Projeto Sistemas Embarcados

Etapa 2 - Biblioteca para controlar o ângulo de um servomotor

Membros: Antônio Farias Araújo Terceiro, Joao Marcos Amorim de Almeida, Jorge Vinícius Santos Castro, Pedro Macêdo Luna, Rennyson Cavalcante Soares.

Ferramentas necessárias:

- Visual Studio Code: **Download**
- ESP-IDF: **Download**

Hardware Necessário:

- ESP-32-WROOM
- 2 Servos:
  - **Wokwi Component**

Firmware

Esquemático do hardware

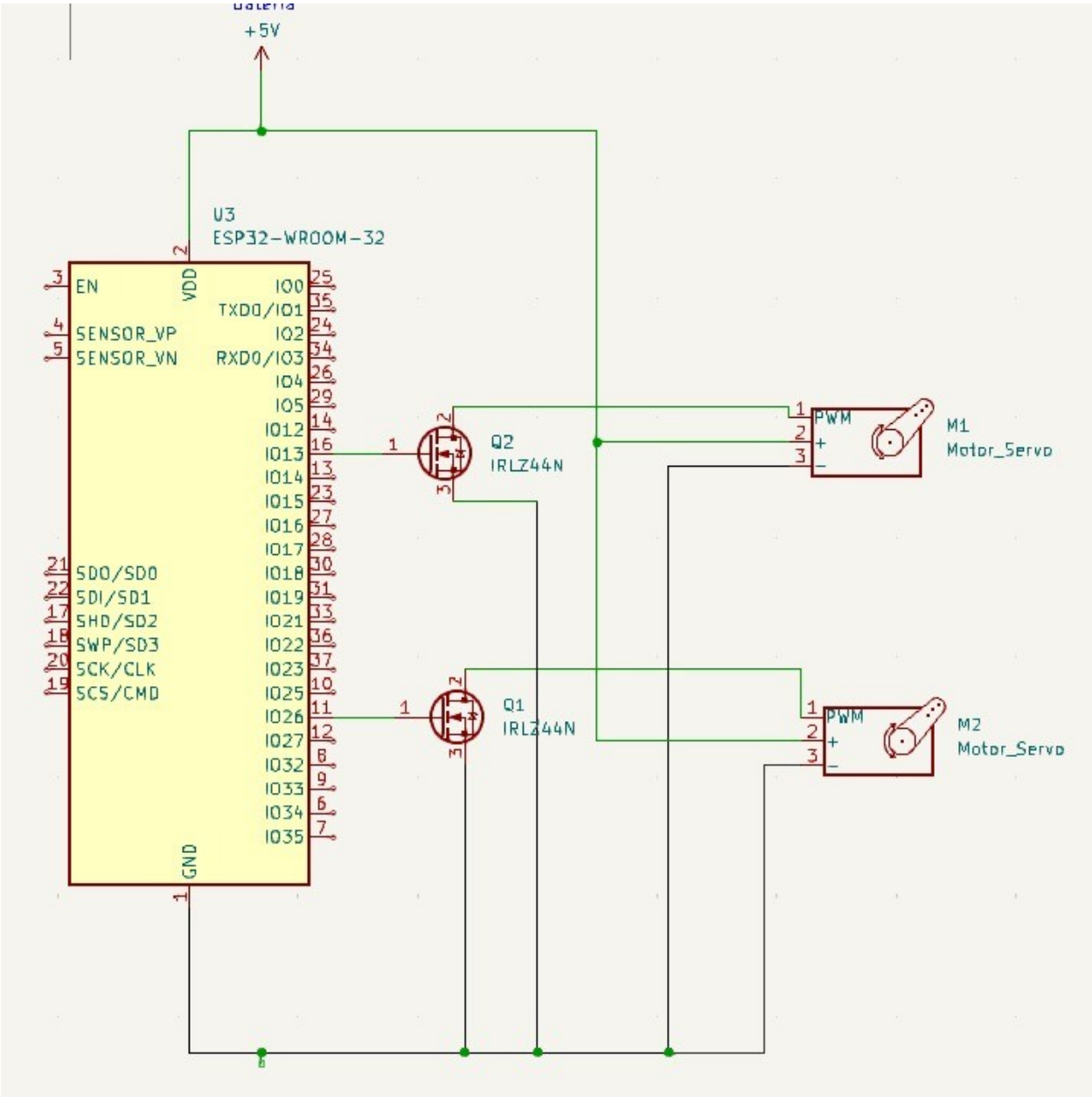
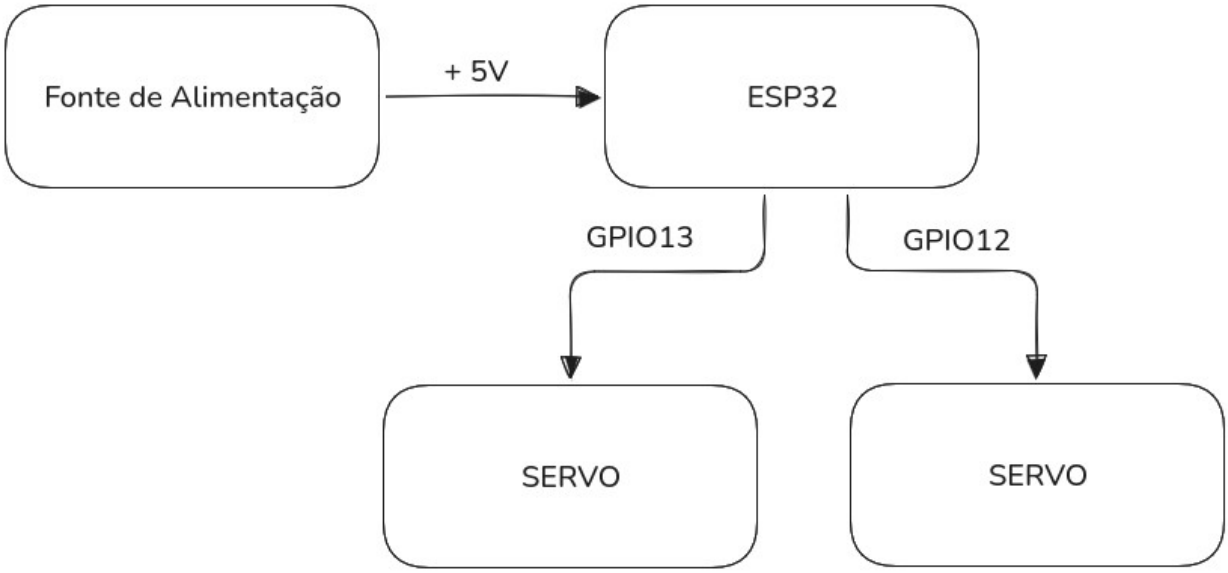
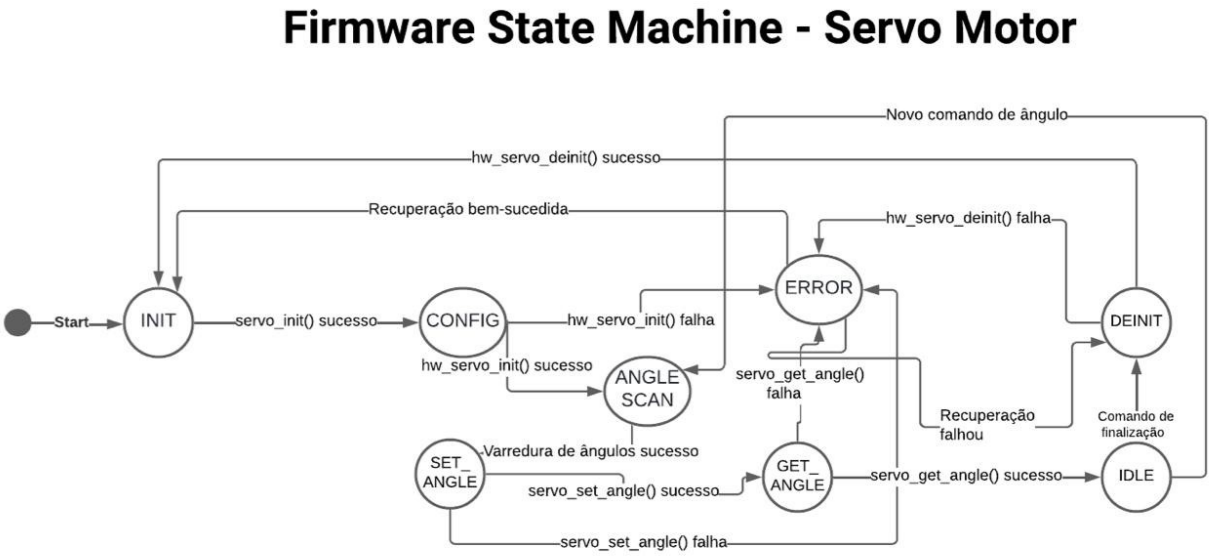


