Układ przerwań

Pliki źródłowe i ich przeznaczenie

Wszystkie te pliki razem tworzą **wbudowany system oparty na mikrokontrolerze lub FPGA.** Zapewniają interfejsy i funkcje do pracy z różnymi urządzeniami peryferyjnymioraz umożliwiają inicjalizację i zarządzanie tymi urządzeniami w głównym programie.

1. crt.S
2. gpio.h
3. interrupt.h
4. linker.ld
5. main.cpp
6. murax.h
7. prescaler.h
8. timer.h
9. uart.h

1. crt.S

Przeznaczenie:

Inicjalizacja systemu

Ten plik jest napisany w assemblerze i odpowiada za początkową inicjalizację systemu:

- **crtStart: Punkt wejścia dla** bootloadera lub resetu sprzętowego.

```
1   .global crtStart
2   .global main
3   .global irqCallback
4
5       .section    .start_jump,"ax",@progbits
6
7   crtStart:           // 0x 8000 0000
8     //long jump to allow crtInit to be anywhere
9     //do it always in 12 bytes
10    lui x2,       %hi(crtInit)
11    addi x2, x2,  %lo(crtInit)
12    jalr x1,x2
13    nop               // 0x 8000 0010
14
15  .section .text
16
17  .global  trap_entry
18  .align 5            //2^5 = 32 = 0x 20
19  trap_entry:
20
21    sw x1,  - 1*4(sp) // 0x 8000 0020
22    sw x5,  - 2*4(sp)
23    sw x6,  - 3*4(sp)
```

- trap_entry: Obsługa przerwań i wyjątków, zapisuje i przywraca stan rejestrów.

```
54      lw x31,  0*4(sp)
55      addi sp,sp,16*4
56      mret
57      .text
58
59
60    crtInit:
61      .option push
62      .option norelax
63      la gp, __global_pointer$
64      .option pop
65      la sp, _stack_start
66
67    bss_init:
68      la a0, _bss_start
69      la a1, _bss_end
70    bss_loop:
71      beq a0,a1,bss_done
72      sw zero,0(a0)
73      add a0,a0,4
74      j bss_loop
75    bss_done:
```

- **crtInit:** Ustawia stos, czyści sekcję .bss, wywołuje konstruktory C++, ustawia przerwania, a następnie wywołuje funkcję main.

```
77   ctors_init:
78     la a0, _ctors_start
79     addi sp,sp,-4
80   ctors_loop:
81     la a1, _ctors_end
82     beq a0,a1,ctors_done
83     lw a3,0(a0)
84     add a0,a0,4
85     sw a0,0(sp)
86     jalr  a3
87     lw a0,0(sp)
88     j ctors_loop
89   ctors_done:
90     addi sp,sp,4
91
92
93     li a0, 0x880     //880 enable timer -
94     csrw mie,a0
95     li a0, 0x1808     //1808 enable inter
96     csrw mstatus,a0
97
98     call main
99   infinitLoop:
100    j infinitLoop
```

2. gpio.h

Przeznaczenie:

Zarządzanie GPIO

Ten plik definiuje strukturę do pracy z rejestrem GPIO:

```c
#ifndef GPIO_H_
#define GPIO_H_



typedef struct
{
  volatile uint32_t INPUT;
  volatile uint32_t OUTPUT;
  volatile uint32_t OUTPUT_ENABLE;
} Gpio_Reg;



#endif /* GPIO_H_ */

```

- **Gpio_Reg: Struktura** do zarządzania stanem wyjść i ich kierunkami.
- **INPUT: Rejestr do** odczytu stanu wejść.
- **OUTPUT: Rejestr do** zapisu stanu wyjść.
- **OUTPUT_ENABLE:** Rejestr do włączania/wyłączania trybu wyjścia.

3. interrupt.h

Przeznaczenie: Zarządzanie przerwaniami

Ten plik definiuje strukturę i funkcje do pracy z kontrolerem przerwań:

- InterruptCtrl_Reg: Struktura do zarządzania przerwaniami.

- PENDINGS: Rejestr do śledzenia bieżących oczekujących przerwań.

