Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Apucarana
Bacharelado em Engenharia da Computação

TRABALHO DE SISTEMAS EMBARCADOS

ATIVIDADE 2

Alarme com máquina de estados

Gabriel Finger Conte[1]
Maria Eduarda Pedroso[2]

APUCARANA
MAIO / 2024

[1] gabcon@alunos.utfpr.edu.br
[2] mpedroso@alunos.utfpr.edu.br

SUMÁRIO

1. INTRODUÇÃO

Atualmente, uma ampla gama de Sistemas Embarcados está disponível e em uso para diversas finalidades específicas. Estes sistemas combinam hardware e software especializados, dedicados a uma única aplicação que requer execução contínua. Em muitos casos, é necessário manter uma referência temporal precisa, seja para registrar eventos com precisão para análise posterior, seja para sincronizar relógios digitais após ajustes iniciais.

Para garantir essa precisão, é comum usar interrupções de hardware acionadas por timers internos em dispositivos, como a biblioteca TimerInterrupt desenvolvida por khoih-prog (2022) para a Arduino IDE. Esses timers internos costumam ter alta precisão devido à sua frequência mantida por cristais de quartzo que oscilam em uma frequência fixa.

As placas de prototipagem, como a Arduino ATMega 2560 mencionada neste trabalho, facilitaram muito o desenvolvimento e teste de aplicações sem custos elevados ou a necessidade de recriar o hardware.

Além disso, muitos sistemas requerem uma interface de comunicação com o usuário, como as telas LCD (Liquid Crystal Display), que se tornaram populares devido à sua compatibilidade com as placas de prototipagem. Para garantir sua portabilidade, várias bibliotecas, como a LiquidCrystal disponibilizada pelo Arduino (2024), foram desenvolvidas.

Este trabalho, uma avaliação parcial para a disciplina de Sistemas Embarcados, apresenta um relógio digital projetado com uma placa Arduino ATMega 2560, utilizando a biblioteca TimerInterrupt para controle de tempo e a biblioteca LiquidCrystal para exibição de horário, alarme e menu de configuração. Utilizando os mecanismos de comunicação serial próprios do Arduino (c2024)

2. DESCRIÇÃO DO PROBLEMA

O projeto proposto visa a implementação de um sistema de alarme utilizando o Arduino. O sistema deve permitir aos usuários ajustarem tanto o horário atual quanto o horário do alarme, utilizando um joystick e um botão. O principal objetivo é criar um dispositivo que exiba o horário atual e o horário do alarme em um display LCD, além de acionar um buzzer quando o horário atual coincidir com o horário do alarme.

Para resolver esse problema, é necessário o uso de máquinas de estados para controlar o fluxo do programa, garantindo que as diferentes funcionalidades sejam executadas de forma organizada e eficiente. O projeto requer a integração de vários componentes, incluindo o Arduino, um display LCD, um joystick, um botão e um buzzer, e a coordenação entre esses componentes para atingir os objetivos de forma satisfatória.

3. EXEMPLO DE APLICAÇÃO

As máquinas de estados são amplamente utilizadas em sistemas embarcados para controlar o fluxo de execução e gerenciar o comportamento dos dispositivos de forma eficiente e organizada. Abaixo, apresentamos três exemplos de sistemas embarcados reais que fazem uso de máquinas de estados:

Os sistemas de automação residencial, como sistemas de controle de iluminação, termostatos e sistemas de segurança, frequentemente empregam máquinas de estados para gerenciar diferentes modos de operação. Por exemplo, um termostato inteligente pode ter estados para aquecimento, resfriamento e modo de economia de energia. Transições entre esses estados são acionadas com base em leituras de sensores de temperatura e ajustes feitos pelo usuário, permitindo que o termostato regule automaticamente a temperatura ambiente de acordo com as necessidades.

Em sistemas de automação industrial, as máquinas de estados são utilizadas para controlar o funcionamento de máquinas, linhas de produção e processos de fabricação. Por exemplo, em uma linha de montagem automatizada, diferentes estados podem representar etapas específicas do processo de produção, como carregamento, montagem, inspeção e descarregamento. As transições entre esses estados são acionadas por sensores de presença, detecção de falhas ou comandos de controle enviados por um sistema de supervisão, garantindo um fluxo de produção eficiente e seguro.

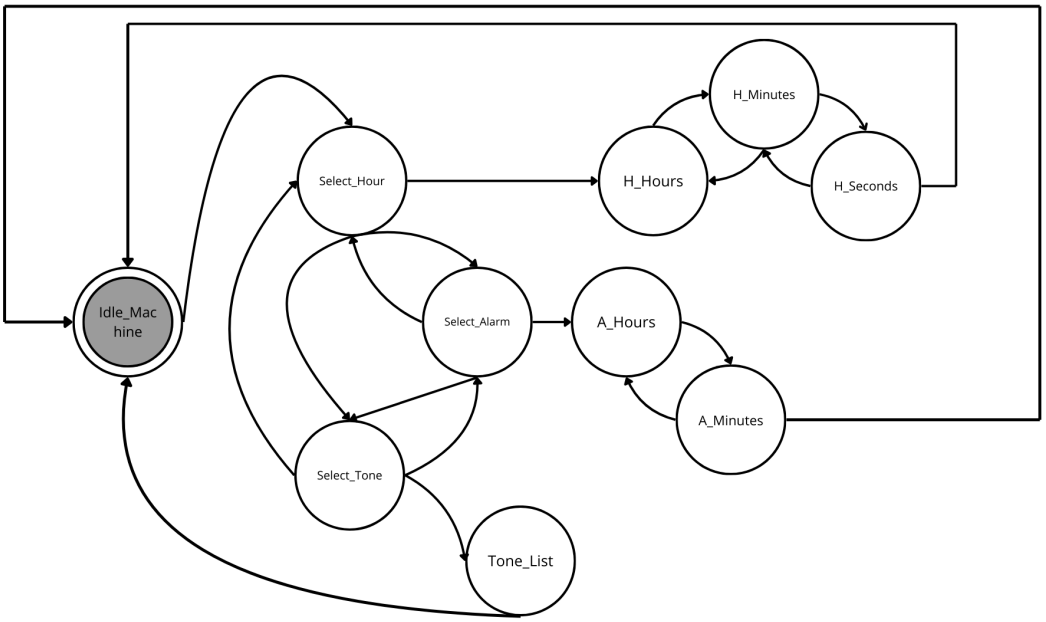
Os veículos autônomos dependem fortemente de máquinas de estados para controlar seu comportamento e tomar decisões em tempo real. Um sistema de controle de veículo autônomo pode ter estados para diferentes modos de direção, como condução em linha reta, mudança de faixa e frenagem de emergência. Além disso, pode haver estados para monitoramento de obstáculos, detecção de sinais de trânsito e interação com outros veículos. As transições entre esses estados são baseadas em dados de sensores, como câmeras, radares e lidar, e em algoritmos de percepção e tomada de decisão, permitindo que o veículo autônomo navegue com segurança em diversos ambientes e situações de tráfego.

Esses exemplos ilustram a versatilidade e a importância das máquinas de estados em uma variedade de sistemas embarcados, desde aplicações domésticas até sistemas críticos de segurança e automação. A utilização de máquinas de estados permite a implementação de sistemas complexos de forma estruturada e eficiente, contribuindo para o desenvolvimento de soluções robustas e confiáveis.

4. DESENVOLVIMENTO

Para desenvolver esse projeto inicialmente foi definido quais estados nosso projeto necessitava para funcionar corretamente, e também quais seriam os próximos estados que o usuário poderia se deslocar de um determinado estado, abaixo temos uma imagem de como ficou os estados do nosso sistema.

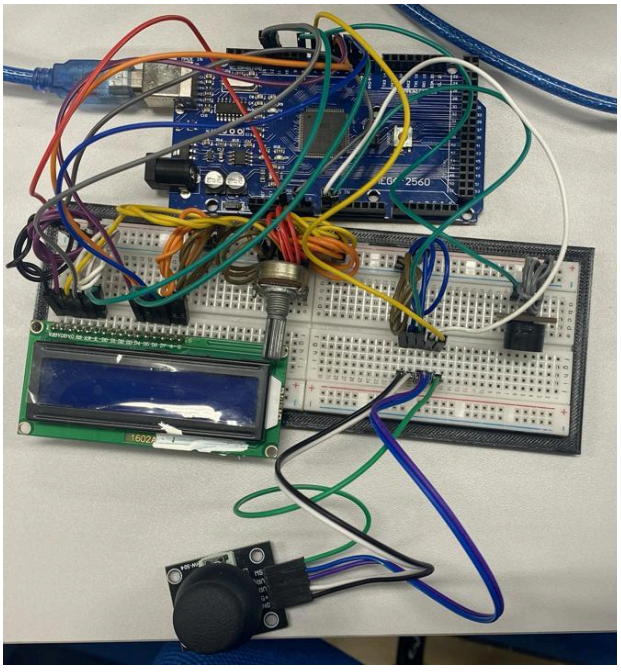
Figura 4.1 - Máquina de estados



Fonte: autoria própria

Além de pensar na parte do código e seus estados, foi necessário projetar o circuito integrando todos os componentes eletrônicos de forma correta com a placa, já inserindo os pinos corretamente que serão utilizados no código.

Figura 4.2 - Circuito implementado



Fonte: autoria própria

4.1. Configurando o Joystick

Antes de iniciar o desenvolvimento do sistema, precisamos identificar e compreender o funcionamento do módulo de Joystick analógico KY-023, disponibilizado para utilização na prática.

Tal módulo trata-se de um Joystic bi-direcional, permitindo assim movimentações tanto verticais quanto horizontais, bem como um botão integrado ao apertar o controle. A fim de controlar o ruído padrão de debouncing do push-button interno, foi utilizado uma variável auxiliar, que mantém o último instante que o botão foi pressionado.

Caso o mesmo seja acionado novamente num intervalo de tempo muito pequeno, provavelmente é por causa do ruído, assim manteve-se um intervalo mínimo para que um novo acionamento do mesmo fosse considerado.

Uma lógica semelhante foi aplicada para controlar a sensibilidade de mudança dos valores do Joystick. Mas ao invés de considerar um intervalo de tempo mínimo para atualizar a direção do controle, tal intervalo controla a velocidade que os valores dependentes dele serão atualizados.

A lógica geral implementada para o Joystick fora separada do arquivo principal, organizada em uma biblioteca denominada “joystickConfig.h”, a qual pode ser analisada minuciosamente no Anexo I do presente relatório. Todavia, a lógica geral do mesmo pode ser conferida no pseudo-código abaixo.

```
Unset
Definição de constantes:
	- JOY_X_PIN: Pino para leitura do eixo X do joystick
	- JOY_Y_PIN: Pino para leitura do eixo Y do joystick
	- JOY_BUTTON_PIN: Pino para leitura do botão do joystick
	- JOYSTIC_DEBOUNCE_INTERVAL: Intervalo de debounce para o joystick
```

```
        - JOYSTIC_DEBOUNCE_INTERVAL_MENU: Intervalo de debounce para o
        joystick no menu

Variáveis globais:
        - lastXdebounce: Último momento de debounce do eixo X do joystick
        - lastYdebounce: Último momento de debounce do eixo Y do joystick
        - lastButtonState: Último estado do botão do joystick

Enumeração de estados para o botão:
        - Idle: Estado ocioso do botão
        - Pressed: Estado pressionado do botão
        - Released: Estado liberado do botão

Enumeração de estados para o joystick:
        - Neutral: Estado neutro do joystick
        - Up: Estado para cima do joystick
        - Down: Estado para baixo do joystick
        - Left: Estado para a esquerda do joystick
        - Right: Estado para a direita do joystick

Classe Button:
        Atributos:
                - pin: Pino de leitura do botão
                - state: Estado atual do botão
                - lastDebounceTime: Último momento de debounce do botão
                - debounceDelay: Intervalo de debounce do botão
        Métodos:
                - Construtor Button(pin): Inicializa o objeto Button com o pino
                especificado e configurações padrão.
                - update(): Atualiza o estado do botão de acordo com a leitura
                do pino e o debounce.
                - getState(): Obtém o estado atual do botão.

Classe Joystick:
        Atributos:
                - xPin: Pino de leitura do eixo X do joystick
```

```
        - yPin: Pino de leitura do eixo Y do joystick
    Métodos:
        - Construtor Joystick(xPin, yPin): Inicializa o objeto Joystick
        com os pinos especificados.
        - getState(): Obtém o estado atual do joystick com base nos
        valores lidos nos pinos X e Y.
```

4.2. Configurando o Timer

Um relógio necessita manter uma base de tempo para funcionar propriamente. Como mencionado anteriormente, no presente projeto utilizou-se especificamente o Timer 1 interno a placa Arduino ATMega 2560.

