# Starting STM8 Microcontrollers

STM8 microcontrollers are 8-bit general purpose microcontrollers from **STMicroelectronics (STM)**. STM is famous mainly for its line of 32-bit ARM Cortex microcontrollers – the STM32s. STM8 microcontrollers are rarely discussed in that context. However, STM8 MCUs are robust and most importantly they come packed with lots of hardware features. Except for the ARM core, 32-bit architecture, performance and some minor differences, STM8s have many peripheral similarities with STM32s. In my opinion, STM8s are equally or sometimes more matched than the popular PICs and AVRs in all areas. Unlike PICs and AVRs however, I have seen STM8s mostly in various SMD packages. Only a handful of STM8 chips are available in Plastic Dual-Inline Package (PDIP)/through-hole packages. I think it is a big reason for which most small industries and hobbyists don't play with them as much as with other 8-bit families. People like to setup their test projects in breadboards, trial PCBs or strip-boards first, prototype and then develop for production. To cope with this issue, STM has provided several affordable STM8 Discovery (Disco) boards to get started with. Besides there are many cheap STM8 breakout-boards from China.
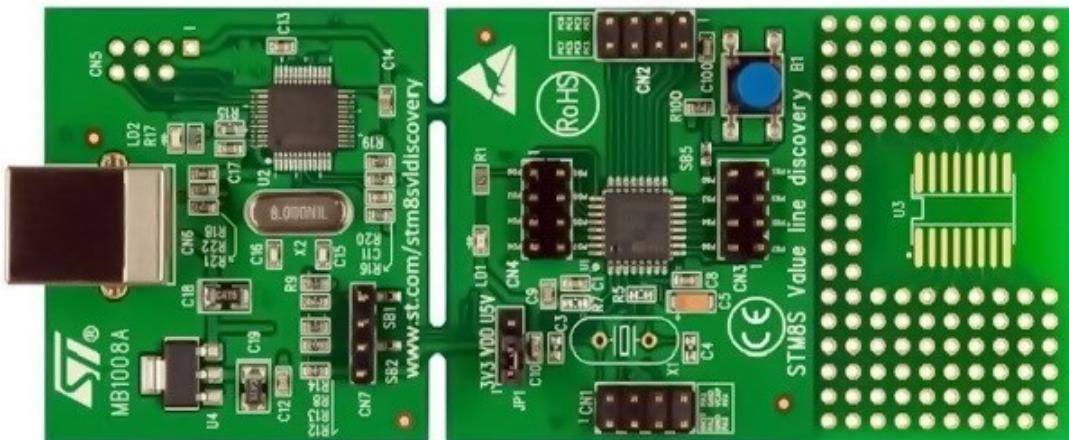
I have experience playing with AVRs, PICs, 8051s, STM32s, MSP430s, TivaC and so on. To be honest, I thought learning about STM8 micros is a pure waste of time and energy. The learning curve will be steep. Things and tools would be different and thus difficult. However, gradually I found these MCUs very useful and there is literally no complexity at all. The main drive factor for learning STM8s is the price factor. They are very cheap. When it comes down to other things, I have not found any book on STM8s written in English. There is literally no 100% complete blog post on the internet that shows the basics. Similarly, same story with tools. I have been using MikroC for AVRs, 8051s and ARMs and it is my favourite but at the time of writing, there's no MikroC compiler for STM8 family. I have also not stumbled upon any Arduino-like IDE that supports STM8 micros. Arduino-based solutions are also not my favourite as they don't go deep and have several limitations. Maybe it is not my luck. After much study and search, I found out that there are a few C compilers for STM8s. However, any new tool is both different and difficult at first. It is not always easy to adapt to new environments. You may never know what unanticipated challenges and harshness a new environment may throw at you even when you reach certain levels of expertise. I also don't want to use any pirated software and so a free compiler was a major requirement. I found out ST Visual Develop and Cosmic COSC compiler are both free tools. Cosmic used to be a paid tool but now it is absolutely free. The only easy thing till then was buying the STM8S Value Line Discovery board for just a few dollars and downloading the stuffs.

The STM8 Family

There are over a hundred STM8 microcontrollers available today. The STM8 family can be simplified into three categorical groups as shown below.
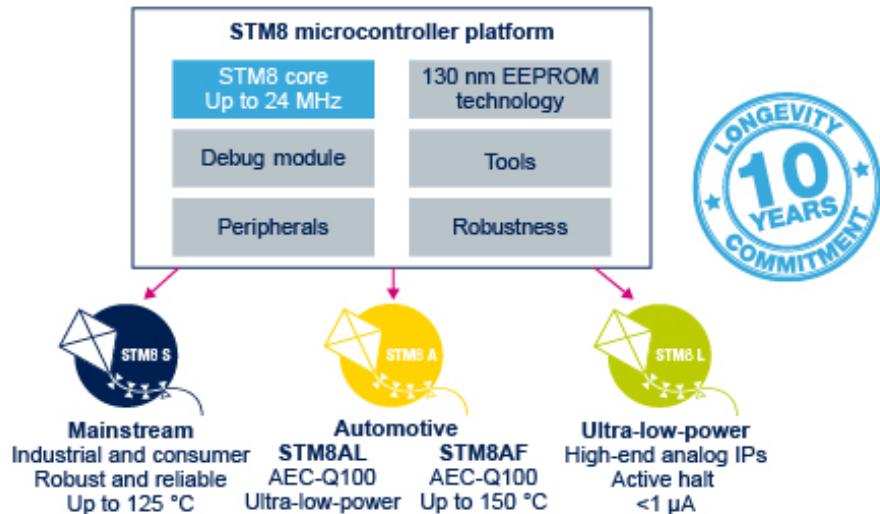

There are subgroups within these groups but broadly speaking these three groups are what by which we can define the entire family. STM8S micros are general purpose robust and reliable micros that can be employed in almost all scopes. This is the most commonly used group and in fact we will be exploring it in this article. They are also cheap and smart. The second group – the STM8A family is intended mainly for automotive industries. This group is packed with additional hardware interfaces like CAN and LIN that are musts according to present-day automotive industry doctrine. The STM8As are also very versatile and are designed to withstand the harsh extremes of an automobile. For instance, STM8As can withstand high temperatures, in excess of 100°C. The last group consists of STM8L micros which are crafted for low power or battery-backed applications. Virtually they consume no power in idle mode. Thus, if you need high power savings or energy cuts in your projects, this group is the right choice. There are also low power editions of automotive-standard STM8 micros that are labelled STM8AL. Apart from all these there is also one variant of STM8 micros that are specifically designed for capacitive touch applications. These are called STM8Ts.

The features and benefits of STM8 micros are numerous and can't simply be expressed in few words. The more you explore, the more you will know. STM8s can be powered with 3.3V or 5V DC power supplies and have built-in brownout detection circuitry. The low power editions can operate at much lower voltages than these values. Official STM8 Discovery boards come with voltage selection jumpers to allow users to select operating voltage level as required. There is very minimum risk of program corruption due to EMI or some other similar unprecedented factors. There is a fail-safe security system for the clock system which ensures that a system based on a STM8 micro won't stop working or get stuck up should its external clock source fail. All internal hardware possesses more features than any other competitive 8-bit microcontroller families that are widely available in the market. The best part is the price benefit. You pay less for most. All these features are well-suited for extremely harsh industrial environments. STM8s are designed with maximum possible combinations of features. Beyond your wildest wet dream, there are many extraordinary stuffs waiting to be unboxed.

Overview of the Discovery Board

For getting started with STM8s, STM has provided several STM8 Disco boards. There are also other third-party boards too. However, I strongly recommend Disco boards for learning and experimental purposes. There are several reasons for this recommendation. One main reason is the fact that all Disco boards come with on-board detachable ST-Link programmers and they are extremely cheap. Shown below is the top layout of a STM8S discovery board.
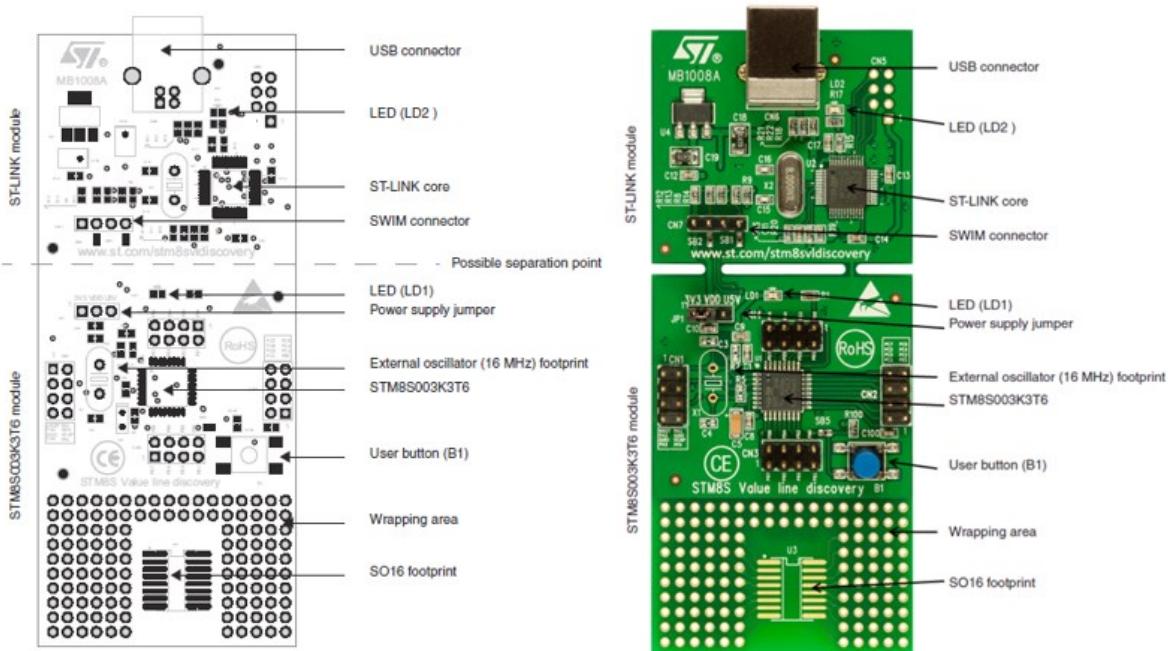
The board I mostly used here in this article hosts a STM8S003K3T6 micro. It is a 32-pin entry-level microcontroller with 8kB flash memory, 1kB RAM and 128-byte true data EEPROM. It comes with some additional hardware – a LED connected to PD0 and a push button connected to PB7. Just as I stated before it also houses a detachable ST-Link V2 programmer. However, I don't recommend separating the programmer from the whole package. The board also has a prototyping area should one needs to prototype something. The overall board has a small form-factor and is a bit longer than a standard credit card. There are several other similar and popular STM8 Discovery boards like the STM8S105 Discovery.
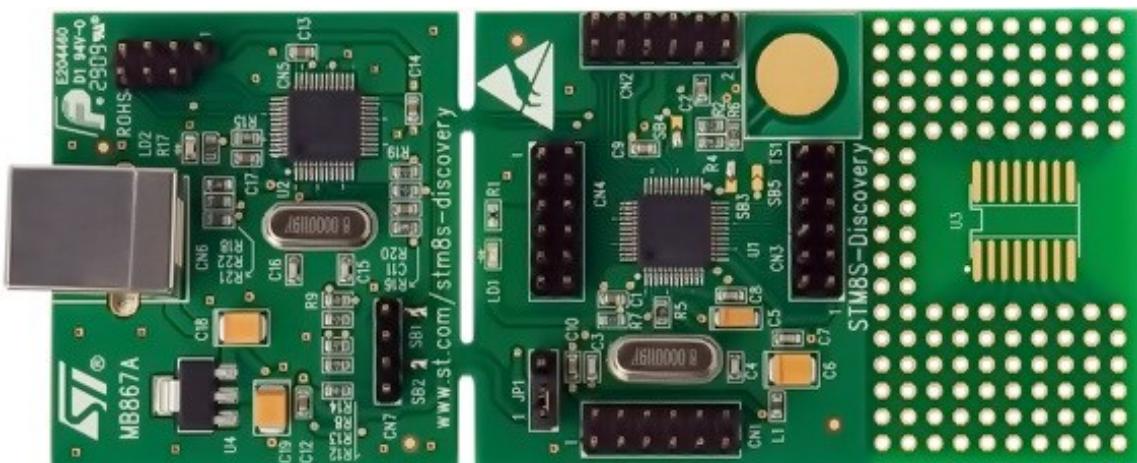
There are also bulks of cheap Chinese minimum system STM8 development boards hosting different STM8 chips. Overall the boards and the chips are so cheap that many simple cheap gadgets from China are based on STM8 MCUs.
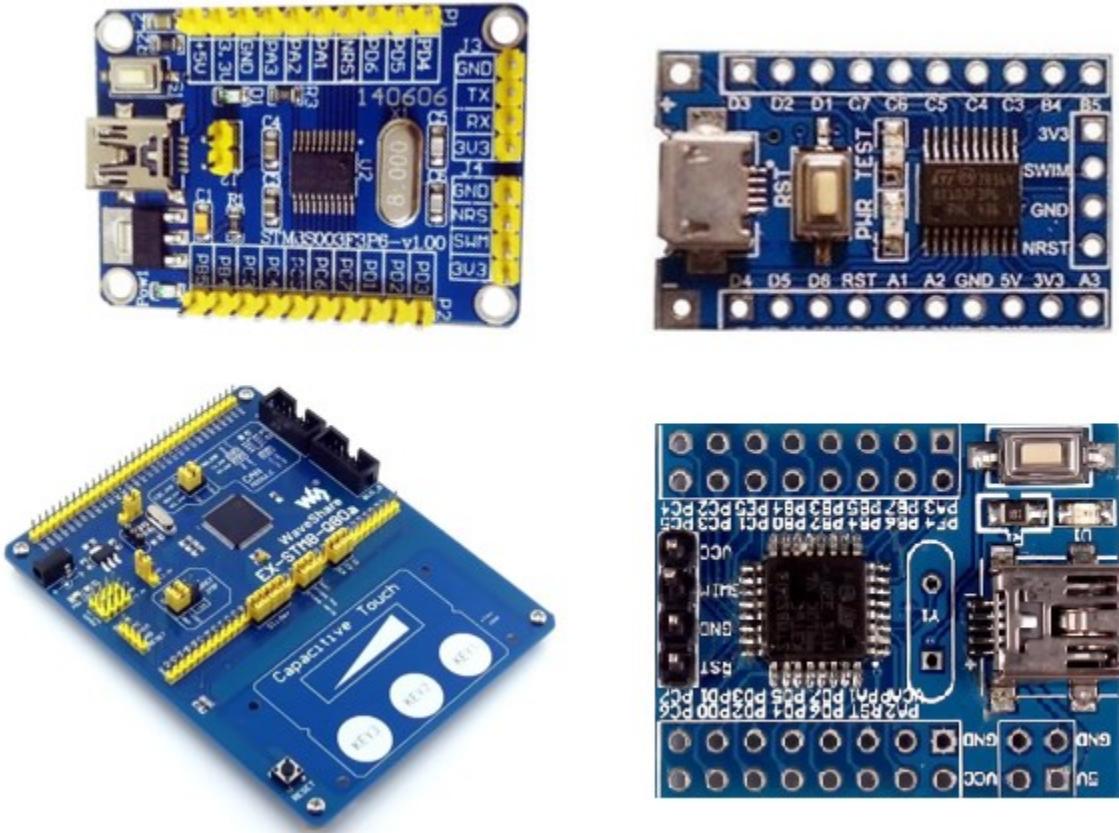
Some cheap STM8-based simple products are shown below:
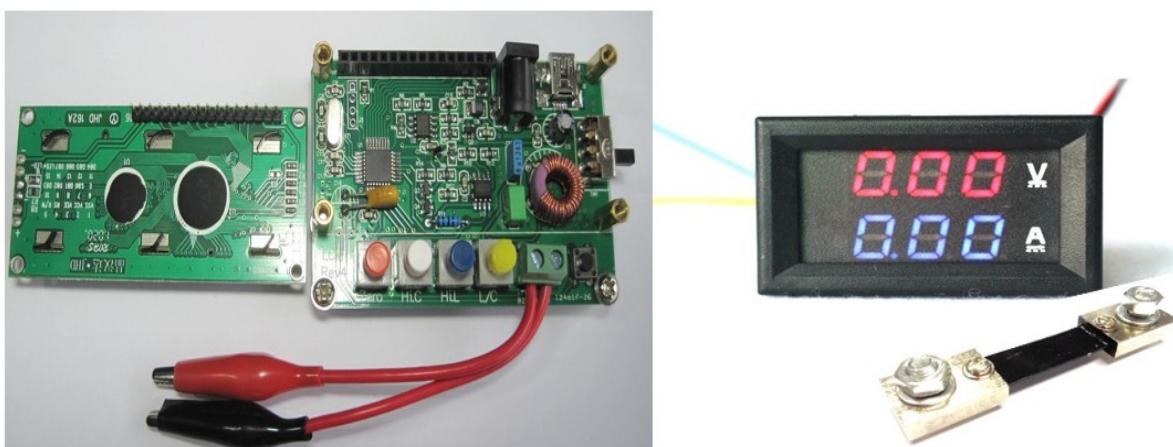
The first one is a cheap DIY LC meter LC-100A. The other one is a simple DC panel meter. These are just examples of simple products. There are many industrial and sophisticated products based on STM8 micros.

Hardware Tools

The list of hardware tools needed is not very long. We will obviously need a STM8 board and I prefer a Discovery board over any other board since it comes with a built-in ST-Link programmer/debugger hardware. If you have some other board like the ones I already showed, you will need a ST-Link programmer. I recommend an additional ST-Link programmer apart from the one available on board.


ST-LINK/V2   ST-LINK/V2-ISOL

ST-Link programmers/debuggers communicate with target STM8 micros via SWIM interface. This interface is the standard for all STM8 micros. Basically, it is a four-wire interface with two wires (VDD and GND) being used for powering the target. The rest two are reset I/O and SWIM I/O. In the official ST-Link V2 programmer unlike other ST-Link programmers, there is a dedicated port for SWIM interface with STM8 inscribed near it. Cheap USB thumb drive-sized ST-Links are also available in the market. They are portable and as good as the official ones. You can also DIY your own ST-Link programmer as the design for it is open source.

Apart from these we will also require some basic electronic lab stuffs like a USB-to-serial converter, connecting/jumper wires, LEDs, buttons, various types of sensors, etc. that are typically found in a common Arduino starter kit.
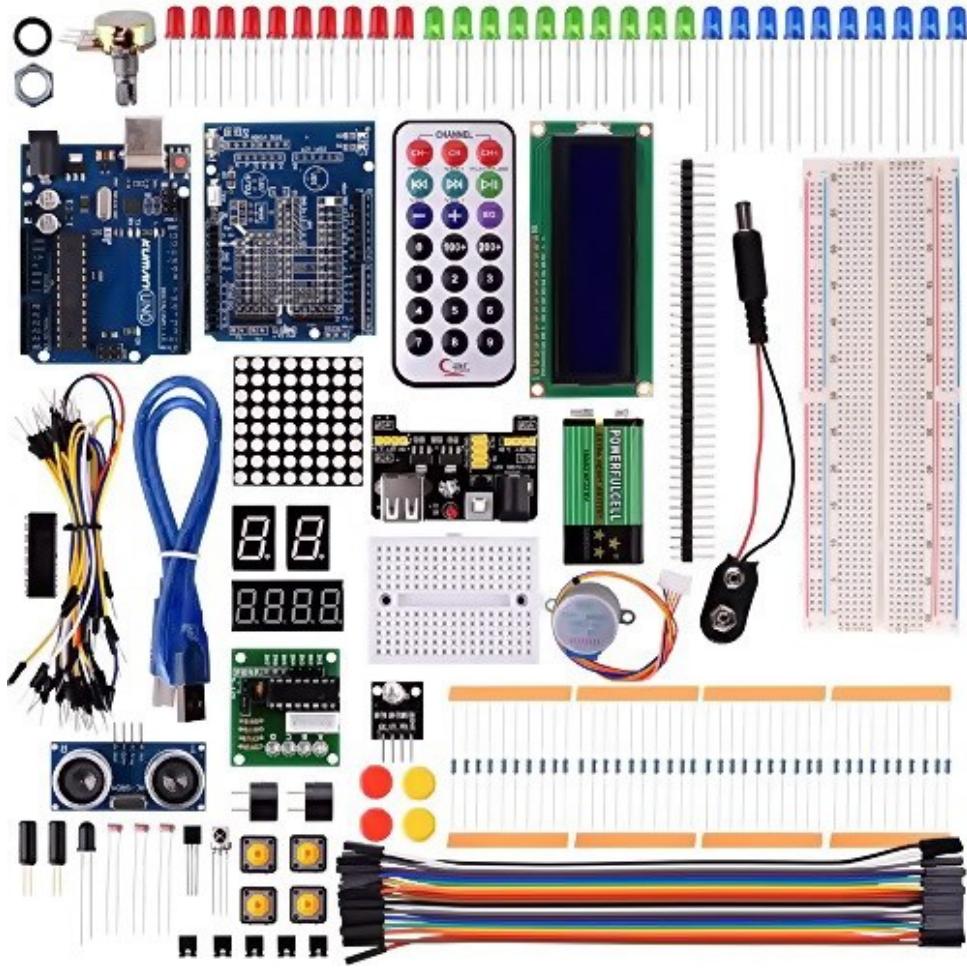
It is yet better if you have either a logic analyser or oscilloscope. A good digital multimeter (DMM) and a well-regulated DC power supply/source are must haves. You can also use a cell phone charging power bank as a power source since Disco boards have USB ports.

Software Tools

Just like any other software developer, my choice of language for software development is C language. I don't want to spend time coding complex stuffs in assembly or other languages. STM8s are also C optimized micros. Apart from these reasons, I chose C language for the fact that STMicroelectronics has provided a **Standard Peripheral Library (SPL)** that is very easy to use. With SPL, it becomes totally unnecessary to program each peripheral register with meaningless numbers and maintain coding sequence. We will never need to access registers for any reason as everything is done under the hood of SPL. All sequences are dealt inside the SPL. All that we will ever need is the clear concept of each hardware block, their working principles, their capabilities and limitations.

We will need an **Integrated Development Environment (IDE)** and a C-language toolchain. The best stuffs you can get your hands on at zero costs are **ST Visual Develop (STVD)** IDE and **Cosmic** C compiler. Both are free but a rather difficult to use at first. STVD also packs with a programmer software tool called **ST Visual Programmer (STVP)**. We will need STVP to upload codes to target STM8 micros.

Cosmic used to be a paid tool just like your PC's antivirus software but at the time of writing this article, the Cosmic team has made it absolutely free for STM8 family. However, to use it you will need to register and acquire a license key via email. Usually this procedure of acquiring license and registration is maintained automatically by the software company's server but with Cosmic it is different story. You will need to wait for some guy at Cosmic end to respond to your license request. It may take a few minutes or even a day but still the best part is getting a full version compiler for nothing.
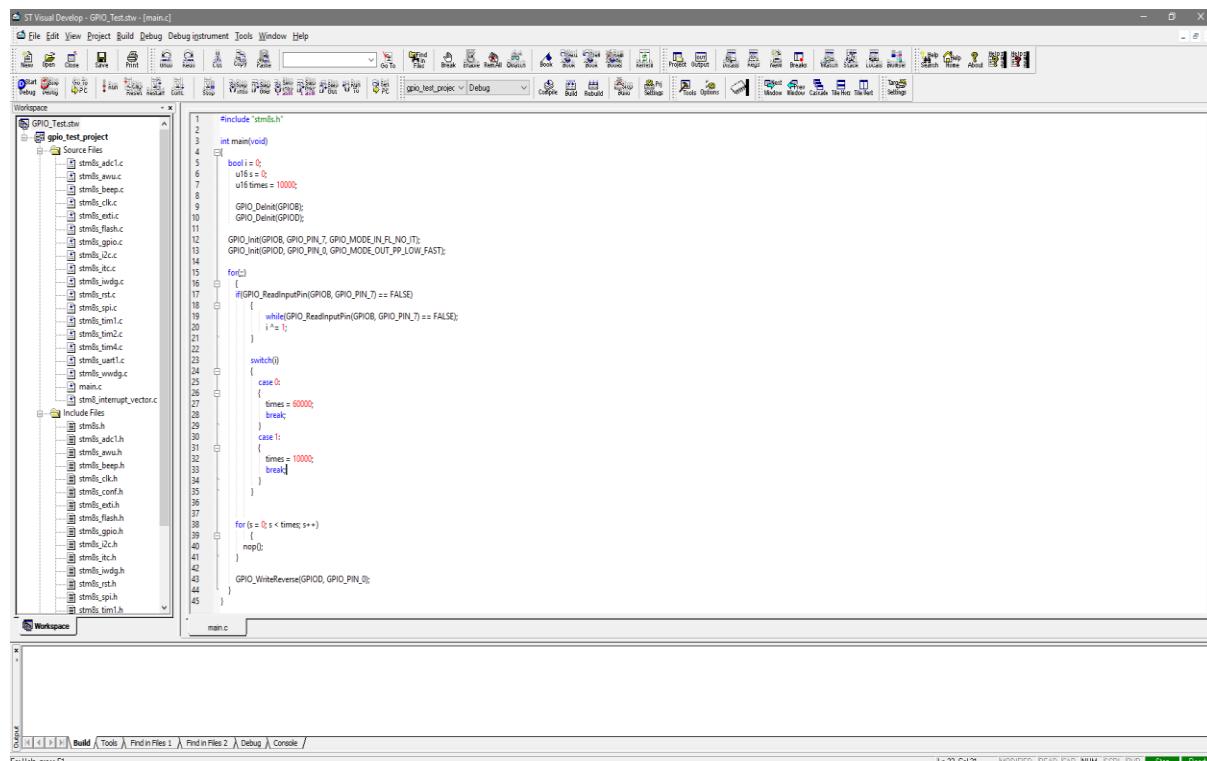
You can get
STVD from here: http://www.st.com/en/development-tools/stvd-stm8.html and
Cosmic C compiler from here: http://www.cosmic-software.com/download.php.

You need to register in order to download both software. For Cosmic you will also need to acquire a free license for it work. So just fill in some basic info about you.

Firstly, we will need to install STVD. Installation procedure is simple and same as typical software installation. Just click next, next and next. After that we will need to install Cosmic C compiler. Again, just next, next and next until the screen as shown below.
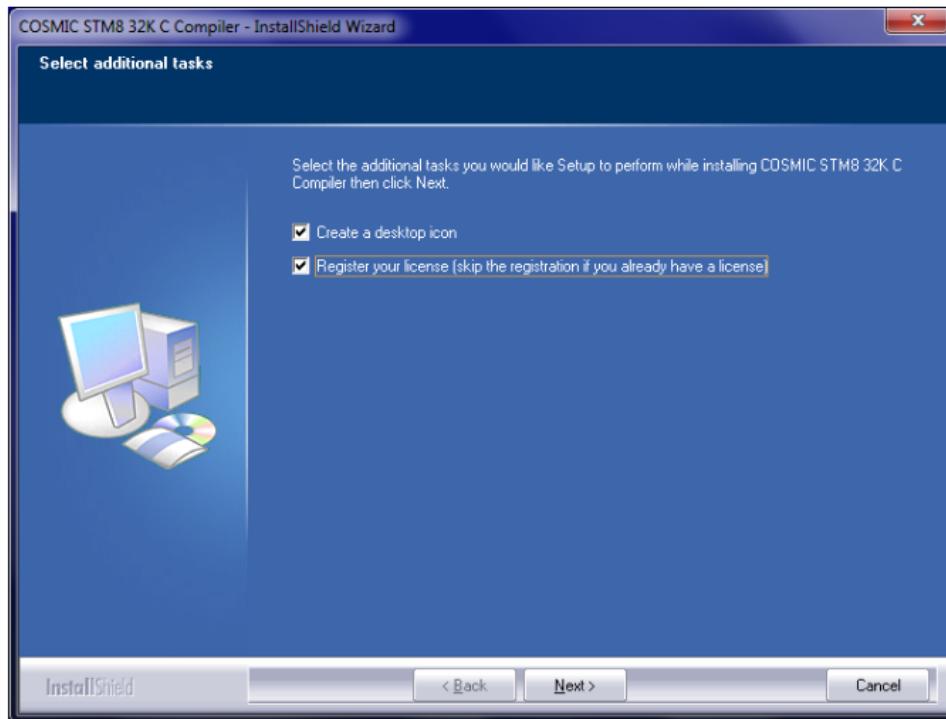
After installation, you'll prompted for licence. You must register your license unless you have already registered. If you have already registered, then you'll be asked if to overwrite registration. You should skip reregistering.
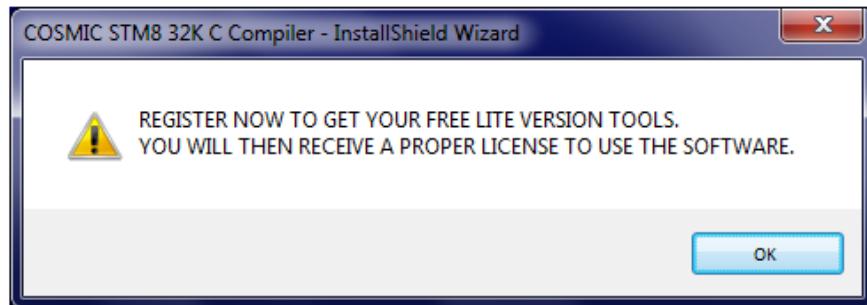
For the first run, you'll get the following screen looking for a valid license.
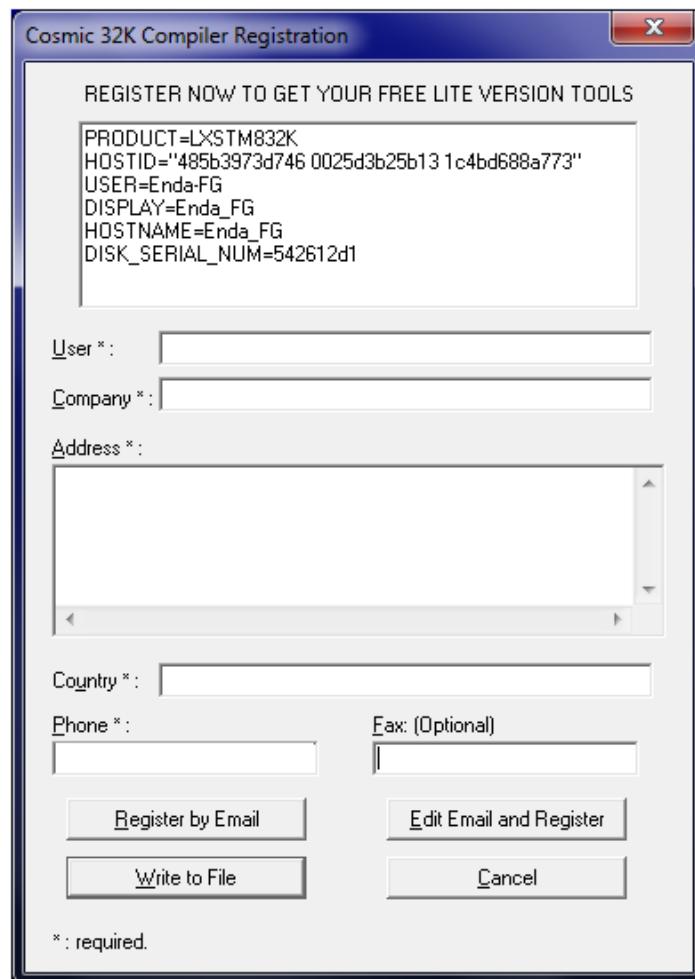
You must fill all the starred (*) points to complete the process of registration. Select **"Write to File"** option and save the file as a text (.txt) file. The file name should be **"CM8_license.txt"**. Send this file to stm8_Free@cosmic.fr with subject "**STM8FSE, STM32 32K License Request**". Now you'll need to wait for the Cosmic team to respond to your request. They'll send you an email back with an electronic key license. The file will have a name like **"license.lic"** and the email will also have some instructions.

This was my emailed license.
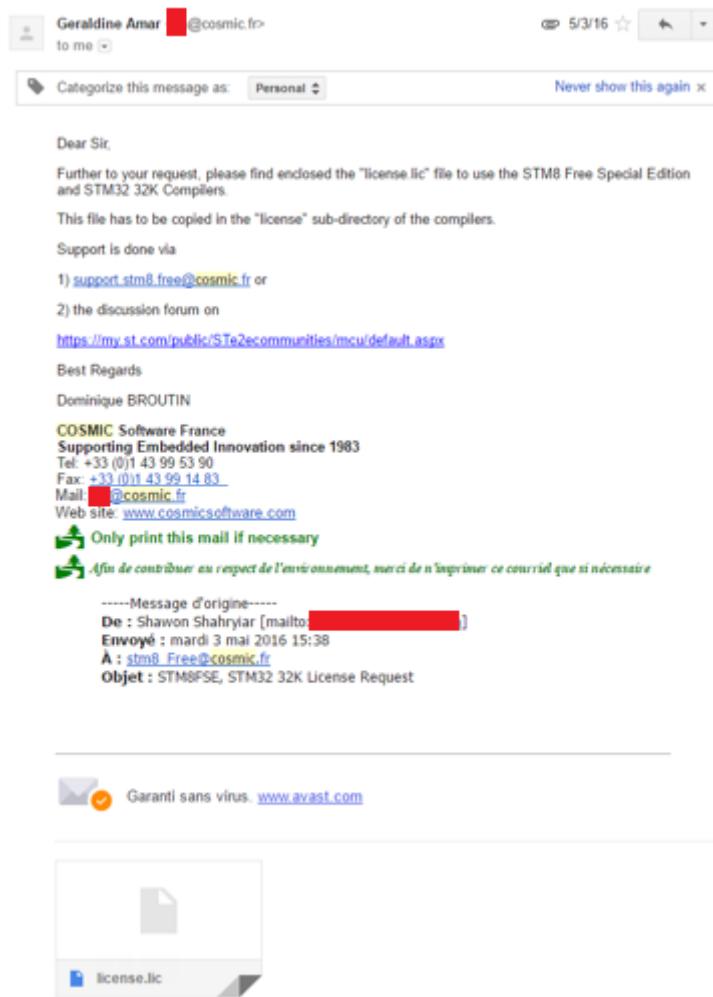
Once you get the license, you'll need to show the software its location and complete the licensing process as shown below. Save the license file in a secured location.
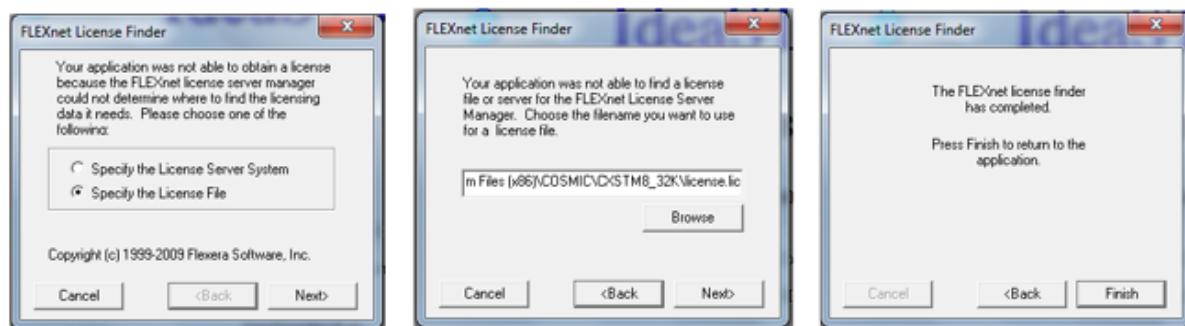
At the end of this process, we can enjoy the compiler without any limitation.

I also recommend that you download Sublime Text (https://www.sublimetext.com) or Notepad++ (https://notepad-plus-plus.org) or Microsoft Visual Code (https://code.visualstudio.com) for viewing your code with ease. These are very cool software. This is not mandatory though.

STM8CubeMX

Should I or should I not feel fortunate was my question at the time of writing this article and that's because STM8CubeMX was released in late February 2017. Yes, that's the time when I was compiling all these STM8 stuffs together. Prior to that I was wondering about a software similar to STM32CubeMX but for STM8s. Back then, I could not find one and raw documentations were only helpers. Although it is still in its early stages of development and still not as robust as its STM32 cousin in terms of code generation capabilities and other areas, we can expect great innovations in the near future. It reminds me of the early days of STM32CubeMX. Not everyone expected it to overcome all the challenges in a very short period of time. At present, we can use STM8CubeMX for common info on STM8 chips like pin assignments/mapping, basic technical specs like memory capacities, possible clock configurations, etc. I can just wonder the potential future integrations and bug fixes. Power consumption calculator is one such tool hopefully to be integrated. STM, most likely, has some serious big plans for it. Nevertheless, we must thank STM for this cool software.

Visit http://www.st.com/en/development-tools/stm8cubemx.html to download STM8CubeMX.

I recommend using it only as reference for now. Don't make yourself dependent on it. There are still many bugs fixes that are yet to be addressed. One example is as in the above screenshot. Notice Timer 4 (TIM4) is missing although STM8S003 micros have this timer. I'm pretty sure the software developers are working hard on such silly issues. For now, it is more like a promise of good tidings to come. This is why I haven't included it in the "must-have" software list and wanted it to be discussed separately.

Preparing the Software Tools

Firstly, we need three major documents before starting to program STM8 micros. These are as follows:

1. STM8 Reference Manual.
   http://www.st.com/content/ccc/resource/technical/document/reference_manual/9a/1b/85/07/ca/eb/4f/dd/CD00190271.pdf/files/CD00190271.pdf/jcr:content/translations/en.CD00190271.pdf

2. Datasheet of the MCU (**STM8S003**) that we'll be using.
   http://www.st.com/content/ccc/resource/technical/document/datasheet/42/5a/27/87/ac/5a/44/88/DM00024550.pdf/files/DM00024550.pdf/jcr:content/translations/en.DM00024550.pdf

3. STM8SVLDiscovery Board User Manual.
   http://www.st.com/content/ccc/resource/technical/document/user_manual/c8/37/11/ba/b5/e7/4c/ee/DM00040810.pdf/files/DM00040810.pdf/jcr:content/translations/en.DM00040810.pdf

These docs will be needed everywhere during learning session. The reference manual states the use and purpose of all the hardware blocks in details. It includes register descriptions, naming conventions, modes of operation of all hardware, etc. However, it does not specify the specifications of a given STM8 micro and that's because it is a generalized literature for all STM8S and STM8AF micros. As we know even within a family of micros, one MCU differs from another in many aspects. Most commonly this variation comes in the form of memory capacities and I/O pin counts. Sometimes electrical specs also vary and so to know the limits and general specs of our target MCUs we need to checkout their respective datasheets. Lastly the Discovery board user manual is most useful for the hardware schematics, pin assignments and layouts. If you are using some other board then I suggest that you acquire at least its schematic.

Now with Cosmic, STVD and STVP installed our software tool setup is almost ready. There are two approaches to STM8 programming. The first uses the traditional concepts of register-access-based coding, meaning you'll have to set up every register on your own. The second way utilizes a specialized method of coding by using standard libraries developed by STM that are both universal and platform independent, meaning your C code will be same for any compiler using these libraries. These libraries are called **Standard Peripheral Libraries (SPL)**. With these libraries, no one will ever need to get down to register-level access. The libraries are so coded that a coder will only have to know a chip's hardware specs and some basics of these hardware. On the coding part, he/she will only have to set properties and desired values. The SPL manages the rest. For instance, when setting up a UART, we will only need to set interrupts, IOs and UART properties like baud rate, parity, etc. All of these setups are done with meaningful numbers and texts.

The STMicroelectronics Standard Peripheral Libraries (SPL) for STM8 microcontrollers can be found here: http://www.st.com/en/embedded-software/stsw-stm8069.html.

I wrote this article using SPL since it will be ridiculous to code STM8s using the old-fashioned way of configuring registers one-by-one manually. Thus, it is a mandatory download item. You should preserve and retain the downloaded SPL zip file fully intact. You may need it when things get messy.

Now make two folders and name them *"inc"* and *"src"*. The *"inc"* folder will be filled with all the header files (*".h"* extension files) from the extracted zip file. Similarly, the *"src"* folder will be holding the source files (*".c"* extension files). For ease of work, it is better to keep these folders secured just like the SPL zip file because every time when we will be making new projects the files in these folders will be needed. You can copy these files to your project folder or you can keep it centrally somewhere. I prefer the former method as doing so will not have any conflicting issue with other projects needing modifications. However, it will cost hard-drive space. This method is however less confusing and trouble-free for beginners. Extract all the files as shown below:

stm8s_adc1.c	stm8s.h
stm8s_awu.c	stm8s_adc1.h
stm8s_beep.c	stm8s_awu.h
stm8s_clk.c	stm8s_beep.h
stm8s_exti.c	stm8s_clk.h
stm8s_flash.c	stm8s_conf.h
stm8s_gpio.c	stm8s_exti.h
stm8s_i2c.c	stm8s_flash.h
stm8s_itc.c	stm8s_gpio.h
stm8s_iwdg.c	stm8s_i2c.h
stm8s_rst.c	stm8s_itc.h
stm8s_spi.c	stm8s_iwdg.h
stm8s_tim1.c	stm8s_rst.h
stm8s_tim2.c	stm8s_spi.h
stm8s_tim4.c	stm8s_tim1.h
stm8s_uart1.c	stm8s_tim2.h
stm8s_wwdg.c	stm8s_tim4.h
	stm8s_uart1.h
	stm8s_wwdg.h

Note that there are more header files than source files. This is because there are two extra header files – **stm8s.h** and **stm8s_conf.h** that define processor type and processor properties. To make things work, we will have to comment one line of the **stm8s_conf.h**. You will find a line at the bottom of this file written as:

```
#define USE_FULL_ASSERT    (1)
```

You need to comment or disable this line, otherwise the compiler will throw tons of error messages. Always check this at the start of a project. Surely, we don't want to assert our code.

Creating a New Code Project

Assuming that STVD, STVP and Cosmic are properly installed, we will see how to create a new project.

1. Firstly, run STVD.

2. Select *File >> New Workspace*.
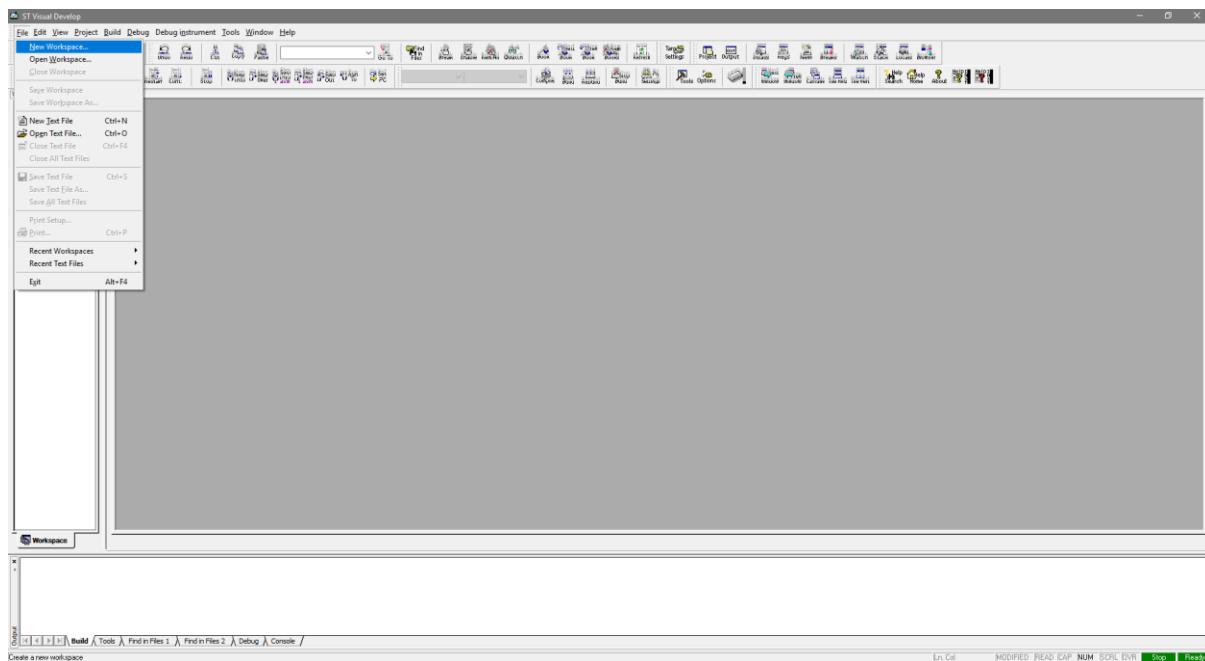
3. Select *Create workspace and project*.
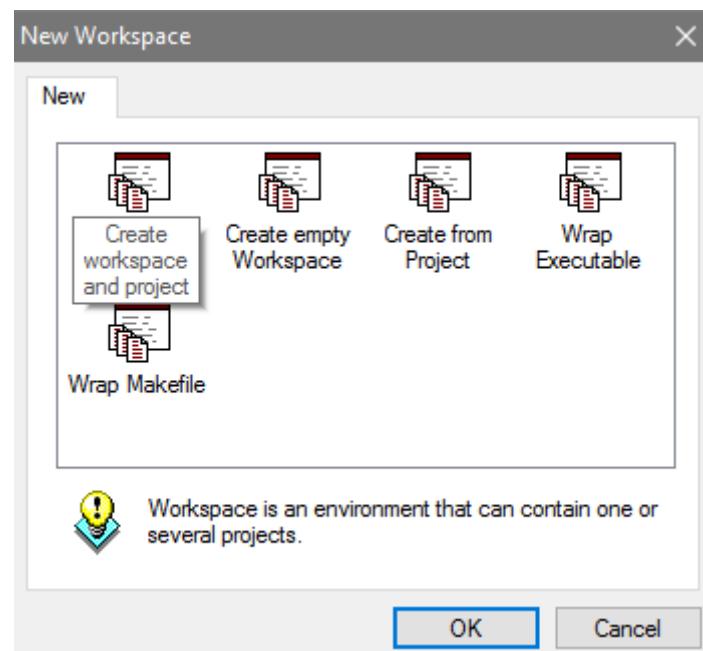
4. Select workspace folder and workspace name.
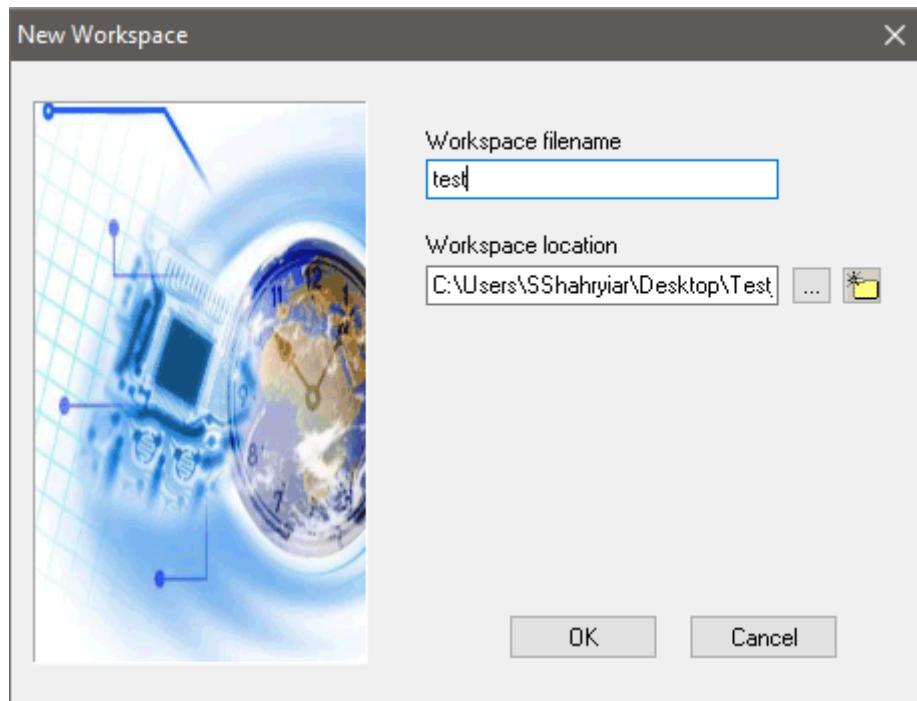
5. Set project name and select toolchain **STM8 Cosmic**. You may need to set the path of your Cosmic compiler's installation location. In my case, it is:

    C:\Program Files (x86)\COSMIC\FSE_Compilers\CXSTM8
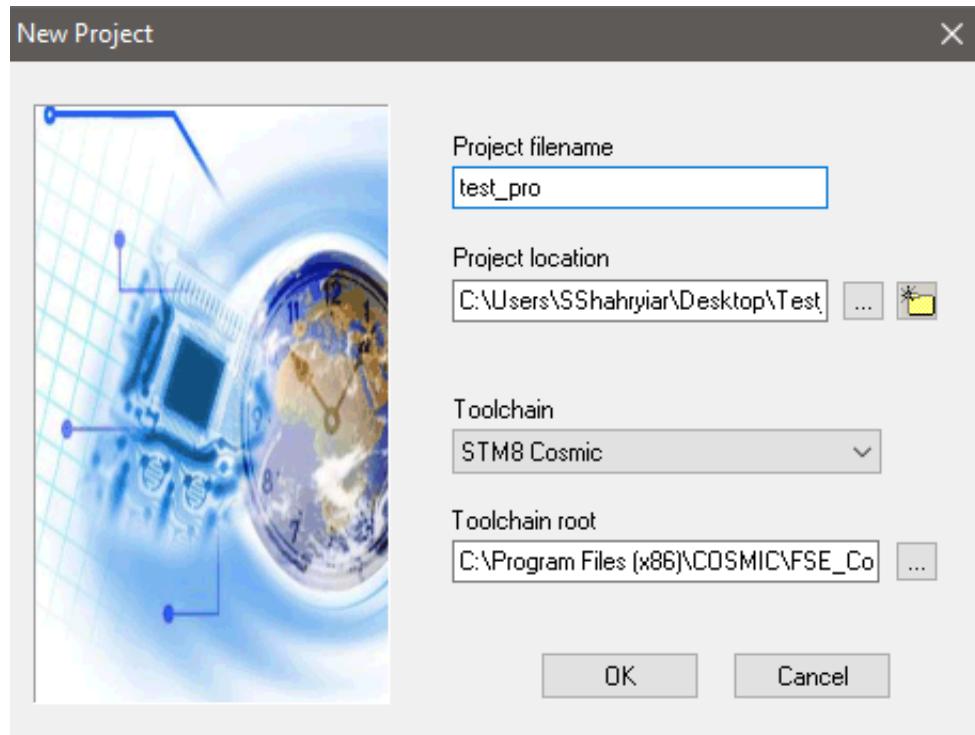
6. Type and select target chip part number. Last two or three digits and letters are enough for finding the correct micro.
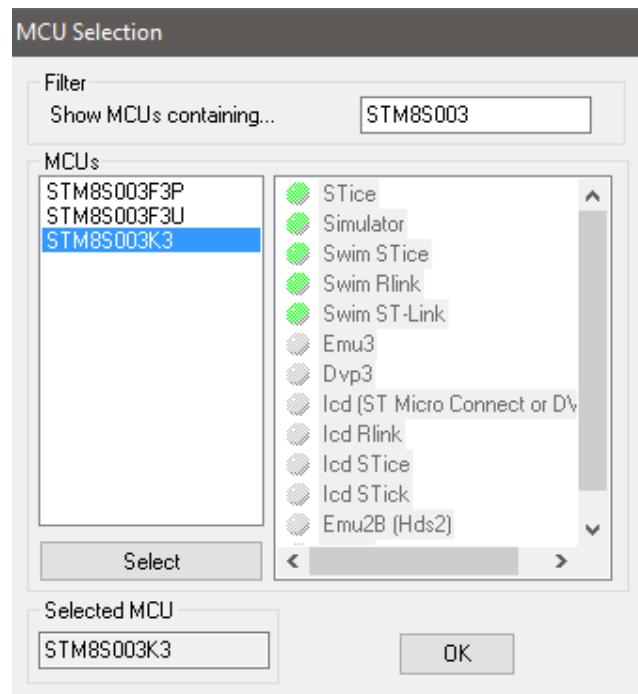
7. Now add the source and header files from the previously mentioned SPL folder.
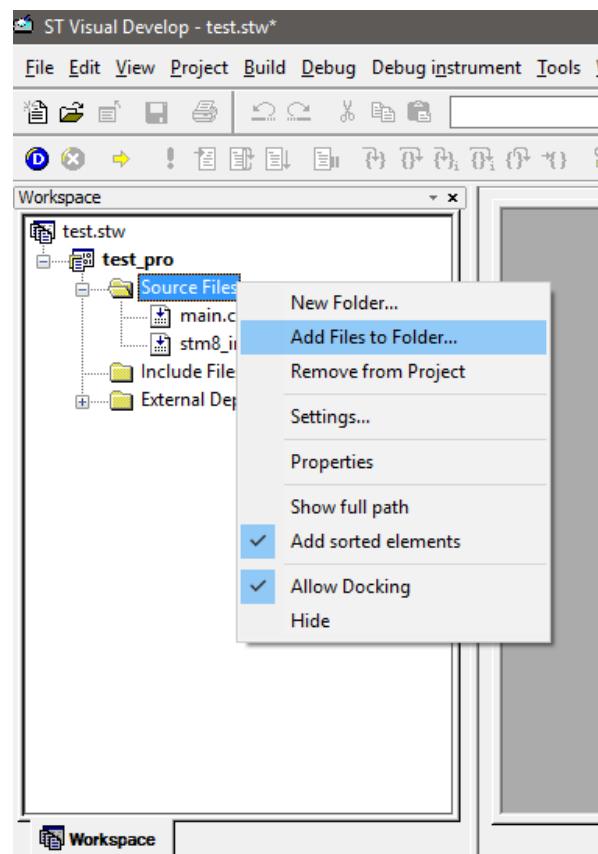
8. After file inclusions, the workspace tab changes as shown below.
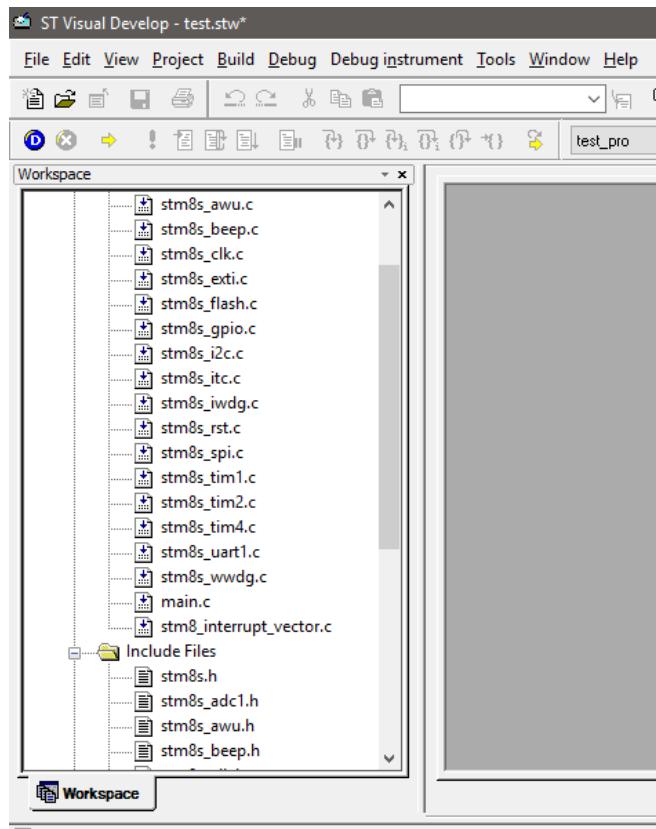
9. Locate and open *main.c* file from the source tab, and then type *#include "stm8s.h"* at the top as shown below:
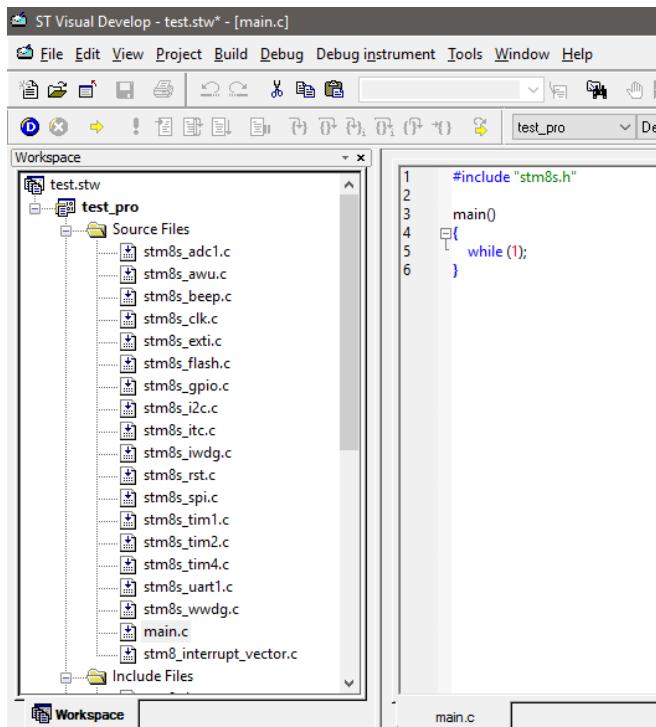
10. You'll have to edit the **STM8S.h** header file and uncomment the chip number you are going to use as shown below:

11. Compile the code once using the key combination **CTRL+F7** or by pressing the compile button. If everything is okay, there should be no error or warning message. The reason for this blank compilation is to use the compiler's powerful code assistant feature. With this feature, we can predict or complete a piece of code line by only writing the first few letters and then pressing **CTRL + SPACE** keys simultaneously.

During compilation, you may get tons of errors for hardware files that are not available in your target STM8S micro. For instance, CAN hardware is not available in STM8S003K3 and so if you have added CAN source and header files you will get an error for that. Once identified by the error messages, the corresponding header and source files for that particular hardware must be removed.

A wiser approach to avoid this issue is to check device datasheet for available hardware in it. The contents section of datasheet shows this as shown below:

STM8CubeMX can also be used for this purpose but some hardware peripherals which do not have any input-output dependency are not shown in it as previously discussed.
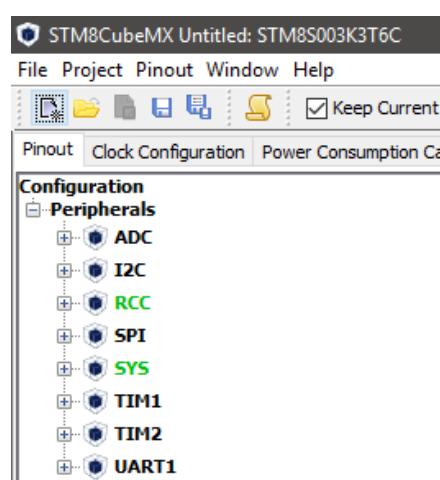
Similarly, one more caution must be observed. Unless your code is using any interrupt, interrupt source and header files (***stm8s_it.h*** and ***stm8s_it.c***) must be excluded. Sometimes it is better to add only those files that you will need to complete a project. For example, if your project is just using GPIOs, it is better to add GPIO files only along with ***stm8s.h*** and ***stm8s_conf.h***. However, I recommend this technique only after you have mastered STM8 coding well because in most cases you will need multiple hardware which have dependencies on each other. As an example, when using SPI, you'll need both GPIO and SPI modules. If you understand these dependencies, it is okay to select files as per need. You can, then, comment out unnecessary hardware module files specified in the ***stm8s.h*** header file and get a faster compilation and build process. After compilation, you should always build/rebuild your project by hitting the ***Build*** or ***Rebuild*** button. This will generate the final ***s19*** output file in either ***Debug*** or ***Release*** folder according to the generation mode selected. If things are in order, there should be no error or warning message.
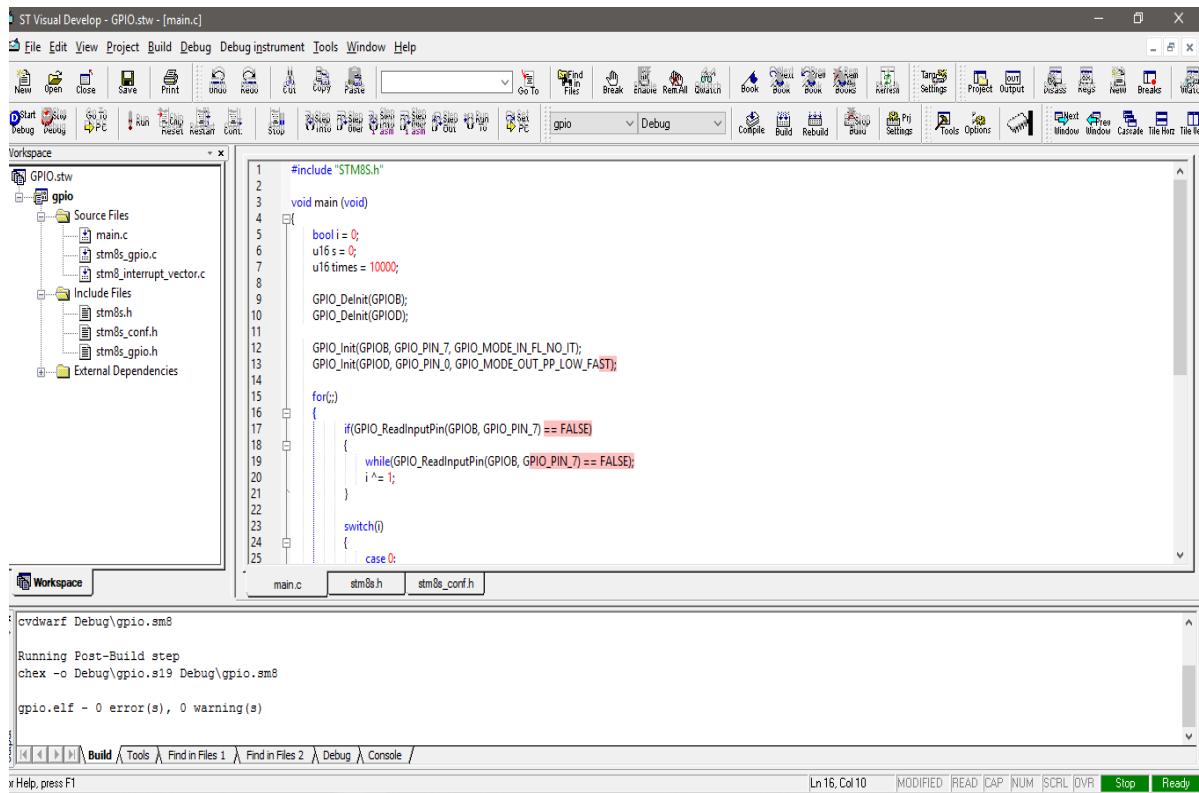
Lastly, I have not found any useful simulation software like Proteus VSM or Electronic Workbench that support STM8 family. Thus, we have to debug our code in real-life with real hardware. It may sound difficult but actually it is not so. We can, however, use such software to make models of STM8 micros and make our PCBs. I don't like simulations as they are not always accurate and real-world type.

One more advice I would like to give to the readers. Please read the SPL help file. It is located in the SPL zip file under the name ***stm8s-a_stdperiph_lib_um.chm***. It explains each function, definition, data structure, all internal hardware modules and how to use them properly. This is a very important document and your best friend in coding STM8 micros. Apart from this document the reference manual is equally important as it details the capabilities of all internal hardware. I won't be detailing the internal hardware much as these docs will be doing so.

Check this video out for details: https://www.youtube.com/watch?v=sqCuSVjevrY.

Uploading Code

Codes generated by Cosmic C compiler have *s19* file extensions. It is similar to typical hex file format, containing user code as hex values. Well since we don't need to modify finally generated output files, it doesn't really matter in which format it is. All we will need is to upload them to our target MCUs. We can do it in two ways – either by using STVP or STVD.

Firstly, let's check the method with STVP. Run STVP software. For the first time the following window will appear. From here we have to select ST-Link programmer, SWIM interface and our target chip.
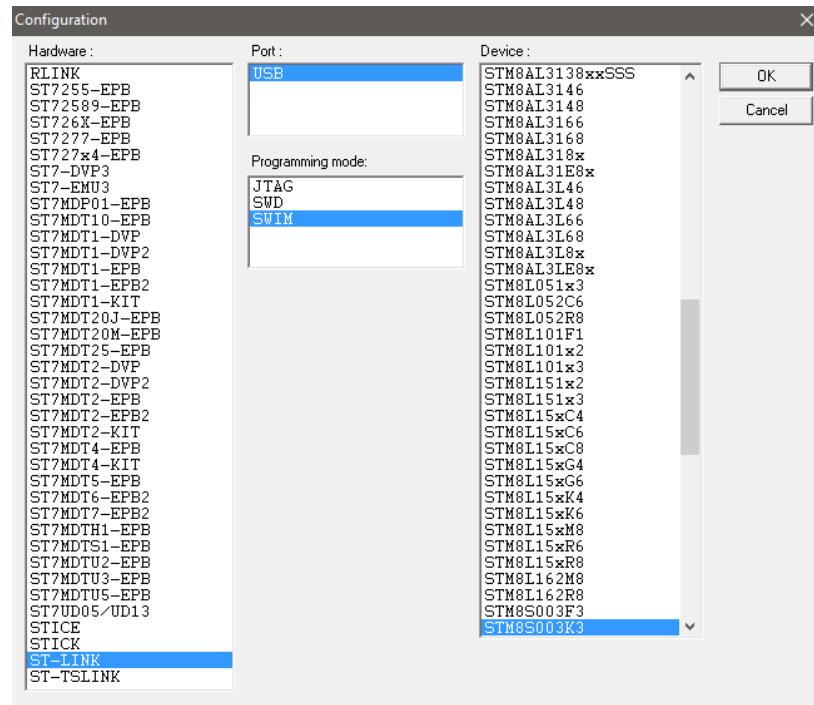
STVP interface looks like any other programmer interface as shown below:
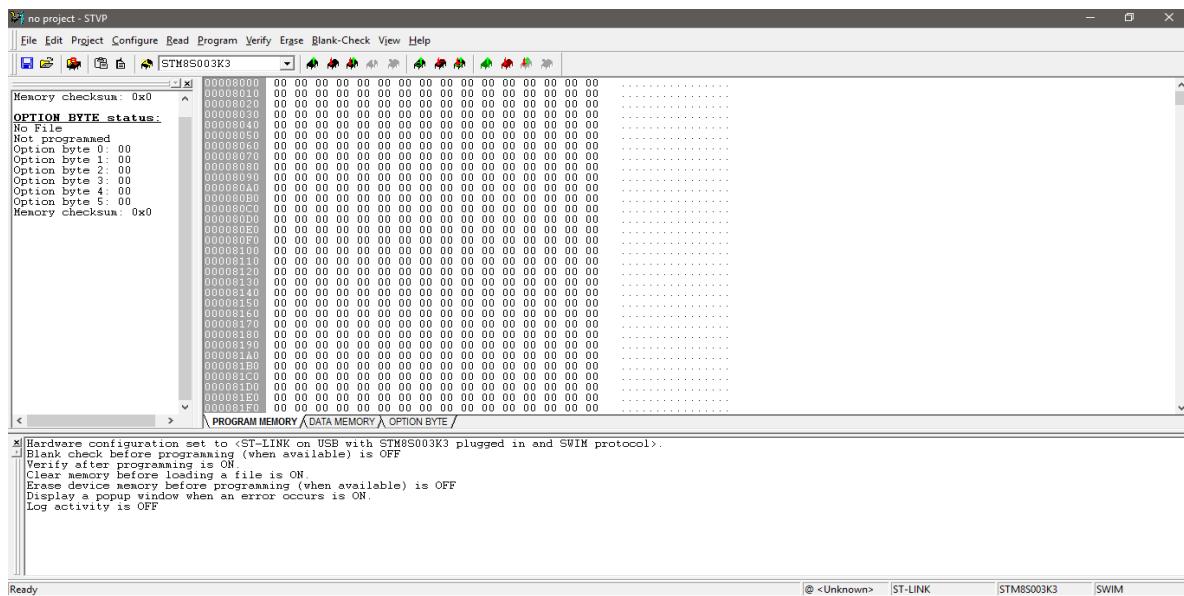
Notice the mid tabs at the bottom of the hex values. From here we can see the hex values for program memory, data/EEPROM memory and configuration settings. The configuration setting bits are intended for setting some special hardware configurations or extending features of the target as well as setting memory readout protection.
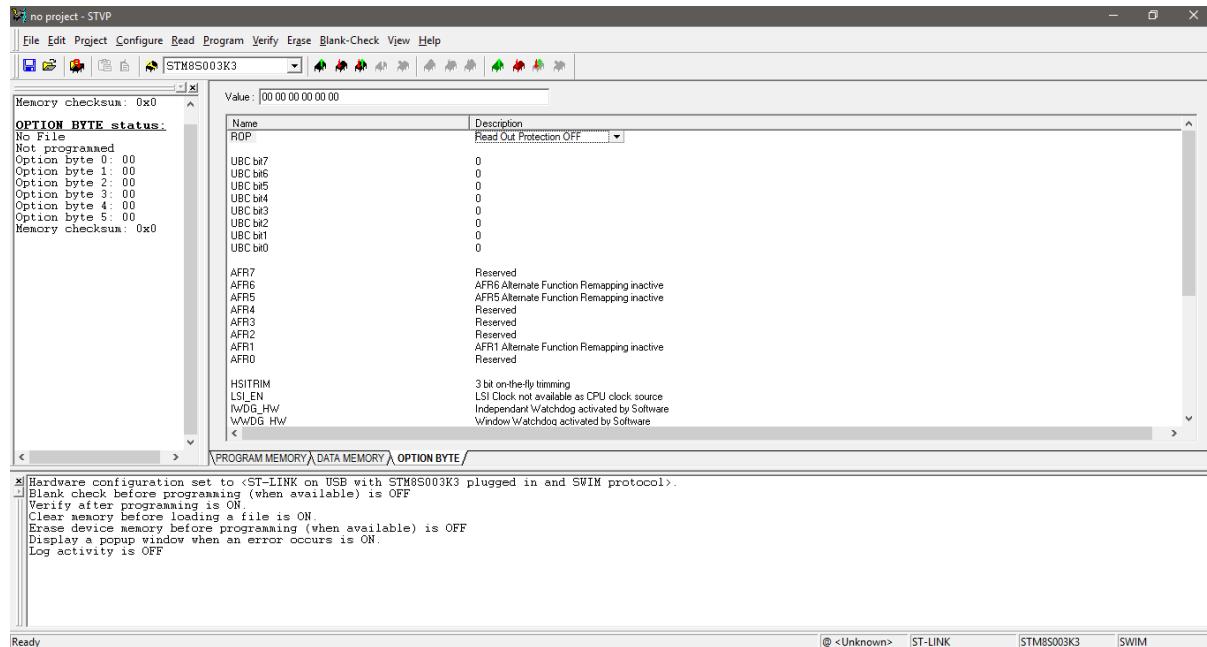
Try not to mess with security or protection bit at first or during tests as it will lock your chip up, rendering it useless. You won't simply be able to write it again until you unlock it. Unless needed, we won't be changing any default configuration bit. One thing to note is the fact that upon new compilation and build, the newly generated output file is automatically reloaded. The rest of the stuffs like loading or saving a s19 file, reading, writing and others are as simple as like with other programmers. I won't be explaining these steps as I assume that readers of this article already know how to do all these from their previous experiences with other MCUs.

Now we will explore how we can upload a code to our target using STVD. After compiling and building a project successfully without any error, the compiler will generate a s19 output file either in **Debug** or **Release** folder depending on which mode of compilation selected. By default, **Debug** mode is selected unless the coder changed it and so our desired s19 file will be in this folder. First, we need to open the programmer interface. We can do that either by clicking the icon as shown below:
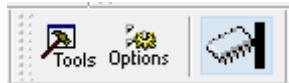
or we can go to **Tools >> Programmer**.

We will get a new window as shown below:
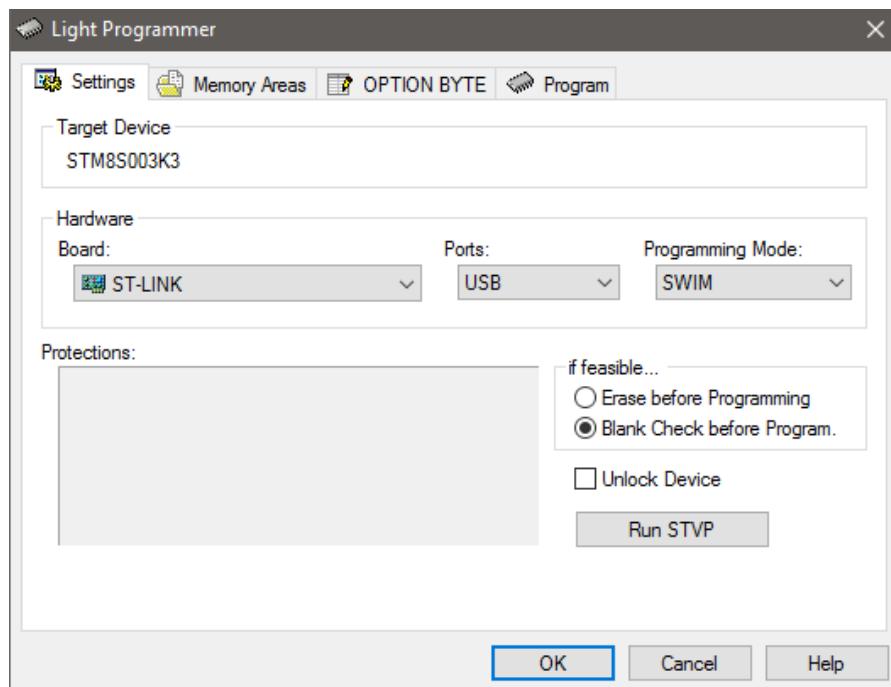
As the name of the new window suggests, it is a light-weight programmer interface but good enough for our purpose. Notice that there are many options and four tabs. Here again we need to select programmer, programming interface (SWIM) and erase/blank check options. Then we go to the next tab to select files for EEPROM (if any) and Program (also Flash/Code) memory as shown below. You can add/remove files just as usually.
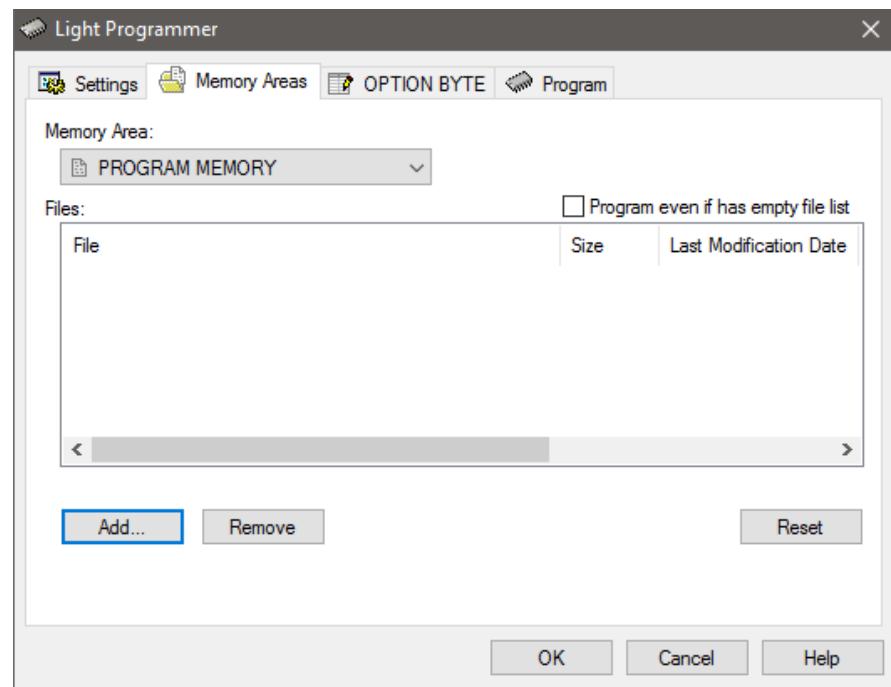
Next, we set configuration bits if needed from the tab as shown below:
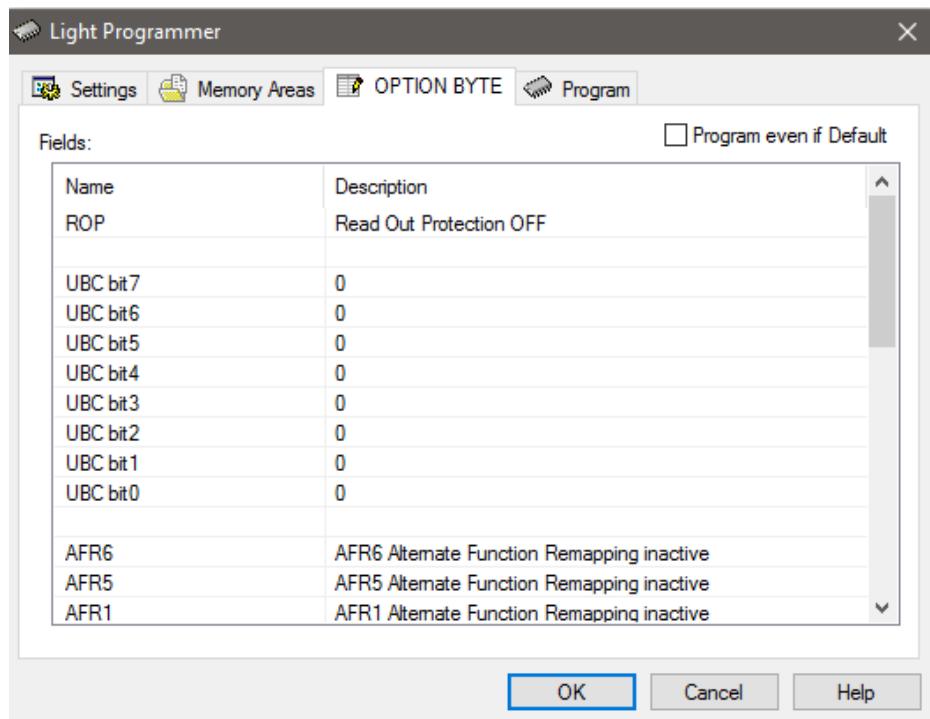
Finally, we are ready to upload code. Just hit the start button and wait for the process to finish.
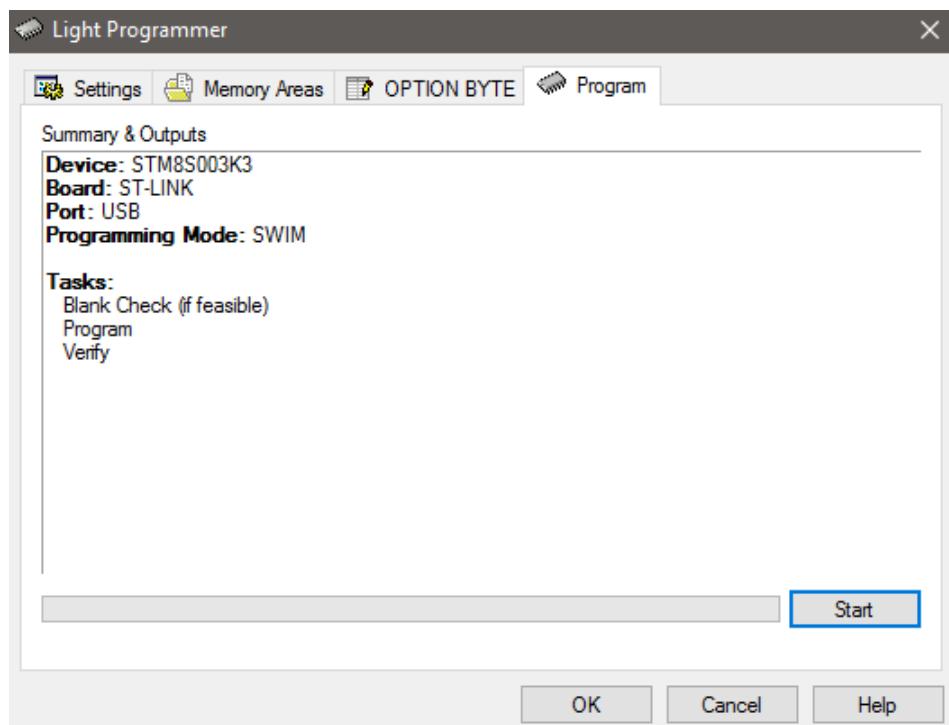
Every time a code is programmed, it is verified automatically.

General Purpose Input Output (GPIO)

The very first *"Hello World"* project that we do with every new embedded device is a simple LED blinking program. Here we do the same. We will be basically testing both input and output function by making a variable flash rate blinking LED. Check the schematic of the Disco board. Check the pins with which the on-board LED and the push button are connected.
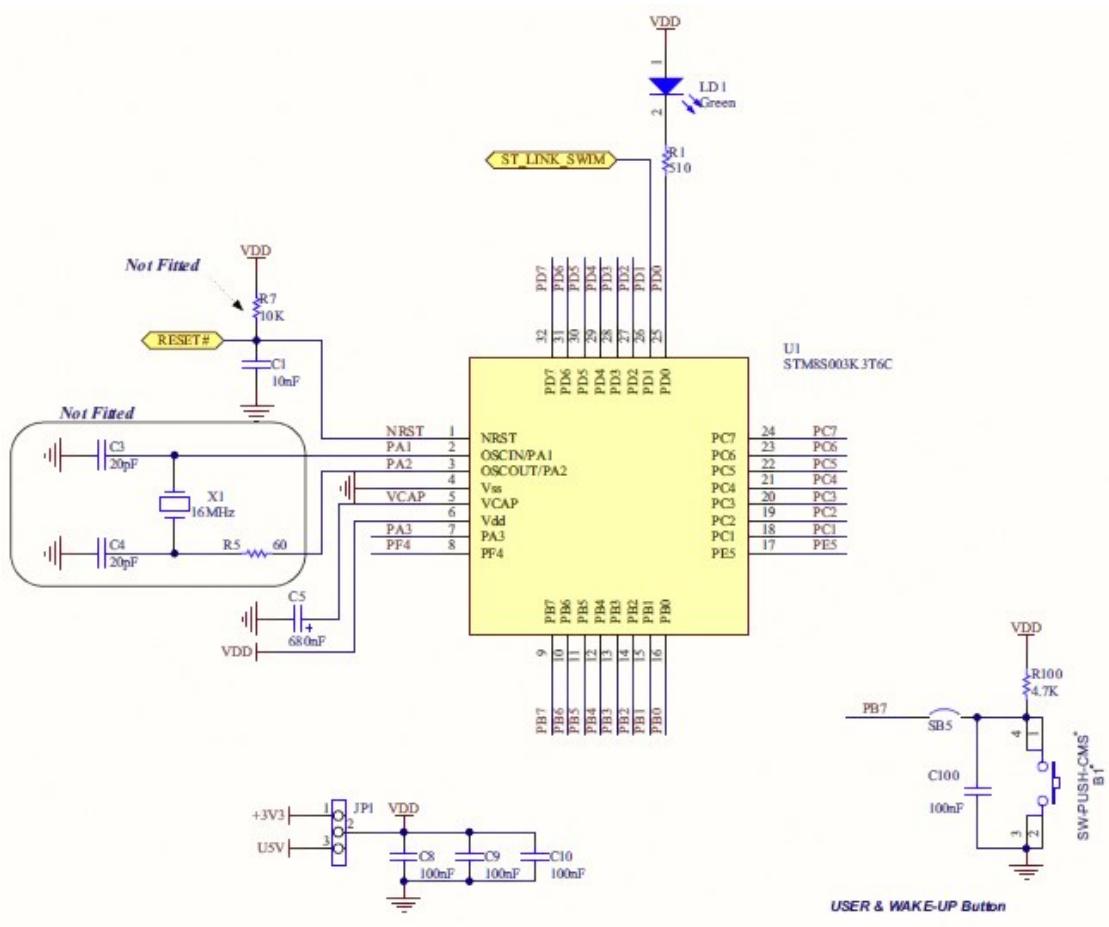
You can also use the STM8CubeMX in board selector mode for this too.

Shown below is the internal block diagram of GPIO pins:
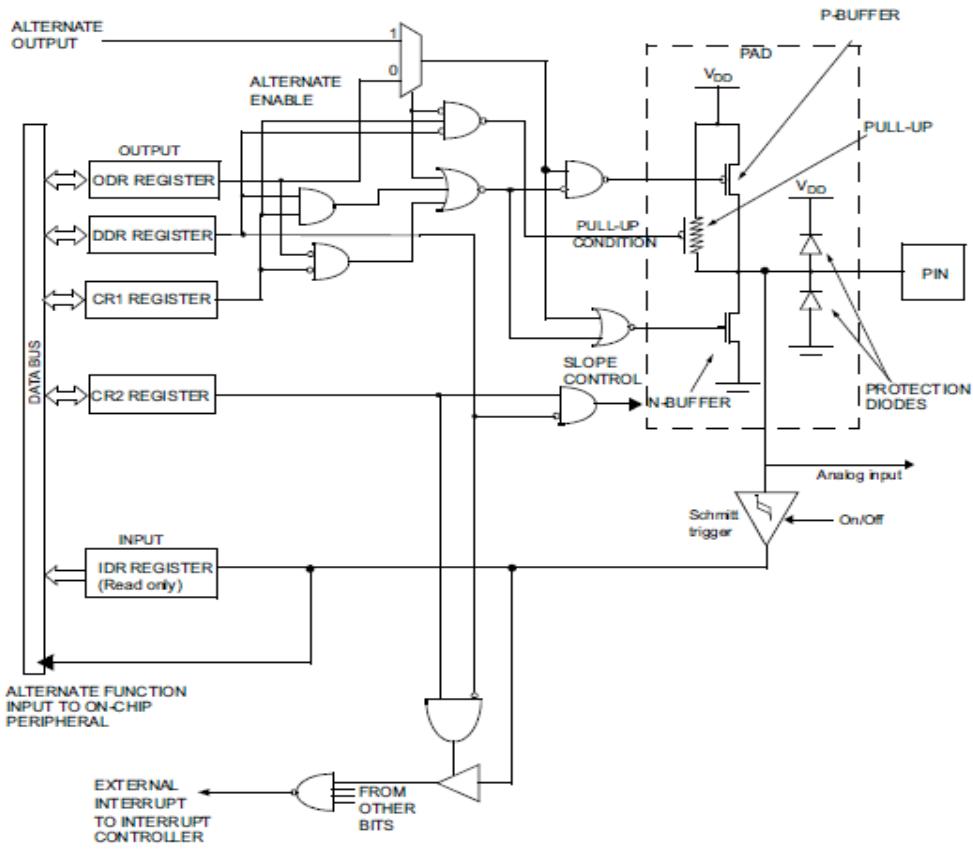
Because each I/O is independently configurable and have many options associated with it, its block looks complex at first sight. Check the various options every I/O possess:

Mode	DDR bit	CR1 bit	CR2 bit	Function	Pull-up	P-buffer	Diodes to V$_{DD}$	to V$_{SS}$
Input	0	0	0	Floating without interrupt	Off	Off		
	0	1	0	Pull-up without interrupt	On			
	0	0	1	Floating with interrupt	Off		On	On
	0	1	1	Pull-up with interrupt	On			
Output	1	0	0	Open drain output	Off	Off		
	1	1	0	Push-pull output		On		
	1	0	1	Open drain output, fast mode		Off		
	1	1	1	Push-pull, fast mode	Off	On		
	1	x	x	True open drain (on specific pins)	Not implemented		Not implemented (1)	

1. The diode connected to V$_{DD}$ is not implemented in true open drain pads. A local protection between the pad and V$_{OL}$ is implemented to protect the device against positive stress.

Shown below are the SPL functions associated with the GPIO module.

GPIO_Public_Functions

STM8S_StdPeriph_Driver

Functions

void	GPIO_DeInit (GPIO_TypeDef *GPIOx)
	Deinitializes the GPIOx peripheral registers to their default reset values.
void	GPIO_ExternalPullUpConfig (GPIO_TypeDef *GPIOx, GPIO_Pin_TypeDef GPIO_Pin, FunctionalState NewState)
	Configures the external pull-up on GPIOx pins.
void	GPIO_Init (GPIO_TypeDef *GPIOx, GPIO_Pin_TypeDef GPIO_Pin, GPIO_Mode_TypeDef GPIO_Mode)
	Initializes the GPIOx according to the specified parameters.
uint8_t	GPIO_ReadInputData (GPIO_TypeDef *GPIOx)
	Reads the specified GPIO input data port.
BitStatus	GPIO_ReadInputPin (GPIO_TypeDef *GPIOx, GPIO_Pin_TypeDef GPIO_Pin)
	Reads the specified GPIO input data pin.
uint8_t	GPIO_ReadOutputData (GPIO_TypeDef *GPIOx)
	Reads the specified GPIO output data port.
void	GPIO_Write (GPIO_TypeDef *GPIOx, uint8_t PortVal)
	Writes data to the specified GPIO data port.
void	GPIO_WriteHigh (GPIO_TypeDef *GPIOx, GPIO_Pin_TypeDef PortPins)
	Writes high level to the specified GPIO pins.
void	GPIO_WriteLow (GPIO_TypeDef *GPIOx, GPIO_Pin_TypeDef PortPins)
	Writes low level to the specified GPIO pins.
void	GPIO_WriteReverse (GPIO_TypeDef *GPIOx, GPIO_Pin_TypeDef PortPins)
	Writes reverse level to the specified GPIO pins.

Observe the code below. This is the power of the ST's SPL. The code is written with no traditional raw register access. Everything here has a meaningful nomenclature, just like regular naming/words of the reference manual/comments. There shouldn't be any issue understanding the code. The code is almost Arduino-like but more efficient and robust. Here we are polling an input pin's state to alter the blink rate of a LED.

