Relatório 2 - Trabalho de Microprocessadores e Microcontrolares

Universidade Federal de Ouro Preto

1st Graziele de Cássia Rodrigues
Discente Engenharia De Computação
Universidade Federal de Ouro Preto
ICEA
21.1.8120

2nd Matheus Fernandes Gomes
Discente Engenharia de Computação
Universidade Federal de Ouro Preto
ICEA
17.2.5941

***Index Terms*—esquemático, componentes, especificações, algoritmo**

I. Esquemático

Utilizando como base a pinagem do Blue Pill, encontrada na imagem 1. Foi desenhado, por meio da plataforma EasyEDA o esquemático para o projeto de controle de motor que está sendo desenvolvido. O esquema se encontra abaixo na Imagem 2.

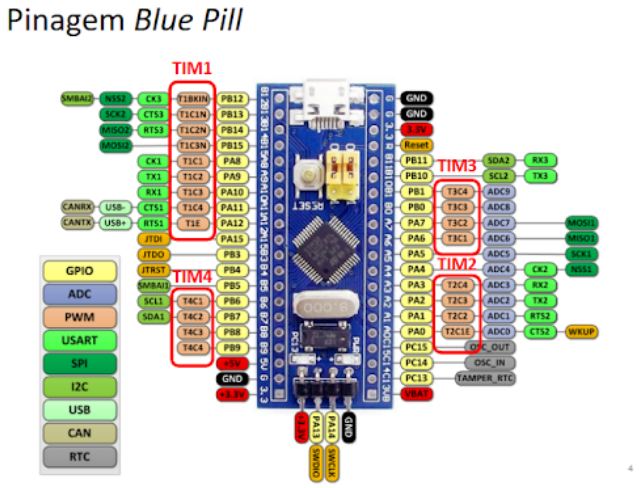

Fig. 1. Pinagem Blue Pill STM32

Perceba que foi optado usar o PA2 para o potenciomêtro, pois é um canal analógico. Enquanto para as saídas dos motores foi utilizado saídas digitais que há presença do timer e portanto pode ser aplicado o PWM, sendo as portas PB6, PB7, PB8 e PB9. Para ver o esquema, encontrado na figura 2, com maior detalhe acesse: https://drive.google.com/file/d/1eFP51GljIfYtS$_W$7XV7xJUKmWC5h9i/view?usp=*sharing*

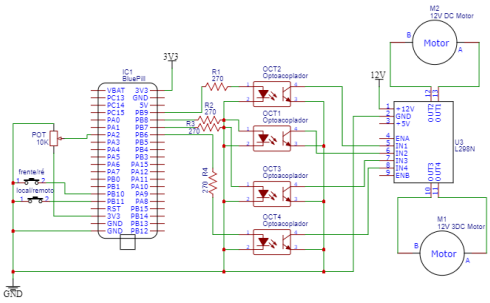

Fig. 2. Esquemático desenvolvido

A. Lista de Componentes

- 4 Optoacopladores familia 4N
- 1 potenciômetro 10K
- 4 resistores 270ohms
- 1 Driver Ponte H LN298N
- 2 Motores DC 12V
- 2 chaves
- 1 STM32F103C8T6

II. Especificações

- **Driver Ponte H L298N:**
  - Tensão Operação: 4.5V - 46V
  - Controle: 1 motor de passo ou 2 motores DC
  - Corrente operação: 2A por canal
  - Tensão Lógica: 4.5V-7V
  - Corrente Lógica: 0-36mA
  - Potência máxima: 25W
- **Octoacopladores:**
  - Tesão Reversa: 5V
  - Corrente adiantada: 60mA
  - Potência dissipada entrada: 100mW
  - Tensão Coletor: 70V

  - Tensão Emissor: 7V
  - Corrente coletor: 100mA
  - Potência dissipada saída 150mW

- **STM32F103C8:**
  - Tensão alimentação: 3.3V
  - Ivdd (Supply Current for Digital): 150mA
  - Ivss(Supply Current for Analog Section): 150mA
  - Corrente saída pinos: 25 mA

- **Motor DC:**
  - Tensão: 12V

III. Funções desenvolvidas

A. Converter sinais analógicos em digitais por meio do método de varredura

É utilizado a função HALADCStart, HALADCPollForConversion e HALADCGetValue para converter sinais analógicos em digitais por meio do método de varredura. A configuração do canal ADC para a varredura está definida no MXADC1Init na função HALADCConfigChanneL. É usado o ADC1 pois o potenciomêtro está conectado no PA2 - ADC1, canal 2.

TABLE I
CONVERTE ANALÓGICO

```
HAL_ADC_Start(&hadc1);
HAL_ADC_PollForConversion(&hadc1, 1);
uint16_t adc_value = HAL_ADC_GetValue
    (&hadc1);
// Mapear o valor do potenci metro
    para a faixa de 0-100
uint16_t duty_cycle = (adc_value *
    100) / 4095;
```

B. Gerar sinais PWM

Considerando que lido o valor do potenciomêtro, convido na faixa de 0 a 100 e salvo em uma variável é feito a cada 1 segundo será aumentando ou diminuido a velocidade do motor até que atinja a velocidade desejada. É utilizado a variável currentdutycycle para guardar o valor atual dos motores.