Para sua configuração, utilizou-se a biblioteca TimerInterrupt desenvolvida por khoih-prog (2022), adaptando o código do seu exemplo TimerInterrupTest. Para tanto, separou-se a lógica em uma biblioteca própria, de modo que:

- “timerConfig.h”: Concentrou-se as definições básicas para utilização do timer, constantes, importação da biblioteca e definição dos cabeçalhos das funções para habilitar, desabilitar, e a função de callback para as interrupções do timer.
- “timerConfig.ino”: Onde foram implementadas de fato as funções. Em especial, a função de callback concentra a lógica de atualização da referência temporal do dispositivo, bem como o controle da frequência de atualização do display LCD. Podendo sua lógica ser melhor visualizada no Anexo I do presente documento, explicada também de maneira simplificada no pseudo-código abaixo:

```
Unset
Função updateCurrentTime():
	- Incrementa a variável currentTick a cada interrupção do timer a 10Hz.
	- Habilita a atualização do LCD (lcdState = true).

	- A cada 10 ticks:
		* Reseta currentTick para 0.
		* Incrementa o segundo atual.
	- A cada 60 segundos:
		* Reseta currentSeconds para 0.
		* Incrementa o minutos atual.
	- A cada 60 minutos:
		* Reseta currentMinute para 0.
		* Incrementa a hora atual.
	- A cada 24 horas:
		* Reseta a hora atual para 0.
```

4.3. Configurando a Máquina de Estados e Alterando a Base de Tempo

Com a base de tempo configurada e o joystick calibrado, passamos para a construção da lógica interna do relógio. Para tanto, como solicitado pelo professor, utilizou-se o conceito de máquina de estados, culminando nos seguintes estados:

- *Idle_Machine*: estado padrão da máquina, onde apenas se mostra o horário atual e o horário definido para o alarme. Caso clicado no botão do joystick, entrará no menu de configurações, passando para o estado *Select_Hour*.
- *Select_Hour*: estado inicial quando entra-se no menu de configurações, caso selecionado passa para o estado *H_Hours*.
- *H_Hours*: estado inicial de configuração da base de tempo do relógio, utilizando o joystick para cima ou para baixo, modifica-se o horário atual incrementando ou decremento o mesmo respectivamente. Caso mova-se o

mesmo para a direita, passará para o estado *H_Minutes*. Por fim, caso aperte novamente o botão, irá manter as alterações e voltar para o *Idle_Machine*.

- *H_Minutes*: permite alterar os minutos da base de tempo do relógio, seguindo o mesmo padrão do *H_Hours*. Caso mova-se o joystick para a esquerda, voltará para o estado *H_Hours*, caso mova-o para a direita passará para o estado *H_Seconds*. Por fim, caso aperte novamente o botão, irá manter as alterações e voltar para o *Idle_Machine*.
- *H_Seconds*: permite alterar os segundos da base de tempo do relógio, seguindo o mesmo padrão do *H_Hours*. Caso mova-se o joystick para a esquerda, voltará para o estado *H_Minutes*. Por fim, caso aperte novamente o botão, irá manter as alterações e voltar para o *Idle_Machine*.
- *Select_Alarm*: estado secundário do menu de configurações, caso selecionado passa para o estado *A_Hours*.
- *A_Hours*: segue a mesma lógica do *H_Hours*, todavia altera a base de tempo do alarme.
- *A_Minutes*: segue a mesma lógica do *H_Minutes*, todavia altera a base de tempo do alarme e não há ação caso mova o joystick para a direita.
- *Select_Tone*: estado terciário do menu de configurações, que caso selecionado passa para o estado *Tone_List*.
- *Tone_List*: estado que permite o usuário escolher o toque do alarme, caso aperte novamente o botão, irá manter as alterações e voltar para o *Idle_Machine*.

A fim de configurar tal comportamento, fora modularizado o mesmo em diferentes funções para facilitar a compreensão e manutenção do código. Assim, para a atualização dos estados propriamente dita, implementou-se a função descrita pelo pseudo-código abaixo, presente no arquivo principal do sistema desenvolvido, disponível no Anexo I do presente.

```
Unset
Função atualizaEstado(pressedButton, joystate):
    Variáveis auxiliares:
       - currXdebounce: Tempo decorrido desde o último debounce do eixo X do
       joystick.
       - currYdebounce: Tempo decorrido desde o último debounce do eixo Y do
       joystick.
Verifica o estado atual do sistema (estadoAtual):
       - Se estadoAtual for Idle_Machine:
             * Se o botão estiver pressionado e seu estado anterior for 1
             (HIGH), muda para Select_Hour.
       - Se estadoAtual for Select_Hour:
             * Se o botão estiver pressionado e seu estado anterior for 1
             (HIGH), muda para H_Hours.
             * Se o joystick estiver sendo movido para cima e o tempo de
             debounce for atingido, muda para Select_Tone.
             * Se o joystick estiver sendo movido para baixo e o tempo de
             debounce for atingido, muda para Select_Alarm.
       - Se estadoAtual for H_Hours, H_Minutes, ou H_Seconds:
             * Se o botão estiver pressionado e seu estado anterior for 1
             (HIGH), retorna para Idle_Machine.
             * Chama a função adjustTime(joystate) para ajustar o tempo do
             relógio.
       - Se estadoAtual for Select_Alarm:
             * Se o botão estiver pressionado e seu estado anterior for 1
             (HIGH), muda para A_Hours.
             * Se o joystick estiver sendo movido para cima e o tempo de
             debounce for atingido, retorna para Select_Hour.
             * Se o joystick estiver sendo movido para baixo e o tempo de
             debounce for atingido, muda para Select_Tone.
       - Se estadoAtual for A_Hours, A_Minutes, ou A_Seconds:
             * Se o botão estiver pressionado e seu estado anterior for 1
             (HIGH), retorna para Idle_Machine.
             * Chama a função adjustAlarmTime(joystate) para ajustar o tempo
             do alarme.
       - Se estadoAtual for Select_Tone:
```

```
            * Se o botão estiver pressionado e seu estado anterior for 1
            (HIGH), muda para Tone_List.
            * Se o joystick estiver sendo movido para baixo e o tempo de
            debounce for atingido, retorna para Select_Hour.
            * Se o joystick estiver sendo movido para cima e o tempo de
            debounce for atingido, muda para Select_Alarm.
      - Se estadoAtual for Tone_List:
            * Se o botão estiver pressionado e seu estado anterior for 1
            (HIGH), retorna para Idle_Machine.
            * Chama a função adjustTone(direction) para selecionar um dos
            toques disponíveis.
      - Caso contrário, se o estadoAtual não for reconhecido, imprime um
      erro no console serial e reseta para Idle_Machine.
```

No mesmo arquivo, implementou-se a função responsável por atualizar a base de tempo atual, conforme detalhado no seguinte pseudo-código:

```
Unset
Função adjustTime(direction):

Variáveis auxiliares:
      - currXdebounce: Tempo decorrido desde o último debounce do eixo X do
      joystick.
      - currYdebounce: Tempo decorrido desde o último debounce do eixo Y do
      joystick.
Verifica o estado atual do sistema (estadoAtual) relacionado ao ajuste do
horário:
- Se estadoAtual for H_Hours (ajuste da hora):
      * Se direction for Up (para cima), incrementa a hora se o tempo de
      debounce for atingido.
      * Se direction for Down (para baixo), decrementa a hora se o tempo de
      debounce for atingido.
      * Se direction for Right (para a direita), muda para o ajuste dos
      minutos se o tempo de debounce do menu for atingido.
```

```
- Se estadoAtual for H_Minutes (ajuste dos minutos):
      * Se direction for Up (para cima), incrementa os minutos se o tempo de
      debounce for atingido.
      * Se direction for Down (para baixo), decrementa os minutos se o tempo
      de debounce for atingido.
      * Se direction for Left (para a esquerda), volta para o ajuste da hora
      se o tempo de debounce do menu for atingido.
      * Se direction for Right (para a direita), muda para o ajuste dos
      segundos se o tempo de debounce do menu for atingido.
- Se estadoAtual for H_Seconds (ajuste dos segundos):
      * Se direction for Up (para cima), incrementa os segundos se o tempo
      de debounce for atingido.
      * Se direction for Down (para baixo), decrementa os segundos se o
      tempo de debounce for atingido.
      * Se direction for Left (para a esquerda), volta para o ajuste dos
      minutos se o tempo de debounce do menu for atingido.
- Se estadoAtual for Idle_Machine (máquina ociosa), não faz nada.
- Caso contrário (estado desconhecido), exibe um erro no console serial,
reseta para Idle_Machine e retorna da função.
```

4.4. Configurando o Alarme

Seguindo a mesma lógica para atualização da base de tempo do relógio, construiu-se a função para atualização do alarme. Todavia, essa lógica fora separada em uma biblioteca distinta, denominada “alarmConfig.h”.

Nela definiu-se o alarme como uma classe, como descrito pelo pseudo-código abaixo. Enquanto a implementação das funções foram configuradas no arquivo “alarmConfig.ino”.

```
Unset
Classe Alarm:
```

```
        Atributos:
                - hour: Hora do alarme
                - minute: Minuto do alarme
        Métodos:
        - Construtor Alarm(hour, minute): Inicializa o objeto Alarm com a hora
        e minuto especificados.
        - isAlarmTime(currentHour, currentMinute): Verifica se é hora do
        alarme comparando com a hora e minuto atuais.

Função adjustAlarmTime(direction):
Variáveis auxiliares:
        - currXdebounce: Tempo decorrido desde o último debounce do eixo X do
        joystick.
        - currYdebounce: Tempo decorrido desde o último debounce do eixo Y do
        joystick.
Verifica o estado atual do sistema (estadoAtual) relacionado ao ajuste do
alarme:
- Se estadoAtual for A_Hours (ajuste da hora do alarme):
        * Se direction for Up (para cima), incrementa a hora do alarme se o
        tempo de debounce for atingido.
        * Se direction for Down (para baixo), decrementa a hora do alarme se o
        tempo de debounce for atingido.
        * Se direction for Right (para a direita), muda para o ajuste dos
        minutos se o tempo de debounce do menu for atingido.
- Se estadoAtual for A_Minutes (ajuste dos minutos do alarme):
        * Se direction for Up (para cima), incrementa os minutos do alarme se
        o tempo de debounce for atingido.
        * Se direction for Down (para baixo), decrementa os minutos do alarme
        se o tempo de debounce for atingido.
        * Se direction for Left (para a esquerda), volta para o ajuste da hora
        se o tempo de debounce do menu for atingido.
- Se estadoAtual for Idle_Machine (máquina ociosa), não faz nada.
- Caso contrário (estado desconhecido), exibe um erro no console serial,
reseta para Idle_Machine e retorna da função.
```

4.5. Configurando o Buzzer

Para adicionar funcionalidades sonoras ao dispositivo, foram desenvolvidos dois arquivos essenciais: "buzzerConfig.h" e "buzzerConfig.cpp", responsáveis por configurar e controlar o buzzer.

- buzzerConfig.h: Este arquivo de cabeçalho contém todas as definições e configurações necessárias para o funcionamento do buzzer. Nele, são especificados o pino de controle do buzzer, as frequências das notas musicais, as melodias a serem reproduzidas (como as músicas de "The Legend of Zelda" e do "Mario"), e as durações correspondentes a cada nota. Além disso, são declaradas as variáveis e as funções responsáveis por controlar o acionamento do buzzer e selecionar as melodias a serem reproduzidas.
- buzzerConfig.cpp: Neste arquivo de implementação, são definidas as funções declaradas no arquivo de cabeçalho. A função adjustTone() é responsável por ajustar qual a musica e o tom do buzzer com base na entrada do joystick, enquanto a função buzz() é encarregada de tocar as melodias selecionadas no buzzer. Ambas as funções utilizam a função millis() para controlar o tempo de execução e garantir a reprodução correta e em looping das melodias sem necessitar de delay.

