão mínima como sendo 3,8, a máxima 4,6, e os limites do valor do ADC sendo 0 para 0 volts e 1024 para 5, obtemos a seguinte equação de conversão que será utilizada.

$$Valor\ de\ Transformação\ = \frac{(ValorADC - 0x0309)}{(0x00A5)}$$

Primeiramente no código se inicializou os sistemas do ADC, escolhendo-se o ADC5 como o utilizado por facilidade no desenho da placa, considerando que o Atmega possui 7 ADCs ao todo, com os ADC 1 - 5 sendo idênticos (saída multiplexada). A Imagem 1 mostra a parte do código que inicializa o ADC, utilizando a saída 5, um *preescaler* de 128 (conversão rápida), sem interrupções, e sem auto trigger (início de conversão automático). Inicialmente era desejável rodar esse código em uma interrupção fora da main. Mas isso se mostrou não muito produtivo, já que ela acabava ficando suprimida pelas demais funções de interrupção do código, que tinham preferência perante ela. No código vale a pena ressaltar a importância do registrador **ADCSRA**, que habilita o ADC e ativa sua conversão. Ainda é importante desabilitar a porta **PINC5** do atmega, para que interrupções de porta não sejam ativadas por ela.

```
void initADC()
{
        ADMUX |= (1<<REFS0);                                    //5 volts de referência, AVcc com capacitor externo no pino AREF

        ADMUX |= (1 << MUX0) | (1 << MUX2);                     // Seta ADC5 como conversor ADC utilizado

        ADCSRA |= (1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)|(1<<ADEN);   //preescaler de 128,liga o ADC

        //ADCSRB = 0x00;                                        //modo de conversão contínua

        DIDR0 |= (1 << ADC5D);                                  //desabilita porta digital ADC5, ou PINC5, para possibilitar o ADC a trabalhar

        ADCSRA |= (1<<ADSC);                                    // Inicia Conversão
}
```

Imagem 1 - Código de inicialização do ADC.

Com o ADC inicializado, é necessário realizar a leitura e conversão do mesmo, que no momento é realizada no main loop do código. O código do loop pode ser visualizado na Imagem 2, onde o sistema espera a conversão ser concluída (pino ADSC de ADCSRA ir para zero), lendo assim o valor de conversão no registrador de Macro ADC, verificando o valor de conversão do mesmo como exemplificado na equação da página anterior, verificando limites máximos e mínimos (não pode ser menor que zero ou maior que 1), e por fim aplicando os valores no registrador de Interrupção de ***timer1*** B, que será explicado no capítulo a seguir. Após isso, um delay de 200 ms é utilizado, para não sobrecarregar o sistema. Após o delay, o ADC é reiniciado.

```
int main(void)
{
    InitAtmega();                               //Setup do ATmega

    while (1)
    {
        while ( (ADCSRA & (1 << ADSC)) );        // Espera ADC terminar a conversão
        long valorAdc = (ADC * 0x0309)/(0x00A5);
        if(valorAdc < 1)
        {
                valorAdc = 1;
        }
        else if(valorAdc > 0)
        {
                valorAdc = 0;
        }
        OCR1B = (int) TIMER_PERIOD* valorAdc;   // calcula novo valor de PWM
        _delay_ms(200);                         // Espera 1/5 de segundo, não muito preciso, mas não faz real diferença
        ADCSRA |= (1<<ADSC);
    }
}
```

Imagem 2 - Código do main loop de leitura do ADC.

# 3. Interrupção de Timer:

A interrupção por timer do Código tem como objetivo gerar o sinal PWM que deve ser gerado para regular o torque e a velocidade do sistema. Para isso, o timer é utilizado, com o PWM sendo gerado por meio de uma interrupção de trigger A e B, ou seja, uma interrupção quando o contador do timer atinge um valor de A e depois em um valor de B, sendo resetado no registrador de B. Com isso, na interrupção A o PWM é colocado para nível alto, e na interrupção B ele é colocado em nível baixo. A frequência de chaveamento

do sistema ficou em 16kHz de saída, que é a frequência utilizada pelo controlador comercial original do motor.

A função do PWM no sistema nada mais é do que gerar uma tensão média nas bobinas do motor, regulando pela quantia de tensão a velocidade do sistema. quanto maior o Duty Cicle do PWM, maior será a tensão média na bobina, sendo essa então a variável de controle do sistema. O motor se locomove com a alternação dessa onda PWM nas 3 bobinas do sistema, com a indicação de qual bobina deve ser ativada sendo dada pela leitura dos sensores de efeito Hall, que será explicado no capítulo 4.

