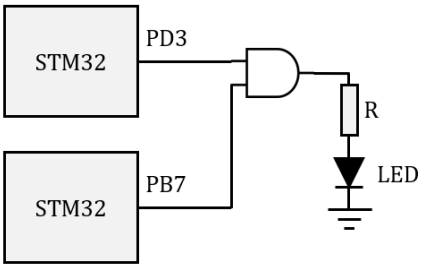
Instituto Federal de Educação, Ciência e Tecnologia - Paraíba (IFPB)
Disciplina: Microprocessadores e microcontroladores.
Professor: Fagner de Araujo Pereira.
Aluno (a): _____

Exercício avaliativo teórico 2 (Peso 50 pontos)

1. Em relação ao microcontrolador STM32F407, como podemos classificar o tipo de computador (RISC ou CISC) e o tipo de arquitetura (Von-Neumann ou Harvard) deste dispositivo?
RISC e Harvard

2. No âmbito dos microcontroladores STM32, explique o que é slew-rate em um pino.
É a velocidade (razão) com a qual a sua tensão de saída de um pino de GPIO pode variar por unidade de tempo, medida em V/µs.

3. Na figura abaixo, é mostrado um diagrama com dois microcontroladores e componentes externos que são ligados na mesma fonte de alimentação. Os pinos apresentados foram previamente configurados como saídas. Os programas 1 e 2 rodam nos microcontroladores e fazem o LED piscar. Considere que o *Programa 2* começa a ser executado com um atraso de 80µs em relação ao *Programa 1*. Nessas condições, responda ao que se pede.




```
//Programa 1
while(1)
{
   GPIOD->ODR &= ~(1 << 3);
   Delay_us(2800);
   GPIOD->ODR |= (1 << 3);
   Delay_us(200);
}
```

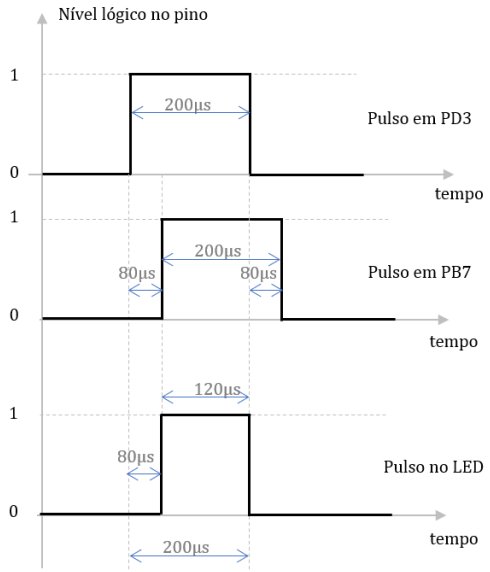
```
//Programa 2
while(1)
{
   GPIOB->ODR &= ~(1 << 7);
   Delay_us(4800);
   GPIOB->ODR |= (1 << 7);
   Delay_us(200);
}
```

a) Com qual frequência o LED piscará?
Analisando os programas, percebemos que o *Programa 1* gera pulsos de 200µs a cada 3ms em PD3, enquanto o *Programa 2* também gera pulsos de 200 µs, mas a cada 5ms em PB7. O LED acenderá quando os dois pulsos coincidirem, uma vez que PD3 e PB7 alimentam uma porta AND e a saída só irá a nível lógico alto quando ambas as entradas forem altas. Dessa forma, isso acontecerá em intervalos de 15ms (que corresponde ao menor múltiplo comum entre 3ms e 5ms). Assim, a frequência de piscadas do LED é de:

