Instituto Politécnico do Cávado e do Ave

Licenciatura em Engenharia de Sistemas Informáticos

Sistemas Embebidos em Tempo Real

Trabalho Prático

Students (Grupo 4) :
Francisco Arantes - 23504
Tiago Oliveira - 16622
Luís Ferreira - 23516

Teacher :
Paulo Macedo

27 de dezembro de 2023

Conteúdo

1 Introdução

Num contexto de crescente digitalização e automação, os sistemas embebidos em tempo real desempenham um papel fundamental na transformação de residências convencionais em espaços inteligentes e eficientes. Este projeto propõe-se a desenvolver e integrar uma série de sistemas embebidos em tempo real num mockup de uma casa real, proporcionando uma perspetiva prática e aplicada das tecnologias emergentes no âmbito da Smart Housing.

O mockup, representando fielmente os diversos espaços de uma residência, servirá como cenário para a implementação de quatro sistemas distintos, cada um abordando desafios específicos relacionados ao conforto, segurança e automação residencial. Ao explorar e integrar esses sistemas, procuramos criar uma sinergia que não só demonstre a aplicação prática dos sistemas embebidos em tempo real, mas também ofereça uma solução holística para tornar a vida quotidiana mais conveniente e eficiente.

Figura 1: Smart Home

Sistema A - Controlo de Iluminação Interior:

O primeiro sistema visa otimizar a iluminação interna com base na intensidade da luz solar, proporcionando assim uma iluminação constante e uma gestão eficiente de energia. O uso de LEDs e sensores LDR proporcionará uma resposta dinâmica ao ambiente, ajustando-se de forma inteligente ás condições de luminosidade.

Sistema B - Controlo de Climatização:

O segundo sistema concentra-se no controlo da temperatura ambiente através de um sistema de ventilação. A ventoinha é acionada automaticamente quando a temperatura ultrapassa 24 graus Celsius, proporcionando não apenas conforto térmico, mas também indicando visualmente o estado do sistema através de LEDs e de um display LCD.

Sistema C - Sistema de Segurança (Alarme):

O terceiro sistema introduz uma camada de segurança, detetando movimentos de intrusos com um sensor PIR. O alarme, composto por um sinal luminoso intermitente e um sinal sonoro característico, reforça a proteção residencial, enquanto um botão de desarme oferece controlo ao utilizador.

Sistema D - Automatização Adicional para Smart Housing:

O quarto sistema é um componente adicional que visa automatizar tarefas diversas na residência, como estacionamento, rega e controlo de acessos. A integração deste sistema adiciona uma camada de versatilidade à casa inteligente.

Além disso, cada sistema será enriquecido com um requisito adicional à escolha do grupo, como a utilização de interrupts, criação de interfaces gráficas para monitorização remota, multitasking ou análise de performance. Este projeto busca, assim, ir além da simples implementação técnica, proporcionando uma visão completa do potencial dos sistemas embebidos em tempo real no contexto da Smart Housing.

2 Análise de Requisitos

A análise de requisitos, funcionais e não funcionais, é essencial para garantir que os sistemas embebidos atendam eficazmente ás necessidades dos utilizadores na *Smart Housing*, estabelecendo critérios operacionais e condições de desempenho.

2.1 Sistema A - Controlo de Iluminação Interior

Requisitos Funcionais:

- **Medição da Intensidade de Luz:** O sistema deve ser capaz de medir a intensidade da luz solar através do sensor *LDR*.
- **Ajuste Dinâmico da Iluminação:** Com base na intensidade medida, o sistema deve ajustar a iluminação interior utilizando *LEDs* em cinco escalas predefinidas.
- **Monitorização em Série:** Apresentar, no monitor de série, os valores de entrada do sensor e saída do *LED* para fins de depuração e acompanhamento.

Requisitos Não Funcionais:

- **Eficiência Energética:** O sistema deve visar a eficiência energética, garantindo que a iluminação é ajustada de forma a otimizar o consumo de energia.
- **Resposta em Tempo Real:** A resposta do sistema ao ajuste da iluminação deve ser rápida e em tempo real para manter uma iluminação constante.

2.2 Sistema B - Controlo de Climatização

Requisitos Funcionais:

- **Controlo de Temperatura:** O sistema deve monitorizar a temperatura ambiente através de um sensor e acionar a ventoinha para arrefecimento quando a temperatura ultrapassar 24°C.
- **Indicação de Estado:** Utilizar *LEDs* para indicar o estado da ventoinha (arrefecimento ou estabilizado) e apresentar o estado no display *LCD*.
- **Controlo do *display*:** A luminosidade do *display LCD* deve ser controlada por meio de um potenciómetro.

Requisitos Não Funcionais:

- **Eficiência Energética:** O sistema deve ser projetado para otimizar o consumo de energia, desligando a ventoinha quando não é necessária.
- **Interface do Utilizador:** Garantir que a informação apresentada no display *LCD* é clara e facilmente compreensível.

2.3 Sistema C - Sistema de Segurança (Alarme)

Requisitos Funcionais:

- **Deteção de Intrusões:** O sistema deve detetar movimentos de intrusos por meio de um sensor *PIR*.
- **Alarme Luminoso e Sonoro:** Acionar um sinal luminoso intermitente e um sinal sonoro de alarme por 10 segundos sempre que for detetado movimento.
- **Desarmamento do Alarme:** Integrar um botão de pressão para desarmar o alarme.

Requisitos Não Funcionais:

- **Segurança:** Garantir a segurança do sistema, permitindo o desarme apenas através do botão designado.
- **Recetividade do Alarme:** O alarme deve repetir o sinal sonoro a intervalos de 5 segundos até ser desarmado.

2.4 Sistema D

Requisitos Funcionais:

- **Controlo de Iluminação:** O sistema deve ser capaz de ligar/desligar luzes em resposta aos comandos recebidos pelo sensor *IR*.
- **Controlo da Garagem:** O sistema deve ativar um motor para abrir a garagem quando receber comandos específicos via *IR*.

Requisitos Não Funcionais:

- **Eficiência Energética:** O sistema deve ser otimizado para minimizar o consumo de energia, desligando luzes ou motores quando não estão em uso.
- **Resposta Rápida ao Comando *IR*:** Garantir que o sistema responde de forma rápida e precisa aos comandos recebidos através do sensor *IR*.

2.5 Requisitos Adicionais

Figura 2: Interrupts

Utilização de *interrupts*:

Implementar *interrupts* em um dos sistemas para garantir uma resposta rápida a eventos específicos.

Figura 3: Monitorização

Interface Gráfica para Monitorização Remota:

Criar uma interface gráfica para monitorização remota em tempo real em um dos sistemas.

Figura 4: *Multitasking*

Multitasking:

Implementar *multitasking*, com *Timer* ou *RTOS*, para gerir concorrentemente o tempo de execução de 2 ou mais componentes em um dos sistemas.

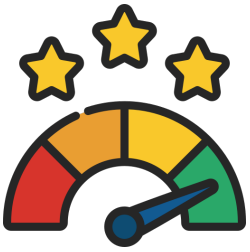

Figura 5: Análise de Performance

Análise de Performance:

Realizar uma análise de performance comparativa entre duas versões do programa de um sistema, considerando a eficiência energética.

3 Especificação do sistema

3.1 Sistema A - Controlo de Iluminação Interior

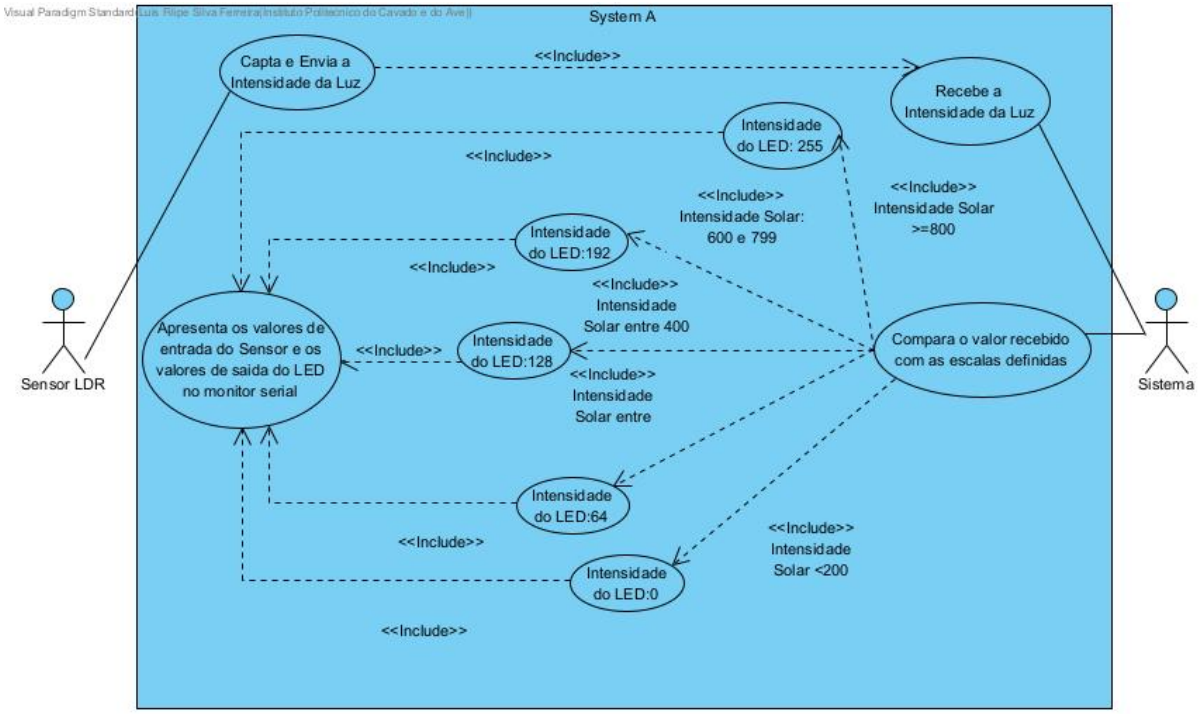

Figura 6: Caso de Usos Sistema A

Arquitetura do Sistema Hardware:

