JX-STM32

Документация на плату микроконтроллера **STM**32 ARM Cortex-M3

www.inexglobal.com

Введение

Отладочные платы STM32 представляют из себя полностью законченные платформы для разработки на основе микроконтроллера **STM32F103 ARM® Cortex™ M3**. Они обеспечивают требуемую эффективность, гибкость, открытость создаваемых решений и идеальную для проверяемых приборов производительность, возможности отладки и периферию. Они состоят из микроконтроллера STM32 с высокопроизводительным ядром ARM Cortex-M3, интерфейса USB 2.0, интерфейса CAN 2.0A/B с широкими возможностями, I^2C, SPI, USART и графического ЖК индикатора.

Платы JX-STM32 включают в себя полную цепь аппаратных средств для выполнения отладки с помощью микроконтроллера и начала разработок в широком ряду применений. Аппаратные средства состоят из SD-платы, GLCD, USB, CAN, I^2C, RS-232 и схемы усилителя звуковой частоты.

Одной из особенностей платы является возможность подключения по стандарту JTAG, позволяющая разработчикам использовать широкий набор средств разработки от сторонних производителей. В дополнение к этому, плата JX-STM32 включает в себя 20-контактный разъем, что является дополнительным инструментом для просмотра временных диаграмм.

Программные средства разработки состоят из средств сторонних производителей, которые представляют совокупность из интегрированной среды разработки и встроенного отладчика/программатора, функционирующего с применением интерфейса JTAG. Разработчики, в первый раз слышащие про этот интерфейс и ядро Cortex™ могут получить информацию о них из ряда начальных руководств, в которых специально созданы справки для разработчиков по особенностям отладки приборов и запуска своих приложений.

Благодаря программно-аппаратным библиотекам от ST и всестороннему описанию возможных применений, микроконтроллеры STM32 позволяют осуществлять полный программный контроль и имеют малое время обработки информации. Великолепная комбинация низкоуровневых эффективных библиотек программных драйверов и всесторонняя поддержка всех распространенных средств разработки предлагает быстрое освоение и оптимизирует процесс разработки.

Список примеров для STM32F103VBT6 в наборе средств разработки Raisonance Rkit-ARM7

Основной каталог

`C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples`

ADC (аналого-цифровой преобразователь)

`C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\ADC`

Большинство примеров может работать с платой JX-STM32 (5 примеров) за исключением 3ADCs_DMA.

Резервный регистр

`C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\BKP`

Все примеры могут работать с платой JX-STM32 (2 примера)

CAN (модуль управления локальной сетью)

`C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\CAN`

Все примеры могут работать с платой JX-STM32 (1 пример)

Процессор Cortex-M3

`C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\CortexM3`

Все примеры могут работать с платой JX-STM32 (2 примера)

DMA (прямой доступ к памяти)

`C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\DMA`

Большинство примеров может работать с платой JX-STM32 (4 примера) за исключением FSMC.

EXTI (внешнее прерывание)

`C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\EXTI`

Все примеры могут работать с платой JX-STM32 (1 пример)

Flash память

`C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\FLASH`

Все примеры могут работать с платой JX-STM32 (2 примера)

GPIO (основное устройство ввода-вывода)

`C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\GPIO`

Все примеры могут работать с платой JX-STM32 (2 примера)

I²C (общая шина)

C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\I2C

Большинство примеров может работать с платой JX-STM32 (4 примера) за исключением M24C08_EEPROM.

IWDG (независимый сторожевой таймер)

C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\IWDG

Все примеры могут работать с платой JX-STM32 (1 пример)

LIB_Debug (библиотека отладки)

C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\DEBUG

Все примеры могут работать с платой JX-STM32 (1 пример)

NVIC (контроллер вектора прерываний)

C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\NVIC

Все примеры могут работать с платой JX-STM32 (6 примеров)

Работа в режиме пониженного энергопотребления

C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\PWR

Все примеры могут работать с платой JX-STM32 (2 примера)

RCC (Real Counter Controller)

C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\RCC

JX-STM32 может выполнить этот пример только после удаления (отключения) 8 Мгц генератора на плате. Требуется хорошо уметь паять для удаления генератора на плате микроконтроллера.

RTC (модуль генератора реального времени)

C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\RTC

Только пример Calender может работать с платой JX-STM32 (1 пример)

SPI (Последовательный периферийный интерфейс)

C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\SPI

Большинство примеров может работать с платой JX-STM32 (5 примеров) за исключением M25P64_FLASH при использовании контактов порта SPI, которые подключены к разъему SD-платы.

SysTick (Таймер системного времени)

C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\SysTick

Все примеры могут работать с платой JX-STM32 (1 пример)

TIM (Основной таймер)

C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\TIM

Все примеры могут работать с платой JX-STM32 (15 примеров). Требуется осциллограф для измерений формы сигнала.

USART (Универсальный синхронный/асинхронный приемопередатчик)

`C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\UART`

Большинство примеров может работать с платой JX-STM32 (9 примеров) за исключением IRDA, Smartcard и Synchronous.

WWDG (Оконный сторожевой таймер)

`C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples\WWDG`

Все примеры могут работать с платой JX-STM32 (1 пример)

STM32

1: STM32F103BVT6 — ARM Cortex-M3 микроконтроллер фирмы STMicroelectronics

1.1 Технические параметры

- Микропроцессор: 32-битный, ARM Cortex-M3 с максимальной тактовой частотой 72 Мгц с одноцикловым умножением и аппаратным делением;
- Память: Flash 128 Кб и статическое ОЗУ (SRAM) 20 Кб;
- Тактовая частота: от 4 до 16 Мгц во внутреннем генераторе, 8 Мгц во внутреннем RC-генераторе, 40 кГц во внутреннем RC-генераторе, 32 кГц — внутренняя частота модуля реального времени и фазовой автоподстройки частоты при тактовой частоте процессора 72 Мгц;
- Напряжение питания: от 2.0 до 3.6 В;
- Управление питанием: сброс при повышении питания (POR), сброс при понижении питания (PDR), программируемый детектор напряжения (PVD). Поддерживаются 3 записываемых режима управления питанием: Sleep (Сон), Stop (Останов), Standby (Резервный);
- Напряжение питания V_{BAT} для модуля тактового генератора реального времени и резервного регистра;
- АЦП: скорость 1 мс, преобразование от 2 до 12 бит, 16 каналов. Поддерживаются входные напряжения от 0 до 3.6 В. Две простые схемы захвата (удержания) и датчик температуры;
- ПДП (DMA): 7 каналов. Поддерживаются таймер, АЦП, SPI, I^2C шина и модуль синхронного/асинхронного приемопередатчика (USART);
- Основное устройство вода/вывода (GPIO): 80-контактный высокоскоростной с максимально разрешенным напряжением + 5 В;
- Отладка: Поддерживается отладка по последовательному каналу (SWD) и JTAG;
- Таймер: 7 элементов:
  - до трех 16-битных таймеров, каждый содержит до 4-х IC/OC/PWM или счетчиков импульсов;
  - 16-битный, 6-канальный таймер с расширенным управлением: до 6 каналов PWM-выхода, генерация по завершению временного интервала и аварийный останов;
  - 2 сторожевых таймера (независимый и оконный);
  - таймер системного времени (SysTick): 24-битный счетчик по спаду импульсов;
- Последовательный интерфейсный модуль: 9 коммуникационных интерфейсов

– 2 интерфейса I^2C шины: поддерживаются SMBus и PMBus;

– 3 модуля синхронного/асинхронного приемопередатчика (USART): поддерживаются ISO 7816, LIN, IrDA и модемные управляющие сигналы;

– 2 модуля последовательного периферийного интерфейса (SPI), скорость 18 Мбит/секунду;

– Контроллер локальной сети (CAN 2.0B активный тип);

– Универсальная последовательная шина (USB 2.0);

Более подробная информация доступна на сайте ***http://www.st.com***

1.2 Архитектура STM32F103VBT6

На рисунке 1-1 показана структурная схема микроконтроллера STM32F103VBT6.

1.2.1 Внутренняя память

STM32F103VBT6 содержит 128 кБайт внутренней Flash-памяти, доступной для хранения программ и данных, и внутреннюю статическую ОЗУ (SRAM). 20 кБайт этой ОЗУ доступны для чтения/записи на тактовой частоте процессора с 0 состояниями ожидания. На рисунке 1-2 показана карта памяти STM32F103VBT6.

1.2.2 Контроллер вектора прерываний (NVIC)

Линейка микроконтроллеров STM32F103xx включает в себя контроллер вектора прерываний, состоящий из 43 маскируемых прерываний (в это число не включено 16 прерываний ядра Cortex-M3) и 16 уровней приоритетов.

- Двусторонняя изоляция контроллера вектора прерываний (NVIC) обеспечивает малую задержку при обработке прерывания;
- Таблица адресов вектора внутренних прерываний располагается прямо в ядре;
- Двусторонняя изоляция контроллера вектора прерываний (NVIC) является интерфейсом ядра;
- Позволяется досрочная обработка прерываний;
- Обработка позже пришедшего прерывания имеет больший приоритет;
- Поддержка вложенных прерываний (tail-chaining);
- Состояние процессора автоматически сохраняется;
- Внутреннее прерывание сохраняется как первое внешнее (без дополнительных инструкций сверху);

Этот аппаратный блок обеспечивает широкие возможности по управлению прерываниями с минимальными задержками.

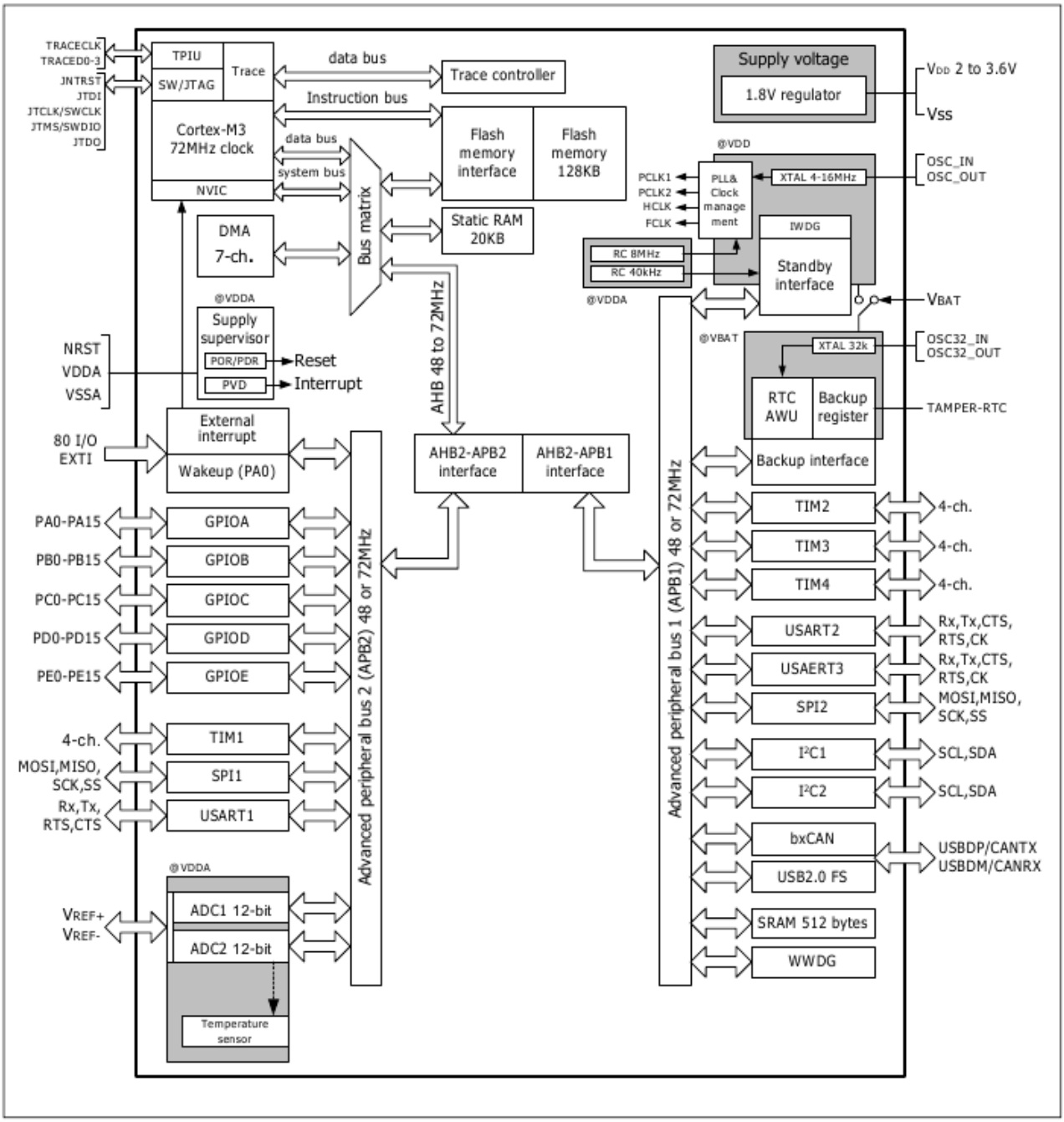


Рисунок 1-1. Структурная схема STM32F103VBT6

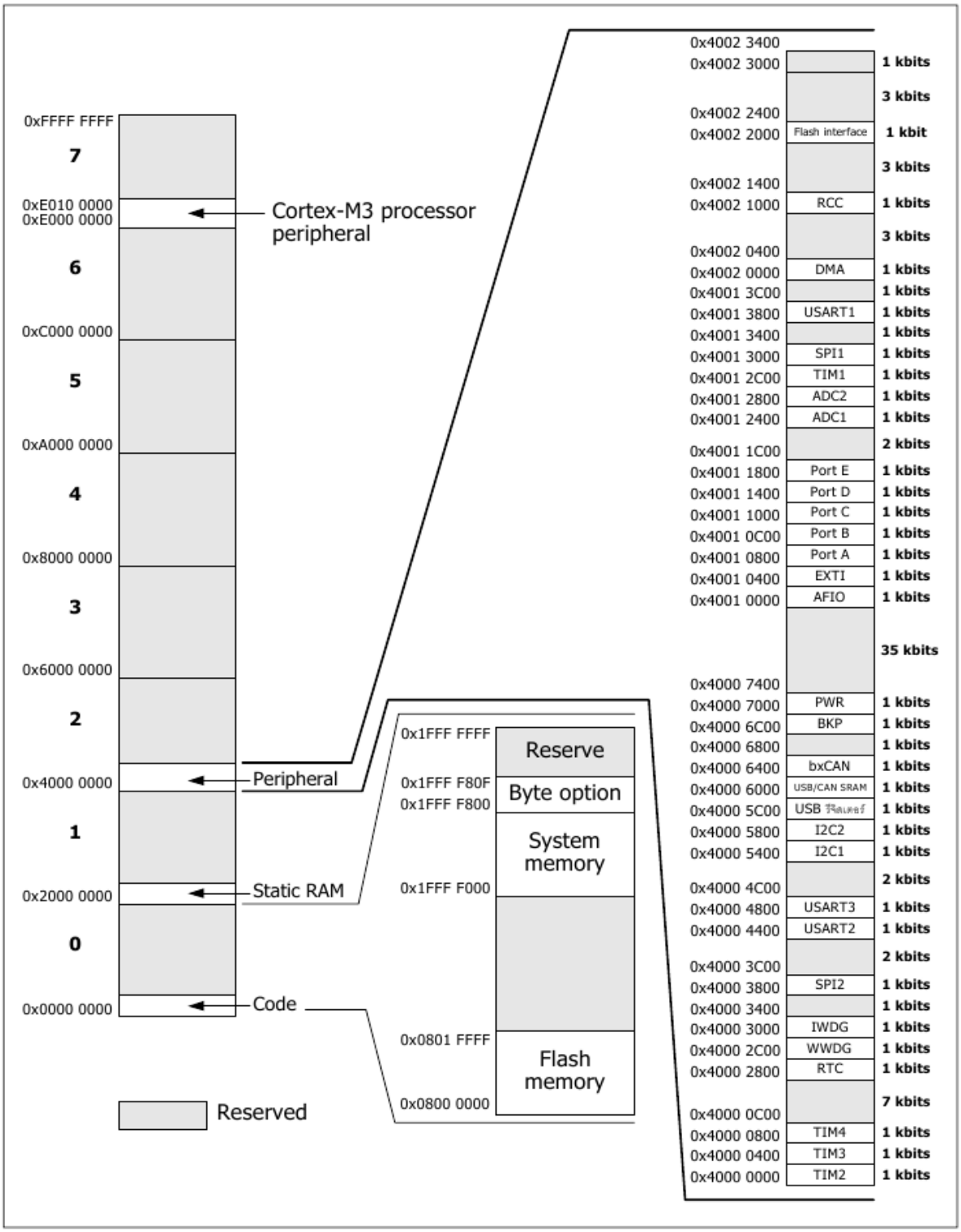


Рисунок 1-2. Карта памяти STM32F103VBT6

1.2.3 Контроллер внешних прерываний/событий (EXTI)

Контроллер внешних прерываний/событий состоит из 16 высокоточных цепей обнаружения, используемых для установки требований возникновения внешних прерываний. Каждая цепь независимо настраивается для выделения импульса (по фронту, по спаду, по фронту и спаду) и может быть независимо маскирован. Ожидающий регистр сохраняет уровень, требующийся для возникновения прерывания. Контроллер внешних прерываний/событий может определить импульс во внешней цепи с длительностью, меньшей чем период внутреннего APB2 тактового генератора. Более 80 портов ввода/вывода может быть подключено к 16 цепям обработки внешних прерываний.

1.2.4 Тактовые генераторы и запуск

Определение системного тактового генератора выполняется при запуске, однако внутренний RC-генератор 8 Мгц определяется как тактовый генератор по умолчанию при сбросе. Внешние генераторы от 4 до 16 Мгц могут быть определены, но считаться неисправными. При возникновении такой ситуации, они должны быть отключены, после чего запускается программное управление через прерывания. Подобным образом, полное управление через прерывания входом системы фазовой автоподстройки частоты (PLL) тактового генератора доступно по необходимости (например при невозможности использования внешнего генератора, резонатора или гетеродина).

Максимальная частота AHB и высокоскоростного ABP достигает 72 Мгц. Максимально допустимая частота низкоскоростного APB составляет 36 Мгц.

1.2.5 Режимы загрузки

При запуске, контакты загрузки используются для выбора одной из трех опций:

- Загрузка из Flash-памяти;
- Загрузка из системной памяти;
- Загрузка из статического ОЗУ;

Загрузчик находится в системной памяти. Он используется для перепрошивки Flash-памяти с использованием первого синхронного/асинхронного передатчика (USART1).

1.2.6 Напряжение питания

Контроллеру STM32F103VBT6 требуется главное напряжение питания (V_{DD}) и напряжение питания аналогового модуля (V_{DDA}) от 2 до 3.6 В для работы. Оно существует как поддерживающее напряжение для модуля тактового генератора реального времени (V_{BAT}) - от 1.8 до 3.6 В от внешней батареи.

1.2.7 Стабилизатор напряжения питания

Это устройство состоит из схемы сброса при повышении питания (POR) и схемы сброса при понижении питания (PDR). Они всегда включены и обеспечивают правильную работу при питании выше/ниже 2 В. Устройство переходит в режим сброса когда V_{DD} опускается ниже установленного порога $V_{por/pdr}$ без необходимости во внешней схеме сброса.

Особенность устройства — встроенный программируемый измеритель напряжения (PVD), который измеряет напряжение питания V_{DD} и сравнивает его

с порогом V_{PVD}. Прерывание происходит когда V_{DD} падает ниже V_{PVD} и/или когда V_{DD} больше порога V_{PVD}. Схема обработки прерываний как обычно генерирует предупреждающее сообщение и/или устанавливает MCU в защищенный режим. Измеритель напряжения (PVD) включается программно.

1.2.8 Регулятор напряжения

Регулятор напряжения может работать в трех режимах: основном (MR), пониженного напряжения питания (LPR) и режиме выключения питания.

- Основной режим (MR) используется при номинальном режиме работы (Run);

- Режим пониженного напряжения питания (LPR) используется в режиме останова (Stop);

- Режим выключения питания используется в резервном (Standby) режиме: выход регулятора имеет высокое сопротивление: схема основной защиты отключает питание, включая нулевое потребление (содержимое регистров и ОЗУ не сохраняется);

Этот регулятор всегда включается после сброса. Он отключается в резервном (Standby) режиме, обеспечивая высокое сопротивление на своем выходе.

1.2.9 Режимы пониженного энергопотребления

В линейке микроконтроллеров STM32F103xx поддерживаются три режима пониженного энергопотребления для достижения наилучшего соотношения между малым энергопотреблением, коротким временем запуска и доступными способами запуска:

- **Сон (Sleep)**: В режиме сна останавливается только процессор (CPU). Вся периферия продолжает работу и может запустить процессор при возникновении соответствующего прерывания/события;

- **Останов (Stop)**: Режим останова позволяет добиться наименьшего энергопотребления пока сохраняется содержимое ОЗУ и регистров. Все тактовые генераторы потребляющие напряжение 1.8 В останавливаются, система фазовой автоподстройки частоты (PLL), HSI и HSE RC-генераторы отключаются. Регулятор напряжения при этом может находиться в нормальном или пониженном энергопотреблении ;

Устройство может переключиться из режима останова с помощью одной из линий контроллера внешних прерываний (EXTI). Контроллер внешних прерываний может реагировать на одну из 16 внешних цепей, выход программируемого измерителя напряжения (PVD), сигнал с генератора реального времени (RTC) или включение USB.

- **Резервный (Standby)**: Резервный режим позволяет достичь наименьшего энергопотребления. Внутренний регулятор напряжения отключен чтобы полностью исключить потребление напряжения 1.8 В. Система фазовой автоподстройки частоты (PLL), HSI и HSE RC-генераторы также отключены. После входа в резервный режим, содержимое ОЗУ и регистров уничтожается за исключением резервных регистров и защиты резервного режима;

Устройство находится в резервном режиме при внешнем сбросе (контакт NRST), при сбросе от независимого сторожевого таймера (IWDG), по фронту

импульса на контакте WKUP или по сигналу с генератора реального времени (RTC).

1.2.10 Генератор реального времени (RTC) и резервные регистры

Генератор реального времени (RTC) и резервные регистры запитываются через переключатель, на который поступает любое из напряжений V_{DD}, если оно есть, или через контакт V_{BAT}. Резервные регистры (десять 16-битных регистров) используются для сохранения данных, когда V_{DD} отсутствует.

Генератор реального времени (RTC) поддерживает набор постоянно запущенных процессов, которые используются соответствующими программами для использования в функциях определения даты и времени, а также делает возможными сигнальное прерывание и периодическое прерывание. Частота генератора, равная 32.768 кГц, задается внешним кварцем, резонатором или гетеродином, внутренним RC-генератором с низким энергопотреблением или высокочастотным внешним генератором с частотой, кратной 128 Гц. Внутренний RC-генератор с низким энергопотреблением имеет стандартную частоту 40 кГц. Генератор реального времени (RTC) может быть откалиброван с использованием внешнего вывода 512 Гц для компенсации естественной девиации частоты кварца.

Генератор реального времени (RTC) представляет из себя 32-битный программируемый счетчик для долговременной работы, использующий регистр сравнения (Compare register) для генерации сигнала. 20-битный предварительный делитель частоты используется как генератор базовых временных интервалов и по умолчанию настроен на генерацию интервала в 1 секунду с помощью генератора с частотой 32.768 кГц.

1.2.11 Независимый сторожевой таймер (IWDG)

Независимый сторожевой таймер (IWDG) основан на 12-битном уменьшающем свое значение счетчике и 8-битном предварительном делителе частоты. Его частота равна 40 кГц, задается независимым внутренним RC-генератором, и поскольку этот генератор работает независимо от главного генератора, таймер IWDG может работать в режиме останова (Stop) и резервном режиме (Standby). Он может использоваться как таймер сброса устройства в случае возникновения проблем или как свободно запускаемый таймер для создания временных задержек. Он может настраиваться как аппаратно, так и программно через устанавливаемые байты. Счетчик может быть остановлен в режиме отладки.

1.2.12 Оконный сторожевой таймер (WWDG)

Оконный сторожевой таймер основан на 7-битном уменьшающем свое значение счетчике, который может быть установлен как свободно запускаемый. Он может использоваться как сторожевой таймер для сброса устройства при возникновении проблем. Его частота задается главным генератором. Он обладает способностью заранее генерировать предупреждающее прерывание, и счетчик может быть остановлен в режиме отладки.

1.2.13 Таймер системного времени (SysTick)

Этот таймер необходим для OS, но может быть также использован как стандартный понижающий свое значение счетчик. Его особенности:

- 24-битный понижающий свое значение счетчик;

- Способность к автоматической перезагрузке;
- Генерация маскируемого системного прерывания когда счетчик равен 0;
- Частота генератора задается программно;

1.2.14 Основной таймер (TIMx)

Это от 1 до 3 стандартно синхронизированных таймеров, встроенных в устройства STM32F103BVT6. Эти таймеры основаны на 16-битном, автоматически перезагружаемом счетчике, способном как понижать, так и повышать свое значение, 16-битном предварительном делителе частоты и имеют 4 независимых канала, каждый из которых может работать в качестве устройства ввода/вывода, PWM или импульсного выхода. Это дает от 4 до 12 устройств ввода/вывода и большого количества импульсных выходов. Они могут работать вместе как 1 таймер с увеличенными возможностями (Advanced Control Timer) путем их соединения (Timer Link) с помощью синхронизации или создания цепочки событий.

Данный счетчик может быть остановлен в режиме отладки.

Некоторые из стандартных таймеров могут быть использованы для генерации импульсов (PWM) на своих выходах. Каждый из этих таймеров имеет независимый ПДП (DMA) для требуемой генерации.

1.2.15 Таймер с увеличенными возможностями (TIM1)

Таймер с увеличенными возможностями (TIM1) можно рассматривать как трехфазный генратор импульсов (PWM) с уплотнением по 6 каналам. Его можно также рассматривать как основной таймер с увеличенными возможностями. 4 независимых канала могут быть использованы в качестве:

- устройства ввода;
- устройства вывода;
- генератора импульсов PWM (треугольных или прямоугольных);
- выхода для одного импульса;
- дополнительных генераторов импульсов с программно задаваемыми задержками между ними;

При настройках в качестве стандартного 16-битного таймера, он имеет те же возможности, что и основной таймер (TIMx). При настройках в качестве 16-битного генератора импульсов (PWM), он имеет возможность полной модуляции (от 0 до 100%).

Данный счетчик может быть остановлен в режиме отладки.

Много возможностей открывается благодаря этим стандартным TIM таймерам, имеющим одинаковую архитектуру. Таймер с увеличенными возможностями (TIM1) также может работать вместе с основными таймерами (TIMx) путем их соединения с помощью синхронизации или создания цепочки событий.

1.2.16 Модуль общей шины (I^2C)

STM32F103BVT6 содержит два интерфейса I^2C шины, которые могут работать в управляющем (multi-master) или управляемом (slave) состояниях. Они поддерживают стандартный и быстрый режимы.

Эти интерфейсы поддерживают двойную адресацию в управляемом (slave) состоянии (только 7 бит) и обе 7/10-битные адресации в управляющем (master) режиме. Присутствует также аппаратный контроль с помощью циклического избыточного кода (CRC).

Данные интерфейсы обслуживаются ПДП (DMA) и поддерживают шины SM Bus 2.0/PM Bus.

1.2.17 Универсальный синхронный/асинхронный приемопередатчик (USART)

Один из USART интерфейсов способен осуществлять связь на скорости до 4.5 Мбит/с. Другие доступные интерфейсы осуществляют связь на скорости до 2.25 Мбит/с. Они обеспечивают аппаратное управление CTS и RTS сигналами, поддерживают кодирование инфракрасного порта (IrDA SIR ENDEC), ISO 7816 и LIN Master/Slave.

Все USART интерфейсы обслуживаются контроллером ПДП (DMA).

1.2.18 Последовательный периферийный интерфейс (SPI)

STM32F103BVT6 включает в себя до двух последовательных периферийных интерфейсов (SPI), которые могут осуществлять связь на скорости более чем 18 Мбит/с в управляемом (Slave) или управляющем (Master) состоянии в полном дуплексном и симплексном режимах. 3-битный предварительный делитель частоты дает 8 частот управляющего режима и их пакет может содержать от 8 до 16 бит. Аппаратный контроль с помощью циклического избыточного кода (CRC) поддерживает базовые режимы SD-карт/MMS.

Оба последовательных периферийных интерфейса обслуживаются контроллером ПДП (DMA).

1.2.19 Контроллер локальной сети (CAN)

Контроллер локальной сети (CAN) удовлетворяет спецификациям 2.0A и B (активный тип) со скоростью передачи до 1 Мбит/с. Он может принимать и передавать как стандартные пакеты с 11-битным заголовком, так и дополнительные пакеты с 29-битным заголовком. Он имеет три буфера при передаче, два FIFO при приеме с 3 уровнями и 14 масштабируемыми фильтрами.

1.2.20 Основное устройство ввода/вывода (GPIO)

Каждый из контактов основного УВВ (GPIO) может быть программно сконфигурирован как выходной (двухтактный или с открытым стоком), как входной (с или без согласования) или как имеющий другое второстепенное назначение. Большинство из контактов основного УВВ (GPIO) совместимы с цифровыми или аналоговыми другими назначениями. Все контакты основного УВВ выдерживают большие токи.

Конфигурация других назначений УВВ может быть закрыта при необходимости выполнения специальной последовательности для защиты от ошибочной записи в регистры ввода/вывода.

Ввод/вывод на шине APB2 может достичь частоты в 18Мгц.

1.2.21 Аналого-цифровой преобразователь (ADC)

Два 12-битных АЦП (ADC) входят в состав STM32F103BVT6 и к каждому из них подключены 16 внешних каналов, выполняющих преобразования в режиме

одинарной точности и в режиме повышенной точности. В режиме повышенной точности, автоматическое преобразование выполняется на выбранной группе аналоговых входов.

Дополнительные логические функции, встроенные в АЦП позволяют:

- параллельное измерение и сохранение;
- последовательное измерение и сохранение;
- однократное шунтирование;

АЦП может обслуживаться контроллером ПДП (DMA).

Возможность аналогового сторожевого таймера позволяет очень точно сравнивать оцифрованное напряжение с одного, нескольких или всех выбранных каналов. Прерывание генерируется когда оцифрованное напряжение выходит за запрограммированные пороги.

События, генерируемые стандартными таймерами (TIMx) и таймером с увеличенными возможностями (TIM1) могут управлять стартовым триггером (start trigger) АЦП, задающим триггером (injection trigger) и триггером ПДП (DMA) соответственно, что позволяет программе синхронизировать аналого-цифровое преобразование и таймеры.

1.2.22 Датчик температуры

Датчик температуры может производить постоянное напряжение в зависимости от изменения температуры.

Пределы этого напряжения находятся между 2 В < V_{DDA} < 3.6 В. Датчик температуры внутри устройства подключен к входному каналу **ADC12_IN16**, который используется для преобразования выходного напряжения датчика в цифровое значение.

1.2.23 Прямой доступ к памяти (DMA)

ПДП является специфической особенностью STM32F103BVT6. Общее назначение 7-канального широконастраиваемого ПДП — возможность управления обменом типа память-память, периферия-память и память-периферия. ПДП-контроллер поддерживает управление буферизацией для избежания генерации прерываний когда контроллер достигает конца буфера.

Каждый канал удовлетворяет требованиям к аппаратному ПДП с поддержкой программных триггеров в каждом канале. Конфигурация выполняется программно и объемы передаваемых данных между источником и приемником независимы друг от друга.

ПДП может быть использован для обмена с основной периферией: последовательным периферийным интрефейсом (SPI), общей шиной (I^2C), универсальным синхронным/асинхронным передатчиком (USART), основными таймерами (TIMx), таймером с увеличенными возможностями (TIM1) и АЦП (ADC).

1.2.24 Универсальная последовательная шина (USB)

STM32F103BVT6 включает в себя устройство USB, совместимое по периферии с полноценным 12 Мбт USB. USB интерфейс осуществляет полноценное (12 Мбит/с) функциональное взаимодействие. Он имеет программно конфигурируемые окончательные установки и поддержку восстановления

работы после обрыва. Необходимые для работы колебания частотой 48 Мгц генерируются внутренним главным генератором с системой фазовой автоподстройкой частоты (PLL).

1.2.25 Последовательный проводной отладочный порт JTAG (SWJ-DP)

ARM SWJ-DP интерфейс встроен в микроконтроллер и состоит из JTAG и последовательного проводного отладочного порта, что позволяет использовать для подключения как последовательный проводной отладчик так и переходное устройство JTAG.

Контакты JTAG TMS и TCK совмещены соответственно с контактами SWDIO, SWCLC и специальной последовательностью TMS контактов, использующихся для переключения между JTAG-DP и SW-DP (последовательный проводной порт).

1.3 Назначение контактов STM32F103VBT6

На рисунке 1-3 показано назначение контактов STM32F103BVT6 в корпусе LQFP-100. Эту документацию следует использовать только если микроконтроллер в этом корпусе.

Назначение каждого контакта полностью показано в Таблице 1-1.

