Быковский С.В., Ключев А.О., Кустарев П.В., Платунов А.Е.

Лабораторный стенд SDK-1.1M
Учебное пособие
(черновой вариант)

ФПИ и КТ
Университет ИТМО
Санкт-Петербург
2019 г

Оглавление

Введение

Учебный лабораторный стенд SDK-1.1M входит в серию стендов SDK-1.1M и является многофункциональным устройством, построенным на базе 32-битного ARM-микроконтроллера. Он предназначен для изучения архитектуры и методов проектирования:

- киберфизических систем и интернета вещей;
- систем на базе микропроцессоров и микроконтроллеров;
- встраиваемых контроллеров и систем сбора данных;
- периферийных блоков вычислительных систем;
- подсистем ввода-вывода встраиваемых систем.

Широкий спектр возможностей стенда SDK-1.1M позволяет использовать его во многих направлениях:

- обучение;
- производство;
- автоматизация управления;
- исследования;
- контрольный пункт управления;
- вычислитель.

Обучение

ВУЗы, колледжи могут использовать стенд SDK-1.1M для проведения учебных и исследовательских работ. Преподаватели могут как разрабатывать свои задания для лабораторных работ, так и использовать готовые, разработанные специалистами ООО «ЛМТ».

Производство

На предприятиях приборостроительных отраслей стенд SDK-1.1M будет удобен для использования в качестве прототипа при разработке разнообразных электронных модулей – контроллеров и приборов.

Автоматизация управления

В распределенных системах сбора данных и управления стенд SDK-1.1M может использоваться в качестве центрального контроллера для подключения GSM-модемов, модулей GPS/ГЛОНАСС, Wi-Fi, ZigBee и т.п.

Исследования

В лабораториях и на производстве SDK-1.1M будет удобен для автоматизации лабораторных исследований и простых технологических процессов.

Контрольный пункт управления

Стенд SDK-1.1M будет полезен в качестве простой панели оператора для организации пункта управления.

Вычислитель

Лабораторный стенд SDK-1.1M подойдет для решения вычислительных задач, т.к. обладает высокой вычислительной мощностью и широким набором интерфейсов ввода-вывода.

Комплектация SDK-1.1M

Процессорные модули SDK-1.1M

Сменные процессорные модули для подключения к несущей плате учебного стенда SDK-1.1M

Таблица. 1. Процессорные модули SDK-1.1M

Характеристики	Type 107	Type 407	Type 427	Type MX6	Type XC7
Вычислитель	STM32F107	STM32F407	STM32F427	i.MX 6ULL	ПЛИС Xilinx
Тактовая частота	72 МГц	168 МГц	180 МГц	до 900 МГц	в разработке
RAM	64 КБ	192 КБ	256 КБ	512 МБ	
Встроенная FLASH-память	256 КБ	1 МБ	2 МБ	-	
Внешняя память SerialFLASH	-	16 МБ	16 МБ	16 МБ	
Micro SD	-	да	да	да	
USB	да	да	да	да	

Несущие платы SDK-1.1M

Таблица. 2. Несущие платы SDK-1.1M

Характеристики	Model A	Model B	Model C
Ethernet	-	-	100 Мбит/с
RS-485	-	да	да
OLED дисплей 128x64	да	да	да
Клавиатура 3x4	да	да	да
Подключение плат SDK-X	-	да	да
Подключение Arduino-совместимых плат	-	-	да
Питание от отладочного USB-порта (micro USB)			
Напряжение	5 В ± 0,5 В		
Ток	до 500 мА		
Питание от внешнего источника постоянного тока			
Напряжение	9÷36 В		
Ток	до 750 мА		

Процессорные модули SDK-1.1M

Микроконтроллер STM32F107VCT6 (Type 107)

- Процессорное ядро: ARM® 32-bit Cortex®-M3 CPU
  - Максимальная частота 72 МГц;
  - Однократное умножение и аппаратное разделение.
- Память
  - От 64 до 256 Кб флеш-памяти;
  - 64 Кб SRAM.
- Управление часами, сбросом, питанием
  - От 2 до 3.6 В на портах ввода/вывода;
  - Power-on reset (POR), Port Direction Register (PDR), Programmable voltage detector (PVD);
  - Кварцевый генератор с частотой от 3 до 25 МГц;
  - Генератор с частотой 32 КГц для часов реального времени (RTC) с калибровкой;
- Малое энергопотребление
  - Режимы сна, остановки, ожидания;
  - Питание от батареи для RTC и резервных регистров;
- 2 12-битных 1 мкс АЦП (16 каналов)
  - Диапазон от 0 до 3.6 В;
  - Датчик температуры
  - Up to 2 MSPS in interleaved mode
- 2 12-битных ЦАП
- 12-ти канальный DMA контроллер
  - Поддерживаемые периферийные устройства: таймеры, АЦП, ЦАП, I^2C, SPI, I^2S и USART.
- Режим отладки
  - Интерфейсы Serial wire debug (SWD) и JTAG
  - Cortex®-M3 Embedded Trace Macrocell™
- 80 быстрых портов ввода/вывода
- Блок вычисления CRC, уникальный 96-битный индефикатор
- 10 таймеров с возможностью переназначения контактов
- 14 интерфейсов связи с возможностью переназначения контактов
  - 2 I^2C интерфейса (SMBus/PMBus);
  - 5 USART;
  - 3 SPI (18 Мбит/с);
  - 2 CAN интерфейса с 512 байт выделенной SRAM;
  - USB 2.0 полноскоростной device/host/OTG контроллер с PHY;
  - USB 2.0 высокоскоротной/полноскоростной device/host/OTG контроллер с выделенной DMA, полноскоростным PHY и ULPI;
  - 10/100 Ethernet MAC с выделенной DMA и SRAM (4 Кб): аппаратная поддержка IEEE1588.

Микроконтроллер STM32F407VGT6 (Type 407)

- Процессорное ядро: ARM® 32-bit Cortex®-M4 CPU с модулем операций с плавающей запятой; частота до 168 МГц; блок защиты памяти (MPU).
- Память:
  - 1 Мбайт FLASH-памяти для программ и данных;
  - 192+4 Кбайт SRAM-памяти, включая 64 Кбайт CCM (core coupled memory);
  - Контроллер внешней памяти, поддерживающий устройства памяти типа Compact Flash, SRAM, PSRAM, NOR и NAND.
- Параллельный интерфейс LCD с режимами 8080 и 6800
- Управление синхронизацией, сбросом, питанием:
  - Напряжение питания и ввода/вывода от 1,8 до 3,6 В;
  - Power-on reset (POR), Power-down reset (PDR);
  - Подключение внешнего кварцевого генератора с частотой от 4 до 26 МГц;
  - Встроенный RC-генератор 16 МГц;
  - Встроенный генератор с частотой 32 кГц для часов реального времени (RTC);
  - Режимы работы с пониженным энергопотреблением.
- 3 12-битных с 2,4 млн выборок/с АЦП: до 24 каналов и 7,2 млн выборок/с в режиме тройного чередования
- 2 12-битных ЦАП
- Отладочные возможности:
  - Интерфейсы JTAG и SWD (Serial wire debug);
  - Cortex®-M3 Embedded Trace Macrocell™.
- Более 70 портов ввода/вывода, устойчивых к уровням сигналов до 5 В
- Стандартные интерфейсы:
  - I^2C (в т.ч. SMBus/PMBus);
  - USART/UART;
  - SPI (до 42 Мбит/с);
  - I^2S (полнодуплексный);
  - CAN с 512 байт выделенной SRAM;
  - SDIO.
- Расширенные возможности подключения:
  - USB 2.0 Full Speed device/host/OTG контроллер с встроенной микросхемой PHY;
  - 10/100 Ethernet MAC с выделенной DMA и SRAM (4 Кбайт): аппаратная поддержка IEEE1588v2.
- 8-14-битный параллельный интерфейс камеры со скоростью до 54 Мбайт/с
- Аппаратный генератор случайных чисел
- Блок вычисления CRC
- 96-битный уникальный идентификатор
- Часы реального времени (RTC)

Микроконтроллер STM32F427ViT6 (Type 427)

- Процессорное ядро: ARM® 32-bit Cortex®-M4 CPU с модулем операций с плавающей запятой, ART ускоритель, обеспечивающий мгновенное выполнение из флеш-памяти, с частотой до 180 МГц, с блоком защиты памяти (MPU).
- Память
  - 2 Мбайт флеш-памяти;
  - 256+4 Кбайт SRAM, включая 64 Кбайт CCM (core coupled memory);
  - Гибкий статический контроллер памяти с 32-битной шиной данных, поддерживающий Compact Flash, SRAM, PSRAM, SDRAM/LPSDR SDRAM, NOR и NAND память.
- Chrom-ART ускоритель для улучшения создания графического содержимого
- Управление часами, сбросом, питанием
  - От 1.7 до 3.6 В на портах ввода/вывода;
  - Power-on reset (POR), Port Direction Register (PDR), Programmable voltage detector (PVD);
  - Кварцевый генератор с частотой от 4 до 26 МГц;
  - Генератор с частотой 32 КГц для часов реального времени (RTC) с калибровкой;
  - Режимы сна, остановки, ожидания;
- 3 12-битных с 2.4 млн выч/с АЦП: до 24 каналов и 7.2 млн выч/с в режиме тройного чередования
- 2 12-битных ЦАП
- Универсальный DMA: 16-поточный котроллер DMA с поддержкой FIFO
- 17 таймеров: 12 16-битных и 2 32-биных таймеров, каждый из которых имеет частоту 180 МГц
- Режим отладки
  - Интерфейсы Serial wire debug (SWD) и JTAG
  - Cortex®-M3 Embedded Trace Macrocell™
- 168 портов ввода/вывода с возможностью прерываний
  - 164 быстрых портов ввода/вывода с частотой 90 МГц;
  - 165 портов, выдерживающих 5 В.
- 21 интерфейс связи
  - 3 I^2C интерфейса (SMBus/PMBus);
  - 4 USART и 4 UART;
  - 6 SPI (45 Мбит/с), 2 с полностью двусторонним I^2S;
  - SAI (serial audio interface)
  - 2 CAN интерфейса с 512 байт выделенной SRAM;
  - Интерфейс SDIO.
- Расширенные возможности подключения
  - USB 2.0 полноскоростной device/host/OTG контроллер с встроенной микросхемой PHY;
  - 10/100 Ethernet MAC с выделенной DMA и SRAM (4 Кб): аппаратная поддержка IEEE1588v2.
- 8-14-битный параллельный интерфейс камеры со скоростью до 54 Мб/с
- Генератор случайных чисел
- Модуль вычислений CRC
- 96-битный уникальный индефикатор
- RTC: точность в секундах, аппаратный календарь

Микропроцессор NXP i.MX 6ULL (Type MX6)

