

Работа с последовательным интерфейсом SPI в программной среде Proteus 8.11

Часть 1

Татьяна Колесникова (beluikluk@gmail.com)

В статье рассматривается проектирование схем микроэлектронных устройств с использованием интерфейса SPI в Proteus на примере его реализации в микроконтроллерах AVR (семейства Mega) и STM32 (семейства Cortex-M3). Описаны особенности написания программного кода для инициализации интерфейса и работы с ним, а также моделирования схем, в которых проводится передача данных через SPI между двумя и тремя устройствами, сконфигурированными как master и slave. Выполнено отображение принятых ведомым устройством данных на экране виртуального терминала. С помощью осциллографа проведён контроль входных/выходных сигналов, присутствующих на выводах устройств схемы.

Введение

Интерфейс SPI (Serial Peripheral Interface – последовательный периферийный интерфейс) является высокоскоростным синхронным последовательным интерфейсом и реализован во всех микроконтроллерах STM32 Cortex-M3 и AVR семейства Mega. Он обеспечивает обмен данными между микроконтроллером и различными периферийными устройствами, такими как АЦП, ЦАП, цифровые потенциометры, FLASH-ПЗУ, другие микросхемы и микроконтроллеры. Каждый модуль SPI может работать в ведущем или подчинённом режиме, что позволяет ему связаться с любой другой интегральной схемой, оснащённой интерфейсом SPI. Ведущий микро-

контроллер можно связать с одним или несколькими ведомыми устройствами. Схема подключения устройств по интерфейсу SPI показана на рис. 1. Связь между устройствами осуществляется с помощью следующих цифровых сигналов:

- MOSI – выход данных для ведущего или вход данных для ведомого устройства;
- MISO – вход данных для ведущего или выход данных для ведомого устройства;
- SCK – сигнал общей синхронизации интерфейса;
- $\overline{SS}$ – выбор ведомого устройства.

Ведущее устройство формирует один или несколько сигналов SS (Slave Select) для выбора ведомых устройств. При этом количество формируемых сигналов соответствует количеству ведомых устройств. Ведомое устройство получит данные только в том случае, если оно было выбрано ведущим.

Передача данных осуществляется посредством линий MOSI и MISO. Процессом передачи данных управляет ведущее устройство (Master), формируя тактовые импульсы через линию SCK. Вывод SCK ведущего микроконтроллера является выходом тактового сигнала, а ведомого микроконтроллера – входом. Одновременно с передачей данных от ведущего к ведомому устройству происходит приём данных ведущим устройством от ведомого по кольцу. Таким образом, за один полный цикл сдвига всех разрядов регистра происходит обмен данными между двумя устройствами. Ведомые устройства не могут обмениваться данными друг с другом.

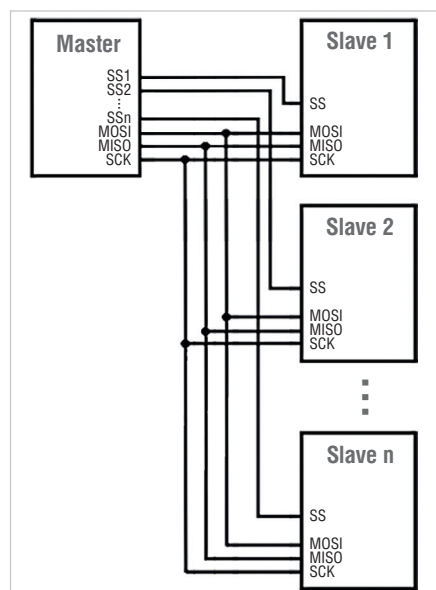


Рис. 1. Схема подключения устройств по интерфейсу SPI

Рассмотрим работу с SPI на примере микроконтроллеров AVR (микросхема ATmega16) и STM32 (микросхема STM32F103C4 [1]), для чего воспользуемся программой компьютерного моделирования электронных схем Proteus [2].

Передача данных через последовательный интерфейс SPI в микроконтроллерах Cortex-M3 в Proteus

Для организации быстродействующей связи с интегральными схемами у микроконтроллеров STM32 Cortex-M3 имеется до трёх модулей SPI [3], предназначенных для полнодуплексной передачи данных на частоте до 18 МГц. Важно обратить внимание, что один модуль SPI подключён к высокоскоростной шине APB2, два других модуля связаны с более низкоскоростной шиной APB1.

Выводы микроконтроллера, используемые модулями SPI, являются линиями ввода/вывода общего назначения. Назначение выводам функции SPI осуществляется в регистрах настройки GPIO. В микроконтроллере STM32F103C4 модулем SPI1 используются линии PA4 – NSS, PA5 – SCK, PA6 – MISO, PA7 – MOSI. Однако, используя регистры AFIO, линии SPI1 можно перенести на выводы PA15, PB3, PB4 и PB5 соответственно.

В модуле SPI имеются следующие регистры ввода/вывода, необходимые для работы интерфейса:

- SPI_DR – регистр данных, содержит посылаемые или принятые 8 или 16 бит данных;
- SPI_CR1, SPI_CR2 – регистры управления, определяют функционирование модуля SPI;
- SPI_SR – регистр состояния, отображает состояние модуля SPI.

Для обнаружения ошибок, возникших при приёме и передаче данных, используются регистры SPI_CRCPR, SPI_RXCRCR и SPI_TXCRCR.

Включение/выключение SPI выполняется установкой шестого бита (SPE) регистра SPI_CR1, седьмой бит (LSBFIRST) задаёт порядок передачи данных, а второй бит (MSTR) этого регистра задаёт выбор режима рабо-

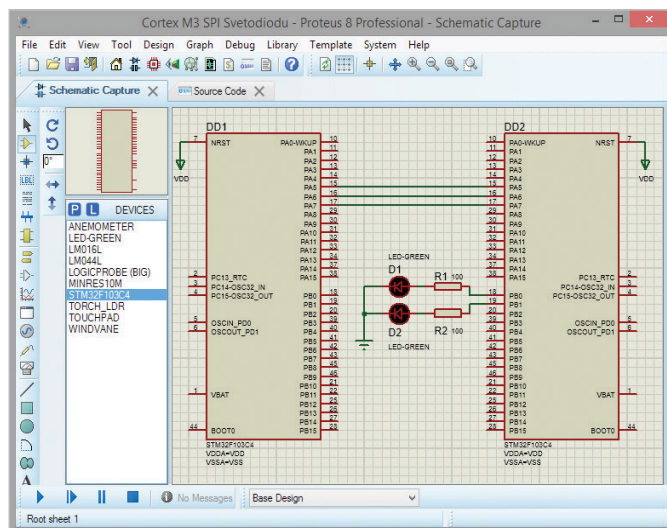


Рис. 2. Демонстрационная схема с использованием двух микроконтроллеров STM32F103C4 и светодиодов