- **Sensor LDR:** Mede a intensidade da luz solar.
- **LED:** Ajusta a iluminação em conformidade com as escalas de intensidade.
- **Monitor de Série:** Apresenta valores de entrada do sensor e saída do LED.

Arquitetura do Sistema Software:

- **Leitura do Sensor:** Captura a intensidade da luz medida pelo sensor LDR.
- **Lógica de Controlo:** Baseada nas escalas de intensidade, determina o ajuste da iluminação.
  1. $\geq$ 800 LED 255;
  2. $\geq$ 600 e $<$ 800 LED 192;
  3. $\geq$ 400 e $<$ 600 LED 128;
  4. $\geq$ 200 e $<$ 400 LED 64;
  5. $<$ 200 LED 0.
- **Controlo do LED:** Ativa o LED de acordo com as escalas definidas.
- **Monitorização:** Apresenta os valores do sensor e LED no monitor de série.

3.2 Sistema B - Controlo de Climatização

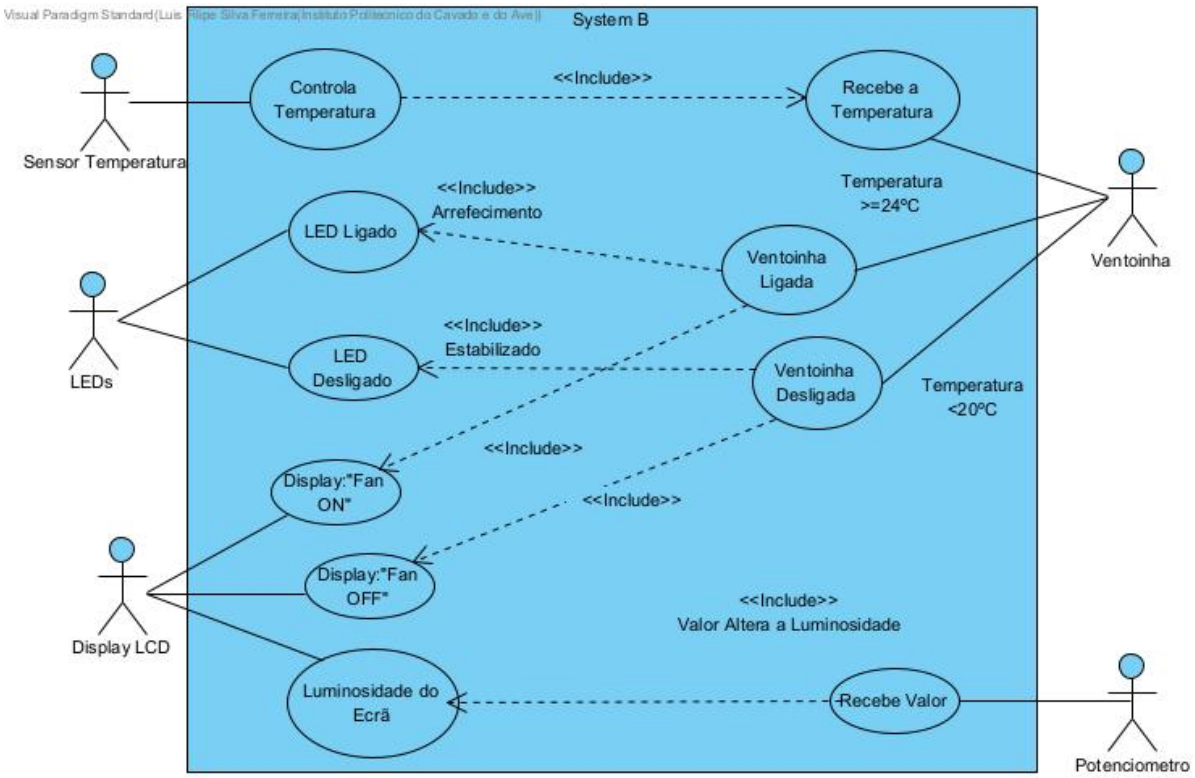

Figura 7: Caso de Usos Sistema B

Arquitetura do Sistema Hardware:

- **Sensor de Temperatura:** Mede a temperatura ambiente.
- **Ventoinha:** Controla o arrefecimento com base na temperatura.
- **LEDs e Display LCD:** Indicam o estado do sistema e a temperatura.
- **Potenciómetro:** Controla a luminosidade do display.

Arquitetura do Sistema Software:

- **Leitura do Sensor:** Captura os valores de temperatura.
- **Lógica de Controlo:** Decide quando ligar ou desligar a ventoinha.
  1. $\geq$ 24°C Ligar ventoinha;
  2. $<$ 20°C Desligar ventoinha.
- **Controlo dos LEDs e Display:** Indica o estado e apresenta a temperatura.
  1. Quando a ventoinha estiver ligada, LED vermelho liga e LED verde desliga;
  2. Quando a ventoinha estiver desligada, LED verde liga e LED vermelho desliga;
  3. Na 1^a linha do LCD apresenta o estado da ventoinha ("Fan ON/ Fan OFF");
  4. Na 2^a linha do LCD apresenta o temperatura ("Temp: xxx^{o}C").
- **Controlo do Potenciómetro:** Ajusta a luminosidade do display.

3.3 Sistema C - Sistema de Segurança (Alarme)

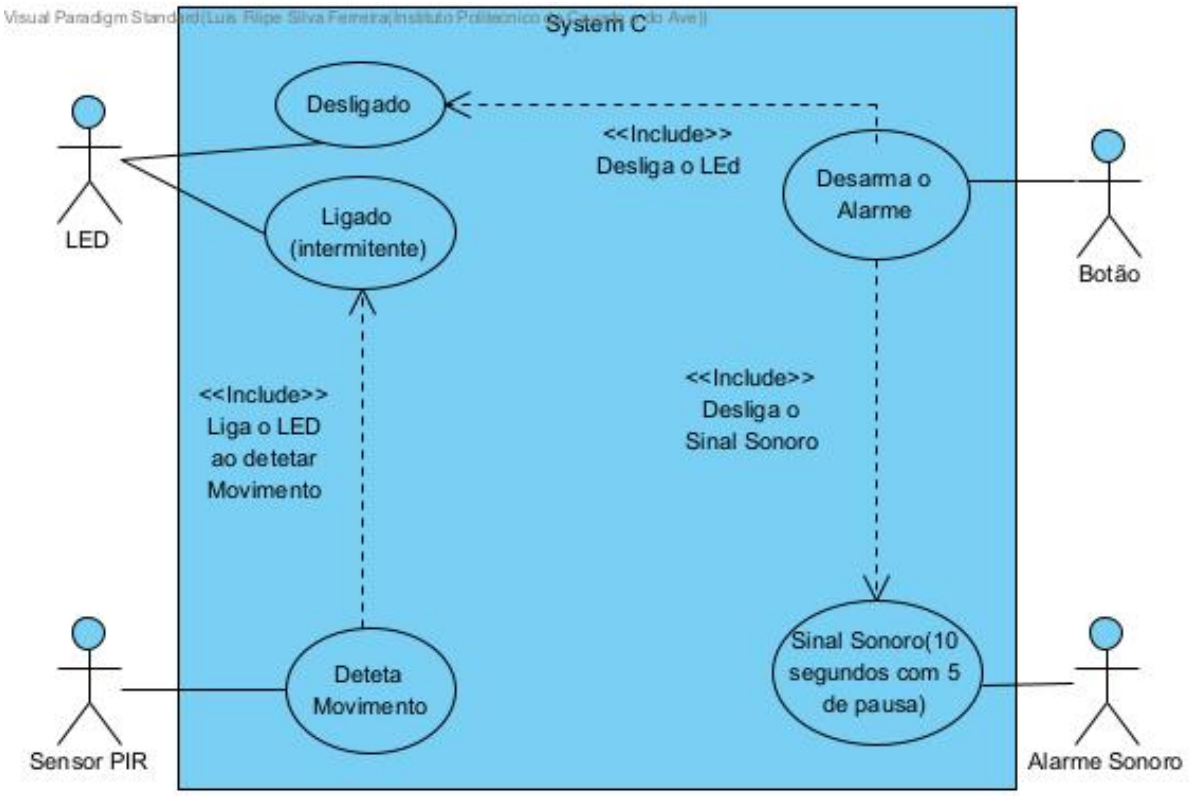

Figura 8: Caso de Usos Sistema C

Arquitetura do Sistema Hardware:

- **Sensor PIR:** Deteção de movimentos intrusivos.
- **LED:** Sinal luminoso intermitente.
- **Buzzer:** Sinal sonoro de alarme.
- **Botão de Pressão:** Para desarmar o alarme.
- **Monitor de Série:** Monitorização do sistema.