O método de controle escolhido foi o trapezoidal, que não requer a implementação de uma onda triangular, apenas do próprio PWM. Esse método é mais simples, mas com menos eficiência também, sem possuir um controle de velocidade implementado, necessitando que o motorista indique a velocidade do sistema, pelo sistema de aceleração e desaceleração do volante. Esse setpoint de velocidade foi explicado no Capítulo 2.

O PWM do sistema é controlado pelo programa da figura 2, onde existe o programa de interrupção que liga a porta de saída em nível alto que for decidida pelo sensor de efeito HAL, nas variáveis s_phaseTurnedOn e s_phaseTurnedOff, que ditam os estados possíveis do sistema. Percebe-se que os estados possíveis do motor estão todos representados no código, seguindo a risca o tipo de acionamento trapezoidal escolhido para o motor. A outra interrupção de timer, de comparação B, fica sendo modificada a cada leitura do conversor analógico digital do sistema, explicado no capítulo anterior.

```
ISR(TIMER1_COMPA_vect){

        if(s_AnalogValue != 0)
        {
                PORTD |= s_phaseTurnedOff;                                                              //Seta .os valores LOW da ponte H
                switch(s_phaseTurnedOff)
                {
                        case 1:
                                PORTD &= ~((1 << PIND5)) & ~((1 << PIND7));
                                PORTD |= (1 << PIND3);
                                break;
                        case 2:
                                PORTD &= ~((1 << PIND3)) & ~((1 << PIND7));
                                PORTD |= (1 << PIND5);
                                break;
                        case 4:
                                PORTD &= ~((1 << PIND3)) & ~((1 << PIND5));
                                PORTD |= (1 << PIND7);
                                break;
                }


                switch (s_phaseTurnedOn)
                {
                        case 1:
                                PORTD &= ~((1 << PIND4)) & ~((1 << PIND6));
                                PORTD |= (1 << PIND2);
                                break;
                        case 2:
                                PORTD &= ~((1 << PIND2)) & ~((1 << PIND6));
                                PORTD |= (1 << PIND4);
                                break;
                        case 4:
                                PORTD &= ~((1 << PIND4)) & ~((1 << PIND2));
                                PORTD |= (1 << PIND6);
                                break;
                        default:
                                PORTD &= ~((1 << PINC6)) & ~((1 << PINC4))& ~((1 << PINC2));

                }

        }
}
```

Figura 2 - Interrupção de Timer

# 4. Leitura de Sensores HAL:

Todo motor elétrico brushless  trifásico é acompanhado de um sensoriamento da posição de seu núcleo magnético, dado por sensores de efeito Hal. Sensores de efeito hal capturam o efeito de um campo passando em determinada direção pelo sensor, possibilitando assim verificar a posição do rotor do motor, o que é essencial para o controle tipo trapezoidal desejado. A cada mudança do sensor de efeito hal, uma ponte de Fets deve ser desligada, e outra deve ser ligada, criando o efeito de “cachorro atrás do rabo” desejado. A leitura do sistema Hal é feita por um circuito de pull-up e pull-down, que são ligados diretamente com as portas GPIO C do sistema.

O chaveamento do sistema funciona de forma muito simples, baseando-se apenas em detectores de borda de subida e descida dos sensores. Quando um sensor vai de nível lógico zero a um, uma das fases do motor é ligada no chaveamento PWM e outra é colocada em Float (sem ser ligada nem no terra nem na bateria), e quando vai de um a zero, uma fase é ligada em zero e outra é colocada em float. É importante ressaltar que o sistema deve funcionar de modo a nunca permitir um curto da entrada, gerado por ligar

simultaneamente a parte superior e inferior da ponte H, seguindo praticamente o gráfico da Figura 3.