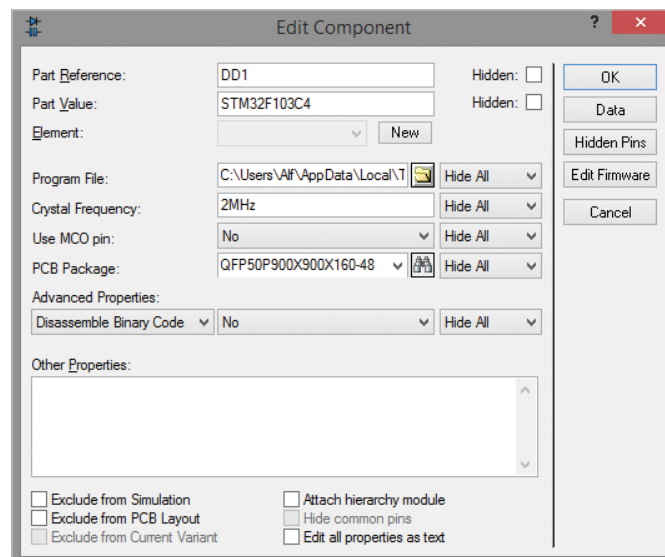


Рис. 3. Настройка частоты работы микроконтроллера DD1

ты интерфейса. При чтении регистра SPI_DR выполняется обращение к буферному регистру приёмника, при записи – к буферному регистру передатчика. Для передачи данных их необходимо записать в регистр передатчика. Принятые данные читаются из регистра приёмника. Для программы существует один регистр с именем SPI_DR. Скорость обмена по SPI определяет блок генератора скорости, который задаёт частоту следования тактовых импульсов. Для этого предназначены разряды BR0, BR1 и BR2 регистра SPI_CR1. Три разряда предполагают наличие восьми значений скорости.

При написании программы инициализации микроконтроллера для передачи данных через интерфейс SPI их записывают в регистр SPI_DR с помощью команды `SPy->DR = data_tx`, где `y` – это номер интерфейса SPI. Окончание передачи контролируется проверкой флага TXE регистра SPI_SR, для чего можно использовать команду `while (!(SPy->SR & SPI_SR_TXE)) { }`. Одновременно с передачей происходит приём данных в регистр SPI_DR. Принятые данные считываются из регистра данных с помощью команды `data_rx = SPy->DR`. Для проверки работоспособности интерфейса SPI в режиме мастера достаточно соединить выводы MISO и MOSI между собой и сравнить переданные данные с полученными. Если они совпадают, это значит, что интерфейс работает правильно.

Управляя значениями битов регистров модуля SPI микроконтроллеров Cortex-M3, реализуют процесс обмена данными. Регистр данных SPI_DR состоит из 16 разрядов данных. В этот

регистр данные записываются для передачи и читаются из него при приёме. В одноранговой шине SPI (где имеется только одно ведущее и одно ведомое устройство) сигнал SS может быть опущен, а соответствующий вывод ведомого устройства подключён к земле.

Передача данных через интерфейс SPI между двумя микроконтроллерами Cortex-M3

Рассмотрим процесс передачи данных между двумя микроконтроллерами Cortex-M3 на примере микросхемы STM32F103C4, для чего создадим в Proteus новый схемный проект и добавим в рабочее поле на вкладке Schematic Capture две микросхемы STM32F103C4, два светодиода, два резистора (100 Ом), символ земли. Соединим компоненты так, как показано на рис. 2, и напомним на языке программирования C программный код управления передачей данных. В микроконтроллере STM32F103C4 имеется только один модуль SPI, к регистрам которого в Proteus в программе инициализации обращаются с указанием номера интерфейса (например, по имени SPI1_SR, SPI1_DR). Обращение без указания номера интерфейса (по имени SPI_SR, SPI_DR) при компиляции кода программы вызывает ошибку. Необходимо отметить, что программа инициализации пишется как для ведущего, так и для ведомого микроконтроллера. Определим микроконтроллер DD1 как ведущий, а микроконтроллер DD2 как ведомый. При этом задачей мастера будет послать управляющий сигнал (кодovou комбинацию), задачей ведомого

устройства – принять его и последовательно включить и выключить оба светодиода.

Перед передачей и приёмом данных необходимо сформировать сигналы выбора для того устройства, с которым будет производиться обмен. Если ведомое устройство одно, то можно использовать сигнал выбора NSS. Если же ведомых устройств несколько, то придётся для каждого из них формировать индивидуальный сигнал выбора. В качестве источников таких сигналов могут выступать свободные выводы портов GPIO.

Для удобства соединения можно отразить в рабочей области микросхему DD1, для чего выделим её левой кнопкой мыши, правой кнопкой мыши вызовем контекстное меню и выберем в нём пункт X-Mirror. В результате микросхема будет отражена по горизонтали в рабочем поле проекта. В таком положении выводы PA4 (NSS), PA5 (SCK), PA6 (MISO), PA7 (MOSI) обеих микросхем соединить намного проще, при этом соединительные линии на схеме будут короче. Для каждого микроконтроллера двойным щелчком левой кнопки мыши откроем окно настроек Edit Component и в поле Crystal Frequency установим частоту работы 2 МГц (см. рис. 3). Кнопкой Hidden Pins для каждого микроконтроллера откроем окно Edit Hidden Power Pins, где выполним согласование скрытых выводов питания и цепей питания (см. рис. 4). В нашем примере в полях Pin VDD и Pin VDDA введём значение VDD, а в полях Pin VSS и Pin VSSA – значение VSS. Нажмём кнопку OK для вступления в силу внесённых изменений.