- Процессорное ядро ARM Cortex-A7:
  - Базовая частота 528 МГц;
  - Частота в разгоне 900 МГц;
- General Interrupt Controller (GIC) с поддержкой 128 прерываний;
- Глобальный таймер;
- Snoop Control Unit (SCU);
- Шина интерфейса вывода L2 кэша Single Master AXI (128 бит);
- NEON MPE сопроцессор:
  - Архитектура обработки мультимедиа SIMD;
  - Регистровый файл NEON с 32x32-битными, 32x64-битными и 16x128-битными регистрами общего назначения;
  - Целочисленный вычислительный конвейер NEON (АЛУ, Сдвиг, MAC);
  - Двойной, одинарный вычислительный конвейер NEON с плавающей запятой (FADD, FMUL);
  - Конвейер NEON загрузки/хранения и перемещения;
  - Обеспечение преобразований между 16-битными, 32-битным, 64-битными форматами с плавающей запятой и целочисленными текстовыми форматами ARM.
- Память: процессор поддерживает высокопроизводительную DRAM, NOR, флэш-память, а также SD-карты. Система памяти состоит из следующих компонентов:
  - Кэш 1 уровня – 32 Кб команд, 32 Кб данных;
  - Кэш 2 уровня – объединенные 128 Кб команд и данных;
  - Загрузочное ПЗУ, включающее 96 Кб высокоуровневой загрузки (HAB);
  - Встроенное ОЗУ быстрого доступа (OCRAM, 128 Кб);
- Встроенные интерфейсы памяти:
  - 16-битная LP-DDR2, 16-битная DDR3-400, LV-DDR3-400;
  - 8-битная NAND флэш-память, включающая поддержку MLC/TLC, 2 Кб, 4 Кб и 8 Кб размера страницы;
  - До 40 бит BCH ECC;
  - 16-битная флэш-память;
  - 16-битная PSRAM, Cellular RAM;
  - Двухканальная/одноканальная QuadSPI флэш-память.
- Дисплеи:
  - LCDIF – поддерживает один параллельный 24-битный LCD дисплей с разрешением 1366x768 и частотой 60 Гц;
  - EPDC – поддерживает прямой драйвер для панелей E-Ink EPD с разрешением 2048x1536 с частотой 106 Гц или 4096x4096 с частотой 20 Гц.
- Параллельный 24-битный порт камеры с частотой 66 МГц;
- Карты расширения: 4 MMC/SD/SDIO порта для карт, поддерживающие:
  - 1-битный или 4-битный режим передачи инструкций для SD и SDIO карт, вплоть до режима UHS-1 SDR-104 (104 Мб/с);
  - 1-битный, 4-битный, 8-битный режим передачи инструкций для MMC карт, вплоть до 52 МГц в SDR и DDR режимах;
  - Совместимость с SD, miniSD, SDIO, SD Combo, MMC, встроенной MMC и встроенной SD-картой;
  - Частота варьируется от 32 КГц до 52 КГц;

    - До 200 Мб/с передачи данных для SD/SDIO карт, использующих 4 параллельных линии данных;
    - До 416 Мб/с передачи данных для MMC карт, использующих 8 параллельных линии данных;
    - До 832 Мб/с передачи данных для MMC/SD карт, использующих 8 параллельных линии данных в режиме DDR;
- 2 высокоскоростных USB 2.0 OTG (до 480 Мбит/с), с встроенной высокоскоростной USB PHY;
- Различные интерфейсы:
    - 3 I^2S/SAI/AC97, работающих со скоростью до 1.4 Мбит/с;
    - ESAI;
    - 8 UART (5 Мбит/с) которые обеспечивают интерфейс RS-232 и поддерживают 9-битный многоточечный режим RS-485;
    - 4 eCSPI, три из которых поддерживают до 52 Мбит/с и один низкоскоростной;
    - 4 I^2C, поддерживающих 400 Кбит/с;
    - Контроллер Ethernet IEEE1588 10/100 Мбит/с;
    - 8 ШИМ-модуляторов;
    - System JTAG Controller (SJC);
    - Порты ввода/вывода с поддержкой прерываний;
    - Порт для клавиатуры 8x8;
    - Sony Philips Digital Interface (SPDIF), Rx и Tx;
    - 2 CAN (1 Мбит/с);
    - 3 таймера Watchdog (WDOG);
    - Асинхронный конвертер частоты дискретизации (ASRC);
    - Medium Quality Sound (MQS).

Периферийные устройства SDK-1.1M

Расширитель ввода/вывода PCA9538PW

PCA9538PW – это 8-битный расширитель портов GPIO с поддержкой прерываний, подключенный по интерфейсу I^{2}C. PCA9538PW состоит из 8-битного регистра конфигурации (вход или выход на выбор), 8-битного регистра входного порта, 8-битного выходного порта и 8-битного регистра инверсии полярности.

В стенде SDK-1.1M установлено два расширителя PCA9538PW для обработки сигналов от различных периферийных устройств.

Часы реального времени MCP79411

MCP79411 – часы/календарь с 1 Кбит встроенной энергонезависимой памяти EEPROM с защищенными от записи областями, работает на частоте 32,768 кГц. Время отслеживается с использованием внутренних счетчиков часов, минут, секунд, дней, месяцев, лет, дней недели. Сигнализация может быть настроена на всех счетчиках вплоть до месяцев. Для использования и настройки MCP79411 поддерживает I^{2}C со скоростью до 400 кГц.

Графический OLED-дисплей WEO012864DL

Монохромный OLED-дисплей 128x64 точки с диагональю 0,96 дюйма, подключенный по интерфейсу I^{2}C. Размеры активной области 21,8 x 10,9 мм. Дисплей оборудован встроенным контроллером IC SSD1306.

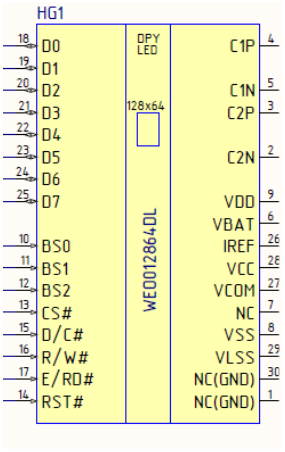

Рис. 1. Графический OLED дисплей WEO012864DL

Ethernet

Учебный стенд SDK-1.1M оборудован разъемом RJ-45 для подключения по Ethernet со скоростью 10/100 Мбит/с.

USB

На процессорной плате размещается имеется разъем USB 2.0 micro B для подключения в режиме host, device или OTG.

Излучатель звука HC0903A

HC0903A – электромагнитный излучатель звука, управляемый прямоугольным периодическим сигналом.

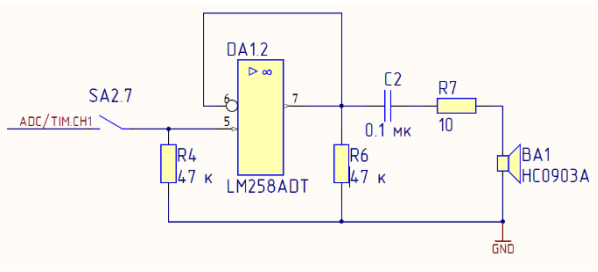

Рис. 2. Излучатель звука HC0903A

Инерционный модуль iNEMO LSM9DS

LSM9DS – это система, состоящая из трехмерного цифрового датчика линейного ускорения, трехмерного цифрового датчика угловой скорости и трехмерного цифрового магнитного датчика. LSM9DS подключен по последовательной шине I^2C.

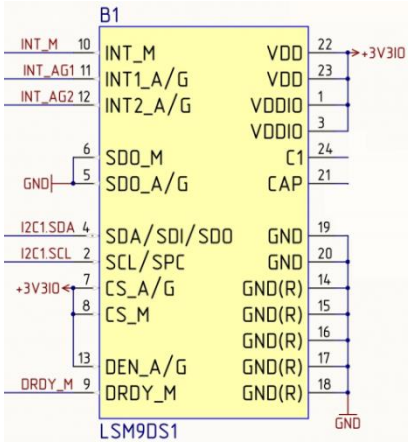

Рис. 3. Инерционный модуль iNEMO LSM9DS

Клавиатура

Клавиатура организована в виде матрицы 3x4, подключенной к расширителю портов. Три бита соответствуют колонкам, четыре бита соответствуют рядам.

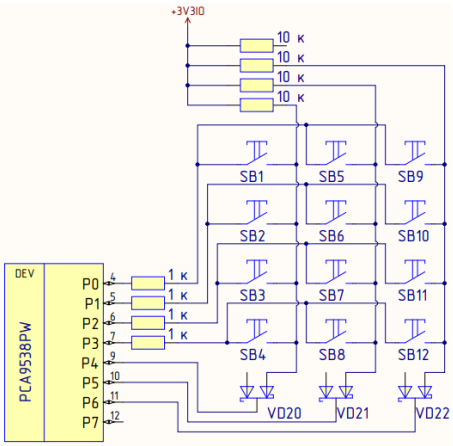

Рис. 4. Матричная клавиатура

Встроенный программатор-отладчик

Для отладки и загрузки программ в память SDK-1.1M используется отладочный разъем USB 2.0 micro B (Debug USB), подключенный к встроенному в стенд программатору-отладчику. Программатор-отладчик обеспечивает подключение к микроконтроллеру по интерфейсу JTAG.

Инструкция по эксплуатации

Общий вид стенда SDK-1.1M

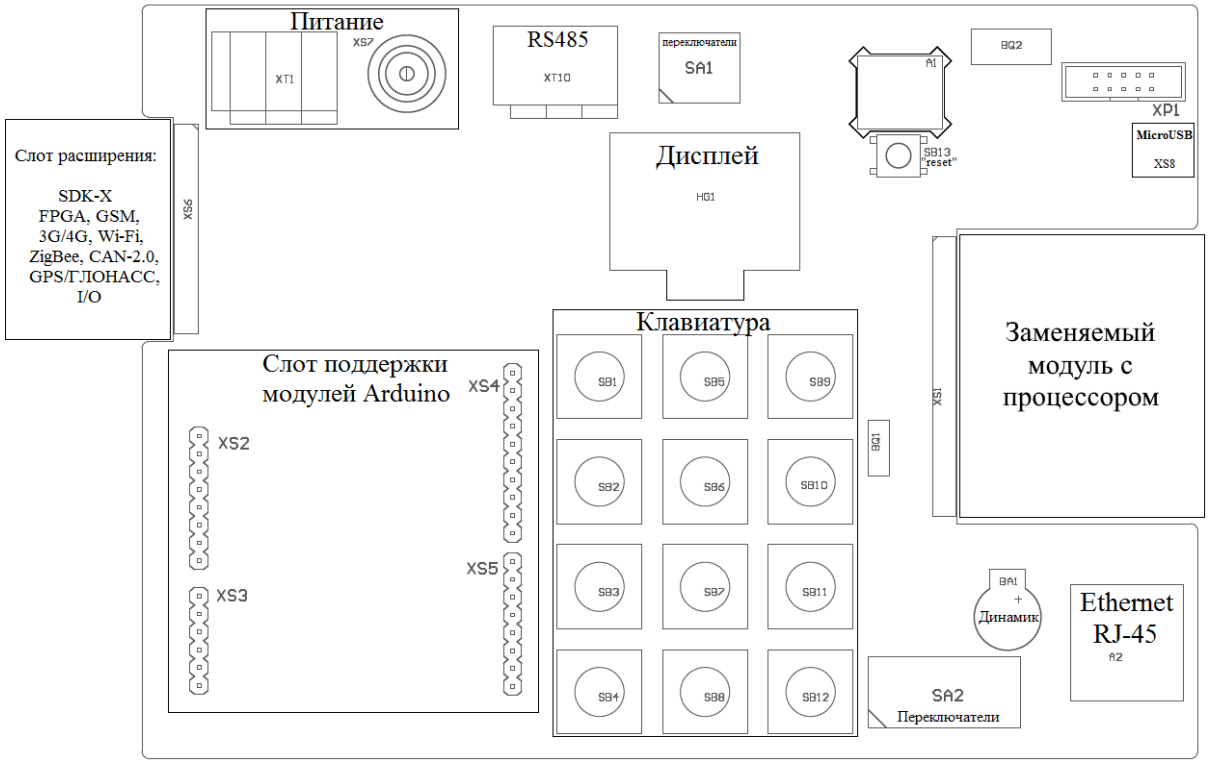

Рис. 5. Общий вид стенда SDK-1.1M

На рисунке представлено схематическое изображение лицевой панели стенда SDK-1.1M. Расшифровка обозначений на схеме дана в таблице.

Таблица. 3. Расшифровка обозначений на схеме лицевой панели стенда SDK-1.1M

Элемент	Описание
Питание (XT1, XS7)	Внешнее питание (если необходимо). Полярность подключения не имеет значения.
XS6	Слот подключения FPGA,GSM, Wi-Fi, GPS/ГЛОНАСС и т.д.
XS2…XS5	Слот для модулей расширения Arduino.
XT10 “RS485”	Разъем взаимодействия SDK-1.1M по интерфесу RS-485.
Переключатели SA1, SA2	Конфигурируемые переключатели.
Дисплей HQ1	Графический OLED дисплей WEO012864DL (см. предыдущий раздел).
Клавиатура SB1…SB12	Клавиатура (см. предыдущий раздел).
SB13 “reset”	Кнопка сброса RESET.
XS8 “MicroUSB”	MicroUSB разъем, предназначенный для программирования SDK-1.1M.
XS1	Разъем подключения платы с микроконтроллером.
Динамик BA1	Излучатель звука HC0903A (см. предыдущий раздел).

