UNIVERSIDADE DE SÃO PAULO

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO - POLI USP

PCS3732 - LABORATÓRIO DE PROCESSADORES

Augusto Vaccarelli Costa	10770197
Eduardo Niza Minosso	11918543
Rodrigo Tei Dalmas	11261660

BIBLIOTECA DE ABSTRAÇÃO DE HARDWARE

Prof. Dr. Bruno Abrantes Basseto

SÃO PAULO - SP

2023

Augusto Vaccarelli Costa 10770197
Eduardo Niza Minosso 11918543
Rodrigo Tei Dalmas 11261660

BIBLIOTECA DE ABSTRAÇÃO DE HARDWARE

SÃO PAULO - SP

2023

SUMÁRIO

INTRODUÇÃO

A placa Evaluator 7T, oferece uma série de funcionalidades. No entanto, a complexidade inerente ao hardware pode dificultar a adoção e a utilização eficaz desses recursos, especialmente para pessoas de menor experiência no manuseio da tecnologia.

Neste cenário, o seguinte trabalho aborda esta problemática ao criar uma biblioteca de abstração de hardware para a placa Evaluator7T. O principal objetivo dessa biblioteca é proporcionar uma interface simplificada e intuitiva para acessar e utilizar as diversas funcionalidades da placa, sem a necessidade de um profundo conhecimento em detalhes de hardware. Assim, será possível acessar as funcionalidades da placa através de funções na linguagem C.

Ao longo deste trabalho, destacaremos os resultados obtidos por meio de testes e experimentos, demonstrando como a utilização da biblioteca pode impactar positivamente a eficiência e eficácia do processo de desenvolvimento e aprendizagem.

1 A placa Evaluator 7T

A placa Evaluator-7T é uma plataforma ARM simples que inclui um conjunto mínimo de recursos principais. Ela é poderosa e flexível o suficiente para funcionar como uma plataforma de avaliação para a tecnologia ARM. A placa permite que você faça o download e depure imagens de software e anexe dispositivos de entrada/saída e periféricos adicionais para experimentação. Os principais componentes da placa incluem um microcontrolador Samsung KS32C50100 RISC, 512KB de flash EPROM, 512KB de SRAM, dois conectores RS232 de 9 pinos, botões de pressão de reset e interrupção, quatro LEDs programáveis pelo usuário e um display LED de sete segmentos, um DIP switch de entrada de usuário de 4 vias, um conector Multi-ICE, um relógio de 10MHz (o processador usa isso para gerar um relógio de 50MHz), um regulador de tensão de 3,3V e outros componentes adicionais necessários.

2 A Biblioteca

2.1 Funções

A biblioteca inclui um coonjunto mínimo de funções para controlar a placa. Entre as funções temos:

- usaGPIO(pino , estado)
  - **Pino** : Apelido do pino
  - **Estado**: Ligado (1, HIGH, True) ou Desligado (0, LOW, False)
- delay(tempo)
  - **Tempo**: Tempo em milissegundos (limite de tempo 32bits/50MHz)
- serial0(baud, protocolo)
  - **Baud**: Baud Rate (*default* 9600)
  - **Protocolo**: Configura protocolo (5N1 até 8E2) (*default* 8N1)
- serial1(baud, protocolo)
  - **Baud**: Baud Rate (default 9600)
  - **Protocolo**: Configura protocolo (5N1 até 8E2) (*default* 8N1)

2.2 Exemplos de uso

```
1    usaGPIO(1, HIGH)
```

```
1    usaGPIO(2, True)
```

```
1    usaGPIO(2, 0)
```

Listing 2.1: Exemplos de uso da função **usaGPIO()**

```
1    delay(1000)
```

```
1    delay(500)
```

Listing 2.2: Exemplos de uso da função **delay**

```
1    serial0(1200, 8E2)
```

```
1    serial0(4800, 5N1)
```

Listing 2.3: Exemplos de uso da função **serial0**

```
1    serial1(4800, 8E2)
```

```
1    serial1(19200, 8N1)
```