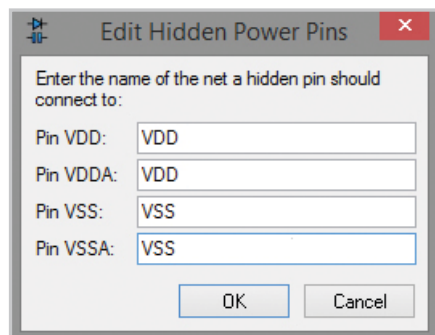


Рис. 4. Подключение цепей питания к скрытым выводам микроконтроллера в окне Edit Hidden Power Pins

Таблица 1. Задание частоты тактового сигнала SCK

BR5	BR4	BR3	Частота сигнала SCK
0	0	0	fPCLK/2
0	0	1	fPCLK/4
0	1	0	fPCLK/8
0	1	1	fPCLK/16
1	0	0	fPCLK/32
1	0	1	fPCLK/64
1	1	0	fPCLK/128
1	1	1	fPCLK/256

Примечание: fPCLK – это тактовая частота микроконтроллера.

Перед выполнением передачи данных необходимо, прежде всего, разрешить работу модуля SPI. Для этого следует установить в единицу шестой бит регистра SPI_CR1. Режим работы определяется состоянием второго бита этого регистра: если бит установлен в 1, микроконтроллер работает в режиме Master, если сброшен в 0 – в режиме Slave. Программно (на языке программирования C) эти действия можно реализовать следующим образом:

```
SPI1->CR1 = (1<<6)|(1<<2); // включение SPI1 в ведущем микроконтроллере
SPI1->CR1 = (1<<6)|(0<<2); // включение SPI1 в ведомом микроконтроллере.
```

Передача данных осуществляется следующим образом. При записи в регистр данных SPI ведущего микроконтроллера запускается генератор тактового сигнала модуля SPI. Данные начинают побитно выдаваться на вывод MOSI устройства Master и соответственно поступать на вывод MOSI устройства Slave. Порядок передачи битов данных определяется состоянием седьмого бита регистра LSBFIRST. Если бит сброшен в 0, первым передаётся младший бит данных, если же установлен в 1 – старший бит. Частота тактового сигнала SCK и соответственно скорость

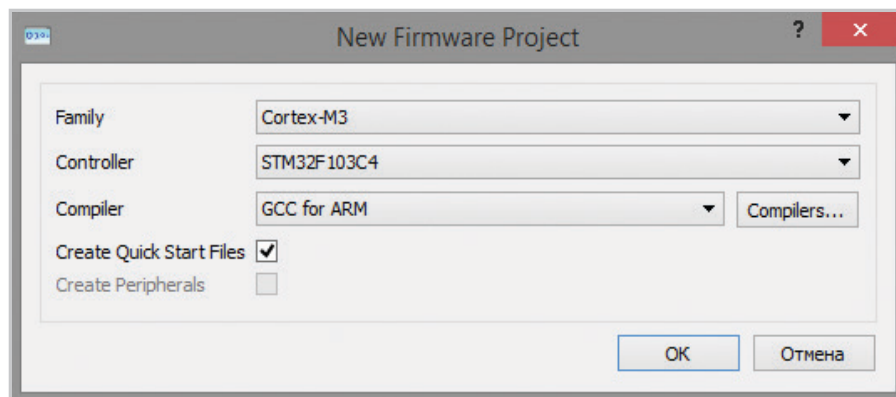


Рис. 5. Окно New Firmware Project

передачи данных по интерфейсу определяются состоянием пятого, четвертого и третьего битов регистра BR ведущего микроконтроллера (см. табл. 1), так как именно он является источником тактового сигнала. Для ведомого микроконтроллера состояние этих битов не имеет значения.

Напишем на языке программирования C следующий код программы инициализации для ведущего микроконтроллера:

```
#include <stm32f1xx.h> // подключение заголовочного файла
int main() { // начало программы
    RCC->APB2ENR |= RCC_APB2ENR_SPI1EN; // включаем тактирование SPI1
    // подсоединение линий порта PA к шине APB2
    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;

    // настройка линий PA5 (SCK), PA6 (MISO), PA7 (MOSI) порта PA
    // биты CNF5, CNF7 = 10, биты MODE5, MODE7 = 11
    // биты CNF6 = 10, биты MODE6 = 00
    GPIOA->CRL = 0xb8b33333;

    // конфигурация SPI1
    SPI1->CR1 = (0<<11) // формат кадра данных 8 бит
    | (0<<7) // направление передачи младшим разрядом вперед
    | (1<<9) // включаем программное управление сигналом NSS
    | (1<<8) // NSS в высоком состоянии
    | (1<<5)|(0<<4)|(0<<3) // скорость передачи данных: F_PCLK/32
    | (1<<2) // режим работы Master (ведущий)
    | (0<<1)|(0<<0) // полярность (0) и фаза тактового сигнала (0)
```

```
| (1<<6); // включаем SPI
// после установки в 1 флага TXE регистра SPI1_SR
while(!(SPI1->SR & SPI_SR_TXE))
{ }
// заполняем буфер передатчика
SPI1->DR = 0b11111110; }
```

Для ведомого микроконтроллера был написан следующий код программы инициализации:

```
#include <stm32f1xx.h> // подключение заголовочного файла
void delay (int dly) // подпрограмма формирования задержки
{ int i;
  for(; dly>0; dly--)
  for ( i=0; i<10000; i++); }

int main() { // начало программы
    RCC->APB2ENR |= RCC_APB2ENR_SPI1EN; // включаем тактирование SPI1
    // подсоединение линий порта PA к шине APB2
    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;
    // подсоединение линий порта PB к шине APB2
    RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;

    // настройка линий PA5 (SCK), PA6 (MISO), PA7 (MOSI) порта PA
    // биты CNF5, CNF7 = 10, биты MODE5, MODE7 = 00
    // биты CNF6 = 10, биты MODE6 = 11
    GPIOA->CRL = 0xb8b33333;
    // настройка линий порта PB
    // биты CNF = 10, биты MODE = 00
    GPIOB->CRL = 0x88888888;

    // конфигурация SPI1
    SPI1->CR1 = (1<<6) | (0<<2); // включаем SPI, режим работы Slave (ведомый)
```