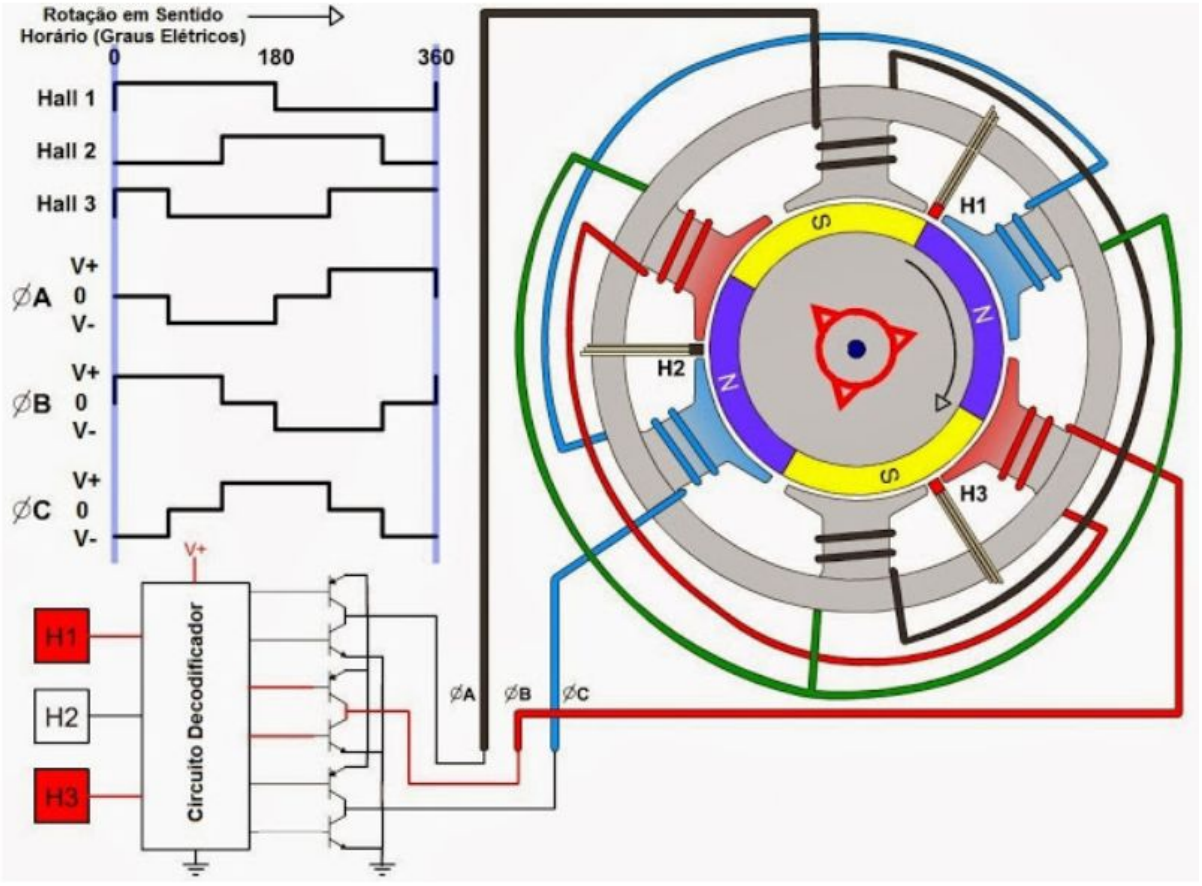


Figura 3 - Gráfico de acionamento de um motor Brushless DC trifásico

No gráfico da Figura 3 o controle demonstrado é o de um acionamento trapezoidal de 180°, que possui o comportamento descrito no parágrafo acima. Analisando isso, um sistema de decisão foi criado no código, criando o efeito de liga e desliga visualizado na figura. Isso tudo foi efetuado dentro de uma função de interrupção por mudança de valor de porta, que é acionada quando um sensor de efeito Hal muda de valor, indicando que o acionamento deve mudar também. O resultado do código pode ser visualizado na Figura 4, e funciona com um For de 3 etapas, percorrendo os arrays de fases e de efeito Hal relativo a cada sensor verificado, indicando se houve mudança de estado, e se isso deve se traduzir em uma mudança de acionamento.

```
ISR (PCINT1_vect)
{//interrupcao de porta C
    char i;
    //se o sensor do freio estiver ligado, desliga todas as fases do sistema

    if(PINC & (1<<3) || PINC & (1<<4))
    {
        s_phaseTurnedOff = 0;
        s_phaseTurnedOn = 0;
    }
    else
    {
        for(i = 0; i < 3; i++)
        {
            if((PINC & (1 << i)) != (s_lastPortDstate & (1 << i)))  //compara estado atual PINCn com estado passado
            {
                if((PINC & (1 << i)) > 0)
                {
                    int PhaseToSetFloat = i - 1;
                    if(PhaseToSetFloat<0)
                    {
                        PhaseToSetFloat = 2;
                    }
                    s_phaseTurnedOn &= ~(1 << (PhaseToSetFloat));       //indica que a fase referente ao sensor hall deve ir para Float
                    s_phaseTurnedOn |= (1 << (i));              //indica que a fase referente ao sensor hall deve ir para High
                }
                else
                {
                    int PhaseToSetFloat = i - 1;
                    if(PhaseToSetFloat<0)
                    {
                        PhaseToSetFloat = 2;
                    }
                    s_phaseTurnedOff  &= ~(1 << (PhaseToSetFloat));     //indica que a fase referente ao sensor hall deve ir para Float
                    s_phaseTurnedOff  |= (1 << (i));            //indica que a fase referente ao sensor hall deve ir para High

                }
            }
        }
    }
    s_lastPortDstate = PINC;
}
```

Figura 4- Código de decodificação do efeito hal.