Listing 2.4: Exemplos de uso da função **serial1**

2.3 Uso dos registradores no microcontrolador

System manager group	Input/output ports	Interrupt controller	UART
SYSCFG	IOPMOD	INTMOD	ULCON1
EXTDBWTH	IOPCON	INTPND	UCON1
ROMCON0	IOPDATA	INTMSK	USTAT1
ROMCON1	-	-	UTXBUF1
ROMCON2	-	-	URXBUF1
-	-	-	URXBUF1

Tabela 1: Lista os registradores usados pelo software do sistema

Relacionando a tabela com o script elaborado em **botao.c** tem-se:

- **IOPMOD**: É um registrador de seleção de modo de pinos no chip. A função ”bit_set (IOPMOD, 5)”é usada para estabelecer o pino 5 como saída configurado para o LED.
- **IOPDATA**: É um registrador de dados de porta de pinos. ”bit_clr (IOPDATA, 5)”é usado para apagar (definir como 0) o LED ligado ao pino 5.
- **IOPCON**: Contém as configurações gerais de controle dos demais registradores. ”IOPCON = (IOPCON, (0b11111)) | 0b11001”está sendo usado para calibrar o registo do interruptor de limite externo.
- **INTMSK**: É o registrador de máscara de interrupção. A linha ”bit_clr(INTMSK, 0)”habilita a interrupção externa 0.

3 Implementações

3.1 Linker

Com base nos aprendizados da disciplina foi estruturado o seguinte arquivo linker para definir as seções do programa e organização da memória:

```
SECTIONS {                    /* arquivo kernel.ld para placa Evaluator7T */
    /*
     * Vetor de reset
     */
    . = 0;
    .reset : { *(.reset) }

    /*
     * Segmentos text e data
     */
    . = 0x8000;
  .text : { *(.text) }
  .data : { *(.data) }

  /*
   * Segmento bss
   */
  inicio_bss = .;
  .bss : { *(.bss) }
  . = ALIGN(4);
  fim_bss = .;

  /*
   * Reserva espaco para a pilha
   */
  . = . + 4096;
  . = ALIGN(8);
  inicio_stack = .;
```

```
29 }
```

Listing 3.1: Arquivo kernel.ld

Vemos nas linhas 5 e 6 que a seção **.reset** está sendo declarada no endereço **0x0**. Esta seção define alguns dados de inicialização.

Nas linhas 11, 12 e 13 vemos a declaração das seções **.text** e **.data** no endereço 0x8000. Os segmentos citados são responsáveis por conter o próprio código em texto, e dados utilizados na execução deste código, respectivamente.

Nas linhas 18 e 19 vemos a declaração do segmento **.bss**, que segue o segmento **.data**. O segmento **.bss** é responsável por armazenar dados não inicializados, de modo que haja economia na memória alocada no segmento **.data**.

Na linha 20 vemos a necessidade do uso da diretiva **ALIGN** que cumpre o propósito de garantir que o endereço de memória atual esteja alinhado num padrão de **8 bytes**.

Enfim, na linha 26 vemos um espaço de 4096 bytes sendo reservados para a pilha, sendo espaço suficiente para a execução dos códigos que seguem.

3.2 botão.c

```
1  SECTIONS {                    /* arquivo kernel.ld para placa
2  #include "evlt7t.h"
3  #include <stdint.h>
4  #include <stdlib.h>
5
6  /**
7   * Tratamento da interrupcao externa 0 (sinal P8, conectado ao
      botao).
8   * Pisca o segundo led.
9   */
10 void
11 trata_irq_ext0(void) {
12    bit_inv(IOPDATA, 5);
13 }
14
15 /**
16  * Funcao de inicializacao do tratamento da interrupcao do
      botao.
```

```
17  */
18 void
19 inicia_botao(void) {
20     /*
21      * Configura a saida do led.
22      */
23     bit_set(IOPMOD, 5);      // configura o pino 5 como saida (
       segundo led).
24     bit_clr(IOPDATA, 5);     // inicia o led apagado.
25
26     /*
27      * Configura P8 como entrada com interrupcao.
28      */
29     bit_clr(IOPMOD, 8);
30     IOPCON = (IOPCON & ~(0b11111)) | 0b11001;    // borda de
       subida
31     bit_clr(INTMSK, 0);       // habilita interrupcao externa 0
32 }
```

Listing 3.2: botao.c

Botao.c destaca-se duas funções principais:

”trata_irq_ext0”: acionada quando há uma interrupção externa 0 (input P8), causa a inversão de estado do bit 5 no registrador IOPDATA, o que resulta em um LED ligado a esse bit piscando.

