

All Collective Personal TOP

Good Bad

Поиск

8L-Course, Part 2 - GPIO

STM8

HIGH-QUALITY PCB

Free shipping

ONLY \$5

FOR 10 PIECES

- Rogers, HDI, aluminum and rigid-flex PCB are available now
- Production time 24 hours

PCB ASS

ONLY

- Component
- Quality a

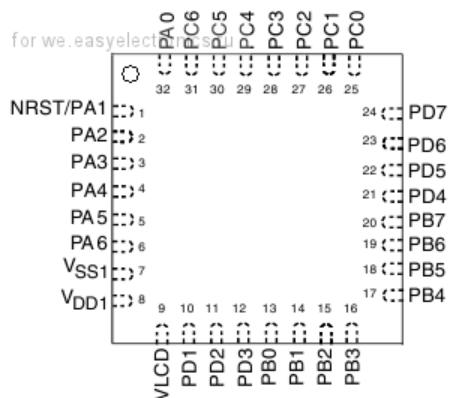
[← Part 1 — Hello LED! Contents Part 3 — Interrupts →](#)

In the last part we started the LED blinker. Now it's time to figure out how it works and how the GPIO module in STM8 is arranged.



Hardware

The ports of the STM8 are designated the same way as those of the AVR — with a letter. Our flasher used port D. Each port consists of **8 pins**, which also resembles the AVR, and any eight-bit microcontrollers in general. However, on cases with a small number of pins, some ports are cut off and half of the pins are missing. In the microcontroller on the module for PB2, this is exactly the case:



But all the peripherals (ADC inputs, timer inputs/outputs) remained on the same pins as in larger cases. This means that you can easily transfer programs from

Live

Comments Publications

penzet → [Sprint Layout in OS X 18](#) → [Software for electronics engineer](#)

Vga → [EmBitz 6](#) → [Software for electronics engineer](#)

Vga → [Rail-to-rail: ideal operational amplifier or a clever marketing ploy? 1](#) → [Theory, measurements and calculations](#)

Flint → [A little more about 1-wire + UART 56](#) → [Hardware connection to the computer.](#)

penzet → [Cross-platform terminal - SerIO 3.x 25](#) → [Software for electronics engineer](#)

From Gorn → [PIP regulator 2](#) → [Algorithms and software solutions](#)

whoim → [CC1101, Treatise on tracing 89](#) → [Blog im. Khomin](#)

OlegG → [Clock on Bluetooth LE module 8](#) → [Cypress PSoC](#)

Technicum505SU → [Nuances of PWM control of a DC motor by a microcontroller 3](#) → [Theory, measurements and calculations](#)

sunjob → [DDS synthesizer AD9833 88](#) → [Blog named after grand1987](#)

dihalt → [W801, LCD screen and tar spoon 2](#) → [Detail](#)

nictrace → [New Arduino-compatible board. 20](#) → [FPGA](#)

sunjob → [Connecting the ARM GCC compiler to CLion 1](#) → [Software for electronics engineers](#)

Vga → [CRC32: on STM32 as on PC or on PC as on STM32 58](#) → [STM32](#)

dmitrij999 → [Capturing images from a USB camera using STM32 6](#) → [STM32](#)

Gilak → [Expanding the capabilities of a simple MC up to an ADC on 2 or 1 pin 8](#) → [Theory, measurements and calculations](#)

sva_omsk → [Lithium ECAD - Russian PCB CAD 40](#) → [Software for electronics engineer](#)

x893 → [W801 - budget controller with Wi-Fi 4](#) → [Details](#)

podkassetnik → [Changing the standard instrument cluster lighting of Logan-like cars 5](#) → [Automotive electronics](#)

Several pins are occupied by power supply. In the picture, there are only two of them (Vss1 and Vdd1), but in multi-pin cases there may be several pairs. Then you need to connect them all, not forgetting a single pair. Otherwise, you can get strange glitches in the microcontroller's operation.

One pin is occupied by a reset signal — NRST. Moreover, it can be programmatically switched to the normal mode and it will become (almost) a normal GPIO pin called A1. "Almost" — because you can't assign an interrupt to it, and you can't disable the pull-up resistor.

Another pin is occupied by power for the built-in LCD controller. If the LCD is not used, then this pin can be left alone.

