8L-Course, Part 1 - Hello LED!

STM8

← Part 0 — Start Contents Part 2 — GPIO →

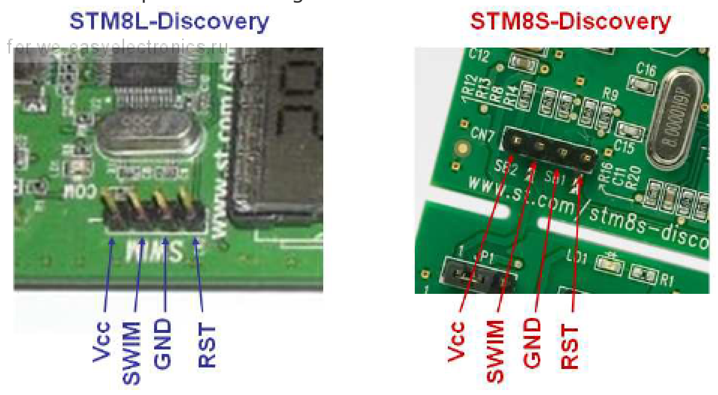
Warning!
If you read the article about the STM8L module, and specifically the part about quick start, you will see a lot of familiar text here and it will be a bit boring for you to read. And don't say later that you weren't warned.

In the previous part we had an overview and a bit of theory (I hope it was enough to understand what was going on), and in this one there will be practice. We will figure out the connection and firmware of the MCU, install and configure IAR, write a simple LED blinker in it and launch it.

In addition, in this part I want to inflict cruel reprisals on uncertainties: we have three hardware options: a single STM8L-Discovery with a debugger, PINBOARD2 + STM8L module (without a debugger, but with a bootloader) or discovery + PINBOARD2. I will assemble all devices and examples on a pinboard, but for those who use other hardware, I will leave a diagram for each example so that you can assemble it on anything.

Let's start with the STM8-Discovery + PINBOARD2 pair.

The discovery boards have an **ST-Link** debugger (a bit flawed, though, since a normal ST-Link can also debug STM32). **The SWIM and RST** lines from it are routed to the pins on the edge of the board:



Live
Comments Publications

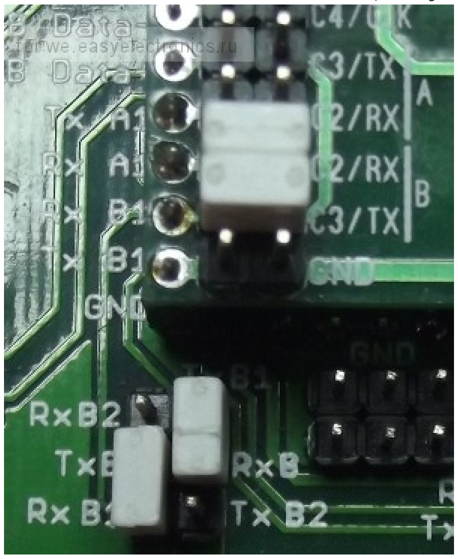
penzet → Sprint Layout in OS X 18 → Software for electronics engineer
Vga → EmBitz 6 → Software for electronics engineer
Vga → Rail-to-rail: ideal operational amplifier or a clever marketing ploy? 1 → Theory, measurements and calculations
Flint → A little more about 1-wire + UART 56 → Hardware connection to the computer.
penzet → Cross-platform terminal - SerIO 3.x 25 → Software for electronics engineer
Gornist → 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 fly in the ointment 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
Gilaks → 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

But the debugger is connected to the MCU, which is on the discovery, and therefore, in order for it to be able to flash external MCUs, it is necessary:
- For STM8L-Discovery: remove the jumpers on the opposite side from the SWIM connector (they are labeled DISCOVERY and ST-LINK)
- For STM8S-Discovery (and SVL there too): unsolder the SB1 and SB2 jumpers near the SWIM connector. Or just break the board in two! THIS IS SPAAARTAA! (I'm not kidding, there are cuts across the board just for breaking the debugger off the breadboard with the MCU).

After these preparations, you can connect the wires to the STM8L module on the pinboard. It is better to connect both SWIM and RESET, although of course, as long as we do not use sleep modes and do not disable SWIM, RESET is not really needed: SWIM is always active. At this point, the preparations are complete and you can already flash something.

For those who only have Discovery, no special preparations are required. Complete Plug&Play.

And those who have PINBOARD2 with an STM8L module, but no Discovery, will have to use a bootloader to flash the MK. Do not be afraid, it is not scary :) First, you need to connect the UART from FT2232 to the MK. Which channel to use (A or B) is not important, as long as you know which COM port corresponds to which channel. I used channel B, the jumpers are installed like this:



Now the boot can communicate with the computer. We could stop here, but there is one problem - **the bootloader waits for a command for only a second after starting** , and then starts the firmware, which is in the flash memory. In other words, you need to press the reset button and within a second after that, have time to poke a button in the firmware program. Gamers accustomed to such things with a honed reaction will not even wrinkle their noses, but for someone it may be inconvenient.
With approximately these thoughts (and a solution plan), I knocked on **the Vga** . After three days and three nights (not true) Vga showed a working hack. The point is that **before sending the start byte to the bootloader, the program pulls the DBUS4** (DTR) pin from high to low - then high again - waits 200 ms - and only then sends the start signal. **DBUS4 is connected by a wire to the RST pin on the MK** .