”inicia_botao”: realiza configurações iniciais, definindo o bit 5 de IOP como saída (presumindo que esteja conectado a um LED) e configuração de pino P8 como entrada, programada para acionar interrupções na borda ascendente.

Ou seja, o código permite que um LED seja alternado entre estados ligado/desligado cada vez que um botão conectado ao pino P8 é pressionado, ilustrando uma estratégia de resposta a interrupções para interações do usuário.

3.3 pipoca.c

```
1 #include <stdint.h>
2 #include <stdlib.h>
3 #include "evlt7t.h"
```

```
4  #include "pipoca.h"
5
6  /**
7   * Tratamento da interrupcao externa 0 (sinal P8, conectado ao
      botao).
8   * Pisca o segundo led.
9   */
10 void trata_irq_ext0(void) {
11    bit_inv(IOPDATA, 5);
12 }
13
14 /**
15  * Funcao de inicializacao do tratamento da interrupcao do
      botao.
16  */
17 void
18 inicia_botao(void) {
19    /*
20     * Configura a saida do led.
21     */
22    bit_set(IOPMOD, 5);      // configura o pino 5 como saida (
      segundo led).
23    bit_clr(IOPDATA, 5);     // inicia o led apagado.
24
25    /*
26     * Configura P8 como entrada com interrupcao.
27     */
28    bit_clr(IOPMOD, 8);
29    IOPCON = (IOPCON & ~(0b11111)) | 0b11001;   // borda de
      subida
30    bit_clr(INTMSK, 0);       // habilita interrupcao externa 0
31 }
32
33 /**
34  * Base de tempo do sistema, incrementado a cada 1 milissegundo
      .
35  */
36 volatile static uint32_t ticks;
37
```

```
38 /**
39  * Tratamento da interrupcao do timer 0.
40  * Chamado por trata_irq().
41  */
42 void trata_irq_timer0(void) {
43    ticks++;
44 }
45
46 /**
47  * Le o valor atual dos ticks com seguranca.
48  * @return Valor atual do contador de ticks.
49  */
50 uint32_t get_ticks(void) {
51    uint32_t res;
52    bit_set(INTMSK, 10);
53    res = ticks;
54    bit_clr(INTMSK, 10);
55    return res;
56 }
57
58 /**
59  * Espera ocioso com base no contador ticks.
60  * @param v Tempo a esperar em unidades de 1ms.
61  */
62 void delay_tempo(uint32_t v) {
63    uint32_t inicio = get_ticks();
64    while((get_ticks() - inicio) < v) ;
65 }
66
67 /**
68  * Configura o timer0 para gerar interrupcao a cada 1ms.
69  */
70 void inicia_timer0(void) {
71    TDATA0 = 49999;         // valor para 0.001s com clock de 50
      MHz
72    TCNT0 = TDATA0;
73    TMOD = (TMOD & (~0b111)) | 0b011;  // ativa o timer 0
74    bit_clr(INTMSK, 10);      // habilita interrupcao do timer 0
75    bit_clr(INTMSK, 21);      // habilita interrupcoes globais
```

```
76 }
77
78
79 /*
80  * buffer de transmissao
81  */
82 static volatile uint32_t ntx = 0;         ///< Quantidade de
     bytes a transmitir
83 static volatile uint8_t *ptx = NULL;      ///< Ponteiro para o
     proximo byte a transmitir
84
85 /**
86  * Tratamento da interrupcao de transmissao da UART 0.
87  * Funcao chamada por trata_irq().
88  */
89 void
90 trata_irq_uart0_tx(void) {
91    if(ntx == 0) return;
92    ntx--;
93    if(ntx == 0) return;
94    ptx++;
95    UTXBUF0 = *ptx;
96 }
97
98 /**
99  * libc nao esta presente.
100  */
101 int
102 strlen(char *s) {
103    int n = 0;
104    while(*s) {
105       n++;
106       s++;
107    }
108    return n;
109 }
110
111 /**
112  * Envia um string atraves da serial, usando interrupcoes.
```

```
113  * @param str String a enviar.
114  */
115 void
116 serie(char *str) {
117    while(ntx) ;              // verifica transmissao pendente
118    ntx = strlen(str);
119    ptx = str;
120    UTXBUF0 = str[0];         // envia o primeiro caractere.
121 }
122
123 /**
124  * Configura a UART 0 (serial do usuario).
125  * (somente transmissao)
126  */
127 void
128 inicia_serial(int baud) {
129    ULCON0 = 0b111;            // 8N2, sem paridade, clock
    interno
130    UCON0 = 0b1001;            // TX e RX habilitado
131    if (baud == 115200){
132     UBRDIV0 = (13 << 4); // 13, valor para 115200 em 50 MHz
133    } else if (baud == 57600){
134     UBRDIV0 = (26 << 4); // 26, valor para 57600 em 50 MHz
135    } else if (baud == 38400){
136     UBRDIV0 = (40 << 4); // 40, valor para 38400 em 50 MHz
137    } else if (baud == 19200){
138     UBRDIV0 = (80 << 4); // 80, valor para 19200 em 50 MHz
139    } else if (baud == 4800){
140     UBRDIV0 = (324 << 4); // 324, valor para 4800 em 50 MHz
141    } else if (baud == 2400){
142     UBRDIV0 = (650 << 4); // 650, valor para 2400 em 50 MHz
143    } else if (baud == 1200){
144     UBRDIV0 = (1301 << 4); // 1301, valor para 1200 em 50 MHz
145    } else {   //valor de excecao para preenchimentos errados
146   UBRDIV0 = (162 << 4); // 162, valor para 9600 em 50 MHz
147    }
148
149    bit_clr(INTMSK, 4);    // habilita interrupcao da
    transmissao.
```

```
150 }
```

