Autor: Wilson Oliveira Lima (TIC370101025)

Relatório do Projeto: Código Morse SOS com LED

Implementação na Placa BitDogLab

EmbarcaTech/Projetos/blink at main · wilssola/EmbarcaTech

1. Visão Geral do Projeto

Este relatório documenta a implementação de um sinal SOS em código Morse utilizando um LED na placa de desenvolvimento BitDogLab. O projeto demonstra conceitos fundamentais de programação de sistemas embarcados utilizando a linguagem C, incorporando controle de GPIO e requisitos precisos de temporização.

2. Requisitos Técnicos

O projeto implementa as seguintes especificações de temporização para o sinal SOS em código Morse:

- Duração do ponto (.): 0,2 segundos LED ACESO
- Duração do traço (-): 0,8 segundos LED ACESO
- Intervalo entre elementos da mesma letra: 0,125 segundos
- Intervalo entre letras: 0,25 segundos
- Intervalo de repetição da mensagem: 3,0 segundos

3. Detalhes da Implementação

3.1 Configuração do Projeto

O projeto foi construído em cima de um projeto de exemplo chamado blink do Raspberry Pi Pico W, todo o desenvolvimento foi realizado a partir desse boilerplate.

3.2 Configuração do Wokwi

Autor: Wilson Oliveira Lima (TIC370101025)

Para rodar o Wokwi no VSCode, foi necessária a criação dos arquivos diagram.json e wokwi.toml. Depois de criado o diagram.json bastou abrir para montar o diagrama visualmente. Já o wokwi.toml foi configurado para apontar os arquivos blink.uf2 e blink.elf do projeto.

3.3 Configuração de Hardware

A implementação utiliza o pino GPIO 13 para controle do LED, embora o código suporte pinos alternativos (11 ou 12) através de uma simples modificação de constante. O sistema é projetado para ser compatível tanto com placas Pico padrão quanto com variantes Pico W, implementando compilação condicional apropriada.

3.4 Arquitetura de Software

A implementação do software segue uma abordagem modular, utilizando várias funções principais:

1. Função de Inicialização:

```
1. int pico_led_init(void)
```

Esta função gerencia a inicialização do GPIO e garante a configuração adequada para as variantes Pico padrão e Pico W.

2. Função de Controle do LED:

```
1. void pico_set_led(bool led_on)
```

Esta função fornece controle abstraído do LED, tratando as diferenças entre GPIO padrão e implementações Pico W.

3. Funções de Elementos do Código Morse:

```
1. void pico_morse_dot()
```

```
2. void pico_morse_dash()
```

Estas funções implementam os elementos fundamentais do código Morse com controle preciso de temporização.

4. Função Principal do Sinal:

```
1. void pico_sos()
```

Esta função orquestra o padrão completo do sinal SOS, gerenciando todos os requisitos de temporização e sequências de sinais.

4. Componentes Principais do Código

4.1 Constantes de Temporização

```
1. #define MORSE_DOT_MS 200
2. #define MORSE_DASH_MS 800
3. #define MORSE_SPACE_MS 250
4. #define MORSE_REPEAT_MS 3000
```

Estas constantes garantem temporização consistente em todos os elementos do código Morse, atendendo precisamente aos requisitos do projeto.

4.2 Loop Principal do Programa

```
1. int main() {
2.     int rc = pico_led_init();
3.     hard_assert(rc == PICO_OK);
4.     while (true) {
5.         pico_sos();
6.     }
7. }
```

O programa principal implementa um loop infinito que transmite continuamente o sinal SOS com os atrasos apropriados.

5. Testes e Verificação

Autor: Wilson Oliveira Lima (TIC370101025)

A implementação foi testada em múltiplos ambientes:

1. Simulador Wokwi Web

2. Integração Wokwi com VS Code

3. Placa de Desenvolvimento BitDogLab física

Todos os ambientes de teste confirmaram a temporização adequada e geração do sinal, com o LED produzindo com sucesso o padrão SOS (... --- ...) de acordo com as especificações.

6. Modularidade e Manutenibilidade do Código

A implementação demonstra boas práticas de engenharia de software através de:

- Separação clara de funções e responsabilidades

- Constantes bem definidas para parâmetros de temporização

- Tratamento de compatibilidade multiplataforma

- Comentários claros e documentação

- Estilo de codificação e formatação consistentes

7. Conclusão

O projeto implementa com sucesso todas as especificações necessárias para o gerador de sinal SOS em código Morse. O código é bem estruturado, manutenível e adequadamente documentado. A implementação demonstra o uso adequado de conceitos e práticas de programação de sistemas embarcados, mantendo a compatibilidade entre diferentes ambientes de desenvolvimento e teste.