As a result, out of 32 pins, we have 28 (or 29 if you turn off the reset) for GPIO. This is what the story will be about.

Electrics

The picture with the pinout, which hangs just above, only shows the names of the pins, without unnecessary details. All the details are collected in the table below ("Medium density STM8L15x pin description", in the datasheet on the MCU). Several parameters are indicated for each pin.

Table 5. STM8L15x pin description

Pin number				Pin name	Type	I/O level	Input			Output			Main function (after reset)	Default alt. functio
UFQFPN48 and LQFP48	UFQFPN32	UFQFPN28	WLCSP28				floating	wpu	Ext. interrupt	High sink/source	OD	PP		
2	1	1	C3	NRST/PA1 ⁽¹⁾	I/O			X		HS	X	X	Reset	PA1
3	2	2	B4	PA2/OSC_IN/[USART1_TX] ⁽³⁾ / [SPI1_MISO] ⁽³⁾	I/O		X	X	X	HS	X	X	Port A2	HSE oscillator [USART1 trans [SPI1 master ir out] /
4	3	3	C4	PA3/OSC_OUT/[USART1_RX] ⁽³⁾ / [SPI1_MOSI] ⁽³⁾	I/O		X	X	X	HS	X	X	Port A3	HSE oscillator [USART1 recei master out/slav
				PA4/TIM2_BKIN/										Timer 2 - break

I/O level — indicates the maximum voltage that can be supplied to the pin. TT — means that the pin can easily withstand a voltage of 3.6V (regardless of the supply voltage of the MCU). And FT means that 5V is not a problem for it. However, there are only two FT pins - C0 and C1, which are used to work with the I2C bus. If this column does not say anything, then the pin cannot be supplied with a voltage higher than the supply voltage of the microcontroller.

Floating Input - shows whether the pin can work as an input without a pull-up resistor (A1 - which has NRST, it cannot - the pull-up is always on there), and **wpu (weak pull-up)** - whether there is a pull-up resistor on this pin (there is none on pins C1 and C0). The mode in which the pin will be at the start of the microcontroller is highlighted in bold - for almost all of them, this is an input without a pull-up. The nominal value of the pull-up resistor is about **40 kOhm** , for all pins (including NRST).

Not all pins can give the same current when configured for output. Those that can deliver up to 20mA are marked SH (in the **High sink/source column**), and the "weak" ones were left without this mark. And these are C0 and C1 again - which can pass a current of only 5 mA through themselves. And there is also pin A0, which, in addition to debugging via SWIM, is designed to connect an IR LED or a similar powerful load and can "pull" a current of up to 80 mA (i.e. it

1-Wire The other arduino ARM
Assembler Atmel AVR C++ compel
DIY enc28j60 ethernet FPGA gcc I2C
AND KEIL LaunchPad LCD led linux
LPCXpresso MSP430 npx PCB PIC
pinboard2 RS-485 RTOS STM32
STM8 STM8L OF UART USB
algorithm assembler ADC the library
power unit detail display an idea tool
competition competition 2 ANGRY
microcontrollers for beginners review
Debug board soldering iron
printed circuit board salary FPGA crafts
purchases programmer programming
Light-emitting diode software scheme
circuit design Technologies
smart House photoresist it's free crap
Times humor

Blogs

Group

AVR	38.98
STM8	37.92
Garbage truck 🗑️	29.53
STM32	28.46
Detail	24.63
Connection of hardware to the computer.	24.04
Circuit design	18.15
Smart House	17.75
MSP430	17.13
LPC1xxx	14.79