Listing 3.3: pipoca.c

A estrutura do código principal Pipoca.c reflete no uso de manipulações de bit para limpar e configurar bits preciosos no microcontrolador e na configuração geral de componentes de hardware, incluindo um botão, um LED, um timer e uma porta serial (UART).

- trata_irq_ext0: Esta função é chamada quando existe uma interrupção externa associada ao sinal P8, possivelmente originada de um botão. Quando essa interrupção ocorre, o segundo LED no microcontrolador é feito para piscar.

- inicia_botao: A funcionalidade do botão é inicializada. Ela configura o LED correspondente para a saída e também define P8 como uma entrada com habilidades de interrupção. Essa configuração permite que o sistema responda quando o botão é pressionado.

- ticks, trata_irq_timer0, get_ticks e delay_tempo: O código cria e manipula uma entidade "ticks", servindo como contadores de tempo para oferecer funcionalidades de atraso. Ticks é incrementado toda vez que a função trata_irq_timer0 é chamada.

- inicia_timer0: Esta configure o timer0 para disparar interrupções a cada 1 milissegundo.

- ntx, ptx, trata_irq_uart0_tx, strlen, serie e inicia_serial: Em conjunto, essas funções e variáveis são usadas para implementar a comunicação via portas seriais (UART 0 en particular). O IR séries recebe uma string como parâmetro e a envia através da porta serial, um byte de cada vez via interrupções.

Em resumo, esta implementação dá ao microcontrolador a habilidade de registrar pressionamentos de botões através de interrupções, executar atrasos precisos com o auxílio de ticks e o timer0, e a comunicação serial/UART caracterizada pela transmissão de strings.

3.4 main.c

```c
#include "evlt7t.h"
#include <stdint.h>
#include <stdlib.h>

#include "timer.h"
#include "serial.h"
#include "botao.h"

void delay(uint32_t t);

/**
 * Tratamento das interrupcoes do ARM.
 */
void __attribute__((interrupt("IRQ")))
trata_irq(void) {
    /*
     * Verifica causa da interrupcao.
     */
    uint32_t pend = INTPND;
    if(bit_is_set(pend, 10)) trata_irq_timer0();
    if(bit_is_set(pend, 4)) trata_irq_uart0_tx();
    if(bit_is_set(pend, 0)) trata_irq_ext0();

    /*
     * Reconhece todas as interrupcoes.
     */
    INTPND = pend;
}

/*
 * Programa principal
 */
int
main(void) {
    /*
     * Configura led (I/O 4) como saida.
     */
    bit_set(IOPMOD, 4);