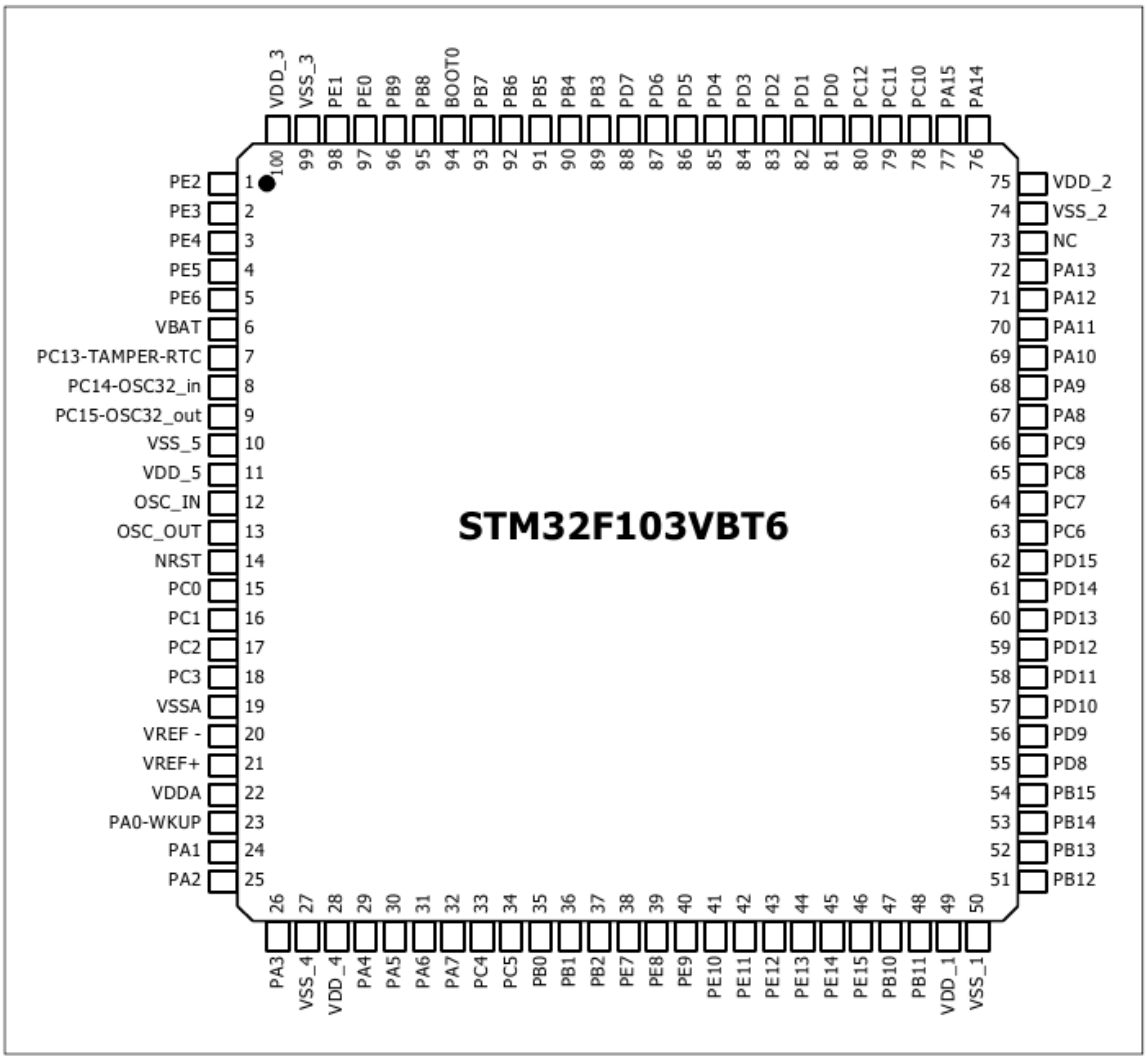


Рисунок 1-3. Назначение контактов STM32F103BVT6

Таблица 1. Назначение контактов STM32F103BVT6

№ контакта	Наименование	Тип	Главное назначение	Другие назначения	
				По умолчанию	Другое
1	PE2	ввод/вывод	PE2	TRACECK	
2	PE3	ввод/вывод	PE3	TRACED0	
3	PE4	ввод/вывод	PE4	TRACED1	
4	PE5	ввод/вывод	PE5	TRACED2	
5	PE6	ввод/вывод	PE6	TRACED3	
6	VBAT	питание от батареи	VBAT		
7	PC13-TIMER-RTC	ввод/вывод	PC13	TIMER-RTC	
8	PC14-OSC32_IN	ввод/вывод	PC14	OSC32_IN	
9	PC15-OSC32_OUT	ввод/вывод	PC15	OSC32_OUT	
10	VSS_S	корпус	VSS_S		
11	VDD_S	ввод/вывод	VDD_S		
12	OSC_IN	ввод	OSC_IN		
13	OSC_OUT	вывод	OSC_OUT		
14	NRST	ввод/вывод	NRST		
15	PC0	ввод/вывод	PC0	ADC12_IN10	
16	PC1	ввод/вывод	PC1	ADC12_IN11	
17	PC2	ввод/вывод	PC2	ADC12_IN12	
18	PC3	ввод/вывод	PC3	ADC12_IN13	
19	VSSA	корпус	VSSA		
20	VREF-	опорное напряжение	VREF-		
21	VREF+	опорное напряжение	VREF+		
22	VDDA	напряжение питания	VDDA		
23	PA0-WKUP	ввод/вывод	PA0	WKUP USART2_CT ADC12_IN0 TIM2_CH1_ETR	
24	PA1	ввод/вывод	PA1	USART2_RTS ADC12_IN1 TIM2_CH2	
25	PA2	ввод/вывод	PA2	USART2_TX ADC12_IN2 TIM2_CH3	
26	PA3	ввод/вывод	PA3	USART2_RX ADC12_IN3 TIM2_CH4	
27	VSS_4	корпус	VSS_4		
28	VDD_4	напряжение питания	VDD_4		
29	PA4	ввод/вывод	PA4	USART2_CK ADC12_IN4 SPI1_NSS	

Таблица 1 (продолжение). Назначение контактов STM32F103BVT6

№ контакта	Наименование	Тип	Главное назначение	Другие назначения	
				По умолчанию	Другое
30	PA5	ввод/вывод	PA5	ADC12_IN5 SPI1_SCK	
31	PA6	ввод/вывод	PA6	ADC12_IN6 SPI1_MISO TIM3_CH1	TIM1_BKIN
32	PA7	ввод/вывод	PA7	ADC12_IN7 SPI1_MOSI TIM3_CH2	TIM1_CH1N
33	PC4	ввод/вывод	PC4	ADC12_IN14	
34	PC5	ввод/вывод	PC5	ADC12_IN15	
35	PB0	ввод/вывод	PB0	ADC12_IN8 TIM3_CH3	TIM1_CH2N
36	PB1	ввод/вывод	PB1	ADC12_IN9 TIM3_CH4	TIM1_CH3N
37	PB2/BOOT1	ввод/вывод	PB2/BOOT1		
38	PE7	ввод/вывод	PE7		TIM1_ETR
39	PE8	ввод/вывод	PE8		TIM1_CH1N
40	PE9	ввод/вывод	PE9		TIM1_CH1
41	PE10	ввод/вывод	PE10		TIM1_CH2N
42	PE11	ввод/вывод	PE11		TIM1_CH2
43	PE12	ввод/вывод	PE12		TIM1_CH3N
44	PE13	ввод/вывод	PE13		TIM1_CH3
45	PE14	ввод/вывод	PE14		TIM4_CH4
46	PE15	ввод/вывод	PE15		TIM1_BKIN
47	PB10	ввод/вывод	PB10	USART3_TX I2C2_SCL	TIM2_CH3
48	PB11	ввод/вывод	PB11	USART3_RX I2C2_SDA	TIM2_CH4
49	VSS_1	корпус	VSS_1		
50	VDD_1	напряжение питания	VDD_1		
51	PB12	ввод/вывод	PB12	USART3_CK SPI2_MSS I2C2_SMBAI TIM1_BKIN	
52	PB13	ввод/вывод	PB13	USART3_CTS SPI2_SCK TIM1_CH1N	
53	PB14	ввод/вывод	PB14	USART3_RTS SPI2_MISO TIM1_CH2N	
54	PB15	ввод/вывод	PB15	SPI2_MOSI TIM1_CH3N	
55	PD8	ввод/вывод	PD8		USART3_TX
56	PD9	ввод/вывод	PD9		USART3_RX
57	PD10	ввод/вывод	PD10		USART3_CK

Таблица 1 (продолжение). Назначение контактов STM32F103BVT6

№ контакта	Наименование	Тип	Главное назначение	Другие назначения	
				По умолчанию	Другое
58	PD11	ввод/вывод	PD11		USART3_CTS
59	PD12	ввод/вывод	PD12		USART3_RTS TIM4_CH1
60	PD13	ввод/вывод	PD13		TIM4_CH2
61	PD14	ввод/вывод	PD14		TIM4_CH3
62	PD15	ввод/вывод	PD15		TIM4_CH4
63	PC6	ввод/вывод	PC6		TIM3_CH1
64	PC7	ввод/вывод	PC7		TIM3_CH2
65	PC8	ввод/вывод	PC8		TIM3_CH3
66	PC9	ввод/вывод	PC9		TIM3_CH4
67	PA8	ввод/вывод	PA8	USART1_CK TIM1_CH1 MC0	
68	PA9	ввод/вывод	PA9	USART1_TX TIM1_CH2	
69	PA10	ввод/вывод	PA10	USART1_RX TIM1_CH3	
70	PA11	ввод/вывод	PA11	USART1_CTS TIM1_CH4 CANRX USBDM	
71	PA12	ввод/вывод	PA12	USART1_RTS TIM1_ETR CANTX USBDP	
72	PA13/JTMS/SWDIO	ввод/вывод	JTMS/SWDIO	PA13	
73	не используется	-	-		
74	VSS_2	корпус	VSS_2		
75	VDD_2	напряжение питания	VDD_2		
76	PA14/JTCK/SWCLC	ввод/вывод	JTCK/SWCLC	PA14	
77	PA15/JTDI	ввод/вывод	JTDI	PA15	TIM2_CH1_ETR SPI1_NSS
78	PC10	ввод/вывод	PC10		USART3_TX
79	PC11	ввод/вывод	PC11		USART3_RX
80	PC12	ввод/вывод	PC12		USART3_CK
81	PD0	ввод/вывод	OSC_IN		CANRX
82	PD1	ввод/вывод	OSC_OUT		CANTX
83	PD2	ввод/вывод	PD2	TIM3_ETR	
84	PD3	ввод/вывод	PD3		USART2_CTS
85	PD4	ввод/вывод	PD4		USART2_RTS
86	PD5	ввод/вывод	PD5		ISART2_TX
87	PD6	ввод/вывод	PD6		USART2_RX
88	PD7	ввод/вывод	PD7		USART2_CK

Таблица 1 (окончание). Назначение контактов STM32F103BVT6

№ контакта	Наименование	Тип	Главное назначение	Другие назначения	
				По умолчанию	Другое
89	PB3/JTDO	ввод/вывод	JTDO	PB3 TRACESWO	TIM2_CH2 SPI1_SCK
90	PB4/JNTRST	ввод/вывод	PA9	PB4	TIM3_CH1 SPI1_MOSI
91	PB5	ввод/вывод	PA10	I2C1_SMBAI	TIM3_CH2 SPI1_SCK
92	PB6	ввод/вывод	PA9	I2C1_SCL TIM4_CH1	USART1_TX
93	PB7	ввод/вывод	PA10	I2C1_SDA TIM4_CH2	USART1_RX
94	BOOT0	ввод	BOOT0		
95	PB8	ввод/вывод	PB8	TIM4_CH3	I2C1_SCL CANRX
96	PB9	ввод/вывод	PB9	TIM4_CH4	I2C1_SDA CANTX
97	PE0	ввод/вывод	PE0	TIM4_ETR	
98	PE1	ввод/вывод	PE1		
99	VSS_3	корпус	VSS_3		
100	VDD_3	напряжение питания	VDD_3		

STM32

2: Программирование на C для микроконтроллера STM32

Мы рекомендуем использовать свободно распространяемое программное обеспечение фирмы **Raisonance (www.raisonance.com)** в программировании на языке C для микроконтроллера STM32. Желающие должны зарегистрироваться перед скачиванием этих программ с **www.raisonance.com** или **http://www.mcu-raisonance.com/mcu_downloads.html**.

Состав програмных средств для программирования микроконтроллера STM32 включает в себя:

1. **Ride7 IDE for ARM** Эту программа может быть использована с рядом программ для разработки, включающим в себя: редактор, отладчик и менеждер проектов, который является общим для нескольких семейств ST-микроконтроллеров с ядром ARM и которые в свою очередь объединены инструментарием, основанным на GNU GCC и программным симулятором.

2. **Rkit-ARM** Это ПО включает в себя компилятор GNU C, поддерживающий ядро ARM Cortex M-3 от STMicroelectronics и встроенные аппаратно-зависимые библиотеки.

3. **Flash Loader Demonstrator** Эта программа является загрузчиком в память микроконтроллера от STMicroelectronics. Пользователь может его свободно скачать с www.st.com. Это программное средство берет HEX-файл, получаемый с помощью C-компилятора, и загружает его во flash-память микроконтроллера STM32 посредством USART (универсального синхронно-асинхронного приемопередатчика). Это программное средство может работать и через преобразователь USB - RS-232 (кабель USB to COM).

Все ПО упаковано вместе с руководством по эксплуатации и записано на CD-ROM.

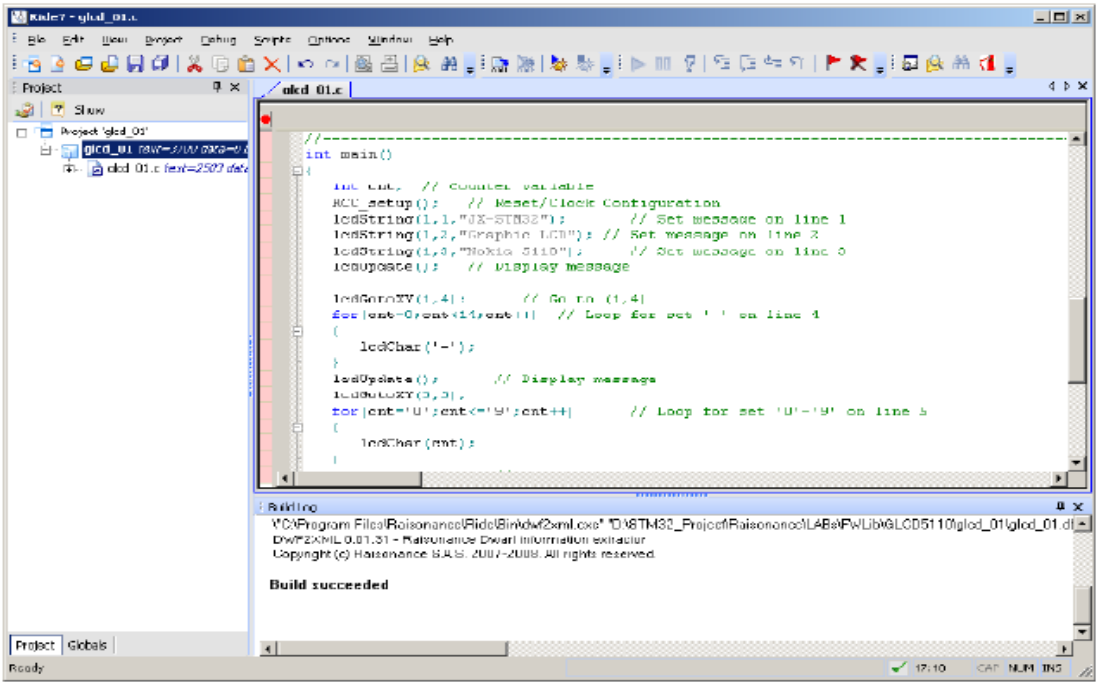


Рисунок 2-1. Главное окно Ride7.

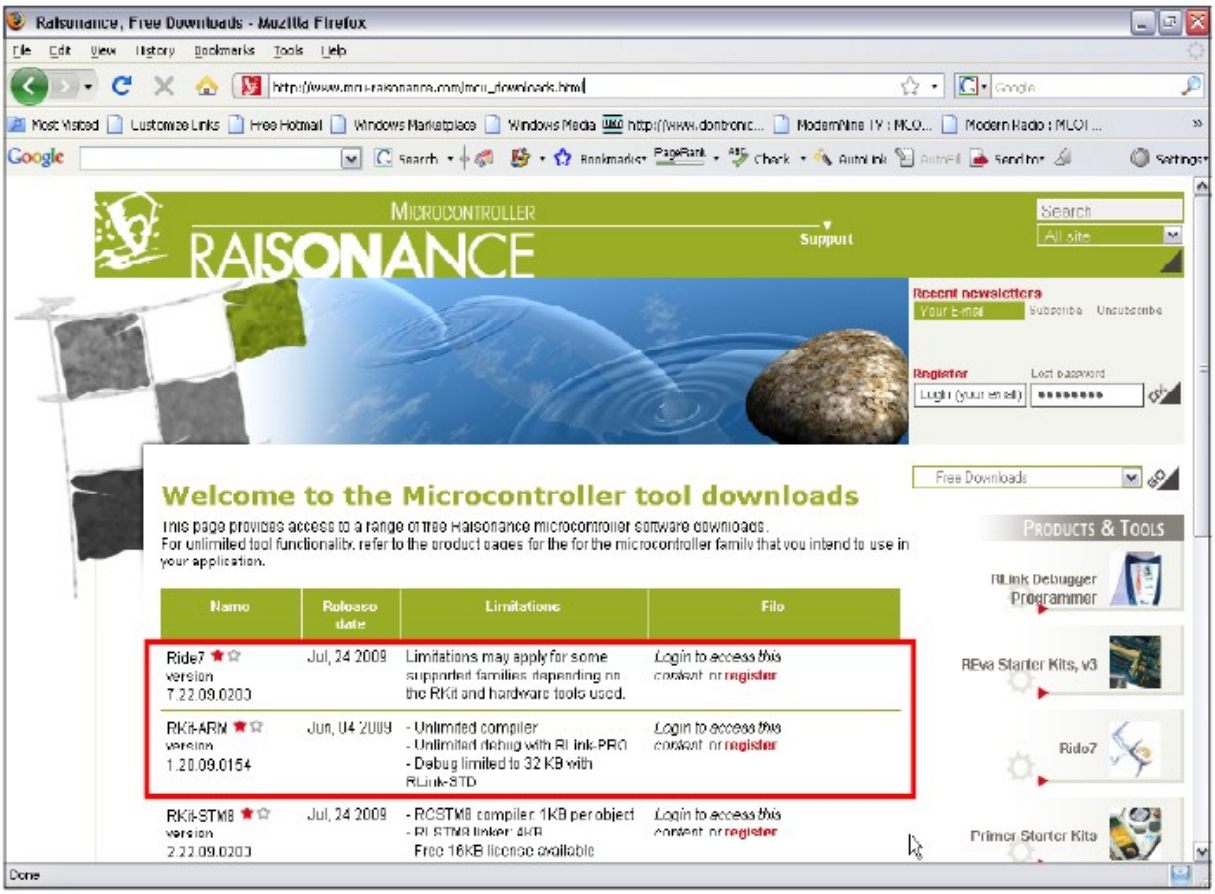
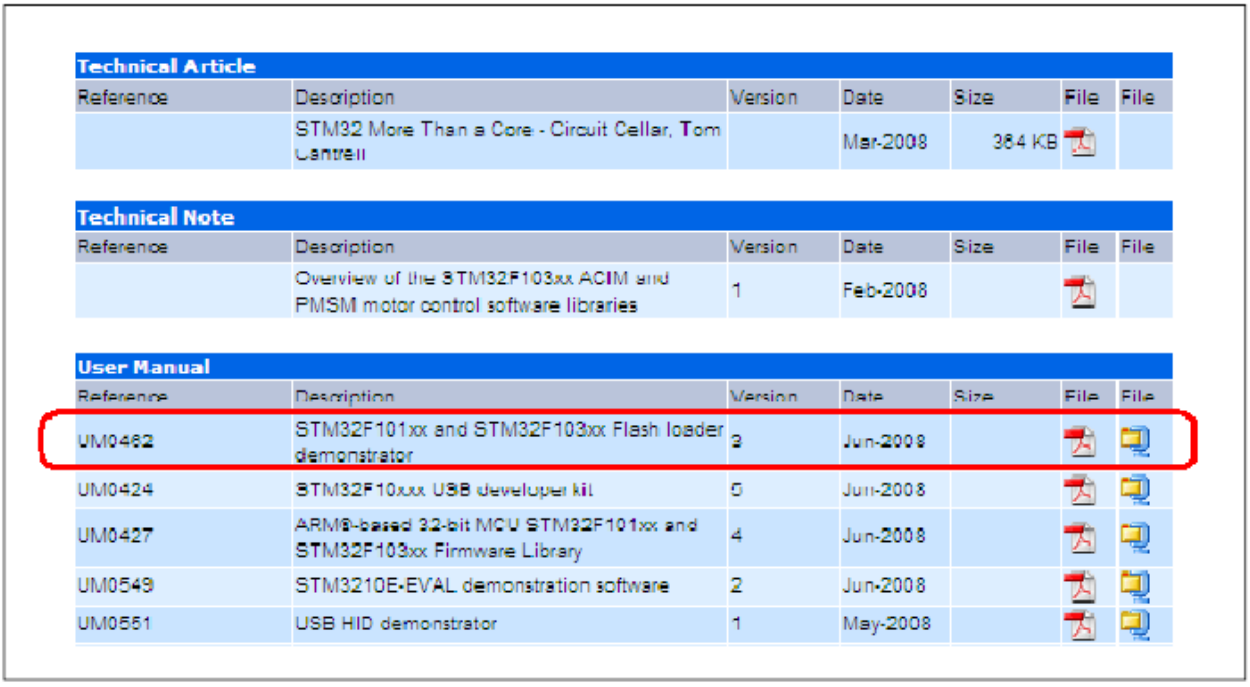


Рисунок 2-2. Web-страница загрузки программных средств ARM7 от Raisonance (www.raisonance.com)



Technical Article

Reference	Description	Version	Date	Size	File	File
	STM32 More Than a Core - Circuit Cellar, Tom Cantrell		Mar-2008	384 KB		

Technical Note

Reference	Description	Version	Date	Size	File	File
	Overview of the STM32F103xx ACIM and PMSM motor control software libraries	1	Feb-2008			

User Manual

Reference	Description	Version	Date	Size	File	File
UM0462	STM32F101xx and STM32F103xx Flash loader demonstrator	2	Jun-2008			
UM0424	STM32F10xxx USB developer kit	5	Jun-2008			
UM0427	ARM®-based 32-bit MCU STM32F101xx and STM32F103xx Firmware Library	4	Jun-2008			
UM0549	STM3210E-EVAL demonstration software	2	Jun-2008			
UM0551	USB HID demonstrator	1	May-2008			

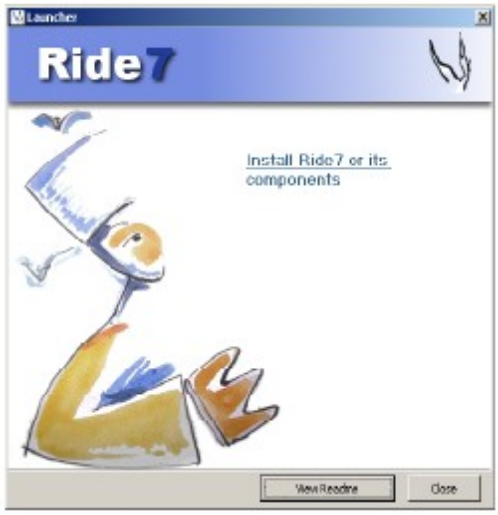
Рисунок 2-3. Web-страница загрузки Flash Loader Demonstrator от STMicroelectronics (www.st.com)

2.1 Установка ПО

Ride7 IDE, Rkit-ARM и Flash Loader требуют для своей работы Windows XP SP2 или более позднюю версию Windows и не меньше 500 Мб места на жестком диске. Ваш компьютер должен иметь свободный последовательный (COM) порт для взаимодействия с платой JX-STM32. Вам потребуется преобразователь USB - RS-232 (кабель USB to COM), если ваш компьютер имеет только USB-порт. Для этой цели наиболее подходит модель UCON_232S (www.inexglobal.com).

2.1.1 Установка Ride7 IDE

(1) Запустите установочный файл Ride7 IDE. Появится первое окно, на котором выберите ссылку **Install Ride7 or its components**.



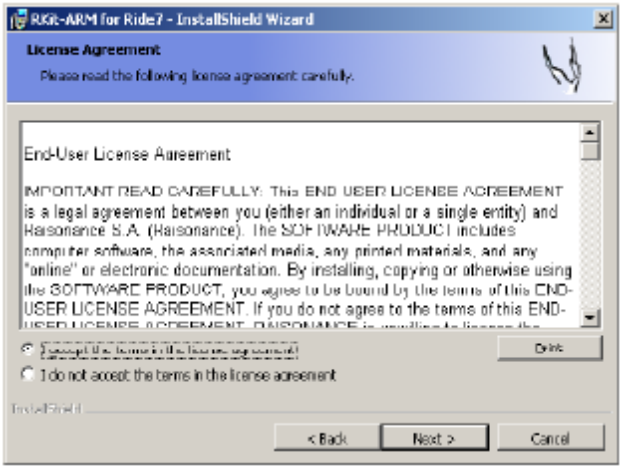
(2) Когда появится окно установки, укажите устанавливаемые компоненты. Выберите путь для установки. По умолчанию **C:\Program Files\Raisonance**. После этого нажмите кнопку **Start install** для запуска установки.

(3) Ожидайте окончания установки.

2.1.2 Установка Rkit-ARM

(1) Запустите установочный файл **Rkit-ARM_BN45A.exe** (версия этой программы может быть другой). Откроется первое окно программы установки. Нажмите кнопку **Next**.

Появится окно **Лицензионного Соглашения**. Выберите пункт **I accept the terms in the license agreement** и нажмите кнопку **Next**. Установка запустится.



По окончании установки появится окно **Завершения установки**. Нажмите кнопку **Finish**. После этого оболочка Rkit-ARM готова к запуску.

2.2 Документация по STM32

После окончания установки Rkit-ARM и Ride7 в каталоги с программами также было скопировано много документации, включающей в себя:

1. **STM32F10x_ref_manual.pdf** - этот файл с руководством расположен в каталоге **C:\Program Files\Raisonance\Ride\Doc\ARM\Manuals**.

2. **STM32-LIB_manual.pdf** или **STM32_Firmware_Library.pdf** - этот файл подробно описывает аппаратно-зависимые библиотеки STM32. Вы можете найти его в каталогах **C:\Program Files\Raisonance\Ride\Doc\ARM\Manuals** и **C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB**.

3. **GettingStartedARM_RIDE7.pdf** - это первое руководство для программирования микроконтроллеров STM32 с помощью Ride7 и Rkit-ARM. Он расположен в каталоге **C:\Program Files\Raisonance\Ride\Doc\ARM**.

4. **Файлы примеров программ** - в Raisonance создано много примеров для устройств STM32. Аппаратное устройство, для которого созданы эти примеры - плата STM32F103B-EVAL (Плата JX-STM32 создана после этой опытной платы). Расположение всех файлов примеров — каталог **C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\examples**.

2.3 JX-STM32: плата на основе микроконтроллера STM32F103VBT6

Эта отладочная плата создается и производится Азиатско-тихоокеанским отделением STMicroelectronics (Тайланд) и Штаб-квартирой фирмы STMicro-electronics (Сингапур)

На Рисунке 2-4 показано расположение элементов на плате JX-STM32.

Технические характеристики данной отладочной платы:

- Основной микроконтроллер - STM32F103VBT6;
- Flash-память - 128 кБ;
- Статическое ОЗУ - 20 кБ;
- Основной кварц - 16 Мгц. Максимальная тактовая частота достигает 72 Мгц с помощью системы фазовой автоподстройки частоты (PLL);
- Кварц 32.768 кГц во внутреннем генераторе реального времени (RTC);
- 2 разъема универсального синхронного-асинхронного приемопередатчика (USART). Поддержка связи в последовательном виде и загрузка кода пользователя с помощью порта USART0;
- Интерфейс USB 2.0;
- Схема управления локальной сетью на основе микросхемы приемопередатчика по локальной сети;
- 4 кнопки, работающих на замыкание при нажатии;
- Джойстик с 4 степенями свободы и высоким входным сопротивлением;
- Усилитель звуковой частоты (ЗЧ) с громкоговорителем на плате и управлением громкостью;
- Схема взаимодействия с графическим дисплеем GLCD5110;

- 8 светодиодов с резистором ограничения по току;
- Интерфейс SD-платы;
- Температурный датчик LM75ADP, управляемый через общую шину (I^2C);
- Контакт свободного порта для взаимодействия с реальным миром (??);
- 350-контактная макетная плата (breadboard);
- Поддержка JTAG-порта. Поддерживаемые средства: R-Link (Raisonance), U-Link (Keil) и ST-LINK;
- Напряжение питания +6 В, 500 мА с напряжением регулятора +3.3 В;

Принципиальная схема платы JX-STM32 показана на рисунках 2-5 и 2-6.

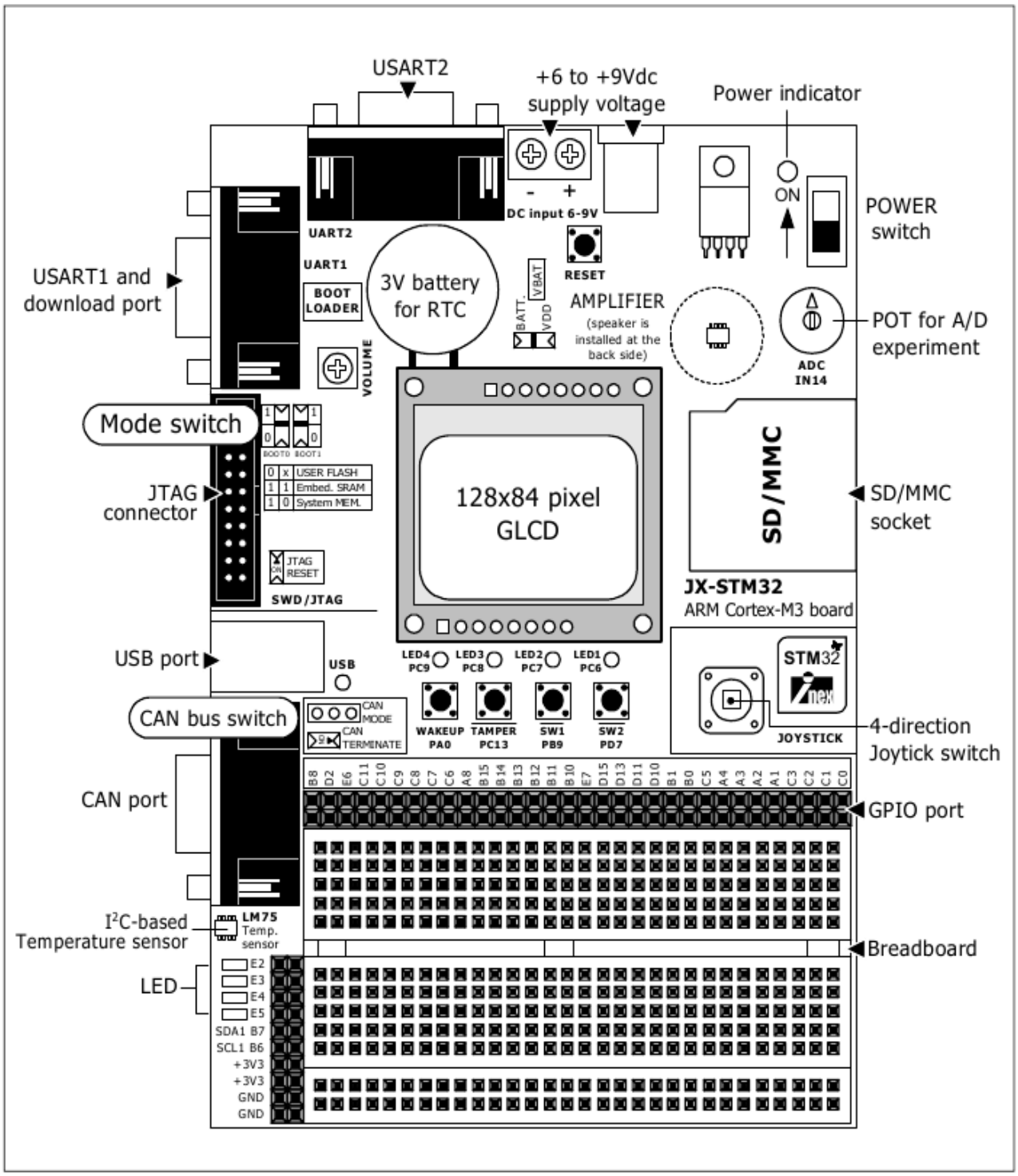


Рисунок 2-4. Расположение элементов на плате JX-STM32

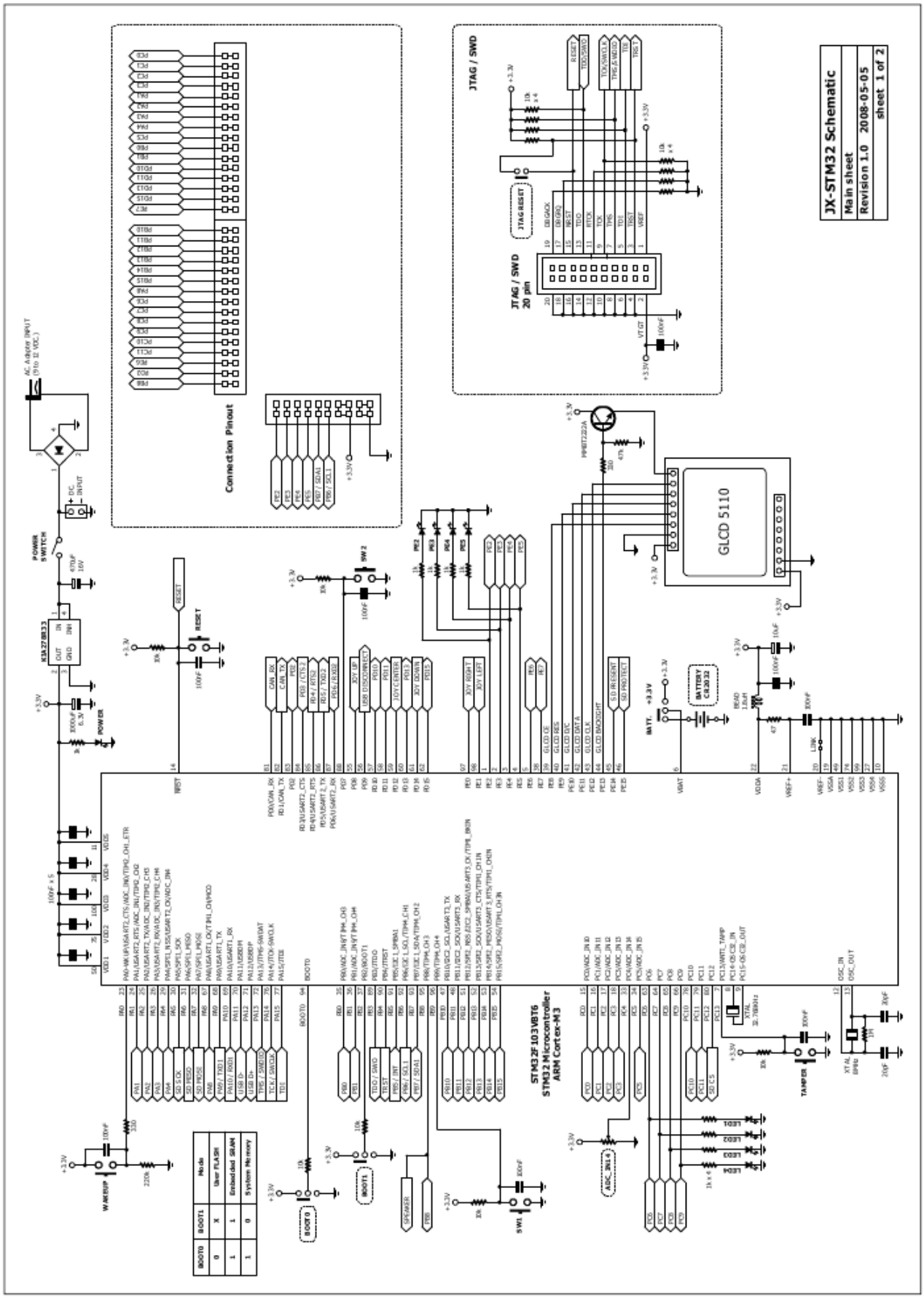


Рисунок 2-5. Принципиальная схема платы JX-STM32 (микроконтроллер и дисплей)

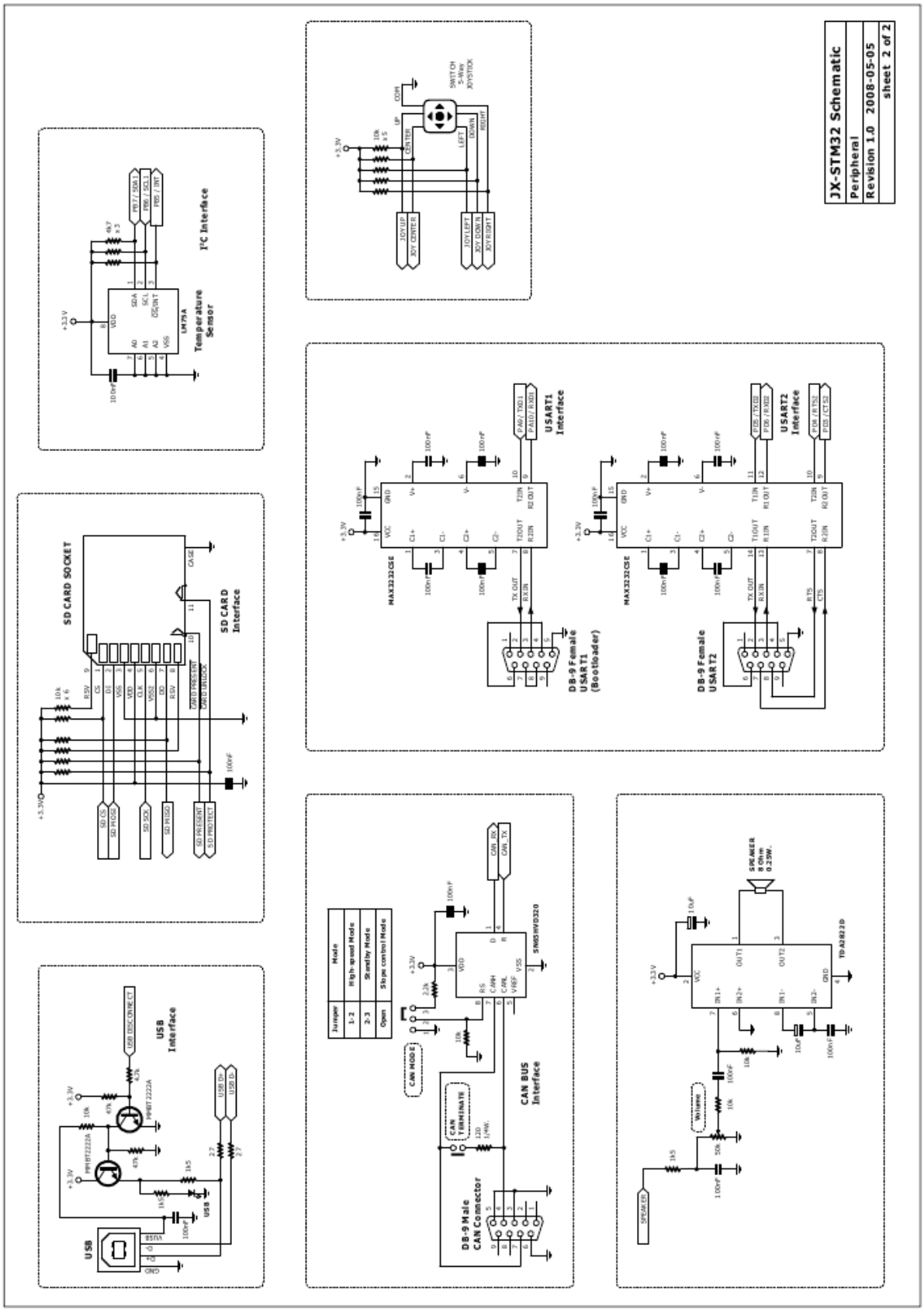
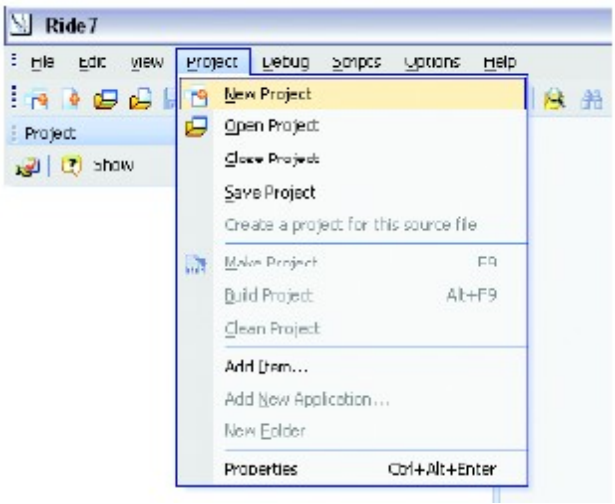


Рисунок 2-6. Принципиальная схема платы JX-STM32
(Устройства ввода-вывода и RS232)

2.4 Как создать файл проекта для STM32 с помощью Ride7

Первый пример для STM32, созданный с помощью Ride7 показывает мигание светодиода на порту PC6.