Arquitetura do Sistema Software:

- **Leitura do Sensor PIR:** Detecta movimentos intrusivos.
- **Lógica de Alarme:** Aciona o LED e o buzzer em resposta ao movimento.
  1. Ativa LED vermelho intermitente;
  2. Sinal sonoro com duração de 10 segundos;
  3. Repetir esse sinal sonoro com intervalos de 5 segundos.
- **Controlo do Botão:** Permite o desarme do alarme, com o uso de um *Interrupt* do tipo *FALLING*.
- **Monitorização:** Apresenta informações no monitor de série.

3.4 Sistema D - Sistema de Controlo por Comando

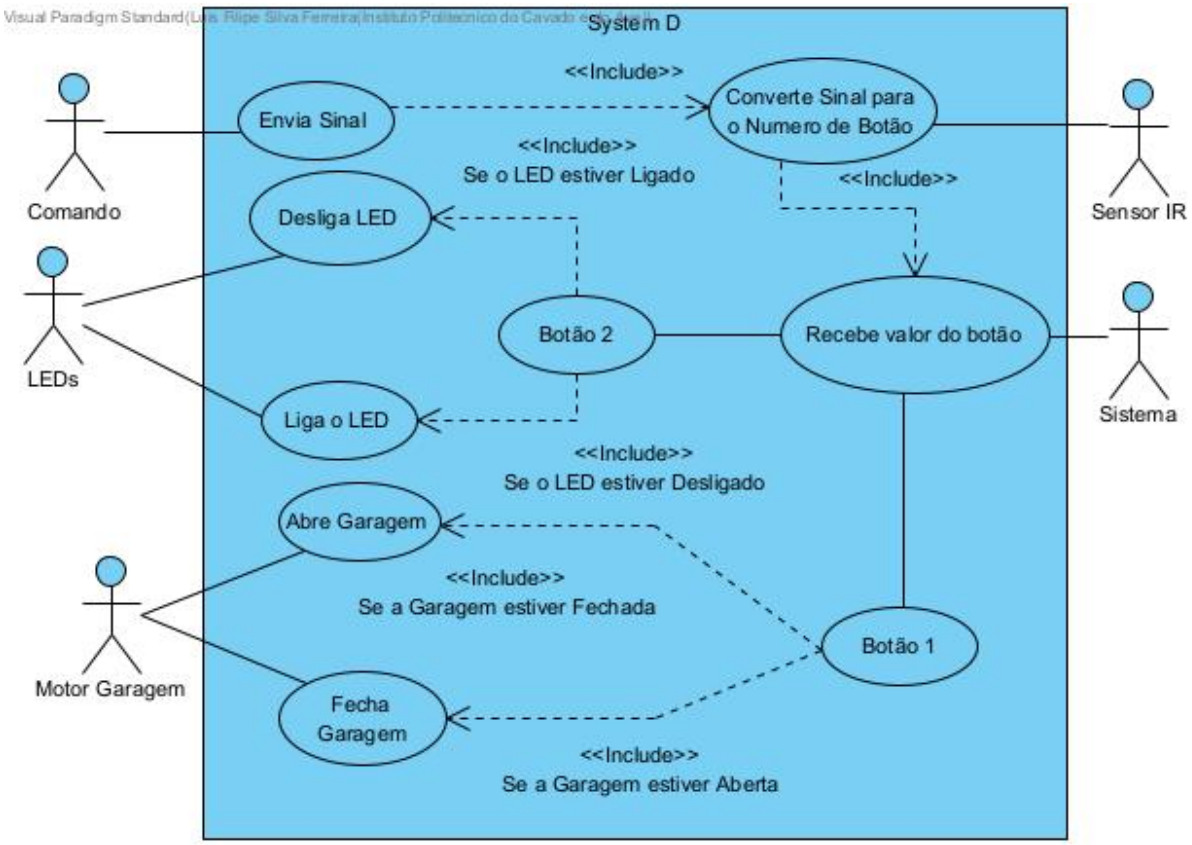

Figura 9: Caso de Usos Sistema D

Arquitetura do Sistema Hardware:

- **Sensor IR:** Receber comandos infravermelhos.
- **LEDs:** Sinal luminoso.
- **Motor:** Motor eletrónico para abrir e fechar a garagem em resposta aos comandos IR..
- **Botão de Pressão:** Para desarmar o alarme.
- **Monitor de Série:** Monitorização do sistema.

Arquitetura do Sistema Software:

- **Leitura do Sensor IR:** Interpretar comandos recebidos.
- **Lógica de Controlo:** Analisar comandos IR recebidos e determinar ações correspondentes.
  1. **Butão 1:** Controla um LED da divisão 3;
  2. **Butão 2:** Controla um LED da divisão 4;
  3. **Butão 3:** Controla o motor da garagem.

4 Modelo de conceção

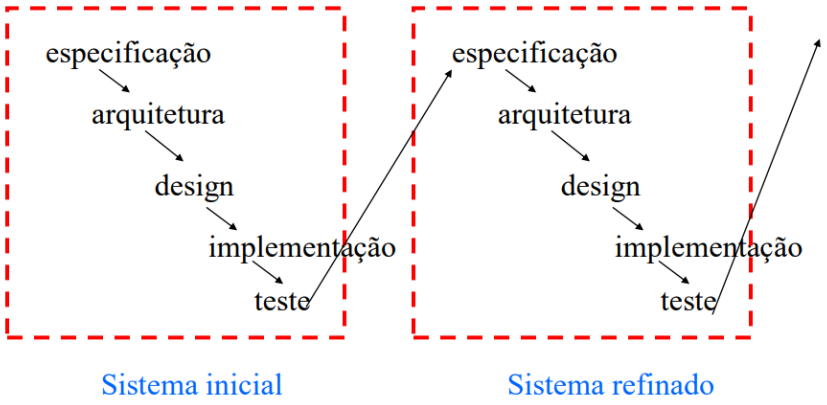

Figura 10: Modelo de Refinamento Sucessivo

O processo de integração dos quatro sistemas independentes foi conduzido de forma gradual, seguindo os princípios do Modelo de Refinamento Sucessivo. Este modelo consiste numa abordagem iterativa composta por cinco fases distintas: *especificação*, *arquitetura*, *design*, *implementação* e *teste*. Em cada iteração, um novo sistema era desenvolvido e posteriormente aprimorado com a integração do sistema subsequente, um método que promoveu a evolução contínua do conjunto, como ilustrado na Figura 10.

Figura 11: Especificação

Especificação:

Cada sistema foi inicialmente especificado, delineando os requisitos e funcionalidades específicos. Os sistemas individuais foram concebidos com base em necessidades particulares, como iluminação, climatização, segurança e automação.

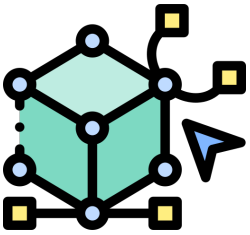

Figura 12: Arquitetura

Arquitetura:

Uma vez especificados, os sistemas foram arquitetados considerando a estrutura geral e a inter-relação entre seus componentes. Foi projetada uma arquitetura modular para facilitar a integração futura e garantir a coesão do conjunto.

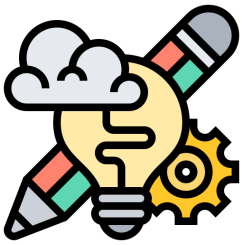

Figura 13: Design

Design:

O design de cada sistema foi elaborado com base na arquitetura definida. Componentes específicos, como LEDs, sensores e atuadores, foram projetados para cumprir suas funções individuais, enquanto elementos comuns foram identificados para promover a interoperabilidade.

Figura 14: Implementação

Implementação:

Cada sistema foi implementado conforme o design estabelecido. A implementação considerou requisitos específicos e a funcionalidade independente de cada sistema, garantindo que cada um pudesse operar autonomamente.

