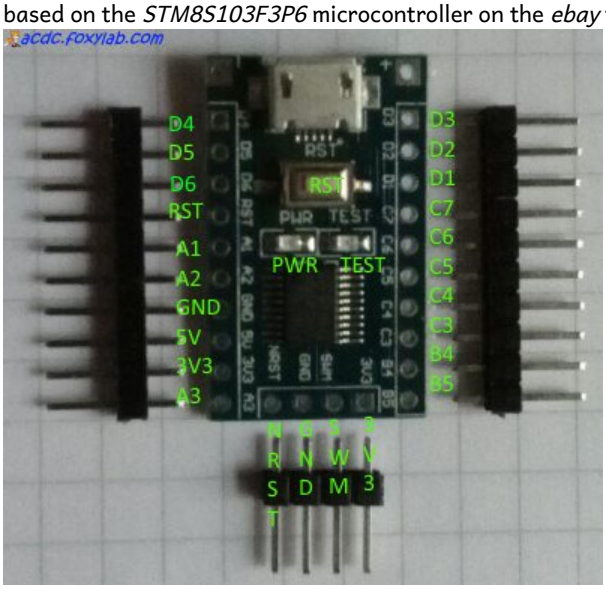
STM8 - programming, firmware and everything

Along with *Arduino , the line of products from STMicroelectronics* is also popular among microcontroller enthusiasts , including 8-bit *STM8* microcontrollers and 32-bit *STM32* microcontrollers (based on the *Cortex* core).

STM8 microcontrollers consist of several lines:
STM8S **- the main line** ,
STM8A - for the automotive industry,
STM8L - with ultra-low power consumption,
STM8T - a capacitive sensor for detecting touch or proximity.

A debug board with an *STM8* microcontroller on board can be purchased for 1 (!) dollar or even cheaper. I bought several such boards based on the *STM8S103F3P6* microcontroller on the *ebay* trading platform :



contact	appointment
D4	PD4/UART_CLK
D5	PD5/TX
D6	PD6/RX
RST	reset
A1	PA1 / Oscin
A2	PA2 / Oscin
GND	Earth
5V	stabilizer input
3V3	stabilizer output
A3	PA3/SS
D3	PD3/Ain4
D2	PD2/Ain3
D1	PD1/SWIM
C7	PC7 / MISO

C6	PC6/MOSI
C5	PC5/SCK
C4	PC4/Ain2
C3	PC3
B4	PB4/SCL (I2C bus)
B5	PB5 / SDA (I2C bus)

Here is an alternative representation of the microcontroller pinout:

```
                            -----------
UART1_CK / TIM2_CH1 / PD4 | 1 20 | PD3/AIN4/TIM2_CH2/ADC_ETR
    UART1_TX / AIN5 / PD5 | 2 19 | PD2/AIN3
    UART1_RX / AIN6 / PD6 | 3 18 | PD1/SWIM
                     NRST | 4 17 | PC7 / SPI_MISO
              OSCIN/PA1 | 5 16 | PC6/SPI_MOSI
             OSCOUT/PA2 | 6 15 | PC5/SPI_CLK
                Vss(GND) | 7 14 | PC4 / TIM1_CH4 / CLK_CCO / AIN2
                VCAP (*1) | 8 13 | PC3/TIM1_CH3/
                Vdd (+Ub) | 9 12 | PB4/I2C_SCL
           TIM2_CH3/PA3 | 10 11 | PB5/I2C_SDA
                            -----------
```

The STM8S103F3P6 microcontroller contains 8 KB of flash memory with a 10,000 erase resource, 640 bytes of *EEPROM* and 1 KB *of RAM* . The clock frequency of the 8-bit *STM8S* series processor is 16 MHz.

The following options can be used to power the board:

- connecting a 4.5...15 V voltage source to the **+** or **5V** and **-** or **GND** contacts ;
- connecting the cable to *the microUSB* connector (this connector is used only for power supply!);
- connecting a 3.3V power source to **the 3V3** and **-** or **GND** contacts .

The board has a stabilizer *AMS1117-3.3* . The stabilizer input is connected to the **5V** contact , and the output is connected to the **3V3** contact .