A2	Разъем RJ-45, для подключения кабеля Ethernet 10/100.

Инструменты для разработки для SDK-1.1M на базе микроконтроллеров STM32 (Type 107, Type 407, Type 427)

Рекомендуется использовать следующее программное обеспечение (https://www.st.com):

- STM32CubeMX;
- System Workbench for STM32.

STM32CubeMX – это графический инструмент, позволяющий конфигурировать микроконтроллеры STM32 и генерировать соответствующие шаблоны проектов с кодом на языке С посредством пошагового процесса.

System Workbench for STM32 – кроссплатформенная свободно распространяемая среда разработки (IDE).

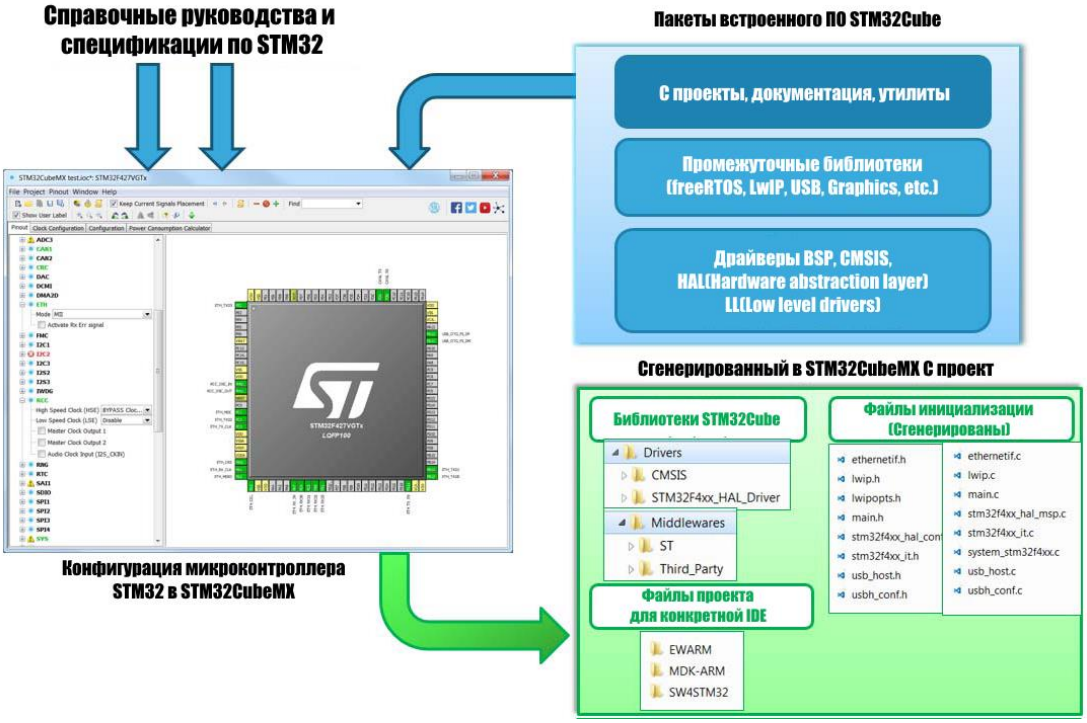

Рис. 6. Схема генерации проекта в программе STM32CubeMX

Установка драйверов

Для программирования и отладки SDK-1.1M оборудован программатором YPROG. Подключение к компьютеру производится через кабель USB-microUSB. После подключения к компьютеру необходимо установить драйвер FTDI.

Рис. 7. Расположение разъема YPROG на стенде SDK-1.1M

Порядок установки драйвера для **Windows:**

1. Скачать и запустить программу Zadig (https://zadig.akeo.ie);
2. Подключить SDK-1.1M к компьютеру;
3. Во вкладке Options выбрать List All Devices (рисунок 8);
4. Выбрать из списка RS232 (Interface 0) или SDK 1.1M Debugger (Interface 0);
5. Выбрать драйвер WinUSB и нажать кнопку установки (рисунок 8).

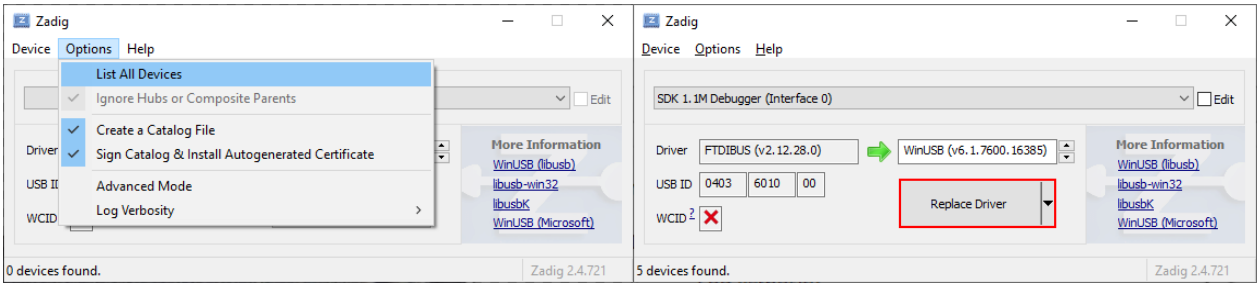

Рис. 8. Процесс установки драйвера

Порядок установки драйвера для **Linux (Debian/Ubuntu):**

1. Открыть терминал;
2. Ввести команду: `sudo apt-get install libusb-1.0-0`;
3. Ввести команду: `sudo nano /etc/udev/rules.d/50-myusb.rules` или `sudo vi /etc/udev/rules.d/50-myusb.rules`;
4. Добавить строку: `SUBSYSTEMS=="usb", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6010",TAG+="uaccess"`
5. Сохранить и закрыть файл;
6. Ввести команду: `sudo udevadm control –reload`;
7. Переподключить SDK-1.1M к компьютеру.

Настройка IDE

Настройки проекта в STM32CubeMX

При создании каждого проекта для SDK-1.1M необходимо добавлять Debug JTAG (5 pins): Вкладка Pinouts & Configuration -> System Core -> SYS -> Debug -> JTAG (5 pins) рисунок 11.

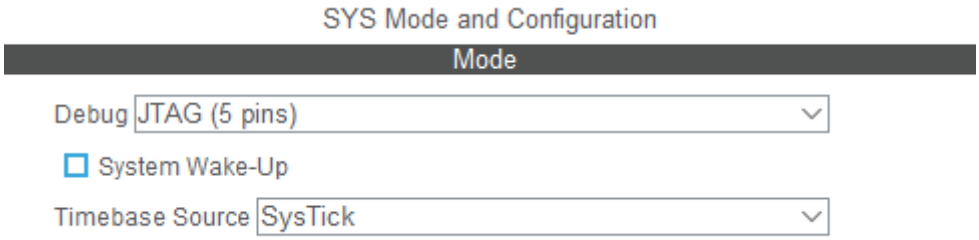

Рис. 9. Параметры Debug

Во вкладке Project Manager рекомендуется использовать следующие параметры:

- Во вкладке Project задать имя проекта и путь к нему;
- В поле Toolchain/IDE – SW4STM32;
- Во вкладке Code Generator в Generated files поставить галочку Generate peripheral initialization as pair of '.c/.h' files per peripheral.

Настройки проекта в System Workbench for STM32

После первой сборки проекта, необходимо добавить два файла:

Для SDK-1.1MC.107:

SDK1_1_M FTDBG.cfg

```
source [find SDK11M_FT.cfg]
set WORKAREASIZE 0x8000
transport select jtag
set CHIPNAME STM32F107VCTx
set BOARDNAME SDK1_1M
# STlink Debug clock frequency
#set CLOCK_FREQ 9000
# use hardware reset, connect under reset
# connect_assert_srst needed if low power mode application running (WFI...)
reset_config srst_only
# srst_nogate connect_assert_srst
set CONNECT_UNDER_RESET 1
source [find target/stm32f1x.cfg]
```

SDK11M_FT.cfg

```
interface ftdi
ftdi_vid_pid 0x0403 0x6010
```

```
ftdi_layout_init 0x0028 0x0ffb
#ftdi_layout_signal OE -data 0x0020
ftdi_layout_signal nSRST -oe 0x0800
#ftdi_set_signal OE 1
```

Для SDK-1.1MC.407:

SDK1_1_M FTDBG.cfg

```
source [find SDK11M_FT.cfg]
set WORKAREASIZE 0x8000
transport select jtag
set CHIPNAME STM32F407VGTx
set BOARDNAME SDK1_1_M
reset_config srst_only
# srst_nogate connect_assert_srst
set CONNECT_UNDER_RESET 1
source [find target/stm32f4x.cfg]
```

SDK11M_FT.cfg

```
interface ftdi
ftdi_vid_pid 0x0403 0x6010
ftdi_layout_init 0x0408 0x0ffb
#ftdi_layout_signal OE -data 0x0020
ftdi_layout_signal nSRST -oe 0x0800
#ftdi_set_signal OE 1
```

Для SDK-1.1MC.427:

SDK1_1_M FTDBG.cfg

```
source [find SDK11M_FT.cfg]
set WORKAREASIZE 0x8000
transport select jtag
set CHIPNAME STM32F427VITx
set BOARDNAME SDK1_1_M
reset_config srst_only
# srst_nogate connect_assert_srst
set CONNECT_UNDER_RESET 1
source [find target/stm32f4x.cfg]
```

SDK11M_FT.cfg

```
interface ftdi
ftdi_vid_pid 0x0403 0x6010
ftdi_layout_init 0x0408 0x0ffb
```

```
#ftdi_layout_signal OE -data 0x0020
ftdi_layout_signal nSRST -oe 0x0800
#ftdi_set_signal OE 1
```

После этого нажать File -> Properties -> Run/Debug Settings -> New -> Ac6 STM32 Debugging. Установить параметры как на рисунках 10 и 11.