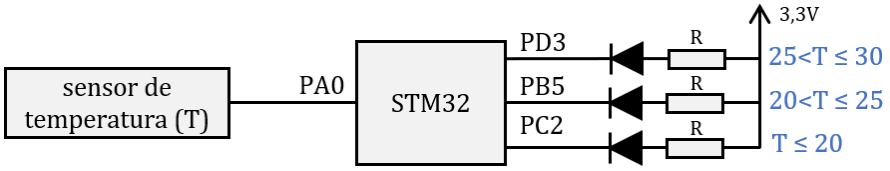
$$f = \frac{1}{T} = \frac{1}{15ms} = 66{,}66Hz.$$

b) Qual a largura do pulso aplicada sobre o LED?
A largura do pulso corresponde ao tempo em que ambas as saídas (PD3 e PB7) se encontram em nível alto. A largura dos pulsos em PD3 e PB7é de 200 µs, mas o pulso em PB7 só inicia 80µs depois, conforme figura abaixo. Além disso, o pulso em PD3 também encerra 80µs antes do pulso em PB7. Assim, as saídas PD3 e PB7 só estarão simultaneamente altas no intervalo de 80µs após o início do pulso em PD3 até o final do seu pulso, isto é:

$$largura = 200\mu s - 80\mu s = 120\mu s.$$



4. No diagrama abaixo, o sensor de temperatura (T) fornece, a cada segundo, uma resposta digital por meio de um pulso cuja largura é variável (PWM). A largura do pulso pode variar de 1ms a 50ms, em passos de 1ms, quando a temperatura varia de 20 °C a 80 °C. Escreva o esboço de um programa, COMENTADO, que faça a leitura do sensor e acione os LEDs de acordo com suas descrições.



Primeiro, a relação entre a largura do pulso e a temperatura deve ser determinada. Temos uma variação de 60 °C (faixa de 20 °C a 80 °C) representada por uma variação de 49ms (faixa de 1ms a 50ms). Isso indica uma relação de $\frac{60}{49}$ °C/ms.

Como a menor largura do pulso é de 1ms e a menor temperatura é de 20 °C, devemos medir a largura do pulso fornecida pelo sensor, subtrair 1ms, multiplicar pela relação encontrada e somar 20 °C ao resultado. Matematicamente, temos:

$$temperatura = (largura - 1ms) * \frac{60\ °C}{49\ ms} + 20\ °C$$

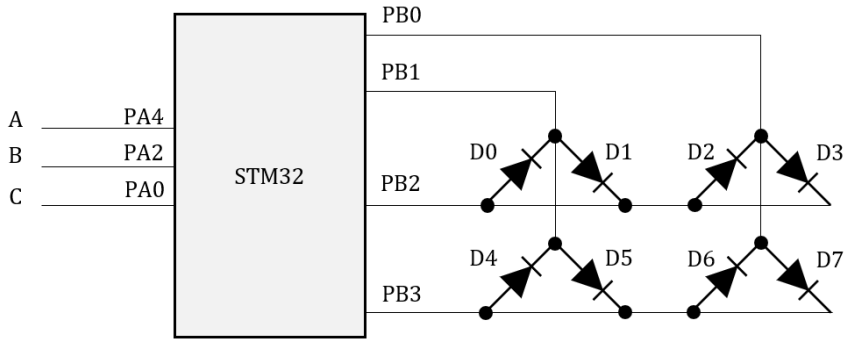
Com isso, uma solução possível é a mostrada no código abaixo:

```
while(1)      //laço infinito
{
   int largura=0;          //variável que mede a largura do pulso
   if(GPIOA->IDR & 1)      //verifica se PA0 é alto, o que marca o início do pulso
   {
      while(GPIOA->IDR & 1)      //permanece nesse laço durante o pulso
      {
         Delay_ms(1);
         ++largura;              //incrementa a variável após cada intervalo de 1ms
      }
      //na saída do while, a variável contém a largura do pulso, em ms

      float temperatura=(largura-1)*(15/59) + 15; //calcula a temperatura

      //testa os limites e acende apenas o LED correspondente
      if(temperatura <= 20)
      {
         GPIOD-ODR |= (1 << 3);
         GPIOB-ODR |= (1 << 5);
         GPIOC-ODR &= ~(1 << 2); //acende o LED em PC2
      }
      if((temperatura > 20) && (temperatura <= 25))
      {
         GPIOD-ODR |= (1 << 3);
         GPIOB-ODR &= ~(1 << 5); //acende o LED em PB5
         GPIOC-ODR |= (1 << 2);
      }
      if((temperatura > 25))
      {
         GPIOD-ODR &= ~(1 << 3); //acende o LED em PD3
         GPIOB-ODR |= (1 << 5);
         GPIOC-ODR |= (1 << 2);
      }
   }
}
```