All blogs

(push-pull) columns show whether the pin can operate in the open collector mode (or "open drain"), when instead of a high level it switches to the input. PP, accordingly, indicates the ability to operate in the Push-pull mode, when the pin can produce both low and high levels. Almost all pins can operate in the Push-pull mode, and if necessary, switch to the open-drain mode. The exception is, again, C0 and C1, which, apart from open-drain, cannot work in any other way. Keep this in mind when distributing pins. **Once again, briefly, about the electrics:** The maximum current output by the pin is 20 mA (except for C1 and C0, which have 5 mA, and A0, which can "pull" up to 80 mA). The total maximum current of the stone is 80 mA (i.e. the current through all the pins should not exceed this mark). The maximum supply voltage is 3.6 V for STM8L and 5.5 V for STM8S. The minimum is 1.8 V for STM8L and 2.7 V for S. Moreover, in STM8L you can disable flash memory and execute code from RAM - then they can work at a voltage of 1.55 V. **Programming** From a programmer's point of view, the GPIO port in STM8 is five registers: **Px_ODR** is the value that is output to the port **Px_IDR** is the current state of the port **Px_DDR** is the direction (input or output) **Px_CR1** is the settings one **Px_CR2** is the settings two (instead of x is the port letter) The first three registers probably do not represent anything new to anyone. Just in case, I will arrange a crash course: Each bit of **Px_DDR** is responsible for the direction of the corresponding pin of the microcontroller. 1 - the pin is configured for output, 0 - for input. The bits in **Px_ODR** set the level on the pin (if it is configured for output). 0 is a low level, 1 is a high level. Moreover, depending on the value in CR1 (see below), the high level can be eliminated - the pin operates in the open drain mode. The STM8 holds the levels very well - without a load, the pin pressed to the ground produces 2.3 mV (and most likely, this is the error of my multimeter). Under a load of 1 mA, the voltage is already 26 mV, and under a load of 20 mA (the maximum value) - 556 mV. If our pin is configured for input, then writing to ODR will not do anything. Reading from Px_ODR is also possible and then we will read the last value written there (even if the pin is currently at a different level). But from **Px_IDR**

you can read the current level on the pins. And here we can briefly finish the description of this register, because there is nothing more to catch here :)

Registers **CR1** and **CR2** work differently depending on the value of the corresponding bit in the DDR register (that is, on whether the pin is configured for input or output).

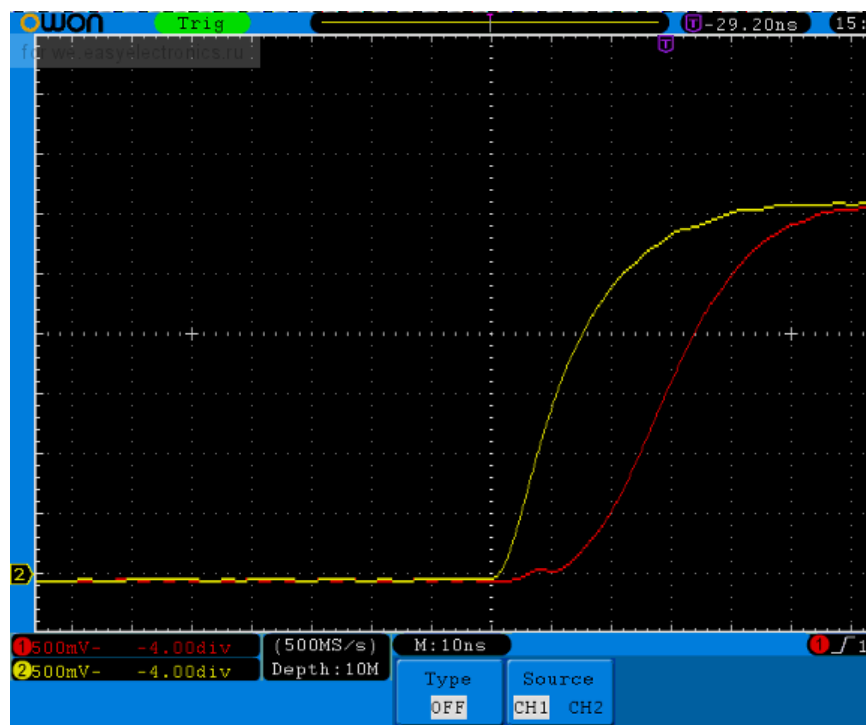
If the pin is configured for input , then the bit in CR1 controls the internal pull-up resistor. One turns it on, and zero turns it off - and the pin just dangles

09/07/2024 10:15 in the air. An input without a pull-up (and not connected to anything) catches interference that switches the triggers in the circuit. Pins hanging without a pull-up can easily add a hundred microamps. In short, if the device should consume as little current as possible, all the pins must be nailed to some constant level. It does not matter, with a pull-up or otherwise - the main thing is that they do not switch from interference.