Hardware Connection
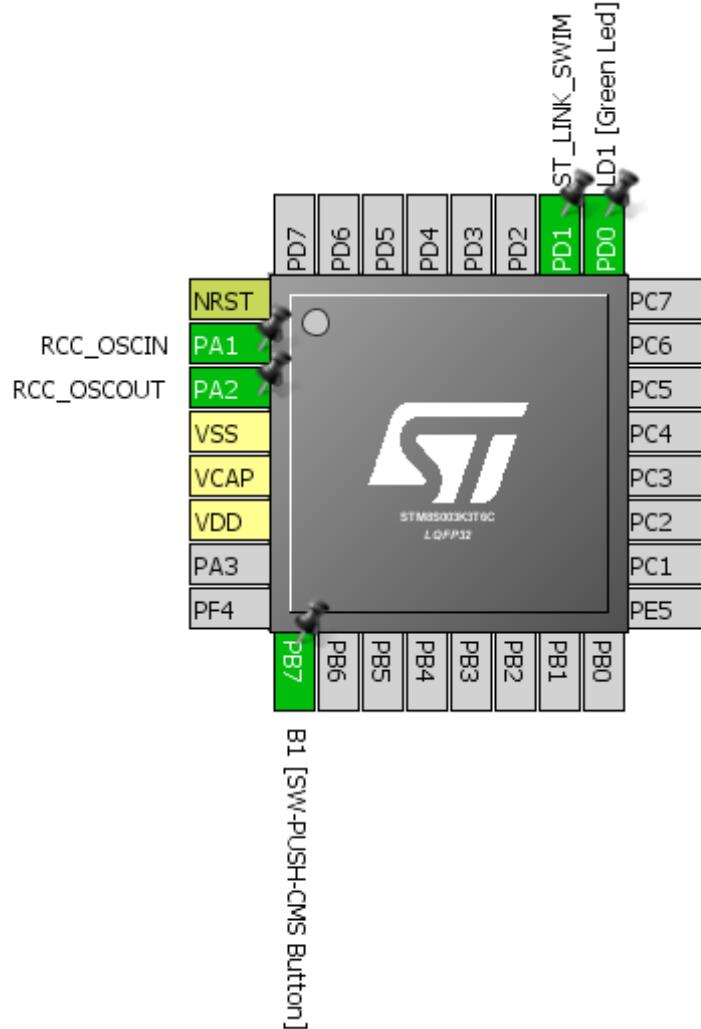
Code Example

```c
#include "stm8s.h"


void GPIO_setup(void);


void main(void)
{
    bool i = FALSE;

    GPIO_setup();

    for(;;)
    {
        if(GPIO_ReadInputPin(GPIOB, GPIO_PIN_7) == FALSE)
        {
```

```
                while(GPIO_ReadInputPin(GPIOB, GPIO_PIN_7) == FALSE);
                i ^= 1;
            }

            switch(i)
            {
                case TRUE:
                {
                    delay_ms(1000);
                    break;
                }
                case FALSE:
                {
                    delay_ms(100);
                    break;
                }
            }

            GPIO_WriteReverse(GPIOD, GPIO_PIN_0);
        };
}


void GPIO_setup(void)
{

    GPIO_DeInit(GPIOB);
    GPIO_Init(GPIOB, GPIO_PIN_7, GPIO_MODE_IN_FL_NO_IT);

    GPIO_DeInit(GPIOD);
    GPIO_Init(GPIOD, GPIO_PIN_0, GPIO_MODE_OUT_PP_LOW_FAST);
}
```

Explanation

The following lines deinitialize the GPIOs we used. Every time you reconfigure or setup a hardware peripheral for the first time you must deinitialize it before using it. Though it is not mandatory, it will remove any chance of wrong/conflicting configurations.

```
GPIO_DeInit(GPIOB);
GPIO_DeInit(GPIOD);
```

After deinitialization, we are good to go for initializing or setting up the GPIOs. Inputs can be with or without internal pull-up resistors. Outputs can be either push-pull totem-pole or open drain types. Each pin can be individually configured and does not have any dependency on another. The following codes set GPIO PB7 as a floating input with no interrupt capability and GPIO PD0 as a fast push-pull output. PB7 is set up as a floating input rather than an internally pulled-up input because the button on the Disco board is already pulled up externally.

```
GPIO_Init(GPIOB, GPIO_PIN_7, GPIO_MODE_IN_FL_NO_IT);
GPIO_Init(GPIOD, GPIO_PIN_0, GPIO_MODE_OUT_PP_LOW_FAST);
```

The remaining part of the code in the main loop is just polling the button's state and altering the delay time for toggling the LED if the button is pressed.

```
for(;;)
{
    if(GPIO_ReadInputPin(GPIOB, GPIO_PIN_7) == FALSE)
    {
        while(GPIO_ReadInputPin(GPIOB, GPIO_PIN_7) == FALSE);
        i ^= 1;
    }

    switch(i)
    {
        case TRUE:
        {
            delay_ms(1000);
            break;
        }
        case FALSE:
        {
            delay_ms(100);
            break;
        }
    }

    GPIO_WriteReverse(GPIOD, GPIO_PIN_0);
};
```

Demo

Video link: https://www.youtube.com/watch?v=Rr1vpfoze4w.
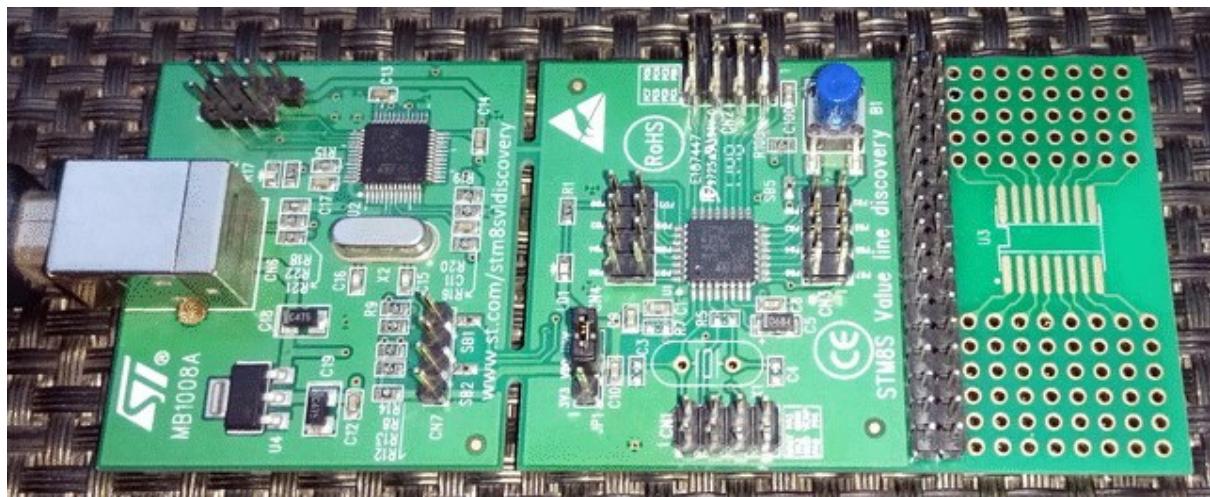
Clock System (CLK)

The internal network of STM8 clock system allows us to tune up operating speeds of peripherals and CPU according to our needs. Software delays and power consumption depend on how the clock system is set.

In STM8 micros, there are three main clock sources – **High Speed Internal Clock (HSI)**, **High Speed External Clock (HSE)** and **Low Speed Internal Clock (LSI)**. The HSI has an oscillating frequency of 16MHz and is an internal RC oscillator with good precision – about 1% tolerant over a wide temperature range. HSE can be an external clock circuitry, temperature-compensated crystal oscillator (TCXO) or ordinary crystal resonator. It accepts all frequencies from 1MHz to 24MHz. Lastly, LSI clock is also an independent internal RC oscillator-based clock source that is mainly intended for idle or low power operating modes and the independent watchdog timer (IWDG). It has a fixed factory calibrated operating frequency of 128kHz and is not as accurate as HSI or HSE. There are also clock dividers/prescalers at various points to scale clocks as per requirement. Mainly two prescalers are what we need – the HSI prescaler and the CPU divider. Peripherals are directly feed by the main clock source. Additionally, there is a clock output pin (CCO) that outputs a given clock frequency. It can be used to clock another micro, generate clock for other devices like logic ICs. It can also be used as a free oscillator or perform clock performance tests. There's fail-safe clock security that makes HSI backup of HSE. Should HSE fail, HSI takes over automatically. Check the internal block diagram below:
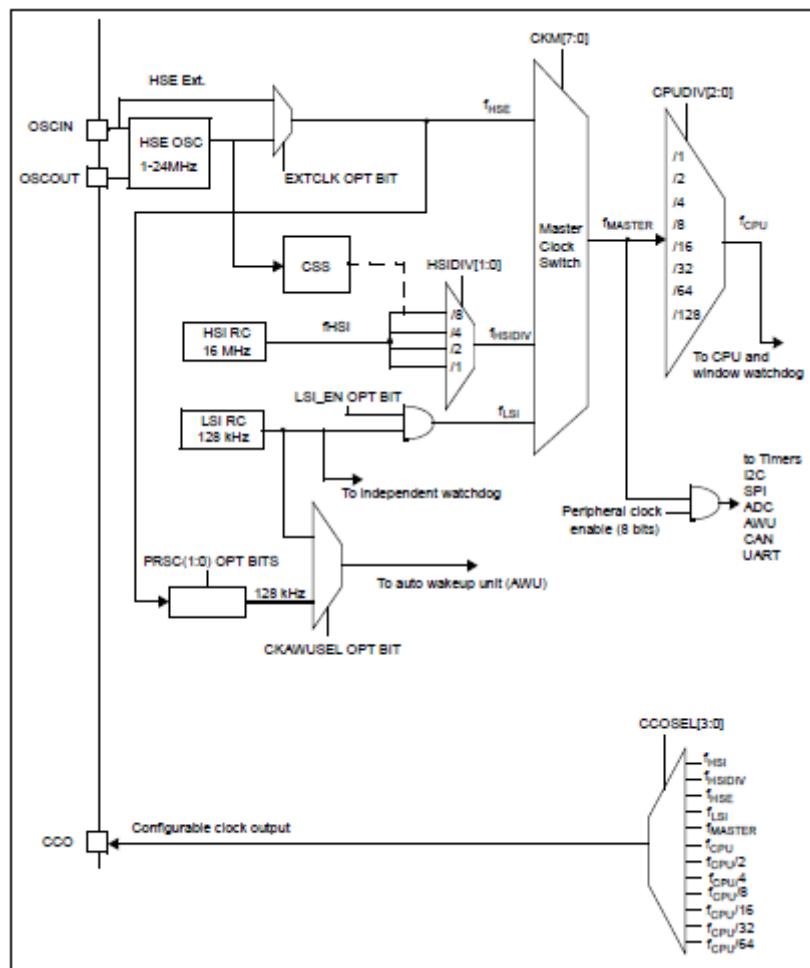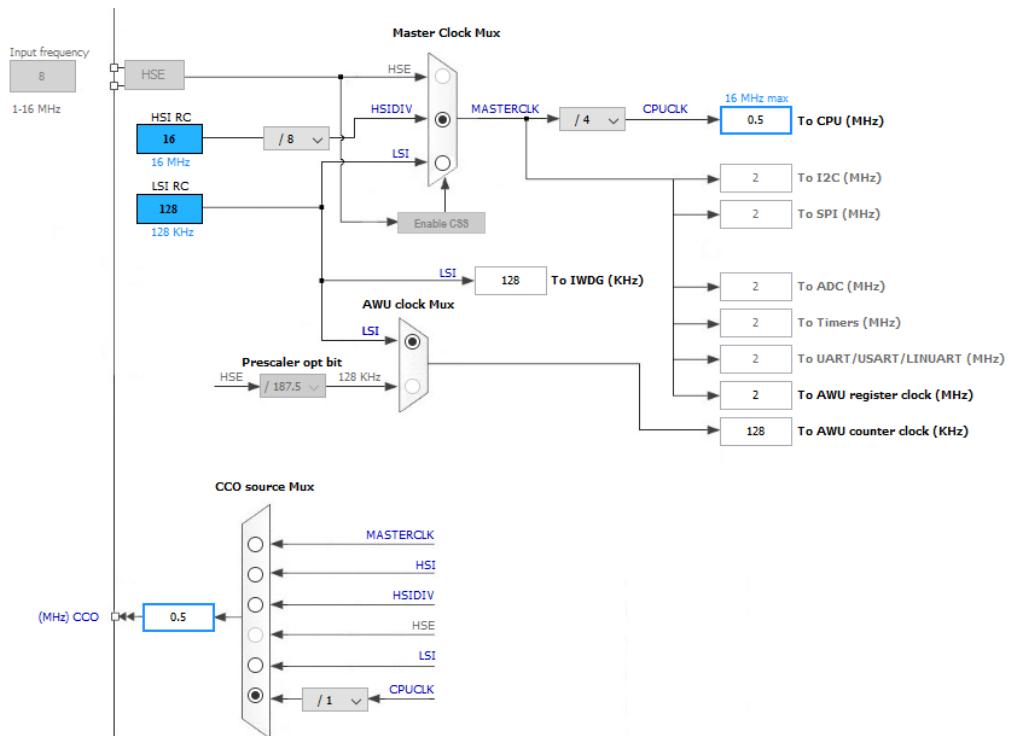
Hardware Connection
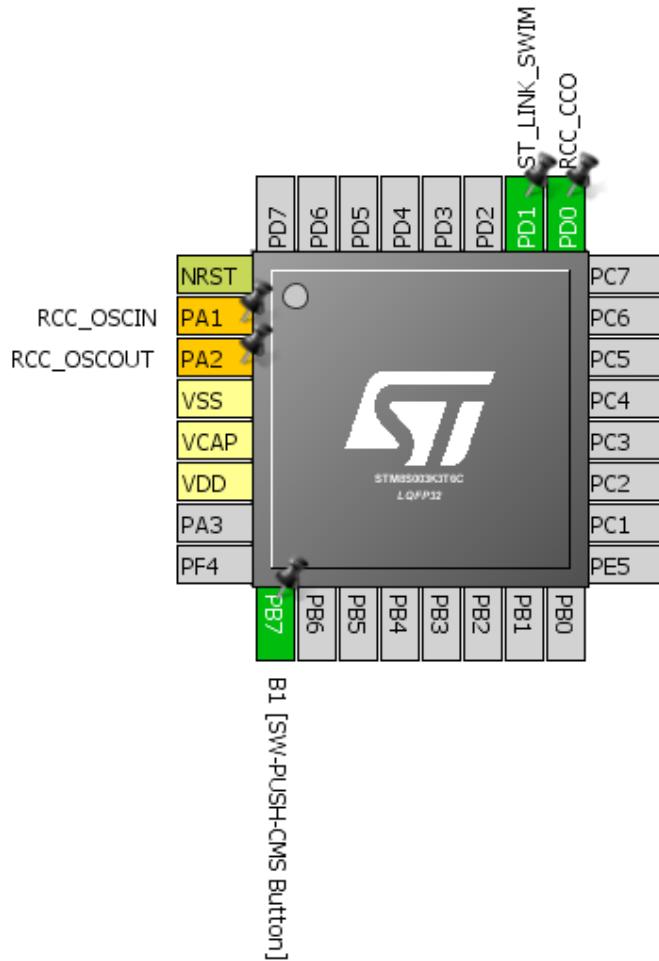
Code Example

The code example below demonstrates how to run the CPU at 2MHz clock using HSI and extract 500kHz clock output from CCO pin using CCO output selection. HSE is divided by 8, i.e. 16MHz divided by 8 equals 2MHz. This 2MHz is the master clock source and further divided four times to get 500kHz.

Note CCO pin is only available in some pins. For example, in STM8S003K3 this pin is either PD0 pin or PC4. We will need to override the default function of PD0 pin to favour for CCO output. To do so, we will need to change Alternate Function (*AFR5*) configuration bit during code upload.

```c
#include "STM8S.h"


#define LED_pin          GPIO_PIN_0
#define LED_port         GPIOD


void setup(void);
void clock_setup(void);
void GPIO_setup(void);


void main(void)
{
    setup();
    GPIO_WriteLow(LED_port, LED_pin);

    while(TRUE)
    {
    };
}


void setup(void)
{
    clock_setup();
    GPIO_setup();
}


void clock_setup(void)
{
    CLK_DeInit();

    CLK_HSECmd(DISABLE);
    CLK_LSICmd(DISABLE);
    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

    CLK_ClockSwitchCmd(ENABLE);
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
    CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV4);

    CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
    DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

    CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
```

```
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);

        CLK_CCOConfig(CLK_OUTPUT_CPU);
        CLK_CCOCmd(ENABLE);
        while(CLK_GetFlagStatus(CLK_FLAG_CCORDY) == FALSE);
}


void GPIO_setup(void)
{
        GPIO_DeInit(LED_port);
        GPIO_Init(LED_port, LED_pin, GPIO_MODE_OUT_OD_HIZ_FAST);
}
```

Explanation

The full explanation of this code is given in the last segment of this article. The only thing I'll describe here is this part:

```
CLK_CCOConfig(CLK_OUTPUT_CPU);
CLK_CCOCmd(ENABLE);
while(CLK_GetFlagStatus(CLK_FLAG_CCORDY) == FALSE);
```

These lines select the clock source that the CCO pin will output, enable the CCO module and wait for it to stabilize. Here I selected the CCO to output CPU clock.

Demo
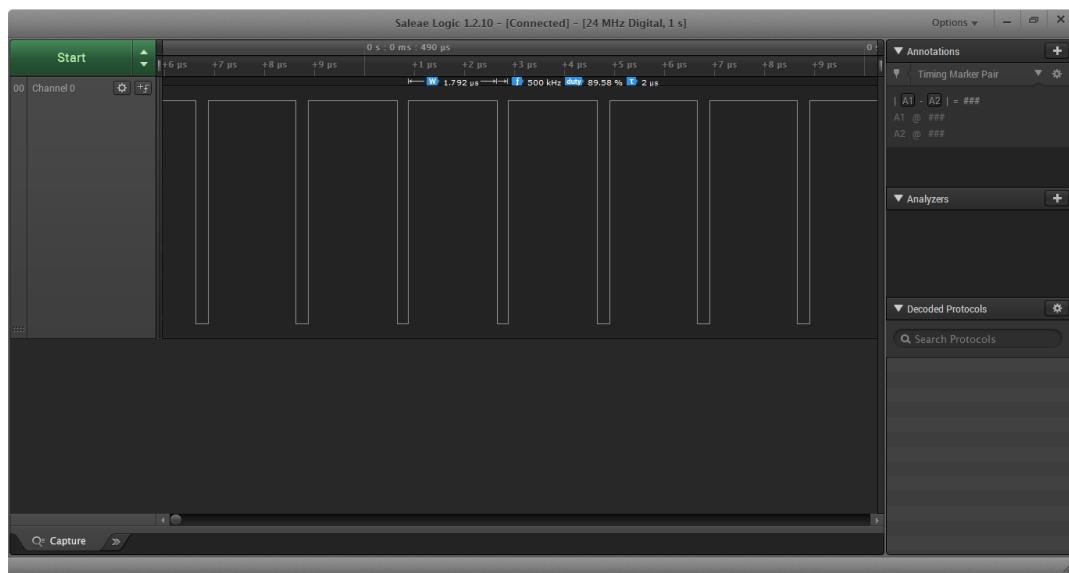
Video link: https://www.youtube.com/watch?v=IeLUc_s3jBE.

External Interrupt (EXTI)

External interrupt is an additional feature of GPIOs in input mode. It makes a micro to respond instantly to changes done on it GPIO input pin(s) by an external events/triggers, skipping other tasks. External interrupts are useful in many scenarios. For instance, an emergency button. Consider the case of a treadmill. You are running at a speed and suddenly you get an ache in one of your ankles. You'll surely want to stop running immediately. Rather decreasing speed in small steps, you can hit the emergency button to quickly stop the treadmill. The emergency button will interrupt all other processes and immediately instruct the treadmill's CPU to decrease speed faster than otherwise possible. Thus, it has high priority over other processes.

In most 8-bit micros, there are few external interrupt pins and very limited options are available for them but that's not the case with STM's micros. In STM8s, almost all GPIO pins have independent external interrupt capability with input Schmitt triggers. Additionally, there's interrupt controller to set interrupt priority.
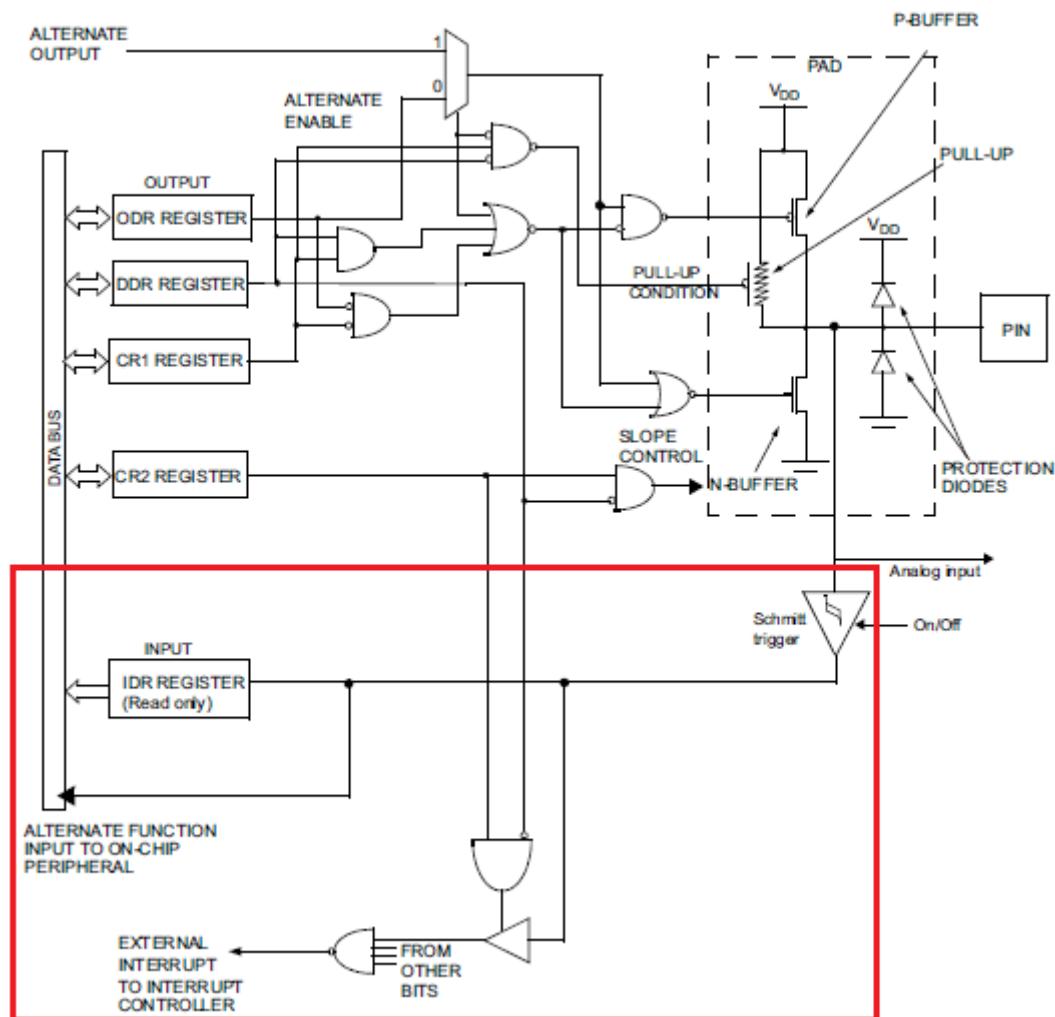
Hardware Connection
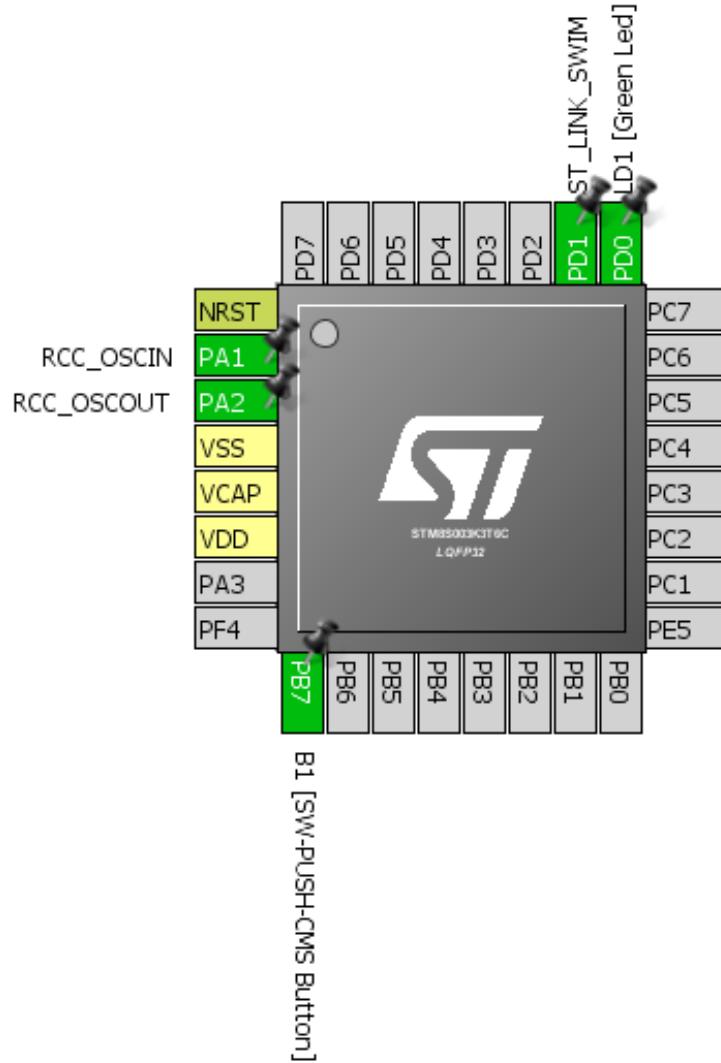
Code Example

We will do the same variable flash rate LED blinking example as in the GPIO example but this time with the DISCO board's button in external interrupt mode. The code here needs special attention as now we will see how interrupts are configured and coded. This code is way different than anything I saw before. I'm telling so because you'll need to follow certain steps unlike other compilers. In other compilers I used so far, all we do is create the interrupt function and tell the compiler the interrupt vector address. Same here too but too many steps involved.

Now check the interrupt vector table of STM8S003 below:

IRQ no.	Source block	Description	Wakeup from Halt mode	Wakeup from Active-halt mode	Vector address
-	RESET	Reset	Yes	Yes	0x00 8000
-	TRAP	Software interrupt	-	-	0x00 8004
0	TLI	External top level interrupt	-	-	0x00 8008
1	AWU	Auto wake up from halt	-	Yes	0x00 800C
2	CLK	Clock controller	-	-	0x00 8010
3	EXTI0	Port A external interrupts	Yes[1]	Yes[1]	0x00 8014
4	EXTI1	Port B external interrupts	Yes	Yes	0x00 8018
5	EXTI2	Port C external interrupts	Yes	Yes	0x00 801C
6	EXTI3	Port D external interrupts	Yes	Yes	0x00 8020
7	EXTI4	Port E external interrupts	Yes	Yes	0x00 8024
8	-	Reserved			0x00 8028
9	-	Reserved			0x00 802C
10	SPI	End of transfer	Yes	Yes	0x00 8030
11	TIM1	TIM1 update/overflow/underflow/ trigger/break	-	-	0x00 8034
12	TIM1	TIM1 capture/compare	-	-	0x00 8038
13	TIM2	TIM2 update /overflow	-	-	0x00 803C
14	TIM2	TIM2 capture/compare	-	-	0x00 8040
15	-	Reserved			0x00 8044
16	-	Reserved			0x00 8048
17	UART1	Tx complete	-	-	0x00 804C
18	UART1	Receive register DATA FULL	-	-	0x00 8050
19	I²C	I²C interrupt	Yes	Yes	0x00 8054
20	-	Reserved			0x00 8058
21	-	Reserved			0x00 805C
22	ADC1	ADC1 end of conversion/analog watchdog interrupt	-	-	0x00 8060
23	TIM4	TIM4 update/overflow	-	-	0x00 8064
24	Flash	EOP/WR_PG_DIS	-	-	0x00 8068
		Reserved			0x00 806C to 0x00 807C

1. Except PA1

You'll find this table not in the reference manual but in the device's datasheet. This table varies with devices and so be sure of correct datasheet. The DISCO board's button is connected to PB7 and so clearly, we will need **IRQ4**, i.e. EXTI1 or PORTB external interrupts. All external interrupts on GPIOB pin are masked in this vector address.

Please note that codes that use peripheral interrupts need **stm8s_it.h** and **stm8s_it.c** files. Therefore, add them if you are to use interrupts.

main.c

```c
#include "stm8s.h"


bool state = FALSE;


void GPIO_setup(void);
void EXTI_setup(void);
void clock_setup(void);


void main(void)
{
    GPIO_setup();
    EXTI_setup();
    clock_setup();

    do
    {
        GPIO_WriteReverse(GPIOD, GPIO_PIN_0);

        if(state == TRUE)
        {
            delay_ms(100);
        }
        else
        {
            delay_ms(1000);
        }

    }while (TRUE);
}


void GPIO_setup(void)
{
    GPIO_DeInit(GPIOB);
    GPIO_Init(GPIOB, GPIO_PIN_7, GPIO_MODE_IN_PU_IT);

    GPIO_DeInit(GPIOD);
    GPIO_Init(GPIOD, GPIO_PIN_0, GPIO_MODE_OUT_PP_LOW_FAST);
}


void EXTI_setup(void)
{
    ITC_DeInit();
    ITC_SetSoftwarePriority(ITC_IRQ_PORTB, ITC_PRIORITYLEVEL_0);

    EXTI_DeInit();
    EXTI_SetExtIntSensitivity(EXTI_PORT_GPIOB, EXTI_SENSITIVITY_FALL_ONLY);
    EXTI_SetTLISensitivity(EXTI_TLISENSITIVITY_FALL_ONLY);

    enableInterrupts();
}
```

```c
void clock_setup(void)
{
        CLK_DeInit();

        CLK_HSECmd(DISABLE);
        CLK_LSICmd(DISABLE);
        CLK_HSICmd(ENABLE);
        while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

        CLK_ClockSwitchCmd(ENABLE);
        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
        CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

        CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
        DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}
```

stm8_interrupt_vector.c

```c
/*      BASIC INTERRUPT VECTOR TABLE FOR STM8 devices
 *      Copyright (c) 2007 STMicroelectronics
 */

 #include "stm8s_it.h"


typedef void @far (*interrupt_handler_t)(void);

struct interrupt_vector {
        unsigned char interrupt_instruction;
        interrupt_handler_t interrupt_handler;
};

//@far @interrupt void NonHandledInterrupt (void)
//{
        /* in order to detect unexpected events during development,
           it is recommended to set a breakpoint on the following instruction
        */
//      return;
//}

extern void _stext();      /* startup routine */


struct interrupt_vector const _vectab[] = {
        {0x82, (interrupt_handler_t)_stext}, /* reset */
        {0x82, NonHandledInterrupt}, /* trap  */
        {0x82, NonHandledInterrupt}, /* irq0  */
        {0x82, NonHandledInterrupt}, /* irq1  */
```

```
        {0x82, NonHandledInterrupt}, /* irq2  */
        {0x82, NonHandledInterrupt}, /* irq3  */
        {0x82, (interrupt_handler_t)EXTI1_IRQHandler}, /* irq4  */
        {0x82, NonHandledInterrupt}, /* irq5  */
        {0x82, NonHandledInterrupt}, /* irq6  */
        {0x82, NonHandledInterrupt}, /* irq7  */
        {0x82, NonHandledInterrupt}, /* irq8  */
        {0x82, NonHandledInterrupt}, /* irq9  */
        {0x82, NonHandledInterrupt}, /* irq10 */
        {0x82, NonHandledInterrupt}, /* irq11 */
        {0x82, NonHandledInterrupt}, /* irq12 */
        {0x82, NonHandledInterrupt}, /* irq13 */
        {0x82, NonHandledInterrupt}, /* irq14 */
        {0x82, NonHandledInterrupt}, /* irq15 */
        {0x82, NonHandledInterrupt}, /* irq16 */
        {0x82, NonHandledInterrupt}, /* irq17 */
        {0x82, NonHandledInterrupt}, /* irq18 */
        {0x82, NonHandledInterrupt}, /* irq19 */
        {0x82, NonHandledInterrupt}, /* irq20 */
        {0x82, NonHandledInterrupt}, /* irq21 */
        {0x82, NonHandledInterrupt}, /* irq22 */
        {0x82, NonHandledInterrupt}, /* irq23 */
        {0x82, NonHandledInterrupt}, /* irq24 */
        {0x82, NonHandledInterrupt}, /* irq25 */
        {0x82, NonHandledInterrupt}, /* irq26 */
        {0x82, NonHandledInterrupt}, /* irq27 */
        {0x82, NonHandledInterrupt}, /* irq28 */
        {0x82, NonHandledInterrupt}, /* irq29 */
};
```

Now check the top parts of the *stm8s_it.h* and *stm8s_it.c* files respectively.

stm8s_it.h

```
27
28      /* Define to prevent recursive inclusion -------------------------------------*/
29      #ifndef __STM8S_IT_H
30        #define __STM8S_IT_H
31
32      @far @interrupt void EXTI1_IRQHandler(void);
33
34      /* Includes ------------------------------------------------------------------*/
35        #include "stm8s.h"
36
37      /* Exported types ------------------------------------------------------------*/
38      /* Exported constants --------------------------------------------------------*/
39      /* Exported macro ------------------------------------------------------------*/
40      /* Exported functions --------------------------------------------------------*/
41      #ifdef _COSMIC_
42        void _stext(void); /* RESET startup routine */
43        INTERRUPT void NonHandledInterrupt(void);
44      #endif /* _COSMIC_ */
```

stm8s_it.c

These must be coded.

Explanation

Most part of the code is same as previous codes and so I won't be going through them again. However, there's something new:

```c
void EXTI_setup(void)
{
    ITC_DeInit();
    ITC_SetSoftwarePriority(ITC_IRQ_PORTB, ITC_PRIORITYLEVEL_0);

    EXTI_DeInit();
    EXTI_SetExtIntSensitivity(EXTI_PORT_GPIOB, EXTI_SENSITIVITY_FALL_ONLY);
    EXTI_SetTLISensitivity(EXTI_TLISENSITIVITY_FALL_ONLY);

    enableInterrupts();
}
```

This function is where we are setting up the external interrupt. The first two lines deinitiate the interrupt controller and set priority while initiating it. It is not mandatory unless you want to set interrupt priority. Then we configure the external interrupt on PORTB pins. We also set the edge that will invoke an interrupt. Finally, we enable global interrupt. There goes the *main.c* file

Now it's time to explain the *stm8_interrupt_vector.c* file. The top part of this file must include this line *#include "stm8s_it.h"*. It must also have the following section commented out:

```c
//@far @interrupt void NonHandledInterrupt (void)
//{
    /* in order to detect unexpected events during development,
       it is recommended to set a breakpoint on the following instruction
    */
//    return;
//}
```

We need to let our compiler know the name of the function that it should call when a particular interrupt is triggered. There are two parts for that. Firstly, the interrupt vector address and secondly the name of the function. This is reason for this line:

```
{0x82, (interrupt_handler_t)EXTI1_IRQHandler}, /* irq4 */
```

Lastly, the **stm8s_it.h** and **stm8s_it.c** files contain the prototype and the function that will execute the interrupt service routine (ISR). In our case, the ISR will change the logic state of the Boolean type variable **state**. This will alter that flashing rate in the main loop.

Demo

Video link: https://www.youtube.com/watch?v=P6qdmWgH-Ls.
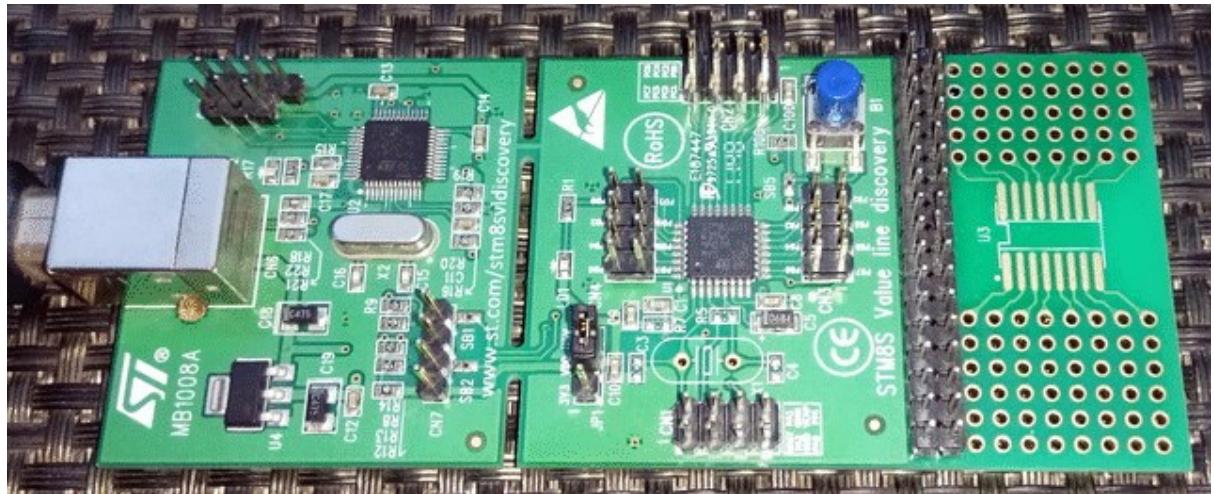
Beeper (BEEP)

The beeper hardware is a sound generation unit. This is a hardware not found in other micros and is useful in scenarios where we need an audible output. An alarm is a good example. The beeper unit uses LSI to generate 1kHz, 2kHz and 4kHz square wave outputs that can be directly feed to a small piezo tweeter (not buzzer). In most STM8 micros, the beeper module's I/O pin (PD4) is not accessible unless alternate function configuration bit is altered during code upload. However, there are few exceptional chips like the STM8S003 in which we don't need to change any configuration bit at all. The beeper module has dependencies with the Auto-Wake-Up (AWU) module.

Hardware Connection
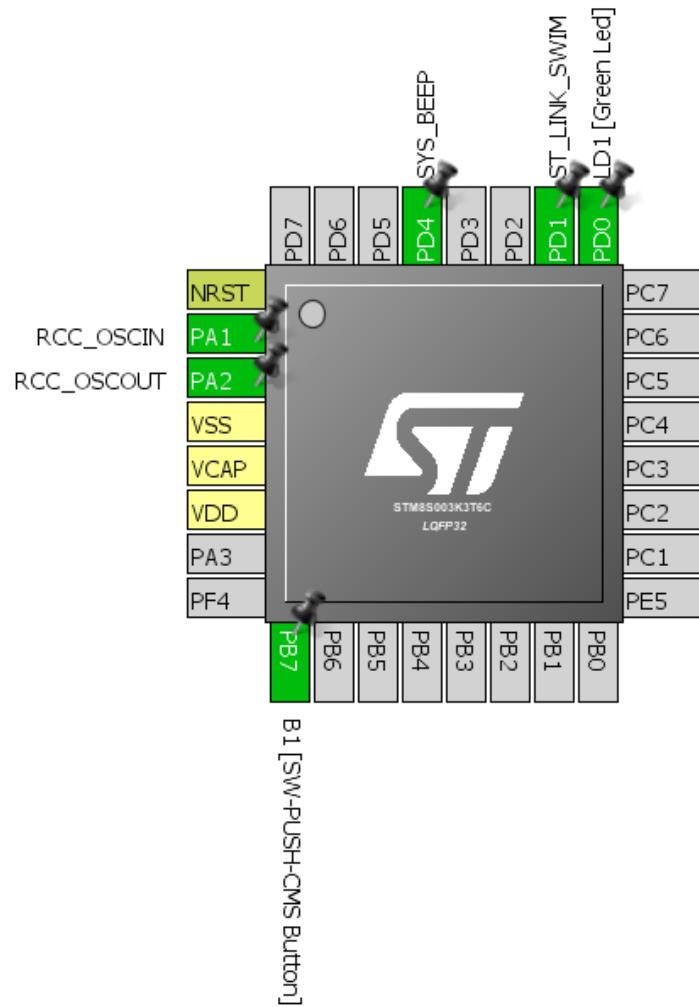
Code Example

```c
#include "STM8S.h"


void clock_setup(void);
void GPIO_setup(void);
void beeper_setup(void);


void main(void)
{
    clock_setup();
    GPIO_setup();
    beeper_setup();

    while(TRUE)
    {
        GPIO_WriteLow(GPIOD, GPIO_PIN_0);
        BEEP_Cmd(ENABLE);
        delay_ms(200);

        GPIO_WriteHigh(GPIOD, GPIO_PIN_0);
        BEEP_Cmd(DISABLE);
        delay_ms(200);
    };
}


void clock_setup(void)
{
    CLK_DeInit();

    CLK_HSECmd(DISABLE);
    CLK_LSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_LSIRDY) == FALSE);
    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

    CLK_ClockSwitchCmd(ENABLE);
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
    CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

    CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
    DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

     CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
     CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
     CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
     CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, ENABLE);
     CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
     CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
     CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
     CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
```

```
{
    GPIO_DeInit(GPIOD);
    GPIO_Init(GPIOD, GPIO_PIN_0, GPIO_MODE_OUT_OD_LOW_FAST);
    GPIO_Init(GPIOD, GPIO_PIN_4, GPIO_MODE_OUT_PP_HIGH_FAST);
}


void beeper_setup(void)
{
    BEEP_DeInit();
    BEEP_LSICalibrationConfig(128000);
    BEEP_Init(BEEP_FREQUENCY_2KHZ);
}
```

Explanation

As stated earlier beeper module is dependent on the AWU module and so we need to enable this module's peripheral clock:

```
CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, ENABLE);
```

We need to set the beeper port pin as an output pin:

```
GPIO_Init(GPIOD, GPIO_PIN_4, GPIO_MODE_OUT_PP_HIGH_FAST);
```

Configuring the beeper is straight. Just like other peripherals, we deinitialize it first and set both LSI and beep frequency. Optionally we can calibrate the LSI.

```
void beeper_setup(void)
{
    BEEP_DeInit();
    BEEP_LSICalibrationConfig(128000);
    BEEP_Init(BEEP_FREQUENCY_2KHZ);
}
```

To activate/deactivate the beeper we need to use the following instructions:

```
BEEP_Cmd(ENABLE);
....
BEEP_Cmd(DISABLE);
```

Demo

Video link: https://www.youtube.com/watch?v=LDPtULsJao8.
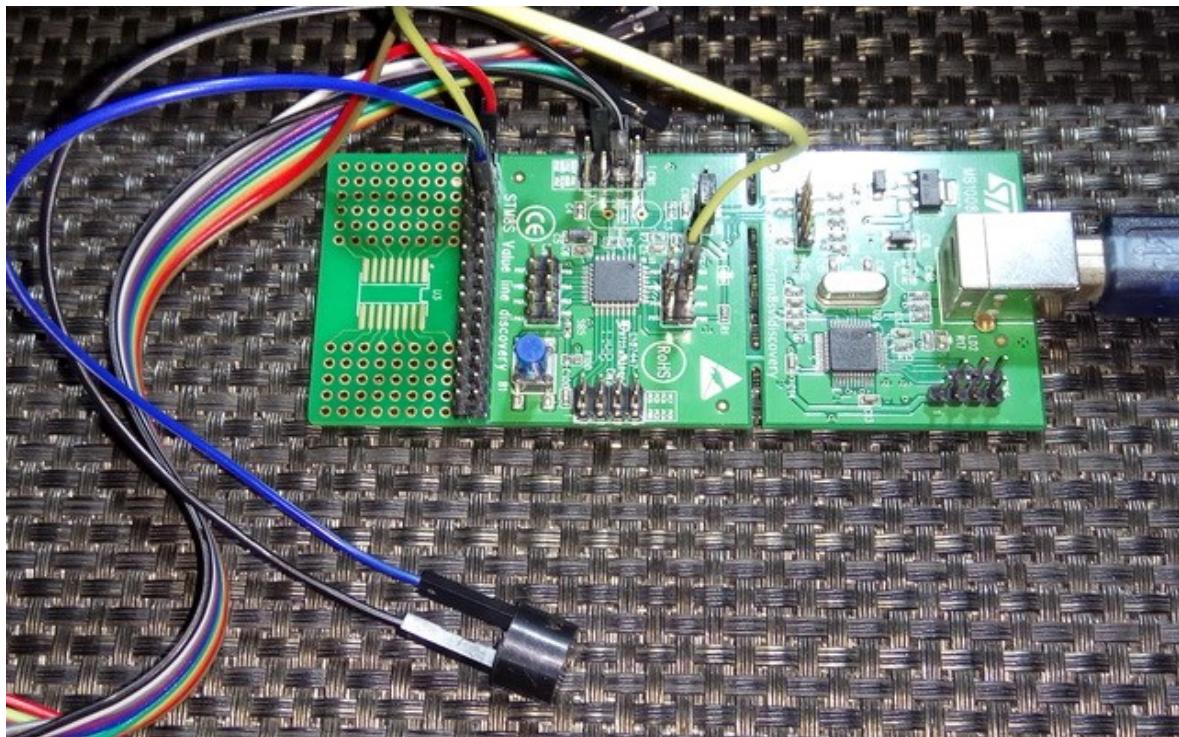
Alphanumeric LCD

Alphanumeric displays are the most common and basic form of displays after seven segments and LED displays. They are useful for projecting multiple data quickly in ways that are otherwise difficult with other kinds of displays.

To interface a LCD with a STM8 micro, we need LCD library. The STM8 SPL does not have a library for such displays and we need to code it on our own. Interfacing a LCDs are not difficult tasks as no special hardware is required to drive LCDs other than GPIOs. However, there are some tasks needed in the software end. We need to include the library files. The process of library inclusion is discussed in the later part of this article as it needs some special attentions. The example I'm sharing here uses 6 GPIO pins from GPIOC. It is an example of 4-bit LCD interfacing – four data pins (D4 – D7), RS and EN pins are all that are needed. The read-write (R/W) pin of the LCD is connected to ground. The layout is as shown below.

Hardware Connection

Code Example

lcd.h
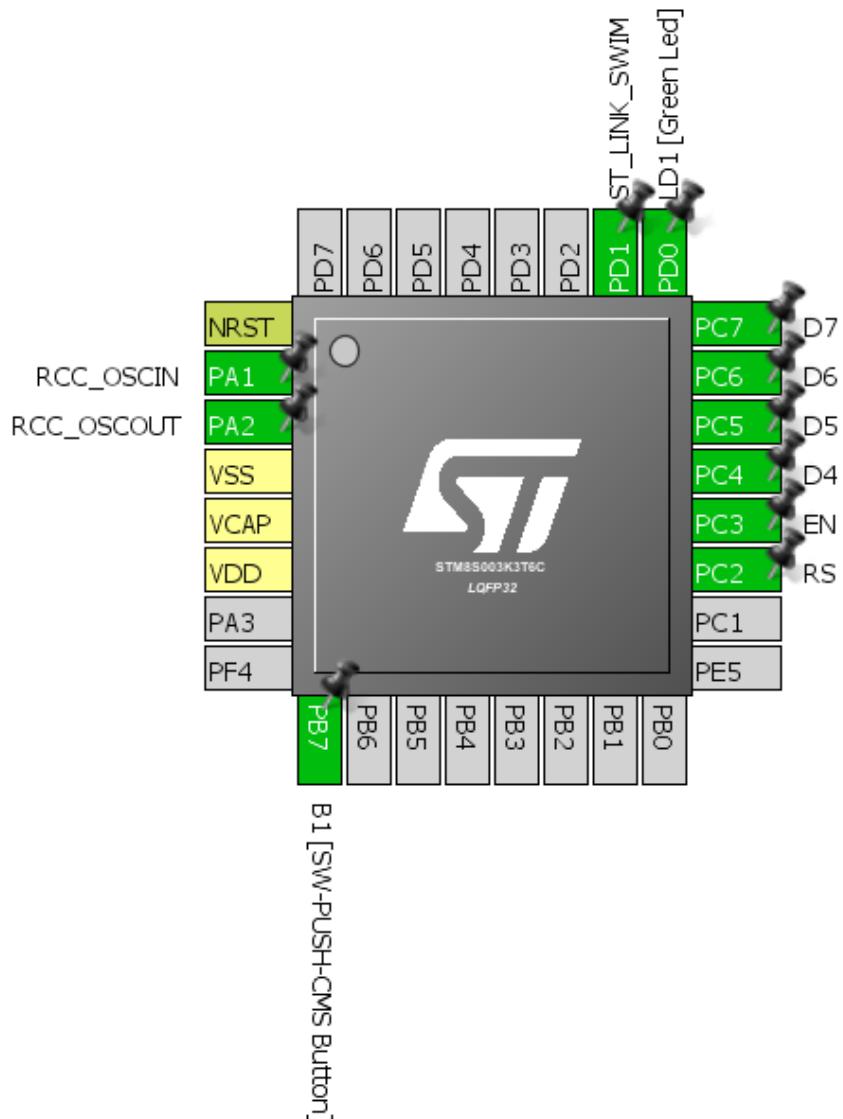
```c
#include "stm8s.h"


#define LCD_PORT                GPIOD

#define LCD_RS                  GPIO_PIN_2
#define LCD_EN                  GPIO_PIN_3
#define LCD_DB4                 GPIO_PIN_4
#define LCD_DB5                 GPIO_PIN_5
#define LCD_DB6                 GPIO_PIN_6
#define LCD_DB7                 GPIO_PIN_7


#define clear_display           0x01
#define goto_home               0x02

#define cursor_direction_inc    (0x04 | 0x02)
#define cursor_direction_dec    (0x04 | 0x00)
#define display_shift           (0x04 | 0x01)
#define display_no_shift        (0x04 | 0x00)

#define display_on              (0x08 | 0x04)
#define display_off             (0x08 | 0x02)
#define cursor_on               (0x08 | 0x02)
#define cursor_off              (0x08 | 0x00)
#define blink_on                (0x08 | 0x01)
#define blink_off               (0x08 | 0x00)

#define _8_pin_interface        (0x20 | 0x10)
#define _4_pin_interface        (0x20 | 0x00)
#define _2_row_display          (0x20 | 0x08)
#define _1_row_display          (0x20 | 0x00)
#define _5x10_dots              (0x20 | 0x40)
#define _5x7_dots               (0x20 | 0x00)

#define DAT                     1
#define CMD                     0


void LCD_GPIO_init(void);
void LCD_init(void);
void LCD_send(unsigned char value, unsigned char mode);
void LCD_4bit_send(unsigned char lcd_data);
void LCD_putstr(char *lcd_string);
void LCD_putchar(char char_data);
void LCD_clear_home(void);
void LCD_goto(unsigned char  x_pos, unsigned char  y_pos);
void toggle_EN_pin(void);
void toggle_io(unsigned char lcd_data, unsigned char bit_pos, unsigned char
pin_num);
```

lcd.c

```c
#include "lcd.h"


void LCD_GPIO_init(void)
{
    GPIO_Init(LCD_PORT, LCD_RS, GPIO_MODE_OUT_PP_HIGH_FAST);
    GPIO_Init(LCD_PORT, LCD_EN, GPIO_MODE_OUT_PP_HIGH_FAST);
    GPIO_Init(LCD_PORT, LCD_DB4, GPIO_MODE_OUT_PP_HIGH_FAST);
    GPIO_Init(LCD_PORT, LCD_DB5, GPIO_MODE_OUT_PP_HIGH_FAST);
    GPIO_Init(LCD_PORT, LCD_DB6, GPIO_MODE_OUT_PP_HIGH_FAST);
    GPIO_Init(LCD_PORT, LCD_DB7, GPIO_MODE_OUT_PP_HIGH_FAST);
    delay_ms(10);
}


void LCD_init(void)
{
     LCD_GPIO_init();
    toggle_EN_pin();

    GPIO_WriteLow(LCD_PORT, LCD_RS);
    GPIO_WriteLow(LCD_PORT, LCD_DB7);
    GPIO_WriteLow(LCD_PORT, LCD_DB6);
    GPIO_WriteHigh(LCD_PORT, LCD_DB5);
    GPIO_WriteHigh(LCD_PORT, LCD_DB4);
    toggle_EN_pin();

    GPIO_WriteLow(LCD_PORT, LCD_DB7);
    GPIO_WriteLow(LCD_PORT, LCD_DB6);
    GPIO_WriteHigh(LCD_PORT, LCD_DB5);
    GPIO_WriteHigh(LCD_PORT, LCD_DB4);
    toggle_EN_pin();

    GPIO_WriteLow(LCD_PORT, LCD_DB7);
    GPIO_WriteLow(LCD_PORT, LCD_DB6);
    GPIO_WriteHigh(LCD_PORT, LCD_DB5);
    GPIO_WriteHigh(LCD_PORT, LCD_DB4);
    toggle_EN_pin();

    GPIO_WriteLow(LCD_PORT, LCD_DB7);
    GPIO_WriteLow(LCD_PORT, LCD_DB6);
    GPIO_WriteHigh(LCD_PORT, LCD_DB5);
    GPIO_WriteLow(LCD_PORT, LCD_DB4);
    toggle_EN_pin();

    LCD_send((_4_pin_interface | _2_row_display | _5x7_dots), CMD);
    LCD_send((display_on | cursor_off | blink_off), CMD);
    LCD_send(clear_display, CMD);
    LCD_send((cursor_direction_inc | display_no_shift), CMD);
}


void LCD_send(unsigned char value, unsigned char mode)
{
    switch(mode)
    {
        case DAT:
```

```c
            {
                GPIO_WriteHigh(LCD_PORT, LCD_RS);
                break;
            }
            case CMD:
            {
                GPIO_WriteLow(LCD_PORT, LCD_RS);
                break;
            }
    }

    LCD_4bit_send(value);
}


void LCD_4bit_send(unsigned char lcd_data)
{
    toggle_io(lcd_data, 7, LCD_DB7);
    toggle_io(lcd_data, 6, LCD_DB6);
    toggle_io(lcd_data, 5, LCD_DB5);
    toggle_io(lcd_data, 4, LCD_DB4);
    toggle_EN_pin();
    toggle_io(lcd_data, 3, LCD_DB7);
    toggle_io(lcd_data, 2, LCD_DB6);
    toggle_io(lcd_data, 1, LCD_DB5);
    toggle_io(lcd_data, 0, LCD_DB4);
    toggle_EN_pin();
}


void LCD_putstr(char *lcd_string)
{
    do
    {
        LCD_send(*lcd_string++, DAT);
    }while(*lcd_string != '\0');
}


void LCD_putchar(char char_data)
{
    LCD_send(char_data, DAT);
}


void LCD_clear_home(void)
{
    LCD_send(clear_display, CMD);
    LCD_send(goto_home, CMD);
}


void LCD_goto(unsigned char  x_pos, unsigned char  y_pos)
{
    if(y_pos == 0)
    {
        LCD_send((0x80 | x_pos), CMD);
    }
    else
```

```c
    {
        LCD_send((0x80 | 0x40 | x_pos), CMD);
    }
}


void toggle_EN_pin(void)
{
    GPIO_WriteHigh(LCD_PORT, LCD_EN);
    delay_ms(2);
    GPIO_WriteLow(LCD_PORT,LCD_EN);
}


void toggle_io(unsigned char lcd_data, unsigned char bit_pos, unsigned char
pin_num)
{
    bool temp = FALSE;

    temp = (0x01 & (lcd_data >> bit_pos));

    switch(temp)
    {
        case TRUE:
        {
            GPIO_WriteHigh(LCD_PORT, pin_num);
            break;
        }

        default:
        {
            GPIO_WriteLow(LCD_PORT, pin_num);
            break;
        }
    }
}
```

main.c

```c
#include "STM8S.h"
#include "lcd.h"


void clock_setup(void);
void GPIO_setup(void);


void main(void)
{
    const char txt1[] = {"MICROARENA"};
    const char txt2[] = {"SShahryiar"};
    const char txt3[] = {"STM8S003K"};
    const char txt4[] = {"Discovery"};

    unsigned char s = 0x00;

    clock_setup();
    GPIO_setup();
```

```c
    LCD_init();
    LCD_clear_home();

    LCD_goto(3, 0);
    LCD_putstr(txt1);
    LCD_goto(3, 1);
    LCD_putstr(txt2);
    delay_ms(4000);

    LCD_clear_home();

    for(s = 0; s < 9; s++)
    {
        LCD_goto((3 + s), 0);
        LCD_putchar(txt3[s]);
        delay_ms(90);
    }
    for(s = 0; s < 9; s++)
    {
        LCD_goto((3 + s), 1);
        LCD_putchar(txt4[s]);
        delay_ms(90);
    }

    while (TRUE);
}


void clock_setup(void)
{
    CLK_DeInit();

    CLK_HSECmd(DISABLE);
    CLK_LSICmd(DISABLE);
    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

    CLK_ClockSwitchCmd(ENABLE);
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
    CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

    CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
    DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

    CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
    GPIO_DeInit(LCD_PORT);
}
```

Explanation

There is very little to explain this code as it involves GPIOs only. The codes for the LCD are coded using all available info on its datasheet, just initialization and working principle. One thing to note, however, is the CPU clock speed. If the CPU clock is too fast, LCDs may not work. This is because most LCDs have a maximum working frequency of 250kHz. It is best to keep this frequency below 200kHz.

```
CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);
```

Demo

Video link: https://www.youtube.com/watch?v=TJg2Tuu4QaQ.
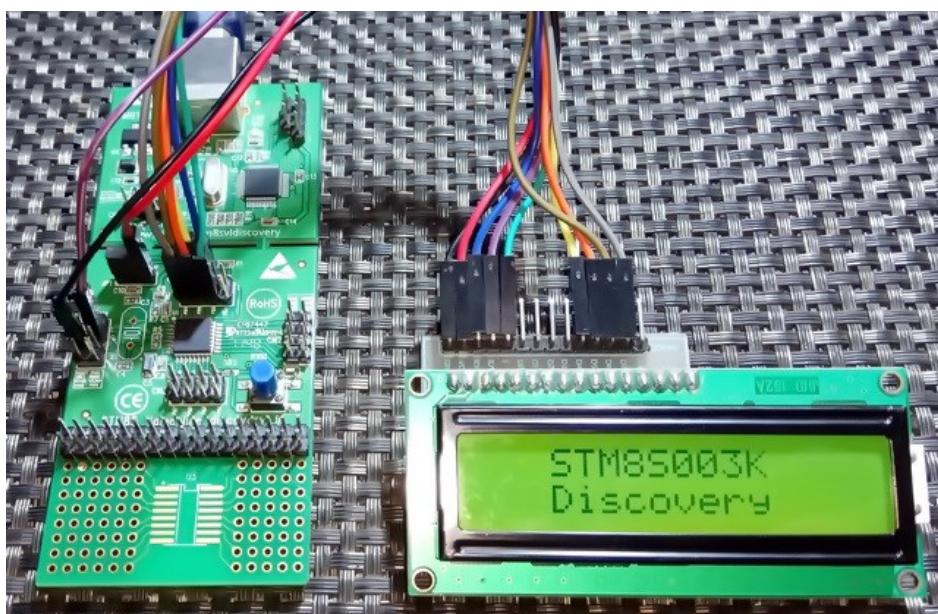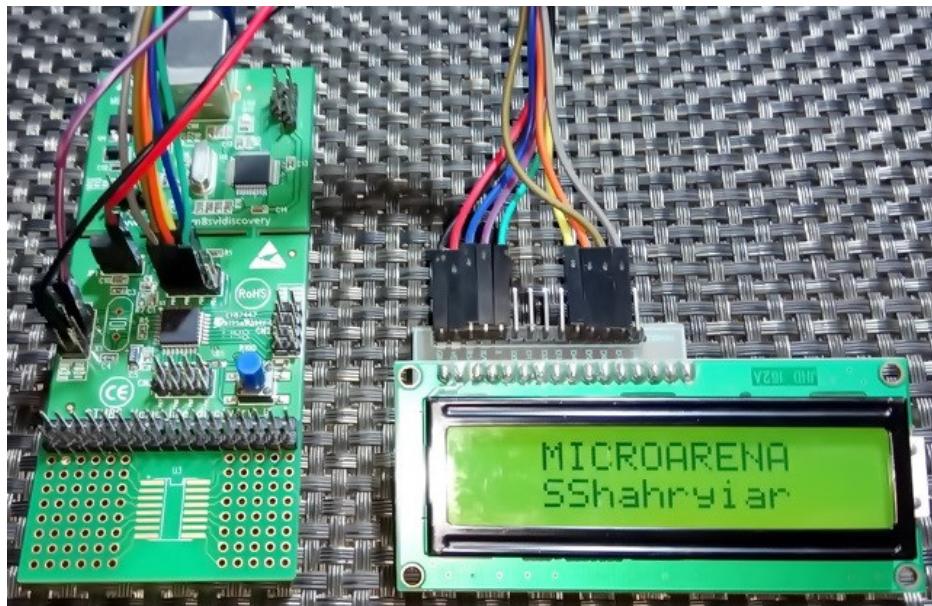
Analog-to-Digital Converter (ADC)

ADC is a very important peripheral in any modern-day microcontroller. It is used to read analogue outputs from sensors, sense voltage levels and so on. For example, we can use an ADC to read a LM35 temperature sensor. The voltage output from the sensor is proportional to temperature and so we can use the voltage info to back-calculate temperature. STM8S003K3 has four ADC channels associated with one ADC block. Other STM8 micros have more ADC channels and ADC blocks. The ADC of STM8 micros is just as same as the ADCs of other micros. There are a few additional features. Shown below is the block diagram of the STM8's ADC peripheral:
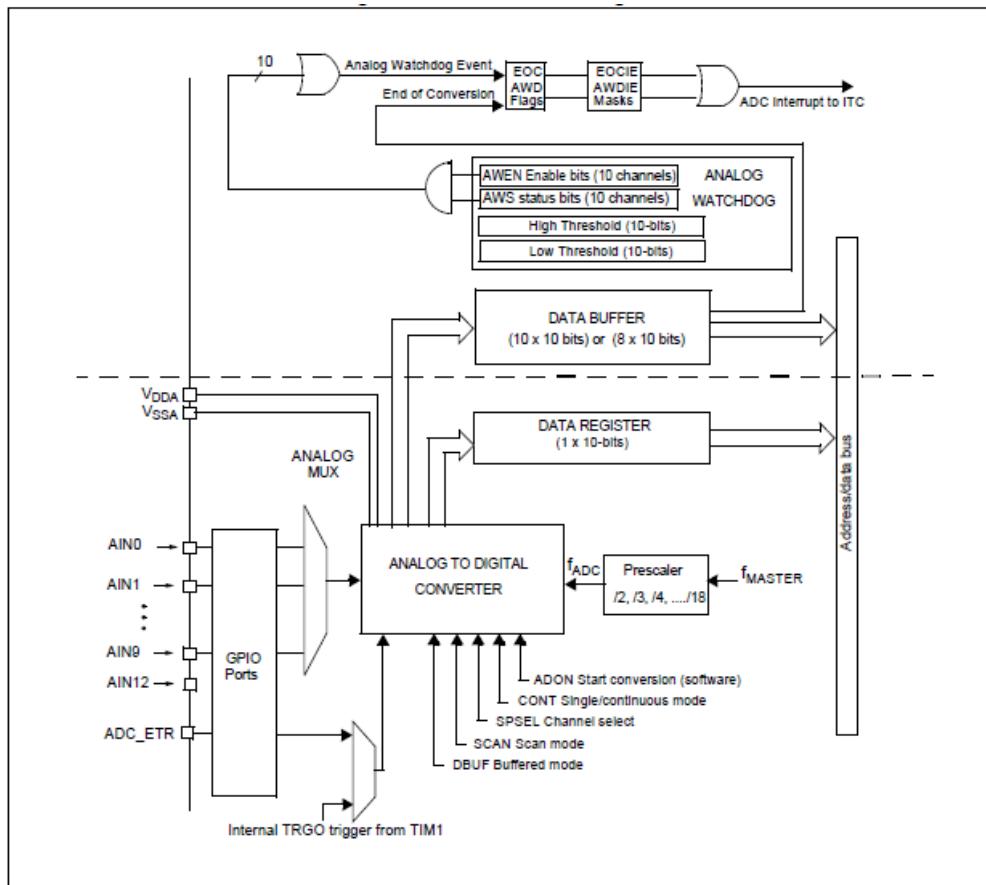
A few things must be noted before using the ADC. These enhance performance significantly:

- Input impedance should be less than 10kΩ.
- It is better to keep ADC clock within or less than 4MHz.
- Schmitt triggers must be disabled.
- Opamp-based input buffer and filter circuits are preferred if possible.
- If the ADC has reference source pins, they should be connected to a precision reference source like LM336. It is recommended to use a good LDO regulator chip otherwise.
- Unused ADC pins should not be configured or disabled. This will reduce power consumption.
- Rather taking single samples, ADC readings should be sampled at fixed regular intervals and averaged to get rid of minute fluctuations in readings.
- Right-justified data alignment should be used as it is most convenient to use.
- PCB/wire tracks leading to ADC channels must be short to reduce interference effects.

Hardware Connection
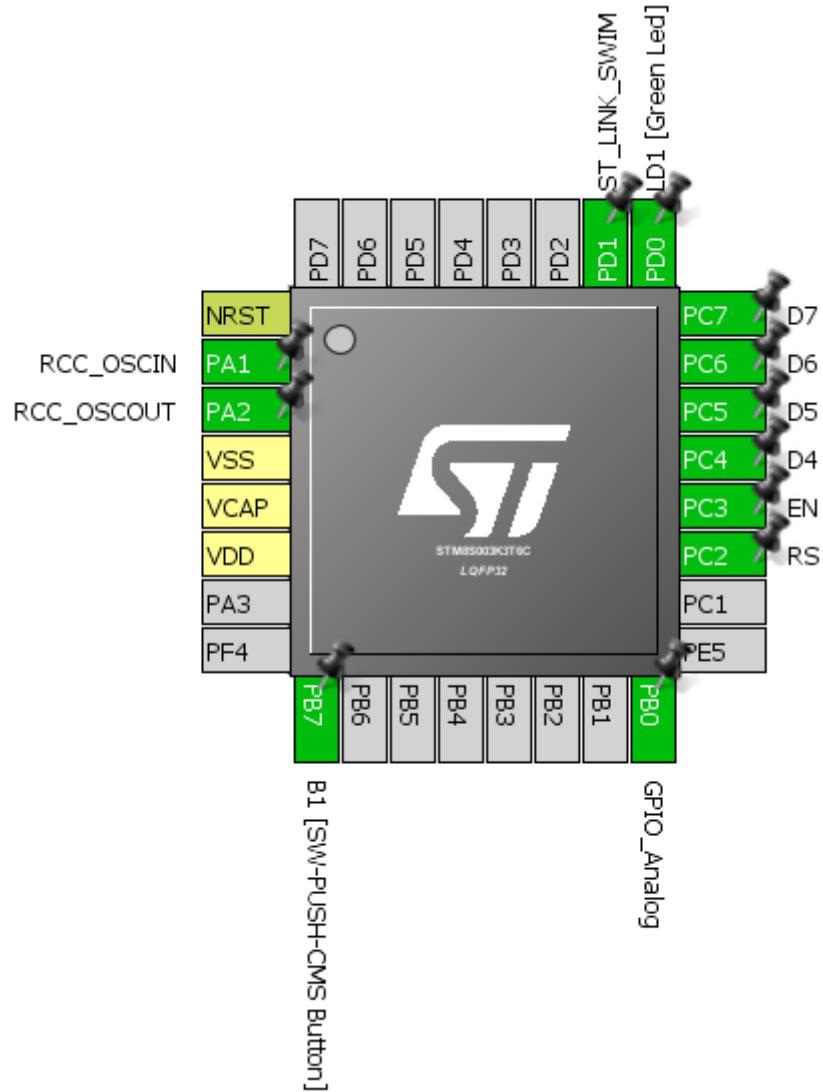
Code Example

```c
#include "STM8S.h"


void clock_setup(void);
void GPIO_setup(void);
void ADC1_setup(void);
void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned int value);


void main()
{
    unsigned int A0 = 0x0000;

    clock_setup();
```

```c
        GPIO_setup();
        ADC1_setup();

        LCD_init();
        LCD_clear_home();

        LCD_goto(0, 0);
        LCD_putstr("STM8 ADC");
        LCD_goto(0, 1);
        LCD_putstr("A0");

        while(TRUE)
        {
                ADC1_StartConversion();
                while(ADC1_GetFlagStatus(ADC1_FLAG_EOC) == FALSE);

                A0 = ADC1_GetConversionValue();
                ADC1_ClearFlag(ADC1_FLAG_EOC);

                lcd_print(4, 1, A0);
                delay_ms(90);
        };
}


void clock_setup(void)
{
        CLK_DeInit();

        CLK_HSECmd(DISABLE);
        CLK_LSICmd(DISABLE);
        CLK_HSICmd(ENABLE);
        while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

        CLK_ClockSwitchCmd(ENABLE);
        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV2);
        CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV4);

        CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
        DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
        GPIO_DeInit(GPIOB);
        GPIO_Init(GPIOB, GPIO_PIN_0, GPIO_MODE_IN_FL_NO_IT);

        GPIO_DeInit(GPIOC);
```

```c
        GPIO_DeInit(GPIOD);
        GPIO_Init(GPIOD, GPIO_PIN_3, GPIO_MODE_IN_PU_NO_IT);
}


void ADC1_setup(void)
{
        ADC1_DeInit();

        ADC1_Init(ADC1_CONVERSIONMODE_CONTINUOUS,
                  ADC1_CHANNEL_0,
                  ADC1_PRESSEL_FCPU_D18,
                  ADC1_EXTTRIG_GPIO,
                  DISABLE,
                  ADC1_ALIGN_RIGHT,
                  ADC1_SCHMITTTRIG_CHANNEL0,
                  DISABLE);

        ADC1_Cmd(ENABLE);
}


void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned int value)
{
        char chr = 0x00;

        chr = ((value / 1000) + 0x30);
        LCD_goto(x_pos, y_pos);
        LCD_putchar(chr);

        chr = (((value / 100) % 10) + 0x30);
        LCD_goto((x_pos + 1), y_pos);
        LCD_putchar(chr);

        chr = (((value / 10) % 10) + 0x30);
        LCD_goto((x_pos + 2), y_pos);
        LCD_putchar(chr);

        chr = ((value % 10) + 0x30);
        LCD_goto((x_pos + 3), y_pos);
        LCD_putchar(chr);
}
```

Explanation

First, we need to enable the peripheral clock of the ADC module:

```
CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, ENABLE);
```

Secondly, we have to set out ADC pin as a floating GPIO with no interrupt capability:

```
GPIO_Init(GPIOB, GPIO_PIN_0, GPIO_MODE_IN_FL_NO_IT);
```

ADC setup needs a few info regarding the desired ADC channel:

```
void ADC1_setup(void)
{
    ADC1_DeInit();

    ADC1_Init(ADC1_CONVERSIONMODE_CONTINUOUS,
            ADC1_CHANNEL_0,
            ADC1_PRESSEL_FCPU_D18,
            ADC1_EXTTRIG_GPIO,
            DISABLE,
            ADC1_ALIGN_RIGHT,
            ADC1_SCHMITTTRIG_CHANNEL0,
            DISABLE);

    ADC1_Cmd(ENABLE);
}
```

The second line of the above function states that we are going to use ADC channel 0 (PB0) with no Schmitt trigger. We are also not going to use external triggers from timer/GPIO modules. Since the master clock is running at 8MHz, the ADC prescaler divides the master/peripheral clock to get a sampling frequency of 444kHz. We are also going to use continuous conversion mode because we want to continually read the ADC input and don't want to measure it in certain intervals. Lastly right-justified data alignment is used as it is easy to read from such.

In the main loop, we need to start ADC conversion and wait for the conversion to complete. We are not using interrupt methods and so we need to poll if ADC conversion has completed. At the end of conversion, we can read the ADC and clear ADC End of Conversion (EOC) flag.

```
ADC1_StartConversion();
while(ADC1_GetFlagStatus(ADC1_FLAG_EOC) == FALSE);

A0 = ADC1_GetConversionValue();
ADC1_ClearFlag(ADC1_FLAG_EOC);
```

The rest of the code is about printing the ADC data on a LCD.

Video link: https://www.youtube.com/watch?v=rx68zPDEZUU.
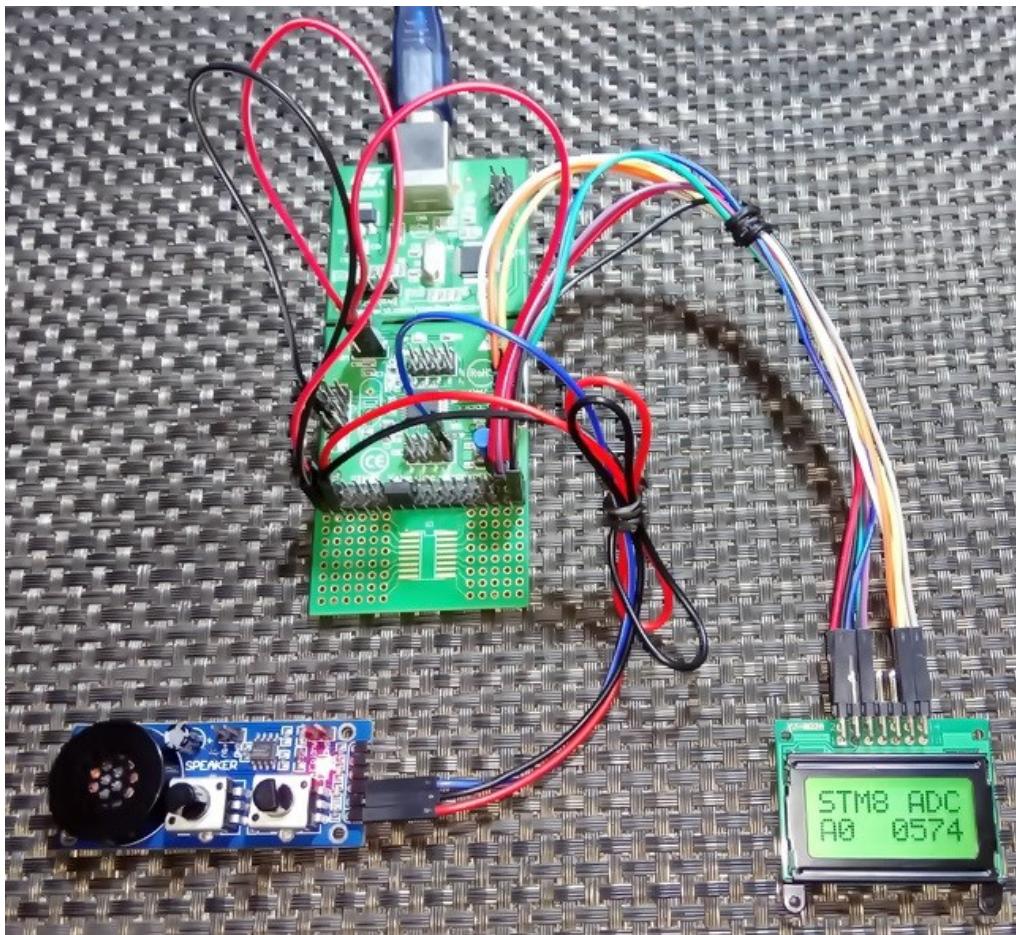
ADC Interrupt

Instead of polling for ADC's end of conversion (EOC) state, it is wise to use ADC interrupt. Just as with any hardware peripheral, interrupt methods make a system highly responsive. Interrupts like these free up the CPU for other tasks. However, it is up to the coder to determine interrupt priorities and look out for situation that may cause too many interrupts to be processed in a short while.

Hardware Connection
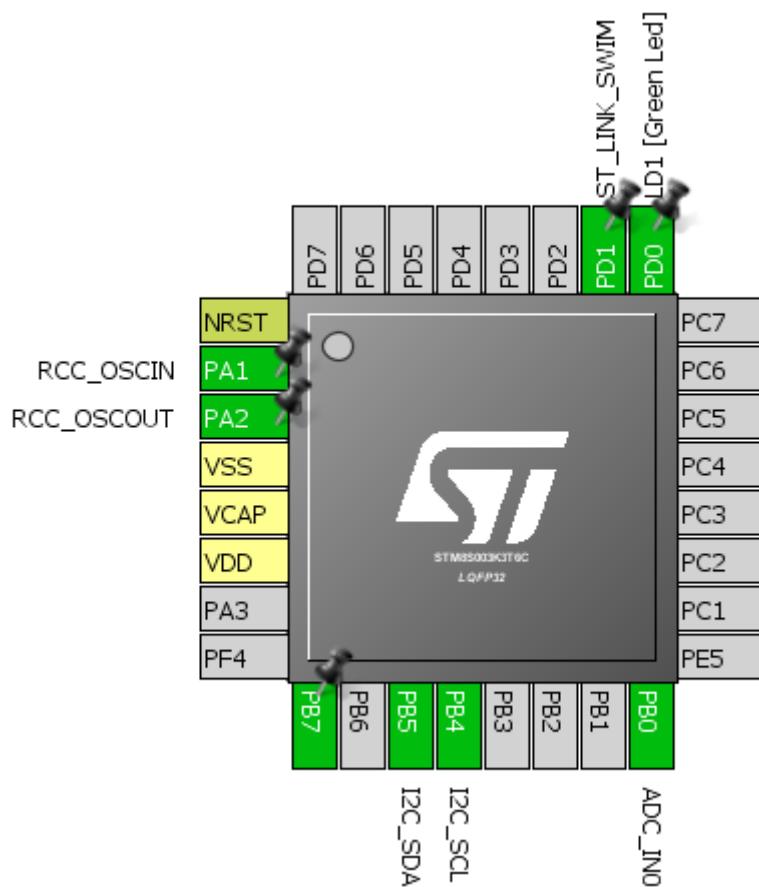
Code Example

stm8s_it.h (top part only)

```c
#ifndef __STM8S_IT_H
#define __STM8S_IT_H

@far @interrupt void ADC_IRQHandler(void);

/* Includes -------------------------------------------------------------------*/
#include "stm8s.h"
....
```

stm8s_it.c (top part only)

```c
#include "stm8s.h"
#include "stm8s_it.h"


extern unsigned int adc_value;


void ADC_IRQHandler(void)
{
      adc_value = ADC1_GetConversionValue();
      ADC1_ClearFlag(ADC1_FLAG_EOC);
}
....
```

stm8_interrupt_vector.c (shortened)

```c
#include "stm8s_it.h"


typedef void @far (*interrupt_handler_t)(void);

struct interrupt_vector {
      unsigned char interrupt_instruction;
      interrupt_handler_t interrupt_handler;
};


extern void _stext();      /* startup routine */


struct interrupt_vector const _vectab[] = {
      {0x82, (interrupt_handler_t)_stext}, /* reset */
      {0x82, NonHandledInterrupt}, /* trap  */
      {0x82, NonHandledInterrupt}, /* irq0  */
      ....
      {0x82, (interrupt_handler_t)ADC_IRQHandler}, /* irq22 */
      ....
      {0x82, NonHandledInterrupt}, /* irq29 */
};
```

main.c

```c
#include "STM8S.h"
#include "lcd.h"


unsigned int adc_value;


unsigned char bl_state;
unsigned char data_value;


void clock_setup(void);
void GPIO_setup(void);
```

```c
void ADC1_setup(void);
void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned int value);


void main(void)
{
        float mv = 0;

        clock_setup();
        GPIO_setup();
        ADC1_setup();

        LCD_init();
        LCD_clear_home();

        LCD_goto(2, 0);
        LCD_putstr("STM8 ADC ISR");
        LCD_goto(0, 1);
        LCD_putstr("A0/mV");

        while(TRUE)
        {
                ADC1_StartConversion();

                mv = (adc_value * 5000.0);
                mv /= 1023.0;

                lcd_print(7, 1, mv);
                lcd_print(12, 1, adc_value);
                GPIO_WriteReverse(GPIOD, GPIO_PIN_0);
                delay_ms(160);
        };
}


void clock_setup(void)
{
        CLK_DeInit();

        CLK_HSECmd(DISABLE);
        CLK_LSICmd(DISABLE);
        CLK_HSICmd(ENABLE);
        while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

        CLK_ClockSwitchCmd(ENABLE);
        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV4);
        CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

        CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
        DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
```

```c
}


void GPIO_setup(void)
{
    GPIO_DeInit(GPIOB);
    GPIO_Init(GPIOB, ((GPIO_Pin_TypeDef)(GPIO_PIN_0 | GPIO_PIN_1)),
GPIO_MODE_IN_FL_NO_IT);

    GPIO_DeInit(GPIOD);
    GPIO_Init(GPIOD, GPIO_PIN_0, GPIO_MODE_OUT_OD_HIZ_FAST);
}


void ADC1_setup(void)
{
    ADC1_DeInit();

    ADC1_Init(ADC1_CONVERSIONMODE_CONTINUOUS,
              ADC1_CHANNEL_0,
              ADC1_PRESSEL_FCPU_D18,
              ADC1_EXTTRIG_GPIO,
              DISABLE,
              ADC1_ALIGN_RIGHT,
              ADC1_SCHMITTTRIG_CHANNEL0,
              DISABLE);

    ADC1_ITConfig(ADC1_IT_EOCIE, ENABLE);
    enableInterrupts();
    ADC1_Cmd(ENABLE);
}


void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned int value)
{
    char chr = 0x00;

    chr = ((value / 1000) + 0x30);
    LCD_goto(x_pos, y_pos);
    LCD_putchar(chr);

    chr = (((value / 100) % 10) + 0x30);
    LCD_goto((x_pos + 1), y_pos);
    LCD_putchar(chr);

    chr = (((value / 10) % 10) + 0x30);
    LCD_goto((x_pos + 2), y_pos);
    LCD_putchar(chr);

    chr = ((value % 10) + 0x30);
    LCD_goto((x_pos + 3), y_pos);
    LCD_putchar(chr);
}
```

Explanation

This example and the first ADC example is all same except for the interrupt part. Note the last three lines below.

```c
void ADC1_setup(void)
{
    ADC1_DeInit();

    ADC1_Init(ADC1_CONVERSIONMODE_CONTINUOUS,
            ADC1_CHANNEL_0,
            ADC1_PRESSEL_FCPU_D18,
            ADC1_EXTTRIG_GPIO,
            DISABLE,
            ADC1_ALIGN_RIGHT,
            ADC1_SCHMITTTRIG_CHANNEL0,
            DISABLE);

    ADC1_ITConfig(ADC1_IT_EOCIE, ENABLE);
    enableInterrupts();
    ADC1_Cmd(ENABLE);
}
```

As can be seen, ***End-of-Conversion (EOC)*** interrupt has been enabled along with global interrupt and the ADC itself.

In the vector mapping file, ADC interrupt is set.

```c
{0x82, (interrupt_handler_t)ADC_IRQHandler}, /* irq22 */
```

When ADC EOC interrupt triggers, the ADC data buffer is read and the EOC flag is cleared.

```c
void ADC_IRQHandler(void)
{
    adc_value = ADC1_GetConversionValue();
    ADC1_ClearFlag(ADC1_FLAG_EOC);
}
```

Demo

Video link: https://www.youtube.com/watch?v=ULSZ4W7eEpU.
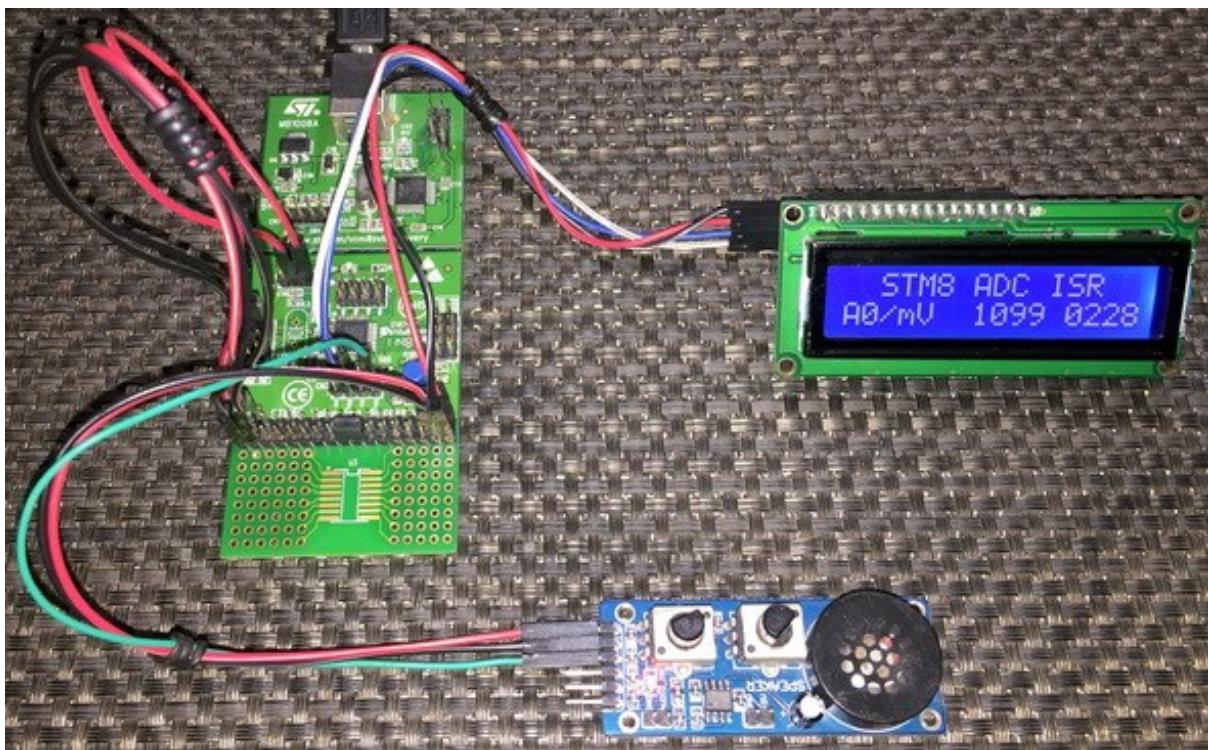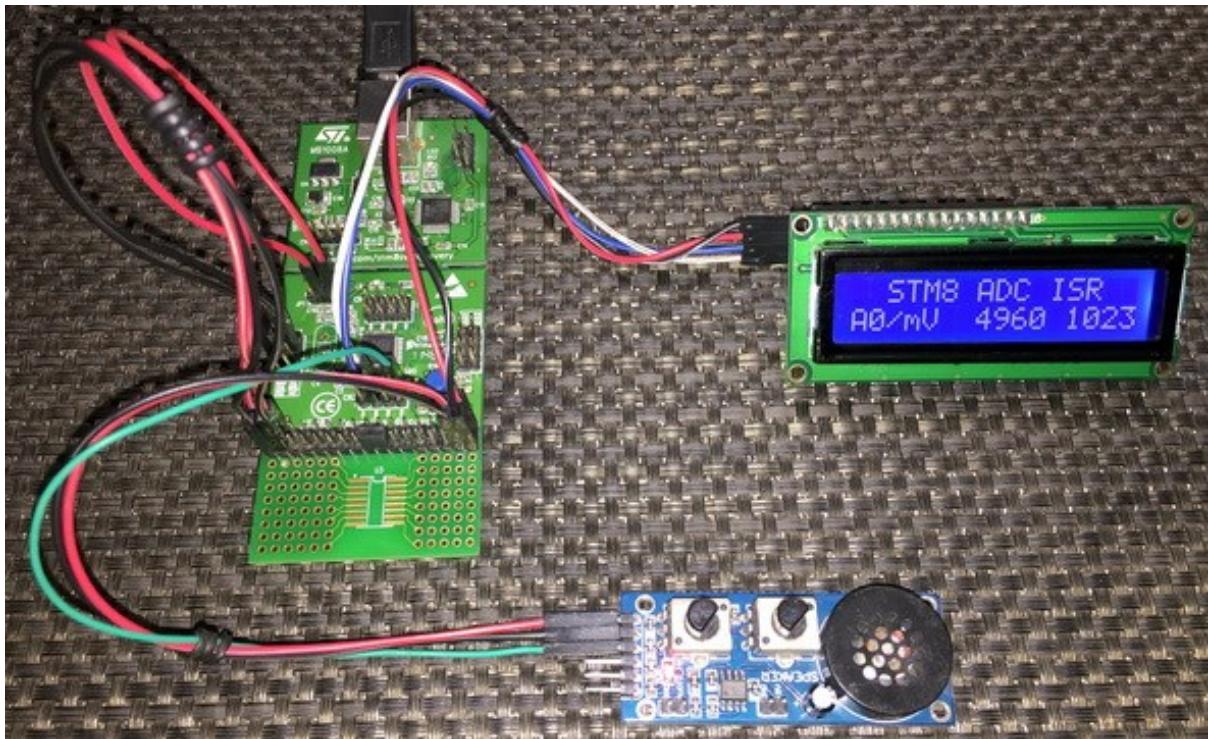
Multi Channel ADC with Scan Mode

When it is required to sense and measure multiple analog voltages, we need to use multiple ADC channels. For example, when we need to build, an electrical energy meter, we need to measure both voltage and current. However, STM8S003K3 has one ADC which is multiplexed over four channels. This does not limit it from acquiring multiple analog signals and the job is easily achieved using ADC scan mode conversion.

In scan mode, the ADC automatically scans a sequence of channels from channel 0 to channel n, keeping the results of AD conversion in data buffer registers. At the end of conversion, the results are ready to be read.