(1) Запустите программу Ride7. Выберите пункт меню **Project->New Project** для создания нового проекта.



(2) Откроется окно **New application**. Выберите нужный микроконтроллер. Имя проекта и его местоположение показаны на рисунке ниже.

(2.1) **Type**: Выберите **New application to be built** (по умолчанию)

(2.2) **Processor**: Укажите нужный микроконтроллер как **STMF103VBT6**

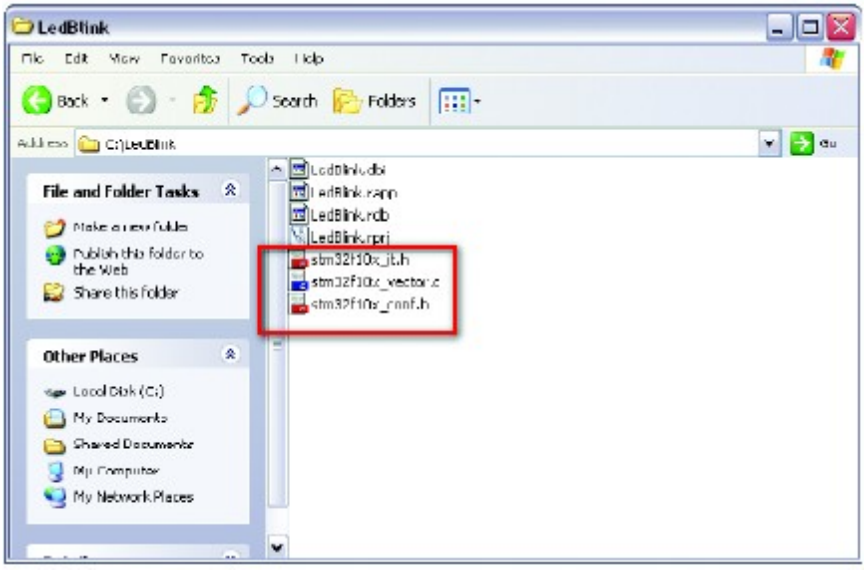
(2.3) **Name**: Назовите проект как **LedBlink**

(2.4) **Location**: Укажите местоположение вашего проекта, например **C:\LedBlink**. Это возможно, если вы заранее создали каталог **LedBlink**.

(2.5) Нажмите кнопку **Finish**.

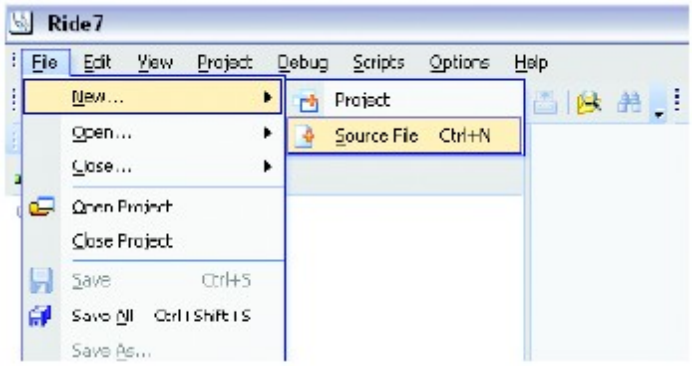
(3) Скопируйте 3 необходимых файла заголовков в этот каталог:

- **stm32f10x_conf.h**; исходное местоположение этого файла в каталоге ***C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\project***;
- **stm32f10x_vector.c**; исходное местоположение этого файла в каталоге ***C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\project\RIDE***;
- **stm32f10x_it.h**; исходное местоположение этого файла в каталоге ***C:\Program Files\Raisonance\Ride\Lib\ARM\STM32F10x_LIB\project***;



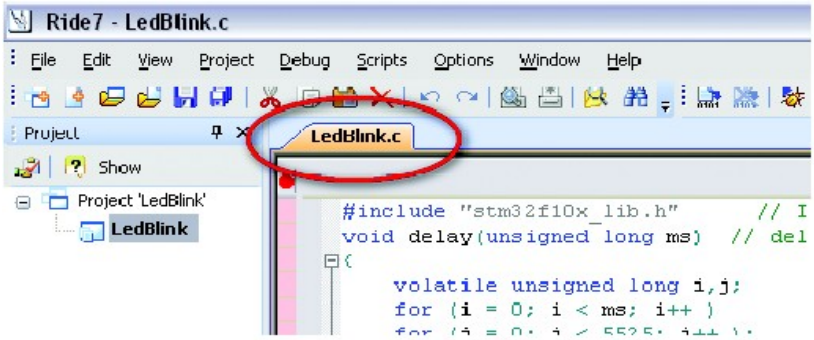
Внимание: Копирование файлов на этом шаге очень важно. Это рекомендуется, когда вы создаете новый проект.

(4) Создайте исходный файл проекта на языке С выбрав пункт меню **File->Source File**.

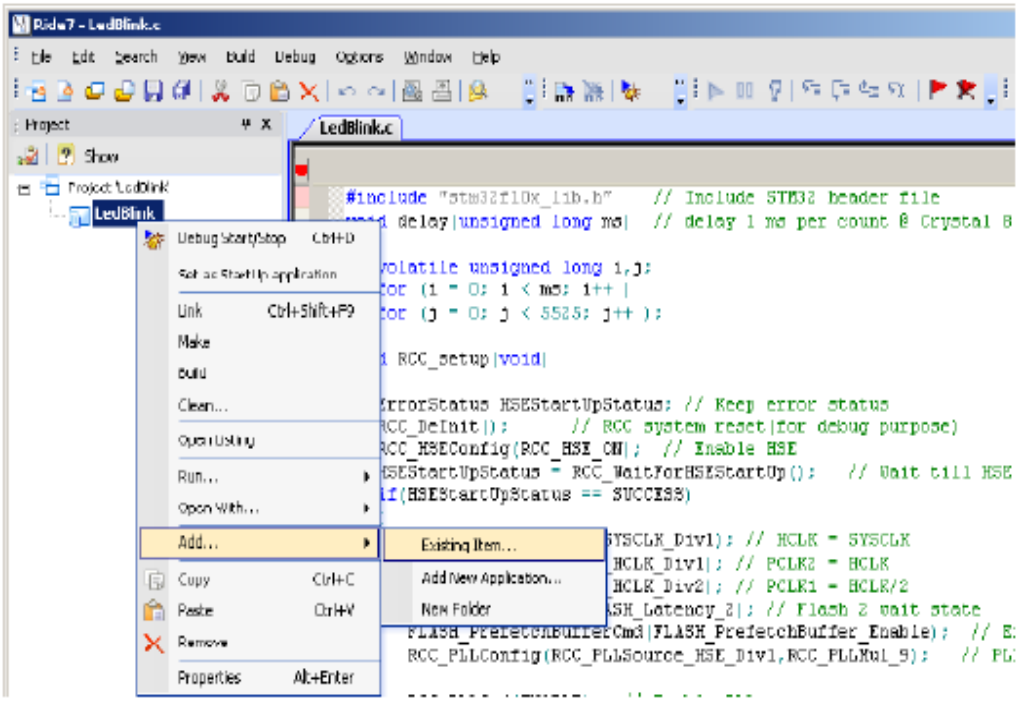


(5) Наберите код на языке С, указанный в Листинге 2-1 в поле редактирования.

(6) Сохраните этот файл выбрав пункт меню **File->Save as...** Укажите имя файла как ***LedBlink.c*** и сохраните его в тот же каталог, куда был сохранен файл ***stm32f10x_conf.h***. Выберите ссылку **Source File**. Имя сохраненного файла появится на экране.



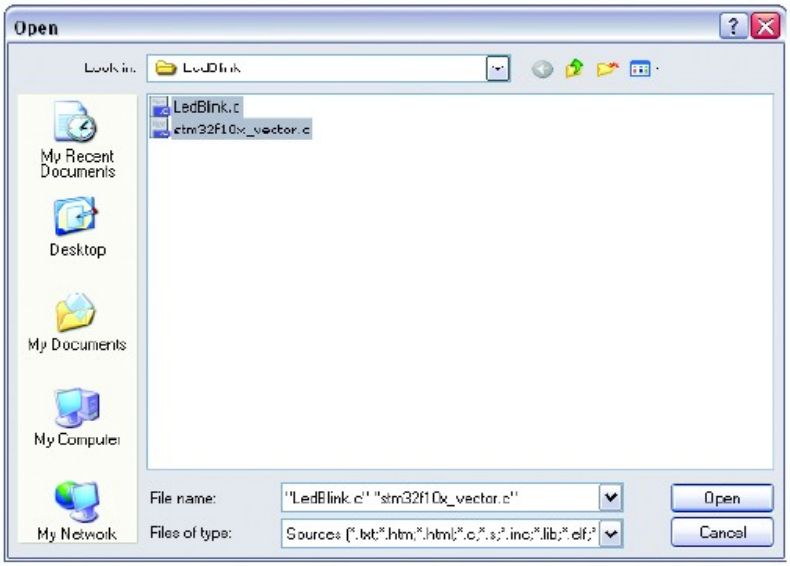
(7) Подключите исходный файл на языке С и **stm32f10x_vector.c** к проекту. Нажмите правой кнопкой мыши на корневой ссылке **LedBlink**. В появившемся всплывающем меню, показанном на рисунке ниже, выберите пункт **Add...->Existing Item**.



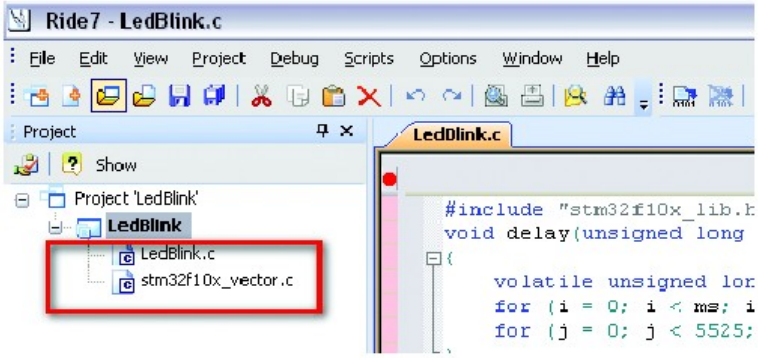
```
#include "stm32f10x_lib.h"                                  // Подключаем STM32 заголовочный файл

void delay(unsigned long ms)
// Задержка 1 мс за цикл при кварце 8.0 Мгц и PLL9x или SYSCLK = 72 Мгц
{
  volatile unsigned long i, j;
  for(i = 0; i < ms; i++)
    for(j=0; j<5525; j++);
}

void RCC_setup(void)
{
  ErrorStatus HSEStartUpStatus;                             // Переменная статуса ошибки
  RCC_DeInit();                                             // Сброс RCC-системы (для отладки)
  RCC_HSEConfig(RCC_HSE_ON);                                // Разрешение HSE
  HSEStartUpStatus = RCC_WaitForHSEStartUp();               // Ждем пока HSE не будет готов
  if(HSEStartUpStatus == SUCCESS)
   {
    RCC_HCLKConfig(RCC_SYSCLK_Div1);                        // HCLK = SYSCLK
    RCC_PCLK2Config(RCC_HCLK_Div1);                         // PCLK2 = HCLK
    RCC_PCLK1Config(RCC_HCLK_Div2);                         // PCLK1 = HCLK/2
    FLASH_SetLatency(FLASH_Latency_2);                      // Состояние ожидания Flash 2
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);   // Доступен предварительный буфер
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);    // PLLCLK = 8 Мгц * 9 = 72 Мгц
    RCC_PLLCmd(ENABLE);                                     // Доступ к PLL разрешен
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);     // Ждем пока PLL не будет готов
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);              // Выбираем PLL как источник частоты
    while(RCC_GetSYSCLKSource() != 0x08);                   // Ждем пока PLL не станет источником
   }
}

void GPIO_setup()
{
  GPIO_InitTypeDef GPIO_InitStructure;
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);     // Разрешаем частоту GPIO
  // Настраиваем PC6, PC7, PC8 и PC9 как двухтактный выход (подключение к светодиоду)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
}

int main()
{
  RCC_setup();                                              // Настройка системных частот
  GPIO_setup();                                             // Настройка портов GPIO
  while(1)                                                  // Бесконечный цикл
   {
     GPIO_WriteBit(GPIOC, GPIO_Pin_6, 1);                   // Светодиод на PC6 ВКЛ
     delay(500);                                            // Задержка 0,5 мс
     GPIO_WriteBit(GPIOC, GPIO_Pin_6, 0);                   // Светодиод на PC6 ВЫКЛ
     delay(500);                                            // Задержка 0,5 мс
   }
}
```

Листинг 2-1. Файл LEDBlink.c; первый пример, демонстрирующий мигание светодиода на порту PC6 микроконтроллера STM32F103VBT6 на плате JX-STM32

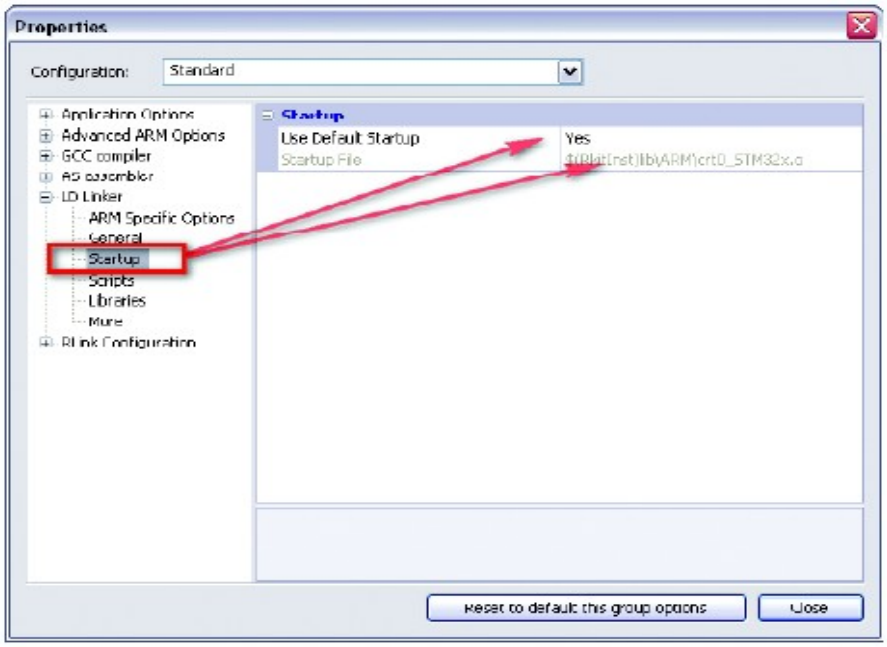
(8) Появится окно выбора файлов. Выберите оба файла: **LedBlink.c** и **stm32f10x_vector.c**.



(9) После подключения файлы **LedBlink.c** и **stm32f10x_vector.c** будут входить в проект, что отобразится в списке слева.



(10) Затем подготовьте проект к компиляции, соответственно настроив его параметры. Выберите пункт меню **Project->Properties**.

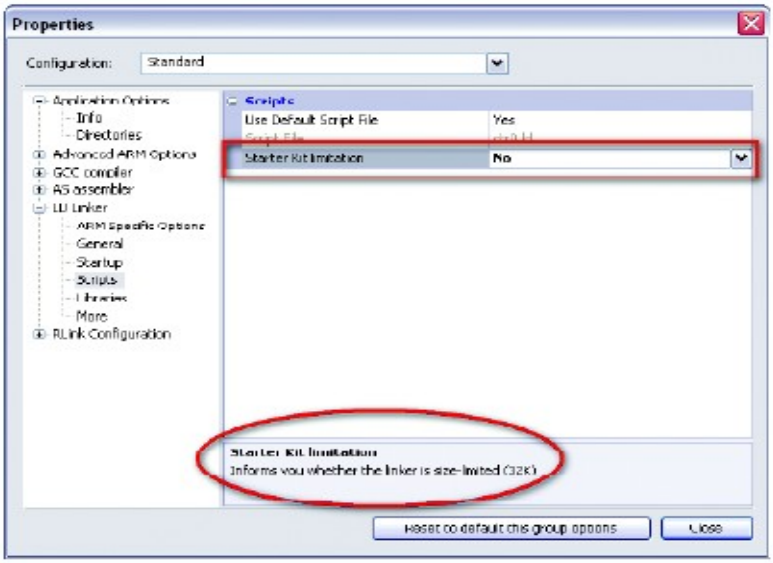


Раскройте пункт **LD Linker**. Выберите подпункт **Startup** и установите 2 следующих параметра:

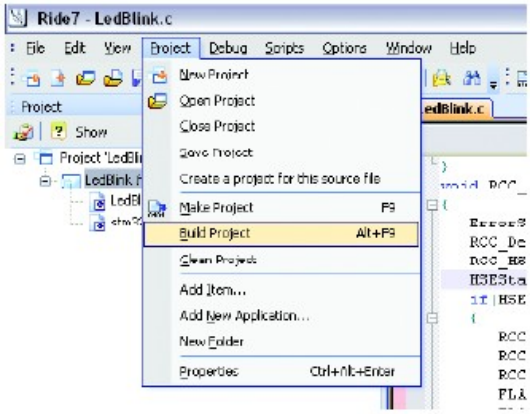
(10.1) **Use Default Startup** - **No**

(10.2) **Startup File** - удалите имя предыдущего файла и оставьте пустое место.

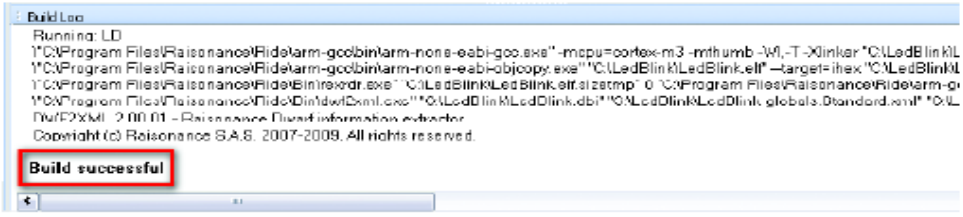
(11) Вернитесь к пункту **LD Linker**. Выберите подпункт **Scripts**, и установите параметр **Starter Kit Limitation** в **No**. **Этот шаг очень важен. Это позволит компилировать для всего ряда используемых микроконтроллеров.** Нажмите кнопку **Close** для подтверждения настроек.



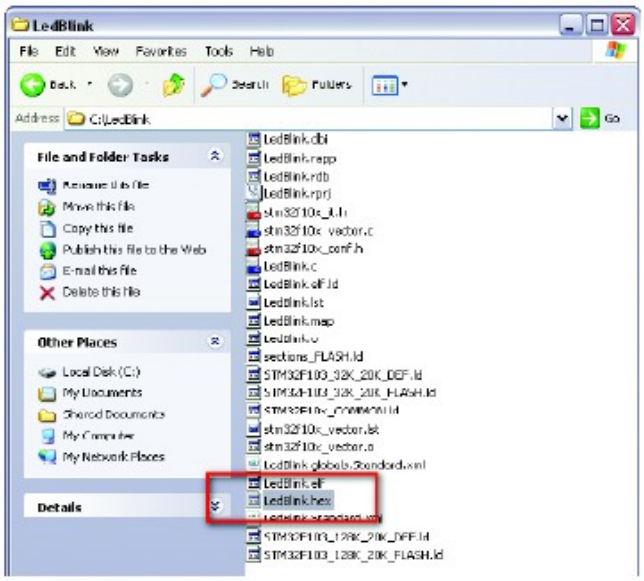
(12) Скомпилируйте проект выбрав меню **Project->Build Project**.



(13) В области показа информации о компил`яции проекта должно появиться сообщение «*Build successful*».



(14) Результирующий файл **LedBlink.hex** создается в каталоге проекта. Этот hex-файл может быть позже загружен в flash-память микроконтроллера STM32.

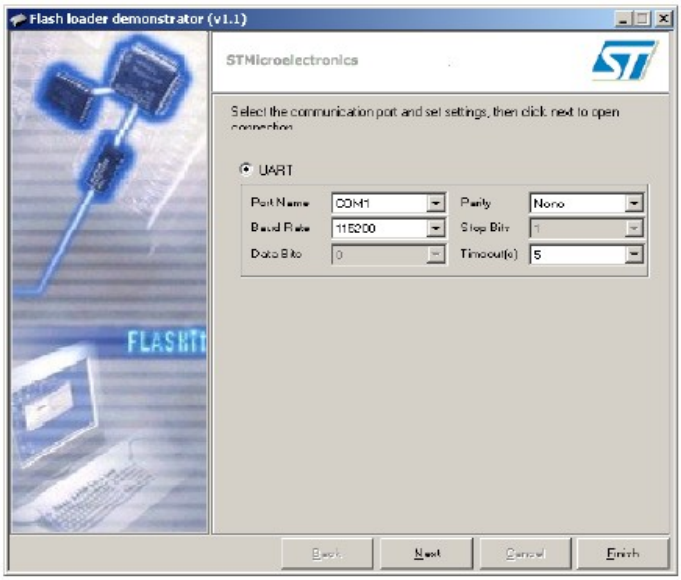

LedBlink
File Edit View Favorites Tools Help
Back Search Folders
Address C:\LedBlink
Go
File and Folder Tasks
Move this file
Copy this file
Publish this file to the Web
E-mail this file
Delete this file
Other Places
Local Disk (C:)
My Documents
Shared Documents
My Network Places
Details
LedBlink.dbi
LedBlink.mpp
LedBlink.rdb
LedBlink.prj
stm32f10x_vector.c
stm32f10x_conf.h
LedBlink.c
LedBlink.lst
LedBlink.map
sections_FLASH.ld
STM32F103_32K_20K_DEF.ld
stm32f10x_vector.lst
stm32f10x_vector.o
LedBlink.elf
LedBlink.hex
STM32F103_128K_20K_DEF.ld
STM32F103_128K_20K_FLASH.ld

2.5 Загрузка и тестирование...

(1) Подайте напряжение питания на плату JX-STM32. Переведите переключатель POWER в положение «включен». Загорится светодиодный индикатор питания.

(2) Подключите загрузочный кабель к порту USART1 на плате JX-STM32 и к COM-порту компьютера. Если ваш компьютер имеет только USB-порт, используйте преобразователь USB - RS-232 (кабель USB to COM). В качестве такового рекомендуется UCON-232S.

(3) Запустите программу **Flash Loader Demonstrator**.



(4) Установите параметры для загрузки:

(4.1) **Port Name** - выберите COM-порт.

(4.2) **Baud Rate** - установите скорость передачи равной **115200 бит в секунду**.

(4.3) **Data Bit** - установите длину байта данных равной **8 бит**.

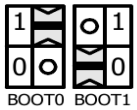
(4.4) **Parity** - установите проверку четности как **None**.

(4.5) **Stop Bit** - установите количество стоп-битов равным **1**.

(4.6) Timeout(s) - установите время ожидания ответа равным **5 секундам**.

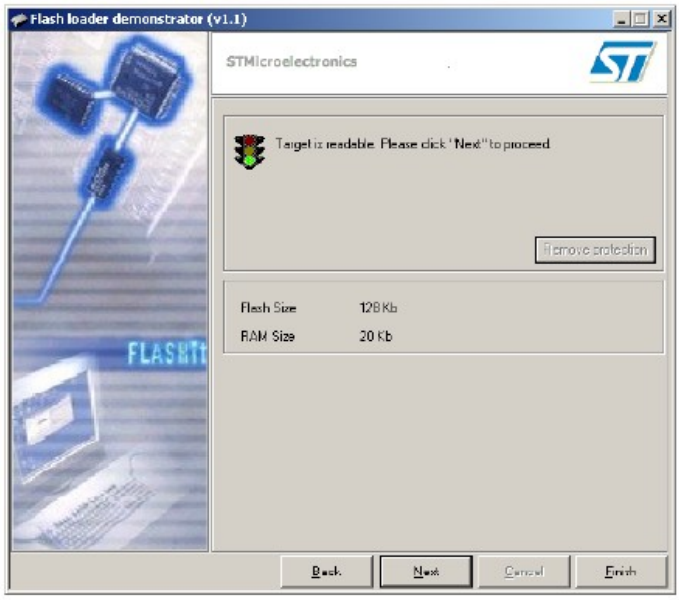
После установки все эти параметры будут использоваться в последующей работе. Пользователь может изменить их в любое время.

(5) Установите плату JX-STM32 в режим программирования. Установите перемычку **BOOT0** в позицию «**1**», перемычку **BOOT1** в позицию «**0**» и нажмите однократно кнопку **RESET**.

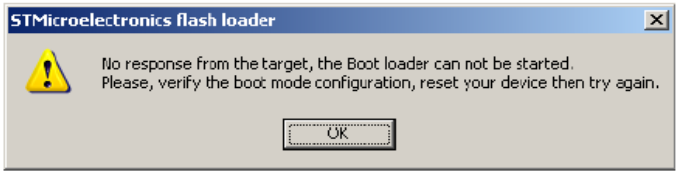


0	x	USER FLASH
1	1	Embed. SRAM
1	0	System MEM.

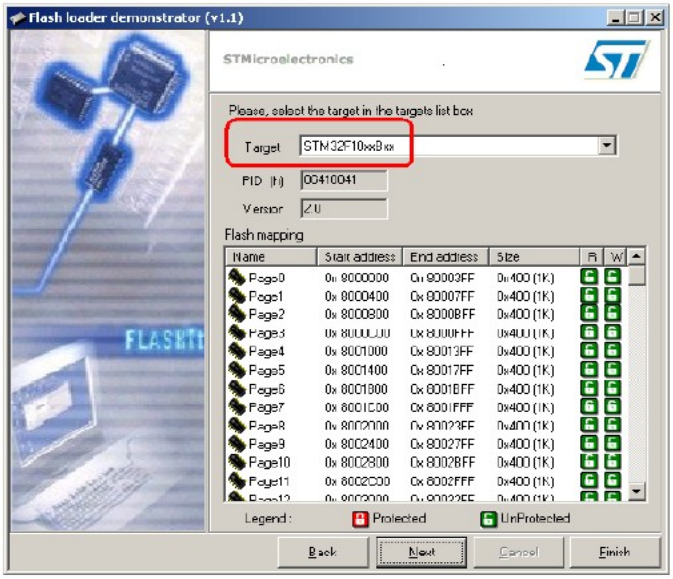
(6) В программе **Flash Loader Demonstrator** однократно нажмите кнопку **Next**. Если подключение установлено, на главном окне программы будет отображаться объем памяти используемого микроконтроллера. В нашем случае это **128 кБ flash-памяти и 20 кБ ОЗУ**. Однократно нажмите кнопку **Next** для перехода к следующему шагу.



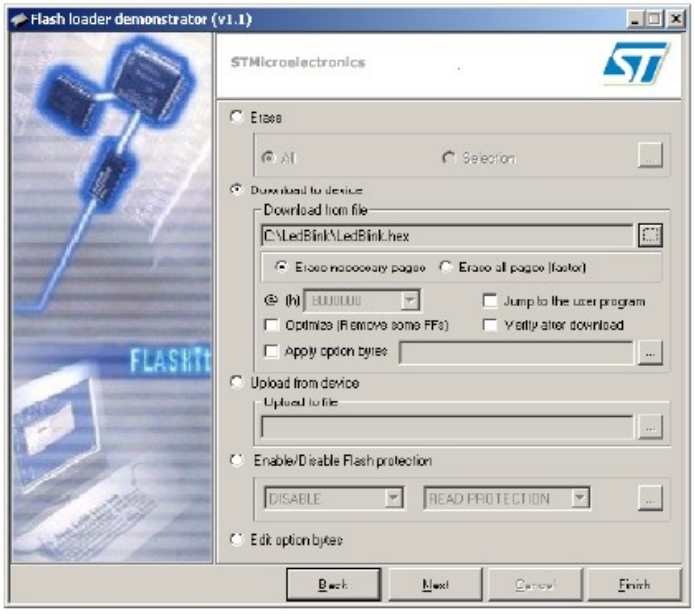
При возникновении некоторых ошибок появится диалоговое окно с сообщением об ошибке. Вам необходимо вернуться назад на все шаги и проверить все подключения, настройки параметров подключений и работоспособность COM-порта.



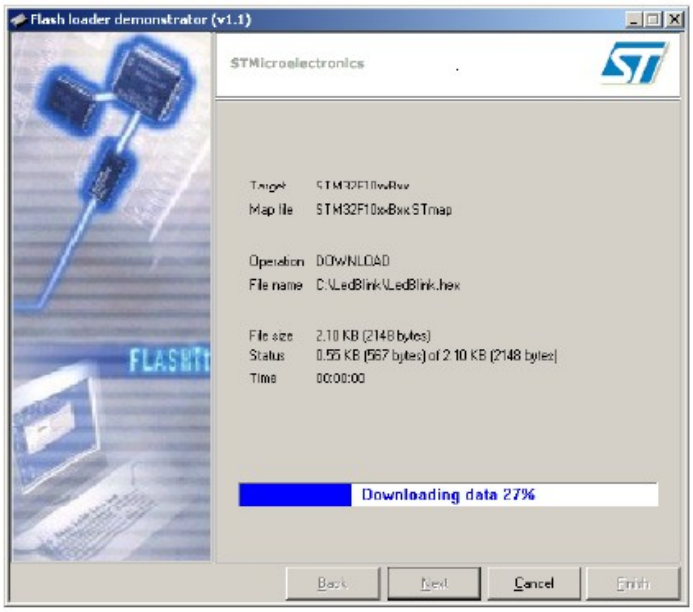
(7) Программа **Flash Loader Demonstrator** показывает информацию об используемом микроконтроллере, как показано на картинке ниже. Вы можете установить опцию защиты кода в этом окне. Нажмите кнопку **Next** для перехода к следующему шагу.



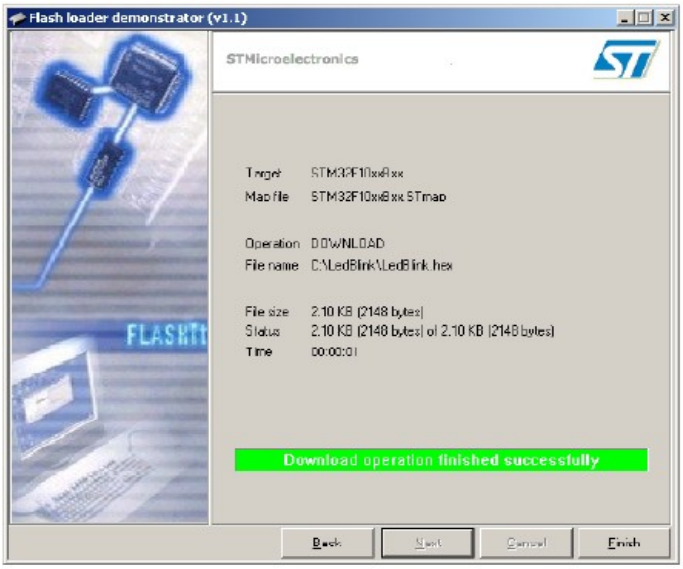
(8) Появится загрузочное окно. Выберите флаг **Download to device**. Выберите файл ***LedBlink.hex*** из ***C:\LedBlink\LedBlink.hex*** и нажмите кнопку **Next** для запуска загрузки.



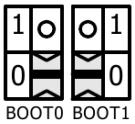
(9) Затем программа Flash Loader Demonstrator переходит в режим загрузки данных в микроконтроллер и показывает как проходит процесс загрузки.



(10) После завершения этого процесса, программа показывает информацию об используемом микроконтроллере, путь к hex-файлу и используемый объем памяти.



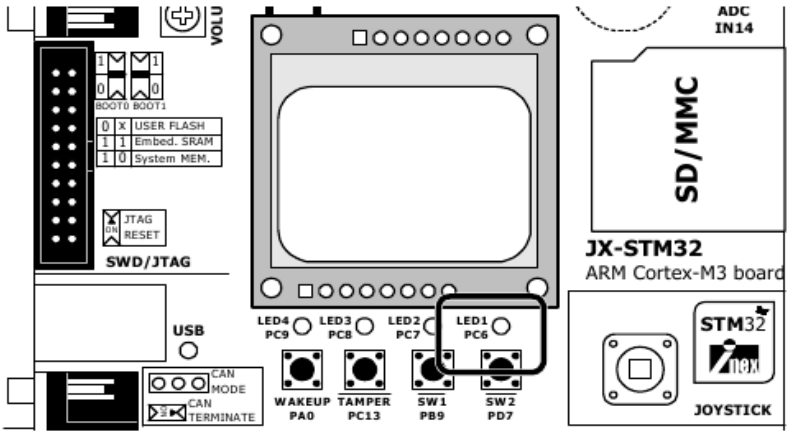
(11) Запустите загруженный код. Установите перемычки **BOOT0** и **BOOT1** в позицию «**0**» и однократно нажмите кнопку **RESET** для запуска программы.



0	X	USER FLASH
1	1	Embed. SRAM
1	0	System MEM.

(12) Наблюдайте работу платы JX-STM32.

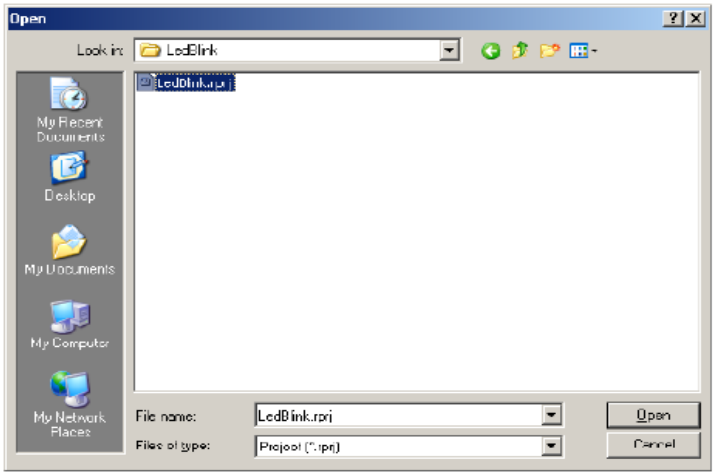
Светодиод на порту PC6 на плате JX-STM32 начинает мигать



2.6 Открытие существующего проекта для редактирования

В случае необходимости редактирования существующего проекта, выберите пункт меню Project->Open Project и укажите путь к используемому проекту. Расширение для файлов проектов Ride7 - *.rprj.

Например, откроем проект ***LedBlink*** для изменения. Открытие данного проекта показано на картинке ниже. После этого редактируем исходный файл на С или изменяем что-нибудь соответственно из ваших настроек.



