



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К домашнему заданию и лабораторным работам
по дисциплине:

**«Техника и элементная база средств цифровой
обработки сигналов»**

Вариант № 13

Выполнил: студент группы РЛ1-91

Рахимов А.

Проверил: преподаватель кафедры РК6

Шеин А.П.

Москва, 2024

СПИСОК ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

АЦП – аналого-цифровой преобразователь.

ЛР – лабораторная работа.

МК – микроконтроллер.

ПК – персональный компьютер.

ТЗ – техническое задание.

ЦАП – цифро-аналоговый преобразователь.

AHB – Advanced High-Performance Bus.

APB – Advanced Peripheral Bus.

CMSIS - Cortex Microcontroller Software Interface Standard.

EXTI – External Interrupts –блок внешних прерываний.

GPIO – General Pin Input-Output – контакты ввода-вывода общего назначения.

HSE – High Speed External oscillator – внешний высокочастотный кварц.

HSI – High Speed Internal crystal – внутренний высокочастотный
RCгенератор.

I2C – Inter-Integrated Circuit - последовательная асимметричная шина для
связи между интегральными схемами внутри электронных приборов.

IRQ – Interrupt Request – запрос на прерывание.

JTAG – Joint Test Action Group - аппаратный интерфейс на базе стандарта
IEEE 1149.1.

LSI – Low Speed Internal generator – внутренний низкочастотный RC-
генератор.

LSE – Low Speed External crystal - внешний низкочастотный кварцевый
генератор.

NVIC – Nested Vectored Interrupt Controller – векторный контроллер
прерываний.

PLL – Phase-Locked Loop – устройство подстройки частоты тактирования.

RCC – Reset and Clock Control – блок перезагрузки и управления
тактированием.

SPI – Serial Peripheral Interface - последовательный периферийный интерфейс.

UART - Universal asynchronous receiver/transmitter – универсальный асинхронный приёмопередатчик.

Содержание

Введение	5
ЛР№1: Минимальный проект с использованием МК	7
А. Лабораторный стенд	7
Б. Среда программирования МК	10
В. Управление портами МК	14
ЛР№2: Прерывание от таймера	20
ЛР№3: Прерывание от кнопки	25
ЛР№4: Индикация	30
Домашнее задание	39
Общие сведения:	39
П1: Тактировка процессора (пересекается с ЛР№1.В)	44
П.2: Прерывание по таймеру (подробнее в ЛР№2)	48
П.3: Отладка через UART	49
П.4: Отладка через символьный дисплей (подробнее в ЛР№4)	54
Заключение	56

Введение

Для выполнения ЛР и ДЗ был разработан лабораторный стенд, включающий минимальный набор для освоения базовых навыков программирования МК. Проприетарное программное обеспечение для разработки, отладки и использования стенда по возможности было заменено на инструменты с открытым исходным кодом. Программы для МК создавались на низкоуровневых языках С и ассемблера. «Низкоуровнеость» поддерживалась и в части работы с регистрами МК. Там, где это было целесообразно, состояние регистров изменялось напрямую битовыми масками.

Для получения навыков разработки встраиваемых систем в любом случае необходимо начать с базовых задач, таких как тактирование, конфигурация портов ввода-вывода, настройка прерываний и простых последовательных интерфейсов. Эти задачи и были выполнены в ходе данной работы.

На этапе отладки и уточнения принципиальной схемы стенд собран на беспаячной макетной плате с «джамперами» в качестве соединений (рис 1). Такое исполнение значительно ускоряет проектирование электронных устройств. Итоговый вариант монтируется на обычной двусторонней макетной плате с металлизированными сквозными отверстиями (рис 2). Для загрузки программ в МК и отладки по UART стенд подключается к ПК.

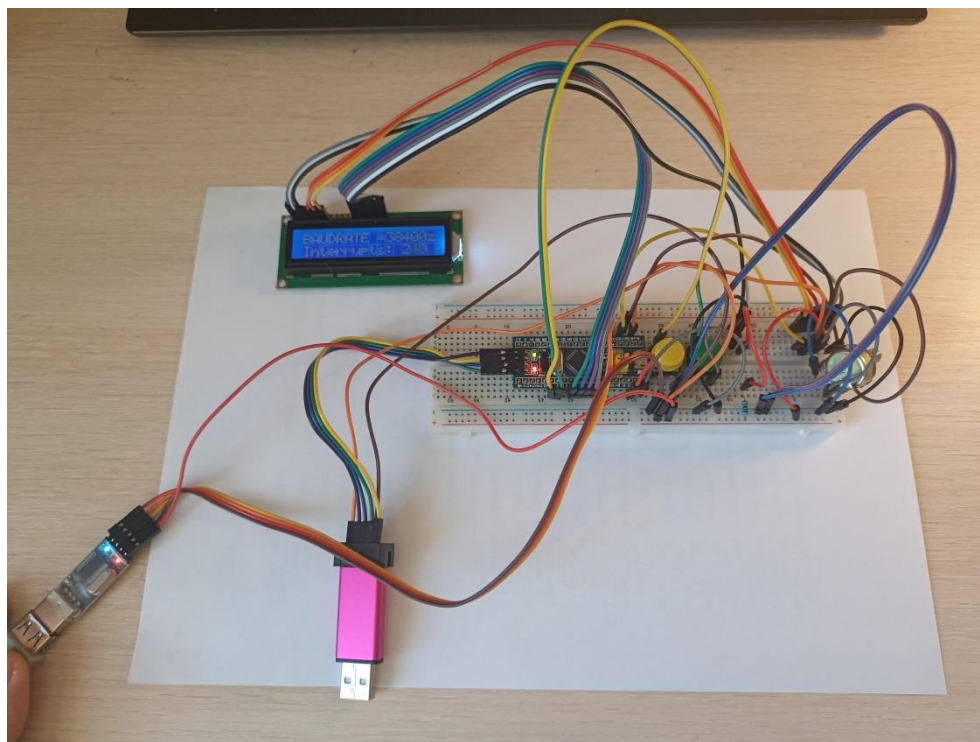


Рис 1 -макет лабораторного стенда.

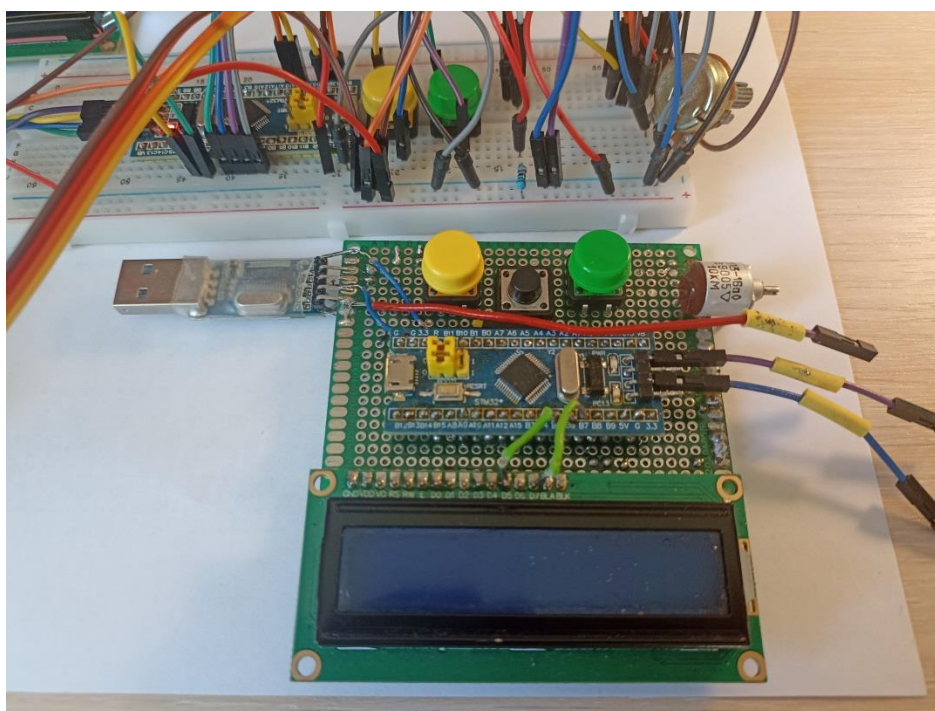


Рис 2 -размещение элементов стенда на плате.

ЛР№1: Минимальный проект с использованием МК

А. Лабораторный стенд

Процессор МК STM32F103C8T6 с ядром ARM CortexM3 на популярной отладочной плате Blue Pill (рис 3) выбран для проекта из-за большого разнообразия учебных материалов и вспомогательных библиотек для работы со всевозможной периферией.

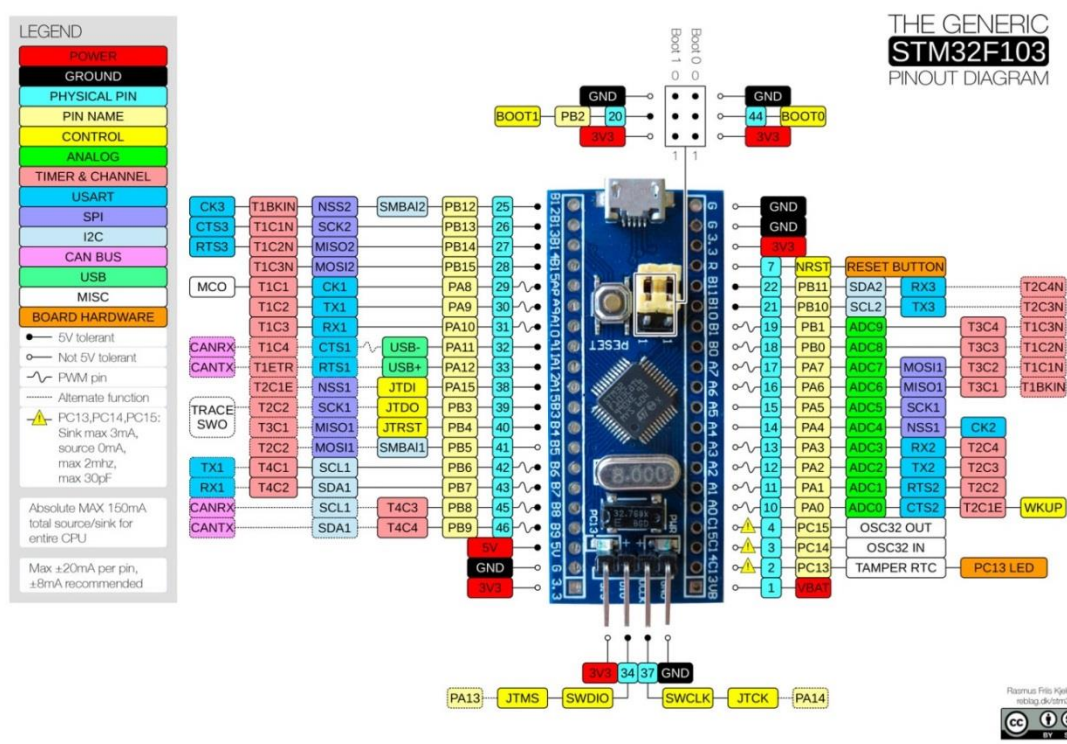
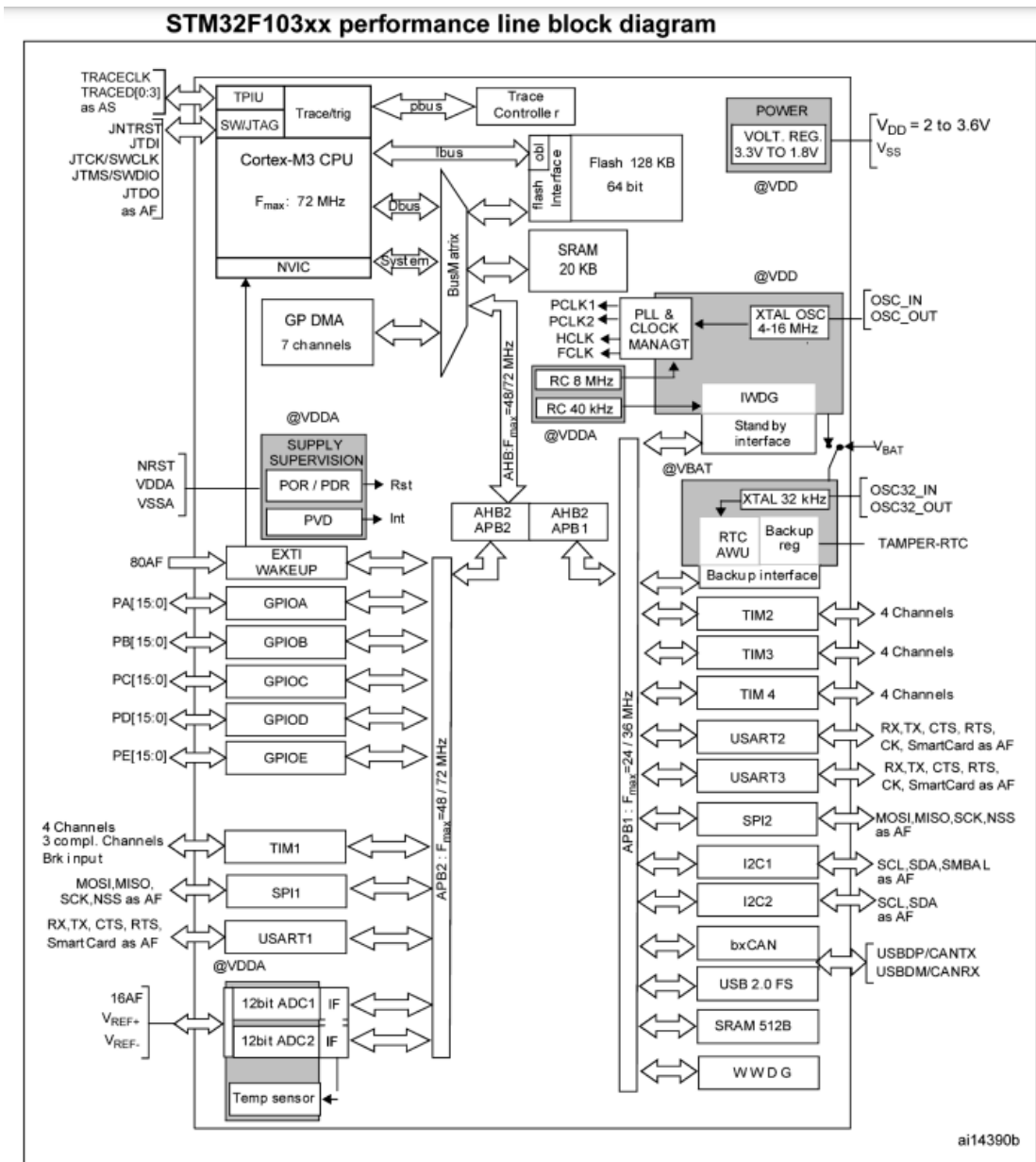


Рис 3 - диаграмма назначения контактов платы Blue Pill .