Hardware Connection
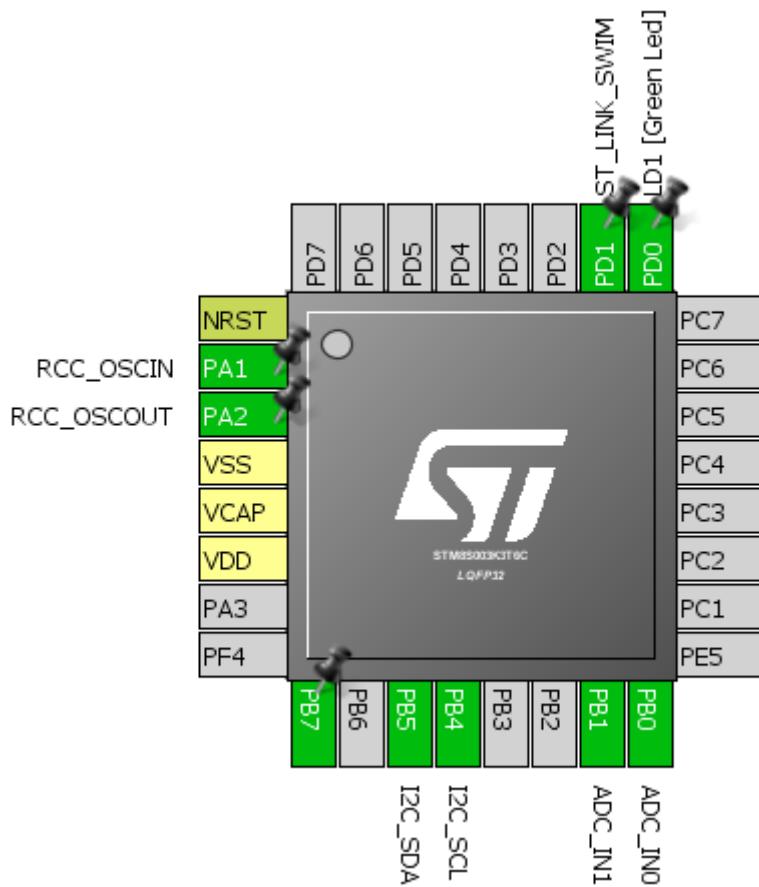
Code Example

```c
#include "STM8S.h"
#include "lcd.h"


unsigned char bl_state;
unsigned char data_value;


void clock_setup(void);
void GPIO_setup(void);
void ADC1_setup(void);
void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned int value);


void main()
{
    unsigned int A0 = 0x0000;
    unsigned int A1 = 0x0000;

    clock_setup();
    GPIO_setup();
    ADC1_setup();

    LCD_init();
    LCD_clear_home();

    LCD_goto(1, 0);
    LCD_putstr("STM8 Multi-ADC");
    LCD_goto(0, 1);
    LCD_putstr("A0");
    LCD_goto(9, 1);
    LCD_putstr("A1");

    while(TRUE)
    {
        ADC1_ScanModeCmd(ENABLE);
        ADC1_StartConversion();
        while(ADC1_GetFlagStatus(ADC1_FLAG_EOC) == FALSE);

        ADC1_ClearFlag(ADC1_FLAG_EOC);

        A0 = ADC1_GetBufferValue(0);
        A1 = ADC1_GetBufferValue(1);

        lcd_print(3, 1, A0);
        lcd_print(12, 1, A1);
        delay_ms(90);
    };
}


void clock_setup(void)
{
    CLK_DeInit();

    CLK_HSECmd(DISABLE);
```

```c
        CLK_LSICmd(DISABLE);
        CLK_HSICmd(ENABLE);
        while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

        CLK_ClockSwitchCmd(ENABLE);
        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV4);
        CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

        CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
        DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
        GPIO_DeInit(GPIOB);
        GPIO_Init(GPIOB, ((GPIO_Pin_TypeDef)(GPIO_PIN_0 | GPIO_PIN_1)),
GPIO_MODE_IN_FL_NO_IT);

        GPIO_DeInit(GPIOD);
        GPIO_Init(GPIOD, GPIO_PIN_0, GPIO_MODE_OUT_OD_HIZ_FAST);
}


void ADC1_setup(void)
{
        ADC1_DeInit();

        ADC1_Init(ADC1_CONVERSIONMODE_CONTINUOUS,
                  ADC1_CHANNEL_0,
                  ADC1_PRESSEL_FCPU_D18,
                  ADC1_EXTTRIG_GPIO,
                  DISABLE,
                  ADC1_ALIGN_RIGHT,
                  ADC1_SCHMITTTRIG_CHANNEL0,
                  DISABLE);

        ADC1_Init(ADC1_CONVERSIONMODE_CONTINUOUS,
                  ADC1_CHANNEL_1,
                  ADC1_PRESSEL_FCPU_D18,
                  ADC1_EXTTRIG_GPIO,
                  DISABLE,
                   ADC1_ALIGN_RIGHT,
                   ADC1_SCHMITTTRIG_CHANNEL1,
                   DISABLE);

        ADC1_ConversionConfig(ADC1_CONVERSIONMODE_CONTINUOUS,

        ((ADC1_Channel_TypeDef)(ADC1_CHANNEL_0 | ADC1_CHANNEL_1)),
```

```
        ADC1_ALIGN_RIGHT);

        ADC1_DataBufferCmd(ENABLE);
        ADC1_Cmd(ENABLE);
}


void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned int value)
{
        char chr = 0x00;

        chr = ((value / 1000) + 0x30);
        LCD_goto(x_pos, y_pos);
        LCD_putchar(chr);

        chr = (((value / 100) % 10) + 0x30);
        LCD_goto((x_pos + 1), y_pos);
        LCD_putchar(chr);

        chr = (((value / 10) % 10) + 0x30);
        LCD_goto((x_pos + 2), y_pos);
        LCD_putchar(chr);

        chr = ((value % 10) + 0x30);
        LCD_goto((x_pos + 3), y_pos);
        LCD_putchar(chr);
}
```

Explanation

The setup here is same as the setup for the other ADC examples. There are a few difference though. Firstly, the channel configuration. Note that here two channels are used and are independently configured. This is so because each channel may have different properties. Secondly, conversion mode and data alignments are set. Thirdly, data buffers are enabled to capture ADC data on different channels.

```
void ADC1_setup(void)
{
        ADC1_DeInit();

        ADC1_Init(ADC1_CONVERSIONMODE_CONTINUOUS,
                  ADC1_CHANNEL_0,
                  ADC1_PRESSEL_FCPU_D18,
                  ADC1_EXTTRIG_GPIO,
                  DISABLE,
                  ADC1_ALIGN_RIGHT,
                  ADC1_SCHMITTTRIG_CHANNEL0,
                  DISABLE);

        ADC1_Init(ADC1_CONVERSIONMODE_CONTINUOUS,
                  ADC1_CHANNEL_1,
                  ADC1_PRESSEL_FCPU_D18,
                  ADC1_EXTTRIG_GPIO,
                  DISABLE,
```

```
                    ADC1_ALIGN_RIGHT,
                    ADC1_SCHMITTTRIG_CHANNEL1,
                    DISABLE);

        ADC1_ConversionConfig(ADC1_CONVERSIONMODE_CONTINUOUS,

        ((ADC1_Channel_TypeDef)(ADC1_CHANNEL_0 | ADC1_CHANNEL_1)),

        ADC1_ALIGN_RIGHT);

        ADC1_DataBufferCmd(ENABLE);
        ADC1_Cmd(ENABLE);
}
```

Polling method is used to start and extract ADC data. Firstly, the ADC is told to do scan conversion. The conversion is then started. We, then, poll the end-of-conversion condition (EOC) status. EOC notifies that all ADC data have been captured and a scan has successfully completed. Finally, the EOC flag is cleared and the data are read from separate data buffers. Everytime a new scan is started the previous buffer data are cleared and are updated with new ones.

```
ADC1_ScanModeCmd(ENABLE);
ADC1_StartConversion();
while(ADC1_GetFlagStatus(ADC1_FLAG_EOC) == FALSE);

ADC1_ClearFlag(ADC1_FLAG_EOC);

A0 = ADC1_GetBufferValue(0);
A1 = ADC1_GetBufferValue(1);
```

Demo

Video link: https://www.youtube.com/watch?v=OJbVsiXAE78.
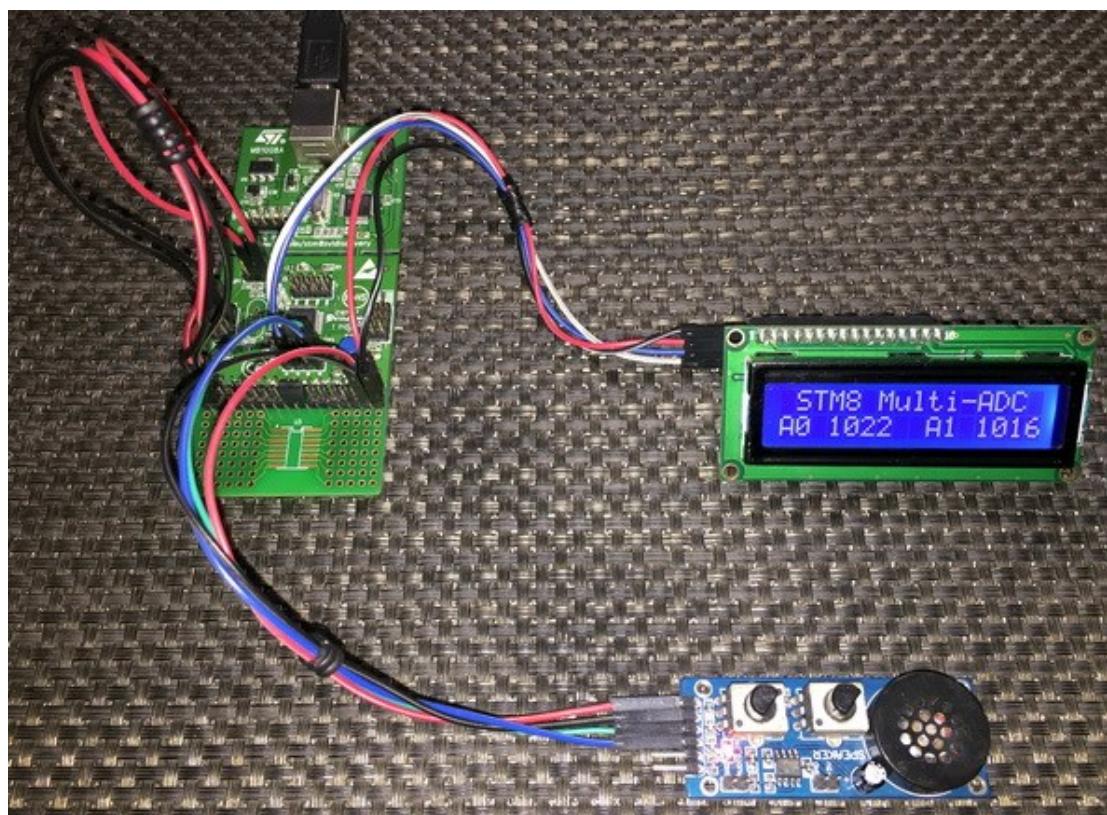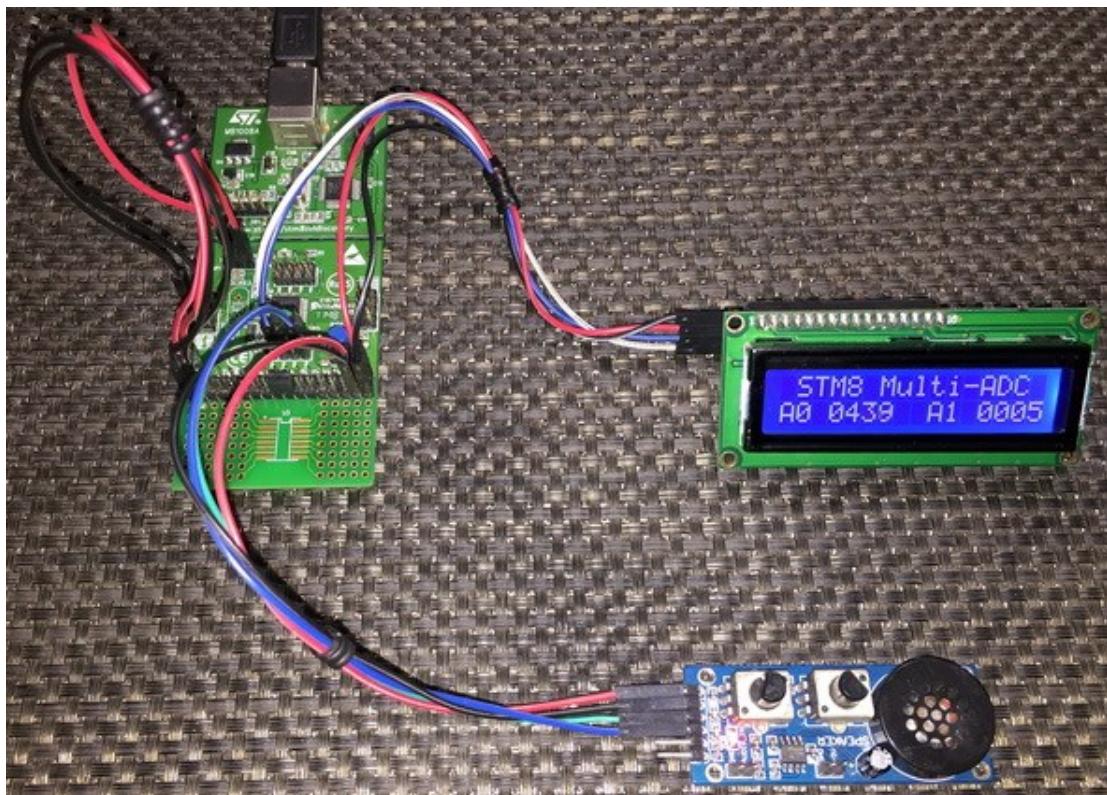
Analog Watchdog (AWD)

The AWD is one additional feature that most microcontrollers in the market do not have. AWD is more like a comparator but with the exception that we can set both the upper and lower limits of this comparator as per our requirement unlike fixed levels in other micros. The region between the upper and lower limits is called guarded zone. Beyond the boundaries of the guarded zone, the AWD unit kicks off.

The AWD unit is very useful in situations where we need to monitor the output of a sensor for example and take quick actions. For instance, consider a temperature controller. We would want the controller to turn on a heater should temperature fall below some level and turn it off when temperature rises to some high value without complex calculations and constant monitoring in our application firmware. In other microcontrollers, we would have accomplished this simple task using conditional *IF-ELSE* statements.
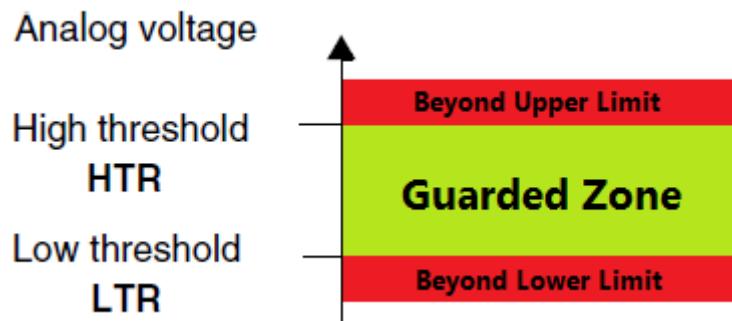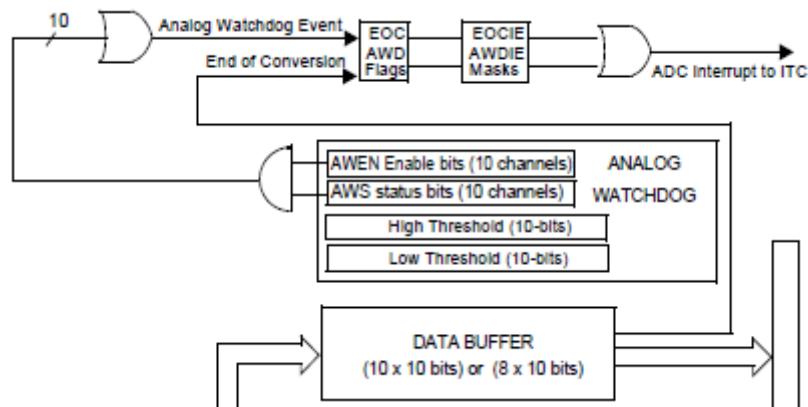
Hardware Connection
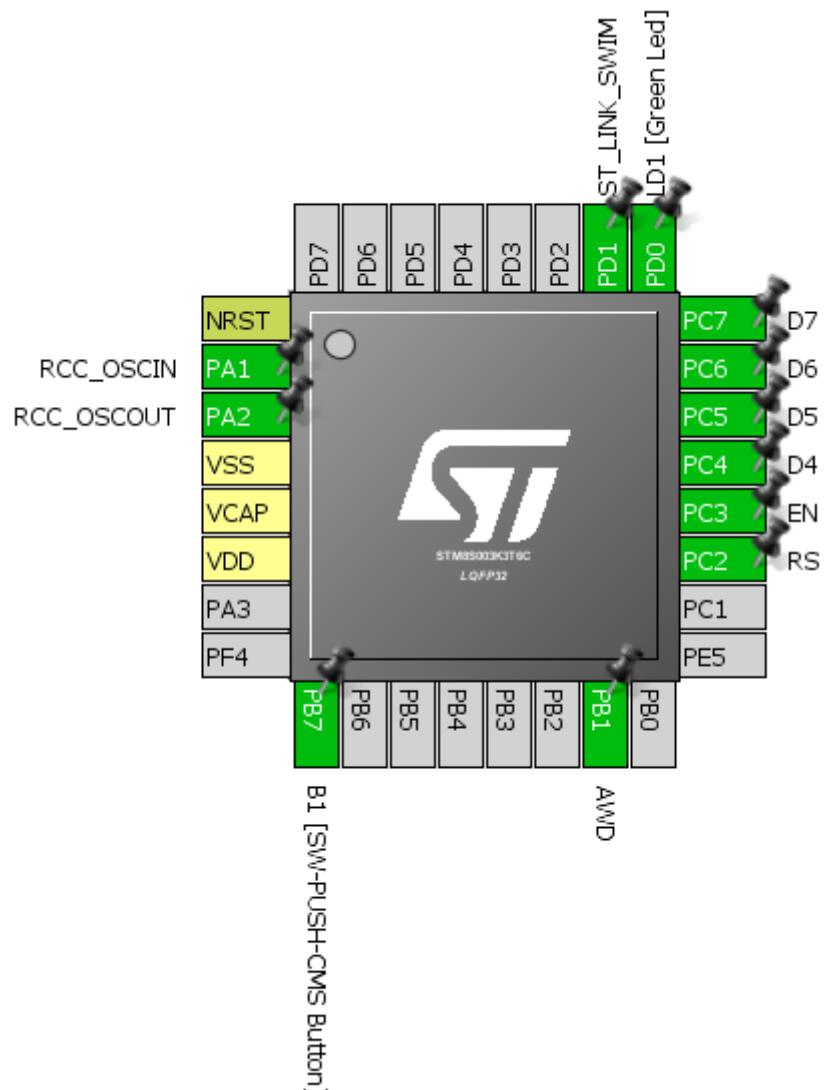
Code Example

```c
#include "STM8S.h"


void clock_setup(void);
void GPIO_setup(void);
void ADC1_setup(void);
void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned int value);


void main()
{
    unsigned int a1 = 0x0000;

    clock_setup();
    GPIO_setup();
    ADC1_setup();
```

```c
        LCD_init();
        LCD_clear_home();

        LCD_goto(0, 0);
        LCD_putstr("STM8 AWD");
        LCD_goto(0, 1);
        LCD_putstr("A1");

        while (TRUE)
        {
                ADC1_ClearFlag(ADC1_FLAG_EOC);

                ADC1_StartConversion();
                while(ADC1_GetFlagStatus(ADC1_FLAG_EOC) == 0);

                a1 = ADC1_GetConversionValue();
                lcd_print(4, 1, a1);

                if(ADC1_GetFlagStatus(ADC1_FLAG_AWD))
                {
                        GPIO_WriteReverse(GPIOD, GPIO_PIN_0);
                        ADC1_ClearFlag(ADC1_FLAG_AWD);
                }
                else
                {
                        GPIO_WriteHigh(GPIOD, GPIO_PIN_0);
                }

                delay_ms(90);
        };
}


void clock_setup(void)
{
        CLK_DeInit();

        CLK_HSECmd(DISABLE);
        CLK_LSICmd(DISABLE);
        CLK_HSICmd(ENABLE);
        while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

        CLK_ClockSwitchCmd(ENABLE);
        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV2);
        CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV4);

        CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
        DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}
```

```c
void GPIO_setup(void)
{
    GPIO_DeInit(GPIOB);
    GPIO_Init(GPIOB, GPIO_PIN_1, GPIO_MODE_IN_FL_NO_IT);

    GPIO_DeInit(GPIOD);
    GPIO_Init(GPIOD, GPIO_PIN_0, GPIO_MODE_OUT_PP_HIGH_FAST);
    GPIO_Init(GPIOD, GPIO_PIN_3, GPIO_MODE_IN_PU_NO_IT);
}


void ADC1_setup(void)
{
    ADC1_DeInit();
    ADC1_Init(ADC1_CONVERSIONMODE_SINGLE,
              ADC1_CHANNEL_1,
              ADC1_PRESSEL_FCPU_D10,
              ADC1_EXTTRIG_GPIO,
              DISABLE,
              ADC1_ALIGN_RIGHT,
              ADC1_SCHMITTTRIG_CHANNEL1,
              DISABLE);

    ADC1_AWDChannelConfig(ADC1_CHANNEL_1, ENABLE);
    ADC1_SetHighThreshold(600);
    ADC1_SetLowThreshold(200);

    ADC1_Cmd(ENABLE);
}


void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned int value)
{
    char chr = 0x00;

    chr = ((value / 1000) + 0x30);
    LCD_goto(x_pos, y_pos);
    LCD_putchar(chr);

    chr = (((value / 100) % 10) + 0x30);
    LCD_goto((x_pos + 1), y_pos);
    LCD_putchar(chr);

    chr = (((value / 10) % 10) + 0x30);
    LCD_goto((x_pos + 2), y_pos);
    LCD_putchar(chr);

    chr = ((value % 10) + 0x30);
    LCD_goto((x_pos + 3), y_pos);
    LCD_putchar(chr);
}
```

Explanation

The code for the AWD example is just as the one demonstrated in the ADC example. However, this time the ADC channel is channel 1 (PB1). Setting up the AWD is simple. We just need to set the limits, specify which channel to be monitored and enable the AWD unit.

```
ADC1_AWDChannelConfig(ADC1_CHANNEL_1, ENABLE);
ADC1_SetHighThreshold(600);
ADC1_SetLowThreshold(200);
```

Here we have set 600 and 200 ADC counts as upper and lower limits respectively.

In the main function, we are simply polling AWD flag. If an AWD (beyond boundary zone) event on PB1 pin occurs the LED on PD0 starts flashing. If PB1 senses voltage between 200 and 600 ADC counts, the LED is turned off, indicating guarded zone.

```
if(ADC1_GetFlagStatus(ADC1_FLAG_AWD))
{
        GPIO_WriteReverse(GPIOD, GPIO_PIN_0);
        ADC1_ClearFlag(ADC1_FLAG_AWD);
}

else
{
        GPIO_WriteHigh(GPIOD, GPIO_PIN_0);
}
```

Demo
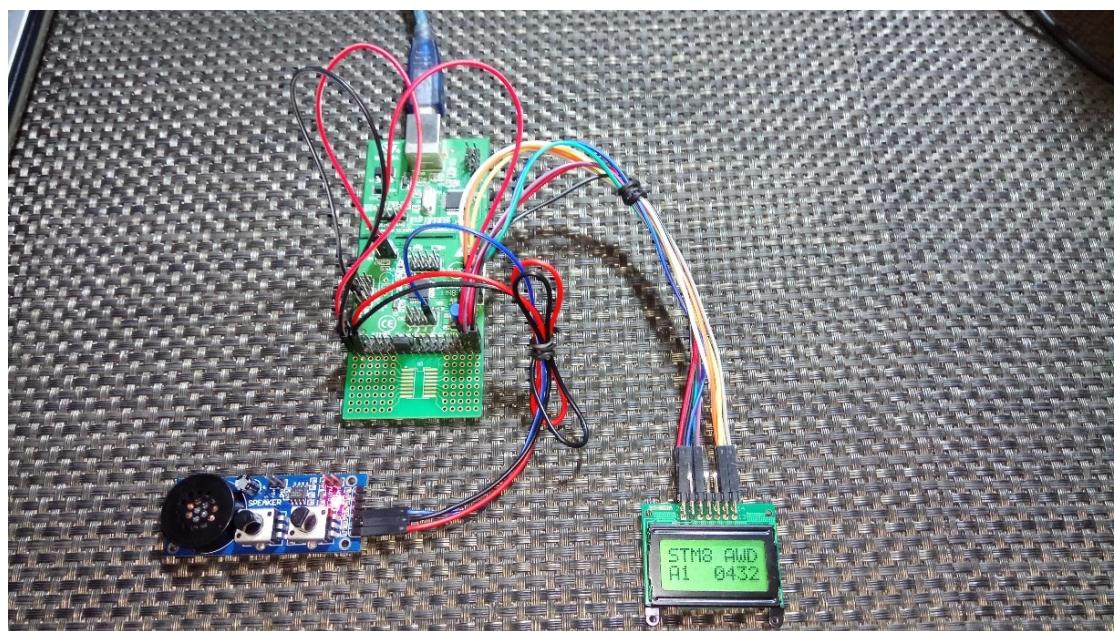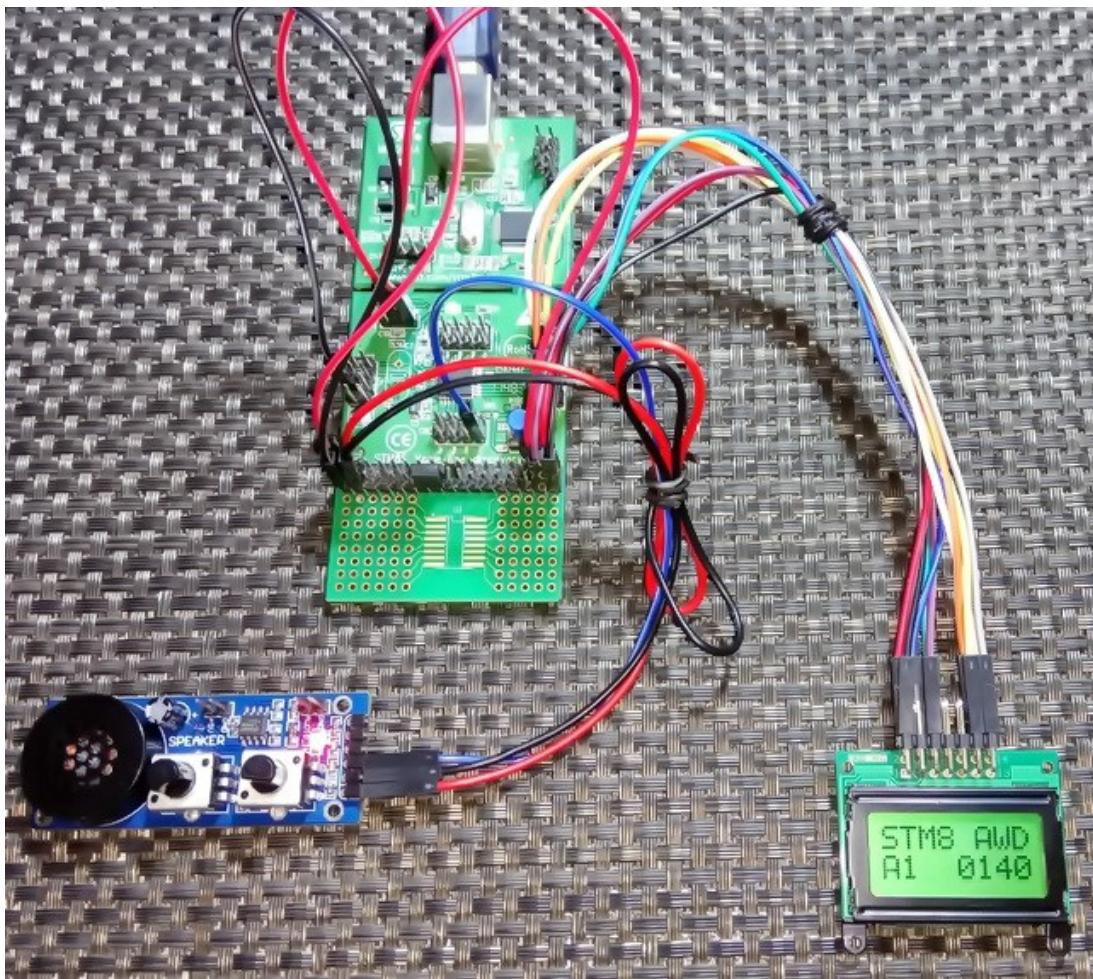
Video link: https://www.youtube.com/watch?v=bvVNuVpeFPk.

Independent Watchdog (IWDG)

The IWDG is just the ordinary watchdog timer we usually find in any modern micro. The purpose of this timer is to recover a micro from an unanticipated event that may result in unresponsive or erratic behaviour. As the name suggests, this timer does not share anything with any other internal hardware peripheral and is clocked by LSI (128kHz) only. Thus, it is invulnerable to main clock (HSE or HSI) failure.
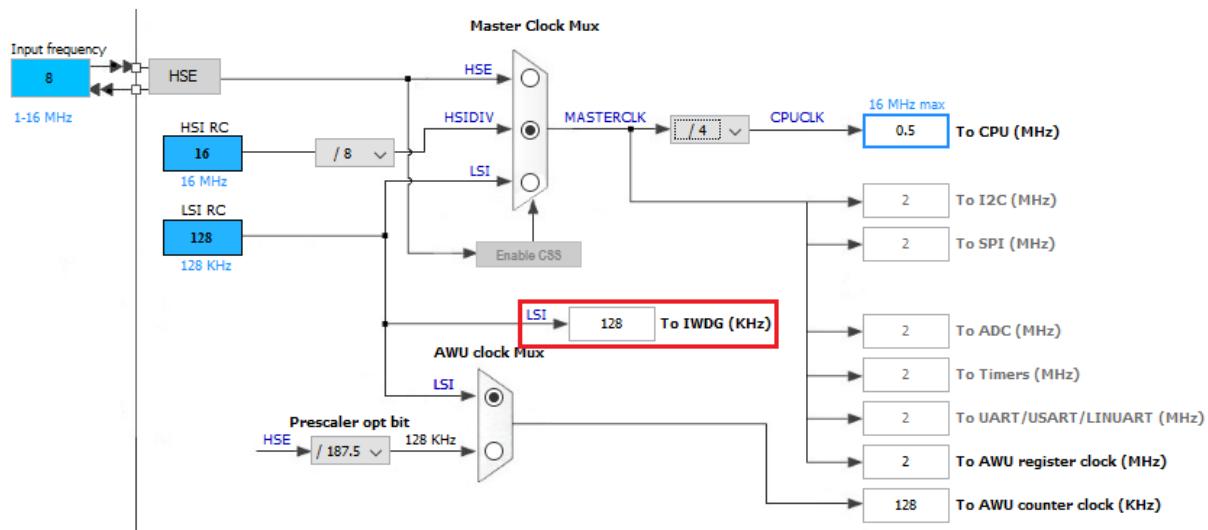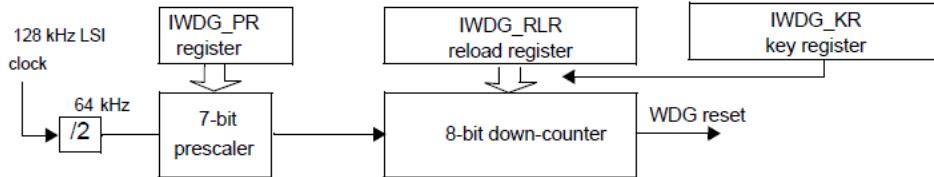
The IWDG works by decrementing a counter, counting time in the process. When the counter hits zero, a reset is issued. Usually we would want that this reset never occurs and so the counter is periodically updated in the application firmware. If for some reason, the counter is not refreshed, a reset will occur, recovering the MCU from an unanticipated situation.

Configuring the IWDG is very easy with SPL. There are certain steps to follow but SPL manages them well internally. All we'll need is to configure the IWDG and reload it periodically before time runs out.

The formula required to calculate timeout is given below:

$$T = 2 \times T_{LSI} \times P \times R$$

where:

T = Timeout period

$T_{LSI} = 1/f_{LSI}$

$P = 2^{(PR[2:0] + 2)}$

$R = RLR[7:0]+1$

Typical values of timeout are as shown below:

Watchdog timeout period (LSI clock frequency = 128 kHz)

Prescaler divider	PR[2:0] bits	Timeout	
		RL[7:0]= 0x00	RL[7:0]= 0xFF
/4	0	62.5 µs	15.90 ms
/8	1	125 µs	31.90 ms
/16	2	250 µs	63.70 ms
/32	3	500 µs	127 ms
/64	4	1.00 ms	255 ms
/128	5	2.00 ms	510 ms
/256	6	4.00 ms	1.02 s

Hardware Connection
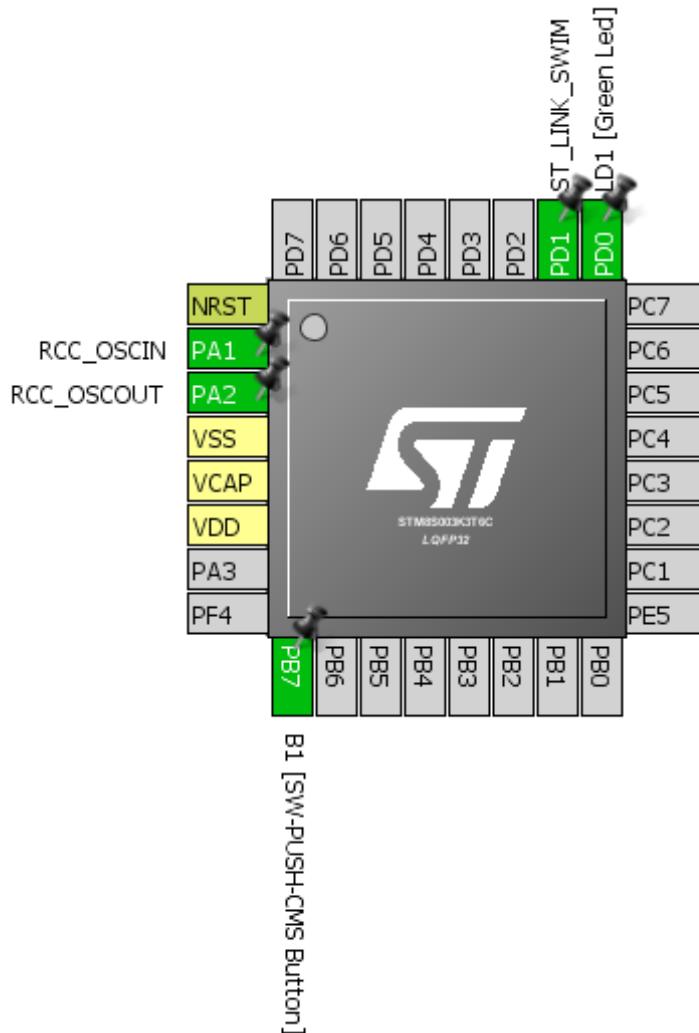
Code Example

```c
#include "STM8S.h"


void clock_setup(void);
void GPIO_setup(void);
void IWDG_setup(void);


void main(void)
{
    unsigned int t = 0;

    clock_setup();
    GPIO_setup();

    GPIO_WriteLow(GPIOD, GPIO_PIN_0);
    for(t = 0; t < 60000; t++);

    IWDG_setup();

    while(TRUE)
    {
        GPIO_WriteReverse(GPIOD, GPIO_PIN_0);
        for(t = 0; t < 1000; t++)
        {
            if(GPIO_ReadInputPin(GPIOB, GPIO_PIN_7) == FALSE)
            {
                IWDG_WriteAccessCmd(IWDG_WriteAccess_Enable);
                IWDG_ReloadCounter();
                IWDG_WriteAccessCmd(IWDG_WriteAccess_Disable);
            }
        }
    };
}


void clock_setup(void)
{
    CLK_DeInit();

    CLK_HSECmd(DISABLE);
    CLK_LSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_LSIRDY) == FALSE);
    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

    CLK_ClockSwitchCmd(ENABLE);
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
    CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV4);

    CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
    DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

    CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
```

```
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
        GPIO_DeInit(GPIOB);
        GPIO_DeInit(GPIOD);

        GPIO_Init(GPIOB, GPIO_PIN_7, GPIO_MODE_IN_PU_NO_IT);
        GPIO_Init(GPIOD, GPIO_PIN_0, GPIO_MODE_OUT_PP_LOW_FAST);
}


void IWDG_setup(void)
{
        IWDG_Enable();
        IWDG_WriteAccessCmd(IWDG_WriteAccess_Enable);
        IWDG_SetPrescaler(IWDG_Prescaler_128);
        IWDG_SetReload(0x99);
        IWDG_WriteAccessCmd(IWDG_WriteAccess_Disable);
}
```

Explanation

In this example, we need not to look at peripheral and CPU clock as IWDG is not dependent on them. Still we can see that the CPU is running at 500kHz speed while the peripherals at 2MHz speed.

```
CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV4);
```

To setup the IWDG, we need to enable it first and then apply *Write Access Protection* key (0x55). We just need to set the prescaler and the counter value. The down counter will start from this value and count down to zero unless refreshed. In this example, the prescaler is set to 128 and reload value is set to 153 (0x99). Thus, with these we get a timeout of approximately 300ms. After entering these values, we must prevent accidental changes in the firmware and so to do so the write access must be disabled.

```
void IWDG_setup(void)
{
        IWDG_Enable();
        IWDG_WriteAccessCmd(IWDG_WriteAccess_Enable);
        IWDG_SetPrescaler(IWDG_Prescaler_128);
        IWDG_SetReload(0x99);
        IWDG_WriteAccessCmd(IWDG_WriteAccess_Disable);
}
```

Disco board's user button and LED are used for the demo. At the very beginning, the LED is lit for some time before the IWDG is configured, indicating the start of the application firmware. In the main loop, the LED is toggled with some delay arranged by a *for* loop. Inside the loop, the button's state is polled.

If the button is kept pressed it will always be in logic low state, reloading the IWDG counter. If its state changes to logic high and 300ms passes out since last button press, a reset is triggered.

```c
GPIO_WriteReverse(GPIOD, GPIO_PIN_0);
for(t = 0; t < 1000; t++)
{
	if(GPIO_ReadInputPin(GPIOB, GPIO_PIN_7) == FALSE)
	{
		IWDG_WriteAccessCmd(IWDG_WriteAccess_Enable);
		IWDG_ReloadCounter();
		IWDG_WriteAccessCmd(IWDG_WriteAccess_Disable);
	}
}
```

Note it is possible to calibrate LSI. It is however rarely needed.

Demo

Video link: https://www.youtube.com/watch?v=05XKoy0ieHo.

Window Watchdog (WWDG)

The WWDG is a bit more advanced watchdog timer. Unlike the IWDG, it will trigger a reset condition if its counter is reloaded earlier or later than a predefined time window. This kind of timer is usually found in high-end microcontrollers like ARMs, ATXMegas and recently released micros. Cool features like this and others make me feel that indeed STM8s are high-end affordable 8-bit alternatives when compared to other traditional 8-bit MCUs.
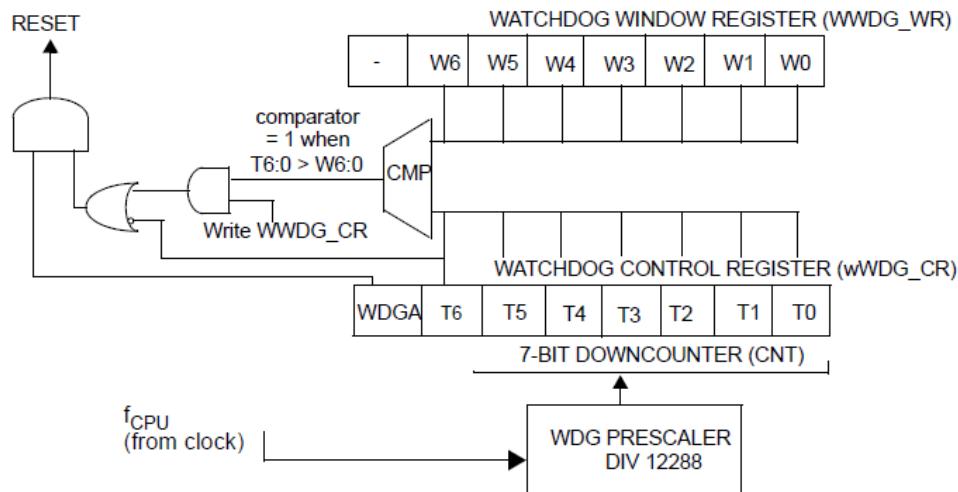
The WWDG works by comparing a down counter against a window register. The counter can only be refreshed when its value is greater than 0x3F and less than window register value. If the counter is refreshed before the value set on window register or when the counter is less than or equal to 0x3F. If the counter hits the value 0x3F, reset automatically triggers. It is programmer's responsibility to refresh the counter at proper time. Note unlike IWDG, WWDG is not independent of main clock.
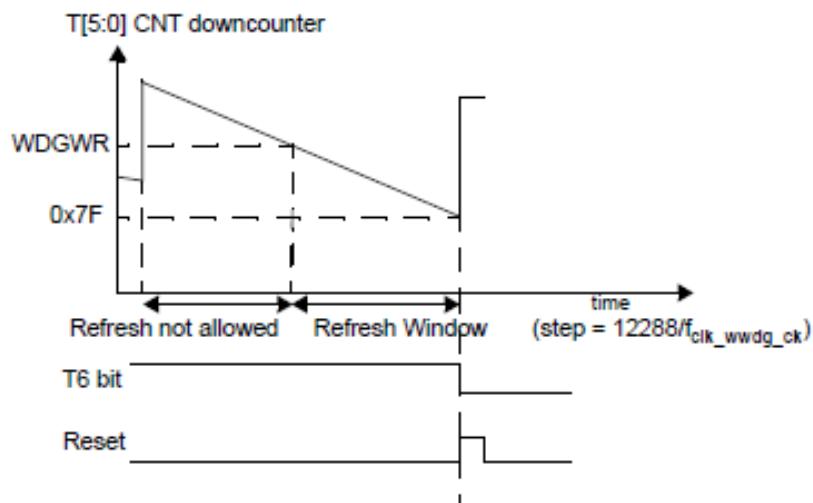
The formula below can be used to calculate the WWDG timeout, t_{WWDG} expressed in ms:

$$t_{WWDG} = T_{CPU} \times 12288 \times (T[5:0] + 1)$$

where T_{CPU} is the peripheral clock period expressed in ms

Hardware Connection
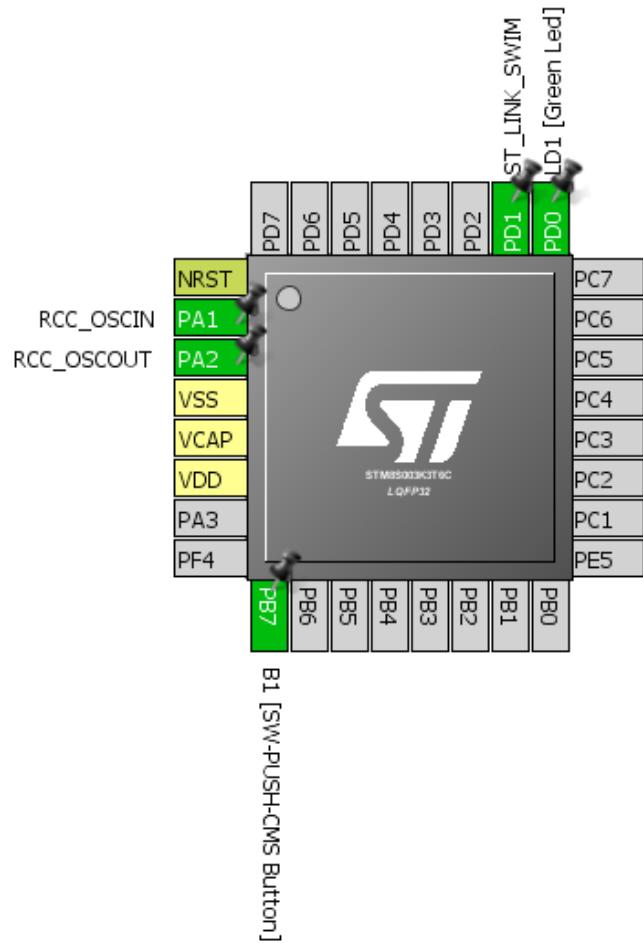
Code Example

The code example here demonstrates WWDG action. Simply Disco board's user LED and button are used once again. When the code starts executing, the LED starts blinking slowly, indicating the start of the code. When the code executes the main loop, the LED blinks rapidly to indicate main loop execution. If the button is pressed randomly the micro is reset because the counter is refreshed before the allowed time. Sometimes the micro may not reset because the counter may be in the allowed window frame – hence the name Window Watchdog.

```c
#include "STM8S.h"


void clock_setup(void);
void GPIO_setup(void);
void WWDG_setup(void);


void main(void)
{
    unsigned char i = 0x00;
```

```c
        clock_setup();
        GPIO_setup();

        for(i = 0x00; i < 0x04; i++)
        {
                GPIO_WriteReverse(GPIOD, GPIO_PIN_0);
                delay_ms(40);
        }

        WWDG_setup();

        while(TRUE)
        {
                if((GPIO_ReadInputPin(GPIOB, GPIO_PIN_7) == FALSE) ||
                ((WWDG_GetCounter() > 0x60) && (WWDG_GetCounter() < 0x7F)))
                {
                        WWDG_SetCounter(0x7F);
                }
                GPIO_WriteReverse(GPIOD, GPIO_PIN_0);
                delay_ms(20);
        };
}


void clock_setup(void)
{
        CLK_DeInit();

        CLK_HSECmd(DISABLE);
        CLK_LSICmd(DISABLE);
        CLK_HSICmd(ENABLE);
        while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

        CLK_ClockSwitchCmd(ENABLE);
        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
        CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV64);

        CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
        DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
        GPIO_DeInit(GPIOB);
        GPIO_Init(GPIOB, GPIO_PIN_7, GPIO_MODE_IN_PU_NO_IT);

        GPIO_DeInit(GPIOD);
        GPIO_Init(GPIOD, GPIO_PIN_0, GPIO_MODE_OUT_OD_HIZ_FAST);
}
```

```
void WWDG_setup(void)
{
        WWDG_Init(0x7F, 0x60);
}
```

Explanation

Unlike other peripherals there's no way to enable/disable watchdogs manually in software as they are always enabled. However, there are configuration bits to select if the IWDG and the WWDG are enabled in software or hardware. They only come in effect when configured. This is cool.

```
AFR7                          Reserved
AFR6                          AFR6 Alternate Function Remapping inactive
AFR5                          AFR5 Alternate Function Remapping inactive
AFR4                          Reserved
AFR3                          Reserved
AFR2                          Reserved
AFR1                          AFR1 Alternate Function Remapping inactive
AFR0                          Reserved

HSITRIM                       3 bit on-the-fly trimming
LSI_EN                        LSI Clock not available as CPU clock source
IWDG_HW                       Independant Watchdog activated by Software
WWDG_HW                       Window Watchdog activated by Software
WWDG_HALT                     No Reset generated on HALT if WWDG active

EXTCLK                        External Crystal connected to OSCIN/OSCOUT
CKAWUSEL                      LSI clock source selected for AWU
PRSC                          16MHz to 128KHz Prescaler
```

For WWDG, we just need to set the value of the down counter and the window register value only.

```
void WWDG_setup(void)
{
        WWDG_Init(0x7F, 0x60);
}
```

We need to monitor the WWDG in order to reload it when it is the right time window.

```
while(TRUE)
{
        if((GPIO_ReadInputPin(GPIOB, GPIO_PIN_7) == FALSE) ||
        ((WWDG_GetCounter() > 0x60) && (WWDG_GetCounter() < 0x7F)))
        {
                WWDG_SetCounter(0x7F);
        }

        GPIO_WriteReverse(GPIOD, GPIO_PIN_0);
        delay_ms(20);
};
```

Remember too early or too late will reset the micro.

Demo

Video link: https://www.youtube.com/watch?v=a_JWHJCh_-o.

Timer Overview

Timers are perhaps the most versatile piece of hardware in any micro. As their name tells, timers are useful for measurement of timed events like frequency, time, phase sequence, etc. and generate time-based events like PWM, waveform, etc.

Timers are also needed for touch sensing applications.

In any STM8 micro, there are three categories of timers. These are:

- Advanced Control Timer (TIM1)
- General Purpose Timers (TIM2, TIM3 & TIM5)
- Basic Timers (TIM4 & TIM6)

The basic working principle of all timers are same with some minor differences. Advanced timers are mainly intended for applications requiring specialized motor control, SMPSs, inverters, waveform generation, pulse width measurements, etc. Then there are general purpose timers that share almost all the features of advanced timer without the advanced features like brake, dead-time control, etc. Basic timers are all same as general purpose timers but lack PWM output/capture input pins and are intended mainly for time base generations. Here's the summary of all timers of STM8 micros:

Timer	Counter resolution	Counter type	Prescaler factor	Capture/compare channels	Complementary outputs	Repetition counter	External trigger input	External break input	Timer synchronization/chaining
TIM1 (advanced control timer)	16-bit	Up/down	Any integer from 1 to 65536	4	3	Yes	1	1	With TIM5/TIM6
TIM2 (general purpose timer)	16-bit	Up	Any power of 2 from 1 to 32768	3	None	No	0	0	No
TIM3 (general purpose timer)	16-bit	Up	Any power of 2 from 1 to 32768	2	None	No	0	0	No
TIM4 (basic timer)	8-bit	Up	Any power of 2 from 1 to 128	0	None	No	0	0	No
TIM5 (general purpose timer)	16-bit	Up	Any power of 2 from 1 to 32768	3	None	No	1 (shared with TIM1)	0	Yes
TIM6 (basic timer)	8-bit	Up	Any power of 2 from 1 to 128	0	None	No	0	0	Yes

Unlike the timers of other micros, STM8 timers have the more functionality that are otherwise only available in some special micros only. Timer cover a significant part of the reference manual. They are so elaborate that it is not possible to describe all of them in just one post. Therefore, here, we'll be exploring the basics only.

Time Base Generation (TIM2)

Time base generation is the most basic property of any timer and is also the most needed requirement in embedded systems. This mode can be used with or without interrupt. We'll first check the method firstly without interrupt and then with interrupt.

With time base generation, we can accurately time stuffs and events that are more precise than using delays, loops or other methods. Time base generation utilizes hardware timers and so work independently from other processes. It has many uses. For instance, with it we can avoid software delays, generate time slots of a Real-Time Operating System (RTOS) and many other tasks.
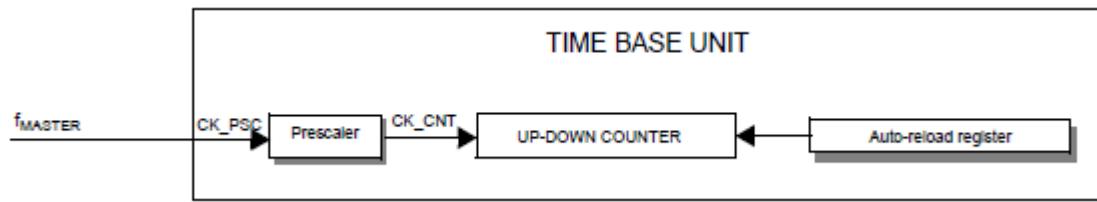
The time base unit for all timers of STM8 is all same. There are a few differences. For example, Timer 1 (TIM1) has a repetition counter. It is like a counter within another counter. Other timers lack this part. All timers can count up while advance timers can count down too.

The basic theory of time base generation is you have a peripheral clock which you would like to scale according to your need. Thus, you prescale it and use the new clock to run a counter. The counter will tick, incrementing count as time flies. It is just like counting from 0 to 100 and repeating from 0 again after reaching 100. Shown below is the generalized formula for finding an important event called timer reload:

$$Time\ Event\ (Timer\ Reload) = \frac{(Prescaler\ \times\ Reptiation\ Counter\ \times\ Counts)}{F_{master}}$$

This is the amount of time that will pass before timer overflow event occurs and the timer restarts from its initial count.

In my example, the peripheral or master clock is set to 2MHz. Thus, to make timer 2 (TIM2) reload after roughly 2 seconds, I have to prescale it by a factor of 2048 and load it with 1952 counts. Note that TIM2 doesn't have a repetition counter and so it is set to 1.

$$Time\ Event\ (Timer\ Reload) = \frac{(2048\ \times 1952)}{2\ \times\ 10^6}$$

$$= 1.998s$$

$$\approx 2s$$

Hardware Connection
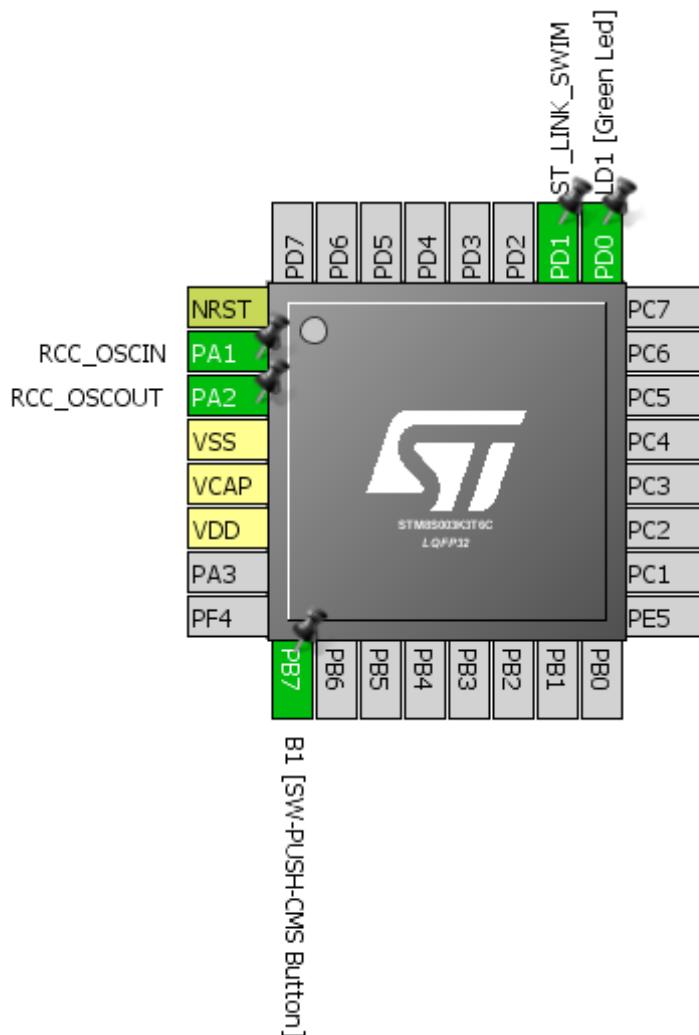
Code Example

In this example, Disco board's user LED is turned-on and off without using any software delay. TIM2 is used to create time delay as such that the code is not stuck in a time-wasting loop.

```c
#include "STM8S.h"


void clock_setup(void);
void GPIO_setup(void);
void TIM2_setup(void);


void main(void)
{
	clock_setup();
	GPIO_setup();
	TIM2_setup();

	while(TRUE)
```

```c
    {
        if(TIM2_GetCounter() > 976)
        {
            GPIO_WriteHigh(GPIOD, GPIO_PIN_0);
        }
        else
        {
            GPIO_WriteLow(GPIOD, GPIO_PIN_0);
        }
    };
}


void clock_setup(void)
{
    CLK_DeInit();

    CLK_HSECmd(DISABLE);
    CLK_LSICmd(DISABLE);
    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

    CLK_ClockSwitchCmd(ENABLE);
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
    CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

    CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
    DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

    CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, ENABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
    GPIO_DeInit(GPIOD);
    GPIO_Init(GPIOD, GPIO_PIN_0, GPIO_MODE_OUT_OD_HIZ_SLOW);
}


void TIM2_setup(void)
{
    TIM2_DeInit();
    TIM2_TimeBaseInit(TIM2_PRESCALER_2048, 1952);
    TIM2_Cmd(ENABLE);
}
```

Explanation

Firstly, the CPU and the peripheral clock is set at 2MHz.

```
CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);
....
CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, ENABLE);
```

As explained earlier, to get 2 second timer reload interval we need to prescale the timer by 2048 and load its counter with 1952. This is what should be the setup for TIM2:

```
void TIM2_setup(void)
{
      TIM2_DeInit();
      TIM2_TimeBaseInit(TIM2_PRESCALER_2048, 1952);
      TIM2_Cmd(ENABLE);
}
```

Our goal is to keep the LED on for 1 second and off for 1 second. It takes 1952 TIM2 counts for 2 second interval and so one second passes when this count is 976. Thus, in the main loop we are checking the value of TIM2's counter. From 0 to 976 counts, the LED is on and from 977 to 1952 counts, the LED is off. Note that the LED's positive end is connected to VDD and so it will turn on only PD0 is low.

```
if(TIM2_GetCounter() > 976)
{
      GPIO_WriteHigh(GPIOD, GPIO_PIN_0);
}

else
{
      GPIO_WriteLow(GPIOD, GPIO_PIN_0);
}
```

Demo

Video link: https://youtu.be/ZstHDHAAHOM.
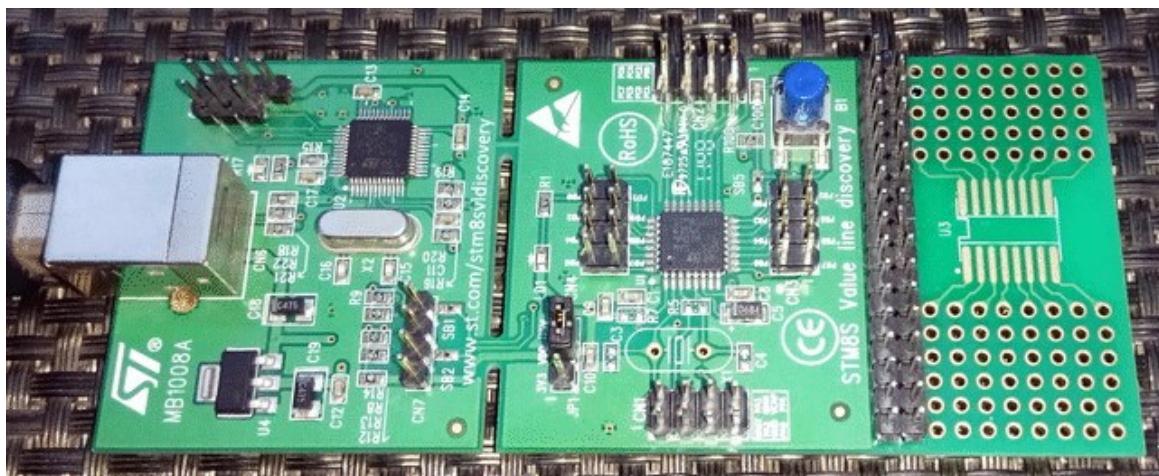
Timer Interrupt (TIM4)

In this example uses the same concepts of the previous example but it is based on timer interrupt – TIM4 interrupt. Timer interrupts are very important interrupts apart from other interrupts in a micro. To me they are highly valuable and useful.

This example demonstrates how to scan and project information on multiple seven segment displays with timer interrupt while the main loop can process the information to be displayed.

Hardware Connection
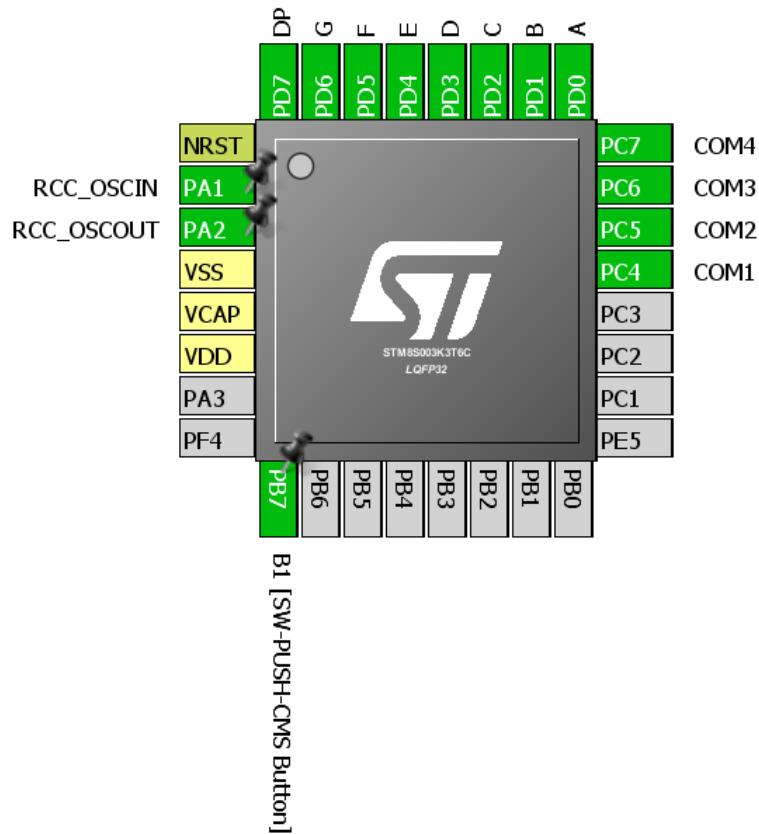
Code Example

main.c

```c
#include "STM8S.h"


unsigned int value = 0x00;

unsigned char n = 0x00;
unsigned char seg = 0x01;
const unsigned char num[0x0A] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8,
0x80, 0x90};
```

```c
void GPIO_setup(void);
void clock_setup(void);
void TIM4_setup(void);


void main(void)
{
    GPIO_setup();
    clock_setup();
    TIM4_setup();

    while (TRUE)
    {
        value++;
        delay_ms(999);
    };
}


void GPIO_setup(void)
{
    GPIO_DeInit(GPIOC);
GPIO_Init(GPIOC, ((GPIO_Pin_TypeDef)(GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 |
GPIO_PIN_7)), GPIO_MODE_OUT_PP_HIGH_FAST);

    GPIO_DeInit(GPIOD);
    GPIO_Init(GPIOD, GPIO_PIN_ALL, GPIO_MODE_OUT_PP_HIGH_FAST);
}


void clock_setup(void)
{
    CLK_DeInit();

    CLK_HSECmd(DISABLE);
    CLK_LSICmd(DISABLE);
    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

    CLK_ClockSwitchCmd(ENABLE);
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
    CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

    CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
    DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

    CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, ENABLE);
}
```

```
void TIM4_setup(void)
{
        TIM4_DeInit();
        TIM4_TimeBaseInit(TIM4_PRESCALER_32, 128);
        TIM4_ITConfig(TIM4_IT_UPDATE, ENABLE);
        TIM4_Cmd(ENABLE);
        enableInterrupts();
}
```

stm8s_it.h (top part only)

```
#ifndef __STM8S_IT_H
#define __STM8S_IT_H

@far @interrupt void TIM4_UPD_IRQHandler(void);



/* Includes -------------------------------------------------------------*/
#include "stm8s.h"
```

stm8s_it.c (top part only)

```
#include "stm8s.h"
#include "stm8s_it.h"


extern unsigned int value;

extern unsigned char n;
extern unsigned char seg;
extern const unsigned char num[10];


void TIM4_UPD_IRQHandler(void)
{
        switch(seg)
        {
                case 1:
                {
                        n = (value / 1000);
                        GPIO_Write(GPIOD, num[n]);
                        GPIO_Write(GPIOC, 0xE0);
                        break;
                }

                case 2:
                {
                        n = ((value / 100) % 10);
                        GPIO_Write(GPIOD, num[n]);
                        GPIO_Write(GPIOC, 0xD0);
                        break;
                }

                case 3:
                {
                        n = ((value / 10) % 10);
                        GPIO_Write(GPIOD, num[n]);
                        GPIO_Write(GPIOC, 0xB0);
```

```
                break;
        }

        case 4:
        {
                n = (value % 10);
                GPIO_Write(GPIOD, num[n]);
                GPIO_Write(GPIOC, 0x70);
                break;
        }
    }

    seg++;
    if(seg > 4)
    {
            seg = 1;
    }
    TIM4_ClearFlag(TIM4_FLAG_UPDATE);
}
```

stm8_interrupt_vector.c (shortened)

```
#include "stm8s_it.h"

typedef void @far (*interrupt_handler_t)(void);

struct interrupt_vector {
        unsigned char interrupt_instruction;
        interrupt_handler_t interrupt_handler;
};


extern void _stext();       /* startup routine */


struct interrupt_vector const _vectab[] = {
        {0x82, (interrupt_handler_t)_stext}, /* reset */
        {0x82, NonHandledInterrupt}, /* trap  */
        {0x82, NonHandledInterrupt}, /* irq0  */
        ....
        {0x82, (interrupt_handler_t)TIM4_UPD_IRQHandler}, /* irq23 */
        ....
        {0x82, NonHandledInterrupt}, /* irq29 */
};
```

Explanation

Both the peripheral and CPU clocks are running at 2MHz.

```
CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);
....
CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, ENABLE);
```

TIM4 is a basic timer and so it is better to use it for such tasks. We initialize it by setting its prescaler to 32 and loading its counter with 128. These values will make TIM4 overflow and interrupt every 2ms – enough time to project info in one seven segment. There are 4 seven segment displays and so within 8ms all four are updated and your eyes see it as if all projecting info at the same time – a trick of vision. Lastly, we need to enable what kind of interrupt we are expecting from the timer and finally enable the global interrupt.

```c
void TIM4_setup(void)
{
    TIM4_DeInit();
    TIM4_TimeBaseInit(TIM4_PRESCALER_32, 128);
    TIM4_ITConfig(TIM4_IT_UPDATE, ENABLE);
    TIM4_Cmd(ENABLE);
    enableInterrupts();
}
```

Remember the first interrupt example? We have to let the compiler know which interrupt we are using. If you look at the datasheet, you'll see that TIM4 update/overflow is located in IRQ23. We need this and so we should make the following change in the *stm8_interrupt_vector.c* file:

```c
{0x82, (interrupt_handler_t)TIM4_UPD_IRQHandler}, /* irq23 */
```

Remember to add the interrupt header and source files as we are going to use interrupt here. Inside the ISR, we do the scanning of each seven segment. Every time an overflow interrupt occurs, a seven segment is changed. At the end of the ISR a counter is incremented to select the next display when new overflow event occurs. Inside the *Switch-Case*, we turn on the seven segment and decide the value that seven segment should show. Finally, the timer overflow/update flag is cleared.

```c
switch(seg)
{
    case 1:
    {
        n = (value / 1000);
        GPIO_Write(GPIOD, num[n]);
        GPIO_Write(GPIOC, 0xE0);
        break;
    }

    case 2:
    {
        n = ((value / 100) % 10);
        GPIO_Write(GPIOD, num[n]);
        GPIO_Write(GPIOC, 0xD0);
        break;
    }

    case 3:
    {
        n = ((value / 10) % 10);
        GPIO_Write(GPIOD, num[n]);
        GPIO_Write(GPIOC, 0xB0);
        break;
    }
```

```
        case 4:
        {
                n = (value % 10);
                GPIO_Write(GPIOD, num[n]);
                GPIO_Write(GPIOC, 0x70);
                break;
        }
}

seg++;

if(seg > 4)
{
        seg = 1;
}

TIM4_ClearFlag(TIM4_FLAG_UPDATE);
```

Demo

Video link: https://youtu.be/Sa20Hf2N4gE.

Creating Delays using Timer

Rather than using loops of no operation instructions, it is wiser to use an unused or free timer to create software delays. It will be both accurate and precise. It can be done with any timer but using a basic timer will be a smart move because basic timers are usually for timebase generation purposes.

Hardware Connection
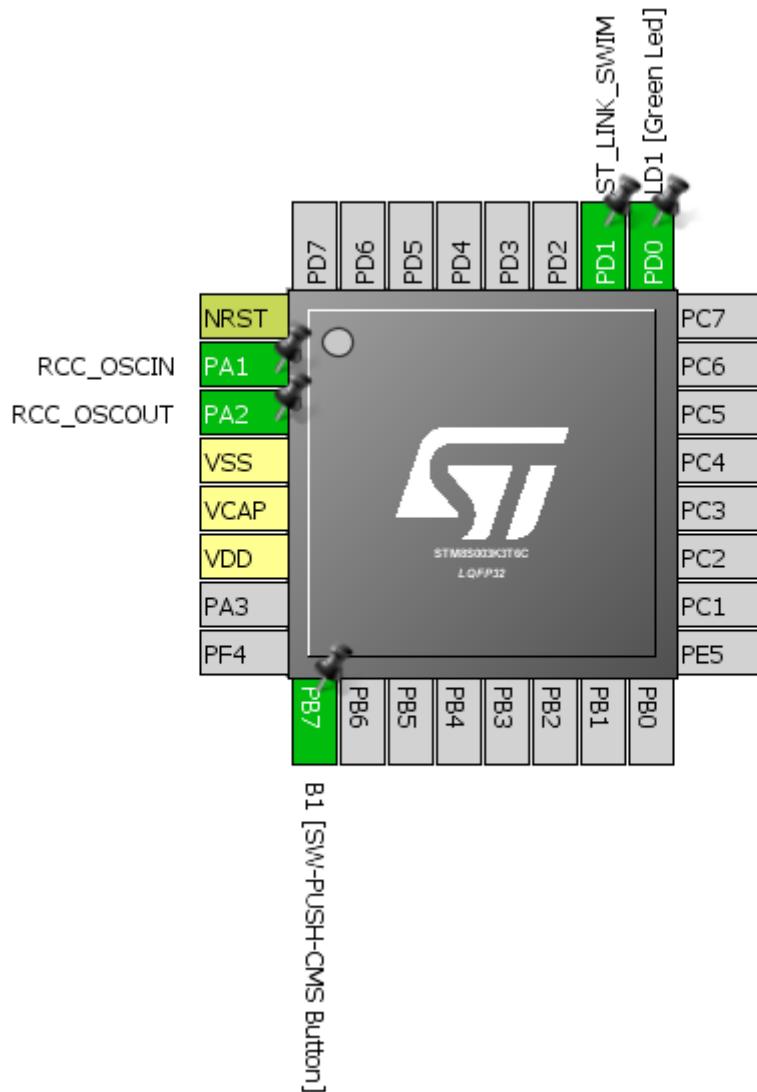
Code Example

```c
#include "STM8S.h"


void clock_setup(void);
void GPIO_setup(void);
void delay_us(signed int us);
void delay_ms(signed int ms);


void main(void)
{
      clock_setup();
      GPIO_setup();

      while(TRUE)
      {
            GPIO_WriteReverse(GPIOD, GPIO_PIN_0);
            delay_ms(400);
      };
}


void clock_setup(void)
{
      CLK_DeInit();

      CLK_HSECmd(DISABLE);
      CLK_LSICmd(DISABLE);
      CLK_HSICmd(ENABLE);
      while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

      CLK_ClockSwitchCmd(ENABLE);
      CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
      CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

      CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
      DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

      CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
      CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
      CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
      CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
      CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
      CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
      CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
      CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, ENABLE);
}


void GPIO_setup(void)
{
      GPIO_DeInit(GPIOD);
      GPIO_Init(GPIOD, GPIO_PIN_0, GPIO_MODE_OUT_OD_HIZ_SLOW);
}
```

```c
void delay_us(signed int us)
{
      TIM4_DeInit();

      if((us <= 200) && (us >= 0))
      {
            TIM4_TimeBaseInit(TIM4_PRESCALER_16, 200);
            TIM4_Cmd(ENABLE);
      }
      else if((us <= 400) && (us > 200))
      {
            us >>= 1;
            TIM4_TimeBaseInit(TIM4_PRESCALER_32, 200);
            TIM4_Cmd(ENABLE);
      }
      else if((us <= 800) && (us > 400))
      {
            us >>= 2;
            TIM4_TimeBaseInit(TIM4_PRESCALER_64, 200);
            TIM4_Cmd(ENABLE);
      }
      else if((us <= 1600) && (us > 800))
      {
            us >>= 3;
            TIM4_TimeBaseInit(TIM4_PRESCALER_128, 200);
            TIM4_Cmd(ENABLE);
      }
      while(TIM4_GetCounter() < us);
      TIM4_ClearFlag(TIM4_FLAG_UPDATE);
      TIM4_Cmd(DISABLE);
}


void delay_ms(signed int ms)
{
      while(ms--)
      {
            delay_us(1000);
      };
}
```

Explanation

The very first important things to decide and note are the clock speeds of peripherals and CPU. Here both are set to 16MHz. Secondly, TIM4 is use to create software delays and so it should be clocked.

```c
CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);
....
CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, ENABLE);
```

The smallest unit of time that we usually use in microcontroller coding is microseconds and so it should be coded first.

```c
void delay_us(signed int us)
{
    TIM4_DeInit();

    if((us <= 200) && (us >= 0))
    {
        TIM4_TimeBaseInit(TIM4_PRESCALER_16, 200);
        TIM4_Cmd(ENABLE);
    }
    else if((us <= 400) && (us > 200))
    {
        us >>= 1;
        TIM4_TimeBaseInit(TIM4_PRESCALER_32, 200);
        TIM4_Cmd(ENABLE);
    }
    else if((us <= 800) && (us > 400))
    {
        us >>= 2;
        TIM4_TimeBaseInit(TIM4_PRESCALER_64, 200);
        TIM4_Cmd(ENABLE);
    }
    else if((us <= 1600) && (us > 800))
    {
        us >>= 3;
        TIM4_TimeBaseInit(TIM4_PRESCALER_128, 200);
        TIM4_Cmd(ENABLE);
    }
    while(TIM4_GetCounter() < us);
    TIM4_ClearFlag(TIM4_FLAG_UPDATE);
    TIM4_Cmd(DISABLE);

}
```

TIM4 is deinitialized first. The input or argument to the *delay_us* function is the amount of time in microseconds we wish to delay. TIM4 is an 8-bit timer and so it can count up to 255. For simplicity, the count has been limited to 200. The first *if-else* conditional statement will configure the timer for periods from 0 to 200 microseconds. Input clock to the timer is prescaled by 16 and so it has a input clock frequency of 1MHz or 1 microsecond tick interval. In the while loop at the bottom of the function, TIM4's counter count is checked against the variable *us*. The loop breaks when the counter exceeds the variable's value, creating a software delay. For bigger delays, the other conditional statements are used.

For more bigger delays, delay milliseconds (*delay_ms*) function is used. This function loops calls of a thousand microseconds.

```c
void delay_ms(signed int ms)
{
    while(ms--)
    {
        delay_us(1000);
    };
}
```

The demo here is just a simple LED blinker.

Demo

Video link: https://www.youtube.com/watch?v=CIQzuBrd260.
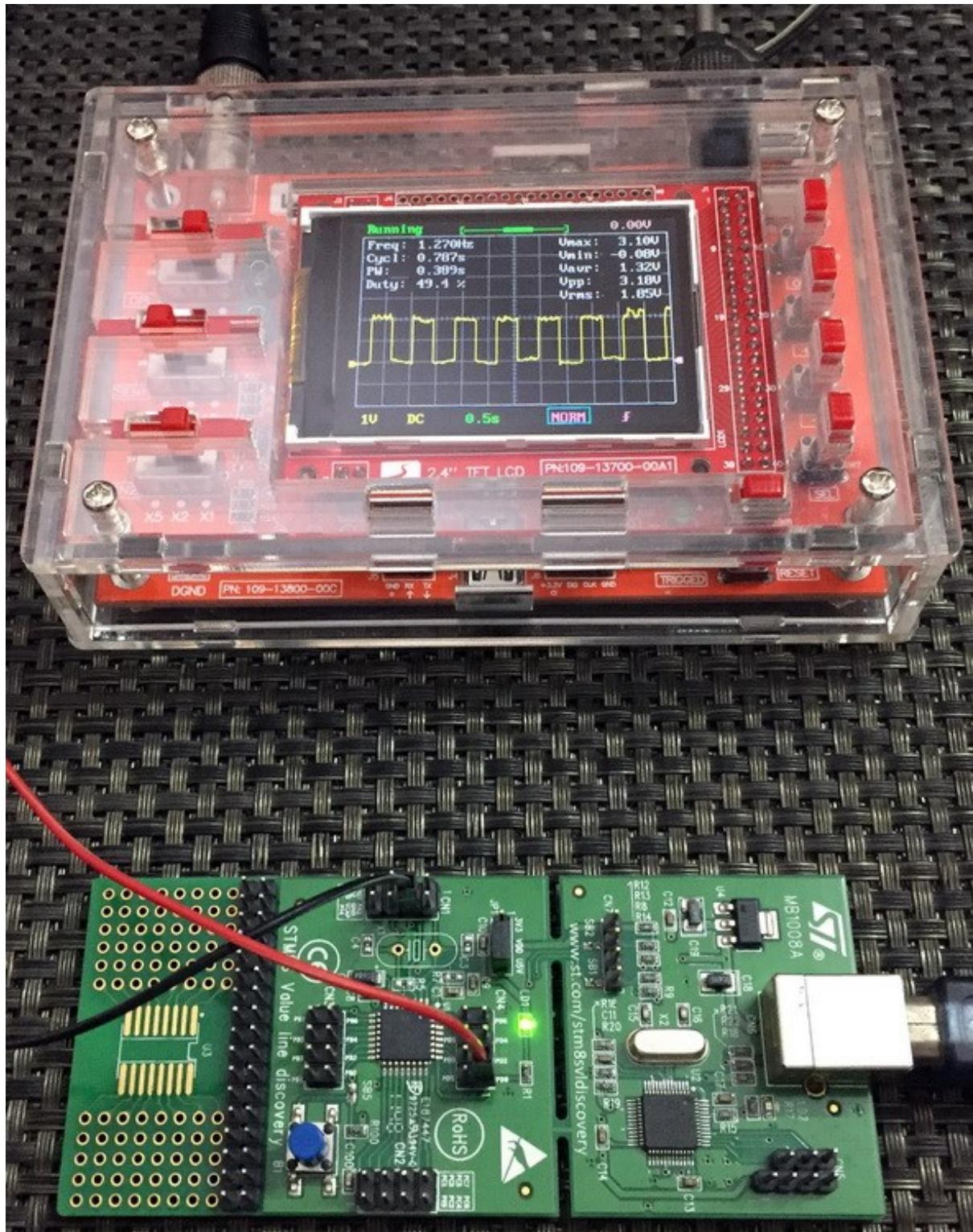
General Purpose Pulse Width Modulation (TIM2 PWM)

Pulse Width Modulation (PWM) is a must-have feature of any microcontroller. PWM has many uses like motor control, SMPSs, lighting control, sound generation, waveform generation, etc. Unlike other micros which have limited PWM channels, STM8 has several PWM channels. For instance, STM8S003K has seven independent PWM channels, three of which belong to TIM2 – a general purpose (GP) timer.

PWMs generated by GP timers are basic PWMs. They can be used for simple tasks like LED brightness control, servo motor control, etc. that don't require advanced features like dead-time, brake or complementary waveform generation. In this section, we will see how to use TIM2 to generate simple or general purpose PWMs.

Please note that in more advanced STM8 micros, timer I/Os are dependent on alternate function configuration bits. Check those bits before uploading codes. In some STM8 micros, the I/Os are also remappable, meaning that the I/Os can be swapped in different GPIOs. Take the help of STM8CubeMx if needed.

Hardware Connection
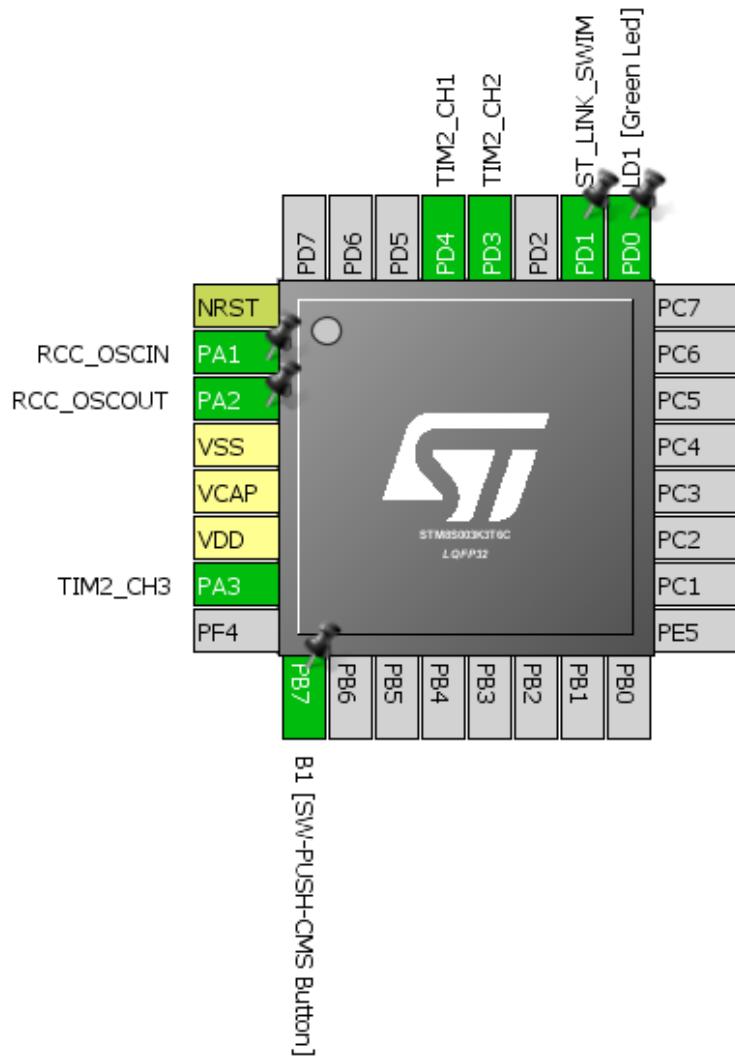
Code Example

This is a pretty simple example. Here all three channels of TIM2 are used to smoothly fade and glow three LEDs connected to the timer channels.

```c
#include "STM8S.h"


void clock_setup(void);
void GPIO_setup(void);
void TIM2_setup(void);


void main(void)
{
    signed int pwm_duty = 0x0000;

    clock_setup();
    GPIO_setup();
    TIM2_setup();

    while(TRUE)
    {
        for(pwm_duty = 0; pwm_duty < 1000; pwm_duty += 10)
        {
            TIM2_SetCompare1(pwm_duty);
            TIM2_SetCompare2(pwm_duty);
            TIM2_SetCompare3(pwm_duty);
            delay_ms(10);
        }
        for(pwm_duty = 1000; pwm_duty > 0; pwm_duty -= 10)
        {
            TIM2_SetCompare1(pwm_duty);
            TIM2_SetCompare2(pwm_duty);
            TIM2_SetCompare3(pwm_duty);
            delay_ms(10);
        }
    };
}


void clock_setup(void)
{
    CLK_DeInit();

    CLK_HSECmd(DISABLE);
    CLK_LSICmd(DISABLE);
    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

    CLK_ClockSwitchCmd(ENABLE);
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
    CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

    CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
    DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

    CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
```

```c
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
        GPIO_DeInit(GPIOA);
        GPIO_Init(GPIOA, GPIO_PIN_3, GPIO_MODE_OUT_PP_HIGH_FAST);

        GPIO_DeInit(GPIOD);
        GPIO_Init(GPIOD, ((GPIO_Pin_TypeDef)GPIO_PIN_3 | GPIO_PIN_4),
GPIO_MODE_OUT_PP_HIGH_FAST);
}


void TIM2_setup(void)
{
        TIM2_DeInit();
        TIM2_TimeBaseInit(TIM2_PRESCALER_32, 1000);
        TIM2_OC1Init(TIM2_OCMODE_PWM1, TIM2_OUTPUTSTATE_ENABLE, 1000,
TIM2_OCPOLARITY_HIGH);
        TIM2_OC2Init(TIM2_OCMODE_PWM1, TIM2_OUTPUTSTATE_ENABLE, 1000,
TIM2_OCPOLARITY_LOW);
        TIM2_OC3Init(TIM2_OCMODE_PWM1, TIM2_OUTPUTSTATE_ENABLE, 1000,
TIM2_OCPOLARITY_HIGH);
        TIM2_Cmd(ENABLE);
}
```

Explanation

Again, the CPU and the peripheral clock is set at 2MHz.

```c
CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);
....
CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, ENABLE);
```

Next, we need to configure the PWM GPIOs as outputs.

```c
void GPIO_setup(void)
{
        GPIO_DeInit(GPIOA);
        GPIO_Init(GPIOA, GPIO_PIN_3, GPIO_MODE_OUT_PP_HIGH_FAST);

        GPIO_DeInit(GPIOD);
        GPIO_Init(GPIOD, ((GPIO_Pin_TypeDef)GPIO_PIN_3 | GPIO_PIN_4),
GPIO_MODE_OUT_PP_HIGH_FAST);
}
```

Just like other microcontrollers, PWM generation involves a timer. Here as said TIM2 is that timer. We need to set time base first before actually configuring the PWM channels.

```c
void TIM2_setup(void)
{
    TIM2_DeInit();
    TIM2_TimeBaseInit(TIM2_PRESCALER_32, 1000);
    TIM2_OC1Init(TIM2_OCMODE_PWM1, TIM2_OUTPUTSTATE_ENABLE, 1000,
TIM2_OCPOLARITY_HIGH);
    TIM2_OC2Init(TIM2_OCMODE_PWM1, TIM2_OUTPUTSTATE_ENABLE, 1000,
TIM2_OCPOLARITY_LOW);
    TIM2_OC3Init(TIM2_OCMODE_PWM1, TIM2_OUTPUTSTATE_ENABLE, 1000,
TIM2_OCPOLARITY_HIGH);
    TIM2_Cmd(ENABLE);
}
```

In the codes above, TIM2 has a time base of 16ms or 62.5kHz. This time base is further divided by the **Output Compare (OC)** unit. Thus, here the 62.5kHz base is further divided by 1000 to get 62.5Hz PWM frequency. The maximum duty cycle is therefore 1000. Additionally, we can set PWM polarity and command the channel if or if not, should it work in an inverted manner.

To change PWM duty, we need to call the following function:

```c
TIM2_SetCompareX(pwm_duty);        //   where X represents channel ID (1, 2 or 3)
```

Note that in STM8 micros, there is a trade-off between duty cycle and PWM frequency. If the PWM resolution, i.e. duty cycle is big then its frequency is small and vice-versa. This is true for all timers.

Demo

Video link: https://www.youtube.com/watch?v=BPS5unUHDz4.
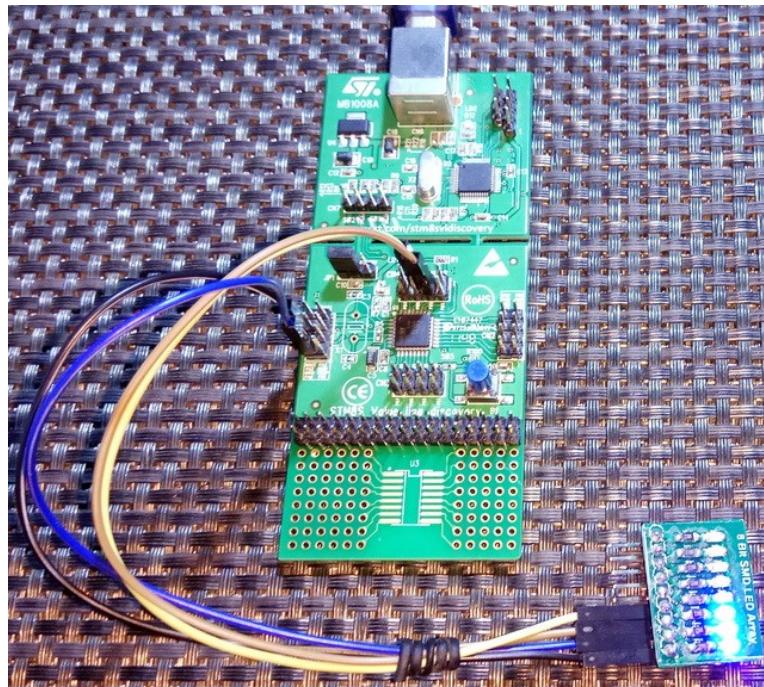
Advanced Pulse Width Modulation (TIM1 PWM)

Timer 1 (TIM1) is an advance timer and so the PWMs generated by it have several additional features that are not available with other timers. For example, it is possible to generate complimentary PWMs with TIM1. Up to three sets of complementary PWMs can be generated. Such PWMs are useful in designing three phase inverters, rectifiers and other power-related tasks. TIM1 PWMs are also very useful for motor control applications. It is also possible to add dead-time and brake. Apart from these TIM1 can also generate PWMs just like GP timers. In this mode however, complimentary PWM outputs are unavailable and up to four independent PWM channels can be obtained.

In this example, I will show how to create complementary PWMs with TIM1 PWM channel 1.

Hardware Connection
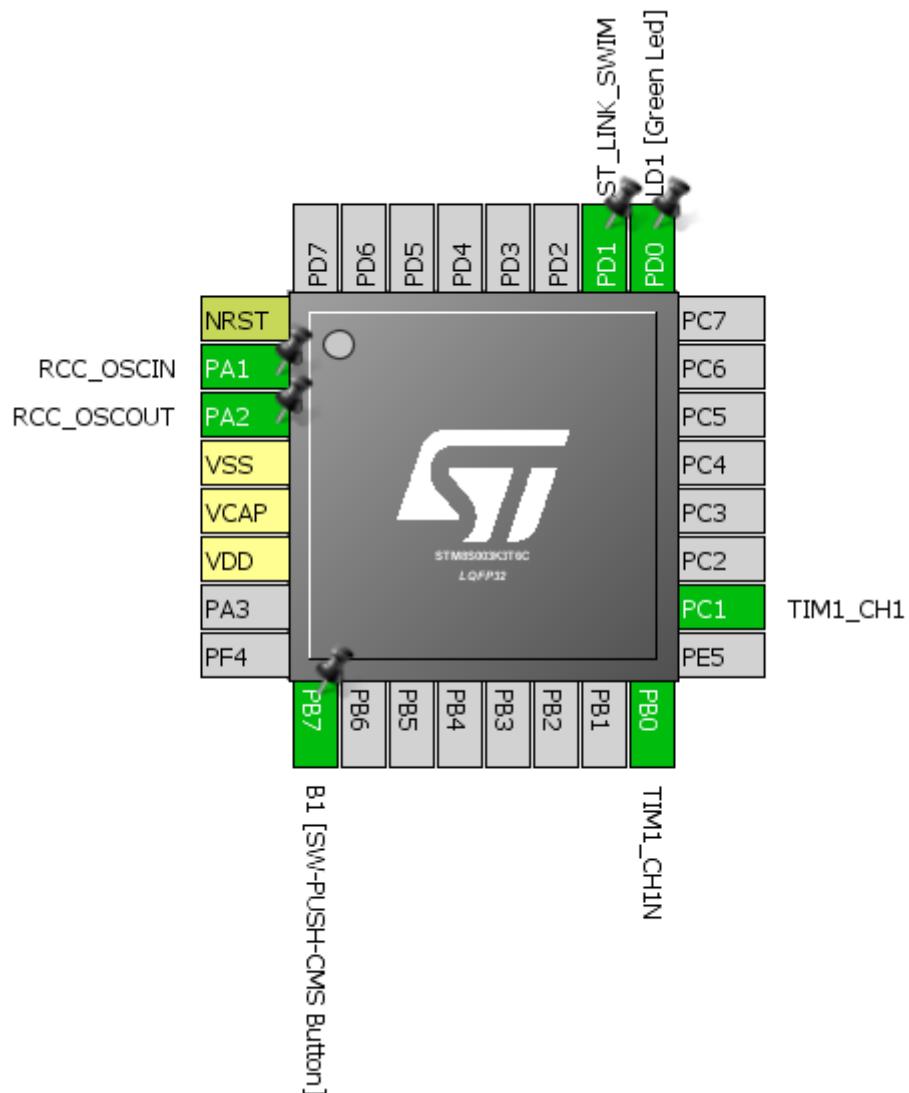
Code Example

```c
#include "STM8S.h"


void clock_setup(void);
void GPIO_setup(void);
void TIM1_setup(void);


void main(void)
{
    signed int i = 0;

    clock_setup();
    GPIO_setup();
    TIM1_setup();

    while(TRUE)
    {
        for(i = 0; i < 1000; i += 1)
        {
            TIM1_SetCompare1(i);
            delay_ms(1);
        }
        for(i = 1000; i > 0; i -= 1)
        {
            TIM1_SetCompare1(i);
            delay_ms(1);
        }
    };
}


void clock_setup(void)
{
    CLK_DeInit();

    CLK_HSECmd(DISABLE);
    CLK_LSICmd(DISABLE);
    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

    CLK_ClockSwitchCmd(ENABLE);
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
    CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

    CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
    DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

    CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, ENABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
```

```
}


void GPIO_setup(void)
{
      GPIO_DeInit(GPIOB);
      GPIO_Init(GPIOB, GPIO_PIN_0, GPIO_MODE_OUT_PP_HIGH_FAST);

      GPIO_DeInit(GPIOC);
      GPIO_Init(GPIOC, GPIO_PIN_1, GPIO_MODE_OUT_PP_HIGH_FAST);
}


void TIM1_setup(void)
{
      TIM1_DeInit();

      TIM1_TimeBaseInit(16, TIM1_COUNTERMODE_UP, 1000, 1);

      TIM1_OC1Init(TIM1_OCMODE_PWM1,
                   TIM1_OUTPUTSTATE_ENABLE,
                   TIM1_OUTPUTNSTATE_ENABLE,
                   1000,
                   TIM1_OCPOLARITY_LOW,
                   TIM1_OCNPOLARITY_LOW,
                   TIM1_OCIDLESTATE_RESET,
                   TIM1_OCNIDLESTATE_RESET);

      TIM1_CtrlPWMOutputs(ENABLE);
      TIM1_Cmd(ENABLE);
}
```

Explanation

This time we used the full 16MHz speed of HSI both for peripheral and CPU clocks:

```
CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);
....
CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, ENABLE);
```

Like as with the previous example PWM output GPIOs are set as outputs:

```
void GPIO_setup(void)
{
      GPIO_DeInit(GPIOB);
      GPIO_Init(GPIOB, GPIO_PIN_0, GPIO_MODE_OUT_PP_HIGH_FAST);

      GPIO_DeInit(GPIOC);
      GPIO_Init(GPIOC, GPIO_PIN_1, GPIO_MODE_OUT_PP_HIGH_FAST);
}
```

TIM1 and OC channel initialization is just like the previous example with some minor differences. The time base generation part seems to have some additional arguments. These are because:

- Unlike other timers, TIM1 prescaler value is not a fixed set of multiples of 2.

- The counting mode is not just up mode counting. Counting mode can also be down counting.

- TIM1 has additional repetition counter.

- Except the basic timers all timers in STM8 are 16-bit timer.

If you open the header file for TIM1, you'll see many functions. Many of these functions are not available with other timers, expressing the power of an advance timer.

Likewise, there are some additional info we must feed when configuring the OC channels. We need to set info about complementary channels even if we don't need them. We can additionally set the default idle states of PWMs apart from polarities.

```c
void TIM1_setup(void)
{
    TIM1_DeInit();

    TIM1_TimeBaseInit(16, TIM1_COUNTERMODE_UP, 1000, 1);

    TIM1_OC1Init(TIM1_OCMODE_PWM1,
                 TIM1_OUTPUTSTATE_ENABLE,
                 TIM1_OUTPUTNSTATE_ENABLE,
                 1000,
                 TIM1_OCPOLARITY_LOW,
                 TIM1_OCNPOLARITY_LOW,
                 TIM1_OCIDLESTATE_RESET,
                 TIM1_OCNIDLESTATE_RESET);

    TIM1_CtrlPWMOutputs(ENABLE);
    TIM1_Cmd(ENABLE);
}
```

To change the duty cycle of a channel, we need to call this function:

```c
TIM1_SetCompareX(duty_cycle);    //   where X represents channel ID (1, 2, 3 or 4)
```

Complementary outputs occur in pairs and so they are interdependent. That's why there is no separate function for outputs labelled **N**, i.e. the inverted ones.

