Лабораторная работа №1
Создание проекта в среде разработки Cube IDE. Работа с портами ввода/вывода

Цель работы: ознакомиться с созданием проектов для платы, рассмотреть их структуру; изучить принципы работы с портами ввода/вывода и организации их взаимодействия.

Оборудование и программное обеспечение: плата UDK32F107V, среда разработки Cube IDE, светодиоды - 2 шт, беспаечная макетная плата, провода соединительные.

Теоретический материал

Микроконтроллер STM32F107VCT построен на ядре arm cortex-m3, имеет ширину шины данных 32-бит, и тактовую частоту ядра 72 МГЦ. Содержит в себе 80 входов/выходов и обладает объемом памяти программ 256 кбайт flash-типа. Объем RAM составляет 64Кб. Имеются в наличии такая периферия, как АЦП 16x12b и ЦАП 2x12b, встроенные интерфейсы can, eth, i2c, irda, lin, spi, uart, usbotg, встроенная периферия dma, por, pwm, voltagedetect, wdt. Питается микроконтроллер напряжением в диапазоне 2…3.6 в.

Светодиоды, задействованные в этом проекте будут подключены к таким портам: PB10 (интегрированный в плату светодиод), PA3 и PC3 (внешние светодиоды). Кнопки для управления состоянием светодиодов подключены к PC13 – Tamper и PA0 – Wake UP. Кнопки установлены прямо на плате UDK32 и могут использоваться в проекте.

Организация задержки

Важнейшей функцией почти всех приложений на микроконтроллерах является вычисление задержек времени в программе (отслеживание таймаутов во время передачи данных, периодический опрос клавиатуры и т. п.). Сложно

представить программу для микроконтроллера, в которой не будет необходимости использовать временные задержки. Реализовать задержку можно несколькими способами:

- HAL_Delay;
- Задержка на основе DWT;
- Задержка на основе SysTick.

Задержка на основе HAL_Delay

В библиотеках HAL есть функция HAL_Delay, реализованная на основе миллисекундного таймера. В качестве параметра она принимает значение задержки, указанное в миллисекундах.Является наиболее простой реализацией задержки, однако при её использовании тормозится вся программа. Для задержки в 500мс требуется написать всего одну строку HAL_Delay(500).

Задержка на основе DWT

Счётчик DWT – это обычный 32-х битный регистр, значение в котором увеличивается на единичку с каждым последующим тактом. В этот регистр можно как писать, так и читать из него, а значит можно с его помощью измерять время выполнения программы в тактах, и организовывать микросекундные задержки. Этот счётчик полностью независимый, и работает не смотря ни на прерывания, ни на другие задержки. Для использования нужно в программе создать такую функцию:

```
#define DWT_CONTROL *(volatile unsigned long *)0xE0001000
#define SCB_DEMCR   *(volatile unsigned long *)0xE000EDFC
void DWT_Init(void)
{
  SCB_DEMCR |= CoreDebug_DEMCR_TRCENA_Msk; // разрешаем использовать счётчик
  DWT_CONTROL |= DWT_CTRL_CYCCNTENA_Msk;  // запускаем счётчик
```

```
}

void delay_micros(uint32_t us)
{
   uint32_t us_count_tic =  us * (SystemCoreClock / 1000000); // получаем кол-во тактов за 1 мкс и умножаем на наше значение
   DWT->CYCCNT = 0U; // обнуляем счётчик
   while(DWT->CYCCNT < us_count_tic);
}
```

Перед бесконечным циклом инициализируем DWT:

```
/* USER CODE BEGIN 2 */
DWT_Init();
/* USER CODE END 2 */
```

И там где нужно, вставляем задержку, например 100 микросекунд:

```
delay_micros(100);
```

Задержка на основе SysTick;

В ядро процессора STM32 встроен 24-битный системный таймер, так называемый SysTick (STK), который считает в обратном направлении от загруженной в таймер величины до нуля. В момент достижения значения 0 счетчик автоматически на следующем тактовом перепаде перезагружает сам себя значением из регистра STK_LOAD, и далее продолжает счет вниз, считая каждый приходящий импульс тактирования. Когда процессор остановлен (halted) для отладки, то счетчик не декрементируется.