The pull-up resistors here are quite weak - about 38-40 kOhm (at room temperature). In real devices (not mock-ups on the table), it is better to put external pull-up resistors on the buttons. With a nominal value of 10 k or less. This is especially important if there are long wires to the buttons, and / or there are sources of interference nearby. Otherwise, you can get false alarms. The

CR2

register in this case [when the pin is configured as an input] is responsible for the interrupt from the pin - if there is one, then the interrupt is allowed. In STM8, you can catch interrupts from any pin (except A1 - NRST). Although there are some tricks here that limit freedom of action. But about interrupts in the next part. **When the pin works as an output**, CR1 is responsible for the operating mode. One is push-pull (i.e. the pin can equally successfully output a low or high level), zero is a parody of an open-drain output (in the original, "pseudo open-drain" :)) - the low level is still output, and instead of a high level, the pin switches to the input mode (in the DDR register, nothing changes, of course). In this mode, it is convenient to work with 1-wire, or with a software implementation of I2C. **CR2**, when the pin is configured for output, is responsible for the "maximum switching speed". Well, this is what is written in the datasheet, but in fact, the steepness of the fronts will depend on the value in this register. Like this:



The yellow channel is attached to the pin with a high switching speed (one in CR2), and the red one is attached to the pin with a low one. Switching occurs simultaneously (with one command), but the voltage rise rate is different. As a result, the fast pin rises about 15-20 nanoseconds earlier than the slow one. Usually, such a small difference does not bother anyone, but if you need to output a high-frequency signal from the MCU pin, or you need to get a signal with steep edges, then setting a bit in CR2 will solve almost all problems (except

In STM8S, the maximum speed of some pins is not configurable, but is hard-coded:

Output speed	O1 = Slow (up to 2 MHz)
	O2 = Fast (up to 10 MHz)
	O3 = Fast/slow programmability with slow as default state after reset
	O4 = Fast/slow programmability with fast as default state after reset

Keep this in mind if you work with them.

We seem to have figured out the GPIO device. Now you can look at the code from the previous part with a fresh look, already understanding exactly what all these registers are responsible for.

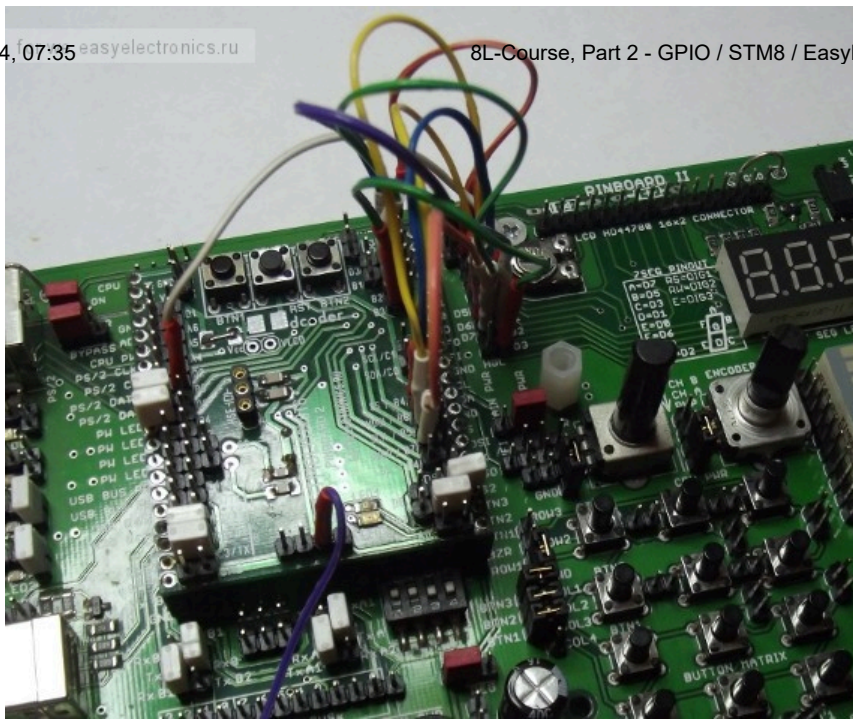
Blinking an LED is a classic, but very boring. Let's do something more complex. For example, an electronic dice: Random number generator. Let's take a seven-segment indicator, on which we will display the numbers 0 - 9 in turn. This count will stop when you press the button, and one number will remain lit on the indicator. And so on until the next button press. The numbers change at a high speed, and the person pressing the button cannot specifically guess the moment and stop the counter on the desired number.