Figura 15: Test

Teste:

Após a implementação individual de cada sistema, foram realizados testes exaustivos para validar a funcionalidade e identificar possíveis melhorias. Esta fase assegurou que cada sistema estivesse operando corretamente antes da etapa de integração.

A integração dos sistemas ocorreu de forma progressiva, a começar com a combinação de dois sistemas e progredindo para a integração completa dos quatro sistemas. Cada etapa de integração foi seguida por uma fase de refinamento, na qual o sistema resultante foi melhorado e otimizado com a inclusão do próximo sistema.

Benefícios da Abordagem de Integração Iterativa:

Figura 16: Desenvolvimento Incremental

Desenvolvimento Incremental:

A integração foi conduzida em passos graduais, permitindo melhorias contínuas e ajustes à medida que novos sistemas eram incorporados.

Figura 17: Validação Contínua

Validação Contínua:

Testes regulares foram realizados durante cada fase de integração, garantindo que cada sistema integrado mantivesse sua funcionalidade e contribuísse para o conjunto de maneira eficaz.

Figura 18: Flexibilidade e Adaptabilidade

Flexibilidade e Adaptabilidade:

A abordagem iterativa permitiu a adaptação constante aos requisitos específicos, possibilitando uma resposta dinâmica às necessidades emergentes.

Ao seguir esta metodologia, alcançou-se uma integração harmoniosa dos quatro sistemas, resultando num conjunto coeso, eficiente e capaz de atender às diversas demandas de um ambiente de Smart Housing.

5 Construção do sistema

Antes de procedermos á montagem de cada respetivo sistema na maqueta, começamos por usar a plataforma *TinkerCad*.

O *TinkerCad* é uma plataforma *Web* gratuita para *design 3D*, eletrónica e codificação, nesta plataforma temos a possibilidade de construir e simular a execução de uma enorme variedade de circuitos. 2

5.1 Sistema A

Os componentes necessários para a construção deste sistema foram essencialmente uma *Breadboard*, 1 *LED*, 1 resistência de 220 *Ohms*, 1 resistência de 10K *Ohms*, sensor de luz, o *Arduino*, um cabo USB tipo A-B para ligar o *Arduino* ao computador e cabos elétricos para ligações entre o *Arduino* e a *breadboard*.

Em suma, consoante o valor da intensidade de luz solar captado pelo sensor de luz, a intensidade do *LED* é ajustada de acordo com 5 escalas e retornado para o monitor de série r o valor de entrada do sensor e saída do *LED*.

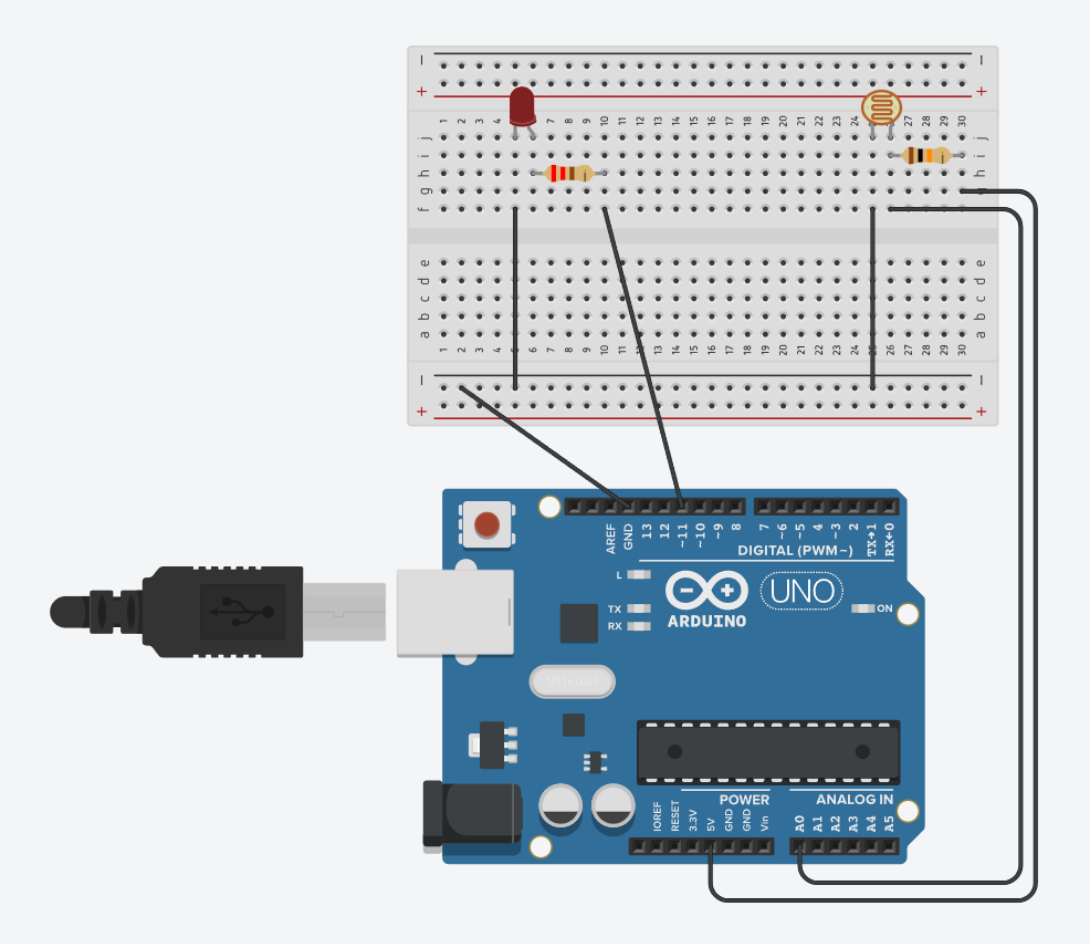

Figura 19: Circuito do sistema A

5.2 Sistema B

Os componentes utilizados para a montagem deste sistema foram uma *breadboard*, 1 *LED* vermelho para quando a ventoinha estiver ligada, 1 *LED* verde para quando a ventoinha desligar-se, 3 resistência de 220 *Ohms*, sensor de temperatura, 1 potenciómetro, 1 *display LCD*, 1 ventoinha, o *Arduino*, um cabo USB tipo A-B para ligar o Arduino ao computador e cabos elétricos para ligações entre o *Arduino* e a *breadboard*.

Para a montagem na maqueta utilizamos um sensor de temperatura e humidade, de modo a, enriquecer o sistema.

Essencialmente o *Arduino* recebe os valores do sensor de temperatura e humidade, quando a temperatura ultrapassar os 24 graus *Celsius* a ventoinha é ligada juntamente com *LED* vermelho, quando a temperatura baixar para para os 20 graus a ventoinha é desligada e o *LED* vermelho desliga-se e o *LED* verde liga-se. Os valores da temperatura e humidade são apresentados no *display LCD* juntamente com o estado da ventoinha (*ON/OFF*). O potenciómetro manipula a luminosidade do *display LCD*.