3: Система генераторов STM32

Микроконтроллер STM32 содержит в себе много периферийных модулей. В состав каждого модуля входит независимый источник тактовой частоты. На Рисунке 3-1 показана система генераторов тактовой частоты микроконтроллера STM32F103VBT6

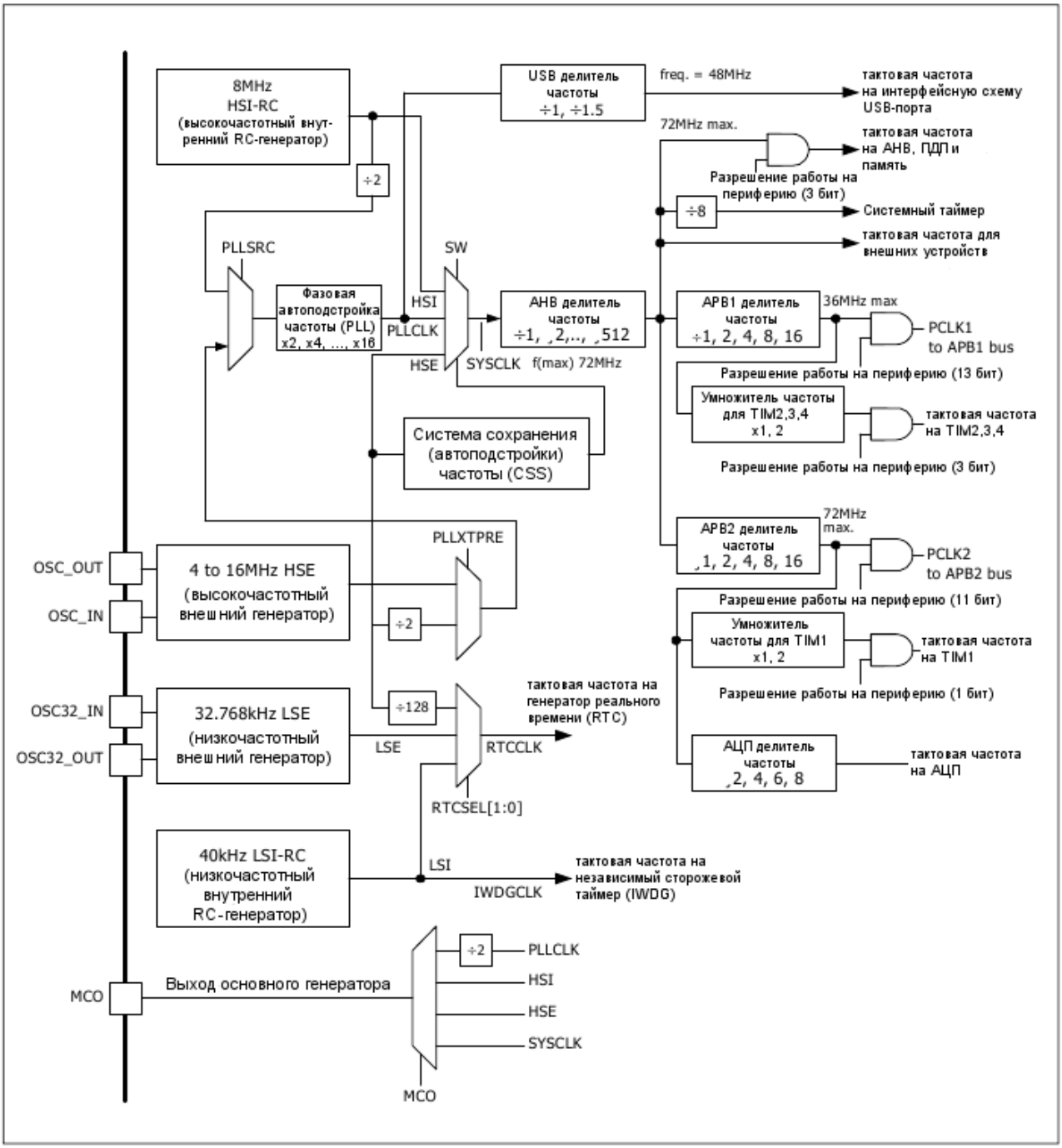


Рисунок 3-1. Схема системы генераторов контроллера STM32F103VBT6

3.1 Работа системы генераторов STM32

Три различных источника сигнала могут быть использованы для получения тактовой частоты системы (SYSCLK):

- Высокочастотный внутренний (HSI) генератор;
- Высокочастотный внешний (HSE) генератор;
- Система фазовой автоподстройки частоты (PLL);

Эти устройства имеют следующие вторичные источники сигнала:

- Низкочастотный внутренний RC-генератор на 40 кГц (LSI RC), который управляет независимым сторожевым таймером (IWDG) и может быть использован генератором реального времени (RTC) для автоматического включения из режима останова (Stop) и резервного режима (Standby);
- Низкочастотный внешний кварцевый генератор на 32,768 кГц (LSE кварц), который может управлять тактовой частотой для генератора реального времени (RTCCLK);

Каждый источник сигнала может независимо друг от друга включаться или выключаться, когда не используется, для оптимизации энергопотребления.

Рассматривая схему системы генераторов на Рисунке 3-1, разработчик может выбрать тактовую частоту системы из 3-х источников сигнала: высокочастотного внутреннего генератора (HSI), высокочастотного внешнего генератора (HSE) и системы фазовой автоподстройки частоты (PLL). Максимально генерируемая частота - 72 Мгц (с помощью системы фазовой автоподстройки частоты (PLL)). После этого указывается значение делителя частоты для установки частоты периферийных устройств или тактовой частоты расширенной высокочастотной шины (AHB).

Расширенные периферийные шины (APB1 и APB2) могут использовать значения делителей частоты для определения частот периферийных устройств. Максимальная частота для APB1 - 36 Мгц, для APB2 - 72 Мгц.

Модуль периферийных устройств APB1 включает в себя:

- TIM2 (Таймер 2);
- TIM3 (Таймер 3);
- TIM4 (Таймер 4);
- WWDG (Оконный сторожевой таймер);
- SPI2 (Последовательный периферийный интерфейс 2);
- USART2 (Универсальный синхронно-асинхронный приемопередатчик 2);
- USART3 (Универсальный синхронно-асинхронный приемопередатчик 3);
- I2C1 (Модуль I^2C - шины 1);
- I2C2 (Модуль I^2C - шины 2);
- USB (Универсальная последовательная шина);
- CAN (Модуль управления локальной сетью);

- BKP (резервный регистр);
- PWR (модуль управления питанием);

Модуль периферийных устройств APB2 включает в себя:

- GPIO A (Основное УВВ A);
- GPIO B (Основное УВВ B);
- GPIO C (Основное УВВ C);
- GPIO D (Основное УВВ D);
- GPIO E (Основное УВВ E);
- ADC1 (Аналого-цифровой преобразователь 1);
- ADC2 (Аналого-цифровой преобразователь 2);
- TIM1 (Таймер 1);
- SPI1 (Последовательный периферийный интерфейс 1);
- USART1 (Универсальный синхронно-асинхронный приемопередатчик 1);

Тактовая частота системы (SYSCLC) ограничивается 64 Мгц когда в качестве источника сигнала выбраны высокочастотный внутренний генератор (HSI) или система фазовой автоподстройки частоты (PLL).

3.2 Регистры сброса и управления частотой

Каждый периферийный модуль из устройств, входящих в состав STM32, имеет свой независимый источник тактовых сигналов. Группа регистров управления этими источниками является **RCC**-регистрами **(Reset and Clock Control)** т.е. регистрами сброса и управления частотой. Эта группа состоит из 10 регистров управления, таких как:

1. **RCC_CR**: регистр управления сигналом тактовой частоты
2. **RCC_CFGR**: регистр формирования сигнала тактовой частоты
3. **RCC_CIR**: регистр прерывания сигнала тактовой частоты
4. **RCC_APB2RSTR**: регистр сброса периферии APB2
5. **RCC_APB1RSTR**: регистр сброса периферии APB1
6. **RCC_AHBENR**: регистр разрешения работы генератора тактовой частоты на периферию расширенной высокочастотной шины (AHB)
7. **RCC_APB2ENR**: регистр разрешения работы генератора тактовой частоты на периферию APB2
8. **RCC_APB1ENR**: регистр разрешения работы генератора тактовой частоты на периферию APB1
9. **RCC_BDCR**: резервный регистр области управления (??)
10. **RCC_CSR**: регистр управления и просмотра состояния

Смотрите справочное руководство для получения более детальной информации по всем регистрам. Его свободно можно загрузить с сайта **www.st.com**.

3.3 Структура RCC-регистров

Структура RCC-регистров определена как **RCC_TypeDef** в файле ***stm32f10x_map.h*** и подробно показана ниже:

```
typedef struct
{
  vu32 CR;
  vu32 CFGR;
  vu32 CIR;
  vu32 APB2RSTR;
  vu32 APB1RSTR;
  vu32 AHBENR;
  vu32 APB2ENR;
  vu32 APB1ENR;
  vu32 BDCR;
  vu32 CSR;
} RCC_TypeDef;
```

Фирма STMicroelectronics создала аппаратно-зависимые библиотеки для каждого важного модуля, в том числе и для группы RCC-регистров. Для полноты информации посмотрите файл **STM32_Firmware_Library.pdf** из каталога ***C:\Program Files\Raisonance\Ride\lib\ARM\STM32F10x_LIB*** после установки Ride7.

3.4 Функции библиотеки RCC-регистров

Для доступа ко всем функциям аппаратно-зависимой библиотеки RCC-регистров вам необходимо скопировать файл ***stm32f10x_conf.h*** в каталог вашего проекта и объявить аппаратно-зависимую библиотеку RCC-регистров с помощью команды ***#define _RCC*** в заголовочном файле.

Данная библиотека содержит 15 функций для работы с RCC-регистрами. Тем не менее в этой главе будут описаны несколько необходимых функций из нее, такие как:

3.4.1 RCC_DeInit

Сбрасывает RCC-настройки генератора тактовой частоты к значениям по умолчанию.

Синтаксис

```
void RCC_DeInit(void)
```

3.4.2 RCC_HSEConfig

Настраивает внешний высокочастотный генератор (HSE);

Синтаксис

```
void RCC_HSEConfig(u32 RCC_HSE)
```

Параметр

RCC_HSE может принимать следующие значения:

RCC_HSE_OFF — HSE-генератор отключен

RCC_HSE_ON — HSE-генератор включен

RCC_HSE_Bypass — HSE-генератор шунтируется внешним генератором тактовой частоты

3.4.3 RCC_WaitForHSEStartUp

Ожидает вступления новых настроек HSE-генератора

Синтаксис

```
ErrorStatus RCC_WaitForHSEStartUp(void)
```

Параметр

SUCCESS — HSE-генератор настроен и готов к работе

ERROR — HSE-генератор не готов к работе

3.4.4 RCC_HSICmd

Разрешает или запрещает работу внутреннего высокочастотного генератора (HSI).

Синтаксис

```
void RCC_HSICmd(FunctionalState NewState)
```

Параметр

NewState может принимать следующие значения:

ENABLE — HSI-генератор отключен

DISABLE — HSI-генератор включен

3.4.5 RCC_PLLConfig

Настраивает источник сигнала тактовой частоты и коэффициент умножения частоты для системы фазовой автоподстройки (PLL).

Синтаксис

```
void RCC_PLLConfig(u32 RCC_PLLSource, u32 RCC_PLLMul)
```

Параметры

RCC_PLLSource может принимать следующие значения:

RCC_PLLSource_HSI_Div2 — устанавливает тактовую частоту равной половине частоты внутреннего высокочастотного генератора (HSI/2)

RCC_PLLSource_HSE_Div1 — устанавливает тактовую частоту равной частоте внешнего высокочастотного генератора (HSE)

RCC_PLLSource_HSE_Div2 — устанавливает тактовую частоту равной половине частоты внешнего высокочастотного генератора (HSE/2)

RCC_PLLMul может быть от 2 до 16:

RCC_PLLMul_2 — частота со входа системы фазовой автоподстройки частоты (PLL) умножается на 2

.

.

RCC_PLLMul_16 — частота со входа системы фазовой автоподстройки частоты (PLL) умножается на 16

3.4.6 RCC_PLLCmd

Разрешает или запрещает работу системы фазовой автоподстройки частоты (PLL).

Синтаксис

```
void RCC_PLLCmd(FunctionalState NewState)
```

Параметр

NewState может принимать следующие значения:

ENABLE — система фазовой автоподстройки (PLL) включена

DISABLE — система фазовой автоподстройки (PLL) отключена

3.4.7 RCC_SYSCLKConfig

Настраивает тактовую частоту системы (SYSCLK).

Синтаксис

```
void RCC_SYSCLKConfig(u32 RCC_SYSCLKSource)
```

Параметр

RCC_SYSCLKSource может принимать следующие значения:

RCC_SYSCLKSource_HSI — устанавливает тактовую частоту системы равной частоте внутреннего высокочастотного генератора (HSI)

RCC_SYSCLKSource_HSE — устанавливает тактовую частоту системы равной частоте внешнего высокочастотного генератора (HSE)

RCC_SYSCLKSource_PLLCLK — устанавливает тактовую частоту системы равной частоте на выходе системы фазовой автоподстройки частоты (PLL)

3.4.8 RCC_GetSYSCLKSource

Определяет тактовую частоту системы (SYSCLK).

Синтаксис

```
u8 RCC_GetSYSCLKSource(void)
```

Параметры

отсутствуют

Возвращаемые значения

0x00 — в качестве тактовой частоты системы используется частота внутреннего высокочастотного генератора (HSI)

0x04 — в качестве тактовой частоты системы используется частота внешнего высокочастотного генератора (HSE)

0x08 — в качестве тактовой частоты системы используется частота на выходе системы фазовой автоподстройки частоты (PLL)

3.4.9 RCC_HCLKConfig

Настраивает тактовую частоту AHB (расширенной высокочастотной шины) (HCLK).

Синтаксис

```
void RCC_HCLKConfig(u32 RCC_HCLKSource)
```

Параметр

RCC_HCLKSource может принимать следующие значения:

RCC_SYSCLK_Div1 — (HCLK = SYSCLK)

RCC_SYSCLK_Div2 — (HCLK = SYSCLK/2)

RCC_SYSCLK_Div4 — (HCLK = SYSCLK/4)

RCC_SYSCLK_Div8 — (HCLK = SYSCLK/8)

RCC_SYSCLK_Div16 — (HCLK = SYSCLK/16)

RCC_SYSCLK_Div64 — (HCLK = SYSCLK/64)

RCC_SYSCLK_Div128 — (HCLK = SYSCLK/128)

RCC_SYSCLK_Div256 — (HCLK = SYSCLK/256)

RCC_SYSCLK_Div512 — (HCLK = SYSCLK/512)

3.4.10 RCC_PCLK1Config

Настраивает тактовую частоту низкочастотной расширенной периферийной шины APB1 (PCLK1).

Синтаксис

```
void RCC_PCLK1Config(u32 RCC_PCLK1)
```

Параметр

RCC_PCLK1 может принимать следующие значения:

RCC_HCLK_Div1 — (PCLK1 = HCLK)

RCC_HCLK_Div2 — (PCLK1 = HCLK/2)

RCC_HCLK_Div4 — (PCLK1 = HCLK/4)

RCC_HCLK_Div8 — (PCLK1 = HCLK/8)

RCC_HCLK_Div16 — (PCLK1 = HCLK/16)

3.4.11 RCC_PCLK2Config

Настраивает тактовую частоту высокочастотной расширенной периферийной шины APB2 (PCLK2).

Синтаксис

```
void RCC_PCLK1Config(u32 RCC_PCLK2)
```

Параметр

RCC_PCLK2 может принимать следующие значения:

RCC_HCLK_Div1 — (PCLK2= HCLK)

RCC_HCLK_Div2 — (PCLK2= HCLK/2)

RCC_HCLK_Div4 — (PCLK2 = HCLK/4)

RCC_HCLK_Div8 — (PCLK2 = HCLK/8)

RCC_HCLK_Div16 — (PCLK2 = HCLK/16)

3.4.12 RCC_GetFlagStatus

Показывает, какой из специальных RCC-флагов установлен или нет.

Синтаксис

```
FlagStatus RCC_GetFlagStatus(u8 RCC_FLAG)
```

Параметр

RCC_FLAG может принимать следующие значения:

RCC_FLAG_HSIRDY — частота внутреннего высокочастотного генератора (HSI) настроена и он готов к работе

RCC_FLAG_HSERDY — частота внешнего высокочастотного генератора (HSE) настроена и он готов к работе

RCC_FLAG_PLLRDY — частота на выходе системы фазовой автоподстройки частоты (PLL) настроена и система готова к работе

RCC_FLAG_LSERDY — частота внешнего низкочастотного генератора (LSE) настроена и он готов к работе

RCC_FLAG_LSIRDY — частота внутреннего низкочастотного генератора (LSI) настроена и он готов к работе

RCC_FLAG_PINRST — Сброс по контакту сброса

RCC_FLAG_PORRST — Сброс при включении и выключении питания

RCC_FLAG_SFTRST — Программный сброс

RCC_FLAG_IWDGRST — Сброс от независимого сторожевого таймера

RCC_FLAG_WWDGRST — Сброс от оконного сторожевого таймера

RCC_FLAG_LPWRRST — Сброс при низком напряжении питания

Возвращаемые значения

SET — состояние RCC-флагов установлено

RESET — состояние RCC-флагов не установлено

3.4.13 RCC_AHBPeriphClockCmd

Разрешает или запрещает подачу сигнала тактовой частоты на устройства расширенной высокочастотной шины (AHB).

Синтаксис

```
void RCC_AHBPeriphClockCmd(u32 RCC_AHBPeriph,
FunctionalState NewState)
```

Параметры

RCC_AHBPeriph может принимать следующие значения:

RCC_AHBPeriph_DMA — подача тактовой частоты на модуль ПДП

RCC_AHBPeriph_SRAM — подача тактовой частоты на ОЗУ

RCC_AHBPeriph_FLITF — подача тактовой частоты на flash-память

NewState может принимать следующие значения:

ENABLE — доступ разрешен

DISABLE — доступ запрещен

Только RCC_AHBPeriph_SRAM и RCC_AHBPeriph_FLITF могут быть запрещены во времени нахождения в режиме сна (Sleep).

3.4.14 RCC_APB2PeriphClockCmd

Разрешает или запрещает подачу сигнала тактовой частоты на устройства высокочастотной расширенной периферийной шины (APB2).

Синтаксис

```
void RCC_APB2PeriphClockCmd(u32 RCC_APB2Periph,
FunctionalState NewState)
```

Параметры

RCC_APB2Periph может принимать следующие значения:

RCC_APB2Periph_AFIO — сигнал тактовой частоты подается на другие модули ввода-вывода

RCC_APB2Periph_GPIOA — подача тактовой частоты на GPIO A

RCC_APB2Periph_GPIOB — подача тактовой частоты на GPIO B

RCC_APB2Periph_GPIOC — подача тактовой частоты на GPIO C

RCC_APB2Periph_GPIOD — подача тактовой частоты на GPIO D

RCC_APB2Periph_GPIOE — подача тактовой частоты на GPIO E

RCC_APB2Periph_ADC1 — подача тактовой частоты на АЦП 1

RCC_APB2Periph_ADC2 — подача тактовой частоты на АЦП 2

RCC_APB2Periph_TIM1 — подача тактовой частоты на TIM1

RCC_APB2Periph_SPI1 — подача тактовой частоты на SPI 1

RCC_APB2Periph_USART1 — подача тактовой частоты на USART 1

RCC_APB2Periph_ALL — подача тактовой частоты на всю шину APB2

NewState может принимать следующие значения:

ENABLE — доступ разрешен

DISABLE — доступ запрещен

3.4.15 RCC_APB1PeriphClockCmd

Разрешает или запрещает подачу сигнала тактовой частоты на устройства низкочастотной расширенной периферийной шины (APB1).

Синтаксис

```
void RCC_APB1PeriphClockCmd(u32 RCC_APB1Periph,
FunctionalState NewState)
```

Параметры

RCC_APB1Periph может принимать следующие значения:

RCC_APB1Periph_TIM2 — подача тактовой частоты на TIM 2

RCC_APB1Periph_TIM3 — подача тактовой частоты на TIM 3

RCC_APB1Periph_TIM4 — подача тактовой частоты на TIM 4

RCC_APB1Periph_WWDG — подача тактовой частоты на оконный сторожевой таймер (WWDG)

RCC_APB1Periph_SPI2 — подача тактовой частоты на SPI 2

RCC_APB1Periph_USART2 — подача тактовой частоты на USART 1

RCC_APB1Periph_USART3 — подача тактовой частоты на USART 2

RCC_APB1Periph_I2C1 — подача тактовой частоты на I^2C 1

RCC_APB1Periph_I2C2 — подача тактовой частоты на I^2C 2

RCC_APB1Periph_USB — подача тактовой частоты на USB

RCC_APB1Periph_CAN — подача тактовой частоты на CAN

RCC_APB1Periph_BKP — подача тактовой частоты резервный регистр

RCC_APB1Periph_PWR — подача тактовой частоты на схему управления питанием

RCC_APB1Periph_ALL — подача тактовой частоты на всю шину APB1

NewState может принимать следующие значения:

ENABLE — доступ разрешен

DISABLE — доступ запрещен

3.5 Установка времени ожидания в регистре FLASH-ACR

По тактовой частоте системы (SYSCLK) можно примерно определить время ожидания, состоящее из времени доступа к flash-памяти и соответствующей обработки данных центральным процессором:

0 состояние ожидания: при частоте от 0 до 24 МГц (нет ожидания)

1 состояние ожидания: при частоте от 24 до 48 МГц

2 состояние ожидания: при частоте от 48 до 72 МГц.

Установка времени ожидания возможна при записи данных в регистр **FLASH_ACR** с помощью следующих функций из аппаратно-зависимой библиотеки:

3.5.1 FLASH_SetLatency

Устанавливает значение кода времени ожидания.

Синтаксис

```
void FLASH_SetLatency(u32 FLASH_Latency)
```

Параметр

FLASH_Latency может принимать следующие значения:

FLASH_Latency_0 — установка 0-го состояния ожидания

FLASH_Latency_1 — установка 1-го состояния ожидания

FLASH_Latency_2 — установка 2-го состояния ожидания

3.5.2 FLASH_PrefetchBufferCmd

Разрешает или запрещает использование предварительного буфера.

Синтаксис

```
void FLASH_PrefetchBufferCmd(u32 FLASH_PrefetchBuffer)
```

Параметр

FLASH_PrefetchBuffer может принимать следующие значения:

FLASH_PrefetchBuffer_Enable — разрешение использования предварительного буфера

FLASH_PrefetchBuffer_Disable — запрещение использования предварительного буфера

3.6 Установка тактовой частоты системы

При каждой разработке кода программы для STM32 первым шагом является установка тактовой частоты системы. Это можно сделать с помощью **настройки RCC-конфигурации** и **FLASH_ACR**.

3.6.1 Общий порядок действий

(1) Разрешите работу с внешним высокочастотным генератором (HSE) или внутренним высокочастотным генератором (HSI).

(2) Установите наличие флага готовности для источника сигнала тактовой частоты.

(3) После установления источника сигнала тактовой частоты в стабильное состояние, настройте параметры всех периферийных модулей STM32.

(4) Установите время ожидания

(5) Если требуется система фазовой автоподстройки частоты (PLL), выполните следующие шаги.

(5.1) Установите источником сигнала тактовой частоты систему фазовой автоподстройки частоты (PLL).

(5.2) Укажите коэффициент умножителя частоты (от 2 до 16).

(5.3) Разрешите использование сигнала тактовой частоты с системы фазовой автоподстройки частоты (PLL).

(5.4) Установите наличие готовности системы фазовой автоподстройки частоты (PLL).

(6) Выберите источник сигнала тактовой частоты системы.

(7) Дождитесь готовности источника сигнала тактовой частоты системы.

3.6.2 Пример кода по настройке тактовой частоты системы в STM32

Пример 1. В качестве источника тактовой частоты системы (8 МГц) выбран внутренний высокочастотный генератор (HSI). Имя функции - ***RCC_setup1***. Ее код на языке С представлен ниже:

```
#include <stm32f10x_lib.h>              //Заголовочный файл для STM32
void RCC_setup1()
{
  RCC_DeInit();
    //Разрешаем работу с внутренним 8 МГц генератором
  RCC_HSICmd(ENABLE);
    //Ждем пока HSI-генератор не будет готов
  while(RCC_GetFlagStatus(RCC_FLAG_HSIRDY)==RESET);
    //Настраиваем частоты периферийных устройств
  RCC_HCLKConfig(RCC_SYSCLK_Div1);      //HCLK = SYSCLK = 8 МГц
  RCC_PCLK2Config(RCC_HCLK_Div1);       //PCLK2 = HCLK/1 = 8 МГц
  RCC_PCLK1Config(RCC_HCLK_Div2);       //PCLK1 = HCLK/2 = 4 МГц
  RCC_ADCCLKConfig(RCC_PCLK2_Div2);     //ADCCLK = PCLK2/2 = 4 МГц
    //Устанавливаем время ожидания в состояние 0
  FLASH_SetLatency(FLASH_Latency_0);
    //Разрешаем доступ к предварительному буферу
  FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
    //Выбираем HSI-генератор как источник SYSCLK
  RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);
    //Ждем готовности источника SYSCLK
  while(RCC_GetSYSCLKSource()!=0x00);
}

int main()
{
  RCC_setup1();            //Пуск с настройкой источника тактовой частоты
  ......................
}
```

Пример 2: В качестве источника тактовой частоты системы выбран внешний высокочастотный генератор (HSE). Имя функции - ***RCC_setup2***. Ее код на языке С представлен ниже:

```
#include <stm32f10x_lib.h>              //Заголовочный файл для STM32
void RCC_setup1()
{
  ErrorStatus HSEStartUpStatus;         //Объявляем переменную ErrorStatus
  RCC_DeInit();
    //Разрешаем работу с внешним высокочастотным генератором (HSE)
  RCC_HSEConfig(RCC_HSE_ON);
    //Ждем пока HSE-генератор не будет готов
  HSEStartUpStatus = RCC_WaitForHSEStartUp();
    //Если HSE-генератор готов к работе
  if(HSEStartUpStatus == SUCCESS)
  {
     //Настраиваем частоты периферийных устройств
    RCC_HCLKConfig(RCC_SYSCLK_Div1);    //HCLK = SYSCLK
    RCC_PCLK2Config(RCC_HCLK_Div1);     //PCLK2 = HCLK/1
    RCC_PCLK1Config(RCC_HCLK_Div2);     //PCLK2 = HCLK/2
    RCC_ADCCLKConfig(RCC_PCLK2_Div6);   //ADCCLK = PCLK2/6
```

```
    //Устанавливаем время ожидания в состояние 0
   FLASH_SetLatency(FLASH_Latency_0);
    //Разрешаем доступ к предварительному буферу
   FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
    //Выбираем HSE-генератор как источник SYSCLK
   RCC_SYSCLKConfig(RCC_SYSCLKSource_HSE);
    //Ждем готовности источника SYSCLK
   while(RCC_GetSYSCLKSource()!=0x04);
  }
}

int main()
{
 RCC_setup2();
 .....................
}
```

Пример 3: В качестве источника тактовой частоты системы выбран внешний высокочастотный генератор (HSE). Частота кварца — 8 МГц. Включаем фазовую автоподсторойку частоты (PLL) для ее умножения в 9 раз, чтобы сделать тактовую частоту системы равной 72 МГц. Имя функции - ***RCC_setup3***. Ее код на языке С представлен ниже:

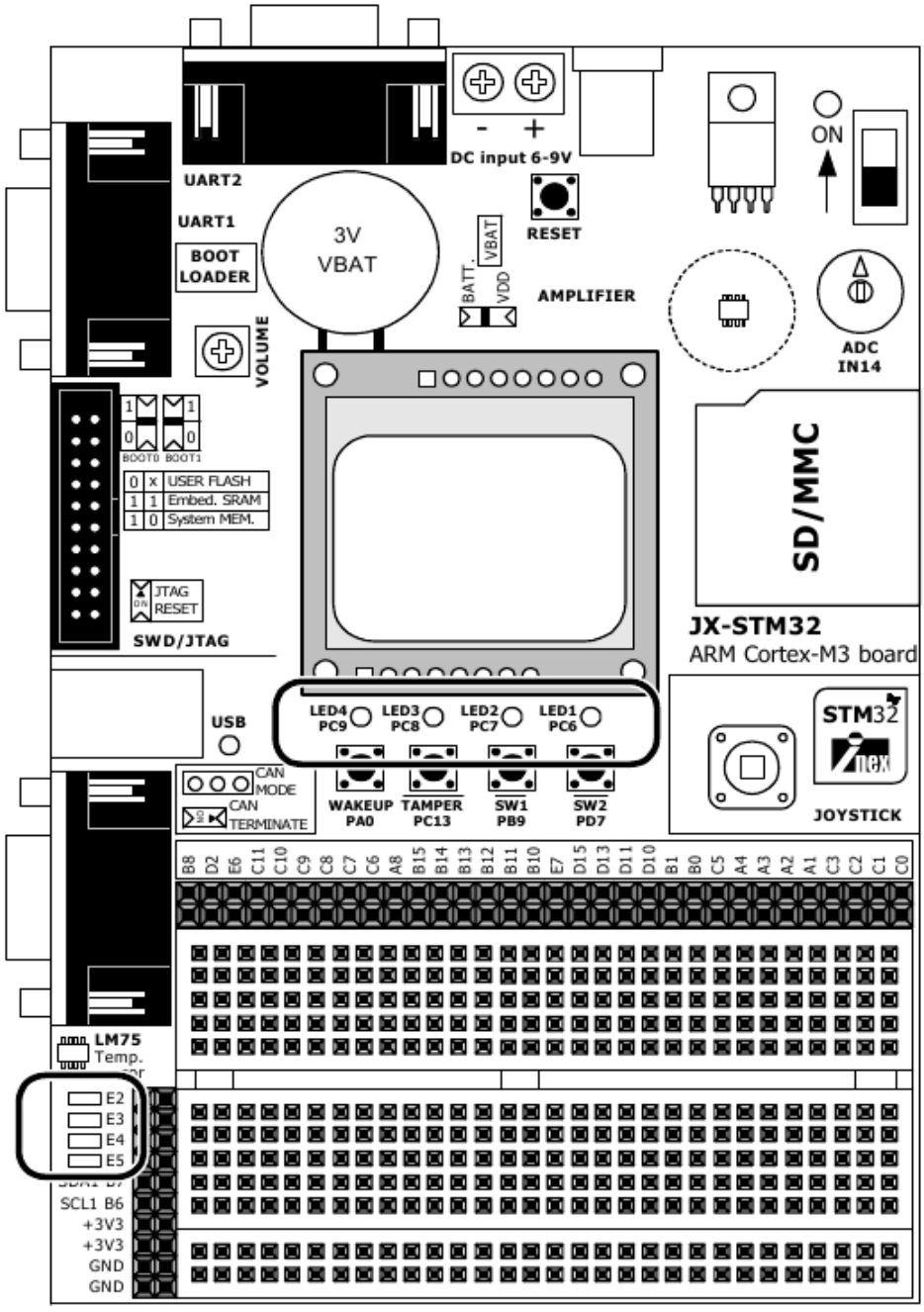
```
#include <stm32f10x_lib.h>              //Заголовочный файл для STM32
void RCC_setup3()
{
 ErrorStatus HSEStartUpStatus;         //Объявляем переменную ErrorStatus
 RCC_DeInit();
   //Разрешаем работу с внешним высокочастотным генератором (HSE)
 RCC_HSEConfig(RCC_HSE_ON);
   //Ждем пока HSE-генератор не будет готов
 HSEStartUpStatus = RCC_WaitForHSEStartUp();
 if(HSEStartUpStatus == SUCCESS)
 {
    //Настраиваем частоты периферийных устройств
  RCC_HCLKConfig(RCC_SYSCLK_Div1);     //HCLK = SYSCLK
  RCC_PCLK2Config(RCC_HCLK_Div1);      //PCLK2 = HCLK/1 = 72 МГц
  RCC_PCLK1Config(RCC_HCLK_Div2);      //PCLK1 = HCLK/2 = 36 МГц
  RCC_ADCCLKConfig(RCC_PCLK2_Div4);    //ADCCLK = PCLK2/4 = 18 МГц
    //Устанавливаем время ожидания в состояние 2
  FLASH_SetLatency(FLASH_Latency_2);
    //Разрешаем доступ к предварительному буферу
  FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
    //Выбираем HSE-генератор с PLL (PLLCLK = 8 МГц * 9 = 72 МГц)
  RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
    //Разрешаем запуск PLL
  RCC_PLLCmd(ENABLE);
    //Ждем, пока PLL не будет готов
  while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);
    //Устанавливаем PLL как источник тактовой частоты системы
  RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
    //Ждем готовности источника SYSCLK
  while(RCC_GetSYSCLKSource() != 0x08);
 }
}

int main()
{
 RCC_setup3();
 .....................
}
```

Все примеры в этой главе используют источник сигнала тактовой частоты, равный 8 МГц.

4: Примеры использования основного УВВ

Эксперимент - 1 : Управление портами вывода (от PC6 до PC9 и от PE2 до PE5)

Этот эксперимент описывает программирование выходных портов микроконтроллера STM32F103VBT6. Происходит управление простыми устройствами на выходах микроконтроллера: светодиодов на портах от PC6 до PC9 и от PE2 до PE5 на плате JX-STM32.



Общий порядок действий

(1.1) Создать проект ***output_01***.

(1.2) Создать исходный файл ***output_01.c*** на языке С в соответствии с Листингом L1-1.

(1.3) Скомпилировать файл проекта и загрузить его в микроконтроллер STM32F103VBT6 на плате JX-STM32. Более подробно об этом можно прочесть в главе 2.

(1.4) Запустить эту программу и наблюдать за работой платы.

Светодиоды на портах от PC6 до PC9 и от PE2 до PE5 мигают 1 раз в 0.5 секунды

```
//*****************************************************************************************//
// Программа: Проверка портов вывода от PC6 до PC9
// Описание: Мигание светодиодов на портах от PC6 до PC9
// Имя файла: output_01.c
// Компилятор C: RkitARM 1.03.0003 для Ride7
// Аппаратно-зависимая библиотека: STMicroelectronics FWlLib V1.0
//*****************************************************************************************//
#include "stm32f10x_lib.h"                          //Подключаем заголовочный файл для STM32

//************************************ Функция задержки ************************************//
void delay(unsigned long ms)
// Задержка 1 мс за цикл при кварце 8.0 Мгц и PLL9x или SYSCLK = 72 Мгц
{
  volatile unsigned long i, j;
  for(i = 0; i < ms; i++)
    for(j = 0; j < 5525; j++);
}

//*******************************Функция установки RCC-регистров*******************************//
void RCC_setup(void)
{
  ErrorStatus HSEStartUpStatus;                     //Переменная статуса ошибки
  RCC_DeInit();                                     //Сброс RCC-системы (для отладки)
  RCC_HSEConfig(RCC_HSE_ON);                        //Разрешаем работу HSE-генератора
  HSEStartUpStatus = RCC_WaitForHSEStartUp();       //Ждем пока HSE-генератор не будет готов
  if(HSEStartUpStatus == SUCCESS)
  {
      //Установка частот периферийных устройств
    RCC_HCLKConfig(RCC_SYSCLK_Div1);                //HCLK = SYSCLK
    RCC_PCLK2Config(RCC_HCLK_Div1);                 //PCLK2 = HCLK
    RCC_PCLK1Config(RCC_HCLK_Div2);                 //PCLK1 = HCLK/2
    FLASH_SetLatency(FLASH_Latency_2);              //Устанавливаем время ожидания в состояние 2
      //Разрешаем доступ к предварительному буферу
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
      //PLLCLK = 8 МГц * 9 = 72 МГц
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
      //Разрешаем работу с PLL
    RCC_PLLCmd(ENABLE);
      //Ожидаем готовность PLL
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);
      //Устанавливаем PLL как источник сигнала тактовой частоты системы
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
      //Ждем пока PLL не станет источником сигнала тактовой частоты системы
    while(RCC_GetSYSCLKSource() != 0x08);
  }
}

//************************** Функция настройки основного УВВ (GPIO) ***************************//
void GPIO_setup()
{
  GPIO_InitTypeDef GPIO_InitStructure;
    //Разрешаем подачу сигнала тактовой частоты на устройства шины APB2
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOE, ENABLE);
```

Листинг L1-1: output_01.c — С-файл для микроконтроллера STM32 содержит пример работы порта вывода. (начало)

```
    //Настраиваем PC6, PC7, PC8, PC9 как выход (подключены к светодиодам)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
    //Настраиваем PE2, PE3, PE4, PE5 как выход (подключены к светодиодам)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOE, &GPIO_InitStructure);
}

//******************************* Основная (главная) функция *********************************//
int main()
{
  RCC_setup();                                      //Настройка сигналов тактовых частот
  GPIO_setup();                                     //Настройка портов основного УВВ (GPIO)
  while(1)                                          //Бесконечный цикл
  {
    GPIO_WriteBit(GPIOC, GPIO_Pin_6, 1);            //Светодиод на PC6 включен
    GPIO_WriteBit(GPIOC, GPIO_Pin_7, 1);            //Светодиод на PC7 включен
    GPIO_WriteBit(GPIOC, GPIO_Pin_8, 1);            //Светодиод на PC8 включен
    GPIO_WriteBit(GPIOC, GPIO_Pin_9, 1);            //Светодиод на PC9 включен
    GPIO_WriteBit(GPIOE, GPIO_Pin_2, 1);            //Светодиод на PE2 включен
    GPIO_WriteBit(GPIOE, GPIO_Pin_3, 1);            //Светодиод на PE3 включен
    GPIO_WriteBit(GPIOE, GPIO_Pin_4, 1);            //Светодиод на PE4 включен
    GPIO_WriteBit(GPIOE, GPIO_Pin_5, 1);            //Светодиод на PE5 включен
    delay(500);
    GPIO_WriteBit(GPIOC, GPIO_Pin_6, 0);            //Светодиод на PC6 выключен
    GPIO_WriteBit(GPIOC, GPIO_Pin_7, 0);            //Светодиод на PC7 выключен
    GPIO_WriteBit(GPIOC, GPIO_Pin_8, 0);            //Светодиод на PC8 выключен
    GPIO_WriteBit(GPIOC, GPIO_Pin_9, 0);            //Светодиод на PC9 выключен
    GPIO_WriteBit(GPIOE, GPIO_Pin_2, 0);            //Светодиод на PE2 выключен
    GPIO_WriteBit(GPIOE, GPIO_Pin_3, 0);            //Светодиод на PE3 выключен
    GPIO_WriteBit(GPIOE, GPIO_Pin_4, 0);            //Светодиод на PE4 выключен
    GPIO_WriteBit(GPIOE, GPIO_Pin_5, 0);            //Светодиод на PE5 выключен
    delay(500);
  }
}
```

Описание кода

Этот код подключает заголовочный файл stm32f10x_lib.h в следующей строке:

```
#include <stm32f10x_lib.h>
```

Разработчик может иметь доступ ко всем ресурсам и аппаратно-зависимым библиотекам. (Сначала нужно скопировать файл **stm32f10x_conf.h** в каталог с вашим проектом).