Using the SDCC compiler

Installing the compiler

When developing for *STM8* microcontrollers , you can use the open (under the *GPLv3* license) *SDCC (Small Device C Compiler suite) compiler suite of the ANSI C* programming language for many architectures - from *Intel 8051* to *STMicroelectronics STM8* . Versions for various OS - *Windows* , *MacOS* , *Linux* - (the current latest stable version is 4.4.0) are available at http://sdcc.sourceforge.net .

For Linux OS, download the latest version (for example, *sdcc-4.4.0-rc2-amd64-unknown-linux2.5.tar.bz2*), unpack and copy the contents of the resulting folder to *./usr/local* .

To check, enter the command

```
sdcc -v
```

(displays the compiler version).

Program development

We write the program code in the ***TST.c*** file :

```
#include <stdint.h>

#define __IO volatile
.........................
}
```

Compiling the program

We compile the program using the command

```
sdcc -mstm8 --std-c99 TST .c
```

When compiling, *a hex* file ***TST.ihx*** is created :

```
:208000008200808382000000820000008200000082000000820000008200000082000004D
:208020008200000082000000820000008200000082000000820000008200000082000030
.....
:00000001FF
```

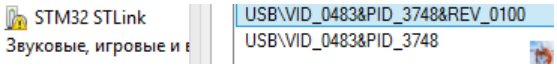
Files with the following extensions are also created:

asm
lk
lst
map
rel
rst
sym

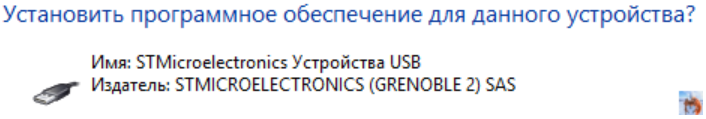
Firmware

To flash the board, I purchased the *ST-LINK V2* programmer on *the eBay* trading platform :

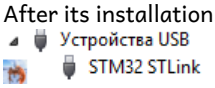
When initially connected to the computer's *USB* port, the programmer is identified as an "unknown device" with *VID* 0483 and *PID* 3748:



The driver for the programmer - *STSW-LINK009* is available on the *ST* website :



After its installation, when reconnected, the programmer is recognized as a " *USB* device ":



The programmer connector has 10 contacts:

Number	Name	Purpose
1	*RST*	reset
2	*SWIM*	*SWIM* interface (for *STM8*)
3	*GND*	Earth
4	*3.3V*	+ 3.3 V
5	*5.0V*	+ 5 V

6	*SWCLK*	synchronization (*SWD* interface, for *STM32*)
7	*SWDIO*	data (*SWD* interface, for *STM32*)
8	*GND*	Earth
9	*3.3V*	+ 3.3 V
10	*5.0V*	+ 5 V

To connect the programmer to the board, I use 4 contacts on the programmer connector and on the board - *3.3V* (*3V3)* , *SWIM* (*SWM*), *GND* , *RST* (*NRST*):

When communicating with the board, the programmer uses the *SWIM* communication protocol (via a single-wire interface - *SWIM* contact).

For flashing I use the *stm8flash* utility , to launch which you should execute the command:

```
stm8flash -c stlinkv2 -p stm8s103f3 -w TST .hex (or TST .ihx)
```

The stm8flash project is hosted on *GitHub* : https://github.com/vdudouyt/stm8flash

The binary version of the project for *Windows* OS (can be taken here) contains two necessary files:

stm8flash.exe - executable file
libusb-1.0.dll - library for accessing *USB* devices

To use the utility under *Linux* OS , you must first install the libraries:

```
sudo apt-get install pkg-config libusb-1.0-0-dev
```

Then download the stm8flash-master.zip project archive , unpack it and run it in the project folder:

for compilation

```
make
```

for installation

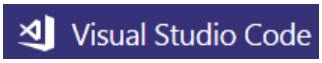
```
sudo make install
```

(the *stm8flash* file is copied to the */usr/local/bin* folder).