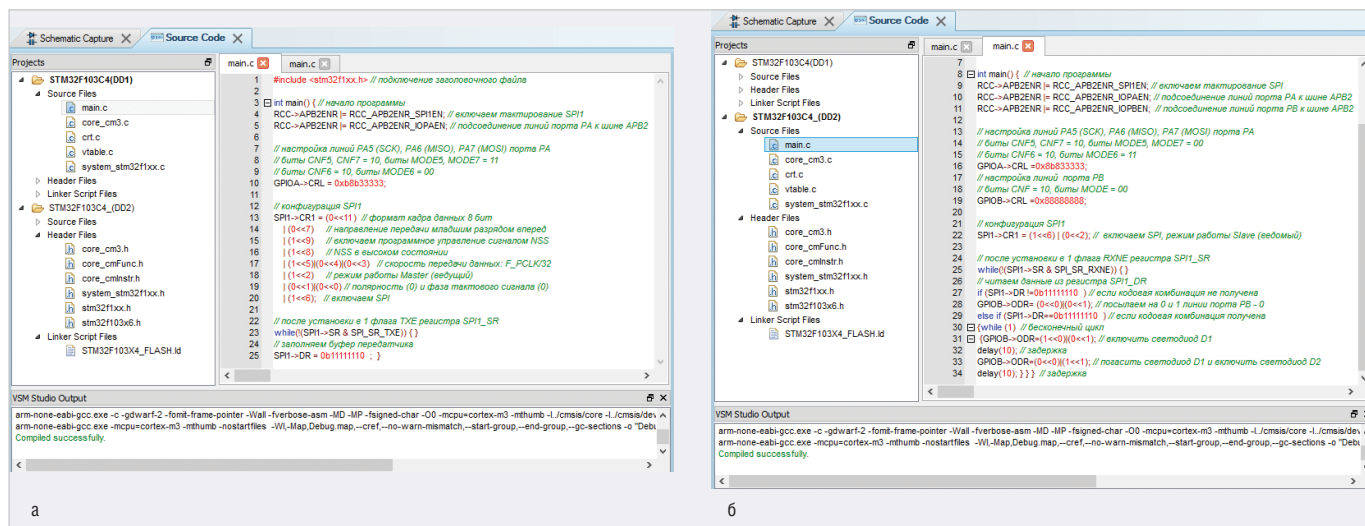


Рис. 6. Вкладка Source Code, код программы инициализации: ведущего микроконтроллера (а) и ведомого микроконтроллера (б)

```
// после установки в 1 флага RXNE
регистра SPI1_SR
while(!(SPI1->SR & SPI_SR_RXNE))
{
}

// читаем данные из регистра SPI1_DR
if (SPI1->DR !=0b11111110) //
если кодовая комбинация не получена
GPIOB->ODR= (0<<0)|(0<<1); //
посылаем лог. 0 на линии PB0 и PB1
порта PB
else if (SPI1->DR==0b11111110) //
если кодовая комбинация получена
{while (1) // бесконечный цикл
{GPIOB->ODR=(1<<0)|(0<<1); //
включить светодиод D1
delay(10); // задержка
GPIOB->ODR=(0<<0)|(1<<1); //
погасить светодиод D1 и включить
светодиод D2
delay(10); } } } // задержка
```

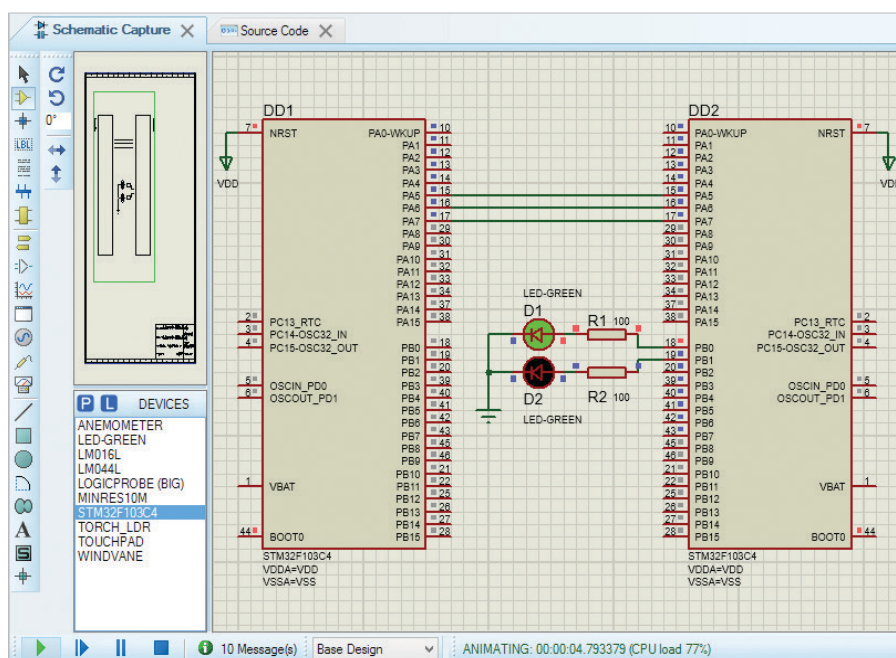


Рис. 7. Моделирование передачи данных между двумя микроконтроллерами STM32F103C4 через интерфейс SPI в программной среде Proteus

В Proteus программа инициализации микроконтроллера вводится на вкладке Source Code. Для её открытия выделяют левой кнопкой мыши символ ведущего микроконтроллера в рабочем поле схемного проекта, правой кнопкой мыши вызывают контекстное меню и выбирают в нём пункт Edit Source Code. В результате откроется окно New Firmware Project (см. рис. 5), в котором устанавливают следующие параметры:

- Family – семейство микроконтроллера (Cortex-M3);
- Controller – модель микроконтроллера (STM32F103C4);
- Compiler – компилятор (GCC for ARM);
- Create Quick Start Files – автоматическое создание заготовки программного кода для микроконтроллера (установим флажок в поле).