Таймер SysTick (Cortex System Timer) специально предназначен для операционных систем реального времени (real-time operating system, RTOS), но может использоваться просто как стандартный счетчик вниз. Его возможности:

• 24-битный счетчик с обратным отсчетом.

• Возможность автозагрузки.
• Маскируемая генерация системного прерывания, когда счетчик достигает 0.
• Программируемый источник тактирования.

Вместе с таймером SysTick написание хорошей функции задержки становится простой задачей. Нужно заранее установить SysTick на требуемую величину, чтобы получить в результате задержки к примеру 1 мс, 100 мкс, 10 мкс и даже 1 мкс. Выбранное значение должно соответствовать потребностям программы, например, если требуемые задержки должны быть в диапазоне единиц и десятков миллисекунд, то подходящим значением для настройки SysTick будет интервал 100 мкс или 1 мс.
В файл stm32f1xx_it.c нужно написать следующий код:

```
/* USER CODE BEGIN PV */
volatile uint32_t timestamp = 0;
/* USER CODE END PV */

void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */
  /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  /* USER CODE BEGIN SysTick_IRQn 1 */
   //сюда попадаем каждую 1 миллисекунду
   timestamp++;
  /* USER CODE END SysTick_IRQn 1 */
}
{
   if (delaystamp < timestamp)
   {
```

```
HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_10); // Инвертирование состояния выхода PB10 (интегрированный в плату светодиод)
    delaystamp = timestamp + 500; //полупериод мигания 0.5 сек.
  }
}
```

Переменная timestamp постоянно увеличивает свое значение каждые 1 мс. Как только пройдет 500 мс, значение timestamp станет больше переменной delaystamp, и выполнится код, который переключит светодиод в противоположное состояние. Потом переменная delaystamp снова загружается на величину, на 500 превышающую timestamp, чем обеспечивается отслеживание очередного интервала задержки 500 мс.

Создание проекта

Для начала выполнения работы нужно запустить Cube IDE и создать в нём новый проект. Для этого требуется нажать "Create new STM32 Project". В появившемся окне выбираем серию STM32F1 и ищем микроконтроллер STM32F107VCT (Рис. 1)