Vga → ROPS (Rem Object Pascal Script) - embedded interpreter of the Pascal language. Plugin PSImport_Classes 3 → Algorithms and software solutions

Full broadcast | RSS

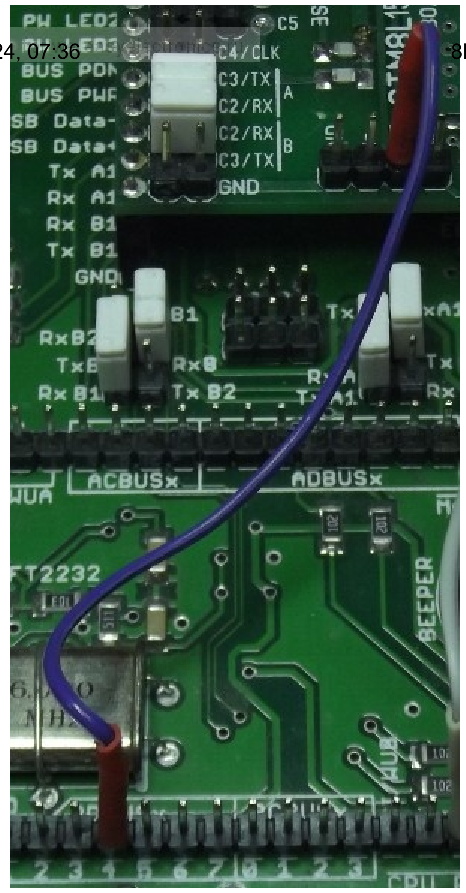
1-Wire Altera arduino ARM Assembler Atmel AVR C++ compel DIY enc28j60 ethernet FPGA gcc I2C IAR KEIL LaunchPad LCD led linux LPCXpresso MSP430 nxp PCB PIC pinboard2 RS-485 RTOS STM32 STM8 STM8L TI UART USB algorithm assembler ADC library power unit detail display idea tool contest competition2 LUT microcontrollers for beginners review Debug board soldering iron printed circuit board pay FPGA crafts purchases programmer programming Light-emitting diode software scheme circuit design Technologies smart House photoresist freebie crap Watch humor

Blogs

Top

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



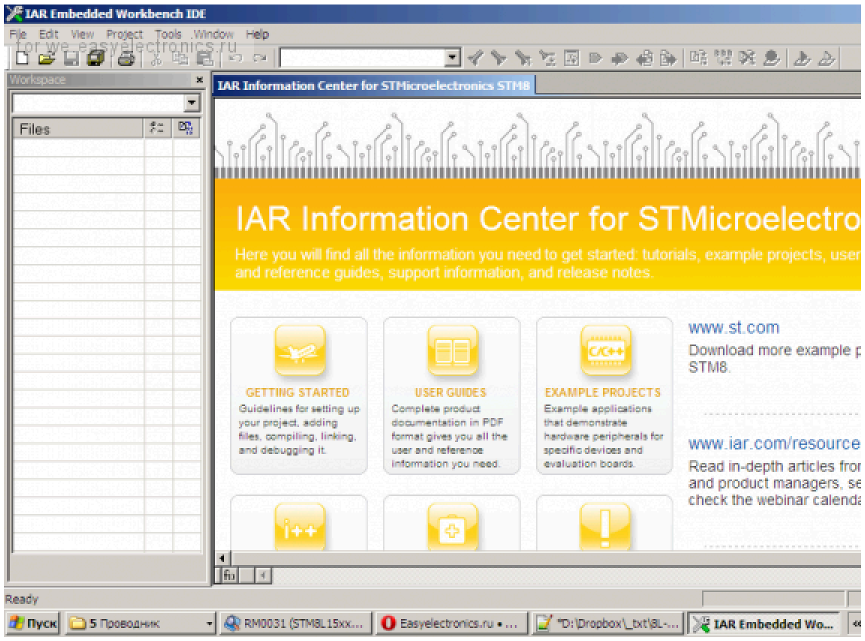
Thus, **the program itself resets the MK** before accessing the bootloader. This means that when flashing the firmware, we don't need to touch the board at all - just click the mouse on the windows. Convenient! (for the laziest, below we will show how not to click the mouse at all :)

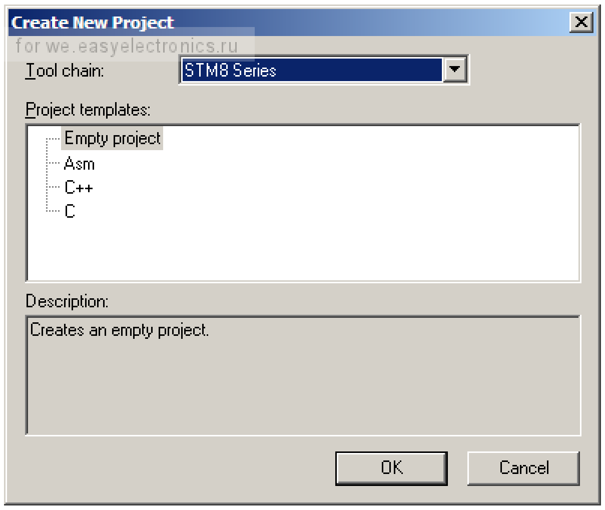
And now let's deal with the software.

To work with STM8, I used **IAR**. There is also COSMIC, which many people praise, but I don’t see much difference. And it’s not that hard to switch to another IDE if necessary.

It lives here . The trial version has **a limit of 8 kilobytes** for the output code (that is, 24k of memory will be lying around idle in any case), but for our purposes this will be enough. We select the **kickstart**

version and get to a long and boring registration. Actually, it was somewhere on rutracker, so if you are too lazy to register, look there. After registration (all sorts of addresses/locations/passwords can be filled in with arbitrary crap), an email with a link is sent to the email. On the page by the link, a download link and a license key (which it is advisable not to lose before installation) will be waiting for us. The software weighs 113 MB, and will download slowly. You can drink some coffee. Immediately after the installation starts, we will be asked for the license number. We copy it from the download page and continue. The next step asks for a license key. At this point, all the hassle with licenses is officially over. You can take off the blindfold and launch the "Jolly Roger" :) You can leave the installation type as full, because there is not much to choose from in the "custom" one. It is installed quickly, you won't even have time to drink coffee (you should have done so when downloading!). After installing IAR itself, you will be asked to install drivers for ST-Link and STIce - agree. They are also installed very quickly and even if you don't have Discovery, they will come in handy for the future. Let's launch it. We are greeted by the main IAR window. Let's try to create a project? Let's go to the **Project -> Create New Project** menu . We are given several project templates to choose from: **Choose C** . IAR will ask where to save and what to name the project. Call it whatever you like, mine is called **"example1_blink"** . Now we have a project in front of us. More precisely, its template: On the left is **the workplace panel** with the project structure. Actually, for now there is only main.c and the output file. Right-click on the project name (in workplace), select **Options** , and we get to **the project settings** . There, first of all, we select our MCU: STM8L152C6 for those who play with discovery, and for STM8L152K6 for pinboard owners. In the C/C++ Compiler section, on the Optimization tab, you can configure code optimization. This is so for the future, so that they don't look for where it is hidden. For now, you can leave it at Low. In **the Output Converter** section, you can (and for those who will flash via bootloader, you need to) configure the generation of output code in **Intel Extended format**