Demo

Video link: https://youtu.be/uclCXH1ZPWU.
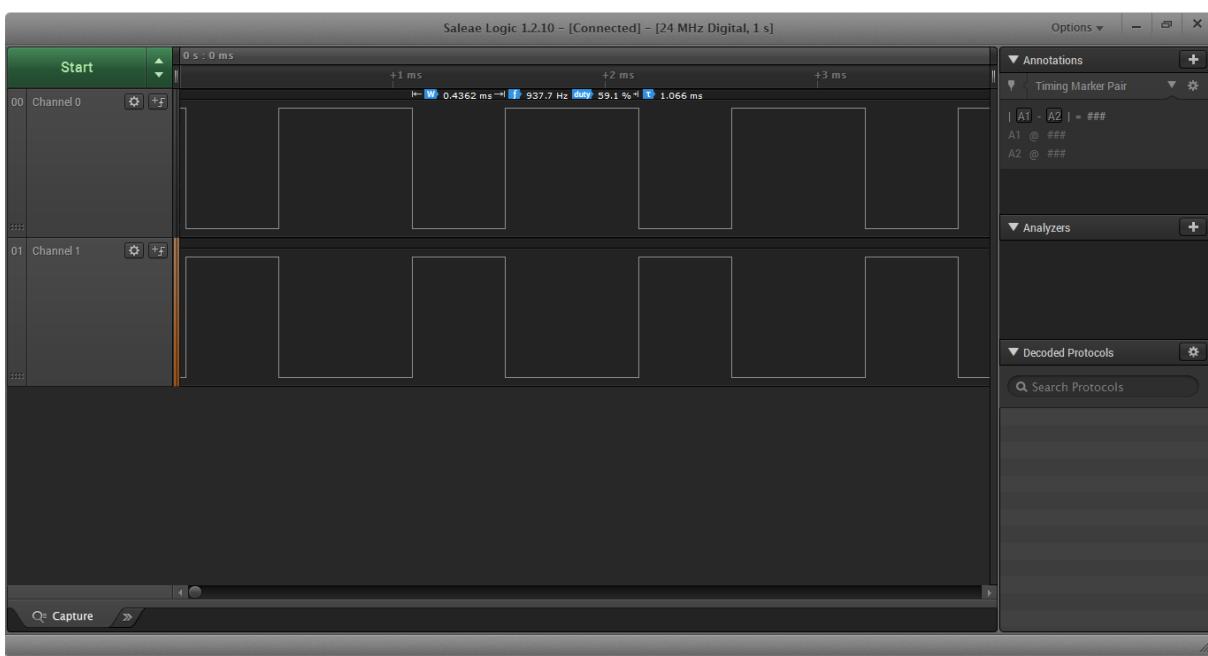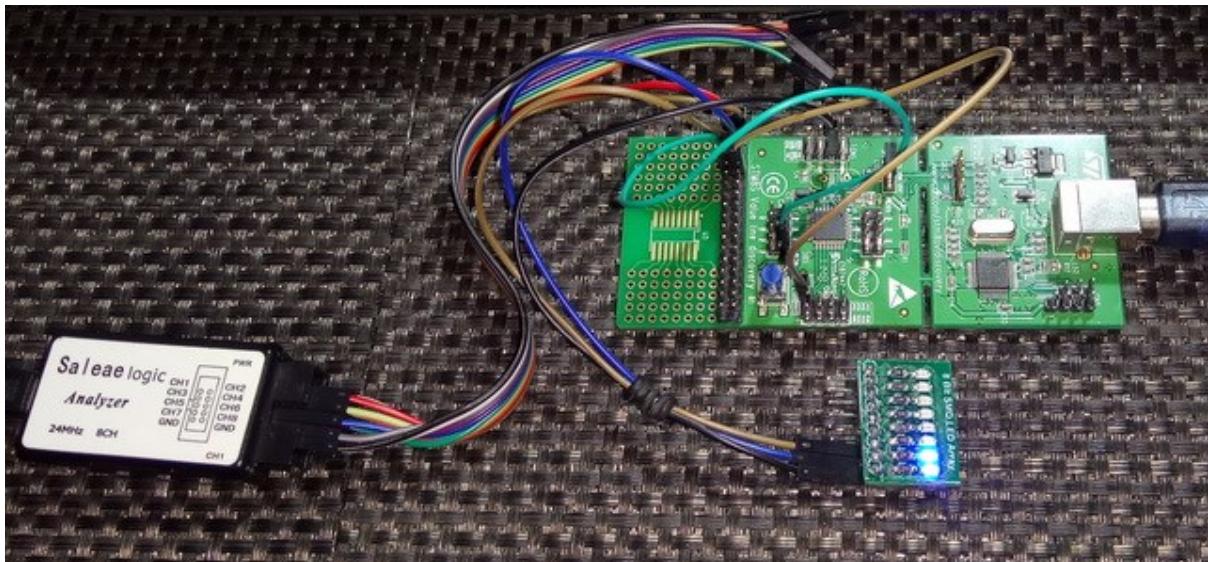
Timer Input Capture (TIM1 & TIM2)

Input capture is needed for measurements of pulses, pulse widths, frequencies, phase detection and similar stuffs. With external interrupts these measurements can be done with some limitations. However, using timer capture has some serious benefits. First of all, accuracy of measurements and secondly timer capture simplifies many tasks as timers themselves time stuffs properly. Dedicated hardware makes stuffs like PWM measurement less complex and resource-friendly too.

STM8 timers have several capture channels just like output compare channels (PWM). The number of input capture channels is same as the number of PWM channels. Except basic timers all timers have input capture option.

Hardware Connection
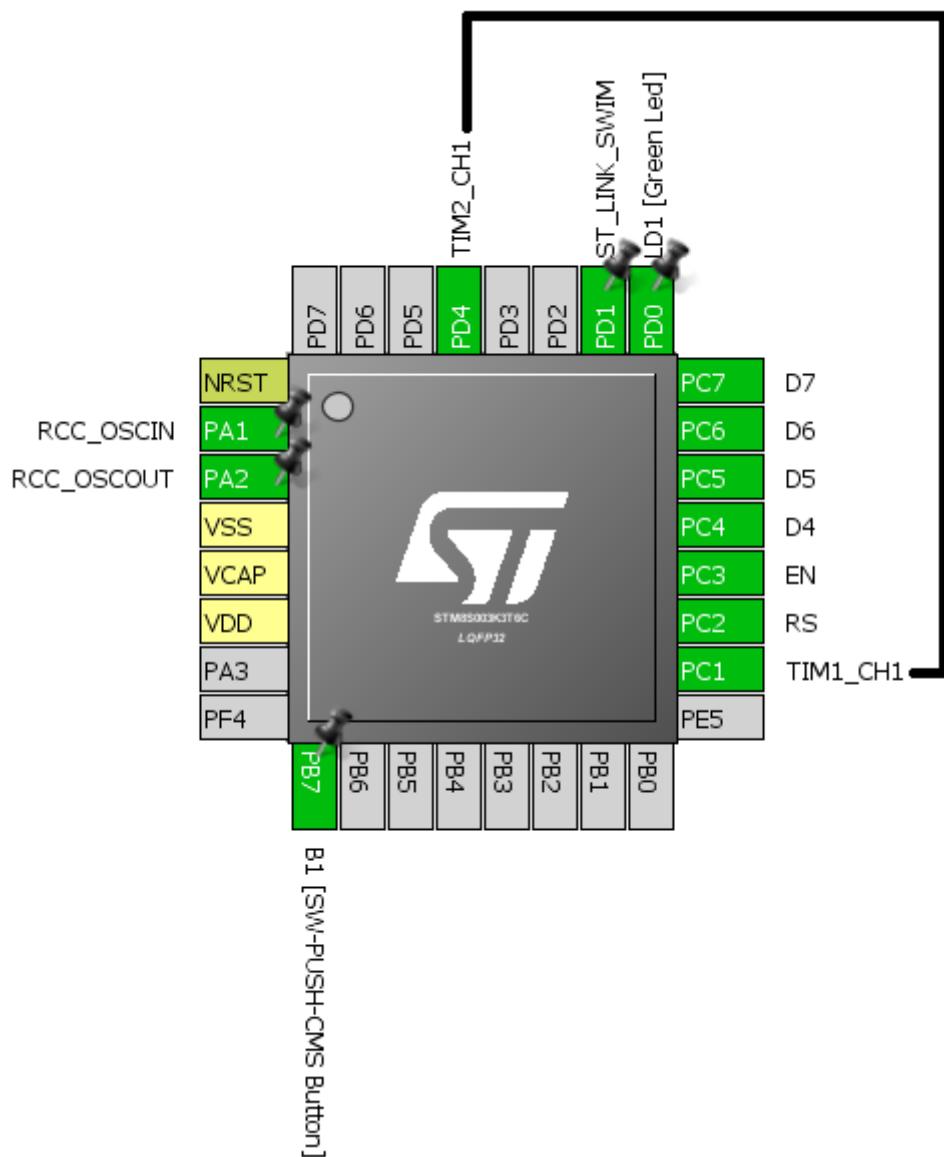
Code Example

In this demo, TIM2 is configured to generate PWM on its CH1 output. TIM1 is configured to capture every rising edge of incoming waveform at its input capture channel CH1. When a capture event occurs, the current time count of TIM1 is saved. By deducting the recent capture count from the previous capture count, we can measure time period of the incoming PWM signal and hence deduce its frequency. If the frequency is too high, TIM1 may overflow and so we need to take care of it too. We, thus, need to check TIM1 overflow event too.

__main.c__

```c
#include "STM8S.h"
#include "lcd.h"


unsigned int overflow_count = 0;
unsigned long pulse_ticks = 0;
unsigned long start_time = 0;
unsigned long end_time = 0;


void clock_setup(void);
void GPIO_setup(void);
void TIM1_setup(void);
void TIM2_setup(void);
void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned long value);


void main(void)
{
    unsigned long time_period = 0;

        clock_setup();
        GPIO_setup();
        TIM1_setup();
        TIM2_setup();
        LCD_init();

        LCD_clear_home();
        LCD_goto(0, 0);
        LCD_putstr("T/ms:");
        delay_ms(10);

        while(TRUE)
        {
                time_period = pulse_ticks;
                lcd_print(0, 1, time_period);
                delay_ms(400);
        };
}


void clock_setup(void)
{
        CLK_DeInit();
```

```c
        CLK_HSECmd(DISABLE);
        CLK_LSICmd(DISABLE);
        CLK_HSICmd(ENABLE);
        while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

        CLK_ClockSwitchCmd(ENABLE);
        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
        CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV2);

        CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
        DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
        GPIO_DeInit(GPIOC);
        GPIO_Init(GPIOC, GPIO_PIN_1, GPIO_MODE_IN_FL_NO_IT);

        GPIO_DeInit(GPIOD);
        GPIO_Init(GPIOD, GPIO_PIN_4, GPIO_MODE_OUT_PP_HIGH_FAST);
}


void TIM1_setup(void)
{
        TIM1_DeInit();
        TIM1_TimeBaseInit(2000, TIM1_COUNTERMODE_UP, 55535, 1);
        TIM1_ICInit(TIM1_CHANNEL_1, TIM1_ICPOLARITY_RISING,
TIM1_ICSELECTION_DIRECTTI, 1, 1);
        TIM1_ITConfig(TIM1_IT_UPDATE, ENABLE);
        TIM1_ITConfig(TIM1_IT_CC1, ENABLE);
        TIM1_Cmd(ENABLE);
        enableInterrupts();
}


void TIM2_setup(void)
{
        TIM2_DeInit();
        TIM2_TimeBaseInit(TIM2_PRESCALER_32, 1250);
        TIM2_OC1Init(TIM2_OCMODE_PWM1, TIM2_OUTPUTSTATE_ENABLE, 1000,
TIM2_OCPOLARITY_LOW);
        TIM2_SetCompare1(625);
        TIM2_Cmd(ENABLE);
}


void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned long value)
{
```

```
        char tmp[6] = {0x20, 0x20, 0x20, 0x20, 0x20, 0x20} ;

        tmp[0] = (((value / 100000) % 10) + 0x30);
        tmp[1] = (((value / 10000) % 10) + 0x30);
        tmp[2] = (((value / 1000) % 10) + 0x30);
        tmp[3] = (((value / 100) % 10) + 0x30);
        tmp[4] = (((value / 10) % 10) + 0x30);
        tmp[5] = ((value % 10) + 0x30);

        LCD_goto(x_pos, y_pos);
        LCD_putstr(tmp);
}
```

stm8_interrupt_vector.c (Interrupt vector address part only)

```
....
{0x82, (interrupt_handler_t)TIM1_UPD_IRQHandler}, /* irq11 */
{0x82, (interrupt_handler_t)TIM1_CH1_CCP_IRQHandler}, /* irq12 */
....
```

stm8s_it.h (Top part only)

```
#ifndef __STM8S_IT_H
#define __STM8S_IT_H


@far @interrupt void TIM1_UPD_IRQHandler(void);
@far @interrupt void TIM1_CH1_CCP_IRQHandler(void);

/* Includes --------------------------------------------------------------------*/
#include "stm8s.h"
....
```

stm8s_it.c (Top part only)

```
#include "stm8s.h"
#include "stm8s_it.h"


extern unsigned int overflow_count;
extern unsigned long pulse_ticks;
extern unsigned long start_time;
extern unsigned long end_time;


void TIM1_UPD_IRQHandler(void)
{
        overflow_count++;
        TIM1_ClearITPendingBit(TIM1_IT_UPDATE);
        TIM1_ClearFlag(TIM1_FLAG_UPDATE);
}


void TIM1_CH1_CCP_IRQHandler(void)
{
        end_time = TIM1_GetCapture1();
        pulse_ticks = ((overflow_count << 16) - start_time + end_time);
```

```
        start_time = end_time;
        overflow_count = 0;
        TIM1_ClearITPendingBit(TIM1_IT_CC1);
        TIM1_ClearFlag(TIM1_FLAG_CC1);
}
....
```

Explanation

The clocks and peripherals are set first. We are using 2MHz peripheral clock and the CPU is running at 0.5MHz.

```
CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV2);
....
CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, ENABLE);
CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, ENABLE);
```

GPIOs must be set too. Since TIM2 is to output PWM, its CH1 must be set output. Likewise, TIM1's CH1 GPIO must be set as an input.

```
void GPIO_setup(void)
{
        GPIO_DeInit(GPIOC);
        GPIO_Init(GPIOC, GPIO_PIN_1, GPIO_MODE_IN_FL_NO_IT);

        GPIO_DeInit(GPIOD);
        GPIO_Init(GPIOD, GPIO_PIN_4, GPIO_MODE_OUT_PP_HIGH_FAST);
}
```

TIM1 need to be configured for input capture. We need to set time base for TIM1 first. It is set as such that TIM1 will overflow every second. Then we set input capture channel by specifying the edge sensitivity, channel, mode and scalars. Since we will be using interrupts, we must enable relevant interrupts.

```
void TIM1_setup(void)
{
        TIM1_DeInit();
        TIM1_TimeBaseInit(2000, TIM1_COUNTERMODE_UP, 55535, 1);
        TIM1_ICInit(TIM1_CHANNEL_1, TIM1_ICPOLARITY_RISING,
TIM1_ICSELECTION_DIRECTTI, 1, 1);
        TIM1_ITConfig(TIM1_IT_UPDATE, ENABLE);
        TIM1_ITConfig(TIM1_IT_CC1, ENABLE);
        TIM1_Cmd(ENABLE);
        enableInterrupts();
}
```

TIM2 is set for PWM generation on its CH1. The generated PWM will have a frequency of 50Hz and 50% duty cycle. The setup of TIM2 should be by now self-explanatory:

```
void TIM2_setup(void)
{
    TIM2_DeInit();
    TIM2_TimeBaseInit(TIM2_PRESCALER_32, 1250);
    TIM2_OC1Init(TIM2_OCMODE_PWM1, TIM2_OUTPUTSTATE_ENABLE, 1000,
TIM2_OCPOLARITY_LOW);
    TIM2_SetCompare1(625);
    TIM2_Cmd(ENABLE);
}
```

In the vector table of **stm8_interrupt_vector.c** file, we need to specify the interrupts we will be using:

```
....
{0x82, (interrupt_handler_t)TIM1_UPD_IRQHandler}, /* irq11 */
{0x82, (interrupt_handler_t)TIM1_CH1_CCP_IRQHandler}, /* irq12 */
....
```

We have to specify the interrupt subroutine (ISR) prototype functions in the **stm8s_it.h** file. These functions are the places where the code will jump when respective interrupt occurs:

```
@far @interrupt void TIM1_UPD_IRQHandler(void);
@far @interrupt void TIM1_CH1_CCP_IRQHandler(void);
```

The ISR functions are coded in the **stm8s_it.c** file:

```
void TIM1_UPD_IRQHandler(void)
{
    overflow_count++;
    TIM1_ClearITPendingBit(TIM1_IT_UPDATE);
    TIM1_ClearFlag(TIM1_FLAG_UPDATE);
}
```

The first part is dealing with TIM1 overflow. If a capture occurs when TIM1 count is near to reset value we need to take this account. This part does so and an overflow counter is incremented.

```
void TIM1_CH1_CCP_IRQHandler(void)
{
    end_time = TIM1_GetCapture1();
    pulse_ticks = ((overflow_count << 16) - start_time + end_time);
    start_time = end_time;
    overflow_count = 0;
    TIM1_ClearITPendingBit(TIM1_IT_CC1);
    TIM1_ClearFlag(TIM1_FLAG_CC1);
}
```

The second part is where TIM1 capture is recorded. Once a rising edge is captured, an interrupt is issued. In the interrupt, we must first save the current TIM1 counter count in the variable named **end_time**. The formula for pulse tick is then computed. Note how the TIM1 overflow is addressed in the formula. The new start time should be the previous capture time because we need to deduct old capture count from new capture count. Lastly, overflow counter, capture flag and pending interrupts are cleared.

In the main loop, we are just displaying the time period of capture in a LCD while everything is being processed in the background inside interrupts:

```
while(TRUE)
{
    time_period = pulse_ticks;
    lcd_print(0, 1, time_period);
    delay_ms(400);
};
```

Demo

Video link: https://youtu.be/bzLUDwuFQTw.
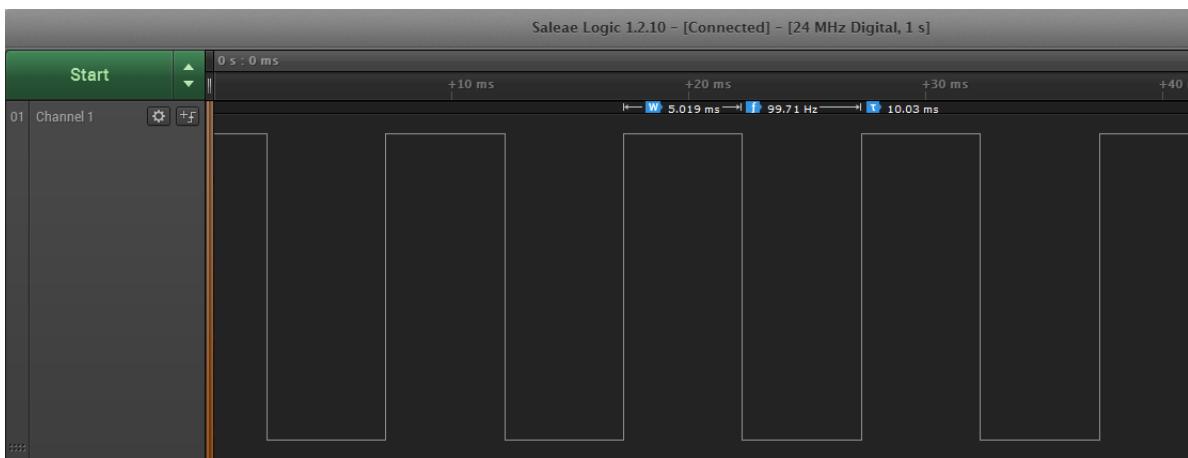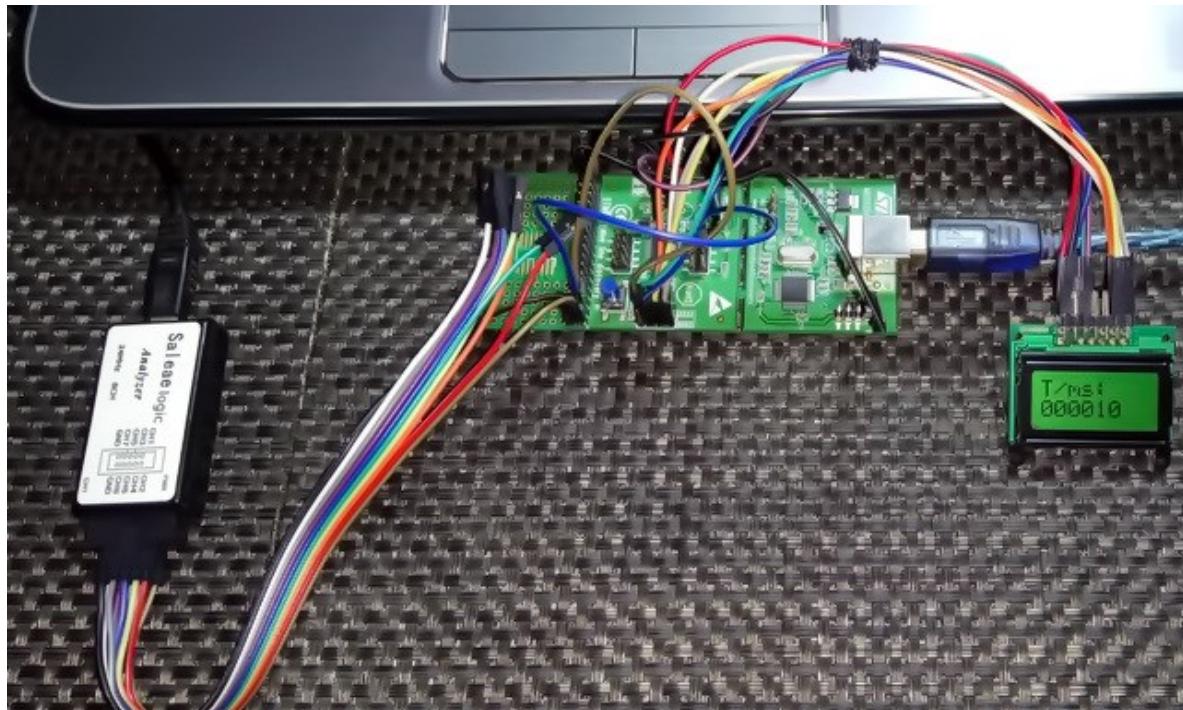
Encoder Interface

In applications like motor control, it is often required to know how much a motor has rotated, how fast it is rotating and in which direction. It is possible to deduce these feedback data with interrupts, timers and even with tricky GPIO coding. However, using methods employing GPIOs, interrupts and timers will add some resource overhead and will also add up extra coding efforts. STM8s have in-built encoder decoding interface that can be used in such scenarios without extra coding or hardware addition.
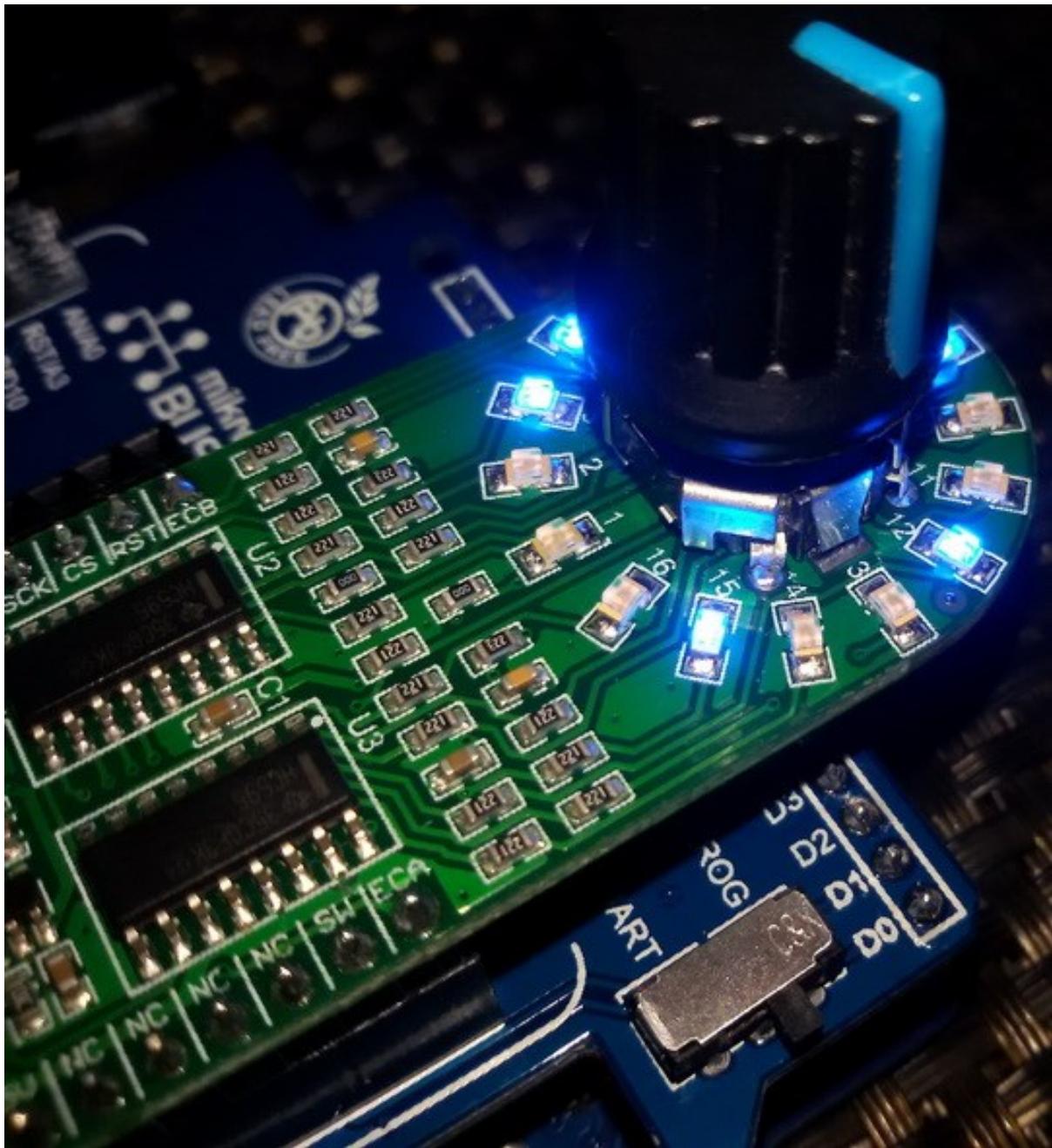
The applications of encoder decoding interface are not only limited to acquiring motor feedbacks. The encoder interface can also be used to interface rotary encoders, linear actuators, digital volume control, etc. Here in this example, I demonstrated this encoder interface with a rotary encoder.

Hardware Connection
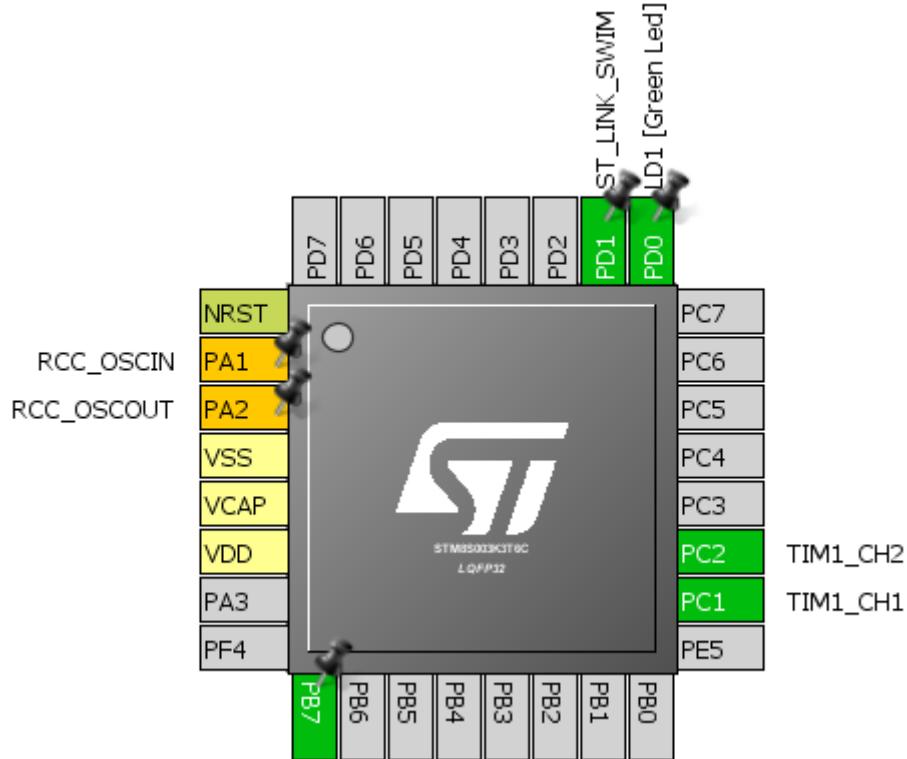
Code Example

```c
#include "STM8S.h"
#include "lcd.h"


#define LED_Port                        GPIOD
#define LED_Pin                         GPIO_PIN_0


unsigned char bl_state;
unsigned char data_value;


void clock_setup(void);
void GPIO_setup(void);
void TIM1_setup(void);
void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned int value);


void main()
{
    unsigned int present_value = 0x0000;
    unsigned int previous_value = 0x0001;

    clock_setup();
    GPIO_setup();
```

```c
        TIM1_setup();

        LCD_init();
        LCD_clear_home();

        LCD_goto(1, 0);
        LCD_putstr("STM8S QEI Test");
        LCD_goto(0, 1);
        LCD_putstr("Value:");

        while(TRUE)
        {
                present_value = TIM1_GetCounter();
                if(present_value != previous_value)
                {
                        lcd_print(12, 1, present_value);
                        GPIO_WriteReverse(LED_Port, LED_Pin);
                }

                previous_value = present_value;
        };
}


void clock_setup(void)
{
        CLK_DeInit();

        CLK_HSECmd(DISABLE);
        CLK_LSICmd(DISABLE);
        CLK_HSICmd(ENABLE);
        while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

        CLK_ClockSwitchCmd(ENABLE);
        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV4);
                CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

        CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI, DISABLE,
                                CLK_CURRENTCLOCKSTATE_ENABLE);

        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
        GPIO_DeInit(GPIOC);
        GPIO_Init(GPIOC,
                ((GPIO_Pin_TypeDef)(GPIO_PIN_1 | GPIO_PIN_2)),
                GPIO_MODE_IN_PU_NO_IT);

        GPIO_DeInit(GPIOD);
```

```
        GPIO_Init(LED_Port, LED_Pin, GPIO_MODE_OUT_OD_HIZ_FAST);
}


void TIM1_setup(void)
{
        TIM1_DeInit();

        TIM1_TimeBaseInit(10, TIM1_COUNTERMODE_UP, 1000, 1);

        TIM1_EncoderInterfaceConfig(TIM1_ENCODERMODE_TI12,
                                    TIM1_ICPOLARITY_FALLING,
                                    TIM1_ICPOLARITY_FALLING);

        TIM1_Cmd(ENABLE);
}


void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned int value)
{
        char tmp[5] = {0x20, 0x20, 0x20, 0x20, '\0'} ;

        tmp[0] = ((value / 1000) + 0x30);
        tmp[1] = (((value / 100) % 10) + 0x30);
        tmp[2] = (((value / 10) % 10) + 0x30);
        tmp[3] = ((value % 10) + 0x30);

        LCD_goto(x_pos, y_pos);
        LCD_putstr(tmp);
}
```

Explanation

A rotary/quadrature encoder gives two outputs that are slightly out of phase:
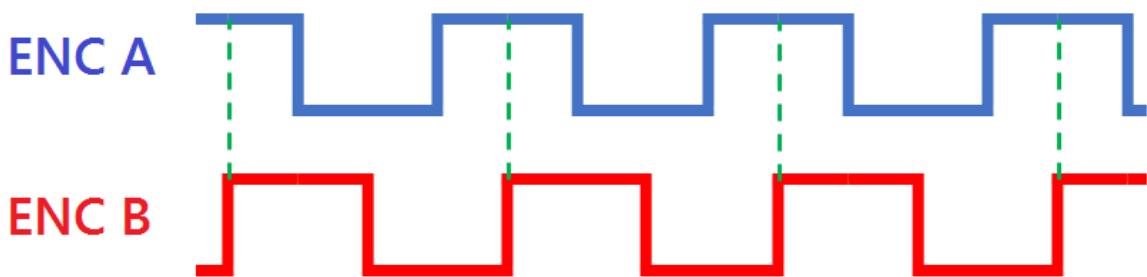
Thus, to interface such an encoder, we will need two timer inputs. **TI12** are those inputs. **TI12** are mapped to **TIM1_CH1** and **TIM1_CH2** respectively. These inputs need to be pulled up unless external pull-up resistors are used.

```
GPIO_DeInit(GPIOC);
GPIO_Init(GPIOC,
        ((GPIO_Pin_TypeDef)(GPIO_PIN_1 | GPIO_PIN_2)),
        GPIO_MODE_IN_PU_NO_IT);
```

The peripheral and CPU clocks are set as follows:

```
CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV4);
CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI, DISABLE,
                      CLK_CURRENTCLOCKSTATE_ENABLE);

CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, ENABLE);
CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, ENABLE);
```

Since TIM1's encoder interface is used, TIM1 peripheral clock is enabled and the I2C clock is enabled for the I2C LCD module.

Then, the main thing to explain here is the way the timer is set for encoder interface:

```
void TIM1_setup(void)
{
      TIM1_DeInit();

      TIM1_TimeBaseInit(10, TIM1_COUNTERMODE_UP, 1000, 1);

      TIM1_EncoderInterfaceConfig(TIM1_ENCODERMODE_TI12,
                                  TIM1_ICPOLARITY_FALLING,
                                  TIM1_ICPOLARITY_FALLING);

      TIM1_Cmd(ENABLE);
}
```

The 1000 in the time base setup is maximum encoder interface count, i.e. the encoder interface's output will span from 0 to 1000.

Active edge	Level on opposite signal (TI1FP1 for TI2, TI2FP2 for TI1)	TI1FP1 signal		TI2FP2 signal	
		Rising	Falling	Rising	Falling
Counting on TI1 only	High	Down	Up	No count	No count
	Low	Up	Down	No count	No count
Counting on TI2 only	High	No count	No count	Up	Down
	Low	No count	No count	Down	Up
Counting on both TI1 and TI2	High	Down	Up	Up	Down
	Low	Up	Down	Down	Up

Since we are using two inputs, the encoder interface is setup according to the third option shown above.

To get current encoder count, TIM1's counter is read as follows:

```
present_value = TIM1_GetCounter();
```

Demo

Video link: https://www.youtube.com/watch?v=YQW0VAG7eM4.
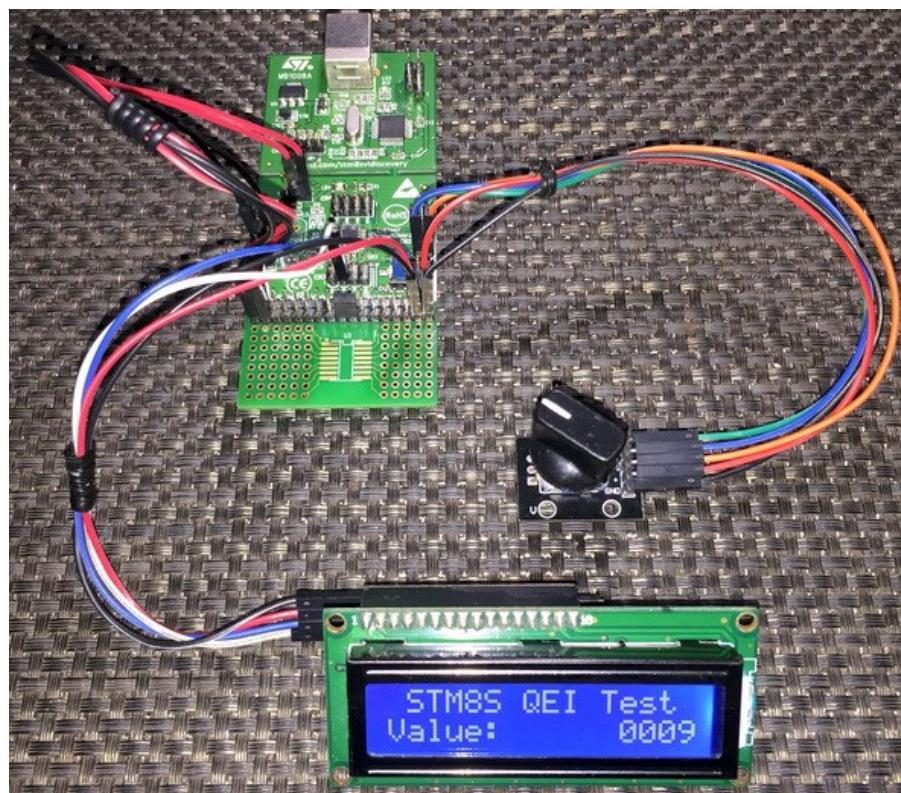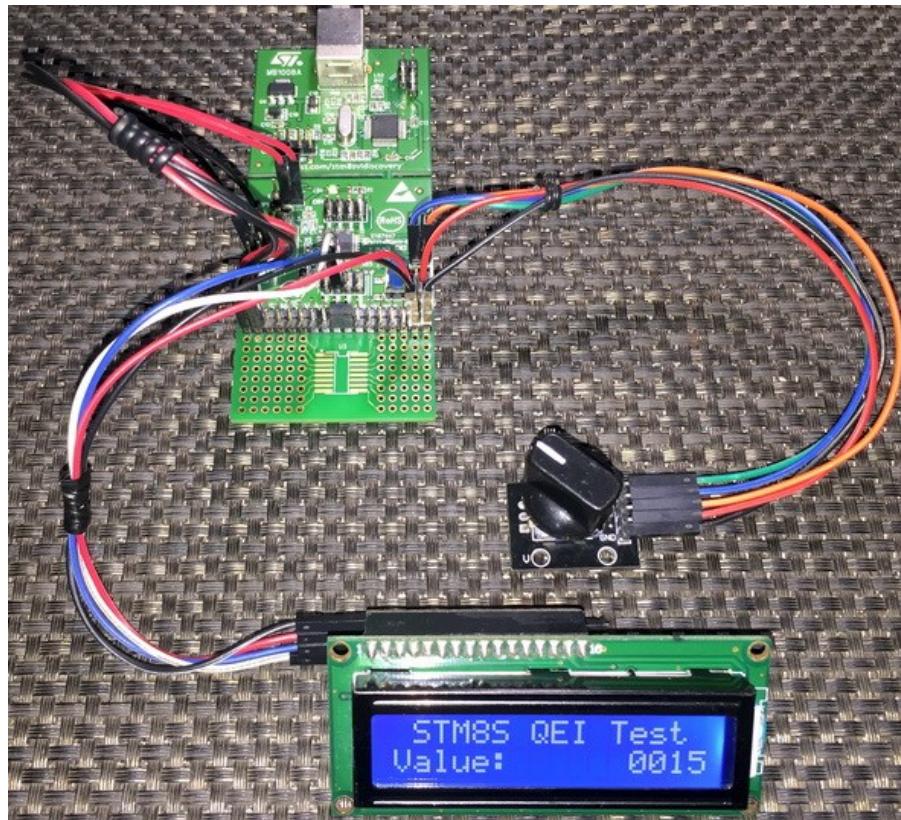
One Pulse Mode (OPM)

One Pulse Mode (OPM) is an additional feature of STM8 timers. It is like a monostable multivibrator or a one-shot timer that can be used to generate a pulse of fixed width after a defined amount of delay. It has many uses like wake-up command generator for other onboard devices/microcontrollers, trigger for sensors like DHT22, etc.

Hardware Connection

Code Example

```c
#include "STM8S.h"


void clock_setup(void);
void GPIO_setup(void);
void TIM1_setup(void);


void main(void)
{
    clock_setup();
    GPIO_setup();
    TIM1_setup();

    while(TRUE)
    {
    };
}
```

```c
void clock_setup(void)
{
    CLK_DeInit();

    CLK_HSECmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSERDY) == FALSE);

    CLK_LSICmd(DISABLE);

    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

    CLK_ClockSwitchCmd(ENABLE);
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
    CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

    CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSE,
    DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, ENABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART2, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER3, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
    GPIO_DeInit(GPIOC);
    GPIO_Init(GPIOC, GPIO_PIN_4, GPIO_MODE_OUT_PP_HIGH_FAST);
    GPIO_WriteHigh(GPIOC, GPIO_PIN_4);
    delay_ms(2000);
}


void TIM1_setup(void)
{
    TIM1_DeInit();

    TIM1_TimeBaseInit(160, TIM1_COUNTERMODE_UP, 10000, 1);

    TIM1_OC4Init(TIM1_OCMODE_PWM2,
                TIM1_OUTPUTSTATE_ENABLE,
                6000,
                TIM1_OCPOLARITY_HIGH,
                TIM1_OCIDLESTATE_RESET);

    TIM1_SelectOnePulseMode(TIM1_OPMODE_SINGLE);
    TIM1_CtrlPWMOutputs(ENABLE);
    TIM1_Cmd(ENABLE);
}
```

Explanation

Firstly, the peripheral and CPU clocks are set at max speed:

```
CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);
....
CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, ENABLE);
```

TIM1 is setup is as simply as we would do for PWM generation. However, for one pulse mode (OPM) we must select one pulse mode.

```
void TIM1_setup(void)
{
    TIM1_DeInit();

    TIM1_TimeBaseInit(160, TIM1_COUNTERMODE_UP, 10000, 1);

    TIM1_OC4Init(TIM1_OCMODE_PWM2,
                 TIM1_OUTPUTSTATE_ENABLE,
                 6000,
                 TIM1_OCPOLARITY_HIGH,
                 TIM1_OCIDLESTATE_RESET);

    TIM1_SelectOnePulseMode(TIM1_OPMODE_SINGLE);
    TIM1_CtrlPWMOutputs(ENABLE);
    TIM1_Cmd(ENABLE);
}
```

At this point it is necessary to describe TIM1 PWM modes. There are two PWM mode – PWM mode 1 and PWM mode 2.

PWM Mode 1

In up-counting mode, the channel is active if the counter is less than CCR1 or else it is inactive. The opposite is true in down-counting mode.

PWM Mode 2

The channel is inactive as long as the counter is less than CCR1 in up- counting mode.

For OPM example here, we need PWM mode 2.

In this example the pulse width time of the one pulse is about 0.25ms and has a delay of 0.375ms. The calculation is very simple. Since 16MHz clock is used, one cycle takes 62.5ns and so 10000 counts equal 0.625ms. Because we are using PWM mode 2, it will take more than 6000 counts or 0.375ms before the one pulse start. The pulse has a width of (10000 - 6000) = 4000 counts or 0.25ms and will end after this time.

Demo
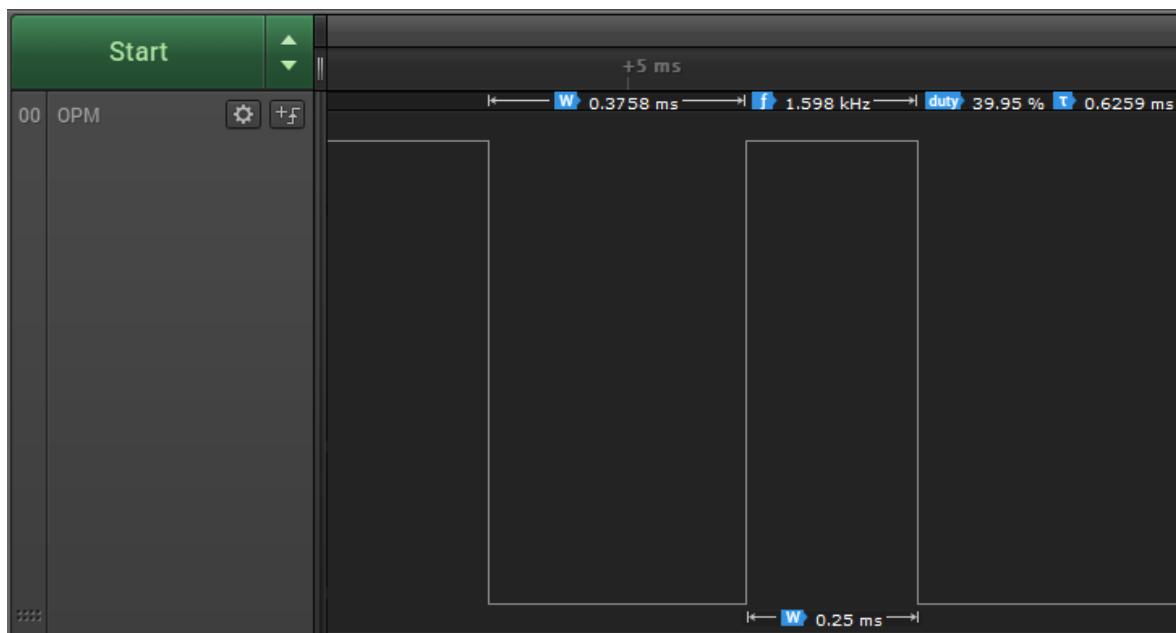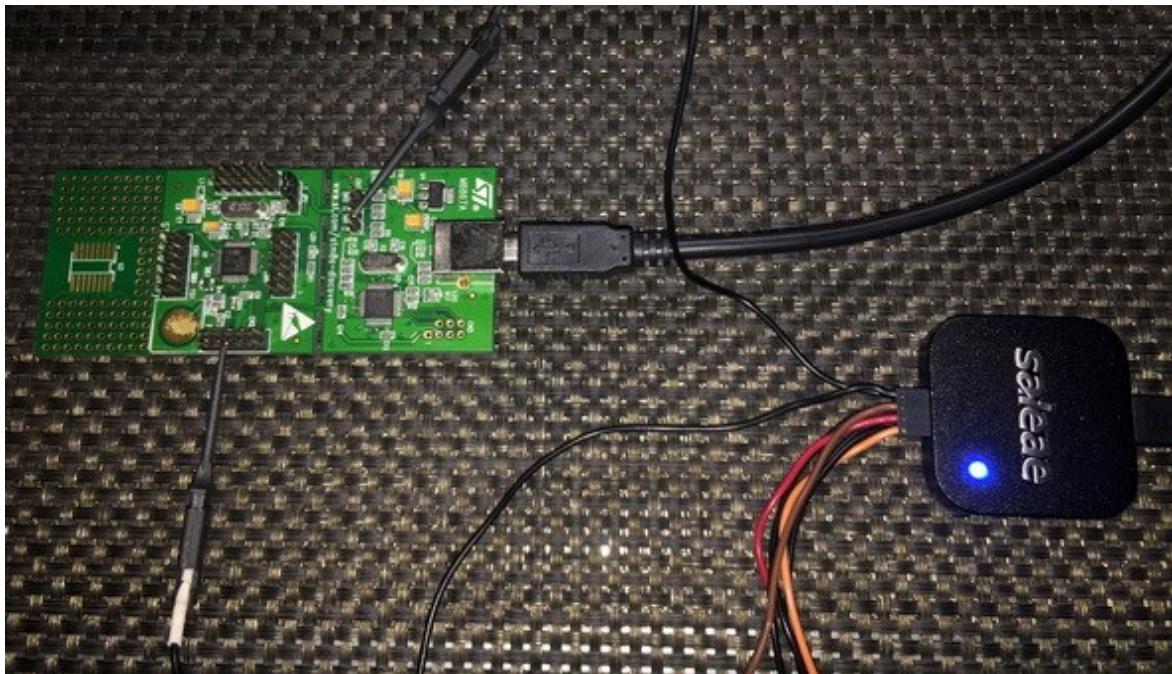
Video link: https://www.youtube.com/watch?v=zJy3_cCYUfY.

PWM Input Mode (PWMI)

With timer capture input we can capture the frequency of an incoming waveform. However, this is not the only stuff that we can do with capture hardware. We can use it to determine the waveform's duty cycle too. Deducing duty cycle enable us to do lot of other stuffs like decoding IR remote signal patterns, measuring time to determine distance measured by an ultrasonic rangefinder sensor, measuring time slot lengths of a one-wire sensor like DHT11 and so on. A very clever feature of STM8 timers is the PWM input capture mode. With this mode we can time the pulse widths of an incoming signal.

Hardware Connection
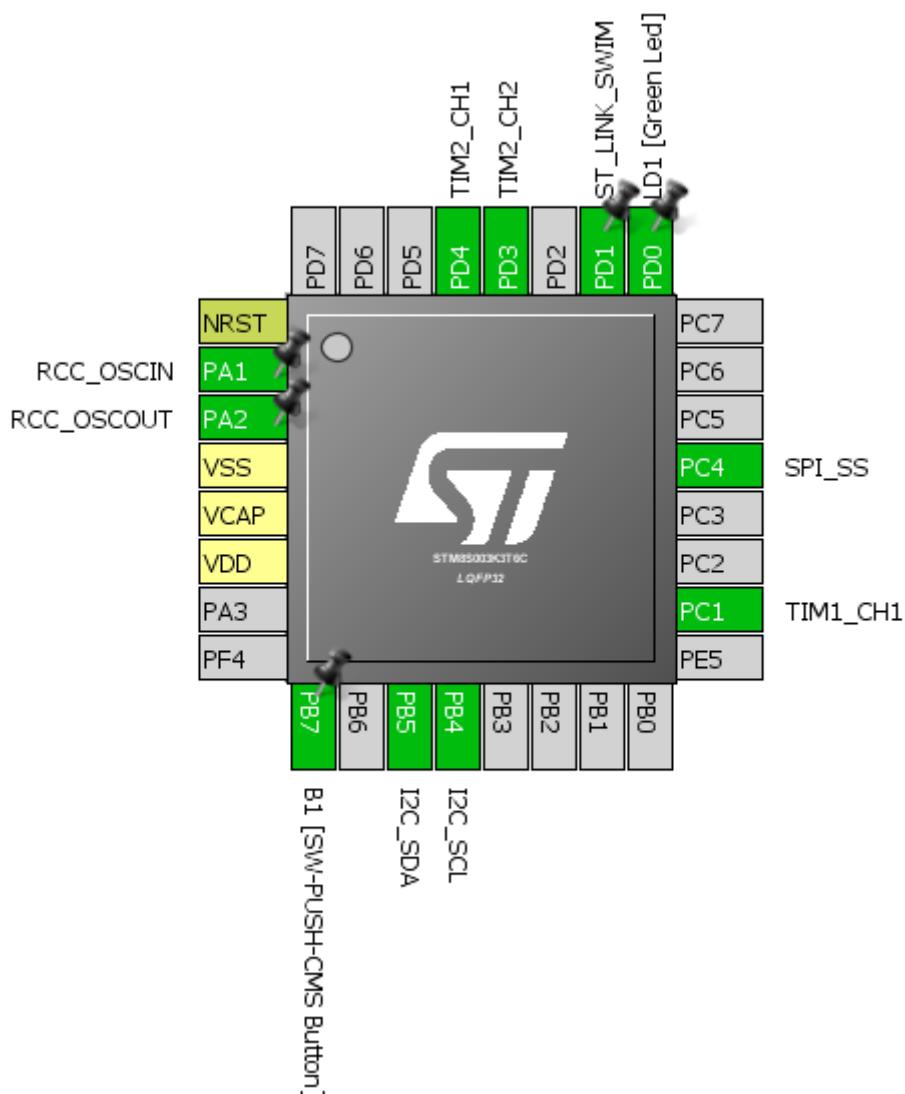
Code Example

stm8s_it.h (top part only)

```c
#ifndef __STM8S_IT_H
#define __STM8S_IT_H


@far @interrupt void TIM1_CH1_CCP_IRQHandler(void);



/* Includes ----------------------------------------------------------*/
#include "stm8s.h"
....
```

stm8s_it.c (top part only)

```c
#include "stm8s.h"
#include "stm8s_it.h"


extern signed long duty_cycle;


void TIM1_CH1_CCP_IRQHandler(void)
{
      duty_cycle = TIM1_GetCapture2();
      TIM1_ClearITPendingBit(TIM1_IT_CC2);
}
....
```

stm8_interrupt_vector.c (shortened)

```c
#include "STM8S.h"
#include "stm8s_it.h"


typedef void @far (*interrupt_handler_t)(void);

struct interrupt_vector {
      unsigned char interrupt_instruction;
      interrupt_handler_t interrupt_handler;
};


extern void _stext();       /* startup routine */

struct interrupt_vector const _vectab[] = {
      {0x82, (interrupt_handler_t)_stext}, /* reset */
      {0x82, NonHandledInterrupt}, /* trap   */
      {0x82, NonHandledInterrupt}, /* irq0   */
      ....
      {0x82, (interrupt_handler_t)TIM1_CH1_CCP_IRQHandler}, /* irq12 */
      ....
      {0x82, NonHandledInterrupt}, /* irq29 */
};
```

main.c

```c
#include "STM8S.h"
#include "lcd.h"


unsigned char bl_state;
unsigned char data_value;

signed long duty_cycle = 0;


void clock_setup(void);
void GPIO_setup(void);
void TIM1_setup(void);
void TIM2_setup(void);
void print_I(unsigned char x_pos, unsigned char y_pos, signed long value);


void main()
{
    unsigned int i = 100;

    clock_setup();
    GPIO_setup();
    TIM1_setup();
    TIM2_setup();
    LCD_init();

    LCD_clear_home();
    LCD_goto(0, 0);
    LCD_putstr("PWM Capture Test");
    LCD_goto(0, 1);
    LCD_putstr("T/ms:");
    delay_ms(10);

    while(TRUE)
    {
        if(GPIO_ReadInputPin(GPIOB, GPIO_PIN_7) == RESET)
        {
            GPIO_WriteLow(GPIOD, GPIO_PIN_0);
            delay_ms(100);
            while(GPIO_ReadInputPin(GPIOB, GPIO_PIN_7) == RESET);
            GPIO_WriteHigh(GPIOD, GPIO_PIN_0);

            i += 100;
            if(i  > 1000)
            {
                i = 100;
            }

            TIM2_SetCompare1(i);
            TIM2_SetCompare2(i);
        }

        print_I(11, 1, duty_cycle);
        delay_ms(100);
    };
}
```

```c
void clock_setup(void)
{
	CLK_DeInit();

	CLK_HSECmd(DISABLE);
	CLK_LSICmd(DISABLE);
	CLK_HSICmd(ENABLE);
	while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

	CLK_ClockSwitchCmd(ENABLE);
	CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
	CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

	CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
	DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

	CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, ENABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, ENABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, ENABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
	GPIO_DeInit(GPIOB);
	GPIO_Init(GPIOB, GPIO_PIN_7, GPIO_MODE_IN_PU_NO_IT);

	GPIO_DeInit(GPIOC);
	GPIO_Init(GPIOC, GPIO_PIN_1, GPIO_MODE_IN_FL_NO_IT);

	GPIO_DeInit(GPIOD);
	GPIO_Init(GPIOD, GPIO_PIN_0, GPIO_MODE_OUT_PP_HIGH_FAST);
	GPIO_Init(GPIOD, GPIO_PIN_3, GPIO_MODE_OUT_PP_HIGH_FAST);
	GPIO_Init(GPIOD, GPIO_PIN_4, GPIO_MODE_OUT_PP_HIGH_FAST);
}


void TIM1_setup(void)
{
	TIM1_DeInit();
	TIM1_TimeBaseInit(2000, TIM1_COUNTERMODE_UP, 55535, 1);
	TIM1_CCxCmd(TIM1_CHANNEL_1, ENABLE);
	TIM1_CCxCmd(TIM1_CHANNEL_2, ENABLE);

	TIM1_PWMIConfig(TIM1_CHANNEL_1,
				TIM1_ICPOLARITY_RISING,
				TIM1_ICSELECTION_DIRECTTI,
				TIM1_ICPSC_DIV1,
				0);

	TIM1_PWMIConfig(TIM1_CHANNEL_2,
				TIM1_ICPOLARITY_FALLING,
```

```c
                         TIM1_ICSELECTION_INDIRECTTI,
                         TIM1_ICPSC_DIV1,
                         0);

    TIM1_SelectInputTrigger(TIM1_TS_TI1FP1);
    TIM1_SelectSlaveMode(TIM1_SLAVEMODE_RESET);

    TIM1_ClearFlag(TIM1_FLAG_CC2);
    TIM1_ITConfig(TIM1_IT_CC2, ENABLE);
    TIM1_Cmd(ENABLE);
    enableInterrupts();
}


void TIM2_setup(void)
{
    TIM2_DeInit();
    TIM2_TimeBaseInit(TIM2_PRESCALER_32, 1250);
    TIM2_OC1Init(TIM2_OCMODE_PWM2, TIM2_OUTPUTSTATE_ENABLE, 1000,
TIM2_OCPOLARITY_LOW);
    TIM2_OC2Init(TIM2_OCMODE_PWM2, TIM2_OUTPUTSTATE_ENABLE, 1000,
TIM2_OCPOLARITY_LOW);
    TIM2_SetCompare1(100);
    TIM2_SetCompare2(100);
    TIM2_Cmd(ENABLE);
}


void print_I(unsigned char x_pos, unsigned char y_pos, signed long value)
{
    char tmp[6] = {0x20, 0x20, 0x20, 0x20, 0x20, '\0'} ;

    tmp[0] = ((value / 10000) + 0x30);
    tmp[1] = (((value / 1000) % 10) + 0x30);
    tmp[2] = (((value / 100) % 10) + 0x30);
    tmp[3] = (((value / 10) % 10) + 0x30);
    tmp[4] = ((value % 10) + 0x30);

    LCD_goto(x_pos, y_pos);
    LCD_putstr(tmp);
}
```

Explanation

The setup and code for this example is same as the timer input capture example. There are few difference though. Unlike the timer input capture example in which TIM2's output waveform's period was measured, here the duty cycle or pulse on time of TIM2's output waveform is measured. Additionally, the following lines of code are added for TIM1 capture hardware:

```
TIM1_PWMIConfig(TIM1_CHANNEL_1,
                TIM1_ICPOLARITY_RISING,
                TIM1_ICSELECTION_DIRECTTI,
                TIM1_ICPSC_DIV1,
                0);

TIM1_PWMIConfig(TIM1_CHANNEL_2,
                TIM1_ICPOLARITY_FALLING,
                TIM1_ICSELECTION_INDIRECTTI,
                TIM1_ICPSC_DIV1,
                0);

TIM1_SelectInputTrigger(TIM1_TS_TI1FP1);
TIM1_SelectSlaveMode(TIM1_SLAVEMODE_RESET);
```

The above code can alternatively be written as follows:

```
TIM1_ICInit(TIM1_CHANNEL_1,
            TIM1_ICPOLARITY_RISING,
            TIM1_ICSELECTION_DIRECTTI,
            TIM1_ICPSC_DIV1,
            0);

TIM1_ICInit(TIM1_CHANNEL_2,
            TIM1_ICPOLARITY_FALLING,
            TIM1_ICSELECTION_INDIRECTTI,
            TIM1_ICPSC_DIV1,
            0);

TIM1_SelectInputTrigger(TIM1_TS_TI1FP1);
TIM1_SelectSlaveMode(TIM1_SLAVEMODE_RESET);
```

The main theme of PWM input measurement is to capture both edges of a pulse and time the difference between these captures. Physically just one GPIO is needed for this purpose but internally two timer channels are assigned for the task. One is set to capture rising edge while the other falling edge. The first channel which is set to capture rising edge, triggers or starts the timer while the second resets it. For the first channel, we do not need to set any interrupt as it will start the timer automatically when it has sensed a rising edge. However, for the second channel, capture interrupt is needed to notify completion of a duty cycle capture.

```
TIM1_ClearFlag(TIM1_FLAG_CC2);
TIM1_ITConfig(TIM1_IT_CC2, ENABLE);
```

Owing to this, channel 2's capture data is extracted to compute duty cycle.

```
duty_cycle = TIM1_GetCapture2();
TIM1_ClearITPendingBit(TIM1_IT_CC2);
```

Demo
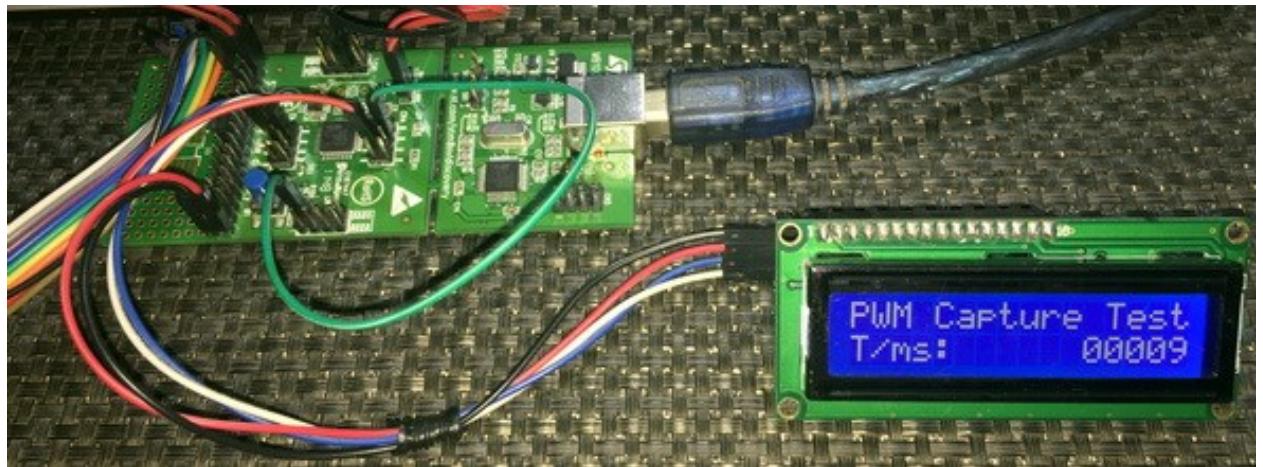
Video link: https://www.youtube.com/watch?v=Gwv3zVbnVvk.

PWM Duty Capture using Software

It is possible to measure pulse widths with only one timer input capture pin and avoid PWMI mode. However, some software tricks must be applied to reliably do the measurements.

Hardware Connection
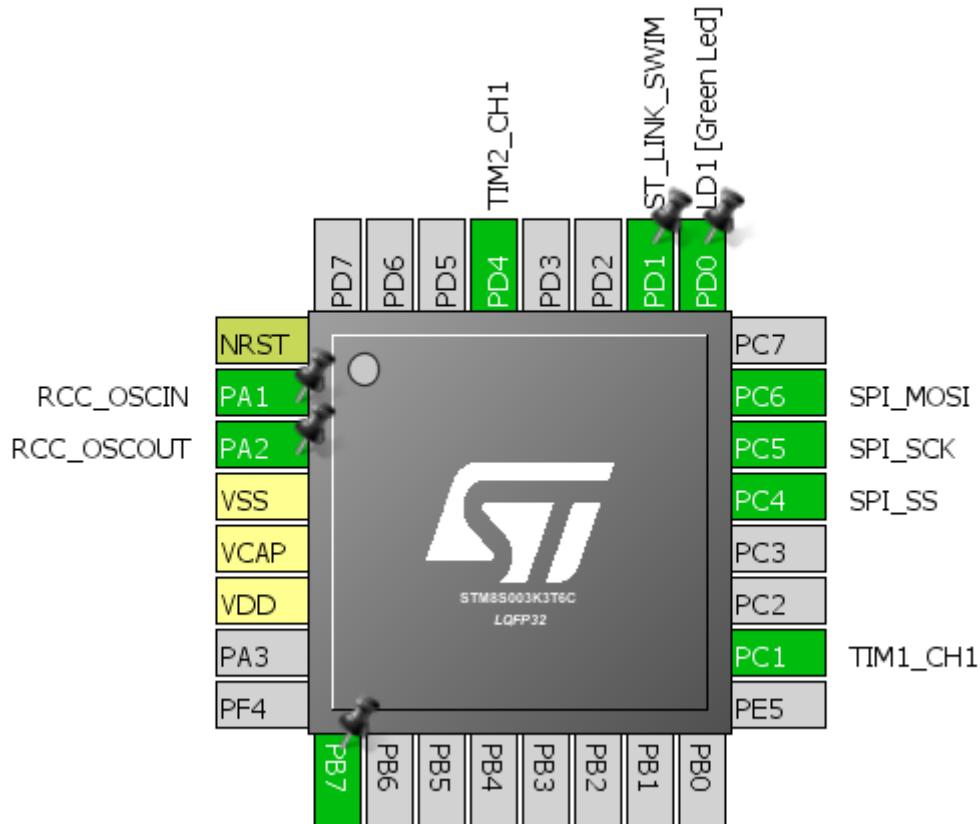
Code Example

stm8s_it.h (top part only)

```
#ifndef __STM8S_IT_H
#define __STM8S_IT_H


@far @interrupt void TIM1_CH1_CCP_IRQHandler(void);


/* Includes ------------------------------------------------------------------*/
#include "stm8s.h"
....
```

stm8s_it.c (top part only)

```c
#include "stm8s.h"
#include "stm8s_it.h"


static bool state = FALSE;

extern unsigned long duty_cycle;
extern unsigned long start_time;
extern unsigned long end_time;


void TIM1_CH1_CCP_IRQHandler(void)
{
    if(TIM1_GetFlagStatus(TIM1_FLAG_CC1))
    {
        if(state == FALSE)
        {
            start_time = TIM1_GetCapture1();
            TIM1_ICInit(TIM1_CHANNEL_1, TIM1_ICPOLARITY_FALLING,
TIM1_ICSELECTION_DIRECTTI, 1, 1);
        }
        else
        {
            end_time = TIM1_GetCapture1();
            TIM1_ICInit(TIM1_CHANNEL_1, TIM1_ICPOLARITY_RISING,
TIM1_ICSELECTION_DIRECTTI, 1, 1);

            duty_cycle = (end_time - start_time);
        }

        state = ~state;
    }

    TIM1_ClearITPendingBit(TIM1_IT_CC1);
    TIM1_ClearFlag(TIM1_FLAG_CC1);
}
....
```

stm8_interrupt_vector.c (shortened)

```c
#include "STM8S.h"
#include "stm8s_it.h"


typedef void @far (*interrupt_handler_t)(void);

struct interrupt_vector {
    unsigned char interrupt_instruction;
    interrupt_handler_t interrupt_handler;
};


extern void _stext();      /* startup routine */

struct interrupt_vector const _vectab[] = {
    {0x82, (interrupt_handler_t)_stext}, /* reset */
```

```
        {0x82, NonHandledInterrupt},  /* trap  */
        {0x82, NonHandledInterrupt},  /* irq0  */
        ....
        {0x82, (interrupt_handler_t)TIM1_CH1_CCP_IRQHandler},  /* irq12 */
        ....
        {0x82, NonHandledInterrupt},  /* irq29 */
};
```

main.c

```c
#include "STM8S.h"
#include "lcd.h"


unsigned char data_value;

unsigned long duty_cycle = 0;
unsigned long start_time = 0;
unsigned long end_time = 0;


void clock_setup(void);
void GPIO_setup(void);
void TIM1_setup(void);
void TIM2_setup(void);
void print_I(unsigned char x_pos, unsigned char y_pos, signed long value);


void main()
{
    unsigned long time_period = 0;

        clock_setup();
        GPIO_setup();
        TIM1_setup();
        TIM2_setup();
        LCD_init();

        LCD_clear_home();
        LCD_goto(0, 0);
        LCD_putstr("PWM Capture Test");
        LCD_goto(0, 1);
        LCD_putstr("T/ms:");
        delay_ms(10);

        while(TRUE)
        {
                print_I(13, 1, duty_cycle);
                delay_ms(900);
        };
}


void clock_setup(void)
{
        CLK_DeInit();

        CLK_HSECmd(DISABLE);
```

```c
        CLK_LSICmd(DISABLE);
        CLK_HSICmd(ENABLE);
        while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

        CLK_ClockSwitchCmd(ENABLE);
        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
        CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

        CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
        DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
        GPIO_DeInit(GPIOC);
        GPIO_Init(GPIOC, GPIO_PIN_1, GPIO_MODE_IN_FL_NO_IT);

        GPIO_DeInit(GPIOD);
        GPIO_Init(GPIOD, GPIO_PIN_4, GPIO_MODE_OUT_PP_HIGH_FAST);
}


void TIM1_setup(void)
{
        TIM1_DeInit();
        TIM1_TimeBaseInit(2000, TIM1_COUNTERMODE_UP, 55535, 1);
        TIM1_CCxCmd(TIM1_CHANNEL_1, ENABLE);
        TIM1_ICInit(TIM1_CHANNEL_1, TIM1_ICPOLARITY_RISING,
TIM1_ICSELECTION_DIRECTTI, 1, 1);
        TIM1_ITConfig(TIM1_IT_CC1, ENABLE);
        TIM1_Cmd(ENABLE);
        enableInterrupts();
}



void TIM2_setup(void)
{
        TIM2_DeInit();
        TIM2_TimeBaseInit(TIM2_PRESCALER_32, 1250);
        TIM2_OC1Init(TIM2_OCMODE_PWM2, TIM2_OUTPUTSTATE_ENABLE, 1000,
TIM2_OCPOLARITY_LOW);
        TIM2_SetCompare1(1125);
        TIM2_Cmd(ENABLE);
}


void print_I(unsigned char x_pos, unsigned char y_pos, signed long value)
{
    unsigned char ch = 0x00;
```

```c
if(value < 0)
{
    LCD_goto(x_pos, y_pos);
        LCD_putchar(0x2D);
    value = -value;
}
else
{
    LCD_goto(x_pos, y_pos);
        LCD_putchar(0x20);
}

if(value > 9999)
{
    ch = ((value / 10000) + 0x30);
    LCD_goto((x_pos + 1), y_pos);
    LCD_putchar(ch);

    ch = (((value % 10000)/ 1000) + 0x30);
    LCD_goto((x_pos + 2), y_pos);
        LCD_putchar(ch);

    ch = (((value % 1000) / 100) + 0x30);
    LCD_goto((x_pos + 3), y_pos);
    LCD_putchar(ch);

    ch = (((value % 100) / 10) + 0x30);
    LCD_goto((x_pos + 4), y_pos);
    LCD_putchar(ch);

    ch = ((value % 10) + 0x30);
    LCD_goto((x_pos + 5), y_pos);
    LCD_putchar(ch);
}

else if((value > 999) && (value <= 9999))
{
    ch = (((value % 10000)/ 1000) + 0x30);
    LCD_goto((x_pos + 1), y_pos);
    LCD_putchar(ch);

    ch = (((value % 1000) / 100) + 0x30);
    LCD_goto((x_pos + 2), y_pos);
    LCD_putchar(ch);

    ch = (((value % 100) / 10) + 0x30);
    LCD_goto((x_pos + 3), y_pos);
    LCD_putchar(ch);

    ch = ((value % 10) + 0x30);
    LCD_goto((x_pos + 4), y_pos);
    LCD_putchar(ch);

    LCD_goto((x_pos + 5), y_pos);
    LCD_putchar(0x20);
}
else if((value > 99) && (value <= 999))
{
```

```c
        ch = (((value % 1000) / 100) + 0x30);
        LCD_goto((x_pos + 1), y_pos);
        LCD_putchar(ch);

        ch = (((value % 100) / 10) + 0x30);
        LCD_goto((x_pos + 2), y_pos);
        LCD_putchar(ch);

        ch = ((value % 10) + 0x30);
        LCD_goto((x_pos + 3), y_pos);
        LCD_putchar(ch);

        LCD_goto((x_pos + 4), y_pos);
        LCD_putchar(0x20);

        LCD_goto((x_pos + 5), y_pos);
        LCD_putchar(0x20);
    }
    else if((value > 9) && (value <= 99))
    {
        ch = (((value % 100) / 10) + 0x30);
        LCD_goto((x_pos + 1), y_pos);
        LCD_putchar(ch);

        ch = ((value % 10) + 0x30);
        LCD_goto((x_pos + 2), y_pos);
        LCD_putchar(ch);

        LCD_goto((x_pos + 3), y_pos);
        LCD_putchar(0x20);

        LCD_goto((x_pos + 4), y_pos);
        LCD_putchar(0x20);

        LCD_goto((x_pos + 5), y_pos);
        LCD_putchar(0x20);
    }
    else
    {
        ch = ((value % 10) + 0x30);
        LCD_goto((x_pos + 1), y_pos);
        LCD_putchar(ch);

        LCD_goto((x_pos + 2), y_pos);
        LCD_putchar(0x20);

        LCD_goto((x_pos + 3), y_pos);
        LCD_putchar(0x20);

        LCD_goto((x_pos + 4), y_pos);
        LCD_putchar(0x20);

        LCD_goto((x_pos + 5), y_pos);
        LCD_putchar(0x20);
    }
}
```

Explanation

Everything here is same as the previous capture example. Since we have to sense both edges for capturing a pulse's width, we will have to do something more in the interrupt routine to take care of this issue.

We assume and select that the first edge that the timer input will capture will be the rising edge of the waveform. Given this fact, the timer is set as follows:

```
TIM1_ICInit(TIM1_CHANNEL_1, TIM1_ICPOLARITY_RISING, TIM1_ICSELECTION_DIRECTTI, 1, 1);
```

So, when the capture interrupt is triggered, we know for sure that it was caused by the rising edge of the pulse captured.

In the input capture interrupt, the very first thing to do is to check the capture-compare flag and then we check last setup state of the timer. Since rising edge capture was initially configured, the first capture count is actually the start time of the pulse. Upon detecting this we must quickly reconfigure the timer capture hardware to sense falling edge.

Now the timer's capture interrupt will be triggered on the falling edge and we have to measure the timer's count on this next interrupt. This second count is the end time of the pulse. The difference between these counts represents the pulse width.

```c
void TIM1_CH1_CCP_IRQHandler(void)
{
    if(TIM1_GetFlagStatus(TIM1_FLAG_CC1))
    {
        if(state == FALSE)
        {
            start_time = TIM1_GetCapture1();
            TIM1_ICInit(TIM1_CHANNEL_1, TIM1_ICPOLARITY_FALLING,
TIM1_ICSELECTION_DIRECTTI, 1, 1);
        }
        else
        {
            end_time = TIM1_GetCapture1();
            TIM1_ICInit(TIM1_CHANNEL_1, TIM1_ICPOLARITY_RISING,
TIM1_ICSELECTION_DIRECTTI, 1, 1);

            duty_cycle = (end_time - start_time);
        }

        state = ~state;
    }

    TIM1_ClearITPendingBit(TIM1_IT_CC1);
    TIM1_ClearFlag(TIM1_FLAG_CC1);
}
```

Every time a TIM1 capture-compare interrupt occurs, the timer's sense edge is altered along with a variable called "state". This variable helps in remembering which edge it detected last time and which count was captured. Finally, before leaving the interrupt sub-routine pending and flag bits are reset.

Demo

Video link: https://www.youtube.com/watch?v=DvXU7NkdZII.

Communication Overview

STM8 microcontrollers are packed with several communication interfaces. These interfaces are needed to communicate with external devices like sensors, actuators, drives, etc. The most commonly used ones are Serial Communication (*UART*), Serial Peripheral Interface (*SPI*) and Inter-Integrated Circuit (*I2C*). There are also other additional more robust communication interfaces like Controller Area Network (*CAN*), Local Interconnect Network (*LIN*), Infrared Data Association (*IrDA*) and *RS-485*. The latter communications will not be discussed here in this article and are kept for future issues. These are methods are, however, not frequently used and are special forms of communications. For example, CAN and LIN are mostly used in automotive industries. Each method communication has its own advantages and disadvantages. Here, we will see the individual basics of various methods of communications.

Comm.	Description	I/O	Max. Speed	Max. Distance	Max. Possible Number of Devices in a Bus
UART	Asynchronous serial point-to-point communication	2	115.2kbps	15m	2 (Point-to-Point)
SPI	Short-range synchronous master-slave serial communication	3/4	4Mbps	0.1m	Multiple
I2C	Synchronous master-slave serial communication using one data and one clock line	2	1Mbps	0.5m	127
RS-485	Asynchronous single master differential 2 wire serial communication	2	115.2kbps	1.2km	Multiple
CAN	Differential communication with multi-master support	2	1Mbps	5km	Multiple
LIN	Asynchronous 2 wire serial communication similar to UART	2	20kbps	40m	Multiple
IrDA	Wireless serial communication using infrared medium	2	115.2kbps	<1m	2 (Point-to-Point)

In STM8 microcontrollers, LIN, IrDA, RS-485 and UART all share the UART hardware peripheral. For other communications, there are dedicated separate hardware. We will now be exploring the basic ones here.

Serial Communication (UART)

Serial communication is perhaps the mostly-used classic communication method for interfacing a PC or other machines with a micro. With just two wires, we can achieve a full-duplex point-to-point communication. Owing to its simplicity and wide usage, it is the communication interface backbone that is used with GSM modems, RF modules, Bluetooth devices like RN-52, Wi-Fi devices like the popular ESP8266, etc. It is also widely used in industries. Other communications rely on it, for example, RS-485, LIN, etc.
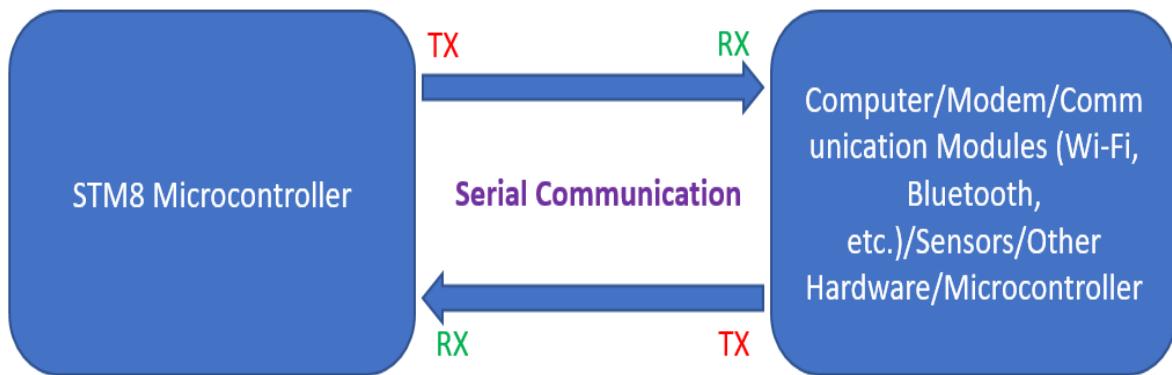
Most STM8s have at least one UART module. Some have more than one. Different UARTs have different features as shown:

UART configurations

Feature	UART1	UART2	UART3	UART4
Asynchronous mode	X	X	X	X
Multiprocessor communication	X	X	X	X
Synchronous communication	X	X	NA	X
Smartcard mode	X	X	NA	X
IrDA mode	X	X	NA	X
Single-wire Half-duplex mode	X	NA	NA	X
LIN master mode	X	X	X	X
LIN slave mode	NA	X	X	X

X = supported; NA = not applicable.

To learn more about UART visit the following link:

https://learn.mikroe.com/uart-serial-communication

The UARTs of STM8 micros are so robust and packed with so many features that it is quite impossible to explain them all in this one article. Here we will explore the basic serial communication only. LIN and IRDA will hopefully be covered in future articles.

Hardware Connection
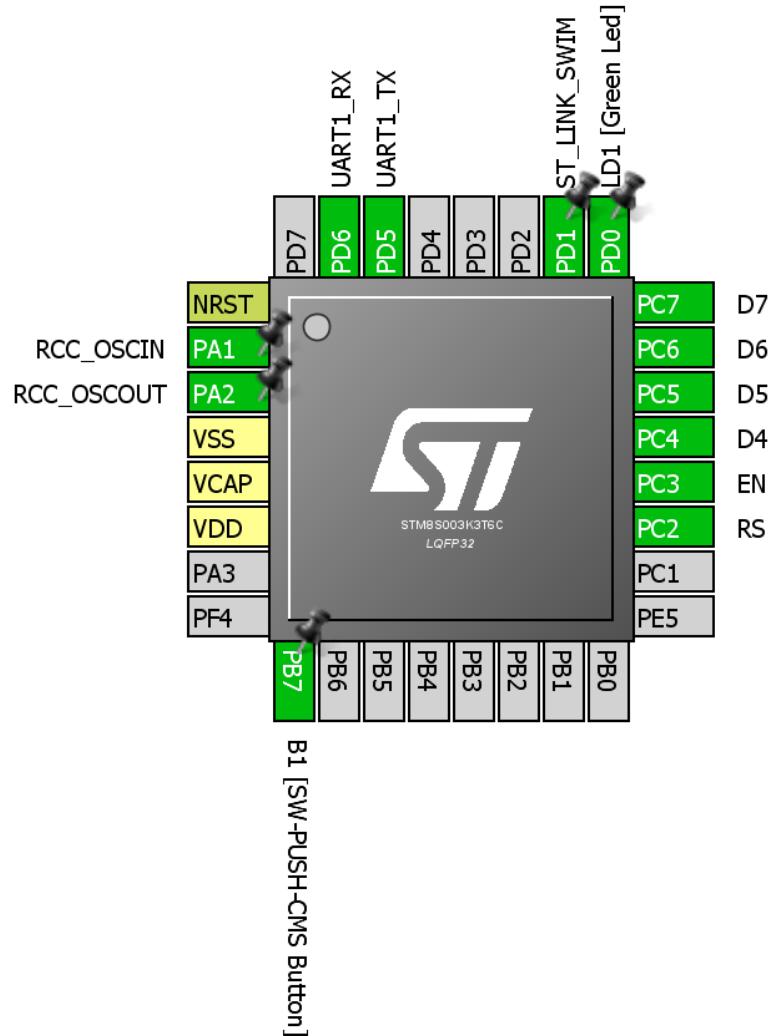
Code Example

```c
#include "STM8S.h"


void clock_setup(void);
void GPIO_setup(void);
void UART1_setup(void);


void main(void)
{
    unsigned char i = 0;
    char ch = 0;

    clock_setup();
```

```c
        GPIO_setup();
        UART1_setup();
        LCD_init();
        LCD_clear_home();

        LCD_goto(0, 0);
        LCD_putstr("TX:");
        LCD_goto(0, 1);
        LCD_putstr("RX:");

        while(TRUE)
        {
                if(UART1_GetFlagStatus(UART1_FLAG_RXNE) == TRUE)
                {
                        ch = UART1_ReceiveData8();
                        LCD_goto(7, 1);
                        LCD_putchar(ch);
                        UART1_ClearFlag(UART1_FLAG_RXNE);
                        UART1_SendData8(i + 0x30);
                }
                if(UART1_GetFlagStatus(UART1_FLAG_TXE) == FALSE)
                {
                        LCD_goto(7, 0);
                        LCD_putchar(i + 0x30);
                        i++;
                }
        };
}


void clock_setup(void)
{
        CLK_DeInit();

        CLK_HSECmd(DISABLE);
        CLK_LSICmd(DISABLE);
        CLK_HSICmd(ENABLE);
        while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

        CLK_ClockSwitchCmd(ENABLE);
        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
        CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

        CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
        DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
```

```
        GPIO_DeInit(GPIOD);

        GPIO_Init(GPIOD, GPIO_PIN_5, GPIO_MODE_OUT_PP_HIGH_FAST);
        GPIO_Init(GPIOD, GPIO_PIN_6, GPIO_MODE_IN_PU_NO_IT);
}


void UART1_setup(void)
{
        UART1_DeInit();

        UART1_Init(9600,
                    UART1_WORDLENGTH_8D,
                    UART1_STOPBITS_1,
                    UART1_PARITY_NO,
                    UART1_SYNCMODE_CLOCK_DISABLE,
                    UART1_MODE_TXRX_ENABLE);

        UART1_Cmd(ENABLE);
}
```

Explanation

The peripheral and CPU clocks are set at 2MHz:

```
CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);
....
CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, ENABLE);
```

The TX-RX GPIO pins are set as output and input respectively:

```
void GPIO_setup(void)
{
        GPIO_DeInit(GPIOD);

        GPIO_Init(GPIOD, GPIO_PIN_5, GPIO_MODE_OUT_PP_HIGH_FAST);
        GPIO_Init(GPIOD, GPIO_PIN_6, GPIO_MODE_IN_PU_NO_IT);
}
```

UART setup is straightforward. We just need to set baud rate, no. of data bits, no. of stop bit, parity and type of communication (synchronous or asynchronous).

```
void UART1_setup(void)
{
        UART1_DeInit();

        UART1_Init(9600,
                    UART1_WORDLENGTH_8D,
                    UART1_STOPBITS_1,
                    UART1_PARITY_NO,
                    UART1_SYNCMODE_CLOCK_DISABLE,
                    UART1_MODE_TXRX_ENABLE);
```

```
        UART1_Cmd(ENABLE);
}
```

In the main code, we are checking both transmission complete and reception complete flags. With these flags, we will know if new data arrived and if it is possible to send a new data.

The first part checks if any new data received. That's why the *IF* condition is checking if the RX buffer is empty or not. If it is not empty then new data must have arrived. The new data (a character here) is fetched and displayed on a LCD. Then we clear the RX buffer not empty flag to enable reception of new data. After that we are sending some data to the host PC over the UART.

```
if(UART1_GetFlagStatus(UART1_FLAG_RXNE) == TRUE)
{
        ch = UART1_ReceiveData8();
        LCD_goto(7, 1);
        LCD_putchar(ch);
        UART1_ClearFlag(UART1_FLAG_RXNE);
        UART1_SendData8(i + 0x30);
}
```

In the second part, we are checking if the last data was sent from our STM8 micro. The data sent is then displayed on LCD.

```
if(UART1_GetFlagStatus(UART1_FLAG_TXE) == FALSE)
{
        LCD_goto(7, 0);
        LCD_putchar(i + 0x30);
        i++;
}
```

Please note that both of these flags are very important.

Demo

Video link: https://youtu.be/uo2tYDUnMmE.
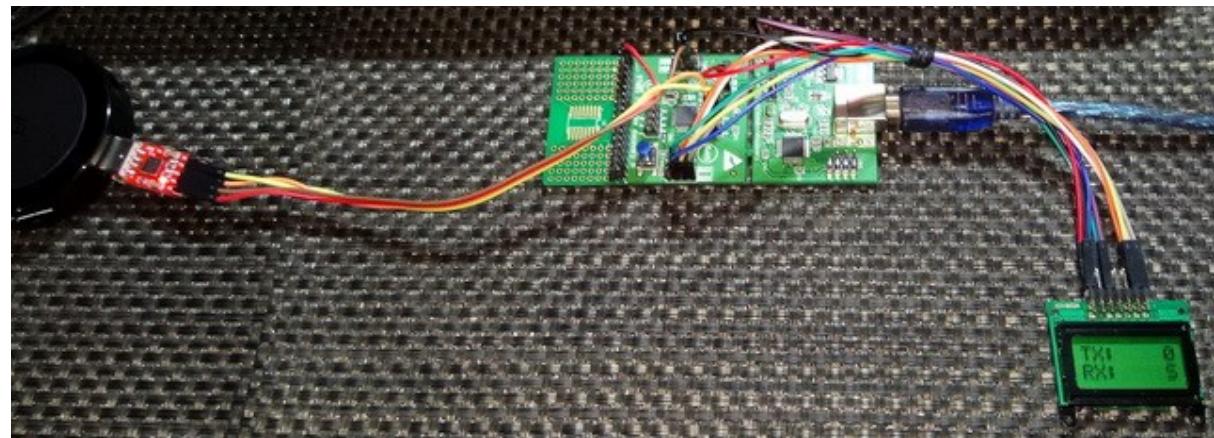
UART Interrupt

We have seen that we can use UART without interrupt methods. However, in most cases, UART interrupts are needed to make a serial communication real-time. In particular UART reception interrupt is of great importance. For example, when it is needed to interface a serial GPS receiver with a host micro, we will need UART reception interrupt because the GPS receiver will be continuously sending GPS data and it will be unwise to poll UART. If reception interrupt is used, the host micro will then be continually receiving GPS data in the background while doing other tasks in the main foreground.

Hardware Connection
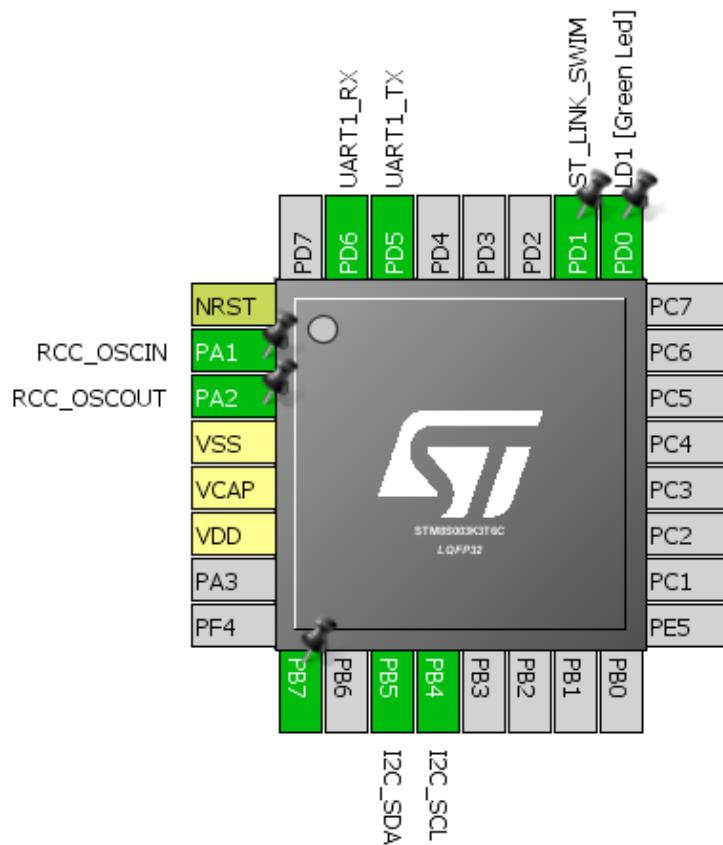
Code Example

stm8s_it.h (top part only)

```
#ifndef __STM8S_IT_H
#define __STM8S_IT_H

@far @interrupt void UART_RX_IRQHandler(void);

/* Includes ------------------------------------------------------------------*/
#include "stm8s.h"
....
```

stm8s_it.c (top part only)

```c
#include "stm8s.h"
#include "stm8s_it.h"


extern unsigned char pos;
extern unsigned char RX_value[16];


void UART_RX_IRQHandler(void)
{
      RX_value[pos] = UART1_ReceiveData8();
      UART1_ClearITPendingBit(UART1_IT_RXNE);
      UART1_ClearFlag(UART1_FLAG_RXNE);

      UART1_SendData8(RX_value[pos]);
      UART1_ClearFlag(UART1_FLAG_TXE);

      pos++;

      if(pos > 15)
      {
            pos = 0;
      }
}
....
```

stm8_interrupt_vector.c (shortened)

```c
#include "stm8s_it.h"


typedef void @far (*interrupt_handler_t)(void);

struct interrupt_vector {
      unsigned char interrupt_instruction;
      interrupt_handler_t interrupt_handler;
};


extern void _stext();      /* startup routine */

struct interrupt_vector const _vectab[] = {
      {0x82, (interrupt_handler_t)_stext}, /* reset */
      {0x82, NonHandledInterrupt}, /* trap  */
      {0x82, NonHandledInterrupt}, /* irq0  */
      ....
      {0x82, (interrupt_handler_t)UART_RX_IRQHandler}, /* irq18 */
      ....
      {0x82, NonHandledInterrupt}, /* irq29 */
};
```

main.c

```c
#include "STM8S.h"
#include "lcd.h"


unsigned char pos;
unsigned char bl_state;
unsigned char RX_value[16];
unsigned char data_value;


void clock_setup(void);
void GPIO_setup(void);
void UART1_setup(void);
void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned int value);


void main(void)
{
    clock_setup();
    GPIO_setup();
    UART1_setup();

    LCD_init();
    LCD_clear_home();

    LCD_goto(1, 0);
    LCD_putstr("STM8S UART ISR");

    while(TRUE)
    {
        LCD_goto(0, 1);
        LCD_putstr(RX_value);
        delay_ms(100);
    };
}


void clock_setup(void)
{
    CLK_DeInit();

    CLK_HSECmd(DISABLE);
    CLK_LSICmd(DISABLE);
    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

    CLK_ClockSwitchCmd(ENABLE);
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV4);
    CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

    CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
    DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

    CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, ENABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, ENABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
```

```c
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
        GPIO_DeInit(GPIOD);
        GPIO_Init(GPIOD, GPIO_PIN_0, GPIO_MODE_OUT_OD_HIZ_FAST);
        GPIO_Init(GPIOD, GPIO_PIN_5, GPIO_MODE_OUT_PP_HIGH_FAST);
        GPIO_Init(GPIOD, GPIO_PIN_6, GPIO_MODE_IN_FL_NO_IT);
}


void UART1_setup(void)
{
        UART1_DeInit();

        UART1_Init(9600,
                    UART1_WORDLENGTH_8D,
                    UART1_STOPBITS_1,
                    UART1_PARITY_NO,
                    UART1_SYNCMODE_CLOCK_DISABLE,
                    UART1_MODE_TXRX_ENABLE);

        UART1_ITConfig(UART1_IT_RXNE, ENABLE);
        enableInterrupts();

        UART1_Cmd(ENABLE);
}


void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned int value)
{
        char chr = 0x00;

        chr = ((value / 1000) + 0x30);
        LCD_goto(x_pos, y_pos);
        LCD_putchar(chr);

        chr = (((value / 100) % 10) + 0x30);
        LCD_goto((x_pos + 1), y_pos);
        LCD_putchar(chr);

        chr = (((value / 10) % 10) + 0x30);
        LCD_goto((x_pos + 2), y_pos);
        LCD_putchar(chr);

        chr = ((value % 10) + 0x30);
        LCD_goto((x_pos + 3), y_pos);
        LCD_putchar(chr);
}
```

Explanation

The UART setup is same as before. The exception is the interrupt part:

```
UART1_ITConfig(UART1_IT_RXNE, ENABLE);
enableInterrupts();
```

These lines enable UART reception interrupt and global interrupt.