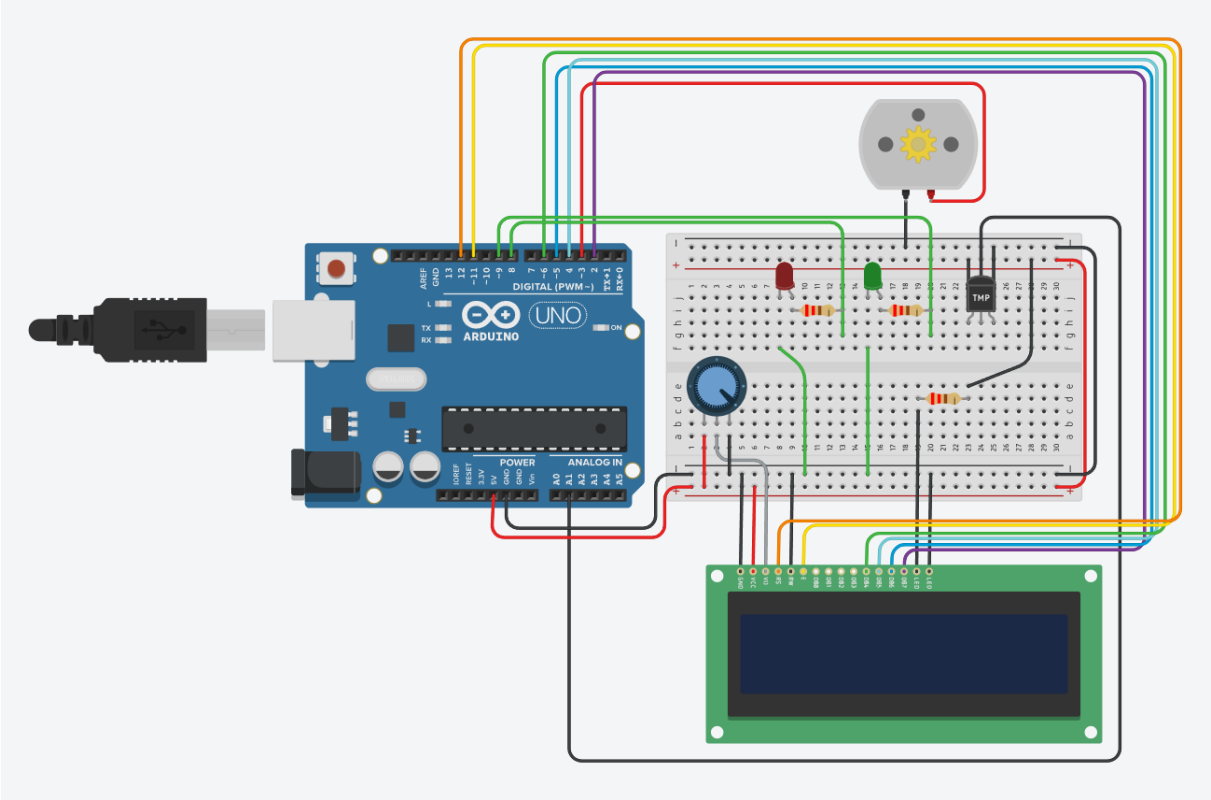

Figura 20: Circuito do sistema B

5.3 Sistema C

Os componentes utilizados para a montagem deste sistema foram uma *breadboard*, 1 *LED* vermelho para ligar-se de forma intermitente quando movimento for detetado, 1, 2 resistência de 220 *Ohms*, sensor de movimento, 1 *buzzer* a servir de alerta, 1 botão para desligar o alarme, o *Arduino*, um cabo USB tipo A-B para ligar o Arduino ao computador e cabos elétricos para ligações entre o *Arduino* e a *breadboard*.

Essencialmente o *Arduino* recebe os valores do sensor de movimento, assim que é detetado movimento, o *LED* liga de forma intermitente e o *buzzer* emite um sinal sonoro com uma duração de 10 segundos com intervalos de 5 segundos, após o botão ser pressionado, o *LED* desliga-se e o *buzzer* deixa de emitir som.

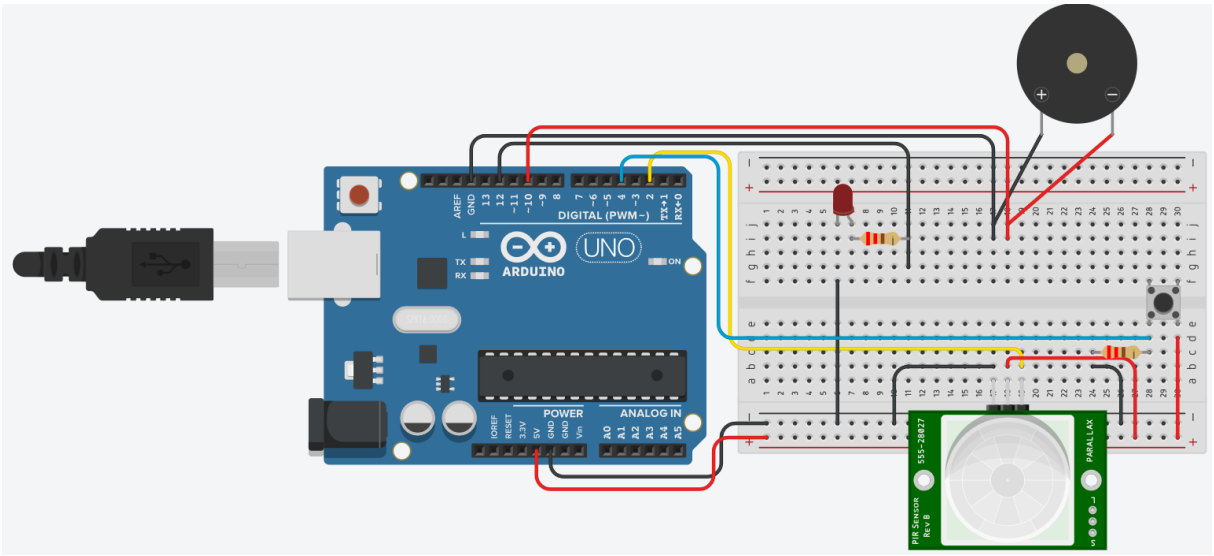

Figura 21: Circuito do sistema C

5.4 Sistema D

Os componentes utilizados para a montagem deste sistema foram uma *breadboard*, 1 motor *servo*, 1 sensor infravermelhos, 1 *LED*, 1 resistência de 220 *Ohms*, 1 comando, o *Arduino*, um cabo USB tipo A-B para ligar o Arduino ao computador e cabos elétricos para ligações entre o *Arduino* e a *breadboard*.

Essencialmente o sensor infravermelhos está constantemente á escuta de um sinal do comando, após receber o sinal proveniente do botão 1 (opcional na implementação) gira 90 graus no sentido contrário dos ponteiros do relógio o motor *servo*, quando receber mais uma vez o sinal do botão 1 gira no sentido dos ponteiros do relógio o motor *servo*.

O mesmo é produzido para o *LED*, o mesmo é ligado e desligado á medida que o sinal do botão 2 é recebido.

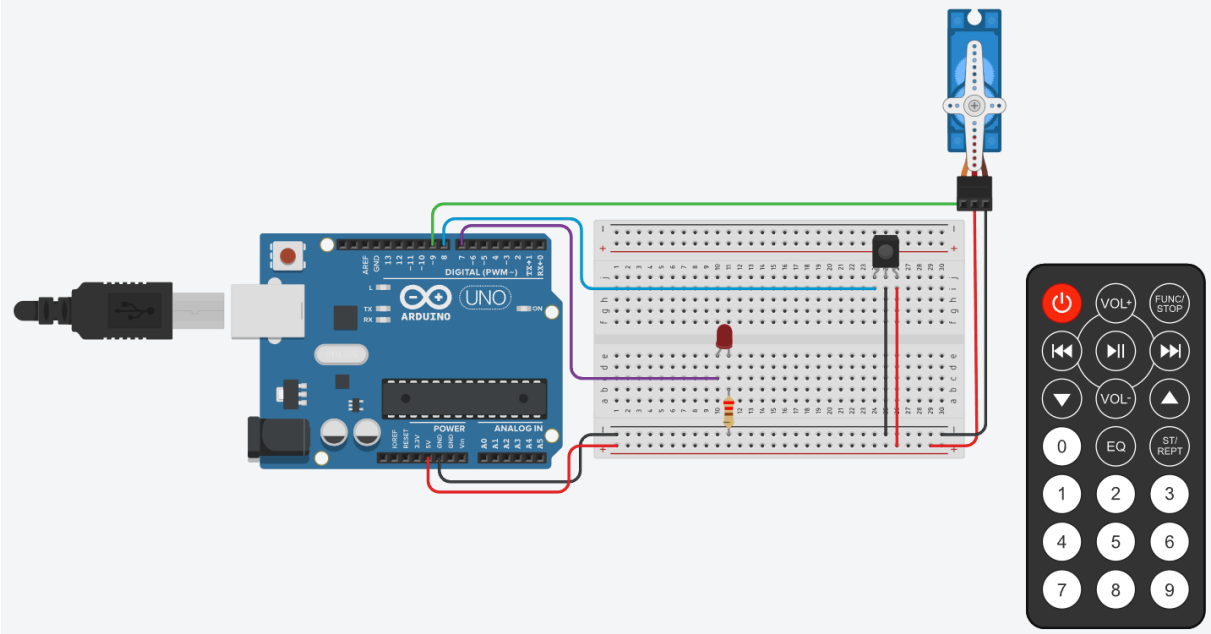

Figura 22: Circuito do sistema D

6 Codificação

6.1 Código Sistema A

```
void TaskSystemA(void *pvParameters) {
  // Avoid compiler warning about unused parameter
  (void)pvParameters;

  // Variables to store light sensor and LED values
  int val_light = 0, val = 0;

  // Infinite loop for continuous task execution
  while (1)
  {
    // Read analog value from the light sensor
    val_light = analogRead(Light);

    // Map the light sensor value to LED brightness
    if (val_light >= 800) val = 255;
    else if (val_light >= 600) val = 192;
    else if (val_light >= 400) val = 128;
    else if (val_light >= 200) val = 64;
    else val = 0;

    // Print light sensor and LED values to Serial Monitor
    Serial.print("Light Sensor: ");
    Serial.print(val_light);
    Serial.print(" Light Value: ");
    Serial.println(val);

    // Adjust LED brightness based on light sensor value
    if (statesLed[0] != 0) analogWrite(Room3, val);
    if (statesLed[1] != 0) analogWrite(Room4, val);

    // Delay to control the update rate of the task
    vTaskDelay(pdMS_TO_TICKS(100));
  }
}
```

Listing 1: SystemA.ino

O código apresenta a implementação de um sistema de controlo de iluminação interior utilizando um Arduino, um sensor de luz (LDR), e LEDs. O conceito subjacente a este sistema é a adaptação dinâmica da iluminação do espaço interior com base na intensidade da luz solar. O sensor LDR realiza leituras da intensidade luminosa ambiente, e o algoritmo mapeia essas leituras para diferentes níveis de brilho dos LEDs, criando cinco escalas

distintas. Desta forma, à medida que a luz solar varia, a iluminação interior é ajustada para garantir uma luminosidade constante e promover uma maior eficiência energética.