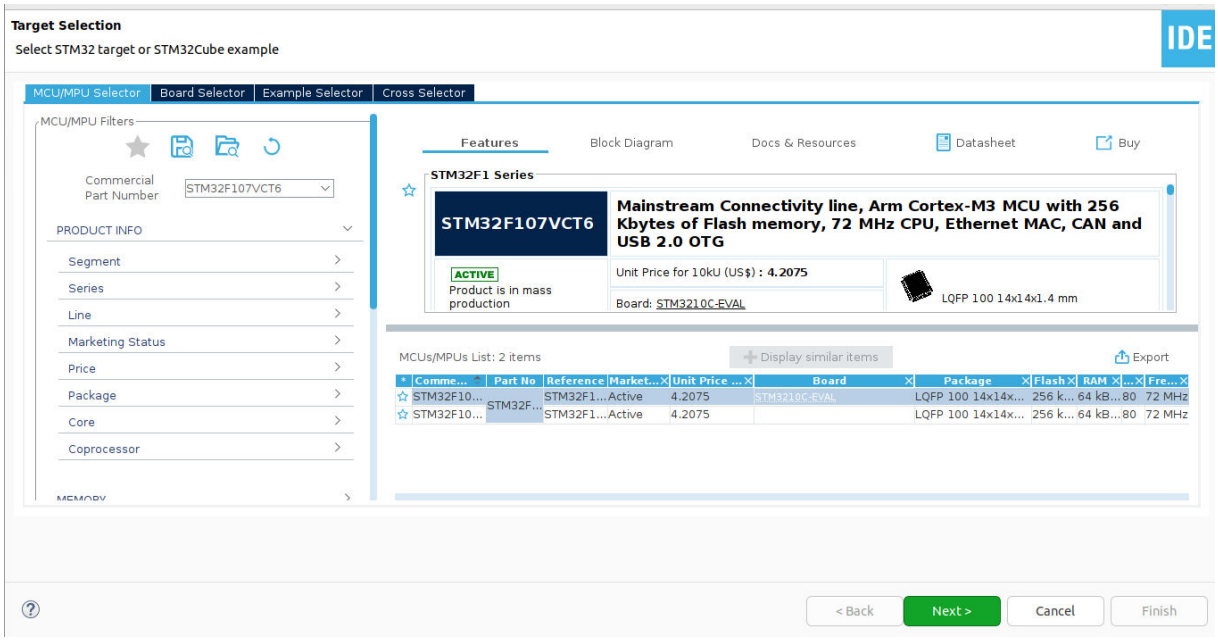

Рис. 1. Окно выбора микроконтроллера

После выбора контроллера откроется окно, в котором нужно будет указать имя проекта и не меняя настроек нажать «Next». В следующем окне выбрать параметр «Add necessary library files as referance in the toolchain project configuration file» и далее «Finish».

Далее откроется окно конфигурирования, где нужно будет настроить параметры микроконтроллера. Для начала необходимо настроить тактовый генератор (рис. 2). Параметры тактового генератора задаются в "System core>RCC", где нужно активировать параметр High Speed Clock: Crystal/Ceramic Resonator. После чего во вкладке "Clock Configuration" в опции HCLK задать частоту 72МГц.

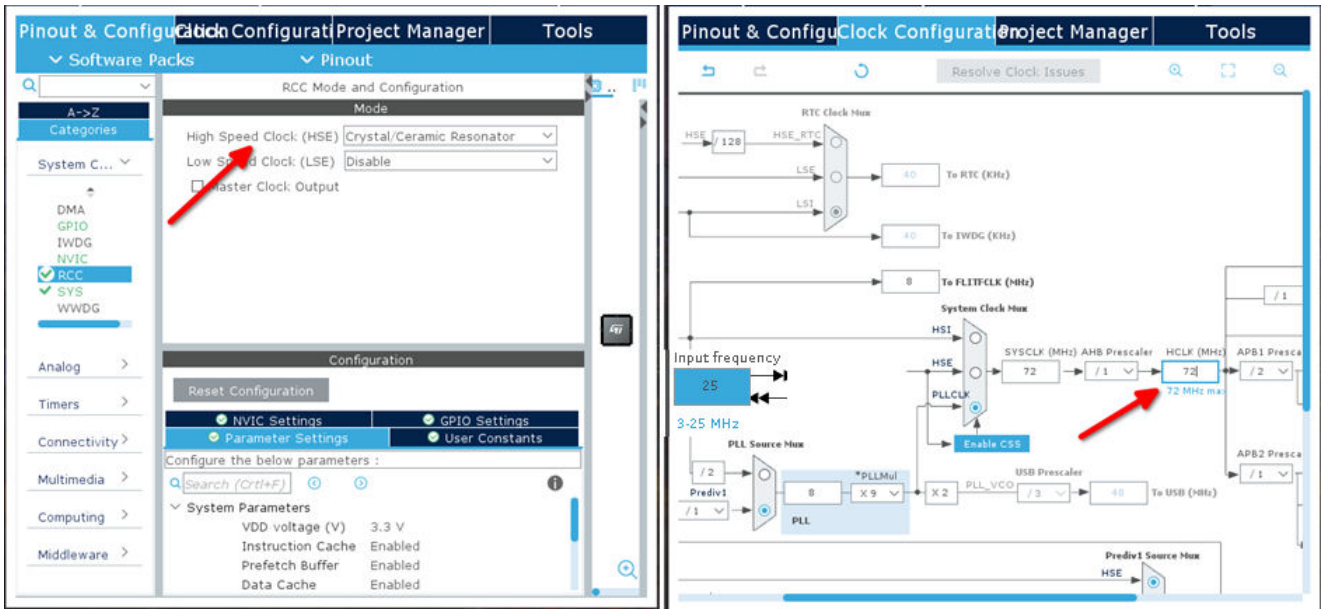

Рис. 2. Окно настройки тактирования микроконтроллера

Также необходимо задать параметры отладчика. Для этого в разделе "System core>SYS" нужно выбрать режим отладки "JTAG (5 pins)" (рис. 3).