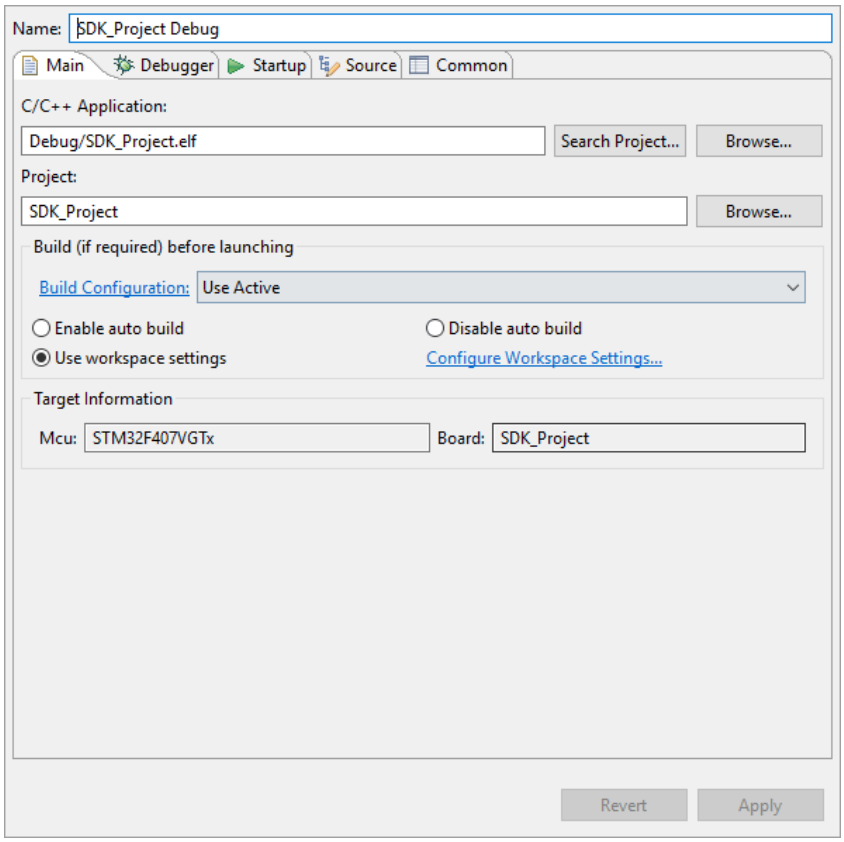

Рис. 10. Параметры Main

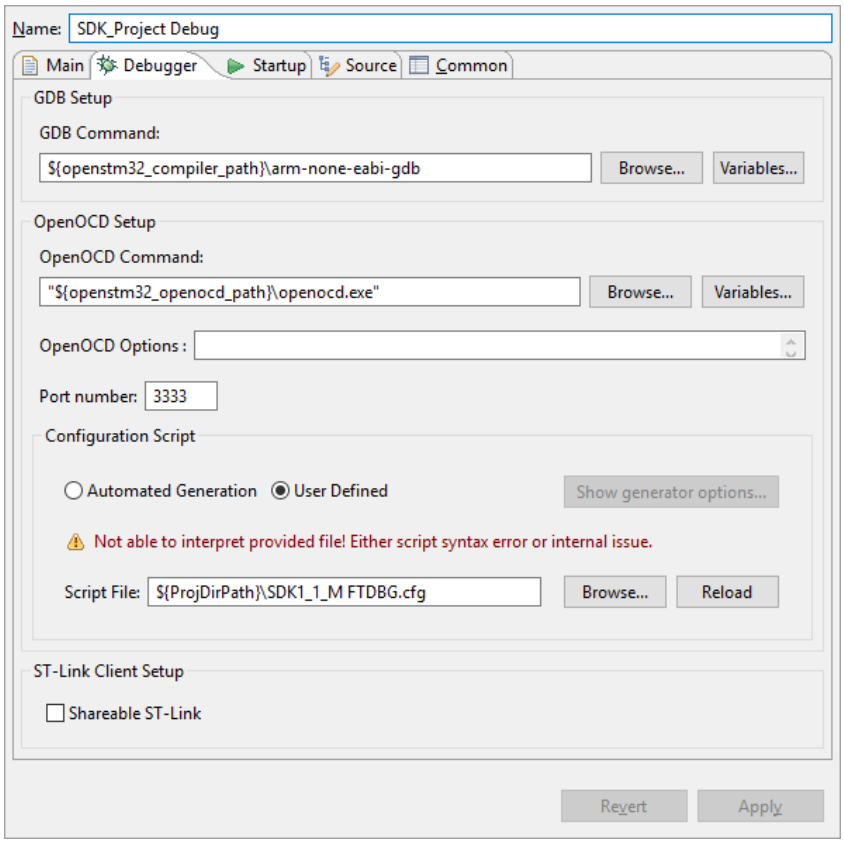

Рис. 11. Параметры Debugger

Требования к лабораторным работам

Требования к выполнению работ

Проект должен быть в виде отдельных файлов, каждый из которых содержит отдельный драйвер или цельную, логически законченную часть программы.

Код должен быть структурированным, хорошо читаемым. Названия переменных и функций должны быть осмысленными и поясняющими назначение данных переменных и функций. На «верхнем уровне» программы (функция main) должна быть только прикладная часть, вся логика работы с аппаратурой скрывается от пользователя «верхнего уровня» и реализуется на уровне драйверов.

Не должно быть одинаковых или сильно похожих блоков кода, желательно реализовывать их с помощью отдельных функций. Крайне желательно минимизировать количество вложенных циклов и свести их к необходимому минимуму.

Требования к отчётам

Отчёт должен содержать:

1. Титульный лист с номером лабораторной работы, её названием, ФИО и группой исполнителей.
2. Номер варианта, задание.
3. Блок-схема прикладного алгоритма.
4. Подробное описание инструментария, который использовался в работе.
5. Исходные коды прикладной программы и использованных библиотек.
6. Выводы, в которых описываются проблемы, которые возникли в ходе работы, и способы их решения.

Защита лабораторной работы

На защите задаются вопросы по:

1. Теории.
2. Инструментальным средствам, которые использовались для работы.
3. Принципиальной схеме стенда SDK.
4. Исходным кодам, включая библиотеки.

Лабораторная работа, «Интерфейсы ввода/вывода общего назначения (GPIO)»

Задание

Разработать и реализовать драйверы GPIO (управление светодиодными индикаторами и обработка нажатий кнопки контроллера SDK-1.1M). Написать программу с использованием разработанных драйверов по алгоритму, соответствующему варианту задания.

Общие сведения

Интерфейс ввода/вывода общего назначения (general-purpose input/output, GPIO) – базовый интерфейс взаимодействия компьютерной системы с внешним миром. С его помощью чаще всего подключаются такие внешние элементы как светодиоды, кнопки, переключатели, осуществляется управление периферийными устройствами и т.д.

GPIO контакты группируются в порты, обычно по 8, 16 или 32 линий. Одни и те же контакты GPIO могут выступать в роли входа или выхода. Режим входа обычно включен по умолчанию, для работы в качестве выхода необходимо настроить соответствующую ножку на выход при помощи управляющего регистра. Каждый контакт настраивается индивидуально – на одном порте могут чередоваться входы и выходы в любых комбинациях.

Управление выходом может быть прямым, из программы (записать 0 или 1), может осуществляться при помощи внутренних устройств (например, реализация широтно-импульсной модуляции на базе таймеров).

В учебном стенде SDK-1.1M с процессорным модулем на базе STM32 есть одна кнопка, подключённая к PC15, и два управляемых светодиода: зеленый, подключенный к PD13, и двухцветный красный/желтый, подключенный к контактам PD14, PD15 (Рис. 1).

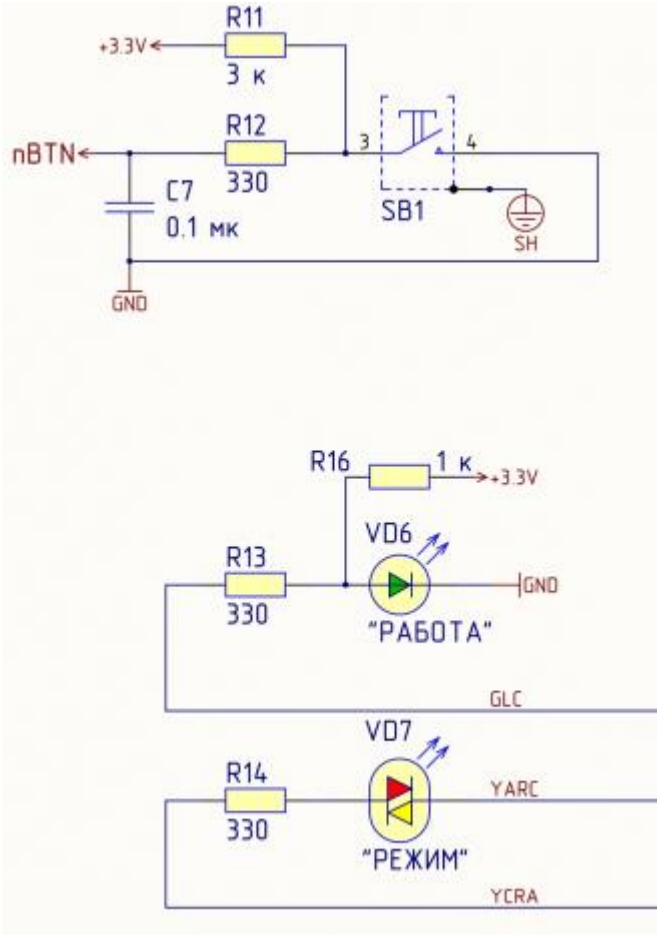

Рис. 12. Кнопка и управляемые светодиоды на принципиальной схеме стенда

Работа с GPIO

Сперва, необходимо произвести настройку Pinout в STM32CubeMX: включить контакты PD13-PD15 в режим GPIO_Output (рисунок 1).

Добавить Debug JTAG (5 pins): Вкладка Pinouts & Configuration -> System Core -> SYS-> Debug -> JTAG (5 pins).

Включить тактовый генератор RCC: Вкладка Pinouts & Configuration -> System Core -> RCC-> High Speed Clock -> Crystal/Ceramic Resonator.

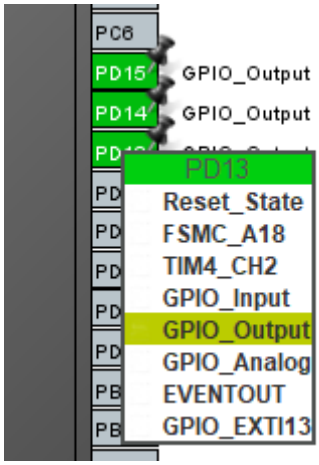

Рис. 13. Окно настройки pinout UART

Для работы с GPIO на STM32 используется библиотека HAL (Hardware Abstract Layer). STM32CubeMX при генерации проекта добавляет в код функцию инициализации UART MX_GPIO_Init.

```
static void MX_GPIO_Init(void)
{
      GPIO_InitTypeDef GPIO_InitStruct = {0};
      __HAL_RCC_GPIOH_CLK_ENABLE();
      __HAL_RCC_GPIOD_CLK_ENABLE();
      __HAL_RCC_GPIOA_CLK_ENABLE();
      __HAL_RCC_GPIOB_CLK_ENABLE();
      HAL_GPIO_WritePin(GPIOD,GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15,
      GPIO_PIN_RESET);
      GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
      GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
      GPIO_InitStruct.Pull = GPIO_NOPULL;
      GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
      HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
}
```

Для переключения состояния светодиода можно использовать стандартную функцию из библиотеки HAL: `HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);`

- `GPIOx` – буква x предназначена для выбора периферийного устройства, может быть (A-K). Для управления светодиодом нужно использовать букву D;
- `GPIO_Pin` – номер управляемого контакта. Например: `GPIO_Pin_13`.

Пример кода:

```
while (1)
  {
        HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_13);
```

```
        HAL_Delay(500);
        HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_13);
        HAL_Delay(500);
        HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_14);
        HAL_Delay(500);
        HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_14);
        HAL_Delay(500);
        HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);
        HAL_Delay(500);
        HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);
        HAL_Delay(500);
}
```

Этот код поочередно включает и выключает светодиоды с задержкой в 500 мс.

Варианты заданий

Вариант 1.

Сымитировать работу светофора пешеходного перехода. В режиме по умолчанию светофор переключает цвета в следующем режиме: зелёный-мигающий зелёный-жёлтый-карсный-зелёный…, при этом период красного значительно больше. При нажатии кнопки происходит переключение с красного на зелёный, но два включения «зелёных» не могут идти сразу друг за другом – между ними должен быть период, больше или равный ¼ периода красного.

Вариант 2.

Реализовать простой имитатор гирлянды с переключением режимов. Должно быть реализовано не менее 4х последовательностей переключения диодов, обязательно с разной частотой мигания. Нажатие кнопки реализует переключение на другой режим. Если режим последний в списке, нажатие кнопки должно переключать на первый режим. После повторного включения режимы должны запускаться с того места, на котором были прерваны.

Вариант 3.

Реализовать «передатчик» азбуки Морзе. Последовательность из нажатий кнопки (короткое – точка, длинное – тире) запоминается, и после сигнала окончания ввода (долгая пауза) начинает «отправляться» при помощи зелёного светодиода, последовательностью быстрых (точка) и долгих (тире) мерцаний. Во время ввода последовательности после каждого нажатия двуцветный диод должен коротким мерцанием индицировать, какой сигнал был введён (мигание жёлтым – точка, мигание красным – тире).

Вариант 4.