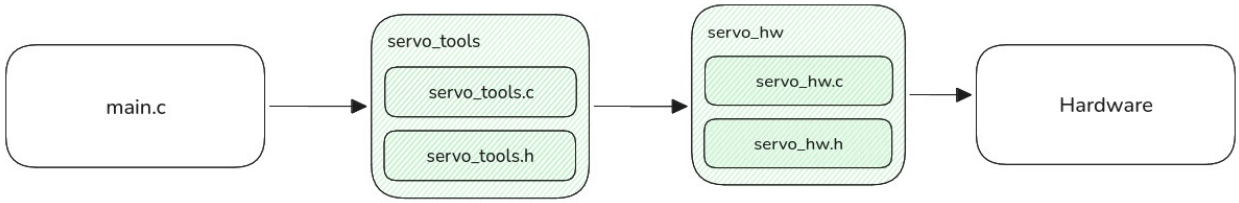
Diagrama de Blocos



Máquina de Estados



Estrutura do Projeto



```
etapa-2-servo/
│
├── build/
│   # Pasta onde os arquivos compilados são armazenados
│
├── components/
│   ├── servo_tools/
│   │   ├── include/
│   │   │   └── servo_tools.h  # Arquivo de cabeçalho para as funções
de controle do servo
│   │   ├── servo_tools.c      # Arquivo fonte para as funções de
controle do servo
│   │   └── CMakeLists.txt     # Arquivo CMake específico para o
componente servo_tools
│   │
│   └── servo_hw/
│       ├── include/
│       │   └── servo_hw.h     # Arquivo de cabeçalho para a abstração
de hardware do servo
│       ├── servo_hw.c         # Arquivo fonte para a abstração de
hardware do servo
│       └── CMakeLists.txt     # Arquivo CMake específico para o
componente servo_hw
│
├── docs/
│   ├── documentation.pdf    # pdf com documentação para firmware e
bibliotecas
│   ├── state_machine.png      # Diagrama de máquina de estados para
firmware
│   └── circuit_diagram.jpg    # Diagrama de circuito para firmware
│
├── main/
│   ├── CMakeLists.txt         # Arquivo CMake para a aplicação
principal
│   └── main.c                 # Arquivo fonte da aplicação principal
│
├── .gitignore                  # Arquivo que especifica quais
arquivos ou diretórios devem ser ignorados pelo Git
│
├── CMakeLists.txt             # Arquivo CMake principal para todo o
projeto
│
└── sdkconfig                  # Arquivo de configuração
(gerado/gerenciado pelo "make menuconfig")
```

Executar projeto

1. Clone o repositório para sua máquina local:

```
git clone https://github.com/pedromacedol/projetos-sistemas-embarcados.git
```

2. Navegue até o diretório da etapa 2 projeto:

```
cd projetos-sistemas-embarcados/etapa-2-servo
```

3. Abra o ESP-IDF
4. Selecione: **Configure ESP-IDF Extension**
5. Selecione: **Use Existing Setup**
6. Selecione: **Advanced / Add .vscode subdirectory files**
7. Selecione: **Build**
8. Feche o ESP-IDF
9. Verifique o status do build

Bibliotecas

servo_tools

Tipos de Dados

ServoConfig

A estrutura `ServoConfig` é usada para configurar os parâmetros de controle de um servo motor.

```
typedef struct
{
    uint8_t gpio_num;
    uint32_t pwm_freq;
    ServoAngle min_angle;
    ServoAngle max_angle;
    uint32_t min_pulse_width;
    uint32_t max_pulse_width;
} ServoConfig;
```

Atributos:

- `gpio_num`: Variável do tipo `uint8_t`, que representa o pino GPIO conectado ao servo.
- `pwm_freq`: Variável do tipo `uint32_t`, que representa a frequência PWM utilizada para controlar o servo.