Конечно, МК линейки STM32 уступают в этом платам Arduino (процессоры AVR), но в промышленности МК STM оказываются гораздо более полезными. Восьмибитным процессорам AVR, пользующимся популярностью у любителей, не хватает многих характеристик. В частности, они имеют низкую рабочую частоту, не такие стабильные параметры, ограниченный набор периферийных устройств и более короткий список инструкций процессора, что в свою очередь также плохо сказывается на производительности.

В STM32 же на аппаратном уровне реализованы контроллеры всех основных интерфейсов (рис 4), гибкая система тактирования и исчерпывающий список настроек процессора внешних устройств. Эта и некоторые другие схемы взяты непосредственно из руководства к процессорам STM32F103



1. $T_A = -40^\circ\text{C}$ to $+105^\circ\text{C}$ (junction temperature up to 125°C).
2. AF = alternate function on I/O port pin.

Рис 4 - функциональная-схема кристалла МК .

Кроме отладочной платы Blue Pill стенд включает в себя программатор ST-Link-v2 для загрузки прошивки, переходник USB-TTL для контроля выполнения программы МК путём вывода текстовых сообщений от МК

на ПК по UART, простой символьный дисплей типа 1602A с параллельным 4-битным интерфейсом, кнопки и резисторы. Принципиальная схема стенда представлена на рис 5. От сервомотора и реле, для управления им на данном этапе пришлось отказаться.

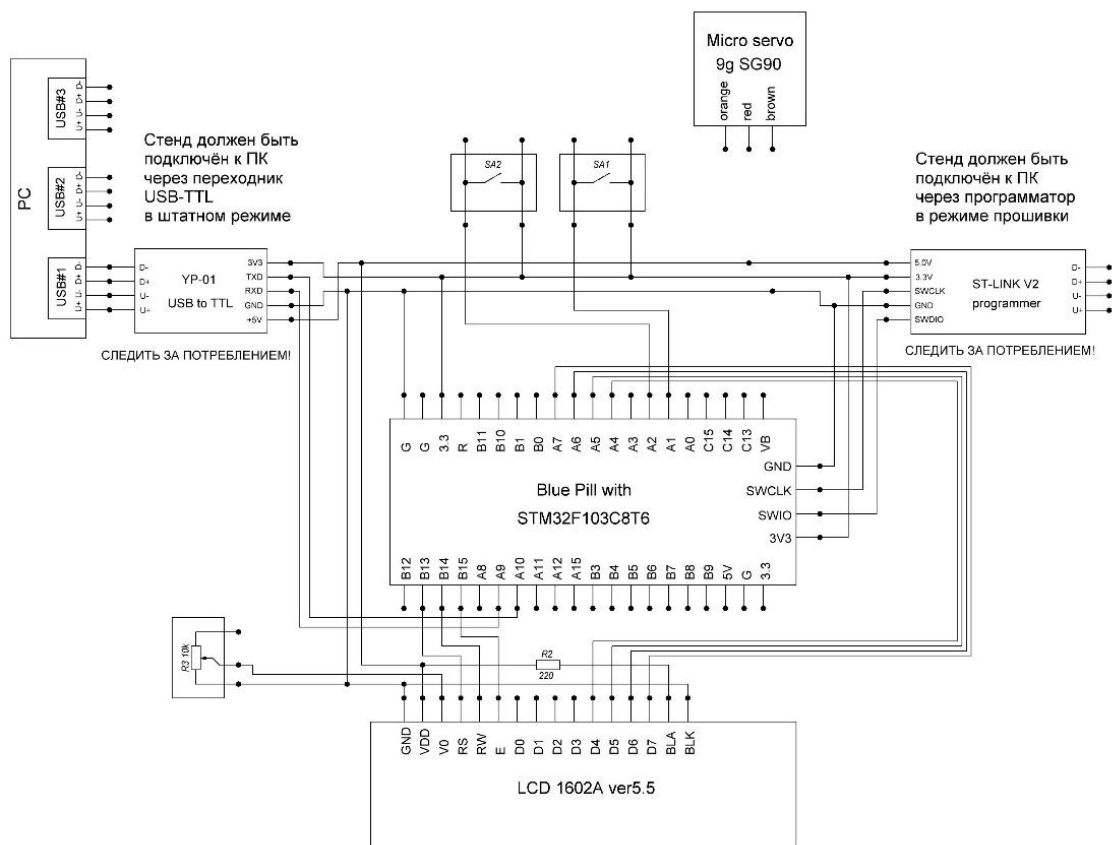


Рис 5 - принципиальная схема лабораторного стенда .

Из-за недостатка на рынке оригинальных моделей устройств, показанных на рис 5, были использованы их полные аналоги. При выполнении простых задач, описанных в отчёте, проблем аппаратного уровня не было выявлено.

Б. Среда программирования МК

Существует несколько подходов к написанию программ для МК. Упрощённо, их можно рассмотреть как более и менее высокоуровневые (рис 6).

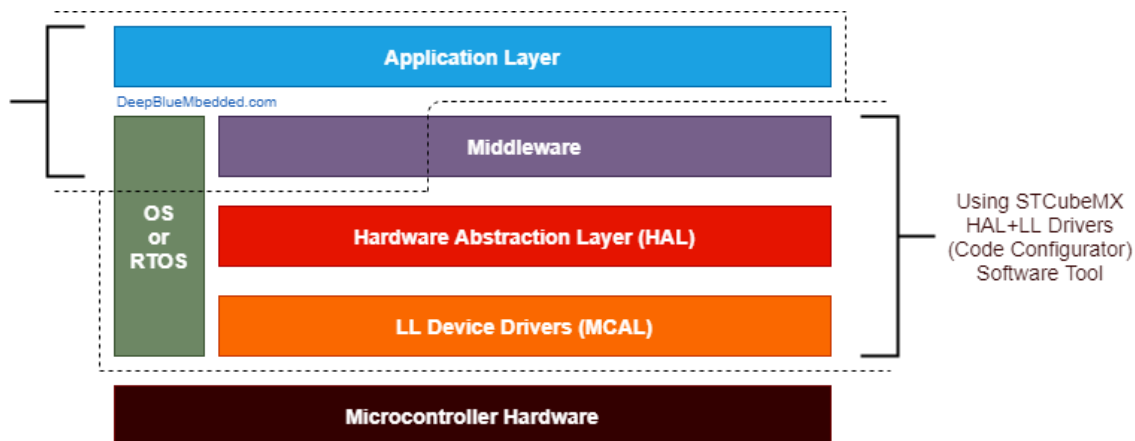


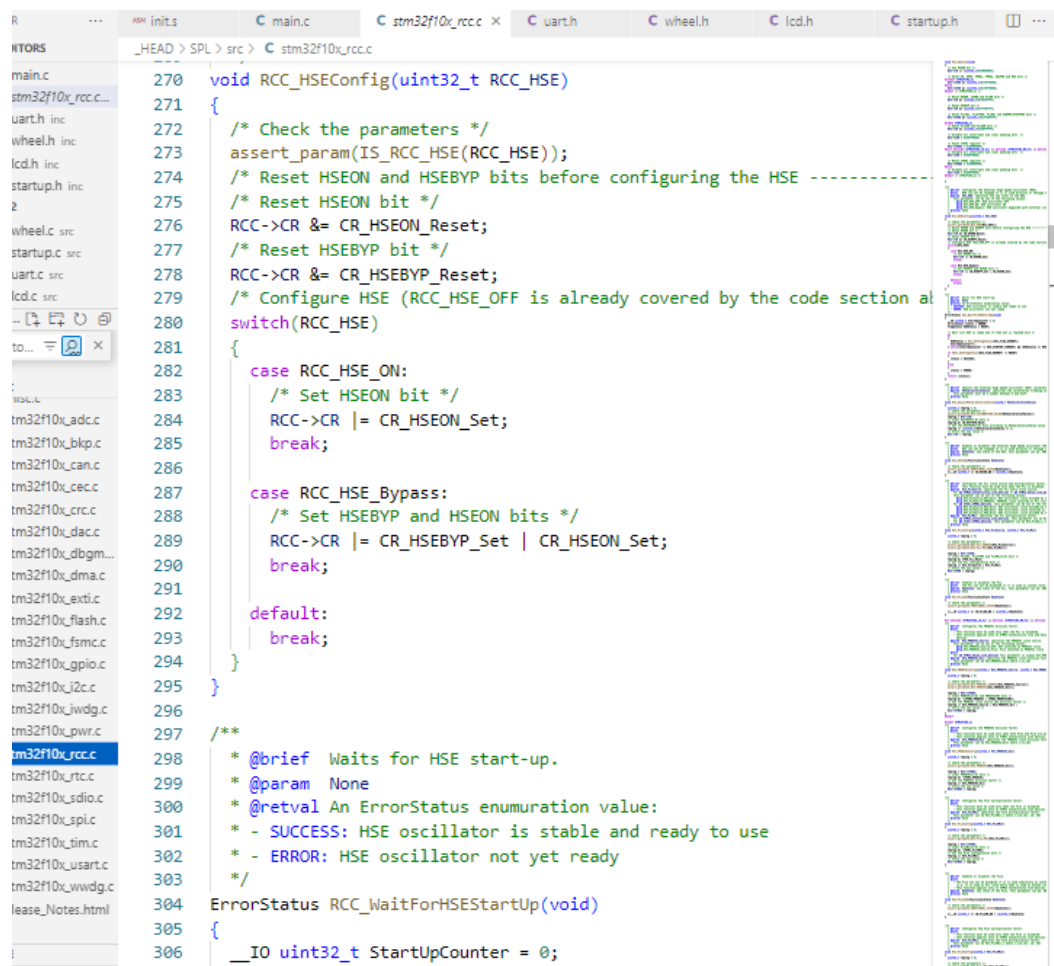
Рис 6 - иерархия модулей во встраиваемом программировании .

На нижний уровень можно поместить программирование на языке ассемблера. Тогда - по аналогии – к низкоуровневым инструментам можно отнести библиотеку CMSIS (пример на рис 7), которая в основном состоит из именованных масок для обращения к битам регистров, в которых хранятся настройки устройств в составе МК и некоторые данные.

```
2330
2331 #define GPIO_CRH_CNF12          ((uint32_t)0x00C0000)
2332 #define GPIO_CRH_CNF12_0        ((uint32_t)0x00040000)
2333 #define GPIO_CRH_CNF12_1        ((uint32_t)0x00080000)
2334
2335 #define GPIO_CRH_CNF13          ((uint32_t)0x00C0000)
2336 #define GPIO_CRH_CNF13_0        ((uint32_t)0x00400000)
2337 #define GPIO_CRH_CNF13_1        ((uint32_t)0x00800000)
2338
2339 #define GPIO_CRH_CNF14          ((uint32_t)0x0C00000)
2340 #define GPIO_CRH_CNF14_0        ((uint32_t)0x04000000)
2341 #define GPIO_CRH_CNF14_1        ((uint32_t)0x08000000)
2342
2343 #define GPIO_CRH_CNF15          ((uint32_t)0xC000000)
2344 #define GPIO_CRH_CNF15_0        ((uint32_t)0x40000000)
2345 #define GPIO_CRH_CNF15_1        ((uint32_t)0x80000000)
2346
2347 /*!<***** Bit definition for GPIO_IDR register *****>
2348 #define GPIO_IDR_IDR0           ((uint16_t)0x0001)
2349 #define GPIO_IDR_IDR1           ((uint16_t)0x0002)
2350 #define GPIO_IDR_IDR2           ((uint16_t)0x0004)
2351 #define GPIO_IDR_IDR3           ((uint16_t)0x0008)
2352 #define GPIO_IDR_IDR4           ((uint16_t)0x0010)
2353 #define GPIO_IDR_IDR5           ((uint16_t)0x0020)
2354 #define GPIO_IDR_IDR6           ((uint16_t)0x0040)
2355 #define GPIO_IDR_IDR7           ((uint16_t)0x0080)
2356 #define GPIO_IDR_IDR8           ((uint16_t)0x0100)
```

Рис 7 - маски для побитового управления портами ввода-вывода из CMSIS.

Для уровня HAL также существует свой одноимённый набор модулей с переменными и функциями. HAL является преемником более старой SPL. OS, RTOS – это операционная система и планировщик задач соответственно. На основе этих «библиотек» программист пишет прошивку МК под свои задачи. Даже без ООП (в языке C отсутствуют классы) можно реализовать любые алгоритмы. Лишь бы хватило выводов и вычислительной мощности МК. Пример функции из SPL приведён на рис 8:



```
270 void RCC_HSEConfig(uint32_t RCC_HSE)
271 {
272     /* Check the parameters */
273     assert_param(IS_RCC_HSE(RCC_HSE));
274     /* Reset HSEON and HSEBYP bits before configuring the HSE -----
275     /* Reset HSEON bit */
276     RCC->CR &= CR_HSEON_Reset;
277     /* Reset HSEBYP bit */
278     RCC->CR &= CR_HSEBYP_Reset;
279     /* Configure HSE (RCC_HSE_OFF is already covered by the code section at
280     switch(RCC_HSE)
281     {
282     case RCC_HSE_ON:
283         /* Set HSEON bit */
284         RCC->CR |= CR_HSEON_Set;
285         break;
286     case RCC_HSE_Bypass:
287         /* Set HSEBYP and HSEON bits */
288         RCC->CR |= CR_HSEBYP_Set | CR_HSEON_Set;
289         break;
290     default:
291         break;
292     }
293 }
294
295 /**
296  * @brief Waits for HSE start-up.
297  * @param None
298  * @retval An ErrorStatus enumeration value:
299  * - SUCCESS: HSE oscillator is stable and ready to use
300  * - ERROR: HSE oscillator not yet ready
301  */
302 ErrorStatus RCC_WaitForHSEStartUp(void)
303 {
304     __IO uint32_t StartUpCounter = 0;
305     uint32_t tmp = 0;
306     while (tmp == 0)
307     {
308         tmp = RCC->CR & CR_HSEON_Set;
309         if (tmp != 0)
310             break;
311         StartUpCounter++;
312         if (StartUpCounter == HSE_STARTUP_TIMEOUT)
313         {
314             return ERROR;
315         }
316     }
317     return SUCCESS;
318 }
```

Рис 8 - функция настройки внешнего кварца из библиотеки SPL .

