SISTEMAS REATIVOS

TRABALHO 1 – RELÓGIO 24H C/ DESPERTADOR

CARLOS LEANDRO – 1122002

LUCAS SANTOS - 1412033

```
class Relogio {
  private:
    byte hora, minuto;

  public:
    Relogio() {
      hora = 0;
      minuto = 0;
    }
    byte get_hora() {···
    }
    void set_hora(byte hora) {···
    }
    byte get_minuto() {···
    }
    void set_minuto(byte minuto) {···
    }
    void incrementar_hora() {···
    }
    void incrementar_minuto() {···
    }
    void clonar(Relogio relogio) {
      hora = relogio.get_hora();
      minuto = relogio.get_minuto();
    }
    byte comparar(Relogio relogio) {
      if (hora == relogio.get_hora()) {
        if (minuto > relogio.get_minuto()) {
          return 1;
        }
        else if (minuto < relogio.get_minuto()) {
          return -1;
        }
      }
      else if (hora > relogio.get_hora()) {
        return 1;
      }
      else if (hora < relogio.get_hora()) {
        return -1;
      }
      return 0;
    }
    void avancar_relogio() {
      incrementar_minuto();
      if (get_minuto() == 0) {
        incrementar_hora();
      }
    }
};
```

A base da aplicação foi o uso da classe relógio.

Este objeto é responsável por armazenar a hora do relógio, além de efetuar todas operações sobre esses horários.

Vale notar que o relógio não é responsável pelo estado do sistema como um todo.

```
//// Volateis
volatile unsigned long salvar_estado;
volatile boolean relogio_config, alarme_config, alarme_ativo;

//// Não volateis
Relogio relogio, alarme, temporario;
unsigned long debounce_delay, debounce_last_time, intervalo_despertador, ultimo_incremento;
/* Segment byte maps for numbers 0 to 9 */
const byte SEGMENT_MAP[] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0X80, 0X90};
/* Byte maps to select digit 1 to 4 */
const byte SEGMENT_SELECT[] = {0xF1, 0xF2, 0xF4, 0xF8};
boolean despertando;
```

Com exceção das variáveis responsáveis pelo display lcd, aqui temos os responsáveis pelo controle de estado do sistema.

As variáveis terminadas em config, definem o que deve ser exibido no display e qual o tratamento esperado para cada um dos botões.

Temos duas instancias de relógio para o relógio em si e o alarme, que pode ser pensado como um relógio em que não iremos passar as horas, além de um auxiliar.

As demais variáveis são uma configuração geral utilizadas para tratar o debounce, espaço entre apitos do buzzer, entre outros.

```
ISR (PCINT1_vect) {
  if ((millis() - debounce_last_time) > debounce_delay) {
    debounce_last_time = millis();
    salvar_estado = millis();
    /*
      Caso o botão 1 seja apertado fora do estado de configuração, entrará no estado de configuração do relogio;
      Caso contrario, incrementará os minutos do que está sendo configurado no momento.
    */
    if (digitalRead(KEY1) == 0) {
      if (!relogio_config && !alarme_config) {···
      }
      else if (relogio_config) {···
      }
      else if (alarme_config) {···
      }
    }
    /*
      Caso o botão 2 seja apertado fora do estado de configuração, entrará no estado de configuração do alarme;
      Caso contrario, incrementará os minutos do que está sendo configurado no momento.
    */
    else if (digitalRead(KEY2) == 0) {
      if (!relogio_config && !alarme_config) {···
      }
      else if (relogio_config) {···
      }
      else if (alarme_config) {···
      }
    }
    /*
      Caso o botão 3 seja apertado no estado de configuração, todas as configurações serão descartadas;
      Caso contrario, ativará ou desativará o alarme, acendendo ou apagando o LED4, respectivamente.
    */
    else if (digitalRead(KEY3) == 0) {
      if (relogio_config || alarme_config) {···
      }
      else {
        alarme_ativo = !alarme_ativo;
        if (alarme_ativo) {···
        }
        else {···
        }
      }
    }
  }
}
```

Este método é responsável por tratar as interrupções geradas pelos botões.

Nele podemos ver os tratamento diferentes dependendo de como está o estado do sistema.

```
/*
  Transiciona os possiveis estados do visor, configuração do relogio e do alarme, avança o relogio e verifica o alarme;
  Caso nennhum botão seja apertado num periodo de 2 segundos, o estado temporario é consolidado, as variaveis resetadas e o respectivo LED apagado.
*/
void loop() {
  if (relogio_config) {
    if ((millis() - salvar_estado) > 2000) {
      relogio.set_hora(temporario.get_hora());
      relogio.set_minuto(temporario.get_minuto());
      resetar_configs();
      ultimo_incremento = millis(); // Começa a rolar o tempo a partir do salvamento do novo estado do relogio
      Serial.println("Configuracoes do relogio salvas com sucesso!");
      digitalWrite(LED1, HIGH);
    }
    else {
      exibir(temporario);
    }
  }
  else if (alarme_config) {
    if ((millis() - salvar_estado) > 2000) {
      alarme.set_hora(temporario.get_hora());
      alarme.set_minuto(temporario.get_minuto());
      resetar_configs();
      Serial.println("Configuracoes do alarme salvas com sucesso!");
      digitalWrite(LED2, HIGH);
    }
    else {
      exibir(temporario);
    }
  }
  else {
    avancar_relogio();
    verificar_alarme();
    exibir(relogio);
  }
}
```

No loop principal temos a definição do que é exibido para cada estado em que o sistema se encontra.

Caso não esteja em uma tela de configuração, o último else incrementa o relógio e verifica o