**Lesson 23**

**HAL. SPI. Shift register 74HC595**

[*Previous lesson*](http://narodstream.ru/stm-urok-22-hal-i2c-i2c-to-lcd2004/) [*Programming STM32 MK*](http://narodstream.ru/programmirovanie-mk-stm32/) [*Next lesson*](http://narodstream.ru/stm-urok-24-hal-spi-led-staticheskaya-indikaciya/)

Today we begin the cycle of training on the **SPI (Serial peripheral interface)** bus .

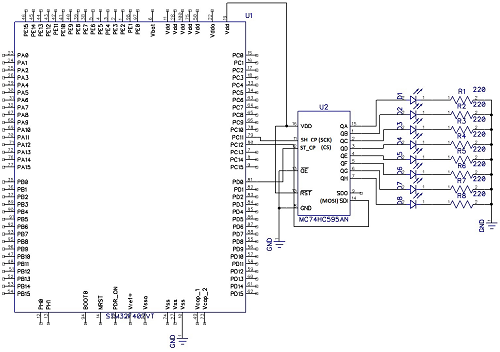
Talk about how this bus works, there is no point, because on this site all information about this already exists, it is enough to look at the [**lesson on this tire**](http://narodstream.ru/avr-urok-24-znakomstvo-s-shinoj-spi/) for AVR.

It is important for us now to learn how to program this bus in **STM32** controllers .

We continue to work with the same board - **STM32F4-Discovery** and connect it to it with a shift register **74HC595** , although many begin to study this bus from the accelerometer, which is already installed on the board. So it's certainly easier, but with accelerometers we'll work a little later, and while we slightly break all the stereotypes and connect the simplest shift register, especially since we have already connected it in the [**lesson on AVR**](http://narodstream.ru/avr-urok-25-spi-podklyuchaem-sdvigovyj-registr-74hc595/) and therefore it will be easier for us to program it and do not have to worry about it narrate.

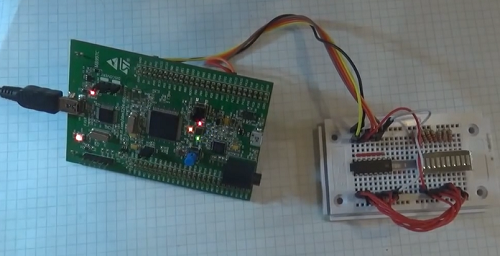
The only thing I recall is that this register is mainly used to convert serial code transmitted via SPI to parallel on an 8-bit bus. A little bit more in detail I told about the bus and the microcircuit in the video version, the link to which is at the bottom of the page.

Here is the circuit for connecting this chip to our controller (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2016/11/Image00-9.png)

The chip on the diagram is slightly different, but we will have it stated in the topic of the lesson.

But that's how it looks practically

[](http://narodstream.ru/wp-content/uploads/2016/11/Image05-10.png)

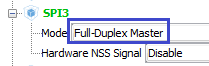
Instead of eight LEDs we have a LED matrix, so it's more convenient and compact.

We do not use the MISO input on the controller, since we do not need any information from the chip.

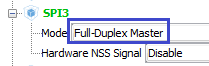
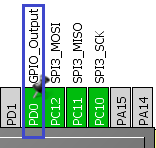
We create the project from MYLCD80, call it SPI595. We remove the files lcd.c and lcd.h from the folders.

Start the Cube. Since the project was created in an earlier version of the Cube, then in response to the dialogue with the three buttons, press "Migrate" so that our project adapts to the new version.

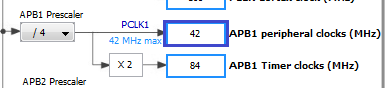
Enable SPI3. Why SPI3, because the tabs on the board on the handy side