Esses arquivos são cruciais para o funcionamento adequado do buzzer no dispositivo, permitindo a reprodução de melodias específicas e proporcionando uma experiência sonora aos usuários.

4.6. Configurando o LCD

Com o intuito de fornecer uma interface visual do sistema para o usuário, utilizou-se o módulo LCD 1602A disponibilizado para a prática.

Para sua configuração, utilizou-se a biblioteca LiquidCrystal mantida pelo Arduino (2022). Para tanto, separou-se a lógica em uma biblioteca própria, de modo que:

- “lcdConfig.h”: Concentrou-se as definições básicas para utilização do lcd, constantes dos pinos, importação da biblioteca e definição do cabeçalho da função para atualização do display.
- “lcdConfig.ino”: Onde foi implementado de fato a função para atualização e exibição no display LCD. Podendo sua lógica ser melhor visualizada no Anexo I do presente documento, explicada também de maneira simplificada no pseudo-código abaixo:

```
Unset
// Declarações de variáveis globais
estadoAtual
currentHour, currentMinute, currentSeconds
alarmHour, alarmMinute
buzzSelect
lcdState

// Função para exibir a hora atual no LCD
função printHour():
  Se estadoAtual é Select_Hour:
    Exibir "*"
  Exibir "Hora: "
  Se currentHour < 10:
    Exibir "0"
  Exibir currentHour
  Exibir ":"
  Se currentMinute < 10:
    Exibir "0"
  Exibir currentMinute
  Exibir ":"
  Se currentSeconds < 10:
    Exibir "0"
```

```
  Exibir currentSeconds
  Se estadoAtual é H_Hours:
    Posicionar cursor em (15, 0)
    Exibir "h"
  Senão se estadoAtual é H_Minutes:
    Posicionar cursor em (15, 0)
    Exibir "m"
  Senão se estadoAtual é H_Seconds:
    Posicionar cursor em (15, 0)
    Exibir "s"

// Função para exibir o horário do alarme no LCD
função printAlarm():
  Se estadoAtual é Select_Alarm:
    Exibir "*"
  Exibir "Alarme: "
  Se alarmHour < 10:
    Exibir "0"
  Exibir alarmHour
  Exibir ":"
  Se alarmMinute < 10:
    Exibir "0"
  Exibir alarmMinute
  Se estadoAtual é A_Hours:
    Posicionar cursor em (15, 1)
    Exibir "h"
  Senão se estadoAtual é A_Minutes:
    Posicionar cursor em (15, 1)
    Exibir "m"
  Senão se estadoAtual é A_Seconds:
    Posicionar cursor em (15, 1)
    Exibir "s"

// Função para exibir a lista de tons de música disponíveis no LCD
função printToneList():
  Se buzzSelect é 1:
```

```
    Exibir "*"
  Exibir "Mario"
  Posicionar cursor em (0, 1)
  Se buzzSelect é 0:
    Exibir "*"
  Exibir "Zelda"

// Função para atualizar o conteúdo exibido no LCD de acordo com o estado
atual do sistema
função updateLCD():
  Se lcdState é verdadeiro:
    lcdState = falso
    Limpar LCD
    Posicionar cursor em (0, 0)
    Se estadoAtual é Tone_List:
      Chamada de função printToneList()
    Senão se estadoAtual é Select_Tone:
      Exibir "*"
      Exibir "Musicas"
    Senão:
      Chamada de função printHour()
      Posicionar cursor em (0, 1)
      Chamada de função printAlarm()
```

5. Configurando o setup() e loop()

A configuração inicial do dispositivo segue o padrão da Arduino IDE e ocorre na função setup() do arquivo principal. Aqui, são estabelecidas diversas operações essenciais: a comunicação serial inicia com uma taxa de 9600 bps para fins de debug, o display LCD é configurado para 16 colunas e 2 linhas, o pino do botão do joystick é definido como entrada com pull-up, e o pino do buzzer é configurado como saída.

Além disso, se o Timer 1 estiver em uso, ele é inicializado com um intervalo específico e associa-se a função TimerHandler como manipulador de interrupção, exibindo mensagens apropriadas para indicação de sucesso ou erro na inicialização. Tal comportamento é simplificado no pseudo-código abaixo:

```
Unset
setup():
      - Inicializa a comunicação serial com taxa de 9600 bps
      - Inicializa o display LCD com 16 colunas e 2 linhas
      - Configura o pino do botão do joystick como entrada com pull-up
      - Configura o pino do buzzer como saída
      - Inicializa o Timer 1 (se estiver em uso) com um intervalo específico
      e associa a função TimerHandler como manipulador de interrupção
```

No loop principal do sistema (loop()), a atualização de diversos elementos fundamentais é realizada. Isso inclui a verificação e atualização do estado do botão do joystick, a obtenção do estado atual do joystick, a atualização do estado do sistema conforme o estado do botão e do joystick.

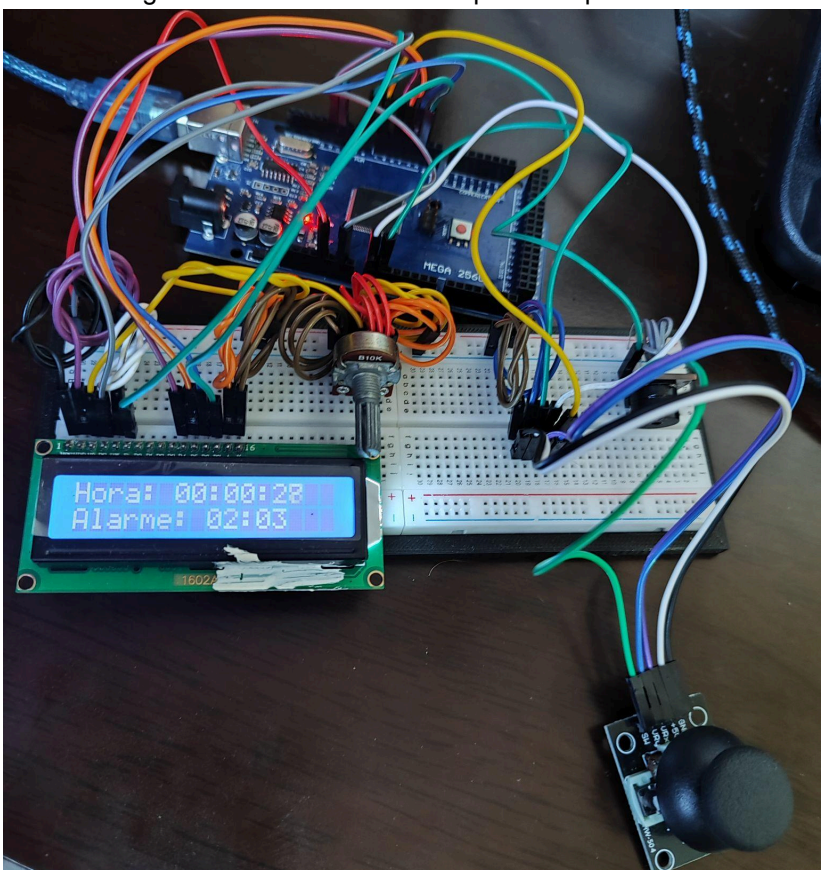
A chamada da função updateLCD() para atualizar o conteúdo do display LCD, a verificação se é a hora de acionar o alarme por meio de um objeto Alarm, e, se necessário, o acionamento do buzzer de acordo com as configurações de hora e alarme previamente definidas. Como no setup, o comportamento da função é simplificado no pseudo-código abaixo, para facilitar a compreensão:

```
Unset
loop():
    - Atualiza o estado do botão do joystick
    - Obtém o estado atual do joystick
    - Atualiza o estado do sistema com base no estado do botão e do joystick
    - Atualiza o conteúdo do display LCD
    - Verifica se é hora de acionar o alarme e aciona o buzzer se necessário
```

6. RESULTADOS

A montagem do circuito, conforme ilustrada na Figura 5.1, foi realizada com sucesso e cada componente foi conectado corretamente e a placa a um computador para permitir a inserção do código na placa.

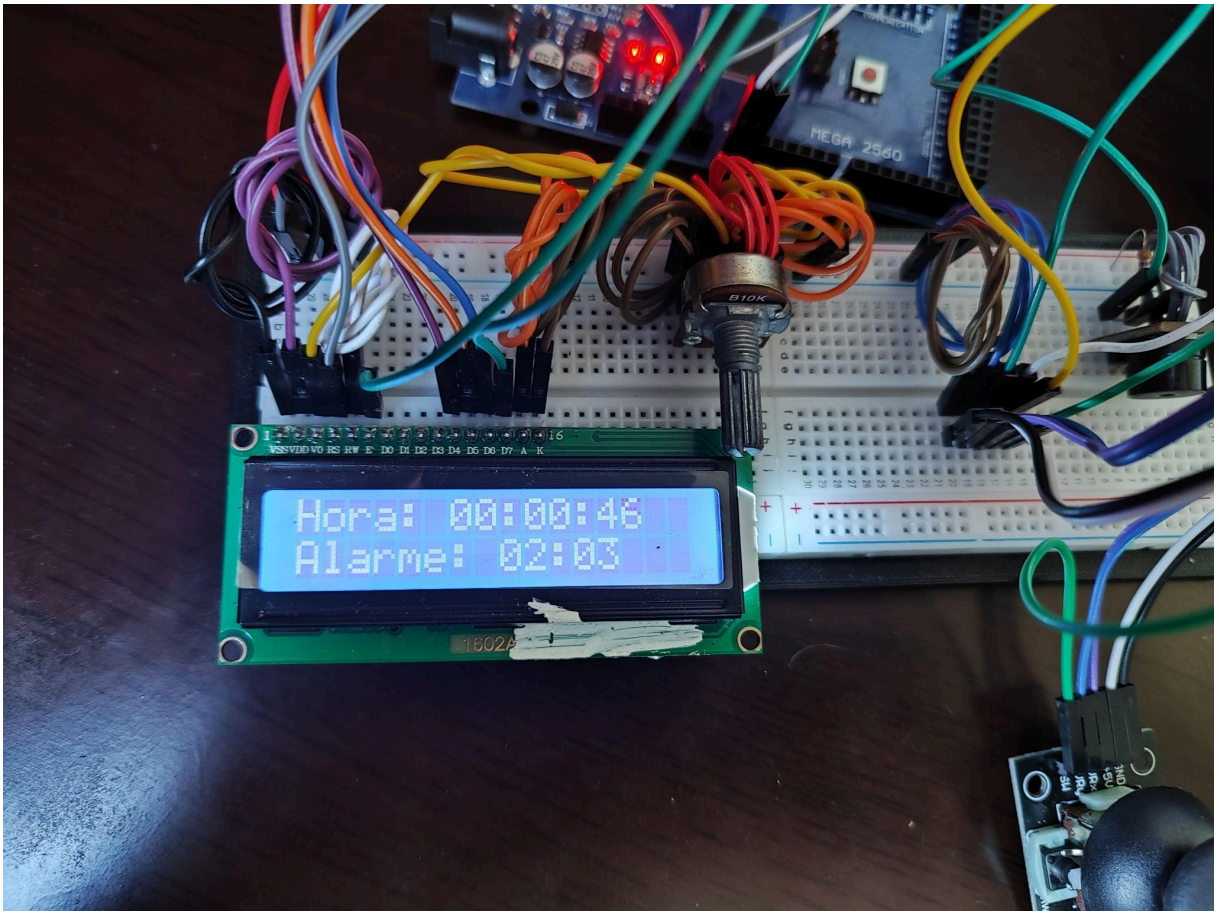
Figura 6.1 - Circuito montado para o experimento



Fonte: autoria própria

Após ter concluído todas as montagens e validado, começamos a observar se a hora estava incrementando de maneira correta, como podemos observar nas figuras 6.2, 6.3 e 6.4 nosso relógio está funcionando de forma efetiva.