Основная (главная) функция начинает свою работу с установки параметров сигналов тактовых частот в функции **RCC_setup**. Затем порты от PC6 до PC9 и от PE2 до PE5 устанавливаются как выходы с помощью функции **GPIO_setup**.

После этого программа будет запущена в цикле **while(1){...}** для передачи данных в порты от PC6 до PC9 и от PE2 до PE5 с помощью инструкции **GPIO_WriteBit**, которая передает либо 1 (светодиод включен), либо 0 (светодиод выключен), и задержкой между посылками в 500 мс, задаваемой функцией **delay**.

Далее представлено подробное описание наиболее важной функции в этом примере кода: **RCC_setup** — функции настройки источника сигнала тактовой частоты для STM32

Листинг L1-1: output_01.c — С-файл для микроконтроллера STM32 содержит пример работы порта вывода. (продолжение)

```
void RCC_setup(void)
{
  ErrorStatus HSEStartUpStatus;                       //Переменная статуса ошибки
  RCC_DeInit();                                       //Сброс RCC-системы (для отладки)
  RCC_HSEConfig(RCC_HSE_ON);                          //Разрешаем работу HSE-генератора
  HSEStartUpStatus = RCC_WaitForHSEStartUp();         //Ждем пока HSE-генератор не будет готов
  if(HSEStartUpStatus == SUCCESS)
  {
      //Установка частот периферийных устройств
    RCC_HCLKConfig(RCC_SYSCLK_Div1);                  //HCLK = SYSCLK
    RCC_PCLK2Config(RCC_HCLK_Div1);                   //PCLK2 = HCLK
    RCC_PCLK1Config(RCC_HCLK_Div2);                   //PCLK1 = HCLK/2
    FLASH_SetLatency(FLASH_Latency_2);                //Устанавливаем время ожидания в состояние 2
      //Разрешаем доступ к предварительному буферу
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
      //PLLCLK = 8 МГц * 9 = 72 МГц
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
      //Разрешаем работу с PLL
    RCC_PLLCmd(ENABLE);
      //Ожидаем готовность PLL
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);
      //Устанавливаем PLL как источник сигнала тактовой частоты системы
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
      //Ждем пока PLL не станет источником сигнала тактовой частоты системы
    while(RCC_GetSYSCLKSource() != 0x08);
  }
}
```

Цель этой функции — установить тактовую частоту системы SYSCLK равной 72 МГц при частоте внешнего высокочастотного генератора (HSE) в 8 МГц и разрешенной фазовой автоподстройке частоты (PLL).

GPIO_setup: устанавливает параметры портов вывода

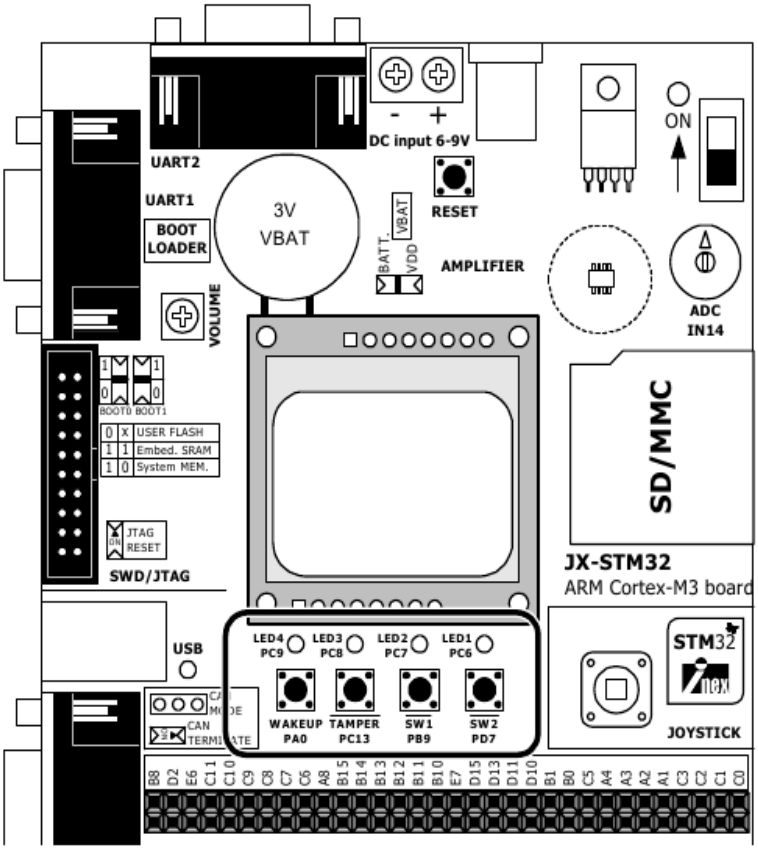
```
void GPIO_setup()
{
  GPIO_InitTypeDef GPIO_InitStructure;
    //Разрешаем подачу сигнала тактовой частоты на устройства шины APB2
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOE, ENABLE);
    //Настраиваем PC6, PC7, PC8, PC9 как выход (подключены к светодиодам)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
    //Настраиваем PE2, PE3, PE4, PE5 как выход (подключены к светодиодам)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOE, &GPIO_InitStructure);
}
```

delay: устанавливает время задержки в миллисекунду при тактовой частоте системы 72 МГц.

Листинг L1-1: output_01.c — C-файл для микроконтроллера STM32 содержит пример работы порта вывода. (окончание)

Эксперимент - 2 : Контроллер ввода/вывода

Этот эксперимент описывает программирование порта ввода/вывода контроллера STM32F103VBT6. Входные данные приходят с кнопок PD7, PB9, PC13 и PA0 для управления выходными устройствами - светодиодами на портах с PC6 до PC9 соответственно на плате JX-STM32.



Общий порядок действий

(2.1) Создать проект ***input_01***.

(2.2) Создать исходный файл на языке С ***input_01.c*** по Листингу L2-1.

(2.3) Скомпилировать файл проекта и загрузить его в микроконтроллер STM32F103VBT6 на плате JX-STM32. Более подробно это описано в главе 2 данного документа.

(2.4) Запустить эту программу. Нажимать кнопки на портах PD7, PB9, PC13 и PA0 и наблюдать как загораются светодиоды на портах с PC6 до PC9.

Кнопка PD7 управляет светодиодом на PC6

Кнопка PB9 управляет светодиодом на PC7

Кнопка PC13 управляет светодиодом на PC8

Кнопка PA0 управляет светодиодом на PC9

```
//**************************************************************************************************//
// Программа: Проверка порта ввода
// Описание: Проверка кнопки на PD7 с помощью мигания светодиода на PC6
// Имя файла: input_01.c
// Компилятор C: RkitARM 1.03.0003 для Ride7
// Аппаратно-зависимая библиотека: STMicroelectronics FWlLib V1.0
//**************************************************************************************************//
#include "stm32f10x_lib.h"                              //Подключаем заголовочный файл для STM32

//********************************** Функция задержки ************************************//
void delay(unsigned long ms)
// Задержка 1 мс за цикл при кварце 8.0 Мгц и PLL9x или SYSCLK = 72 Мгц
{
  volatile unsigned long i, j;
  for(i = 0; i < ms; i++)
    for(j = 0; j < 5525; j++);
}

//*****************************Функция установки RCC-регистров*******************************//
void RCC_setup(void)
{
  ErrorStatus HSEStartUpStatus;                         //Переменная статуса ошибки
  RCC_DeInit();                                         //Сброс RCC-системы (для отладки)
  RCC_HSEConfig(RCC_HSE_ON);                            //Разрешаем работу HSE-генератора
  HSEStartUpStatus = RCC_WaitForHSEStartUp();           //Ждем пока HSE-генератор не будет готов
  if(HSEStartUpStatus == SUCCESS)
  {
      //Установка частот периферийных устройств
    RCC_HCLKConfig(RCC_SYSCLK_Div1);                    //HCLK = SYSCLK
    RCC_PCLK2Config(RCC_HCLK_Div1);                     //PCLK2 = HCLK
    RCC_PCLK1Config(RCC_HCLK_Div2);                     //PCLK1 = HCLK/2
    FLASH_SetLatency(FLASH_Latency_2);                  //Устанавливаем время ожидания в состояние 2
      //Разрешаем доступ к предварительному буферу
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
      //PLLCLK = 8 МГц * 9 = 72 МГц
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
      //Разрешаем работу с PLL
    RCC_PLLCmd(ENABLE);
      //Ожидаем готовность PLL
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);
      //Устанавливаем PLL как источник сигнала тактовой частоты системы
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
      //Ждем пока PLL не станет источником сигнала тактовой частоты системы
    while(RCC_GetSYSCLKSource() != 0x08);
  }
}

//************************** Функция настройки основного УВВ (GPIO) ****************************//
void GPIO_setup()
{
  GPIO_InitTypeDef GPIO_InitStructure;
    //Разрешаем подачу сигнала тактовой частоты на GPIOA, GPIOB, GPIOC и GPIOD
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB |
   RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOD, ENABLE);
    //Настраиваем PC6, PC7, PC8, PC9 как выходы (подключены к светодиодам)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
    //Настраиваем PA0 как вход (подключен к кнопке)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
    //Настраиваем PB9 как вход (подключен к кнопке)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOB, &GPIO_InitStructure);
    //Настраиваем PC13 как вход (подключен к кнопке)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
```

Листинг L2-1: input_01.c — C-файл для микроконтроллера STM32 содержит пример работы портов ввода/вывода (начало).

```
    //Настраиваем PD7 как вход (подключен к кнопке)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOD, &GPIO_InitStructure);
}

//****************************** Основная (главная) функция *********************************//
int main()
{
  RCC_setup();                                    //Настройка сигналов тактовых частот
  GPIO_setup();                                   //Настройка портов основного УВВ (GPIO)
  while(1)                                        //Бесконечный цикл
  {
      //Кнопка PD7 нажата?
    if(GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_7)==0)
    {
        //Переключаем светодиод на PC6
      GPIO_WriteBit(GPIOC, GPIO_Pin_6, (BitAction)(1-GPIO_ReadOutputDataBit(GPIOC, GPIO_Pin_6)));
      delay(200);                                 //Задержка 0.2 с
    }
      //Кнопка PB9 нажата?
    if(GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_9)==0)
    {
        //Переключаем светодиод на PC7
      GPIO_WriteBit(GPIOC, GPIO_Pin_7, (BitAction)(1-GPIO_ReadOutputDataBit(GPIOC, GPIO_Pin_7)));
      delay(200);                                 //Задержка 0.2 с
    }
      //Кнопка PC13 нажата?
    if(GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_13)==0)
    {
        //Переключаем светодиод на PC8
      GPIO_WriteBit(GPIOC, GPIO_Pin_8, (BitAction)(1-GPIO_ReadOutputDataBit(GPIOC, GPIO_Pin_8)));
      delay(200);                                 //Задержка 0.2 с
    }
      //Кнопка PA0 нажата? (тока нет)
    if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0)==1)
    {
        //Переключаем светодиод на PC9
      GPIO_WriteBit(GPIOC, GPIO_Pin_9, (BitAction)(1-GPIO_ReadOutputDataBit(GPIOC, GPIO_Pin_9)));
      delay(200);                                 //Задержка 0.2 с
    }
  }
}
```

Описание кода

Этот код подключает заголовочный файл stm32f10x_lib.h в следующей строке:

```
#include <stm32f10x_lib.h>
```

Разработчик может иметь доступ ко всем ресурсам и аппаратно-зависимым библиотекам. (Сначала нужно скопировать файл **stm32f10x_conf.h** в каталог с вашим проектом).

Основная (главная) функция начинает свою работу с установки параметров сигналов тактовых частот в функции **RCC_setup**. Затем порты от PC6 до PC9 устанавливаются как выходы, а порты PD7, PB9, PC13 и PA0 как входы с помощью функции **GPIO_setup**.

После этого программа будет запущена в цикле **while(1){...}** для проверки состояния портов ввода с помощью инструкции **GPIO_ReadInputDataBit**. Если принимаемый бит равен 0, то кнопка на этом порту нажата. Процессор будет инвертировать данные и посылать в порты с PC6 по PC9 следующую информацию:

Вариант 1: Если нажата кнопка PD7, данные на PC6 преобразуются в 0, светодиод включен.

Вариант 2: Если нажата кнопка PB9, данные на PC7 преобразуются в 0, светодиод включен.

Вариант 3: Если нажата кнопка PC13, данные на PC8 преобразуются в 0, светодиод включен.

Вариант 4: Если нажата кнопка PA0, данные на PC9 преобразуются в 0, светодиод включен.

В каждое нажатие программа выполняет задержку 200 мс для исключения реакции на дребезг контактов при нажатии кнопки.

Листинг L2-1: input_01.c — C-файл для микроконтроллера STM32 содержит пример работы портов ввода/вывода (продолжение).

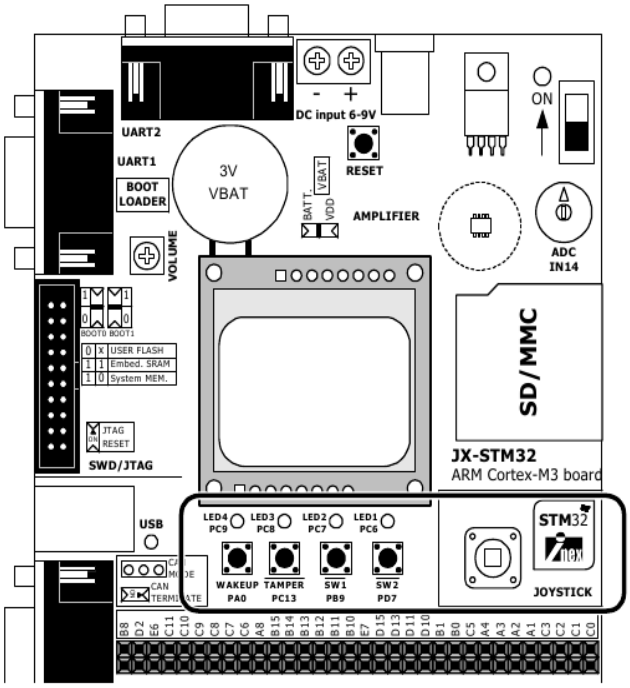
Инвертирование данных происходит во время выполнения инструкции **GPIO_WriteBit**. В этой инструкции происходит чтение текущих данных из соответствующего порта вывода с помощью инструкции **GPIO_ReadOutputDataBit**, которые вычитаются из 1. Затем происходит их преобразование к типу **BitAction** как SET (или 1) и RESET (или 0).

Листинг L2-1: input_01.c — C-файл для микроконтроллера STM32 содержит пример работы портов ввода/вывода (окончание).

STM32

Эксперимент - 3 : Контроллер ввода/вывода с джойстиком

Этот эксперимент описывает взаимодействие с кнопкой 5 степенями свободы, которая по-другому называется «Джойстик». Установим основное УВВ (GPIO) как порт ввода для определения состояния джойстика. С него мы будем получать входные данные и показывать их на портах вывода от PC6 до PC9.



Общий порядок действий

(3.1) Создать проект ***joystick_01***

(3.2) Создать исходный файл на языке С ***joystick_01.c*** по Листингу L3-1.

(3.3) Скомпилировать файл проекта и загрузить его в микроконтроллер STM32F103VBT6 на плате JX-STM32. Более подробно это описано в главе 2 данного документа.

(3.4) Запустить эту программу. Сдвинуть рычажок джойстика и нажать кнопку в центре. Наблюдать за диодами на портах с PC6 по PC9.

(3.4.1) Сдвинули рычажок в верхнее положение (JOY_UP).
Мигнул светодиод на PC6.

(3.4.2) Сдвинули рычажок в нижнее положение (JOY_DOWN).
Мигнул светодиод на PC7.

(3.4.3) Сдвинули рычажок в левое положение (JOY_LEFT).
Мигнул светодиод на PC8.

(3.4.4) Сдвинули рычажок в правое положение (JOY_RIGHT).
Мигнул светодиод на PC9.

(3.4.5) Нажали на рычажок (JOY_SELECT).
Мигнули все светодиоды на портах с PC6 по PC9.

```
//**************************************************************************************************//
// Программа: Проверка порта джойстика
// Описание: Проверка работы джойстика при условиях:
//    - движение вверх (JOY_UP) включает светодиод на PC6;
//    - движение вниз (JOY_DOWN) включает светодиод на PC7;
//    - движение влево (JOY_LEFT) включает светодиод на PC8;
//    - движение вправо (JOY_RIGHT) включает светодиод на PC9;
//    - нажатие на джойстик (JOY_SELECT) включает светодиоды на PC6, PC7, PC8 и PC9;
// Имя файла: joystick_01.c
// Компилятор C: RkitARM 1.03.0003 для Ride7
// Аппаратно-зависимая библиотека: STMicroelectronics FWlLib V1.0
//**************************************************************************************************//
#include "stm32f10x_lib.h"                          //Подключаем заголовочный файл для STM32

//************************************ Функция задержки ************************************//
void delay(unsigned long ms)
// Задержка 1 мс за цикл при кварце 8.0 Мгц и PLL9x или SYSCLK = 72 Мгц
{
  volatile unsigned long i, j;
  for(i = 0; i < ms; i++)
    for(j = 0; j < 5525; j++);
}

//*****************************Функция установки RCC-регистров*******************************//
void RCC_setup(void)
{
  ErrorStatus HSEStartUpStatus;                     //Переменная статуса ошибки
  RCC_DeInit();                                     //Сброс RCC-системы (для отладки)
  RCC_HSEConfig(RCC_HSE_ON);                        //Разрешаем работу HSE-генератора
  HSEStartUpStatus = RCC_WaitForHSEStartUp();       //Ждем пока HSE-генератор не будет готов
  if(HSEStartUpStatus == SUCCESS)
  {
     //Установка частот периферийных устройств
    RCC_HCLKConfig(RCC_SYSCLK_Div1);                //HCLK = SYSCLK
    RCC_PCLK2Config(RCC_HCLK_Div1);                 //PCLK2 = HCLK
    RCC_PCLK1Config(RCC_HCLK_Div2);                 //PCLK1 = HCLK/2
    FLASH_SetLatency(FLASH_Latency_2);              //Устанавливаем время ожидания в состояние 2
      //Разрешаем доступ к предварительному буферу
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
      //PLLCLK = 8 МГц * 9 = 72 МГц
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
      //Разрешаем работу с PLL
    RCC_PLLCmd(ENABLE);
      //Ожидаем готовность PLL
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);
      //Устанавливаем PLL как источник сигнала тактовой частоты системы
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
      //Ждем пока PLL не станет источником сигнала тактовой частоты системы
    while(RCC_GetSYSCLKSource() != 0x08);
  }
}
//************************** Функция настройки основного УВВ (GPIO) ***************************//
void GPIO_setup()
{
  GPIO_InitTypeDef GPIO_InitStructure;
    //Разрешаем подачу сигнала тактовой частоты на GPIOC, GPIOD и GPIOE
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOD | RCC_APB2Periph_GPIOE,
   ENABLE);
    //Настраиваем PC6, PC7, PC8, PC9 как выходы (подключены к светодиодам)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
    //Настраиваем PD8, PD12, PD14 как входы (подключены к джойстику)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_12 | GPIO_Pin_14;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOD, &GPIO_InitStructure);
    //Настраиваем PE0, PE1 как входы (подключены к джойстику)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOE, &GPIO_InitStructure);
}
```

Листинг L3-1: joystick_01.c — C-файл для микроконтроллера STM32 содержит пример работы портов ввода/вывода с джойстиком (начало).

```
int main()
{
  RCC_setup();                                    //Настройка сигналов тактовых частот
  GPIO_setup();                                   //Настройка портов основного УВВ (GPIO)
  while(1)                                        //Бесконечный цикл
  {
      // JOY_UP - джойстик двигают вверх (PD8)
    if(GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_8)==0)
    {
      GPIO_WriteBit(GPIOC, GPIO_Pin_6, 1);        //Светодиод на PC6 включен
      delay(200);                                 //Задержка 0.2 мс
      GPIO_WriteBit(GPIOC, GPIO_Pin_6, 0);        //Светодиод на PC6 выключен
    }
      // JOY_DOWN - джойстик двигают вниз (PD14)
    if(GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_14)==0)
    {
      GPIO_WriteBit(GPIOC, GPIO_Pin_7, 1);        //Светодиод на PC7 включен
      delay(200);                                 //Задержка 0.2 мс
      GPIO_WriteBit(GPIOC, GPIO_Pin_7, 0);        //Светодиод на PC7 выключен
    }
      // JOY_LEFT - джойстик двигают влево (PE1)
    if(GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_1)==0)
    {
      GPIO_WriteBit(GPIOC, GPIO_Pin_8, 1);        //Светодиод на PC8 включен
      delay(200);                                 //Задержка 0.2 мс
      GPIO_WriteBit(GPIOC, GPIO_Pin_8, 0);        //Светодиод на PC8 выключен
    }
      // JOY_LEFT - джойстик двигают вправо (PE0)
    if(GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_0)==0)
    {
      GPIO_WriteBit(GPIOC, GPIO_Pin_9, 1);        //Светодиод на PC9 включен
      delay(200);                                 //Задержка 0.2 мс
      GPIO_WriteBit(GPIOC, GPIO_Pin_9, 0);        //Светодиод на PC9 выключен
    }
      // JOY_SELECT – нажатие на джойстик (PD12)
    if(GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_12)==0)
    {
      GPIO_WriteBit(GPIOC, GPIO_Pin_6, 1);        //Светодиод на PC6 включен
      GPIO_WriteBit(GPIOC, GPIO_Pin_7, 1);        //Светодиод на PC7 включен
      GPIO_WriteBit(GPIOC, GPIO_Pin_8, 1);        //Светодиод на PC8 включен
      GPIO_WriteBit(GPIOC, GPIO_Pin_9, 1);        //Светодиод на PC9 включен
      delay(200);                                 //Задержка 0.2 мс
      GPIO_WriteBit(GPIOC, GPIO_Pin_6, 0);        //Светодиод на PC6 выключен
      GPIO_WriteBit(GPIOC, GPIO_Pin_7, 0);        //Светодиод на PC7 выключен
      GPIO_WriteBit(GPIOC, GPIO_Pin_8, 0);        //Светодиод на PC8 выключен
      GPIO_WriteBit(GPIOC, GPIO_Pin_9, 0);        //Светодиод на PC9 выключен
    }
  }
}
```

Описание кода

Этот код подключает заголовочный файл stm32f10x_lib.h в следующей строке:

```
#include <stm32f10x_lib.h>
```

Разработчик может иметь доступ ко всем ресурсам и аппаратно-зависимым библиотекам. (Сначала нужно скопировать файл **stm32f10x_conf.h** в каталог с вашим проектом).

Основная (главная) функция начинает свою работу с установки параметров сигналов тактовых частот в функции **RCC_setup**. Затем порты от PC6 до PC9 устанавливаются как выходы, а порты PD8, PD12, PD14, PE0 и PE1 как входы для определения состояния джойстика с помощью функции **GPIO_setup**.

После этого программа будет запущена в цикле **while(1){...}** для проверки состояния портов ввода с помощью инструкции **GPIO_ReadInputDataBit**. Процессор будет инвертировать данные и посылать в порты с PC6 по PC9 следующую информацию:

Листинг L3-1: joystick_01.c — C-файл для микроконтроллера STM32 содержит пример работы портов ввода/вывода с джойстиком (продолжение).

Вариант 1 — PD8 равен 0: Джойстик двигают вверх (JOY_UP). Диод на порту PC6 загорается на 200 мс.

Вариант 2 — PD14 равен 0: Джойстик двигают вниз (JOY_DOWN). Диод на порту PC7 загорается на 200 мс.

Вариант 3 — PE1 равен 0: Джойстик двигают влево (JOY_LEFT). Диод на порту PC8 загорается на 200 мс.

Вариант 4 — PE0 равен 0: Джойстик двигают вправо (JOY_RIGHT). Диод на порту PC9 загорается на 200 мс.

Вариант 5 — PD12 равен 0: Происходит нажатие на джойстик (JOY_SELECTION). Диоды на портах с PC6 по PC9 загораются на 200 мс.

Интересное назначение у кода функции GPIO_setup. Смотрите более подробно эту функцию ниже.

```
void GPIO_setup()
{
  GPIO_InitTypeDef GPIO_InitStructure;
    //Разрешаем подачу сигнала тактовой частоты на GPIOC, GPIOD и GPIOE
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOD | RCC_APB2Periph_GPIOE,
   ENABLE);
    //Настраиваем PC6, PC7, PC8, PC9 как выходы (подключены к светодиодам) режим 50 МГц
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
    //Настраиваем PD8, PD12, PD14 как входы (подключены к джойстику)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_12 | GPIO_Pin_14;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOD, &GPIO_InitStructure);
    //Настраиваем PE0, PE1 как входы (подключены к джойстику)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOE, &GPIO_InitStructure);
}
```

Листинг L3-1: joystick_01.c — C-файл для микроконтроллера STM32 содержит пример работы портов ввода/вывода с джойстиком (окончание).

5: Примеры внешних прерываний

Порядок разработки программ с внешними прерываниями

(1) Определить подходящие источники сигналов тактовой частоты для микропроцессорного ядра STM32 и периферии.

(2) Установить основное УВВ (GPIO) в режим порта ввода. Рекомендуется установить тип меняющегося ввода (Input Floating). Необходимо разрешить доступ сигнала тактовой частоты к GPIO и к AFIO на периферийной шине APB2.

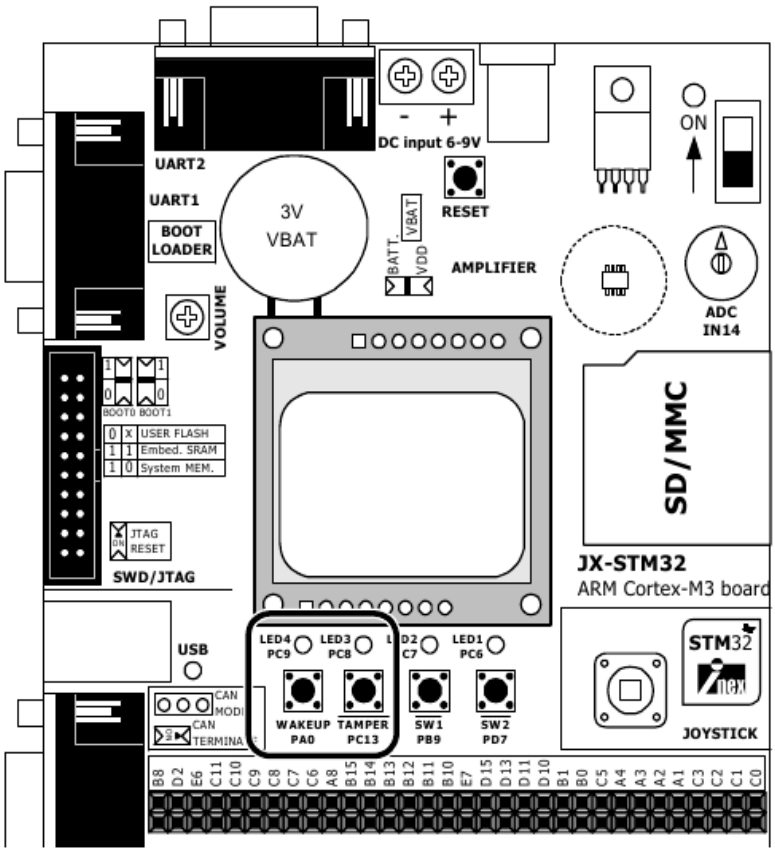
(3) Настроить параметры модуля внешних прерываний (EXTI). Выбрать границу определения сигнала прерывания.

(4) Разрешить внешние прерывания (EXTI) в контроллере вектора прерываний (NVIC).

(5) Объявить функцию обработки прерываний.

Эксперимент - 4 : Внешние прерывания

Этот эксперимент показывает работу внешних прерываний (EXTI). При написании программы внешние прерывания должны быть настроены для определения падения уровня сигнала прерывания на контактах PA0 и PC13. PA0 соответствует линии 0 внешних прерываний, а PC13 - линии 13. Оба контакта соединены с кнопками на плате. Светодиоды на PC8 и PC9 соответствуют устройству отображения в этом эксперименте.



```
//*****************************************************************************************//
// Программа    : Проверка внешних прерываний (EXTI) на портах PA0 и PC13
// Описание     : Включение/выключение светодиода на PC9 при нажатии кнопки на PA0
//              : Включение/выключение светодиода на PC6 при нажатии кнопки на PC13
//              : (Падение уровня при нажатии)
// Имя файла    : switch_int_01.c
// Компилятор C : RkitARM 1.03.0003 для Ride7
//*****************************************************************************************//
#include "stm32f10x_lib.h"                          //Подключаем заголовочный файл для STM32

//*****************************Функция установки RCC-регистров*****************************//
void RCC_setup(void)
{
  ErrorStatus HSEStartUpStatus;                     //Переменная статуса ошибки
  RCC_DeInit();                                     //Сброс RCC-системы (для отладки)
  RCC_HSEConfig(RCC_HSE_ON);                        //Разрешаем работу HSE-генератора
  HSEStartUpStatus = RCC_WaitForHSEStartUp();       //Ждем пока HSE-генератор не будет готов
  if(HSEStartUpStatus == SUCCESS)
  {
      //Установка частот периферийных устройств
    RCC_HCLKConfig(RCC_SYSCLK_Div1);                //HCLK = SYSCLK
    RCC_PCLK2Config(RCC_HCLK_Div1);                 //PCLK2 = HCLK
    RCC_PCLK1Config(RCC_HCLK_Div2);                 //PCLK1 = HCLK/2
    FLASH_SetLatency(FLASH_Latency_2);              //Устанавливаем время ожидания в состояние 2
      //Разрешаем доступ к предварительному буферу
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
      //PLLCLK = 8 МГц * 9 = 72 МГц
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
      //Разрешаем работу с PLL
    RCC_PLLCmd(ENABLE);
      //Ожидаем готовность PLL
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);
      //Устанавливаем PLL как источник сигнала тактовой частоты системы
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
      //Ждем пока PLL не станет источником сигнала тактовой частоты системы
    while(RCC_GetSYSCLKSource() != 0x08);
  }
}

//************************** Функция настройки основного УВВ (GPIO) ***************************//
void GPIO_setup()
{
  GPIO_InitTypeDef GPIO_InitStructure;
    //Разрешаем подачу сигнала тактовой частоты на GPIOB и GPIOC
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOC, ENABLE);
    //Настраиваем PC8 и PC9 как выходы (подключены к светодиодам)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
    //Настраиваем PC13 как вход (линия 13 внешних прерываний (EXTI))
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOD, &GPIO_InitStructure);
    //Настраиваем PE0 как вход (линия 0 внешних прерываний (EXTI))
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOE, &GPIO_InitStructure);
}

//***************** Функция настройки контроллера вектора прерываний (NVIC) ******************//
void NVIC_setup()
{
  NVIC_InitTypeDef NVIC_InitStructure;
    //Устанавливаем бит приоритета прерываний
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
    //Разрешаем прерывание 0 (EXTI0)
  NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQChannel;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
```

Листинг L4-1: switch_int_01.c — C-файл для микроконтроллера STM32 содержит пример работы внешних прерываний (начало).

```
    //Разрешаем прерывание 13 (EXTI13)
  NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQChannel;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}

//********************* Функция настройки внешних прерываний (EXTI) ***************************//
void EXTI_setup()
{
  EXTI_InitTypeDef EXTI_InitStructure;
    //Разрешение доступа к УВВ (AFIO) на APB2 через внешние прерывания
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
    //Подключаем линию 0 прерываний (EXTI Line0) к порту PA0
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource0);
    //Устанавливаем EXTI Line0 для генерации прерывания при падении уровня сигнала
  EXTI_InitStructure.EXTI_Line = EXTI_Line0;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);
    //Подключаем линию 13 прерываний (EXTI Line13) к порту PC13
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource13);
    //Устанавливаем EXTI Line13 для генерации прерывания при падении уровня сигнала
  EXTI_InitStructure.EXTI_Line = EXTI_Line13;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);
}

//***************** Функция реакции на прерывание по линии 0 (EXTI Line0) **********************//
void EXTI0_IRQHandler(void)
{
    //Проверяем наличие бита прерывания на EXTI line 0
  if(EXTI_GetITStatus(EXTI_Line0) != RESET)
  {
      //Переключаем светодиод на порту PC9
    GPIO_WriteBit(GPIOC, GPIO_Pin_9, (BitAction)((1-GPIO_ReadOutputDataBit(GPIOC, GPIO_Pin_9))));
    EXTI_ClearITPendingBit(EXTI_Line0);              //Очищаем бит прерывания на EXTI line 0
  }
}

//***************** Функция реакции на прерывание по линиям от 10 до 15 ************************//
void EXTI15_10_IRQHandler(void)
{
    //Проверяем наличие бита прерывания на EXTI line 13
  if(EXTI_GetITStatus(EXTI_Line13) != RESET)
  {
      //Переключаем светодиод на порту PC8
    GPIO_WriteBit(GPIOC, GPIO_Pin_8, (BitAction)((1-GPIO_ReadOutputDataBit(GPIOC, GPIO_Pin_8))));
    EXTI_ClearITPendingBit(EXTI_Line13);             //Очищаем бит прерывания на EXTI line 13
  }
}

int main()
{
  RCC_setup();                                       //Настройка сигналов тактовых частот
  GPIO_setup();                                      //Настройка портов основного УВВ (GPIO)
  EXTI_setup();                                      //Настройка внешних прерываний (EXTI)
  NVIC_setup();                                      //Настройка контроллера вектора прерываний
  while(1);                                          //Прерываемая программа
}
```

Описание кода

Этот код подключает заголовочный файл stm32f10x_lib.h в следующей строке:

```
#include <stm32f10x_lib.h>
```

Разработчик может иметь доступ ко всем ресурсам и аппаратно-зависимым библиотекам. (Сначала нужно скопировать файл **stm32f10x_conf.h** в каталог с вашим проектом).

Листинг L4-1: switch_int_01.c — С-файл для микроконтроллера STM32 содержит пример работы внешних прерываний (продолжение).

Этот код состоит из большого количества функций. Ниже мы опишем их назначение:

RCC_setup - настраивает тактовые частоты, необходимые для работы частей системы, и показывает их источники.

SYSCLK = 72 МГц из внешнего высокочастотного 8 МГц генератора (HSE) с использованием системы фазовой автоподстройки частоты для умножения исходной тактовой частоты в 9 раз.

HCLK = SYSCLK

PCLK2 = HCLK/1 = 72 МГц

PCLK1 = HCLK/2 = 36 МГц

LATENCY - 2 состояние ожидания (так как 48 МГц < SYSCLK <= 72 МГц)

GPIO_setup — устанавливает режим работы основного УВВ (GPIO).

PC8 и PC9 устанавливаются в режим двухтактного выхода (push-pull).

PA0 и PC13 устанавливаются в режим аналогового входа (floating) для определения сигнала внешнего прерывания.

EXTI_setup — устанавливает режим работы модуля внешних прерываний (EXTI).

PA0 и PC13 устанавливаются как линии 0 и 13 внешних прерываний соответственно. Прерывания происходят при падении уровней сигналов.

NVIC_setup — устанавливает режим обслуживания прерываний в модуле вектора обработки прерываний (NVIC).

Происходит выбор источника прерываний как внешние прерывания на линиях 0 и 13. По умолчанию, прерывание на линии 0 более приоритетное, чем прерывание на линии 13.

После этого программа будет запущена в цикле **while(1){...}** для определения падения уровней сигналов прерываний на PA0 и PC13. Если это происходит, выполняется обработка прерывания.

Вариант 1 — происходит прерывание на линии 0. Процессор переходит на выполнение **EXTI0_IRQHandler** - функции обработки прерывания по линии 0. Эта функция инвертирует состояние светодиода (включает/выключает) на порту PC6 и очищает флаг прерывания для последующей работы.

Вариант 2 — происходит прерывание на линии 13. Процессор переходит на выполнение **EXTI15_10_IRQHandler** - функции обработки прерывания по линиям от 10 до 15. Эта функция инвертирует состояние светодиода (включает/выключает) на порту PC9 и очищает флаг прерывания для последующей работы.

Листинг L4-1: switch_int_01.c — C-файл для микроконтроллера STM32 содержит пример работы внешних прерываний (окончание).