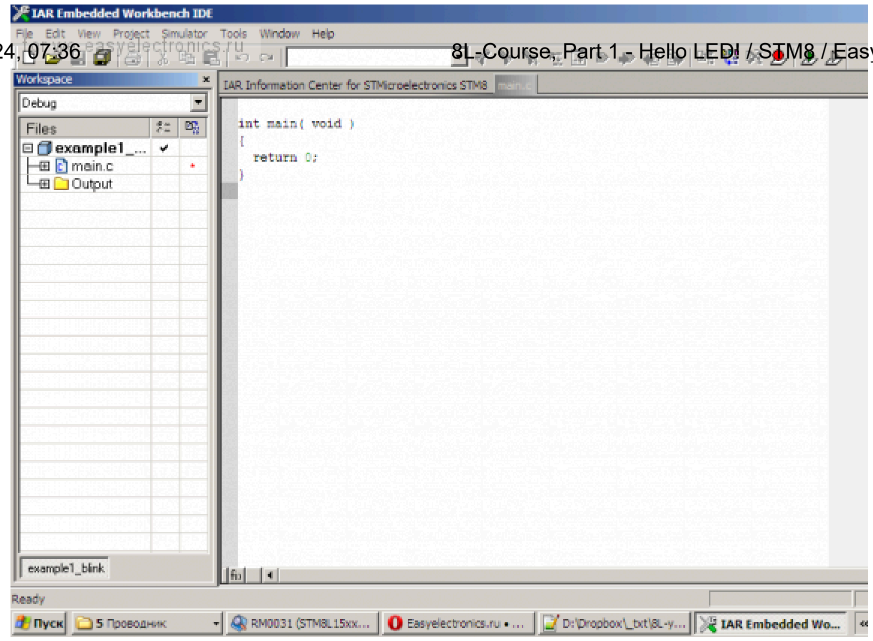

IAR Embedded Workbench IDE
File Edit View Project Tools Window Help
for we.easyelectronics.ru
Workspace
Files
IAR Information Center for STMicroelectronics STM8
IAR Information Center for STMicroelectro
Here you will find all the information you need to get started: tutorials, example projects, user and reference guides, support information, and release notes.
GETTING STARTED
Guidelines for setting up your project, adding files, compiling, linking, and debugging it.
USER GUIDES
Complete product documentation in PDF format gives you all the user and reference information you need.
EXAMPLE PROJECTS
Example applications that demonstrate hardware peripherals for specific devices and evaluation boards.
www.st.com
Download more example p STM8.
www.iar.com/resource
Read in-depth articles fror and product managers, se check the webinar calenda
Ready
Пуск
5 Проводник
RM0031 (STM8L15xx...
Easyelectronics.ru • ...
"D:\Dropbox_txt\8L-...
IAR Embedded Wo...



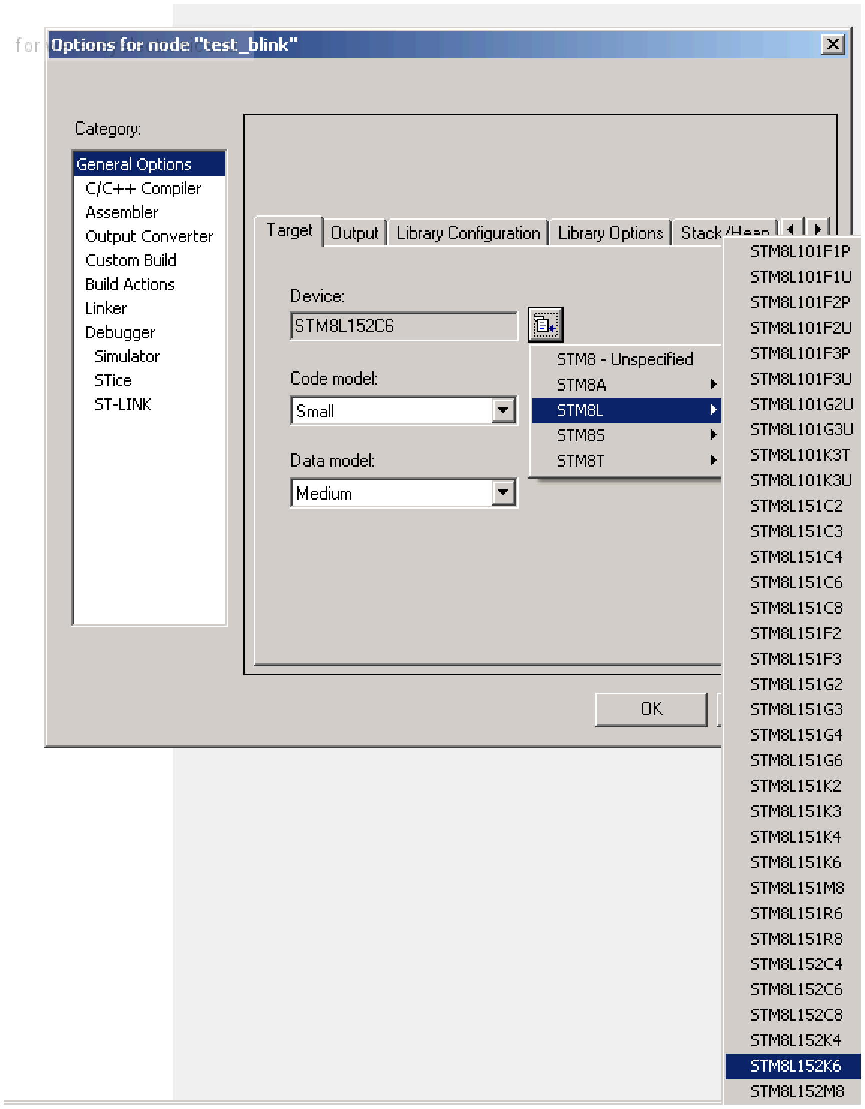
Tool chain: STM8 Series

Project templates:
- Empty project
- Asm
- C++
- C

Description:
Creates an empty project.