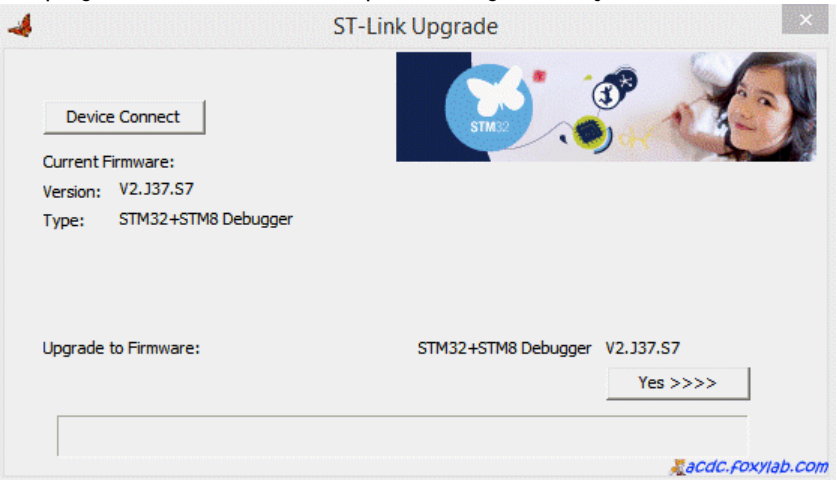
After launching the utility, it flashes the program into memory and reports the number of bytes written:

```
D:\STM8\TST>stm8flash -c stlinkv2 -p stm8s103f3 -w TST.hex
Determine FLASH area
Writing Intel hex file 261 bytes at 0x8000... OK
Bytes written: 261
```


After flashing the above LED blinking program, it starts blinking with a period of about six seconds.



The programmer firmware can be updated using the utility available for download on the *ST website* - *STSW-LINK007* :

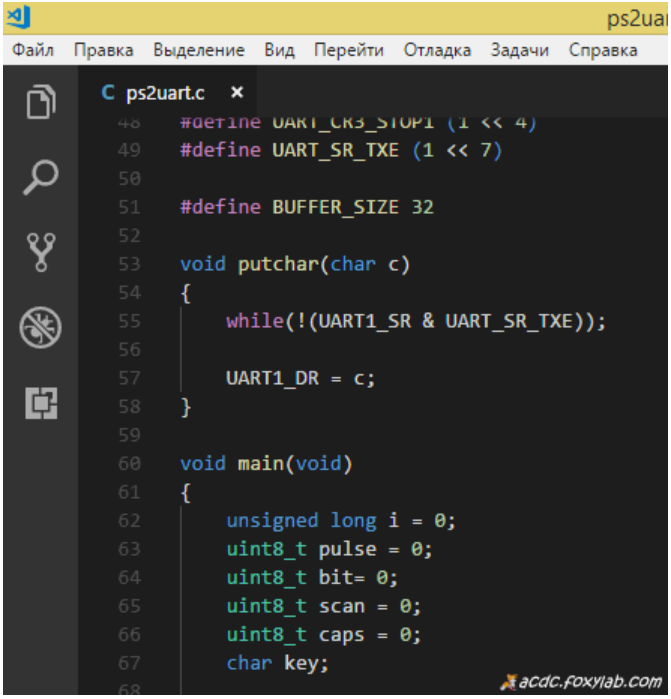


After launching the application (for *Windows* - *ST-LinkUpgrade.exe*), you need to connect the programmer to *the USB* port of the computer, press the **Device Connect** button - this will display the current version of the programmer firmware (*Version*) and its type (*Type*), as well as the version to which the firmware can be updated (*Upgrade to Firmware). To start the firmware update process, you need to press the* **Yes>>>>** button .

Working in the Visual Studio Code editor

To write the source code of the program, as well as to automate the process of compilation and firmware, it is convenient to use the free *Visual Studio Code* editor from *Microsoft* (*download page*).

Here's what the program's source code looks like in *Visual Studio Code* :



For ease of use, you should create a folder (for example, *sdcc*) in which the project files for *STM8* will be located . Then you should add this folder to the workspace. In the same folder, we place the files:

- *compile.cmd* - with contents: **c:/sdcc** /bin/sdcc -mstm8 --std-c99 %~n1.c , where **c:/sdcc** is the *SDCC compiler folder*
- *flash.cmd* - with contents: stm8flash -c stlinkv2 -p stm8s103f3 -w %1
- *stm8flash.exe*
- *libusb-1.0.dll*

In this folder, you should create *a .vscode* folder , in which you should place *a tasks.json* file with the following contents:

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "compile",
```

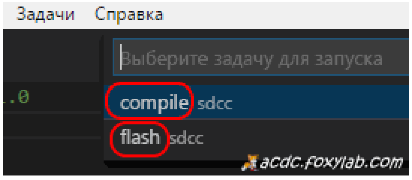
```
"type": "shell",
"command": "compile ${file}",          "problemMatcher": []       }     ] }
{fileDirname} \ ${fileBasenameNoExtension}.ihx",
```

```
{file}",
        "problemMatcher": []
    },
    {
        "label": "flash",
        "type": "shell",
        "command": "flash
```

This file describes two tasks:

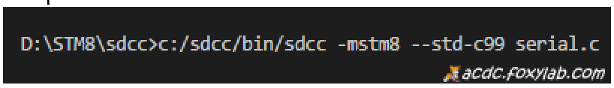
- *compile* - compiles the currently open source code file (with the *.c* extension)
- *flash* - flashing a previously compiled *hex* file (with the *.ihx* extension)

By selecting the appropriate task (" *Tasks* " > " *Run task...* ")

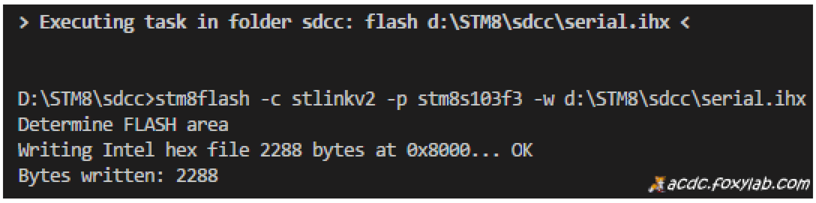


you can run:

compilation



firmware



Helpful hints

working with ports

Setting base port addresses:

port *A*

```
#define GPIOA_BaseAddress 0x5000
```

Port *B*

```
#define GPIOB_BaseAddress 0x5005
```

port *C*

```
#define GPIOC_BaseAddress 0x500A
```

Port *D*

```
#define GPIOD_BaseAddress 0x500F
```

Setting the port mode (*DDR* - port direction register):

Setting the output of port *N X* (*P **XN***) to output

```
GPIO X ->DDR |= (1 << N );
```

Setting up the output of port *N* *X* (*P* ***XN***) to input

```
GPIO X ->DDR &= ~(1 << N );
```

output (*ODR* - output data register):

setting the output *N* of port *X* (*P* ***XN***) to the state "1" :

```
GPIO X ->ODR |= (1 << N );
```

setting the output *N* of port *X* (*P* ***XN***) to the state "0" :

```
GPIO X ->ODR &= ~(1 << N );
```

Including header files

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
```

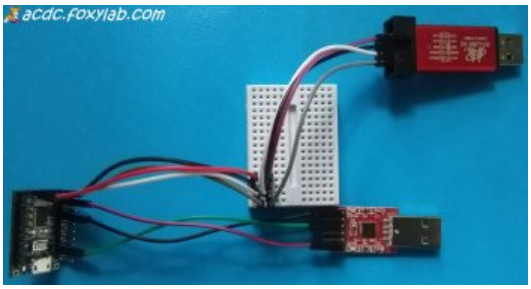
delays

```
// constants for CLK #define CLK_DIVR (*(volatile uint8_t *)0x50c6)
...
static void delay(uint32_t t)
{
   while(t--) {}
}
...
delay(44000000UL); //delay for one minute
```

working with UART

```
//constants for CLK
#define CLK_DIVR (*(volatile uint8_t *)0x50c6)
#define CLK_PCKENR1 (*(volatile uint8_t *)0x50c7)
//constants for UART
#define UART1_SR (*(volatile uint8_t *)0x5230)
#define UART1_DR (*(volatile uint8_t *)0x5231)
#define UART1_BRR1 (*(volatile uint8_t *)0x5232)
#define UART1_BRR2 (*(volatile uint8_t *)0x5233)
#define UART1_CR2 (*(volatile uint8_t *)0x5235)
#define UART1_CR3 (*(volatile uint8_t *)0x5236)
#define UART_CR2_TEN (1 << 3)
#define UART_CR3_STOP2 (1 << 5)
#define UART_CR3_STOP1 (1 << 4)
#define UART_SR_TXE (1 << 7)
...
void putchar(char c) // output character to UART
{
   while(!(UART1_SR & UART_SR_TXE));
   UART1_DR = c;
}
...
CLK_DIVR = 0x00; //setting the clock frequency to 16 MHz
CLK_PCKENR1 = 0xFF; //enable peripherals
UART1_CR2 = UART_CR2_TEN; //TX and RX enable
UART1_CR3 &= ~(UART_CR3_STOP1 | UART_CR3_STOP2); //1 stop bit
UART1_BRR2 = 0x03; UART1_BRR1 = 0x68; //9600 baud
...
printf("Hello,world!\r\n"); //output string to UART
```