Figura 6.2 - Hora em 46 segundos

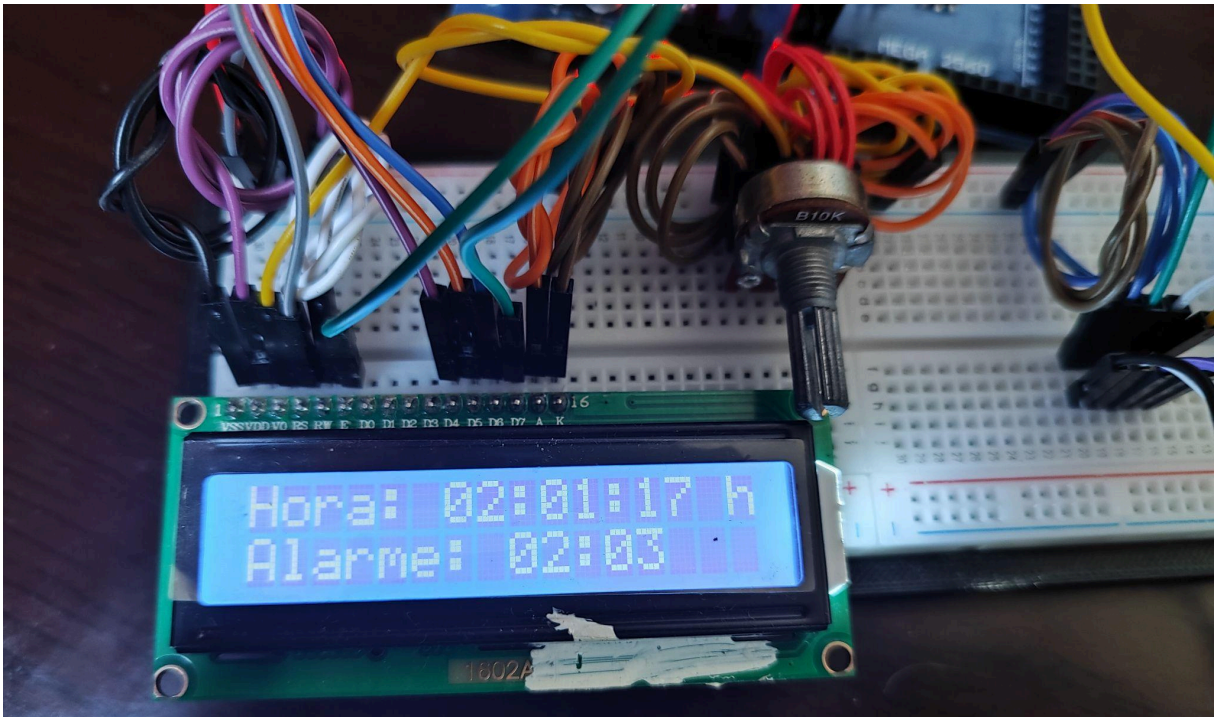


Fonte: autoria própria

Figura 6.3 - Hora em 46 segundos

Fonte: autoria própria

Figura 6.4 - Hora em 2 horas, 1 minuto e 17 segundos



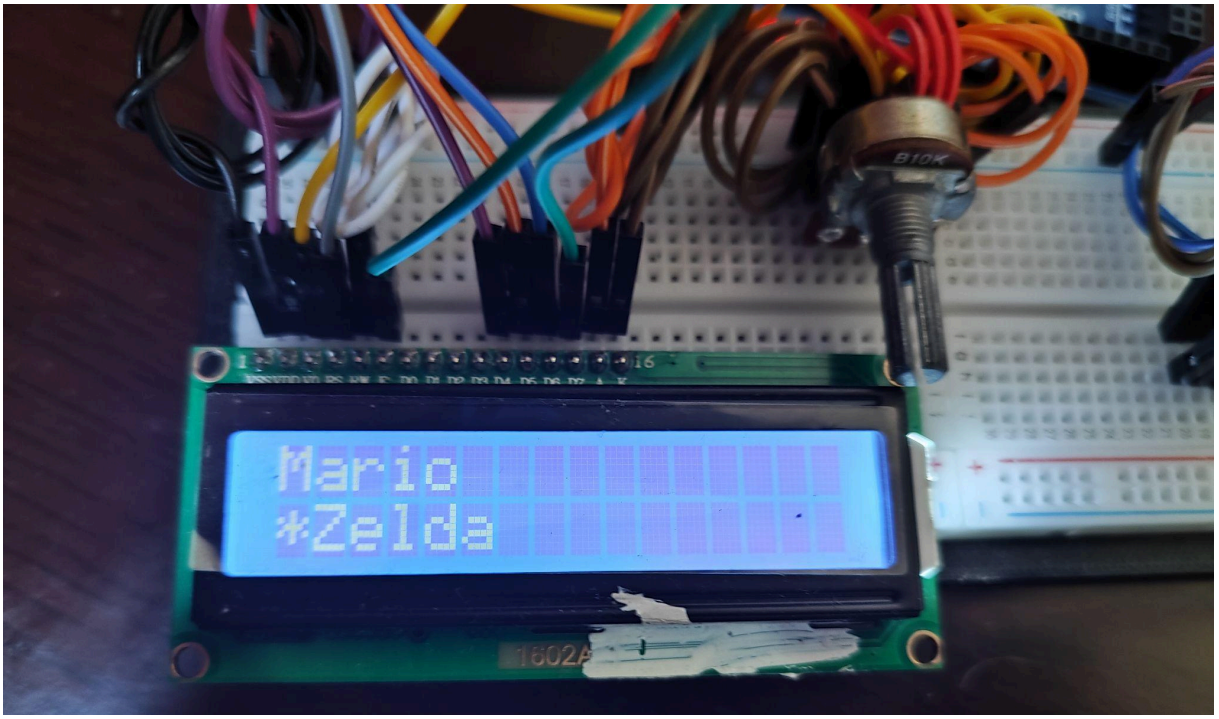
Fonte: autoria própria

Quando nos referimos ao funcionamento correto do alarme, deixamos o relógio ligado funcionando e incrementando as horas até chegar em 02:03, que era o valor escolhido para o alarme e ao chegar nesse valor nosso sistema dispara corretamente de acordo com a música escolhida. Na figura abaixo podemos observar o menu de música e quais músicas o usuário pode escolher.

Figura 6.5 - Menu de musica

Fonte: autoria própria

Figura 6.6 - Músicas atuais do projeto



Fonte: autoria própria

7. CONCLUSÃO

A implementação do sistema de alarme utilizando Arduino e máquinas de estados demonstrou ser uma abordagem eficaz para resolver o problema proposto. Ao longo do projeto, exploramos os conceitos de máquinas de estados e sua aplicação prática no desenvolvimento de sistemas embarcados. Através da divisão do programa em estados discretos e da definição de transições entre esses estados, conseguimos controlar o fluxo de execução do programa de forma organizada e modular, facilitando o desenvolvimento e a manutenção do sistema.

A utilização de máquinas de estados permitiu uma melhor estruturação do código, tornando-o mais legível e fácil de entender. Isso facilitou a implementação de funcionalidades específicas, como o ajuste do horário, a exibição dos horários no display LCD e o acionamento do buzzer quando necessário. Além disso, a modularização do código em classes para cada componente do sistema contribuiu para uma melhor organização e reutilização de código, promovendo uma abordagem de programação mais orientada a objetos.

O projeto também proporcionou uma oportunidade de praticar habilidades de resolução de problemas e trabalho em equipe, além de expandir nosso conhecimento sobre o funcionamento do Arduino e sua integração com diversos componentes eletrônicos. Através da implementação deste sistema de alarme, adquirimos uma compreensão mais profunda das aplicações práticas das máquinas de estados em sistemas embarcados e sua importância para o desenvolvimento de soluções confiáveis e eficientes.

Em resumo, a implementação do sistema de alarme com Arduino utilizando máquinas de estados foi bem-sucedida, alcançando os objetivos propostos e proporcionando uma experiência valiosa de aprendizado e desenvolvimento de habilidades. Este projeto serve como base para futuras explorações e aplicações de máquinas de estados em outros contextos de sistemas embarcados.

8. REFERÊNCIAS

khoih-prog. 2022. TimerInterrupt [Software repository]. GitHub. Disponível em: https://github.com/khoih-prog/TimerInterrupt. Acesso em: 09 de maio de 2024.

Arduino. 2024. LiquidCrystal [Documentação]. Disponível em: https://www.arduino.cc/reference/en/libraries/liquidcrystal/. Acesso em: 09 de maio de 2024.

Arduino. c2024. Serial [Documentação]. Disponível em: https://www.arduino.cc/reference/en/language/functions/communication/serial/. Acesso em: 09 de maio de 2024.