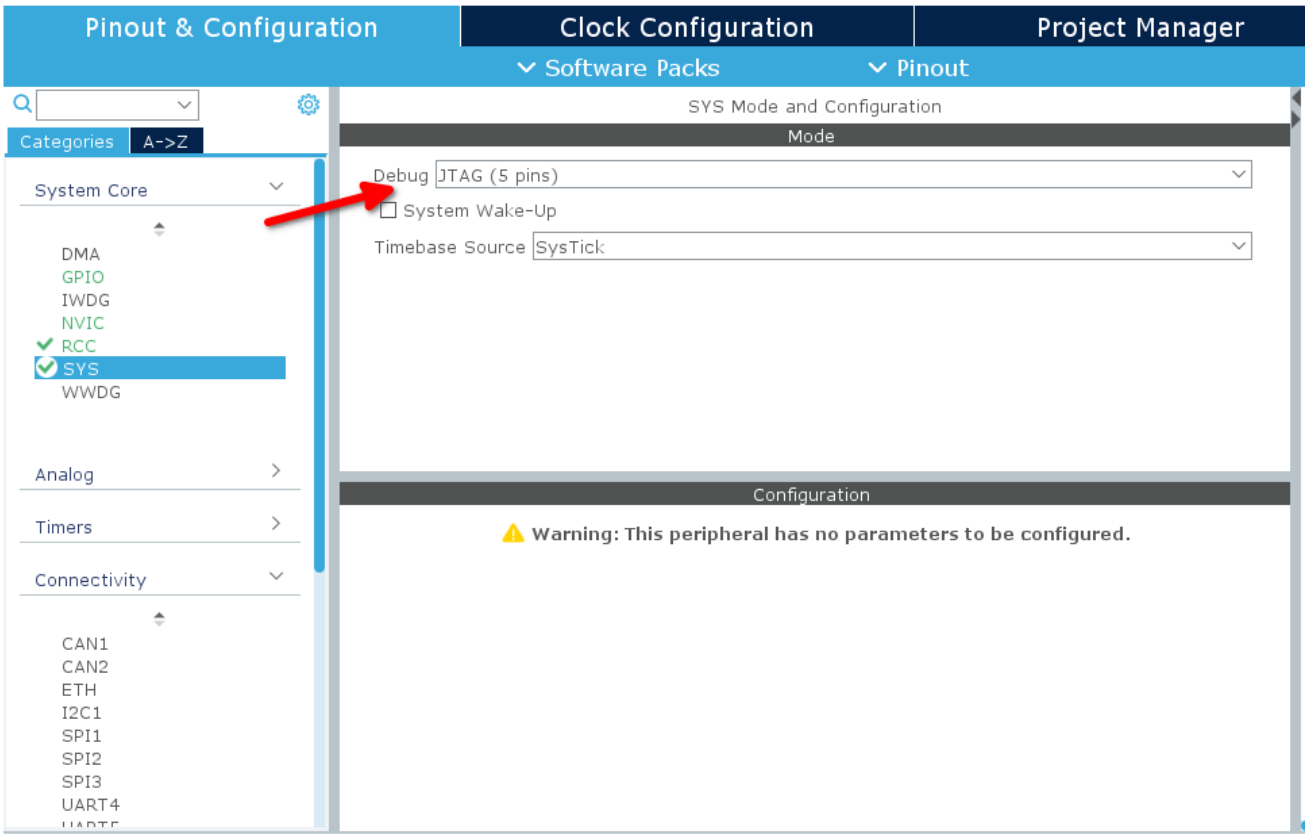

Рис. 3. Окно выбора режима отладки

Следует отметить, что вышеописанные настройки необходимо производить каждый раз при создании проекта.

На этом первоначальная настройка завершена. Теперь необходимо задать настройки портов ввода/вывода. В начале работы в исходных данных указаны порты, которые будут задействованы в проекте. В соответствии с этими данными производим настройку портов. Переходим к "Pinout view" и ищем порты: PB10, PC3 и PC13. Для настройки порта нужно нажать на номер порта и в параметрах выбрать нужную опцию. PB10 и PC3 настраиваем как GPIO_Output, а PC3 настраиваем как GPIO_Input. На рис. 4 видно, как настроить порт на выход. Аналогично производим эту настройку для всех оставшихся портов.