O código em execução contínua (loop infinito) lê o valor analógico do sensor de luz, determina o nível de brilho correspondente e ajusta a intensidade dos LEDs de acordo. Os valores de entrada do sensor e a saída dos LEDs são apresentados no monitor de série para acompanhamento e depuração.

Essencialmente, este sistema representa uma abordagem prática para criar um ambiente iluminado de forma inteligente, adaptando-se de forma eficaz ás condições de luz ambiente, proporcionando simultaneamente uma gestão eficiente da energia.

6.2 Código Sistema B

```
float hum, temp; // Global variables to store humidity and
   temperature

void TaskSystemB(void *pvParameters) {
  // Avoid compiler warning about unused parameter
  (void)pvParameters;

  // Flag to track whether cooling is active
  bool isCooling = false;

  // Infinite loop for continuous task execution
  while (1) {
    // Read humidity and temperature from the DHT sensor
    hum = dht.readHumidity();
    temp = dht.readTemperature();

    // Update LCD with temperature and humidity values
    lcd.setCursor(5, 1);
    lcd.print(int(temp));
    lcd.setCursor(13, 1);
    lcd.print(int(hum));

    // Cooling control based on temperature
    if (!isCooling && temp > 24) {
      digitalWrite(Room1R, HIGH);
      digitalWrite(Room1G, LOW);
      digitalWrite(AC, HIGH);
      lcd.setCursor(0, 0);
      lcd.print("Fan ON ");
      isCooling = true;
    } else if (isCooling && temp < 20) {
      digitalWrite(Room1R, LOW);
      digitalWrite(Room1G, HIGH);
      digitalWrite(AC, LOW);
      lcd.setCursor(0, 0);
      lcd.print("Fan OFF");
      isCooling = false;
    }

    // Delay to control the update rate of the task
    vTaskDelay(pdMS_TO_TICKS(100));
  }
}

void requestEvent() {
```

```
  // Respond to I2C master request with temperature and
   humidity values
46  Wire.write((uint8_t*)&temp, sizeof(float));
  Wire.write((uint8_t*)&hum, sizeof(float));
48 }
```

Listing 2: SystemB.ino

O código apresenta a implementação de um sistema de controlo de climatização que utiliza um sensor de temperatura e humidade (DHT sensor), uma ventoinha, LEDs e um display LCD. O sistema regula a temperatura ambiente ativando a ventoinha quando a temperatura ultrapassa os 24 graus Celsius e desativando-a quando a temperatura é inferior a 20 graus Celsius.

O display LCD exibe informações relevantes, como o estado da ventoinha ("Fan ON"ou "Fan OFF") na 1ª linha e a temperatura atual na 2ª linha. Além disso, dois LEDs indicam visualmente o estado do ambiente: o LED vermelho liga quando a ventoinha está em funcionamento, enquanto o LED verde indica que o ambiente está estabilizado.

O controle da luminosidade do display LCD é realizado através de um potenciómetro, permitindo ajustar a visibilidade conforme necessário. Adicionalmente, o código responde a solicitações de um mestre I2C, fornecendo os valores de temperatura e humidade quando solicitados.

Em resumo, este sistema visa criar um ambiente climatizado e eficiente, adaptando-se dinamicamente às condições de temperatura e proporcionando feedback visual por meio dos LEDs e do display LCD.

6.3 Código Sistema C

```
// Function to play a tone on the buzzer
void playTone(int frequency) {
  unsigned int period = 1000000 / frequency;
  unsigned int halfPeriod = period / 2;
  unsigned int cycles = frequency * 125 / 1000;
  unsigned int i;

  // Generate the specified tone
  for (i = 0; i < cycles; ++i) {
    digitalWrite(Buzzer, HIGH);
    delayMicroseconds(halfPeriod);
    digitalWrite(Buzzer, LOW);
    delayMicroseconds(halfPeriod);
  }
}
```

Listing 3: SystemC.ino

Este código implementa uma função chamada playTone que é responsável por gerar um som em um buzzer com base em uma frequência especificada.

```
bool isActive = false, found = false;

void TaskSystemC(void *pvParameters) {
  // Avoid compiler warning about unused parameter
  (void)pvParameters;

  unsigned long time;
  bool isToAwait = false;
  byte ledState = HIGH;

  // Infinite loop for continuous task execution
  while (1) {
    // Update LCD with isActive state
    lcd.setCursor(8, 0);
    lcd.print(isActive);

    // If not active, delay and continue to the next iteration
    if (!isActive) {
      digitalWrite(Buzzer, LOW);
      digitalWrite(Room2, LOW);
      vTaskDelay(pdMS_TO_TICKS(100));
      continue;
    }

    // PIR sensor detected motion
    if (digitalRead(PIR) && !found) {
      time = millis();
      found = true;
      isToAwait = false;
    }

    // Execute actions when motion is detected
    if (found && !isToAwait) {
      playTone(1000);
      digitalWrite(Room2, ledState);
      ledState = !ledState;

      // Check if the elapsed time is greater than 10 seconds
      if (millis() - time > 10000) {
        digitalWrite(Buzzer, LOW);
        ledState = LOW;
        digitalWrite(Room2, LOW);
        isToAwait = true;
        time = millis();
      }
    }
```

```
47
    // Wait for 5 seconds after the action and reset
49  if (isToAwait && millis() - time > 5000) {
      isToAwait = false;
51    time = millis();
    }
53
    // Delay to control the update rate of the task
55  vTaskDelay(pdMS_TO_TICKS(100));
  }
57}
```

Listing 4: SystemC.ino

Este código representa a implementação de um sistema de segurança que utiliza um sensor de movimento PIR (Passive Infrared). Quando o sistema está ativo (isActive é verdadeiro) e o sensor PIR deteta movimento, são desencadeadas várias ações de alarme. Estas ações incluem a ativação de um sinal sonoro (Buzzer), a alternância do estado de um LED (Room2) entre ligado e desligado para criar um efeito intermitente vermelho, e a medição do tempo de duração dessas ações.

O sistema aguarda 5 segundos após cada acionamento do alarme antes de poder ser ativado novamente, garantindo intervalos entre eventos consecutivos. O tempo decorrido desde o início até o final de cada iteração do ciclo é registado e impresso no terminal série para monitorização de desempenho.

Este sistema de segurança proporciona alertas visuais e sonoros quando é detetado movimento pelo sensor PIR, e é possível desativá-lo ao pressionar um botão, embora a implementação específica do botão não esteja incluída no código fornecido.

```
void interruptHandler() {
  // Toggle the isActive state and reset variables
  isActive = !isActive;
  found = false;
  delay(10); // Small delay to debounce the button
  digitalWrite(Buzzer, LOW);
  digitalWrite(Room2, LOW);
}
```

Listing 5: SystemC.ino

Este código implementa uma função chamada interruptHandler que é responsável por parar o Buzzer e desativar o alarme quando o butão é pressionado.

6.4 Código Sistema D

```
// Return -1, if supplied code does not map to a key.
int mapCodeToButton(unsigned long code) {
  // Check for codes from this specific remote
  if ((code & 0x0000FFFF) == 0x0000FF00) {

    // No longer need the lower 16 bits. Shift the code by 16
    // to make the rest easier.
    code >>= 16;
    // Check that the value and inverse bytes are complementary
    .
    if (((code >> 8) ^ (code & 0x00FF)) == 0x00FF) {
      return code & 0xFF; // Extract the lower 8 bits as the
   button code
    }
  }
  return -1; // Return -1 if the code doesn't match the
   expected format
}

int readInfrared() {
  int result = -1;
  // Check if we've received a new code
  if (IrReceiver.decode()) {
    // Get the infrared code
    unsigned long code = IrReceiver.decodedIRData.
   decodedRawData;
    // Map it to a specific button on the remote
    result = mapCodeToButton(code);
    // Enable receiving of the next value
    IrReceiver.resume();
  }
  return result;
}
```

Listing 6: SystemD.ino

Este código implementa uma função chamada *readInfrared* que é responsável por parar receber o valor do infravermelho do comando quando é pressionado, e também implementa uma função chamada *mapCodeToButton* que é responsável por descodificar o hexadecimal do valor do comando para um valor mais simples de ler.

```
void openGate() {
  for (int i = 5; i < 170; i++) {
    myservo.write(i);
    delay(30);
  }
}

void closeGate() {
  for (int i = 170; i > 5; i--) {
    myservo.write(i);
    delay(30);
  }
}
```

Listing 7: SystemD.ino

Este código implementa duas funções chamadas *openGate* e *closeGate* que são responsáveis por abrir e fechar o portão.

```
void TaskSystemD(void *pvParameters)
{
  (void)pvParameters;
  unsigned long time;
  int button = -1;
  bool isOpen = false;

  while (1)
  {
    button = readInfrared();

    switch (button) {
      case 12: // Button 1
        // Toggle LED state for Room3
        statesLed[0] = !statesLed[0];
        digitalWrite(Room3, LOW);
        break;
      case 24: // Button 2
        // Toggle LED state for Room4
        statesLed[1] = !statesLed[1];
        digitalWrite(Room4, LOW);
        break;
      case 94: // Button 3
        // Toggle gate state (open/close)
        if (isOpen) closeGate();
        else openGate();
        isOpen = !isOpen;
        break;
```

```
29        default:
            // No action for other buttons
31          break;
        }
33
      vTaskDelay(pdMS_TO_TICKS(100));
35    }
}
```

Listing 8: SystemD.ino

A função *TaskSystemD* desempenha um papel crucial num sistema mais amplo que envolve o controlo de LEDs e de um portão através de um comando remoto infravermelho.