O design modular permite modificações fáceis e potenciais melhorias, como suporte a padrões adicionais de código Morse ou ajuste de parâmetros de temporização. O projeto serve como uma base sólida para a compreensão de conceitos de programação de sistemas embarcados usando a placa de desenvolvimento BitDogLab.

8. Código-fonte

blink.c

```c
#include "pico/stdlib.h"

// Pico W devices use a GPIO on the WIFI chip for the LED,
// so when building for Pico W, CYW43_WL_GPIO_LED_PIN will be defined
#ifdef CYW43_WL_GPIO_LED_PIN
#include "pico/cyw43_arch.h"
#endif

// Define the delay for the LED in milliseconds if not already defined
#ifndef LED_DELAY_MS
#define LED_DELAY_MS 125
#endif

// Define the default LED pin for the Pico device
#define PICO_DEFAULT_LED_PIN 13

// Define the timing for Morse code elements in milliseconds
#define MORSE_DOT_MS 200
#define MORSE_DASH_MS 800
#define MORSE_SPACE_MS 250
#define MORSE_REPEAT_MS 3000

// Perform initialisation of the LED
int pico_led_init(void) {
#if defined(PICO_DEFAULT_LED_PIN)
    // A device like Pico that uses a GPIO for the LED will define PICO_DEFAULT_LED_PIN
    // so we can use normal GPIO functionality to turn the led on and off
    gpio_init(PICO_DEFAULT_LED_PIN);
    gpio_set_dir(PICO_DEFAULT_LED_PIN, GPIO_OUT);
    return PICO_OK;
#elif defined(CYW43_WL_GPIO_LED_PIN)
    // For Pico W devices we need to initialise the driver etc
    return cyw43_arch_init();
#endif
}

// Turn the LED on or off
void pico_set_led(bool led_on) {
#if defined(PICO_DEFAULT_LED_PIN)
    // Just set the GPIO on or off
    gpio_put(PICO_DEFAULT_LED_PIN, led_on);
#elif defined(CYW43_WL_GPIO_LED_PIN)
    // Ask the wifi "driver" to set the GPIO on or off
    cyw43_arch_gpio_put(CYW43_WL_GPIO_LED_PIN, led_on);
#endif
}

// Function to blink a Morse code dot
void pico_morse_dot() {
    pico_set_led(true);
    sleep_ms(MORSE_DOT_MS);
    pico_set_led(false);
    sleep_ms(MORSE_DOT_MS);
}

// Function to blink a Morse code dash
void pico_morse_dash() {
    pico_set_led(true);
    sleep_ms(MORSE_DASH_MS);
    pico_set_led(false);
    sleep_ms(MORSE_DOT_MS);
}

// Function to blink SOS in Morse code
```

```c
void pico_sos() {
    pico_morse_dot();
    sleep_ms(LED_DELAY_MS);
    pico_morse_dot();
    sleep_ms(LED_DELAY_MS);
    pico_morse_dot();

    sleep_ms(MORSE_SPACE_MS);
    pico_morse_dash();
    sleep_ms(LED_DELAY_MS);
    pico_morse_dash();
    sleep_ms(LED_DELAY_MS);
    pico_morse_dash();

    sleep_ms(MORSE_SPACE_MS);
    pico_morse_dot();
    sleep_ms(LED_DELAY_MS);
    pico_morse_dot();
    sleep_ms(LED_DELAY_MS);
    pico_morse_dot();
    sleep_ms(MORSE_SPACE_MS);

    sleep_ms(MORSE_REPEAT_MS);
}

// Main function to initialize the LED and blink SOS in an infinite loop
int main() {
    int rc = pico_led_init();
    hard_assert(rc == PICO_OK);
    while (true) {
        pico_sos();
    }
}

```

diagram.json

```json
{
  "version": 1,
  "author": "Anonymous maker",
  "editor": "wokwi",
  "parts": [
    { "type": "board-pi-pico-w", "id": "pico", "top": 0, "left": 0, "attrs": {} },
    {
      "type": "wokwi-led",
      "id": "led1",
      "top": 34.8,
      "left": -149.8,
      "attrs": { "color": "red" }
    },
    {
      "type": "wokwi-resistor",
      "id": "r1",
      "top": 139.2,
      "left": -163.75,
      "rotate": 90,
      "attrs": { "value": "1000" }
    }
  ],
  "connections": [
    [ "pico:GP0", "$serialMonitor:RX", "", [] ],
    [ "pico:GP1", "$serialMonitor:TX", "", [] ],
    [ "led1:C", "r1:1", "green", [ "v0" ] ],
    [ "r1:2", "pico:GND.4", "green", [ "h0" ] ],
    [ "led1:A", "pico:GP13", "green", [ "v0" ] ]
  ],
  "dependencies": {}
}
```