- MASKS: Rejestr do maskowania (włączania/wyłączania) przerwań.

- interruptCtrl_init: Funkcja do inicjalizacji kontrolera p

```
#ifndef INTERRUPTCTRL_H_
#define INTERRUPTCTRL_H_

#include <stdint.h>

typedef struct
{
  volatile uint32_t PENDINGS;
  volatile uint32_t MASKS;
} InterruptCtrl_Reg;

static void interruptCtrl_init(InterruptCtrl_Re
    reg->MASKS = 0;
    reg->PENDINGS = 0xFFFFFFFF;
}

#endif /* INTERRUPTCTRL_H_ */
```

4. linker.ld

Przeznaczenie: Rozmieszczenie sekcji programu w pamięci

Ten plik jest skryptem linkera, który definiuje rozmieszczenie różnych sekcji programu w pamięci:

```
13  MEMORY {
14    RAM      (rwx): ORIGIN = 0x80000000, LENGTH = 128k
15  }
16
17  _stack_size = DEFINED(_stack_size) ? _stack_size : 8192;
18  _heap_size = DEFINED(_heap_size) ? _heap_size : 8192;
19
20  SECTIONS {
21
22    ._vector ORIGIN(RAM): {
23      *crt.o(.start_jump);
24      *crt.o(.text);
25    } > RAM
26
27    ._user_heap (NOLOAD):
28    {
29      . = ALIGN(8);
30      PROVIDE ( end = . );
31      PROVIDE ( _end = . );
32      PROVIDE ( _heap_start = .);
33      . = . + _heap_size;
34      . = ALIGN(8);
35      PROVIDE ( _heap_end = .);
36    } > RAM
37
```

- MEMORY: Definiuje dostępny obszar pamięci urządzenia.
- SECTIONS: Opisuje rozmieszczenie różnych sekcji (kod, dane, stos, sterta) w pamięci.

5. main.cpp

Przeznaczenie: Główny program

Ten plik zawiera główną program

```
12  #include <stdint.h>
13
14  #include "murax.h"
15
16  void print(const char*str){
17      while(*str){
18  //      uart_write(UART,*str);
19          str++;
20      }
21  }
22  void println(const char*str){
23      print(str);
24  //  uart_write(UART,'\n');
25  }
26
27  volatile int mati = 0;
28
29
```

- print i println: Funkcje do wyświetlania łańcuchów znaków.

- TimeR: Klasa do pracy z timerem.

```
30  // Modern C++ aproach
31  // ---
32  class TimeR {
33
34  public:
35      static inline  uint32_t readValue() {
36          return ptr->VALUE;
37      }
38
39  private:
40          static const Timer_Reg* ptr ;
41  };
42
43  const Timer_Reg* TimeR::ptr = reinterpret_cast<const Timer_Reg*>(0xF0020040);
44  // ---
45
```

```
46  int main() {
47
48  //      println("hello world arty a7 v1");
49
50
51          interruptCtrl_init(TIMER_INTERRUPT);
52          prescaler_init(TIMER_PRESCALER);
53          timer_init(TIMER_A);
54
55          TIMER_PRESCALER->LIMIT = 0x10;
56
57          TIMER_A->LIMIT = 0x10000-1;
58          TIMER_A->CLEARS_TICKS = 0x10002;
59
60          TIMER_INTERRUPT->PENDINGS = 0xF;
61          TIMER_INTERRUPT->MASKS = 0x1;
62
63      // Thread 0
64      while( 1){
65          ++mati;
66      }
67
68  }
69
70  extern "C" void irqCallback(){
71
72  static volatile int count = 0;
73      ++count;
74     TIMER_INTERRUPT->PENDINGS = 1;
75
76  }
```

- main: Główna funkcja programu, inicjalizuje przerwania, preskaler i timer, a następnie wykonuje nieskończoną pętlę.
- irqCallback: Funkcja obsługi przerwań, zwiększa licznik i resetuje przerwanie timera.