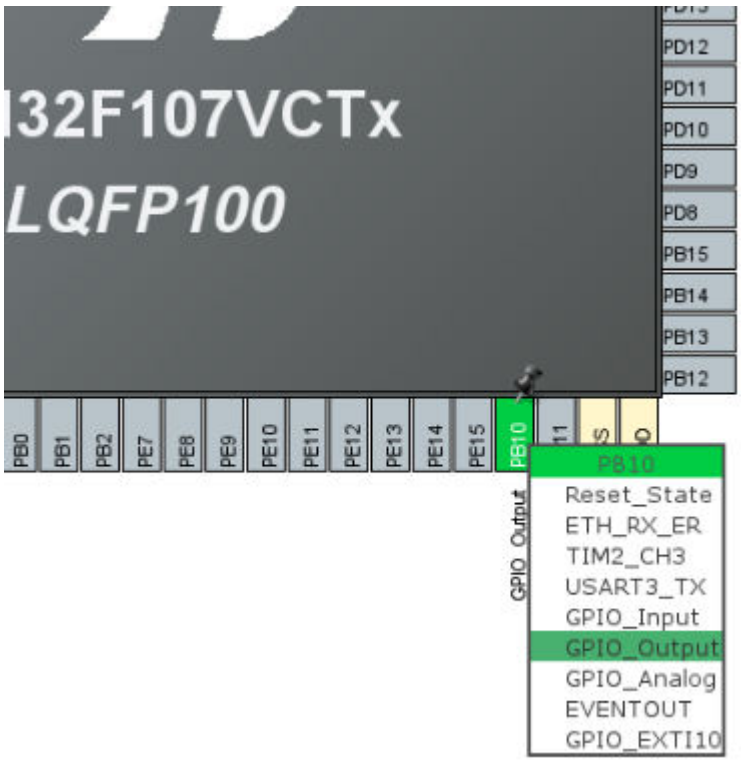

Рис. 4. Настройка порта PB10 на выход

По окончании конфигурации проекта нажимаем «Ctrl + S», программа сгенерирует код и можно приступать к программированию. Выбрав в боковом меню файл main.c, требуется нажать "Build". Если всё настроено правильно, то компиляция будет выполнена без ошибок, а в консоли будет написано "Build finished" (рис. 8).

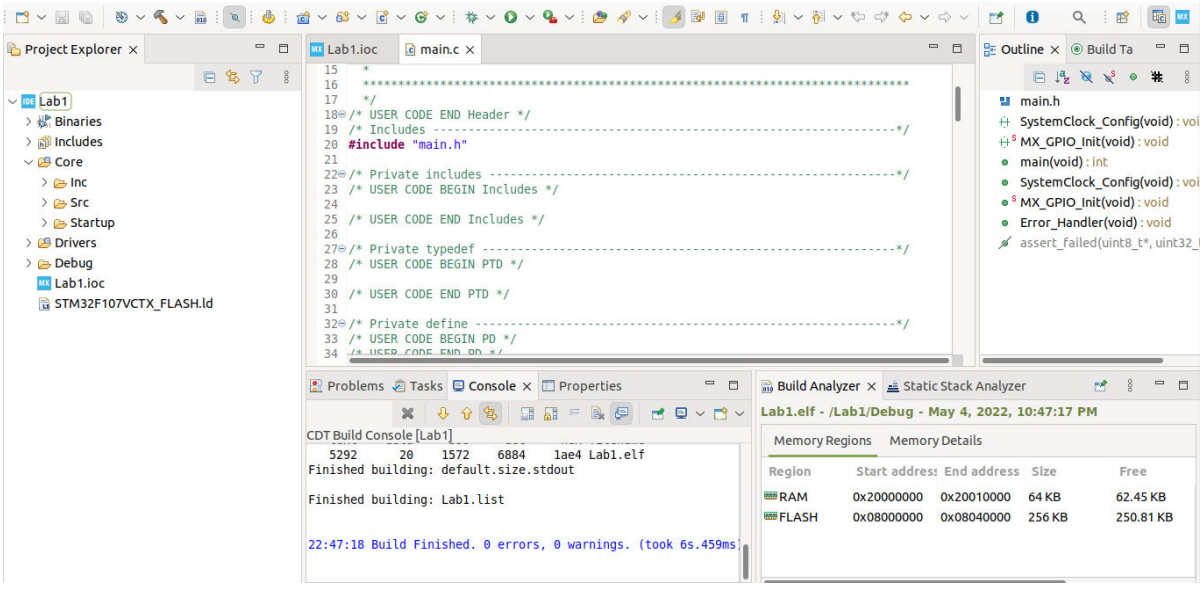

Рис. 8. Компиляция проекта

Теперь можно приступать к разработке своей первой программы. В файле main.c ищем:

```
int main(){
while (1)
  {
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
//
```