- `min_angle`: Variável do tipo `ServoAngle`, que representa o ângulo mínimo permitido para o movimento do servo.
- `max_angle`: Variável do tipo `ServoAngle`, que representa o ângulo máximo permitido para o movimento do servo.
- `min_pulse_width`: Variável do tipo `uint32_t`, que representa a largura mínima do pulso em microssegundos.
- `max_pulse_width`: Variável do tipo `uint32_t`, que representa a largura máxima do pulso em microssegundos.

ServoAngle

ServoAngle é um tipo uint16_t que representa o ângulo de um servo motor, geralmente em graus ou milésimos de grau.

```
typedef uint16_t ServoAngle;
```

Funções

`servo_init`:

Inicializa o servomotor com base na configuração fornecida (pino GPIO, frequência PWM, etc.).

```
esp_err_t servo_init(ServoConfig *config);
```

Parâmetros:

- **config**: Ponteiro para uma estrutura `ServoConfig` que contém as configurações do servomotor.
  - **Tipo**: `ServoConfig *`
  - **Descrição**: Deve apontar para uma estrutura `ServoConfig` válida que especifica os parâmetros de configuração, como pino GPIO e frequência PWM.

Valor Retornado:

- **Tipo**: `esp_err_t`
- **Valores**:
  - `ESP_OK`: Sucesso na inicialização.
  - `ESP_ERR_INVALID_ARG`: Falha na inicialização devido a argumentos inválidos.

Exemplo de Uso

```
#include <stdio.h>
#include "esp_err.h"
#include "servo_tools.h"
```

```c
int main(void) {
    ServoConfig config = {
        .gpio_pin = 18,
        .pwm_freq = 50,
        .min_pulse_width = 1000,
        .max_pulse_width = 2000
    };

    esp_err_t result_servo = servo_init(&config);

    if (result_servo == ESP_OK) {
        printf("Servomotor inicializado com sucesso.\n");
    } else {
        printf("Falha na inicialização do servomotor. Código do erro:
%d\n", result_servo);
    }

    return 0;
}
```

`servo_set_angle`:

Configura o ângulo do servomotor com base na configuração fornecida.

```c
esp_err_t servo_set_angle(ServoConfig *config, ServoAngle angle);
```

Parâmetros:

- **config**:
  - **Tipo**: ServoConfig *
  - **Descrição**: Ponteiro para a estrutura ServoConfig que contém a configuração do servomotor.
- **quaternion**:
  - **Tipo**: ServoAngle *
  - **Descrição**: Ângulo desejado para o servomotor.

Valor Retornado:

- ESP_OK: Sucesso na configuração do ângulo.
- ESP_FAIL: Falha na configuração do ângulo.

```c
#include <stdio.h>
#include "esp_err.h"
#include "servo_tools.h"
```

```c
int main(void) {
    ServoConfig config = {
        .min_pulse_width = 500,
        .max_pulse_width = 2500,
        .max_angle = 180
    };

    ServoAngle angle = 90;

    esp_err_t result = servo_set_angle(&config, angle);

    if (result == ESP_OK) {
        printf("Ângulo do servomotor ajustado para %d graus.\n",
angle);
    } else {
        printf("Erro ao ajustar o ângulo do servomotor. Código do
erro: %d\n", result);
    }

    return 0;
}
```

`servo_get_angles`:

Obtém o ângulo atual do servomotor com base na configuração fornecida.

```c
esp_err_t servo_get_angle(const ServoConfig *config, ServoAngle
*angle);
```

Parâmetros

- **config**:
  - **Tipo**: `const ServoConfig *`
  - **Descrição**: Ponteiro constante para a estrutura `ServoConfig` que contém a configuração do servomotor.
- **angle**:
  - **Tipo**: `ServoAngle *`
  - **Descrição**: Ponteiro para uma variável do tipo `ServoAngle` onde o ângulo atual do servomotor será armazenado.

Valor Retornado

- **ESP_OK**: Sucesso na obtenção do ângulo.
- **ESP_FAIL**: Falha na obtenção do ângulo.