TABLE II
PWM

```
{
        // Incrementar a largura de pulso em
            passos de 10% por segundo
        if (__HAL_TIM_GET_FLAG(&htim1,
            TIM_FLAG_UPDATE)){ //para atualizar a
            cada 1 segundo
                __HAL_TIM_CLEAR_FLAG(&htim1,
                    TIM_FLAG_UPDATE);
                if (current_duty_cycle <
                    duty_cycle) {
                        current_duty_cycle += 10;
                        if (current_duty_cycle >
                            duty_cycle) {
                                current_duty_cycle
                                    = duty_cycle;
                        }
                } else if (current_duty_cycle >
                    duty_cycle) {
                        current_duty_cycle -= 10;
                        if (current_duty_cycle <
                            duty_cycle) {
                                current_duty_cycle
                                    = duty_cycle;
                        }
                }
                __HAL_TIM_SET_COMPARE(&htim4,
                    TIM_CHANNEL_1, (htim4.Init.
                    Period * current_duty_cycle) /
                     100);
                __HAL_TIM_SET_COMPARE(&htim4,
                    TIM_CHANNEL_2, (htim4.Init.
                    Period * current_duty_cycle) /
                     100);
                __HAL_TIM_SET_COMPARE(&htim4,
                    TIM_CHANNEL_3, (htim4.Init.
                    Period * current_duty_cycle) /
                     100);
                __HAL_TIM_SET_COMPARE(&htim4,
                    TIM_CHANNEL_4, (htim4.Init.
                    Period * current_duty_cycle) /
                     100);
        }
}
```

C. Controlar Saídas

O motores estão conectados nas portas PB6, PB7, PB8 E PB9 que possuem o TIMER4, sendo respectivamente o canal 1,2,3 e 4. É utilizado a função __HAL_TIM_SET_COMPARE para controlar de PWM. As linhas de código com essa função configuram o valor de comparação para o canal x do Timer 4, o que afeta o ciclo de trabalho de um sinal PWM gerado por esse timer. O valor do ciclo de trabalho é baseado na variável currentdutycycle, o resultado é ajustado para o intervalo permitido pelo hardware (geralmente 0 a htim4.Init.Period) e a divisão por 100 é realizada para converter o ciclo de trabalho de uma representação percentual (0 a 100) para a escala utilizada pelo timer.

TABLE III
SAÍDAS

```
{
        __HAL_TIM_SET_COMPARE(&htim4,
            TIM_CHANNEL_1, (htim4.Init.
            Period * current_duty_cycle
            ) / 100);
        __HAL_TIM_SET_COMPARE(&htim4,
            TIM_CHANNEL_2, (htim4.Init.
            Period * current_duty_cycle
            ) / 100);
        __HAL_TIM_SET_COMPARE(&htim4,
            TIM_CHANNEL_3, (htim4.Init.
            Period * current_duty_cycle
            ) / 100);
        __HAL_TIM_SET_COMPARE(&htim4,
            TIM_CHANNEL_4, (htim4.Init.
            Period * current_duty_cycle
            ) / 100);
}
```

D. Encoder

TABLE IV
ENCODER

```
void delay_ms(uint32_t ms) {
    uint32_t start_time = millis;
    while ((millis - start_time) < ms) {
        // Aguarda at  que o tempo
            desejado seja alcan ado
    }
}

void TIM3_IRQHandler(void) {
    if (TIM3->SR & TIM_SR_UIF) {
        // Limpa a flag de interrup  o
            do Timer 3
        TIM3->SR &= ~TIM_SR_UIF;

        // Incrementa o contador de
            milissegundos
        millis++;
    }
}

void init_delay_timer(void) {
    // Ativar o clock do Timer 3
    RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;

    // Configurar o Timer 3 para gerar
        interrup  o a cada 1 segundo
    TIM3->PSC = 7199;  // Prescaler para
        contar a cada 1 segundo
    TIM3->ARR = 9999;  // Valor de
        recarga para contar at  9999 (1
        segundo)

    // Habilitar interrup  o de estouro
        (UIF) do Timer 3
    TIM3->DIER |= TIM_DIER_UIE;

    // Configurar a prioridade da
        interrup  o
    NVIC_SetPriority(TIM3_IRQn, 0);
    NVIC_EnableIRQ(TIM3_IRQn);

    // Iniciar o Timer 3
    TIM3->CR1 |= TIM_CR1_CEN;
}
```

E. Contar os pulsos de entrada na borda de subida e enviar o valor do contador através da USART

.

TABLE V
USUART

```
{
  RCC->APB2ENR |= (0xFC| (1<<14)); //
      enable GPIO clocks and USART1 clock
  RCC->APB1ENR |= (1<<0);          /* enable
       TIM2 clock */

  usart1_init();                   /*
      initialize the usart1 */

  GPIOA->CRL = 0x44444484;         /* PA1(
      CH2): input pull-up */
  GPIOA->ODR |= (1<<1);

  TIM2->CCMR1 = 0x0000;  /* no filter */
  TIM2->CCER = 0; /* CC2P = 0 (rising) */
  TIM2->SMCR = 0x67;      /* TIM2_CH2 as
      clock source */
  TIM2->ARR = 50000-1; /* count from 0 to
       49999 then roll over to 0 */
  TIM2->CR1 = 1;           /* start counting
       up */

  while(1) {
    usart1_sendInt(TIM2->CNT);  /* send
        the counter value through serial
        */
    usart1_sendStr("\n\r"); /* go to new
        line */
    delay_ms(1000);
  }
}
```