ANEXO I - CÓDIGO

- **Arquivo Principal “clockATMega.ino”**

```C/C++
/**
 * @file main.ino
 * @brief Função principal e funções auxiliares para o sistema de alarme e relógio.
 * @author Gabriel Finger Conte e Maria Eduarda Pedroso
 *
 * Este arquivo contém a função principal do sistema de alarme e relógio, juntamente com
 * funções auxiliares para ajustar o horário e o alarme, atualizar o estado do sistema e
 * controlar a interface de usuário.
 */

// Variáveis para armazenar os horários do alarme e atual
int alarmHour = 0;       ///< Hora do alarme
int alarmMinute = 0;     ///< Minuto do alarme
int alarmSecond = 0;     ///< Segundo do Alarme
int currentTick = 0;     ///< Contador de tick atual
int currentSeconds = 0;  ///< Segundos atuais
int currentHour = 0;     ///< Hora atual
int currentMinute = 0;   ///< Minuto atual
boolean lcdState;        ///< Flag de atualização do display LCD

// Estados para o relógio
enum MachineStates {
  Idle_Machine,
  Select_Hour,
  H_Hours,
  H_Minutes,
  H_Seconds,
```

```
  Select_Alarm,
  A_Hours,
  A_Minutes,
  A_Seconds, /* Não utilizado, para utilização caso desejado*/
  Select_Tone,
  Tone_List
};

MachineStates estadoAtual;  ///< Estado atual da máquina de estados

#include "timerConfig.h"    /* Configuração e Funções com o Timer */
#include "joystickConfig.h" /* Definições e Configurações do Joystic */
#include "alarmConfig.h"    /* Definições e Configurações do Alarme */
#include "buzzerConfig.h"   /* Definições e Configurações do Buzzer */
#include "lcdConfig.h"      /* Definições e Configurações do LCD */

static Joystick joystick(JOY_X_PIN, JOY_Y_PIN);   ///< Objeto Joystick para
leitura dos valores do joystick
JoystickState joyState;                                ///< Estado atual do
joystick
static Button button(JOY_BUTTON_PIN);               ///< Objeto Button para
leitura do estado do botão

/**
 * Função para atualizar o estado do sistema com base na entrada do botão e
do joystick.
 * Esta função verifica o estado atual do sistema e atualiza o estado de
acordo com as ações do usuário.
 * @param pressedButton Indica se o botão está pressionado.
 * @param joystate O estado atual do joystick.
 */
void atualizaEstado(bool pressedButton, JoystickState joystate);

/**
 * Função para ajustar o horário do sistema com base na entrada do joystick.
```

```
 * Esta função permite ajustar a hora, minuto e segundo do sistema utilizando as direções do joystick.
 * @param direction A direção do joystick (Up, Down, Left, Right).
 */
void adjustTime(JoystickState direction);

/**
 * Função principal de inicialização do sistema.
 * Esta função configura os dispositivos (LCD, joystick, buzzer, timer) e inicia o loop principal do sistema.
 */
void setup() {
   Serial.begin(9600);   ///< Inicializa a comunicação serial com a taxa de transmissão de 9600 bps
  while (!Serial) {};  // Aguarda até que a comunicação serial esteja pronta

  // Inicialização do display LCD
  lcd.begin(16, 2);  ///< Inicializa o display LCD com 16 colunas e 2 linhas

  // Inicialização do joystick e botão
   pinMode(JOY_BUTTON_PIN, INPUT_PULLUP);   ///< Configura o pino do botão do joystick como entrada com pull-up

  // Inicialização do buzzer
  pinMode(BUZZER_PIN, OUTPUT);  ///< Configura o pino do buzzer como saída

  /* Inicializa o timer */
#if USE_TIMER_1
  ITimer1.init();  ///< Inicializa o Timer 1
  if (ITimer1.attachInterruptInterval(TIMER_INTERVAL_MS, TimerHandler)) {
       Serial.print(F("Starting   ITimer1  OK,  millis()  =  "));   ///< Exibe mensagem de inicialização bem-sucedida do Timer 1
     Serial.println(millis());                                ///< Exibe o valor atual do contador de milissegundos
  } else {
```

```
    Serial.println(F("Can't set ITimer1. Select another freq. or timer"));
///< Exibe mensagem de erro se não for possível configurar o Timer 1
                                                                        }
// if
#endif

}  // setup

/**
 * Loop principal do sistema.
 * Este loop executa continuamente as tarefas principais do sistema, como
atualização de estado, interface com o usuário e verificação de alarme.
 */
void loop() {

  button.update();  ///< Atualiza o estado do botão

  joyState = joystick.getState();  ///< Obtém o estado atual do joystick

   atualizaEstado(!button.getState(), joyState);   ///< Atualiza o estado do
sistema com base no estado do botão e do joystick

   lastButtonState = button.getState();   ///< Atualiza o estado do último
botão pressionado

  // Atualiza o display LCD
  updateLCD();  ///< Atualiza o conteúdo do display LCD

  // Verifica se é hora do alarme e aciona o buzzer
   Alarm alarm(alarmHour, alarmMinute);                    ///< Objeto Alarm
para verificação do alarme
   if (alarm.isAlarmTime(currentHour, currentMinute)) {   ///< Verifica se é
hora de acionar o alarme
    buzz();                                              ///< Aciona o buzzer
  } else {
    noTone(BUZZER_PIN);  // Desliga o buzzer
```

```
    buzzIndex = 0;
  }

}  // loop

/**
 * Função para atualizar o estado do sistema com base na entrada do botão e do joystick.
 * Esta função verifica o estado atual do sistema e atualiza o estado de acordo com as ações do usuário.
 * @param pressedButton Indica se o botão está pressionado.
 * @param joystate O estado atual do joystick.
 */
void atualizaEstado(bool pressedButton, JoystickState joystate) {
  // Variáveis auxiliáres para verificar o debounce e sensibilidade no ajuste com os joysticks
  unsigned long currXdebounce = millis() - lastXdebounce;
  unsigned long currYdebounce = millis() - lastYdebounce;

  // Verifica o estado atual
  switch (estadoAtual) {
    case Idle_Machine:
         // Se estiver em espera, caso aperte o botão entra no menu de configuração.
      if (pressedButton && lastButtonState == 1) {
        estadoAtual = Select_Hour;
      }  // if
      break;

    case Select_Hour:
      // Se quiser alterar a hora do relógio, aperta o botão
      if (pressedButton && lastButtonState == 1) {
        estadoAtual = H_Hours;
      }  // if
```

```
      // Com o ajuste de sensibilidade/debounce verifica se deseja selecionar
para alterar o alarme
      if (joystate == Down) {
        if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL_MENU) {
          lastYdebounce = millis();
          estadoAtual = Select_Alarm;
        }  // if currYdebounce
      }    // joystate

      if (joystate == Up) {
        if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL_MENU) {
          lastYdebounce = millis();
          estadoAtual = Select_Tone;
        }  // if currYdebounce
      }    // joystate
      break;

    case H_Hours:
      // Se quiser finalizar a alteração aperta o botão
      if (pressedButton && lastButtonState == 1) {
        estadoAtual = Idle_Machine;
      }  // if
      // Chama a função para alterar o tempo do relógio
      adjustTime(joystate);
      break;

    case H_Minutes:
      // Se quiser finalizar a alteração aperta o botão
      if (pressedButton && lastButtonState == 1) {
        estadoAtual = Idle_Machine;
      }  // if
      // Chama a função para alterar o tempo do relógio
      adjustTime(joystate);

      break;
```

```
    case H_Seconds:
      // Se quiser finalizar a alteração aperta o botão
      if (pressedButton && lastButtonState == 1) {
        estadoAtual = Idle_Machine;
      }
      // Chama a função para alterar o tempo do relógio
      adjustTime(joystate);
      break;

    case Select_Alarm:
      // Se quiser alterar a hora do alarme, aperta o botão
      if (pressedButton && lastButtonState == 1) {
        estadoAtual = A_Hours;
      }  // if

      // Com o ajuste de sensibilidade/debounce verifica se deseja selecionar
para alterar o tempo do relógio
      if (joystate == Up) {
        if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL_MENU) {
          lastYdebounce = millis();
          estadoAtual = Select_Hour;
        }  // if currYdebounce
      }    // if joystate
      if (joystate == Down) {
        if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL_MENU) {
          lastYdebounce = millis();
          estadoAtual = Select_Tone;
        }  // if currYdebounce
      }    // if joystate
      break;

    case A_Hours:
      // Se quiser finalizar a alteração aperta o botão
      if (pressedButton && lastButtonState == 1) {
        estadoAtual = Idle_Machine;
      }  // if
```

```
      // Chama a função para alterar o tempo do alarme
      adjustAlarmTime(joystate);
      break;

    case A_Minutes:
      // Se quiser finalizar a alteração aperta o botão
      if (pressedButton && lastButtonState == 1) {
        estadoAtual = Idle_Machine;
      }  // if
      // Chama a função para alterar o tempo do alarme
      adjustAlarmTime(joystate);
      break;

     case A_Seconds:   // Como não esta implementado, volta para Idle caso
entre
      estadoAtual = Idle_Machine;
      // Se quiser finalizar a alteração aperta o botão
      //     if (pressedButton && lastButtonState == 1) {
      //       estadoAtual = Idle_Machine;
      //     }// if
      //     // Chama a função para alterar o tempo do alarme
      //     adjustAlarmTime(joystate);
      break;

    case Select_Tone:
      // Se quiser alterar a música do alarme, aperta o botão
      if (pressedButton && lastButtonState == 1) {
        estadoAtual = Tone_List;
      }  // if

      // Com o ajuste de sensibilidade/debounce verifica se deseja selecionar
para alterar o tempo do relógio ou do alarme
      if (joystate == Up) {
        if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL_MENU) {
          lastYdebounce = millis();
          estadoAtual = Select_Alarm;
```

```
        }  // if currYdebounce
      }    // if joystate
      if (joystate == Down) {
        if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL_MENU) {
          lastYdebounce = millis();
          estadoAtual = Select_Hour;
        }  // if currYdebounce
      }    // if joystate
      break;

    case Tone_List:
      // Se quiser salvar a música escolhida, aperta o botão
      if (pressedButton && lastButtonState == 1) {
        estadoAtual = Idle_Machine;
      }  // if
      adjustTone(joystate);
      break;

    default:
      // Se deu ruim, informa no serial e reseta o estado atual
          Serial.print("Erro updateState - Estado Desconhecido, resetando
estado...");
      Serial.println(estadoAtual);
      estadoAtual = Idle_Machine;
      break;
  }  // switch

}  // atualizaEstado

/**
 * Função para ajustar o horário do sistema com base na entrada do joystick.
 * Esta função permite ajustar a hora, minuto e segundo do sistema utilizando
as direções do joystick.
 * @param direction A direção do joystick (Up, Down, Left, Right).
 */
void adjustTime(JoystickState direction) {
```

```
  // Variáveis auxiliares para verificar o debounce e sensibilidade no ajuste
com os joysticks
  unsigned long currXdebounce = millis() - lastXdebounce;  ///< Tempo desde o
último debounce do eixo X do joystick
  unsigned long currYdebounce = millis() - lastYdebounce;  ///< Tempo desde o
último debounce do eixo Y do joystick

  switch (estadoAtual) {                                    ///< Verifica o
estado atual do sistema relacionado ao ajuste do horário
    case H_Hours:                                          ///< Estado para
ajuste da hora
      switch (direction) {                                  ///< Verifica a
direção do joystick
        case Up:                                             ///< Aumenta a
hora
          if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL) {  ///< Verifica se
passou o tempo de debounce
            lastYdebounce = millis();                       ///< Atualiza o
tempo de debounce
           currentHour++;                                  ///< Incrementa a
hora
            if (currentHour >= 24) {                        ///< Verifica se
ultrapassou as 24 horas
              currentHour = 0;                             ///< Volta para 0
horas
           }                                               // if currentHour
            }                                                          // if
currYdebounce
         break;

        case Down:                                           ///< Diminui a
hora
          if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL) {  ///< Verifica se
passou o tempo de debounce
            lastYdebounce = millis();                       ///< Atualiza o
tempo de debounce
```

```
            currentHour--;                                      ///< Decrementa a
hora
             if (currentHour < 0) {                             ///< Verifica se
está abaixo de 0 horas
                currentHour = 23;                               ///< Volta para
23 horas
            }                                                   // if currentHour
             }                                                          // if
currYdebounce
          break;

        case Left:  ///< Nenhuma ação para a esquerda
          break;

         case Right:                                                ///< Muda
para o ajuste dos minutos
               if (currXdebounce > JOYSTIC_DEBOUNCE_INTERVAL_MENU) {   ///<
Verifica se passou o tempo de debounce do menu
                lastXdebounce = millis();                           ///<
Atualiza o tempo de debounce do menu
              estadoAtual = H_Minutes;                              ///< Muda
para o ajuste dos minutos
          }                                                   // if
          break;

        default:  ///< Nenhuma ação para outras direções
          break;
      }  // switch
      break;

    case H_Minutes:                                             ///< Estado para
ajuste dos minutos
       switch (direction) {                                     ///< Verifica a
direção do joystick
         case Up:                                               ///< Aumenta os
minutos
```

```
            if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL) {  ///< Verifica se
passou o tempo de debounce
                lastYdebounce = millis();                      ///< Atualiza o
tempo de debounce
                currentMinute++;                               ///< Incrementa
os minutos
               if (currentMinute >= 60) {                      ///< Verifica se
ultrapassou os 60 minutos
                 currentMinute = 0;                           ///< Volta para 0
minutos
                 }                                                          // if
currentMinute
              }                                                             // if
currYdebounce
            break;

          case Down:                                           ///< Diminui os
minutos
            if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL) {  ///< Verifica se
passou o tempo de debounce
                lastYdebounce = millis();                      ///< Atualiza o
tempo de debounce
                currentMinute--;                               ///< Decrementa
os minutos
               if (currentMinute < 0) {                        ///< Verifica se
está abaixo de 0 minutos
                  currentMinute = 59;                          ///< Volta para
59 minutos
                 }                                                          // if
currentMinute
              }                                                             // if
currYdebounce
            break;

          case Left:                                                ///< Volta
para o ajuste da hora
```

```
                if (currXdebounce > JOYSTIC_DEBOUNCE_INTERVAL_MENU) {   ///<
Verifica se passou o tempo de debounce do menu
                lastXdebounce = millis();                                ///<
Atualiza o tempo de debounce do menu
              estadoAtual = H_Hours;                                  ///< Volta
para o ajuste da hora
            }                                                        // if
            break;

          case Right:                                                 ///< Muda
para o ajuste dos segundos
                if (currXdebounce > JOYSTIC_DEBOUNCE_INTERVAL_MENU) {   ///<
Verifica se passou o tempo de debounce do menu
                lastXdebounce = millis();                                ///<
Atualiza o tempo de debounce do menu
              estadoAtual = H_Seconds;                                ///< Muda
para o ajuste dos segundos
            }                                                        // if
            break;

         default:  ///< Nenhuma ação para outras direções
           break;
       }  // switch
       break;

     case H_Seconds:                                                ///< Estado para
ajuste dos segundos
        switch (direction) {                                         ///< Verifica a
direção do joystick
          case Up:                                                   ///< Aumenta os
segundos
            if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL) {  ///< Verifica se
passou o tempo de debounce
              lastYdebounce = millis();                              ///< Atualiza o
tempo de debounce
```

```
            currentSeconds++;                                   ///< Incrementa
os segundos
            if (currentSeconds >= 60) {                         ///< Verifica se
ultrapassou os 60 segundos
             currentSeconds = 0;                                ///< Volta para 0
segundos
              }                                                         // if
currentSeconds
            }                                                           // if
currYdebounce
          break;

         case Down:                                             ///< Diminui os
segundos
           if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL) {     ///< Verifica se
passou o tempo de debounce
             lastYdebounce = millis();                          ///< Atualiza o
tempo de debounce
             currentSeconds--;                                  ///< Decrementa
os segundos
             if (currentSeconds < 0) {                          ///< Verifica se
está abaixo de 0 segundos
               currentSeconds = 59;                             ///< Volta para
59 segundos
              }                                                         // if
currentSeconds
            }                                                           // if
currYdebounce
          break;

         case Left:                                                 ///< Volta
para o ajuste dos minutos
              if (currXdebounce > JOYSTIC_DEBOUNCE_INTERVAL_MENU) {   ///<
Verifica se passou o tempo de debounce do menu
               lastXdebounce = millis();                            ///<
Atualiza o tempo de debounce do menu
```

```
            estadoAtual = H_Minutes;                                ///< Volta
para o ajuste dos minutos
          }                                                       // if
          break;

        case Right:  ///< Nenhuma ação para a direita
          break;

        default:  ///< Nenhuma ação para outras direções
          break;
      }  //switch
      break;

    case Idle_Machine:  ///< Nenhuma ação quando a máquina está ociosa
      break;

                                                                  default:
///< Resetar o estado se estiver em um estado desconhecido
           Serial.print("Erro  adjustTime  -  Estado  Desconhecido,  resetando
estado...");  ///< Exibe mensagem de erro no monitor serial
                                                Serial.println(estadoAtual);
///< Exibe o estado desconhecido no monitor serial
                                        estadoAtual      =      Idle_Machine;
///< Reseta para o estado de máquina ociosa
                                                                  return;
///< Retorna da função
                                                                }
// switch

}  // adjustTime
```