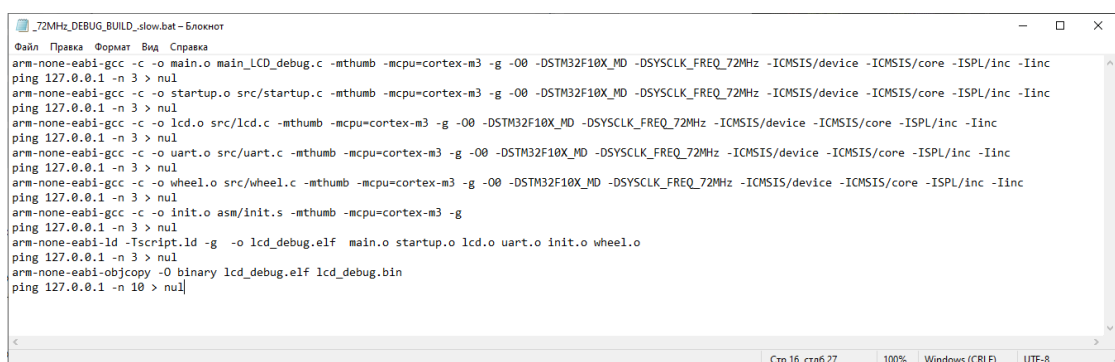
Часто инженеры сами переписывают некоторые низкоуровневые драйвера под себя. Зачем? Потому что готовые библиотеки и средства генерации кода могут не подойти для выполнения конкретной задачи. В

любой программе для ЭВМ некоторые части будут более нагружены, чем другие. Отсюда возникает потребность в максимально возможной оптимизации кода. Разработчик получает больше контроля над МК, если обращается к регистрам сам напрямую, а также лучше представляет себе его работу на каждом отдельном этапе, вследствие чего быстрее находит и исправляет ошибки. Однако, недостатками такого подхода могут являться неоправданно долгая разработка простых программ и плохая читаемость кода, особенно для программиста, который не сталкивался с библиотеками низкого уровня.

Для выполнения разработки стенда понадобились следующие инструменты:

- Visual Studio Code - одна из самых распространённых сред программирования в настоящее время. Но в данной работе использовалась лишь в качестве редактора программного кода (рис 8).

- ARM GNU Toolchain с консольным интерфейсом для сборки версий проекта. Инструмент создаёт файлы объектов, выполняет линковку и компиляцию. Создание batch-файлов (пример на рис 9) немного упростило и ускорило работу с ним. В файле script.ld проекта лежат указатели на области памяти конкретного процессора.



```
arm-none-eabi-gcc -c -o main.o main_lcd_debug.c -mthumb -mcpu=cortex-m3 -g -O0 -DSTM32F10X_MD -DSYSCLK_FREQ_72MHz -ICMSIS/device -ICMSIS/core -ISPL/inc -Iinc
ping 127.0.0.1 -n 3 > nul
arm-none-eabi-gcc -c -o startup.o src/startup.c -mthumb -mcpu=cortex-m3 -g -O0 -DSTM32F10X_MD -DSYSCLK_FREQ_72MHz -ICMSIS/device -ICMSIS/core -ISPL/inc -Iinc
ping 127.0.0.1 -n 3 > nul
arm-none-eabi-gcc -c -o lcd.o src/lcd.c -mthumb -mcpu=cortex-m3 -g -O0 -DSTM32F10X_MD -DSYSCLK_FREQ_72MHz -ICMSIS/device -ICMSIS/core -ISPL/inc -Iinc
ping 127.0.0.1 -n 3 > nul
arm-none-eabi-gcc -c -o uart.o src/uart.c -mthumb -mcpu=cortex-m3 -g -O0 -DSTM32F10X_MD -DSYSCLK_FREQ_72MHz -ICMSIS/device -ICMSIS/core -ISPL/inc -Iinc
ping 127.0.0.1 -n 3 > nul
arm-none-eabi-gcc -c -o wheel.o src/wheel.c -mthumb -mcpu=cortex-m3 -g -O0 -DSTM32F10X_MD -DSYSCLK_FREQ_72MHz -ICMSIS/device -ICMSIS/core -ISPL/inc -Iinc
ping 127.0.0.1 -n 3 > nul
arm-none-eabi-gcc -c -o init.o asm/init.s -mthumb -mcpu=cortex-m3 -g
ping 127.0.0.1 -n 3 > nul
arm-none-eabi-ld -Tscript.ld -g -o lcd_debug.elf main.o startup.o lcd.o uart.o init.o wheel.o
ping 127.0.0.1 -n 3 > nul
arm-none-eabi-objcopy -O binary lcd_debug.elf lcd_debug.bin
ping 127.0.0.1 -n 10 > nul
```

Рис 9 - инструкция для сборки проекта с помощью ARM GNU Toolchain.

- Утилитой STM32 ST-LINK (рис 10) бинарный файл прошивки загружался в МК. Функции проверки целостности чипа и обнуления памяти программ также были там доступны.

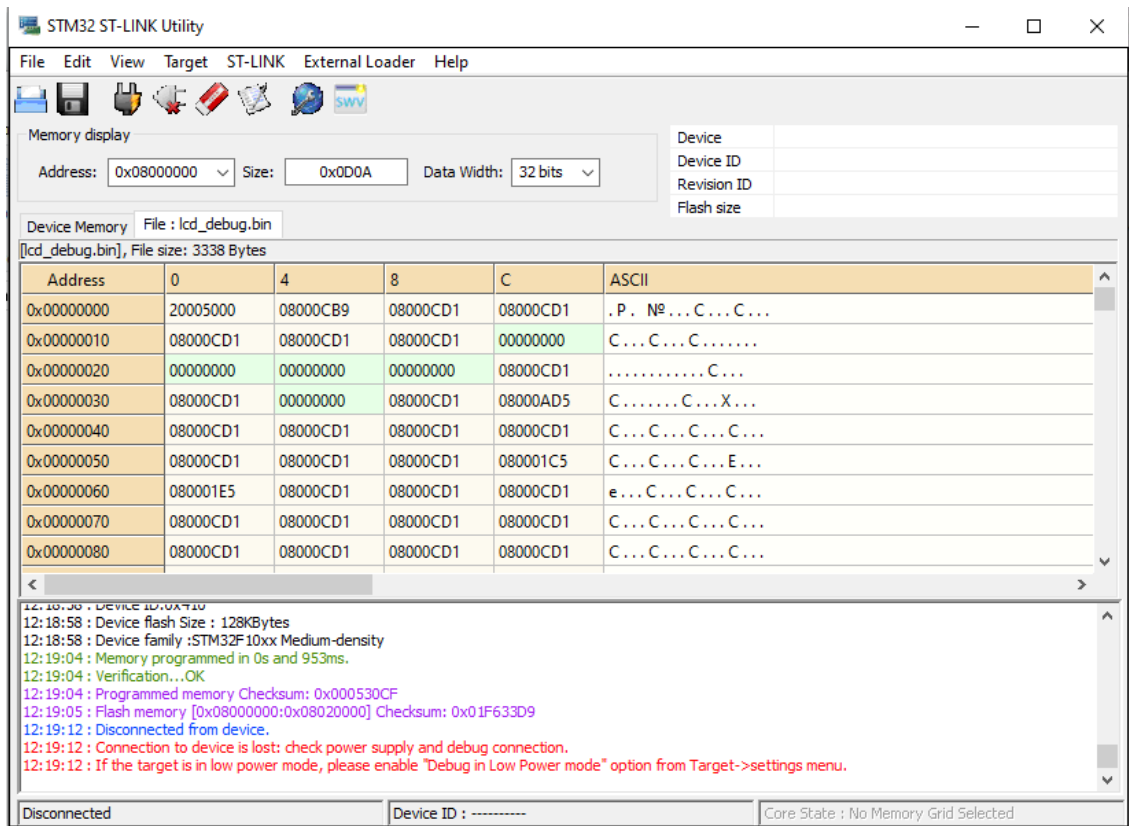


Рис 10 -представление прошивки в памяти программ .

- Terminal by Bray позволяет получать с МК сообщения по протоколу UART. Пример процесса отладки приведён на рис 11.

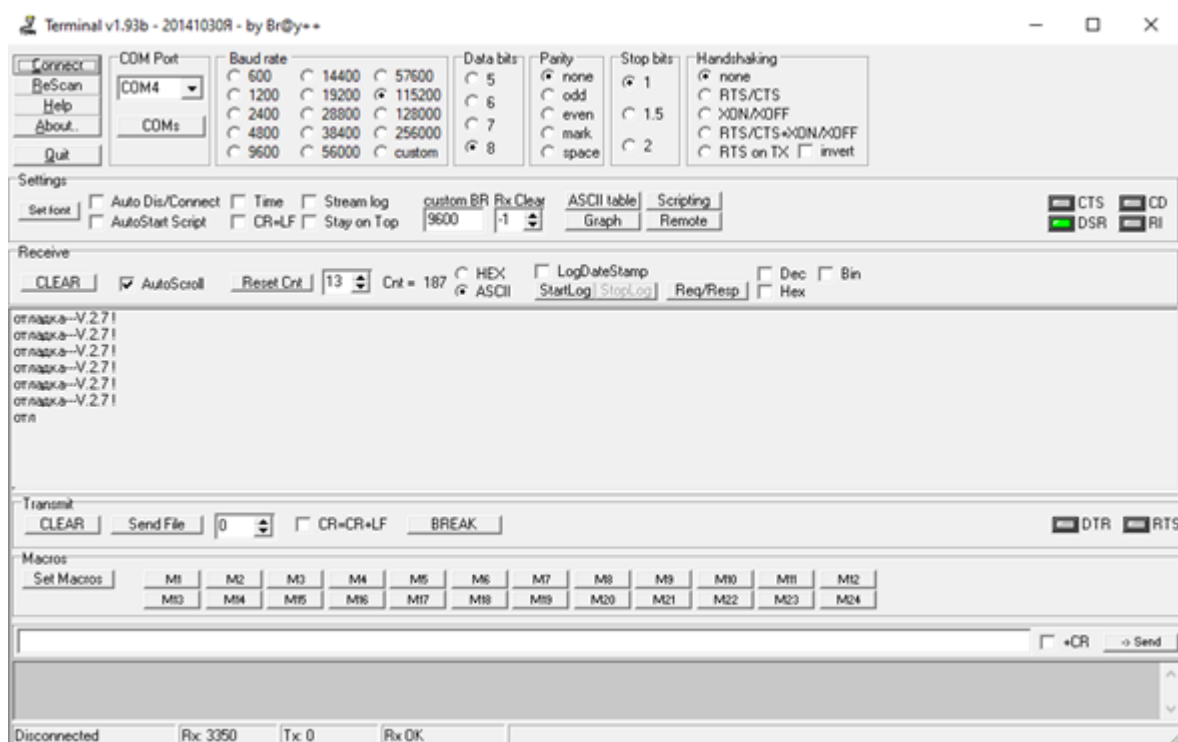


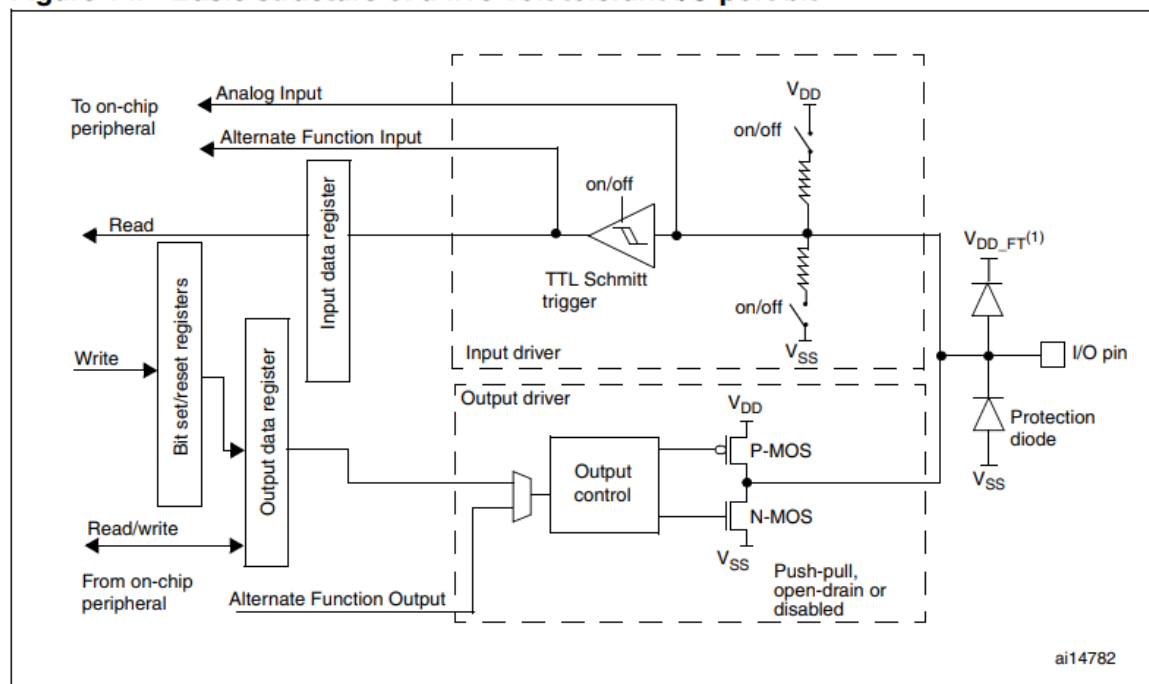
Рис 11 - интерфейс терминала .

- Cube MX и Keil uVision в проекте не использовались.

В. Управление портами МК

Порт МК по сути является параллельный регистром, выводы которого «торчат наружу». Если говорить точнее, к выводам порта подключены одновременно сразу несколько регистров, но при выполнении простых операций ввода-вывода не обязательно помнить значения каждого их бита наизусть. Особенно, если пользоваться библиотекой HAL или её аналогами. Схема отдельного вывода порта общего назначения МК показана на рис 12:

Figure 14. Basic structure of a five-volt tolerant I/O port bit



1. V_{DD_FT} is a potential specific to five-volt tolerant I/Os and different from V_{DD} .

Рис 12 - схема ножки порта .

Далее настройку портов на примере обычного мигания встроенным светодиодом. В начале GPIO-порт, как и любую другую периферию, нужно затактировать. А это, в свою очередь, требует настройки частоты процессора. Схема тактирования МК STM32F1 демонстрируется на рис 13:

--- предделители для шин АНВ и APB2 и APB1 устанавливаются равными 1,

--- предделитель PLL также хадаётся равным 1

--- устанавливается множитель PLL в зависимости от требуемой частоты процессора

--- PLL выбирается источником тактирования ядра

--- выполняется проверка того, что PLL действительно стал источником тактирования

```
_HEAD > src > C startup.c
89
90
91 // ----- SYS CLK TO REQUIRED FREQ -----
92
93 #ifdef FCPU_8 /* PLL configuration: PLLCLK = HSE * 1 = 8 MHz */
94 SetSysClockToHSE_Directly(); // uncomment func def later
95 #endif
96
97 /*
98  Sets System clock frequency to desirable freq and configure HCLK, PCLK2
99  and PCLK1 prescalers.
100  This function should be used only after reset.
101  // allowed cpu_freq_in_MHz values: 16, 24, 32, 40, 48, 56, 64, 72
102  */
103 void SetSysClockToReqFreq(uint8_t cpu_freq_in_MHz)
104 {
105
106     /* SYSCLK, HCLK, PCLK2 and PCLK1 configuration -----*/
107     /* Enable HSE */
108     RCC->CR |= ((uint32_t)RCC_CR_HSEON);
109
110     /* Wait till HSE is ready and if Time out is reached exit */
111     while (!(RCC->CR & RCC_CR_HSERDY))
112
113     /* Enable Prefetch Buffer */
114     FLASH->ACR |= FLASH_ACR_PRFTBE;
115
116     /* Flash 2 wait state */
117     FLASH->ACR &= (uint32_t)((uint32_t)~FLASH_ACR_LATENCY);
118     FLASH->ACR |= (uint32_t)FLASH_ACR_LATENCY_2;
119
120
121     /* HCLK = SYSCLK */
122     RCC->CFGR |= (uint32_t)RCC_CFGR_HPRE_DIV1;
123
124     /* PCLK2 = HCLK */
125     RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE2_DIV1;
126
```

Рис 14 - начало функции настройки МК на заданную частоту .

