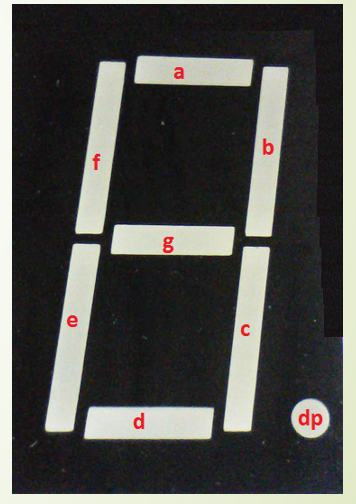
**HAL. Static indication**

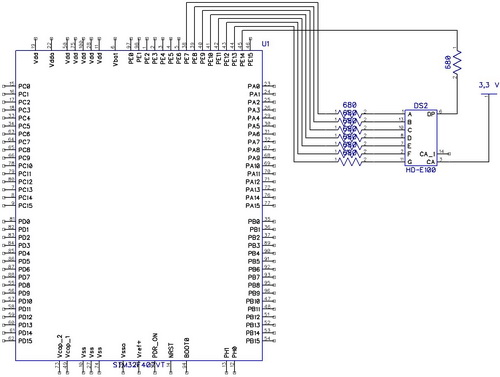
Today we will try to connect a seven-segment LED indicator to the STM32 controller using the HAL library. We have already connected such an indicator to the AVR microcontroller, it's time to try and do this with the STM controller. I think we will have no difficulties in this.

**Seven-segment LED indicator** is the purchase of LED segments located in a certain order



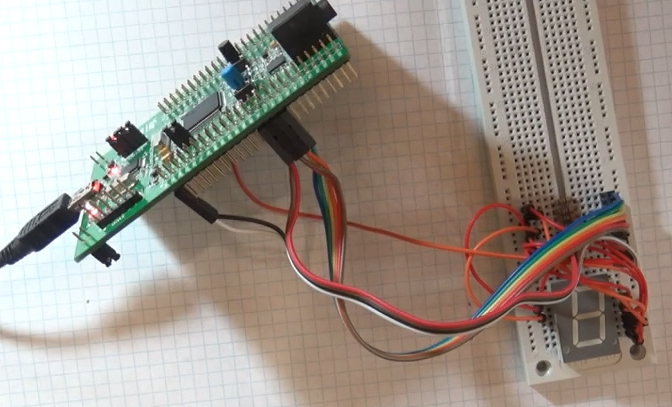
As we see, the indicator is not seven-segmented at all, but eight. Segments are usually denoted by Latin letters in alphabetical order, and the point - by the symbols dp.

We will connect our indicator here in this way (click the button to enlarge the image)

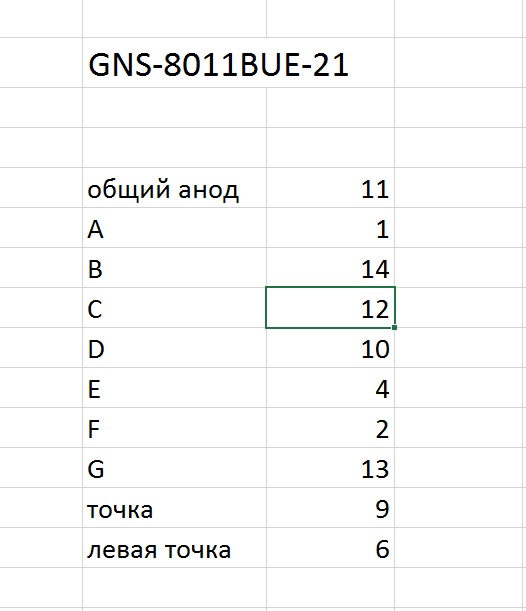
[](http://narodstream.ru/wp-content/uploads/2016/12/Image03-1.jpg)

Do not forget to also use resistors to provide current limiting through the legs of the ports. Since I have an indicator with a common anode, the common wire was connected to the 3.3-volt power leg.

And here's how it looks almost

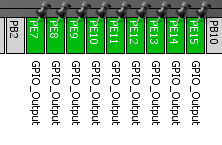


The purpose of the legs on my indicator, that is, on the one that I connected the following



As before, the project is created from the very first (TEST001), call it LED\_STAT

Start the cube, turn off all ports, turn on the output ports PE7-PE15.



In the settings in Configuration do not touch anything at all.

We generate the project.