I used an indicator installed on the PINBOARD. Its segments are output to the same pins where the data bus from the LCD display is located, and they light up with a high level (i.e. an indicator with a common cathode). The segments are connected to port B on the MK:

```
B0 - D0 (Сегмент E)
B1 - D1 (D)
B2 - D2 (Точка)
B3 - D3 (C)
B4 - D4 (G)
B5 - D5 (B)
B6 - D6 (F)
B7 - D7 (A)
```

Because port B is the only "full" port in the TQFP32 MK. The others are missing one or a couple of pins.

The common terminals from the discharges are connected via transistors to ground (to activate the discharge, you need to apply a high level to open the transistor) and are connected to the RS, R/W and E pins. We only need one discharge for now, so we just connect the required terminal to MAIN PWR - the transistor will be constantly open.



In order not to manually count the codes responsible for the symbol of each digit, we use this table:

```
//Закомментировать, если используется индикатор с общим анодом (0 = заже
#define COMMON_CATHODE

// Сегмент индикатора  Пин
#define segment_A 7
#define segment_B 5
#define segment_C 3
#define segment_D 1
#define segment_E 0
#define segment_F 6
#define segment_G 4
#define segment_DP 2

#ifdef COMMON_CATHODE
extern const char numbers[10] = {
/*0*/ (1 << segment_F) | (1 << segment_E) | (1 << segment_D) | (1 << seg
/*1*/ (1 << segment_C) | (1 << segment_B),
/*2*/ (1 << segment_D) | (1 << segment_E) | (1 << segment_G) | (1 << seg
/*3*/ (1 << segment_G) | (1 << segment_D) | (1 << segment_C) | (1 << seg
/*4*/ (1 << segment_G) | (1 << segment_F) | (1 << segment_C) | (1 << seg
/*5*/ (1 << segment_G) | (1 << segment_F) | (1 << segment_D) | (1 << seg
/*6*/ (1 << segment_G) | (1 << segment_F) | (1 << segment_E) | (1 << seg
/*7*/ (1 << segment_C) | (1 << segment_B) | (1 << segment_A),
/*8*/ (1 << segment_G) | (1 << segment_F) | (1 << segment_E) | (1 << seg
/*9*/ (1 << segment_G) | (1 << segment_F) | (1 << segment_D) | (1 << seg

#define DecimalPoint (1<<segment_DP)
#define ClearDisplay 0
#else
extern const char numbers[10] = {
/*0*/ ~((1 << segment_F) | (1 << segment_E) | (1 << segment_D) | (1 << s
/*1*/ ~((1 << segment_C) | (1 << segment_B)),
/*2*/ ~((1 << segment_D) | (1 << segment_E) | (1 << segment_G) | (1 << s
/*3*/ ~((1 << segment_G) | (1 << segment_D) | (1 << segment_C) | (1 << s
/*4*/ ~((1 << segment_G) | (1 << segment_F) | (1 << segment_C) | (1 << s
/*5*/ ~((1 << segment_G) | (1 << segment_F) | (1 << segment_D) | (1 << s
```



```

/*6*/ ~((1 << segment_G) | (1 << segment_F) | (1 << segment_E) | (1 << s
07:35 ~((1 << segment_C) | (1 << segment_D) | (1 << segment_B) | (1 << s
/*7*/ ~((1 << segment_G) | (1 << segment_F) | (1 << segment_E) | (1 << s
/*8*/ ~((1 << segment_G) | (1 << segment_F) | (1 << segment_E) | (1 << s
/*9*/ ~((1 << segment_G) | (1 << segment_F) | (1 << segment_D) | (1 << s

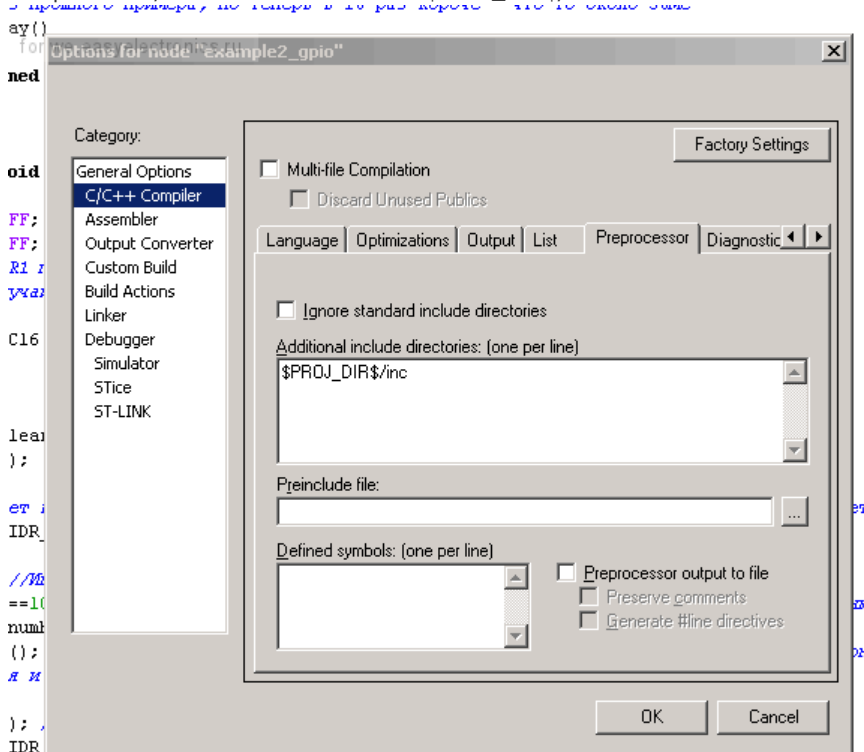
#define DecimalPoint ~(1<<segment_DP)
#define ClearDisplay 255
#endif
};