Процедура SetSysClockToReqFreq() вызывается в SystemInit() из startup.c. И только после завершения их работы мы можем начать работу с портами GPIO. Конечно, предварительно их затактировав (рис 15):

```

59
60 void gpio_init()
61 {
62     // --- enable GPIO sync ----
63     RCC -> APB2ENR |= RCC_APB2Periph_GPIOA;
64     RCC -> APB2ENR |= RCC_APB2Periph_GPIOB;
65     RCC -> APB2ENR |= RCC_APB2Periph_GPIOC;
66
67     // --- GPIO setup ----
68     GPIOC -> CRH &= ~(uint32_t)(0xf<<20); // Режим 00 - Push-Pull
69     GPIOC -> CRH |= (uint32_t)(0x3<<20); // Скорость 11. (Max Speed 50MHz)
70     GPIOC -> BSRR = GPIO_BSRR_BR13; // LED OFF??
71
72     GPIOA -> CRL = (uint32_t)(0x33333333); // Тоже Push-Pull 50MHz
73     GPIOA -> CRH = (uint32_t)(0x33333333);
74     GPIOB -> CRL = (uint32_t)(0x33333333);
75     GPIOB -> CRH = (uint32_t)(0x33333333);
76 }
77

```

Рис 15 - функция настройки портов ввода-вывода .

Строки 67-75 отвечают за конфигурацию режима работы выводов («пинов»). Действительно, каждый из них (независимо от других) может быть настроен в нужный режим. Список режимов есть здесь (рис 16):

Table 18. Port bit configuration table

Configuration mode		CNF1	CNF0	MODE1	MODE0	PxODR register		
Выход GPIO т.е. просто порт	Тяни-толкай	0	0	01 10 11 see Table 19		0 or 1		
	Открытый коллектор		1			0 or 1		
Альтернативный режим т.е. от периферии	Тяни-толкай	1	0					Пофигу
	Открытый коллектор		1					Пофигу
Вход	Аналоговый (АЦП)	0	0	00				Пофигу
	Вход с Hi-Z		1					Пофигу
	Вход, подтяжка вниз	1	0			0		
	Вход, подтяжка вверх					1		

Table 19. Output MODE bits

MODE[1:0]	Meaning	
00	Reserved	
01	Max. output speed	10 MHz
10	Max. output speed	2 MHz
11	Max. output speed	50 MHz

Рис 16 - возможные режимы работы каждого разряда портов ввода-вывода общего назначения.

Непосредственно смена режима производится записью в регистры GPIOx_CRL и GPIOx_CRH (4 бита для каждого пина). Запись GPIOx означает, что существуют соответствующие регистры для каждого порта (GPIOA, GPIOB, GPIOC и т. д.), а буква L в CRL обозначает принадлежность к выводам под номерами 0-7. (CRH соответственно 8-15)

General-purpose and alternate-function I/Os (GPIOs and AFIOs)

RM0008

8.2.2

Port configuration register high (GPIOx_CRH) (x=A..G)

Address offset: 0x04

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2

CNFy[1:0]: Port x configuration bits (y= 8 .. 15)

These bits are written by software to configure the corresponding I/O port.

Refer to [Table 18: Port bit configuration table on page 148](#).

In input mode (MODE[1:0]=00):

00: Analog mode

01: Floating input (reset state)

10: Input with pull-up / pull-down

11: Reserved

In output mode (MODE[1:0] > 00):

00: General purpose output push-pull

01: General purpose output Open-drain

10: Alternate function output Push-pull

11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0

MODEy[1:0]: Port x mode bits (y= 8 .. 15)

These bits are written by software to configure the corresponding I/O port.

Refer to [Table 18: Port bit configuration table on page 148](#).

00: Input mode (reset state)

01: Output mode, max speed 10 MHz.

10: Output mode, max speed 2 MHz.

11: Output mode, max speed 50 MHz.

8.2.3

Port input data register (GPIOx_IDR) (x=A..G)

8.2.3 Port input data register (GPIOx_IDR) (x=A..G)

Рис 17 - регистр настройки режима порта .

А ведь у многих выводов МК существует одна-две альтернативные функции. Они примерно описаны диаграммой на рисунке 3 (стр. 7) . Таблицы переопределений функций и альтернативных режимов выводов можно найти в мануале.

Встроенный светодиод соединён с выводом 13 порта C, который тоже был настроен. Теперь можно помогать светодиодом:

```

234     for(;;)
235     {
236         // tim_4Hz_delay();
237         delay_ms(250);
238         itrpts_counter = itrpts_counter + 1;
239         GPIOC -> ODR ^= GPIO_Pin_13; // led toggle
240     }

```

ЛР№2: Прерывание от таймера

Программа МК записывается во Flash-память, которую можно считать энергонезависимой. Инструкции выполняются процессором последовательно и, в случае STM32F103 (рис 18) на выполнение одной инструкции уходит не менее одного такта процессора.

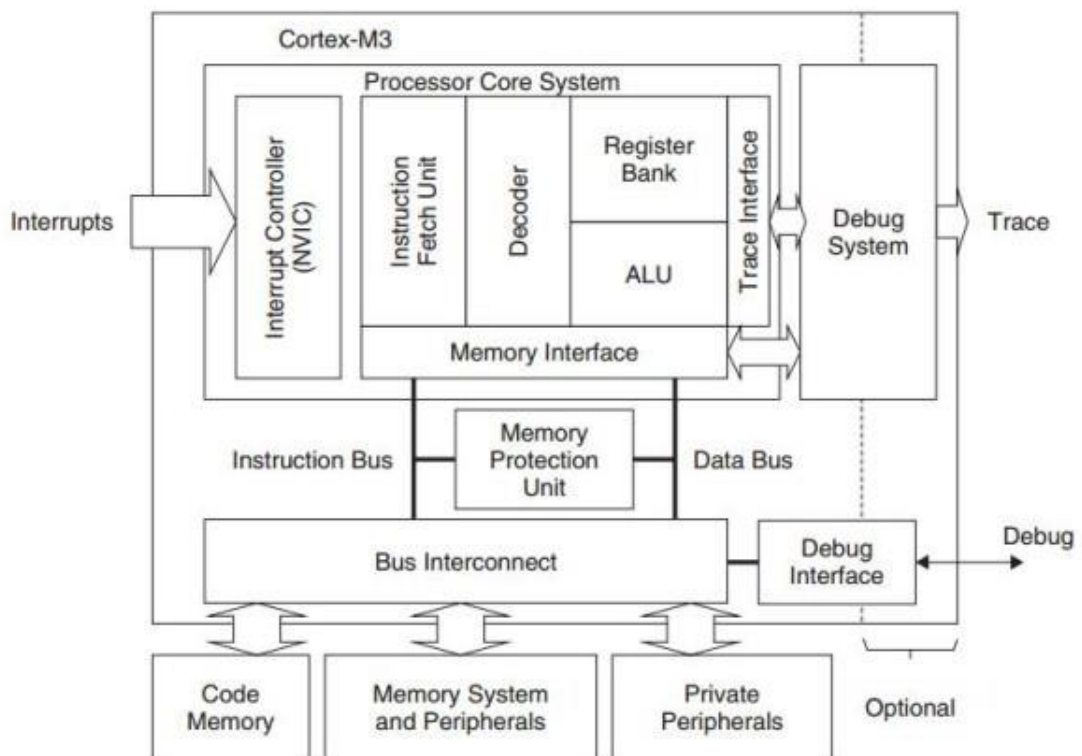


Рис 18 - блок-схема процессора Cortex-M3.

В учебных проектах может не быть заметно, как быстро могут быть достигнуты пределы вычислительной мощности МК. Максимальная частота работы рассматриваемого процессора – всего 72МГц . Одна операция деления с плавающей точкой может занимать во времени десятки и сотни тактов. И хорошо, если задача не требует успевать выполнять вычисления в реальном времени с большим потоком данных.

Действительно, в мире достаточно систем, требующих громоздких вычислений в реальном времени. При этом, часто необходимо выполнять несколько разных задач, которые ещё и имеют сложную зависимость друг от

друга. Возникает необходимость выполнять эти задачи параллельно и грамотно выбирать время для переключения между ними.

Здесь нужно вспомнить, какие функции в программировании называются блокирующими. На уровне МК объяснить их особенность можно на примере функции временной задержки. «В лоб» задержку в 1 секунду можно реализовать, вызвав функцию, которая запустит пустой цикл с количеством итераций, примерно численно равным частоте МК в Гц (рис 19).

```
136 void dummy_loop(uint32_t count)
137 {
138     while(--count);
139 }
140
141
142 void dummy_led_dim(uint8_t n)
143 {
144     for(uint8_t i=0; i<n; i++)
145     {
146         GPIOC -> BSRR = GPIO_BSRR_BS13;
147         dummy_loop(2500000);
148         GPIOC -> BSRR = GPIO_BSRR_BR13;
149         dummy_loop(500000);
150     }
151 }
152
```

Рис 19 - блокирующая задержка.

Но МК STM32 имеет единственное ядро Cortex-M3. Как процессор будет одновременно выполнять основную программу и считать такты? К счастью, на кристалле помимо ядра размещается ещё и периферия. Много периферии. И многие периферийные устройства способны выполнять некоторые элементарные задачи самостоятельно. Речь идёт про встроенные таймеры. Чтобы реализация задержки (допустим, для того, чтобы один раз в секунду моргать светодиодом) не останавливала выполнение других инструкций основной программы, существует система прерываний.

Прерывания можно условно поделить на внешние и внутренние. Внешние прерывания обычно так или иначе являются результатом изменения напряжения на контактах портов, которые настроены, как входные. Этот случай будет разобран в следующей ЛР. Сейчас же рассмотрим работу

системного таймера, то, как он генерирует прерывание, и как его правильно обрабатывать.

Регистры управления обычными таймерами показаны на рис __: TIMx_SMCR отвечает за привязку таймера к внешнему источнику. TIMx_SR за хранение текущего статуса таймеров, а TIMx_CR1 и TIMx_CR2 позволяют задавать настройки таймеров (такие как значение счётчика, направление счёта шаг, прерывания и т. д.)

Table 82. TIM1&TIM8 register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																	
0x00	TIMx_CR1	Reserved																						CKD [1:0]		ARPE		CMS [1:0]		DIR		OPM		URS		UDIS		CEN												
	Reset value																							0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	TIMx_CR2	Reserved																		OIS4		OIS3N		OIS3		OIS2N		OIS2		OIS1N		OIS1		THS		MMS[2:0]		CCDS		CCUS		Reserved		CCPC						
	Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x08	TIMx_SMCR	Reserved																ETP		ECE		ETPS [1:0]		ETF[3:0]				MSM		TS[2:0]				Reserved		SMS[2:0]														
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
0x0C	TIMx_DIER	Reserved																TDE		COMDE		CC4DE		CC3DE		CC2DE		CC1DE		UDE		BIE		TIE		COMIE		CC4IE		CC3IE		CC2IE		CC1IE		UIE				
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x10	TIMx_SR	Reserved																		CC4OF		CC3OF		CC2OF		CC1OF		Reserved		BIF		TIF		COMIF		CC4IF		CC3IF		CC2IF		CC1IF		UIF						
	Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	TIMx_EGR	Reserved																						BG		TG		COM		CC4G		CC3G		CC2G		CC1G		UG												
	Reset value																							0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0									

Рис 20 -регистры для управления таймерами и их состояние после перезагрузки МК.

А мы будем использовать именно системный таймер SysTick: И с его помощью определим пару задержек, которые будут отслеживать пройденное время, считая количество прерываний от таймера. Частота периодических прерываний получается равной 4Гц и 1кГц соответственно (рис 21).

```

25
26 void tim_4Hz_delay(){
27     // ----- SysTick CONFIG -----
28     if (SysTick_Config(TICKS_BEFORE_CYCLIC_INTERRUPT)) //
29     {
30         while(1); // error
31     }
32     four_hz_timer = 1;
33     while(four_hz_timer) {
34         asm("wfi");
35     };
36     SysTick -> LOAD &= ~(SysTick_CTRL_ENABLE_Msk);    // disable SysTick
37 }
38
39 void delay_ms(__IO uint32_t val) {
40     // ----- SysTick CONFIG -----
41     if (SysTick_Config(TICKS_IN_MS)) //
42     {
43         while(1); // error
44     }
45     ms_timer = val;
46     while(ms_timer) {
47         asm("wfi");
48     };
49     SysTick -> LOAD &= ~(SysTick_CTRL_ENABLE_Msk);    // disable SysTick
50 }
51
69
70 void SysTick_Handler(void)
71 {
72     {
73         if (four_hz_timer)
74             four_hz_timer--;
75         if (ms_timer)
76             ms_timer--;
77         if (us_timer)
78             us_timer--;
79     }
80

```

Рис 21 - таймеры, генерирующие прерывания с разным периодом в основе функций-задержек.

Для упрощения задачи в данном конкретном случае используем функции из библиотеки для работы с ядром C-M3.

Опишем алгоритм на примере delay_ms():

--- Функция принимает значение задержки в миллисекундах . Тип `_IO` (volatile) нужен, чтобы компилятор случайно не «выоптимизировал» переменную из программы из-за наличия прерываний.

--- Вызов `SysTick_Config` проверяет готовность таймера и одновременно задаёт значение тактов, которое он должен посчитать

--- переменной `ms_timer` присваивается значение `мс`, которые нужно отсчитать.

--- если таймер обнулится, функция уходит ждать прерывания

--- Прерывание происходит по прошествии одной мс, и системой вызывается обработчик прерываний от системного таймера `SysTick_Handler()`;

--- если какая либо из задержек активна, обработчик уменьшает счётчик циклов таймера на 1.

--- Так продолжается, пока счётчик `ms_timer` не обнулиться, и не выполнится последняя строка в `delay_ms()`, отключающая таймер.

В результате зелёный диод (рис 25 стр. 23) моргает с заданной частотой. Виной этому вызов задержки, использующей прерывания системного таймера (рис 22)

```
232     char string_for_ic[4];
233     uint16_t itrpts_counter = 0;
234     for(;;)
235     {
236         // tim_4Hz_delay(); // роли в ошибке не играет
237         delay_ms(250);
238         itrpts_counter = itrpts_counter + 1;
239         GPIOC -> ODR ^= GPIO_Pin_13; // led toggle
240
241         lcd_msg(2,12, itoa(itrpts_counter, string_for_ic, 10)); // PAR/
```

Рис 22 -регистры для управления таймерами и их состояние после перезагрузки МК.