IAR Embedded Workbench IDE
File Edit View Project Simulator Tools Window Help
Workspace
Debug
Files
example1_...
main.c
Output
example1_blink
IAR Information Center for STMicroelectronics STM8
main.c
Ready

```
int main( void )
{
  return 0;
}
```



Device: STM8L152C6

Code model: Small

Data model: Medium

. The .hex file that will need to be given to the flashing program is located in the project folder in the /Debug/Exe subfolder. And it has the same name as the project.

And for those who work via ST-Link on discovery, there is a **debugger** section . There, **instead of Simulator, you need to select ST-Link** so that IAR works with the hardware debugger, and not with the simulator.

We are done with the project settings, now we need to connect the header file with the description of the peripherals, registers, etc.

At the very beginning of main.c, we insert the line

```
#include "iostm8l151k6.h"
```

(who works on Discovery — writes iostm8l151 **with** 6). Or does not write — it will work anyway, because the MCs are almost identical :)

For the sake of idle curiosity, you can right-click on the file name in the code and select **Open "iostm8l151k6.h"** to see what's inside. And inside are register defines, structures for accessing individual bits, masks for setting bits... in general, 300 KB of such crap:

```
/* Port A data output latch register */
#ifdef __IAR_SYSTEMS_ICC__
typedef struct
{
  unsigned char ODR0        : 1;
  unsigned char ODR1        : 1;
  unsigned char ODR2        : 1;
  unsigned char ODR3        : 1;
  unsigned char ODR4        : 1;
  unsigned char ODR5        : 1;
  unsigned char ODR6        : 1;
  unsigned char ODR7        : 1;
} __BITS_PA_ODR;
#endif
__IO_REG8_BIT(PA_ODR,      0x5000, __READ_WRITE, __BITS_PA_ODR);
```

This is a description of the structure for accessing the ODR register of port A (for those familiar with AVR, it is the same as the PORT register for them).
You can access it like this:

```
PA_ODR = 0x42;
```

Or, to individual bits, like this:

```
PA_ODR_bit.ODR2 = 1;
```

And also compare (although the ODR register is a bad example here - it is usually written to, not read):

```
if (PA_ODR_bit.ODR2 == 1)
```

— Now, perhaps, we can start coding? — An impatient reader will ask
. — Yes, we can. — I will answer. — But before that, we can do one clever trick.

Judge for yourself — we will have to create a whole bunch of projects for this MK during the course (and later). And the settings that we have just changed will also have to be set for each of them. It is inconvenient. Maybe this can be simplified somehow? It is possible: **create your own project template** with a header file and settings.

Go to the folder “C:\Program Files\IAR Systems\Embedded Workbench 6.0 Kickstart\stm8\config\template\project” — IAR stores project templates there. In it, create a subfolder and call it, for example, 8L_examples (this name will not be displayed in the list of templates, so it is not important). Now we go to the folder with our project, take the main.c and .ewp files (it has the same name as the project) from there, and throw them into the folder with the template. Rename the .ewp file to templproj.ewp.

Now we return to the folder where the folders with templates are. There is a file C.projtempl there. We make a copy of it and call it the same as the folder with our new template (8L_examples). Open the copy with some notepad. Inside, you only need to change the name of the template and its description. For example, like this:

```
<?xml version="1.0" encoding="iso-8859-1"?>

<template>
  <displayname>8L-examples</displayname>
    <description>STM8L example template (152K6)
    </description>
  <files>
    <file>$PROJ_DIR$\main.c</file>
  </files>
</template>
```

Save. That's it, you can create a project with all the settings and the included header file in one click.

Now you can start writing the code. Or rather, since we don't know anything yet, but we really want to check the work, we can start copying the code:

```

#include "iostm8l151k6.h"

//Задержка на цикле. Выбрана на глаз и равна примерно половине секунды.
void SomeDelay()
{
  for (unsigned long delay_count=0; delay_count<300000; delay_count++);
};

int main( void )
{
  PC_DDR_bit.DDR7 = 1; //Настраиваем 7й пин порта C на выход
  PC_CR1_bit.C17 = 1; //Переключаем его в режим push-pull (это когда он 
  //и низкий и высокий уровень), а то по-умолчанию он прикидывается пино
  //(это когда может выдавать только низкий уровень, а вместо высокого п

  while (1) //В цикле будем мигать светодиодом
  {
   PC_ODR_bit.ODR7 = 1; //Переключаем пин в высокий уровень - светодиод
   SomeDelay();  //Задержка в 0.5 сек
   PC_ODR_bit.ODR7 = 0; //Пин в низкий уровень - светик тухнет
   SomeDelay();
  };

  return 0;
}
```

By the way, an attentive reader has already noticed (and others will notice after flashing) that **there is no pin C7 on the STM8L module for PINBOARD2** . Therefore, nothing will blink there. It was not me who was greedy when laying out the layout, but STM when designing the MK in the LQFP32 case. So, you will have to correct the program for blinking with another pin. For example, D7 (which is connected to the on-board LED with a jumper). Then you need to change all calls to the registers of port C to the registers of port D. In short, the first line of main() will now look like this:

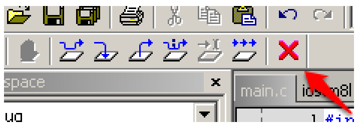
```
PD_DDR_bit.DDR7 = 1;
```

Well, and so on in the same vein.