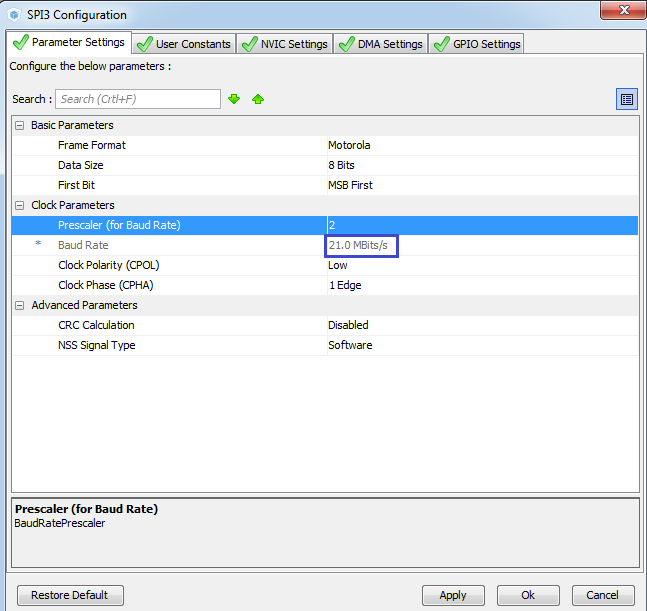
We turn PD0 on the output, it is also next to these legs and we need it for Chip Select

Because SPI3 is on the periphery of APB1 (see Reference Manual), then adjust it for the maximum frequency in Clock Configuration



Let's see SPI3 in Configuration



Judging by the technical documentation for the chip, with this frequency, it must cope. It is claimed 100 MHz.

We do not touch anything else. Generate the project and run it. Set up the programmer for auto-cutting.

Remove the declaration of lcd.h from main.h.

Also remove all the code from main.c, well, of course not all, and which we created ourselves, we do not need to remove the self-generated ones.

We declare in main.c some of the defenses for controlling the Chip Select leg

/ \* USER CODE BEGIN PV \* /

/ \* Private variables ------------------- \* /

**#define cs\_set () HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_0, GPIO\_PIN\_RESET)**

**#define cs\_reset () HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_0, GPIO\_PIN\_SET)**

**#define cs\_strob () cs\_reset (); cs\_set ()**

/ \* USER CODE END PV \* /

Let's start writing code in the main () function - enable the chip selection

  / \* USER CODE BEGIN 2 \* /

**cs\_set ();**

Create a buffer for the data to be sent to the SPI, because this requires a transfer function

/ \* Private variables ------------------- \* /

**uint8\_t aTxBuffer [1] = {0};**

#define cs\_set () HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_0, GPIO\_PIN\_RESET)

On page 814 of STM32F4\_HAL\_User Manual.pdf, we find the function of data transfer to the SPI. Let's write her code in the project

  / \* USER CODE BEGIN 2 \* /

        cs\_set ();

**HAL\_SPI\_Transmit (& hspi3, (uint8\_t \*) aTxBuffer, 1, 5000);**

Before calling this function, put a number in the buffer, for example 1

        cs\_set ();

**aTxBuffer [0] = 0x01;**

        HAL\_SPI\_Transmit (& hspi3, (uint8\_t \*) aTxBuffer, 1, 5000);

Now we need a pulse on the cs leg (page 5 of the chip's datasheet)

        HAL\_SPI\_Transmit (& hspi3, (uint8\_t \*) aTxBuffer, 1, 5000);

**cs\_strob ();**

Now let's send another digit after a short delay

        cs\_strob ();

**HAL\_Delay (1000);**

**aTxBuffer [0] = 0xFF;**

**HAL\_SPI\_Transmit (& hspi3, (uint8\_t \*) aTxBuffer, 1, 5000);**

**HAL\_Delay (1000);**

**cs\_strob ();**

  / \* USER CODE END 2 \* /

Let's sew controller, let's see the result



Well, let's now organize some counter for our matrix.

Declare a variable for the counter

int main (void)

{

  / \* USER CODE BEGIN 1 \* /

**uint8\_t i = 0;**

  / \* USER CODE END 1 \* /

Now let's add to the infinite counter itself

  while (1)

  {

**for (i = 0; i <= 255; i ++)**

**{**

**aTxBuffer [0] = i;**

**HAL\_SPI\_Transmit (& hspi3, (uint8\_t \*) aTxBuffer, 1, 5000);**

**cs\_strob ();**

**HAL\_Delay (100);**

**}**

                / \* USER CODE END WHILE \* /

Again we'll sew and see



In the [next lesson](http://narodstream.ru/stm-urok-24-hal-spi-led-staticheskaya-indikaciya/) we will try to connect the seven-segment LED indicator to the same shift register and direct them already.