На практике более правильным было бы поместить строку с переключением диода непосредственно в обработчик прерываний. А для разных периодов прерываний использовать разные таймеры.

ЛР№3: Прерывание от кнопки

В прошлой лабораторной работе были рассмотрены прерывания от системного таймера. Теперь перейдем к настройке внешних прерываний (EXTI).

Контроллер внешних прерываний показан на рисунке 23

Figure 20. External interrupt/event controller block diagram

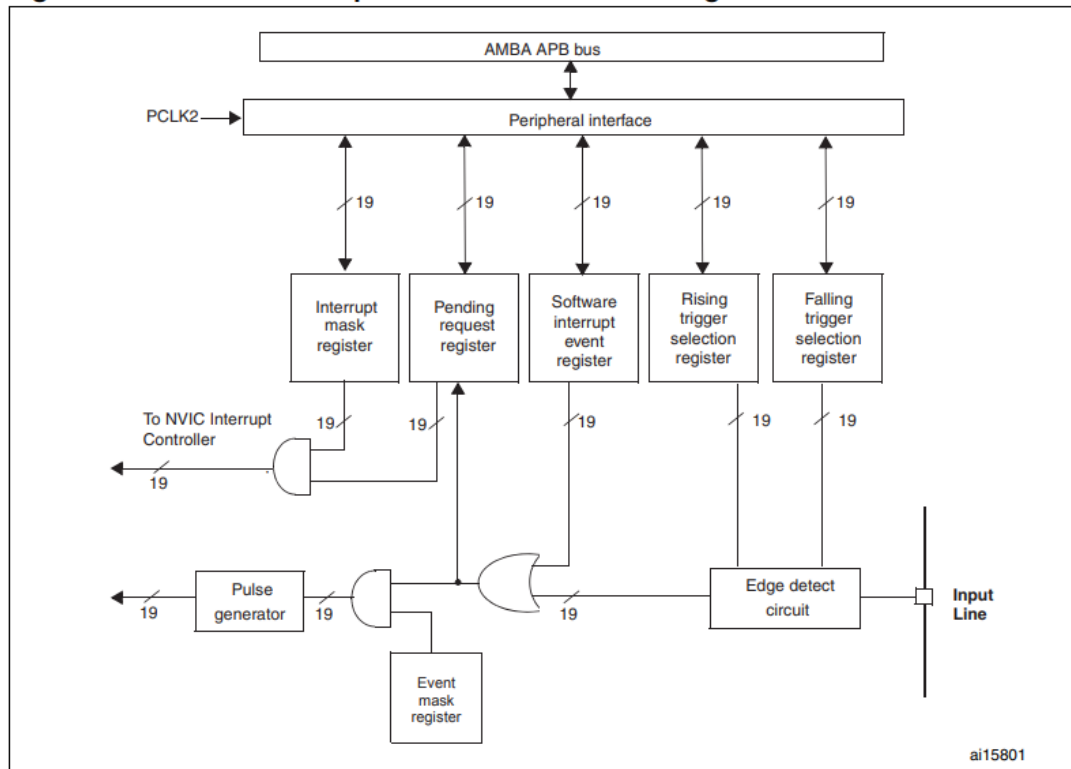


Рис 23 - блок-схема контроллера внешних прерываний .

В простейшем случае источником внешних прерываний может служить кнопка: А лучше две. Каждая на свой порт:

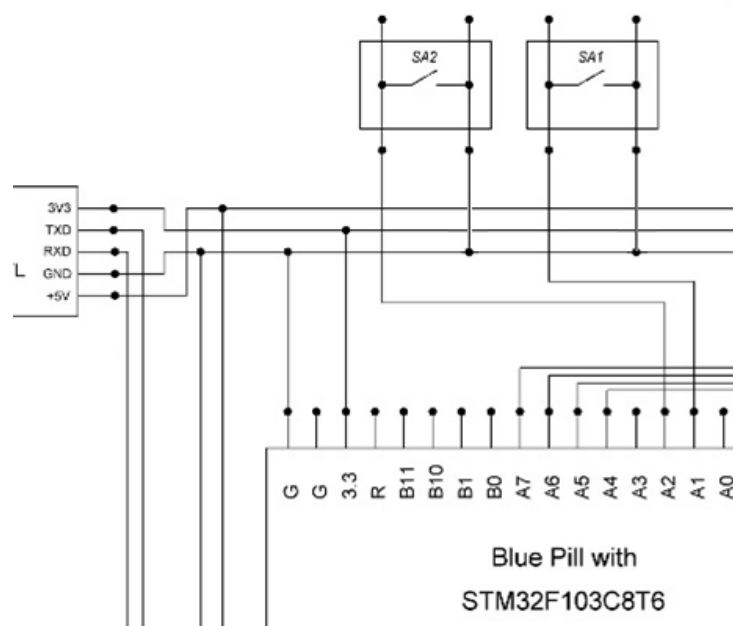


Рис 24 -подключение кнопок.

Кнопки подключены между контактами МК и землёй.

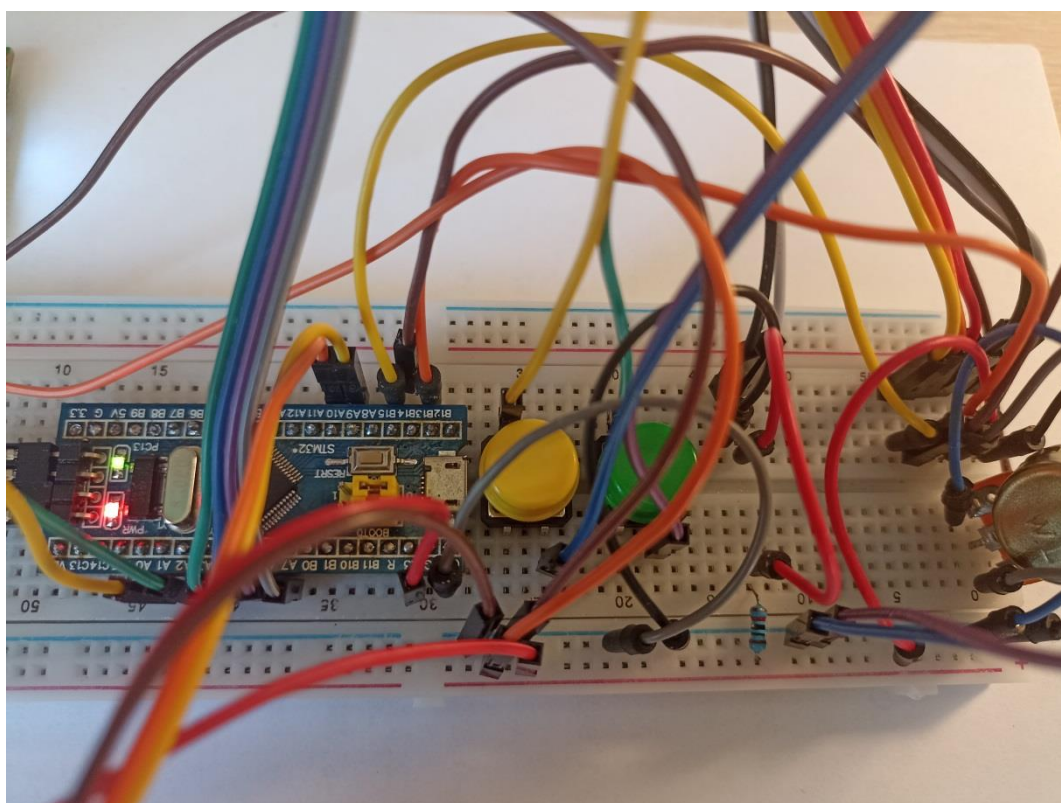


Рис 25 -подключение кнопок между землёй и вводом МК.

Идея следующая: если ножки 1 и 2 порта А подтянуть к напряжению питания (Pull-up mode) и настроить прерывания по падению напряжения на них, то при нажатии кнопки прерывание на соответствующем контроллере будет срабатывать и запускать обработчик.

Важно помнить, что настроить независимые прерывания от PA1 и, например, PB1 нельзя. Это принципиально невозможно из-за устройства контроллера EXTI:

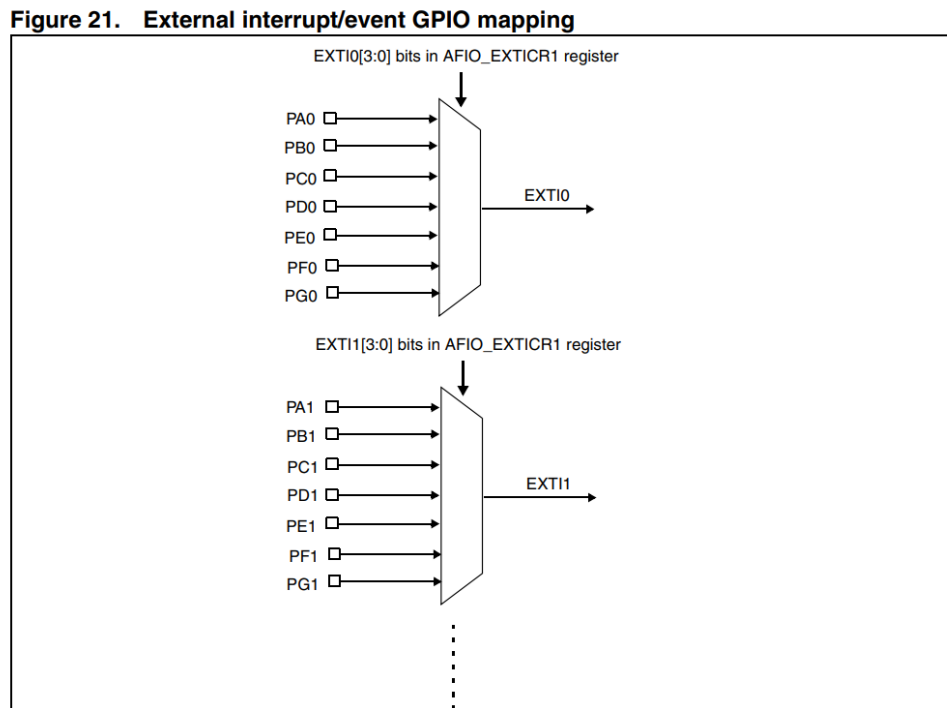


Рис 26 - эскиз контроллера внешних прерываний.

Прерывание по кнопке можно настроить следующим образом (рис 27, 28).

Здесь можно увидеть, как были затактированы нужные порты, включён альтернативный режим для этих портов, выставлен необходимый режим для них для двух контактов и разрешены внешние прерывания по спаду напряжения:

```

void buttons_intrpts_init()
{
    // --- BUTTONS setup ----
    /*
        PA1_pulled_up - button 1 to gnd
        PA2_pulled_up - button 2 to gnd
    */
    // Надо AFIO затактировать. Иначе будет трудно уловимый глюк!
    RCC -> APB2ENR |= (
        RCC_APB2ENR_IOPAEN | RCC_APB2ENR_IOPBEN | RCC_APB2ENR_AFIOEN
    );

    // Выставляем режим порта в CNF для битов 1,2 Режим 10 = PullUp(Down)
    //Ставим первый бит CRL
    GPIOA -> CRL    &= ~(GPIO_CRL_CNF1 | GPIO_CRL_CNF2);    // Занулим заранее
    GPIOA -> CRL    |= GPIO_CRL_CNF1_1 | GPIO_CRL_CNF2_1;    // Выставим бит 1

    // Выставляем режим для 1,2 пина. Режим MODE_00 = Input
    GPIOA -> CRL    &= ~(GPIO_CRL_MODE1 | GPIO_CRL_MODE2);

    // Установили бит 1,2 в ODR включив PULL UP.
    GPIOA -> BSRR = GPIO_BSRR_BS2 | GPIO_BSRR_BS1;

    /*
        Настраиваем EXTI1 и EXTI2 на выводы порта A. Они в первом регистре оба.
        А регистры в виде массива описаны в CMSIS
        Регистры EXTICR сгруппированы в массив, а массив считается с нуля!
        Т.е. первый EXTICR регистр в 0 ячейке
        второй регистр в первой, третий во 2
    */
    AFIO -> EXTICR[0] |= AFIO_EXTICR1_EXTI1_PA | AFIO_EXTICR1_EXTI2_PA;

    // Прерывание по падению уровня на пинах 1 и 2 порта привязанного к EXTI
    EXTI -> FTSR |= (EXTI_FTSR_TR1 | EXTI_FTSR_TR2);

    NVIC_EnableIRQ(EXTI1_IRQn);
    NVIC_EnableIRQ(EXTI2_IRQn);
    NVIC_SetPriority(EXTI2_IRQn, 2);

    // Разрешаем прерывания в периферии
    EXTI -> IMR |= (EXTI_IMR_MR1 | EXTI_IMR_MR2);
}

```

Рис 27 - настройка внешних прерываний от кнопки .

Далее представлены обработчики прерываний:

```

void EXTI1_IRQHandler(void) // Обработчик EXTI 1
{
    dummy_led_dim(7);
    // Сбрасываем флаг прерывания единицей
    EXTI -> PR |= EXTI_PR_PR1;
}

void EXTI2_IRQHandler(void) // Обработчик EXTI 2
{
    dummy_led_dim(13);
    // Сбрасываем флаг прерывания
    EXTI -> PR |= EXTI_PR_PR2;
}

```

Рис 28 - обработчики прерываний от кнопок. Для каждой кнопки – свой.

Задача о обработчиков несложная – моргнуть диодом несколько раз. Количество вспышек зависит от того, какая кнопка была нажата. Иногда по нажатию одной кнопки срабатывают оба прерывания (судя по тому, что количество вспышек суммируется). Это можно исправить, устранив взаимное влияние кнопок. Скорее всего хватит небольшого электролитического конденсатора, который сгладит выбросы напряжения на шину питания.

Видео работы стенда, где видно и результат срабатывания прерываний есть в файловом архиве.

ЛР№4: Индикация

Существуют множество разных типов устройств вывода информации от ЭВМ. Для графической и текстовой информации могут быть использованы и дорогостоящие мониторы с высокой плотностью пикселей, частотой обновления и точной кодировкой любого цвета, и простые монохромные дисплеи. Выбор устройства зависит от решаемой задачи. От требований к информации, которую нужно отображать.

Простейшая индикация в электронных устройствах – это мигание встроенным светодиодом, которое было описано в ЛР№1.В. В данной же ЛР текстовая информация от МК будет выводиться на символьный дисплей. Именно на символьный (рис 29).



Рис 29 - ЖК-дисплей типа 1602А.