Just as with other interrupts, UART reception interrupt is enabled in the STM8S interrupt vector map file.

```
{0x82, (interrupt_handler_t)UART_RX_IRQHandler}, /* irq18 */
```

When a valid UART data is received by the UART RX pin, an UART reception interrupt is triggered. UART buffer is read. UART reception pending bit and reception flag are both cleared to pave way for further receptions. The data received is stored in an array for displaying it later and is immediately transmitted back through UART TX pin.

In the STM8 interrupt source file the following lines of code are added to do all these tasks:

```
void UART_RX_IRQHandler(void)
{
      RX_value[pos] = UART1_ReceiveData8();
      UART1_ClearITPendingBit(UART1_IT_RXNE);
      UART1_ClearFlag(UART1_FLAG_RXNE);

      UART1_SendData8(RX_value[pos]);
      UART1_ClearFlag(UART1_FLAG_TXE);

      pos++;

      if(pos > 15)
      {
            pos = 0;
      }
}
```

Demo
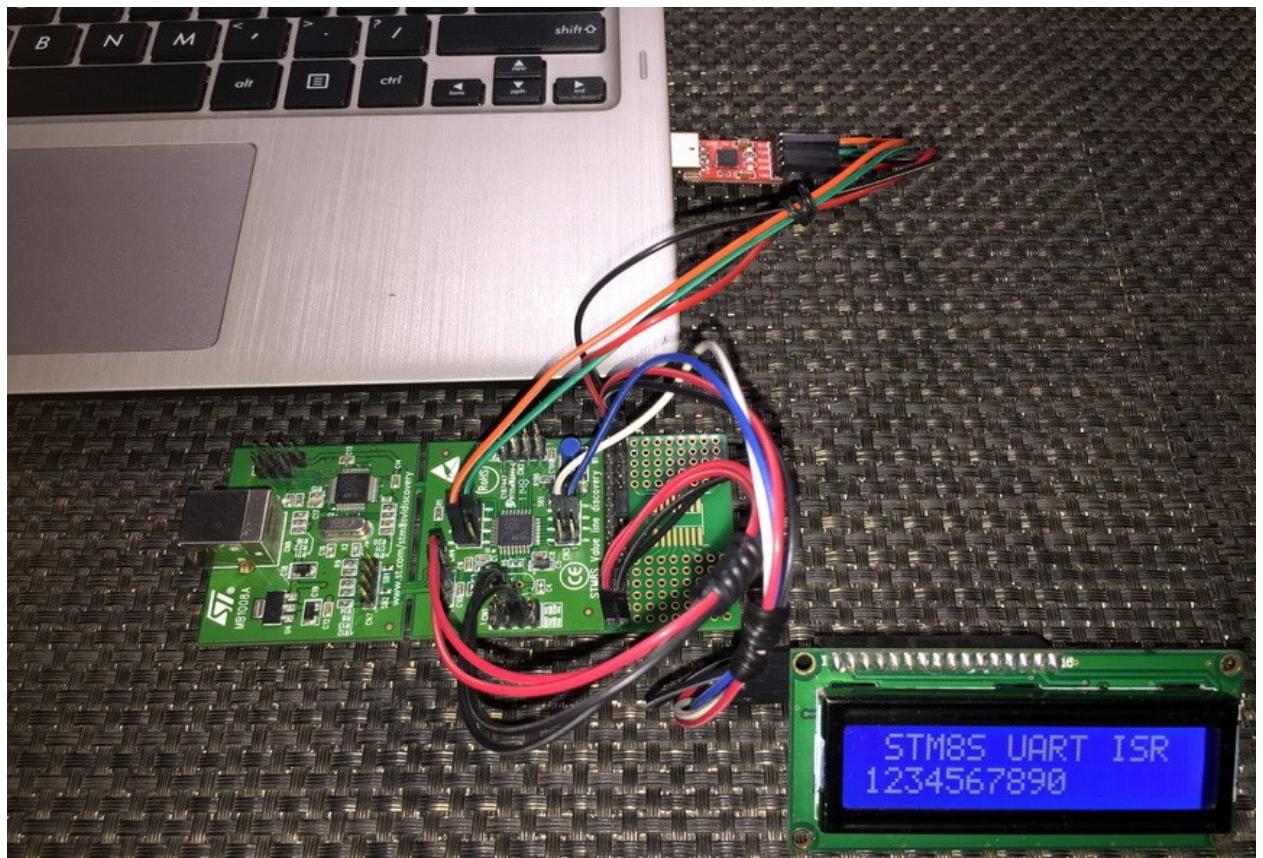
Video link: https://www.youtube.com/watch?v=4fE5-aza_48.

Serial Peripheral Interface (SPI)

SPI communication is a popular onboard synchronous communication method and is used by a number of devices including various sensors, TFT and OLED displays, GPIO expanders, PWM controller ICs, memory chips, addon support devices, etc.

There is always one master device in a SPI communication bus which generates clock and select slave(s). Master sends commands to slave(s). Slave(s) responds to commands sent by the master. The number of slaves in a SPI bus is virtually unlimited. Except the chip selection pin, all SPI devices in a bus can share the same clock and data pins.

Typical full-duplex SPI bus requires four basic I/O pins. The red ones are for SPI master while the green ones are for slave(s). Their naming suggests their individual purpose.

- *Master-Out-Slave-In (MOSI)* connected to *Slave-Data-In (SDI)*.
- *Master-In-Slave-Out (MIS0)* connected to *Slave-Data-Out (SDO)*.
- *Serial Clock (SCLK)* connected to *Slave Clock (SCK)*.
- *Slave Select (SS)* connected to *Chip Select (CS)*.
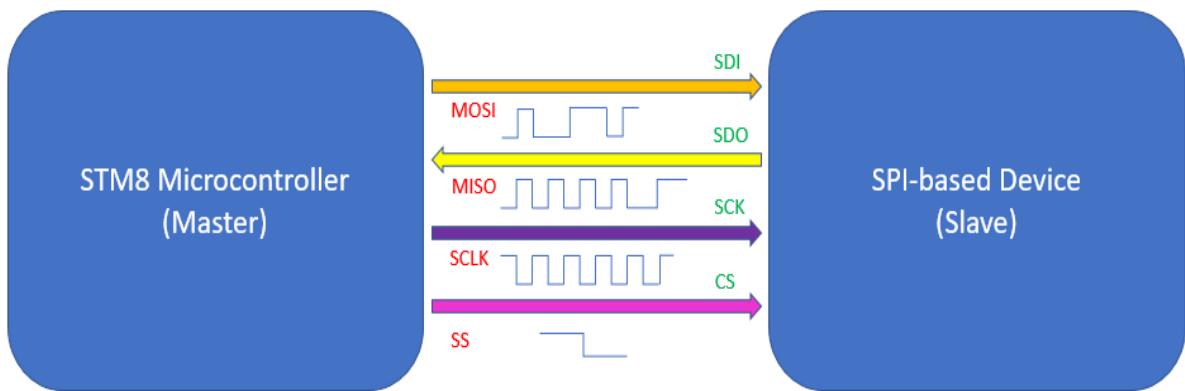
In general, if you wish to know more about SPI bus here are some cool links:

- https://learn.mikroe.com/spi-bus

- https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi

- http://ww1.microchip.com/downloads/en/devicedoc/spi.pdf

- http://tronixstuff.com/2011/05/13/tutorial-arduino-and-the-spi-bus

- https://embeddedmicro.com/tutorials/mojo/serial-peripheral-interface-spi

- http://www.circuitbasics.com/basics-of-the-spi-communication-protocol

STM8s have SPI hardware that are more capable than the SPI hardware found in other micros. An additional feature of STM8's SPI is the hardware *Cyclic Redundancy Check (CRC)*. This feature ensures reliable data communication between devices.

Hardware Connection
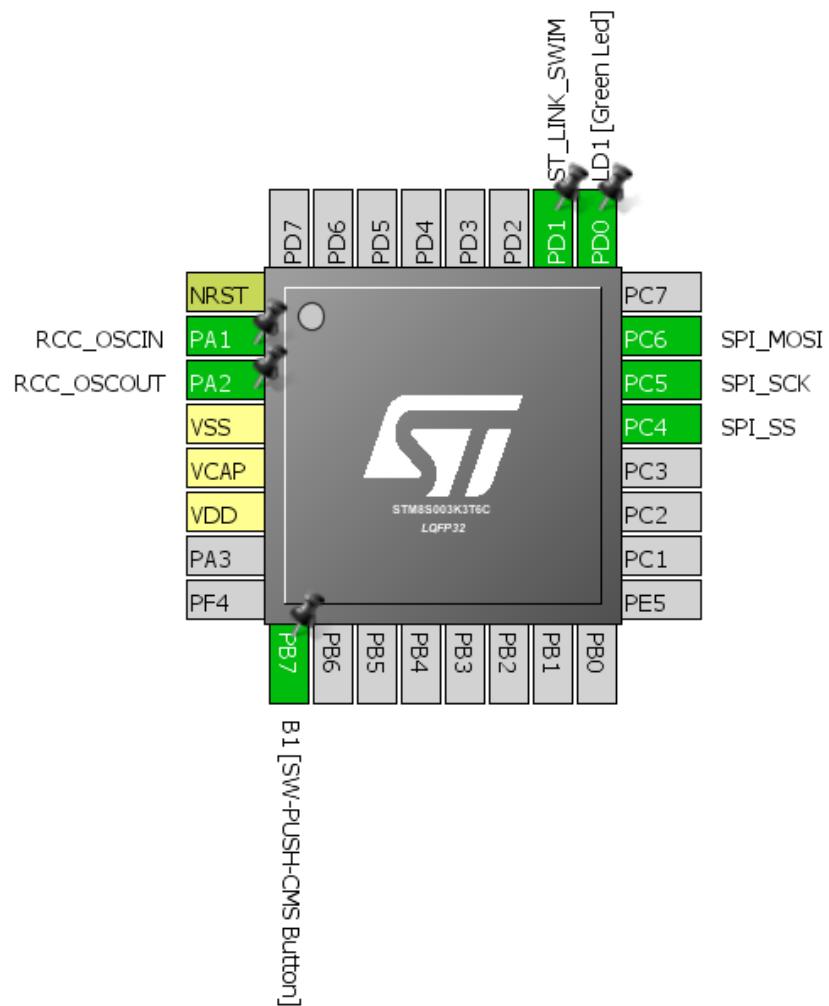
Code Example

main.c

```c
#include "STM8S.h"
#include "MAX72XX.h"


void clock_setup(void);
void GPIO_setup(void);
void SPI_setup(void);


void main(void)
{
    unsigned char i = 0x00;
    unsigned char j = 0x00;

    volatile unsigned char temp[8] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00};
```

```c
    const unsigned char text[96] =
    {
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x7E, 0x04, 0x08, 0x08, 0x04, 0x7E, 0x00,         //M
        0x00, 0x42, 0x42, 0x7E, 0x7E, 0x42, 0x42, 0x00,         //I
        0x00, 0x3C, 0x42, 0x42, 0x42, 0x42, 0x24, 0x00,         //C
        0x00, 0x7E, 0x1A, 0x1A, 0x1A, 0x2A, 0x44, 0x00,         //R
        0x00, 0x3C, 0x42, 0x42, 0x42, 0x42, 0x3C, 0x00,         //O
        0x00, 0x7C, 0x12, 0x12, 0x12, 0x12, 0x7C, 0x00,         //A
        0x00, 0x7E, 0x1A, 0x1A, 0x1A, 0x2A, 0x44, 0x00,         //R
        0x00, 0x7E, 0x7E, 0x4A, 0x4A, 0x4A, 0x42, 0x00,         //E
        0x00, 0x7E, 0x04, 0x08, 0x10, 0x20, 0x7E, 0x00,         //N
        0x00, 0x7C, 0x12, 0x12, 0x12, 0x12, 0x7C, 0x00,         //A
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
    };


    clock_setup();
    GPIO_setup();
    SPI_setup();
    MAX72xx_init();

    while(TRUE)
    {
        for(i = 0; i < sizeof(temp); i++)
        {
            temp[i] = 0x00;
        }

        for(i = 0; i < sizeof(text); i++)
        {
            for(j = 0; j < sizeof(temp); j++)
            {
                temp[j] = text[(i + j)];
                MAX72xx_write((1 + j), temp[j]);
                delay_ms(9);
            }
        }
    };
}


void clock_setup(void)
{
    CLK_DeInit();

    CLK_HSECmd(DISABLE);
    CLK_LSICmd(DISABLE);
    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

    CLK_ClockSwitchCmd(ENABLE);
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
    CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

    CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
    DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

    CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, ENABLE);
```

```c
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
        GPIO_DeInit(GPIOC);
        GPIO_Init(GPIOC, ((GPIO_Pin_TypeDef)GPIO_PIN_5 | GPIO_PIN_6),
GPIO_MODE_OUT_PP_HIGH_FAST);
}


void SPI_setup(void)
{
        SPI_DeInit();
        SPI_Init(SPI_FIRSTBIT_MSB,
                SPI_BAUDRATEPRESCALER_2,
                SPI_MODE_MASTER,
                SPI_CLOCKPOLARITY_HIGH,
                SPI_CLOCKPHASE_1EDGE,
                SPI_DATADIRECTION_1LINE_TX,
                SPI_NSS_SOFT,
               0x00);
        SPI_Cmd(ENABLE);
}
```

MAX72xx.h

```c
#include "STM8S.h"


#define CS_pin                              GPIO_PIN_4
#define CS_port                             GPIOC

#define NOP                                 0x00
#define DIG0                                0x01
#define DIG1                                0x02
#define DIG2                                0x03
#define DIG3                                0x04
#define DIG4                                0x05
#define DIG5                                0x06
#define DIG6                                0x07
#define DIG7                                0x08

#define decode_mode_reg                     0x09
#define intensity_reg                       0x0A
#define scan_limit_reg                      0x0B
#define shutdown_reg                        0x0C
#define display_test_reg                    0x0F

#define shutdown_cmd                        0x00
#define run_cmd                             0x01
```

```c
#define no_test_cmd                                      0x00
#define test_cmd                                         0x01
#define digit_0_only                                     0x00
#define digit_0_to_1                                     0x01
#define digit_0_to_2                                     0x02
#define digit_0_to_3                                     0x03
#define digit_0_to_4                                     0x04
#define digit_0_to_5                                     0x05
#define digit_0_to_6                                     0x06
#define digit_0_to_7                                     0x07

#define No_decode_for_all                                0x00
#define Code_B_decode_digit_0                            0x01
#define Code_B_decode_digit_0_to_3                       0x0F
#define Code_B_decode_for_all                            0xFF


void MAX72xx_init(void);
void MAX72xx_write(unsigned char address, unsigned char value);
```

MAX72xx.c

```c
#include "MAX72xx.h"


void MAX72xx_init(void)
{
    GPIO_Init(CS_port, CS_pin, GPIO_MODE_OUT_PP_HIGH_FAST);

    MAX72xx_write(shutdown_reg, run_cmd);
    MAX72xx_write(decode_mode_reg, 0x00);
    MAX72xx_write(scan_limit_reg, 0x07);
    MAX72xx_write(intensity_reg, 0x04);
    MAX72xx_write(display_test_reg, test_cmd);
    delay_ms(10);
    MAX72xx_write(display_test_reg, no_test_cmd);
}


void MAX72xx_write(unsigned char address, unsigned char value)
{
    while(SPI_GetFlagStatus(SPI_FLAG_BSY));
    GPIO_WriteLow(CS_port, CS_pin);

    SPI_SendData(address);
    while(!SPI_GetFlagStatus(SPI_FLAG_TXE));

    SPI_SendData(value);
    while(!SPI_GetFlagStatus(SPI_FLAG_TXE));

    GPIO_WriteHigh(CS_port, CS_pin);
}
```

Explanation

This time we are again using the max peripheral and CPU clock:

```c
CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);
....
CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, ENABLE);
```

We also need to set the GPIOs:

```c
#define CS_pin          GPIO_PIN_4
#define CS_port         GPIOC
....
....
GPIO_DeInit(GPIOC);
GPIO_Init(CS_port, CS_pin, GPIO_MODE_OUT_PP_HIGH_FAST);
GPIO_Init(GPIOC, ((GPIO_Pin_TypeDef)GPIO_PIN_5 | GPIO_PIN_6),
GPIO_MODE_OUT_PP_HIGH_FAST);
```

Note we can use definitions to make things meaningful. The GPIOs should be configured as fast I/Os because SPI communication is faster than simple GPIO operations.

Now for the SPI configuration part. Assuming you know how to interpret timing diagrams and understand device datasheets, SPI configuration should not be a problem. Here in the case of MAX7219, we have configured the SPI port as to send MSB first, we have also selected a high speed peripheral clock, we have made the STM8 SPI act like a master with proper SPI mode and we have set the sort of duplex. The last two parameters are not important as we are not using hardware slave select option and CRC feature.

```c
void SPI_setup(void)
{
    SPI_DeInit();
    SPI_Init(SPI_FIRSTBIT_MSB,
            SPI_BAUDRATEPRESCALER_2,
            SPI_MODE_MASTER,
            SPI_CLOCKPOLARITY_HIGH,
            SPI_CLOCKPHASE_1EDGE,
            SPI_DATADIRECTION_1LINE_TX,
            SPI_NSS_SOFT,
            0x00);
    SPI_Cmd(ENABLE);
}
```

The timing diagram of MAX7219 suggests that CS should be low in order for MAX7219 to receive data.
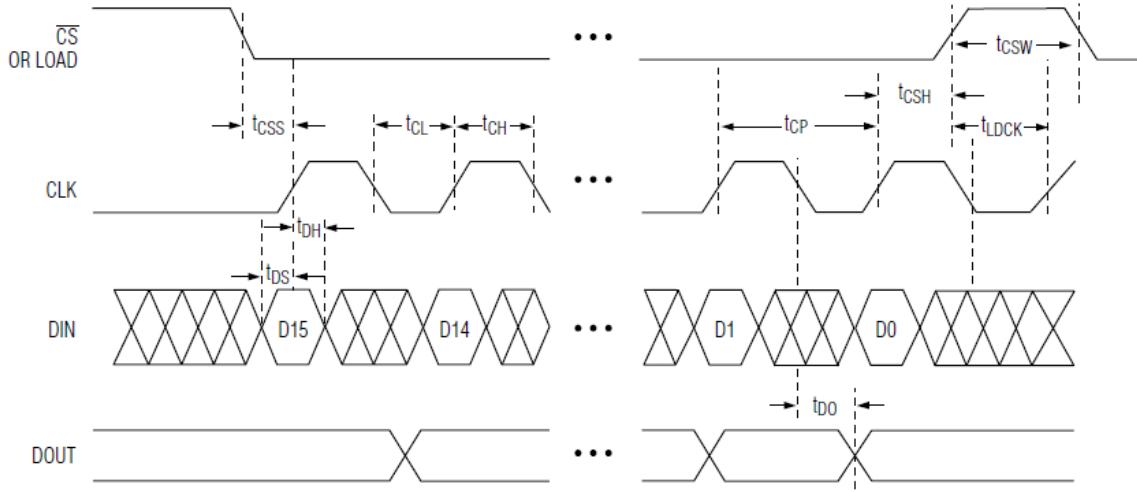
Then it suggests that when idle, clock must be high, data transfer is done on every rising edge of the clock. All these are what required for setting up the SPI hardware.

```c
void MAX72xx_write(unsigned char address, unsigned char value)
{
    while(SPI_GetFlagStatus(SPI_FLAG_BSY));
    GPIO_WriteLow(CS_port, CS_pin);

    SPI_SendData(address);
    while(!SPI_GetFlagStatus(SPI_FLAG_TXE));

    SPI_SendData(value);
    while(!SPI_GetFlagStatus(SPI_FLAG_TXE));

    GPIO_WriteHigh(CS_port, CS_pin);
}
```

Before sending data to MAX7219, we must check if the SPI hardware is busy for some reason. We set CS low by setting STM8's Slave Select (SS) pin (PC4) low. Then we send address and data. Every time we send something we must wait until it has completely been sent out. Finally, we set CS high to latch sent data. This function is what we will need to set MAX7219 things up and also to update displays.

The demo here is that of a MAX7219-based scrolling dot-matrix display. Letters of **MICROARENA** – the name of my Facebook page are scrolled one after another.

Demo

Video link: https://youtu.be/O7mre-bzsGE.
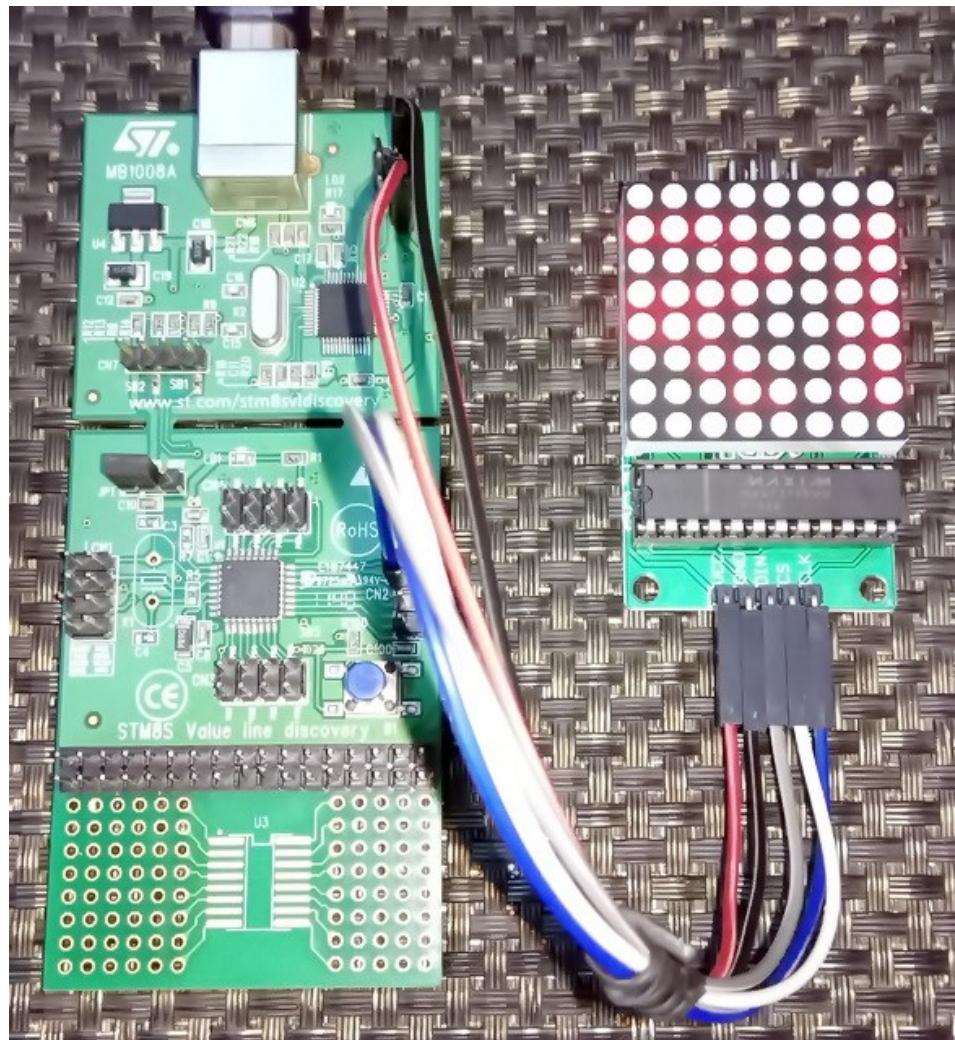
Software SPI – MAX6675

Software SPI is basically bit-banging ordinary GPIOs to emulate SPI signals. Software SPI is rarely needed unless hardware SPI peripheral is unavailable or hardware SPI pins are used up for some other tasks. Though it is slow and require extra coding and therefore extra overhead compared to hardware SPI, it is sometimes very helpful in debugging SPI-based hardware as it offers full control over all SPI signals.

To demo software SPI here, I have used MAX6675 Cold-Junction-Compensated K-Thermocouple-to-Digital Converter. It is interfaced with a typical K-type thermocouple and sends out the temperature sensed by the thermocouple.

Hardware Connection
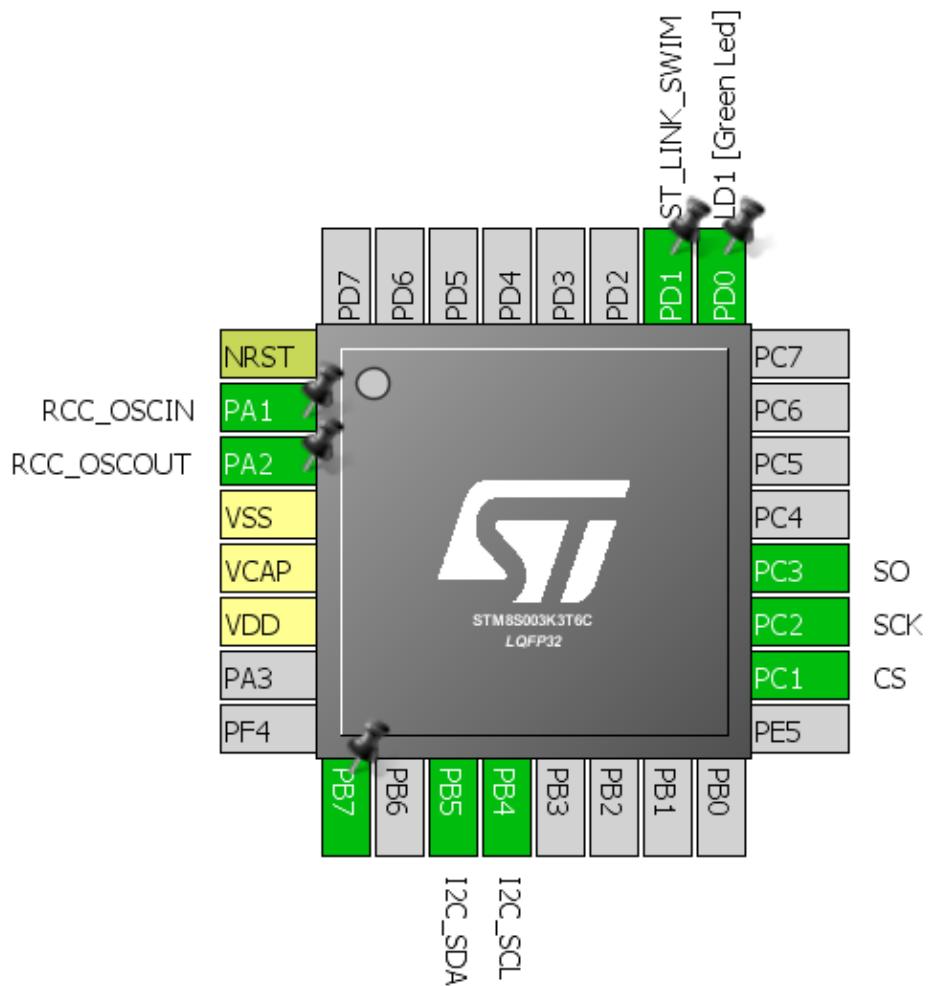
Code Example

MAX6675.h

```c
#include "STM8S.h"


#define SW_SPI_Port      GPIOC

#define CS_pin           GPIO_PIN_1
#define SCK_pin          GPIO_PIN_2
#define SO_pin           GPIO_PIN_3

#define CS_OUT_HIGH()    GPIO_WriteHigh(SW_SPI_Port, CS_pin)
#define CS_OUT_LOW()     GPIO_WriteLow(SW_SPI_Port, CS_pin)
#define SCK_OUT_HIGH()   GPIO_WriteHigh(SW_SPI_Port, SCK_pin)
#define SCK_OUT_LOW()    GPIO_WriteLow(SW_SPI_Port, SCK_pin)

#define SO_IN()          GPIO_ReadInputPin(SW_SPI_Port, SO_pin)

#define T_min            0
#define T_max            1024

#define count_max        4096

#define no_of_pulses     16

#define deg_C            0
#define deg_F            1
#define tmp_K            2

#define open_contact     0x04
#define close_contact    0x00

#define scalar_deg_C     0.25
#define scalar_deg_F_1   1.8
#define scalar_deg_F_2   32.0
#define scalar_tmp_K     273.0

#define no_of_samples    16


void MAX6675_init(void);
unsigned char MAX6657_get_ADC(unsigned int *ADC_data);
float MAX6675_get_T(unsigned int ADC_value, unsigned char T_unit);
```

MAX6675.c

```c
#include "MAX6675.h"


void MAX6675_init()
{
   GPIO_DeInit(SW_SPI_Port);

   GPIO_Init(SW_SPI_Port,
             ((GPIO_Pin_TypeDef)(SCK_pin | CS_pin)),
```

```c
            GPIO_MODE_OUT_PP_HIGH_FAST);

    GPIO_Init(SW_SPI_Port, SO_pin, GPIO_MODE_IN_FL_NO_IT);

    CS_OUT_HIGH();
    SCK_OUT_HIGH();
}


unsigned char MAX6657_get_ADC(unsigned int *ADC_data)
{
    unsigned char samples = no_of_samples;
    unsigned char clk_pulses = 0;
    unsigned int temp_data = 0;
    unsigned long avg_value = 0;


    while(samples > 0)
    {
        clk_pulses = no_of_pulses;
        temp_data = 0;

        CS_OUT_LOW();

        while(clk_pulses > 0)
        {
            temp_data <<= 1;

            if(SO_IN())
            {
                temp_data |= 1;
            }

            SCK_OUT_HIGH();
            SCK_OUT_LOW();

            clk_pulses--;
        };

        CS_OUT_HIGH();
        temp_data &= 0x7FFF;

        avg_value += temp_data;

        samples--;
        delay_ms(10);
    };

    temp_data = (avg_value >> 4);

    if((temp_data & 0x04) == close_contact)
    {
        *ADC_data = (temp_data >> 3);
        return close_contact;
    }
    else
    {
        *ADC_data = (count_max + 1);
        return open_contact;
```

```c
        }
}


float MAX6675_get_T(unsigned int ADC_value, unsigned char T_unit)
{
    float tmp = 0.0;

    tmp = (((float)ADC_value) * scalar_deg_C);

    switch(T_unit)
    {
        case deg_F:
        {
            tmp *= scalar_deg_F_1;
            tmp += scalar_deg_F_2;
            break;
        }
        case tmp_K:
        {
            tmp += scalar_tmp_K;
            break;
        }
        default:
        {
            break;
        }
    }

    return tmp;
}
```

main.c

```c
#include "STM8S.h"
#include "MAX6675.h"
#include "lcd.h"


unsigned char bl_state;
unsigned char data_value;

const unsigned char symbol[0x08] =
{
    0x00, 0x06, 0x09, 0x09, 0x06, 0x00, 0x00, 0x00
};


void clock_setup(void);
void lcd_symbol(void);
void print_C(unsigned char x_pos, unsigned char y_pos, signed int value);
void print_I(unsigned char x_pos, unsigned char y_pos, signed long value);
void print_D(unsigned char x_pos, unsigned char y_pos, signed int value, unsigned
char points);
void print_F(unsigned char x_pos, unsigned char y_pos, float value, unsigned char
points);
```

```c
void main(void)
{
        unsigned char state = close_contact;
        unsigned int ti = 0x0000;
        float tf = 0.0;

        clock_setup();

        MAX6675_init();

        LCD_init();
        LCD_clear_home();
        lcd_symbol();

        LCD_goto(0, 0);
        LCD_putstr("STM8 SW-SPI Test");

        while(TRUE)
        {
                state = MAX6657_get_ADC(&ti);

                switch(state)
                {
                    case open_contact:
                    {
                            LCD_goto(0, 1);
                            LCD_putstr("     Error!      ");
                            delay_ms(200);
                            LCD_goto(0, 1);
                            LCD_putstr("                ");
                            break;
                    }
                    case close_contact:
                    {
                            tf = MAX6675_get_T(ti, deg_C);

                            LCD_goto(0, 1);
                            LCD_putstr("T/            ");

                            LCD_goto(2, 1);
                            LCD_send(0, DAT);
                            LCD_goto(3, 1);
                            LCD_putstr("C:");

                            print_F(9, 1, tf, 2);
                            delay_ms(100);

                            break;
                    }
                }
        };
}


void clock_setup(void)
{
        CLK_DeInit();

        CLK_HSECmd(DISABLE);
```

```c
        CLK_LSICmd(DISABLE);
        CLK_HSICmd(ENABLE);
        while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

        CLK_ClockSwitchCmd(ENABLE);
        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);

        CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

        CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO,
                              CLK_SOURCE_HSI,
                              DISABLE,
                              CLK_CURRENTCLOCKSTATE_ENABLE);

        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void lcd_symbol(void)
{
    unsigned char s = 0;

    LCD_send(0x40, CMD);

    for(s = 0; s < 8; s++)
    {
        LCD_send(symbol[s], DAT);
    }

    LCD_send(0x80, CMD);
}


void print_C(unsigned char x_pos, unsigned char y_pos, signed int value)
{
    unsigned char ch[5] = {0x20, 0x20, 0x20, 0x20, '\0'};

    if(value < 0x00)
    {
        ch[0] = 0x2D;
        value = -value;
    }
    else
    {
        ch[0] = 0x20;
    }

    if((value > 99) && (value <= 999))
    {
        ch[1] = ((value / 100) + 0x30);
        ch[2] = (((value % 100) / 10) + 0x30);
        ch[3] = ((value % 10) + 0x30);
```

```c
    }
    else if((value > 9) && (value <= 99))
    {
        ch[1] = (((value % 100) / 10) + 0x30);
        ch[2] = ((value % 10) + 0x30);
        ch[3] = 0x20;
    }
    else if((value >= 0) && (value <= 9))
    {
        ch[1] = ((value % 10) + 0x30);
        ch[2] = 0x20;
        ch[3] = 0x20;
    }

    LCD_goto(x_pos, y_pos);
    LCD_putstr(ch);
}


void print_I(unsigned char x_pos, unsigned char y_pos, signed long value)
{
    unsigned char ch[7] = {0x20, 0x20, 0x20, 0x20, 0x20, 0x20, '\0'};

    if(value < 0)
    {
        ch[0] = 0x2D;
        value = -value;
    }
    else
    {
        ch[0] = 0x20;
    }

    if(value > 9999)
    {
        ch[1] = ((value / 10000) + 0x30);
        ch[2] = (((value % 10000)/ 1000) + 0x30);
        ch[3] = (((value % 1000) / 100) + 0x30);
        ch[4] = (((value % 100) / 10) + 0x30);
        ch[5] = ((value % 10) + 0x30);
    }

    else if((value > 999) && (value <= 9999))
    {
        ch[1] = (((value % 10000)/ 1000) + 0x30);
        ch[2] = (((value % 1000) / 100) + 0x30);
        ch[3] = (((value % 100) / 10) + 0x30);
        ch[4] = ((value % 10) + 0x30);
        ch[5] = 0x20;
    }
    else if((value > 99) && (value <= 999))
    {
        ch[1] = (((value % 1000) / 100) + 0x30);
        ch[2] = (((value % 100) / 10) + 0x30);
        ch[3] = ((value % 10) + 0x30);
        ch[4] = 0x20;
        ch[5] = 0x20;
    }
    else if((value > 9) && (value <= 99))
```

```c
    {
        ch[1] = (((value % 100) / 10) + 0x30);
        ch[2] = ((value % 10) + 0x30);
        ch[3] = 0x20;
        ch[4] = 0x20;
        ch[5] = 0x20;
    }
    else
    {
        ch[1] = ((value % 10) + 0x30);
        ch[2] = 0x20;
        ch[3] = 0x20;
        ch[4] = 0x20;
        ch[5] = 0x20;
    }

    LCD_goto(x_pos, y_pos);
    LCD_putstr(ch);
}


void print_D(unsigned char x_pos, unsigned char y_pos, signed int value, unsigned char points)
{
    char ch[5] = {0x2E, 0x20, 0x20, '\0'};

    ch[1] = ((value / 100) + 0x30);

    if(points > 1)
    {
        ch[2] = (((value / 10) % 10) + 0x30);

            if(points > 1)
            {
                ch[3] = ((value % 10) + 0x30);
            }
    }

    LCD_goto(x_pos, y_pos);
    LCD_putstr(ch);
}


void print_F(unsigned char x_pos, unsigned char y_pos, float value, unsigned char points)
{
    signed long tmp = 0x0000;

    tmp = value;
    print_I(x_pos, y_pos, tmp);
    tmp = ((value - tmp) * 1000);

    if(tmp < 0)
    {
        tmp = -tmp;
    }

    if(value < 0)
    {
```

```c
            value = -value;
                LCD_goto(x_pos, y_pos);
            LCD_putchar(0x2D);
        }
        else
        {
            LCD_goto(x_pos, y_pos);
            LCD_putchar(0x20);
        }

        if((value >= 10000) && (value < 100000))
        {
            print_D((x_pos + 6), y_pos, tmp, points);
        }
        else if((value >= 1000) && (value < 10000))
        {
            print_D((x_pos + 5), y_pos, tmp, points);
        }
        else if((value >= 100) && (value < 1000))
        {
            print_D((x_pos + 4), y_pos, tmp, points);
        }
        else if((value >= 10) && (value < 100))
        {
            print_D((x_pos + 3), y_pos, tmp, points);
        }
        else if(value < 10)
        {
            print_D((x_pos + 2), y_pos, tmp, points);
        }
}
```

Explanation

A decent implementation of software SPI first requires I/Os and their purposes to be defined at first:

```c
#define SW_SPI_Port      GPIOC

#define CS_pin           GPIO_PIN_1
#define SCK_pin          GPIO_PIN_2
#define SO_pin           GPIO_PIN_3

#define CS_OUT_HIGH()    GPIO_WriteHigh(SW_SPI_Port, CS_pin)
#define CS_OUT_LOW()     GPIO_WriteLow(SW_SPI_Port, CS_pin)
#define SCK_OUT_HIGH()   GPIO_WriteHigh(SW_SPI_Port, SCK_pin)
#define SCK_OUT_LOW()    GPIO_WriteLow(SW_SPI_Port, SCK_pin)

#define SO_IN()          GPIO_ReadInputPin(SW_SPI_Port, SO_pin)
```

Trust me defining stuffs this way saves both time and helps in debugging.
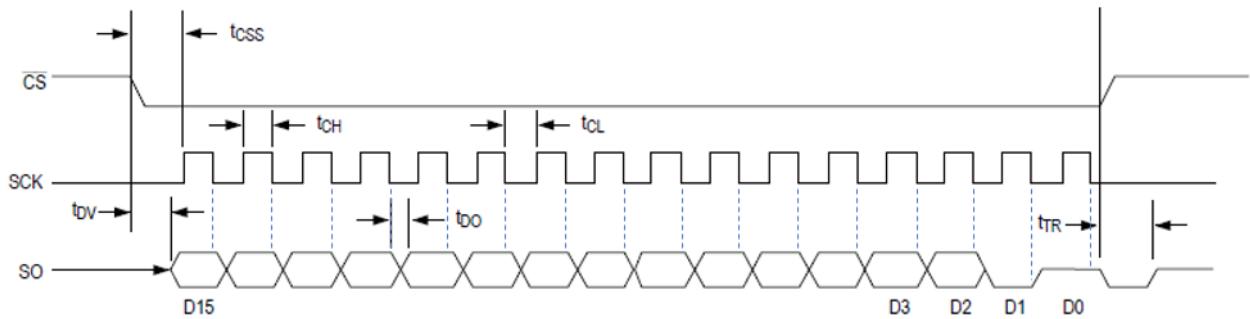
We must code signal patterns as per timing diagram in the device's datasheet (shown above). The datasheet of MAX6675 states that sensed data should be read on falling edges of the serial clock with chip select pin held low. This is what we have to code:

```c
CS_OUT_LOW();

while(clk_pulses > 0)
{
    temp_data <<= 1;

    if(SO_IN())
    {
        temp_data |= 1;
    }

    SCK_OUT_HIGH();
    SCK_OUT_LOW();

    clk_pulses--;
};

CS_OUT_HIGH();
```

Note that **Chip Select (CS)** is held low to begin communication. **Serial Clock (SCK)** is held high first and the set low to emulate a falling edge of the serial clock. In total sixteen such transitions are needed to extract all data from MAX6675 via serial input pin.

The rest of the code is data processing and conversion, and then finally data display. I'm sure you'll understand.

Video link: https://www.youtube.com/watch?v=yD-1a70MfV8.
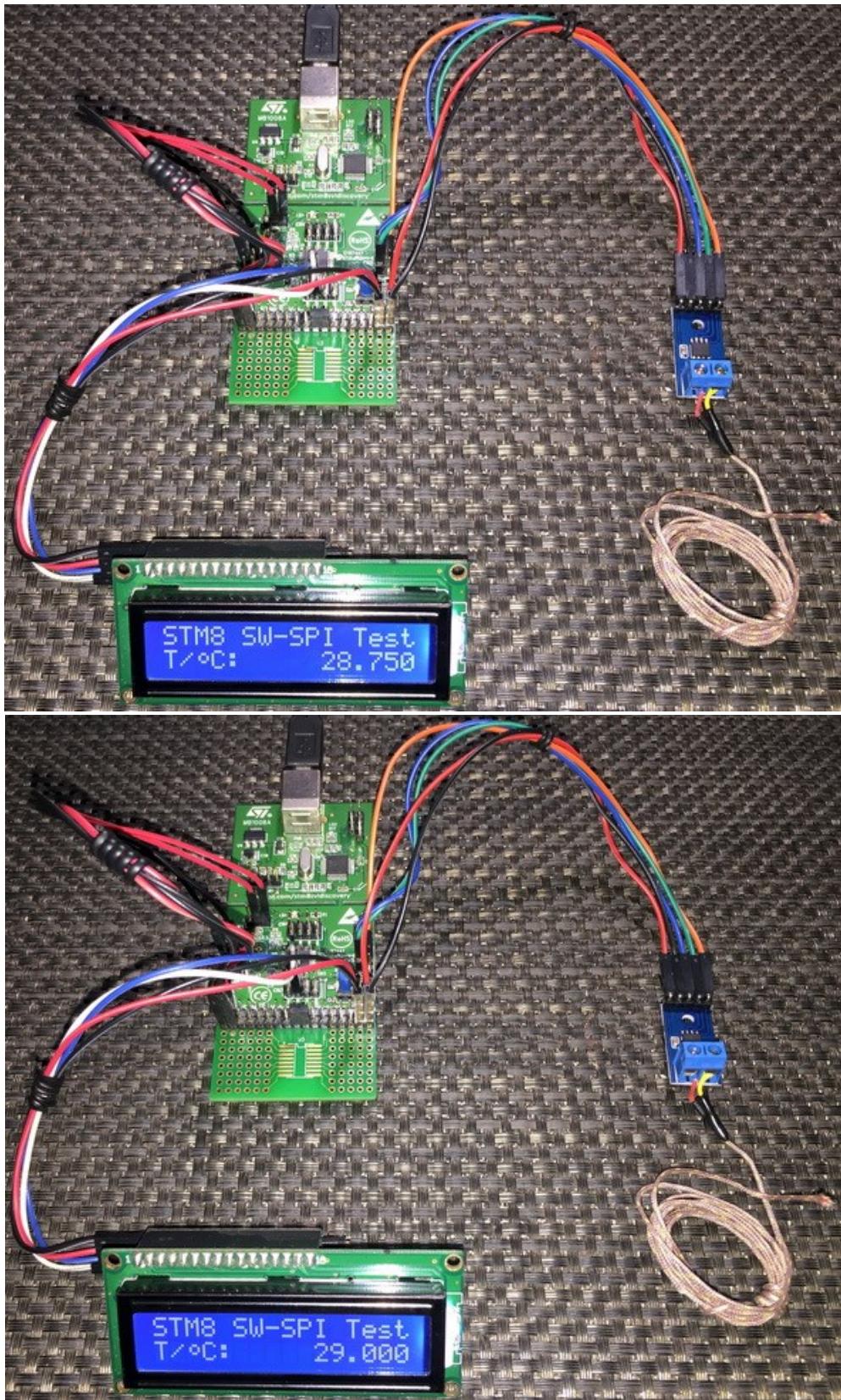
Driving LCD with Hardware SPI and by Bit-Banging Method

Bit-banging and SPI-based communications are very popular in the embedded system world. These methods are not limited to communication with sensors, drives, etc. They can be used for driving an Alphanumeric LCD with a few pins. Usually with shift registers like 74HC595 and CD4094B, we can make such display interfaces very simply. It is also possible to do so with SPI port expanders like MCP23S17 but those will be expensive solutions. All that we will need is to create a way to translate LCD I/O operations to SPI signals.
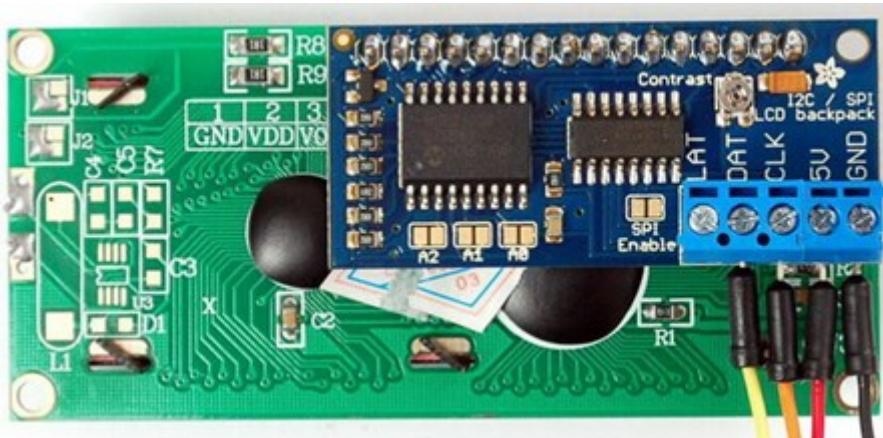
The main advantages of using SPI-based/bit-banging-based LCDs include immunity from noise and EMI, simplicity, reduction of GPIO usage and addressability. However, the downsides are slower refresh rates and requirement of additional coding and memory spaces. Even with these pluses and minuses, driving a LCD with such arrangements is often a requirement, especially when it comes to low pin count micros and fast debugging.

Hardware Connection

Code Example

lcd.h
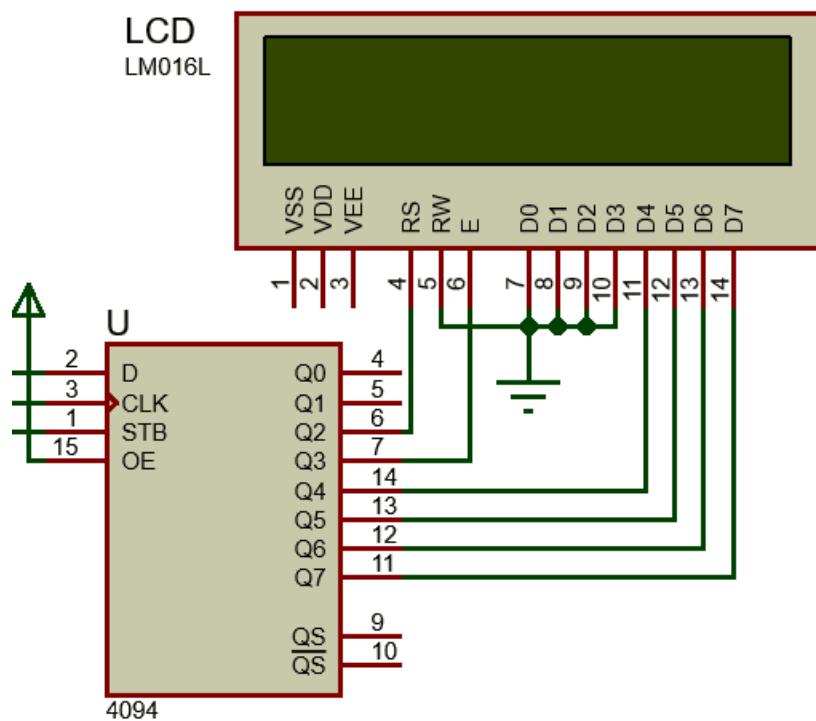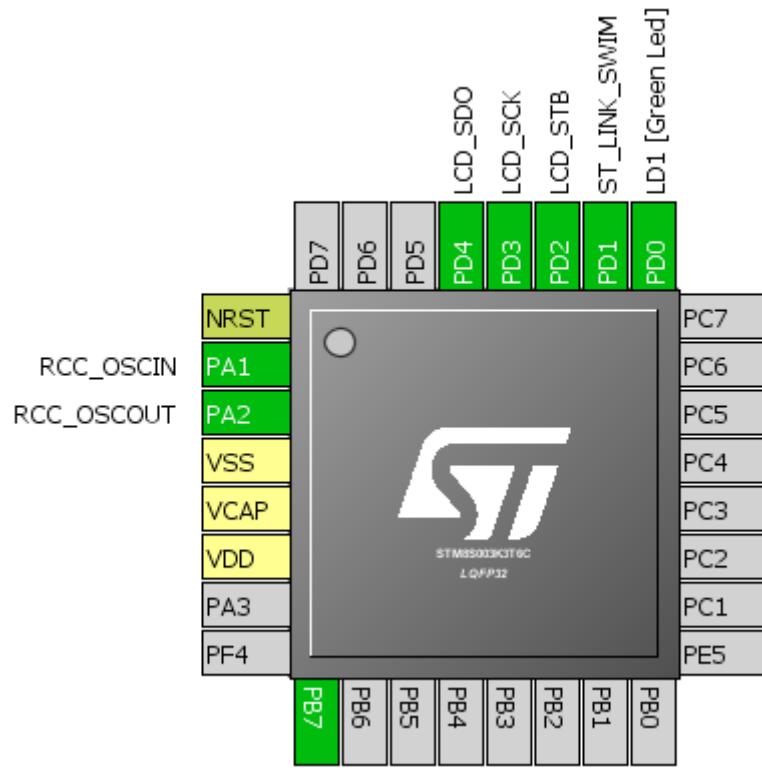
```c
#include "stm8s.h"


#define LCD_PORT                        GPIOD

#define LCD_STB_pin                     GPIO_PIN_2
#define LCD_SCK_pin                     GPIO_PIN_3
#define LCD_SDI_pin                     GPIO_PIN_4

#define LCD_STB_HIGH()                  GPIO_WriteHigh(LCD_PORT, LCD_STB_pin)
#define LCD_STB_LOW()                   GPIO_WriteLow(LCD_PORT, LCD_STB_pin)
#define LCD_SCK_HIGH()                  GPIO_WriteHigh(LCD_PORT, LCD_SCK_pin)
#define LCD_SCK_LOW()                   GPIO_WriteLow(LCD_PORT, LCD_SCK_pin)
#define LCD_SDI_HIGH()                  GPIO_WriteHigh(LCD_PORT, LCD_SDI_pin)
#define LCD_SDI_LOW()                   GPIO_WriteLow(LCD_PORT, LCD_SDI_pin)


#define clear_display                   0x01
#define goto_home                       0x02

#define cursor_direction_inc            (0x04 | 0x02)
#define cursor_direction_dec            (0x04 | 0x00)
#define display_shift                   (0x04 | 0x01)
#define display_no_shift                (0x04 | 0x00)

#define display_on                      (0x08 | 0x04)
#define display_off                     (0x08 | 0x02)
#define cursor_on                       (0x08 | 0x02)
#define cursor_off                      (0x08 | 0x00)
#define blink_on                        (0x08 | 0x01)
#define blink_off                       (0x08 | 0x00)

#define _8_pin_interface                (0x20 | 0x10)
#define _4_pin_interface                (0x20 | 0x00)
#define _2_row_display                  (0x20 | 0x08)
#define _1_row_display                  (0x20 | 0x00)
#define _5x10_dots                      (0x20 | 0x40)
#define _5x7_dots                       (0x20 | 0x00)

#define dly                             2

#define DAT                             1
#define CMD                             0


extern unsigned char data_value;


void SIPO(void);
void LCD_init(void);
void LCD_toggle_EN(void);
void LCD_send(unsigned char value, unsigned char mode);
void LCD_4bit_send(unsigned char lcd_data);
void LCD_putstr(char *lcd_string);
```

```c
void LCD_putchar(char char_data);
void LCD_clear_home(void);
void LCD_goto(unsigned char x_pos, unsigned char y_pos);
```

lcd.c

```c
#include "lcd.h"


void SIPO(void)
{
    unsigned char bit = 0x00;
    unsigned char clk = 0x08;
    unsigned char temp = 0x00;

    temp = data_value;
    LCD_STB_LOW();

    while(clk > 0)
    {
        bit = ((temp & 0x80) >> 0x07);
        bit &= 0x01;

        switch(bit)
        {
            case 0:
            {
                LCD_SDI_LOW();
                break;
            }
            default:
            {
                LCD_SDI_HIGH();
                break;
            }
        }

        LCD_SCK_HIGH();

        temp <<= 0x01;
        clk--;

        LCD_SCK_LOW();
    };

    LCD_STB_HIGH();
}


void LCD_init(void)
{
    unsigned char t = 0x00;

    GPIO_Init(LCD_PORT,
            ((GPIO_Pin_TypeDef)(LCD_STB_pin | LCD_SCK_pin | LCD_SDI_pin)),
            GPIO_MODE_OUT_PP_HIGH_FAST);

    data_value = 0x08;
```

```c
    SIPO();
    delay_ms(10);

    LCD_send(0x33, CMD);
    LCD_send(0x32, CMD);

    LCD_send((_4_pin_interface | _2_row_display | _5x7_dots), CMD);
    LCD_send((display_on | cursor_off | blink_off), CMD);
    LCD_send((clear_display), CMD);
    LCD_send((cursor_direction_inc | display_no_shift), CMD);
}


void LCD_toggle_EN(void)
{
    data_value |= 0x08;
    SIPO();
    delay_ms(dly);
    data_value &= 0xF7;
    SIPO();
    delay_ms(dly);
}


void LCD_send(unsigned char value, unsigned char mode)
{
    switch(mode)
    {
        case DAT:
        {
            data_value |= 0x04;
            break;
        }
        default:
        {
            data_value &= 0xFB;
            break;
        }
    }

    SIPO();
    LCD_4bit_send(value);
}


void LCD_4bit_send(unsigned char lcd_data)
{
    unsigned char temp = 0x00;

    temp = (lcd_data & 0xF0);
    data_value &= 0x0F;
    data_value |= temp;
    SIPO();
    LCD_toggle_EN();

    temp = (lcd_data & 0x0F);
    temp <<= 0x04;
    data_value &= 0x0F;
    data_value |= temp;
```

```c
    SIPO();
    LCD_toggle_EN();
}


void LCD_putstr(char *lcd_string)
{
    while(*lcd_string != '\0')
    {
        LCD_putchar(*lcd_string++);
    }
}


void LCD_putchar(char char_data)
{
    if((char_data >= 0x20) && (char_data <= 0x7F))
    {
        LCD_send(char_data, DAT);
    }
}


void LCD_clear_home(void)
{
    LCD_send(clear_display, CMD);
    LCD_send(goto_home, CMD);
}


void LCD_goto(unsigned char x_pos,unsigned char y_pos)
{
    if(y_pos == 0)
    {
        LCD_send((0x80 | x_pos), CMD);
    }
    else
    {
        LCD_send((0x80 | 0x40 | x_pos), CMD);
    }
}
```

main.c

```c
#include "STM8S.h"
#include "lcd.h"


unsigned char data_value;


void clock_setup(void);
void GPIO_setup(void);
void show_value(unsigned char value);


void main(void)
{
```

```c
    const char txt1[] = {"MICROARENA"};
    const char txt2[] = {"SShahryiar"};
    const char txt3[] = {"STM8S003K3"};
    const char txt4[] = {"Discovery!"};

    unsigned char s = 0x00;

    clock_setup();
    GPIO_setup();
    LCD_init();

    LCD_clear_home();

    LCD_goto(3, 0);
    LCD_putstr(txt1);
    LCD_goto(3, 1);
    LCD_putstr(txt2);
    delay_ms(2600);

    LCD_clear_home();

    for(s = 0; s < 10; s++)
    {
        LCD_goto((3 + s), 0);
        LCD_putchar(txt3[s]);
        delay_ms(60);
    }
    for(s = 0; s < 10; s++)
    {
        LCD_goto((3 + s), 1);
        LCD_putchar(txt4[s]);
        delay_ms(60);
    }
        delay_ms(2600);

    s = 0;
    LCD_clear_home();

    LCD_goto(3, 0);
    LCD_putstr(txt1);

    while(1)
    {
        show_value(s);
        s++;
        delay_ms(200);
    };
}


void clock_setup(void)
{
    CLK_DeInit();
    CLK_HSECmd(DISABLE);
    CLK_LSICmd(DISABLE);

    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);
```

```c
    CLK_ClockSwitchCmd(ENABLE);
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
    CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

    CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
                          DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

    CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
    GPIO_DeInit(LCD_PORT);
}


void show_value(unsigned char value)
{
    char ch = 0x00;

    ch = ((value / 100) + 0x30);
    LCD_goto(6, 1);
    LCD_putchar(ch);

    ch = (((value / 10) % 10) + 0x30);
    LCD_goto(7, 1);
    LCD_putchar(ch);

    ch = ((value % 10) + 0x30);
    LCD_goto(8, 1);
    LCD_putchar(ch);
}
```

Explanation

The code demoed here is same as the other LCD codes in this article so there is not much to explain. The I/O operations of the LCD are handled using a CD4094B *Serial-In-Parallel-Out (SIPO)* shift register. This shift register here acts like an output expander. With just three STM8 GPIOs we are able to interface a 4-bit LCD that needs at least six GPIOs to work.

CD4094B can be operated by either bit-banging GPIOs, i.e. using software SPI or using hardware SPI. If we are to use software-based SPI then we need to code the entire SPI operation as shown below:

```c
void SIPO(void)
{
    unsigned char bit = 0x00;
    unsigned char clk = 0x08;
    unsigned char temp = 0x00;

    temp = data_value;
    LCD_STB_LOW();

    while(clk > 0)
    {
        bit = ((temp & 0x80) >> 0x07);
        bit &= 0x01;

        switch(bit)
        {
            case 0:
            {
                LCD_SDI_LOW();
                break;
            }
            default:
            {
                LCD_SDI_HIGH();
                break;
            }
        }

        LCD_SCK_HIGH();

        temp <<= 0x01;
        clk--;

        LCD_SCK_LOW();
    };

    LCD_STB_HIGH();
}
```

This function does the work of SPI clock generation, chip selection and data bit shifting.

If hardware SPI is used, we have to initialize SPI hardware, use hardware SPI pins and use SPI write technique to send data to shift register. The process of clock generation and data shifting is done inside the SPI hardware and so we have nothing else to do.

```c
void SPI_setup(void)
{
    SPI_DeInit();

    SPI_Init(SPI_FIRSTBIT_MSB,
            SPI_BAUDRATEPRESCALER_64,
            SPI_MODE_MASTER,
            SPI_CLOCKPOLARITY_LOW,
            SPI_CLOCKPHASE_1EDGE,
            SPI_DATADIRECTION_1LINE_TX,
            SPI_NSS_SOFT,
            0x00);

    SPI_Cmd(ENABLE);
}


void LCD_write(void)
{
    while(SPI_GetFlagStatus(SPI_FLAG_BSY));
    GPIO_WriteHigh(GPIOC, GPIO_PIN_4);
    SPI_SendData(data_value);
    GPIO_WriteLow(GPIOC, GPIO_PIN_4);
    while(!SPI_GetFlagStatus(SPI_FLAG_TXE));
}
```

Just as with any hardware peripheral, we have to enable SPI peripheral clock before we use it.

```c
CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, ENABLE);
```

The rest of the code is just about translating and transferring GPIO patterns to the shift register. Remember everything is same the only this is the fact that the shift register here works as a GPIO expander. The code here works as per hardware connections shown. In the market, there is no ready-made SPI LCD module like the one I used here. It is my own designed. If different shift register or different hardware layout is used then you must modify the code accordingly.

Demo
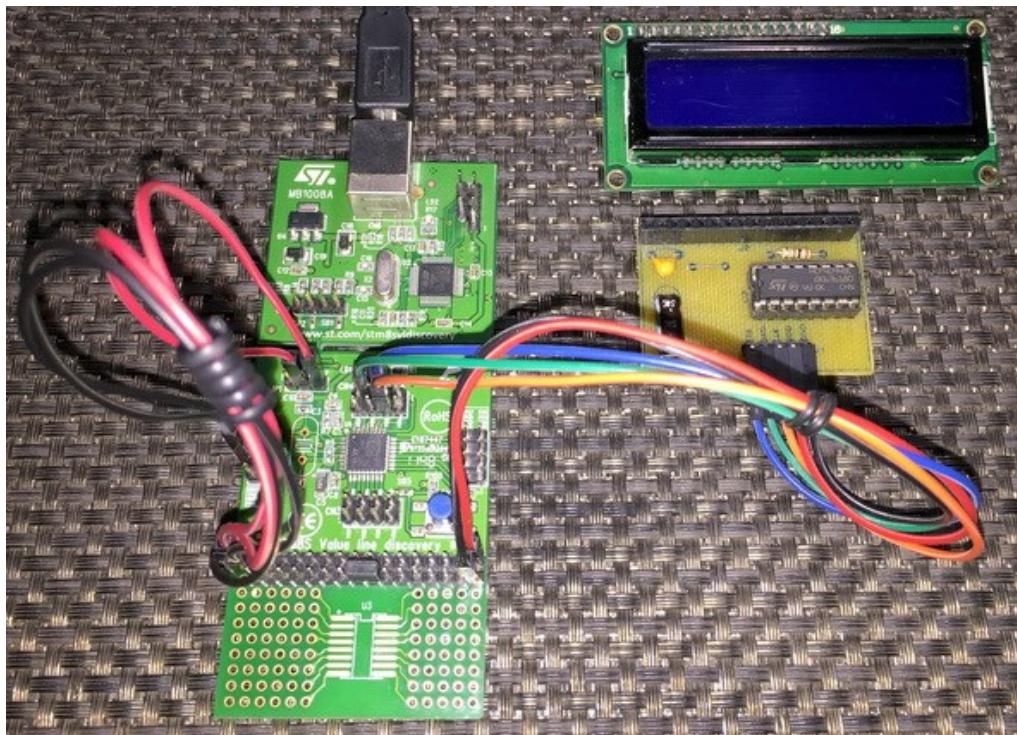
Video link: https://www.youtube.com/watch?v=WZpp8x1kf5k.

TFT Display – ST7735

Alphanumeric LCD are well-suited for most applications that require no graphical objects to be projected on a display screen. However, to make interfaces interactive and user-friendly, designers sometimes like things to be projected graphically. For example, consider an oscilloscope. In such cases, we need to use a graphical LCD (GLCD) or TFT display. Compared to GLCDs, TFTs are much more interactive, smarter and cheaper. Most GLCDs are monochrome and fat while TFTs are just the opposites. Because of these reasons, TFTs are more popular than GLCDs.
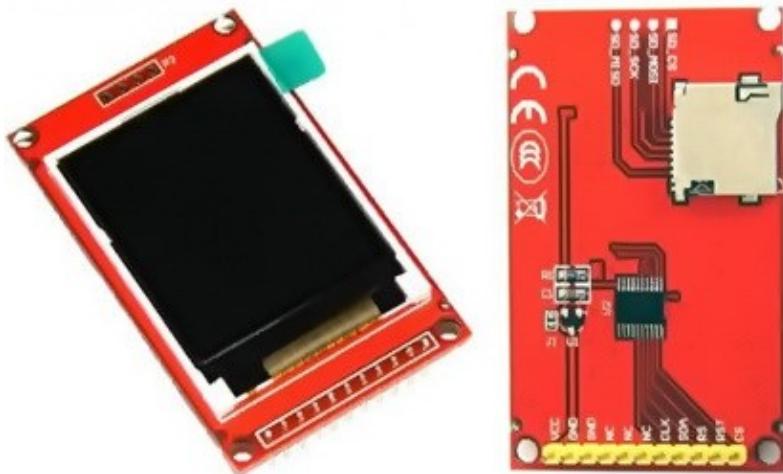
At present TFT displays are very common in the embedded-system market. There are many models of TFT displays, varying in screen size, resolution and other properties. ILI9325, IL9341, SSD1289, SSD1963, etc. are a few examples of common TFT display chips. However, most these TFT drivers need about sixteen plus GPIOs to function as they use 8080 display interface to communicate with host devices. ST7735-based TFT display stands differently in this regard. It needs SPI instead of 8080 display interface and it is good enough for most application. Adding a resistive touch screen on top of the display and SD card support will make it smarter.

Hardware Connection

Code Example

font.h

```c
static const unsigned char font[][5] =
{
     {0x00, 0x00, 0x00, 0x00, 0x00} // 20
    ,{0x00, 0x00, 0x5f, 0x00, 0x00} // 21 !
    ,{0x00, 0x07, 0x00, 0x07, 0x00} // 22 "
    ,{0x14, 0x7f, 0x14, 0x7f, 0x14} // 23 #
    ,{0x24, 0x2a, 0x7f, 0x2a, 0x12} // 24 $
    ,{0x23, 0x13, 0x08, 0x64, 0x62} // 25 %
    ,{0x36, 0x49, 0x55, 0x22, 0x50} // 26 &
    ,{0x00, 0x05, 0x03, 0x00, 0x00} // 27 '
    ,{0x00, 0x1c, 0x22, 0x41, 0x00} // 28 (
    ,{0x00, 0x41, 0x22, 0x1c, 0x00} // 29 )
    ,{0x14, 0x08, 0x3e, 0x08, 0x14} // 2a *
    ,{0x08, 0x08, 0x3e, 0x08, 0x08} // 2b +
    ,{0x00, 0x50, 0x30, 0x00, 0x00} // 2c ,
    ,{0x08, 0x08, 0x08, 0x08, 0x08} // 2d -
    ,{0x00, 0x60, 0x60, 0x00, 0x00} // 2e .
    ,{0x20, 0x10, 0x08, 0x04, 0x02} // 2f /
    ,{0x3e, 0x51, 0x49, 0x45, 0x3e} // 30 0
    ,{0x00, 0x42, 0x7f, 0x40, 0x00} // 31 1
    ,{0x42, 0x61, 0x51, 0x49, 0x46} // 32 2
    ,{0x21, 0x41, 0x45, 0x4b, 0x31} // 33 3
```

```
,{0x18, 0x14, 0x12, 0x7f, 0x10} // 34 4
,{0x27, 0x45, 0x45, 0x45, 0x39} // 35 5
,{0x3c, 0x4a, 0x49, 0x49, 0x30} // 36 6
,{0x01, 0x71, 0x09, 0x05, 0x03} // 37 7
,{0x36, 0x49, 0x49, 0x49, 0x36} // 38 8
,{0x06, 0x49, 0x49, 0x29, 0x1e} // 39 9
,{0x00, 0x36, 0x36, 0x00, 0x00} // 3a :
,{0x00, 0x56, 0x36, 0x00, 0x00} // 3b ;
,{0x08, 0x14, 0x22, 0x41, 0x00} // 3c <
,{0x14, 0x14, 0x14, 0x14, 0x14} // 3d =
,{0x00, 0x41, 0x22, 0x14, 0x08} // 3e >
,{0x02, 0x01, 0x51, 0x09, 0x06} // 3f ?
,{0x32, 0x49, 0x79, 0x41, 0x3e} // 40 @
,{0x7e, 0x11, 0x11, 0x11, 0x7e} // 41 A
,{0x7f, 0x49, 0x49, 0x49, 0x36} // 42 B
,{0x3e, 0x41, 0x41, 0x41, 0x22} // 43 C
,{0x7f, 0x41, 0x41, 0x22, 0x1c} // 44 D
,{0x7f, 0x49, 0x49, 0x49, 0x41} // 45 E
,{0x7f, 0x09, 0x09, 0x09, 0x01} // 46 F
,{0x3e, 0x41, 0x49, 0x49, 0x7a} // 47 G
,{0x7f, 0x08, 0x08, 0x08, 0x7f} // 48 H
,{0x00, 0x41, 0x7f, 0x41, 0x00} // 49 I
,{0x20, 0x40, 0x41, 0x3f, 0x01} // 4a J
,{0x7f, 0x08, 0x14, 0x22, 0x41} // 4b K
,{0x7f, 0x40, 0x40, 0x40, 0x40} // 4c L
,{0x7f, 0x02, 0x0c, 0x02, 0x7f} // 4d M
,{0x7f, 0x04, 0x08, 0x10, 0x7f} // 4e N
,{0x3e, 0x41, 0x41, 0x41, 0x3e} // 4f O
,{0x7f, 0x09, 0x09, 0x09, 0x06} // 50 P
,{0x3e, 0x41, 0x51, 0x21, 0x5e} // 51 Q
,{0x7f, 0x09, 0x19, 0x29, 0x46} // 52 R
,{0x46, 0x49, 0x49, 0x49, 0x31} // 53 S
,{0x01, 0x01, 0x7f, 0x01, 0x01} // 54 T
,{0x3f, 0x40, 0x40, 0x40, 0x3f} // 55 U
,{0x1f, 0x20, 0x40, 0x20, 0x1f} // 56 V
,{0x3f, 0x40, 0x38, 0x40, 0x3f} // 57 W
,{0x63, 0x14, 0x08, 0x14, 0x63} // 58 X
,{0x07, 0x08, 0x70, 0x08, 0x07} // 59 Y
,{0x61, 0x51, 0x49, 0x45, 0x43} // 5a Z
,{0x00, 0x7f, 0x41, 0x41, 0x00} // 5b [
,{0x02, 0x04, 0x08, 0x10, 0x20} // 5c ?
,{0x00, 0x41, 0x41, 0x7f, 0x00} // 5d ]
,{0x04, 0x02, 0x01, 0x02, 0x04} // 5e ^
,{0x40, 0x40, 0x40, 0x40, 0x40} // 5f _
,{0x00, 0x01, 0x02, 0x04, 0x00} // 60 `
,{0x20, 0x54, 0x54, 0x54, 0x78} // 61 a
,{0x7f, 0x48, 0x44, 0x44, 0x38} // 62 b
,{0x38, 0x44, 0x44, 0x44, 0x20} // 63 c
,{0x38, 0x44, 0x44, 0x48, 0x7f} // 64 d
,{0x38, 0x54, 0x54, 0x54, 0x18} // 65 e
,{0x08, 0x7e, 0x09, 0x01, 0x02} // 66 f
,{0x0c, 0x52, 0x52, 0x52, 0x3e} // 67 g
,{0x7f, 0x08, 0x04, 0x04, 0x78} // 68 h
,{0x00, 0x44, 0x7d, 0x40, 0x00} // 69 i
,{0x20, 0x40, 0x44, 0x3d, 0x00} // 6a j
,{0x7f, 0x10, 0x28, 0x44, 0x00} // 6b k
,{0x00, 0x41, 0x7f, 0x40, 0x00} // 6c l
,{0x7c, 0x04, 0x18, 0x04, 0x78} // 6d m
,{0x7c, 0x08, 0x04, 0x04, 0x78} // 6e n
```

```
    ,{0x38, 0x44, 0x44, 0x44, 0x38} // 6f o
    ,{0x7c, 0x14, 0x14, 0x14, 0x08} // 70 p
    ,{0x08, 0x14, 0x14, 0x18, 0x7c} // 71 q
    ,{0x7c, 0x08, 0x04, 0x04, 0x08} // 72 r
    ,{0x48, 0x54, 0x54, 0x54, 0x20} // 73 s
    ,{0x04, 0x3f, 0x44, 0x40, 0x20} // 74 t
    ,{0x3c, 0x40, 0x40, 0x20, 0x7c} // 75 u
    ,{0x1c, 0x20, 0x40, 0x20, 0x1c} // 76 v
    ,{0x3c, 0x40, 0x30, 0x40, 0x3c} // 77 w
    ,{0x44, 0x28, 0x10, 0x28, 0x44} // 78 x
    ,{0x0c, 0x50, 0x50, 0x50, 0x3c} // 79 y
    ,{0x44, 0x64, 0x54, 0x4c, 0x44} // 7a z
    ,{0x00, 0x08, 0x36, 0x41, 0x00} // 7b {
    ,{0x00, 0x00, 0x7f, 0x00, 0x00} // 7c |
    ,{0x00, 0x41, 0x36, 0x08, 0x00} // 7d }
    ,{0x10, 0x08, 0x08, 0x10, 0x08} // 7e ?
    ,{0x78, 0x46, 0x41, 0x46, 0x78} // 7f ?
};
```

ST7735.h

```c
#include "STM8S.h"
#include "font.h"


#define SPI_PORT                        GPIOC
#define CTL_PORT                        GPIOD

#define CS_pin                          GPIO_PIN_7
#define RS_pin                          GPIO_PIN_5
#define RST_pin                         GPIO_PIN_6
#define SCK_pin                         GPIO_PIN_5
#define SDA_pin                         GPIO_PIN_6

#define ST7735_NOP                      0x00
#define ST7735_SWRESET                  0x01
#define ST7735_RDDID                    0x04
#define ST7735_RDDST                    0x09
#define ST7735_RDDPM                    0x0A
#define ST7735_RDD_MADCTL               0x0B
#define STT7735_RDD_COLMOD              0x0C
#define ST7735_RDDIM                    0x0D
#define ST7735_RDDSM                    0x0E

#define ST7735_SLPIN                    0x10
#define ST7735_SLPOUT                   0x11
#define ST7735_PTLON                    0x12
#define ST7735_NORON                    0x13

#define ST7735_INVOFF                   0x20
#define ST7735_INVON                    0x21
#define ST7735_GAMSET                   0x26
#define ST7735_DISPOFF                  0x28
#define ST7735_DISPON                   0x29
#define ST7735_CASET                    0x2A
#define ST7735_RASET                    0x2B
#define ST7735_RAMWR                    0x2C
#define ST7735_RAMRD                    0x2E
```

```c
#define ST7735_PTLAR                    0x30
#define ST7735_TEOFF                    0x34
#define ST7735_TEON                     0x35
#define ST7735_MADCTL                   0x36
#define ST7735_IDMOFF                   0x38
#define ST7735_IDMON                    0x39
#define ST7735_COLMOD                   0x3A

#define ST7735_RDID1                    0xDA
#define ST7735_RDID2                    0xDB
#define ST7735_RDID3                    0xDC
#define ST7735_RDID4                    0xDD

#define ST7735_FRMCTR1                  0xB1
#define ST7735_FRMCTR2                  0xB2
#define ST7735_FRMCTR3                  0xB3
#define ST7735_INVCTR                   0xB4
#define ST7735_DISSET5                  0xB6

#define ST7735_PWCTR1                   0xC0
#define ST7735_PWCTR2                   0xC1
#define ST7735_PWCTR3                   0xC2
#define ST7735_PWCTR4                   0xC3
#define ST7735_PWCTR5                   0xC4
#define ST7735_VMCTR1                   0xC5

#define ST7735_RDID1                    0xDA
#define ST7735_RDID2                    0xDB
#define ST7735_RDID3                    0xDC
#define ST7735_RDID4                    0xDD

#define ST7735_PWCTR6                   0xFC

#define ST7735_GMCTRP1                  0xE0
#define ST7735_GMCTRN1                  0xE1

#define     BLACK                       0x0000
#define     BLUE                        0x001F
#define     RED                         0xF800
#define     GREEN                       0x07E0
#define     CYAN                        0x07FF
#define     MAGENTA                     0xF81F
#define     YELLOW                      0xFFE0
#define     WHITE                       0xFFFF

#define     MADCTL_MY                   0x80
#define     MADCTL_MX                   0x40
#define     MADCTL_MV                   0x20
#define     MADCTL_ML                   0x10
#define     MADCTL_RGB                  0x08
#define     MADCTL_MH                   0x04

#define ST7735_TFTWIDTH                 128
#define ST7735_TFTLENGTH                160

#define CMD                             0x0
#define DAT                             0x1
```

```c
#define SQUARE                                 0x00
#define ROUND                                  0x01

#define NO                                     0x00
#define YES                                    0x01


extern unsigned int width;
extern unsigned int length;


unsigned int Swap_Colour(unsigned int colour);
unsigned int Color565(unsigned char r, unsigned char g, unsigned char b);
void SPI_setup(void);
void ST7735_HW_init(void);
void ST7735_Write(unsigned char value, unsigned char mode);
void ST7735_Reset(void);
void ST7735_init(void);
void ST7735_Word_Write(unsigned int value);
void ST7735_Set_Addr_Window(signed int xs, signed int ys, signed int xe, signed int
ye);
void ST7735_RAM_Address_Set(void);
void ST7735_Invert_Display(unsigned char i);
void ST7735_Set_Rotation(unsigned char m);
void TFT_fill(unsigned int colour);
void Draw_Pixel(signed int x_pos, signed int y_pos, unsigned int colour);
void Draw_Line(signed int x1, signed int y1, signed int x2, signed int y2, unsigned
int colour);
void Draw_Rectangle(signed int x1, signed int y1, signed int x2, signed int y2,
unsigned char fill, unsigned char type, unsigned int colour, unsigned int
back_colour);
void Draw_Circle(signed int xc, signed int yc, signed int radius, unsigned char
fill, unsigned int colour);
void Draw_V_Line(signed int x1, signed int y1, signed int y2, unsigned colour);
void Draw_H_Line(signed int x1, signed int x2, signed int y1, unsigned colour);
void Draw_Triangle(signed int x1, signed int y1, signed int x2, signed int y2,
signed int x3, signed int y3, unsigned char fill, unsigned int colour);
void Draw_Font_Pixel(signed int x_pos, signed int y_pos, unsigned int colour,
unsigned char pixel_size);
void print_char(signed int x_pos, signed int  y_pos, unsigned char font_size,
unsigned int colour, unsigned int back_colour, char ch);
void print_str(signed int x_pos, signed int y_pos, unsigned char font_size, unsigned
int colour, unsigned int back_colour, char *ch);
void print_C(signed int x_pos, signed int y_pos, unsigned char font_size, unsigned
int colour, unsigned int back_colour, signed int value);
void print_I(signed int x_pos, signed int y_pos, unsigned char font_size, unsigned
int colour, unsigned int back_colour, signed int value);
void print_D(signed int x_pos, signed int y_pos, unsigned char font_size, unsigned
int colour, unsigned int back_colour, unsigned int value, unsigned char points);
void print_F(signed int x_pos, signed int y_pos, unsigned char font_size, unsigned
int colour, unsigned int back_colour, float value, unsigned char points);
```

ST7735.c

```c
#include "ST7735.h"


extern unsigned int width;
```

```c
extern unsigned int length;


unsigned int Swap_Colour(unsigned int colour)
{
    return ((colour << 0x000B) | (colour & 0x07E0) | (colour >> 0x000B));
}


unsigned int Color565(unsigned char r, unsigned char g, unsigned char b)
{
    return (((r & 0xF8) << 0x08) | ((g & 0xFC) << 0x03) | (b >> 0x03));
}


void SPI_setup(void)
{
    SPI_DeInit();

    SPI_Init(SPI_FIRSTBIT_MSB,
                SPI_BAUDRATEPRESCALER_2,
                SPI_MODE_MASTER,
                SPI_CLOCKPOLARITY_HIGH,
                SPI_CLOCKPHASE_1EDGE,
                SPI_DATADIRECTION_1LINE_TX,
                SPI_NSS_SOFT,
                0x00);

    SPI_Cmd(ENABLE);
}


void ST7735_HW_init(void)
{
    GPIO_Init(SPI_PORT,      ((GPIO_Pin_TypeDef)(SCK_pin       |      SDA_pin)),
GPIO_MODE_OUT_PP_HIGH_FAST);
    GPIO_Init(CTL_PORT,  ((GPIO_Pin_TypeDef)(CS_pin  |   RS_pin  |   RST_pin)),
GPIO_MODE_OUT_PP_HIGH_FAST);
    SPI_setup();
    delay_ms(10);
}


void ST7735_Write(unsigned char value, unsigned char mode)
{
    while(SPI_GetFlagStatus(SPI_FLAG_BSY));
    GPIO_WriteLow(CTL_PORT, CS_pin);

        switch(mode)
    {
        case CMD:
        {
            GPIO_WriteLow(CTL_PORT, RS_pin);
            break;
        }

        case DAT:
        {
            GPIO_WriteHigh(CTL_PORT, RS_pin);
```

```c
            break;
        }
    }

    SPI_SendData(value);
    while(!SPI_GetFlagStatus(SPI_FLAG_TXE));

    GPIO_WriteHigh(CTL_PORT, CS_pin);
}


void ST7735_Reset(void)
{
    GPIO_WriteLow(CTL_PORT, RST_pin);
    delay_ms(2);
    GPIO_WriteHigh(CTL_PORT, RST_pin);
    delay_ms(2);
}


void ST7735_init(void)
{
    ST7735_HW_init();
    ST7735_Reset();

    ST7735_Write(ST7735_SWRESET, CMD);
    delay_us(150);
    ST7735_Write(ST7735_SLPOUT, CMD);
    delay_us(150);

    ST7735_Write(ST7735_FRMCTR1, CMD);
    ST7735_Write(0x01, DAT);
    ST7735_Write(0x2C, DAT);
    ST7735_Write(0x2D, DAT);

    ST7735_Write(ST7735_FRMCTR2, CMD);
    ST7735_Write(0x01, DAT);
    ST7735_Write(0x2C, DAT);
    ST7735_Write(0x2D, DAT);

    ST7735_Write(ST7735_FRMCTR3, CMD);
    ST7735_Write(0x01, DAT);
    ST7735_Write(0x2C, DAT);
    ST7735_Write(0x2D, DAT);
    ST7735_Write(0x01, DAT);
    ST7735_Write(0x2C, DAT);
    ST7735_Write(0x2D, DAT);

    ST7735_Write(ST7735_INVCTR, CMD);
    ST7735_Write(0x07, DAT);

    ST7735_Write(ST7735_PWCTR1, CMD);
    ST7735_Write(0xA2, DAT);
    ST7735_Write(0x02, DAT);
    ST7735_Write(0x84, DAT);

    ST7735_Write(ST7735_PWCTR1, CMD);
    ST7735_Write(0xC5, DAT);
```

```c
ST7735_Write(ST7735_PWCTR2, CMD);
ST7735_Write(0x0A, DAT);
ST7735_Write(0x00, DAT);

ST7735_Write(ST7735_PWCTR3, CMD);
ST7735_Write(0x8A, DAT);
ST7735_Write(0x2A, DAT);

ST7735_Write(ST7735_PWCTR4, CMD);
ST7735_Write(0x8A, DAT);
ST7735_Write(0xEE, DAT);

ST7735_Write(ST7735_PWCTR5, CMD);
ST7735_Write(0x0E, DAT);

ST7735_Write(ST7735_VMCTR1, CMD);
ST7735_Write(0x00, DAT);

ST7735_Write(ST7735_COLMOD, CMD);
ST7735_Write(0x05, DAT);

ST7735_Write(ST7735_MADCTL, CMD);
ST7735_Write(0xC8, DAT);

ST7735_RAM_Address_Set();

ST7735_Write(ST7735_GMCTRP1, CMD);
ST7735_Write(0x02, DAT);
ST7735_Write(0x1C, DAT);
ST7735_Write(0x07, DAT);
ST7735_Write(0x12, DAT);
ST7735_Write(0x37, DAT);
ST7735_Write(0x32, DAT);
ST7735_Write(0x29, DAT);
ST7735_Write(0x2D, DAT);
ST7735_Write(0x29, DAT);
ST7735_Write(0x25, DAT);
ST7735_Write(0x2B, DAT);
ST7735_Write(0x39, DAT);
ST7735_Write(0x00, DAT);
ST7735_Write(0x01, DAT);
ST7735_Write(0x03, DAT);
ST7735_Write(0x10, DAT);

ST7735_Write(ST7735_GMCTRN1, CMD);
ST7735_Write(0x03, DAT);
ST7735_Write(0x1D, DAT);
ST7735_Write(0x07, DAT);
ST7735_Write(0x06, DAT);
ST7735_Write(0x2E, DAT);
ST7735_Write(0x2C, DAT);
ST7735_Write(0x29, DAT);
ST7735_Write(0x2D, DAT);
ST7735_Write(0x2E, DAT);
ST7735_Write(0x2E, DAT);
ST7735_Write(0x37, DAT);
ST7735_Write(0x3F, DAT);
ST7735_Write(0x00, DAT);
ST7735_Write(0x00, DAT);
```

```c
    ST7735_Write(0x02, DAT);
    ST7735_Write(0x10, DAT);

    ST7735_Write(ST7735_NORON, CMD);
    delay_ms(10);

    ST7735_Write(ST7735_DISPON, CMD);
    delay_ms(100);

    ST7735_Write(ST7735_RAMWR, CMD);
    delay_ms(100);
}


void ST7735_Word_Write(unsigned int value)
{
    ST7735_Write(((value & 0xFF00) >> 0x08), DAT);
    ST7735_Write((value & 0x00FF), DAT);
}


void ST7735_Set_Addr_Window(signed int xs, signed int ys, signed int xe, signed int ye)
{
    ST7735_Write(ST7735_CASET, CMD);
    ST7735_Write(0x00, DAT);
    ST7735_Write(xs, DAT);
    ST7735_Write(0x00, DAT);
    ST7735_Write(xe, DAT);

    ST7735_Write(ST7735_RASET, CMD);
    ST7735_Write(0x00, DAT);
    ST7735_Write(ys, DAT);
    ST7735_Write(0x00, DAT);
    ST7735_Write(ye, DAT);

    ST7735_Write(ST7735_RAMWR, CMD);
}


void ST7735_RAM_Address_Set(void)
{
     ST7735_Set_Addr_Window(0x00, 0x00, 0x7F, 0x9F);
}


void ST7735_Invert_Display(unsigned char i)
{
     ST7735_Write(i, CMD);
}


void ST7735_Set_Rotation(unsigned char m)
{
    unsigned char rotation = 0x00;

    ST7735_Write(ST7735_MADCTL, CMD);
    rotation = (m % 4);
```

```c
    switch(rotation)
    {
        case 0:
        {
            ST7735_Write((MADCTL_MX | MADCTL_MY | MADCTL_RGB), DAT);
            width  = ST7735_TFTWIDTH;
            length = ST7735_TFTLENGTH;
            break;
        }
        case 1:
        {
            ST7735_Write((MADCTL_MY | MADCTL_MV | MADCTL_RGB), DAT);
            width  = ST7735_TFTLENGTH;
            length = ST7735_TFTWIDTH;
            break;
        }
        case 2:
        {
            ST7735_Write((MADCTL_RGB), DAT);
            width = ST7735_TFTWIDTH;
            length = ST7735_TFTLENGTH;
            break;
        }
        case 3:
        {
            ST7735_Write((MADCTL_MX | MADCTL_MV | MADCTL_RGB), DAT);
            width = ST7735_TFTLENGTH;
            length = ST7735_TFTWIDTH;
            break;
        }
    }
}


void TFT_fill(unsigned int colour)
{
    signed int i = 0x00;
    signed int j = 0x00;

    ST7735_Set_Addr_Window(0, 0, (width - 1), (length - 1));

    for(j = length; j > 0; j--)
    {
        for(i = width; i > 0; i--)
        {
            ST7735_Word_Write(colour);
        }
    }
}


void Draw_Pixel(signed int x_pos, signed int y_pos, unsigned int colour)
{
    ST7735_Set_Addr_Window(x_pos, y_pos, (1 + x_pos), (1 + y_pos));
    ST7735_Word_Write(colour);
}
```

```c
void Draw_Line(signed int x1, signed int y1, signed int x2, signed int y2, unsigned
int colour)
{
    signed int dx = 0x0000;
    signed int dy = 0x0000;
    signed int stepx = 0x0000;
    signed int stepy = 0x0000;
    signed int fraction = 0x0000;

    dy = (y2 - y1);
    dx = (x2 - x1);

    if (dy < 0)
    {
        dy = -dy;
        stepy = -1;
    }
    else
    {
        stepy = 1;
    }

    if (dx < 0)
    {
        dx = -dx;
        stepx = -1;
    }
    else
    {
        stepx = 1;
    }

    dx <<= 0x01;
    dy <<= 0x01;

    Draw_Pixel(x1, y1, colour);

    if (dx > dy)
    {
        fraction = (dy - (dx >> 1));
        while (x1 != x2)
        {
            if (fraction >= 0)
            {
                y1 += stepy;
                fraction -= dx;
            }
            x1 += stepx;
            fraction += dy;

            Draw_Pixel(x1, y1, colour);
        }
    }
    else
    {
        fraction = (dx - (dy >> 1));

        while (y1 != y2)
        {
```

```c
            if (fraction >= 0)
            {
                x1 += stepx;
                fraction -= dy;
            }
            y1 += stepy;
            fraction += dx;
            Draw_Pixel(x1, y1, colour);
        }
    }
}


void Draw_Rectangle(signed int x1, signed int y1, signed int x2, signed int y2,
unsigned char fill, unsigned char type, unsigned int colour, unsigned int
back_colour)
{
    unsigned char i = 0x00;
    unsigned char xmin = 0x00;
    unsigned char xmax = 0x00;
    unsigned char ymin = 0x00;
    unsigned char ymax = 0x00;

    if(fill != NO)
    {
        if(x1 < x2)
        {
            xmin = x1;
            xmax = x2;
        }
        else
        {
            xmin = x2;
            xmax = x1;
        }

        if(y1 < y2)
        {
            ymin = y1;
            ymax = y2;
        }
        else
        {
            ymin = y2;
            ymax = y1;
        }

        for(; xmin <= xmax; ++xmin)
        {
            for(i = ymin; i <= ymax; ++i)
            {
                Draw_Pixel(xmin, i, colour);
            }
        }
    }

    else
    {
        Draw_Line(x1, y1, x2, y1, colour);
```

```c
            Draw_Line(x1, y2, x2, y2, colour);
            Draw_Line(x1, y1, x1, y2, colour);
            Draw_Line(x2, y1, x2, y2, colour);
        }

        if(type != SQUARE)
        {
            Draw_Pixel(x1, y1, back_colour);
            Draw_Pixel(x1, y2, back_colour);
            Draw_Pixel(x2, y1, back_colour);
            Draw_Pixel(x2, y2, back_colour);
        }
}


void Draw_Circle(signed int xc, signed int yc, signed int radius, unsigned char
fill, unsigned int colour)
{
    signed int a = 0x0000;
    signed int b = 0x0000;
    signed int p = 0x0000;

    b = radius;
    p = (1 - b);

    do
    {
        if(fill != NO)
        {
            Draw_Line((xc - a), (yc + b), (xc + a), (yc + b), colour);
            Draw_Line((xc - a), (yc - b), (xc + a), (yc - b), colour);
            Draw_Line((xc - b), (yc + a), (xc + b), (yc + a), colour);
            Draw_Line((xc - b), (yc - a), (xc + b), (yc - a), colour);
        }
        else
        {
            Draw_Pixel((xc + a), (yc + b), colour);
            Draw_Pixel((xc + b), (yc + a), colour);
            Draw_Pixel((xc - a), (yc + b), colour);
            Draw_Pixel((xc - b), (yc + a), colour);
            Draw_Pixel((xc + b), (yc - a), colour);
            Draw_Pixel((xc + a), (yc - b), colour);
            Draw_Pixel((xc - a), (yc - b), colour);
            Draw_Pixel((xc - b), (yc - a), colour);
        }

        if(p < 0)
        {
            p += (0x03 + (0x02 * a++));
        }
        else
        {
            p += (0x05 + (0x02 * ((a++) - (b--))));
        }
    }while(a <= b);
}


void Draw_V_Line(signed int x1, signed int y1, signed int y2, unsigned colour)
```

```c
{
    signed int pos = 0;
    signed int temp = 0;

    if(y1 > y2)
    {
        swap(&y1, &y2);
    }

    while(y2 > (y1 - 1))
    {
        Draw_Pixel(x1, y2, colour);
        y2--;
    }
}


void Draw_H_Line(signed int x1, signed int x2, signed int y1, unsigned colour)
{
    signed int pos = 0;
    signed int temp = 0;

    if(x1 > x2)
    {
        swap(&x1, &x2);
    }

    while(x2 > (x1 - 1))
    {
        Draw_Pixel(x2, y1, colour);
        x2--;
    }
}


void Draw_Triangle(signed int x1, signed int y1, signed int x2, signed int y2,
signed int x3, signed int y3, unsigned char fill, unsigned int colour)
{
    signed int a = 0;
    signed int b = 0;
    signed int sa = 0;
    signed int sb = 0;
    signed int yp = 0;
    signed int last = 0;
    signed int dx12 = 0;
    signed int dx23 = 0;
    signed int dx13 = 0;
    signed int dy12 = 0;
    signed int dy23 = 0;
    signed int dy13 = 0;

    switch(fill)
    {
        case YES:
        {
            if(y1 > y2)
            {
                swap(&y1, &y2);
                swap(&x1, &x2);
```

```c
        }
        if(y2 > y3)
        {
            swap(&y3, &y2);
            swap(&x3, &x2);
        }
        if(y1 > y2)
        {
            swap(&y1, &y2);
            swap(&x1, &x2);
        }

        if(y1 == y3)
        {
            a = b = x1;

            if(x2 < a)
            {
                a = x2;
            }
            else if(x2 > b)
            {
                b = x2;
            }
            if(x2 < a)
            {
                a = x3;
            }
            else if(x3 > b)
            {
                b = x3;
            }

            Draw_H_Line(a, (a + (b - (a + 1))), y1, colour);
            return;
        }

        dx12 = (x2 - x1);
        dy12 = (y2 - y1);
        dx13 = (x3 - x1);
        dy13 = (y3 - y1);
        dx23 = (x3 - x2);
        dy23 = (y3 - y2);
        sa = 0,
        sb = 0;

        if(y2 == y3)
        {
            last = y2;
        }
        else
        {
            last = (y2 - 1);
        }

        for(yp = y1; yp <= last; yp++)
        {
            a = (x1 + (sa / dy12));
            b = (x1 + (sb / dy13));
```

```c
                sa += dx12;
                sb += dx13;
                if(a > b)
                {
                    swap(&a, &b);
                }
                Draw_H_Line(a, (a + (b - (a + 1))), yp, colour);
            }

            sa = (dx23 * (yp - y2));
            sb = (dx13 * (yp - y1));
            for(; yp <= y3; yp++)
            {
                a = (x2 + (sa / dy23));
                b = (x1 + (sb / dy13));
                sa += dx23;
                sb += dx13;

                if(a > b)
                {
                    swap(&a, &b);
                }
                Draw_H_Line(a, (a + (b - (a + 1))), yp, colour);
            }


            break;
        }
        default:
        {
            Draw_Line(x1, y1, x2, y2, colour);
            Draw_Line(x2, y2, x3, y3, colour);
            Draw_Line(x3, y3, x1, y1, colour);
            break;
        }
    }
}


void Draw_Font_Pixel(signed int x_pos, signed int y_pos, unsigned int colour,
unsigned char pixel_size)
{
    unsigned char i = 0x00;

    ST7735_Set_Addr_Window(x_pos, y_pos, (x_pos + pixel_size - 1), (y_pos +
pixel_size - 1));

    for(i = 0x00; i < (pixel_size * pixel_size); i++)
    {
        ST7735_Word_Write(colour);
    }
}


void print_char(signed int x_pos, signed int  y_pos, unsigned char font_size,
unsigned int colour, unsigned int back_colour, char ch)
{
    unsigned char i = 0x00;
    unsigned char j = 0x00;
```

```c
    char value = 0x00;

    if(font_size < 0)
    {
        font_size = 1;
    }

    if(x_pos < font_size)
    {
        x_pos = font_size;
    }

    for (i = 0x00; i < 0x05; i++)
    {
        for (j = 0x00; j < 0x08; j++)
        {
            value = 0x00;
            value = ((font[((unsigned char)ch) - 0x20][i]));

            if((value >> j)  & 0x01)
            {
                Draw_Font_Pixel(x_pos, y_pos, colour, font_size);
            }
            else
            {
                Draw_Font_Pixel(x_pos, y_pos, back_colour, font_size);
            }

            y_pos = y_pos + font_size;
        }
        y_pos -= (font_size << 0x03);
        x_pos += font_size;
    }

    x_pos += font_size;

    if(x_pos > width)
    {
        x_pos = (font_size + 0x01);
        y_pos += (font_size << 0x03);
    }
}


void print_str(signed int x_pos, signed int y_pos, unsigned char font_size, unsigned int colour, unsigned int back_colour, char *ch)
{
    do
    {
        print_char(x_pos, y_pos, font_size, colour, back_colour, *ch++);
        x_pos += (font_size * 0x06);
    }while((*ch >= 0x20) && (*ch <= 0x7F) && (*ch != '\n'));
}


void print_C(signed int x_pos, signed int y_pos, unsigned char font_size, unsigned int colour, unsigned int back_colour, signed int value)
{
```

```c
        char ch[4] = {0x20, 0x20, 0x20, 0x20};

        if(value < 0x00)
        {
            ch[0] = 0x2D;
            value = -value;
        }
        else
        {
            ch[0] = 0x20;
        }

        if((value > 99) && (value <= 999))
        {
            ch[1] = ((value / 100) + 0x30);
            ch[2] = (((value % 100) / 10) + 0x30);
            ch[3] = ((value % 10) + 0x30);
        }
        else if((value > 9) && (value <= 99))
        {
            ch[1] = (((value % 100) / 10) + 0x30);
            ch[2] = ((value % 10) + 0x30);
            ch[3] = 0x20;
        }
        else if((value >= 0) && (value <= 9))
        {
            ch[1] = ((value % 10) + 0x30);
            ch[2] = 0x20;
            ch[3] = 0x20;
        }

        print_str(x_pos, y_pos, font_size, colour, back_colour, ch);
}


void print_I(signed int x_pos, signed int y_pos, unsigned char font_size, unsigned
int colour, unsigned int back_colour, signed int value)
{
    char ch[6] = {0x20, 0x20, 0x20, 0x20, 0x20, 0x20};

    if(value < 0)
    {
        ch[0] = 0x2D;
        value = -value;
    }
    else
    {
        ch[0] = 0x20;
    }

    if(value > 9999)
    {
        ch[1] = ((value / 10000) + 0x30);
        ch[2] = (((value % 10000) / 1000) + 0x30);
        ch[3] = (((value % 1000) / 100) + 0x30);
        ch[4] = (((value % 100) / 10) + 0x30);
        ch[5] = ((value % 10) + 0x30);
    }
```

```c
        else if((value > 999) && (value <= 9999))
        {
            ch[1] = (((value % 10000) / 1000) + 0x30);
            ch[2] = (((value % 1000) / 100) + 0x30);
            ch[3] = (((value % 100) / 10) + 0x30);
            ch[4] = ((value % 10) + 0x30);
            ch[5] = 0x20;
        }
        else if((value > 99) && (value <= 999))
        {
            ch[1] = (((value % 1000) / 100) + 0x30);
            ch[2] = (((value % 100) / 10) + 0x30);
            ch[3] = ((value % 10) + 0x30);
            ch[4] = 0x20;
            ch[5] = 0x20;
        }
        else if((value > 9) && (value <= 99))
        {
            ch[1] = (((value % 100) / 10) + 0x30);
            ch[2] = ((value % 10) + 0x30);
            ch[3] = 0x20;
            ch[4] = 0x20;
            ch[5] = 0x20;
        }
        else
        {
            ch[1] = ((value % 10) + 0x30);
            ch[2] = 0x20;
            ch[3] = 0x20;
            ch[4] = 0x20;
            ch[5] = 0x20;
        }

        print_str(x_pos, y_pos, font_size, colour, back_colour, ch);
}


void print_D(signed int x_pos, signed int y_pos, unsigned char font_size, unsigned
int colour, unsigned int back_colour, unsigned int value, unsigned char points)
{
        char ch[5] = {0x2E, 0x20, 0x20, 0x20, 0x20};

        ch[1] = ((value / 1000) + 0x30);

        if(points > 1)
        {
            ch[2] = (((value % 1000) / 100) + 0x30);

            if(points > 2)
            {
                ch[3] = (((value % 100) / 10) + 0x30);

                if(points > 3)
                {
                    ch[4] = ((value % 10) + 0x30);
                }
            }
        }
```

```c
    print_str(x_pos, y_pos, font_size, colour, back_colour, ch);
}


void print_F(signed int x_pos, signed int y_pos, unsigned char font_size, unsigned
int colour, unsigned int back_colour, float value, unsigned char points)
{
    signed long tmp = 0x0000;

    tmp = value;
    print_I(x_pos, y_pos, font_size, colour, back_colour, tmp);
    tmp = ((value - tmp) * 10000);

    if(tmp < 0)
    {
        tmp = -tmp;
    }

    if((value >= 9999) && (value < 99999))
    {
        print_D((x_pos + (0x24 * font_size)), y_pos, font_size, colour, back_colour,
tmp, points);
    }
    else if((value >= 999) && (value < 9999))
    {
        print_D((x_pos + (0x1E * font_size)), y_pos, font_size, colour, back_colour,
tmp, points);
    }
    else if((value >= 99) && (value < 999))
    {
        print_D((x_pos + (0x18 * font_size)), y_pos, font_size, colour, back_colour,
tmp, points);
    }
    else if((value >= 9) && (value < 99))
    {
        print_D((x_pos + (0x12 * font_size)), y_pos, font_size, colour, back_colour,
tmp, points);
    }
    else if(value < 9)
    {
        print_D((x_pos + (0x0C * font_size)), y_pos, font_size, colour, back_colour,
tmp, points);

        if(value < 0)
        {
            print_char(x_pos, y_pos, font_size, colour, back_colour, 0x2D);
        }
        else
        {
            print_char(x_pos, y_pos, font_size, colour, back_colour, 0x20);
        }
    }
}
```

main.c

```c
#include "STM8S.h"
#include "ST7735.h"


unsigned int width;
unsigned int length;


void setup_clock(void);
void setup_GPIOs(void);


void main(void)
{
    float f = -0.09;
    signed int i = -9;
    signed char c = 127;

    setup_clock();
    setup_GPIOs();
    ST7735_init();

    ST7735_Set_Rotation(0x01);

    TFT_fill(Swap_Colour(GREEN));
    Draw_Circle(79, 63, 20, YES, Swap_Colour(RED));
    delay_ms(4000);

    TFT_fill(Swap_Colour(WHITE));

    Draw_Circle(6, 6, 4, YES, RED);
    Draw_Circle(153, 6, 4, YES, RED);
    Draw_Circle(6, 121, 4, YES, RED);
    Draw_Circle(153, 121, 4, YES, RED);

    delay_ms(1000);

    Draw_Line(14, 0, 14, 127, CYAN);
    Draw_Line(145, 0, 145, 127, CYAN);
    Draw_Line(0, 14, 159, 14, CYAN);
    Draw_Line(0, 113, 159, 113, CYAN);

    delay_ms(1000);

    Draw_Rectangle(17, 17, 142, 110, YES, ROUND, BLUE, WHITE);
    delay_ms(1000);

    print_str(22, 58, 2, YELLOW, BLUE, "MicroArena");
    delay_ms(4000);

    TFT_fill(BLACK);
    print_str(20, 90, 1, YELLOW, BLACK, "www.fb.com/MicroArena");

    while(1)
    {
        print_F(60, 20, 1, BLUE, BLACK, f, 2);
        print_C(60, 40, 1, RED, BLACK, c);
```

```c
            print_I(60, 60, 1, GREEN, BLACK, i);
            f += 0.01;
            c -= 1;
            i += 1;
            if(c < -128)
            {
                c = 127;
            }
            delay_ms(60);
    };
}


void setup_clock(void)
{
    CLK_DeInit();

    CLK_HSECmd(ENABLE);
    CLK_LSICmd(DISABLE);
    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSERDY) == FALSE);

    CLK_ClockSwitchCmd(ENABLE);
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
    CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

    CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSE,
    DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

    CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, ENABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART2, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER3, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void setup_GPIOs(void)
{
    GPIO_DeInit(SPI_PORT);
    GPIO_DeInit(CTL_PORT);
}
```

Explanation

Well that's a hell lot code. Explaining the working of the ST7735 TFT display is beyond the scope of this article. The goal here is just the integration of such displays with STM8s. The library I coded here has several functions which however need some explanation.