Реализовать двоичный двухразрядный счётчик на светодиодах, с возможностью вычитания. Быстрое нажатие кнопки должно прибавлять 1 к отображаемому на диодах двоичному числу, по переполнению счётчика должна отображаться простая анимация: мигание обоими светодиодами, затем количество миганий зелёным светодиодом, равное количеству переполнений. Долгое нажатие кнопки должно вычитать 1 от отображаемого на диодах двоичного числа, если происходит вычитание из нуля – уменьшается на 1 количество переполнений, и отображается анимация, аналогичная анимации при переполнении.

Вариант 5.

Реализовать «кодовый замок». После ввода единственно верной последовательности из не менее чем восьми коротких и длинных нажатий должен загореться зелёный светодиод,

индицирующий «открытие» замка. Светодиод горит некоторое время, потом гаснет и система вновь переходит в «режим ввода». Каждый неправильно введённый элемент последовательности должен сопровождаться миганием красного светодиода и сбросом в «начало», каждый правильный – миганием жёлтого. После трёх неправильных вводов начинает мигать красный диод, и через некоторое время возвращается в «режим ввода».

Лабораторная работа, «Последовательный интерфейс UART»

Задание

Разработать и написать драйверы UART для учебно-лабораторного стенда SDK-1.1M, *с использованием* и *без использования* прерываний. Написать тестовую программу для разработанных драйверов, которая выполняет определенную вариантом задачу.

Общие сведения

Интерфейс UART(USART) широко применяется в вычислительной технике для связи между цифровыми устройствами и фактически является стандартом «де-факто» для подключения периферийных устройств (самый распространённый пример – различные беспроводные модемы).

С помощью UART соединяются отдельные микросхемы, физически располагающиеся близко, как правило – на одной плате. Для передачи сигналов между конструктивно самостоятельными устройствами и на большие расстояния сигнал UART необходимо пропустить через приёмопередатчик, преобразующий передаваемый сигнал в сигналы таких стандартов, как RS-232 (позволяет передавать сигнал на расстояние порядка 15м) или RS-485 (осуществляет передачу сигнала на километры, при соблюдении некоторых условий и подключении ретрансляторов).

Современные микроконтроллеры практически всегда имеют в своём составе отдельную функциональную единицу, контроллер UART(USART), аппаратно реализующую данный интерфейс. Такие контроллеры самостоятельно переключают соответствующие входы/выходы микроконтроллера, формируют и считывают сигналы. Управление контроллера процессором осуществляется через запись/чтение в соответствующие регистры, подключённые к системной шине.

UART – последовательный интерфейс передачи данных. Это предполагает одну сигнальную линию (провод, дорожку на плате) для передачи данных в одном направлении, по которой информационные биты передаются друг за другом, последовательно.

Стандарт UART является чисто асинхронным интерфейсом, но реализующий его контроллер, как правило, может настраиваться в широких пределах и функционировать как в синхронном, так и асинхронном режимах.

Синхронный режим предполагает наличие средств синхронизации передатчика и приемника. Как правило, для синхронизации используют специальную линию для передачи тактовых импульсов. Информация в канале данных считывается приемником только в те моменты, когда на линии синхронизации сигнал активный.

В *асинхронном режиме* посылке очередного байта информации предшествует специальный *старт-бит*, сигнализирующий о начале передачи (обычно логический «0»). Затем следуют биты данных (их обычно 8), за которыми может следовать дополнительный бит (его наличие зависит от режима передачи, обычно этот бит выполняет функцию контроля четности). Завершается посылка *стоп-битом* (логическая «1»), длина которого (длительность единичного состояния линии) может соответствовать длительности передачи 1, 1.5 («полтора стоп-бита») или 2 бит. Стоп-бит гарантирует некоторую выдержку между соседними посылками, при этом пауза между ними может быть сколь угодно долгой (без учета понятия «тайм-аута»). Для асинхронного режима предусмотрен ряд стандартных скоростей обмена: 50, 75, 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600 и 115200 bps.

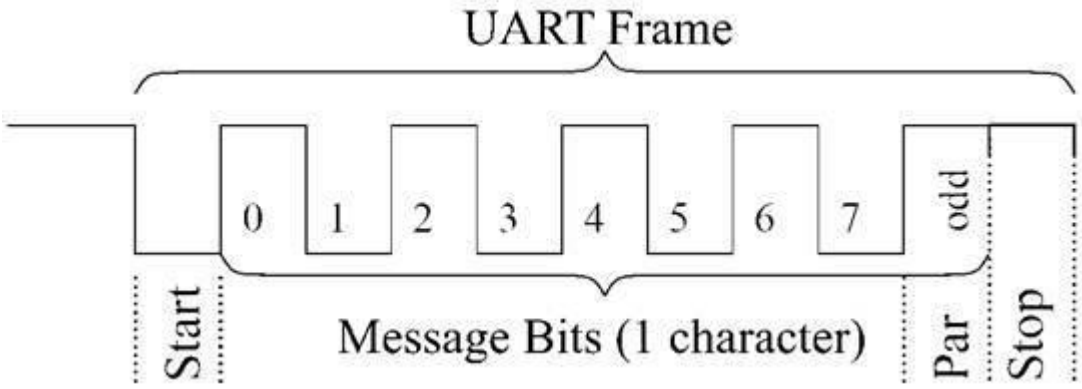

Рис. 14. Временная диаграмма передачи сообщения через UART

Скорость изменения передатчиком состояния сигнальной линии измеряется в бодах (baud) – количестве изменений состояния линии за одну секунду. В простейшем случае в линии имеется всего два состояния сигнала, т.е. одним состоянием кодируется один бит, и тогда скорость изменения состояния в бодах совпадает со скоростью передачи двоичной информации, bps (bit per second, бит/с). Однако, при использовании других методов модуляции возможны несколько состояний сигнала, что позволяет одним состоянием кодировать сразу несколько передаваемых бит, и здесь скорость передачи данных bps превышает скорость изменения сигнала baud.

Интерфейс UART является дуплексным каналом передачи данных. Вообще, обмен данными бывает:

1. Дуплексным – предполагает прием и передачу данных одновременно (UART, RS-232);
2. Полудуплексным – данные передаются в одном направлении с возможностью смены направления (RS-485);
3. Симплексным – данные передаются только в одном направлении.

Передача и прием данных по UART

Сперва, необходимо произвести настройку Pinout в STM32CubeMX:

UART/USART подключается в разделе Connectivity со следующими параметрами:

- Режим (Mode) – асинхронный (asynchronous);
- Скорость (Baud Rate) – 115200 bps;
- Длина сообщения – 8 бит;
- Четность (Parity) – None;
- Стоп-бит – 1;
- Направление передачи (Data direction) – Receive and Transmit.

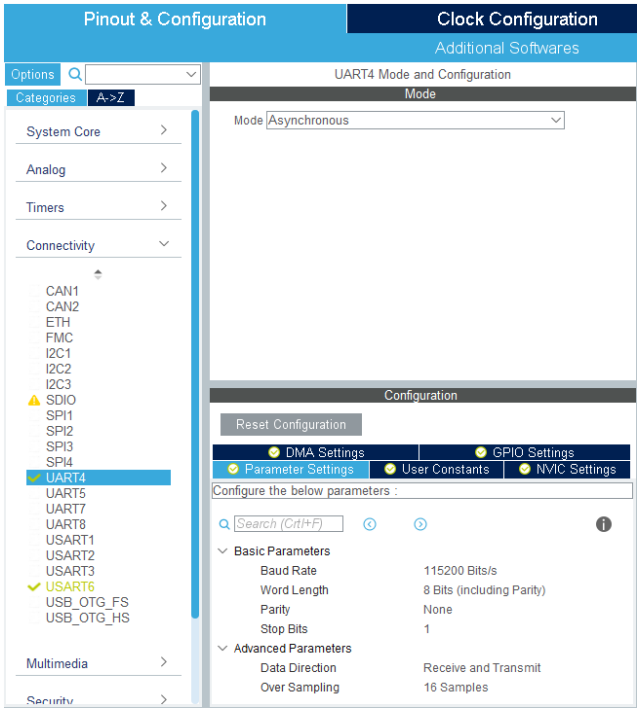

Рис. 15. Окно настройки pinout UART

Для работы с UART на микроконтроллерах STM32 используется библиотека HAL (Hardware Abstract Layer). STM32CubeMX при генерации проекта добавляет в код функцию инициализации UART MX_UART4_Init().

```
static void MX_UART4_Init(void)
{
  huart4.Instance = UART4;
  huart4.Init.BaudRate = 115200;
  huart4.Init.WordLength = UART_WORDLENGTH_8B;
  huart4.Init.StopBits = UART_STOPBITS_1;
  huart4.Init.Parity = UART_PARITY_NONE;
  huart4.Init.Mode = UART_MODE_TX_RX;
  huart4.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart4.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart4) != HAL_OK)
  {
    Error_Handler();
  }
```

В библиотеке HAL существует две стандартных функции, отвечающих за прием и передачу информации:

HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout);

HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout);

- `*huart` – указатель на структуру UART_HandleTypeDef, которая содержит информацию о конфигурации для указанного модуля UART;
- `*pData` – указатель на буфер данных;
- `Size` – размер данных которые будут переданы или приняты;
- `Timeout` – длительность времени ожидания.

Варианты заданий

Для всех вариантов реализуется два варианта программы: с использованием прерываний и без.

Драйвер UART должен каждый принимаемый контроллером символ отсылать назад – так называемое «эхо». Каждое новое сообщение от стенда должно выводиться с новой строки.

Связь между стендом и персональным компьютером (ПК) осуществляется с использованием терминальной программы – например, putty. Физически связь между ПК и стендом осуществляется через тот же кабель USB, через который происходит прошивка и отладка – со стороны SDK находится микросхема, которая управляется с помощью USB и формирует последовательные сигналы UART, которые подаются на входы контроллера.

Вариант 1.

Доработать программу «светофор», добавив возможность отключения кнопки и задание величины таймаута (период, в течение которого горит красный).

Должны обрабатываться следующие команды, посылаемые через UART:

- *?* – в ответ контроллер должен прислать состояние, которое отображается в данный момент на диодах: *red, yellow, green, blinking green,* режим – *mode 1* или *mode 2,* величину таймаута (сколько горит красный) – *timeout …*, и задействованы ли прерывания или нет – символ *I* или *P*;
- *set mode 1* или *set mode 2* – установить режим работы светофора, когда обрабатываются или игнорируются нажатия кнопки;
- *set timeout X* – установить таймаут;
- *set interrupts on* или *set interrupts off* – включить или выключить прерывания в драйвере.

Вариант 2.

Доработать программу «гирлянда», включив возможность добавления четырёх новых последовательностей миганий светодиодов, и задающейся для каждой последовательности индивидуально частоты переключения состояний. Каждая вводимая последовательность должна иметь от двух до восьми состояний. В один момент времени может гореть только один диод (или не гореть ни один). Смена отображаемой в данный момент последовательности должна осуществляться нажатием кнопки или командой, посылаемой через UART.

Должны обрабатываться следующие команды, посылаемые через UART:

- new xx – ввести новую последовательность, где x – это одна из букв g, r, y, n; «g» соответствует включению зелёного светодиода, «r» - красного, «y» - жёлтого, «n» означает, что ни один светодиод не горит; количество вводимых значений может быть от двух до восьми, ввод завершается либо по нажатию enter, либо после ввода восьми значений; после окончания ввода последовательности мерцаний стенд должен послать сообщение произвольного содержания, приглашающее ввести частоту мерцаний светодиодов (должны предусматриваться минимум три градации); ввод частоты мерцаний заканчивается по нажатию *enter*;
- *set x* – сделать активной последовательность мерцаний *x*, где *x* – порядковый номер;
- *set interrupts on* или *set interrupts off* – включить или выключить прерывания в драйвере.

Вариант 3.

Доработать программу, посылающую сигналы азбуки Морзе для латинского алфавита. Последовательность точек и тире, полученных нажатием кнопки стенда, должна дешифроваться и отправляться через последовательный канал.