При конфигурировании проекта через Cube MX, создаются файлы с таким содержимым. Свой код можно писать только в:

```
 /* USER CODE BEGIN XX */
<программный код>
  /* USER CODE END XX  */
```

в противном случае при повторной генерации кода через CodeMX, ваш код, который написан за пределами USER CODE, будет утерян. Если всё сделано правильно, то код сохранится и никуда не исчезнет.

Следовательно приступаем к написанию кода. В бесконечный цикл while добавляем код для мигания светодиода, встроенного в плату UDK-32F107V.

```
while (1)
  {
HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_10); // Инвертирование состояния выхода PB10 (интегрированный в плату светодиод)
HAL_Delay(1000);  //задержка включения/выключения ветодиода
  }
 /* USER CODE END WHILE */
 /* USER CODE BEGIN 3 */
 /* USER CODE END 3 */
```

После добавления кода, нужно нажать "Build" и если код успешно скомпилируется, загрузить его в плату нажатием "Run>Debug". После прошивки код нужно запустить, нажав "Resume", либо остановить отладку, нажав "Stop". Результатом выполненной прошивки будет мигающий красный светодиод на плате рядом с синим светодиодом питания (рис. 9).

Рис. 9. Светодиод, подключённый к порту PB10

Теперь усложним задачу, отключив встроенный в плату светодиоди подключив два внешних: зелёный - PC3 и красный - PA3. Светодиоды будут мигать по очереди, а скорость переключения будет регулироваться удержанием кнопок Tamper - PC13 и Wake UP – PA0. Кнопки подключены по разной схеме (Рис. 10). Wake Up при нажатии подтягивается к +3.3V, а Tamper подтягивается к земле (GND), следовательно код у них будет немного отличаться.

```
if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == 0){ //если нажата WakeUP
}
if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == 1){  //если нажата Tamper
}
```

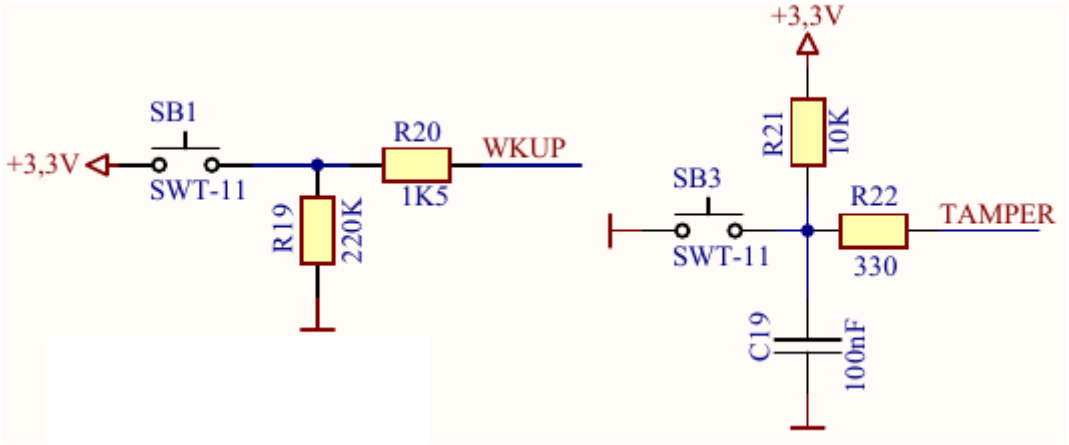

Рис. 10. Схема подключения кнопок

```
/* USER CODE BEGIN Includes */
#define constrain(amt,low,high) ((amt)<(low)?(low):((amt)>(high)?(high):(amt)))
//создаём макрос для задания пределов значений
/* USER CODE END Includes */

/* USER CODE BEGIN PV */
int led_delay=500; //переменная для хранения задержки
/* USER CODE END PV */

/* USER CODE BEGIN WHILE */
while (1)
{
if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == 0){ //если нажата WakeUP, увеличиваем задержку
led_delay -= 50; //декремент
}

if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == 1){ //если нажата Tamper, уменьшаем задержку
led_delay += 50; //инкремент
}
```

HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_3); // Инвертирование состояния выхода.