This is what the assembled circuit looks like from a microcontroller, *USB-UART* converter and programmer:



working with a timer

```
//timer registers
#define TIM1_CR1 (*(volatile uint8_t *)0x5250)
#define TIM1_IER (*(volatile uint8_t *)0x5254)
#define TIM1_SR1 (*(volatile uint8_t *)0x5255)
#define TIM1_CNTRH (*(volatile uint8_t *)0x525E)
#define TIM1_CNTRL (*(volatile uint8_t *)0x525F)
#define TIM1_PSCRH (*(volatile uint8_t *)0x5260)
#define TIM1_PSCRL (*(volatile uint8_t *)0x5261)
...
TIM1_PSCRH = 0x09; //setting up the timer prescaler
TIM1_PSCRL = 0x89;
TIM1_CR1 = 0x01; //turn on the timer
TIM1_IER = 0x01; //enable timer interrupts
__asm__ ("rim"); //enable interrupts
...
//timer interrupt handler
void TIM1_overflow_Handler() __interrupt(11)
{
    TIM1_SR1 &= ~1; //reset interrupt flag
    //performing the required actions
}
```

(1/16000000)*65536*prescaler = interval in seconds
10 seconds = 2441 0x0989

working with ADC

```
typedef struct ADC1_struct
 {
  __IO uint8_t DB0RH; /*!< ADC1 Data Buffer Register (MSB) */
  __IO uint8_t DB0RL; /*!< ADC1 Data Buffer Register (LSB) */
  __IO uint8_t DB1RH; /*!< ADC1 Data Buffer Register (MSB) */
  __IO uint8_t DB1RL; /*!< ADC1 Data Buffer Register (LSB) */
  __IO uint8_t DB2RH; /*!< ADC1 Data Buffer Register (MSB) */
  __IO uint8_t DB2RL; /*!< ADC1 Data Buffer Register (LSB) */
  __IO uint8_t DB3RH; /*!< ADC1 Data Buffer Register (MSB) */
  __IO uint8_t DB3RL; /*!< ADC1 Data Buffer Register (LSB) */
  __IO uint8_t DB4RH; /*!< ADC1 Data Buffer Register (MSB) */
  __IO uint8_t DB4RL; /*!< ADC1 Data Buffer Register (LSB) */
  __IO uint8_t DB5RH; /*!< ADC1 Data Buffer Register (MSB) */
  __IO uint8_t DB5RL; /*!< ADC1 Data Buffer Register (LSB) */
  __IO uint8_t DB6RH; /*!< ADC1 Data Buffer Register (MSB) */
  __IO uint8_t DB6RL; /*!< ADC1 Data Buffer Register (LSB) */
  __IO uint8_t DB7RH; /*!< ADC1 Data Buffer Register (MSB) */
  __IO uint8_t DB7RL; /*!< ADC1 Data Buffer Register (LSB) */
  __IO uint8_t DB8RH; /*!< ADC1 Data Buffer Register (MSB) */
  __IO uint8_t DB8RL; /*!< ADC1 Data Buffer Register (LSB) */
  __IO uint8_t DB9RH; /*!< ADC1 Data Buffer Register (MSB) */
  __IO uint8_t DB9RL; /*!< ADC1 Data Buffer Register (LSB) */
  uint8_t RESERVED[12]; /*!< Reserved byte */
  __IO uint8_t CSR; /*!< ADC1 control status register */
  __IO uint8_t CR1; /*!< ADC1 configuration register 1 */
  __IO uint8_t CR2; /*!< ADC1 configuration register 2 */
  __IO uint8_t CR3; /*!< ADC1 configuration register 3 */
  __IO uint8_t DRH; /*!< ADC1 Data high */
```

```
  __IO uint8_t DRL; /*!< ADC1 Data low */
  __IO uint8_t TDRH; /*!< ADC1 Schmitt trigger disable register high */
  __IO uint8_t TDRL; /*!< ADC1 Schmitt trigger disable register low */
  __IO uint8_t HTRH; /*!< ADC1 high threshold register High*/
  __IO uint8_t HTRL; /*!< ADC1 high threshold register Low*/
  __IO uint8_t LTRH; /*!< ADC1 low threshold register high */
  __IO uint8_t LTRL; /*!< ADC1 low threshold register low */
  __IO uint8_t AWSRH; /*!< ADC1 watchdog status register high */
  __IO uint8_t AWSRL; /*!< ADC1 watchdog status register low */
  __IO uint8_t AWCRH; /*!< ADC1 watchdog control register high */
  __IO uint8_t AWCRL; /*!< ADC1 watchdog control register low */
 }
 ADC1_TypeDef;
#define ADC1_BaseAddress 0x53E0
#define ADC1 ((ADC1_TypeDef *) ADC1_BaseAddress)
...
//reading data from ADC
unsigned int val=0;
ADC1->CSR |= ((0x0F) channel ); //channel selection
ADC1->CR2 |= (1<<3); //data is right aligned
ADC1->CR1 |= (1<<0); //enable ADC
ADC1->CR1 |= (1<<0); //start conversion
while(((ADC1->CSR)&(1<<7))== 0); //waiting for conversion to complete
val |= (unsigned int)ADC1->DRL;
val |= (unsigned int)ADC1->DRH<<8;
ADC1->CR1 &= ~(1<<0); //stop conversion
val &= 0x03ff; //result
```