6. murax.h

Przeznaczenie: Integracja urządzeń peryferyjnych

Ten plik zawiera nagłówki dla różnych urządzeń peryferyjnych i definiuje makra dla ich adresów w pamięci:

```
#ifndef __MURAX_H__
#define __MURAX_H__

#include "timer.h"
#include "prescaler.h"
#include "interrupt.h"
#include "gpio.h"
#include "uart.h"

#define GPIO_A    ((Gpio_Reg*)(0xF0000000))
#define TIMER_PRESCALER ((Prescaler_Reg*)0xF0020000)
#define TIMER_INTERRUPT ((InterruptCtrl_Reg*)0xF0020010)
#define TIMER_A ((Timer_Reg*)0xF0020040)
#define TIMER_B ((Timer_Reg*)0xF0020050)
#define UART      ((Uart_Reg*)(0xF0010000))

#endif /* __MURAX_H__ */
```

Makra do dostępu do rejestrów urządzeń peryferyjnych (GPIO, timery, UART) po ich podstawowych adresach w pamięci.

7. prescaler.h

Przeznaczenie: Zarządzanie preskalerem

Ten plik definiuje strukturę i funkcję do pracy z preskalerem:

```c
#ifndef PRESCALERCTRL_H_
#define PRESCALERCTRL_H_

#include <stdint.h>


typedef struct
{
  volatile uint32_t LIMIT;
} Prescaler_Reg;

static void prescaler_init(Prescaler_Reg* reg){

}

#endif /* PRESCALERCTRL_H_ */
```

- Prescaler_Reg: Struktura do ustawiania limitu preskalera.
- prescaler_init: Inicjalizacja preskalera.

8. timer.h

Przeznaczenie: Zarządzanie timerem

Ten plik definiuje strukturę i funkcję do pracy z timerem:

```c
#ifndef TIMERCTRL_H_
#define TIMERCTRL_H_

#include <stdint.h>


typedef struct
{
  volatile uint32_t CLEARS_TICKS;
  volatile uint32_t LIMIT;
  volatile uint32_t VALUE;
} Timer_Reg;

static void timer_init(Timer_Reg *reg){
    reg->CLEARS_TICKS  = 0;
    reg->VALUE = 0;
}


#endif /* TIMERCTRL_H_ */
```

- Timer_Reg: Struktura do zarządzania timerem.
- timer_init: Inicjalizacja timera.

9. uart.h

Przeznaczenie: Zarządzanie UART

Ten plik definiuje strukturę i funkcje do pracy z UART

```c
#ifndef UART_H_
#define UART_H_


typedef struct
{
  volatile uint32_t DATA;
  volatile uint32_t STATUS;
  volatile uint32_t CLOCK_DIVIDER;
  volatile uint32_t FRAME_CONFIG;
} Uart_Reg;

enum UartParity {NONE = 0,EVEN = 1,ODD = 2};
enum UartStop {ONE = 0,TWO = 1};

typedef struct {
    uint32_t dataLength;
    enum UartParity parity;
    enum UartStop stop;
    uint32_t clockDivider;
} Uart_Config;

static uint32_t uart_writeAvailability(Uart_Reg *reg){
    return (reg->STATUS >> 16) & 0xFF;
}
static uint32_t uart_readOccupancy(Uart_Reg *reg){
    return reg->STATUS >> 24;
}

static void uart_write(Uart_Reg *reg, uint32_t data){
    while(uart_writeAvailability(reg) == 0);
    reg->DATA = data;
}

static void uart_applyConfig(Uart_Reg *reg, Uart_Config *config){
    reg->CLOCK_DIVIDER = config->clockDivider;
    reg->FRAME_CONFIG = ((config->dataLength-1) << 0) | (config->parity << 8) | (config->stop << 16);
}

#endif /* UART_H_ */
```

- Uart_Reg: Struktura do zarządzania UART.
- Uart_Config: Struktura do konfigurowania UART.
- uart_writeAvailability, uart_readOccupancy: Funkcje do sprawdzania dostępności zapisu i odczytu danych.
- uart_write: Funkcja do zapisu danych do UART.
- uart_applyConfig: Funkcja do stosowania konfiguracji UART.

Dziękuje za uwage!

Borys Rybalchuk 1i