Let's try to flash it?

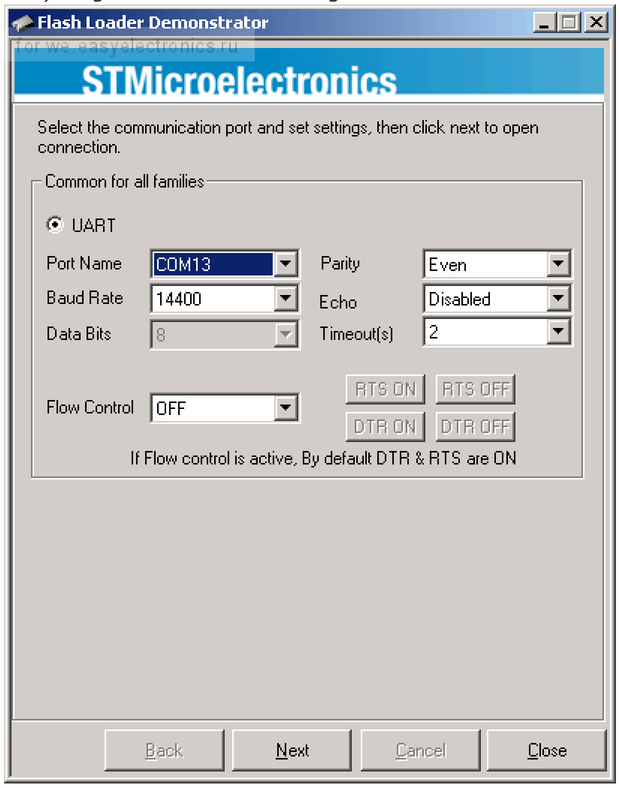
If you are working via **ST-Link** , then simply press **Ctrl+D** (or select Project -> Download and Debug in the menu). Before that, of course, you need to connect the ST-Link correctly (if you are working with a Pinboard MK). The program will be loaded into the MK memory, the debugger will start, and will stop at the first line of the main() function. Now you can step through the code with the **F7** button , watching how the LED on the board lights up and then goes out. Or you can **run the code with the F5** key .

After you have played with debugging, you can finish it by pressing the button with a cross.



After debugging is complete, the MK will reboot and continue to work . This is a note for you, in case you ever make a device with some tricky initialization at startup (setting servomotors to the initial position or something like that) - immediately after closing the debugger, the initialization will happen again. Sometimes it is annoying. Therefore, you either need to press RESET to the ground so that the MC does not start up again, or make a cycle at the very beginning of the program with the expectation of some event (pressing a button, for example), and only after it - initialization and other code.

For those who use a bootloader for firmware, everything will also be very simple: launch **FlashLoader Demonstrator** (here you can get a "hacked" one). It greets us with a settings window:



Flash Loader Demonstrator

STMicroelectronics

Select the communication port and set settings, then click next to open connection.

Common for all families

UART

Port Name: COM13
Baud Rate: 14400
Data Bits: 8
Flow Control: OFF
Parity: Even
Echo: Disabled
Timeout(s): 2

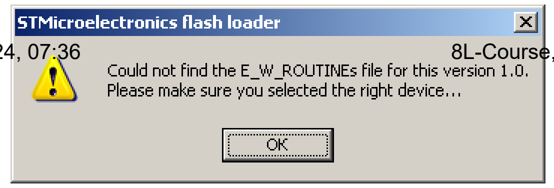
RTS ON RTS OFF DTR ON DTR OFF

If Flow control is active, By default DTR & RTS are ON

Back Next Cancel Close

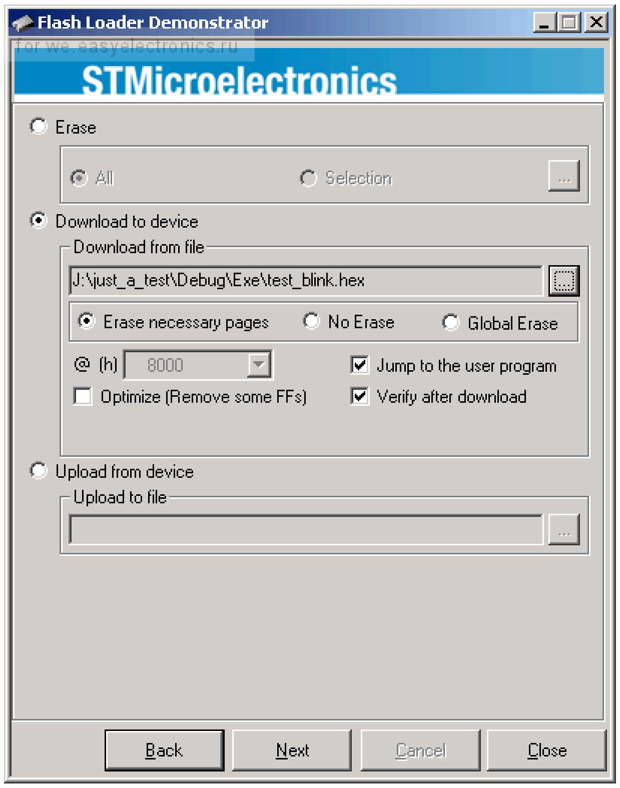
Here we select the COM port, and leave the rest as is. The bootloader, by the way, **determines the required speed itself** - cool, right?

Press next and, if everything is ok, the program will complain that it cannot find the file with the memory layout for our MK.


STMicroelectronics flash loader

Could not find the E_W_ROUTINEs file for this version 1.0.
Please make sure you selected the right device...