Контроллеры таких устройств часто принципиально не рассчитаны на переключение состояния пикселей в произвольном порядке. Вместо этого на экране расположены фиксированные позиции для отображения символов. Шрифт также обычно определен производителем заранее.

Таким образом, монохромный символьный ЖК-дисплей типа 1602А имеет 2 строки по 16 символов в каждой. Каждому отдельному символу соответствует уникальный двоичный код (рис 30).

NO.7066-0A

b7-b4 b3-b0	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	CG RAM (1)			0	1	2	3	4	5	6	7	8	9	A	B	C
0001	(2)		!	@	A	B	C	D	E	F	G	H	I	J	K	L
0010	(3)		"	#	\$	%	&	'	()	*	+	=	<	>	?
0011	(4)		#	3	0	5	c	s			1	7	T	E	S	o
0100	(5)		\$	4	D	T	a	t			\	I	K	P	U	a

Рис 30 - устройство таблицы символов в памяти контроллера дисплея.

Данная модель имеет параллельный 8-ми или 4-ёх битный интерфейс (рис32), в отличие от аналогичных моделей с последовательным интерфейсом I²C

.Битность выбирается программным образом. И вообще, МК должен «общаться» с дисплеем посредством команд (рис 31).

Instruction Table:

Instruction	Instruction Code										Description	Description Time (270KHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRAM, and set DDRAM address to "00H" from AC	1.52 ms
Return Home	0	0	0	0	0	0	0	0	1	x	Set DDRAM address to "00H" from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.	1.52 ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 us
Display ON/OFF	0	0	0	0	0	0	1	D	C	B	D=1:entire display on C=1:cursor on B=1:cursor position on	37 us
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	x	x	Set cursor moving and display shift control bit, and the direction, without changing DDRAM data.	37 us

Рис 31 - часть списка команд дисплея .

Например, для выполнения команды на удаление текста с экрана по линия DB0 должна иметь высокий логический уровень. И обрабатывается около 1.5 мс. Минимальные временные задержки на обработку команд всегда должны выдерживаться программой МК, чтобы текст отображался правильно.

Два контроллера дисплея, конструктивно расположенные на его обратной стороне имеют множество соединений, но для разработчика доступна только малая часть. На плату выведены следующие контакты (рис 32):

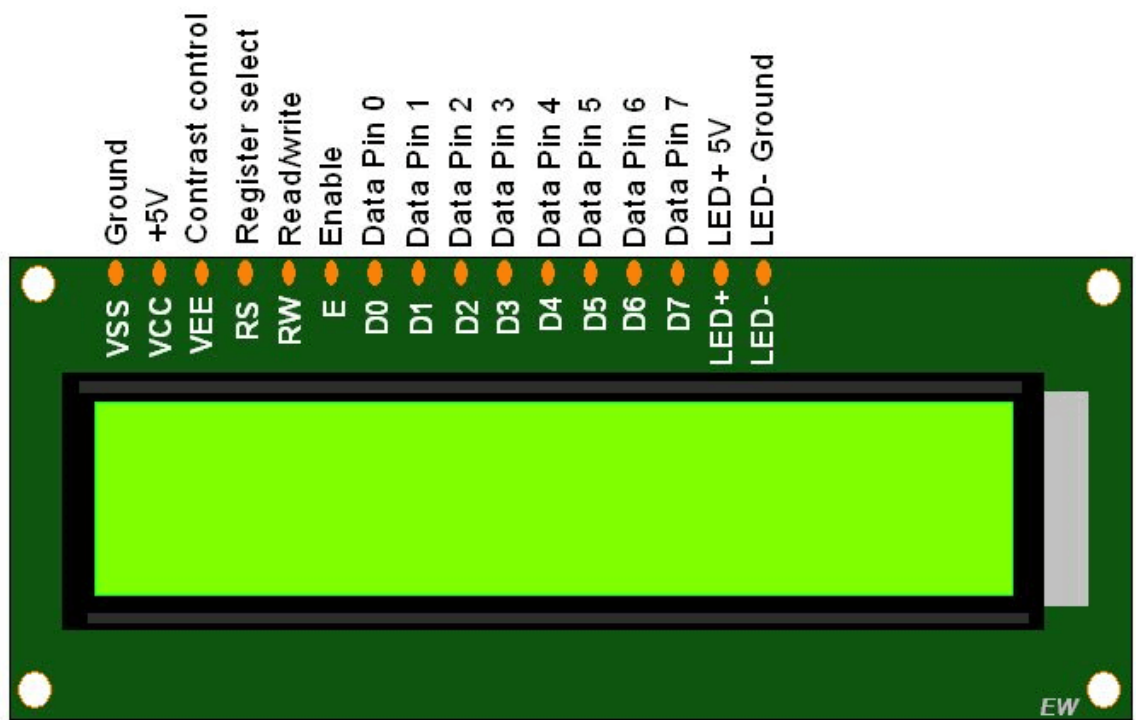


Рис 32 -назначение выводов дисплея .

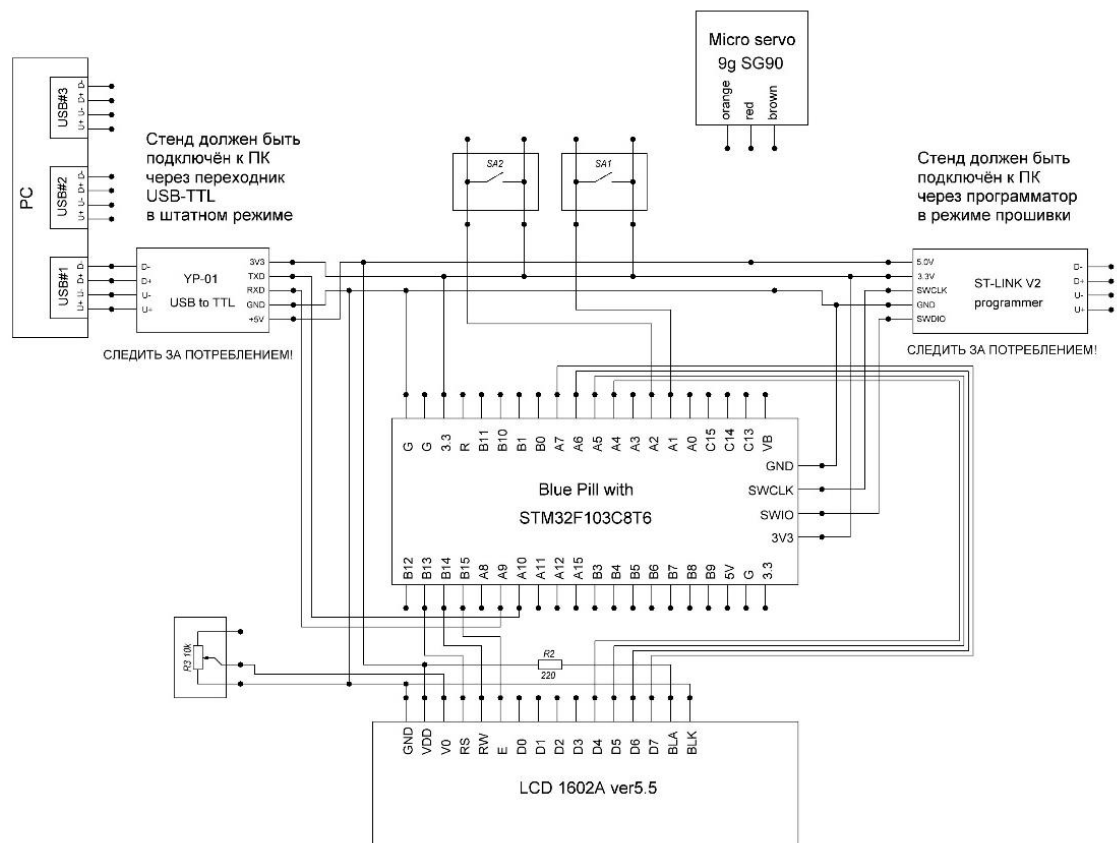


Рис 33 - принципиальная схема лабораторного стенда .

--- LED +, LED- (или BLA, BLK, как на нашей плате) – нужны для питания фоновой подсветки экрана. Эта схема изолирована от остальных элементов.

--- VSS, VCC – пины питания цифровой сигнальной части. Т. е. контроллеров дисплея, их обвязки и самого дисплея. Напряжение питания - 5В (пин VCC). Оно отличается от рабочего напряжения STM32 (3,3В), поэтому в схеме лабораторного стенда организован отдельная 5-Вольтовая линия

--- VEE (V0) – вход для регулирования контрастности (аналоговый)

--- RS –выбор между разными списками команд)

--- RW – переключение между чтением и записью

--- E – для сигнала синхронизации (дисплей настроен на передний фронт синхроимпульса)

--- D0, D1 ... D7 – шина параллельной передачи данных. Также используется и для отправки команд.

Выбор наиболее простого и популярного дисплея снова даёт возможность пользоваться библиотеками с открытым исходным кодом. В сети нетрудно найти код даже под связку этих конкретных моделей экрана и МК. Однако, этот код нельзя сразу использовать в проекте. Как минимум из-за того, что нужно редактировать конфигурацию портов МК под схему стенда, использовать другие функции-таймеры и изменять идентификаторы для повышения читаемости и производительности кода.

Переработанный код частично представлен на листингах (рис 34 - 36).

Для начала хочется обернуть задание логического уровня для управляющих линий:

```

7 void lcd_rs_high(){
8     GPIOB -> BSRR = GPIO_Pin_13; // поставили 1
9 }
10
11 void lcd_rs_low(){
12     GPIOB -> BRR = GPIO_Pin_13; // поставили 0
13 }
14
15 void lcd_rw_high(){
16     GPIOB -> BSRR = GPIO_Pin_14;
17 }
18
19 void lcd_rw_low(){
20     GPIOB -> BRR = GPIO_Pin_14;
21 }
22
23 void lcd_e_high(){
24     GPIOB -> BSRR = GPIO_Pin_15;
25 }
26
27 void lcd_e_low(){
28     GPIOB -> BRR = GPIO_Pin_15;
29 }
30

```

Рис 34 - конфигурация управляющих линий .

Затем определить управление шиной данных:

```

void send_bits_type_1(unsigned char data){
    GPIOA -> ODR &= 0xff0f;
    GPIOA -> ODR |= (data & 0x00f0);
}

void send_bits_type_2(unsigned char data){
    GPIOA -> ODR &= 0xff0f;
    GPIOA -> ODR |= ((data << 4) & 0x00f0);
}

void send_in_init_type_1(){
    GPIOA -> ODR &= 0xff0f;
    GPIOA -> ODR |= 0x30; // 8 bit communication mode
}

void send_in_init_type_2(){
    GPIOA -> ODR &= 0xff0f;
    GPIOA -> ODR |= 0x20; // 4 bit communication mode
}

```

Рис 35 -функции для передачи данных через параллельный интерфейс .

А затем в соответствии с инструкцией к дисплею собрать функции передачи данных и команд из задержек и переключателей пинов:

```

HEAD > src > lcd.c
56 void lcd_data(unsigned char data)
57 {
58     //pin_output(4,7); // set it up in main
59
60     lcd_rs_high();
61     lcd_rw_low();
62     delay_ms(1); // 0.01 should be!!
63     lcd_e_high();
64     delay_ms(1); // 0.005 should be!!
65     send_bits_type_1(data);
66     delay_ms(1); // 0.01 should be!!
67     lcd_e_low();
68
69     delay_ms(1); // 0.02 should be!!
70
71     lcd_e_high();
72     delay_ms(1); // 0.005 should be!!
73     send_bits_type_2(data);
74     delay_ms(1); // 0.01 should be!!
75     lcd_e_low();
76 }
77
78
79 void lcd_cmd(unsigned char data)
80 {
81     // pin_output(0,11); // set it up in main
82     lcd_rs_low();
83     lcd_rw_low();
84     delay_ms(1); // 0.01 should be!!
85     lcd_e_high();
86     delay_ms(1); // 0.005 should be!!
87     send_bits_type_1(data);
88     delay_ms(1); // 0.01 should be!!
89     lcd_e_low();
90
91     delay_ms(1); // 0.02 should be!!
92
93 }

```

Рис 36 - формирование команд управления экраном.

На всякий случай интервалы между посылками были увеличены, чтобы отладка дисплея прошла быстрее и кол-во непредвиденных ошибок было снижено. А оптимизировать скорость передачи можно потом. Целиком модуль lcd.c расположен в папке src в архиве, прикреплённом к отчёту.

А для демонстрации связи дисплея с МК, он должен быть проинициализирован в main.c и принять команды на отображение текста (рис 37, 38):

```

158 char string_for_f_cpu[2];
159 char string_for_baudrate[6];
160
161 void lcd_demo()
162 {
163     // --- LCD setup ---
164     /*
165     Pin setup : 4 bits data communication
166
167     PA8  -> RS
168     PB10 -> RW
169     PB11 -> E
170
171     PA4  -> DB4
172     PA5  -> DB5
173     PA6  -> DB6
174     PA7  -> DB7
175     */
176     //Digital_Input(PA,11); // Initialize push button
177
178     lcd_init(); // Initialize the LCD screen to work with 4 bits data interface
179     // lcd_msg(Line_number 1 or 2, Position within the line, String to be shown)
180     lcd_msg(1, 1, "Rahimov");
181     lcd_msg(2, 6, "RL1-104");
182     delay_ms(1000);
183     lcd_msg(1,0, "Bench 13 var:  ");
184     delay_ms(1000);
185     lcd_msg(2,0, "Ver ____:  "); // PARAMETRIZE!
186

```

Рис 37 - конфигурация портов дисплея и вызов его инициализации .

```

186     lcd_msg(2,0, "Ver ____:  "); // PARAMETRIZE!
187     delay_ms(1000);
188     lcd_msg(2,4, "0.5.2");
189     delay_ms(3000);
190     lcd_msg(1,0, "CPU_FREQ = __MHz");
191     delay_ms(1000);
192     lcd_msg(1,11, itoa(F_CPU_MHZ, string_for_f_cpu, 10));
193     delay_ms(1000);
194     lcd_msg(2,0, "INTRP_FREQ = 4Hz");
195     delay_ms(3000);
196     lcd_msg(1,0, "BAUDRATE =");
197     delay_ms(1000);
198     lcd_msg(1,10, itoa(BAUDRATE, string_for_baudrate, 10));
199     delay_ms(1000);
200     lcd_msg(2,0, "Interrupts: ");
201     delay_ms(1000);
202     lcd_msg(2,12, "0  ");
203     delay_ms(1000);
204 }
205

```

Рис 38 - вывод информации о параметрах МК на дисплей .

Как видно из рис 39, параметры МК, отображаемые на экране параметризованы. Функция `itoa` здесь отвечает за преобразование чисел в строки



Рис 39 - отображение текста на дисплее..

Изменяя аргументы функций-задержек в модуле `lcd.c` можно добиться постепенного появления текста на экране (один символ за другим). Либо организовать движение текста с заданной скоростью.