Общий порядок действий

(4.1) Создать проект ***switch_int_01***.

(4.2) Создать исходный файл на языке C ***switch_int.c*** по Листингу L4-1.

(4.3) Скомпилировать файл проекта и загрузить его в микроконтроллер STM32F103VBT6 на плате JX-STM32. Более подробно это описано в главе 2 данного документа.

(4.4) Запустить эту программу. Нажимать кнопки на PA0 и PC13 и наблюдать за работой.

Если нажата кнопка на PA0, происходит внешнее прерывание. Светодиод на PC9 включится и выключится при последующем нажатии кнопки на PA0.

Если нажата кнопка на PC13, происходит внешнее прерывание. Светодиод на PC8 включится и выключится при последующем нажатии кнопки на PC13.

6: Пример взаимодействия с графическим ЖК-дисплеем

6.1 Технические характеристики дисплея GLCD5110

- Модуль графического ЖК-дисплея № LPH7366. Разрешение 84x48 пикселей;
- Встроенный контроллер PCD8544 с интерфейсом по шине последовательного периферийного интерфейса;
- Встроенная светодиодная подсветка, управляемая программно;
- Напряжение питания от +3В до 3.3В. Максимальный ток - 10 мА (с учетом светодиодной подсветки);
- Может показывать стандартные символы размером 5x7 пикселов по 14 символов в строке, всего 6 строк или монохромное графическое изображение;
- Размер 4.5x4.5 см;
- Удобен (прост) при подключении;
- Можно подключить ко многим популярным микроконтроллерам
  - *если уровень логической единицы равен +3 В, пользователь может подключить ЖК-дисплей напрямую к порту микроконтроллера;*
  - *если уровень логической единицы равен +5 В, пользователь должен подключать дисплей через схему сопряжения, чтобы опустить уровень логической единицы с 5 В до 3 или 3.3 В или не выше уровня напряжения питания GLCD5110;*

На Рисунке 6-1 показаны размеры и обозначения контактов GLCD5110;

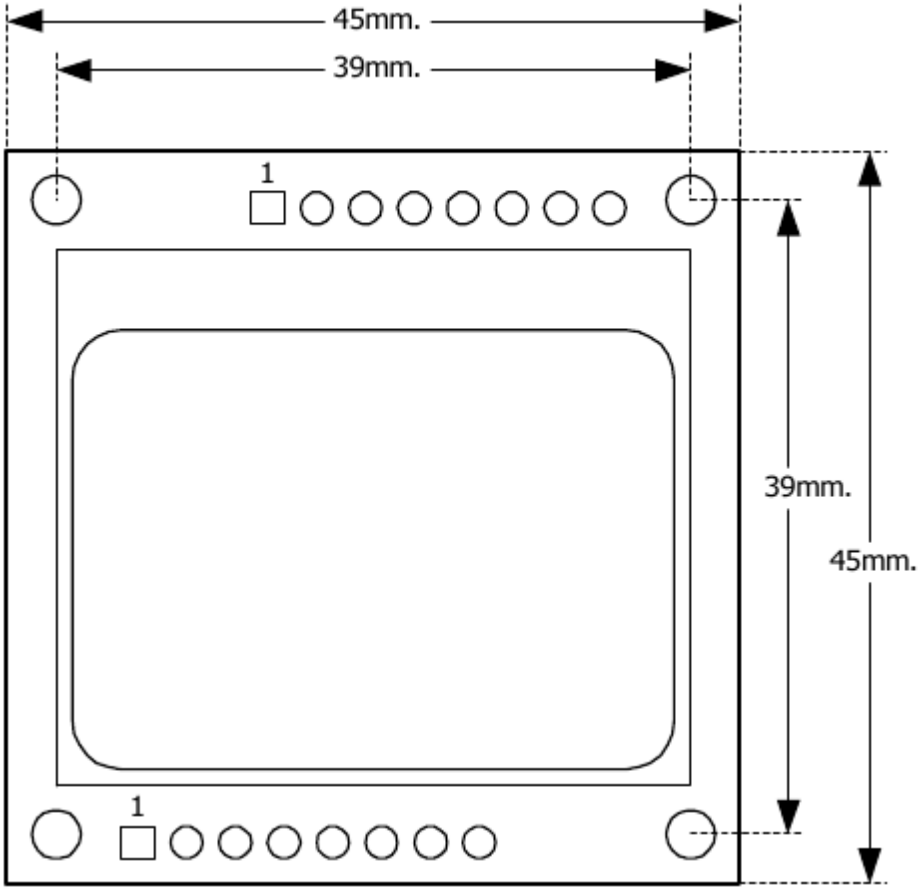
На Рисунке 6-2 показана схема соединения с микроконтроллером;

6.2 Взаимодействие с JX-STM32

Соединения дисплея GLCD5110 с платой JX-STM32 показаны ниже:

GLCD5110	STM32F103VBT6
SDIN	PE11
SCLK	PE12
SCE	PE8
RESET	PE9
D/C	PE10
LED	PE13

На Рисунке 6-3 показана схема соединения с платой JX-STM32;



№	Наименование	Назначение
1	Vcc	Напряжение питания от +2.7 до +3.3 В
2	GND	Корпус
3	SCE	Контакт доступа - активный уровень низкий
4	RESET	Контакт сброса - активный уровень низкий
5	D/C	Контакт выбора Данные/Команда *0 - запись данных* *1 - запись команды*
6	SDIN	Вход последовательных данных
7	SCLK	Вход сигнала тактовой частоты
8	LED	Контакт управления светодиодной подсветкой - активный уровень высокий (включая резистор ограничения по току 330 Ом)

Рисунок 6-1. Размеры и обозначения контактов дисплея GLCD5110

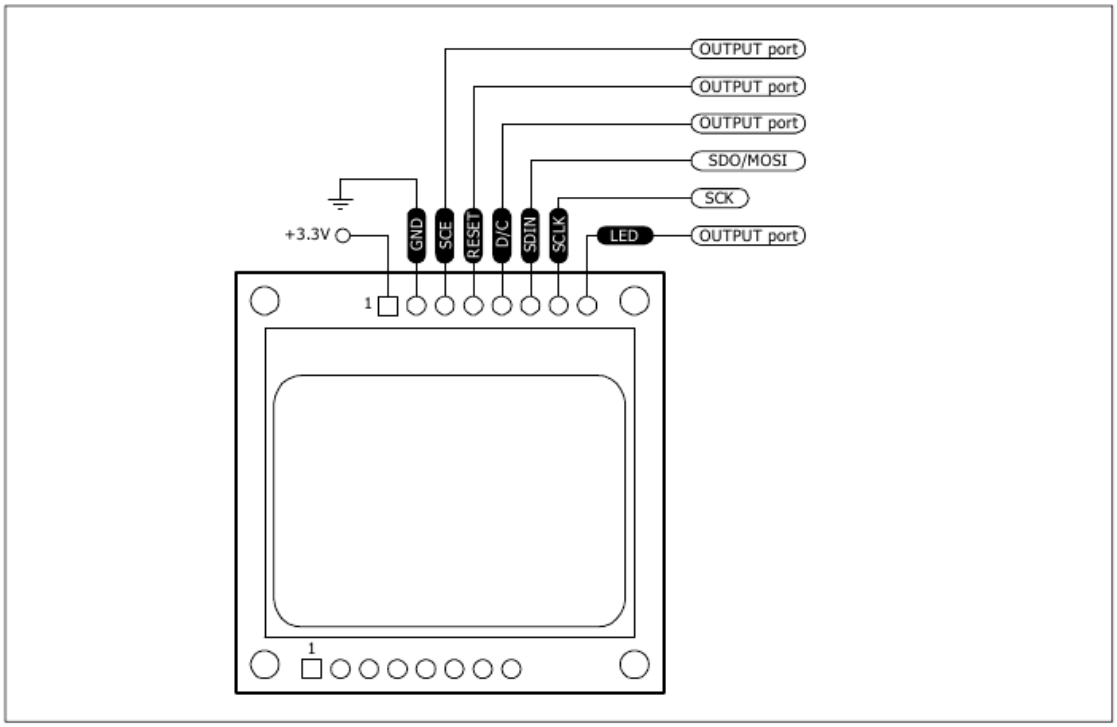


Рисунок 6-2. Схема подключения дисплея GLCD5110 к микроконтроллеру

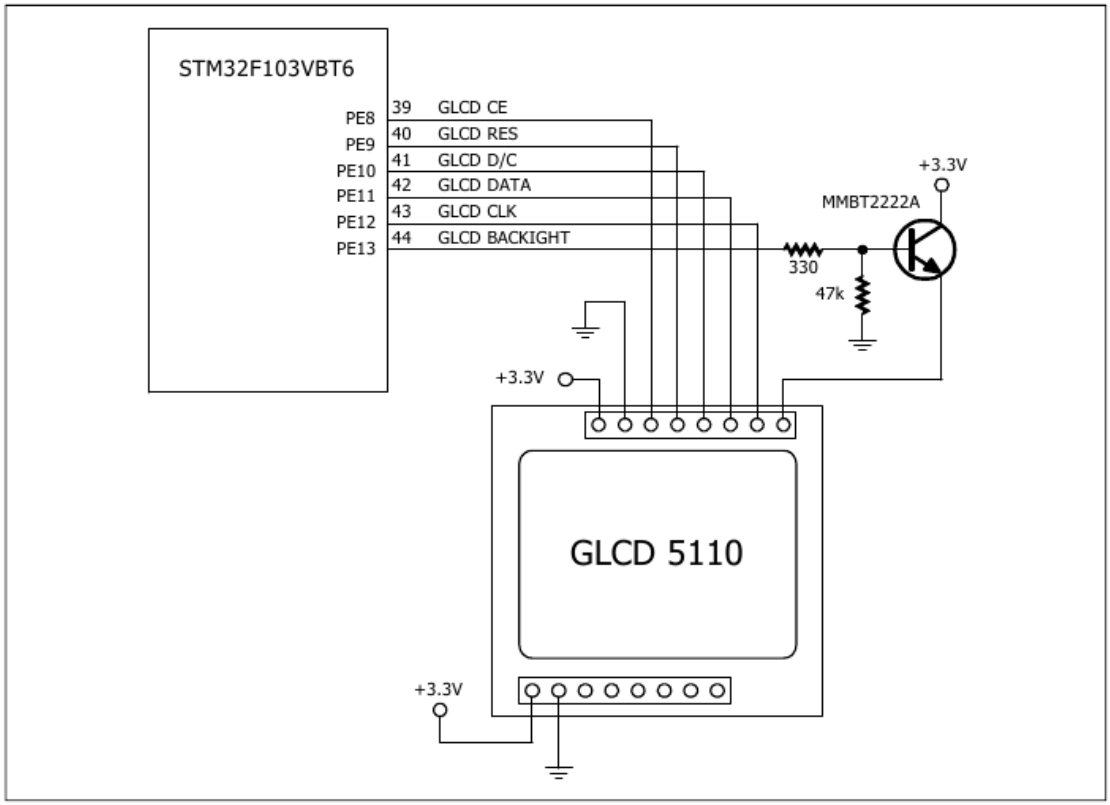


Рисунок 6-3. Схема подключения дисплея GLCD5110 к микроконтроллеру STM32F103VBT6 на плате JX-STM32

6.3 Программирование STM32 для управления дисплеем GLCD5110

Управление дисплеем GLCD5110 с помощью STM32 основано на библиотеке, которая называется ***glcd5110.h***. Эта библиотека создана с помощью компании Innovative Experiment (INEX). Она находится на CD-ROM, прилагаемом к плате JX-STM32.

Вам необходимо подключить эту библиотеку как заголовочный файл в исходном файле на языке С с помощью инструкции ***#include "glcd5110.h"***. Определение пути нахождения этой библиотеки возможно двумя методами:

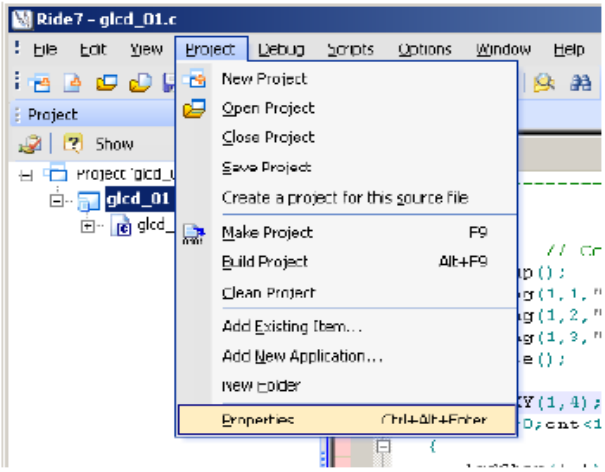
1. Определение пути по методу 1

Скопируйте файл glcd5110.h в каталог с разрабатываемым проектом. Компилятор найдет эту библиотеку автоматически.

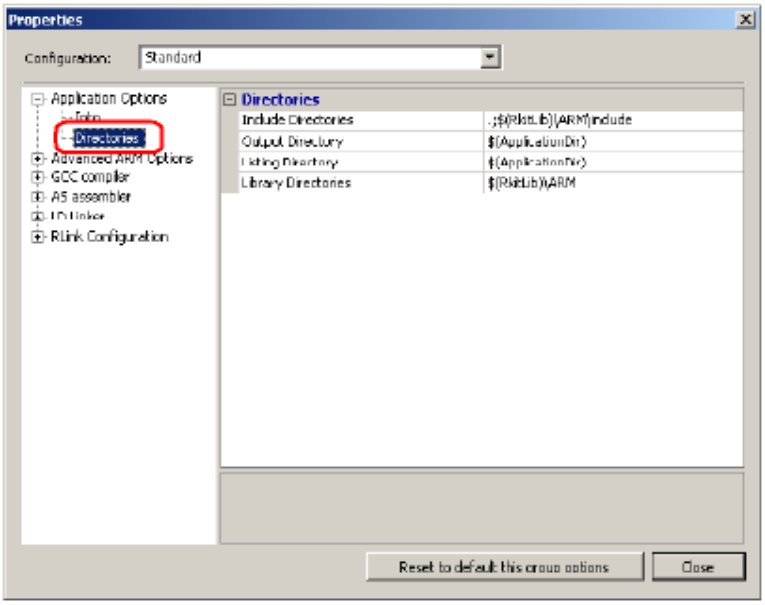
2. Определение пути по методу 2

Для определения пути по этому методу необходима Ride7. Выполните следующие шаги:

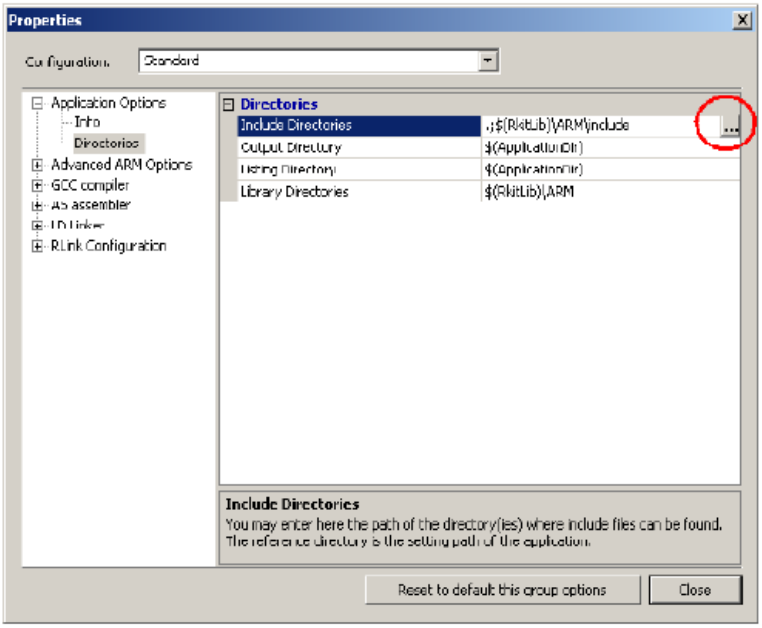
(2.1) Выберите пункт меню **Project -> Properties**



(2.2) Откроется окно **Properties**. В списке слева в пункте **Application Options** выберите подпункт **Directories**

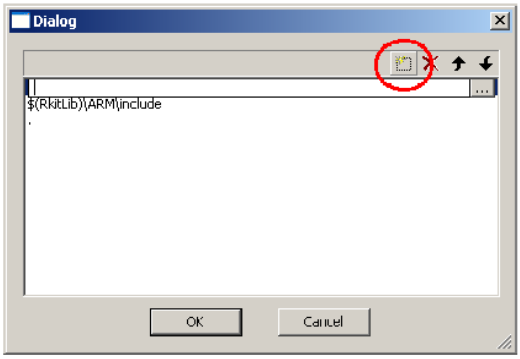


(2.3) В правой стороне окна выберите строку **Include Directories**, после чего нажмите на кнопку **[...]** в правом конце строки для задания пути по умолчанию.



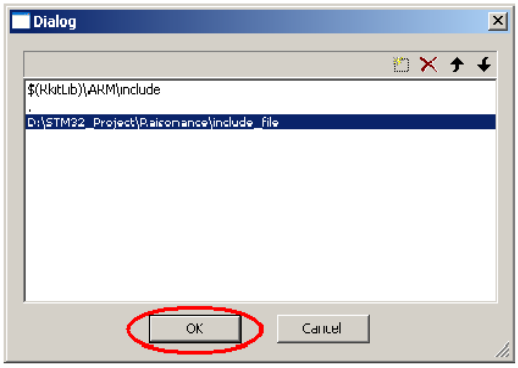
(2.4) Откроется окно **Dialog**. Нажмите на кнопку **New** (первая слева в верхнем ряду кнопок) для определения нового пути. В нашем случае это:

D:\STM32_Project\Raisonance\include_file



Таким образом можно определить местоположение путей библиотек и заголовочных файлов для каждого разработчика в отдельности.

(2.5) После окончательной установки пути к файлу glcd5110.h нажмите кнопку OK для подтверждения.



Примечание: Библиотека ***glcd5110.h*** автоматически устанавливает режимы работы портов PE8 и PE13 как выходы.

Эксперимент - 5 : Отображение символов на дисплее GLCD5110

Этот эксперимент показывает простое управление GLCD5110 с помощью STM32F103VBT6. Код программы будет заставлять GLCD5110 показывать простой текст на своём экране.

Общий порядок действий

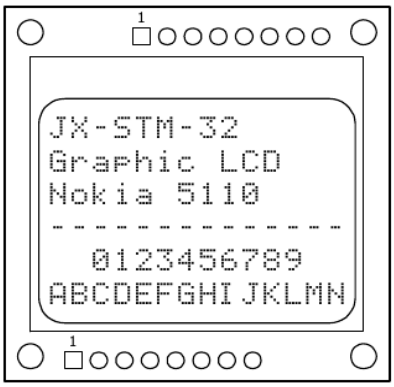
(5.1) Создать проект ***glcd_01***.

(5.2) Создать исходный файл на языке С ***glcd_01.c*** по Листингу L5-1.

(5.3) Скомпилировать файл проекта и загрузить его в микроконтроллер STM32F103VBT6 на плате JX-STM32. Более подробно это описано в главе 2 данного документа.

(5.4) Запустить эту программу. Наблюдать работу платы JX-STM32 на экране дисплея GLCD5110.

Дисплей GLCD5110 будет показывать то, что изображено на картинке ниже:



И светодиодная подсветка GLCD5110 будет мигать 1 раз в 0.5 секунды

```
//*****************************************************************************************//
// Программа                       : Проверка дисплея GLCD5110
// Описание                        : Простые сообщения для показа их на дисплее GLCD5110
// Имя файла                       : glcd_01.c
// Компилятор C                    : RkitARM 1.03.0003 для Ride7
// Аппаратно-зависимая библиотека : STMicroelectronics FWlLib V1.0
//*****************************************************************************************//
#include "stm32f10x_lib.h"                          //Подключаем заголовочный файл для STM32
#include "glcd5110.h"                               //Подключаем файл для работы с GLCD5110

//*******************************Функция установки RCC-регистров*********************************//
void RCC_setup(void)
{
  ErrorStatus HSEStartUpStatus;                     //Переменная статуса ошибки
  RCC_DeInit();                                     //Сброс RCC-системы (для отладки)
  RCC_HSEConfig(RCC_HSE_ON);                        //Разрешаем работу HSE-генератора
  HSEStartUpStatus = RCC_WaitForHSEStartUp();       //Ждем пока HSE-генератор не будет готов
  if(HSEStartUpStatus == SUCCESS)
  {
     //Установка частот периферийных устройств
    RCC_HCLKConfig(RCC_SYSCLK_Div1);                //HCLK = SYSCLK
    RCC_PCLK2Config(RCC_HCLK_Div1);                 //PCLK2 = HCLK
    RCC_PCLK1Config(RCC_HCLK_Div2);                 //PCLK1 = HCLK/2
    FLASH_SetLatency(FLASH_Latency_2);              //Устанавливаем время ожидания в состояние 2
      //Разрешаем доступ к предварительному буферу
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
      //PLLCLK = 8 МГц * 9 = 72 МГц
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
      //Разрешаем работу с PLL
    RCC_PLLCmd(ENABLE);
      //Ожидаем готовность PLL
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);
      //Устанавливаем PLL как источник сигнала тактовой частоты системы
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
      //Ждем пока PLL не станет источником сигнала тактовой частоты системы
    while(RCC_GetSYSCLKSource() != 0x08);
  }
}

//******************************* Основная (главная) функция **********************************//
int main()
{
  int cnt;                                          //счетчик
  RCC_setup();                                      //Настройка сигналов тактовых частот
  lcdString(1, 1, "JX-STM32");                      //Сообщение в строку 1
  lcdString(1, 2, "Graphic LCD");                   //Сообщение в строку 2
  lcdString(1, 3, "Nokia 5110");                    //Сообщение в строку 3
  lcdUpdate();                                      //Показываем сообщения на экране дисплея
  lcdGotoXY(1, 4);                                  //Переходим на 1 столбец, 4 строку
  for(cnt=0; cnt<14; cnt++)                         //Цикл для рисования "-" в строку 4
  {
    lcdChar('-');
  }
  lcdUpdate();                                      //Показываем сообщение на экране дисплея
  lcdGotoXY(3, 5);                                  //Переходим на 3 столбец, 5 строку
  for(cnt='0'; cnt<='9'; cnt++)                     //Цикл для рисования '0'—'9' в строке 5
  {
    lcdChar(cnt);
  }
  lcdUpdate();                                      //Показываем сообщения на экране дисплея
  while(1)                                          //Бесконечный цикл мигания подсветки
  {
    lcdBackLight(1);                                //Подсветка включена
    delay_ms(500);                                  //Задержка 500 мс
    lcdBackLight(0);                                //Подсветка отключена
    delay_ms(500);                                  //Задержка 500 мс
  }
}
```

Листинг L5-1: glcd_01.c — C-файл для микроконтроллера STM32 содержит пример отображения символов на экране дисплея GLCD5110 (начало).

Описание кода

Этот код включает в себя 2 заголовочных файла:

```
#include <stm32f10x_lib.h>
#include "glcd5110.h"
```

Разработчик может иметь доступ ко всем ресурсам и аппаратно-зависимым библиотекам. (Сначала нужно скопировать файлы ***stm32f10x_conf.h*** и ***glcd5110.h*** в каталог с вашим проектом).

Этот код состоит из большого количества функций. Ниже мы опишем их назначение:

RCC_setup - настраивает тактовые частоты, необходимые для работы частей системы, и показывает их источники.

SYSCLK = 72 МГц из внешнего высокочастотного 8 МГц генератора (HSE) с использованием системы фазовой автоподстройки частоты для умножения исходной тактовой частоты в 9 раз.

HCLK = SYSCLK

PCLK2 = HCLK/1 = 72 МГц

PCLK1 = HCLK/2 = 36 МГц

LATENCY - 2 состояние ожидания (так как 48 МГц < SYSCLK <= 72 МГц)

Функции объявленные в файле glcd5110.h:

lcdString — определяет сообщение для дисплея;

lcdUpdate — отправляет сообщения на дисплей;

lcdGotoXY — определяет позицию на экране дисплея;

lcdBackLight — включает/выключает подсветку дисплея;

Листинг L5-1: glcd_01.c — C-файл для микроконтроллера STM32 содержит пример отображения символов на экране дисплея GLCD5110 (окончание).

Эксперимент - 6 : Отображение простых графических объектов на GLCD5110

Этот эксперимент показывает простое управление GLCD5110 с помощью STM32F103VBT6. Код программы будет заставлять GLCD5110 показывать простые графические объекты на своём экране.

Общий порядок действий

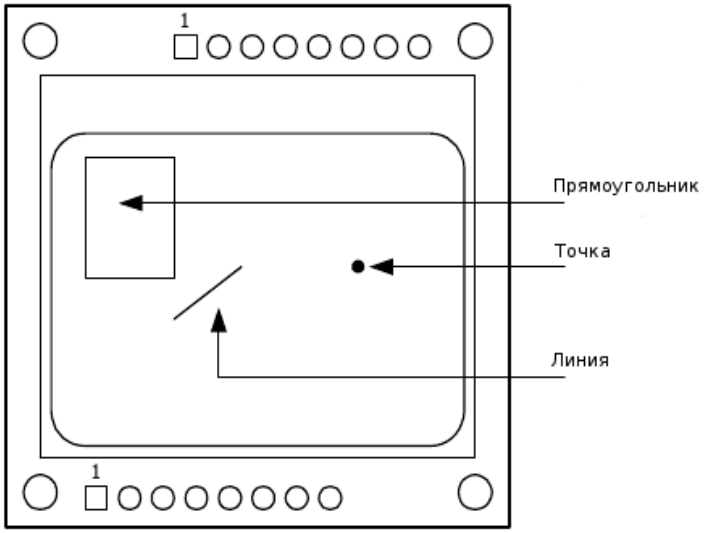
(6.1) Создать проект ***glcd_02***.

(6.2) Создать исходный файл на языке C ***glcd_02.c*** по Листингу L6-1.

(6.3) Скомпилировать файл проекта и загрузить его в микроконтроллер STM32F103VBT6 на плате JX-STM32. Более подробно это описано в главе 2 данного документа.

(6.4) Запустить эту программу. Наблюдать работу платы JX-STM32 на экране дисплея GLCD5110.

Дисплей GLCD5110 будет показывать то, что изображено на картинке ниже:



```
//*****************************************************************************************//
// Программа                       : Проверка дисплея GLCD5110
// Описание                        : Простые графические объекты для показа их на дисплее GLCD5110
// Имя файла                       : glcd_02.c
// Компилятор C                    : RkitARM 1.03.0003 для Ride7
// Аппаратно-зависимая библиотека : STMicroelectronics FWlLib V1.0
//*****************************************************************************************//
#include "stm32f10x_lib.h"                              //Подключаем заголовочный файл для STM32
#include "glcd5110.h"                                   //Подключаем файл для работы с GLCD5110

//*******************************Функция установки RCC-регистров*******************************//
void RCC_setup(void)
{
  ErrorStatus HSEStartUpStatus;                         //Переменная статуса ошибки
  RCC_DeInit();                                         //Сброс RCC-системы (для отладки)
  RCC_HSEConfig(RCC_HSE_ON);                            //Разрешаем работу HSE-генератора
  HSEStartUpStatus = RCC_WaitForHSEStartUp();           //Ждем пока HSE-генератор не будет готов
  if(HSEStartUpStatus == SUCCESS)
  {
     //Установка частот периферийных устройств
    RCC_HCLKConfig(RCC_SYSCLK_Div1);                    //HCLK = SYSCLK
    RCC_PCLK2Config(RCC_HCLK_Div1);                     //PCLK2 = HCLK
    RCC_PCLK1Config(RCC_HCLK_Div2);                     //PCLK1 = HCLK/2
    FLASH_SetLatency(FLASH_Latency_2);                  //Устанавливаем время ожидания в состояние 2
      //Разрешаем доступ к предварительному буферу
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
      //PLLCLK = 8 МГц * 9 = 72 МГц
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
      //Разрешаем работу с PLL
    RCC_PLLCmd(ENABLE);
      //Ожидаем готовность PLL
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);
      //Устанавливаем PLL как источник сигнала тактовой частоты системы
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
      //Ждем пока PLL не станет источником сигнала тактовой частоты системы
    while(RCC_GetSYSCLKSource() != 0x08);
  }
}

//******************************* Основная (главная) функция *********************************//
int main()
{
  RCC_setup();                                          //Настройка сигналов тактовых частот
  lcdPixel(60, 20, 1);                                  //Рисуем точку на (60, 20)
  lcdLine(40, 30, 25, 40, 1);                           //Рисуем линию от (40, 30) до (25, 40)
  lcdRect(5, 5, 25, 25, 0, 1);                          //Рисуем прямоугольник от (5, 5) до (25, 25)
  lcdUpdate();                                          //Показываем сообщение на экране дисплея
  while(1);                                             //Бесконечный цикл
}
```

Описание кода

Этот код включает в себя 2 заголовочных файла:

```
#include <stm32f10x_lib.h>
#include "glcd5110.h"
```

Разработчик может иметь доступ ко всем ресурсам и аппаратно-зависимым библиотекам. (Сначала нужно скопировать файлы ***stm32f10x_conf.h*** и ***glcd5110.h*** в каталог с вашим проектом).

Новые функции из glcd5110.h, используемые в этом коде:

lcdPixel — определяет позицию точки;

lcdLine — определяет позицию для создания прямой линии;

lcdRect — определяет позицию для создания прямоугольника;

Листинг L6-1: glcd_02.c — C-файл для микроконтроллера STM32 содержит пример отображения графических объектов на экране дисплея GLCD5110.

Эксперимент - 7 : Показ индикатора выполнения (Progress Bar)

Этот эксперимент показывает простое управление GLCD5110 с помощью STM32F103VBT6. Код программы будет заставлять GLCD5110 показывать простой индикатор выполнения (Progress Bar) на своём экране.

Общий порядок действий

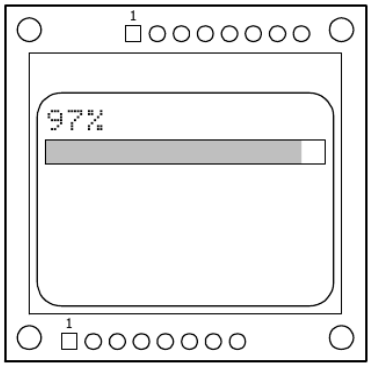
(7.1) Создать проект ***glcd_03***.

(7.2) Создать исходный файл на языке C ***glcd_03.c*** по Листингу L7-1.

(7.3) Скомпилировать файл проекта и загрузить его в микроконтроллер STM32F103VBT6 на плате JX-STM32. Более подробно это описано в главе 2 данного документа.

(7.4) Запустить эту программу. Наблюдать работу платы JX-STM32 на экране дисплея GLCD5110.

Дисплей GLCD5110 будет показывать то, что изображено на картинке ниже:



На экране дисплея в первой строке будет показываться значение отношения закрашенной части ко всей площади индикатора. Сам индикатор будет нарисован на следующей строке. Координаты его левой стороны (0, 10). Его размеры: 83 пикселя по ширине и 5 пикселей по высоте.

Значение закрашенной части индикатора выполнения увеличивается каждые 0.2 секунды и после достижения 100% сбрасывается на 0, после чего процесс повторяется.

```
//*****************************************************************************************//
// Программа                      : Проверка дисплея GLCD5110
// Описание                       : Показ индикатора выполнения на дисплее GLCD5110
// Имя файла                      : glcd_03.c
// Компилятор C                   : RkitARM 1.03.0003 для Ride7
// Аппаратно-зависимая библиотека : STMicroelectronics FWlLib V1.0
//*****************************************************************************************//
#include "stm32f10x_lib.h"                          //Подключаем заголовочный файл для STM32
#include "glcd5110.h"                               //Подключаем файл для работы с GLCD5110
#include "stdio.h"                                  //Подключаем файл с функцией sprintf

//*****************************Функция установки RCC-регистров*******************************//
void RCC_setup(void)
{
  ErrorStatus HSEStartUpStatus;                     //Переменная статуса ошибки
  RCC_DeInit();                                     //Сброс RCC-системы (для отладки)
  RCC_HSEConfig(RCC_HSE_ON);                        //Разрешаем работу HSE-генератора
  HSEStartUpStatus = RCC_WaitForHSEStartUp();       //Ждем пока HSE-генератор не будет готов
  if(HSEStartUpStatus == SUCCESS)
  {
      //Установка частот периферийных устройств
    RCC_HCLKConfig(RCC_SYSCLK_Div1);                //HCLK = SYSCLK
    RCC_PCLK2Config(RCC_HCLK_Div1);                 //PCLK2 = HCLK
    RCC_PCLK1Config(RCC_HCLK_Div2);                 //PCLK1 = HCLK/2
    FLASH_SetLatency(FLASH_Latency_2);              //Устанавливаем время ожидания в состояние 2
      //Разрешаем доступ к предварительному буферу
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
      //PLLCLK = 8 МГц * 9 = 72 МГц
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
      //Разрешаем работу с PLL
    RCC_PLLCmd(ENABLE);
      //Ожидаем готовность PLL
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);
      //Устанавливаем PLL как источник сигнала тактовой частоты системы
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
      //Ждем пока PLL не станет источником сигнала тактовой частоты системы
    while(RCC_GetSYSCLKSource() != 0x08);
  }
}

//****************************** Основная (главная) функция *********************************//
int main()
{
  char str[15];                                     //Буфер для хранения строки в процентах
  int percentage=0;                                 //Сохранение процентов индикатора выполнения
  RCC_setup();                                      //Настройка сигналов тактовых частот
  while(1)                                          //Бесконечный цикл
  {
    sprintf(str, "%d%%      ", percentage);         //Преобразуем число в строку
    lcdString(1, 1, str);                           //В строке 1 — количество процентов
      //Рисуем индикатор выполнения от (0, 10) ширина 83 пикселя, высота 5 пикселей
    lcdProgBar(0, 10, 83, 5, percentage);
    lcdUpdate();                                    //Показываем сообщение на экране дисплея
    delay_ms(200);                                  //Задержка перед следующим циклом
    percentage++;                                   //Увеличиваем количество процентов
    if(percentage>100)                              //К-во процентов больше 100?
    {
      percentage=0;                                 //Устанавливаем проценты в 0
    }
  }
}
```

Листинг L7-1: glcd_03.c — C-файл для микроконтроллера STM32 содержит пример показа индикатора выполнения на экране дисплея GLCD5110.

7: Пример использования аналого-цифрового преобразователя

В состав STM32F103VBT6 входит 2 модуля 12-битного аналого-цифрового преобразования (АЦП). Они являются аналого-цифровыми преобразователями последовательного приближения. Они имеют до 18 мультиплексных каналов, позволяющих измерять сигналы от 16 внешних и двух внутренних источников. Аналого-цифровое преобразование может производиться в однократном, непрерывном режимах, режиме сканирования или прерывистом режиме на различных каналах. Результат преобразования сохраняется в 16-битных регистрах данных, выравнивающих его по левым или правым разрядам.

Наличие аналогового сторожевого таймера позволяет программно определять когда входное напряжение выходит за указанные пользователем нижний и верхний пороги.

7.1 Основные характеристики

- 12-битное преобразование
- Генерация прерывания в конце преобразования и событие аналогового сторожевого таймера
- Однократный и непрерывный режимы преобразования;
- Режим сканирования каналов для автоматического преобразования на каналах от 0 до n;
- Автоматическая калибровка;
- Сравнение данных с помощью встроенного устройства выравнивания данных;
- Программируемая временная выборка по каналам;
- Внешняя схема выбора (переключатель) между стандартным и инжектирующим преобразованием;
- Прерывистый режим преобразования;
- Двойной режим работы (для устройств, использующих 2 АЦП);
- Время преобразования: 1 мкс на 56 МГц (1.17 мкс на 72 МГц);
- Требования к напряжению питания: от 2.4 до 3.6 В;
- Напряжение на входе: $V_{REF-} \le V_{IN} \le V_{REF+}$;
- Прямой доступ к памяти при стандартном преобразовании;

7.2 Назначение контактов модулей АЦП

В STM32F103VBT6 есть 21 контакт, связанный с модулем АЦП, в том числе:

V_{REF+} - Вход внешнего опорного напряжения. Диапазон: 2.4 В ≤ $V_{REF+} \le V_{DDA}$.

V_{DDA} - Напряжение питания модуля АЦП. Диапазон: 2.4 В ≤ $V_{DDA} \le 3.6$ В.

V_{REF-} - Вход внешнего опорного отрицательного напряжения. Равно V_{SSA}.

V_{SSA} - Корпус модуля АЦП.

ADCx_IN[15:0] - Аналоговые входы. Всего 16 контактов.