Select it yourself from the list above - **STM8L_32K** . Press next. Next, the firmware offers us to select the desired action: Check **the Download to device**


Flash Loader Demonstrator

( ) Erase
(● All) ( ) Selection

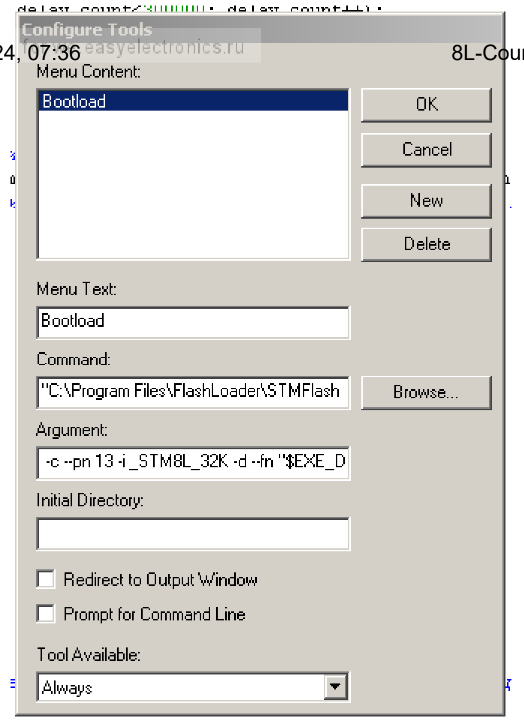
(● Download to device)
Download from file
J:\just_a_test\Debug\Exe\test_blink.hex
(● Erase necessary pages) ( ) No Erase ( ) Global Erase
@ (h) 8000
[✓] Jump to the user program
[ ] Optimize (Remove some FFs)
[✓] Verify after download

( ) Upload from device
Upload to file

box and select the hex file that IAR issued for uploading (I remind you, it is located in the project folder in /Debug/Exe). Press next and watch the firmware progress. After the firmware is complete, the bootloader transfers control to the main program and the LED on the board will start blinking. Victory! But in addition to the cute and friendly graphical interface, we have **a harsh console firmware** . And it can **be attached to IAR** so that the firmware occurs with one click of a button. The console utility is in the same folder as everything else. It is called " **STMFlashLoader.exe**

" (and the graphic one is STMicroelectronics flash loader.exe). Nearby, in the Doc folder, there is a file UM0462.pdf in which you can find a description of all the commands that the flasher understands. Therefore, I will not go into detail here.

In order to attach it to IAR, go to the **Tools -> Configure Tools** menu . There, click " **New** " and fill in the fields:



Menu Text: Bootload

Command: "C:\Program Files\FlashLoader\STMFlash

Argument: -c --pn 13 -i _STM8L_32K -d --fn "$EXE_D

Initial Directory:

☐ Redirect to Output Window

☐ Prompt for Command Line

Tool Available: Always

Menu Text is just a name that will be displayed in the menu and it does not affect anything.

In the **Command** field , write the full path to our console flasher. Mine is in "C:\Program Files\FlashLoader\STMFlashLoader.exe"

In the **Argument** field , write

```
-c --pn 13 -i _STM8L_32K -d --fn "$EXE_DIR$\$PROJ_FNAME$.hex" -r --a 800
```

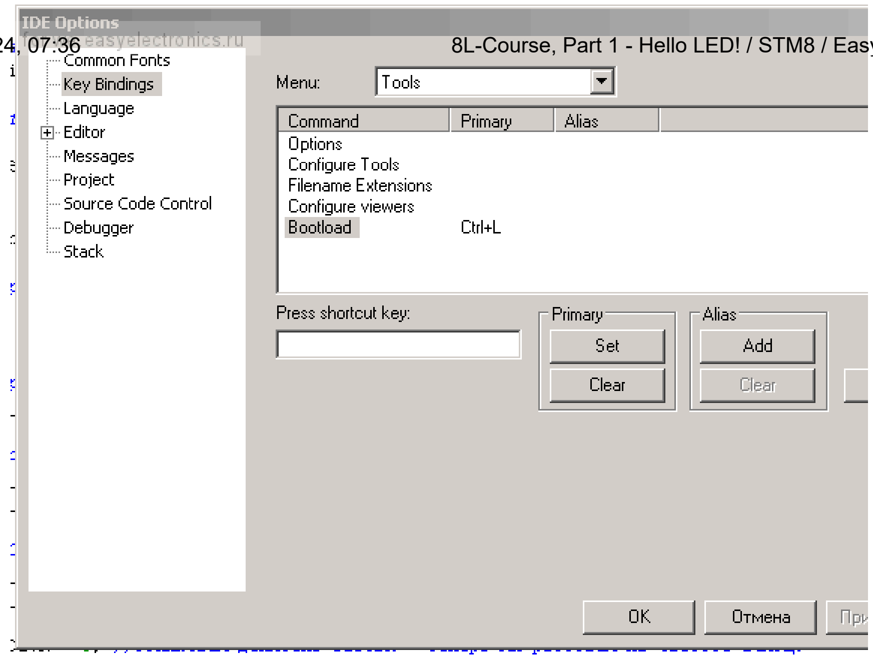
The numbers after --pn are the COM port number used to communicate with the bootloader, yours will most likely be different.
What follows --fn is the path to the hex file. Instead of EXE_DIR, IAR will substitute the folder with the project executable files, and instead of $PROJ_FNAME$, the project name. Yes, if for some reason you gave it a different name when setting up the hex file generation, then you will need to fix it here. Otherwise, leave it as is.

-r --a 8000 means starting the firmware from address 0x8000 (this is the beginning of the program memory). If you remove this key, the firmware will start only after resetting the MC. In some cases, this can be convenient.

There is one problem with console flashing - after the firmware is completed and launched, the program does not exit, but hangs and persistently asks for "press any key". And it will be more convenient for us if it closes itself. What to do? You can fix the source code (they are in the Src folder), but I didn't have Visual Studio, so I had to look for a particularly tricky method. And I found it. The program handles the **-v** parameter , but the handler is empty, and if the parameter is at the very end, the program ends without any questions. That's the story of the last key in the line :)

Now click OK and go to the tools menu: the bootload item has appeared there. If you click it, the console firmware will start and upload the firmware to the MK.