Когда все значения установлены, нажмём на кнопку OK, в результате в проект будет добавлена вкладка Source Code, на которой и необходимо ввести код программы управления ведущим микроконтроллером (см. рис. 6а). Перейти на вкладку для написания программы инициализации ведомого микроконтроллера (см. рис. 6б) можно таким же образом, как и для ведущего, однако в этом случае на вкладку будет добавлена отдельная закладка. После того как в рабочей области проекта собрана схема, а на вкладке Source Code введён код программы для всех микроконтроллеров проекта, кнопкой Run the simulation (кнопка находится в левом нижнем углу окна программы) можно запустить модели-

рование (см. рис. 7). Для компиляции кода программы, написанного на языке программирования C, для микроконтроллеров STM32 Cortex-M3 в Proteus применяется компилятор GCC for ARM. Проанализируем работу демонстрационной схемы, представленной на рис. 7. На вкладке Source Code программным путём были даны указания ведущему микроконтроллеру через интерфейс SPI1 отправить ведомому микроконтроллеру кодовую комбинацию. Программа ведомого микроконтроллера находится в ожидании установки в 1 флага RXNE регистра SPI1_SR. Как только по интерфейсу SPI1 получена кодовая комбинация от ведущего микроконтроллера, запускается подпрограмма, которая даёт указания

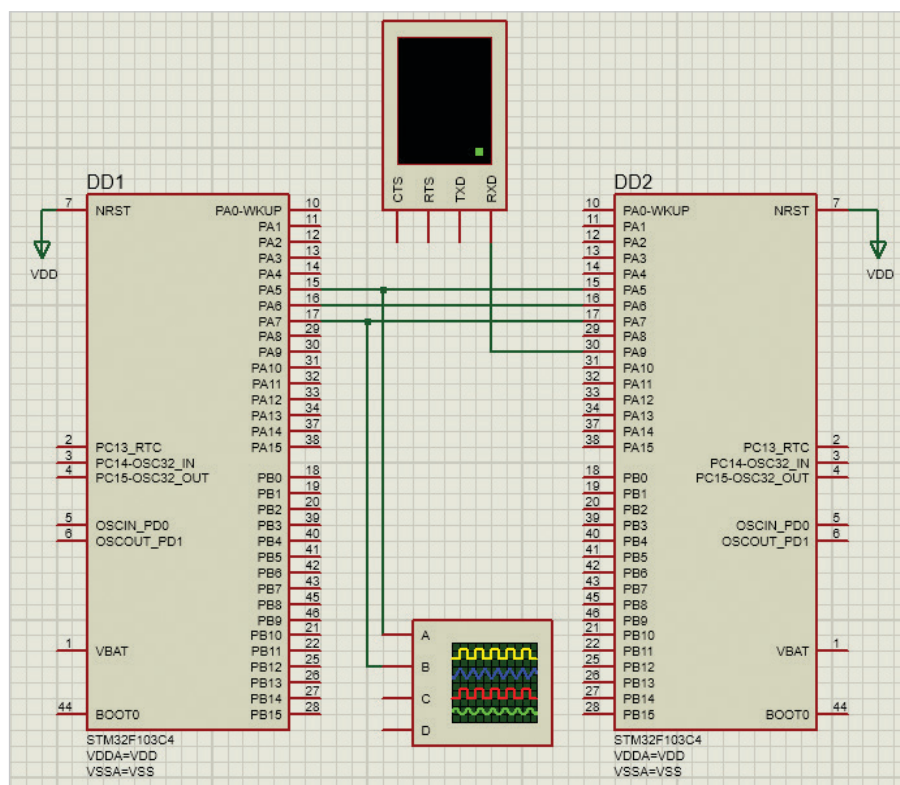


Рис. 8. Демонстрационная схема с использованием двух микроконтроллеров STM32F103C4, виртуального терминала и осциллографа

ведомому микроконтроллеру вывести на линии PB0 и PB1 порта PB значения логической 1 и 0 соответственно, которые удерживаются на этих линиях при помощи команды задержки. Затем на линии PB0 и PB1 выводятся значения логического 0 и 1 соответственно, после чего по истечении времени задержки выполнение этого фрагмента программы повторяется. После запуска моделирования при помощи двух светодиодов, подключённых к линиям PB0 и PB1, мы можем проверить правильность работы программы – светодиоды подсвечиваются и гаснут поочередно.

Передача инициируется записью передаваемых данных (в 8- или 16-битном формате) в буферный регистр передатчика, т.е. в регистр данных SPI_DR. После чего автоматически сбрасывается первый бит регистра SPI_SR (TXE = 0), что говорит о том, что буфер передатчика уже не пуст. При этом устанавливается в 1 и седьмой бит регистра SPI_SR (BSY = 1), что означает, что интерфейс занят. После этого данные пересылаются из регистра SPI_DR в сдвиговый регистр передатчика. Передача данных осуществляется посредством линий MOSI и MISO.

Сдвиговые регистры ведущего и ведомого устройства объединяются линиями связи в единый сдвиговый регистр.

Процессом передачи данных управляет ведущее устройство, формируя тактовые импульсы через линию SCK. Одновременно с передачей данных от ведущего к ведомому устройству происходит приём данных ведущим устройством от ведомого по кольцу. Таким образом, за один полный цикл сдвига всех разрядов регистра происходит обмен данными между двумя устройствами. При написании программного кода необходимо указывать номер модуля, к которому мы обращаемся. В нашем примере это SPI1.

Рассмотрим ещё один пример, где ведущий микроконтроллер пересылает через интерфейс SPI1 комбинацию символов английского алфавита ведомому микроконтроллеру, который выводит принятые данные через интерфейс USART [4] на экран виртуального терминала. Для чего создадим в Proteus новый схемный проект, добавим в рабочее поле на вкладке Schematic Capture две микросхемы STM32F103C4 и соединим их так, как показано на рис. 8. Щёлкнув левой кнопкой мыши на панели INSTRUMENTS (см. рис. 9) строку с названием VIRTUAL TERMINAL, а затем строку OSCILLOSCOPE, разместим мышью в рабочем поле проекта виртуальный терминал и виртуальный осциллограф, которым воспользуемся для просмотра осциллограммы рабо-

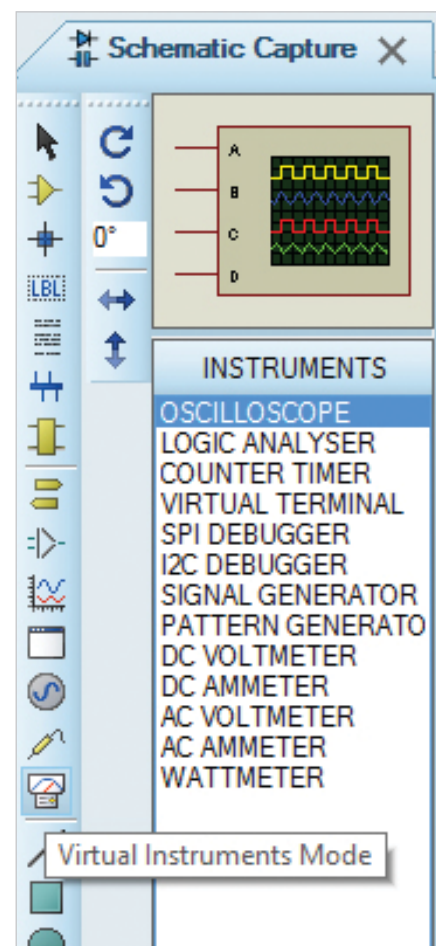


Рис. 9. Открытие панели INSTRUMENTS с помощью пиктограммы Virtual Instruments Mode

ты SPI. Подсоединим вывод PA9 (TXD) микроконтроллера DD2 к выводу RXD виртуального терминала, а выводы PA5 (SCK) и PA7 (MOSI) – к каналам A и B осциллографа.