Символы, получаемые стендом через последовательный канал, должны шифроваться и «проигрываться» с помощью светодиода. Должна быть возможность ввода до восьми символов – они должны запоминаться стендом и проигрываться последовательно. Если в момент «мигания» приходят новые символы – они добавляются в очередь.

Считывание нажатий кнопки и отправка дешифрованных символов должна функционировать одновременно с приёмом через последовательный канал и миганием светодиода.

Включение/выключение прерываний должно осуществляться отправкой с ПК символа «+».

Вариант 4.

Доработать программу счётчика до калькулятора.

Ввод значений производится через последовательный порт из терминальной программы, запущенной на компьютере: *xxxxxyxxxxx*=, где x – десятичные цифры, y – знак (+, -, *, /). Ввод чисел завершается либо знаком операции (для первого числа), либо знаком равно (для второго числа), либо после ввода пяти цифр.

Размерность результата и обоих операндов должна быть short int, и должна быть предусмотрена защита от переполнения. В случае выполнения недопустимых операций (ответ или вводимые числа больше, чем размер переменных в памяти) должен загораться красный диод, а в последовательный канал вместо ответа выводиться слово *error*.

Включение/отключение прерываний должно осуществляться нажатием кнопки на SDK и сопровождаться отправкой в последовательный порт сообщения произвольного содержания, сообщающего, какой режим (с прерываниями или без) включен.

Вариант 5.

Доработать программу кодового замка.

После ввода единственно верной последовательности из не более чем восьми латинских букв без учёта регистра и цифр должен загореться зелёный светодиод, индицирующий «открытие» замка. Светодиод горит некоторое время, потом гаснет и система вновь переходит в «режим ввода». Каждый неправильно введённый элемент последовательности должен сопровождаться миганием красного светодиода и сбросом в «начало», каждый правильный – миганием жёлтого. После трёх неправильных вводов начинает мигать красный диод, и через некоторое время система вновь возвращается в «режим ввода».

Должно быть предусмотрено изменение отпирающей последовательности, что производится следующей последовательностью действий:

- ввод символа +;
- ввод новой последовательности, который завершается либо по нажатию enter, либо по достижению восьми значений;
- стенд отправляет сообщение произвольного содержания, спрашивая сделать ли последовательность активной, и запрашивает подтверждение вводом символа *y*;
- после ввода *y* введённая последовательность устанавливается как активная.

Включение/отключение прерываний должно осуществляться нажатием кнопки на SDK и сопровождаться отправкой в последовательный порт сообщения произвольного содержания, сообщающего, какой режим (с прерываниями или без) включен.

Лабораторная работа, «I²C и OLED дисплей»

Модуль OLED, общие сведения

Дисплейный модуль WEO012865D имеет диагональ 0.96 дюйма и разрешение 128x64 точек. За работу модуля отвечает микросхема SSD1306BZ, обменивающаяся данными с микропроцессором через интерфейс I²C.

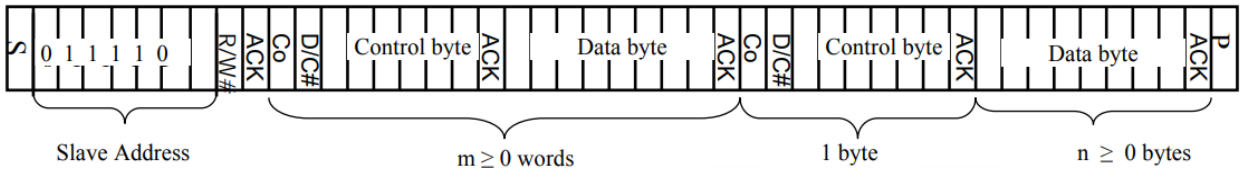

Рис. 16. Формат данных, передающихся через шину I²C.

Co – бит продолжения;

D/C# – бит выбор данных или команды;

ACK – подверженнее;

SA0 – бит адреса подчиненного устройства;

R/W# – бит выбора чтения или записи;

S – начальное состояние;

P – конечное состояние;

Режим записи I²C:

1. Ведущее устройство инициализирует передачу данных через состояние Start. Это состояние достигается сменой SDA с 1 в 0, пока SCL остается 1. (Рис. 4).
2. Адрес подчиненного устройства соответствует состоянию Start. Адрес может быть «b0111100» или «b0111101» при изменении SA0 с 0 на 1.
3. Режим записи устанавливается путем установки бита R/W в логический 0.
4. Сигнал подтверждения будет сгенерирован после получения одного байта данных, включая ведомый адрес и бит R/W#. (Рис. 3).
5. После передачи адреса подчиненного устройства управляющий байт или байт данных могут быть отправлены через SDA. Управляющий байт состоит из битов Co и D/C#, за которыми следуют шесть нулей.
    a. Если бит Co установлен как логический «0», передача следующей информации будет содержать только байты данных.
    b. Если бит D/C# установлен на логической «0», он определяет следующий байт данных как команду. Если бит D/C # установлен в логическую «1» определяет следующий байт данных как данные, которые будут храниться в GDDRAM. Указатель адреса столбца GDDRAM будет увеличиваться на единицу автоматически после каждой записи данных.
6. Бит подтверждения будет генерироваться после получения каждого управляющего байта или байта данных.
7. Режим записи будет завершен, когда будет применено состояние Stop. Это состояние достигается сменой SDA из 0 в 1, пока SCL остается 1. (Рис. 4).

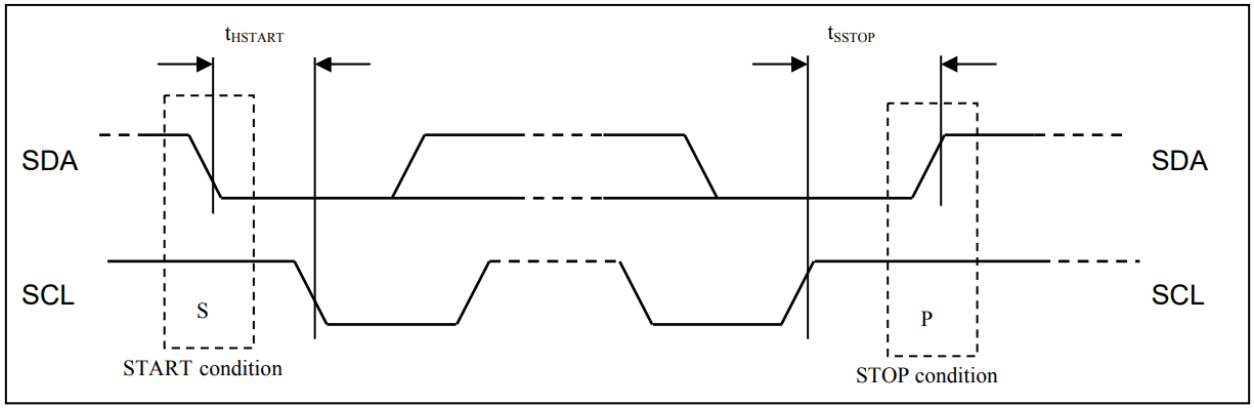

Рис. 17. Определение состояний Start и Stop.

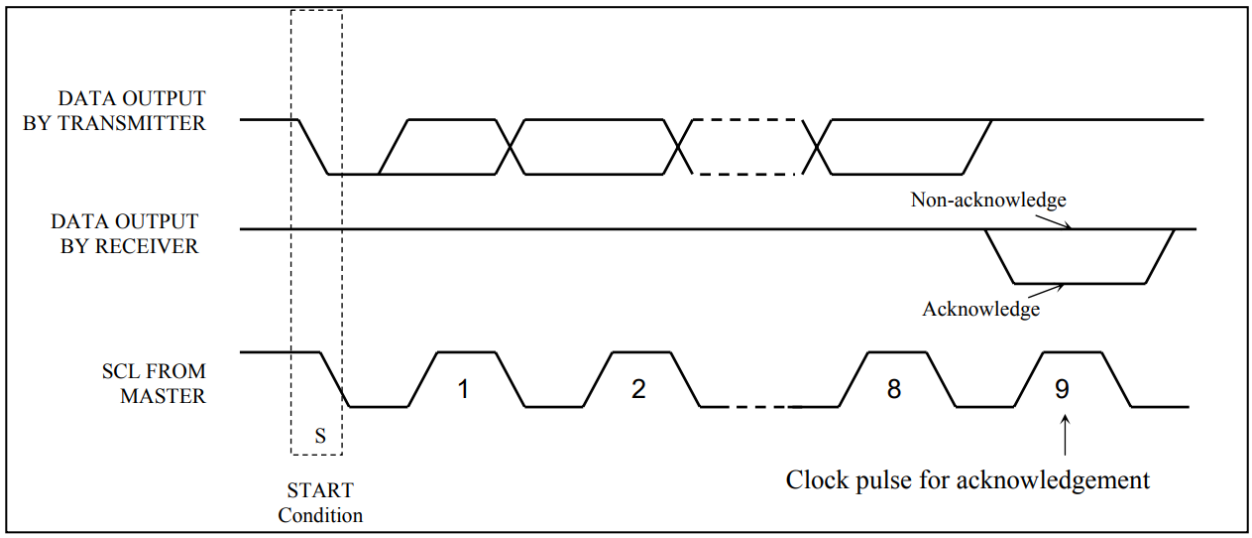

Рис. 18. Определение состояния подтверждения.

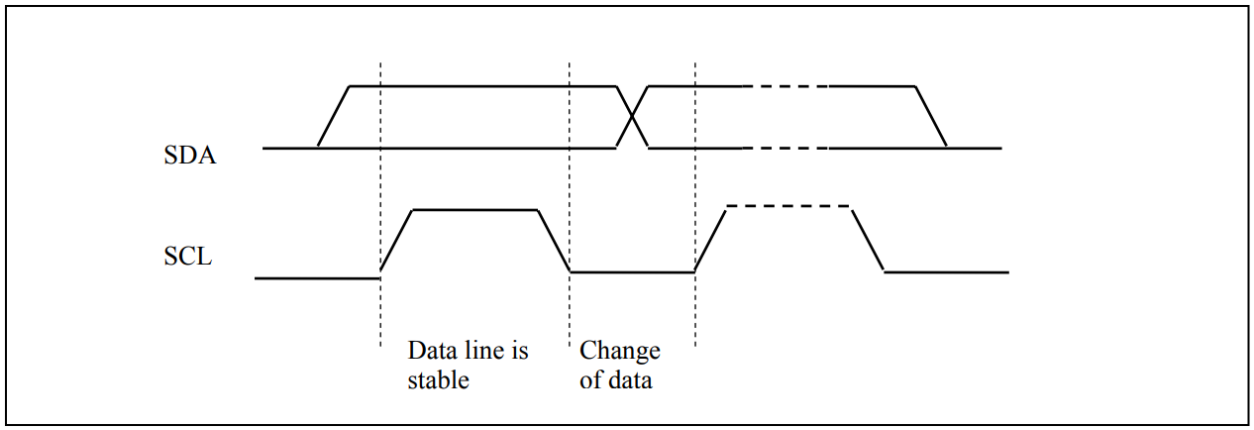

Рис. 19. Определение состояния передачи данных.

GDDRAM – это статическое ОЗУ с битовой привязкой, содержащее отображаемый битовый массив. Размер оперативной памяти составляет 128x64 бита, и ОЗУ разделено на восемь

страниц, от PAGE0 до PAGE7, которые используются для монохромного матричного дисплея 128x64. (Рис. 7).

Когда один байт данных записывается в GDDRAM, все данные изображения строк на одной и той же странице текущего столбца заполняются (то есть заполняется весь столбец (8 бит), на который указывает указатель адреса столбца). Бит данных D0 записывается в верхний ряд, а бит данных D7 записывается в нижний ряд.