```

First, there are defines that determine the order in which the indicator segments are connected to the MC pins. There is also a define that determines the indicator type ("common anode" or "common cathode"). Depending on them, then an array numbers[] is created containing the codes of numbers from 0 to 9. In order to output a number to the indicator, you just need to take the corresponding element of the array and output it to the port.

In order not to clog up main, I have allocated this table in a separate file `7seg_table.h`, which is lying around in the project folder in the `inc` subfolder. In IAR, to connect a file to a project, you first need to add it to the list on the workspace panel (on the left). Right-click on it and select `Add -> Files...` Add our file. In general, this is not necessary for `.h` files (we did not connect `iostm81151k6.h` this way) - this is only necessary for `.c` files containing executable code.

If you try to build the project now, the compiler will complain that it doesn't know where your `7seg_table.h` is. By default, it looks for included files in the folder with IAR header files and in the folder with the project itself. And ours is in `/inc`. In order to explain to the compiler where to look for files, go to the project settings and select the C/C++ Compiler section there, and in it the Preprocessor tab. There, in the Additional include directories field, we write the path to our folder with the header file. It is better to write using IAR environment variables, so that the project can then be easily built on another computer and/or from another folder: `"$PROJ_DIR$/inc"`.



When searching for a file, IAR will replace \$PROJ_DIR\$ with the path to the folder with the project.

```
#include "7seg_table.h"
extern const char numbers[10];
```

We will fully use port B to work with the indicator, which means that all its pins must be configured for output.

```
PB_DDR = 0xFF;
```

The indicator segments light up with a high level, and the pins by default operate in open drain mode and cannot output a high level. Therefore, we raise all the bits in PB_CR1 to switch the port to push-pull mode:

```
PB_CR1 = 0xFF;
```

We also need a button. There are a whole bunch of them on the PINBOARD, but three can be easily thrown onto the processor module pins, designated BTN1, BTN2 and BTN3. The BTN2 button is connected to pin D6 - that's what we'll use.