To make it even more convenient, you can assign a key combination to the firmware command. To do this, go to **tools -> options -> Key bindings** . There, from the menu list, select Tools and write a hotkey for our command:



Command	Primary	Alias
Options		
Configure Tools		
Filename Extensions		
Configure viewers		
Bootload	Ctrl+L	

With such a console gadget and a hack from Vga, the firmware of the stone is done without any extra movements, with one click of a button. And that's cool!

Well, that's all for today. Next time we'll figure out what we've just done and how it all works — there will be a GPIO analysis. Stay tuned!

IAR project with our flasher

← Part 0 — Start Contents Part 2 — GPIO →

STM8 , STM8L , IAR

+7 December 11, 2012, 16:56 **dcoder** 1

Files in topic: 1_FirstSteps.zip

Comments (37)

RSS Collapse / Expand

Here..., will there ever be a course like Hello TFT panel... -1

Valio
December 11, 2012, 20:26

yes yes and without its own controller - to form an image through dma. and with a controller it is not interesting - the nogodryg is the same everywhere 0

FogBRD
December 11, 2012, 20:57 ↑

So you want a course "LCD matrix controller for dummies"
Have you ever torn off the controller board from the TFT matrix? It has dozens of pins on several sides, and hundreds on the larger ones. And they are also applied directly to the glass. How do you suggest connecting to them?:) 0

"And the next step will be to connect 90*120 contacts to our wires"

Ozzie
December 11, 2012, 10:27 PM ↑

It is necessary to assume that FogBRD does not mean a bare TFT matrix (there are no such), but with drivers with a MIPI DPI interface (these are usually found in all kinds of mobile devices, such as tablets). Unlike the controller, the drivers only have an interface converter and a glass driver, so they need to be 0 fed with data at a frequency of Width * Height * Framerate (and even a little more). Usually, a separate controller (TCON) does this; in the case of mobile devices it is usually built into the SoC as one of the output interfaces of the built-in video adapter).

Vga
December 11, 2012, 10:36 PM

Absolutely right.
The article series is designed to teach how to use resources, I understand.
Well, they should have shown how to use DMA.

0

FogBRD
December 12, 2012, 4:47 PM

As far as I know, there are simply no resources there. The only thing that can handle a TFT panel at an acceptable speed is STM32 with FSMC. The cheap STM32F100C4 was also forced to work on the panel, but there were big limitations, and it gave out about ten frames per second at 320x240, and most panels want 60Hz and simply do not work with insufficient refresh rate.

0

Vga
December 12, 2012, 5:53 PM

There are plenty of mobile phone screens with a good diagonal. They are cheap, but I would like to use them.
Well, they turned the atmega onto a 320x240 panel. Why won't this stm8 turn it. Especially older models with a clock speed of 24 MHz

0

FogBRD
December 12, 2012, 21:41

Mobile phone screens usually have brains. And give me a link to the project where the atmega spins a brainless TFT panel.

0

Vga
December 12, 2012, 21:45

a 320x240 5" panel costs 12 USD. I understand there is no alternative, with such a diagonal you have to have brains.
www.sat.cc.ua/myavr.html

0

FogBRD
December 12, 2012, 21:52

Ah, there is external memory and a processor with its support. Among STMs, this is STM32 with FSMC. You can run a display on them. On the others, the problem is that the memory buses are only inside, and accordingly, DMA works only inside the crystal. And there is simply not enough RAM to store the frame buffer.

0

Vga
December 12, 2012, 10:21 PM

Thanks for the article.

How is the "hacked" FlashLoader Demonstrator different from the regular one? It's supposedly free anyway?
And why didn't the IAR optimizer throw away the empty loop ?

```
for(unsigned long delay_count=0; delay_count<300000; delay_count++);
```

Features of the optimizer or is optimization simply disabled in your project?

0

e_mc2
December 11, 2012, 10:08 PM

Read the article. It has answers to both of your questions.

0

angel5a
December 11, 2012, 10:16 PM ↑

Oops, got it.

> *Vga has revealed a working hack.*

I thought it was about hacking as finding an original solution to a problem, and FlashLoader always behaves like that. I didn't realize that Vga actually modified the program.
And regarding the optimization options - I really screwed up.

0

e_mc2
December 11, 2012, 10:37 PM ↑

> *How is the "hacked" FlashLoader Demonstrator different from the regular one? It's supposedly free, isn't it?*

Reset pulls before sewing. "Hacked" because this functionality had to be added through the ass (ST stole the sources and GUI, and actually the libraries for communicating with the boot, there are only sources for the console utility and wrapper to the library).
Also dcoder already wrote about this "hacked FlashLoader".

+1

Vga
December 11, 2012, 10:23 PM ↑

> *Also dcoder already wrote about this "hacked FlashLoader".*

I can't find anything (local search doesn't find anything, and dcoder has a lot of flocks). If it's not too much trouble, please give me a link.

Definitely respect to you.

How did you make the changes? Disassembling the code and a binary patch?

0

e_mc2
December 11, 2012, 11:40 PM ↑

> *I can't find anything (local search doesn't find anything, and dcoder has a lot of flocks).*

Here . However, I also had to search by browsing through his articles) Although Lifelover recommends using Google("%searchword% site:we.easyelectronics.ru").