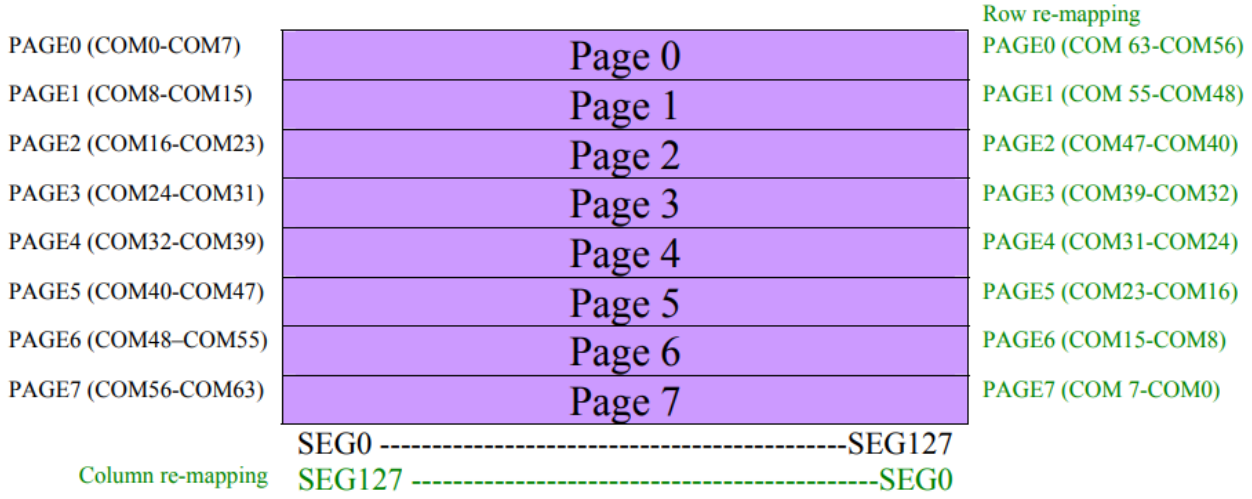

Рис. 20. Структура страниц GDDRAM.

Описание команд

1. Установить начальный адрес нижнего столбца для режима адресации страницы (00h~0Fh).
2. Установить более высокий начальный адрес столбца для режима адресации страницы (10h~1Fh).
3. Установить режим адресации памяти (20h).
4. Установить адрес столбца (21h).
5. Установить адрес страницы (22h).
6. Установить начальную строку дисплея (40h~7Fh).
7. Установить контрастный контроль для BANK0 (81h).
8. Установить повторное отображение сегмента (A0h/A1h).
9. Включить весь дисплей (A4h/A5h).
10. Установить нормальный/инвертированный режим дисплея (A6h/A7h).
11. Установить мультиплексное соотношение (A8h).
12. Установить состояние дисплея Вкл/Выкл (AEh/AFh).
13. Установить начальный адрес для режима адресации страницы (B0h~B7h).
14. Установить направление сканирования COM-выхода (C0h/C8h).
15. Установить смещение дисплея (D3h).
16. Установить частоту деления дисплея/частоту генератора (D5h).

17. Установить период предварительной зарядки (D9h).
18. Настройка конфигурации аппаратного обеспечения COM-выводов (DAh).

Функции для работы с дисплеем

- **SetCursor()** – установка курсора по заданным координатам.
- **DrawPixel()** – выведение на экран пикселя белого или черного цвета по заданным координатам.
- **WriteChar()** – выведение символа на дисплей. Необходимо создать библиотеку шрифтов, где каждый символ представлен в виде последовательности байтов.

  Пример:

  Символ «S» 7x10 будет выглядеть так: 0x38, 0x44, 0x40, 0x30, 0x08, 0x04, 0x44, 0x38, 0x00, 0x00.

  00**111**000 0x38
  0**1**000**1**00 0x44
  0**1**000000 0x40
  00**11**0000 0x30
  0000**1**000 0x08
  00000**1**00 0x04
  0**1**000**1**00 0x44
  00**111**000 0x38
  00000000 0x00
  00000000 0x00

- **WriteString()** – выведение строки на дисплей.
- **Fill()** – заполняет дисплей одним цветом (белым или черным).
- **UpdateScreen()** – записывает все изменения в память и выводит на экран.
- **Init()** – функция инициализации дисплея, в память контроллера отправляется конфигурация (0xAE, 0xA6, 0x20, 0x00, 0x21, 0x00, 0x7F, 0x22, 0x00, 0x07, 0xA1, 0xC8, 0xA8, 0x3F, 0x81, 0x7F, 0x8D, 0x14, 0xAF) по адресу **0x00**, параметры курсора устанавливаются на 0.

Варианты заданий

Вариант 1.

Написать программу, реализующую монитор состояний светодиодов. При отправке по UART со стороны персонального компьютера символа «1» меняется состояние зеленого светодиода, «2» - желтого, «3» - красного. Выводить на дисплей информацию о состояниях всех светодиодов. Например: горит только желтый светодиод, на дисплее будет выведено:

```
Green off
Yellow on
Red off
```

Вариант 2.

Написать программу, реализующую функции простого текстового редактора. Необходиом выводить на дисплей символы с клавиатуры, отправляемые по UART согласно таблице ASCII, при нажатии клавиши “Backspace” один символ стирается.

Вариант 3.

Написать программу, реализующую калькулятор с операциями сложения, вычитания, умножения и деления. С клавиатуры персонального компьютера по UART отправляется выражение, которое выводится на дисплей, при нажатии кнопки “=” на дисплее остается только ответ. Клавиша “Backspace” очищает дисплей.

Вариант 4.

Написать программу, реализующую функции электронного секундомера. Точность измерения времени – десятые доли секунды. Управление производится посредством двух кнопок программируемой клавиатуры SDK-1.1M (Старт/Стоп и Сброс).

Вариант 5.

Написать программу, рисующую на дисплее различные геометрические фигуры: квадрат, круг или треугольник, в зависимости от посылаемого с персонального компьютера через UART символа (“s”, “c”, “t”).

Вариант 6.

Написать программу, реализующую функции электронных часов. На дисплее должны отображаться часы, минуты и секунды.

Лабораторная работа, «Клавиатура»

Описание клавиатуры

Клавиатура SDK-1.1M организована в виде матрицы 4x3. Связь с процессором осуществляется через интерфейс I^{2}C. Строки и столбцы клавиатуры подключены к ногам 8-битной микросхемы PCA9538 (рис. 8). К портам P0-P3 подключены ряды, к P4, P5, P6 подключены колонки. Доступ к колонкам и рядам организован как чтение/запись определенного байта памяти, где каждый бит отвечает за определенную ногу микросхемы PCA9538.

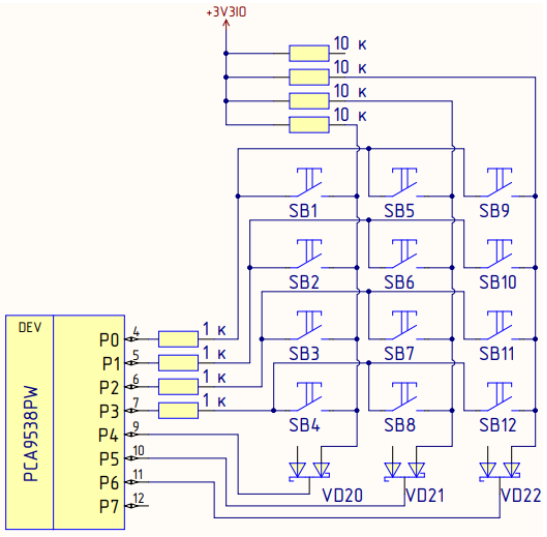

Рис. 21. Схема подключения кнопок к PCA9538.

Работа с PCA9538 по I^{2}C

Последний бит адреса подчиненного устройства определяет операцию (чтение или запись), которая должна быть выполнена. Low (L) означает операцию записи, High (H) означает операцию чтения (таблица 1).

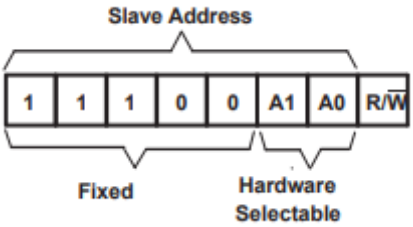

Рис. 22. Адрес подчиненного устройства.

Таблица. 4. Варианты I^{2}C адресов.

INPUTS		I^{2}C BUS SLAVE ADDRESS
A1	A0	
L	L	112 (decimal), 70 (hexadecimal)
L	H	113 (decimal), 71 (hexadecimal)
H	L	114 (decimal), 72 (hexadecimal)
H	H	115 (decimal), 73 (hexadecimal)

После успешного подтверждения байта адреса ведущее устройство шины отправляет командный байт (таблица 2), который хранится в контрольном регистре в PCA9538. Два бита этого командного байта определяют операцию (чтение или запись) и внутренний регистр (вход, выход, инверсия полярности или конфигурация), которые будут затронуты. Этот регистр можно записать или прочитать через шину I^2C. Байт команды отправляется только во время передачи записи. После того, как командный байт был отправлен, адрес, к которому был адресован регистр, продолжает обращаться к операциям чтения до тех пор, пока не будет отправлен новый байт команды.

Таблица. 5. Командный байт.

CONTROL REGISTER BITS		COMMAND BYTE (HEX)	REGISTER	PROTOCOL	POWER-UP DEFAULT
B1	B0				
0	0	0x00	Input Port	Read byte	XXXX XXXX
0	1	0x01	Output Port	Read/write byte	1111 1111
1	0	0x02	Polarity Inversion	Read/write byte	0000 0000
1	1	0x03	Configuration	Read/write byte	1111 1111

Описание регистров

Регистр входного порта (регистр 0) отражает входящие логические уровни выводов, независимо от того, определен ли вывод как вход или выход регистром конфигурации. Действует только на операцию чтения. Записи в эти регистры не имеют никакого эффекта. Значение по умолчанию X определяется применяемым извне логическим уровнем. Перед операцией чтения передается запись с командным байтом, чтобы указать устройству I^2C, что в следующий раз осуществляется доступ к регистру входного порта.

Таблица. 6. Входной регистр.

Бит	**I7**	**I6**	**I5**	**I4**	**I3**	**I2**	**I1**	**I0**
Стандартное значение	X	X	X	X	X	X	X	X

Регистр выходного порта (регистр 1) показывает исходящие логические уровни выводов, определенных как выходы регистром конфигурации. Битовые значения в этом регистре не влияют на выводы, определенные как входы. В свою очередь, чтения из этого регистра отражают значение, которое находится в триггере, управляющем выбором выхода, а не фактическое значение входа.

Таблица. 7. Выходной регистр.

Бит	**O7**	**O6**	**O5**	**O4**	**O3**	**O2**	**O1**	**O0**
Стандартное значение	1	1	1	1	1	1	1	1

Регистр инверсии полярности (регистр 2) позволяет инвертировать полярность выводов, определенных как входы регистра конфигурации. Если бит в этом регистре установлен (записан с 1), соответствующая полярность контакта порта инвертирована. Если бит в этом регистре очищается (записывается с 0), исходная полярность соответствующего контакта порта сохраняется.

Таблица. 8. Регистр инверсии полярности.

Бит	**N7**	**N6**	**N5**	**N4**	**N3**	**N2**	**N1**	**N0**
Стандартное значение	0	0	0	0	0	0	0	0

Регистр конфигурации (регистр 3) устанавливает назначения ввода/вывода. Если бит в этом регистре установлен в 1, соответствующий вывод порта активируется как вход. Если бит в этом регистре принимает значение 0, соответствующий вывод порта активируется как выход.

Таблица. 9. Конфигурационный регистр.

Бит	**C7**	**C6**	**C5**	**C4**	**C3**	**C2**	**C1**	**C0**
Стандартное значение	1	1	1	1	1	1	1	1

Запись и чтение данных на PCA9538

Запись:

Данные передаются на PCA9538 путем отправки адреса устройства и установки младшего значащего бита в логический 0. Байт команды отправляется после адреса и определяет, какой регистр получает данные, следующие за байтом команды. Нет ограничений на количество байтов данных, отправленных во время одной передачи.