Em cada iteração do ciclo, o código verifica se há novos códigos provenientes do comando remoto infravermelho através da função *readInfrared*.

Ao detetar um novo código, a função mapCodeToButton é utilizada para associar esse código a um botão específico no comando remoto. A tarefa, então, realiza ações específicas com base no botão pressionado.

- Botão 1 (Código 12): Alternar o estado do LED associado à Room3. Se estiver aceso, é desligado; se estiver apagado, é ligado.
- Botão 2 (Código 24): Alternar o estado do LED associado à Room4 da mesma forma que o Botão 1.
- Botão 3 (Código 94): Alternar o estado do portão entre aberto e fechado. Se o portão estiver aberto, será fechado, e vice-versa. O estado atual do portão é mantido numa variável chamada isOpen.

Esta tarefa é fundamental para a interação do sistema com o comando remoto infravermelho, permitindo ao utilizador controlar a iluminação (LEDs) e o estado do portão remotamente.

7 Testes/Resultados

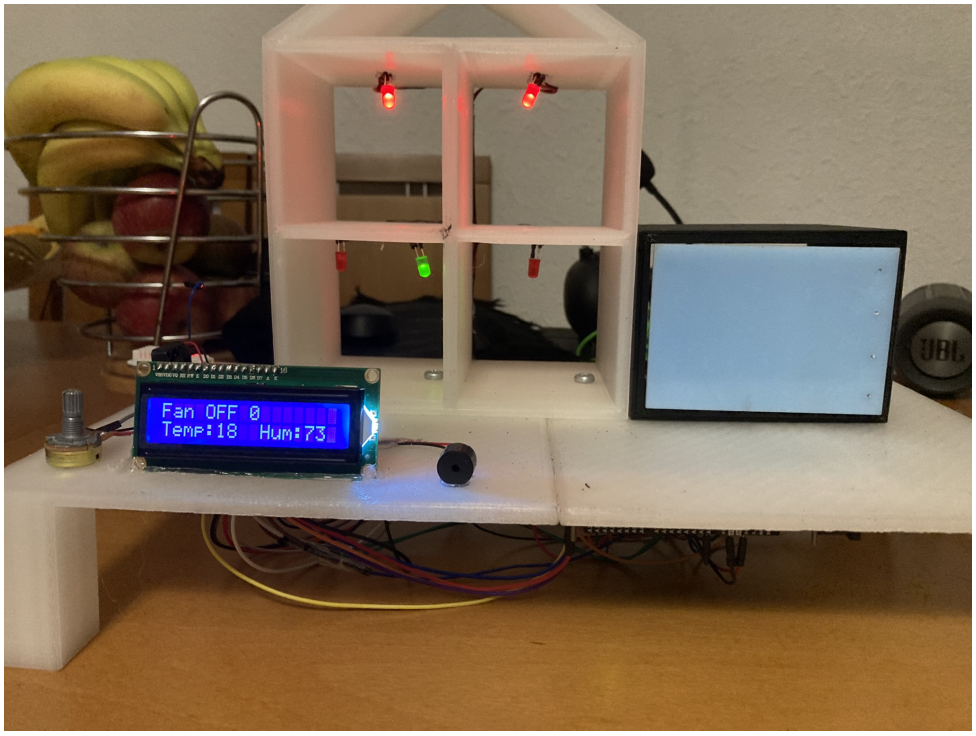

Figura 23: Casa

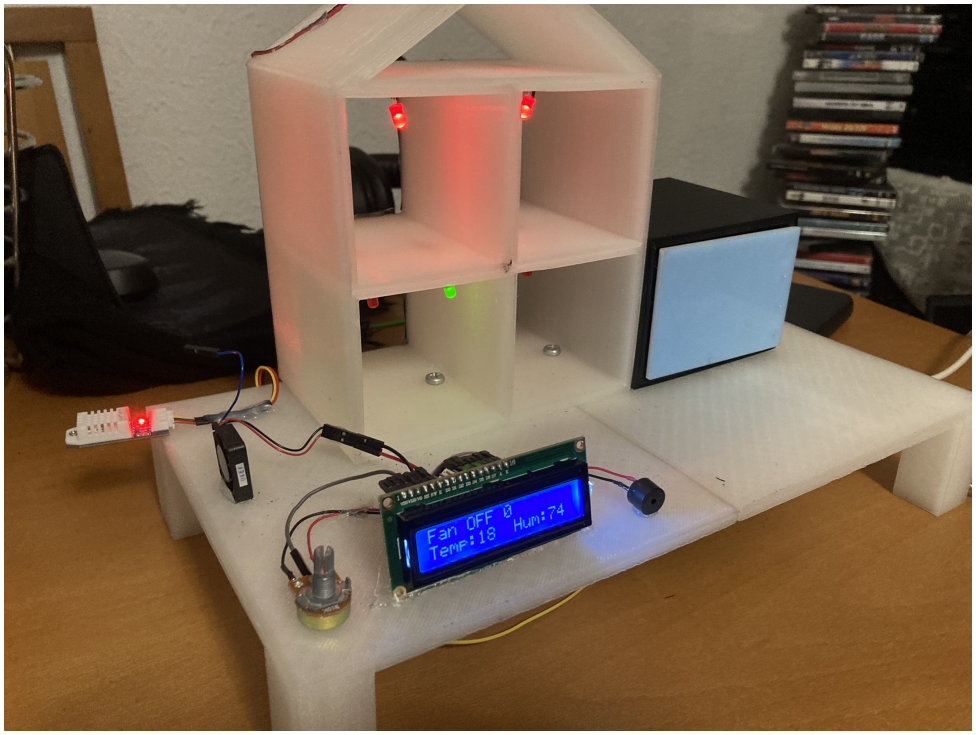

Figura 24: Casa

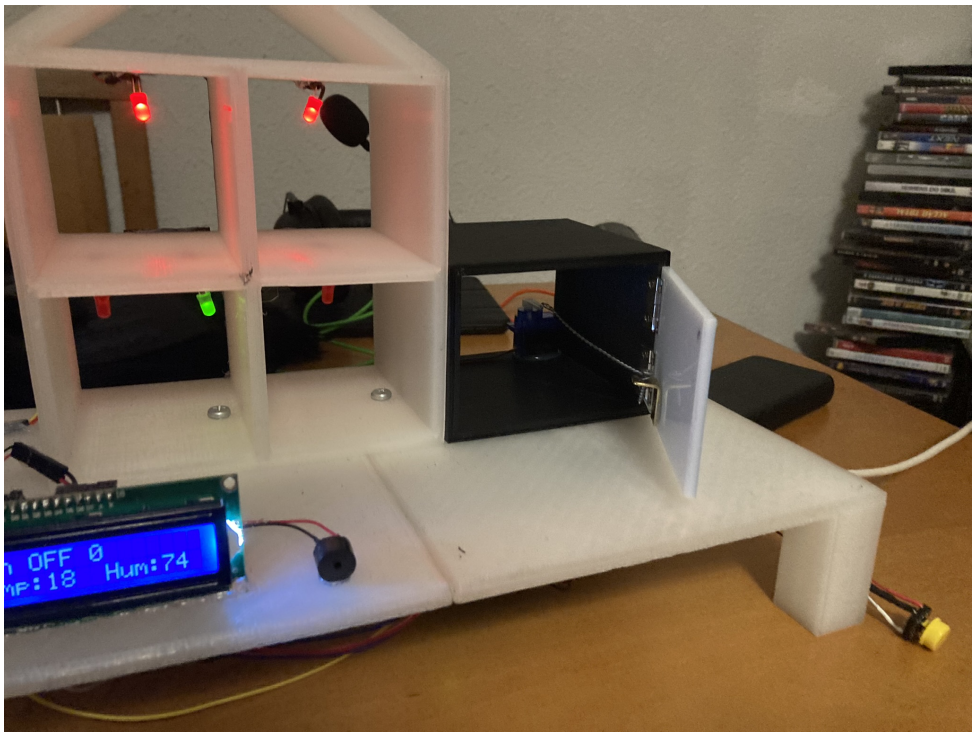

Figura 25: Casa

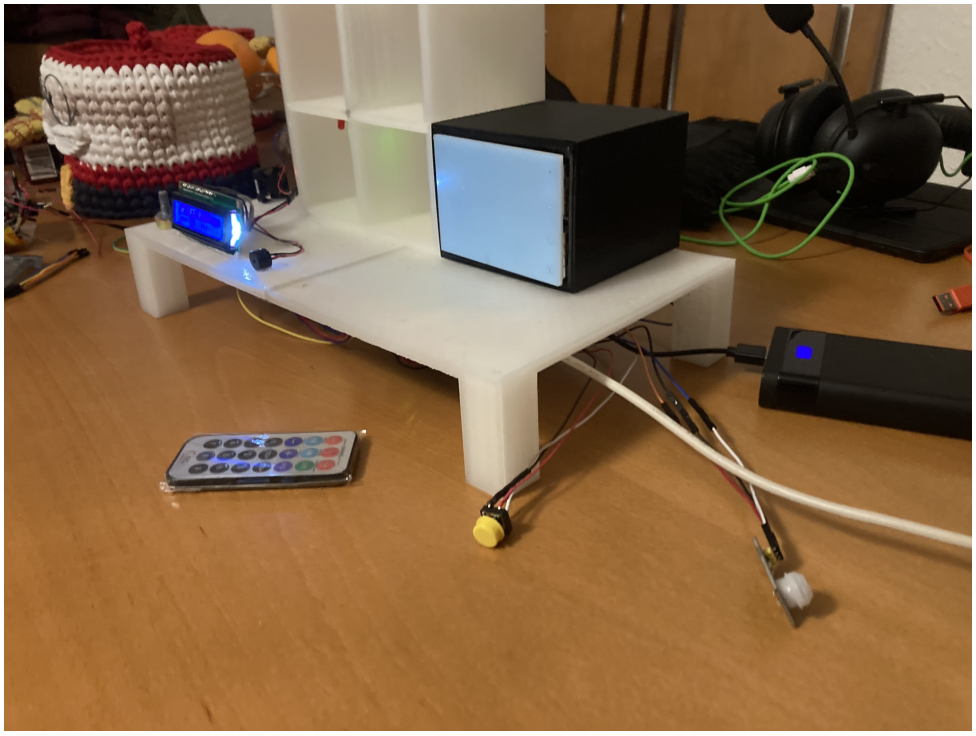

Figura 26: Casa

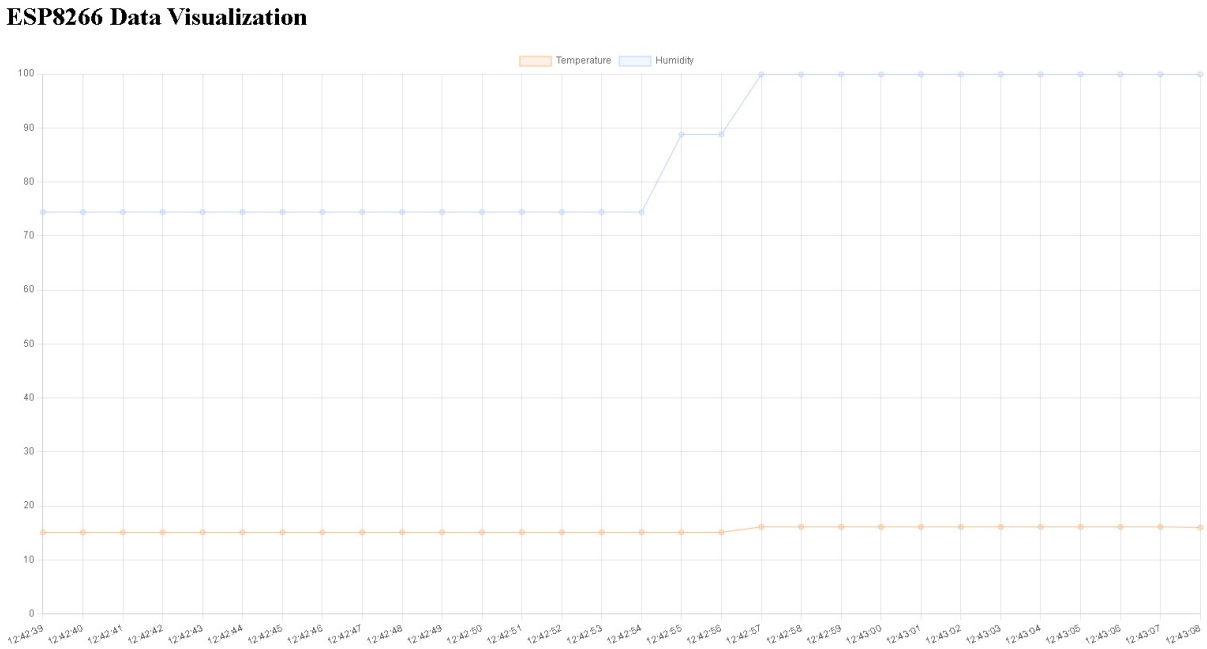

Figura 27: Website demonstração da temperatura

7.1 Código não Otimizado Sistema C

```
void TaskSystemCWorst(void *pvParameters) {
  // Avoid compiler warning about unused parameter
  (void)pvParameters;

  long time;
  bool isToAwait = false;
  byte ledState = HIGH;

  // Infinite loop for continuous task execution
  while (1) {
    // Update LCD with isActive state
    lcd.setCursor(8, 0);
    lcd.print(isActive);

    if (isActive) {
      // PIR sensor detected motion
      if (digitalRead(PIR) == HIGH) {
        if (!found) {
          time = millis();
          found = true;
          isToAwait = false;
        }
      }
      // Execute actions when motion is detected
      if (found) {
        if (!isToAwait) {
          playTone(1000);
          digitalWrite(Room2, ledState);
          ledState = !ledState;

          // Check if the elapsed time is greater than 10
   seconds
          if (millis() - time >= 10000) {
            digitalWrite(Buzzer, LOW);
            ledState = LOW;
            digitalWrite(Room2, LOW);
            isToAwait = true;
            time = millis();
          }
        }
      }

      // Wait for 5 seconds after the action and reset