Exemplo de Uso

```c
#include <stdio.h>
#include "esp_err.h"
```

```c
#include "servo_tools.h"

int main(void) {
    ServoConfig config = {
        .min_pulse_width = 500,
        .max_pulse_width = 2500,
        .max_angle = 180
    };

    ServoAngle current_angle;

    esp_err_t result = servo_get_angle(&config, &current_angle);

    if (result == ESP_OK) {
        printf("Ângulo atual do servomotor: %d graus.\n",
current_angle);
    } else {
        printf("Erro ao obter o ângulo do servomotor. Código do erro:
%d\n", result);
    }

    return 0;
}
```

sensor_imu

Funções

`hw_servo_init`

Inicializa o servomotor e aloca os recursos necessários.

```c
esp_err_t hw_servo_init(uint8_t gpio_num);
```

Valor Retornado:

- **ESP_OK**: Sucesso na inicialização.
- **ESP_FAIL**: Erro durante a inicialização.

Parâmetros:

- **gpio_num**:
  - **Tipo**: `uint8_t`
  - **Descrição**: Número do GPIO onde o PWM será habilitado.

Exemplo de Uso

```c
#include <stdio.h>
#include "esp_err.h"
#include "servo_hw.h"
```

```
int main(void) {
    uint8_t gpio_num = 18;
    esp_err_t result = hw_servo_init(gpio_num);

    if (result == ESP_OK) {
        printf("Servomotor inicializado com sucesso no GPIO %d.\n",
gpio_num);
    } else {
        printf("Erro ao inicializar o servomotor no GPIO %d. Código do
erro: %d\n", gpio_num, result);
    }

    return 0;
}
```

`hw_servo_set_pulse_width`:

Define a largura de pulso para o controle do ângulo do servo.

```
esp_err_t hw_servo_set_pulse_width(uint8_t gpio_num, uint32_t
pulse_width_us);
```

Parâmetros:

- **gpio_num**:
  - **Tipo**: `uint8_t`
  - **Descrição**: Número do GPIO onde o PWM está habilitado.
- **pulse_width_us**:
  - **Tipo**: `uint32_t`
  - **Descrição**: Largura do pulso em microssegundos (us) a ser definida para o controle do ângulo do servo.

Valor Retornado:

- **ESP_OK**: Sucesso ao definir a largura de pulso.
- **ESP_FAIL**: Erro ao definir a largura de pulso.

Exemplo de Uso

```
#include <stdio.h>
#include "esp_err.h"
#include "servo_hw.h"

int main(void) {
    uint8_t gpio_num = 18;
    uint32_t pulse_width_us = 1500;

    esp_err_t init_result = hw_servo_init(gpio_num);
```

```
    if (init_result != ESP_OK) {
        printf("Erro ao inicializar o servomotor no GPIO %d. Código do
erro: %d\n", gpio_num, init_result);
        return init_result;
    }

    esp_err_t set_pulse_result = hw_servo_set_pulse_width(gpio_num,
pulse_width_us);
    if (set_pulse_result == ESP_OK) {
        printf("Largura de pulso definida para %d microssegundos no
GPIO %d.\n", pulse_width_us, gpio_num);
    } else {
        printf("Erro ao definir a largura de pulso no GPIO %d. Código
do erro: %d\n", gpio_num, set_pulse_result);
    }

    hw_servo_deinit(gpio_num);

    return 0;
}
```

`hw_servo_deinit`:

Desinicializa o servomotor e libera os recursos alocados.

```
esp_err_t hw_servo_deinit(uint8_t gpio_num);
```

Parâmetros:

- **gpio_num**:
  - **Tipo**: `uint8_t`
  - **Descrição**: Número do GPIO onde o PWM será desabilitado.

Valor Retornado:

- **ESP_OK**: Sucesso na desinicialização.
- **ESP_FAIL**: Erro durante a desinicialização.

Exemplo de Uso

```
#include <stdio.h>
#include "esp_err.h"
#include "servo_tools.h"

int main(void) {
    ServoConfig config = {
        .min_pulse_width = 500,
        .max_pulse_width = 2500,
        .max_angle = 180
    };
```

```
    esp_err_t result = hw_servo_deinit(&config);

    if (result == ESP_OK) {
        printf("Hardware do servomotor desinicializado com sucesso.\
n");
    } else {
        printf("Erro ao desinicializar o hardware do servomotor.
Código do erro: %d\n", result);
    }

    return 0;
}
```