It is necessary to set the ADC channel number corresponding to the input used (for example, pin *D2* - channel 3).

The voltage can be determined by multiplying the value read from the ADC by *VCC* /1023, where *VCC* is the supply voltage on the *3.3V* bus .
For example, when connecting the 3.3V output of the USB-UART converter to the 3.3V STM input , *the* voltage *on* it *was* 3.24 V. In this case, the scaling factor is 3.24/1023 = 0.00317 V.

Example of a simple project that blinks an LED

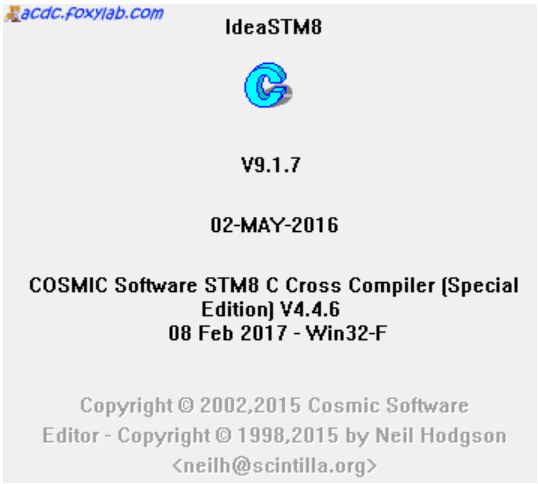
```
#include <stdint.h>
#include <stdio.h>

#define CLK_DIVR (*(volatile uint8_t *)0x50c6)
#define CLK_PCKENR1 (*(volatile uint8_t *)0x50c7)

#define __IO volatile

typedef struct GPIO_struct
{
  __IO uint8_t ODR;
  __IO uint8_t IDR;
  __IO uint8_t DDR;
  __IO uint8_t CR1;
  __IO uint8_t CR2;
}
GPIO_TypeDef;

#define GPIOB_BaseAddress 0x5005
#define GPIOB ((GPIO_TypeDef *) GPIOB_BaseAddress)


#define TIM1_CR1 (*(volatile uint8_t *)0x5250)
#define TIM1_IER (*(volatile uint8_t *)0x5254)
#define TIM1_SR1 (*(volatile uint8_t *)0x5255)
#define TIM1_CNTRH (*(volatile uint8_t *)0x525E)
#define TIM1_CNTRL (*(volatile uint8_t *)0x525F)
#define TIM1_PSCRH (*(volatile uint8_t *)0x5260)
#define TIM1_PSCRL (*(volatile uint8_t *)0x5261)

volatile uint8_t led = 0;

void TIM1_overflow_Handler() __interrupt(11)
```

```
{
    TIM1_SR1 &= ~1;
    if (led == 1) {
      GPIOB->ODR |= (1 << 5);
    }
    else
    {
      GPIOB->ODR &= ~(1 << 5);
    }
    led ^= 1;
}


void main(void)
{
   CLK_DIVR = 0x00;
   CLK_PCKENR1 = 0xFF;

   GPIOB->DDR |= (1 << 5);
   GPIOB->ODR |= (1 << 5);

   TIM1_PSCRH = 0x00;
   TIM1_PSCRL = 0xF4;
   TIM1_CR1 = 0x01;
   TIM1_IER = 0x01;
   __asm__ ("rim");
   while(1)
   {
      __asm__("WFI");
   }
}
```