The following functions set display inversion and orientation.

```
void ST7735_Invert_Display(unsigned char i);
void ST7735_Set_Rotation(unsigned char m);
```

For graphics, the following functions can be used:

```
/*
Fills the entire TFT area with the designated colour.
*/

void TFT_fill(unsigned int colour);

/*
Draws a pixel of designated colour on coordinate (x_pos, y_pos).
*/

void Draw_Pixel(signed int x_pos, signed int y_pos, unsigned int colour);

/*
Draws a line of designated colour from point (x1, y1) to (x2, y2).
*/

void Draw_Line(signed int x1, signed int y1, signed int x2, signed int y2, unsigned
int colour);


/*
Draws a rectangle from point (x1, y1) to (x2, y2). "Fill" states whether or not the
rectangular area is solid filled. "Type" states whether the edges of the rectangle
are round or square. "Colour" states the colour of the edges of the rectangle and
"back colour" states the colour of the area inside the rectangle.
*/

void Draw_Rectangle(signed int x1, signed int y1, signed int x2, signed int y2,
unsigned  char  fill,  unsigned  char  type,  unsigned  int  colour,  unsigned  int
back_colour);

/*
Draws a circle of a given radius with center at (xc, yc). Argument "radius" states
radius of the circle in pixels. "Fill" states whether or not the circular area is
solid filled. "Colour" states colour of the circle.
*/


void Draw_Circle(signed int xc, signed int yc, signed int radius, unsigned char
fill, unsigned int colour);

/*
Draws a vertical line of designated colour from point (x1, y1) to point (x1, y2)
```

```
*/

void Draw_V_Line(signed int x1, signed int y1, signed int y2, unsigned colour);

/*
Draws a horizontal line of designated colour from point (x1, y1) to point (x2, y1)
*/

void Draw_H_Line(signed int x1, signed int x2, signed int y1, unsigned colour);

/*
Draws a triangle of designated colour in the area enclosed by points (x1, y1), (x2,
y2) and (x3, y3). "Fill" states whether or not the occupied area is solid filled.
*/

void Draw_Triangle(signed int x1, signed int y1, signed int x2, signed int y2,
signed int x3, signed int y3, unsigned char fill, unsigned int colour);
```

For printing texts, the following functions can be used:

```
/*
Prints a single character "ch" at (x_pos, y_pos) with size stated by "font_size".
"Colour" states font colour and "back_colour" states the font background colour.
*/

void  print_char(signed int x_pos, signed int  y_pos, unsigned char  font_size,
unsigned int colour, unsigned int back_colour, char ch);

/*
Prints a string of characters from point (x_pos, y_pos) with size stated by
"font_size". "Colour" states font colour and "back_colour" states the font
background colour.
*/

void print_str(signed int x_pos, signed int y_pos, unsigned char font_size, unsigned
int colour, unsigned int back_colour, char *ch);

/*
Prints a character type variable in "value" at (x_pos, y_pos) with size stated by
"font_size". "Colour" states font colour and "back_colour" states the font
background colour.
*/

void print_C(signed int x_pos, signed int y_pos, unsigned char font_size, unsigned
int colour, unsigned int back_colour, signed int value);

/*
Prints an integer type variable in "value" at (x_pos, y_pos) with size stated by
"font_size". "Colour" states font colour and "back_colour" states the font
background colour.
*/

void print_I(signed int x_pos, signed int y_pos, unsigned char font_size, unsigned
int colour, unsigned int back_colour, signed int value);

/*
```

```
Prints a float type variable in "value" at (x_pos, y_pos) with size stated by
"font_size". "Colour" states font colour and "back_colour" states the font
background colour. Points states the number of digits after decimal point.
*/

void print_F(signed int x_pos, signed int y_pos, unsigned char font_size, unsigned
int colour, unsigned int back_colour, float value, unsigned char points);
```

Initialize the TFT display before using it by including the following initialization function in the main file:

```
ST7735_init();
```

Since the TFT display communicates by using SPI communication method, enable the SPI peripheral clock and necessary GPIOs:

```
CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, ENABLE);
```

Font.h contains the character fonts and must be included to print texts, symbols and numbers.

One word of caution, by the way, is to look out for possible stack overflow and memory model related issues. If such a case arise then you should check the memory model of your code from project settings as shown:
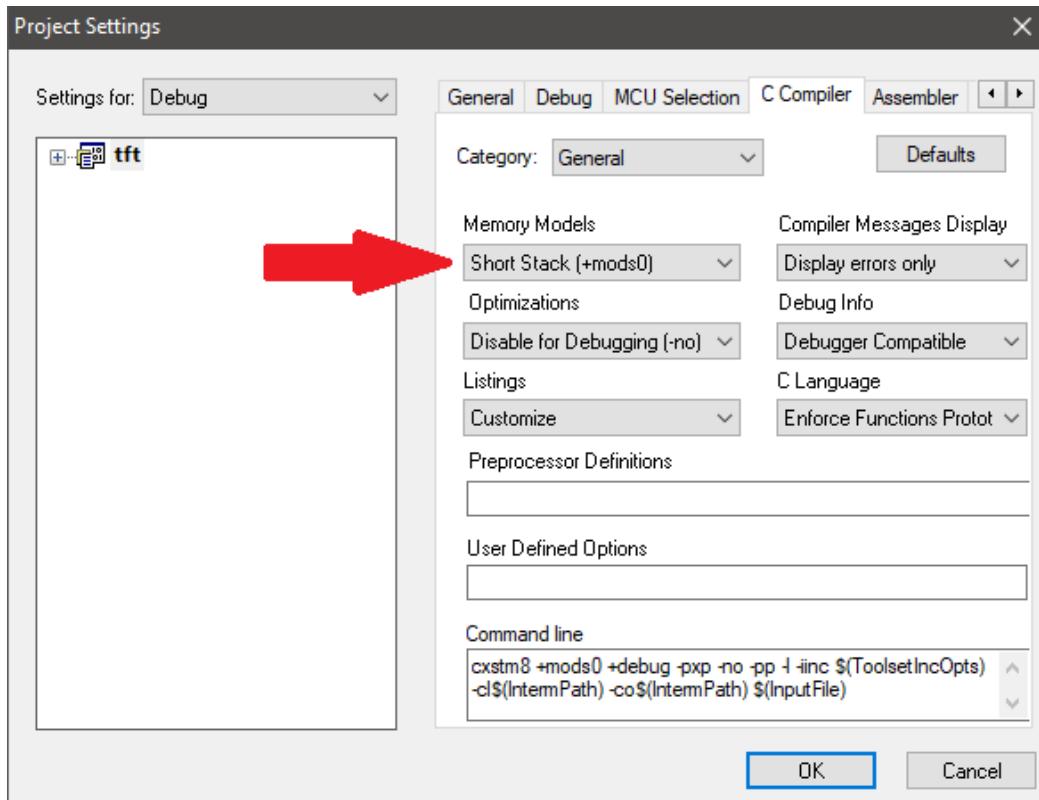
Usually this case occurs when we try to declare a large array or do something similar. By default, short stack memory model is selected and most projects will flawlessly work with this memory model. However, for some projects especially those related to TFT, OLED, graphical displays, GSM, GPS and similar, long stack model is a must or else you won't be able to compile your code any way.

Demo
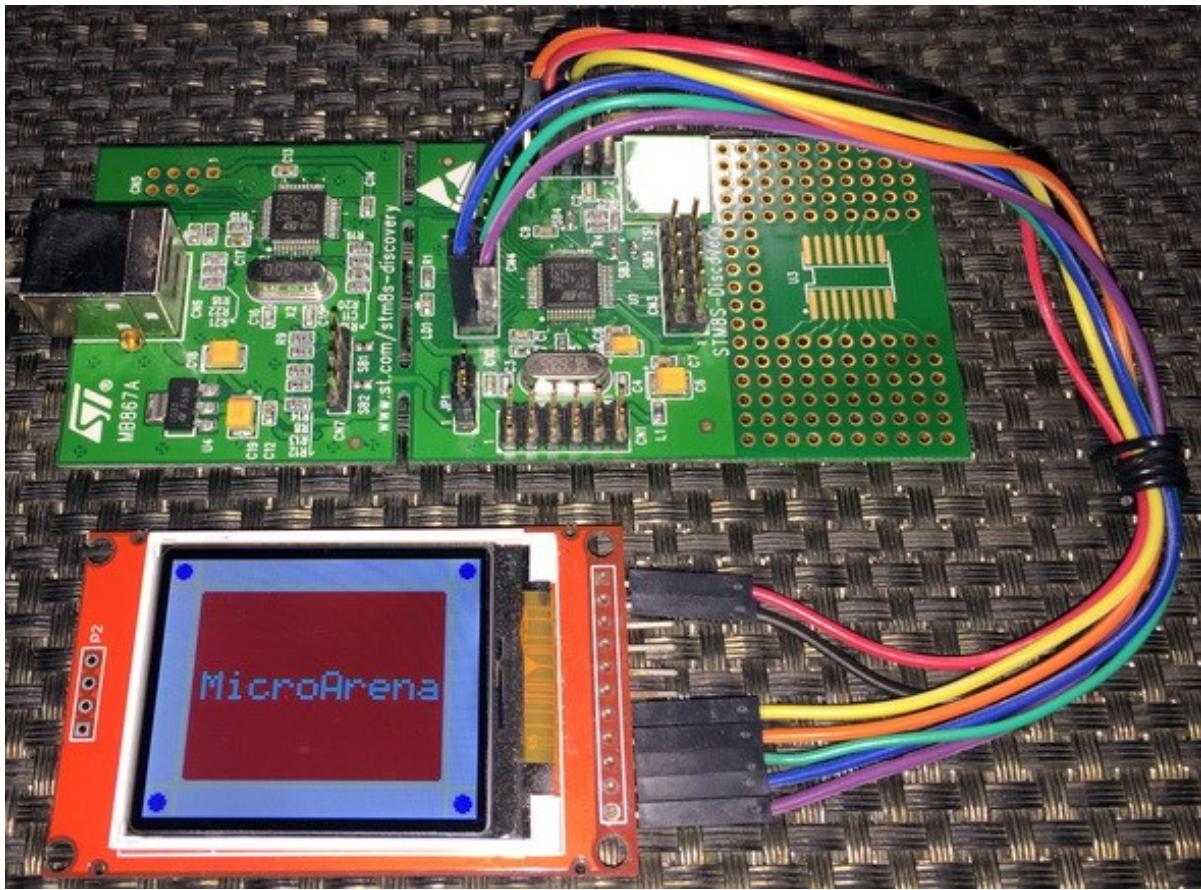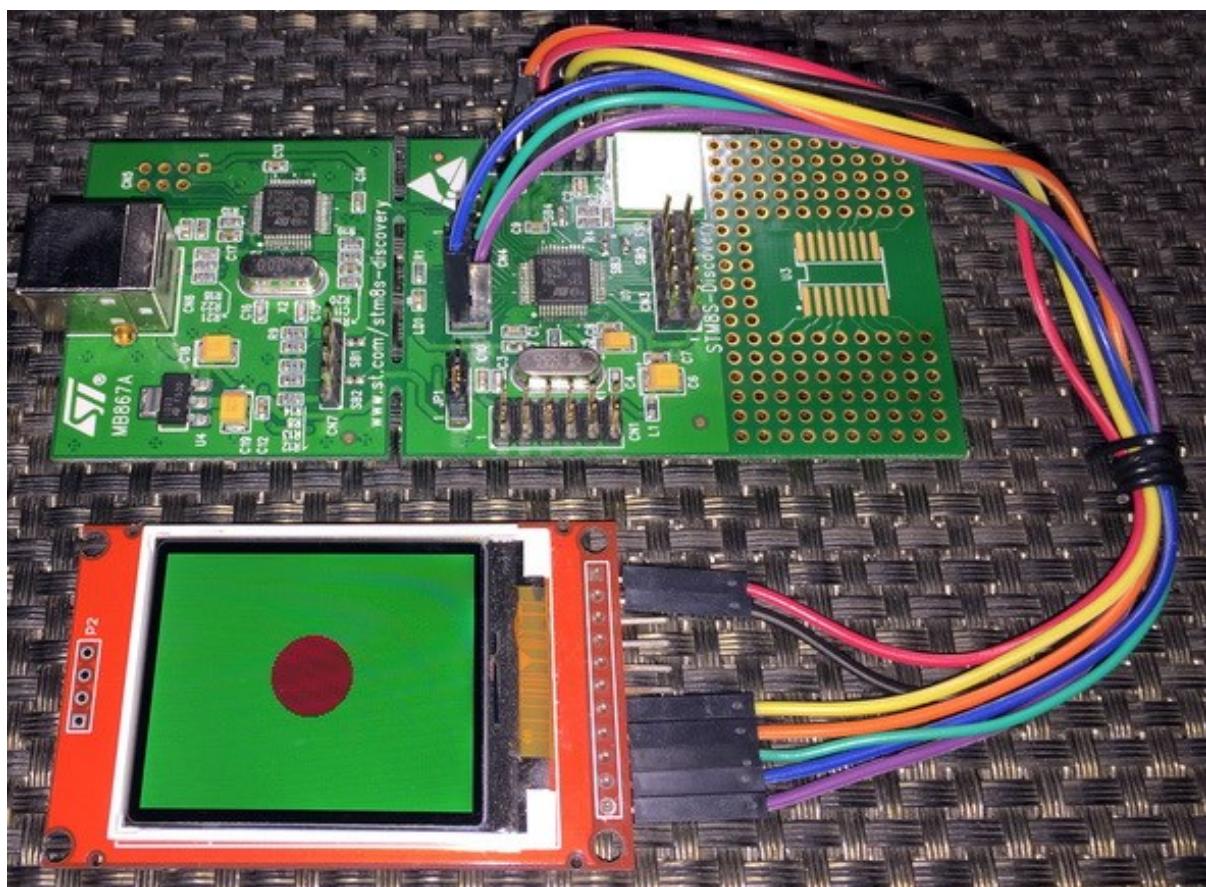
Video link: https://www.youtube.com/watch?v=R-NL_7BuCwM.

Inter-Integrated Circuit (I2C)

I2C is another popular form of on board synchronous serial communication developed by NXP (Philips). It just uses two wires for communication and so it is also referred as **Two Wire Interface (TWI)**. Just like SPI, I2C is widely used in interfacing real-time clocks (RTC), digital sensors, LCDs, memory chips and so on. It is as much as popular as SPI but compared to SPI it is slower and have some limitations. Up to 127 devices can be connected in an I2C bus. In an I2C bus, however, it is not possible, by conventional means, to interface devices with same device addresses or devices with different logic voltage levels without logic level converters and so on. Still however, I2C is very popular because these issues rarely arise and because of its simplicity. Unlike other communications, there is no hassle of pin/wire swapping as two wires connect straight to the bus – SDA to SDA and SCL to SCL.
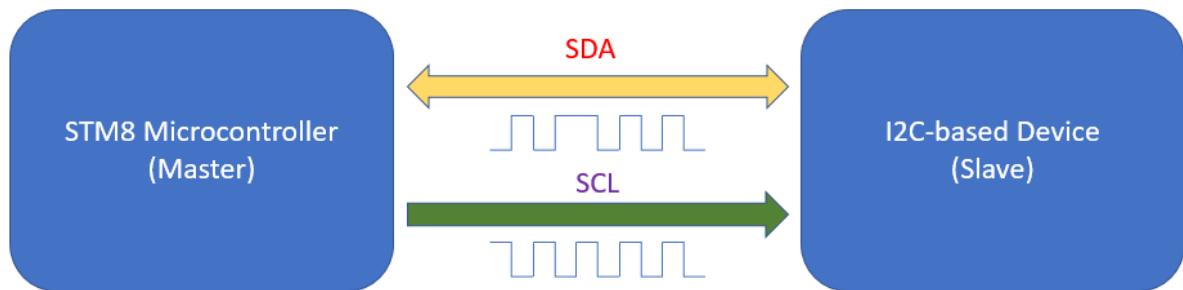
Just like SPI, an I2C bus must contain one master device (usually a microcontroller) and one or more slaves. The master is solely responsible for generating clock signals and initiating communication. Communication starts when master sends out a slave's ID with read/write command. The slave reacts to this command by processing the request from the master and sending out data.

To know more about I2C interface visit the following links:

- https://learn.mikroe.com/i2c-everything-need-know

- https://learn.sparkfun.com/tutorials/i2c

- http://www.ti.com/lsds/ti/interface/i2c-overview.page

- http://www.robot-electronics.co.uk/i2c-tutorial

- https://www.i2c-bus.org/i2c-bus

- http://i2c.info

Other protocols like SMBus and I2S have similarities with I2C and so learning about I2C advances learning these too.

Hardware Connection
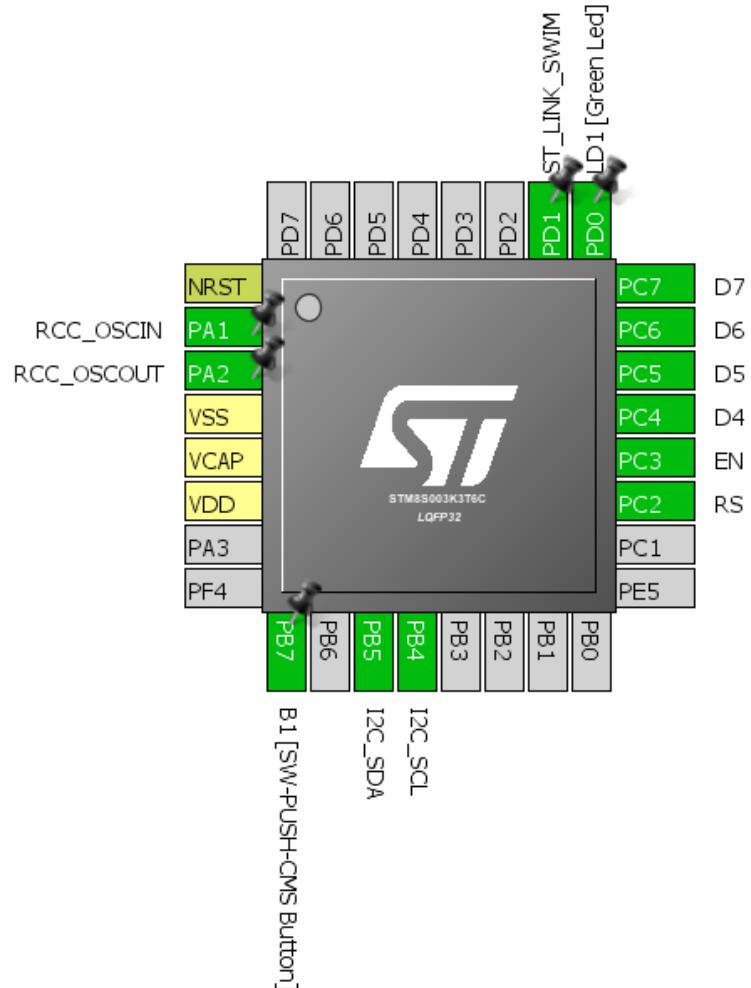
Code Example

This code demonstrates how to interface BH1750 I2C digital light sensor with STM8S003K3. A LCD is used to display the light sensor's output in lux.

main.c

```c
#include "STM8S.h"
#include "BH1750.h"
#include "lcd.h"


void clock_setup(void);
void GPIO_setup(void);
void I2C_setup(void);
void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned int value);
```

```c
void main(void)
{
    unsigned int LX = 0x0000;
    unsigned int tmp = 0x0000;

    clock_setup();
    GPIO_setup();
    I2C_setup();
    LCD_init();
    BH1750_init();

    LCD_clear_home();
    LCD_goto(0, 0);
    LCD_putstr("STM8 I2C");
    LCD_goto(0, 1);
    LCD_putstr("Lx");
    delay_ms(10);

    while(TRUE)
    {
        tmp = get_lux_value(cont_L_res_mode, 20);

        if(tmp > 10)
        {
            LX = tmp;
        }
        else
        {
            LX = get_lux_value(cont_H_res_mode1, 140);
        }

        lcd_print(3, 1, LX);
        delay_ms(200);
    };
}


void clock_setup(void)
{
    CLK_DeInit();

    CLK_HSECmd(DISABLE);
    CLK_LSICmd(DISABLE);
    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

    CLK_ClockSwitchCmd(ENABLE);
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
    CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV2);

    CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
    DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

    CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, ENABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
```

```c
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
    GPIO_DeInit(GPIOB);
    GPIO_Init(GPIOB, GPIO_PIN_4, GPIO_MODE_OUT_OD_HIZ_FAST);
    GPIO_Init(GPIOB, GPIO_PIN_5, GPIO_MODE_OUT_OD_HIZ_FAST);
}


void I2C_setup(void)
{
    I2C_DeInit();
    I2C_Init(100000,
            BH1750_addr,
            I2C_DUTYCYCLE_2,
            I2C_ACK_CURR,
            I2C_ADDMODE_7BIT,
            (CLK_GetClockFreq() / 1000000));

     I2C_Cmd(ENABLE);
}


void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned int value)
{
    char tmp[5] = {0x20, 0x20, 0x20, 0x20, 0x20} ;

    tmp[0] = ((value / 10000) + 0x30);
    tmp[1] = (((value / 1000) % 10) + 0x30);
    tmp[2] = (((value / 100) % 10) + 0x30);
    tmp[3] = (((value / 10) % 10) + 0x30);
    tmp[4] = ((value % 10) + 0x30);

    LCD_goto(x_pos, y_pos);
    LCD_putstr(tmp);
}
```

BH1750.h

```c
#include "STM8S.h"


#define    BH1750_addr               0x46

#define   power_down                 0x00
#define   power_up                   0x01
#define   reset                      0x07
#define   cont_H_res_mode1           0x10
#define   cont_H_res_mode2           0x11
#define   cont_L_res_mode            0x13
#define   one_time_H_res_mode1       0x20
#define   one_time_H_res_mode2       0x21
#define   one_time_L_res_mode        0x23
```

```c
void BH1750_init(void);
void BH1750_write(unsigned char cmd);
unsigned int BH1750_read_word(void);
unsigned int get_lux_value(unsigned char mode, unsigned int delay_time);
```

BH1750.c

```c
#include "BH1750.h"


void BH1750_init(void)
{
    delay_ms(10);
    BH1750_write(power_down);
}


void BH1750_write(unsigned char cmd)
{
    I2C_GenerateSTART(ENABLE);
    while(!I2C_CheckEvent(I2C_EVENT_MASTER_MODE_SELECT));

    I2C_Send7bitAddress(BH1750_addr, I2C_DIRECTION_TX);
    while(!I2C_CheckEvent(I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

    I2C_SendData(cmd);
    while(!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTED));

    I2C_GenerateSTOP(ENABLE);
}


unsigned int BH1750_read_word(void)
{
    unsigned long value = 0x0000;
    unsigned char num_of_bytes = 0x02;
    unsigned char bytes[2] = {0x00, 0x00};

    while(I2C_GetFlagStatus(I2C_FLAG_BUSBUSY));

    I2C_GenerateSTART(ENABLE);
    while(!I2C_CheckEvent(I2C_EVENT_MASTER_MODE_SELECT));

    I2C_Send7bitAddress(BH1750_addr, I2C_DIRECTION_RX);
    while(!I2C_CheckEvent(I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

    while(num_of_bytes)
    {
        if(I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_RECEIVED))
        {
            if(num_of_bytes == 0)
            {
                I2C_AcknowledgeConfig(I2C_ACK_NONE);
                I2C_GenerateSTOP(ENABLE);
            }

            bytes[(num_of_bytes - 1)] = I2C_ReceiveData();
            num_of_bytes--;
        }
```

```c
    };

    value = ((bytes[1] << 8) | bytes[0]);

    return value;
}


unsigned int get_lux_value(unsigned char mode, unsigned int delay_time)
{
    unsigned long lux_value = 0x00;
    unsigned char dly = 0x00;
    unsigned char s = 0x08;

    while(s)
    {
        BH1750_write(power_up);
        BH1750_write(mode);
        lux_value += BH1750_read_word();
        for(dly = 0; dly < delay_time; dly += 1)
        {
            delay_ms(1);
        }
        BH1750_write(power_down);
        s--;
    }

    lux_value >>= 3;

    return ((unsigned int)lux_value);
}
```

Explanation

Firstly, both the CPU and peripheral clocks are set. Note the CPU is slower than last few examples. This has no significance.

```c
CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV2);
....
CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, ENABLE);
```

I2C I/Os are set as open drain outputs because they have external pull-up resistors that terminate the bus I/Os to VDD lines. **SCL** pin is always an output from host microcontroller's end however **SDA** pin's direction varies with reading and writing operations. This is automatically done by the I2C hardware.

```c
void GPIO_setup(void)
{
    GPIO_DeInit(GPIOB);
    GPIO_Init(GPIOB, GPIO_PIN_4, GPIO_MODE_OUT_OD_HIZ_FAST);
    GPIO_Init(GPIOB, GPIO_PIN_5, GPIO_MODE_OUT_OD_HIZ_FAST);
}
```

I2C setup has many parameters to set, firstly the I2C bus clock speed, then its own ID, clock duty cycle, address mode, acknowledgement type and clock speed of the peripheral. Here the own ID and slave ID are both set same because we are not using our STM8 as a slave. It doesn't matter. You can also set something else.

```c
void I2C_setup(void)
{
    I2C_DeInit();
    I2C_Init(100000,
             BH1750_addr,
             I2C_DUTYCYCLE_2,
             I2C_ACK_CURR,
             I2C_ADDMODE_7BIT,
             (CLK_GetClockFreq() / 1000000));

    I2C_Cmd(ENABLE);
}
```

If you have used compilers with built-in I2C library before then you may get some hiccups studying the following part. This is because those built-in libraries accomplish many tasks in the background that you never felt necessary. Flags and acknowledgments are such stuffs that are mostly automatically dealt by the compiler and ignore by most users. Personally, I had to struggle with these before settling this code. Another big difference is fact that the SPL's functions, their operations and nomenclatures for I2C are different than most I2C libraries one has seen before. Lastly, I2C examples with STM's SPL on the internet are rare and most of them demonstrate I2C example with 24 series EEPROMs only. I wanted to do something different and so I used BH1750 I2C digital sensor instead of repeating another EEPROM example.

```c
unsigned int BH1750_read_word(void)
{
    unsigned long value = 0x0000;
    unsigned char num_of_bytes = 0x02;
    unsigned char bytes[2] = {0x00, 0x00};

    while(I2C_GetFlagStatus(I2C_FLAG_BUSBUSY));

    I2C_GenerateSTART(ENABLE);
    while(!I2C_CheckEvent(I2C_EVENT_MASTER_MODE_SELECT));

    I2C_Send7bitAddress(BH1750_addr, I2C_DIRECTION_RX);
    while(!I2C_CheckEvent(I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

    while(num_of_bytes)
    {
        if(I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_RECEIVED))
        {
            if(num_of_bytes == 0)
            {
                I2C_AcknowledgeConfig(I2C_ACK_NONE);
                I2C_GenerateSTOP(ENABLE);
            }

            bytes[(num_of_bytes - 1)] = I2C_ReceiveData();
            num_of_bytes--;
        }
```

```
    };

    value = ((bytes[1] << 8) | bytes[0]);

    return value;
}
```

As with SPI, we need to check first if I2C hardware is free. We, then, initiate a I2C start condition and check master/slave mode selection. Next, we send out slave device's ID or address with read command, signalling that we wish to read from the slave. Again, a flag is checked before continuing. Here the sensor gives 16-bit light output data and so we need to extract two 8-bit data values. This is done in the *while* loop. At the end of data extraction process, we must generate stop as well as take care of acknowledgement. Finally, the two bytes are joined to form a word value representing light output.

The following function simplifies the task of determining lux value from the sensor. It selects mode of operation and latency. We will call this function in the main loop to extract average light value in lux.

```
unsigned int get_lux_value(unsigned char mode, unsigned int delay_time)
{
    unsigned long lux_value = 0x00;
    unsigned char dly = 0x00;
    unsigned char s = 0x08;

    while(s)
    {
        BH1750_write(power_up);
        BH1750_write(mode);
        lux_value += BH1750_read_word();
        for(dly = 0; dly < delay_time; dly += 1)
        {
            delay_ms(1);
        }
        BH1750_write(power_down);
        s--;
    }

    lux_value >>= 3;

    return ((unsigned int)lux_value);
}
```

Demo

Video link: https://youtu.be/bpwki1RCOXU.
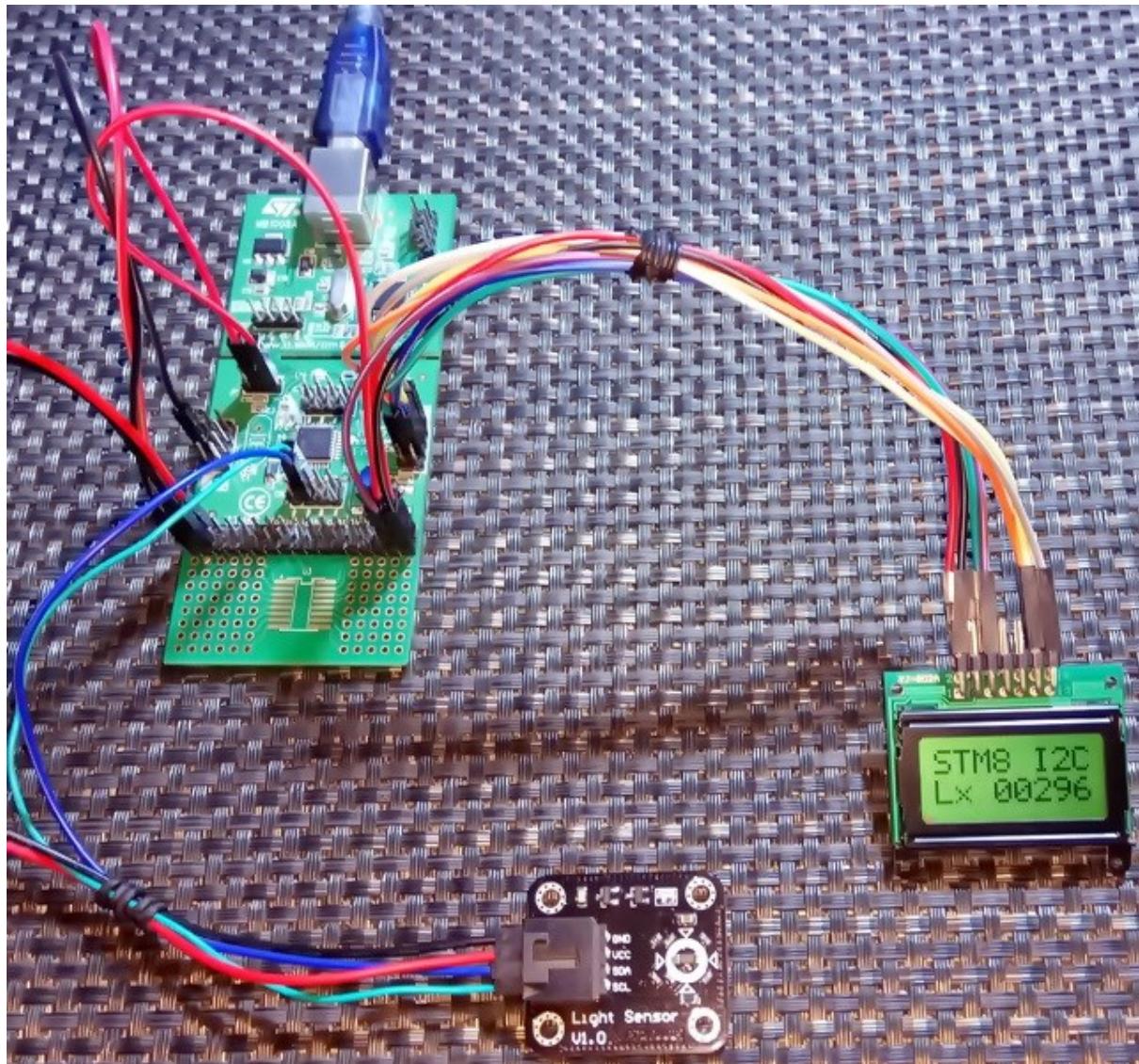
Software I2C – DS1307

Software-based I2C is not a big requirement in case of STM8s because STM8 chips have hardware I2C blocks. However, for some reason if we are unable to use hardware I2C blocks, we can implement software-based I2C by bit-banging ordinary GPIOs. I2C protocol is itself slow compared to SPI and other protocols and so implementing software-based I2C will not significantly affect communication speed and overall throughput in most applications. However, extra coding is required and this in turn adds some coding and resource overheads.

Hardware Connection
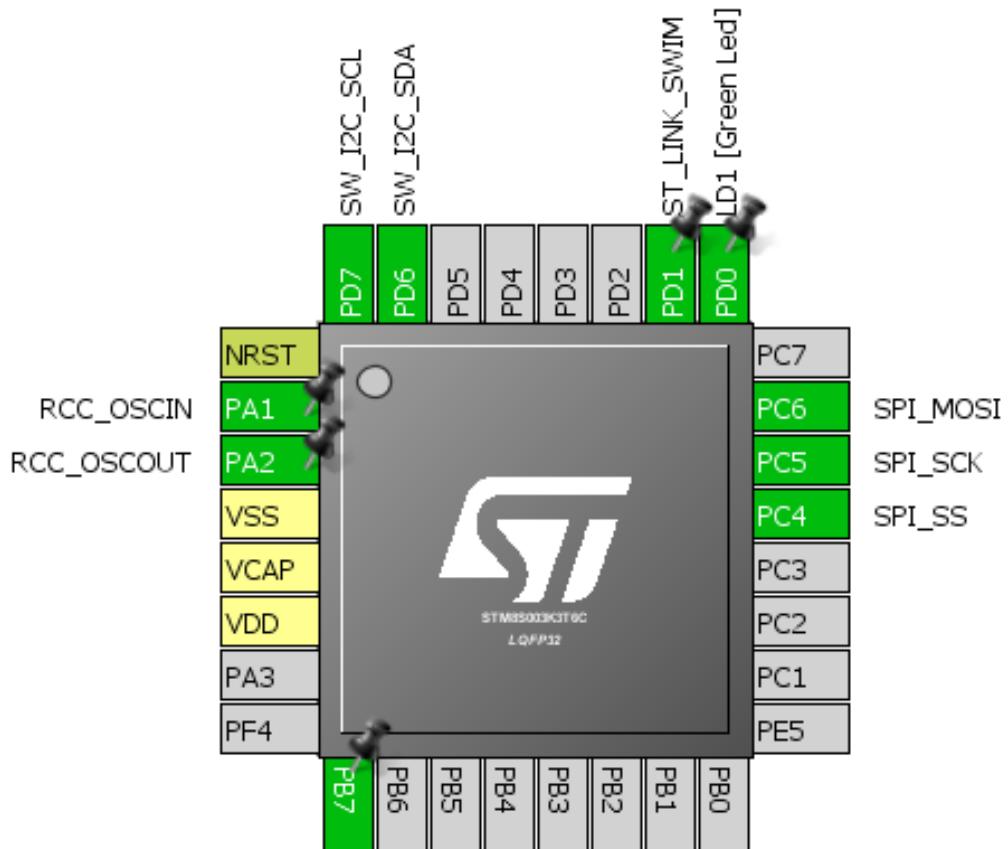
Code Example

SW_I2C.h

```c
#include "STM8S.h"


#define SW_I2C_port             GPIOD

#define SDA_pin                 GPIO_PIN_6
#define SCL_pin                 GPIO_PIN_7

#define SW_I2C_OUT()            do{GPIO_DeInit(SW_I2C_port);
GPIO_Init(SW_I2C_port, SDA_pin, GPIO_MODE_OUT_PP_LOW_FAST); GPIO_Init(SW_I2C_port,
SCL_pin, GPIO_MODE_OUT_PP_LOW_FAST);}while(0)

#define SW_I2C_IN()             do{GPIO_DeInit(SW_I2C_port);
GPIO_Init(SW_I2C_port, SDA_pin, GPIO_MODE_IN_FL_NO_IT); GPIO_Init(SW_I2C_port,
SCL_pin, GPIO_MODE_OUT_PP_LOW_FAST);}while(0)

#define SDA_HIGH()              GPIO_WriteHigh(SW_I2C_port, SDA_pin)
#define SDA_LOW()               GPIO_WriteLow(SW_I2C_port, SDA_pin)
#define SCL_HIGH()              GPIO_WriteHigh(SW_I2C_port, SCL_pin)
#define SCL_LOW()               GPIO_WriteLow(SW_I2C_port, SCL_pin)

#define SDA_IN()                GPIO_ReadInputPin(SW_I2C_port, SDA_pin)

#define I2C_ACK                 0xFF
#define I2C_NACK                0x00

#define I2C_timeout             1000


void SW_I2C_init(void);
void SW_I2C_start(void);
void SW_I2C_stop(void);
unsigned char SW_I2C_read(unsigned char ack);
void SW_I2C_write(unsigned char value);
void SW_I2C_ACK_NACK(unsigned char mode);
unsigned char SW_I2C_wait_ACK(void);
```

SW_I2C.c

```c
#include "SW_I2C.h"


void SW_I2C_init(void)
{
    SW_I2C_OUT();
    delay_ms(10);
    SDA_HIGH();
    SCL_HIGH();
}


void SW_I2C_start(void)
{
```

```c
    SW_I2C_OUT();
    SDA_HIGH();
    SCL_HIGH();
    delay_us(40);
    SDA_LOW();
    delay_us(40);
    SCL_LOW();
}


void SW_I2C_stop(void)
{
    SW_I2C_OUT();
    SDA_LOW();
    SCL_LOW();
    delay_us(40);
    SDA_HIGH();
    SCL_HIGH();
    delay_us(40);
}


unsigned char SW_I2C_read(unsigned char ack)
{
    unsigned char i = 0x08;
    unsigned char j = 0x00;

    SW_I2C_IN();

    while(i > 0x00)
    {
        SCL_LOW();
        delay_us(20);
        SCL_HIGH();
        delay_us(20);
        j <<= 1;

        if(SDA_IN() != 0x00)
        {
            j++;
        }

        delay_us(10);
        i--;
    };

    switch(ack)
    {
        case I2C_ACK:
        {
            SW_I2C_ACK_NACK(I2C_ACK);;
            break;
        }
        default:
        {
            SW_I2C_ACK_NACK(I2C_NACK);;
            break;
        }
    }
```

```c
        return j;
}


void SW_I2C_write(unsigned char value)
{
    unsigned char i = 0x08;

    SW_I2C_OUT();
    SCL_LOW();

    while(i > 0x00)
    {

        if(((value & 0x80) >> 0x07) != 0x00)
        {
            SDA_HIGH();
        }
        else
        {
            SDA_LOW();
        }


        value <<= 1;
        delay_us(20);
        SCL_HIGH();
        delay_us(20);
        SCL_LOW();
        delay_us(20);
        i--;
    };
}


void SW_I2C_ACK_NACK(unsigned char mode)
{
    SCL_LOW();
    SW_I2C_OUT();

    switch(mode)
    {
        case I2C_ACK:
        {
            SDA_LOW();
            break;
        }
        default:
        {
            SDA_HIGH();
            break;
        }
    }

    delay_us(20);
    SCL_HIGH();
    delay_us(20);
    SCL_LOW();
```

```c
}

unsigned char SW_I2C_wait_ACK(void)
{
    signed int timeout = 0x0000;

    SW_I2C_IN();

    SDA_HIGH();
    delay_us(10);
    SCL_HIGH();
    delay_us(10);

    while(SDA_IN() != 0x00)
    {
        timeout++;

        if(timeout > I2C_timeout)
        {
            SW_I2C_stop();
            return 1;
        }
    };

    SCL_LOW();

    return 0x00;
}
```

DS1307.h

```c
#include "SW_I2C.h"


#define sec_reg             0x00
#define min_reg             0x01
#define hr_reg              0x02
#define day_reg             0x03
#define date_reg            0x04
#define month_reg           0x05
#define year_reg            0x06
#define control_reg         0x07

#define DS1307_addr         0xD0
#define DS1307_WR           DS1307_addr
#define DS1307_RD           0xD1


void DS1307_init(void);
unsigned char DS1307_read(unsigned char address);
void DS1307_write(unsigned char address, unsigned char value);
unsigned char bcd_to_decimal(unsigned char value);
unsigned char decimal_to_bcd(unsigned char value);
void get_time(void);
void set_time(void);
```

DS1307.c

```c
#include "DS1307.h"


extern struct
{
    unsigned char s;
    unsigned char m;
    unsigned char h;
    unsigned char dy;
    unsigned char dt;
    unsigned char mt;
    unsigned char yr;
}time;


void DS1307_init(void)
{
    SW_I2C_init();
    DS1307_write(control_reg, 0x00);
}


unsigned char DS1307_read(unsigned char address)
{
    unsigned char value = 0x00;

    SW_I2C_start();
    SW_I2C_write(DS1307_WR);
    SW_I2C_write(address);

    SW_I2C_start();
    SW_I2C_write(DS1307_RD);
    value = SW_I2C_read(I2C_NACK);
    SW_I2C_stop();

    return value;
}


void DS1307_write(unsigned char address, unsigned char value)
{
    SW_I2C_start();
    SW_I2C_write(DS1307_WR);
    SW_I2C_write(address);
    SW_I2C_write(value);
    SW_I2C_stop();
}


unsigned char bcd_to_decimal(unsigned char value)
{
    return ((value & 0x0F) + (((value & 0xF0) >> 0x04) * 0x0A));
}


unsigned char decimal_to_bcd(unsigned char value)
{
```

```c
        return (((value / 0x0A) << 0x04) & 0xF0) | ((value % 0x0A) & 0x0F);
}


void get_time(void)
{
        time.s = DS1307_read(sec_reg);
        time.s = bcd_to_decimal(time.s);

        time.m = DS1307_read(min_reg);
        time.m = bcd_to_decimal(time.m);

        time.h = DS1307_read(hr_reg);
        time.h = bcd_to_decimal(time.h);

        time.dy = DS1307_read(day_reg);
        time.dy = bcd_to_decimal(time.dy);

        time.dt = DS1307_read(date_reg);
        time.dt = bcd_to_decimal(time.dt);

        time.mt = DS1307_read(month_reg);
        time.mt = bcd_to_decimal(time.mt);

        time.yr = DS1307_read(year_reg);
        time.yr = bcd_to_decimal(time.yr);
}


void set_time(void)
{
        time.s = decimal_to_bcd(time.s);
        DS1307_write(sec_reg, time.s);

        time.m = decimal_to_bcd(time.m);
        DS1307_write(min_reg, time.m);

        time.h = decimal_to_bcd(time.h);
        DS1307_write(hr_reg, time.h);

        time.dy = decimal_to_bcd(time.dy);
        DS1307_write(day_reg, time.dy);

        time.dt = decimal_to_bcd(time.dt);
        DS1307_write(date_reg, time.dt);

        time.mt = decimal_to_bcd(time.mt);
        DS1307_write(month_reg, time.mt);

        time.yr = decimal_to_bcd(time.yr);
        DS1307_write(year_reg, time.yr);
}
```

main.c

```c
#include "STM8S.h"
#include "lcd.h"
#include "SW_I2C.h"
```

```c
#include "DS1307.h"


#define Button_port        GPIOB
#define Button_pin         GPIO_PIN_7


struct
{
    unsigned char s;
    unsigned char m;
    unsigned char h;
    unsigned char dy;
    unsigned char dt;
    unsigned char mt;
    unsigned char yr;
}time;

bool set = FALSE;

unsigned char menu = 0;
unsigned char data_value;


void clock_setup(void);
void GPIO_setup(void);
void show_value(unsigned char x_pos, unsigned char y_pos, unsigned char value);
void display_time(void);
void setup_time(void);
unsigned char adjust(unsigned char x_pos, unsigned char y_pos, signed char
value_max, signed char value_min, signed char value);


void main(void)
{
    clock_setup();
    GPIO_setup();

    DS1307_init();

    LCD_init();
    LCD_clear_home();

    LCD_goto(0, 0);
    LCD_putstr("STM8 SW-I2C Test");

    while(1)
    {
        if((GPIO_ReadInputPin(Button_port, Button_pin) == FALSE) && (set == FALSE))
        {
            delay_ms(1000);
            if(GPIO_ReadInputPin(Button_port, Button_pin) == FALSE)
            {
                while(GPIO_ReadInputPin(Button_port, Button_pin) == FALSE);
                delay_ms(1000);

                menu = 0;
                set = TRUE;
            }
```

```c
        }

        if(set)
        {
                setup_time();
        }
        else
        {
                get_time();
                display_time();
        }
    };
}


void clock_setup(void)
{
        CLK_DeInit();

        CLK_HSECmd(DISABLE);
        CLK_LSICmd(DISABLE);
        CLK_HSICmd(ENABLE);
        while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

        CLK_ClockSwitchCmd(ENABLE);
        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV2);
        CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

        CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
        DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
        GPIO_DeInit(GPIOB);
        GPIO_Init(Button_port, Button_pin, GPIO_MODE_IN_PU_NO_IT);
}


void show_value(unsigned char x_pos, unsigned char y_pos, unsigned char value)
{
    char ch[0x03] = {0x20, 0x20, '\0'};

    ch[0] = (((value / 10) % 10) + 0x30);
    ch[1] = ((value % 10) + 0x30);

    LCD_goto(x_pos, y_pos);
    LCD_putstr(ch);
}
```

```c
void display_time(void)
{
	show_value(4, 1, time.h);
	show_value(7, 1, time.m);
	show_value(10, 1, time.s);

	LCD_goto(6, 1);
	LCD_putchar(':');
	LCD_goto(9, 1);
	LCD_putchar(':');
	delay_ms(100);
}


void setup_time(void)
{
	switch(menu)
	{
		case 0:
		{
			time.h = adjust(4, 1, 23, 0, time.h);
			break;
		}
		case 1:
		{
			time.m = adjust(7, 1, 59, 0, time.m);
			break;
		}
		case 2:
		{
			time.s = adjust(10, 1, 59, 0, time.s);
			break;
		}
	}
}


unsigned char adjust(unsigned char x_pos, unsigned char y_pos, signed char
value_max, signed char value_min, signed char value)
{
	if(GPIO_ReadInputPin(Button_port, Button_pin) == FALSE)
	{
		delay_ms(900);

		if(GPIO_ReadInputPin(Button_port, Button_pin) == FALSE)
		{
			while(GPIO_ReadInputPin(Button_port, Button_pin) == FALSE);
			menu++;

			if(menu >= 3)
			{
				LCD_goto(3, 1);
				LCD_putchar(' ');
				LCD_goto(12, 1);
				LCD_putchar(' ');
				set_time();
				set = FALSE;
```

```
                              return;
                    }
            }
      }

      else
      {
            value++;

            if(value > value_max)
            {
                  value = value_min;
            }
      }

      LCD_goto(3, 1);
      LCD_putchar('<');
      LCD_goto(12, 1);
      LCD_putchar('>');

      LCD_goto(x_pos, y_pos);
      LCD_putstr("  ");
      delay_ms(90);

      show_value(x_pos, y_pos, value);
      delay_ms(90);

      return value;
}
```

Explanation

SW_I2C.h and **SW_I2C.c** files translate the entire I2C communication operations with the manipulation of ordinary GPIOs. Any GPIO pin pair can be used for software I2C. Just define the software I2C port, and the serial data (SDA) and serial clock (SDA) pins:

```
#define SW_I2C_port              GPIOD

#define SDA_pin                  GPIO_PIN_6
#define SCL_pin                  GPIO_PIN_7
```

The functions in these files are self-explanatory and so I'm not going to explain them here.

The rest of the code is about using the software I2C library with DS1307 real time clock (RTC) chip to make a real time clock.

Demo

Video link: https://www.youtube.com/watch?v=OeafNmbCnZ8.
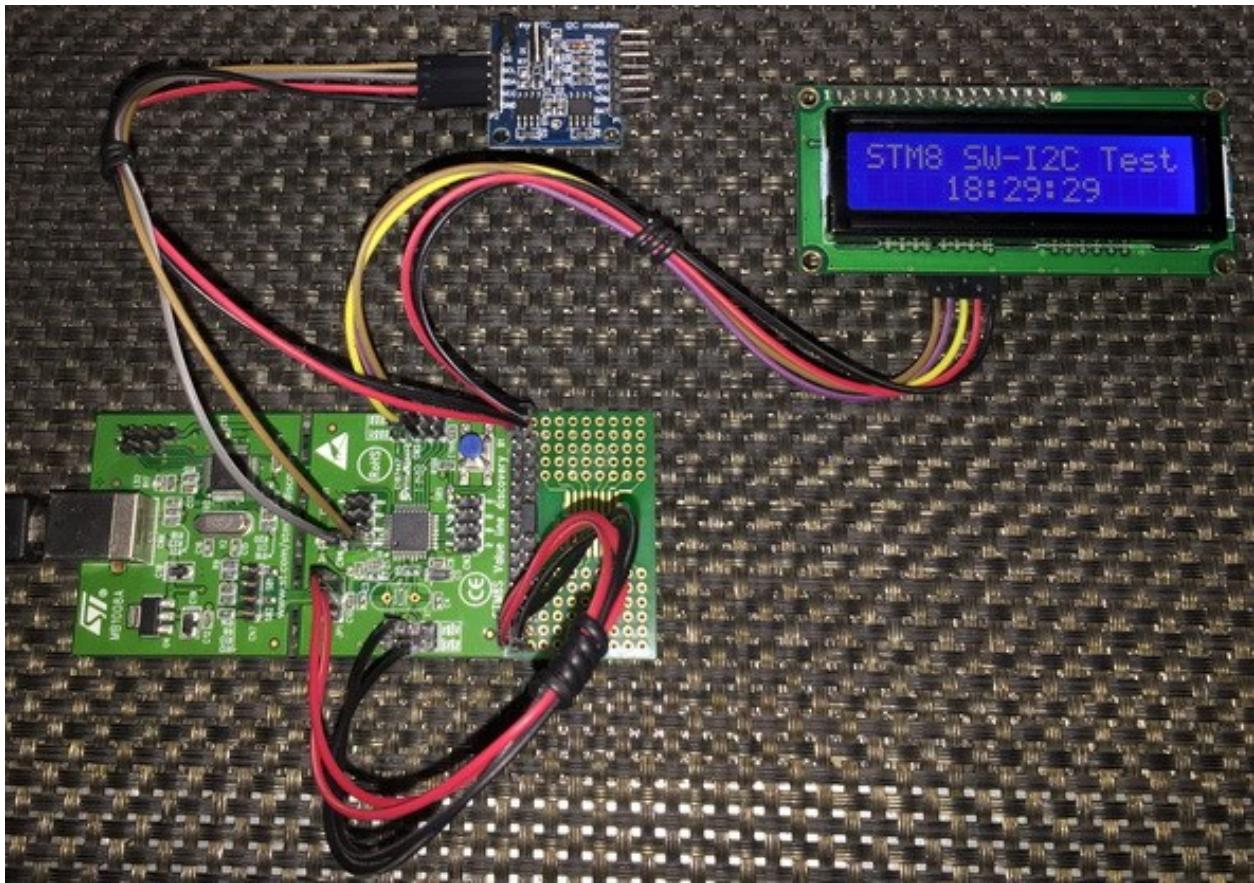
Driving LCD with I2C Interface

Recently ST released an 8 pin STM8 micro called STM8S001J3. This is a tiny but powerful microcontroller with lot of potentials. There are several other STM8 micros that have low pin counts and there are also times when we have to complete projects with least possible hardware and at low costs. We, then, don't have the luxury of doing things in obvious ways. We need to use some supporting external hardware to get things done in such situations. Typically to interface a LCD in 4-bit mode with a micro it requires at least six GPIO pins. However, we can avoid so by using I2C port expanders like PCF8574. Since it is I2C based, it will just occupy two pins instead of six.
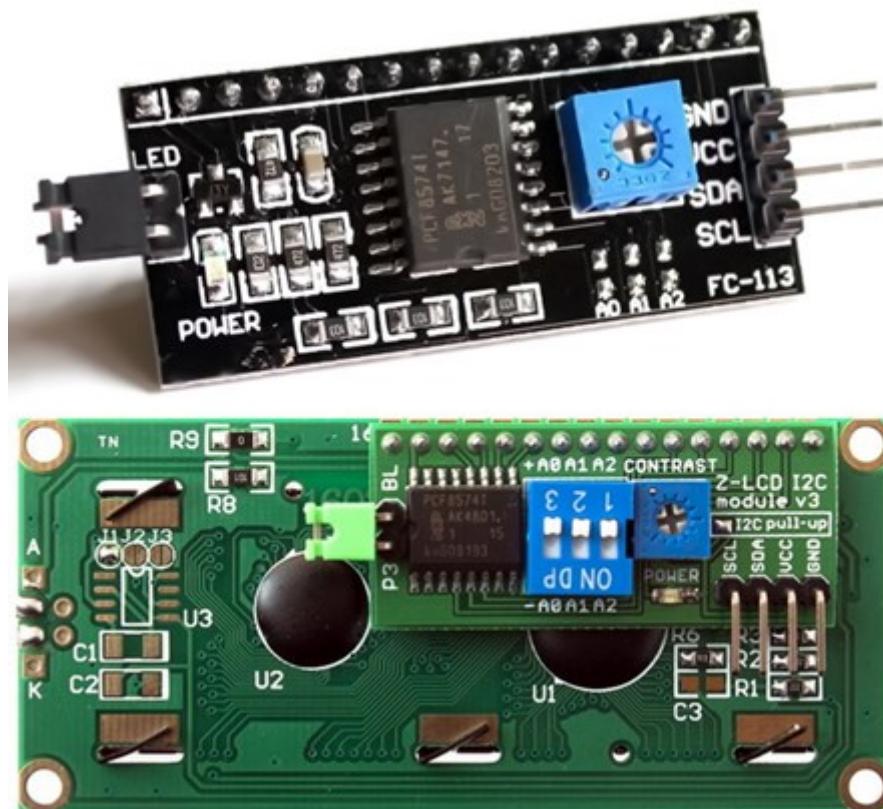
Owing to its popularity and simplicity, in the market there is a cheap and widely-available 2-wire LCD module based on PCF8574T. There are a few advantages of this module. Firstly, it is based on a PCF8574T chip that is made by NXP (*a.k.a* Philips). NXP happens to be the founder of I2C communication protocol and so the chip is well documented in terms of I2C communication. Secondly, there are three external address selection bits which can be used to address multiple LCDs coexisting on the same I2C bus. Lastly the module is compact and readily plug-and-playable. Additionally, the possibility of LCD data corruption due to noise and EMI is significantly reduced.

However, I2C communication is itself a slow communication and so interfacing displays via I2C may yield in slower performances and slower refresh rates. Additionally, extra coding and therefore extra memory spaces are needed.

Hardware Connection
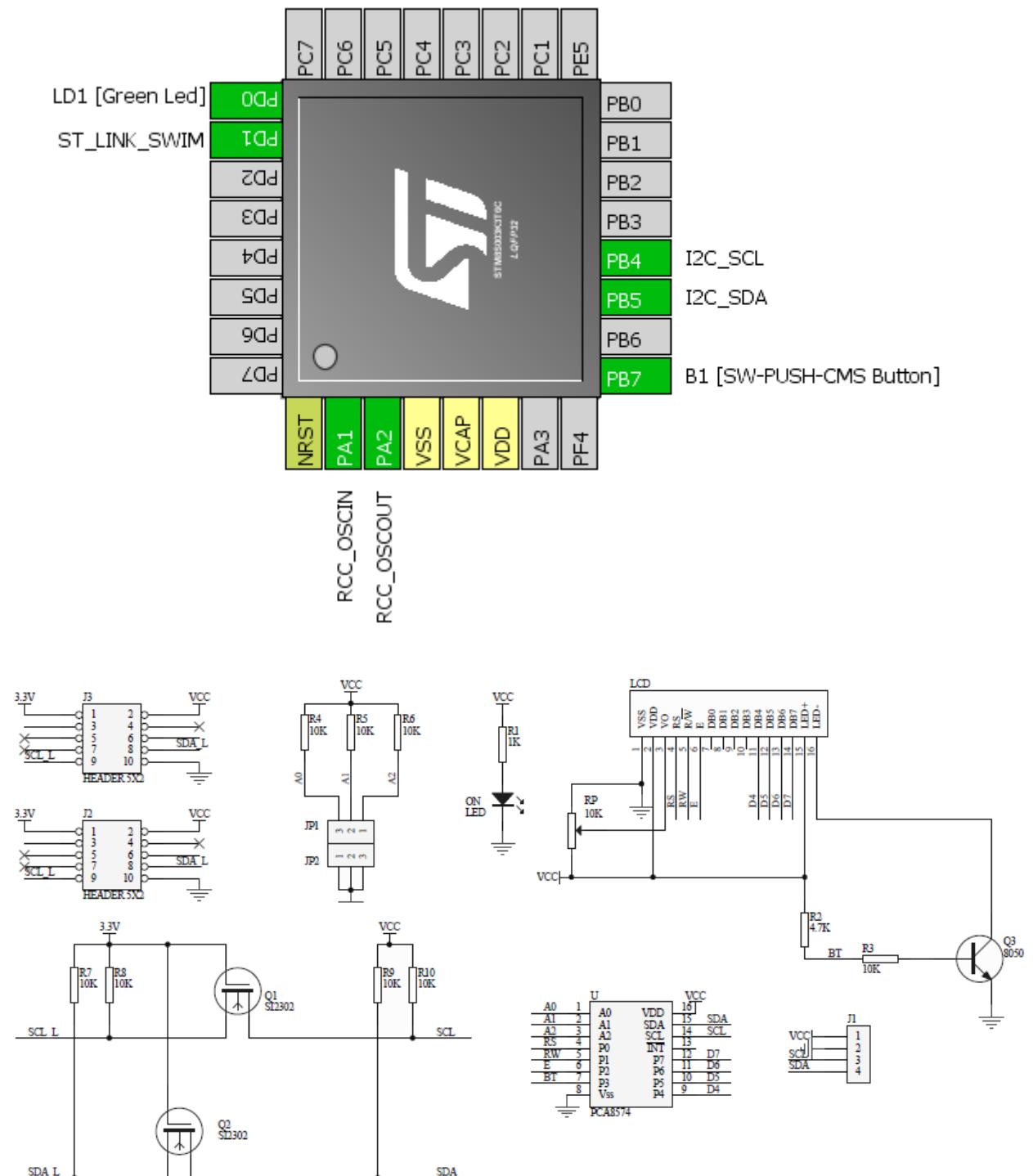
Code Example

PCF8574.h

```c
#include "stm8s.h"


#define I2C_PORT                            GPIOB

#define SDA_pin                             GPIO_PIN_5
#define SCL_pin                             GPIO_PIN_4

#define PCF8574_address                     0x4E



void I2C_GPIO_setup(void);
void I2C_setup(void);
void PCF8574_init(void);
unsigned char PCF8574_read(void);
void PCF8574_write(unsigned char data_byte);
```

PCF8574.c

```c
#include "PCF8574.h"


void I2C_GPIO_setup(void)
{
    GPIO_Init(I2C_PORT,
              ((GPIO_Pin_TypeDef)(SCL_pin | SDA_pin)),
              GPIO_MODE_OUT_PP_HIGH_FAST);
}


void I2C_setup(void)
{
    I2C_DeInit();
    I2C_Init(100000,
                PCF8574_address,
                I2C_DUTYCYCLE_2,
                I2C_ACK_CURR,
                I2C_ADDMODE_7BIT,
                (CLK_GetClockFreq() / 1000000));
    I2C_Cmd(ENABLE);
}


void PCF8574_init(void)
{
    I2C_GPIO_setup();
    I2C_setup();
}


unsigned char PCF8574_read(void)
{
    unsigned char port_byte = 0x00;
```

```c
    unsigned char num_of_bytes = 0x01;

    while(I2C_GetFlagStatus(I2C_FLAG_BUSBUSY));

    I2C_GenerateSTART(ENABLE);
    while(!I2C_CheckEvent(I2C_EVENT_MASTER_MODE_SELECT));

    I2C_Send7bitAddress(PCF8574_address, I2C_DIRECTION_RX);
    while(!I2C_CheckEvent(I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

    while(num_of_bytes)
    {
        if(I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_RECEIVED))
        {
            if(num_of_bytes == 0)
            {
                I2C_AcknowledgeConfig(I2C_ACK_NONE);
                I2C_GenerateSTOP(ENABLE);
            }

            port_byte = I2C_ReceiveData();

            num_of_bytes--;
        }
    };

    return port_byte;
}


void PCF8574_write(unsigned char data_byte)
{
    I2C_GenerateSTART(ENABLE);
    while(!I2C_CheckEvent(I2C_EVENT_MASTER_MODE_SELECT));

    I2C_Send7bitAddress(PCF8574_address, I2C_DIRECTION_TX);
    while(!I2C_CheckEvent(I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

    I2C_SendData(data_byte);
    while(!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTED));

    I2C_GenerateSTOP(ENABLE);
}
```

lcd.h

```c
#include "stm8s.h"
#include "PCF8574.h"


#define clear_display                          0x01
#define goto_home                              0x02

#define cursor_direction_inc        (0x04 | 0x02)
#define cursor_direction_dec        (0x04 | 0x00)
#define display_shift               (0x04 | 0x01)
#define display_no_shift            (0x04 | 0x00)
```

```c
#define display_on                          (0x08 | 0x04)
#define display_off                         (0x08 | 0x02)
#define cursor_on                           (0x08 | 0x02)
#define cursor_off                          (0x08 | 0x00)
#define blink_on                            (0x08 | 0x01)
#define blink_off                           (0x08 | 0x00)

#define _8_pin_interface                    (0x20 | 0x10)
#define _4_pin_interface                    (0x20 | 0x00)
#define _2_row_display                      (0x20 | 0x08)
#define _1_row_display                      (0x20 | 0x00)
#define _5x10_dots                          (0x20 | 0x40)
#define _5x7_dots                           (0x20 | 0x00)

#define BL_ON                                          1
#define BL_OFF                                         0

#define dly                                            2

#define DAT                                            1
#define CMD                                            0


extern unsigned char bl_state;
extern unsigned char data_value;


void LCD_init(void);
void LCD_toggle_EN(void);
void LCD_send(unsigned char value, unsigned char mode);
void LCD_4bit_send(unsigned char lcd_data);
void LCD_putstr(char *lcd_string);
void LCD_putchar(char char_data);
void LCD_clear_home(void);
void LCD_goto(unsigned char x_pos, unsigned char y_pos);
```

lcd.c

```c
#include "lcd.h"

void LCD_init(void)
{
    PCF8574_init();
    delay_ms(100);

    bl_state = BL_ON;
    data_value = 0x04;
    PCF8574_write(data_value);

    delay_ms(10);

    LCD_send(0x33, CMD);
    LCD_send(0x32, CMD);

    LCD_send((_4_pin_interface | _2_row_display | _5x7_dots), CMD);
    LCD_send((display_on | cursor_off | blink_off), CMD);
    LCD_send((clear_display), CMD);
```

```c
    LCD_send((cursor_direction_inc | display_no_shift), CMD);
}


void LCD_toggle_EN(void)
{
    data_value |= 0x04;
    PCF8574_write(data_value);
    delay_ms(dly);
    data_value &= 0xF9;
    PCF8574_write(data_value);
    delay_ms(dly);
}


void LCD_send(unsigned char value, unsigned char mode)
{
    switch(mode)
    {
        case CMD:
        {
            data_value &= 0xF4;
            break;
        }
        case DAT:
        {
            data_value |= 0x01;
            break;
        }
    }

    switch(bl_state)
    {
        case BL_ON:
        {
            data_value |= 0x08;
            break;
        }
        case BL_OFF:
        {
            data_value &= 0xF7;
            break;
        }
    }

    PCF8574_write(data_value);
    LCD_4bit_send(value);
    delay_ms(dly);
}


void LCD_4bit_send(unsigned char lcd_data)
{
    unsigned char temp = 0x00;

    temp = (lcd_data & 0xF0);
    data_value &= 0x0F;
    data_value |= temp;
    PCF8574_write(data_value);
```

```
        LCD_toggle_EN();

        temp = (lcd_data & 0x0F);
        temp <<= 0x04;
        data_value &= 0x0F;
        data_value |= temp;
        PCF8574_write(data_value);
        LCD_toggle_EN();
}


void LCD_putstr(char *lcd_string)
{
        do
        {
                LCD_putchar(*lcd_string++);
        }while(*lcd_string != '\0') ;
}


void LCD_putchar(char char_data)
{
        if((char_data >= 0x20) && (char_data <= 0x7F))
        {
                LCD_send(char_data, DAT);
        }
}


void LCD_clear_home(void)
{
        LCD_send(clear_display, CMD);
        LCD_send(goto_home, CMD);
}


void LCD_goto(unsigned char x_pos,unsigned char y_pos)
{
        if(y_pos == 0)
        {
                LCD_send((0x80 | x_pos), CMD);
        }
        else
        {
                LCD_send((0x80 | 0x40 | x_pos), CMD);
        }
}
```

main.c

```
#include "STM8S.h"
#include "lcd.h"


unsigned char bl_state;
unsigned char data_value;
```

```c
void clock_setup(void);
void GPIO_setup(void);
void show_value(unsigned char value);


void main(void)
{
  const char txt1[] = {"MICROARENA"};
  const char txt2[] = {"SShahryiar"};
  const char txt3[] = {"STM8S003K3"};
  const char txt4[] = {"Discovery!"};

  unsigned char s = 0x00;

  clock_setup();
  GPIO_setup();
  LCD_init();

    LCD_clear_home();

    LCD_goto(3, 0);
    LCD_putstr(txt1);
    LCD_goto(3, 1);
    LCD_putstr(txt2);
    delay_ms(2600);

    LCD_clear_home();

    for(s = 0; s < 10; s++)
    {
        LCD_goto((3 + s), 0);
        LCD_putchar(txt3[s]);
        delay_ms(60);
    }
    for(s = 0; s < 10; s++)
    {
        LCD_goto((3 + s), 1);
        LCD_putchar(txt4[s]);
        delay_ms(60);
    }
  delay_ms(2600);

    s = 0;
    LCD_clear_home();

    LCD_goto(3, 0);
    LCD_putstr(txt1);

    while(1)
    {
        show_value(s);
        s++;
        delay_ms(200);
    };
}


void clock_setup(void)
{
```

```c
    CLK_DeInit();

    CLK_HSECmd(DISABLE);
    CLK_LSICmd(DISABLE);

    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

    CLK_ClockSwitchCmd(ENABLE);
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
    CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

    CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
    DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

    CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, ENABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
    GPIO_DeInit(I2C_PORT);
}


void show_value(unsigned char value)
{
    char ch = 0x00;

    ch = ((value / 100) + 0x30);
    LCD_goto(6, 1);
    LCD_putchar(ch);

    ch = (((value / 10) % 10) + 0x30);
    LCD_goto(7, 1);
    LCD_putchar(ch);

    ch = ((value % 10) + 0x30);
    LCD_goto(8, 1);
    LCD_putchar(ch);
}
```

Explanation

The LCD example is nothing different from other LCD examples except for the I2C implementation part. For that we need to code for the PCF8574 I2C port expander first. SDA and SCL pins are initialized in the beginning and this is followed by I2C hardware initialization.

```
void I2C_GPIO_setup(void)
{
    GPIO_Init(I2C_PORT,
              ((GPIO_Pin_TypeDef)(SCL_pin | SDA_pin)),
              GPIO_MODE_OUT_PP_HIGH_FAST);
}


void I2C_setup(void)
{
    I2C_DeInit();
    I2C_Init(100000,
                PCF8574_address,
                I2C_DUTYCYCLE_2,
                I2C_ACK_CURR,
                I2C_ADDMODE_7BIT,
                (CLK_GetClockFreq() / 1000000));
    I2C_Cmd(ENABLE);
}
```

Don't forget to enable I2C peripheral clock:

```
CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);
....
CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, ENABLE);
```

We don't need to read the I2C port expander, just need to write it and so we just need the write operation part only. I have still included the reading part.

```
void PCF8574_write(unsigned char data_byte)
{
   I2C_GenerateSTART(ENABLE);
   while(!I2C_CheckEvent(I2C_EVENT_MASTER_MODE_SELECT));

   I2C_Send7bitAddress(PCF8574_address, I2C_DIRECTION_TX);
   while(!I2C_CheckEvent(I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

   I2C_SendData(data_byte);
   while(!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTED));

   I2C_GenerateSTOP(ENABLE);
}
```

The rest of the code is just about coding the LCD as we would do with ordinary connections and running it. Therefore, the only theme here is to use PCF8574 to handle the I/O operations for driving the LCD connected with it.