```

```
40      /*
41       * Aciona funcoes de inicializacao.
42       */
43      inicia_botao();
44      inicia_timer0();
45      inicia_serial();
46
47      serie("Hello people!\r\n");
48
49      /*
50       * Repetir indefinidamente.
51       */
52      for(;;) {
53          bit_inv(IOPDATA, 4);
54          delay_tempo(50);
55      }
56  }
```

Listing 3.4: main.c

A rotina trata_irq() configura o tratamento global de interrupções do sistema, sendo invocado quando há uma interrupção ocorrido (”IRQ”). A função usa o valor de INTPND (o registo que guarda as interrupções pendentes) para depois verificar qual procedimento de atendimento a interrupção deve ser chamado conforme sugere o vetor de interrupções.

Conforme descrito, o código proporciona tratamento de interrupção para a serial UART0 via trata_irq_uart0_tx(), para timer0 através do trata_irq_timer0(), e uma interrupção externa na esperança de invocar trata_irq_ext0(). Após o atendimento da interrupção, todas as interrupções pendentes são reconhecidas e o fluxo de execução é novamente direcionado para a rotina interrompida.

A main() configura um LED na posição 4 para saída e depois inicializa algumas funções relacionadas ao botão (inicia_botao()), ao timer0 (inicia_timer0()) e a porta serial (inicia_serial()). Captura duas ações práticas: gera um olá mundo através da porta serial, imprimindo a frase ”Hello people!”. Este processo esta encapsulado com o procedimento serie(”Hello people!”).

Pós esta fase de inicialização, infunde num laço de repetição indefinida que inverte continuamente o estado do LED 4 (com uso da rotina bit_inv() para troca de estado, alternando entre ON e OFF) e invoca um *delay* de 500ms a cada iteração (O intervalo

para a transição de estados deste LED).

Em síntese, o código fornece a estrutura básica para operar o evaluator7t para uma aplicação do tipo pipoca explode-explode(pisca-pisca), persistindo uma comunicação série ativa para transmitir mensagens de strings visando certa interação entre utilizador e o microcontrolador. Desta forma, incorpora partes da inicialização e configuração, declaração de interrupções, manipulação da ferramenta UART e implementação do laço de espera em um programa baseado em microcontroladores.

3.5 serial.c

```
1  #include "evlt7t.h"
2  #include <stdint.h>
3  #include <stdlib.h>
4
5
6  /*
7   * buffer de transmissao
8   */
9  static volatile uint32_t ntx = 0;          ///< Quantidade de
       bytes a transmitir
10 static volatile uint8_t *ptx = NULL;       ///< Ponteiro para o
       proximo byte a transmitir
11
12 /**
13  * Tratamento da interrupcao de transmissao da UART 0.
14  * Funcao chamada por trata_irq().
15  */
16 void
17 trata_irq_uart0_tx(void) {
18     if(ntx == 0) return;
19     ntx--;
20     if(ntx == 0) return;
21     ptx++;
22     UTXBUF0 = *ptx;
23 }
24
25 /**
```

```
26  * libc nao esta presente.
27  */
28 int
29 strlen(char *s) {
30    int n = 0;
31    while(*s) {
32       n++;
33       s++;
34    }
35    return n;
36 }
37
38 /**
39  * Envia um string atraves da serial, usando interrupcoes.
40  * @param str String a enviar.
41  */
42 void
43 serie(char *str) {
44    while(ntx) ;              // verifica transmissao pendente
45    ntx = strlen(str);
46    ptx = str;
47    UTXBUF0 = str[0];         // envia o primeiro caractere.
48 }
49
50 /**
51  * Configura a UART 0 (serial do usuario).
52  * (somente transmissao)
53  */
54 void
55 inicia_serial(void) {
56    ULCON0 = 0b111;           // 8 bits, sem paridade, clock
   interno
57    UCON0 = 0b1000;           // TX habilitado
58    UBRDIV0 = (26 << 4);      // valor para 57600 em 50 MHz
59    bit_clr(INTMSK, 4);       // habilita interrupcao da
   transmissao.
60 }
```

Listing 3.5: serial.c

serial.c oferece funcionalidades para a comunicação serial, especificamente para a UART 0, no microcontrolador evaluator7t.