HAL_Delay(led_delay); //задержка

HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_3); // Инвертирование состояния выхода.

HAL_Delay(led_delay);

led_delay = constrain(led_delay, 50, 2000); //задаём пределы значений задержки 50..2000мс

}

/* USER CODE END WHILE */

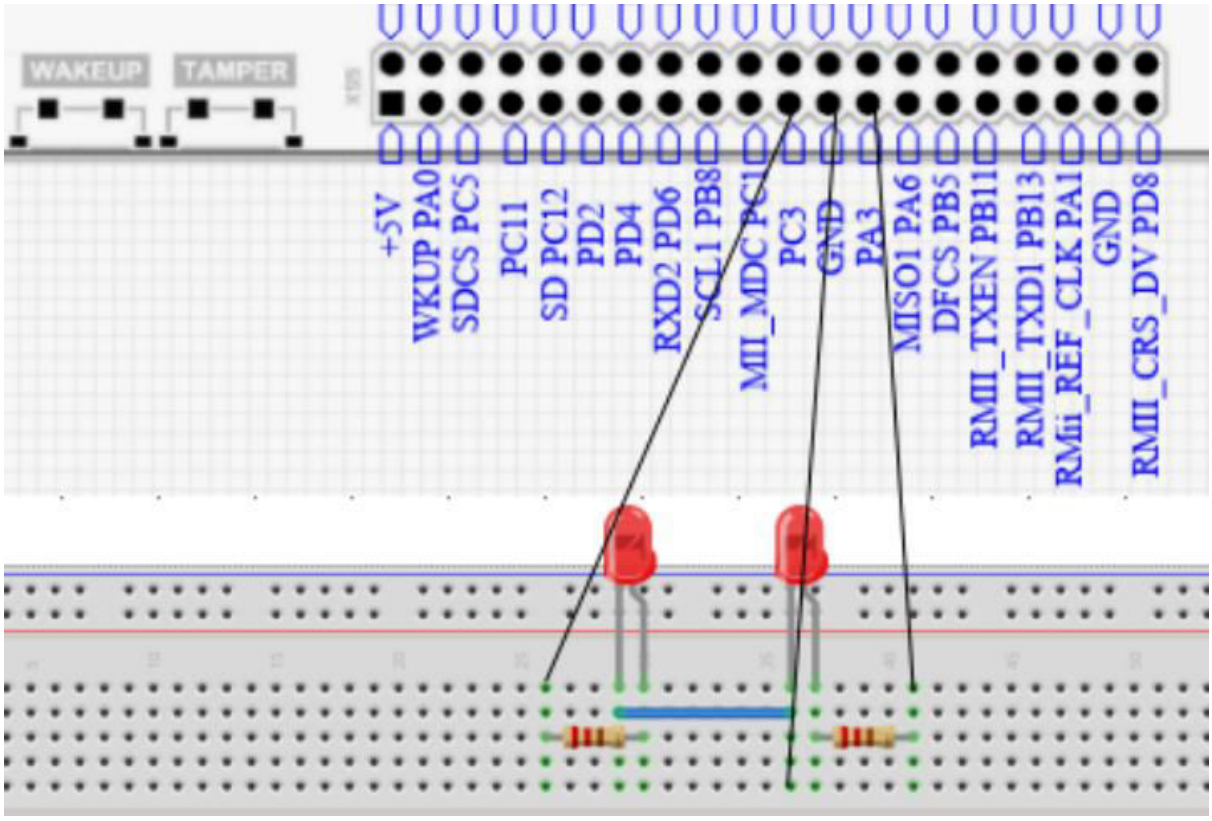

Рис. 11. Собранная схема

Ход работы

1. На основе кода примера приложения создать свой проект в среде разработки и проверить его работоспособность.
2. Ознакомиться с работой используемых функций, просмотрев исходный код, а также комментарии в исходных файлах библиотеки и в режиме отладки.
3. Реализовать необходимую функциональность.
4. Запрограммировать плату и продемонстрировать работу программы.