Чтение:

Сначала ведущее устройство шины должно отправить адрес PCA9538 с младшим значащим битом, установленным на логический 0. Байт команды отправляется после адреса и определяет, к какому регистру обращаются. После перезапуска адрес устройства отправляется снова, но на этот раз младший значащий бит устанавливается на логический 1. После этого данные из регистра, определенного командным байтом, отправляются PCA9538. После перезапуска значение регистра, определенного байтом команды, совпадает с регистром, к которому осуществляется доступ при перезапуске. Данные синхронизируются в регистре по переднему фронту тактового импульса ACK. На количество байтов данных, полученных в одной передаче чтения ограничения нет, но когда получен последний байт, ведущее устройство шины не должно подтверждать данные.

Лабораторная работа, «Прерывания»

Общие сведения

Прерывание – процесс переключения ЦП с одной программы на другую по внешнему сигналу с сохранением информации для последующего возобновления прерванной программы.

Основная цель введения прерываний – реализация асинхронного режима работы и распараллеливания работы отдельных устройств вычислительного комплекса. Механизм прерываний реализуется аппаратно-программными средствами.

Каждое событие, требующее прерывания, сопровождается сигналом прерывания, оповещающим об этом вычислительную машину, и называемым запросом прерывания.

Обработка прерывания выполняется в три этапа:

1. Прекращение выполнения текущей программы.

   Для корректного возврата к выполнению программы после обработки прерывания необходимо сохранить контекст программы – содержимое caller-saved регистров микроконтроллера. Содержимое регистров сохраняется в стек.

2. Переход к выполнению программы обработчик.

   Определяется источник прерывания и соответствующий вектор прерывания. Адрес вектора прерывания записывается в регистр счетчика команд. После чего осуществляется переход к программе-обработчику прерывания и ее выполнение.

3. Возврат управления прерванной программе.

   Для корректного возврата управления необходимо восстановить контекст программы из стека. Последней командой программы обработки прерывания должна быть команда, которая осуществляет возврат в основную программу и восстановление предварительно сохраненного счетчика команд.

Вектор прерывания – вектор начального состояния обработчик прерывания и содержит всю необходимую информацию для перехода к обработчику, в том числе его начальный адрес. Каждому типу прерываний соответствует свой вектор прерывания, который инициализирует выполнение соответствующего обработчика. Обычно векторы прерывания хранятся в специально выделенных фиксированных ячейках памяти с короткими адресами, представляющих собой таблицу векторов прерываний. Для перехода к соответствующей прерывающей программе процессор должен располагать вектором прерывания и адресом этого вектора. По этому адресу, как правило, находится команда безусловного перехода к подпрограмме обработки прерывания.

Приоритеты прерываний – это механизм, позволяющий установить определенный порядок обработки запросов прерываний. При наличии нескольких запросов прерывания, поступивших одновременно, приоритет прерывания будет определять, какой из поступивших запросов будет обработан в первую очередь.

Вложенные прерывания – механизм, позволяющий осуществлять обработку поступившего прерывания в период обработки другого прерывания. В случае, если во время обработки прерывания поступает запрос на прерывание с более высоким уровнем приоритета, управление передается обработчику прерывания более высокого приоритета, при этом работа обработчика прерывания с более низким уровнем приоритета приостанавливается.

Максимальное число программ, которые могут приостанавливать друг друга называется глубиной прерываний.

Задания

Реализовать обработчик прерываний, а также основную программу, которая будет прерываться (мигание светодиодов, вывод текста на дисплей и т.д) в соответствии с вариантом.

Вариант	**Основная программа**	**Обработчик прерывания**	**Источник прерывания**	**Длительность**
1	Переключение светодиодов	Включается красный и зеленый светодиод	Таймер	2с
2	Вывод на дисплей результата арифметической операции	Вывод на экран «Interrupt»	Кнопка	1с
3	Переключение светодиодов	Светодиоды не горят	Dip-переключатель	2с
4	Вывод на дисплей результата арифметической операции	Вывод на экран «Interrupt»	Таймер	2с
5	Переключение светодиодов	Включается желтый и зеленый светодиод	Кнопка	1с
6	Вывод на дисплей результата арифметической операции	Вывод на экран «Interrupt»	Dip-переключатель	2с

Лабораторная работа, «Операционная система реального времени FreeRTOS»

Общие сведения

FreeRTOS – многозадачная операционная система реального времени для встраиваемых систем. За переключение между задачами отвечает диспетчер, работающий по прерыванию от таймера. Задача выглядит как обычная функция Си, которая выполняет некоторое действие в бесконечном цикле. Для STM32 используется CMSIS-RTOS API.

Пример задачи, переключающей состояние светодиода раз 500мс:

```
void Start_ledTask(void const * argument)
{
  for(;;)
  {
        HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_13);
        osDelay(500);
  }
}
```

Так как задачи выполняются асинхронно, то при использовании общей переменной данные могут теряться. Для обеспечения правильного обмена данными между задачами используются очереди. Одни задачи кладут данные в очередь, другие оттуда читают. За переполнением очереди и чтением после окончания записи следит диспетчер. Если очередь пуста/переполнена, то та задача, которая собирается считать/записать туда данные переводится в состояние WAIT и диспетчер выведет ее из этого состояния только когда очередь будет готова принять/отдать данные.

Семафоры бывают двух типов: бинарные и счетные. Работа бинарного семафора очень похода на работу флага. То есть, когда некое событие поднимает флаг, а программа этот флаг отслеживает в цикле. В FreeRTOS функцию этого цикла выполняет диспетчер. Счетный семафор работает по такому же принципу. Разница лишь в том, что у него состояние это не единичный флаг, а некая переменная. Задача, которая выдает семафор эту переменную увеличивает, а которая считывает – уменьшает.

Таблица. 10. Описание API FreeRTOS.

API	Описание
osThreadCreate	Начать выполнение потока
osThreadTerminate	Закончить выполнение потока
osThreadYield	Передвать выполнение другому потоку
osThreadGetId	Получить индефикатор потока для ссылки на него
osThreadSetPriority	Изменить приоритет выполнения потока
osThreadGetPriority	Получить текущий приоритет выполнения потока
osDelay	Подождать в течение указанного времени
osWait	Ожидать события типа Signal, Message, Mail
osTimerCreate	Определить атрибуты Callback таймера
osTimerStart	Запустить таймер с назначением времени
osSignalSet	Установить сигнальные флаги потока
osSignalClear	Сбросить сигнальные флаги потока
osMutexCreate	Инициализация мьютекса

osMutexWait	Получить мьютекс или подождать, пока он не станет доступным
osMutexRelease	Освободить мьютекс
osMutexDelete	Удалить мьютекс
osSemaphoreCreate	Инициализация семафора
osSemaphoreWait	Получить семафор или подождать, пока он не станет доступным
osSemaphoreRelease	Освободить семафор
osSemaphoreDelete	Удалить семафор
osMessageCreate	Инициализация очереди

FreeRTOS и STM32CubeMX

Для того чтобы установить FreeRTOS, необходимо назначить один из таймеров микроконтроллера (TIM1) на генерацию системных тиков. Конфигурация FreeRTOS производится в разделе Middleware. В этом разделе можно конфигурировать:

- Задачи;
- Очереди;
- Семафоры;
- Таймеры;
- Мьютексы.

Лабораторная работа, «Введение в LwIP»

Общие сведения

Учебный стенд SDK-1.1M оборудован разъемом RJ-45. KSZ8081 – это приемопередатчик физического уровня Ethenet для передачи данных через витую пару со скоростью 10/100 Мбит/с. В микроконтроллерах STM32 стек TCP/IP представлен в виде LwIP (Lightweight IP). Стек LwIP предлагает три типа API:

- Raw API;
- Netconn API;
- Socket API.

Архитектура стека LwIP построена на модели, включающей четыре уровня абстракции:

- Прикладной уровень содержит все протоколы для передачи данных между процессами.
- Транспортный уровень обрабатывает соединения между хостами.
- Интернет уровень соединяет независимые сети, устанавливая межсетевое взаимодействие.
- Уровень сетевого интерфейса содержит технологии коммуникации для одного сегмента локальной сети.

Рис. 23. Архитектура стека LwIP.

Raw API

Raw API – это родной интерфейс LwIP. Он позволяет использовать обратные вызовы функций (callbacks) внутри стека. То есть перед началом работы со стеком необходимо присвоить указатели на функции-обработчики событий, которые в процессе работы будут вызываться внутри LwIP.

Таблица. 11. Описание RAW API TCP

Функции API		**Описание**
Установка TCP-соединения	tcp_new	Создает новый TCP PCB (блок управления протоколом)
	tcp_bind	Привязывает TCP PCB к локальному IP-адресу и порту
	tcp_listen	Начало процесса прослушки
	tcp_accept	Назначает функцию обратного вызова, которая будет вызваться при получении нового соединения TCP
	tcp_accepted	Сообщает стеку LwIP, что входящее соединение TCP принято
	tcp_connect	Подключение к удаленному хосту TCP
Отправка данных через TCP	tcp_write	Записывает данные в очередь на отправку
	tcp_sent	Назначает функцию обратного вызова, которая будет вызваться при подтверждении данных удаленным хостом
	tcp_output	Принудительно отправляет данные в очередь
Получение данных через TCP	tcp_recv	Назначает функцию обратного вызова, которая будет вызываться при поступлении новых данных
	tcp_recved	Вызывается, когда приложение обработало входящий пакет данных
Опрос приложений	tcp_poll	Назначает функции обратного вызова, которые будут вызываться периодически. Используется приложением для проверки того, имеются ли оставшиеся данные приложения, которые необходимо отправить, или существуют ли соединения, которые необходимо закрыть
Закрытие и прерывание соединения	tcp_close	Закрывает TCP-соединение с удаленным хостом.
	tcp_err	Назначает функцию обратного вызова для обработки соединений, прерванных LwIP из-за ошибок
	tcp_abort	Прерывает TCP-соединение

Таблица. 12. Описание RAW API UDP

Функции API	Описание
udp_new	Создает новый UDP PCB
udp_remove	Удаляет UDP PCB
udp_bind	Привязывает UDP PCB к локальному IP-адресу и порту
udp_connect	Устанавливает UDP PCB к удаленному IP-адресу и порту
udp_disconnect	Удаляет IP-адрес и порт UDP PCB
udp_send	Отправляет данные через UDP
udp_recv	Определяет функцию обратного вызова, которая вызывается при получении данных

Netconn API

Netconn API – высокоуровневый последовательный интерфейс, разработанный поверх RAW API. Этот интерфейс требует наличия операционной системы реального времени (RTOS) и поддерживает многопоточные операции.

Таблица. 13. Описание Netconn API

Функции API	Описание
netconn_new	Создает новое соединение
netconn_delete	Удаляет существующее соединение
netconn_bind	Привязывает соединение к локальному IP-адресу и порту
netconn_connect	Присоединяется у удаленному IP-адресу и порту
netconn_send	Отправляет данные по подключенному в данный момент удаленному IP-адресу или порту
netconn_recv	Принимает данные от netconn
netconn_listen	Устанавливает TCP-соединение в режим прослушки
netconn_accept	Принимает входящее соединение в режиме прослушки TCP-соединения
netconn_write	Отправляет данные в подключенный TCP netconn

netconn_close	Закрывает TCP-соединение, не удаляя его

Socket API

Socket API – высокоуровневый интерфейс сокетов, разработанный поверх Netconn API. Этот интерфейс обеспечивает высокую переносимость написанных программ, потому что является стандартизированным API.

Таблица. 14. Описание Socket API

Функции API	**Описание**
soket	Создает новый сокет
bind	Связывает сокет с IP-адресом и портом
listen	Прослушать соединения сокетов
connect	Соединяет сокет с удаленным IP-адресом и портом
accept	Принимает новое соединение на сокете
read	Читает данные из сокета
write	Записывает данные в сокет
close	Закрывает сокет (с удалением)

Задание

Необходимо реализовать на SDK-1.1M программу для обмена данными с компьютером, используя один из интерфейсов LwIP.