В окне настроек Edit Component в поле Crystal Frequency для каждого микроконтроллера установим частоту его работы 2 МГц. Кнопкой Hidden Pins откроем окно Edit Hidden Power Pins, где выполним согласование скрытых выводов питания и цепей питания. В окне настроек терминала (см. рис. 10) определим значения следующих параметров:

- Baud Rate – скорость обмена данными (9600 бод);
- Data Bits – формат пакета данных (8 бит);
- Parity – контроль чётности (отсутствует – NONE);
- Stop Bits – количество стоповых битов (1).

Окна настроек открывают двойным щелчком левой кнопки мыши по размещённому на схеме компоненту.

Напишем на языке программирования C для ведущего (DD1) и для ведомо-

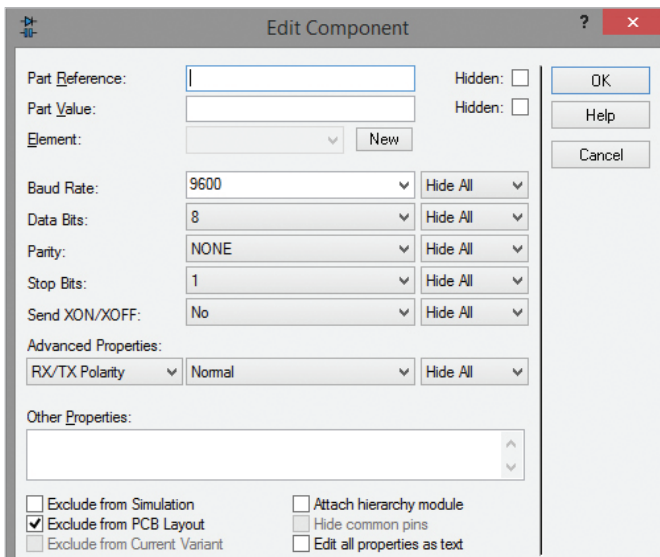


Рис. 10. Окно настроек виртуального терминала

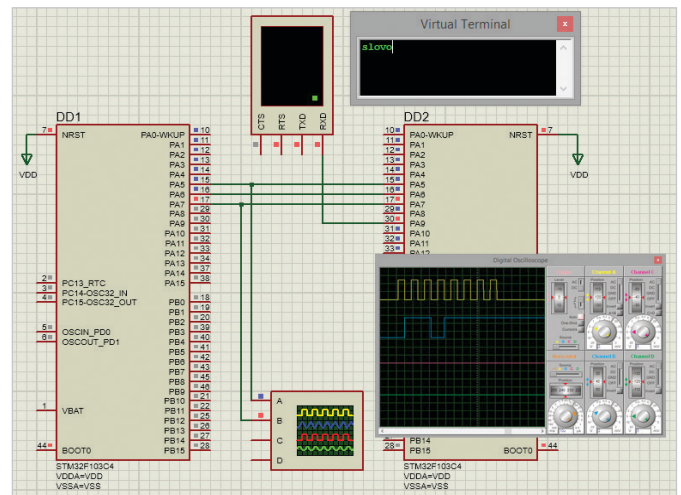


Рис. 11. Приём ведомым микроконтроллером через интерфейс SPI данных и их вывод на экран виртуального терминала через интерфейс USART

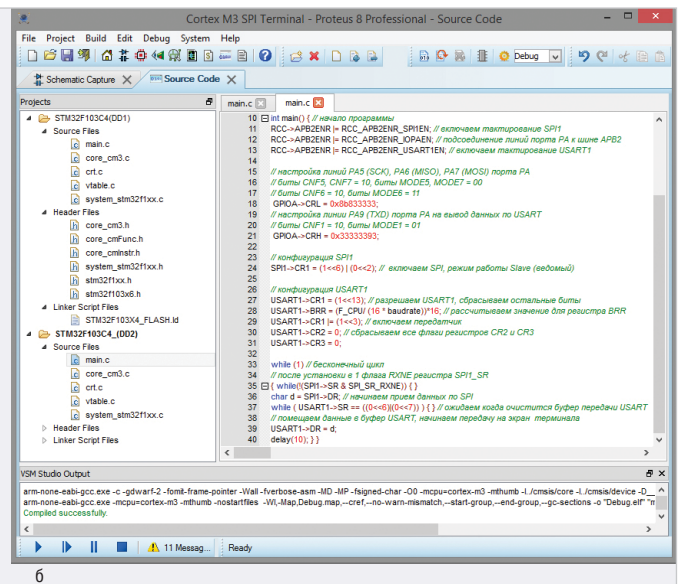
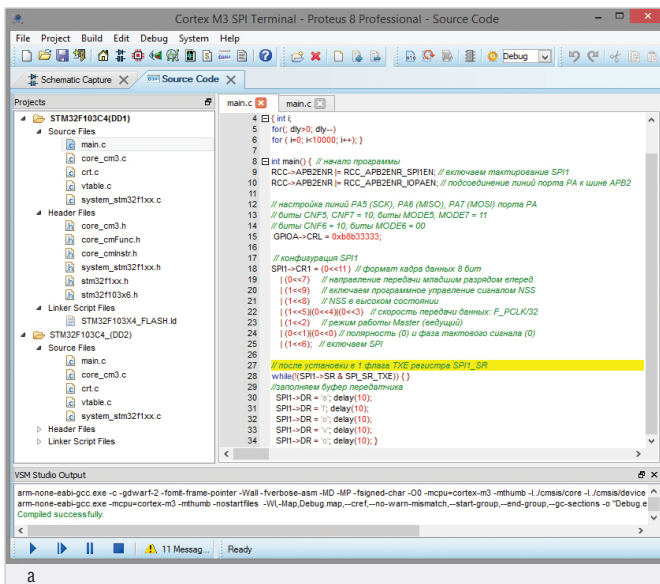


Рис. 12. Передача данных между двумя устройствами через интерфейс SPI. Вкладка Source Code, код программы инициализации: ведущего (а) и ведомого (б) микроконтроллера

мого (DD2) микроконтроллеров программный код управления передачей данных. Для получения осциллограммы работы интерфейса SPI настроим параметры осциллографа так, как показано на рис. 11.