- **Biblioteca “alarmConfig.h”**

```C/C++
/**
 * @file alarmConfig.h
 * @brief Configuração do alarme e funções relacionadas ao ajuste do horário do alarme.
 * @author Gabriel Finger Conte e Maria Eduarda Pedroso
 *
 * Este arquivo contém a definição da classe Alarm para representar o alarme e a função para ajustar
 * o horário do alarme com base na entrada do joystick.
*/

#ifndef alarmConfig_H
#define alarmConfig_H

// Classe para o alarme
class Alarm {
private:
  int hour;    ///< Hora do alarme
  int minute;  ///< Minuto do alarme

public:
  /**
   * Construtor da classe Alarm.
   * Inicializa o objeto Alarm com a hora e minuto especificados.
   * @param hour A hora do alarme.
   * @param minute O minuto do alarme.
   */
  Alarm(int hour, int minute)
    : hour(hour), minute(minute) {}

  /**
   * Método para verificar se é hora do alarme.
    * Este método compara a hora e o minuto do alarme com a hora e o minuto atuais para determinar se é hora do alarme.
   * @param currentHour A hora atual.
   * @param currentMinute O minuto atual.
```

```
   * @return Verdadeiro se for hora do alarme, falso caso contrário.
   */
  bool isAlarmTime(int currentHour, int currentMinute) {
    return (hour == currentHour && minute == currentMinute);
  }
};

/**
 * Função para ajustar o horário do alarme com base na entrada do joystick.
 * Esta função permite ajustar a hora e o minuto do alarme utilizando as direções do joystick.
 * @param direction A direção do joystick (Up, Down, Left, Right).
 */
void adjustAlarmTime(JoystickState direction);

#endif
```

- **Implementação da Biblioteca “alarmConfig.ino”**

```
C/C++
/**
 * @file alarmConfig.h
 * @brief Configuração do alarme e funções relacionadas ao ajuste do horário do alarme.
 * @author Gabriel Finger Conte e Maria Eduarda Pedroso
 *
 * Este arquivo contém a definição da função para ajustar o horário do alarme com base na entrada do joystick.
*/

/**
 * Função para ajustar o horário do alarme com base na entrada do joystick.
```

```
 * Esta função permite ajustar a hora e o minuto do alarme utilizando as direções do joystick.
 * @param direction A direção do joystick (Up, Down, Left, Right).
 */
void adjustAlarmTime(JoystickState direction) {
  unsigned long currXdebounce = millis() - lastXdebounce;  ///< Tempo desde o último debounce do eixo X do joystick
  unsigned long currYdebounce = millis() - lastYdebounce;  ///< Tempo desde o último debounce do eixo Y do joystick

  switch (estadoAtual) {                                    ///< Verifica o estado atual do sistema relacionado ao ajuste do alarme
    case A_Hours:                                           ///< Estado para ajuste da hora do alarme
      switch (direction) {                                  ///< Verifica a direção do joystick
        case Up:                                            ///< Aumenta a hora do alarme
          if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL) {  ///< Verifica se passou o tempo de debounce
            lastYdebounce = millis();                       ///< Atualiza o tempo de debounce
            alarmHour++;                                    ///< Incrementa a hora do alarme
            if (alarmHour >= 24) {                          ///< Verifica se ultrapassou as 24 horas
              alarmHour = 0;                                ///< Volta para 0 horas
            }                                               // if alarmHour
          }                                                 // if currYdebounce
          break;

        case Down:                                          ///< Diminui a hora do alarme
```

```
            if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL) {  ///< Verifica se
passou o tempo de debounce
                lastYdebounce = millis();                       ///< Atualiza o
tempo de debounce
                alarmHour--;                                   ///< Decrementa a
hora do alarme
                if (alarmHour < 0) {                            ///< Verifica se
está abaixo de 0 horas
                    alarmHour = 23;                             ///< Volta para
23 horas
                }                                              // if alarmHour
                break;
            }  // if currYdebounce

        case Left:  ///< Nenhuma ação para a esquerda
            break;

        case Right:                                                ///< Muda
para o ajuste dos minutos
                if (currXdebounce > JOYSTIC_DEBOUNCE_INTERVAL_MENU) {   ///<
Verifica se passou o tempo de debounce do menu
                    lastXdebounce = millis();                          ///<
Atualiza o tempo de debounce do menu
                    estadoAtual = A_Minutes;                           ///< Muda
para o ajuste dos minutos
                }                                                  // if
                break;

        default:  ///< Nenhuma ação para outras direções
            break;
    }
    break;

    case A_Minutes:                                                ///< Estado para
ajuste dos minutos do alarme
```

```
      switch (direction) {                                    ///< Verifica a
direção do joystick
        case Up:                                              ///< Aumenta os
minutos do alarme
          if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL) {  ///< Verifica se
passou o tempo de debounce
            lastYdebounce = millis();                         ///< Atualiza o
tempo de debounce
            alarmMinute++;                                    ///< Incrementa
os minutos do alarme
           if (alarmMinute >= 60) {                          ///< Verifica se
ultrapassou os 60 minutos
            alarmMinute = 0;                                 ///< Volta para 0
minutos
          }                                                  // if alarmMinute
           }                                                           // if
currYdebounce
        break;

        case Down:                                            ///< Diminui os
minutos do alarme
          if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL) {  ///< Verifica se
passou o tempo de debounce
            lastYdebounce = millis();                         ///< Atualiza o
tempo de debounce
            alarmMinute--;                                    ///< Decrementa
os minutos do alarme
           if (alarmMinute < 0) {                            ///< Verifica se
está abaixo de 0 minutos
              alarmMinute = 59;                               ///< Volta para
59 minutos
          }                                                  // if alarmMinute
           }                                                           // if
currYdebounce
        break;
```

```
         case Left:                                                    ///< Volta
para o ajuste da hora
               if (currXdebounce > JOYSTIC_DEBOUNCE_INTERVAL_MENU) {   ///<
Verifica se passou o tempo de debounce do menu
               lastXdebounce = millis();                               ///<
Atualiza o tempo de debounce do menu
             estadoAtual = A_Hours;                                    ///< Volta
para o ajuste da hora
          }                                                          // if
          break;

        case Right:  ///< Nenhuma ação para a direita
          break;

        default:  ///< Nenhuma ação para outras direções
          break;
      }  // switch
      break;

    case A_Seconds:                                                ///< Estado para
ajuste dos segundos do alarme
       switch (direction) {                                         ///< Verifica a
direção do joystick
         case Up:                                                   ///< Aumenta os
segundos do alarme
           if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL) {  ///< Verifica se
passou o tempo de debounce
             lastYdebounce = millis();                              ///< Atualiza o
tempo de debounce
             alarmSecond++;                                         ///< Incrementa
os minutos do alarme
            if (alarmSecond >= 60) {                               ///< Verifica se
ultrapassou os 60 segundos
              alarmSecond = 0;                                   ///< Volta para 0
segundos
           }                                                    // if alarmSecond
```

```
            }                                                       // if
currYdebounce
          break;

         case Down:                                          ///< Diminui os
segundos do alarme
           if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL) {  ///< Verifica se
passou o tempo de debounce
             lastYdebounce = millis();                       ///< Atualiza o
tempo de debounce
             alarmSecond--;                                  ///< Decrementa
os segundos do alarme
            if (alarmSecond < 0) {                           ///< Verifica se
está abaixo de 0 segundos
               alarmSecond = 59;                             ///< Volta para
59 segundos
            }                                               // if alarmSecond
            }                                                       // if
currYdebounce
          break;

         case Left:                                             ///< Volta
para o ajuste da hora
              if (currXdebounce > JOYSTIC_DEBOUNCE_INTERVAL_MENU) {   ///<
Verifica se passou o tempo de debounce do menu
               lastXdebounce = millis();                            ///<
Atualiza o tempo de debounce do menu
             estadoAtual = A_Hours;                             ///< Volta
para o ajuste da hora
          }                                                 // if
          break;

        case Right:  ///< Nenhuma ação para a direita
          break;

        default:  ///< Nenhuma ação para outras direções
```

```
            break;
        }  // switch
        break;

    case Idle_Machine:  ///< Nenhuma ação quando a máquina está ociosa
        break;

                                                                    default:
///< Resetar o estado se estiver em um estado desconhecido
            Serial.print("Erro adjustAlarm - Estado Desconhecido, resetando
estado...");  ///< Exibe mensagem de erro no monitor serial
                                                Serial.println(estadoAtual);
///< Exibe o estado desconhecido no monitor serial
                                        estadoAtual      =      Idle_Machine;
///< Reseta para o estado de máquina ociosa
                                                                    return;
///< Retorna da função
                                                                        }
// switch
}  // adjustAlarmTime
```