Consiste em quatro funções principais.

trata_irq_uart0_tx: Responsável por tratar as interrupções de transmissão da UART 0. São enviados em sequência os bytes que ainda não foram transmitidos.

strlen: Presente em virtude da ausência da standard library (libc), essa implementação da função strlen retorna o tamanho de uma string dada.

serie: Esta função faz o envio de uma string através da porta serial do dispositivo UART diafragmático em irq.

inicia_serial: Função encarregada de configurar a UART 0 para transmitir dados, definindo o tamanho da palavra (8 bits), modo de operação (sem paridade e clock interno) e a taxa de transmissão (correspondendo a 57.600 bps em um clock de 50MHz).

O código incorpora ainda duas variáveis de buffer de transmissão, ntx para armazenar o número de bytes restantes a serem transmitidos e ptx para registrar o endereço do próximo byte a ser transmitido.

Em resumo, é um programa direto e de objetivo único - implementar funcionalidades não bloqueantes para transmissão serial no envio de mensagens de string pela porta UART na evaluator7t.

3.6 timer.c

```
1  #include "evlt7t.h"
2  #include <stdint.h>
3  #include <stdlib.h>
4
5  /**
6   * Base de tempo do sistema, incrementado a cada 0.1 segundos.
7   */
8  volatile static uint32_t ticks;
9
10 /**
11  * Tratamento da interrupcao do timer 0.
12  * Chamado por trata_irq().
13  */
14 void
```

```
15 trata_irq_timer0(void) {
16    ticks++;
17 }
18
19 /**
20  * Le o valor atual dos ticks com seguranca.
21  * @return Valor atual do contador de ticks.
22  */
23 uint32_t
24 get_ticks(void) {
25    uint32_t res;
26    bit_set(INTMSK, 10);
27    res = ticks;
28    bit_clr(INTMSK, 10);
29    return res;
30 }
31
32 /**
33  * Espera ocioso com base no contador ticks.
34  * @param v Tempo a esperar em unidades de 0.1s.
35  */
36 void
37 delay_tempo(uint32_t v) {
38    uint32_t inicio = get_ticks();
39    while((get_ticks() - inicio) < v) ;
40 }
41
42 /**
43  * Configura o timer0 para gerar interrupcao a cada 100ms.
44  */
45 void
46 inicia_timer0(void) {
47    TDATA0 = 4999999;         // valor para 0.1s com clock de 50
      MHz
48    TCNT0 = TDATA0;
49    TMOD = (TMOD & (~0b111)) | 0b011;  // ativa o timer 0
50    bit_clr(INTMSK, 10);      // habilita interrupcao do timer 0
51    bit_clr(INTMSK, 21);      // habilita interrupcoes globais
```

```
52 }
```

Listing 3.6: timer.c

Timer.c estabelece o funcionamento de um timer repetitivo com manipulação de interrupções dedicado a prover uma base de tempo para eventual utilização em outras partes do programa. A relevância first significativa repousa na variável de ticks, uma variável volátil de 32 bits que serve como a base temporal do sistema, incrementada a cada 100 milissegundos (0.1 segundos). Ocorrem quatros funções principais no código: trata_irq_timer0: Esta função é direccionada na happening de uma interrupção do timer 0, simplesmente incrementando o valor do ticks. get_ticks: Proporcionando leitura segura do valor atual do contador ticks, lembrando que é disable e predisable as interrupções do timer para prevenir alterações durante o processo de leitura. delay_tempo: Aceitando um parâmetro v, é providenciada um tempo delay baseando-se fora do contador ticks. Permanece inoperante até de v número de incrementos do contador ticks desde a chamada dessa função. inicia_timer0: Contribuição ao timer 0, configurando o mesmo para produzir uma interrupção a cada 100 milissegundos (0.1 segundos), cujo ocorrido incrementa o contador ticks. Neste contexto, o código configura uma base de tempo sistemático robusta através de interrupções recebido-se uma exterior indicação na passagem do tempo global, que é significante no microcontroladores e aplicações do tempo-real onde a domesticação de operação, routine tarefas e executar tarefas regularmente necessitam perflexibilidade exata referente ao tempo acontecido.