7.3 Краткое описание аппаратно-зависимой библиотеки АЦП-модуля

Наименование функции	Описание
ADC_DeInit	Сбрасывает периферийные регистры АЦП к их значениям по умолчанию
ADC_Init	Инициализирует (производит начальные настройки) АЦП
ADC_Cmd	Включает или отключает АЦП-модуль
ADC_DMACmd	Включает или отключает разрешение прямого доступа к памяти (ПДП) модуля АЦП
ADC_ITConfig	Включает или отключает разрешение прерываний от модуля АЦП
ADC_ResetCalibration	Сбрасывает регистры калибровки выбранного канала АЦП
ADC_GetResetCalibrationStatus	Читает состояние сброшенных регистров калибровки выбранного канала АЦП
ADC_StartCalibration	Запускает процесс калибровки выбранного канала АЦП
ADC_GetCalibrationStatus	Читает состояние калибровки выбранного канала АЦП
ADC_SoftwareStartConvCmd	Включает или отключает программный запуск аналого-цифрового преобразования выбранного канала АЦП
ADC_RegularChannelConfig	Настраивает для выбранного канала АЦП стандартного преобразования соответствующий ему приоритет и время преобразования
ADC_GetConversionValue	Возвращает последний результат аналого-цифрового преобразования для канала стандартного преобразования
ADC_AnalogWatchdogCmd	Включает или выключает аналоговый сторожевой таймер в однократном/непрерывном режимах на каналах стандартного или инжектирующего преобразования
ADC_AnalogWatchdogThresholdsConfig	Устанавливает верхний и нижний пороги для аналогового сторожевого таймера
ADC_AnalogWatchdogSingleChannelConfig	Устанавливает канал однократного преобразования для аналогового сторожевого таймера
ADC_GetFlagStatus	Проверяет, установлены ли флаги АЦП или нет
ADC_ClearFlag	Сбрасывает флаги АЦП
ADC_GetITStatus	Проверяет, какое из прерываний АЦП произошло, а какое нет
ADC_ClearITPendingBit	Очищает биты произошедших прерываний АЦП

Эксперимент - 8 : Чтение аналогового сигнала

В этом эксперименте показано чтение аналогового сигнала на порту PC4 или на аналоговом входе канала 14. Сигнал с потенциометра 10 кОм является источником аналогового напряжения, которое подается на вход АЦП. Преобразованные цифровые данные передаются для показа на дисплее GLCD5110 на плате JX-STM32.

Принципиальная схема для проведения эксперимента с АЦП изображена на Рисунке L8-1.

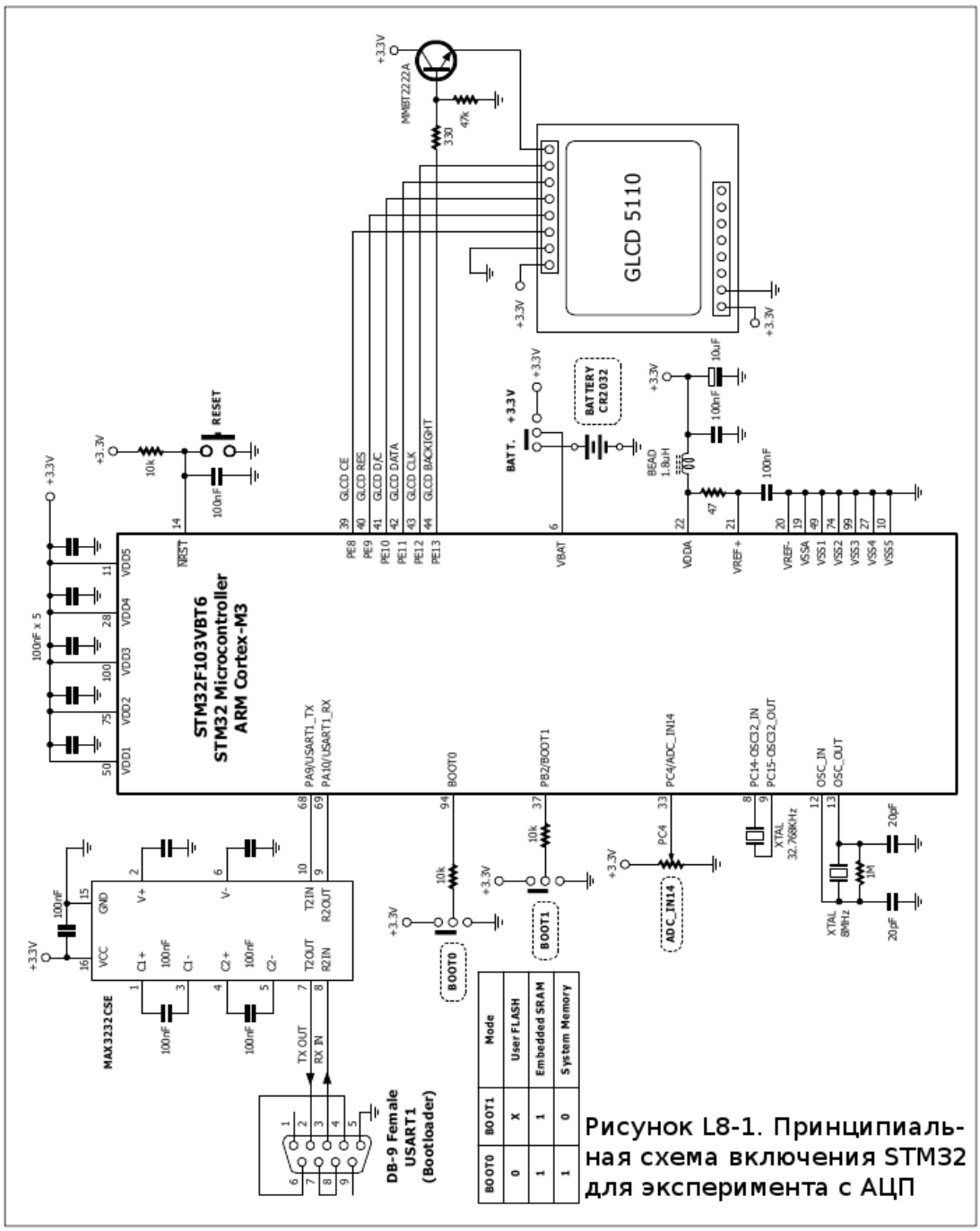


Рисунок L8-1. Принципиальная схема включения STM32 для эксперимента с АЦП

Общий порядок действий

(8.1) Создать проект ***adc_01***.

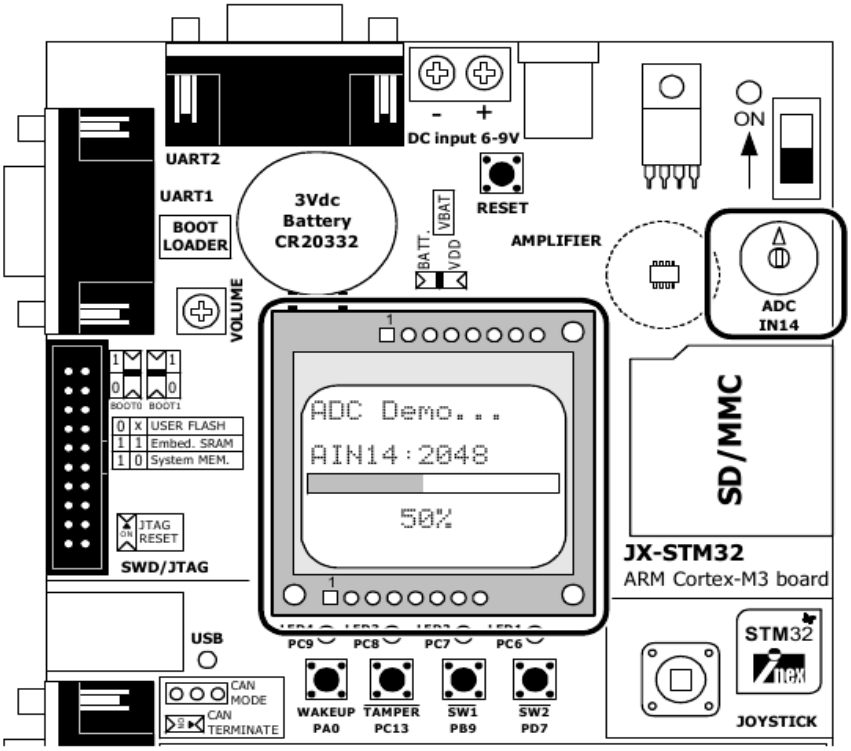
(8.2) Создать исходный файл на языке С ***adc_01.c*** по Листингу L8-1.

(8.3) Скомпилировать файл проекта и загрузить его в микроконтроллер STM32F103VBT6 на плате JX-STM32. Более подробно это описано в главе 2 данного документа.

(8.4) Запустить эту программу. Покрутить ручку потенциометра на порту PC4 платы JX-STM32.

(8.5) Наблюдать как значение экране дисплея GLCD5110 связано с регулировкой потенциометра.

Дисплей GLCD5110 будет показывать то, что показано на картинке ниже:



Значение названное AIN14: цифровые данные, преобразованные АЦП. Диапазон от 0 до 4095.

Значение под индикатором выполнения: относительный коэффициент от 0% до 100% (т. е. 100% = 4095).

Все значения могут быть изменены с помощью регулировки потенциометра на PC4/ADC14 входном порту.

```
//*****************************************************************************************//
// Программа                       : Проверка АЦП
// Описание                        : Получаем аналоговое значение для показа на GLCD5110
//                                 : с помощью опроса EOC флага
// Имя файла                       : adc_01.c
// Компилятор C                    : RkitARM 1.03.0003 для Ride7
// Аппаратно-зависимая библиотека : STMicroelectronics FWlLib V1.0
//*****************************************************************************************//
#include "stm32f10x_lib.h"                          //Подключаем заголовочный файл для STM32
#include "glcd5110.h"                               //Подключаем файл для работы с GLCD5110
#include "stdio.h"                                  //Подключаем файл с функцией sprintf
//*******************************Функция установки RCC-регистров*******************************//
void RCC_setup(void)
{
  ErrorStatus HSEStartUpStatus;                     //Переменная статуса ошибки
  RCC_DeInit();                                     //Сброс RCC-системы (для отладки)
  RCC_HSEConfig(RCC_HSE_ON);                        //Разрешаем работу HSE-генератора
  HSEStartUpStatus = RCC_WaitForHSEStartUp();       //Ждем пока HSE-генератор не будет готов
  if(HSEStartUpStatus == SUCCESS)
  {
      //Установка частот периферийных устройств
    RCC_HCLKConfig(RCC_SYSCLK_Div1);                //HCLK = SYSCLK
    RCC_PCLK2Config(RCC_HCLK_Div1);                 //PCLK2 = HCLK
    RCC_PCLK1Config(RCC_HCLK_Div2);                 //PCLK1 = HCLK/2
    FLASH_SetLatency(FLASH_Latency_2);              //Устанавливаем время ожидания в состояние 2
      //Разрешаем доступ к предварительному буферу
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
      //PLLCLK = 8 МГц * 9 = 72 МГц
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
      //Разрешаем работу с PLL
    RCC_PLLCmd(ENABLE);
      //Ожидаем готовность PLL
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);
      //Устанавливаем PLL как источник сигнала тактовой частоты системы
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
      //Ждем пока PLL не станет источником сигнала тактовой частоты системы
    while(RCC_GetSYSCLKSource() != 0x08);
  }
}
//************************** Функция настройки основного УВВ (GPIO) ****************************//
void GPIO_setup()
{
  GPIO_InitTypeDef GPIO_InitStructure;
    //Разрешаем подачу сигнала тактовой частоты на устройства шины APB2 (PC)
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
    //Настраиваем PC4 как аналоговый вход (АЦП канал 14)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
}
//*****************************Функция настройки параметров АЦП*********************************//
void ADC_setup()
{
  ADC_InitTypeDef ADC_InitStructure;
    //Разрешаем подачу сигнала тактовой частоты на устройства шины APB2 (АЦП1)
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
    //Настройка АЦП1
  ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
  ADC_InitStructure.ADC_ScanConvMode = DISABLE;
  ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
  ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfChannel = 1;
  ADC_Init(ADC1, &ADC_InitStructure);
    //Настройка канала 14 стандартного преобразования АЦП1
  ADC_RegularChannelConfig(ADC1, ADC_Channel_14, 1, ADC_SampleTime_13Cycles5);
  ADC_Cmd(ADC1, ENABLE);                            //Разрешение работы АЦП1
  ADC_ResetCalibration(ADC1);                       //Сброс регистров калибровки АЦП1
  while(ADC_GetResetCalibrationStatus(ADC1));       //Ждем пока регистры сбросятся
  ADC_StartCalibration(ADC1);                       //Пуск калибровки АЦП1
  while(ADC_GetCalibrationStatus(ADC1));            //Ждем пока не закончится калибровка
}
```

Листинг L8-1: adc_01.c — C-файл для микроконтроллера STM32 демонстрирует работу модуля АЦП (начало).

```
//****************************** Основная (главная) функция ********************************//
int main()
{
  unsigned long adc_val=0, percentage;              //Для хранения аналоговых значений
  char str[14];                                     //Для хранения преобразованной строки
  RCC_setup();                                      //Настройка сигналов тактовых частот
  GPIO_setup();                                     //Настройка портов основного УВВ (GPIO)
  ADC_setup();                                      //Настройка АЦП
  lcdString(1, 1, "ADC Demo...");                   //Сообщение в строку 1 дисплея
  lcdUpdate();                                      //Показываем на экране дисплея
  while(1)                                          //Бесконечный цикл
  {
    ADC_Cmd(ADC1, ENABLE);                          //Включаем АЦП1
      //Ждем когда АЦП1 включится (установится флаг EOC)
    while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC)==RESET);
      //Получаем аналоговое значение с АЦП1, канала 14 (PC4)
    adc_val = ADC_GetConversionValue(ADC1);
      //Преобразуем аналоговое значение в проценты (0-100% эквивалентно 0-4095)
    percentage = (adc_val*100)/4095;
    sprintf(str, "AIN14: %d    ", adc_val);         //Преобразуем аналоговое значение в строку
    lcdString(1, 3, str);                           //Сообщение в строку 3 дисплея
      //Показываем индикатор выполнения с (0, 26) длина 83, высота 5 пикселей
    lcdProgBar(0, 26, 83, 5, percentage);
    sprintf(str, "%d%%    ", percentage);           //Преобразуем проценты в строку
    lcdString(6, 5, str);                           //Сообщение в строку 5, столбец 6 дисплея
    lcdUpdate();                                    //Показываем сообщения на экране дисплея
    delay_ms(50);                                   //Задержка 50 мс
  }
}
```

Описание кода

Этот код подключает 3 заголовочных файла:

```
#include <stm32f10x_lib.h>
#include "glcd5110.h"
#include "stdio.h"
```

Разработчик может использовать все ресурсы и аппаратно-зависимые библиотеки. (необходимо скопировать файлы **stm32f10x_conf.h** и **glcd5110.h** в каталог с вашим проектом) а также функцию sprintf для преобразования в строку.

Работа программы может быть разделена на 3 части, показанные ниже:

1. Функция **RCC_setup** - настраивает тактовые частоты, необходимые для работы частей системы.

   1.1 HCLK = SYSCLK

   1.2 PCLK2 = HCLK/1 = 72 МГц

   1.3 PCLK1 = HCLK/2 = 36 МГц

   1.4 ADCCLK = PCLK2/4 = 18 МГц

   LATENCY - 2 состояние ожидания (так как 48 МГц < SYSCLK <= 72 МГц)

2. Функция **GPIO_setup** - настраивает режим работы основного УВВ (GPIO). В этой части порт PC4 настраивается как аналоговый вход АЦП.
3. Функция **ADC_setup** - настраивает работу АЦП. В этом участке кода модуль АЦП устанавливается в независимый режим работы. Выбирается АЦП1, стандартный режим преобразования, аналоговый вход 14, время преобразования равно 13.5 циклов модуля АЦП.

Листинг L8-1: adc_01.c — C-файл для микроконтроллера STM32 демонстрирует работу модуля АЦП (окончание).

8: Пример использования универсального синхронно-асинхронного приемопередатчика (USART).

Универсальный синхронно-асинхронный приемопередатчик (USART) в микроконтроллере STM23F103VBT6 предоставляет гибкие возможности полного дуплексного обмена данными с внешним оборудованием по промышленному асинхронному последовательному формату данных стандарта NRZ. USART обеспечивает очень широкий диапазон скоростей передачи данных используя дискретный генератор скоростей передачи.

Приемопередатчик поддерживает синхронную связь в одну сторону и полудуплексную связь по одному проводу (каналу). Он также поддерживает локальную объединенную сеть (LIN - local interconnection network), протоколы взаимодействия со смарт-картами, инфракрасными портами ввода-вывода (IrDA), спецификации SIR ENDEC и работу с модемом (CTS/RTS). С его помощью можно обеспечить мультипроцессорное взаимодействие.

Высокоскоростной обмен данными возможен с использованием прямого доступа к памяти в мультибуферной конфигурации.

8.1 Основные характеристики

- Полный дуплексный асинхронный обмен;
- Формат стандарта NRZ (Mark/Space);
- Системы дискретного генератора скоростей передачи. Программируемые скорости приема и передачи свыше 4.5 Мбит/с;
- Программируемая длина слова данных (8 или 9 бит);
- Настраиваемая поддержка стоп-битов (1 или 2 стоп-бита);
- Выход тактовой частоты передатчика для синхронизации передачи;
- Возможность эмуляции смарт-карты;
- Полудуплексная связь по одному проводу;
- Настраиваемая мультибуферная связь с использованием ПДП (полного доступа к памяти);
- Отдельные биты доступа для приемника и передатчика;
- Флаги контроля обмена: буфер приемника заполнен, буфер передатчика пуст и завершение проверки флагов передачи;
- Контроль четности: передается бит четности и проверяется четность принятого байта данных;
- Четыре флага обнаружения ошибок: ошибка переполнения, аппаратная ошибка, ошибка синхронизации, ошибка четности;
- Источники прерываний по флагам: изменение CTS, регистр передаваемых данных пуст, передача завершена, регистр принимаемых данных заполнен, приняты ошибочные данные, ошибка переполнения, ошибка

синхронизации, аппаратная ошибка и ошибка четности;

- Мультипроцессорное взаимодействие - входит в режим отсутствия ответа если не происходит соответствие адресов;
- Входит в режим отсутствия ответа (приняты ошибочные данные или произошла остановка по адресу);
- Два режима запуска приемника: управляющий бит (MSB, 9-ый бит), ошибочные данные;

8.2 Общее описание

Приемопередатчик подключается к другому устройству с помощью трех контактов (см. *Рисунок 8-1*). Непосредственная связь между любыми USART требует как минимум 2 контакта: вход принимаемых данных (**RX** - Receive Data In) и выход передаваемых данных (**TX** - Transmit Data Out):

RX: вход принимаемых данных это вход последовательных данных. Метод выборки с запасом по частоте дискретизации используется для восстановления данных с помощью различий между действительно входными данными и шумом.

TX: выход передаваемых данных. Когда передатчик отключен, выходной контакт возвращается в конфигурацию своего порта ввода-вывода. Когда передатчик включен, но ничего не передает, контакт TX находится в высоком уровне. В режимах передачи по одному проводу и связи со смарт-картами, этот контакт используется для передачи и приема данных.

По этим контактам последовательные данные передаются и принимаются в стандартном режиме работы USART в виде пакетов, включающих в себя:

- Неиспользуемую для передачи или приема последовательность данных в начале пакета;
- Старт-бит;
- Слово данных (8 или 9 бит) с наименее значимым битом впереди (младший бит спереди);
- 0.5, 1, 1.5, 2 стоп-бита, определяющих, когда пакет завершён;
- Этот интерфейс использует дискретный генератор скоростей передачи - с 12-битной мантиссой и 4-битным порядком;
- Регистр состояния (USART_SR);
- Регистр данных (USART_DR);
- Регистр управления скоростью передачи (USART_BRR) - 12-битная мантисса и 4-битный порядок;
- Регистр контроля времени (USART_GTPR) для случая работы со смарт-картой;

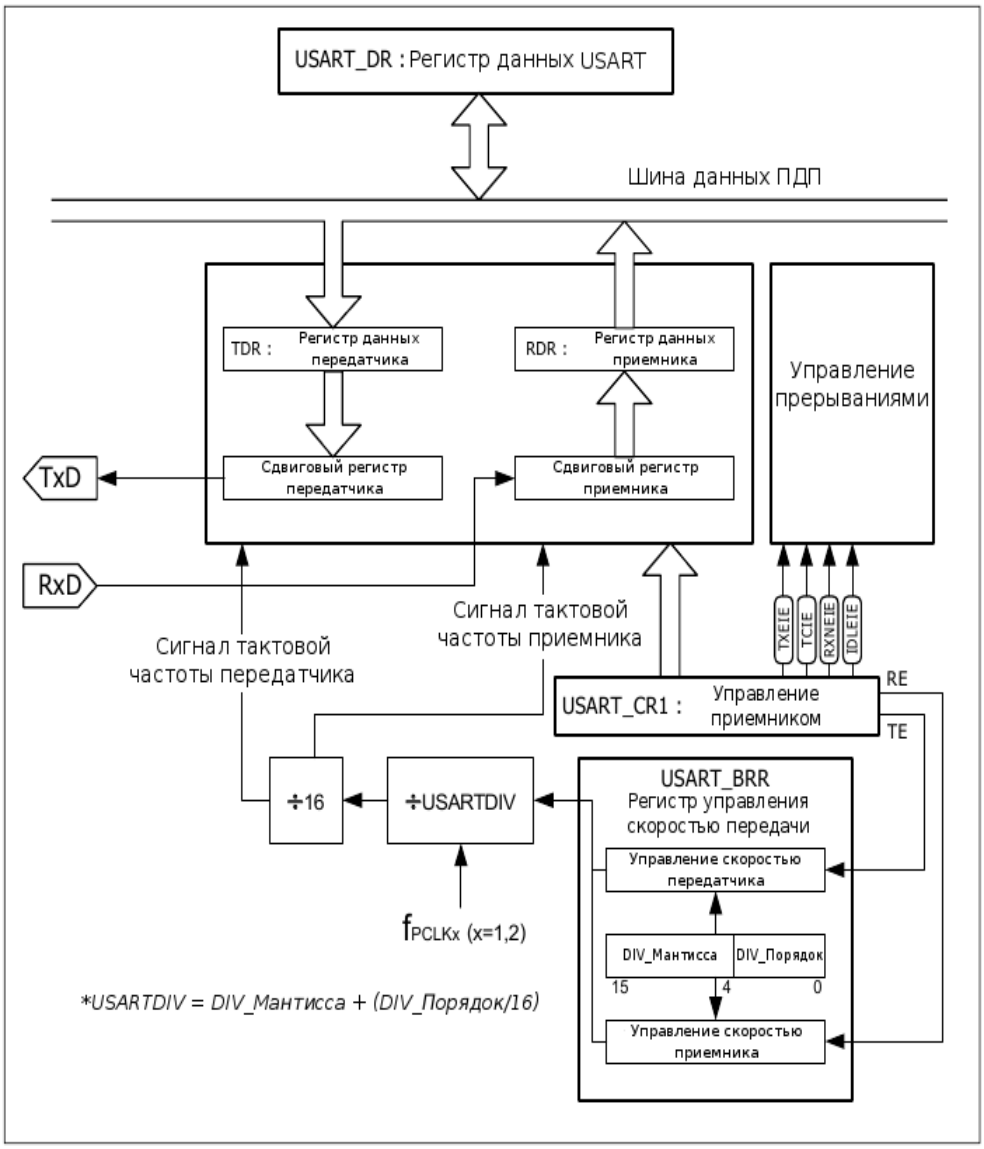


Рисунок 8-1. Структурная схема универсального синхронного-асинхронного приемопередатчика (USART).

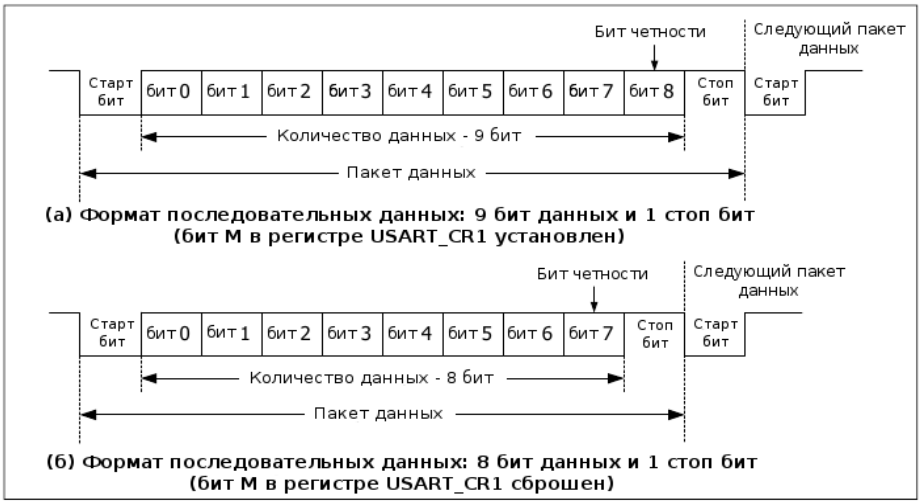


Рисунок 8-2. Формат пакета данных для случаев 8 и 9 бит.

8.3 Последовательность работы с передатчиком

(8.3.1) Включаем модуль USART установкой бита **UE** в регистре **USART_CR1**.

(8.3.2) Выбираем длину пакета данных равной 8 или 9 бит устанавливая бит **M** (8 бит - 0, 9 бит - 1) в регистре **USART_CR1**.

(8.3.3) Устанавливаем количество стоп-битов с помощью записи в регистр **USART_CR2** соответствующего значения.

(8.3.4) Выбираем прямой доступ к памяти (**DMAT**) в регистре **USART_CR3**, если имеет место мультибуферная коммуникация. Настраиваем регистр ПДП, так как он необходим в мультибуферной коммуникации.

(8.3.5) Устанавливаем бит **TE** регистра **USART_CR1** для отправки пустого пакета при установке связи.

(8.3.6) Указываем значение скорости передачи в регистре **USART_BRR**.

(8.3.7) Записываем передаваемые данные в регистр данных **USART_DR**.

Необходимо установить работу порта как порт вывода перед включением модуля USART в режиме передатчика.

8.4 Последовательность работы с приемником

(8.4.1) Включаем модуль USART установкой бита **UE** в регистре **USART_CR1**.

(8.4.2) Выбираем длину пакета данных равной 8 или 9 бит устанавливая бит **M** (8 бит - 0, 9 бит - 1) в регистре **USART_CR1**.

(8.4.3) Устанавливаем количество стоп-битов с помощью записи в регистр **USART_CR2** соответствующего значения.

(8.4.4) Выбираем прямой доступ к памяти (**DMAT**) в регистре **USART_CR3**, если имеет место мультибуферная коммуникация. Настраиваем регистр ПДП, так как он необходим в мультибуферной коммуникации.

(8.4.5) Устанавливаем бит **TE** регистра **USART_CR1** для отправки пустого пакета при установке связи.

(8.4.6) Указываем значение скорости передачи в регистре **USART_BRR**.

(8.4.7) Устанавливаем бит RE в регистре USART_CR1 для включения приёмника USART.

При приёме данных происходит следующая последовательность действий:

- Бит **RXNE** = «1». Его наличие означает окончание переноса данных из сдвигового регистра в регистр данных приёмника (**RDR**);
- Прерывание происходит если установлен бит **RXNEIE** (**RXNEIE** = «1»);
- При возникновении некоторых ошибок, будет установлен соответствующий флаг ошибки;
- Бит **RXNE** автоматически очищается (**RXNE** = «0») после считывания данных из регистра **USART_DR**.

Эксперимент - 9 : Опрос USART1

Этот эксперимент демонстрирует работу USART1 микроконтроллера STM32F103VBT6 в режиме эха. Устройство ожидает ввода данных с клавиатуры компьютера через последовательный порт, после чего отсылает их назад на компьютер для показа в терминальной программе. USART определяет наличие данных, проводя в цикле проверку флага готовности. Этот метод называется «Опрос».

Принципиальная схема включения микроконтроллера изображена на рисунке L9-1.

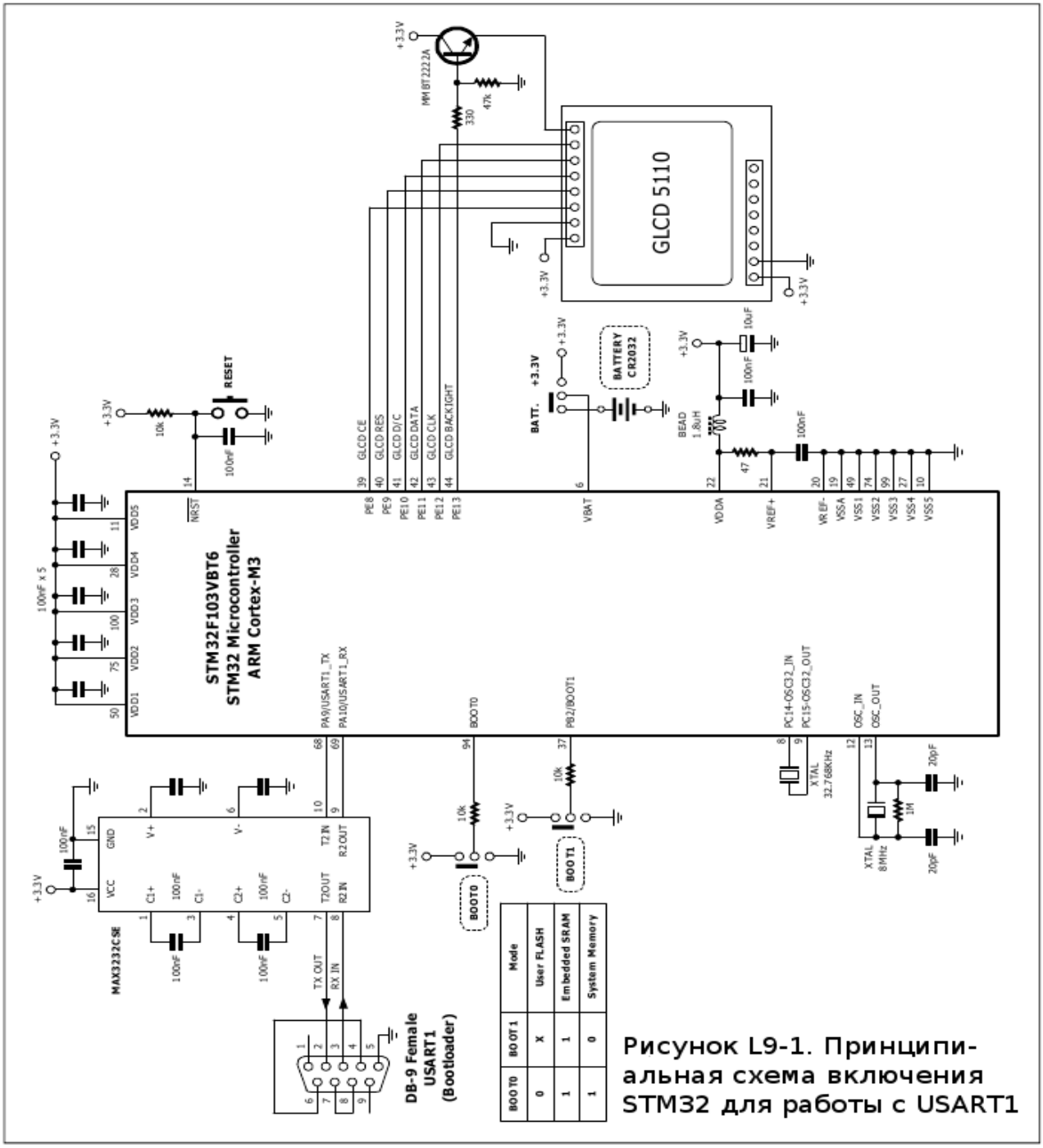


Рисунок L9-1. Принципиальная схема включения STM32 для работы с USART1

Общий порядок действий

(9.1) Создать проект ***usart_01***.

(9.2) Создать исходный файл на языке С ***usart_01.c*** по Листингу L9-1.

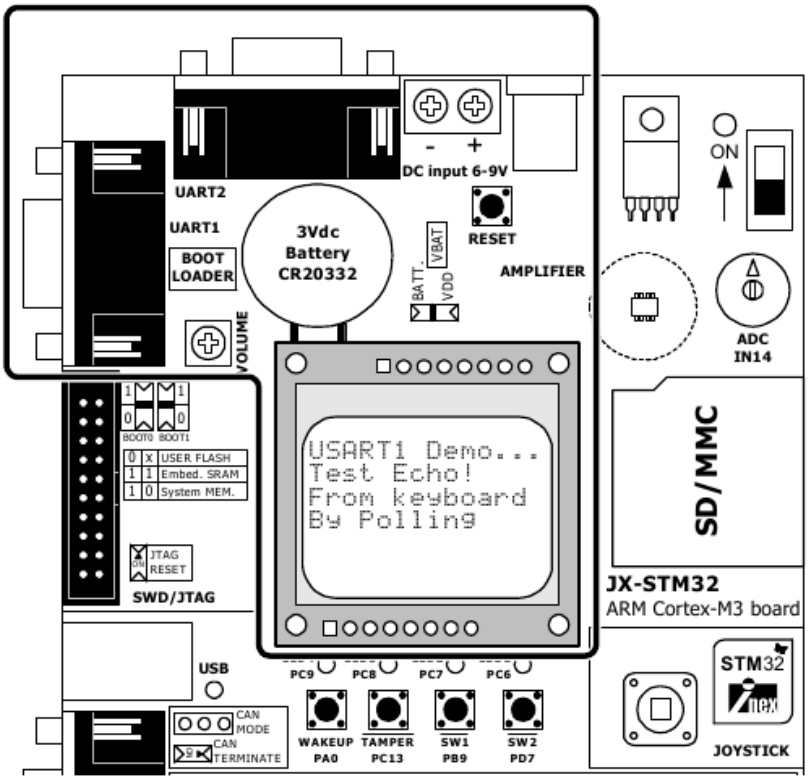
(9.3) Скомпилировать файл проекта и загрузить его в микроконтроллер STM32F103VBT6 на плате JX-STM32. Более подробно это описано в главе 2 данного документа.

(9.4) Закрыть программу Flash loader.

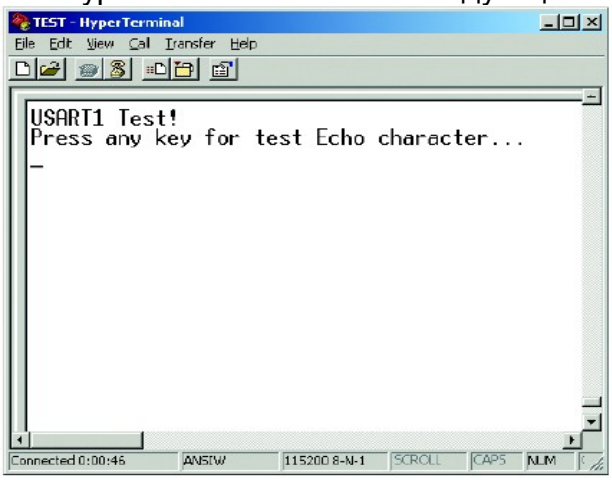
(9.5) Открыть программу Hyper Terminal. Установить следующие параметры связи: скорость передачи (baudrate) - 115200, размер байта (data) - 8 бит, четность (parity) - Нет, стоп-бит - 1.

(9.6) Запустить программу в микроконтроллере.