- **Biblioteca “buzzerConfig.h”**

```
C/C++
/**
 * @file buzzerConfig.h
 * @brief Configuração do buzzer e função para acioná-lo.
 * @author Gabriel Finger Conte e Maria Eduarda Pedroso
 *
 * Este arquivo contém a definição do pino para o buzzer e a função para
acioná-lo.
*/
```

```
#ifndef buzzerConfig_H
#define buzzerConfig_H

// Definição do pino para o buzzer
#define BUZZER_PIN 8  ///< Pino de controle do buzzer

// Definição das frequências das notas musicais
#define NOTE_C4 261   // Dó (C4)
#define NOTE_CS4 277  // Dó# / Réb (C#4 / Db4)
#define NOTE_D4 293   // Ré (D4)
#define NOTE_DS4 311  // Ré# / Mib (D#4 / Eb4)
#define NOTE_E4 329   // Mi (E4)
#define NOTE_F4 349   // Fá (F4)
#define NOTE_FS4 369  // Fá# / Solb (F#4 / Gb4)
#define NOTE_G4 392   // Sol (G4)
#define NOTE_GS4 415  // Sol# / Láb (G#4 / Ab4)
#define NOTE_A4 440   // Lá (A4)
#define NOTE_AS4 466  // Lá# / Síb (A#4 / Bb4)
#define NOTE_B4 493   // Si (B4)
#define NOTE_C5 523   // Dó (C5)
#define NOTE_CS5 554  // Dó# / Réb (C#5 / Db5)
#define NOTE_D5 587   // Ré (D5)
#define NOTE_DS5 622  // Ré# / Mib (D#5 / Eb5)
#define NOTE_E5 659   // Mi (E5)
#define NOTE_F5 698   // Fá (F5)
#define NOTE_FS5 739  // Fá# / Solb (F#5 / Gb5)
#define NOTE_G5 783   // Sol (G5)
#define NOTE_GS5 830  // Sol# / Láb (G#5 / Ab5)
#define NOTE_A5 880   // Lá (A5)
#define NOTE_AS5 932  // Lá# / Síb (A#5 / Bb5)
#define NOTE_B5 987   // Si (B5)
#define NOTE_C6 1046  // Dó (C6)

unsigned long lastBuzz = millis();  ///< Último momento em que o buzzer foi
acionado
```

```
int buzzIndex = 0;                          ///< Índice para percorrer as
melodias/durações
int tom;                                ///< Tom a ser tocado no buzzer
int duracao;                                ///< Duração da nota a ser tocada no
buzzer
boolean nextBuzz = true;                ///< Indica se deve tocar a próxima nota
ou esperar passar o tempo
uint8_t buzzSelect = 0;                 ///< Seletor da melodia a ser tocada no
alarme
const uint8_t maxToneIndex = 1;

// Melodia da música Saria's Song de The Legend of Zelda
const int melodia_zelda[] = {
  0, NOTE_F4, NOTE_A4, NOTE_B4, 0,
  NOTE_F4, NOTE_A4, NOTE_B4, 0,
  NOTE_F4, NOTE_A4, NOTE_B4, NOTE_E5, NOTE_D5, 0,
  NOTE_B4, NOTE_C5, NOTE_B4, NOTE_G4, NOTE_E4, 0,
  NOTE_D4, NOTE_E4, NOTE_G4, NOTE_E4, 0,

  NOTE_F4, NOTE_A4, NOTE_B4, 0,
  NOTE_F4, NOTE_A4, NOTE_B4, 0,
  NOTE_F4, NOTE_A4, NOTE_B4, NOTE_E5, NOTE_D5, 0,
  NOTE_B4, NOTE_C5, NOTE_E5, NOTE_B4, NOTE_G4, 0,
  NOTE_B4, NOTE_G4, NOTE_D4, NOTE_E4
};

// Durações das notas
const int duracoes_zelda[] = {
  1, 2, 2, 3, 1,
  2, 2, 3, 1,
  2, 2, 2, 2, 4, 1,
  2, 2, 2, 2, 4, 1,
  2, 2, 2, 4, 1,
  2, 2, 3, 1,
  2, 2, 3, 1,
  2, 2, 2, 2, 4, 1,
```

```
  2, 2, 2, 2, 4, 1,
  2, 2, 2, 4
};

const int maxIndex_zelda = sizeof(melodia_zelda) / sizeof(melodia_zelda[0]);

// Melodia do Theme de Mario
const int melodia_mario[] = {
  0, NOTE_E5, 0, NOTE_E5, 0, NOTE_E5, NOTE_C5, NOTE_E5, NOTE_G5, NOTE_G4, 0,
  NOTE_C5, NOTE_G4, NOTE_E4, NOTE_A4, NOTE_B4, NOTE_AS4, NOTE_A4, NOTE_G4, 0,
   NOTE_E5,  NOTE_G5,  NOTE_A5,  NOTE_F5,  NOTE_G5,  NOTE_E5,  NOTE_C5,  NOTE_D5,
NOTE_B4, 0,

  NOTE_G5, NOTE_FS5, NOTE_F5, NOTE_DS5, NOTE_E5, 0,
  NOTE_GS4, NOTE_A4, NOTE_C5, NOTE_A4, NOTE_C5, NOTE_D5, 0,
  NOTE_G5, NOTE_FS5, NOTE_F5, NOTE_DS5, NOTE_E5, 0,
  NOTE_C6, 0, NOTE_C6, 0, NOTE_C6,
};

// Durações das notas
const int duracoes_mario[] = {
  1, 1, 0, 1, 0, 2, 1, 1, 2, 2, 1,
  2, 2, 2, 1, 2, 1, 1, 2, 1,
  1, 1, 2, 1, 2, 2, 1, 1, 2, 1,
  2, 2, 2, 1, 2, 1, 1, 2, 1,
  1, 1, 2, 1, 2, 2, 1, 1, 2, 1,

  1, 1, 1, 1, 2, 1,
  1, 1, 2, 1, 1, 2, 1,
  1, 1, 1, 1, 2, 1,
  1, 0, 1, 0, 1
};

const int maxIndex_mario = sizeof(melodia_mario) / sizeof(melodia_mario[0]);

/**
```

```
 * Função para acionar o buzzer com uma frequência específica.
  * Esta função verifica o tempo desde o último acionamento do buzzer e
controla o acionamento do buzzer de acordo com o tempo de delay especificado.
*/
void buzz();

/**
 * Ajusta o tom da música selecionada utilizando o joystick.
 * @param direction A direção do joystick (Up para aumentar o tom, Down para
diminuir o tom).
 */
void adjustTone(JoystickState direction);

#endif
```

- **Implementação da Biblioteca “buzzerConfig.ino”**

```
C/C++
/**
   @file buzzerConfig.h
   @brief Configuração do buzzer e função para acioná-lo.
   @author Gabriel Finger Conte e Maria Eduarda Pedroso

   Este arquivo contém a definição da função para acionar o buzzer.
*/

/**
 * Ajusta o tom da música selecionada utilizando o joystick.
 * @param direction A direção do joystick (Up para aumentar o tom, Down para
diminuir o tom).
 */
void adjustTone(JoystickState direction) {
```

```
  // Variáveis auxiliares para verificar o debounce e sensibilidade no ajuste com os joysticks
  unsigned long currXdebounce = millis() - lastXdebounce;  ///< Tempo desde o último debounce do eixo X do joystick
  unsigned long currYdebounce = millis() - lastYdebounce;  ///< Tempo desde o último debounce do eixo Y do joystick

  switch (direction) {                                     ///< Verifica a direção do joystick
    case Up:                                               ///< Aumenta o índice da música selecionada
       if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL) {   ///< Verifica se passou o tempo de debounce
         lastYdebounce = millis();                         ///< Atualiza o tempo de debounce
       if (buzzSelect < maxToneIndex) {
         buzzSelect++;   ///< Incrementa o índice da música selecionada
         buzzIndex = 0;  ///< Reinicia o índice da melodia
       }
     }  // if currYdebounce
     break;

    case Down:                                             ///< Diminui o índice da música selecionada
       if (currYdebounce > JOYSTIC_DEBOUNCE_INTERVAL) {   ///< Verifica se passou o tempo de debounce
         lastYdebounce = millis();                         ///< Atualiza o tempo de debounce
       if (buzzSelect > 0) {
         buzzSelect--;   ///< Decrementa o índice da música selecionada
         buzzIndex = 0;  ///< Reinicia o índice da melodia
       }
     }  // if currYdebounce
     break;

   case Left:  ///< Nenhuma ação para a esquerda
```

```
      break;

    case Right:  ///< Nenhuma ação para a direita
      break;

    default:  ///< Nenhuma ação para outras direções
      break;
  }  // switch

}  // adjustTone

/**
 * Toca uma nota musical no buzzer de acordo com a melodia selecionada.
 * A melodia pode ser a do jogo Zelda ou do jogo Mario.
 * @param buzzSelect Seleção da melodia a ser tocada (0 para Zelda, 1 para
Mario).
 */
void buzz() {
  // Variáveis para controle do tempo
  unsigned long tempoAtual = millis();

  // Seleciona a melodia com base na variável buzzSelect
  switch (buzzSelect) {
    case 0:
      tom = melodia_zelda[buzzIndex];
      duracao = duracoes_zelda[buzzIndex] * 200;
      break;
    case 1:
      tom = melodia_mario[buzzIndex];
      duracao = duracoes_mario[buzzIndex] * 200;
      break;
    default:
      tom = melodia_zelda[buzzIndex];
      duracao = duracoes_zelda[buzzIndex] * 200;
      break;
  }
```

```
// Verifica se é uma pausa ou uma nota musical
if (tom == 0) {  // Pausa
  if (nextBuzz) {
    nextBuzz = false;
    noTone(BUZZER_PIN);
  }  // if

  if (tempoAtual - lastBuzz >= duracao + 50) {
    buzzIndex++;
    lastBuzz = tempoAtual;
    nextBuzz = true;
  }       // if
} else {  // Nota musical
  if (nextBuzz) {
    nextBuzz = false;
    tone(BUZZER_PIN, tom, duracao * 10);
  }  // if

  if (tempoAtual - lastBuzz >= (duracao + 50)) {
    buzzIndex++;
    lastBuzz = tempoAtual;
    nextBuzz = true;
  }  // if
}    // if

// Verifica se chegou ao final da melodia e reinicia se necessário
switch (buzzSelect) {
  case 0:
    if (buzzIndex >= maxIndex_zelda) {
      buzzIndex = 0;
    }  // if
    break;
  case 1:
    if (buzzIndex >= maxIndex_mario) {
      buzzIndex = 0;
```

```
      }  // if
      break;
    default:
      if (buzzIndex >= maxIndex_zelda) {
        buzzIndex = 0;
      }  // if
      break;
  }  // switch buzzSelect
}  // buzz
```

- **Implementação da Biblioteca “joystickConfig.h”**

```
C/C++
/**
 * @file joystickConfig.h
 * @brief Configuração do joystick e botão, incluindo as máquinas de estado
para o botão e o joystick.
 * @author Gabriel Finger Conte e Maria Eduarda Pedroso
 *
 * Este arquivo contém a definição dos pinos para o joystick e botão, bem
como as classes para o botão e o joystick.
*/

#ifndef joystickConfig_H
#define joystickConfig_H

// Definição dos pinos para o joystick e botão
#define JOY_X_PIN A0                          ///< Pino para a leitura do eixo
X do joystick
#define JOY_Y_PIN A1                          ///< Pino para a leitura do eixo
Y do joystick
#define JOY_BUTTON_PIN 7                            ///< Pino para o botão do
joystick
```

```
#define JOYSTIC_DEBOUNCE_INTERVAL 200       ///< Intervalo de debounce para o
joystick
#define JOYSTIC_DEBOUNCE_INTERVAL_MENU 300  ///< Intervalo de debounce para o
joystick no menu

unsigned long lastXdebounce = millis();  ///< Último momento de debounce do
eixo X do joystick
unsigned long lastYdebounce = millis();  ///< Último momento de debounce do
eixo Y do joystick
bool lastButtonState = 1;                   ///< Último estado do botão do
joystick

// Estados para o botão
enum ButtonState {
  Idle,     ///< Estado ocioso do botão
  Pressed,  ///< Estado pressionado do botão
  Released  ///< Estado liberado do botão
};


// Estados para o joystick
enum JoystickState {
  Neutral,  ///< Estado neutro do joystick
  Up,       ///< Estado para cima do joystick
  Down,     ///< Estado para baixo do joystick
  Left,     ///< Estado para a esquerda do joystick
  Right     ///< Estado para a direita do joystick
};


// Classe do botão
class Button {
private:
  int pin;                          ///< Pino de leitura do botão
  ButtonState state;                ///< Estado atual do botão
  unsigned long lastDebounceTime;   ///< Último momento de debounce do botão
```

```
  unsigned long debounceDelay;     ///< Intervalo de debounce do botão

public:
  /**
   * Construtor da classe Button.
    * Inicializa o objeto Button com o pino especificado e configurações
padrão.
   * @param pin O pino de leitura do botão.
   */
  Button(int pin)
    : pin(pin), state(Idle), lastDebounceTime(0), debounceDelay(50) {}

  /**
   * Método para atualizar o estado do botão.
   * Este método realiza o debounce do botão e atualiza seu estado de acordo
com o tempo e a leitura do pino.
   */
  void update() {
    int buttonState = digitalRead(pin);

    switch (state) {
      case Idle:
        if (buttonState == HIGH) {
          state = Pressed;
          lastDebounceTime = millis();
        }  // if
        break;

      case Pressed:
                if (buttonState == LOW && millis() - lastDebounceTime >
debounceDelay) {
          state = Released;
        }  // if
        break;

      case Released:
```

```
        if (buttonState == LOW) {
          state = Idle;
        }  // if
        break;
    }  // switch
  }    // update

  /**
   * Método para obter o estado atual do botão.
   * @return O estado atual do botão (Idle, Pressed ou Released).
   */
  ButtonState getState() {
    return state;
  }  // getState
};

// Classe do joystick
class Joystick {
private:
  int xPin;  ///< Pino de leitura do eixo X do joystick
  int yPin;  ///< Pino de leitura do eixo Y do joystick

public:
  /**
   * Construtor da classe Joystick.
   * Inicializa o objeto Joystick com os pinos especificados.
   * @param xPin O pino de leitura do eixo X do joystick.
   * @param yPin O pino de leitura do eixo Y do joystick.
   */
  Joystick(int xPin, int yPin)
    : xPin(xPin), yPin(yPin) {}

  /**
   * Método para obter o estado atual do joystick.
    * Este método lê os valores dos pinos X e Y do joystick e determina o
estado do joystick (Neutral, Up, Down, Left ou Right).
```

```
   * @return O estado atual do joystick.
   */
  JoystickState getState() {
    int xValue = analogRead(xPin);
    int yValue = analogRead(yPin);

    if (xValue < 100) {
      return Left;
    } else if (xValue > 800) {
      return Right;
    } else if (yValue < 100) {
      return Down;
    } else if (yValue > 800) {
      return Up;
    } else {
      return Neutral;
    }  // if-else
  }    // getState
};

#endif
```