Create and add files main.h, led.h, led.c

In main.h and led.c we connect

#include "led.h"

in main.c

#include "main.h"

in led.h

#include "stm32f4xx\_hal.h"

#include "main.h"

In the file led.h write defines (or macros) for the segments, so that it would be more convenient to work with them in the code

#define SA GPIO\_PIN\_7

#define SB GPIO\_PIN\_8

#define SC GPIO\_PIN\_9

#define SD GPIO\_PIN\_10

#define SE GPIO\_PIN\_11

#define SF GPIO\_PIN\_12

#define SG GPIO\_PIN\_13

#define SH GPIO\_PIN\_14

#define SA\_SET HAL\_GPIO\_WritePin (GPIOE, SA, GPIO\_PIN\_RESET)

#define SA\_RESET HAL\_GPIO\_WritePin (GPIOE, SA, GPIO\_PIN\_SET)

#define SB\_SET HAL\_GPIO\_WritePin (GPIOE, SB, GPIO\_PIN\_RESET)

#define SB\_RESET HAL\_GPIO\_WritePin (GPIOE, SB, GPIO\_PIN\_SET)

#define SC\_SET HAL\_GPIO\_WritePin (GPIOE, SC, GPIO\_PIN\_RESET)

#define SC\_RESET HAL\_GPIO\_WritePin (GPIOE, SC, GPIO\_PIN\_SET)

#define SD\_SET HAL\_GPIO\_WritePin (GPIOE, SD, GPIO\_PIN\_RESET)

#define SD\_RESET HAL\_GPIO\_WritePin (GPIOE, SD, GPIO\_PIN\_SET)

#define SE\_SET HAL\_GPIO\_WritePin (GPIOE, SE, GPIO\_PIN\_RESET)

#define SE\_RESET HAL\_GPIO\_WritePin (GPIOE, SE, GPIO\_PIN\_SET)

#define SF\_SET HAL\_GPIO\_WritePin (GPIOE, SF, GPIO\_PIN\_RESET)

#define SF\_RESET HAL\_GPIO\_WritePin (GPIOE, SF, GPIO\_PIN\_SET)

#define SG\_SET HAL\_GPIO\_WritePin (GPIOE, SG, GPIO\_PIN\_RESET)

#define SG\_RESET HAL\_GPIO\_WritePin (GPIOE, SG, GPIO\_PIN\_SET)

#define SH\_SET HAL\_GPIO\_WritePin (GPIOE, SH, GPIO\_PIN\_RESET)

#define SH\_RESET HAL\_GPIO\_WritePin (GPIOE, SH, GPIO\_PIN\_SET)

In the led.c we write a function for outputting the digits to the indicator.

The code for each leg is written with the condition that the segment will light up if there is a low level on this leg, and if it is high, it will not be

void segchar (uint8\_t seg)

{

        switch (seg)

        {

                case 1:

                        SA\_RESET; SB\_SET; SC\_SET; SD\_RESET; SE\_RESET; SF\_RESET; SG\_RESET; SH\_RESET;

                        break;

                case 2:

                        SA\_SET; SB\_SET; SC\_RESET; SD\_SET; SE\_SET; SF\_RESET; SG\_SET; SH\_RESET;

                        break;

                case 3:

                        SA\_SET; SB\_SET; SC\_SET; SD\_SET; SE\_RESET; SF\_RESET; SG\_SET; SH\_RESET;

                        break;

                case 4:

                        SA\_RESET; SB\_SET; SC\_SET; SD\_RESET; SE\_RESET; SF\_SET; SG\_SET; SH\_RESET;

                        break;

                case 5:

                        SA\_SET; SB\_RESET; SC\_SET; SD\_SET; SE\_RESET; SF\_SET; SG\_SET; SH\_RESET;

                        break;

                case 6:

                        SA\_SET; SB\_RESET; SC\_SET; SD\_SET; SE\_SET; SF\_SET; SG\_SET; SH\_RESET;

                        break;

                case 7:

                        SA\_SET; SB\_SET; SC\_SET; SD\_RESET; SE\_RESET; SF\_RESET; SG\_RESET; SH\_RESET;

                        break;

                case 8:

                        SA\_SET; SB\_SET; SC\_SET; SD\_SET; SE\_SET; SF\_SET; SG\_SET; SH\_RESET;

                        break;

                case 9:

                        SA\_SET; SB\_SET; SC\_SET; SD\_SET; SE\_RESET; SF\_SET; SG\_SET; SH\_RESET;

                        break;

                case 0:

                        SA\_SET; SB\_SET; SC\_SET; SD\_SET; SE\_SET; SF\_SET; SG\_RESET; SH\_RESET;

                        break;

        }

}

We write a prototype for it in led.h

void segchar (uint8\_t seg);

In the main module main.c in the main () function we add the code so that at the start the indicator does not shine

MX\_GPIO\_Init ();

  / \* USER CODE BEGIN 2 \* /

**HAL\_GPIO\_WritePin (GPIOE, GPIO\_PIN\_7 | GPIO\_PIN\_8 | GPIO\_PIN\_9 | GPIO\_PIN\_10**

**| GPIO\_PIN\_11 | GPIO\_PIN\_12 | GPIO\_PIN\_13 | GPIO\_PIN\_14,**

**GPIO\_PIN\_SET);**

        / \* USER CODE END 2 \* /

Also in main () we add a variable for the account

int main (void)

{

  / \* USER CODE BEGIN 1 \* /

**uint8\_t i = 0;**

  / \* USER CODE END 1 \* /

Also adding code for an account in an infinite loop

  while (1)

  {

**for (i = 0; i <10; i ++)**

**{**

**segchar (i);**

**HAL\_Delay (1000);**

**}**

        / \* USER CODE END WHILE \* /

We will collect the code, we'll sew the controller and see the result (in the video version of course it looks much more dynamic)