Development in IdeaSTM8 environment

Setting up the development environment

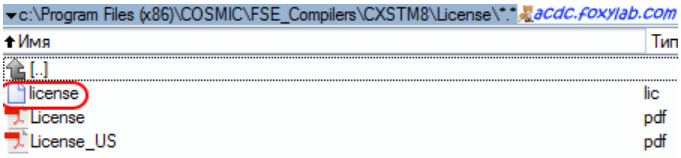
For programming for *STM8* microcontrollers , you can use *the IdeaSTM8* development environment from *Cosmic Software* (in the *CXSTM8 special edition package* version - available since March 2016, has no restrictions):



To download the distribution, follow this link: http://cosmicsoftware.com/download_stm8_32k.php .

In order to use the cross-compiler from *Cosmic Software,* you must register before downloading, specifying your name (*Name*), company name (*Company*), country (*Other*), email address (*E-mail*), and then clicking the " *Submit* " button to send the information.

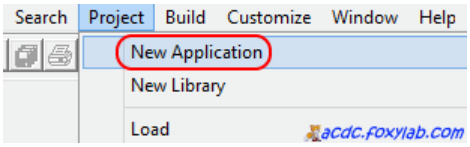
In version 4.4.6, the size of the distribution (*cxstm8_FSE_stm32_32K.exe*) is 20.7 MB. To obtain an annual (then renewable) free license, you must click the " *Register on the Web* " button during installation , which will send the license file to the email address specified during registration. The license is tied to the computer on which the compiler is installed (using *HOSTNAME* , *HOSTID* , etc.). After receiving the *license.lic* file , you should place it in the *\COSMIC\FSE_Compilers\CXSTM8\License* folder :



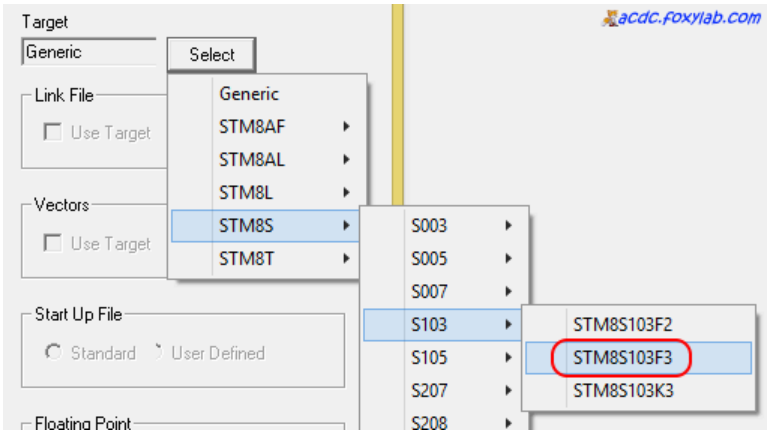
Program development

As an example, let's create a program for blinking an LED (*Hello, world!* in the world of microcontrollers) *TEST* , located on the board and connected to contact *PB.5* .