      if (isToAwait) {
        if (millis() - time >= 5000) {
```

```
            isToAwait = false;
46          time = millis();
          }
48      }
    }
50    else {
      digitalWrite(Buzzer, LOW);
52    digitalWrite(Room2, LOW);
    }
54
    // Delay to control the update rate of the task
56  vTaskDelay(pdMS_TO_TICKS(100));
  }
58}
```

Listing 9: SystemC.ino

Com o uso do código para o Sistema C implementou-se de outra forma para testar a eficiência de um código bem feito para um código mal feito, e como se pode ver este código para além de ser mais difícil, também não está otimizado.

8 Conclusão

Ao longo do desenvolvimento deste projeto, cada grupo empenhou-se na conceção e implementação de sistemas embebidos destinados ao controlo de iluminação, climatização, segurança e tarefas adicionais para uma residência inteligente. A proposta inicial incluía a integração harmoniosa desses sistemas, culminando numa solução completa e funcional.

A etapa final do projeto consistiu na integração bem-sucedida de todos os sistemas desenvolvidos num único ambiente coeso. A fusão dessas funcionalidades proporcionou uma residência inteligente totalmente operacional.

Contudo, ao longo do processo de desenvolvimento, deparamos-nos com desafios e limitações. Restrições de recursos, tanto em termos de hardware como de software, foram enfrentadas. A complexidade da integração de múltiplos sistemas também se revelou um ponto crítico, exigindo uma abordagem cuidadosa para garantir a eficácia e a estabilidade do sistema unificado.

Apesar dos desafios, o resultado final demonstra o sucesso do projeto na implementação de sistemas embebidos para Smart Housing. Cada grupo contribuiu de forma significativa para a criação de uma casa inteligente funcional e eficiente em termos energéticos.

Este projeto proporcionou uma valiosa experiência prática, destacando a importância da integração de sistemas para alcançar soluções inovadoras e funcionais. As dificuldades enfrentadas ao longo do caminho servem como oportunidades de aprendizagem, contribuindo para o crescimento técnico e aperfeiçoamento das habilidades de engenharia de sistemas.

9 Bibliografia

1. Github do Projeto: https://github.com/TinoMeister/SETR;
2. Plataforma TinkerCad: https://www.tinkercad.com/;
3. Download Arduino IDE: https://www.arduino.cc/en/software;
4. Bibliotecas do Arduino: https://www.arduino.cc/libraries/