Дисплей GLCD5110 будет показывать то, что показано на картинке ниже:



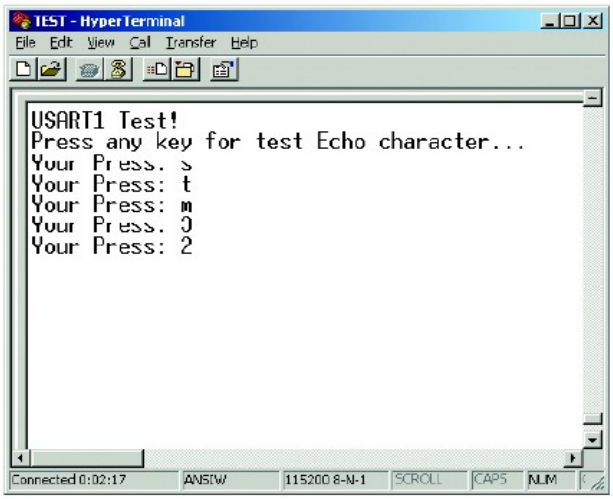
Окно программы Hyper Terminal покажет следующее сообщение:



(9.7) Нажимать клавиши на клавиатуре, при этом окно программы Hyper Terminal должно быть активным. При нажатии на какую-либо клавишу на экране программы Hyper Terminal появится сообщение:

Your Press: [нажатая клавиша]

(9.8) Нажать 5 клавиш: s, t, m, 3, и 2. Hyper Terminal покажет:




```
//*****************************************************************************************//
// Программа                       : Проверка USART1
// Описание                        : Возврат символа с клавиатуры компьютера через программу
//                                 : Hyper Terminal с помощью опроса флагов приёма и передачи
// Имя файла                       : usart_01.c
// Компилятор C                    : RkitARM 1.03.0003 для Ride7
// Аппаратно-зависимая библиотека : STMicroelectronics FWlLib V1.0
//*****************************************************************************************//
#include "stm32f10x_lib.h"                          //Подключаем заголовочный файл для STM32
#include "glcd5110.h"                               //Подключаем файл для работы с GLCD5110
#include "stdio.h"                                  //Подключаем файл с функцией sprintf
//*****************************Функция установки RCC-регистров*******************************//
void RCC_setup(void)
{
  ErrorStatus HSEStartUpStatus;                     //Переменная статуса ошибки
  RCC_DeInit();                                     //Сброс RCC-системы (для отладки)
  RCC_HSEConfig(RCC_HSE_ON);                        //Разрешаем работу HSE-генератора
  HSEStartUpStatus = RCC_WaitForHSEStartUp();       //Ждем пока HSE-генератор не будет готов
  if(HSEStartUpStatus == SUCCESS)
  {
      //Установка частот периферийных устройств
    RCC_HCLKConfig(RCC_SYSCLK_Div1);                //HCLK = SYSCLK
    RCC_PCLK2Config(RCC_HCLK_Div1);                 //PCLK2 = HCLK
    RCC_PCLK1Config(RCC_HCLK_Div2);                 //PCLK1 = HCLK/2
    FLASH_SetLatency(FLASH_Latency_2);              //Устанавливаем время ожидания в состояние 2
      //Разрешаем доступ к предварительному буферу
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
      //PLLCLK = 8 МГц * 9 = 72 МГц
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
      //Разрешаем работу с PLL
    RCC_PLLCmd(ENABLE);
      //Ожидаем готовность PLL
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);
      //Устанавливаем PLL как источник сигнала тактовой частоты системы
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
      //Ждем пока PLL не станет источником сигнала тактовой частоты системы
    while(RCC_GetSYSCLKSource() != 0x08);
  }
}
```

Листинг L9-1: usart_01.c — C-файл для микроконтроллера STM32 демонстрирует работу модуля USART1 в режиме опроса (начало).

```
//************************* Функция настройки основного УВВ (GPIO) ***************************//
void GPIO_setup()
{
  GPIO_InitTypeDef GPIO_InitStructure;
    //Разрешаем подачу сигнала тактовой частоты на устройства шины APB2 (GPIOA)
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO, ENABLE);
    //Настраиваем передатчик USART1 (PA9) как выход
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
    //Настраиваем приёмник USART1 (PA10) как вход
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
}
//*****************************Функция настройки USART1************************************//
void USART1_setup()
{
  USART_InitTypeDef USART_InitStructure;
    //Подаем сигнал тактовой частоты на USART1
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
    //Настраиваем параметры USART1
  USART_InitStructure.USART_BaudRate = 115200;
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
  USART_Init(USART1, &USART_InitStructure);          //Сохраняем настройки USART1
  USART_Cmd(USART1, ENABLE);                         //Подключаем USART1
}
//********************Функция передачи одного символа через USART1*****************************//
void usart1_putc(unsigned char c)
{
    //Ждем пока не будет возможна передача
  while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)==RESET);
    //Передаем символ
  USART_SendData(USART1, (int)c);
}
//************************Функция передачи строки через USART1*********************************//
void usart1_puts(unsigned char *s)
{
  while(*s)                                          //Проверка на конец строки
  {
    usart1_putc(*s++);                               //Передаём 1 символ
  }
}
//*****************Функция ожидания и приёма символа через USART1******************************//
int usart1_getc()
{
    //Ждем пока данные не будут приняты
  while(USART_GetFlagStatus(USART1, USART_FLAG_RXNE)==RESET);
    //Возвращаем символ
  return(USART_ReceiveData(USART1));
}
//****************************** Основная (главная) функция *********************************//
int main()
{
  char str[14];                                      //Сохраняем преобразованную строку
  int msg;                                           //Сохраняем символ
  RCC_setup();                                       //Настройка сигналов тактовых частот
  GPIO_setup();                                      //Настройка портов основного УВВ (GPIO)
  USART1_setup();                                    //Настройка приёмопередатчика USART1
  lcdString(1, 1, "USART1 Demo...");                 //Сообщение в 1-ую строку дисплея
  lcdString(1, 2, "--------");                       //Сообщение во 2-ую строку дисплея
  lcdString(1, 3, "Test Echo!");                     //Сообщение в 3-ую строку дисплея
  lcdString(1, 4, "from keyboard");                  //Сообщение в 4-ую строку дисплея
```

Листинг L9-1: usart_01.c — C-файл для микроконтроллера STM32 демонстрирует работу модуля USART1 в режиме опроса (продолжение).

```
  lcdString(1, 5, "by Polling");                    //Сообщение в 5-ую строку дисплея
  lcdUpdate();                                      //Показываем сообщения на экране дисплея
    //Передаем начальные сообщения
  usart1_puts("\rUSART1 Test!\r\n");
  usart1_puts("Press any key for test Echo character...\r\n");
  while(1)                                          //Бесконечный цикл
  {
    msg = usart1_getc();                            //Ожидаем и принимаем символ
    usart1_puts("Your press: ");                    //Передаём сообщение
    usart1_putc(msg);                               //Передаём принятый символ
    usart1_puts("\r\n");                            //Переход на новую строку
  }
}
```

Описание кода

Этот код подключает 3 заголовочных файла:

```
#include <stm32f10x_lib.h>
#include "glcd5110.h"
#include "stdio.h"
```

Разработчик может использовать все ресурсы и аппаратно-зависимые библиотеки. (необходимо скопировать файлы **stm32f10x_conf.h** и **glcd5110.h** в каталог с вашим проектом) а также функцию sprintf для преобразования в строку.

Код программы содержит 6 важных функций, представленных ниже:

1. Функция **RCC_setup** - настраивает тактовые частоты, необходимые для работы частей системы.

   1.1 HCLK = SYSCLK

   1.2 PCLK2 = HCLK/1 = 72 МГц

   1.3 PCLK1 = HCLK/2 = 36 МГц

   1.4 ADCCLK = PCLK2/4 = 18 МГц

   LATENCY - 2 состояние ожидания (так как 48 МГц < SYSCLK <= 72 МГц)

2. Функция **GPIO_setup** - настраивает режим работы основного УВВ (GPIO). В этой части порт PA9 настраивается как выход передатчика последовательного порта, PA10 как вход приёмника последовательного порта.
3. Функция **USART1_setup** - настраивает работу USART1. В этом участке кода производятся следующие настройки: скорость передачи устанавливается в 115200 бит в секунду, размер пакета данных - 8 бит, низкоуровневый контроль отключается, отсутствует бит четности и количество стоп-битов равно 1.
4. Функция **usart1_getc** - получает символ из приёмного буфера USART1.
5. Функция **usart1_putc** - отправляет символ в передающий буфер USART1.
6. Функция **usart1_puts** — посылает сообщение в передающий буфер USART1.

Основная программа будет запущена в цикле **while(1){...}** для определения нажатия клавиши на клавиатуре с помощью функции **usart1_getc()**. После получения символа он будет отправлен назад с помощью функции **usart1_putc()** и будет отображён на экране компьютера в окне программы Hyper Terminal.

Функция **usart1_puts()** используется для передачи титульного сообщения и некоторых ASCII-команд, таких как **\r** (возврат каретки) и **\n** (переход на новую строку).

Дисплей GLCD5110 показывает некоторые из титульных сообщений с помощью функций lcdString() и lcdUpdate() чтобы сообщить пользователю о работе платы JX-STM32.

Листинг L9-1: usart_01.c — C-файл для микроконтроллера STM32 демонстрирует работу модуля USART1 в режиме опроса (окончание).

Эксперимент - 10 : Опрос USART2

Этот эксперимент похож на эксперимент-9. Изменён только модуль USART: с USART1 на USART2. В большинстве случаев изменения в коде представляют настройку порта PD5 как передающего контакта USART2, а PD6 как приёмного контакта USART2.

Для USART2 на плате JX-STM32 назначены альтернативные контакты. Таким образом, разработчик должен использовать схему включения как указано ниже:

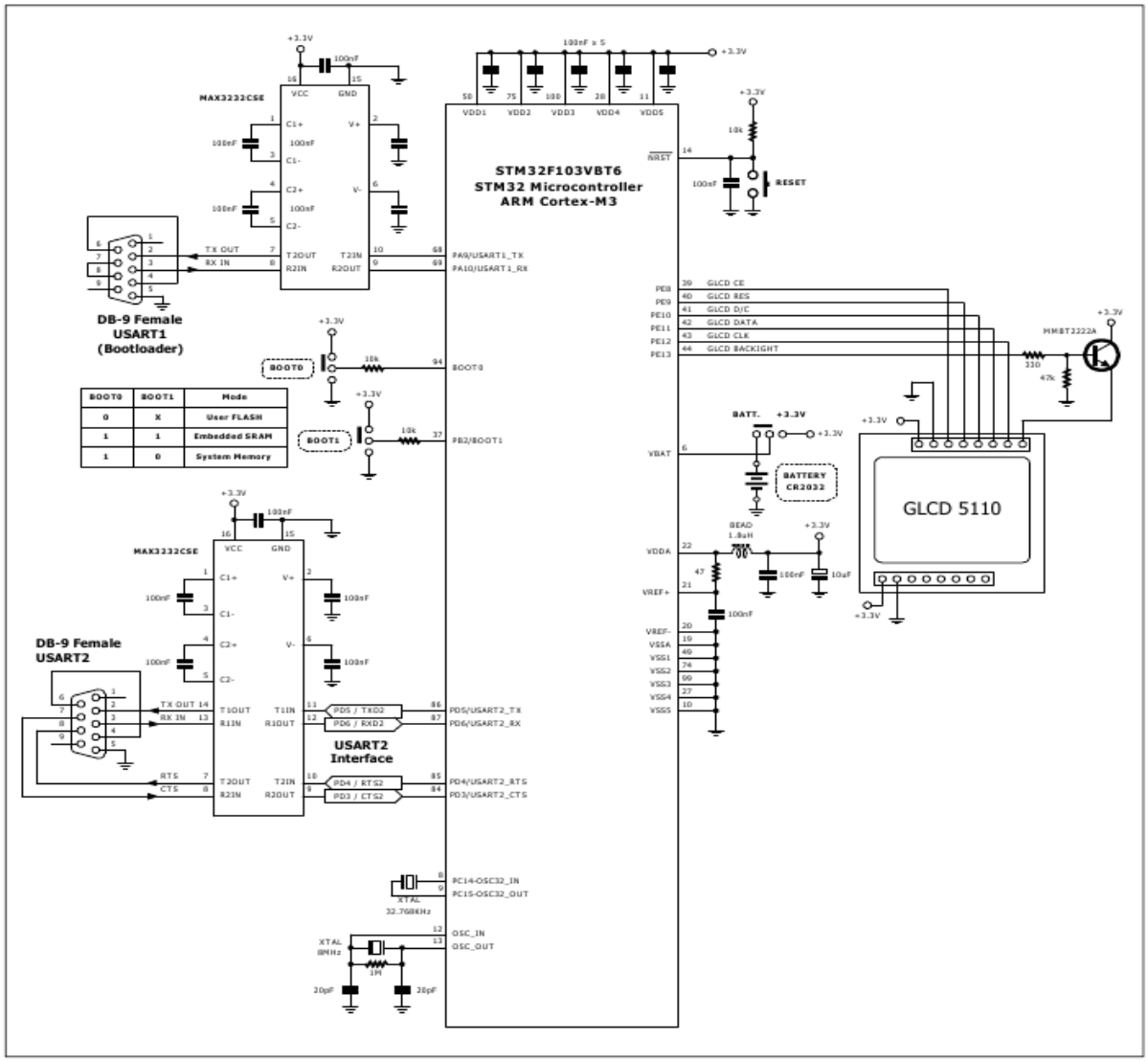


Рисунок L10-1. Принципиальная схема включения STM32 для работы с USART2

Имя контакта	По умолчанию	Переделка
USART2_CTS	PA0	PD3
USART2_RTS	PA1	PD4
USART2_TX	***PA2***	***PD5***
USART2_RX	***PA3***	***PD6***
USART2_CK	PA4	PD7

Общий порядок действий

(10.1) Создать проект ***usart_02***.

(10.2) Создать исходный файл на языке С ***usart_02.c*** по Листингу L10-1.

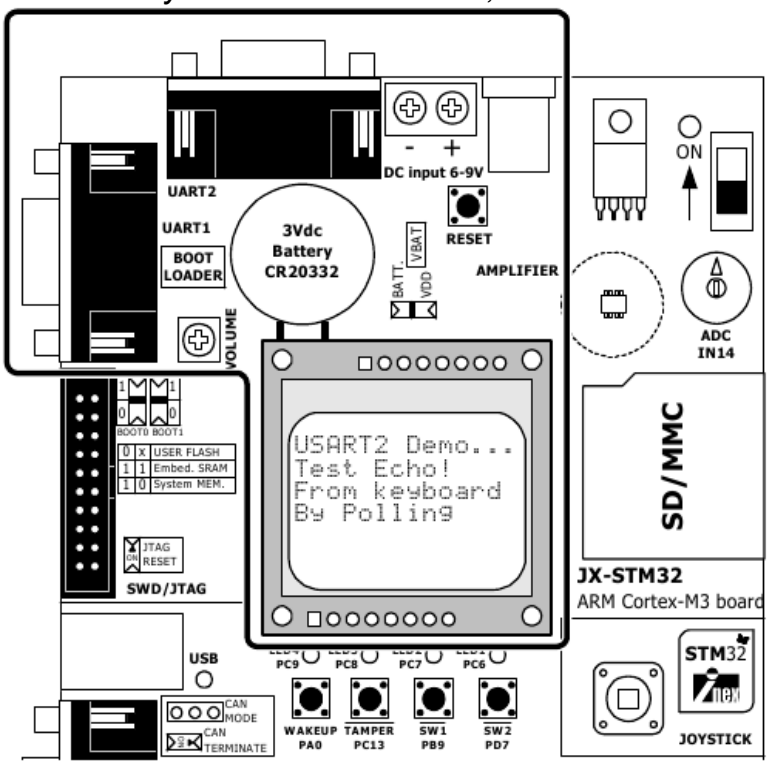
(10.3) Скомпилировать файл проекта и загрузить его в микроконтроллер STM32F103VBT6 на плате JX-STM32. Более подробно это описано в главе 2 данного документа.

(10.4) Закрыть программу Flash loader.

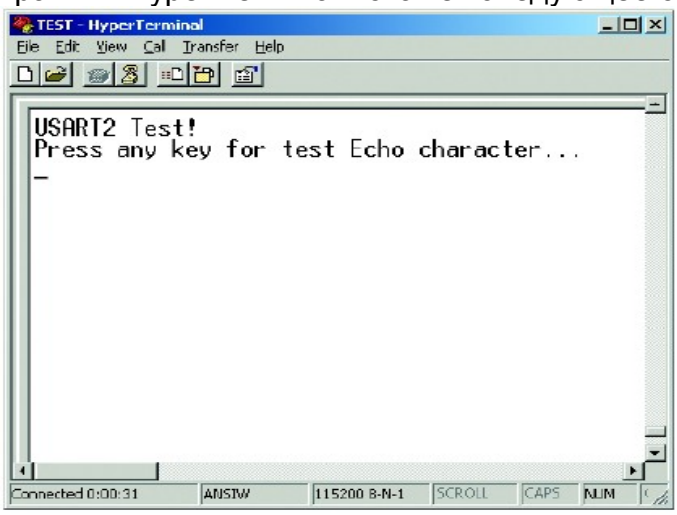
(10.5) Открыть программу Hyper Terminal. Установить следующие параметры связи: скорость передачи (baudrate) - 115200, размер байта (data) - 8 бит, четность (parity) - Нет, стоп-бит - 1.

(10.6) Запустить программу в микроконтроллере.

Дисплей GLCD5110 будет показывать то, что показано на картинке ниже:



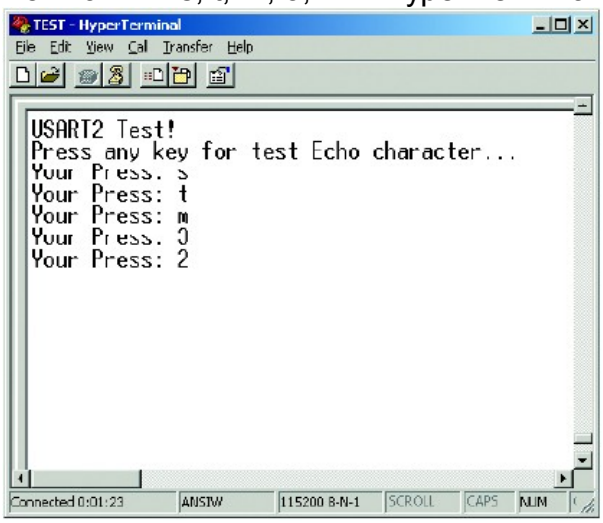
Окно программы Hyper Terminal покажет следующее сообщение:



(10.7) Нажимать клавиши на клавиатуре, при этом окно программы Hyper Terminal должно быть активным. При нажатии на какую-либо клавишу на экране программы Hyper Terminal появится сообщение:

Your Press: [нажатая клавиша]

(10.8) Нажать 5 клавиш: s, t, m, 3, и 2. Hyper Terminal покажет:




```
//*****************************************************************************************//
// Программа                       : Проверка USART2
// Описание                        : Возврат символа с клавиатуры компьютера через программу
//                                 : Hyper Terminal с помощью опроса флагов приёма и передачи
// Имя файла                       : usart_02.c
// Компилятор C                    : RkitARM 1.03.0003 для Ride7
// Аппаратно-зависимая библиотека : STMicroelectronics FWlLib V1.0
//*****************************************************************************************//
#include "stm32f10x_lib.h"                          //Подключаем заголовочный файл для STM32
#include "glcd5110.h"                               //Подключаем файл для работы с GLCD5110
#include "stdio.h"                                  //Подключаем файл с функцией sprintf
//*****************************Функция установки RCC-регистров*******************************//
void RCC_setup(void)
{
  ErrorStatus HSEStartUpStatus;                     //Переменная статуса ошибки
  RCC_DeInit();                                     //Сброс RCC-системы (для отладки)
  RCC_HSEConfig(RCC_HSE_ON);                        //Разрешаем работу HSE-генератора
  HSEStartUpStatus = RCC_WaitForHSEStartUp();       //Ждем пока HSE-генератор не будет готов
  if(HSEStartUpStatus == SUCCESS)
  {
      //Установка частот периферийных устройств
    RCC_HCLKConfig(RCC_SYSCLK_Div1);                //HCLK = SYSCLK
    RCC_PCLK2Config(RCC_HCLK_Div1);                 //PCLK2 = HCLK
    RCC_PCLK1Config(RCC_HCLK_Div2);                 //PCLK1 = HCLK/2
    FLASH_SetLatency(FLASH_Latency_2);              //Устанавливаем время ожидания в состояние 2
      //Разрешаем доступ к предварительному буферу
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
      //PLLCLK = 8 МГц * 9 = 72 МГц
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
      //Разрешаем работу с PLL
    RCC_PLLCmd(ENABLE);
      //Ожидаем готовность PLL
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);
      //Устанавливаем PLL как источник сигнала тактовой частоты системы
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
      //Ждем пока PLL не станет источником сигнала тактовой частоты системы
    while(RCC_GetSYSCLKSource() != 0x08);
  }
}
```

Листинг L10-1: usart_02.c — C-файл для микроконтроллера STM32 демонстрирует работу модуля USART2 в режиме опроса (начало).

```
//************************** Функция настройки основного УВВ (GPIO) ***************************//
void GPIO_setup()
{
  GPIO_InitTypeDef GPIO_InitStructure;
    //Разрешаем подачу сигнала тактовой частоты на устройства шины APB2 (GPIOD)
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD | RCC_APB2Periph_AFIO, ENABLE);
  GPIO_PinRemapConfig(GPIO_Remap_USART2,ENABLE);
    //Настраиваем передатчик USART2 (PD5) как выход
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOD, &GPIO_InitStructure);
    //Настраиваем приёмник USART2 (PD6) как вход
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOD, &GPIO_InitStructure);
}
//*****************************Функция настройки USART2************************************//
void USART2_setup()
{
  USART_InitTypeDef USART_InitStructure;
    //Подаем сигнал тактовой частоты на USART2
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
    //Настраиваем параметры USART2
  USART_InitStructure.USART_BaudRate = 115200;
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
  USART_Init(USART2, &USART_InitStructure);          //Сохраняем настройки USART2
  USART_Cmd(USART2, ENABLE);                         //Подключаем USART2
}
//********************Функция передачи одного символа через USART2*****************************//
void usart2_putc(unsigned char c)
{
    //Ждем пока не будет возможна передача
  while(USART_GetFlagStatus(USART2, USART_FLAG_TXE)==RESET);
    //Передаем символ
  USART_SendData(USART2, (int)c);
}
//***********************Функция передачи строки через USART2*********************************//
void usart2_puts(unsigned char *s)
{
  while(*s)                                          //Проверка на конец строки
  {
    usart2_putc(*s++);                               //Передаём 1 символ
  }
}
//*****************Функция ожидания и приёма символа через USART2*******************************//
int usart2_getc()
{
    //Ждем пока данные не будут приняты
  while(USART_GetFlagStatus(USART2, USART_FLAG_RXNE)==RESET);
    //Возвращаем символ
  return(USART_ReceiveData(USART2));
}
//******************************** Основная (главная) функция *********************************//
int main()
{
  char str[14];                                      //Сохраняем преобразованную строку
  int msg;                                           //Сохраняем символ
  RCC_setup();                                       //Настройка сигналов тактовых частот
  GPIO_setup();                                      //Настройка портов основного УВВ (GPIO)
  USART2_setup();                                    //Настройка приёмопередатчика USART2
  lcdString(1, 1, "USART2 Demo...");                 //Сообщение в 1-ую строку дисплея
  lcdString(1, 2, "--------");                       //Сообщение во 2-ую строку дисплея
  lcdString(1, 3, "Test Echo!");                     //Сообщение в 3-ую строку дисплея
  lcdString(1, 4, "from keyboard");                  //Сообщение в 4-ую строку дисплея
```

Листинг L10-1: usart_02.c — C-файл для микроконтроллера STM32 демонстрирует работу модуля USART2 в режиме опроса (продолжение).

```
  lcdString(1, 5, "by Polling");                    //Сообщение в 5-ую строку дисплея
  lcdUpdate();                                      //Показываем сообщения на экране дисплея
    //Передаем начальные сообщения
  usart2_puts("\rUSART2 Test!\r\n");
  usart2_puts("Press any key for test Echo character...\r\n");
  while(1)                                          //Бесконечный цикл
  {
    msg = usart2_getc();                            //Ожидаем и принимаем символ
    usart2_puts("Your press: ");                    //Передаём сообщение
    usart2_putc(msg);                               //Передаём принятый символ
    usart2_puts("\r\n");                            //Переход на новую строку
  }
}
```

Описание кода

Код программы похож на код программы эксперимента-9. Различия в функции **GPIO_setup**, поскольку для USART2 на плате JX-STM32 назначены альтернативные контакты. Код этой функции представлен ниже.

```
    //Настраиваем передатчик USART2 (PD5) как выход
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOD, &GPIO_InitStructure);
    //Настраиваем приёмник USART2 (PD6) как вход
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOD, &GPIO_InitStructure);
```

Функция **USART2_setup** - настраивает работу USART2. В этом участке кода производятся следующие настройки: скорость передачи устанавливается в 115200 бит в секунду, размер пакета данных - 8 бит, низкоуровневый контроль отключается, отсутствует бит четности и количество стоп-битов равно 1.

Функция **usart2_getc** - получает символ из приёмного буфера USART2.

Основная программа будет запущена в цикле **while(1){...}** для определения нажатия клавиши на клавиатуре с помощью функции **usart2_getc()**. После получения символа он будет отправлен назад с помощью функции **usart2_putc()** и будет отображён на экране компьютера в окне программы Hyper Terminal.

Функция **usart2_puts()** используется для передачи титульного сообщения и некоторых ASCII-команд, таких как **\r** (возврат каретки) и **\n** (переход на новую строку).

Дисплей GLCD5110 показывает некоторые из титульных сообщений с помощью функций lcdString() и lcdUpdate() чтобы сообщить пользователю о работе платы JX-STM32.

Листинг L10-1: usart_02.c — C-файл для микроконтроллера STM32 демонстрирует работу модуля USART2 в режиме опроса (окончание).

Эксперимент - 11 : Работа с USART1 с помощью прерываний

Этот эксперимент показывает работу USART1 микроконтроллера STM32F103VBT6 в режиме эха как и в эксперименте 9, но не методом опроса, а по прерываниям. Принципиальная схема включения микроконтроллера такая же, как и в эксперименте-9.

Общий порядок действий

(11.1) Создать проект ***usart_03***.

(11.2) Создать исходный файл на языке С ***usart_03.c*** по Листингу L11-1.

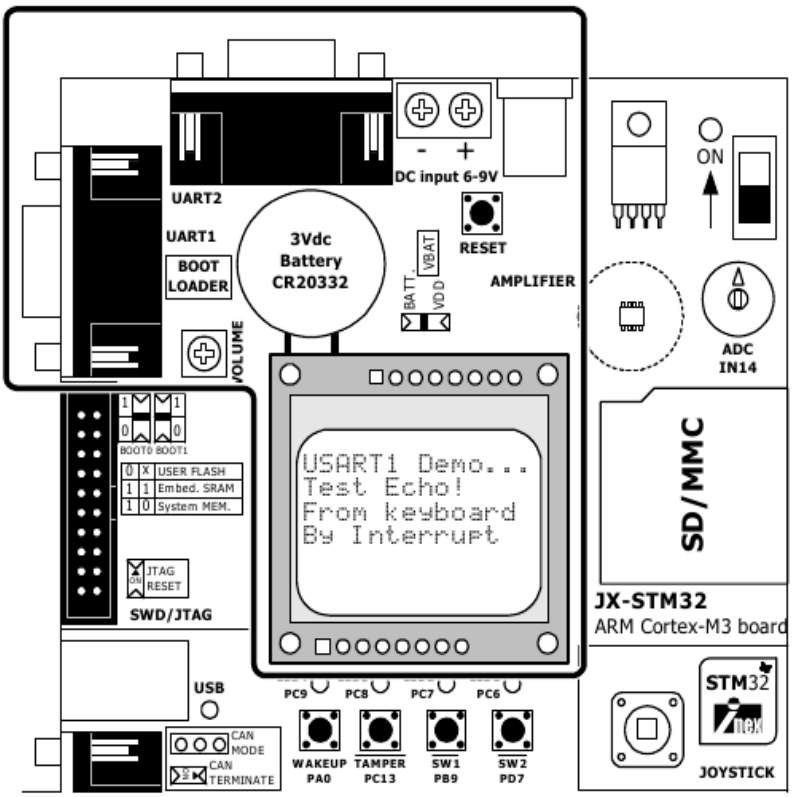
(11.3) Скомпилировать файл проекта и загрузить его в микроконтроллер STM32F103VBT6 на плате JX-STM32. Более подробно это описано в главе 2 данного документа.

(11.4) Закрыть программу Flash loader.

(11.5) Открыть программу Hyper Terminal. Установить следующие параметры связи: скорость передачи (baudrate) - 115200, размер байта (data) - 8 бит, четность (parity) - Нет, стоп-бит - 1.

(11.6) Запустить программу в микроконтроллере.

Дисплей GLCD5110 будет показывать то, что показано на картинке ниже:



(11.7) Выполнить все остальные шаги, как и в эксперименте-9.

```
//*****************************************************************************************//
// Программа                       : Проверка USART1 через прерывания
// Описание                        : Возврат символа с клавиатуры компьютера через программу
//                                 : Hyper Terminal с помощью прерываний
// Имя файла                       : usart_03.c
// Компилятор C                    : RkitARM 1.03.0003 для Ride7
// Аппаратно-зависимая библиотека : STMicroelectronics FWlLib V1.0
//*****************************************************************************************//
#include "stm32f10x_lib.h"                          //Подключаем заголовочный файл для STM32
#include "glcd5110.h"                               //Подключаем файл для работы с GLCD5110
#include "string.h"                                 //Подключаем файл с функцией sprintf
#define TxBufferSize   100                          //Максимальное к-во передаваемых байт
u8 TxBuffer[TxBufferSize]={0};                      //Буфер передатчика (обнуленный)
vu8 TxCounter=0;                                    //Количество передаваемых байт
vu8 RxMessage=0;                                    //Принимаемые данные
vu8 RxUpdate=0;                                     //Флаг, сообщающий о поступлении данных
vu8 TxReady=1;                                      //Флаг, сообщающий о готовности передатчика
//*****************************Функция установки RCC-регистров*******************************//
void RCC_setup(void)
{
  ErrorStatus HSEStartUpStatus;                     //Переменная статуса ошибки
  RCC_DeInit();                                     //Сброс RCC-системы (для отладки)
  RCC_HSEConfig(RCC_HSE_ON);                        //Разрешаем работу HSE-генератора
  HSEStartUpStatus = RCC_WaitForHSEStartUp();       //Ждем пока HSE-генератор не будет готов
  if(HSEStartUpStatus == SUCCESS)
  {
     //Установка частот периферийных устройств
    RCC_HCLKConfig(RCC_SYSCLK_Div1);                //HCLK = SYSCLK
    RCC_PCLK2Config(RCC_HCLK_Div1);                 //PCLK2 = HCLK
    RCC_PCLK1Config(RCC_HCLK_Div2);                 //PCLK1 = HCLK/2
    RCC_ADCCLKConfig(RCC_PCLK2_Div4);               // ADCCLK = PCLK2/4
    FLASH_SetLatency(FLASH_Latency_2);              //Устанавливаем время ожидания в состояние 2
      //Разрешаем доступ к предварительному буферу
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
      //PLLCLK = 8 МГц * 9 = 72 МГц
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
      //Разрешаем работу с PLL
    RCC_PLLCmd(ENABLE);
      //Ожидаем готовность PLL
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);
      //Устанавливаем PLL как источник сигнала тактовой частоты системы
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
      //Ждем пока PLL не станет источником сигнала тактовой частоты системы
    while(RCC_GetSYSCLKSource() != 0x08);
  }
}
//************************** Функция настройки основного УВВ (GPIO) ****************************//
void GPIO_setup()
{
  GPIO_InitTypeDef GPIO_InitStructure;
    //Разрешаем подачу сигнала тактовой частоты на устройства шины APB2 (GPIOA)
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO, ENABLE);
    //Настраиваем передатчик USART1 (PA9) как выход
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
    //Настраиваем приёмник USART1 (PA10) как вход
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
}
//************************Функция настройки вектора прерываний (NVIC)*****************************//
void NVIC_setup(void)
{
  NVIC_InitTypeDef NVIC_InitStructure;
    //Разрешаем прерывание на USART1
  NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQChannel;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
```

Листинг L11-1: usart_03.c — C-файл для микроконтроллера STM32 демонстрирует работу модуля USART1 по прерываниям (начало).

```
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}
//*****************************Функция настройки USART1************************************//
void USART1_setup()
{
  USART_InitTypeDef USART_InitStructure;
    //Подаем сигнал тактовой частоты на USART1
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
    //Настраиваем параметры USART1
  USART_InitStructure.USART_BaudRate = 115200;
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
  USART_Init(USART1, &USART_InitStructure);          //Сохраняем настройки USART1
  USART_ITConfig(USART1, USART_IT_TXE, DISABLE);     //Запрещаем прерывание USART1 на передачу
  USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);     //Разрешаем прерывание USART1 на приём
  USART_Cmd(USART1, ENABLE);                         //Подключаем USART1
}
//***********************Функция передачи строки через USART1*********************************//
void usart1_puts(unsigned char *s)
{
  while(TxReady == 0);                               //Проверяем флаг готовности к передаче
  TxCounter = 0;                                     //Очищаем количество передаваемых байт
  strcpy(TxBuffer, s);                               //Копируем данные в передающий буфер
  USART_ITConfig(USART1, USART_IT_TXE, ENABLE);      //Разрешаем прерывание USART1 на передачу
  TxReady = 0;                                       //Очищаем флаг готовности (передатчик занят)
}
//***********************Функция передачи символа через USART1*********************************//
void usart1_putc(unsigned char c)
{
  while(TxReady == 0);                               //Проверяем флаг готовности к передаче
  TxCounter = 0;                                     //Очищаем количество передаваемых байт
  TxBuffer[0] = c;                                   //Копируем символ в передающий буфер
  TxBuffer[1] = '\n';                                //Конец строки
  USART_ITConfig(USART1, USART_IT_TXE, ENABLE);      //Разрешаем прерывание USART1 на передачу
  TxReady = 0;                                       //Очищаем флаг готовности (передатчик занят)
}
//***********************Функция обработки прерываний USART1*********************************//
void USART1_IRQHandler(void)
{
    //Обработка прерываний по приему
  if(USART_GetITStatus(USART1,USART_IT_RXNE)!=RESET) //Произошло прерывание по приёму?
  {
    RxMessage = USART_ReceiveData(USART1);           //Читаем 1 байт из регистра прм. данных
    RxUpdate = 1;                                    //Устанавливаем флаг приёма данных
    USART_ClearITPendingBit(USART1, USART_IT_RXNE);  //Очищаем бит прерывания по приему
  }
    //Обработка прерываний по передаче
  if(USART_GetITStatus(USART1,USART_IT_TXE)!=RESET)  //Произошло прерывание по передаче?
  {
    USART_SendData(USART1,TxBuffer[TxCounter++]);    //Записываем 1 байт в регистр прд. данных
    USART_ClearITPendingBit(USART1, USART_IT_TXE);   //Очищаем бит прерывания по передаче
      //Проверка на конец передачи или превышения размера буфера для передачи
    if(TxBuffer[TxCounter] == 0 || TxCounter > TxBufferSize)
    {
      USART_ITConfig(USART1, USART_IT_TXE, DISABLE); //Запрещаем прерывание USART1 на передачу
      TxReady=1;                                     //Устанавливаем флаг готовности передатчика
    }
  }
}
//****************************** Основная (главная) функция *********************************//
int main()
{
  char str[14];                                      //Сохраняем преобразованную строку
  RCC_setup();                                       //Настройка сигналов тактовых частот
  GPIO_setup();                                      //Настройка портов основного УВВ (GPIO)
  USART1_setup();                                    //Настройка приёмопередатчика USART1
  NVIC_setup();                                      //Настройка вектора прерываний (NVIC)
```

Листинг L11-1: usart_03.c — C-файл для микроконтроллера STM32 демонстрирует работу модуля USART1 по прерываниям (продолжение).

```
  lcdString(1, 1, "USART2 Demo...");                //Сообщение в 1-ую строку дисплея
  lcdString(1, 2, "--------");                      //Сообщение во 2-ую строку дисплея
  lcdString(1, 3, "Test Echo!");                    //Сообщение в 3-ую строку дисплея
  lcdString(1, 4, "from keyboard");                 //Сообщение в 4-ую строку дисплея
  lcdString(1, 5, "by Interrupt");                  //Сообщение в 5-ую строку дисплея
  lcdUpdate();                                      //Показываем сообщения на экране дисплея
    //Передаем начальные сообщения
  usart1_puts("\rUSART2 Test!\r\n");
  usart1_puts("Press any key for test Echo character! (Interrupt Method)\r\n");
  while(1)                                          //Бесконечный цикл
  {
    if(RxUpdate==1)                                 //Ожидаем входные данные с клавиатуры
    {
      RxUpdate = 0;                                 //
      usart1_puts("Your press: ");                  //Передаём сообщение
      usart1_putc(RxMessage);                       //Передаём принятый символ
      usart1_puts("\r\n");                          //Переход на новую строку
    }
  }
}
```

Описание кода

В начале данного кода подключаются заголовочные файлы и устанавливаются некоторые важные параметры, показанные ниже:

```
#include "stm32f10x_lib.h"                          //Подключаем заголовочный файл для STM32
#include "glcd5110.h"                               //Подключаем файл для работы с GLCD5110
#include "string.h"                                 //Подключаем файл с функцией sprintf
#define TxBufferSize   100                          //Максимальное к-во передаваемых байт
u8 TxBuffer[TxBufferSize]={0};                      //Буфер передатчика (обнуленный)
vu8 TxCounter=0;                                    //Количество передаваемых байт
vu8 RxMessage=0;                                    //Принимаемые данные
vu8 RxUpdate=0;                                     //Флаг, сообщающий о поступлении данных
vu8 TxReady=1;                                      //Флаг, сообщающий о готовности передатчика
```

Функция **USART1_setup** настраивает работу USART1 и разрешает доступ к его прерываниям для приема и передачи. Флаг **TXE** установлен когда передатчик готов передавать данные, а флаг **RXNE** установлен, когда приёмником приняты данные.

Две важные функции: **NVIC_setup()** и **USART1_IRQHandler()**. **NVIC_setup()** используется для настройки источника прерываний USART1 в контроллере вектора прерываний (NVIC). **USART1_IRQHandler()** - функция реакции на прерывания от USART1.

Листинг L11-1: usart_03.c — C-файл для микроконтроллера STM32 демонстрирует работу модуля USART1 по прерываниям (окончание).

Содержание