The pin for the button should be configured as an input with a pull-up resistor. That is, there should be zero in Px_DDR (there is already 0x00 after reset), and one in Px_CR1.

```
PD_CR1_bit.C16 = 1;
```

We're done with the setup, let's write the main loop

```
while(1)
{
    PB_ODR = ClearDisplay; //Сбрасываем все пины порта B - индикатор тухнет
    SomeDelay();

    //Цикл будет выполняться до тех пор, пока не нажмут кнопку (тогда бй б
    while (PD_IDR_bit.IDR6 == 1)
    {
        value++; //Инкрементируем счетчик
        if (value==10) value=0; //Проверяем - не ушел ли он за предел (индика
        PB_ODR = numbers[value]; //Выводим число на индикатор
        SomeDelay(); //Если не сделать задержку, то пропадет красивый эффект
        //солятся и будет 8 с немного разной яркостью сегментов.
    };
    SomeDelay(); //После того как нажали кнопку, делаем задержку для антид
    while (PD_IDR_bit.IDR6 == 0); //Ждем, пока кнопку отпустят.
    SomeDelay(); //Опять антидребезг
    while (PD_IDR_bit.IDR6 == 1); // Ждем, пока нажмут
    SomeDelay();
    while (PD_IDR_bit.IDR6 == 0); //... и снова отпустят

    //В итоге наша шарманка будет считать цифры, пока не нажмут кнопку.
    //И продолжать счет только после повторного нажатия
};
```

You can build the project and run it. Just don't forget to correct 7seg_table.h if you use another indicator type or another connection.

That's probably all. Next time we'll look at interrupts from pins (and interrupts in general). Stay tuned!

[← Part 1 — Hello LED! Contents Part 3 — Interrupts →](#)

STM8 , STM8L, GPIO


Comments (13)

[RSS](#) [Collapse](#) / [Expand](#)

Now with the JPI (pardon the swearing) everything is not so...

Where are the detailed explanations (to another) novice bydlokoder about the total maximum current of the stone, about the differences between HS pins and regular ones, about the differences in the types of connectors on the output-input pins, how many power inputs? Speed, edges, initial pin capacities, etc... What is this explanation with segmenters for, what is the speech about, as indicated in the topic? Not a word about the alternativeness of the selection functions and self-tuning of the pin...

0

 **champion**
December 16, 2012, 20:10

No need to downvote him! No, really, overall valio is right. I myself noticed that there was a lot more that could have been written. But I noticed it when I left-clicked on the publish button.

Fortunately, we don't have a periodical here, but the live internet - you can listen to criticism, add something and correct it.

Now I feel sleepy, so all the editing will be tomorrow.

Kisses to everyone in this chat

+1

 **dcdoder**
December 16, 2012, 20:40

If I remember correctly, the target audience was defined in [Who Needs It?](#) And given the above, and after reading the post, it is clear that the person asking the question did not read (either the introduction, or the lesson, or both at once).

0

 **angel5a**
December 17, 2012, 1:42 PM


Half of the questions valio listed are STM8 specific, so they are well worth considering.

0

 **Vga**
December 17, 2012, 1:45 PM

PA0 can "sink" 80 mA, but only source 25, like all the others. :)

0

 **Katz**
December 17, 2012, 11:33

Good day! The comment is not on topic, but still: it would be interesting to know what simulators are available for STM8? Despite the mega debugging capabilities, it is still somehow unethical to constantly torture the crystal... I think beginners are very interested in this problem

0

 **Fountain-G**
December 18, 2012, 10:22

... during the year - this is a minimum of 166 guaranteed reflashes daily. Try to breed at least 50. But if you spend \$ 10 on a new set - the toad chokes, there is RAM. Everything is really bad, then hypothetically in IAR.

0

 **champion**
December 18, 2012, 11:32

Eh... I really wanted to hear: don't be stupid, download Proteus, it already has support...

0

Why, if there is debugging on live hardware, which will obviously work as it is seen Oo

0



thermophysicist
December 18, 2012, 1:06 PM

Well, what is this "why"? I know, I work with pleasure. But with Avrs (and peaks, damn them...) I got used to the simulator, the thing is also very pleasant and certainly would not hurt

0



Fountain-G
December 18, 2012, 1:20 PM

what about DMA??

0



the scalds
December 19, 2012, 21:07

Atension!

I accidentally rewrote this squalor a bit. If anyone cares, you can reread it :) The thread is requested by **valio** , who will say that it hasn't gotten any better :)

0



dcoder
09 February 2013, 14:38

It can be added that in reset some pins are pulled up to the power supply (for STM8L15X, these are PB0 and PB4).

0



TheLongRunSmoke
April 15, 2014, 14:14

Only registered and authorized users can leave comments.