> *How did you make the changes? Disassembling the code and a binary patch?*

No, it's much simpler. For some reason, there is a wrapper library around the compiled boot client library, it consists of functions like wrapfoo(args){return foo(args);}. There I chose a couple of suitable functions and added the pin twitching code to them. By the way, somewhere in the dll there is a structure of four boolean values, preceded by a marker string. There you can use the hexeditor to choose how the function is intercepted, which pin and in which direction to twitch)

```
#ifdef RESET_MCU_HACK
const struct
{
        char MARKER[21];
        char UseRTS;
        char Invert;
        char HackComOpen;
        char HackInitBL;
} resetMCUHackOptions = {{'R', 'E', 'S', 'E', 'T', '_', 'M', '(
#endif
```

+1

Vga
December 12, 2012, 03:34 ↑

+2

By the way, about the console flasher. It can be used with an unmodified dll (or disable the hack in the modified one by zeroing out 4 bytes after RESET_MCU_HACK_MARKER with the hexadecitor). There are RTS/DTS control commands and a pause command (in the one I modified, there is no pause in the original). So you can add -Dts --Hi -Pause 200 -Dts --Lo -Pause 100 to the beginning of the command line (or the same thing, but swap Hi and Lo, I don't remember, you can also use -Rts instead of -Dts).

Vga
December 12, 2012, 03:44

I got IAR Embedded Workbench for ARM 6.40.2.3992 (CD-EWARM-6402-3992) from here

0

PipEtS
December 12, 2012, 19:53

ex.ua seems to work only with Ukrainian IPs, needs to be checked.

0

PipEtS
December 12, 2012, 19:55 ↑

What's the point of IAR for ARM?

0

Vga
December 12, 2012, 19:58 ↑

Regarding ST-Link, I did not find any mention of STM32VL-Discovery. These boards were sent out en masse, many people have them. Is it possible to combine STM32VL-Discovery + PINBOARD2?

0

anakost
December 15, 2012, 12:59

ST-LINKs on Discovery are cut down and support only the family installed on the board. So STM32VL DISCOVERY with PB2 can be used, but only with the STM32 processor kit, instead of the native CoLink debugger for PB2. And for the STM8 processor kit, only DISCOVERY of the STM8 family are suitable - today these are STM8S, STM8SVL, STM8L. The first two options are more convenient, but the third one has a glass in the kit.

0

Vga
December 15, 2012, 14:31 ↑

I dug around on the internet and found this on the forum: Your text to link...
In short - " **STM8 needs SWIM, but here it's SWD** ".

0

anakost
December 15, 2012, 15:08

The gist is the same. On STM8*-DISCOVERY JTAG (aka SWD) is locked, on STM32*-DESCOVERY SWIM is locked and only in the standalone ST-LINK both interfaces are unlocked.

0

Vga
December 15, 2012, 15:25 ↑

Let me correct you: on STM8 - SWIM and on STM32 - SWD. Besides, JTAG and SWIM/SWD are completely different interfaces.

0

kusovn
07 April 2013, 01:11 ↑

SWD, as applied to ARMs, is a two-wire version of JTAG, and ST-LINK itself in the STM32 version is called ST-LINK JTAG, even if only SWD is output.

> *Let me correct you: on STM8 it's SWIM and on STM32 it's SWD.*

Read it more carefully. I said that on STM8-DISCOVERY JTAG *is locked* . That is, only SWIM works.

0

Vga
07 April 2013, 02:33 ↑

0

I'm terribly sorry, I screwed up a bit)

kusovn
08 April 2013, 09:16 ↑

By the way, if you read the topic more carefully, there is a funny moment - according to STM, STM8S-DISCOVERY can program only STM8S, but not STM8L. However, there are only three ST-LINK firmware - ST-LINK JTAG, ST-LINK SWIM (this is the firmware on all STM8 Discoveries) and ST-LINK JTAG/SWIM. Apparently, STM assumes that ST-LINKs on Discoveries have thrown out the level translator and therefore it does not work with MCU series designed for other voltages. 0

Vga
08 April 2013, 11:28 ↑

I don't know about the 8-k, of which I only have the STM8L-DISCOVERY and haven't tried to flash other MCs from this series, but I connected the F103 chip to the STM32F4-DISCOVERY, it sees it and flashes it normally. 0

kusovn
08 April 2013, 18:05 ↑

But JTAG and ST-LINK don't have any matchers. Apparently, they all work from 3.3V. But STM8 of different series - from 1.65 to 5.5V, so the full-fledged one has a 74LVC2T45 buffer (or something like that). 0

Vga
08 April 2013, 19:53 ↑

Where in IAR in simulator mode (without hardware) can I see the number of cycles, execution time and other real characteristics of the project? 0

Chapa
March 24, 2013, 21:51

In the registers window (View/Registers, select "CPU Registers" from the list) there are variables:
- CYCLECOUNTER - the number of cycles since the reset;
- CCSTEP - the number of cycles for the last action (depends on the mode and the action itself);
- CCTIMER1 and 2 - just cycle counters that the user can reset themselves (can be used to count intervals, etc.). 0

SerjT
08 April 2013, 14:30 ↑

The last timers can not only be reset, but also have their own values set. 0

SerjT
08 April 2013, 14:33 ↑

Something I didn't understand: for new VL microcontrollers such as STM8S003, for example, there is no iostm8s003x.h file with directly filled in defines like PB_ODR? 0

They suggest including the necessary controller in the stm8s.h file to connect the corresponding defines, but they already use SPL, and there everything is done through structures: instead of PB_ODR, you have to do it through GPIOB->ODR.

Maybe there is this iostm8s003x.h file after all? I downloaded examples for stm3sVldiscovery and looked in the Cosmic and STVD folders - there are some for other families, but not for s00x.

maxgrind
August 26, 2013, 5:58 PM

There, all the addresses seem to match with 103. At least GPIO, UART, timers, clock system - that's what I checked in practice. 0

Katz
August 28, 2013, 09:47 ↑

(upd) Everything is the same there. For example, here is the header content for STM8S003F3 from the latest version of STVD:

```
/* STM8S003F3.h */
#ifdef MCU_NAME
#define STM8S003F3 1
#endif
#include "STM8S103F.h"
```

0

Katz
August 29, 2013, 09:59 ↑

Yes. It's true that it's in \STMicroelectronics\st_toolset\include\
I don't know how I missed it. Maybe after I installed the new version of STVD and new headers appeared there.

0

maxgrind
August 29, 2013, 12:39 ↑

Only registered and authorized users can leave comments.