Код программы инициализации для ведущего микроконтроллера (см. рис. 12а):

```
#include <stm32f1xx.h> // подключение заголовочного файла

void delay (int dly) // подпрограмма формирования задержки
{ int i;
  for(; dly>0; dly--)
  for ( i=0; i<10000; i++); }

int main() { // начало программы
```

```
RCC->APB2ENR |= RCC_APB2ENR_SPI1EN; // включаем тактирование SPI1
// подсоединение линий порта PA к шине APB2
RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;

// настройка линий PA5 (SCK), PA6 (MISO), PA7 (MOSI) порта PA
// биты CNF5, CNF7 = 10, биты MODE5, MODE7 = 11
// биты CNF6 = 10, биты MODE6 = 00
GPIOA->CRL = 0xb8b33333;

// конфигурация SPI1
SPI1->CR1 = (0<<11) // формат кадра данных 8 бит
| (0<<7) // направление передачи младшим разрядом вперед
```

```
| (1<<9) // включаем программное управление сигналом NSS
| (1<<8) // NSS в высоком состоянии
| (1<<5)|(0<<4)|(0<<3) // скорость передачи данных: F_PCLK/32
| (1<<2) // режим работы Master (ведущий)
| (0<<1)|(0<<0) // полярность (0) и фаза тактового сигнала (0)
| (1<<6); // включаем SPI

// после установки в 1 флага TXE регистра SPI1_SR
while(!(SPI1->SR & SPI_SR_TXE))
{ }

//заполняем буфер передатчика
SPI1->DR = 's'; delay(10);
SPI1->DR = 'l'; delay(10);
SPI1->DR = 'o'; delay(10);
```

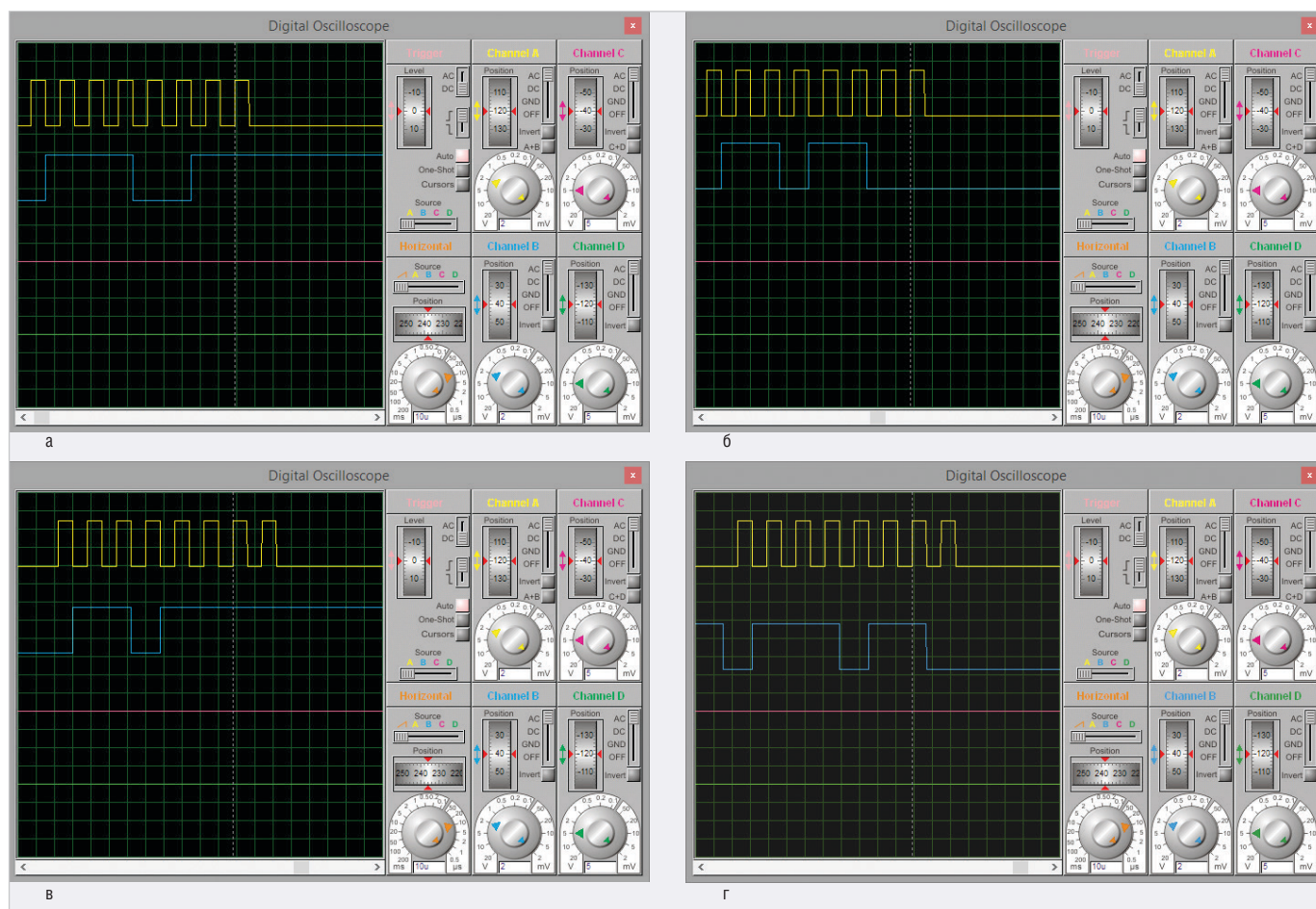


Рис. 13. Осциллограммы передачи данных через интерфейс SPI. Передача символов: «s» (а), «f» (б), «o» (в) и «v» (г), двоичные коды которых (01110011, 01101100, 01101111 и 01110110 соответственно) на осциллограмме отображены голубым цветом

```
SPI1->DR = 'v'; delay(10);
SPI1->DR = 'o'; delay(10); }
```