Домашнее задание

Общие сведения:

Для выполнения ДЗ используется тот же самый стенд с STM32 во главе. Программа для МК разбита на следующие модули:

--- `init.s` для загрузки таблицы обработчиков прерываний в память микроконтроллера

--- `wheel.c` включает вспомогательные функции, такие как настройка таймеров, задание прерываний по ним, разные виды индикации встроенными светодиодами, функции для преобразования данных из одного типа в другой и пр.

--- В модуле `startup.c` вводятся функции, отвечающие за тактировку процессора, шин данных, периферии и их конфигурацию. Эти функции вызываются один раз – в момент подачи питания на МК. Или в момент его перезагрузки.

```

190 // ----- SYSTEM INIT (called at reset) -----
191 void SystemInit(void)
192 {
193     RCC->CR |= (uint32_t)0x00000001; // enable HSI, 8 MHz
194
195     /* Reset SW, HPRE, PPRE1, PPRE2, ADCPRE and MCO bits */
196 #ifndef STM32F10X_CL
197     RCC->CFGR &= (uint32_t)0xF8FF0000;
198 #else
199     RCC->CFGR &= (uint32_t)0xF0FF0000;
200 #endif /* STM32F10X_CL */
201
202     /* Reset HSEON, CSSON and PLLON bits */
203     RCC->CR &= (uint32_t)0xFEFFFFFF;
204
205     /* Reset HSEBYP bit */
206     RCC->CR &= (uint32_t)0xFFFFBFFFF;
207
208     /* Reset PLLSRC, PLLXTPRE, PLLMUL and USBPRE/OTGFSPRE bits */
209     RCC->CFGR &= (uint32_t)0xFF80FFFF;
210
211 #ifdef STM32F10X_CL
212     /* Reset PLL2ON and PLL3ON bits */
213     RCC->CR &= (uint32_t)0xEBFFFFFF;
214
215     /* Disable all interrupts and clear pending bits */
216     RCC->CIR = 0x00FF0000;
217
218     /* Reset CFGR2 register */
219     RCC->CFGR2 = 0x00000000;
220 #elif defined (STM32F10X_LD_VL) || defined (STM32F10X_MD_VL) || (defined STM32F10X_HD_VL)
221     /* Disable all interrupts and clear pending bits */
222     RCC->CIR = 0x009F0000;
223
224     /* Reset CFGR2 register */
225     RCC->CFGR2 = 0x00000000;
226 }

```

Рис 40 - начало функции инициализации МК .

--- в модуле uart.c, определяет работу с интерфейсом UART (универсальный асинхронный приёмопередатчик)


```

wheel.c startup.c uart.c x lcd.c
_HEAD > src > C uart.c
1  #include "uart.h"
2  #include "wheel.h"
3  #define len 8
4
5
6  // ----- LIB FUNCS -----
7  void usart1_print_number(uint32_t num)
8  {
9      uint8_t n[len];
10     char *s=n+(len-1);
11     *s=0;          // EOL
12     do {
13         * (--s) = (uint32_t)(num%10 + 0x30);
14         num=num/10;
15     } while (num > 0);
16     usart1_print_string(s);
17 }
18
19
20 void usart1_send_char(uint32_t ch)
21 {
22     USART1->DR=ch;
23     while(!(USART1->SR & USART_FLAG_TXE));
24 }
25
26
27 void usart1_print_string(char *str)
28 {
29     while (*str)
30     {
31         usart1_send_char((uint32_t)*str++);
32     }
33 }
34

```

Рис 41 - функции передачи данных по UART .

--- lcd.c, - модуль для отправки текста на ЖК-дисплей, и контроля за ним:

```

_HEAD > src > C lcd.c
147
148 void lcd_msg(unsigned char line_1_2, unsigned char pos_0_16, char msg[])
149 {
150
151     // led_dim(3);
152
153     short pos = 0;
154     if(line_1_2==1)
155     {
156         pos = 0;
157     }
158     else if(line_1_2==2)
159     {
160         pos = 0x40;
161     }
162     lcd_cmd(0x80 +pos + pos_0_16);
163     delay_ms(1); // NEEDS 0.1
164     lcd_send(msg);
165
166     // led_dim(3);
167
168 }
169

```

Рис 42 - функция передачи данных на ЖК-дисплей.

В main.c –импортируются все остальные модули для их использования при инициализации контроллера и в главном цикле.

```
_HEAD > C main.c
207 // ----- MAIN -----
208 int main()
209 {
210     // system_init called from startup directly
211     // но для красоты можно ещё раз вызвать здесь
212     SystemInit();
213
214     SetSysClockToReqFreq(F_CPU_MHZ); // п.1 ДЗ
215
216     // п.2 ДЗ вынесен в wheel.c
217
218     gpio_init();
219
220     uart_init();
221
222     buttons_intrpts_init();
223
224     __enable_irq(); // NEED TESTING!
225
226     led_dim(7);
227
228     lcd_demo(); // п.4 ДЗ
229
230     led_dim(3);
231 }
```

Рис 43 - инициализация контроллера и периферии в модуле main.c.

```

232 char string_for_ic[4];
233 uint16_t itrpts_counter = 0;
234 for(;;)
235 {
236     // tim_4Hz_delay(); // роли в ошибке не играет
237     delay_ms(250);
238     itrpts_counter = itrpts_counter + 1;
239     GPIOC -> ODR ^= GPIO_Pin_13; // led toggle
240
241     lcd_msg(2,12, itoa(itrpts_counter, string_for_ic, 10)); // PARA
242     usart1_send_string_of_30_chars( // n.3 д3
243         // string transmission fails
244         'C','u','r','r','e','n','t',
245         ' ','F','_', 'C','P','U',
246         '=', '4','8',
247         'M','H','z',
248         ' ','-','-','-','-','-','- ',
249         '-','-','-','- ', 0x0000000D
250     );
251     usart1_send_string_of_30_chars( // n.3 д3
252         // string transmission fails
253         'D','e','b','u','g',' ','V',
254         '0','.', '5','.', 'x',' ',
255         0x000000F0, 0x000000E0, 0x000000E1, 0x000000EE,
256         0x000000F2, 0x000000E0, 0x000000E5, 0x000000F2,
257         ' ','-','-','-','- ',
258         '-','-','-','- ', 0x0000000D
259     );
260 }
261 }
262

```

Рис 44 - вечный цикл в модуле main.c .

Также были созданы соответствующие заголовочные файлы.

Возможные параметры задания приведены здесь:

Частоты тактирования от HSE (МГц): 8, 16, 24, 32, 40, 48, 56, 64, 72, 80

Периоды прерываний (в секундах). Да, пусть это только периоды, а не частоты): 0.25, 0.5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

Технические скорости UART (кбит/с): 9.6, 19.2, 38.4, 57.6, 115.2.

Необходимо добиться того, чтобы проект работал для любой комбинации этих трёх параметров ТЗ. Для демонстрация работы и отладки будут использованы значения 13го варианта:

вар = 13	F_CPU = 48 МГц	F_INTRPT = 4 Гц	BAUD = 38.4 кбит/с
----------	----------------	-----------------	--------------------

П1: Тактировка процессора (пересекается с ЛР№1.В)

Написать функцию, настраивающую частоту микроконтроллера от внешнего кварца 8 МГц (источник сигнала HSE) на заданную частоту. В качестве параметра функции передается желаемое значение частоты.

Для работы с регистрами нужно постоянно обращаться к руководству по МК серии STM32F103x. Основной информацией является описание структуры регистров, которые отвечают за управление процессором и периферией. Для настройки частоты процессора необходимо будет производить записи в регистры группы RCC:

Table 16. RCC register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0										
0x000	RCC_CR	Reserved						PLL RDY	PLL ON	Reserved						CSSON	HSEBYP	HSERDY	HSEON	HSICAL[7:0]						HSITRIM[4:0]						Reserved	HSIRDY	HSION									
	Reset value							0	0							0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1							
0x004	RCC_CFGR	Reserved						MCO [2:0]		Reserved	USBPRE	PLLMUL[3:0]				PLLXTPRE	PLLSRC	ADC PRE [1:0]	PPRE2 [2:0]		PPRE1 [2:0]		HPRE[3:0]			SWS [1:0]		SW [1:0]															
	Reset value							0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0									
0x008	RCC_CIR	Reserved									CSSC	Reserved	PLL RDY	HSERDY	HSIRDY	LSERDY	LSIRDY	Reserved			PLL RDY	HSERDY	HSIRDY	LSERDY	LSIRDY	CSSF	Reserved		PLL RDY	HSERDY	HSIRDY	LSERDY	LSIRDY										
	Reset value										0	0	0	0	0	0	0	0				0	0	0	0	0	0			0	0	0	0	0									
0x00C	RCC_APB2RSTR	Reserved										TIM11RST	TIM10RST	TIM9RST	Reserved			ADC3RST	USART1RST	TIM8RST	SPI1RST	TIM1RST	ADC2RST	ADC1RST	IOPGRST	IOPFRST	IOPERST	IOPDRST	IOPCRST	IOPBRST	IOPARST	Reserved	AFIORST										
	Reset value											0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0								
0x010	RCC_APB1RSTR	Reserved	DACRST	PWRRST	BKPRST	Reserved	CANRST	Reserved	USBRST	I2C2RST	I2C1RST	UART5RST	UART4RST	USART3RST	USART2RST	Reserved	SPI3RST	SPI2RST	Reserved		WWDGRST	Reserved		TIM14RST	TIM13RST	TIM12RST	TIM7RST	TIM6RST	TIM5RST	TIM4RST	TIM3RST	TIM2RST											
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			0			0	0	0	0	0	0	0	0	0	0										
0x014	RCC_AHBENR	Reserved																				SDIOEN	Reserved	FSMCEN	Reserved	CRCEN	Reserved	FLITFEN	Reserved	SRAMEN	DM2AEN	DM1AEN											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0											
0x018	RCC_APB2ENR	Reserved										TIM11EN	TIM10EN	TIM9EN	Reserved			ADC3EN	USART1EN	TIM8EN	SPI1EN	TIM1EN	ADC2EN	ADC1EN	IOPGEN	IOPFEN	IOPDEN	IOPCEN	IOPBEN	IOPAEN	Reserved	AFIOEN											
	Reset value											0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0										
0x01C	RCC_APB1ENR	Reserved	DACEN	PWREN	BKPEN	Reserved	CANEN	Reserved	USBEN	I2C2EN	I2C1EN	UART5EN	UART4EN	USART3EN	USART2EN	Reserved	SPI3EN	SPI2EN	Reserved		WWDGEN	Reserved		TIM14EN	TIM13EN	TIM12EN	TIM7EN	TIM6EN	TIM5EN	TIM4EN	TIM3EN	TIM2EN											
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			0			0	0	0	0	0	0	0	0	0	0										
0x020	RCC_BDCR	Reserved															BDRST	RTCEN	Reserved					RTC SEL [1:0]	Reserved					LSEBYP	LSERDY	LSEON											
	Reset value																0	0						0						0	0	0											
0x024	RCC_CSR	LPWRSTF	WWDGRSTF	IWDGRSTF	SFTRSTF	PORRSTF	PINRSTF	Reserved	RMVF	Reserved																						LSIRDY	LSION										
	Reset value	0	0	0	0	1	1	0	0																						0	0	0										

Рис 45 - регистры перезагрузки и тактирования. Их состояние по умолчанию .

Регистр настройки тактирования процессора показан на рис 46. В нём разработчик помимо прочего может задать:

- множитель частоты при тактировании от PLL
- предделитель частоты процессора
- источник тактирования для PLL
- предделитель тактирования АЦП
- предделитель тактирования шин для работы с периферией
- выбор источника тактирования процессора и статус этого выбора

6.3.2 Clock configuration register (RCC_CFGR)

Address offset: 0x04

Reset value: 0x0000 0000

Access: $0 \leq \text{wait state} \leq 2$, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					MCO[2:0]			Res.	USB PRE	PLLMUL[3:0]				PLL XTPRE	PLL SRC
					rw	rw	rw		rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC PRE[1:0]		PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

Рис 46 -области регистра настройки тактирования процессора.

Особый интерес представляют биты с 18 по 21. От их значения зависит множитель частоты PLL, который нужно будет рассчитывать в зависимости от требуемой частоты процессора МК. Важно отметить, что частота 80МГц превышает предельно допустимое значение.

Bits 21:18 **PLLMUL**: PLL multiplication factor

These bits are written by software to define the PLL multiplication factor. These bits can be written only when PLL is disabled.

Caution: The PLL output frequency must not exceed 72 MHz.

0000: PLL input clock x 2
 0001: PLL input clock x 3
 0010: PLL input clock x 4
 0011: PLL input clock x 5
 0100: PLL input clock x 6
 0101: PLL input clock x 7
 0110: PLL input clock x 8
 0111: PLL input clock x 9
 1000: PLL input clock x 10
 1001: PLL input clock x 11
 1010: PLL input clock x 12
 1011: PLL input clock x 13
 1100: PLL input clock x 14
 1101: PLL input clock x 15
 1110: PLL input clock x 16
 1111: PLL input clock x 16

Рис 47 - ключ для настройки множителя частоты PLL из руководства пользователя .

Настройка тактовой частоты процессора и расчёт вспомогательных параметров частично приведены на рис 48:

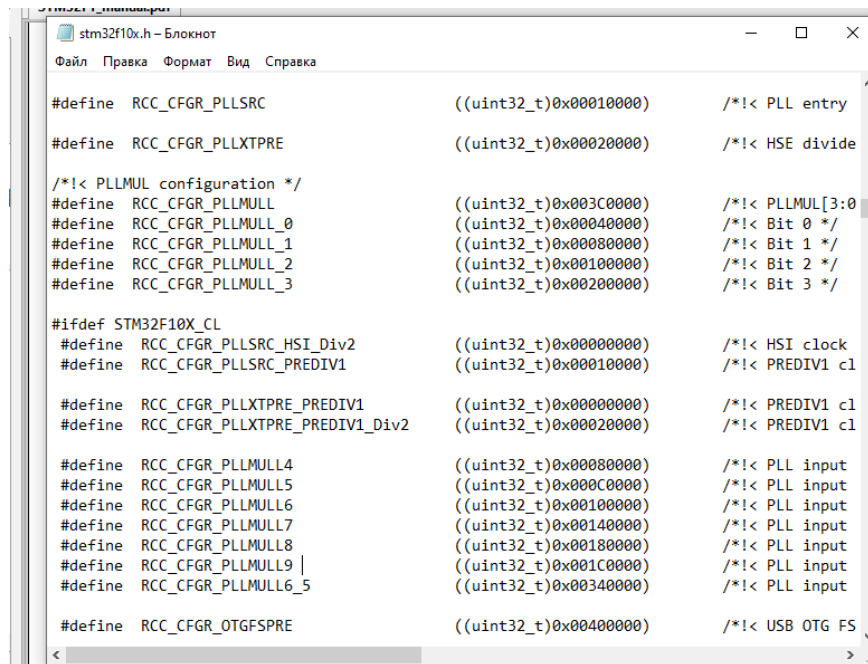


Рис 49 - пример списка именованных битовых масок для работы с регистрами «библиотеки» CMSIS .