Create a new project by executing the *New Application* command :



We select the STM8S103F3 microcontroller as the target platform :

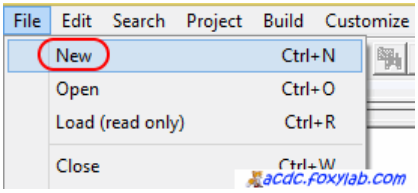


We copy the header file with definitions *stm8s.h* into the project folder , having previously uncommented the definition of the used microcontroller *STM8S103* :

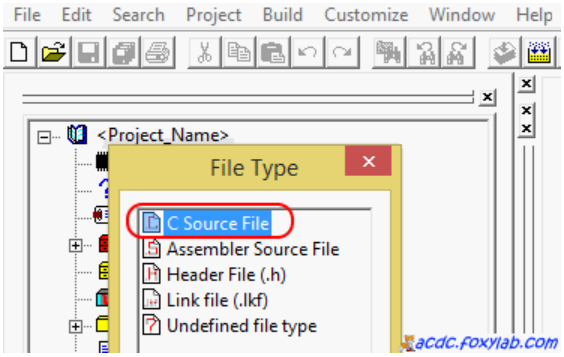
```
/* #define STM8AF626x */ /*!< STM8A Medium density devices */
  #define STM8S103 /*!< STM8S Low density devices */
/* #define STM8S903 */ /*!< STM8S Low density devices */
```

Create a new file (*tst.c*) with the source code:

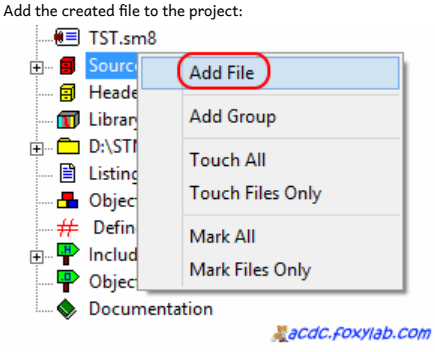
create a file:



select as the file type - file with source code in C:



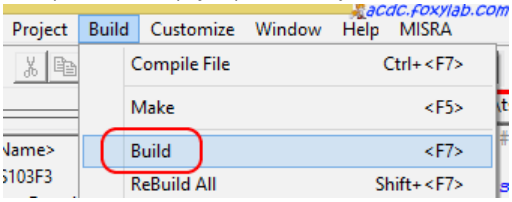
Add the created file to the project:



We write the program code in the created file:

```
#include <stm8s.h>

static void delay(uint32_t t) //delay procedure
{
  while(t--) {}
}

int main(void)
{
  GPIOB->DDR |= (1 << 5); //setting pin PB.5 to output
  GPIOB->CR1 |= (1 << 5); //setting up pin PB.5 as push-pull, can be skipped
  GPIOB->ODR |= (1 << 5); //output 1 to port
  while(1)
  {
    GPIOB->ODR |= (1 << 5); //output 1 to port
    delay(100000UL); //delay
    GPIOB->ODR &= ~(1 << 5); //output 0 to port
    delay(100000UL); //delay
  }
}
```

To compile and build the project, press the F7 key or run the *Build* command :



As a result of the build, a file with the project name and the . *sm8* extension (*TST.sm8*) is created in the project folder.

To convert a . *sm8 file into a hex* file ready for flashing into a microcontroller, I use *COSMIC Software Hexa Translator* (*chex.exe*) with the command:

```
chex -o TST .hex -fi TST .sm8    ,
```

where **TST** is the name of the project.

The resulting *hex* file (*TST.hex*) contains the information needed to flash the microcontroller:

```
:20800000820080808200000082000000820000008200000082000000820000008200000050
:20802000820000008200000082000000820000008200000082000000820000008200000030
:20804000820000008200000082000000820000008200000082000000820000008200000010
:20806000820000008200000082000000820000008200000082000000820000008200000F0
:20808000AE03FF94CD809F20FE961C0003CD80F3961C0003A601CD80C9CD80DF26EB8172FB
:2080A0001A5007721A5005721A5005AE86A089AE000189ADD45B04721B5005AE86A089AE2B
:2080C000000189ADC45B0420DE40EB03E703250EE6026A024D2607E60126017A6A01819C1F
:2080E0003D00260E3D0126083D0226043D032702A6018188F6B700E601B701E602B702E64E
:0581000003B7038481B8
:00FFFF0101
```

To flash this file into the microcontroller memory, you can use the *ST-LINK V2 programmer described above.*

My projects on STM8

PS/2 - UART Converter

The converter for the *cpm4nano* project allows you to connect *a PS/2* keyboard via *a* serial port (*UART*).

Remote controlled airboat

The STM8 microcontroller receives the IR remote control signal and controls the movement of the airboat model .

Ultraviolet radiation source from a DRL lamp

The STM8 microcontroller controls the power supply for the UV lamp .

To be continued