5. Um pino configurado como saída em dreno aberto (*open drain*) é capaz de fornecer 0V quando acionado com nível lógico baixo, mas não é capaz de fornecer tensão quando acionado com nível lógico alto. Ao invés disso, o transistor de saída do pino fica com o terminal dreno em aberto (daí o nome dreno aberto), o que faz com que não chegue qualquer tensão ao pino, que se comporta como um circuito aberto. A partir desse conceito, observe o diagrama abaixo que mostra um microcontrolador recebendo um código de 3 bits (ABC, com C sendo o bit menos significativo) e acionando 8 LEDs (D0 a D7) a partir de 4 pinos de saída. Escreva um programa, COMENTADO, que faça a leitura do código de 3 bits e acione o LED respectivo ao código.



Para cada código, os dois pinos que acionam o LED devem ser configurados como saídas *push-pull* e os outros dois pinos devem ser configurados como saídas *open drain* em nível alto *(*ou como entradas*)* para não fornecerem tensão aos outros LEDs. Assim, uma solução possível é a mostrada no código abaixo:

```
while(1)       //laço infinito
{
   //lê os pinos de entrada e organiza os bits para formar o código ABC;
   int codigo = ((GPIOA->IDR & (1<<4)) >> 2) | ((GPIOA->IDR & (1<<2)) >> 1) | (GPIOA->IDR & 1);

   //testa o código e acende o LED correspondente
   switch(código)
   {
      case 0:  //acende D0
         GPIOB->OTYPER &= ~((1<<2) | (1<<1));       //PB2 e PB1 com saída push-pull
         GPIOB->OTYPER |=  ((1<<0) | (1<<3));       //PB0 e PB3 com saída open drain
         GPIOB->ODR |= ((1<<2) | (1<<0) | (1<<3));  //nível alto em PB0, PB2 e PB3
         GPIOB->ODR &= ~(1<<1);                     //nível baixo em PB1
         break;

      case 1:  //acende D1
         GPIOB->OTYPER &= ~((1<<2) | (1<<1));       //PB2 e PB1 com saída push-pull
         GPIOB->OTYPER |=  ((1<<0) | (1<<3));       //PB0 e PB3 com saída open drain
         GPIOB->ODR |= ((1<<1) | (1<<0) | (1<<3));  //nível alto em PB0, PB1 e PB3
         GPIOB->ODR &= ~(1<<2);                     //nível baixo em PB2
         break;

      case 2:  //acende D2
         GPIOB->OTYPER &= ~((1<<2) | (1<<0));       //PB2 e PB0 com saída push-pull
         GPIOB->OTYPER |=  ((1<<1) | (1<<3));       //PB1 e PB3 com saída open drain
         GPIOB->ODR |= ((1<<2) | (1<<1) | (1<<3));  //nível alto em PB2, PB1 e PB3
         GPIOB->ODR &= ~(1<<0);                     //nível baixo em PB0
         break;

      ...
      ...

      case 6:  //acende D6
         GPIOB->OTYPER &= ~((1<<0) | (1<<3));       //PB0 e PB3 com saída push-pull
         GPIOB->OTYPER |=  ((1<<1) | (1<<2));       //PB1 e PB2 com saída open drain
         GPIOB->ODR |= ((1<<3) | (1<<2) | (1<<1));  //nível alto em PB3, PB2 e PB1
         GPIOB->ODR &= ~(1<<0);                     //nível baixo em PB0
         break;


      case 7:  //acende D7
         GPIOB->OTYPER &= ~((1<<0) | (1<<3));       //PB0 e PB3 com saída push-pull
         GPIOB->OTYPER |=  ((1<<1) | (1<<2));       //PB1 e PB2 com saída open drain
         GPIOB->ODR |= ((1<<0) | (1<<1) | (1<<2));  //nível alto em PB0, PB1 e PB2
         GPIOB->ODR &= ~(1<<3);                     //nível baixo em PB3
         break;
   }
}
```