П.2: Прерывание по таймеру (подробнее в ЛРН№2)

Написать функцию, настраивающую прерывание по таймеру с заданной частотой, в зависимости от настроенной частоты микроконтроллера. Обеспечить индикацию срабатывания прерывания с помощью светодиодов.

```

38
39 void delay_ms(__IO uint32_t val) {
40     // ----- SysTick CONFIG -----
41     if (SysTick_Config(TICKS_IN_MS)) //
42     {
43         while(1); // error
44     }
45     ms_timer = val;
46     while(ms_timer) {
47         asm("wfi");
48     };
49     SysTick -> LOAD &= ~(SysTick_CTRL_ENABLE_Msk); // disable SysTick
50 }
51

```

Рис 50 -функция задержки, использующая прерывания системного таймера.


```

70 void SysTick_Handler(void)
71 {
72     if (four_hz_timer)
73         four_hz_timer--;
74     if (ms_timer)
75         ms_timer--;
76     if (us_timer)
77         us_timer--;
78 }
79
80
81
82 // ----- LED FUNCS -----
83 void led_blink_once()
84 {
85     GPIOC -> BSRR = GPIO_BSRR_BS13; // 1
86     delay_ms(500);
87     GPIOC -> BSRR = GPIO_BSRR_BR13; // 0
88     delay_ms(500);
89 }
90

```

Рис 51 - обработчик прерываний системного таймера и индикация его срабатывания диодом.

На практике более правильным было бы поместить строку с переключением диода непосредственно в обработчик прерываний. А для разных периодов прерываний использовать разные таймеры.

П.3: Отладка через UART

Написать функцию, настраивающую универсальный приемопередатчик UART на заданную скорость передачи информации, в зависимости от настроенной частоты микроконтроллера. Обеспечить передачу на ПК по прерыванию из п. 2 данных о частоте микроконтроллера.

Заданная функция и результаты её работы показаны на рисунках 52 и 53:соответственно:

```

83 uint32_t calc_brr_bits(uint8_t F_CPU_MHZ, uint32_t BAUDRATE)
84 { // BRR_total_value = AHBxCLK / TrueBaudRate
85   // all AHB mults must be 1 for result to be correct
86   uint32_t f_cpu_in_hz = ((uint32_t)F_CPU_MHZ) * 1000000;
87   uint32_t brr_mantissa = f_cpu_in_hz / (BAUDRATE * 16);
88   uint32_t brr_fraction = 0; // округлим пока. боимся деления с плав. точкой
89   // APPROX INT DEVISION. NEEDS CHECKING!
90   return (brr_mantissa << 4) + brr_fraction; // mantissa and fraction
91 }
92
93
94 void uart_init()
95 {
96   // --- UART setup ----
97   /*
98   PA8 - CK1 (пока не нужен)
99   PA9 - TX1
100  PA10 - RX1
101  PA11 - CTS1 (пока не нужен)
102  PA12 - RTS1 (пока не нужен)
103  */
104  RCC -> APB2ENR |= RCC_APB2Periph_USART1;
105  GPIOA -> CRH &= ~(uint32_t)(0xf<<4);
106  // тут медленный альтернативный пуш-пулл для PA9
107  GPIOA -> CRH |= (uint32_t)(0xa<<4);
108  uint32_t brr_bits = calc_brr_bits(F_CPU_MHZ, BAUDRATE);
109  USART1 -> BRR = ((uint16_t)brr_bits); // BaudRate Register
110  // 0x271 --> 0000 0010 0111 0001 --> 115200 при F_cpu_MHz=72
111  USART1 -> CR1 |= (USART_CR1_UE_Set | USART_Mode_Tx);
112 }
113

```

Рис 52 - функция настройки UART .

Отдельной задачей стояла настройка технической скорости («бодрейта») в зависимости от частоты процессора. Функция `calc_brr_bits()` решает эту задачу даже не смотря на округление дробной части значения для регистра BRR.

Table 194. USART mode configuration⁽¹⁾

USART modes	USART1	USART2	USART3	UART4	UART5
Asynchronous mode	X	X	X	X	X
Hardware Flow Control	X	X	X	NA	NA
Multibuffer Communication (DMA)	X	X	X	X	NA
Multiprocessor Communication	X	X	X	X	X
Synchronous	X	X	X	NA	NA
Smartcard	X	X	X	NA	NA
Half-Duplex (Single-Wire mode)	X	X	X	X	X
IrDA	X	X	X	X	X
LIN	X	X	X	X	X

1. X = supported; NA = not applicable.

Рис 55 - режимы работы UART и их доступность на каждом отдельном «порту».

Figure 288. USART example of synchronous transmission

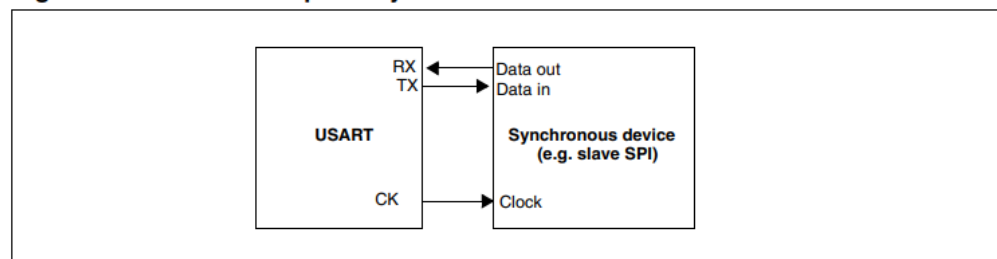


Figure 289. USART data clock timing diagram (M=0)

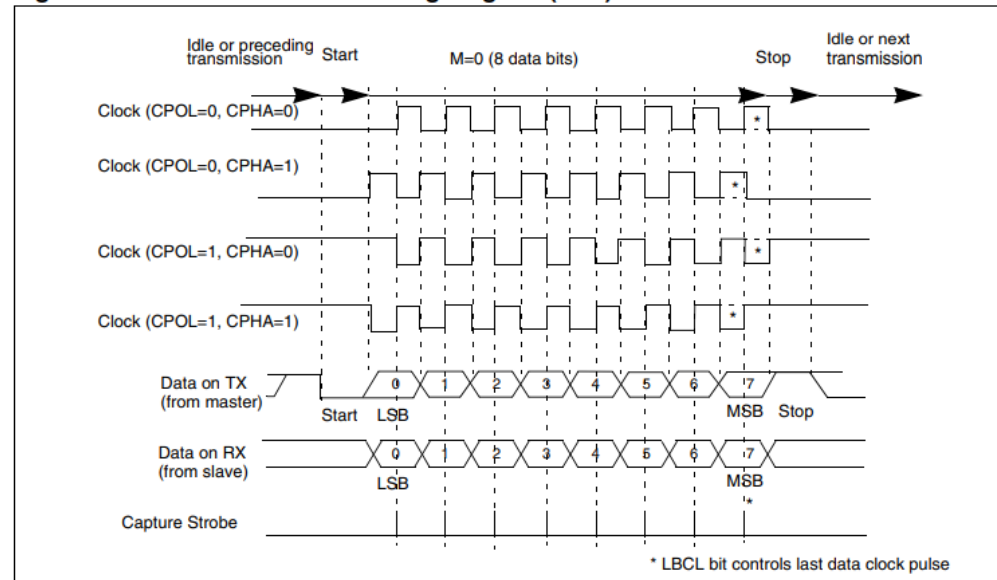


Рис 56 - временные диаграммы контроллера UART .

26.6.3 Baud rate register (USART_BRR)

Note: The baud counters stop counting if the TE or RE bits are disabled respectively.

Address offset: 0x08

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, forced by hardware to 0.

Bits 15:4 **DIV_Mantissa[11:0]**: mantissa of USARTDIV

These 12 bits define the mantissa of the USART Divider (USARTDIV)

Bits 3:0 **DIV_Fraction[3:0]**: fraction of USARTDIV

These 4 bits define the fraction of the USART Divider (USARTDIV)

Рис 57 - регистр задания технической скорости интерфейсов UART.

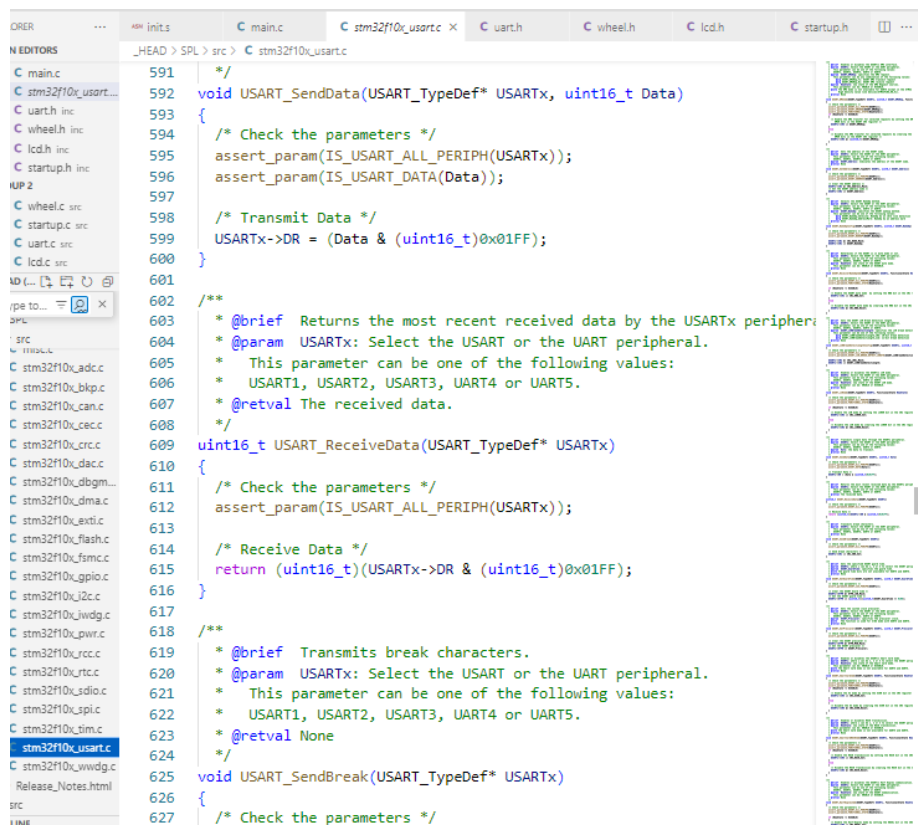


Рис 58 - функция отправки данных по UART предлагаемая библиотекой SPL .

П.4: Отладка через символьный дисплей (подробнее в ЛР№4)

Написать функцию, обеспечивающую вывод на дисплей информации: ФИО, группа, частота микроконтроллера, частота прерываний, скорость UART и количество произведенных прерываний.

Реализация заданной функции показана на листинге (рис 59, 60):

```
162 void lcd_demo()
163 {
164     // --- LCD setup ---
165     /*
166         Pin setup : 4 bits data communication
167
168         PA8  -> RS
169         PB10 -> RW
170         PB11 -> E
171
172         PA4  -> DB4
173         PA5  -> DB5
174         PA6  -> DB6
175         PA7  -> DB7
176     */
177     //Digital_Input(PA,11); // Initialize push button
178
179     lcd_init(); // Initialize the LCD screen to work with 4 bits data interface
180     // lcd_msg(line_number 1 or 2, Position within the line, String to be shown)
181     lcd_msg(1, 1, "Rahimov");
182     lcd_msg(2, 6, "RL1-104");
183     delay_ms(1000);
184     lcd_msg(1,0, "Bench 13 var:  ");
185     delay_ms(1000);
186     lcd_msg(2,0, "Ver ____:  "); // PARAMETRIZE!
```

Рис 59 - функция вывода данных на дисплей. Часть 1.

```

186     lcd_msg(2,0, "Ver ____:   "); // PARAMETRIZE!
187     delay_ms(1000);
188     lcd_msg(2,4, "0.5.2");
189     delay_ms(3000);
190     lcd_msg(1,0, "CPU_FREQ = __MHz");
191     delay_ms(1000);
192     lcd_msg(1,11, itoa(F_CPU_MHZ, string_for_f_cpu, 10));
193     delay_ms(1000);
194     lcd_msg(2,0, "INTRP_FREQ = 4Hz");
195     delay_ms(3000);
196     lcd_msg(1,0, "BAUDRATE =");
197     delay_ms(1000);
198     lcd_msg(1,10, itoa(BAUDRATE, string_for_baudrate, 10));
199     delay_ms(1000);
200     lcd_msg(2,0, "Interrupts: ");
201     delay_ms(1000);
202     lcd_msg(2,12, "0   ");
203     delay_ms(1000);
204 }
205

```

Рис 60 - функция вывода данных на дисплей. Часть 2.

Параметры задаются в начале модуля main.c и их изменение (после перепрошивки) отражается в тексте, который отображается на ЖК-дисплее.

Счётчик прерываний отображается «в реальном времени» на экране (рис 61). Прерывания происходят с заданной частотой и создаются системным таймером.



Рис 61 -отображения счётчика прерываний.

Заключение

Лабораторный стенд выполнил свою задачу. Освоены базовые навыки программирования МК, в т. ч. на уровне регистров. Схему и прошивку можно без конца усложнять. Запускать периферию и подключать дополнительные внешние устройства. Уже готовые и разобранные модули послужат шаблоном для будущих учебных и рабочих проектов.

Есть смысл уделить особое внимание беспроводным технологиям (Wi-Fi, Bluetooth, Zigbee, RFID, цифровым приёмникам вообще и не только), ввиду направления программы обучения.

Стенд выходит недорогим и компактным, а значит может быть использован молодыми специалистами и студентами для знакомства со сферой встраиваемого программирования.

Некоторые использованные материалы:

- [] <https://easyelectronics.ru/arm-uchebnyj-kurs-vneshnie-preryvaniya.html>
- [] <https://easyelectronics.ru/arm-uchebnyj-kurs-usart.html>
- [] <https://prog-cpp.ru/c-directives/>
- [] <https://gnutoolchains.com/arm-eabi/openocd/>
- [] <https://gnutoolchains.com/arm-eabi/>
- [] https://count-zero.ru/2018/stm32_start/#1
- [] <https://www.electronicshub.org/program-stm32f103c8t6-using-keil-uvision/>
- [] <https://narodstream.ru/programmirovanie-mk-stm32/>
- [] <https://narodstream.ru/stm-urok-179-displej-tft-240x320-spi-chast-1/>
- [] <https://habr.com/ru/articles/721184/>
- [] <https://www.st.com/en/microcontrollers-microprocessors/stm32f1-series.html>
- [] <https://stm32-base.org/guides/getting-started.html>

А также:

- [] методическое пособие по МК1986 компании «Миландр»
- [] руководство пользователя по МК линейки STM32F1
- [] описание библиотек для работы с символьными дисплеями.