Практическое задание №3

Волков Егор Алексеевич

20 января 2025 г.

1 Задание

- Включите на МК STM32 два интерфейса SPI и передайте данные с одного на другой. Подтверждением может быть скриншот отладчика.

2 Описание работы

Моргать будем встроенным светодиодом блупила.
Хостовая ОС: Debian 12.
Версия OpenOCD: 0.12.0
Версия arm-gcc: 12.2

2.1 Подключение библиотек

- `libopencm3/stm32/rcc.h` - используется для управления тактированием периферийных устройств, включая порты GPIO и модули прерываний.
- `libopencm3/stm32/gpio.h` - предоставляет функции для настройки и управления пинами GPIO, такими как настройка входов и выходов, а также режимов работы.
- `libopencm3/stm32/exti.h` - используется для настройки и работы с внешними прерываниями (EXTI), включая выбор линии, триггеры и активацию прерываний.
- `libopencm3/cm3/nvic.h` - предоставляет функции для настройки и активации прерываний в контроллере прерываний NVIC.

2.2 Настройка периферии

- **Функция** `gpio_setup()`: Включает тактирование для портов `GPIOA` и `GPIOC`, настраивает пин `PC13` как выход с частотой 2 МГц в режиме Push-Pull для управления встроенным светодиодом. Также настраивает пин `PA0` как вход с плавающим состоянием (не подтянутым, не заземленным).
- **Функция** `exti_setup()`: Включает тактирование модуля `AFIO` (для настройки альтернативных функций), настраивает внешний источник прерывания `EXTI0` на пин `PA0`, задает срабатывание прерыва-

ния на оба фронта (восходящий и нисходящий) и активирует запрос прерывания на линии `EXTI0`.

2.3 Реализация прерывания

- **Функция** `exti0_isr()`: Обработчик прерывания для линии `EXTI0`. При срабатывании прерывания переключается состояние пина `PC13` (светодиод), что индицирует факт срабатывания внешнего прерывания. Затем сбрасывается флаг прерывания для линии `EXTI0` через вызов `exti_reset_request()`.

2.4 Основной цикл

Основной цикл программы выполняет бесконечный цикл `while(1)` с инструкцией `nop`, которая позволяет процессору находиться в режиме ожидания до тех пор, пока не произойдёт внешнее прерывание на пине `PA0`.

3 Код blink_exti.c

```
/**
* @file blink_exti.c
* @author Volkov Egor (gole00201@gmail.com)
* @brief
* @version 0.1
* @date 2025-01-14
* @copyright Copyright (c) 2025
*/

#include <libopencm3/stm32/exti.h>
#include <libopencm3/cm3/nvic.h>
#include "gpio_init.h"

static void
exti_setup(void){
    rcc_periph_clock_enable(RCC_AFIO);
    nvic_enable_irq(NVIC_EXTI0_IRQ);
    exti_select_source(EXTI0, GPIOA);
    exti_set_trigger(EXTI0, EXTI_TRIGGER_BOTH);
    exti_enable_request(EXTI0);
}

void
exti0_isr(void){
    gpio_toggle(GPIOC, GPIO13);
    exti_reset_request(EXTI0);
}

int
main(void){
    gpio_setup();
    exti_setup();
    while(1){
        __asm volatile("nop");
    }
    return 0; // Houston, we have a problem =)
}
```

4 Код init.c

```
#include "init.h"

void
gpio_setup(void){
    rcc_periph_clock_enable(RCC_GPIOA);
    rcc_periph_clock_enable(RCC_GPIOC);
    gpio_set_mode(GPIOC, GPIO_MODE_OUTPUT_2_MHZ,
                  GPIO_CNF_OUTPUT_PUSHPULL, GPIO13);
    gpio_set_mode(GPIOA, GPIO_MODE_INPUT,
                  GPIO_CNF_INPUT_FLOAT, GPIO0);
}

void
exti0_isr(void){
    gpio_toggle(GPIOC, GPIO13);
    exti_reset_request(EXTI0);
}
```

5 Структура каталога с проектом

За основу взят: `https://github.com/libopencm3/libopencm3-template`

```
├── 2_task.pdf
├── 2_task.tex
├── blink_exti
│   ├── blink_exti.c
│   ├── init.c
│   ├── init.h // Стоит завести отдельные папки src и inc
│   └── Makefile
└── rules.mk
```

Путь к libopencm3 прописан в Makefile **(поменять при необходимости)**:

```
OPENCM3_DIR=/opt/libopencm3
```

Просто единицы трансляции прописываем дополительные, чтобы работала многофайловая структура:

```
CFILES = blink_exti.c init.c
```

6 Алгоритм прошивки

Используется блупил + stlink-v2 + openocd. Таргет "flash"для makefile:

```
$(OOCD) -f interface/$(OOCD_INTERFACE).cfg \
-f target/$(OOCD_TARGET).cfg \
-c "program $(realpath $(*).elf) verify reset exit" \
-c "set CPUTAPID 0" \ # Fuck yeah Chinees bluepill!
$(NULL)
```

7 Аппаратное подавление дребезга

Для аппаратного подавления дребезга контактов используется RC-фильтр, который сглаживает сигнал с кнопки перед подачей его на вход микроконтроллера. Описание схемы приведено ниже.

7.1 Компоненты:

- Резистор R (10 кОм).
- Конденсатор C (1 мкФ).
- Кнопка, подключенная к пину $PA0$.

7.2 Описание схемы подключения:

- Один вывод кнопки подключается к V_CC (например, 3.3 В).
- Другой вывод кнопки подключается к пину $PA0$ через RC-фильтр:
  - Резистор R подключается между выводом кнопки и землёй (GND).
  - Конденсатор C подключается параллельно кнопке, то есть между выводом кнопки и GND.
- Пин микроконтроллера $PA0$ подключается к точке между кнопкой и резистором.

Оно работало я клянусь (x2). Архив прекреплю на диск. Вот кстати именно за такую простоту я и как-то когда увидел эту библиотеку так порадовался. Все крайне понятно. Все контролируемо. Но при этом нет такого большого количества визуального шума когда пишешь

код под контроллер используя окошечные средства. При этом все в одном месте и не нужно что-то где-то под ВПНом искать. Все сразу из коробки работает без лишних вопросов. Если все же вопросы появляются то спокойно смотришь что происходит в API, в доках гуглишь что за регистры такие там дергаются и что за биты там выставляются и понимания прибавляется. Это как по мне лучше чем (дальше чистое ИМХО, я учусь и ещё ничего не знаю) набор каких-то весьма сомнительных структур, которые черт пойми где и как в конечном итоге обрабаыватюся. В общем я и на курсе с ардуино считал что AVR-LIBC $>$ Arduino ядро. Видимо у меня Торвальдс-трянка.