Demo

Video link: https://www.youtube.com/watch?v=2Z1apI0u8BA.
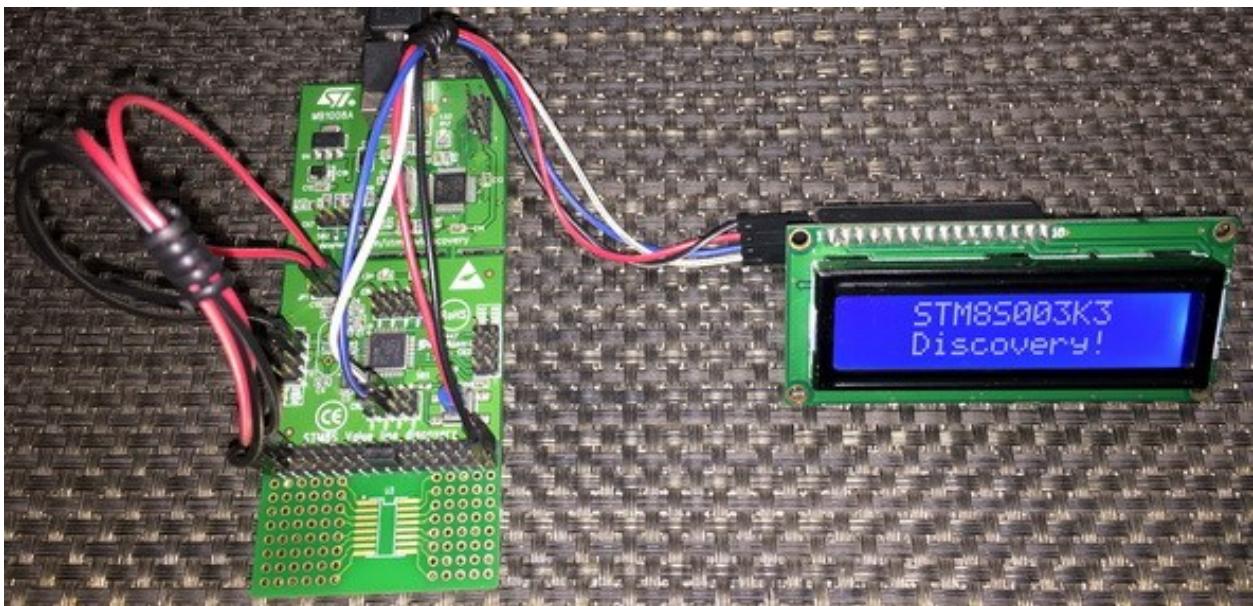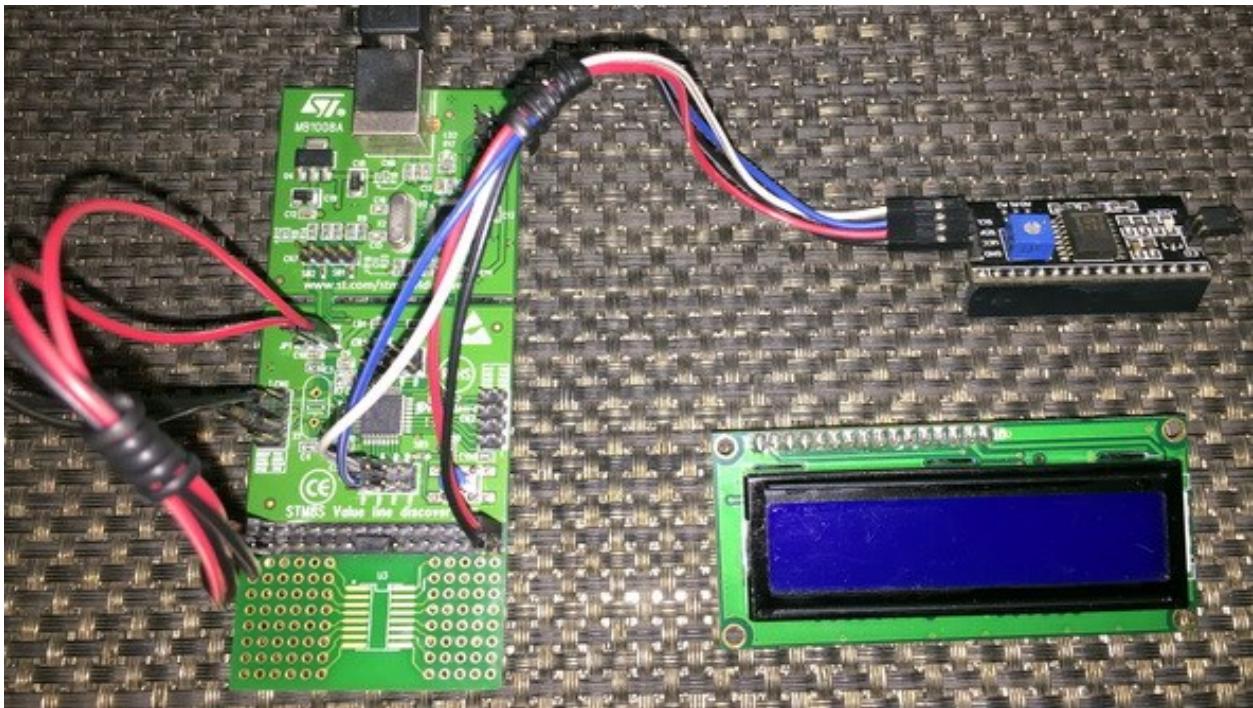
OLED Display – SSD1306/SSD1309

At present LED technology is quickly conquering the world with new innovations and wonders. Organic LED (OLED) displays are just one example of such wonders. In the market, there are many OLED display modules available, varying in size and resolution. Most of them are based on SSD1306/SSD1309 and have a resolution of 128x64 bits. Most of them are either fully monochrome or dual colour-based. There are other OLED displays with full RGB colour support. SSD1306 display chip supports both I2C and SPI communication methods. It is nice to integrate a tiny and stylish graphical display as such in projects.
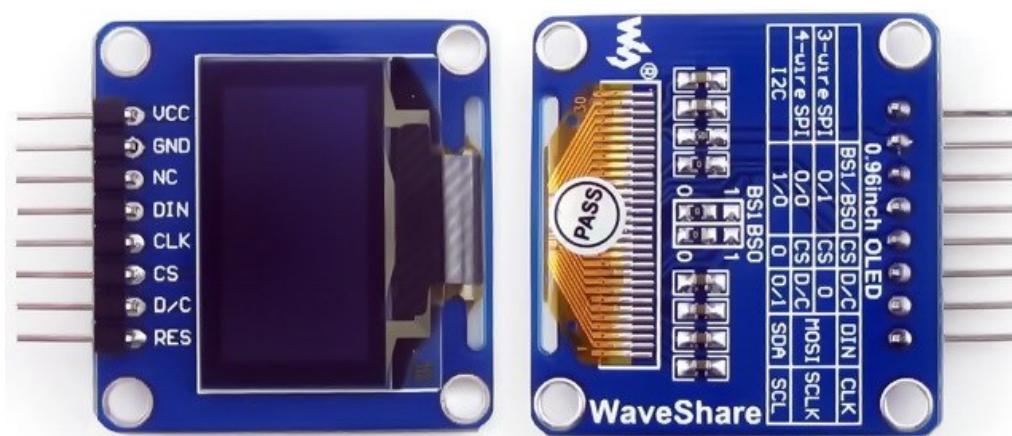
Hardware Connection
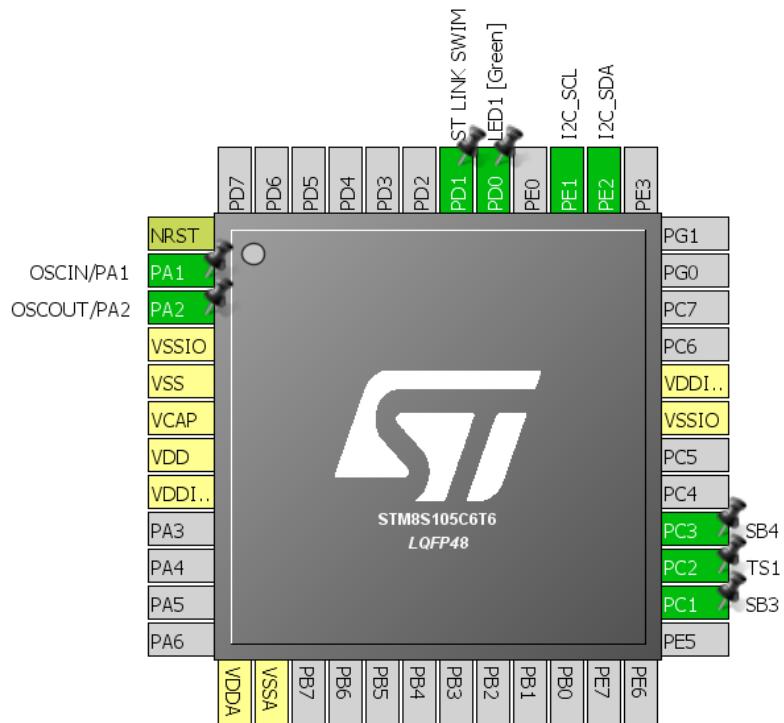
Code Example

font.h

```c
static const unsigned char font_regular[92][6] =
{
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00,    // sp
    0x00, 0x00, 0x00, 0x2f, 0x00, 0x00,    // !
    0x00, 0x00, 0x07, 0x00, 0x07, 0x00,    // "
    0x00, 0x14, 0x7f, 0x14, 0x7f, 0x14,    // #
    0x00, 0x24, 0x2a, 0x7f, 0x2a, 0x12,    // $
    0x00, 0x62, 0x64, 0x08, 0x13, 0x23,    // %
    0x00, 0x36, 0x49, 0x55, 0x22, 0x50,    // &
    0x00, 0x00, 0x05, 0x03, 0x00, 0x00,    // '
    0x00, 0x00, 0x1c, 0x22, 0x41, 0x00,    // (
    0x00, 0x00, 0x41, 0x22, 0x1c, 0x00,    // )
    0x00, 0x14, 0x08, 0x3E, 0x08, 0x14,    // *
    0x00, 0x08, 0x08, 0x3E, 0x08, 0x08,    // +
    0x00, 0x00, 0x00, 0xA0, 0x60, 0x00,    // ,
    0x00, 0x08, 0x08, 0x08, 0x08, 0x08,    // -
    0x00, 0x00, 0x60, 0x60, 0x00, 0x00,    // .
    0x00, 0x20, 0x10, 0x08, 0x04, 0x02,    // /
    0x00, 0x3E, 0x51, 0x49, 0x45, 0x3E,    // 0
    0x00, 0x00, 0x42, 0x7F, 0x40, 0x00,    // 1
    0x00, 0x42, 0x61, 0x51, 0x49, 0x46,    // 2
    0x00, 0x21, 0x41, 0x45, 0x4B, 0x31,    // 3
    0x00, 0x18, 0x14, 0x12, 0x7F, 0x10,    // 4
    0x00, 0x27, 0x45, 0x45, 0x45, 0x39,    // 5
    0x00, 0x3C, 0x4A, 0x49, 0x49, 0x30,    // 6
    0x00, 0x01, 0x71, 0x09, 0x05, 0x03,    // 7
    0x00, 0x36, 0x49, 0x49, 0x49, 0x36,    // 8
    0x00, 0x06, 0x49, 0x49, 0x29, 0x1E,    // 9
    0x00, 0x00, 0x36, 0x36, 0x00, 0x00,    // :
    0x00, 0x00, 0x56, 0x36, 0x00, 0x00,    // ;
    0x00, 0x08, 0x14, 0x22, 0x41, 0x00,    // <
    0x00, 0x14, 0x14, 0x14, 0x14, 0x14,    // =
    0x00, 0x00, 0x41, 0x22, 0x14, 0x08,    // >
    0x00, 0x02, 0x01, 0x51, 0x09, 0x06,    // ?
    0x00, 0x32, 0x49, 0x59, 0x51, 0x3E,    // @
    0x00, 0x7C, 0x12, 0x11, 0x12, 0x7C,    // A
    0x00, 0x7F, 0x49, 0x49, 0x49, 0x36,    // B
    0x00, 0x3E, 0x41, 0x41, 0x41, 0x22,    // C
    0x00, 0x7F, 0x41, 0x41, 0x22, 0x1C,    // D
    0x00, 0x7F, 0x49, 0x49, 0x49, 0x41,    // E
    0x00, 0x7F, 0x09, 0x09, 0x09, 0x01,    // F
    0x00, 0x3E, 0x41, 0x49, 0x49, 0x7A,    // G
    0x00, 0x7F, 0x08, 0x08, 0x08, 0x7F,    // H
    0x00, 0x00, 0x41, 0x7F, 0x41, 0x00,    // I
    0x00, 0x20, 0x40, 0x41, 0x3F, 0x01,    // J
    0x00, 0x7F, 0x08, 0x14, 0x22, 0x41,    // K
    0x00, 0x7F, 0x40, 0x40, 0x40, 0x40,    // L
    0x00, 0x7F, 0x02, 0x0C, 0x02, 0x7F,    // M
    0x00, 0x7F, 0x04, 0x08, 0x10, 0x7F,    // N
    0x00, 0x3E, 0x41, 0x41, 0x41, 0x3E,    // O
    0x00, 0x7F, 0x09, 0x09, 0x09, 0x06,    // P
    0x00, 0x3E, 0x41, 0x51, 0x21, 0x5E,    // Q
    0x00, 0x7F, 0x09, 0x19, 0x29, 0x46,    // R
    0x00, 0x46, 0x49, 0x49, 0x49, 0x31,    // S
```

```c
    0x00, 0x01, 0x01, 0x7F, 0x01, 0x01,    // T
    0x00, 0x3F, 0x40, 0x40, 0x40, 0x3F,    // U
    0x00, 0x1F, 0x20, 0x40, 0x20, 0x1F,    // V
    0x00, 0x3F, 0x40, 0x38, 0x40, 0x3F,    // W
    0x00, 0x63, 0x14, 0x08, 0x14, 0x63,    // X
    0x00, 0x07, 0x08, 0x70, 0x08, 0x07,    // Y
    0x00, 0x61, 0x51, 0x49, 0x45, 0x43,    // Z
    0x00, 0x00, 0x7F, 0x41, 0x41, 0x00,    // [
    0x00, 0x02, 0x04, 0x08, 0x10, 0x20,    // \92
    0x00, 0x00, 0x41, 0x41, 0x7F, 0x00,    // ]
    0x00, 0x04, 0x02, 0x01, 0x02, 0x04,    // ^
    0x00, 0x40, 0x40, 0x40, 0x40, 0x40,    // _
    0x00, 0x00, 0x01, 0x02, 0x04, 0x00,    // '
    0x00, 0x20, 0x54, 0x54, 0x54, 0x78,    // a
    0x00, 0x7F, 0x48, 0x44, 0x44, 0x38,    // b
    0x00, 0x38, 0x44, 0x44, 0x44, 0x20,    // c
    0x00, 0x38, 0x44, 0x44, 0x48, 0x7F,    // d
    0x00, 0x38, 0x54, 0x54, 0x54, 0x18,    // e
    0x00, 0x08, 0x7E, 0x09, 0x01, 0x02,    // f
    0x00, 0x18, 0xA4, 0xA4, 0xA4, 0x7C,    // g
    0x00, 0x7F, 0x08, 0x04, 0x04, 0x78,    // h
    0x00, 0x00, 0x44, 0x7D, 0x40, 0x00,    // i
    0x00, 0x40, 0x80, 0x84, 0x7D, 0x00,    // j
    0x00, 0x7F, 0x10, 0x28, 0x44, 0x00,    // k
    0x00, 0x00, 0x41, 0x7F, 0x40, 0x00,    // l
    0x00, 0x7C, 0x04, 0x18, 0x04, 0x78,    // m
    0x00, 0x7C, 0x08, 0x04, 0x04, 0x78,    // n
    0x00, 0x38, 0x44, 0x44, 0x44, 0x38,    // o
    0x00, 0xFC, 0x24, 0x24, 0x24, 0x18,    // p
    0x00, 0x18, 0x24, 0x24, 0x18, 0xFC,    // q
    0x00, 0x7C, 0x08, 0x04, 0x04, 0x08,    // r
    0x00, 0x48, 0x54, 0x54, 0x54, 0x20,    // s
    0x00, 0x04, 0x3F, 0x44, 0x40, 0x20,    // t
    0x00, 0x3C, 0x40, 0x40, 0x20, 0x7C,    // u
    0x00, 0x1C, 0x20, 0x40, 0x20, 0x1C,    // v
    0x00, 0x3C, 0x40, 0x30, 0x40, 0x3C,    // w
    0x00, 0x44, 0x28, 0x10, 0x28, 0x44,    // x
    0x00, 0x1C, 0xA0, 0xA0, 0xA0, 0x7C,    // y
    0x00, 0x44, 0x64, 0x54, 0x4C, 0x44,    // z
    0x14, 0x14, 0x14, 0x14, 0x14, 0x14     // horiz lines
};
```

SSD1306.h

```c
#include "STM8S.h"


#define I2C_PORT                                GPIOE

#define SCL_pin                                 GPIO_PIN_1
#define SDA_pin                                 GPIO_PIN_2


#define SSD1306_I2C_Address                     0x78


#define DAT                                     0x60
#define CMD                                     0x00


#define Set_Lower_Column_Start_Address_CMD      0x00
```

```c
#define Set_Higher_Column_Start_Address_CMD        0x10
#define Set_Memory_Addressing_Mode_CMD             0x20
#define Set_Column_Address_CMD                     0x21
#define Set_Page_Address_CMD                       0x22
#define Set_Display_Start_Line_CMD                 0x40
#define Set_Contrast_Control_CMD                   0x81
#define Set_Charge_Pump_CMD                        0x8D
#define Set_Segment_Remap_CMD                      0xA0
#define Set_Entire_Display_ON_CMD                  0xA4
#define Set_Normal_or_Inverse_Display_CMD          0xA6
#define Set_Multiplex_Ratio_CMD                    0xA8
#define Set_Display_ON_or_OFF_CMD                  0xAE
#define Set_Page_Start_Address_CMD                 0xB0
#define Set_COM_Output_Scan_Direction_CMD          0xC0
#define Set_Display_Offset_CMD                     0xD3
#define Set_Display_Clock_CMD                      0xD5
#define Set_Pre_charge_Period_CMD                  0xD9
#define Set_Common_HW_Config_CMD                   0xDA
#define Set_VCOMH_Level_CMD                        0xDB
#define Set_NOP_CMD                                0xE3

#define Horizontal_Addressing_Mode                 0x00
#define Vertical_Addressing_Mode                   0x01
#define Page_Addressing_Mode                       0x02

#define Disable_Charge_Pump                        0x00
#define Enable_Charge_Pump                         0x04

#define Column_Address_0_Mapped_to_SEG0            0x00
#define Column_Address_0_Mapped_to_SEG127          0x01

#define Normal_Display                             0x00
#define Entire_Display_ON                          0x01

#define Non_Inverted_Display                       0x00
#define Inverted_Display                           0x01

#define Display_OFF                                0x00
#define Display_ON                                 0x01

#define Scan_from_COM0_to_63                       0x00
#define Scan_from_COM63_to_0                       0x08

#define x_size                                     128
#define x_max                                      x_size
#define x_min                                      0
#define y_size                                     64
#define y_max                                      8
#define y_min                                      0

#define ON                                         1
#define OFF                                        0

#define YES                                        1
#define NO                                         0

#define ROUND                                      1
#define SQUARE                                     0
```

```c
#define buffer_size                                                    1024//(x_max * y_max)


extern unsigned char buffer[buffer_size];


void I2C_setup(void);
void OLED_HW_setup(void);
void OLED_init(void);
void OLED_write(unsigned char value, unsigned char control_byte);
void OLED_gotoxy(unsigned char x_pos, unsigned char y_pos);
void OLED_fill(unsigned char bmp_data);
void OLED_print_Image(const unsigned char *bmp, unsigned char pixel);
void OLED_clear_screen(void);
void OLED_clear_buffer(void);
void OLED_cursor(unsigned char x_pos, unsigned char y_pos);
void OLED_draw_bitmap(unsigned char xb, unsigned char yb, unsigned char xe,
unsigned char ye, unsigned char *bmp_img);
void OLED_print_char(unsigned char x_pos, unsigned char y_pos, unsigned char ch);
void OLED_print_string(unsigned char x_pos, unsigned char y_pos, unsigned char
*ch);
void OLED_print_chr(unsigned char x_pos, unsigned char y_pos, signed int value);
void OLED_print_int(unsigned char x_pos, unsigned char y_pos, signed long value);
void OLED_print_decimal(unsigned char x_pos, unsigned char y_pos, unsigned int
value, unsigned char points);
void OLED_print_float(unsigned char x_pos, unsigned char y_pos, float value,
unsigned char points);
void Draw_Pixel(unsigned char x_pos, unsigned char y_pos, unsigned char colour);
void Draw_Line(signed int x1, signed int y1, signed int x2, signed int y2,
unsigned char colour);
void Draw_Rectangle(signed int x1, signed int y1, signed int x2, signed int y2,
unsigned char fill, unsigned char colour, unsigned char type);
void Draw_Circle(signed int xc, signed int yc, signed int radius, unsigned char
fill, unsigned char colour);
```

SSD1306.c

```c
#include "fonts.h"
#include "SSD1306.h"


void I2C_setup(void)
{
    I2C_DeInit();

    I2C_Init(100000,
            SSD1306_I2C_Address,
            I2C_DUTYCYCLE_2,
            I2C_ACK_CURR,
            I2C_ADDMODE_7BIT,
            (CLK_GetClockFreq() / 1000000));

    I2C_Cmd(ENABLE);
}


void OLED_HW_setup(void)
{
```

```c
        GPIO_Init(I2C_PORT, SCL_pin, GPIO_MODE_OUT_OD_HIZ_FAST);
        GPIO_Init(I2C_PORT, SDA_pin, GPIO_MODE_OUT_OD_HIZ_FAST);

    I2C_setup();
}


void OLED_init(void)
{
    OLED_HW_setup();
    delay_ms(100);

    OLED_write((Set_Display_ON_or_OFF_CMD | Display_OFF), CMD);
    OLED_write(Set_Multiplex_Ratio_CMD, CMD);
    OLED_write(0x3F, CMD);
    OLED_write(Set_Display_Offset_CMD, CMD);
    OLED_write(0x00, CMD);
    OLED_write(Set_Display_Start_Line_CMD, CMD);
    OLED_write((Set_Segment_Remap_CMD | Column_Address_0_Mapped_to_SEG127), CMD);
    OLED_write((Set_COM_Output_Scan_Direction_CMD | Scan_from_COM63_to_0), CMD);
    OLED_write(Set_Common_HW_Config_CMD, CMD);
    OLED_write(0x12, CMD);
    OLED_write(Set_Contrast_Control_CMD, CMD);
    OLED_write(0xFF, CMD);
    OLED_write(Set_Entire_Display_ON_CMD, CMD);
    OLED_write(Set_Normal_or_Inverse_Display_CMD, CMD);
    OLED_write(Set_Display_Clock_CMD, CMD);
    OLED_write(0x80, CMD);
    OLED_write(Set_Pre_charge_Period_CMD, CMD);
    OLED_write(0x25, CMD);
    OLED_write(Set_VCOMH_Level_CMD, CMD);
    OLED_write(0x20, CMD);
    OLED_write(Set_Page_Address_CMD, CMD);
    OLED_write(0x00, CMD);
    OLED_write(0x07, CMD);
    OLED_write(Set_Page_Start_Address_CMD , CMD);
    OLED_write(Set_Higher_Column_Start_Address_CMD, CMD);
    OLED_write(Set_Lower_Column_Start_Address_CMD, CMD);
    OLED_write(Set_Memory_Addressing_Mode_CMD, CMD);
    OLED_write(0x02, CMD);
    OLED_write(Set_Charge_Pump_CMD, CMD);
    OLED_write(0x14, CMD);
    OLED_write((Set_Display_ON_or_OFF_CMD | Display_ON), CMD);
}


void OLED_write(unsigned char value, unsigned char control_byte)
{
    while(I2C_GetFlagStatus(I2C_FLAG_BUSBUSY));

    I2C_GenerateSTART(ENABLE);
    while(!I2C_CheckEvent(I2C_EVENT_MASTER_MODE_SELECT));

    I2C_Send7bitAddress(SSD1306_I2C_Address, I2C_DIRECTION_TX);
    while(!I2C_CheckEvent(I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

    I2C_SendData(control_byte);
    while(!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTING));
```

```c
        I2C_SendData(value);
        while(!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTED));

        I2C_GenerateSTOP(ENABLE);
}


void OLED_gotoxy(unsigned char x_pos, unsigned char y_pos)
{
        OLED_write((Set_Page_Start_Address_CMD + y_pos), CMD);
        OLED_write(((x_pos & 0x0F) | Set_Lower_Column_Start_Address_CMD), CMD);
        OLED_write((((x_pos & 0xF0) >> 0x04) | Set_Higher_Column_Start_Address_CMD),
CMD);
}


void OLED_fill(unsigned char bmp_data)
{
        unsigned char x_pos = 0x00;
        unsigned char page = 0x00;

        for(page = 0; page < y_max; page++)
        {
                OLED_gotoxy(x_min, page);

                I2C_GenerateSTART(ENABLE);
                while(!I2C_CheckEvent(I2C_EVENT_MASTER_MODE_SELECT));

                I2C_Send7bitAddress(SSD1306_I2C_Address, I2C_DIRECTION_TX);
                while(!I2C_CheckEvent(I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

                I2C_SendData(DAT);
                while(!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTING));

                for(x_pos = x_min; x_pos < x_max; x_pos++)
                {
                        I2C_SendData(bmp_data);
                                while(!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTED));
                }

                I2C_GenerateSTOP(ENABLE);
        }
}


void OLED_print_Image(const unsigned char *bmp, unsigned char pixel)
{
        unsigned char x_pos = 0;
        unsigned char page = 0;

        if(pixel != OFF)
        {
                pixel = 0xFF;
        }
        else
        {
                pixel = 0x00;
        }
```

```c
        for(page = 0; page < y_max; page++)
        {
            OLED_gotoxy(x_min, page);

            I2C_GenerateSTART(ENABLE);
            while(!I2C_CheckEvent(I2C_EVENT_MASTER_MODE_SELECT));

            I2C_Send7bitAddress(SSD1306_I2C_Address, I2C_DIRECTION_TX);
            while(!I2C_CheckEvent(I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

            I2C_SendData(DAT);
            while(!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTING));

            for(x_pos = x_min; x_pos < x_max; x_pos++)
            {
                I2C_SendData((*bmp++ ^ pixel));
                while(!I2C_CheckEvent(I2C_EVENT_MASTER_BYTE_TRANSMITTED));
            }

                I2C_GenerateSTOP(ENABLE);
        }
}


void OLED_clear_screen(void)
{
    OLED_fill(0x00);
}


void OLED_clear_buffer(void)
{
    unsigned int s = 0x0000;

    for(s = 0; s < buffer_size; s++)
    {
        buffer[s] = 0x00;
    }
}


void OLED_cursor(unsigned char x_pos, unsigned char y_pos)
{
    unsigned char s = 0x00;

    if(y_pos != 0x00)
    {
        if(x_pos == 1)
        {
            OLED_gotoxy(0x00, (y_pos + 0x02));
        }
        else
        {
            OLED_gotoxy((0x50 + ((x_pos - 0x02) * 0x06)), (y_pos + 0x02));
        }

        for(s = 0x00; s < 0x06; s++)
        {
            OLED_write(0xFF, DAT);
```

```c
        }
    }
}


void OLED_draw_bitmap(unsigned char xb, unsigned char yb, unsigned char xe,
unsigned char ye, unsigned char *bmp_img)
{
    unsigned int s = 0x0000;
    unsigned char x_pos = 0x00;
    unsigned char y_pos = 0x00;

    for(y_pos = yb; y_pos <= ye; y_pos++)
    {
        OLED_gotoxy(xb, y_pos);
        for(x_pos = xb; x_pos < xe; x_pos++)
        {
            OLED_write(bmp_img[s++], DAT);
        }
    }
}


void OLED_print_char(unsigned char x_pos, unsigned char y_pos, unsigned char ch)
{
  unsigned char s = 0x00;
  unsigned char chr = 0x00;

  chr = (ch - 0x20);

  if(x_pos > (x_max - 0x06))
  {
    x_pos = 0x00;
    y_pos++;
  }
  OLED_gotoxy(x_pos, y_pos);

  for(s = 0x00; s < 0x06; s++)
  {
    OLED_write(font_regular[chr][s], DAT);
  }
}


void OLED_print_string(unsigned char x_pos, unsigned char y_pos, char *ch)
{
  do
  {
    OLED_print_char(x_pos, y_pos, *ch++);
    x_pos += 0x06;
  }while((*ch >= 0x20) && (*ch <= 0x7F) && (*ch != '\n'));
}


void OLED_print_chr(unsigned char x_pos, unsigned char y_pos, signed int value)
{
    char ch[5] = {0x20, 0x20, 0x20, 0x20, '\n'};

    if(value < 0)
```

```c
    {
        ch[0] = 0x2D;
        value = -value;
    }
    else
    {
        ch[0] = 0x20;
    }

    if((value > 99) && (value <= 999))
    {
        ch[1] = ((value / 100) + 0x30);
        ch[2] = (((value % 100) / 10) + 0x30);
        ch[3] = ((value % 10) + 0x30);
    }
    else if((value > 9) && (value <= 99))
    {
        ch[1] = ((value / 10) + 0x30);
        ch[2] = ((value % 10) + 0x30);
        ch[3] = 0x20;
    }
    else if((value >= 0) && (value <= 9))
    {
        ch[1] = ((value % 10) + 0x30);
        ch[2] = 0x20;
        ch[3] = 0x20;
    }

    OLED_print_string(x_pos, y_pos, ch);
}


void OLED_print_int(unsigned char x_pos, unsigned char y_pos, signed long value)
{
    char ch[7] = {0x20, 0x20, 0x20, 0x20, 0x20, 0x20, '\n'};

    if(value < 0)
    {
        ch[0] = 0x2D;
        value = -value;
    }
    else
    {
        ch[0] = 0x20;
    }

    if(value > 9999)
    {
        ch[1] = ((value / 10000) + 0x30);
        ch[2] = (((value % 10000) / 1000) + 0x30);
        ch[3] = (((value % 1000) / 100) + 0x30);
        ch[4] = (((value % 100) / 10) + 0x30);
        ch[5] = ((value % 10) + 0x30);
    }

    else if((value > 999) && (value <= 9999))
    {
        ch[1] = ((value / 1000) + 0x30);
        ch[2] = (((value % 1000) / 100) + 0x30);
```

```c
      ch[3] = (((value % 100) / 10) + 0x30);
      ch[4] = ((value % 10) + 0x30);
      ch[5] = 0x20;
   }
   else if((value > 99) && (value <= 999))
   {
      ch[1] = ((value / 100) + 0x30);
      ch[2] = (((value % 100) / 10) + 0x30);
      ch[3] = ((value % 10) + 0x30);
      ch[4] = 0x20;
      ch[5] = 0x20;
   }
   else if((value > 9) && (value <= 99))
   {
      ch[1] = ((value / 10) + 0x30);
      ch[2] = ((value % 10) + 0x30);
      ch[3] = 0x20;
      ch[4] = 0x20;
      ch[5] = 0x20;
   }
   else
   {
      ch[1] = ((value % 10) + 0x30);
      ch[2] = 0x20;
      ch[3] = 0x20;
      ch[4] = 0x20;
      ch[5] = 0x20;
   }

   OLED_print_string(x_pos, y_pos, ch);
}


void OLED_print_decimal(unsigned char x_pos, unsigned char y_pos, unsigned int
value, unsigned char points)
{
   char ch[6] = {0x2E, 0x20, 0x20, 0x20, 0x20, '\n'};

   ch[1] = ((value / 1000) + 0x30);

   if(points > 1)
   {
      ch[2] = (((value % 1000) / 100) + 0x30);

      if(points > 2)
      {
         ch[3] = (((value % 100) / 10) + 0x30);

         if(points > 3)
         {
            ch[4] = ((value % 10) + 0x30);
         }
      }
   }

   OLED_print_string(x_pos, y_pos, ch);
}
```

```c
void OLED_print_float(unsigned char x_pos, unsigned char y_pos, float value,
unsigned char points)
{
  signed long tmp = 0;

  tmp = value;
  OLED_print_int(x_pos, y_pos, tmp);
  tmp = ((value - tmp) * 10000);

  if(tmp < 0)
  {
    tmp = -tmp;
  }

  if((value >= 10000) && (value < 100000))
  {
    OLED_print_decimal((x_pos + 36), y_pos, tmp, points);
  }
  else if((value >= 1000) && (value < 10000))
  {
    OLED_print_decimal((x_pos + 30), y_pos, tmp, points);
  }
  else if((value >= 100) && (value < 1000))
  {
    OLED_print_decimal((x_pos + 24), y_pos, tmp, points);
  }
  else if((value >= 10) && (value < 100))
  {
    OLED_print_decimal((x_pos + 18), y_pos, tmp, points);
  }
  else if(value < 10)
  {
    OLED_print_decimal((x_pos + 12), y_pos, tmp, points);

    if(value < 0)
    {
        OLED_print_char(x_pos, y_pos, 0x2D);
    }
    else
    {
        OLED_print_char(x_pos, y_pos, 0x20);
    }
  }
}


void Draw_Pixel(unsigned char x_pos, unsigned char y_pos, unsigned char colour)
{
    unsigned char value = 0x00;
    unsigned char page = 0x00;
    unsigned char bit_pos = 0x00;

    page = (y_pos / y_max);
    bit_pos = (y_pos - (page * y_max));
    value = buffer[((page * x_max) + x_pos)];

    if((colour & YES) != NO)
    {
        value |= (1 << bit_pos);
```

```c
    }
    else
    {
        value &= (~(1 << bit_pos));
    }

    buffer[((page * x_max) + x_pos)] = value;
    OLED_gotoxy(x_pos, page);
    OLED_write(value, DAT);
}


void Draw_Line(signed int x1, signed int y1, signed int x2, signed int y2,
unsigned char colour)
{
    signed int dx = 0x0000;
    signed int dy = 0x0000;
    signed int stepx = 0x0000;
    signed int stepy = 0x0000;
    signed int fraction = 0x0000;

    dy = (y2 - y1);
    dx = (x2 - x1);

    if (dy < 0)
    {
        dy = -dy;
        stepy = -1;
    }
    else
    {
        stepy = 1;
    }

    if (dx < 0)
    {
        dx = -dx;
        stepx = -1;
    }
    else
    {
        stepx = 1;
    }

    dx <<= 1;
    dy <<= 1;

    Draw_Pixel(x1, y1, colour);

    if(dx > dy)
    {
        fraction = (dy - (dx >> 1));
        while (x1 != x2)
        {
            if(fraction >= 0)
            {
                y1 += stepy;
                fraction -= dx;
            }
```

```c
                x1 += stepx;
                fraction += dy;

                Draw_Pixel(x1, y1, colour);
            }
        }
        else
        {
            fraction = (dx - (dy >> 1));
            while (y1 != y2)
            {
                if (fraction >= 0)
                {
                    x1 += stepx;
                    fraction -= dy;
                }

                y1 += stepy;
                fraction += dx;

                Draw_Pixel(x1, y1, colour);
            }
        }
}

void Draw_Rectangle(signed int x1, signed int y1, signed int x2, signed int y2,
unsigned char fill, unsigned char colour, unsigned char type)
{
    unsigned short i = 0x00;
    unsigned short xmin = 0x00;
    unsigned short xmax = 0x00;
    unsigned short ymin = 0x00;
    unsigned short ymax = 0x00;

    if(fill != NO)
    {
        if(x1 < x2)
        {
            xmin = x1;
            xmax = x2;
        }
        else
        {
            xmin = x2;
            xmax = x1;
        }

        if(y1 < y2)
        {
            ymin = y1;
            ymax = y2;
        }
        else
        {
            ymin = y2;
            ymax = y1;
        }
```

```c
        for(; xmin <= xmax; ++xmin)
        {
            for(i = ymin; i <= ymax; ++i)
            {
                Draw_Pixel(xmin, i, colour);
            }
        }
    }

    else
    {
        Draw_Line(x1, y1, x2, y1, colour);
        Draw_Line(x1, y2, x2, y2, colour);
        Draw_Line(x1, y1, x1, y2, colour);
        Draw_Line(x2, y1, x2, y2, colour);
    }

    if(type != SQUARE)
    {
        Draw_Pixel(x1, y1, ~colour);
        Draw_Pixel(x1, y2, ~colour);
        Draw_Pixel(x2, y1, ~colour);
        Draw_Pixel(x2, y2, ~colour);
    }
}


void Draw_Circle(signed int xc, signed int yc, signed int radius, unsigned char
fill, unsigned char colour)
{
    signed int a = 0x0000;
    signed int b = 0x0000;
    signed int P = 0x0000;

    b = radius;
    P = (1 - b);

    do
    {
        if(fill != NO)
        {
            Draw_Line((xc - a), (yc + b), (xc + a), (yc + b), colour);
            Draw_Line((xc - a), (yc - b), (xc + a), (yc - b), colour);
            Draw_Line((xc - b), (yc + a), (xc + b), (yc + a), colour);
            Draw_Line((xc - b), (yc - a), (xc + b), (yc - a), colour);
        }
        else
        {
            Draw_Pixel((xc + a), (yc + b), colour);
            Draw_Pixel((xc + b), (yc + a), colour);
            Draw_Pixel((xc - a), (yc + b), colour);
            Draw_Pixel((xc - b), (yc + a), colour);
            Draw_Pixel((xc + b), (yc - a), colour);
            Draw_Pixel((xc + a), (yc - b), colour);
            Draw_Pixel((xc - a), (yc - b), colour);
            Draw_Pixel((xc - b), (yc - a), colour);
        }
```

```c
        if(P < 0)
        {
            P += (3 + (2 * a++));
        }
        else
        {
            P += (5 + (2 * ((a++) - (b--))));
        }
    }while(a <= b);
}
```

main.c

```c
#include "STM8S.h"
#include "SSD1306.h"


unsigned char buffer[buffer_size];


void setup_clock(void);
void setup_GPIOs(void);


void main(void)
{
    unsigned char p = 0;

    signed int i = -9;

    float f = -10.0;

    setup_clock();
    setup_GPIOs();
    OLED_init();

    while(1)
    {
        i = -9;
        f = -10.0;

        OLED_fill(0x00);
        OLED_fill(0xFF);
        delay_ms(600);
        OLED_fill(0xAA);
        delay_ms(600);
        OLED_fill(0x55);
        delay_ms(600);

        OLED_clear_buffer();
        OLED_fill(0x00);
        OLED_print_string(36, 0, "MicroArena");
        OLED_print_string(16, 1, "fb.com/MicroArena");
        Draw_Line(0, 19, 127, 19, ON);
        Draw_Line(0, 60, 127, 60, ON);
        delay_ms(900);
        Draw_Line(3, 16, 3, 63, ON);
        Draw_Line(124, 16, 124, 63, ON);
```

```c
        delay_ms(900);
        Draw_Rectangle(122, 58, 5, 21, OFF, ON, SQUARE);
        delay_ms(900);
        Draw_Circle(63, 40, 7, ON, ON);
        delay_ms(2000);

        OLED_fill(0x00);
        OLED_clear_buffer();
        OLED_print_string(36, 0, "SShahyriar");
        Draw_Rectangle(27, 21, 100, 58, ON, ON, ROUND);
        delay_ms(900);
        Draw_Circle(63, 40, 9, ON, OFF);
        delay_ms(2000);

        OLED_fill(0x00);
        OLED_clear_buffer();
        for(p = 0; p < 254; p++)
        {
            f += 0.1;
            i += 1;

            OLED_print_float(42, 2, f, 1);
            OLED_print_int(42, 3, i);
            OLED_print_chr(42, 4, p);

            delay_ms(99);
        }
    };
}


void setup_clock(void)
{
    CLK_DeInit();

    CLK_HSECmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSERDY) == FALSE);

    CLK_LSICmd(DISABLE);

    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

    CLK_ClockSwitchCmd(ENABLE);
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
    CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

    CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSE,
    DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

    CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, ENABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART2, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER3, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
```

```
}

void setup_GPIOs(void)
{
      GPIO_DeInit(I2C_PORT);
}
```

Explanation

Explaining the working of the OLED display is beyond the scope of this article and so I will rather explain the useful functions of the OLED display library.

To use the OLED display library, you need to include the font library and initialize the OLED display in the main function.

```
OLED_init();
```

As said earlier, SSD1306 supports both I2C and SPI communication methods. Here I2C-based method is used and so we need to initialize I2C GPIOs and I2C peripheral clock.

```
CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, ENABLE);
```

The following functions are for general printing:

```
/*
Fills the whole screen with the designated pattern.
*/

void OLED_fill(unsigned char bmp_data);

/*
Prints a bitmap on the whole screen. Pixel sets whether or not to invert the pixel
states.
*/

void OLED_print_Image(const unsigned char *bmp, unsigned char pixel);

/*
Clears the entire screen.
*/

void OLED_clear_screen(void);

/*
Clears the data buffer that holds pixel data.
*/

void OLED_clear_buffer(void);
```

```
/*
Draws a bitmap in the area defined by points (xb, yb) and (xe, ye).
*/

void OLED_draw_bitmap(unsigned char xb, unsigned char yb, unsigned char xe, unsigned
char ye, unsigned char *bmp_img);
```

The following functions are used printing texts and numbers:

```
/*
Prints a single character at position (x_pos, y_pos).
*/

void OLED_print_char(unsigned char x_pos, unsigned char y_pos, unsigned char ch);

/*
Prints a string of characters at position (x_pos, y_pos).
*/

void OLED_print_string(unsigned char x_pos, unsigned char y_pos, unsigned char *ch);

/*
Prints a character type variable at position (x_pos, y_pos).
*/

void OLED_print_chr(unsigned char x_pos, unsigned char y_pos, signed int value);

/*
Prints a integer type variable at position (x_pos, y_pos).
*/


void OLED_print_int(unsigned char x_pos, unsigned char y_pos, signed long value);


/*
Prints a float type variable at position (x_pos, y_pos). Argument "points" states
the number of points to print after the decimal point.
*/

void OLED_print_float(unsigned  char  x_pos,  unsigned  char  y_pos,  float  value,
unsigned char points);
```

The following functions are for projecting basic graphical objects:

```
/*
Draws a pixel at coordinate (x_pos, y_pos).
*/

void Draw_Pixel(unsigned char x_pos, unsigned char y_pos, unsigned char colour);

/*
Draws a line from point (x1, y1) to (x2, y2).
*/

void Draw_Line(signed int x1, signed int y1, signed int x2, signed int y2, unsigned
char colour);
```

```c
/*
Draws a rectangle from point (x1, y1) to (x2, y2). "Fill" states whether or not the
area inside the rectangle is solid filled. "Type" states the edge type –
Round/Square.
*/

void Draw_Rectangle(signed int x1, signed int y1, signed int x2, signed int y2,
unsigned char fill, unsigned char colour, unsigned char type);

/*
Draws a circle with centre at point (xc, yc). "Radius" states the radius of the
circle in pixels. "Fill" states whether or not to solid fill the enclosed circular
area.
*/

void Draw_Circle(signed int xc, signed int yc, signed int radius, unsigned char
fill, unsigned char colour);

/*
In all functions, colour states whether or not to enable the pixel(s) at the
point(s). It doesn't represent actual colour since the displays are not colour
displays.
*/
```

Don't forget to switch to long stack memory model.

Video link: https://www.youtube.com/watch?v=kLOG4GPU-mw.
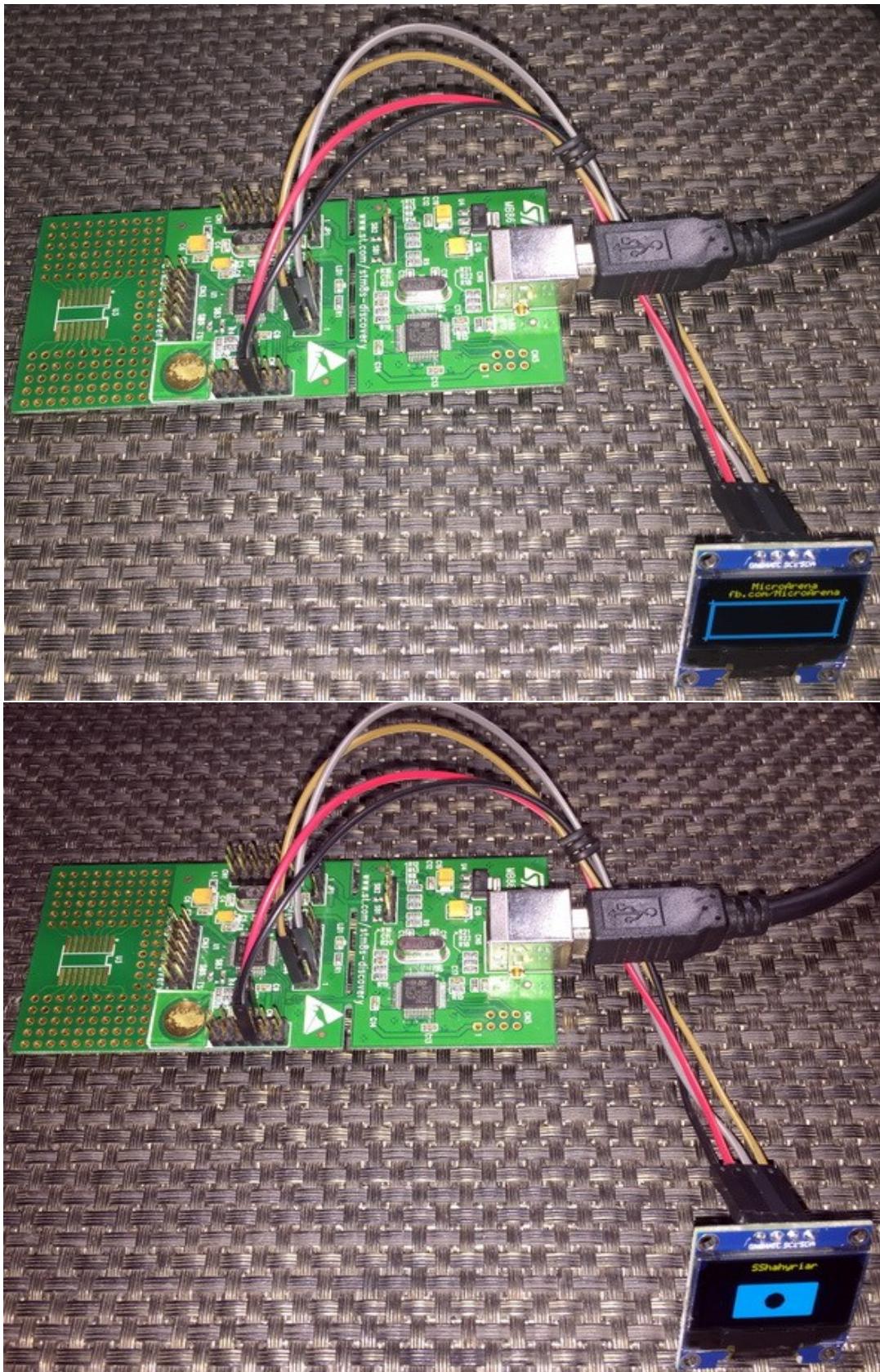
One Wire (OW) – DS18B20

One Wire (OW) communication is not as much as popular as I2C, SPI or serial communication. However, there are a number of devices that use one-wire communication method. DS18B20 digital temperature sensor from Dallas is one such example. For OW communication, we do not need any special hardware. Only GPIO bit-banging is all that is needed. Additionally, we can use a timer if we do not want to use software-based delays.

Hardware Connection
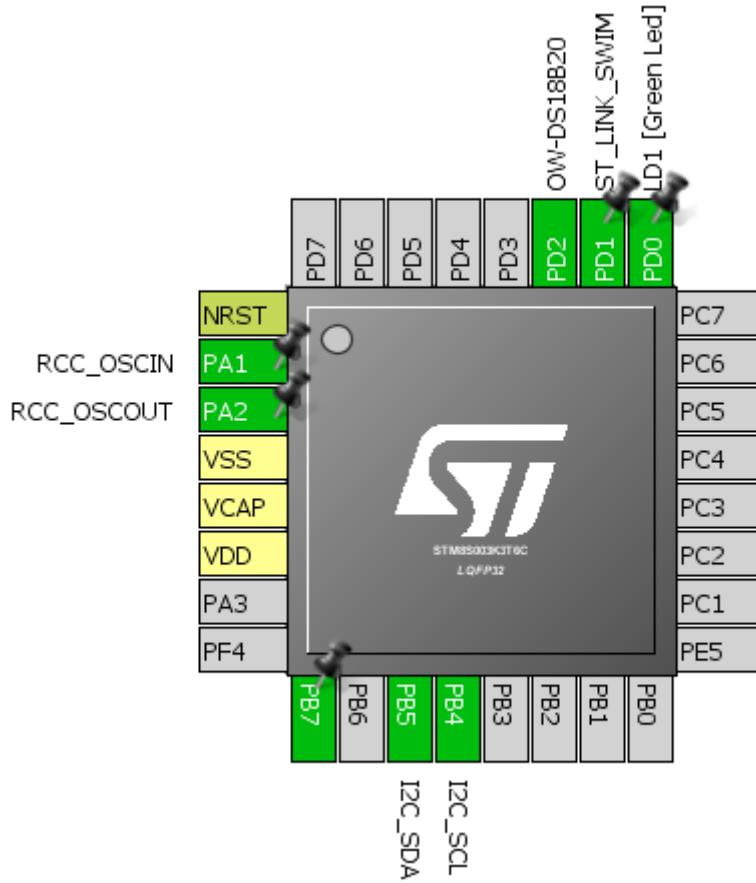
Code Example

one_wire.h

```
#include "STM8S.h"


#define DS18B20_port       GPIOD
#define DS18B20_pin        GPIO_PIN_2

#define DS18B20_INPUT()    do{GPIO_DeInit(DS18B20_port);    GPIO_Init(DS18B20_port,
DS18B20_pin, GPIO_MODE_IN_FL_NO_IT);}while(0)
```

```c
#define  DS18B20_OUTPUT()    do{GPIO_DeInit(DS18B20_port);  GPIO_Init(DS18B20_port,
DS18B20_pin, GPIO_MODE_OUT_PP_HIGH_FAST);}while(0)

#define DS18B20_IN()         GPIO_ReadInputPin(DS18B20_port, DS18B20_pin)

#define DS18B20_OUT_LOW() GPIO_WriteLow(DS18B20_port, DS18B20_pin)
#define DS18B20_OUT_HIGH()GPIO_WriteHigh(DS18B20_port, DS18B20_pin)


bool onewire_reset(void);
void onewire_write_bit(bool bit_value);
bool onewire_read_bit(void);
void onewire_write(unsigned char value);
unsigned char onewire_read(void);
```

one_wire.c

```c
#include "one_wire.h"


bool onewire_reset(void)
{
        bool res = FALSE;

        DS18B20_OUTPUT();
        DS18B20_OUT_LOW();
        delay_us(480);
        DS18B20_OUT_HIGH();
        delay_us(60);

        DS18B20_INPUT();
        res = DS18B20_IN();
        delay_us(480);

        return res;
}


void onewire_write_bit(bool bit_value)
{
        DS18B20_OUTPUT();
        DS18B20_OUT_LOW();

        if(bit_value)
        {
                delay_us(104);
                DS18B20_OUT_HIGH();
        }
}


bool onewire_read_bit(void)
{
        DS18B20_OUTPUT();
        DS18B20_OUT_LOW();
        DS18B20_OUT_HIGH();
         delay_us(15);
```

```c
        DS18B20_INPUT();

        return(DS18B20_IN());
}


void onewire_write(unsigned char value)
{
        unsigned char s = 0;

        DS18B20_OUTPUT();

        while(s < 8)
        {
                if((value & (1 << s)))
                {
                        DS18B20_OUT_LOW();
                        delay_cycles(1);
                        DS18B20_OUT_HIGH();
                        delay_us(60);
                }

                else
                {
                         DS18B20_OUT_LOW();
                        delay_us(60);
                        DS18B20_OUT_HIGH();
                        delay_cycles(1);
                }
                s++;
        }
}


unsigned char onewire_read(void)
{
        unsigned char s = 0x00;
        unsigned char value = 0x00;

        while(s < 8)
        {
                DS18B20_OUTPUT();

                DS18B20_OUT_LOW();
                delay_cycles(1);
                DS18B20_OUT_HIGH();

                DS18B20_INPUT();
                if(DS18B20_IN())
                {
                    value |=  (1 << s);
                }

                delay_us(60);
                s++;
        }

        return value;
}
```

DS18B20.h

```c
#include "STM8S.h"
#include "one_wire.h"


#define convert_T          0x44
#define read_scratchpad    0xBE
#define write_scratchpad   0x4E
#define copy_scratchpad    0x48
#define recall_E2          0xB8
#define read_power_supply  0xB4
#define skip_ROM           0xCC

#define resolution         12



void DS18B20_init(void);
float DS18B20_get_temperature(void);
```

DS18B20.c

```c
#include "DS18B20.h"


void DS18B20_init(void)
{
    onewire_reset();
    delay_ms(100);
}


float DS18B20_get_temperature(void)
{
    unsigned char msb = 0x00;
    unsigned char lsb = 0x00;
    register float temp = 0.0;

    onewire_reset();
    onewire_write(skip_ROM);
    onewire_write(convert_T);

    switch(resolution)
    {
        case 12:
        {
            delay_ms(750);
            break;
        }
        case 11:
        {
            delay_ms(375);
            break;
        }
        case 10:
        {
            delay_ms(188);
```

```c
                break;
            }
            case 9:
            {
                delay_ms(94);
                break;
            }
        }


    onewire_reset();

    onewire_write(skip_ROM);
    onewire_write(read_scratchpad);

    lsb = onewire_read();
    msb = onewire_read();

    temp = msb;
    temp *= 256.0;
    temp += lsb;


    switch(resolution)
    {
        case 12:
        {
            temp *= 0.0625;
            break;
        }
        case 11:
        {
            temp *= 0.125;
            break;
        }
        case 10:
        {
            temp *= 0.25;
            break;
        }
        case 9:
        {
            temp *= 0.5;
            break;
        }
    }

    delay_ms(40);

    return (temp);
}
```

main.c

```c
#include "STM8S.h"
#include "DS18B20.h"
#include "lcd.h"
```

```c
unsigned char bl_state;
unsigned char data_value;

const unsigned char symbol[8] =
{
    0x00, 0x06, 0x09, 0x09, 0x06, 0x00, 0x00, 0x00
};


void clock_setup(void);
void GPIO_setup(void);
void lcd_symbol(void);
void print_C(unsigned char x_pos, unsigned char y_pos, signed int value);
void print_I(unsigned char x_pos, unsigned char y_pos, signed long value);
void print_D(unsigned char x_pos, unsigned char y_pos, signed int value, unsigned
char points);
void print_F(unsigned char x_pos, unsigned char y_pos, float value, unsigned char
points);


void main(void)
{
    float temp = 0.0;

    clock_setup();
    GPIO_setup();

    DS18B20_init();

    LCD_init();
    LCD_clear_home();
    lcd_symbol();

    LCD_goto(0, 0);
    LCD_putstr("STM8 DS18B20 Ex.");

    LCD_goto(0, 1);
    LCD_putstr("T/ C");
    LCD_goto(2, 1);
    LCD_send(0, DAT);

    while(TRUE)
    {
        temp = DS18B20_get_temperature();
        print_F(9, 1, temp, 3);
        delay_ms(600);
    };
}


void clock_setup(void)
{
    CLK_DeInit();

    CLK_HSECmd(DISABLE);
    CLK_LSICmd(DISABLE);
    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);
```

```c
        CLK_ClockSwitchCmd(ENABLE);
        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);

        CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

        CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO,
                              CLK_SOURCE_HSI,
                              DISABLE,
                              CLK_CURRENTCLOCKSTATE_ENABLE);

        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
    GPIO_DeInit(DS18B20_port);
}


void lcd_symbol(void)
{
   unsigned char s = 0;

   LCD_send(0x40, CMD);

   for(s = 0; s < 8; s++)
   {
       LCD_send(symbol[s], DAT);
   }

   LCD_send(0x80, CMD);
}


void print_C(unsigned char x_pos, unsigned char y_pos, signed int value)
{
    unsigned char ch[5] = {0x20, 0x20, 0x20, 0x20, '\0'};

    if(value < 0x00)
    {
        ch[0] = 0x2D;
        value = -value;
    }
    else
    {
        ch[0] = 0x20;
    }

    if((value > 99) && (value <= 999))
    {
        ch[1] = ((value / 100) + 0x30);
```

```c
            ch[2] = (((value % 100) / 10) + 0x30);
            ch[3] = ((value % 10) + 0x30);
        }
        else if((value > 9) && (value <= 99))
        {
            ch[1] = (((value % 100) / 10) + 0x30);
            ch[2] = ((value % 10) + 0x30);
            ch[3] = 0x20;
        }
        else if((value >= 0) && (value <= 9))
        {
            ch[1] = ((value % 10) + 0x30);
            ch[2] = 0x20;
            ch[3] = 0x20;
        }

        LCD_goto(x_pos, y_pos);
    LCD_putstr(ch);
}


void print_I(unsigned char x_pos, unsigned char y_pos, signed long value)
{
    unsigned char ch[7] = {0x20, 0x20, 0x20, 0x20, 0x20, 0x20, '\0'};

    if(value < 0)
    {
        ch[0] = 0x2D;
        value = -value;
    }
    else
    {
        ch[0] = 0x20;
    }

    if(value > 9999)
    {
        ch[1] = ((value / 10000) + 0x30);
        ch[2] = (((value % 10000)/ 1000) + 0x30);
        ch[3] = (((value % 1000) / 100) + 0x30);
        ch[4] = (((value % 100) / 10) + 0x30);
        ch[5] = ((value % 10) + 0x30);
    }

    else if((value > 999) && (value <= 9999))
    {
        ch[1] = (((value % 10000)/ 1000) + 0x30);
        ch[2] = (((value % 1000) / 100) + 0x30);
        ch[3] = (((value % 100) / 10) + 0x30);
        ch[4] = ((value % 10) + 0x30);
        ch[5] = 0x20;
    }
    else if((value > 99) && (value <= 999))
    {
        ch[1] = (((value % 1000) / 100) + 0x30);
        ch[2] = (((value % 100) / 10) + 0x30);
        ch[3] = ((value % 10) + 0x30);
        ch[4] = 0x20;
        ch[5] = 0x20;
    }
```

```c
    }
    else if((value > 9) && (value <= 99))
    {
        ch[1] = (((value % 100) / 10) + 0x30);
        ch[2] = ((value % 10) + 0x30);
        ch[3] = 0x20;
        ch[4] = 0x20;
        ch[5] = 0x20;
    }
    else
    {
        ch[1] = ((value % 10) + 0x30);
        ch[2] = 0x20;
        ch[3] = 0x20;
        ch[4] = 0x20;
        ch[5] = 0x20;
    }

    LCD_goto(x_pos, y_pos);
    LCD_putstr(ch);
}


void print_D(unsigned char x_pos, unsigned char y_pos, signed int value, unsigned char points)
{
    char ch[5] = {0x2E, 0x20, 0x20, '\0'};

    ch[1] = ((value / 100) + 0x30);

    if(points > 1)
    {
        ch[2] = (((value / 10) % 10) + 0x30);

            if(points > 1)
            {
                ch[3] = ((value % 10) + 0x30);
            }
    }

    LCD_goto(x_pos, y_pos);
    LCD_putstr(ch);
}

void print_F(unsigned char x_pos, unsigned char y_pos, float value, unsigned char points)
{
    signed long tmp = 0x0000;

    tmp = value;
    print_I(x_pos, y_pos, tmp);
    tmp = ((value - tmp) * 1000);

    if(tmp < 0)
    {
        tmp = -tmp;
    }
```

```c
    if(value < 0)
    {
        value = -value;
            LCD_goto(x_pos, y_pos);
        LCD_putchar(0x2D);
    }
    else
    {
        LCD_goto(x_pos, y_pos);
        LCD_putchar(0x20);
    }

    if((value >= 10000) && (value < 100000))
    {
        print_D((x_pos + 6), y_pos, tmp, points);
    }
    else if((value >= 1000) && (value < 10000))
    {
        print_D((x_pos + 5), y_pos, tmp, points);
    }
    else if((value >= 100) && (value < 1000))
    {
        print_D((x_pos + 4), y_pos, tmp, points);
    }
    else if((value >= 10) && (value < 100))
    {
        print_D((x_pos + 3), y_pos, tmp, points);
    }
    else if(value < 10)
    {
        print_D((x_pos + 2), y_pos, tmp, points);
    }
}
```

Explanation

One wire communication is detailed in these application notes from Maxim:

https://www.maximintegrated.com/en/app-notes/index.mvp/id/126

https://www.maximintegrated.com/en/app-notes/index.mvp/id/162

These notes are all that are needed for implementing the one wire communication interface for DS18B20. Please go through these notes. The codes are self-explanatory and are implemented from the code examples in these app notes.

Demo

Video link: https://www.youtube.com/watch?v=CGpO-OLjtSQ.
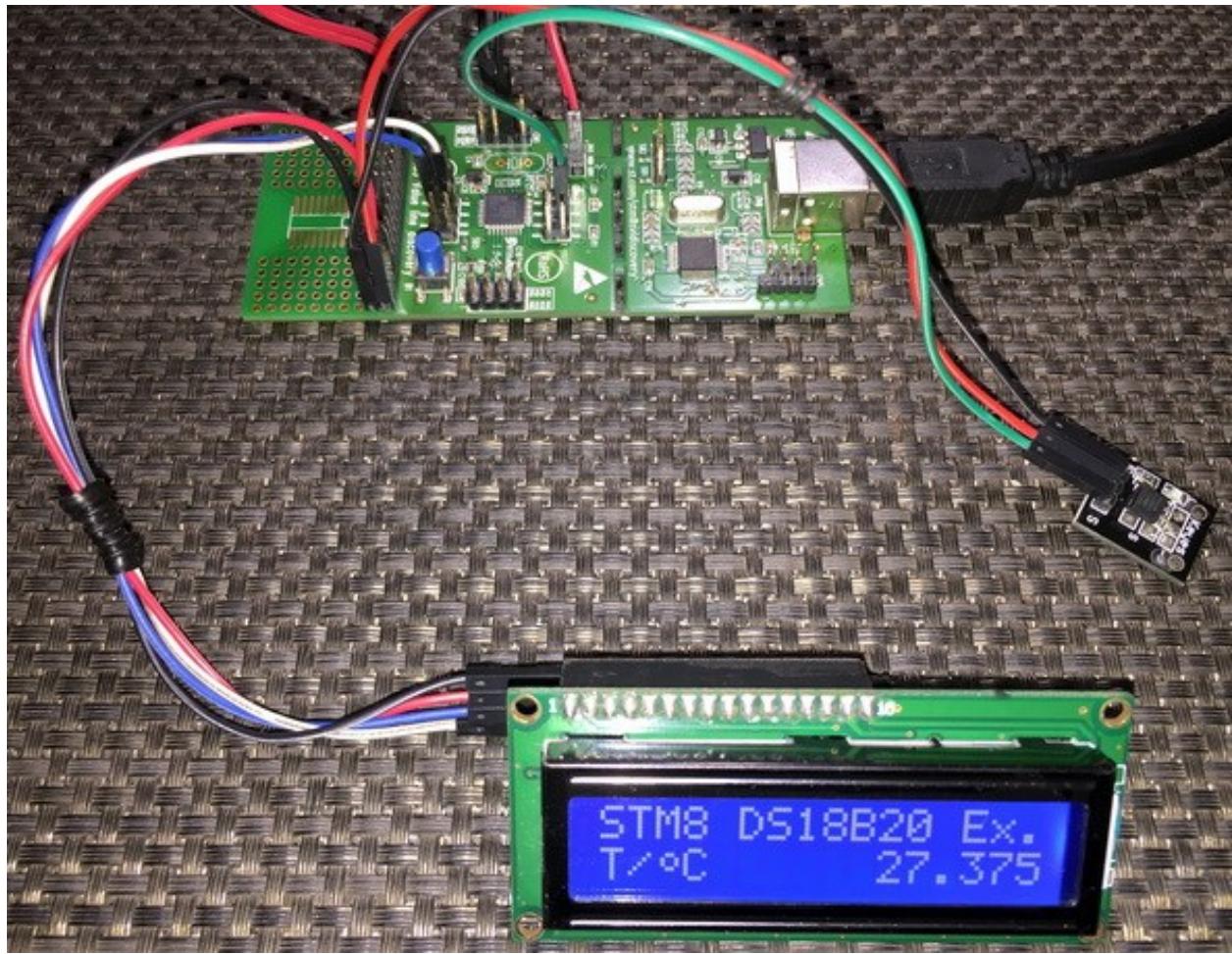
One Wire – DHT11

Just like DS18B20, DHT11 one wire digital relative humidity and temperature sensor also uses one wire communication method to communicate with its host micro. It uses time-slotting mechanism to transmit-receive relative humidity and temperature data. However, the method of communication is different.

Hardware Connection
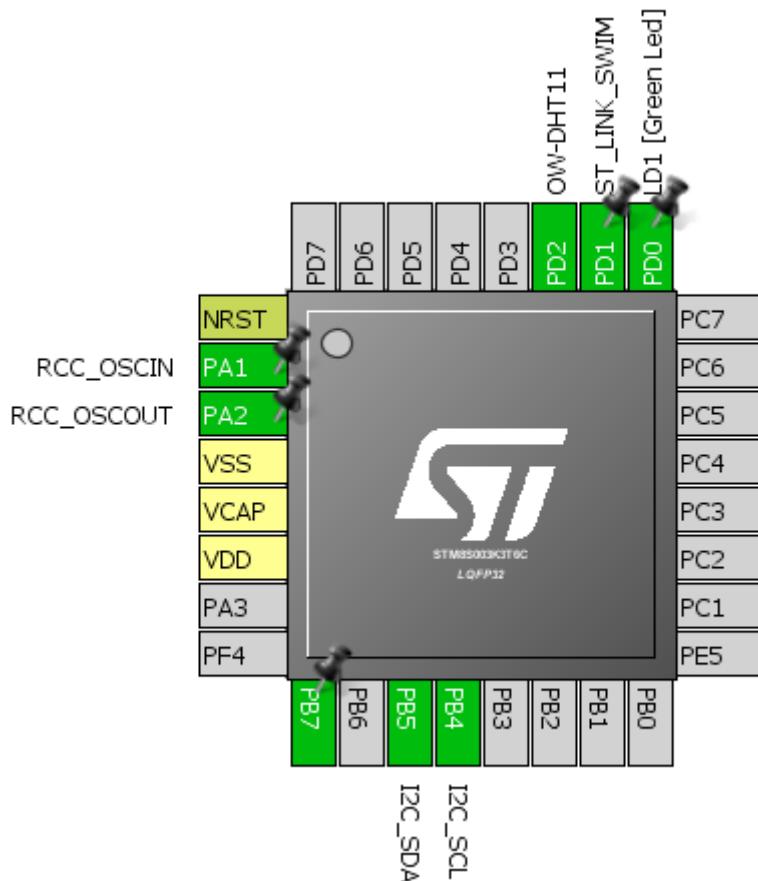
Code Example

DHT11.h

```c
#include "STM8S.h"


#define DHT11_port          GPIOD
#define DHT11_pin           GPIO_PIN_2

#define DHT11_INPUT()     do{GPIO_DeInit(DHT11_port); GPIO_Init(DHT11_port,
DHT11_pin, GPIO_MODE_IN_FL_NO_IT);}while(0)
#define DHT11_OUTPUT()    do{GPIO_DeInit(DHT11_port); GPIO_Init(DHT11_port,
DHT11_pin, GPIO_MODE_OUT_PP_HIGH_FAST);}while(0)
```

```
#define DHT11_IN()           GPIO_ReadInputPin(DHT11_port, DHT11_pin)

#define DHT11_OUT_LOW()   GPIO_WriteLow(DHT11_port, DHT11_pin)
#define DHT11_OUT_HIGH()  GPIO_WriteHigh(DHT11_port, DHT11_pin)


extern unsigned char values[5];



void DHT11_init(void);
unsigned char get_byte(void);
unsigned char get_data(void);
```

DHT11.c

```
#include "DHT11.h"


void DHT11_init(void)
{
    DHT11_INPUT();
    delay_ms(1000);
}


unsigned char get_byte(void)
{
    unsigned char s = 0x08;
    unsigned char value = 0x00;

    DHT11_INPUT();

    while(s > 0)
    {
        value <<= 1;

        while(DHT11_IN() == FALSE);
        delay_us(30);

        if(DHT11_IN())
        {
            value |= 1;
        }

        while(DHT11_IN());
        s--;
    }

    return value;
}


unsigned char get_data(void)
{
    bool chk = FALSE;
    unsigned char s = 0x00;
    unsigned char check_sum = 0x00;
```

```c
    DHT11_OUTPUT();

    DHT11_OUT_HIGH();
    DHT11_OUT_LOW();

    delay_ms(18);

    DHT11_OUT_HIGH();

    delay_us(26);
    DHT11_INPUT();

    chk = DHT11_IN();

    if(chk == TRUE)
    {
        return 1;
    }

    delay_us(80);

    chk = DHT11_IN();

    if(chk == FALSE)
    {
        return 2;
    }

    delay_us(80);

    for(s = 0; s <= 4; s += 1)
    {
        values[s] = get_byte();
    }

    DHT11_OUTPUT();
    DHT11_OUT_HIGH();

    for(s = 0; s < 4; s++)
    {
    check_sum += values[s];
    }

    if(check_sum != values[4])
    {
        return 3;
    }
    else
    {
        return 0;
    }
}
```

main.c

```c
#include "STM8S.h"
#include "lcd.h"
#include "DHT11.h"


unsigned char bl_state;
unsigned char data_value;
unsigned char values[5];

const unsigned char symbol[8] =
{
    0x00, 0x06, 0x09, 0x09, 0x06, 0x00, 0x00, 0x00
};


void clock_setup(void);
void GPIO_setup(void);
void lcd_symbol(void);
unsigned int make_word(unsigned char HB, unsigned char LB);
void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned char value);


void main(void)
{
    unsigned char state = 0x00;

    unsigned int RH = 0x0000;
    unsigned int t = 0x0000;

    clock_setup();
    GPIO_setup();

    DHT11_init();

    LCD_init();
    LCD_clear_home();
    lcd_symbol();

    while(TRUE)
    {
        GPIO_WriteReverse(GPIOD, GPIO_PIN_0);

        state = get_data();

        switch(state)
        {
            case 1:
            {
            }
            case 2:
            {
                LCD_clear_home();
                LCD_putstr("No Sensor Found!");
                break;
            }
            case 3:
            {
```

```c
                            LCD_clear_home();
                            LCD_putstr("Checksum Error!");
                            break;
                    }
                    default:
                    {
                            LCD_goto(0, 0);
                            LCD_putstr("R.H/ %:          ");

                            RH = make_word(values[0], values[1]);
                            lcd_print(14, 0, values[0]);

                            LCD_goto(0, 1);
                            LCD_putstr("Tmp/");
                            LCD_goto(4, 1);
                            LCD_send(0, DAT);
                            LCD_goto(5, 1);
                            LCD_putstr("C:");

                            t = make_word(values[2], values[3]);
                            if((values[2] & 0x80) == 1)
                            {
                                    LCD_goto(13, 1);
                                    LCD_putstr("-");
                            }
                            else
                            {
                                    LCD_goto(13, 1);
                                    LCD_putstr(" ");
                            }

                            lcd_print(14, 1, values[2]);
                            break;
                    }
            }
            delay_ms(1000);
        };
}


void clock_setup(void)
{
        CLK_DeInit();

        CLK_HSECmd(DISABLE);
        CLK_LSICmd(DISABLE);
        CLK_HSICmd(ENABLE);
        while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

        CLK_ClockSwitchCmd(ENABLE);
        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);

        CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

        CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO,
                                CLK_SOURCE_HSI,
                        DISABLE,
                        CLK_CURRENTCLOCKSTATE_ENABLE);
```

```c
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
        GPIO_DeInit(DHT11_port);
}


void lcd_symbol(void)
{
        unsigned char s = 0;

    LCD_send(0x40, CMD);

    for(s = 0; s < 8; s++)
    {
        LCD_send(symbol[s], DAT);
    }

    LCD_send(0x80, CMD);
}


unsigned int make_word(unsigned char HB, unsigned char LB)
{
  unsigned int value = 0x0000;

  value = HB;
  value <<= 8;
  value |= LB;

  return value;
}


void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned char value)
{
        char chr = 0x00;

        chr = ((value / 10) + 0x30);
        LCD_goto(x_pos, y_pos);
        LCD_putchar(chr);

        chr = ((value % 10) + 0x30);
        LCD_goto((x_pos + 1), y_pos);
        LCD_putchar(chr);
}
```

Explanation

Unlike I2C, SMBUS, UART and other communication methods, one wire communication has no fixed communication standard. A perfect example is the difference between the way of communicating with DHT11 and DS18B20.
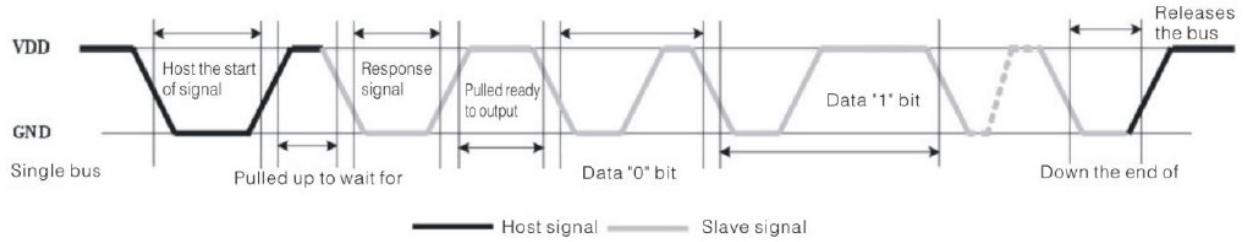
Shown above is the timing diagram of DHT11. If you compare the timings for ones and zeroes in both devices you'll notice that these timings are way different. Same goes for the data, command and control. Here again the datasheet of DHT11 is used to create the library for DHT11 and again the process is just manipulation of a single GPIO pin.

Demo

Video link: https://www.youtube.com/watch?v=-n4obnWGrN4.
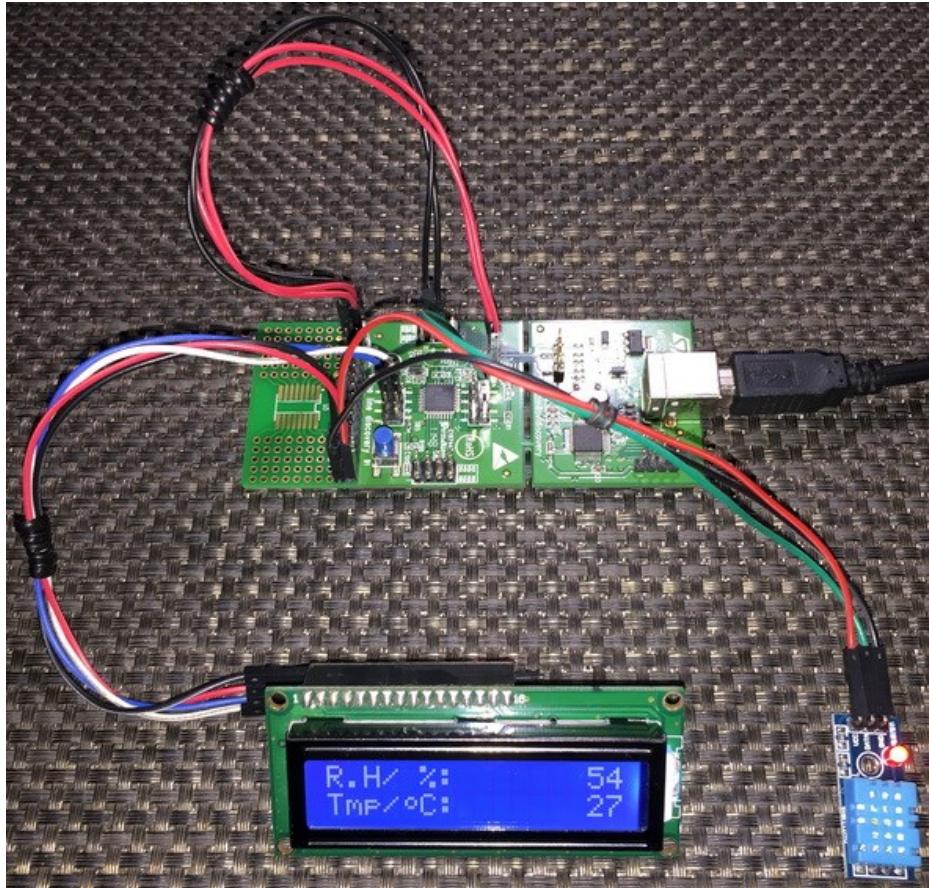
Bit-Banging PCD8544 GLCD

Without I2C, SPI or other forms of communications, it is possible to interface graphical displays with microcontrollers. By bit-banging GPIO pins it is possible to integrate graphical displays like PCD8544. This display was the display of several early Nokia phones like the famous Nokia 5110. At present, these LCDs are sold as ready-made GLCD modules ready to be integrated.

Though PCD8544 supports SPI communication method, here in this example bit-banging method is used. Thus, we can realize it being driven by software SPI instead of hardware-based SPI.

Hardware Connection

Code Example

font.h

```c
static const unsigned char font[][5] =
{
     {0x00, 0x00, 0x00, 0x00, 0x00} // 20
    ,{0x00, 0x00, 0x5f, 0x00, 0x00} // 21 !
    ,{0x00, 0x07, 0x00, 0x07, 0x00} // 22 "
    ,{0x14, 0x7f, 0x14, 0x7f, 0x14} // 23 #
    ,{0x24, 0x2a, 0x7f, 0x2a, 0x12} // 24 $
    ,{0x23, 0x13, 0x08, 0x64, 0x62} // 25 %
    ,{0x36, 0x49, 0x55, 0x22, 0x50} // 26 &
    ,{0x00, 0x05, 0x03, 0x00, 0x00} // 27 '
    ,{0x00, 0x1c, 0x22, 0x41, 0x00} // 28 (
    ,{0x00, 0x41, 0x22, 0x1c, 0x00} // 29 )
    ,{0x14, 0x08, 0x3e, 0x08, 0x14} // 2a *
    ,{0x08, 0x08, 0x3e, 0x08, 0x08} // 2b +
    ,{0x00, 0x50, 0x30, 0x00, 0x00} // 2c ,
    ,{0x08, 0x08, 0x08, 0x08, 0x08} // 2d -
    ,{0x00, 0x60, 0x60, 0x00, 0x00} // 2e .
    ,{0x20, 0x10, 0x08, 0x04, 0x02} // 2f /
    ,{0x3e, 0x51, 0x49, 0x45, 0x3e} // 30 0
    ,{0x00, 0x42, 0x7f, 0x40, 0x00} // 31 1
    ,{0x42, 0x61, 0x51, 0x49, 0x46} // 32 2
    ,{0x21, 0x41, 0x45, 0x4b, 0x31} // 33 3
    ,{0x18, 0x14, 0x12, 0x7f, 0x10} // 34 4
    ,{0x27, 0x45, 0x45, 0x45, 0x39} // 35 5
    ,{0x3c, 0x4a, 0x49, 0x49, 0x30} // 36 6
    ,{0x01, 0x71, 0x09, 0x05, 0x03} // 37 7
    ,{0x36, 0x49, 0x49, 0x49, 0x36} // 38 8
    ,{0x06, 0x49, 0x49, 0x29, 0x1e} // 39 9
    ,{0x00, 0x36, 0x36, 0x00, 0x00} // 3a :
    ,{0x00, 0x56, 0x36, 0x00, 0x00} // 3b ;
    ,{0x08, 0x14, 0x22, 0x41, 0x00} // 3c <
    ,{0x14, 0x14, 0x14, 0x14, 0x14} // 3d =
    ,{0x00, 0x41, 0x22, 0x14, 0x08} // 3e >
    ,{0x02, 0x01, 0x51, 0x09, 0x06} // 3f ?
    ,{0x32, 0x49, 0x79, 0x41, 0x3e} // 40 @
    ,{0x7e, 0x11, 0x11, 0x11, 0x7e} // 41 A
    ,{0x7f, 0x49, 0x49, 0x49, 0x36} // 42 B
    ,{0x3e, 0x41, 0x41, 0x41, 0x22} // 43 C
    ,{0x7f, 0x41, 0x41, 0x22, 0x1c} // 44 D
    ,{0x7f, 0x49, 0x49, 0x49, 0x41} // 45 E
    ,{0x7f, 0x09, 0x09, 0x09, 0x01} // 46 F
    ,{0x3e, 0x41, 0x49, 0x49, 0x7a} // 47 G
    ,{0x7f, 0x08, 0x08, 0x08, 0x7f} // 48 H
    ,{0x00, 0x41, 0x7f, 0x41, 0x00} // 49 I
    ,{0x20, 0x40, 0x41, 0x3f, 0x01} // 4a J
    ,{0x7f, 0x08, 0x14, 0x22, 0x41} // 4b K
    ,{0x7f, 0x40, 0x40, 0x40, 0x40} // 4c L
    ,{0x7f, 0x02, 0x0c, 0x02, 0x7f} // 4d M
    ,{0x7f, 0x04, 0x08, 0x10, 0x7f} // 4e N
    ,{0x3e, 0x41, 0x41, 0x41, 0x3e} // 4f O
    ,{0x7f, 0x09, 0x09, 0x09, 0x06} // 50 P
    ,{0x3e, 0x41, 0x51, 0x21, 0x5e} // 51 Q
    ,{0x7f, 0x09, 0x19, 0x29, 0x46} // 52 R
    ,{0x46, 0x49, 0x49, 0x49, 0x31} // 53 S
```

```
    ,{0x01, 0x01, 0x7f, 0x01, 0x01} // 54 T
    ,{0x3f, 0x40, 0x40, 0x40, 0x3f} // 55 U
    ,{0x1f, 0x20, 0x40, 0x20, 0x1f} // 56 V
    ,{0x3f, 0x40, 0x38, 0x40, 0x3f} // 57 W
    ,{0x63, 0x14, 0x08, 0x14, 0x63} // 58 X
    ,{0x07, 0x08, 0x70, 0x08, 0x07} // 59 Y
    ,{0x61, 0x51, 0x49, 0x45, 0x43} // 5a Z
    ,{0x00, 0x7f, 0x41, 0x41, 0x00} // 5b [
    ,{0x02, 0x04, 0x08, 0x10, 0x20} // 5c ?
    ,{0x00, 0x41, 0x41, 0x7f, 0x00} // 5d ]
    ,{0x04, 0x02, 0x01, 0x02, 0x04} // 5e ^
    ,{0x40, 0x40, 0x40, 0x40, 0x40} // 5f _
    ,{0x00, 0x01, 0x02, 0x04, 0x00} // 60 `
    ,{0x20, 0x54, 0x54, 0x54, 0x78} // 61 a
    ,{0x7f, 0x48, 0x44, 0x44, 0x38} // 62 b
    ,{0x38, 0x44, 0x44, 0x44, 0x20} // 63 c
    ,{0x38, 0x44, 0x44, 0x48, 0x7f} // 64 d
    ,{0x38, 0x54, 0x54, 0x54, 0x18} // 65 e
    ,{0x08, 0x7e, 0x09, 0x01, 0x02} // 66 f
    ,{0x0c, 0x52, 0x52, 0x52, 0x3e} // 67 g
    ,{0x7f, 0x08, 0x04, 0x04, 0x78} // 68 h
    ,{0x00, 0x44, 0x7d, 0x40, 0x00} // 69 i
    ,{0x20, 0x40, 0x44, 0x3d, 0x00} // 6a j
    ,{0x7f, 0x10, 0x28, 0x44, 0x00} // 6b k
    ,{0x00, 0x41, 0x7f, 0x40, 0x00} // 6c l
    ,{0x7c, 0x04, 0x18, 0x04, 0x78} // 6d m
    ,{0x7c, 0x08, 0x04, 0x04, 0x78} // 6e n
    ,{0x38, 0x44, 0x44, 0x44, 0x38} // 6f o
    ,{0x7c, 0x14, 0x14, 0x14, 0x08} // 70 p
    ,{0x08, 0x14, 0x14, 0x18, 0x7c} // 71 q
    ,{0x7c, 0x08, 0x04, 0x04, 0x08} // 72 r
    ,{0x48, 0x54, 0x54, 0x54, 0x20} // 73 s
    ,{0x04, 0x3f, 0x44, 0x40, 0x20} // 74 t
    ,{0x3c, 0x40, 0x40, 0x20, 0x7c} // 75 u
    ,{0x1c, 0x20, 0x40, 0x20, 0x1c} // 76 v
    ,{0x3c, 0x40, 0x30, 0x40, 0x3c} // 77 w
    ,{0x44, 0x28, 0x10, 0x28, 0x44} // 78 x
    ,{0x0c, 0x50, 0x50, 0x50, 0x3c} // 79 y
    ,{0x44, 0x64, 0x54, 0x4c, 0x44} // 7a z
    ,{0x00, 0x08, 0x36, 0x41, 0x00} // 7b {
    ,{0x00, 0x00, 0x7f, 0x00, 0x00} // 7c |
    ,{0x00, 0x41, 0x36, 0x08, 0x00} // 7d }
    ,{0x10, 0x08, 0x08, 0x10, 0x08} // 7e ?
    ,{0x78, 0x46, 0x41, 0x46, 0x78} // 7f ?
};
```

PCD8544.h

```c
#include "STM8S.h"
#include "font.h"


#define BL_pin                          GPIO_PIN_4
#define RST_pin                         GPIO_PIN_7
#define CE_pin                          GPIO_PIN_0
#define DC_pin                          GPIO_PIN_1
#define MOSI_pin                        GPIO_PIN_6
#define SCK_pin                         GPIO_PIN_5
```

```c
#define PCD8544_port_1                      GPIOC
#define PCD8544_port_2                      GPIOG

#define BL_OUT_LOW()                        GPIO_WriteLow(PCD8544_port_1, BL_pin)
#define BL_OUT_HIGH()                       GPIO_WriteHigh(PCD8544_port_1, BL_pin)
#define RST_OUT_LOW()                       GPIO_WriteLow(PCD8544_port_1, RST_pin)
#define RST_OUT_HIGH()                      GPIO_WriteHigh(PCD8544_port_1, RST_pin)
#define CE_OUT_LOW()                        GPIO_WriteLow(PCD8544_port_2, CE_pin)
#define CE_OUT_HIGH()                       GPIO_WriteHigh(PCD8544_port_2, CE_pin)
#define DC_OUT_LOW()                        GPIO_WriteLow(PCD8544_port_2, DC_pin)
#define DC_OUT_HIGH()                       GPIO_WriteHigh(PCD8544_port_2, DC_pin)
#define MOSI_OUT_LOW()                      GPIO_WriteLow(PCD8544_port_1,MOSI_pin)
#define MOSI_OUT_HIGH()                     GPIO_WriteHigh(PCD8544_port_1, MOSI_pin)
#define SCK_OUT_LOW()                       GPIO_WriteLow(PCD8544_port_1, SCK_pin)
#define SCK_OUT_HIGH()                      GPIO_WriteHigh(PCD8544_port_1, SCK_pin)

#define PCD8544_set_Y_addr                  0x40
#define PCD8544_set_X_addr                  0x80

#define PCD8544_set_temp                    0x04
#define PCD8544_set_bias                    0x10
#define PCD8544_set_VOP                     0x80

#define PCD8544_power_down                  0x04
#define PCD8544_entry_mode                  0x02
#define PCD8544_extended_instruction        0x01

#define PCD8544_display_blank               0x00
#define PCD8544_display_normal              0x04
#define PCD8544_display_all_on              0x01
#define PCD8544_display_inverted            0x05

#define PCD8544_function_set                0x20
#define PCD8544_display_control             0x08

#define CMD                                 0
#define DAT                                 1

#define X_max                               84
#define Y_max                               48
#define Rows                                6

#define BLACK                               0
#define WHITE                               1
#define PIXEL_XOR                           2

#define ON                                  1
#define OFF                                 0

#define NO                                  0
#define YES                                 1

#define buffer_size                         504


void setup_GLCD_GPIOs(void);
void PCD8544_write(unsigned char type, unsigned char value);
void PCD8544_reset(void);
```

```c
void PCD8544_init(void);
void PCD8544_backlight_state(unsigned char value);
void PCD8544_set_contrast(unsigned char value);
void PCD8544_set_cursor(unsigned char x_pos, unsigned char y_pos);
void PCD8544_print_char(unsigned char ch, unsigned char colour);
void PCD8544_print_custom_char(unsigned char *map);
void PCD8544_fill(unsigned char bufr);
void PCD8544_clear_buffer(unsigned char colour);
void PCD8544_clear_screen(unsigned char colour);
void PCD8544_print_image(const unsigned char *bmp);
void PCD8544_print_string(unsigned char x_pos, unsigned char y_pos, unsigned char
*ch, unsigned char colour);
void print_char(unsigned char x_pos, unsigned char y_pos, unsigned char ch, unsigned
char colour);
void print_string(unsigned char x_pos, unsigned char y_pos, unsigned char *ch,
unsigned char colour);
void print_chr(unsigned char x_pos, unsigned char y_pos, signed int value, unsigned
char colour);
void print_int(unsigned char x_pos, unsigned char y_pos, signed long value, unsigned
char colour);
void print_decimal(unsigned char x_pos, unsigned char y_pos, unsigned int value,
unsigned char points, unsigned char colour);
void print_float(unsigned char x_pos, unsigned char y_pos, float value, unsigned
char points, unsigned char colour);
void Draw_Pixel(unsigned char x_pos, unsigned char y_pos, unsigned char colour);
void Draw_Line(signed int x1, signed int y1, signed int x2, signed int y2, unsigned
char colour);
void Draw_Rectangle(signed int x1, signed int y1, signed int x2, signed int y2,
unsigned char fill, unsigned char colour);
void Draw_Circle(signed int xc, signed int yc, signed int radius, unsigned char
fill, unsigned char colour);
```

PCD8544.c

```c
#include "PCD8544.h"


extern unsigned char PCD8544_buffer[X_max][Rows];


void setup_GLCD_GPIOs(void)
{
    GPIO_Init(PCD8544_port_1,
            ((GPIO_Pin_TypeDef)(BL_pin | RST_pin | MOSI_pin | SCK_pin)),
             GPIO_MODE_OUT_PP_HIGH_FAST);

    GPIO_Init(PCD8544_port_2,
            ((GPIO_Pin_TypeDef)(DC_pin | CE_pin)),
             GPIO_MODE_OUT_PP_HIGH_FAST);

    delay_ms(10);
}


void PCD8544_write(bool mode, unsigned char value)
{
    unsigned char s = 0x08;
```

```c
    switch(mode)
    {
        case DAT:
        {
            DC_OUT_HIGH();
            break;
        }

        default:
        {
            DC_OUT_LOW();
            break;
        }
    }

    CE_OUT_LOW();

    while(s > 0)
    {
        SCK_OUT_LOW();

        if((value & 0x80) == 0)
        {
            MOSI_OUT_LOW();
        }
        else
        {
            MOSI_OUT_HIGH();
        }

        value <<= 1;
        SCK_OUT_HIGH();
        s--;
    };

    CE_OUT_HIGH();
}


void PCD8544_reset(void)
{
    RST_OUT_LOW();
    delay_us(100);
    RST_OUT_HIGH();
}


void PCD8544_init(void)
{
    setup_GLCD_GPIOs();
    PCD8544_reset();
    PCD8544_write(CMD, (PCD8544_extended_instruction | PCD8544_function_set));
    PCD8544_write(CMD, (PCD8544_set_bias | 0x02));
    PCD8544_set_contrast(0x39);
    PCD8544_write(CMD, PCD8544_set_temp);
    PCD8544_write(CMD, (PCD8544_display_normal | PCD8544_display_control));
    PCD8544_write(CMD, PCD8544_function_set);
    PCD8544_write(CMD, PCD8544_display_all_on);
    PCD8544_write(CMD, PCD8544_display_normal);
```

```c
        PCD8544_clear_buffer(OFF);
}


void PCD8544_backlight_state(bool mode)
{
    switch(mode)
    {
        case ON:
        {
            BL_OUT_LOW();
            break;
        }

        default:
        {
            BL_OUT_HIGH();
            break;
        }
    }
}


void PCD8544_set_contrast(unsigned char value)
{
    PCD8544_write(CMD, (PCD8544_extended_instruction | PCD8544_function_set));
    PCD8544_write(CMD, (PCD8544_set_VOP | (value & 0x7F)));
    PCD8544_write(CMD, PCD8544_function_set);
}


void PCD8544_set_cursor(unsigned char x_pos, unsigned char y_pos)
{
    PCD8544_write(CMD, (PCD8544_set_X_addr | x_pos));
    PCD8544_write(CMD, (PCD8544_set_Y_addr | y_pos));
}


void PCD8544_print_char(unsigned char ch, unsigned char colour)
{
    unsigned char s = 0;
    unsigned char chr = 0;

    for(s = 0; s <= 4; s++)
    {
        chr = font[(ch - 0x20)][s];
        if(colour == BLACK)
        {
            chr = ~chr;
        }
        PCD8544_write(DAT, chr);
    }
}


void PCD8544_print_custom_char(unsigned char *map)
{
    unsigned char s = 0;
```

```c
    for(s = 0; s <= 4; s++)
    {
        PCD8544_write(DAT, *map++);
    }
}


void PCD8544_fill(unsigned char bufr)
{
    unsigned int s = 0;

    PCD8544_set_cursor(0, 0);

    for(s = 0; s < buffer_size; s++)
    {
        PCD8544_write(DAT, bufr);
    }
}


void PCD8544_clear_buffer(unsigned char colour)
{
    unsigned char x_pos = 0;
    unsigned char y_pos = 0;

    for(x_pos; x_pos < X_max; x_pos++)
    {
        for(y_pos; y_pos < Rows; y_pos++)
        {
            PCD8544_buffer[x_pos][y_pos] = colour;
        }
    }
}


void PCD8544_clear_screen(unsigned char colour)
{
    unsigned char x_pos = 0;
    unsigned char y_pos = 0;

    for(y_pos = 0; y_pos < Rows; y_pos++)
    {
        for(x_pos = 0; x_pos < X_max; x_pos++)
        {
            PCD8544_print_string(x_pos, y_pos, " ", colour);
        }
    }
}


void PCD8544_print_image(const unsigned char *bmp)
{
    unsigned int s = 0;

    PCD8544_set_cursor(0, 0);

    for(s = 0; s < buffer_size; s++)
    {
        PCD8544_write(DAT, bmp[s]);
```

```c
        }
}


void PCD8544_print_string(unsigned char x_pos, unsigned char y_pos, unsigned char
*ch, unsigned char colour)
{
    PCD8544_set_cursor(x_pos, y_pos);

    do
    {
        PCD8544_print_char(*ch++, colour);
    }while((*ch >= 0x20) && (*ch <= 0x7F) && (*ch != '\0'));
}


void print_chr(unsigned char x_pos, unsigned char y_pos, signed int value,
unsigned char colour)
{
    unsigned char ch = 0x00;

    if(value < 0)
    {
        PCD8544_set_cursor(x_pos, y_pos);
        PCD8544_print_char(0x2D, colour);
        value = -value;
    }
    else
    {
        PCD8544_set_cursor(x_pos, y_pos);
        PCD8544_print_char(0x20, colour);
    }

     if((value > 99) && (value <= 999))
     {
        ch = (value / 100);
        PCD8544_set_cursor((x_pos + 6), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = ((value % 100) / 10);
        PCD8544_set_cursor((x_pos + 12), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = (value % 10);
        PCD8544_set_cursor((x_pos + 18), y_pos);
        PCD8544_print_char((48 + ch), colour);
     }
     else if((value > 9) && (value <= 99))
     {
        ch = ((value % 100) / 10);
        PCD8544_set_cursor((x_pos + 6), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = (value % 10);
        PCD8544_set_cursor((x_pos + 12), y_pos);
        PCD8544_print_char((48 + ch), colour);

        PCD8544_set_cursor((x_pos + 18), y_pos);
        PCD8544_print_char(0x20, colour);
```

```c
        }
        else if((value >= 0) && (value <= 9))
        {
            ch = (value % 10);
            PCD8544_set_cursor((x_pos + 6), y_pos);
            PCD8544_print_char((48 + ch), colour);

            PCD8544_set_cursor((x_pos + 12), y_pos);
            PCD8544_print_char(0x20, colour);

            PCD8544_set_cursor((x_pos + 18), y_pos);
            PCD8544_print_char(0x20, colour);
        }
}


void print_int(unsigned char x_pos, unsigned char y_pos, signed long value,
unsigned char colour)
{
    unsigned char ch = 0x00;

    if(value < 0)
    {
        PCD8544_set_cursor(x_pos, y_pos);
        PCD8544_print_char(0x2D, colour);
        value = -value;
    }
    else
    {
        PCD8544_set_cursor(x_pos, y_pos);
        PCD8544_print_char(0x20, colour);
    }

    if(value > 9999)
    {
        ch = (value / 10000);
        PCD8544_set_cursor((x_pos + 6), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = ((value % 10000)/ 1000);
        PCD8544_set_cursor((x_pos + 12), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = ((value % 1000) / 100);
        PCD8544_set_cursor((x_pos + 18), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = ((value % 100) / 10);
        PCD8544_set_cursor((x_pos + 24), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = (value % 10);
        PCD8544_set_cursor((x_pos + 30), y_pos);
        PCD8544_print_char((48 + ch), colour);
    }

    else if((value > 999) && (value <= 9999))
    {
        ch = ((value % 10000)/ 1000);
```

```c
        PCD8544_set_cursor((x_pos + 6), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = ((value % 1000) / 100);
        PCD8544_set_cursor((x_pos + 12), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = ((value % 100) / 10);
        PCD8544_set_cursor((x_pos + 18), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = (value % 10);
        PCD8544_set_cursor((x_pos + 24), y_pos);
        PCD8544_print_char((48 + ch), colour);

        PCD8544_set_cursor((x_pos + 30), y_pos);
        PCD8544_print_char(0x20, colour);
    }
    else if((value > 99) && (value <= 999))
    {
        ch = ((value % 1000) / 100);
        PCD8544_set_cursor((x_pos + 6), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = ((value % 100) / 10);
        PCD8544_set_cursor((x_pos + 12), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = (value % 10);
        PCD8544_set_cursor((x_pos + 18), y_pos);
        PCD8544_print_char((48 + ch), colour);

        PCD8544_set_cursor((x_pos + 24), y_pos);
        PCD8544_print_char(0x20, colour);

        PCD8544_set_cursor((x_pos + 30), y_pos);
        PCD8544_print_char(0x20, colour);
    }
    else if((value > 9) && (value <= 99))
    {
        ch = ((value % 100) / 10);
        PCD8544_set_cursor((x_pos + 6), y_pos);
        PCD8544_print_char((48 + ch), colour);

        ch = (value % 10);
        PCD8544_set_cursor((x_pos + 12), y_pos);
        PCD8544_print_char((48 + ch), colour);

        PCD8544_set_cursor((x_pos + 18), y_pos);
        PCD8544_print_char(0x20, colour);

        PCD8544_set_cursor((x_pos + 24), y_pos);
        PCD8544_print_char(0x20, colour);

        PCD8544_set_cursor((x_pos + 30), y_pos);
        PCD8544_print_char(0x20, colour);
    }
    else
    {
```

```c
            ch = (value % 10);
            PCD8544_set_cursor((x_pos + 6), y_pos);
            PCD8544_print_char((48 + ch), colour);

            PCD8544_set_cursor((x_pos + 12), y_pos);
            PCD8544_print_char(0x20, colour);

            PCD8544_set_cursor((x_pos + 18), y_pos);
            PCD8544_print_char(0x20, colour);

            PCD8544_set_cursor((x_pos + 24), y_pos);
            PCD8544_print_char(0x20, colour);

            PCD8544_set_cursor((x_pos + 30), y_pos);
            PCD8544_print_char(0x20, colour);
        }
}


void print_decimal(unsigned char x_pos, unsigned char y_pos, unsigned int value,
unsigned char points, unsigned char colour)
{
    unsigned char ch = 0x00;

    PCD8544_set_cursor(x_pos, y_pos);
    PCD8544_print_char(0x2E, colour);

    ch = (value / 1000);
    PCD8544_set_cursor((x_pos + 6), y_pos);
    PCD8544_print_char((48 + ch), colour);

    if(points > 1)
    {
        ch = ((value % 1000) / 100);
        PCD8544_set_cursor((x_pos + 12), y_pos);
        PCD8544_print_char((48 + ch), colour);


        if(points > 2)
        {
            ch = ((value % 100) / 10);
            PCD8544_set_cursor((x_pos + 18), y_pos);
            PCD8544_print_char((48 + ch), colour);

            if(points > 3)
            {
                ch = (value % 10);
                PCD8544_set_cursor((x_pos + 24), y_pos);
                PCD8544_print_char((48 + ch), colour);;
            }
        }
    }
}


void print_float(unsigned char x_pos, unsigned char y_pos, float value, unsigned
char points, unsigned char colour)
{
    signed long tmp = 0x00;
```

```c
    tmp = ((signed long)value);
    print_int(x_pos, y_pos, tmp, colour);
    tmp = ((value - tmp) * 10000);

    if(tmp < 0)
    {
        tmp = -tmp;
    }

    if((value >= 10000) && (value < 100000))
    {
        print_decimal((x_pos + 36), y_pos, tmp, points, colour);
    }
    else if((value >= 1000) && (value < 10000))
    {
        print_decimal((x_pos + 30), y_pos, tmp, points, colour);
    }
    else if((value >= 100) && (value < 1000))
    {
        print_decimal((x_pos + 24), y_pos, tmp, points, colour);
    }
    else if((value >= 10) && (value < 100))
    {
        print_decimal((x_pos + 18), y_pos, tmp, points, colour);
    }
    else if(value < 10)
    {
        print_decimal((x_pos + 12), y_pos, tmp, points, colour);
        if(value < 0)
        {
            PCD8544_set_cursor(x_pos, y_pos);
            PCD8544_print_char(0x2D, colour);
        }
        else
        {
            PCD8544_set_cursor(x_pos, y_pos);
            PCD8544_print_char(0x20, colour);
        }
    }
}

void Draw_Pixel(unsigned char x_pos, unsigned char y_pos, unsigned char colour)
{
    unsigned char row = 0;
    unsigned char value = 0;

    if((x_pos < 0) || (x_pos >= X_max) || (y_pos < 0) || (y_pos >= Y_max))
    {
        return;
    }

    row = (y_pos >> 3);

    value = PCD8544_buffer[x_pos][row];

    if(colour == BLACK)
    {
```

```c
            value |= (1 << (y_pos % 8));
        }
        else if(colour == WHITE)
        {
            value &= (~(1 << (y_pos % 8)));
        }
        else if(colour == PIXEL_XOR)
        {
            value ^= (1 << (y_pos % 8));
        }

        PCD8544_buffer[x_pos][row] = value;

        PCD8544_set_cursor(x_pos, row);
        PCD8544_write(DAT, value);
}


void Draw_Line(signed int x1, signed int y1, signed int x2, signed int y2,
unsigned char colour)
{
    signed int dx = 0x0000;
    signed int dy = 0x0000;
    signed int stepx = 0x0000;
    signed int stepy = 0x0000;
    signed int fraction = 0x0000;

    dy = (y2 - y1);
    dx = (x2 - x1);

    if (dy < 0)
    {
        dy = -dy;
        stepy = -1;
    }
    else
    {
        stepy = 1;
    }

    if (dx < 0)
    {
        dx = -dx;
        stepx = -1;
    }
    else
    {
        stepx = 1;
    }

    dx <<= 0x01;
    dy <<= 0x01;

    Draw_Pixel(x1, y1, colour);

    if (dx > dy)
    {
        fraction = (dy - (dx >> 1));
        while (x1 != x2)
```

```c
            {
                if (fraction >= 0)
                {
                    y1 += stepy;
                    fraction -= dx;
                }
                x1 += stepx;
                fraction += dy;

                Draw_Pixel(x1, y1, colour);
            }
        }
        else
        {
            fraction = (dx - (dy >> 1));

            while (y1 != y2)
            {
                if (fraction >= 0)
                {
                    x1 += stepx;
                    fraction -= dy;
                }
                y1 += stepy;
                fraction += dx;
                Draw_Pixel(x1, y1, colour);
            }
        }
}

void Draw_Rectangle(signed int x1, signed int y1, signed int x2, signed int y2,
unsigned char fill, unsigned char colour)
{
    unsigned char i = 0x00;
    unsigned char xmin = 0x00;
    unsigned char xmax = 0x00;
    unsigned char ymin = 0x00;
    unsigned char ymax = 0x00;

    if(fill != NO)
    {
        if(x1 < x2)
        {
            xmin = x1;
            xmax = x2;
        }
        else
        {
            xmin = x2;
            xmax = x1;
        }

        if(y1 < y2)
        {
            ymin = y1;
            ymax = y2;
        }
        else
```

```c
        {
            ymin = y2;
            ymax = y1;
        }

        for(; xmin <= xmax; ++xmin)
        {
            for(i = ymin; i <= ymax; ++i)
            {
                Draw_Pixel(xmin, i, colour);
            }
        }
    }

    else
    {
        Draw_Line(x1, y1, x2, y1, colour);
        Draw_Line(x1, y2, x2, y2, colour);
        Draw_Line(x1, y1, x1, y2, colour);
        Draw_Line(x2, y1, x2, y2, colour);
    }
}

void Draw_Circle(signed int xc, signed int yc, signed int radius, unsigned char
fill, unsigned char colour)
{
    signed int a = 0x0000;
    signed int b = 0x0000;
    signed int p = 0x0000;

    b = radius;
    p = (1 - b);

    do
    {
        if(fill != NO)
        {
            Draw_Line((xc - a), (yc + b), (xc + a), (yc + b), colour);
            Draw_Line((xc - a), (yc - b), (xc + a), (yc - b), colour);
            Draw_Line((xc - b), (yc + a), (xc + b), (yc + a), colour);
            Draw_Line((xc - b), (yc - a), (xc + b), (yc - a), colour);
        }
        else
        {
            Draw_Pixel((xc + a), (yc + b), colour);
            Draw_Pixel((xc + b), (yc + a), colour);
            Draw_Pixel((xc - a), (yc + b), colour);
            Draw_Pixel((xc - b), (yc + a), colour);
            Draw_Pixel((xc + b), (yc - a), colour);
            Draw_Pixel((xc + a), (yc - b), colour);
            Draw_Pixel((xc - a), (yc - b), colour);
            Draw_Pixel((xc - b), (yc - a), colour);
        }

        if(p < 0)
        {
            p += (0x03 + (0x02 * a++));
        }
```

```
        else
        {
            p += (0x05 + (0x02 * ((a++) - (b--))));
        }
    }while(a <= b);
}
```

main.c

```c
#include "STM8S.h"
#include "PCD8544.h"


unsigned char PCD8544_buffer[X_max][Rows];


void setup_clock(void);
void setup_GPIOs(void);


void main(void)
{
    unsigned char g = 0;
    const unsigned char txt1[11] = {"MicroArena"};
    const unsigned char txt2[11] = {"SSHAHRYIAR"};

    signed char c = -9;
    signed int i = -66;
    float f = -0.04;

    setup_clock();
    setup_GPIOs();
    PCD8544_init();

    PCD8544_backlight_state(ON);
    delay_ms(1000);
    PCD8544_backlight_state(OFF);
    delay_ms(1000);

    PCD8544_clear_screen(WHITE);

    PCD8544_backlight_state(ON);
    Draw_Rectangle(2, 2, 81, 45, OFF, BLACK);
    delay_ms(200);

    Draw_Circle(6, 6, 2, ON, BLACK);
    delay_ms(200);
    Draw_Circle(77, 6, 2, ON, BLACK);
    delay_ms(200);
    Draw_Circle(77, 41, 2, ON, BLACK);
    delay_ms(200);
    Draw_Circle(6, 41, 2, ON, BLACK);
    delay_ms(200);

    Draw_Line(2, 11, 10, 11, BLACK);
    Draw_Line(73, 11, 81, 11, BLACK);
    delay_ms(200);
    Draw_Line(2, 36, 10, 36, BLACK);
```

```c
        Draw_Line(73, 36, 81, 36, BLACK);
        delay_ms(200);
        Draw_Line(11, 45, 11, 2, BLACK);
        Draw_Line(72, 45, 72, 2, BLACK);
        delay_ms(200);

        PCD8544_backlight_state(OFF);
        delay_ms(400);

        PCD8544_backlight_state(ON);

        for(g = 0; g <= 9; g++)
        {
            PCD8544_set_cursor((18 + (g * 5)), 2);
            PCD8544_print_char(txt1[g], WHITE);
            delay_ms(90);
        }

        for(g = 0; g <= 9; g++)
        {
            PCD8544_set_cursor((18 + (g * 5)), 3);
            PCD8544_print_char(txt2[g], WHITE);
            delay_ms(90);
        }
        delay_ms(4000);

        PCD8544_clear_screen(WHITE);

        PCD8544_print_string(1, 2, "CHR:", WHITE);
        PCD8544_print_string(1, 3, "INT:", WHITE);
        PCD8544_print_string(1, 4, "FLT:", WHITE);

        while(1)
        {
            print_chr(26, 2, c, WHITE);
            print_int(26, 3, i, WHITE);
            print_float(26, 4, f, 2, WHITE);
            c++;
            i++;
            f += 0.01;
            delay_ms(400);
        };
}


void setup_clock(void)
{
        CLK_DeInit();

        CLK_HSECmd(ENABLE);
        while(CLK_GetFlagStatus(CLK_FLAG_HSERDY) == FALSE);

        CLK_LSICmd(DISABLE);

        CLK_HSICmd(ENABLE);
        while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

        CLK_ClockSwitchCmd(ENABLE);
        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
```

```c
        CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

        CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSE,
        DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER3, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void setup_GPIOs(void)
{
        GPIO_DeInit(PCD8544_port_1);
        GPIO_DeInit(PCD8544_port_2);
}
```

Explanation

This display shares the same function demonstrated in the OLED display and so I am not going to repeat their uses again.