- **Biblioteca “lcdConfig.h”**

```
C/C++
/**
   @file lcdConfig.h
    @brief Configuração do display LCD e função para atualização do conteúdo
exibido.
   @author Gabriel Finger Conte e Maria Eduarda Pedroso

     Este arquivo contém a definição dos pinos para o display LCD e a
declaração da função
```

```
   para atualizar o conteúdo exibido no LCD.
*/

#ifndef lcdConfig_H
#define lcdConfig_H

#include <LiquidCrystal.h>

// Definição dos pinos para o display LCD
#define RS 12  ///< Pino RS do LCD
#define EN 11  ///< Pino EN do LCD
#define D4 5   ///< Pino D4 do LCD
#define D5 4   ///< Pino D5 do LCD
#define D6 3   ///< Pino D6 do LCD
#define D7 2   ///< Pino D7 do LCD

LiquidCrystal lcd(RS, EN, D4, D5, D6, D7);  ///< Objeto LiquidCrystal para
controle do LCD

  /**
 * Exibe a hora atual no display LCD, incluindo a seleção das horas do
alarme, se aplicável.
 */
  void printHour();

/**
 * Exibe o horário do alarme no display LCD, incluindo a seleção das horas,
minutos ou segundos do alarme, se aplicável.
 */
void printAlarm();

  /**
 * Exibe a lista de tons de música disponíveis no display LCD, indicando o
tom selecionado com um asterisco.
 */
  void printToneList();
```

```
/**
 * Atualiza o conteúdo exibido no display LCD de acordo com o estado atual do sistema.
 * Esta função atualiza o conteúdo do LCD de acordo com o estado atual do sistema, mostrando a hora, o alarme ou a lista de tons de música.
 */
void updateLCD();

#endif
```

- **Implementação da Biblioteca “lcdConfig.ino”**

```
C/C++
/**
   @file lcdConfig.h
    @brief Configuração do display LCD e função para atualização do conteúdo exibido.
   @author Gabriel Finger Conte e Maria Eduarda Pedroso

    Este arquivo contém a lógica da função para atualizar o conteúdo exibido no LCD.
*/

/**
 * Exibe a hora atual no display LCD, incluindo a seleção das horas do alarme, se aplicável.
 */
void printHour() {
   if (estadoAtual == Select_Hour) {   ///< Verifica se o estado atual é a seleção da hora
     lcd.print("*");                   ///< Exibe um asterisco para indicar a seleção da hora
```

```
  }
  lcd.print("Hora: ");
  if (currentHour < 10) {  ///< Verifica se a hora é menor que 10 para exibir um zero à esquerda
    lcd.print("0");
  }
  lcd.print(currentHour);  ///< Exibe a hora
  lcd.print(":");
  if (currentMinute < 10) {  ///< Verifica se os minutos são menores que 10 para exibir um zero à esquerda
    lcd.print("0");
  }
  lcd.print(currentMinute);  ///< Exibe os minutos
  lcd.print(":");
  if (currentSeconds < 10) {  ///< Verifica se os segundos são menores que 10 para exibir um zero à esquerda
    lcd.print("0");
  }
  lcd.print(currentSeconds);              ///< Exibe os segundos
  if (estadoAtual == H_Hours) {           ///< Verifica se o estado atual é a seleção das horas do alarme
    lcd.setCursor(15, 0);                 ///< Posiciona o cursor no final da primeira linha
     lcd.print("h");                          ///< Exibe "h" para indicar as horas do alarme
  } else if (estadoAtual == H_Minutes) {  ///< Verifica se o estado atual é a seleção dos minutos do alarme
    lcd.setCursor(15, 0);                 ///< Posiciona o cursor no final da primeira linha
     lcd.print("m");                          ///< Exibe "m" para indicar os minutos do alarme
  } else if (estadoAtual == H_Seconds) {  ///< Verifica se o estado atual é a seleção dos segundos do alarme
    lcd.setCursor(15, 0);                 ///< Posiciona o cursor no final da primeira linha
```

```
     lcd.print("s");                          ///< Exibe "s" para indicar os
segundos do alarme
  }
}

/**
 * Exibe o horário do alarme no display LCD, incluindo a seleção das horas,
minutos ou segundos do alarme, se aplicável.
 */
void printAlarm() {
   if (estadoAtual == Select_Alarm) {   ///< Verifica se o estado atual é a
seleção do alarme
     lcd.print("*");                   ///< Exibe um asterisco para indicar a
seleção do alarme
  }
  lcd.print("Alarme: ");
   if (alarmHour < 10) {   ///< Verifica se a hora do alarme é menor que 10
para exibir um zero à esquerda
    lcd.print("0");
  }
  lcd.print(alarmHour);  ///< Exibe a hora do alarme
  lcd.print(":");
  if (alarmMinute < 10) {   ///< Verifica se os minutos do alarme são menores
que 10 para exibir um zero à esquerda
    lcd.print("0");
  }
  lcd.print(alarmMinute);                 ///< Exibe os minutos do alarme
  if (estadoAtual == A_Hours) {           ///< Verifica se o estado atual é a
seleção das horas do alarme
    lcd.setCursor(15, 1);                 ///< Posiciona o cursor no final da
segunda linha
     lcd.print("h");                       ///< Exibe "h" para indicar as
horas do alarme
  } else if (estadoAtual == A_Minutes) {  ///< Verifica se o estado atual é a
seleção dos minutos do alarme
```

```
    lcd.setCursor(15, 1);                ///< Posiciona o cursor no final da
segunda linha
     lcd.print("m");                         ///< Exibe "m" para indicar os
minutos do alarme
  } else if (estadoAtual == A_Seconds) {  ///< Verifica se o estado atual é a
seleção dos segundos do alarme
    lcd.setCursor(15, 1);                ///< Posiciona o cursor no final da
segunda linha
     lcd.print("s");                         ///< Exibe "s" para indicar os
segundos do alarme
  }
}

/**
 * Exibe a lista de tons de música disponíveis no display LCD, indicando o
tom selecionado com um asterisco.
 */
void printToneList() {
  if (buzzSelect == 1) {  ///< Verifica se o tom selecionado é o de Mario
    lcd.print("*");        ///< Exibe um asterisco para indicar a seleção do
tom
  }
  lcd.print("Mario");
  lcd.setCursor(0, 1);    ///< Posiciona o cursor no início da segunda linha
  if (buzzSelect == 0) {  ///< Verifica se o tom selecionado é o de Zelda
    lcd.print("*");        ///< Exibe um asterisco para indicar a seleção do
tom
  }
  lcd.print("Zelda");
}

/**
 * Atualiza o conteúdo exibido no display LCD de acordo com o estado atual do
sistema.
 * Esta função atualiza o conteúdo do LCD de acordo com o estado atual do
sistema, mostrando a hora, o alarme ou a lista de tons de música.
```

```
 */
void updateLCD() {

  if (lcdState) {
    lcdState = false;
    lcd.clear();          ///< Limpa o LCD
    lcd.setCursor(0, 0);  ///< Posiciona o cursor no início da primeira linha
    if (estadoAtual == Tone_List) {
      printToneList();
    } else if (estadoAtual == Select_Tone) {
      lcd.print("*");
      lcd.print("Musicas");
    } else {
      printHour();
        lcd.setCursor(0, 1);   ///< Posiciona o cursor no início da segunda
linha
      printAlarm();
    }
  }
}
```

- **Biblioteca “timerConfig.h”**

```
C/C++
/**
   @file timerConfig.h
                                    @note            Adaptado            de
https://github.com/khoih-prog/TimerInterrupt/blob/master/examples/TimerInterr
uptTest/TimerInterruptTest.ino
   @brief Configurações do timer.
   @author Gabriel Finger Conte e Maria Eduarda Pedroso
```

```
    Este arquivo contém as definições das constantes relacionadas ao timer,
bem como a inclusão da biblioteca TimerInterrupt.h
     e a declaração das funções para habilitar, desabilitar e lidar com
interrupções do timer.
*/


#ifndef timerConfig_H
#define timerConfig_H


/* Definição de constantes de Timer */
#define TIMER_INTERRUPT_DEBUG 2
#define _TIMERINTERRUPT_LOGLEVEL_ 2
#define USE_TIMER_1 true
#define TIMER_INTERVAL_MS 100

#if    (defined(__AVR_ATmega644__)    ||    defined(__AVR_ATmega644A__)    ||
defined(__AVR_ATmega644P__)      ||      defined(__AVR_ATmega644PA__)      ||
defined(ARDUINO_AVR_UNO)        ||        defined(ARDUINO_AVR_NANO)        ||
defined(ARDUINO_AVR_MINI)       ||       defined(ARDUINO_AVR_ETHERNET)       ||
defined(ARDUINO_AVR_FIO)         ||         defined(ARDUINO_AVR_BT)         ||
defined(ARDUINO_AVR_LILYPAD)       ||       defined(ARDUINO_AVR_PRO)       ||
defined(ARDUINO_AVR_NG)     ||     defined(ARDUINO_AVR_UNO_WIFI_DEV_ED)     ||
defined(ARDUINO_AVR_DUEMILANOVE)    ||    defined(ARDUINO_AVR_FEATHER328P)    ||
defined(ARDUINO_AVR_METRO)     ||     defined(ARDUINO_AVR_PROTRINKET5)     ||
defined(ARDUINO_AVR_PROTRINKET3)  ||  defined(ARDUINO_AVR_PROTRINKET5FTDI)  ||
defined(ARDUINO_AVR_PROTRINKET3FTDI))
#define USE_TIMER_2 true
#warning Using Timer1
#else
#define USE_TIMER_3 true
#warning Using Timer3
#endif
```

```
/* Importando a Biblioteca do Timer */
#include "TimerInterrupt.h"


/**
   Função para desabilitar o temporizador.
*/
void disableTimer();


/**
   Função para reabilitar o temporizador.
*/
void enableTimer();


/**
   Função que irá ser executada pelo Timer
*/
void TimerHandler();


#endif
```

- **Implementação da Biblioteca “timerConfig.ino”**

```
C/C++
/**
   @file timerConfig.ino
   @brief Implementação das funções relacionadas à configuração do timer.
   @author Gabriel Finger Conte e Maria Eduarda Pedroso
```

```
      Este arquivo contém a implementação das funções relacionadas à
configuração do timer,
   incluindo a desabilitação, reabilitação e atualização do horário atual.
*/

/**
   Função para desabilitar o temporizador.
*/
void disableTimer() {
  ITimer1.disableTimer();
}  // disableTimer

/**
   Função para reabilitar o temporizador.
*/
void enableTimer() {
  ITimer1.enableTimer();
}  // enableTimer

/**
   Função para atualizar o horário atual.
    Aqui é realizada a lógica para atualizar o horário com base no tempo
decorrido.
*/
void updateCurrentTime() {
  // Atualiza a cada interrupção do timer à 10Hz
  currentTick++;
  /* Habilita a atualização do LCD*/
  lcdState = true;

  if (currentTick >= 10) {  // A cada 10 ticks atualiza 1 segundo
    currentTick = 0;
    currentSeconds++;
    if (currentSeconds >= 60) {  // A cada 60 segundos atualiza 1 minuto
      currentSeconds = 0;
      currentMinute++;
```

```
      if (currentMinute >= 60) {  // A cada 60 minutos atualiza 1 hora
        currentMinute = 0;
        currentHour++;
          if (currentHour >= 24) {   // A cada 24 horas, reseta para a hora
atual para 0
          currentHour = 0;
        }  // if currentHour
      }    // if currentMinutes
    }      // if currentSeconds
  }        // if currentTick
}  // updateCurrentTime

/**
   Função chamada pelo Timer para atualizar o horário atual.
*/
void TimerHandler() {
  updateCurrentTime();
}  // TimerHandler
```