Код программы инициализации для ведомого микроконтроллера (см. рис. 126):

```
#include <stm32f1xx.h> // подключение заголовочного файла
#define F_CPU 2000000 // рабочая частота контроллера
#define baudrate 9600L // скорость обмена данными

void delay (int dly) // подпрограмма формирования задержки
{ int i;
  for(; dly>0; dly--)
  for ( i=0; i<10000; i++); }

int main() { // начало программы
  RCC->APB2ENR |= RCC_APB2ENR_SPI1EN; // включаем тактирование SPI1
  // подсоединение линий порта PA к шине APB2
  RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;
  RCC->APB2ENR |= RCC_APB2ENR_USART1EN; // включаем тактирование USART1
```

```
// настройка линий PA5 (SCK), PA6 (MISO), PA7 (MOSI) порта PA
// биты CNF5, CNF7 = 10, биты MODE5, MODE7 = 00
// биты CNF6 = 10, биты MODE6 = 11
GPIOA->CRL = 0x8b833333;
// настройка линии PA9 (TXD) порта PA на вывод данных по USART
// биты CNF1 = 10, биты MODE1 = 01
GPIOA->CRH = 0x33333393;

// конфигурация SPI1
SPI1->CR1 = (1<<6) | (0<<2); // включаем SPI, режим работы Slave (ведомый)

// конфигурация USART1
USART1->CR1 = (1<<13); // разрешаем USART1, сбрасываем остальные биты
USART1->BRR = (F_CPU/ (16 * baudrate))*16; // рассчитываем значение для регистра BRR
USART1->CR1 |= (1<<3); // включаем передатчик
USART1->CR2 = 0; // сбрасываем все флаги регистров CR2 и CR3
USART1->CR3 = 0;
while (1) // бесконечный цикл
```

```
// после установки в 1 флага RXNE регистра SPI1_SR
{ while(!(SPI1->SR & SPI_SR_RXNE)) { }
  char d = SPI1->DR; // начинаем приём данных по SPI
  // ожидаем когда очистится буфер передачи USART
  while ( USART1->SR == ((0<<6)|(0<<7)) ) { }
  // помещаем данные в буфер USART, начинаем передачу на экран терминала
  USART1->DR = d;
  delay(10); } }
```

После того как в рабочей области проекта собрана схема, а на вкладке Source Code введён код программы, можно запускать моделирование, для чего предусмотрена кнопка Run the simulation в левом нижнем углу окна программы. Как видно на рис. 12, компиляция закончена успешно – в коде программы отсутствуют ошибки. Разработанный проект (см. рис. 11) функционирует верно – на экран виртуального терминала была выведена указанная в коде программы комбинация символов. Осциллограм-

мы передачи данных через интерфейс SPI между микроконтроллерами DD1 и DD2 показаны на рис. 13.

Таким образом, чтобы передать данные через SPI между двумя микроконтроллерами STM32F103C4 в ведущем микроконтроллере, необходимо:

- включить тактирование выбранного модуля SPIx (где x – номер модуля) и порта ввода/вывода, через который будет вестись передача данных;
- настроить режим работы линий синхронизации и передачи данных на вывод данных с альтернативной функцией, записав в соответствующие разряды регистров конфигурации линий GPIO нужную комбинацию бит;
- разрешить работу с выбранным модулем SPIx;
- управляя значениями битов регистра SPIx_CR1, перевести интерфейс SPIx в режим Master (флаг MSTR) и задать скорость передачи (флаг BR [2:0]), размер кадра данных (флаг DFF), направление передачи (флаг LSBFIRST), источник сигнала NSS (флаг SSM), полярность (флаг CPOL) и фазу (флаг CPHA) тактового сигнала;

- после установки в 1 флага TXE регистра SPIx_SR записать данные в регистр SPIx_DR.

Для настройки интерфейса SPI в режим ведомого устройства выполняются следующие действия:

- включают тактирование выбранного модуля SPIx (где x – номер модуля) и порта ввода/вывода, через который будет вестись приём данных;
- настраивают режим работы линий синхронизации и приёма данных на ввод данных, записав в соответствующие разряды регистров конфигурации линий GPIO нужную комбинацию бит;
- разрешают работу с выбранным модулем SPIx;
- переводят интерфейс SPIx в режим Slave;
- после установки в 1 флага RXNE регистра SPIx_SR читают данные из регистра SPIx_DR.

Важно обратить внимание, что шина APB2 может работать с максимальным быстродействием 72 МГц, а быстродействие шины APB1 ограничено частотой 36 МГц. По умолчанию тактирование отключено, и перед началом работы с любым

периферийным устройством необходимо разрешить подачу на него тактового сигнала, что выполняется разработчиком программно в регистрах RCC_APB2ENR, RCC_APB1ENR. Шина APB2 обслуживает контроллеры последовательных интерфейсов USART1, SPI1 и порты ввода/вывода общего назначения (GPIO), шина APB1 обслуживает контроллеры интерфейсов USART2, USART3, SPI2, SPI3.

Литература

1. STM32F103x4, STM32F103x6 MCU Datasheet. STMicroelectronics. 2009.
2. Proteus VSM Help. Labcenter Electronics. 2020.
3. STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced ARM-based 32-bit MCUs. Reference manual. STMicroelectronics. 2010.
4. Колесникова Т. Работа с универсальным синхронно/асинхронным приёмо-передатчиком USART в программной среде Proteus 8.11 // Современная электроника. 2021. № 8.
5. 8-bit AVR Microcontroller with 16K Bytes In-System Programmable Flash. ATmega16, ATmega16L. Atmel Corporation. 2010.



IF/RF & Microwave Design

advantex UNO-2XM

100 кГц – 21 ГГц

Малошумящий синтезатор частот

Уровень фазового шума: -140 дБн/Гц
при отстройке 10 кГц @1 ГГц

Время перестройки: < 60 мкс

Шаг перестройки: 0.0001 Гц

Фазовый шум, нормированный к 1 ГГц

Phase Noise, dBc/Hz

- 1 GHz
- 2 GHz
- 3 GHz
- 5 GHz
- 7 GHz
- 10 GHz
- 13 GHz
- 15 GHz
- 17 GHz
- 20 GHz
- Spec Mask

Offset, Hz

Мощность выходного сигнала: -5...+15 дБм, шаг 0,5 дБ

Опорный сигнал: 1-250 МГц

Выход опорной частоты: 10/100 МГц

Рабочий диапазон: -40...+60 °C

Габаритные размеры: 30,5 × 87,5 × 195,0 мм

Сделано в России
www.advantex.ru