3.7 startup.s

```
1  /*
2   * Vetor de interrupcoes.
3   */
4  .section .reset, "ax"
5  .org 0
6  vetor:
7  b reset          // reset
8  b panic          // instrucao nao definida
9  b panic          // interrupcao de software
10 b panic          // abort de instrucao
11 b panic          // abort de dados
```

```
12 nop
13 b trata_irq      // interrupcao (tratamento em C)
14 b panic          // interrupcao rapida
15
16 reset:
17   b start
18
19 panic:
20   b panic
21
22 .text
23 .global delay, start
24
25 /*
26  * Ponto de entrada do programa.
27  */
28 start:
29    /*
30     * Configura modo do processador e pilha
31     */
32    mov r0, #0b10010               // modo IRQ
33    msr cpsr, r0
34    mov r13, #0x8000
35
36    mov r0, #0b10011               // modo SVR
37    msr cpsr, r0
38    ldr r13, =inicio_stack
39
40    /*
41     * Zera area bss.
42     */
43    mov r0, #0
44    ldr r1, =inicio_bss
45    ldr r2, =fim_bss
46 loop:
47    cmp r1, r2
48    bge cont
49    str r0, [r1], #4
50    b loop;
```

```
51 cont:
52    b main
53
54 /*
55  * Funcao utilitaria delay.
56  */
57 delay:
58    subs r0, r0, #1
59    bne delay
60    mov pc, lr
```

Listing 3.7: startup.s

Esse programa codificado em linguagem Assembly voltado para microcontrolador evaluator7t apresentando um procedimento de inicialização do sistema e uma função de atraso básica. O Assembly lista o vetor de endereços onde os manipuladores de várias interrupções podem ser encontrados. Nele quase todas as causas de interrupção são tratadas pela função de ”panic”que trava o sistema em um loop infinito. A exceção é a interrupção do tipo IRQ que é tratada pela função em C trata_irq. No ponto de entrada start do programa, diferentes modos de CPU e pilhas são configurados. Além disso, a área de memória BSS é posta a zero. Após estas operações de arranque inicial, o programa entra no ponto de entrada principal implementado em C, a função main. Também é implementada uma função utilitária delay, que realiza um retardamento simples de tempo consumindo através de um decréscimo progressivo do valor do registrador r0 até que este chegue ao valor zero. De forma concisa, este código assume as operações de preparação antes de entrar em uma rotina principal de programa verdadeiramente útil, que está pressupostamente escrita em linguagem C conforme indicado pela chamada para main na rotina start. Além disso, provê uma função utilitária que implementa um mecanismo crasso de realização de uma operação de atraso não-preemptiva.

CONCLUSÃO

Através da concepção e implementação da biblioteca, demonstramos como a abstração de hardware pode efetivamente permitir que os desenvolvedores explorem as capacidades da placa Evaluator 7T sem se perderem nos detalhes de baixo nível. A biblioteca fornece uma interface intuitiva e simplificada para interagir com as funcionalidades do hardware, permitindo que o processo de aprendizagem seja facilitado.

Ao analisar os resultados dos testes realizados, ficou evidente que a biblioteca demonstrou sua eficácia ao simplificar o acesso às funcionalidades da placa, resultando em ganhos mensuráveis de produtividade e em uma redução do esforço necessário para desenvolver aplicações complexas.

No entanto, reconhecemos que sempre há espaço para melhorias e expansões futuras. Novas funcionalidades podem ser adicionadas à biblioteca, tornando-a mais abrangente e adaptável a uma variedade maior de cenários de aplicação.

Em suma, a Biblioteca de Abstração de Hardware para a placa Evaluator 7T não apenas simplifica o processo de aprendizagem, mas também ressalta a importância geral da abstração de hardware como uma estratégia eficaz para enfrentar os desafios de complexidade associados à tecnologia de ponta. Também é notável a aplicação de diversos conceitos lecionados na disciplina, demonstrando domínio das ferramentas aprendidas ao longo do curso.

Referências

1 UNKNOWN. **ARM Evaluator-7T Board User Guide**. [S.l.]: Unknown, 2000.