Demo

Video link: https://www.youtube.com/watch?v=TLNRtVpHabY.
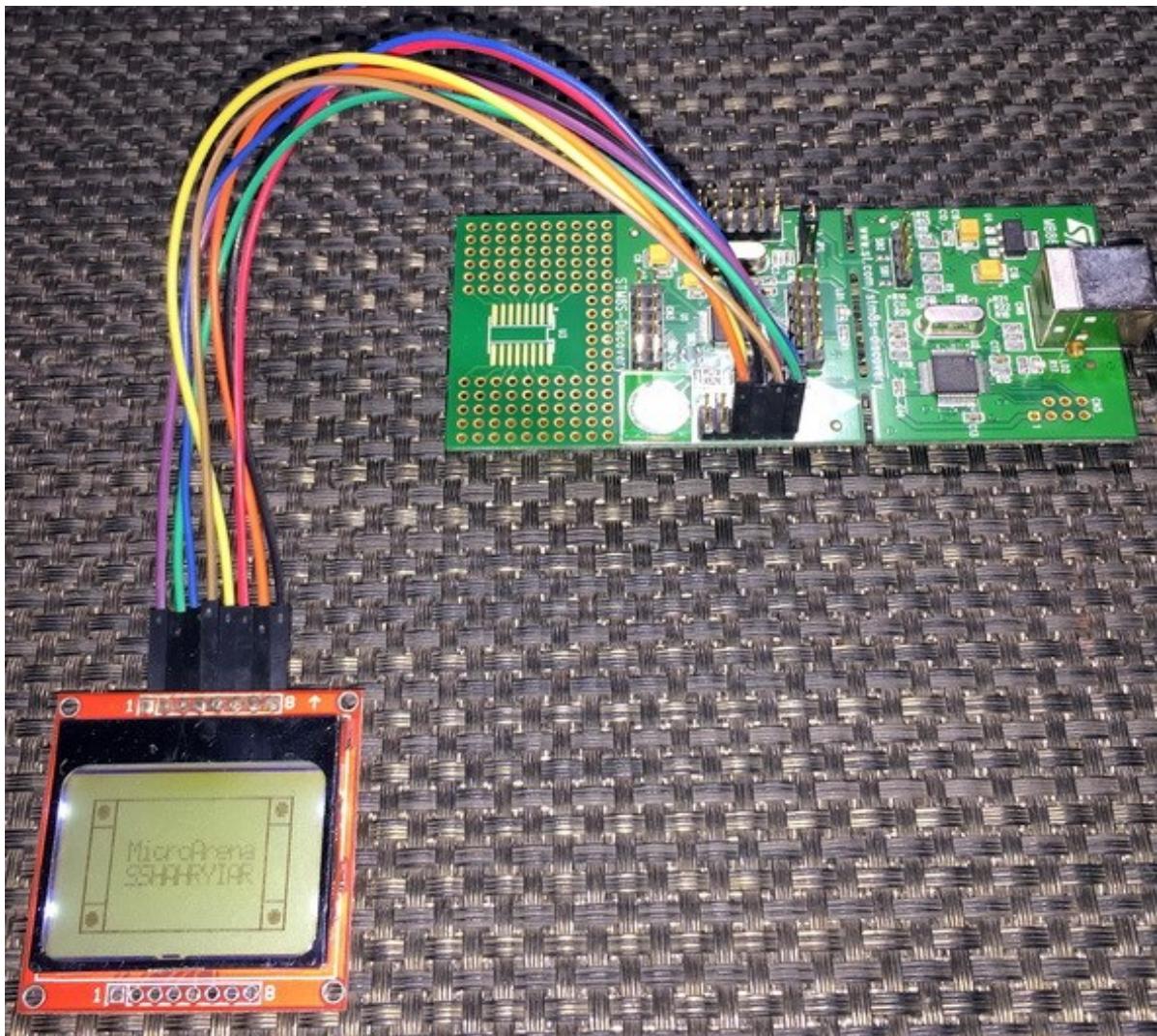
Capacitive Touch Overview

Electronics and embedded-systems advancing fast. Designers and users are moving toward electronic solutions over mechanical ones. Touch screens, touchpads and touch switches are getting popular like never before and are replacing traditional mechanical switches, knobs, etc. ST like many other microcontroller manufacturers have provided resources for implementing capacitive touch buttons. However, the algorithm for implementing reliable capacitive touch sensors is a bit complicated especially when it comes to an 8-bit microcontroller. For this reason, ST has provided capacitive touch libraries for its STM8 and STM32 product line. These libraries along with SPL enable us to quickly and effectively add capacitive touch functionality in our designs.
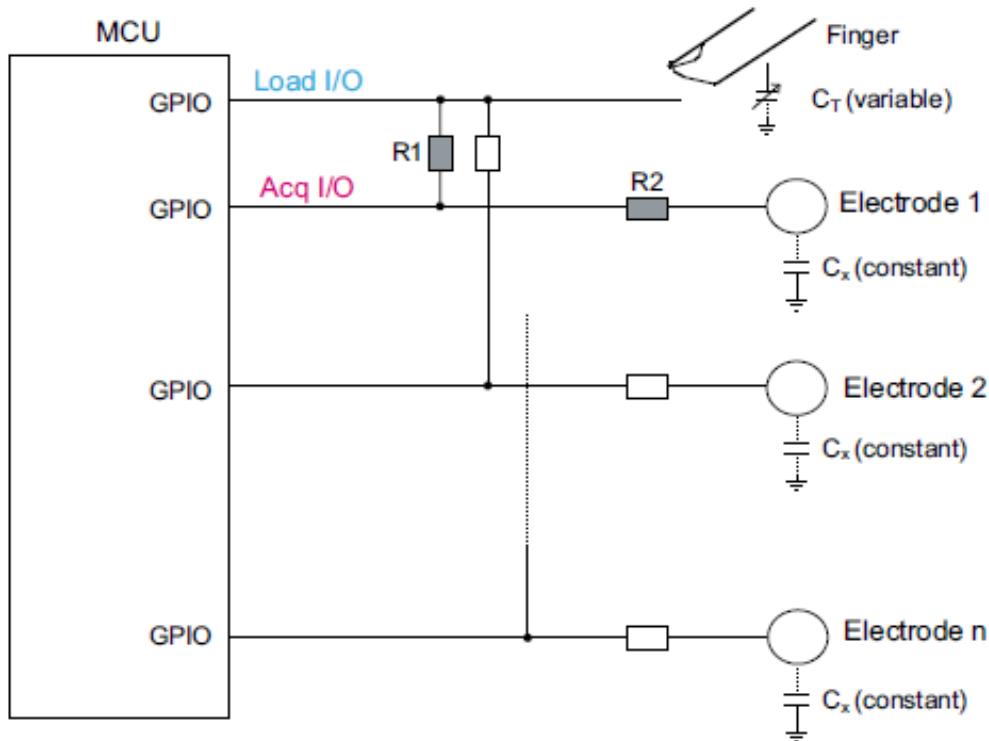
In STM8s, RC acquisition principle is used to implement capacitive touch sensors. No special hardware is required for implementing RC-based capacitive touch buttons/sliders/wheels. Just timers and GPIOs are all that are needed to get the job done. RC acquisition method detects a human touch on a capacitive touch sensor (acquisition pin/pins) by measuring small variations of the touch electrode's capacitance. Capacitance exists between acquisition electrodes and ground. The touch electrode capacitance is periodically charged and discharged through a fixed resistor called load resistor by altering the logic state of the load I/O. A touch electrode's capacitance depends on its area, relative dielectric constant of the insulator in-between, the relative permittivity of air and the distance between the two electrodes. Additionally, an active shield is used to get rid of parasitic capacitance between touch electrode(s) and ground. Parasitic capacitances originate due to environmental effects and other foreign objects. An active shield surrounds touch electrode(s) and maintain same in-phase potential. This leads to zero potential difference between the electrode(s) and shield, and thereby no parasitic capacitance builds up.

ST's application note _**AN2927**_ details capacitive touch sensing.

Single Capacitive Touch

The STM8S105C6T6 Discovery board features a capacitive touch button. A part of this Discovery board's schematic is shown below:
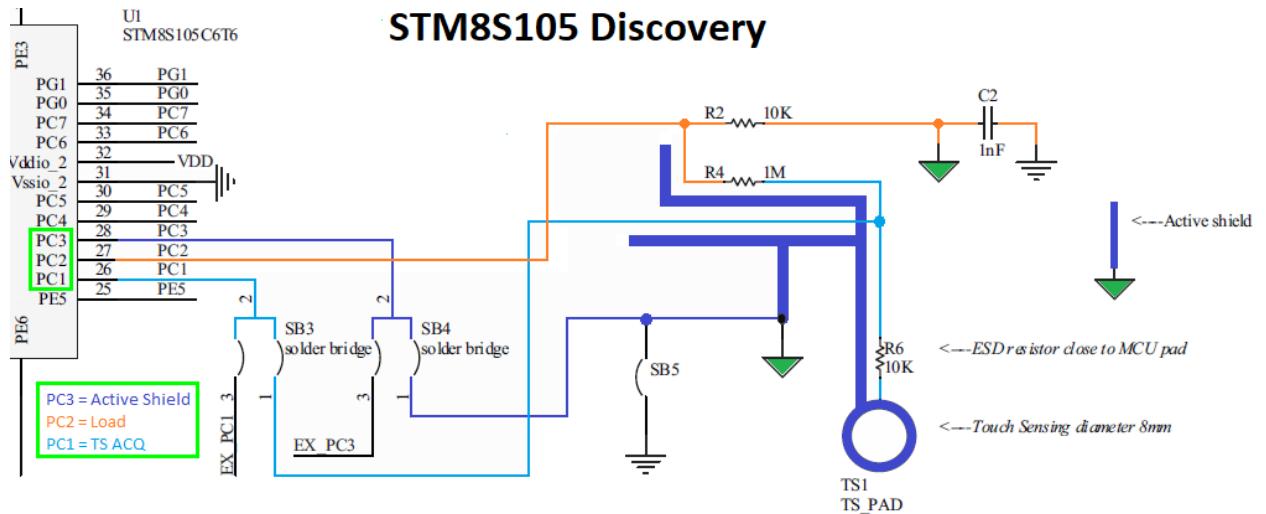
Note the highlighted sections. These are what that should be focused on for now. The purple lines on the right side are parts of the active shield. The orange line is the load or charge-discharge I/O. The light blue lines represent the touch acquisition I/O. The physical attributes of the capacitive touch button is documented in the user manual of the STM8S105 Discovery board. These properties are just as important as the electrical diagram of the touch button because these physical properties set its characteristics.

Hardware Connection
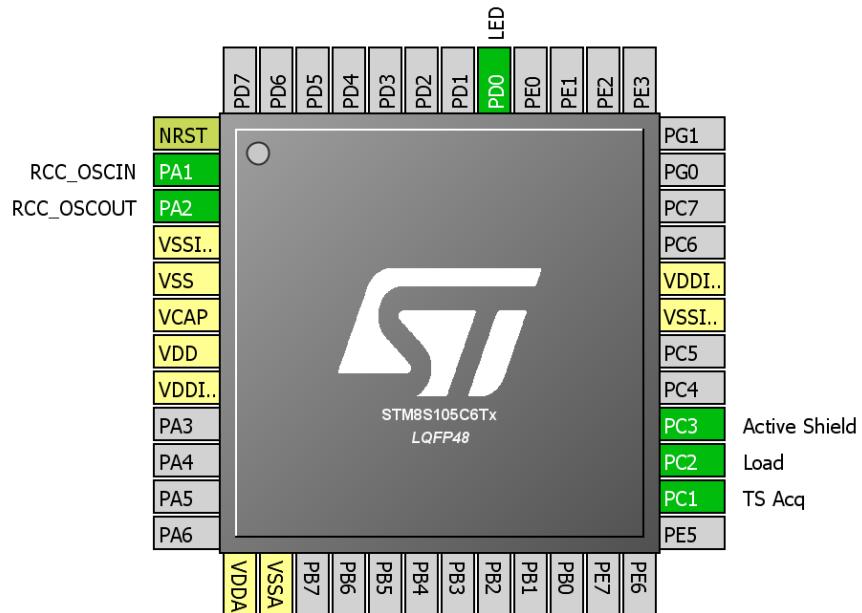
Code Example

stm8_tsl_conf.h

```c
/* Define to prevent recursive inclusion ------------------------------------*/
#ifndef __TSL_CONF_H
#define __TSL_CONF_H



//==============================================================================
//
// 1) MCU SELECTION
//
// 1 = STM8S/A
//
//==============================================================================

#define MCU_SELECTION (1)



//==============================================================================
//
// 2) ACQUISITION TIMER SELECTION (TIMACQ)
//
// Set the acquisition timer and its counter high register address.
//
// The timer you select must be a *16-bit timer*, have a *8-bit prescaler* and
// must be different of the TIMTICK timer described below (TIM1, TIM2 or TIM3
// for example).
//
//==============================================================================

#define TIMACQ          (TIM3)
#define TIMACQ_CNTR_ADD  (0x5328)



//==============================================================================
//
// 3) GENERIC TIMEBASE TIMER SELECTION (TIMTICK)
//
// Set the generic timebase timer.
//
// The timer you select must be a *basic 8-bit timer* and must be different
// of the TIMACQ timer described above (TIM4 for example).
//
// Warning: The selected timer update/overflow interrupt vector must point to
// the TSL_Timer_ISR() interrupt routine.
//
//==============================================================================

#define TIMTICK  (TIM4)



//==============================================================================
//
// 4) REFERENCE LOAD I/O DEFINITION
//
// Set the port
```

```c
// Set the pin mask
//
//==============================================================================

#define LOADREF_PORT_ADDR   (GPIOC_BaseAddress)  /**< LOADREF pin GPIO base address
*/

#define LOADREF_BIT         (0x04)              /**< LOADREF pin mask */



//==============================================================================
//
// 5) SINGLE CHANNEL KEYS DEFINITION - PORT 1
//
// Set the number of keys
// Set the port
// Set the pins mask
//
// Warning: This port is mandatory and one key at least must be defined.
//
//==============================================================================

#define SCKEY_P1_KEY_COUNT  (1)  /**< Single channel key Port 1: Number of keys
used (value from 1 to 8) */

#define SCKEY_P1_PORT_ADDR  (GPIOC_BaseAddress)  /**< Single channel key Port 1:
GPIO base address */

#define SCKEY_P1_A  (0x02)  /**< Single channel key Port 1: 1st key mask */
#define SCKEY_P1_B  (0)  /**< Single channel key Port 1: 2nd key mask */
#define SCKEY_P1_C  (0)  /**< Single channel key Port 1: 3rd key mask */
#define SCKEY_P1_D  (0)  /**< Single channel key Port 1: 4th key mask */
#define SCKEY_P1_E  (0)     /**< Single channel key Port 1: 5th key mask */
#define SCKEY_P1_F  (0)     /**< Single channel key Port 1: 6th key mask */
#define SCKEY_P1_G  (0)     /**< Single channel key Port 1: 7th key mask */
#define SCKEY_P1_H  (0)     /**< Single channel key Port 1: 8th key mask */

#define SCKEY_P1_DRIVEN_SHIELD_MASK (0x08)



//==============================================================================
//
// 6) SINGLE CHANNEL KEYS DEFINITION - PORT 2
//
// Set the number of keys
// Set the port
// Set the pins mask
//
// Note: This port is optional. Set SCKEY_P2_KEY_COUNT to 0 to not use this port.
//
//==============================================================================

#define SCKEY_P2_KEY_COUNT  (0)  /**< Single channel key Port 2: Number of keys
used (value from 0 to 8) */

#define SCKEY_P2_PORT_ADDR  (GPIOD_BaseAddress)  /**< Single channel key Port 2:
GPIO base address */

#define SCKEY_P2_A  (0)     /**< Single channel key Port 2: 1st key mask */
```

```c
#define SCKEY_P2_B  (0)        /**< Single channel key Port 2: 2nd key mask */
#define SCKEY_P2_C  (0)        /**< Single channel key Port 2: 3rd key mask */
#define SCKEY_P2_D  (0)        /**< Single channel key Port 2: 4th key mask */
#define SCKEY_P2_E  (0)        /**< Single channel key Port 2: 5th key mask */
#define SCKEY_P2_F  (0)        /**< Single channel key Port 2: 6th key mask */
#define SCKEY_P2_G  (0)        /**< Single channel key Port 2: 7th key mask */
#define SCKEY_P2_H  (0)        /**< Single channel key Port 2: 8th key mask */

#define SCKEY_P2_DRIVEN_SHIELD_MASK (0x00)



//================================================================================
//
// 7) SINGLE CHANNEL KEYS DEFINITION - PORT 3
//
// Set the number of keys
// Set the port
// Set the pins mask
//
// Note: This port is optional. Set SCKEY_P3_KEY_COUNT to 0 to not use this port.
//
//================================================================================

#define SCKEY_P3_KEY_COUNT  (0)  /**< Single channel key Port 3: Number of keys
used (value from 0 to 8) */

#define SCKEY_P3_PORT_ADDR  (GPIOE_BaseAddress)  /**< Single channel key Port 3:
GPIO base address */

#define SCKEY_P3_A  (0)        /**< Single channel key Port 3: 1st key mask */
#define SCKEY_P3_B  (0)        /**< Single channel key Port 3: 2nd key mask */
#define SCKEY_P3_C  (0)        /**< Single channel key Port 3: 3rd key mask */
#define SCKEY_P3_D  (0)        /**< Single channel key Port 3: 4th key mask */
#define SCKEY_P3_E  (0)        /**< Single channel key Port 3: 5th key mask */
#define SCKEY_P3_F  (0)        /**< Single channel key Port 3: 6th key mask */
#define SCKEY_P3_G  (0)        /**< Single channel key Port 3: 7th key mask */
#define SCKEY_P3_H  (0)        /**< Single channel key Port 3: 8th key mask */

#define SCKEY_P3_DRIVEN_SHIELD_MASK (0x00)



//================================================================================
//
// 8) NUMBER OF MULTI CHANNEL KEYS AND NUMBER OF CHANNELS USED
//
// Set the total number of multi channel keys used (0, 1 or 2)
// Set the number of channels (5 or 8)
//
//================================================================================

#define NUMBER_OF_MULTI_CHANNEL_KEYS  (0)  /**< Number of multi channel keys
(value from 0 to 2) */
#define CHANNEL_PER_MCKEY             (5)  /**< Number of channels per key
(possible values are 5 or 8 only) */



//================================================================================
//
// 9) MULTI CHANNEL KEY 1 DEFINITION
```

```c
//
// Set the port used
// Set the pins mask
//
// Note: This key is optional
//
//==============================================================================

#if NUMBER_OF_MULTI_CHANNEL_KEYS > 0

#define MCKEY1_A_PORT_ADDR  (GPIOD_BaseAddress)  /**< Multi channel key 1: 1st
channel port */
#define MCKEY1_A            (0x40)               /**< Multi channel key 1: 1st
channel mask */
#define MCKEY1_B_PORT_ADDR  (GPIOD_BaseAddress)  /**< Multi channel key 1: 2nd
channel port */
#define MCKEY1_B            (0x20)               /**< Multi channel key 1: 2nd
channel mask */
#define MCKEY1_C_PORT_ADDR  (GPIOD_BaseAddress)  /**< Multi channel key 1: 3rd
channel port */
#define MCKEY1_C            (0x10)               /**< Multi channel key 1: 3rd
channel mask */
#define MCKEY1_D_PORT_ADDR  (GPIOD_BaseAddress)  /**< Multi channel key 1: 4th
channel port */
#define MCKEY1_D            (0x08)               /**< Multi channel key 1: 4th
channel mask */
#define MCKEY1_E_PORT_ADDR  (GPIOD_BaseAddress)  /**< Multi channel key 1: 5th
channel port */
#define MCKEY1_E            (0x04)               /**< Multi channel key 1: 5th
channel mask */
#define MCKEY1_F_PORT_ADDR  (0)                  /**< Multi channel key 1: 6th
channel port */
#define MCKEY1_F            (0)                  /**< Multi channel key 1: 6th
channel mask */
#define MCKEY1_G_PORT_ADDR  (0)                  /**< Multi channel key 1: 7th
channel port */
#define MCKEY1_G            (0)                  /**< Multi channel key 1: 7th
channel mask */
#define MCKEY1_H_PORT_ADDR  (0)                  /**< Multi channel key 1: 8th
channel port */
#define MCKEY1_H            (0)                  /**< Multi channel key 1: 8th
channel mask */

#define MCKEY1_TYPE (0)                          /**< Multi channel key 1 type:
0=wheel (zero between two electrodes), 1=slider (zero in the middle of one
electrode) */
#define MCKEY1_LAYOUT_TYPE (0)                   /**< Multi channel key 1 layout
type: 0=interlaced, 1=normal */

#define MCKEY1_DRIVEN_SHIELD_MASK (0x00)

#endif



//==============================================================================
//
// 10) MULTI CHANNEL KEY 2 DEFINITION
//
// Set the port used
```

```c
// Set the pins mask
//
// Note: This key is optional.
//
//============================================================================

#if NUMBER_OF_MULTI_CHANNEL_KEYS > 1

#define MCKEY2_A_PORT_ADDR  (GPIOE_BaseAddress)  /**< Multi channel key 2: 1st
channel port */
#define MCKEY2_A            (0x01)               /**< Multi channel key 2: 1st
channel mask */
#define MCKEY2_B_PORT_ADDR  (GPIOE_BaseAddress)  /**< Multi channel key 2: 2nd
channel port */
#define MCKEY2_B            (0x02)               /**< Multi channel key 2: 2nd
channel mask */
#define MCKEY2_C_PORT_ADDR  (GPIOE_BaseAddress)  /**< Multi channel key 2: 3rd
channel port */
#define MCKEY2_C            (0x04)               /**< Multi channel key 2: 3rd
channel mask */
#define MCKEY2_D_PORT_ADDR  (GPIOE_BaseAddress)  /**< Multi channel key 2: 4th
channel port */
#define MCKEY2_D            (0x08)               /**< Multi channel key 2: 4th
channel mask */
#define MCKEY2_E_PORT_ADDR  (GPIOE_BaseAddress)  /**< Multi channel key 2: 5th
channel port */
#define MCKEY2_E            (0x10)               /**< Multi channel key 2: 5th
channel mask */
#define MCKEY2_F_PORT_ADDR  (0)                  /**< Multi channel key 2: 6th
channel port */
#define MCKEY2_F            (0)                  /**< Multi channel key 2: 6th
channel mask */
#define MCKEY2_G_PORT_ADDR  (0)                  /**< Multi channel key 2: 7th
channel port */
#define MCKEY2_G            (0)                  /**< Multi channel key 2: 7th
channel mask */
#define MCKEY2_H_PORT_ADDR  (0)                  /**< Multi channel key 2: 8th
channel port */
#define MCKEY2_H            (0)                  /**< Multi channel key 2: 8th
channel mask */

#define MCKEY2_TYPE (0)                          /**< Multi channel key 2 type:
0=wheel (zero between two electrodes), 1=slider (zero in the middle of one
electrode) */
#define MCKEY2_LAYOUT_TYPE (0)                   /**< Multi channel key 2 layout
type: 0=interlaced, 1=normal */

#define MCKEY2_DRIVEN_SHIELD_MASK (0x00)

#endif


//============================================================================
//
// 11) ELECTRODES MASKS USED ON EACH GPIO
//
// Define the electrodes mask for each GPIO used (SCKeys + MCKeys but not LOADREF)
//
//============================================================================
```

```c
#define GPIOA_ELECTRODES_MASK  (0x00)   /**< Electrodes mask for GPIOA */
#define GPIOB_ELECTRODES_MASK  (0x00)   /**< Electrodes mask for GPIOB */
#define GPIOC_ELECTRODES_MASK  (0x0A)   /**< Electrodes mask for GPIOC */
#define GPIOD_ELECTRODES_MASK  (0x00)   /**< Electrodes mask for GPIOD */
#define GPIOE_ELECTRODES_MASK  (0x00)   /**< Electrodes mask for GPIOE */
#define GPIOF_ELECTRODES_MASK  (0x00)   /**< Electrodes mask for GPIOF */
#define GPIOG_ELECTRODES_MASK  (0x00)   /**< Electrodes mask for GPIOG */
#define GPIOH_ELECTRODES_MASK  (0x00)   /**< Electrodes mask for GPIOH */
#define GPIOI_ELECTRODES_MASK  (0x00)   /**< Electrodes mask for GPIOI */


//==============================================================================
//
// 12) TSL PARAMETERS CONFIGURATION
//
//==============================================================================

/** @addtogroup TSL_parameters_RC
  * @{ */

// IO acquisition
#define SCKEY_ACQ_NUM                     (3)  /**< Single channel key acquisition
number - N (value from 1 to 255) */
#define SCKEY_ADJUST_LEVEL                (1)  /**< Single channel key adjustment
level (value from 0 to 255) */
#define MCKEY_ACQ_NUM                     (6)  /**< Multi channel key acquisition
number - N (value from 1 to 255) */
#define MCKEY_ADJUST_LEVEL                (1)  /**< Multi channel key adjustment
level (value from 0 to 255) */

// IO acquisition number of rejected values and measure guardbands
#define MAX_REJECTED_MEASUREMENTS      (5)   /**< Max number of rejected
measurements allowed (value from 0 to 255) */
#define MAX_MEAS_COEFF             (0x011A)  /**< Max measure guardband
(MSB=integer part, LSB=decimal part) */
#define MIN_MEAS_COEFF             (0x00E6)  /**< Min measure guardband
(MSB=integer part, LSB=decimal part) */

// Thresholds
#define SCKEY_DETECTTHRESHOLD_DEFAULT         (10)  /**< Single channel key
detection threshold (value from 1 to 127) */
#define SCKEY_ENDDETECTTHRESHOLD_DEFAULT       (8)  /**< Single channel key end
detection threshold (value from 1 to 127) */
#define SCKEY_RECALIBRATIONTHRESHOLD_DEFAULT (-10)  /**< Single channel key
calibration threshold (value from -1 to -128) */
#define MCKEY_DETECTTHRESHOLD_DEFAULT         (30)  /**< Multi channel key
detection threshold (value from 1 to 127) */
#define MCKEY_ENDDETECTTHRESHOLD_DEFAULT      (20)  /**< Multi channel key end
detection threshold (value from 1 to 127) */
#define MCKEY_RECALIBRATIONTHRESHOLD_DEFAULT (-30)  /**< Multi channel key
calibration threshold (value from -1 to -128) */

// MCKey resolution
#define MCKEY_RESOLUTION_DEFAULT                  (4)  /**< Multi channel key
resolution (value from 1 to 8) */

// MCKey Direction Change process
```

```c
#define MCKEY_DIRECTION_CHANGE_ENABLED               (1)   /**< Multi channel key
direction change enable (1) or disable (0) switch */
#define MCKEY_DIRECTION_CHANGE_MAX_DISPLACEMENT     (255)  /**< Multi channel key
direction change maximum displacement (value from 0 to 255) */
#define MCKEY_DIRECTION_CHANGE_INTEGRATOR_DEFAULT    (1)   /**< Multi channel key
direction change integrator (value from 1 to 255) */
#define MCKEY_DIRECTION_CHANGE_THRESHOLD_DEFAULT    (10)   /**< Multi channel key
direction change threshold (value from 1 to 255) */

// Integrators
#define DETECTION_INTEGRATOR_DEFAULT        (2)   /**< Detection Integrator =
Debounce Filter (value from 0 to 255) */
#define END_DETECTION_INTEGRATOR_DEFAULT    (2)   /**< End detection Integrator =
Debounce Filter (from 0 to 255) */
#define RECALIBRATION_INTEGRATOR_DEFAULT   (10)   /**< Calibration integrator (value
from 1 to 255) */

// IIR Filter
#define ECS_TIME_STEP_DEFAULT  (20)   /**< Sampling frequency, multiple of 10ms */
#define ECS_TEMPO_DEFAULT      (20)   /**< Delay after detection, multiple of 100ms
*/
#define ECS_IIR_KFAST_DEFAULT  (20)   /**< K factor for fast filtering */
#define ECS_IIR_KSLOW_DEFAULT  (10)   /**< K factor for slow filtering */

// Detection Timeout
#define DTO_DEFAULT  (0)   /**< 1s unit (value from 0 (= infinite!) to 255) */

// Automatic Calibration
#define NEGDETECT_AUTOCAL (1)   /**< 0 (Enable negative threshold for noise), 1
(Enable autocalibration)  */

// Acquisition values limits
#define SCKEY_MIN_ACQUISITION    (50) /**< Single channel key minimum acquisition
value */
#define SCKEY_MAX_ACQUISITION  (3000) /**< Single channel key maximum acquisition
value */
#define MCKEY_MIN_ACQUISITION   (150) /**< Multi channel key minimum acquisition
value */
#define MCKEY_MAX_ACQUISITION  (5000) /**< Multi channel key maximum acquisition
value */

// Optional parameters for Delta Normalization Process (for Multi channel keys
only).
// The MSB is the integer part, the LSB is the real part:
// For example to apply a factor 1.10:
// 0x01 to the MSB
// 0x1A to the LSB (0.1 x 256 = 25.6 -> 26 = 0x1A)
// Final value to define is: 0x011A

#define MCKEY1_DELTA_COEFF_A (0x0100) /**< MCKey1 Channel A parameter */
#define MCKEY1_DELTA_COEFF_B (0x0100) /**< MCKey1 Channel B parameter */
#define MCKEY1_DELTA_COEFF_C (0x0100) /**< MCKey1 Channel C parameter */
#define MCKEY1_DELTA_COEFF_D (0x0100) /**< MCKey1 Channel D parameter */
#define MCKEY1_DELTA_COEFF_E (0x0100) /**< MCKey1 Channel E parameter */
#define MCKEY1_DELTA_COEFF_F (0x0100) /**< MCKey1 Channel F parameter */
#define MCKEY1_DELTA_COEFF_G (0x0100) /**< MCKey1 Channel G parameter */
#define MCKEY1_DELTA_COEFF_H (0x0100) /**< MCKey1 Channel H parameter */

#define MCKEY2_DELTA_COEFF_A (0x0100) /**< MCKey2 Channel A parameter */
```

```c
#define MCKEY2_DELTA_COEFF_B (0x0100) /**< MCKey2 Channel B parameter */
#define MCKEY2_DELTA_COEFF_C (0x0100) /**< MCKey2 Channel C parameter */
#define MCKEY2_DELTA_COEFF_D (0x0100) /**< MCKey2 Channel D parameter */
#define MCKEY2_DELTA_COEFF_E (0x0100) /**< MCKey2 Channel E parameter */
#define MCKEY2_DELTA_COEFF_F (0x0100) /**< MCKey2 Channel F parameter */
#define MCKEY2_DELTA_COEFF_G (0x0100) /**< MCKey2 Channel G parameter */
#define MCKEY2_DELTA_COEFF_H (0x0100) /**< MCKey2 Channel H parameter */

// Interrupt synchronisation
#define IT_SYNC  (1) /**< Interrupt synchronisation. (=1) Allow to synchronize the
aquisition with a flag set in an interrupt routine */

// Spread spectrum
#define SPREAD_SPECTRUM       (1) /**< Spread spectrum. (=1) Add a variable delay
between acquisitions */
#define SPREAD_COUNTER_MIN    (1) /**< Spread min value */
#define SPREAD_COUNTER_MAX    (20) /**< Spread max value */

// RTOS Management of the acquisition (instead of the timebase interrupt sub-
routine
#define RTOS_MANAGEMENT     (0) /**< The Timebase routine is launched by the
application instead to be managed through a timebase interrupt routine */
// Timer Callback to allow the user to add its own function called from the timer
interrupt sub-routine
#define TIMER_CALLBACK (0)     /**< if (1) Allows the use of a callback function in
the timer interrupt. This function will be called every 0.5ms. The callback
function must be defined inside the application and have the following prototype
FAR void USER_TickTimerCallback(void);   */
/** @} */


//=============================================================================
//
// DEFINITIONS CHECK. DO NOT TOUCH ANYTHING BELOW !!!
//
//=============================================================================

#include "stm8_tsl_checkconfig.h"

#endif /* __TSL_CONF_H */
```

stm8_interrupt_vector.c (shortened)

```c
#include "stm8s_it.h"
#include "STM8_TSL_API.h"

typedef void @far (*interrupt_handler_t)(void);

struct interrupt_vector {
    unsigned char interrupt_instruction;
    interrupt_handler_t interrupt_handler;
};


extern void _stext();     /* startup routine */

struct interrupt_vector const _vectab[] = {
    {0x82, (interrupt_handler_t)_stext}, /* reset */
```

```
        {0x82, NonHandledInterrupt},  /* trap */
        {0x82, NonHandledInterrupt},  /* irq0 */
        ....
        {0x82, (interrupt_handler_t)TSL_Timer_ISR},  /* irq23 */
        ....
        {0x82, NonHandledInterrupt},  /* irq29 */
};
```

main.c

```c
#include "STM8S.h"
#include "stm8_tsl_conf.h"
#include "STM8_TSL_API.h"


void setup_clock(void);
void setup_GPIO(void);
void setup_capacitive_touch(void);


void main(void)
{
    bool state;

    setup_clock();
    setup_GPIO();
    setup_capacitive_touch();

    while(1)
    {
        TSL_Action();

        if ((TSL_GlobalSetting.b.CHANGED) && (TSLState == TSL_IDLE_STATE))
        {
            TSL_GlobalSetting.b.CHANGED = 0;

            if (sSCKeyInfo[0].Setting.b.DETECTED)
            {
                state ^= 1;
            }
        }

        switch(state)
        {
            case 1:
            {
                delay_ms(60);
                break;
            }

            default:
            {
                delay_ms(120);
                break;
            }
        }

        GPIO_WriteReverse(GPIOD, GPIO_PIN_0);
```

```c
        };
}


void setup_clock(void)
{
        CLK_DeInit();

        CLK_HSECmd(ENABLE);
        CLK_LSICmd(DISABLE);
        CLK_HSICmd(ENABLE);
        while(CLK_GetFlagStatus(CLK_FLAG_HSERDY) == FALSE);

        CLK_ClockSwitchCmd(ENABLE);
        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
        CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

        CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSE,
        DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER3, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, ENABLE);
}


void setup_GPIO(void)
{
        GPIO_DeInit(GPIOD);
        GPIO_Init(GPIOD, GPIO_PIN_0, GPIO_MODE_OUT_PP_LOW_SLOW);
}


void setup_capacitive_touch(void)
{
        TSL_Init();

        sSCKeyInfo[0].Setting.b.IMPLEMENTED = 1;
        sSCKeyInfo[0].Setting.b.ENABLED = 1;

        enableInterrupts();
}
```

Explanation

Firstly, include the header and source files for all hardware peripherals as with any other project. Go to touch library installation folder. In my case it is as follows:

C:\Program Files (x86)\STMicroelectronics\STM8S_TouchSensing_Lib_V2.5.0\Libraries\STM8S_TouchSensing_Driver

Copy the header and source files from there to you project include (***inc***) and source (***src***) folders respectively. Note there is a file with an odd name. Some part of its name is in block letter:

stm8_tsl_conf_RC_TOADAPT.h

Rename it to:

stm8_tsl_conf.h

and then edit it. This file states the GPIOs and the timers to be used and their purposes. We just have to edit some part of this file only and we do not need to edit or change any other touch library file. The header file itself is helpful just as other SPL files and is well documented.

In this file, the very first things to edit are the timers. Two timers are needed. The first is for acquisition and the second for a time base. The header file suggests which timers can be used for these purposes and we have to stick to those unless these timers are used for some other tasks. It is better to use a 16-bit timer like TIM2/3 for acquisition and a basic time like TIM4 for time base.

```
//==============================================================================
//
// 2) ACQUISITION TIMER SELECTION (TIMACQ)
//
// Set the acquisition timer and its counter high register address.
//
// The timer you select must be a *16-bit timer*, have a *8-bit prescaler* and
// must be different of the TIMTICK timer described below (TIM1, TIM2 or TIM3
// for example).
//
//==============================================================================

#define TIMACQ           (TIM3)
#define TIMACQ_CNTR_ADD  (0x5328)


//==============================================================================
//
// 3) GENERIC TIMEBASE TIMER SELECTION (TIMTICK)
//
// Set the generic timebase timer.
//
// The timer you select must be a *basic 8-bit timer* and must be different
// of the TIMACQ timer described above (TIM4 for example).
//
// Warning: The selected timer update/overflow interrupt vector must point to
// the TSL_Timer_ISR() interrupt routine.
//
```

```
//=============================================================================

#define TIMTICK  (TIM4)


//=============================================================================
```

Next define the load I/O pin:

```
//=============================================================================
//
// 4) REFERENCE LOAD I/O DEFINITION
//
// Set the port
// Set the pin mask
//
//=============================================================================

#define LOADREF_PORT_ADDR  (GPIOC_BaseAddress)  /**< LOADREF pin GPIO base address
*/

#define LOADREF_BIT        (0x04)               /**< LOADREF pin mask */


//=============================================================================
```

Then, we define the actual touch button(s) and shield I/O pins. Since one capacitive touch key is available on the board, the number of capacitive touch sensor under capacitive touch Port 1 is set one. The correct pins are also masked according to their uses.

```
//=============================================================================
//
// 5) SINGLE CHANNEL KEYS DEFINITION - PORT 1
//
// Set the number of keys
// Set the port
// Set the pins mask
//
// Warning: This port is mandatory and one key at least must be defined.
//
//=============================================================================

#define SCKEY_P1_KEY_COUNT  (1)  /**< Single channel key Port 1: Number of keys
used (value from 1 to 8) */

#define SCKEY_P1_PORT_ADDR  (GPIOC_BaseAddress)  /**< Single channel key Port 1:
GPIO base address */

#define SCKEY_P1_A  (0x02)  /**< Single channel key Port 1: 1st key mask */
#define SCKEY_P1_B  (0)      /**< Single channel key Port 1: 2nd key mask */
....
#define SCKEY_P1_H  (0)      /**< Single channel key Port 1: 8th key mask */

#define SCKEY_P1_DRIVEN_SHIELD_MASK (0x08)


//=============================================================================
```

Finally, we mask all the I/Os used except the load I/O pin:

```
//==============================================================================
//
// 11) ELECTRODES MASKS USED ON EACH GPIO
//
// Define the electrodes mask for each GPIO used (SCKeys + MCKeys but not LOADREF)
//
//==============================================================================

#define GPIOA_ELECTRODES_MASK  (0x00)  /**< Electrodes mask for GPIOA */
....
#define GPIOC_ELECTRODES_MASK  (0x0A)  /**< Electrodes mask for GPIOC */
#define GPIOD_ELECTRODES_MASK  (0x00)  /**< Electrodes mask for GPIOD */
....
#define GPIOI_ELECTRODES_MASK  (0x00)  /**< Electrodes mask for GPIOI */
```

That finishes single capacitive touch configuration! There are other settings but they are rarely changed.

In the interrupt vector mapping file, include the following touch API header file:

```
#include "STM8_TSL_API.h"
```

Also designate the timebase timer (TIMTICK) ISR vector. Here it is TIM4 vector – IRQ23.

```
{0x82, (interrupt_handler_t)TSL_Timer_ISR},  /* irq23 */
```

Capacitive touch sensing requires lot of signal processing apart from hardware and memory resources. Processing data in turn consumes time and so to keep things responsive and real-time, we must run the CPU and peripherals at max speed:

```
CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);
....
CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER3, ENABLE);
CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, ENABLE);
```

Note that except the timers needed for capacitive touch no additional hardware has been clocked.

Touch I/O(s) to be used need special configuration and are not configured like regular GPIOs:

```
void setup_capacitive_touch(void)
{
    TSL_Init();

    sSCKeyInfo[0].Setting.b.IMPLEMENTED = 1;
    sSCKeyInfo[0].Setting.b.ENABLED = 1;

    enableInterrupts();
}
```

The first line of this function initiates the timers and sets the GPIOs used for capacitive touch. The second and third lines state which keys are implemented and enabled. Finally, interrupts are enabled for TIMTICK timer.

So how do we know that the button has been touched? Well the idea is similar to port change interrupt:

```
TSL_Action();

if ((TSL_GlobalSetting.b.CHANGED) && (TSLState == TSL_IDLE_STATE))
{
      TSL_GlobalSetting.b.CHANGED = 0;

      if (sSCKeyInfo[0].Setting.b.DETECTED)
      {
            //Do something....
      }
}
```

The function in the first line should always be called in the main as this function is responsible for touch data processing. The first *if* condition checks if any change in the touch I/O(s) have occurred. If a change has been detected, the key that caused the changed is then identified by the second *if* condition.

The demo here is a variable flash rate LED flasher in which the flash rate is altered by touching the Discovery board's touch button.

Demo

Video link: https://www.youtube.com/watch?v=aRriGKqvdxQ.

Multi Capacitive Touch

It is also possible to design multiple capacitive touch sensors/buttons with the SPL touch library. There is no hardware limitation. Multiple capacitive touch sensors can be used to implement sliders, rotary encoders and knobs, gesture sensors and so on.
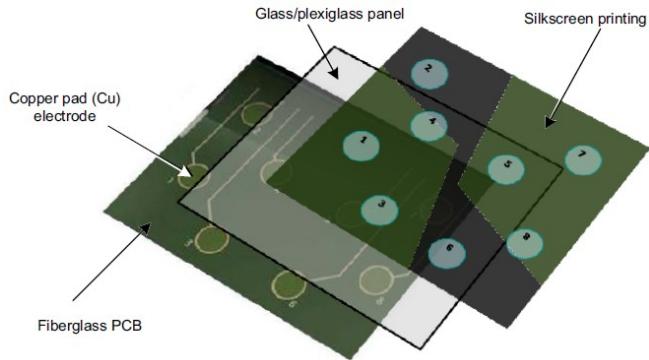
For demoing multi capacitive touch, I used a non-Discovery board with three onboard touch keys, LEDs and other hardware. The hardware schematic of this board is shared below.

Hardware Connection

Code Example

stm8_tsl_conf.h

```c
#ifndef __TSL_CONF_H
#define __TSL_CONF_H



//=============================================================================
//
// 1) MCU SELECTION
//
// 1 = STM8S/A
//
//=============================================================================

#define MCU_SELECTION (1)


//=============================================================================
//
// 2) ACQUISITION TIMER SELECTION (TIMACQ)
//
// Set the acquisition timer and its counter high register address.
//
// The timer you select must be a *16-bit timer*, have a *8-bit prescaler* and
// must be different of the TIMTICK timer described below (TIM1, TIM2 or TIM3
// for example).
//
//=============================================================================

#define TIMACQ           (TIM3)
#define TIMACQ_CNTR_ADD  (0x5328)


//=============================================================================
//
// 3) GENERIC TIMEBASE TIMER SELECTION (TIMTICK)
//
// Set the generic timebase timer.
//
// The timer you select must be a *basic 8-bit timer* and must be different
// of the TIMACQ timer described above (TIM4 for example).
//
// Warning: The selected timer update/overflow interrupt vector must point to
// the TSL_Timer_ISR() interrupt routine.
//
//=============================================================================

#define TIMTICK  (TIM4)


//=============================================================================
//
// 4) REFERENCE LOAD I/O DEFINITION
//
// Set the port
// Set the pin mask
```

```c
//
//=============================================================================

#define LOADREF_PORT_ADDR  (GPIOA_BaseAddress)  /**< LOADREF pin GPIO base address */

#define LOADREF_BIT        (0x40)               /**< LOADREF pin mask */



//=============================================================================
//
// 5) SINGLE CHANNEL KEYS DEFINITION - PORT 1
//
// Set the number of keys
// Set the port
// Set the pins mask
//
// Warning: This port is mandatory and one key at least must be defined.
//
//=============================================================================

#define SCKEY_P1_KEY_COUNT  (3)  /**< Single channel key Port 1: Number of keys used (value from 1 to 8) */

#define SCKEY_P1_PORT_ADDR  (GPIOB_BaseAddress)  /**< Single channel key Port 1: GPIO base address */

#define SCKEY_P1_A  (0x40)  /**< Single channel key Port 1: 1st key mask */
#define SCKEY_P1_B  (0x20)  /**< Single channel key Port 1: 2nd key mask */
#define SCKEY_P1_C  (0x10)  /**< Single channel key Port 1: 3rd key mask */
#define SCKEY_P1_D  (0)     /**< Single channel key Port 1: 4th key mask */
#define SCKEY_P1_E  (0)     /**< Single channel key Port 1: 5th key mask */
#define SCKEY_P1_F  (0)     /**< Single channel key Port 1: 6th key mask */
#define SCKEY_P1_G  (0)     /**< Single channel key Port 1: 7th key mask */
#define SCKEY_P1_H  (0)     /**< Single channel key Port 1: 8th key mask */

#define SCKEY_P1_DRIVEN_SHIELD_MASK (0x80)


//=============================================================================
//
// 6) SINGLE CHANNEL KEYS DEFINITION - PORT 2
//
// Set the number of keys
// Set the port
// Set the pins mask
//
// Note: This port is optional. Set SCKEY_P2_KEY_COUNT to 0 to not use this port.
//
//=============================================================================

#define SCKEY_P2_KEY_COUNT  (0)  /**< Single channel key Port 2: Number of keys used (value from 0 to 8) */

#define SCKEY_P2_PORT_ADDR  (GPIOD_BaseAddress)  /**< Single channel key Port 2: GPIO base address */

#define SCKEY_P2_A  (0)     /**< Single channel key Port 2: 1st key mask */
#define SCKEY_P2_B  (0)     /**< Single channel key Port 2: 2nd key mask */
```

```c
#define SCKEY_P2_C  (0)      /**< Single channel key Port 2: 3rd key mask */
#define SCKEY_P2_D  (0)      /**< Single channel key Port 2: 4th key mask */
#define SCKEY_P2_E  (0)      /**< Single channel key Port 2: 5th key mask */
#define SCKEY_P2_F  (0)      /**< Single channel key Port 2: 6th key mask */
#define SCKEY_P2_G  (0)      /**< Single channel key Port 2: 7th key mask */
#define SCKEY_P2_H  (0)      /**< Single channel key Port 2: 8th key mask */

#define SCKEY_P2_DRIVEN_SHIELD_MASK (0x00)



//==============================================================================
//
// 7) SINGLE CHANNEL KEYS DEFINITION - PORT 3
//
// Set the number of keys
// Set the port
// Set the pins mask
//
// Note: This port is optional. Set SCKEY_P3_KEY_COUNT to 0 to not use this port.
//
//==============================================================================

#define SCKEY_P3_KEY_COUNT  (0)  /**< Single channel key Port 3: Number of keys
used (value from 0 to 8) */

#define SCKEY_P3_PORT_ADDR  (GPIOE_BaseAddress)  /**< Single channel key Port 3:
GPIO base address */

#define SCKEY_P3_A  (0)      /**< Single channel key Port 3: 1st key mask */
#define SCKEY_P3_B  (0)      /**< Single channel key Port 3: 2nd key mask */
#define SCKEY_P3_C  (0)      /**< Single channel key Port 3: 3rd key mask */
#define SCKEY_P3_D  (0)      /**< Single channel key Port 3: 4th key mask */
#define SCKEY_P3_E  (0)      /**< Single channel key Port 3: 5th key mask */
#define SCKEY_P3_F  (0)      /**< Single channel key Port 3: 6th key mask */
#define SCKEY_P3_G  (0)      /**< Single channel key Port 3: 7th key mask */
#define SCKEY_P3_H  (0)      /**< Single channel key Port 3: 8th key mask */

#define SCKEY_P3_DRIVEN_SHIELD_MASK (0x00)


//==============================================================================
//
// 8) NUMBER OF MULTI CHANNEL KEYS AND NUMBER OF CHANNELS USED
//
// Set the total number of multi channel keys used (0, 1 or 2)
// Set the number of channels (5 or 8)
//
//==============================================================================

#define NUMBER_OF_MULTI_CHANNEL_KEYS  (0)  /**< Number of multi channel keys
(value from 0 to 2) */
#define CHANNEL_PER_MCKEY             (5)  /**< Number of channels per key
(possible values are 5 or 8 only) */



//==============================================================================
//
// 9) MULTI CHANNEL KEY 1 DEFINITION
//
```

```c
// Set the port used
// Set the pins mask
//
// Note: This key is optional
//
//=============================================================================

#if NUMBER_OF_MULTI_CHANNEL_KEYS > 0

#define MCKEY1_A_PORT_ADDR  (GPIOD_BaseAddress)  /**< Multi channel key 1: 1st
channel port */
#define MCKEY1_A            (0x40)               /**< Multi channel key 1: 1st
channel mask */
#define MCKEY1_B_PORT_ADDR  (GPIOD_BaseAddress)  /**< Multi channel key 1: 2nd
channel port */
#define MCKEY1_B            (0x20)               /**< Multi channel key 1: 2nd
channel mask */
#define MCKEY1_C_PORT_ADDR  (GPIOD_BaseAddress)  /**< Multi channel key 1: 3rd
channel port */
#define MCKEY1_C            (0x10)               /**< Multi channel key 1: 3rd
channel mask */
#define MCKEY1_D_PORT_ADDR  (GPIOD_BaseAddress)  /**< Multi channel key 1: 4th
channel port */
#define MCKEY1_D            (0x08)               /**< Multi channel key 1: 4th
channel mask */
#define MCKEY1_E_PORT_ADDR  (GPIOD_BaseAddress)  /**< Multi channel key 1: 5th
channel port */
#define MCKEY1_E            (0x04)               /**< Multi channel key 1: 5th
channel mask */
#define MCKEY1_F_PORT_ADDR  (0)                  /**< Multi channel key 1: 6th
channel port */
#define MCKEY1_F            (0)                  /**< Multi channel key 1: 6th
channel mask */
#define MCKEY1_G_PORT_ADDR  (0)                  /**< Multi channel key 1: 7th
channel port */
#define MCKEY1_G            (0)                  /**< Multi channel key 1: 7th
channel mask */
#define MCKEY1_H_PORT_ADDR  (0)                  /**< Multi channel key 1: 8th
channel port */
#define MCKEY1_H            (0)                  /**< Multi channel key 1: 8th
channel mask */

#define MCKEY1_TYPE (1)                          /**< Multi channel key 1 type:
0=wheel (zero between two electrodes), 1=slider (zero in the middle of one
electrode) */
#define MCKEY1_LAYOUT_TYPE (0)                   /**< Multi channel key 1 layout
type: 0=interlaced, 1=normal */

#define MCKEY1_DRIVEN_SHIELD_MASK (0x00)

#endif


//=============================================================================
//
// 10) MULTI CHANNEL KEY 2 DEFINITION
//
// Set the port used
// Set the pins mask
//
```

```c
// Note: This key is optional.
//
//=============================================================================

#if NUMBER_OF_MULTI_CHANNEL_KEYS > 1

#define MCKEY2_A_PORT_ADDR  (GPIOE_BaseAddress)  /**< Multi channel key 2: 1st
channel port */
#define MCKEY2_A            (0x01)               /**< Multi channel key 2: 1st
channel mask */
#define MCKEY2_B_PORT_ADDR  (GPIOE_BaseAddress)  /**< Multi channel key 2: 2nd
channel port */
#define MCKEY2_B            (0x02)               /**< Multi channel key 2: 2nd
channel mask */
#define MCKEY2_C_PORT_ADDR  (GPIOE_BaseAddress)  /**< Multi channel key 2: 3rd
channel port */
#define MCKEY2_C            (0x04)               /**< Multi channel key 2: 3rd
channel mask */
#define MCKEY2_D_PORT_ADDR  (GPIOE_BaseAddress)  /**< Multi channel key 2: 4th
channel port */
#define MCKEY2_D            (0x08)               /**< Multi channel key 2: 4th
channel mask */
#define MCKEY2_E_PORT_ADDR  (GPIOE_BaseAddress)  /**< Multi channel key 2: 5th
channel port */
#define MCKEY2_E            (0x10)               /**< Multi channel key 2: 5th
channel mask */
#define MCKEY2_F_PORT_ADDR  (0)                  /**< Multi channel key 2: 6th
channel port */
#define MCKEY2_F            (0)                  /**< Multi channel key 2: 6th
channel mask */
#define MCKEY2_G_PORT_ADDR  (0)                  /**< Multi channel key 2: 7th
channel port */
#define MCKEY2_G            (0)                  /**< Multi channel key 2: 7th
channel mask */
#define MCKEY2_H_PORT_ADDR  (0)                  /**< Multi channel key 2: 8th
channel port */
#define MCKEY2_H            (0)                  /**< Multi channel key 2: 8th
channel mask */

#define MCKEY2_TYPE (0)                          /**< Multi channel key 2 type:
0=wheel (zero between two electrodes), 1=slider (zero in the middle of one
electrode) */
#define MCKEY2_LAYOUT_TYPE (0)                   /**< Multi channel key 2 layout
type: 0=interlaced, 1=normal */

#define MCKEY2_DRIVEN_SHIELD_MASK (0x00)

#endif



//=============================================================================
//
// 11) ELECTRODES MASKS USED ON EACH GPIO
//
// Define the electrodes mask for each GPIO used (SCKeys + MCKeys but not LOADREF)
//
//=============================================================================
#define GPIOA_ELECTRODES_MASK  (0x00)  /**< Electrodes mask for GPIOA */
#define GPIOB_ELECTRODES_MASK  (0xF0)  /**< Electrodes mask for GPIOB */
```

```c
#define GPIOC_ELECTRODES_MASK   (0x00)   /**< Electrodes mask for GPIOC */
#define GPIOD_ELECTRODES_MASK   (0x00)   /**< Electrodes mask for GPIOD */
#define GPIOE_ELECTRODES_MASK   (0x00)   /**< Electrodes mask for GPIOE */
#define GPIOF_ELECTRODES_MASK   (0x00)   /**< Electrodes mask for GPIOF */
#define GPIOG_ELECTRODES_MASK   (0x00)   /**< Electrodes mask for GPIOG */
#define GPIOH_ELECTRODES_MASK   (0x00)   /**< Electrodes mask for GPIOH */
#define GPIOI_ELECTRODES_MASK   (0x00)   /**< Electrodes mask for GPIOI */


//=============================================================================
//
// 12) TSL PARAMETERS CONFIGURATION
//
//=============================================================================

/** @addtogroup TSL_parameters_RC
  * @{ */

// IO acquisition
#define SCKEY_ACQ_NUM                   (3)   /**< Single channel key acquisition
number - N (value from 1 to 255) */
#define SCKEY_ADJUST_LEVEL              (2)   /**< Single channel key adjustment
level (value from 0 to 255) */
#define MCKEY_ACQ_NUM                   (6)   /**< Multi channel key acquisition
number - N (value from 1 to 255) */
#define MCKEY_ADJUST_LEVEL              (2)   /**< Multi channel key adjustment
level (value from 0 to 255) */

// IO acquisition number of rejected values and measure guardbands
#define MAX_REJECTED_MEASUREMENTS       (20)   /**< Max number of rejected
measurements allowed (value from 0 to 255) */
#define MAX_MEAS_COEFF              (0x011A)   /**< Max measure guardband
(MSB=integer part, LSB=decimal part) */
#define MIN_MEAS_COEFF             (0x00E6)   /**< Min measure guardband
(MSB=integer part, LSB=decimal part) */

// Thresholds
#define SCKEY_DETECTTHRESHOLD_DEFAULT        (30)   /**< Single channel key
detection threshold (value from 1 to 127) */
#define SCKEY_ENDDETECTTHRESHOLD_DEFAULT     (18)   /**< Single channel key end
detection threshold (value from 1 to 127) */
#define SCKEY_RECALIBRATIONTHRESHOLD_DEFAULT  (-18)   /**< Single channel key
calibration threshold (value from -1 to -128) */
#define MCKEY_DETECTTHRESHOLD_DEFAULT        (30)   /**< Multi channel key
detection threshold (value from 1 to 127) */
#define MCKEY_ENDDETECTTHRESHOLD_DEFAULT     (20)   /**< Multi channel key end
detection threshold (value from 1 to 127) */
#define MCKEY_RECALIBRATIONTHRESHOLD_DEFAULT  (-30)   /**< Multi channel key
calibration threshold (value from -1 to -128) */

// MCKey resolution
#define MCKEY_RESOLUTION_DEFAULT                 (4)   /**< Multi channel key
resolution (value from 1 to 8) */

// MCKey Direction Change process
#define MCKEY_DIRECTION_CHANGE_ENABLED              (1)   /**< Multi channel key
direction change enable (1) or disable (0) switch */
#define MCKEY_DIRECTION_CHANGE_MAX_DISPLACEMENT     (255)   /**< Multi channel key
direction change maximum displacement (value from 0 to 255) */
```

```c
#define MCKEY_DIRECTION_CHANGE_INTEGRATOR_DEFAULT    (1)  /**< Multi channel key
direction change integrator (value from 1 to 255) */
#define MCKEY_DIRECTION_CHANGE_THRESHOLD_DEFAULT    (10)  /**< Multi channel key
direction change threshold (value from 1 to 255) */

// Integrators
#define DETECTION_INTEGRATOR_DEFAULT      (2)  /**< Detection Integrator =
Debounce Filter (value from 0 to 255) */
#define END_DETECTION_INTEGRATOR_DEFAULT   (2)  /**< End detection Integrator =
Debounce Filter (from 0 to 255) */
#define RECALIBRATION_INTEGRATOR_DEFAULT  (10)  /**< Calibration integrator (value
from 1 to 255) */

// IIR Filter
#define ECS_TIME_STEP_DEFAULT  (20)  /**< Sampling frequency, multiple of 10ms */
#define ECS_TEMPO_DEFAULT      (20)  /**< Delay after detection, multiple of 100ms
*/
#define ECS_IIR_KFAST_DEFAULT  (20)  /**< K factor for fast filtering */
#define ECS_IIR_KSLOW_DEFAULT  (10)  /**< K factor for slow filtering */

// Detection Timeout
#define DTO_DEFAULT  (0)  /**< 1s unit (value from 0 (= infinite!) to 255) */

// Automatic Calibration
#define NEGDETECT_AUTOCAL (1)  /**< 0 (Enable negative threshold for noise), 1
(Enable autocalibration)  */

// Acquisition values limits
#define SCKEY_MIN_ACQUISITION   (50) /**< Single channel key minimum acquisition
value */
#define SCKEY_MAX_ACQUISITION  (3000) /**< Single channel key maximum acquisition
value */
#define MCKEY_MIN_ACQUISITION   (150) /**< Multi channel key minimum acquisition
value */
#define MCKEY_MAX_ACQUISITION  (5000) /**< Multi channel key maximum acquisition
value */

// Optional parameters for Delta Normalization Process (for Multi channel keys
only).
// The MSB is the integer part, the LSB is the real part:
// For example to apply a factor 1.10:
// 0x01 to the MSB
// 0x1A to the LSB (0.1 x 256 = 25.6 -> 26 = 0x1A)
// Final value to define is: 0x011A

#define MCKEY1_DELTA_COEFF_A (0x0100) /**< MCKey1 Channel A parameter */
#define MCKEY1_DELTA_COEFF_B (0x0100) /**< MCKey1 Channel B parameter */
#define MCKEY1_DELTA_COEFF_C (0x0100) /**< MCKey1 Channel C parameter */
#define MCKEY1_DELTA_COEFF_D (0x0100) /**< MCKey1 Channel D parameter */
#define MCKEY1_DELTA_COEFF_E (0x0100) /**< MCKey1 Channel E parameter */
#define MCKEY1_DELTA_COEFF_F (0x0100) /**< MCKey1 Channel F parameter */
#define MCKEY1_DELTA_COEFF_G (0x0100) /**< MCKey1 Channel G parameter */
#define MCKEY1_DELTA_COEFF_H (0x0100) /**< MCKey1 Channel H parameter */

#define MCKEY2_DELTA_COEFF_A (0x0100) /**< MCKey2 Channel A parameter */
#define MCKEY2_DELTA_COEFF_B (0x0100) /**< MCKey2 Channel B parameter */
#define MCKEY2_DELTA_COEFF_C (0x0100) /**< MCKey2 Channel C parameter */
#define MCKEY2_DELTA_COEFF_D (0x0100) /**< MCKey2 Channel D parameter */
#define MCKEY2_DELTA_COEFF_E (0x0100) /**< MCKey2 Channel E parameter */
```

```c
#define MCKEY2_DELTA_COEFF_F (0x0100) /**< MCKey2 Channel F parameter */
#define MCKEY2_DELTA_COEFF_G (0x0100) /**< MCKey2 Channel G parameter */
#define MCKEY2_DELTA_COEFF_H (0x0100) /**< MCKey2 Channel H parameter */

// Interrupt synchronisation
#define IT_SYNC  (1) /**< Interrupt synchronisation. (=1) Allow to synchronize the
aquisition with a flag set in an interrupt routine */

// Spread spectrum
#define SPREAD_SPECTRUM       (1) /**< Spread spectrum. (=1) Add a variable delay
between acquisitions */
#define SPREAD_COUNTER_MIN   (1) /**< Spread min value */
#define SPREAD_COUNTER_MAX   (20) /**< Spread max value */

// RTOS Management of the acquisition (instead of the timebase interrupt sub-
routine
#define RTOS_MANAGEMENT     (0) /**< The Timebase routine is launched by the
application instead to be managed through a timebase interrupt routine */
// Timer Callback to allow the user to add its own function called from the timer
interrupt sub-routine
#define TIMER_CALLBACK (0)    /**< if (1) Allows the use of a callback function in
the timer interrupt. This function will be called every 0.5ms. The callback
function must be defined inside the application and have the following prototype
FAR void USER_TickTimerCallback(void);  */
/** @} */


//==============================================================================
//
// DEFINITIONS CHECK. DO NOT TOUCH ANYTHING BELOW !!!
//
//==============================================================================

#include "stm8_tsl_checkconfig.h"

#endif /* __TSL_CONF_H */
```

stm8_interrupt_vector.c (shortened)

```c
#include "stm8s_it.h"
#include "STM8_TSL_API.h"

typedef void @far (*interrupt_handler_t)(void);

struct interrupt_vector {
    unsigned char interrupt_instruction;
    interrupt_handler_t interrupt_handler;
};

extern void _stext();    /* startup routine */

struct interrupt_vector const _vectab[] = {
    {0x82, (interrupt_handler_t)_stext}, /* reset */
    {0x82, NonHandledInterrupt}, /* trap  */
    {0x82, NonHandledInterrupt}, /* irq0  */
    ....
    {0x82, (interrupt_handler_t)TSL_Timer_ISR}, /* irq23 */
    ....
```

```
        {0x82, NonHandledInterrupt}, /* irq29 */
};
```

main.c

```c
#include "STM8S.h"
#include "stm8_tsl_conf.h"
#include "STM8_TSL_API.h"


void setup_clock(void);
void setup_GPIO(void);
void setup_capacitive_touch(void);


void main(void)
{
      bool state1 = FALSE;
      bool state2 = FALSE;
      bool state3 = FALSE;

      setup_clock();
      setup_GPIO();
      setup_capacitive_touch();

      while(TRUE)
      {
            TSL_Action();

            if((TSL_GlobalSetting.b.CHANGED == TRUE) && (TSLState ==
TSL_IDLE_STATE))
            {
                  TSL_GlobalSetting.b.CHANGED = FALSE;

                  if(sSCKeyInfo[0].Setting.b.DETECTED)
                  {
                        if(state1 == FALSE)
                        {
                              GPIO_WriteHigh(GPIOD, GPIO_PIN_0);
                              state1 = TRUE;
                        }
                        else
                        {
                              GPIO_WriteLow(GPIOD, GPIO_PIN_0);
                              state1 = FALSE;
                        }
                  }

                  if(sSCKeyInfo[1].Setting.b.DETECTED)
                  {
                        if(state2 == FALSE)
                        {
                              GPIO_WriteHigh(GPIOD, GPIO_PIN_2);
                              state2 = TRUE;
                        }
                        else
                        {
                              GPIO_WriteLow(GPIOD, GPIO_PIN_2);
```

```c
                                state2 = FALSE;
                        }
                }

                if(sSCKeyInfo[2].Setting.b.DETECTED)
                {
                        if(state3 == FALSE)
                        {
                                GPIO_WriteHigh(GPIOD, GPIO_PIN_3);
                                state3 = TRUE;
                        }
                        else
                        {
                                GPIO_WriteLow(GPIOD, GPIO_PIN_3);
                                state3 = FALSE;
                        }
                }
            }
        }
}


void setup_clock(void)
{
    CLK_DeInit();

    CLK_HSECmd(ENABLE);
    CLK_LSICmd(DISABLE);
    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSERDY) == FALSE);

    CLK_ClockSwitchCmd(ENABLE);
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
    CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

    CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSE,
    DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

    CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART3, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER3, ENABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, ENABLE);
}


void setup_GPIO(void)
{
    GPIO_DeInit(GPIOD);
    GPIO_Init(GPIOD, ((GPIO_Pin_TypeDef)(GPIO_PIN_0 | GPIO_PIN_2 |
GPIO_PIN_3)), GPIO_MODE_OUT_PP_LOW_FAST);
}
```

```c
void setup_capacitive_touch(void)
{
    unsigned char i = 0;

    TSL_Init();

    for(i = 0; i < 3; i++)
    {
        sSCKeyInfo[i].Setting.b.IMPLEMENTED = 1;
        sSCKeyInfo[i].Setting.b.ENABLED = 1;
    }

    enableInterrupts();
}
```

Explanation

Every process in this example is same as the last one with some differences in GPIOs.

```c
//============================================================================
//
// 5) SINGLE CHANNEL KEYS DEFINITION - PORT 1
//
// Set the number of keys
// Set the port
// Set the pins mask
//
// Warning: This port is mandatory and one key at least must be defined.
//
//============================================================================

#define SCKEY_P1_KEY_COUNT  (3)  /**< Single channel key Port 1: Number of keys
used (value from 1 to 8) */

#define SCKEY_P1_PORT_ADDR  (GPIOB_BaseAddress)  /**< Single channel key Port 1:
GPIO base address */

#define SCKEY_P1_A  (0x40)  /**< Single channel key Port 1: 1st key mask */
#define SCKEY_P1_B  (0x20)  /**< Single channel key Port 1: 2nd key mask */
#define SCKEY_P1_C  (0x10)  /**< Single channel key Port 1: 3rd key mask */
....
#define SCKEY_P1_H  (0)     /**< Single channel key Port 1: 8th key mask */

#define SCKEY_P1_DRIVEN_SHIELD_MASK (0x80)


//============================================================================
```

Since there are three capacitive touch keys, the key count is three and these keys of capacitive touch Port 1 are designated as previously by proper bit-level masking. Similarly, the shield pin is also defined. Note load pin can reside in some other GPIO port.

This demo code simply alters the on-off states of the three onboard LED when their respective touch buttons are pressed.

Demo

Video link: https://www.youtube.com/watch?v=NG21NVdndXc.
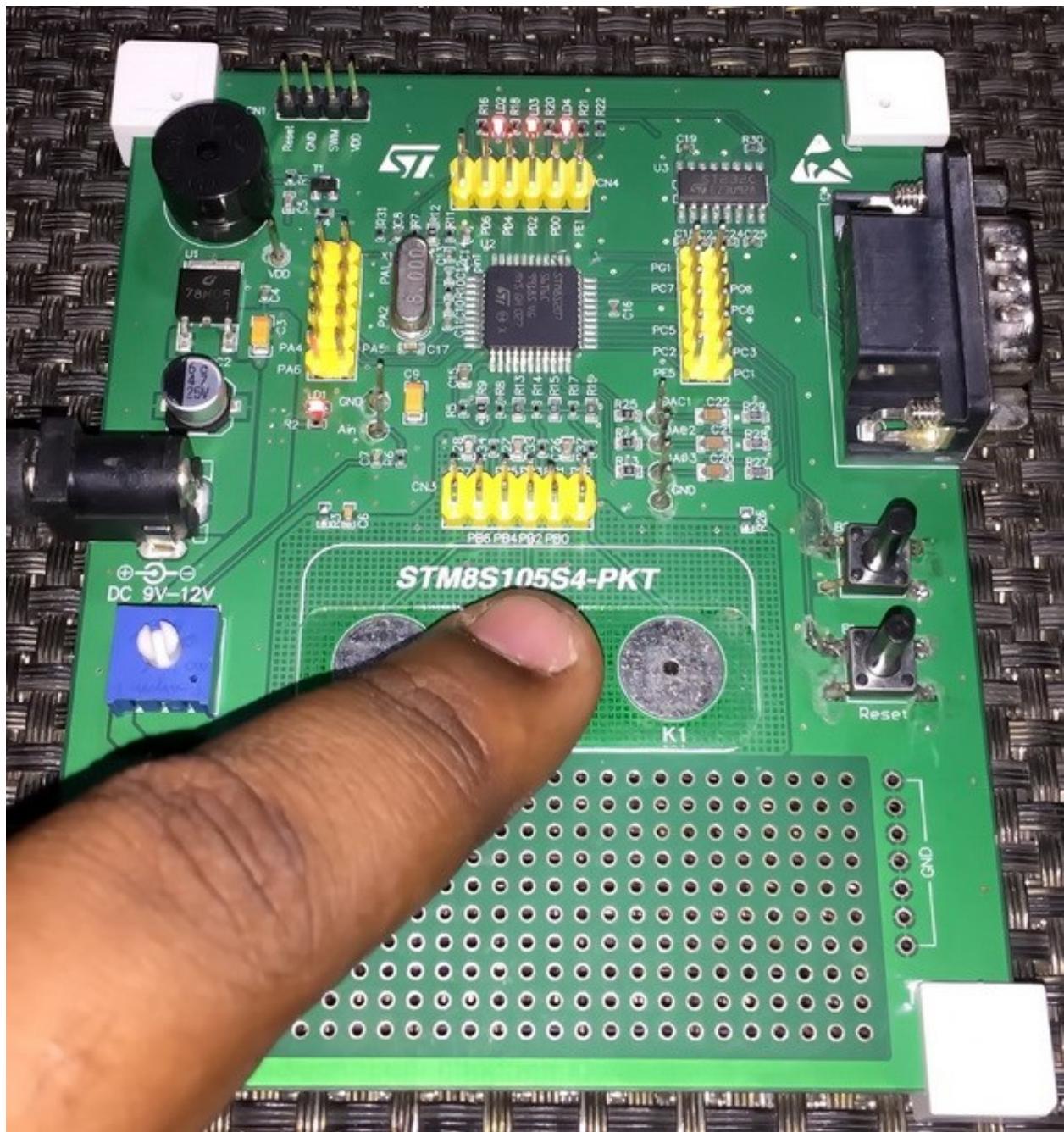
Low Power Sleep Mode

Low power sleep mode is a major requirement in any modern era microcontroller and that is because present-day modern gadgets use batteries as primary energy source. For instance, consider a smart watch like the Mikroelektronika's HexiWear. Unlike previous generation of watches which used tiny LCDs and LED displays, this watch uses an OLED screen to project info. Most of the time we do not keep our eyes fixed at our watch and so it is unnecessary to keep the display powered up. Thus, it is a good idea to put the display and other unimportant functionalities of the watch to sleep. This suspended state in turn save limited battery energy to a great extent. The same trick can also be applied to STM8s.

There are a number of ways to reduce power consumption of a design. These include:

- Using low frequency clocks.
- Slowing down or switching to low speed clocks when fast processing is not needed.
- Stopping or deinitializing unused hardware peripheral modules.
- Using lower power supply voltages without compromising brownout limits.
- Suspending the CPU until certain event(s) take place.
- Avoiding polling-based or blocking codes.
- Optimizing and organizing codes so that tasks are quickly and efficiently done.
- Reducing design sizes and using SMD components.
- Following recommended design notes and layout.

In terms of coding, including low power sleep mode in a code needs well-organized code planning. A developer must probe and formulate plans for both wakeup and sleep events. This is so because if a MCU goes to sleep then it must wakeup upon fulfillment of certain event(s) or else it will appear to have got stuck in an unprecedented loop and will behave unresponsively. Similarly, it is wise to put it to sleep as much as possible in order to reduce power consumption. Thus, events like interrupts are very important in this regard.

Hardware Connection

Code Example

```c
#include "stm8s.h"


bool state = FALSE;


void GPIO_setup(void);
void EXTI_setup(void);
void clock_setup(void);


void main(void)
{
        unsigned char i = 0;
```

```c
        GPIO_setup();
        EXTI_setup();
        clock_setup();

        for(i = 0; i <= 6; i++)
        {
                GPIO_WriteReverse(GPIOD, GPIO_PIN_0);
                delay_ms(900);
        }
        wfi();

        while (TRUE)
        {
                for(i = 0; i < 4; i++)
                {
                        GPIO_WriteReverse(GPIOD, GPIO_PIN_0);
                        delay_ms(600);
                }
                for(i = 0; i < 4; i++)
                {
                        GPIO_WriteReverse(GPIOD, GPIO_PIN_0);
                        delay_ms(300);
                }
                halt();
        }
}


void GPIO_setup(void)
{
        GPIO_DeInit(GPIOB);
        GPIO_Init(GPIOB, GPIO_PIN_7, GPIO_MODE_IN_PU_IT);

        GPIO_DeInit(GPIOD);
        GPIO_Init(GPIOD, GPIO_PIN_0, GPIO_MODE_OUT_PP_LOW_FAST);
}


void EXTI_setup(void)
{
        ITC_DeInit();
        ITC_SetSoftwarePriority(ITC_IRQ_PORTB, ITC_PRIORITYLEVEL_0);

        EXTI_DeInit();
        EXTI_SetExtIntSensitivity(EXTI_PORT_GPIOB, EXTI_SENSITIVITY_FALL_ONLY);
        EXTI_SetTLISensitivity(EXTI_TLISENSITIVITY_FALL_ONLY);

        rim();
}


void clock_setup(void)
{
        CLK_DeInit();

        CLK_HSECmd(DISABLE);
        CLK_LSICmd(DISABLE);
        CLK_HSICmd(ENABLE);
```

```
        while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

        CLK_ClockSwitchCmd(ENABLE);
        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
        CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

        CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
        DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}
```

Explanation

As the hardware connections suggest, Discovery board's built-in LED and button are used. The button is set as an external interrupt source. In the main code shown below, all hardware and clock are set. When the board is powered, the built-in LED flashes slowly three times indicating the beginning of the application code. After the flashing has stopped, there is no other activity. This is because of the function

wfi() – Wait for Interrupt. When this function executes, the CPU is halted until an interrupt occurs. Here, the built-in button comes into play. It introduces the required interrupt. This interrupt causes the CPU to wake up and the CPU continues executing next instructions. Again, we'll see fast variable rate flashes of the built-in LED and then no repetition of the same activity despite being in the main loop. This is because of the *halt()* function. This function also does the same thing as the wait for interrupt function. Thus, halt function also suspends CPU activities, reducing power consumption.

```
void main(void)
{
        unsigned char i = 0;

        GPIO_setup();
        EXTI_setup();
        clock_setup();

        for(i = 0; i <= 6; i++)
        {
                GPIO_WriteReverse(GPIOD, GPIO_PIN_0);
                delay_ms(900);
        }
        wfi();

        while (TRUE)
        {
                for(i = 0; i < 4; i++)
                {
                        GPIO_WriteReverse(GPIOD, GPIO_PIN_0);
```

```
            delay_ms(600);
        }
        for(i = 0; i < 4; i++)
        {
            GPIO_WriteReverse(GPIOD, GPIO_PIN_0);
            delay_ms(300);
        }
        halt();
    }
}
```

Demo
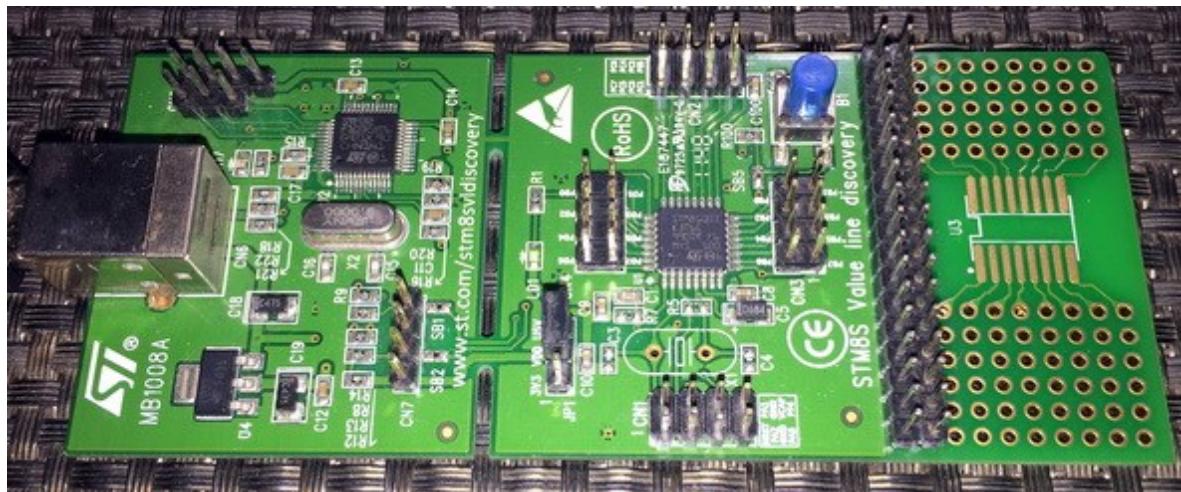
Video link: https://www.youtube.com/watch?v=vTCWKv8is_s.

Auto Wakeup Mode (AWU)

Imagine that you have to design a data acquisition system like a solar-powered weather station that will log data periodically. You can guess that there is a limitation of available energy to keep up the system and continuous monitoring is unnecessary. Thus, weather data is periodically obtained and the system has no other task rather than to sleep. In situations like this we have to rely on STM8's low power mode with auto wakeup feature.
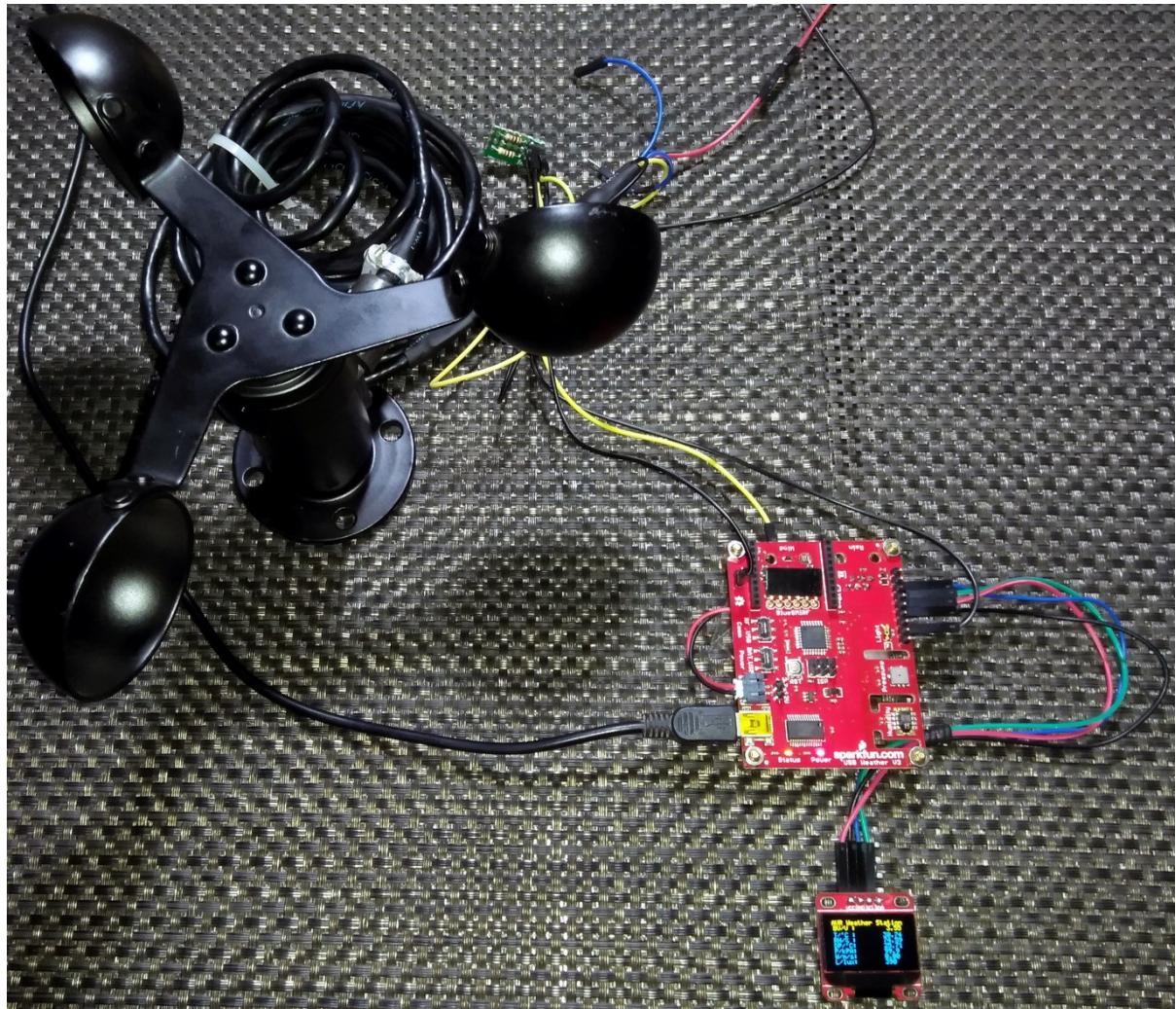
The Auto Wakeup Unit (AWU) of STM8 microcontrollers is like an alarm clock. All we have to do is to set the time for wake up and put our device to sleep. Time ticks and the CPU wakes up to do assigned tasks once the time is over.

Hardware Connection
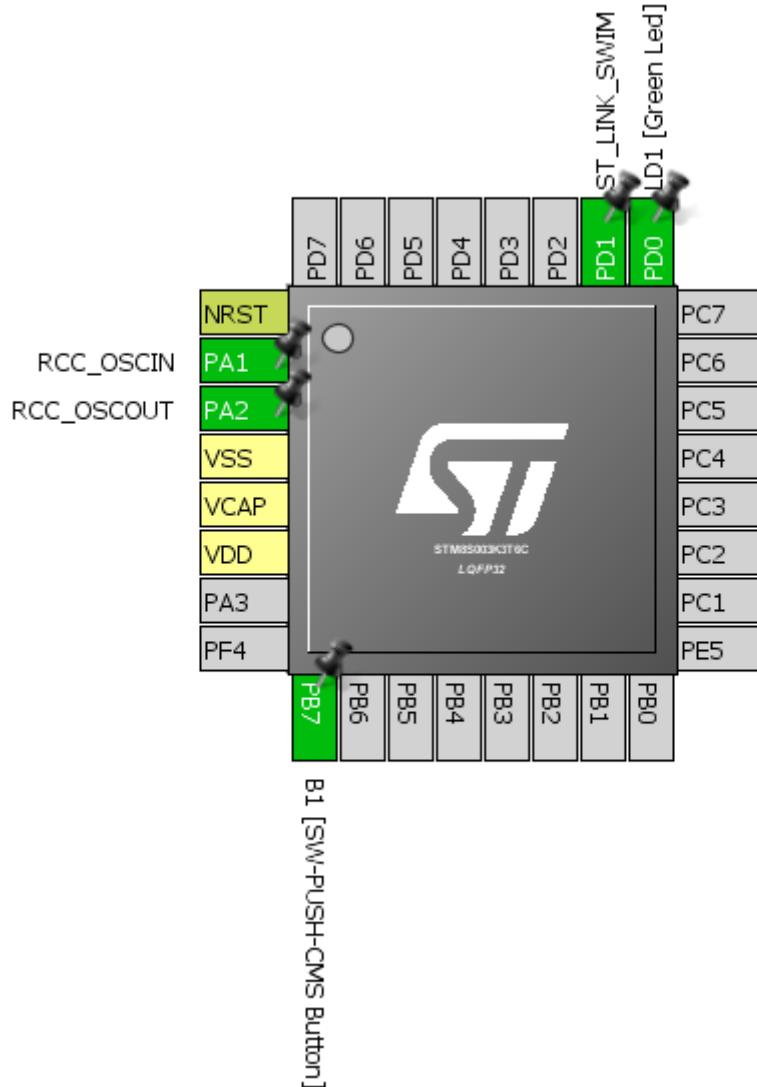
Code Example

```c
#include "STM8S.h"


void clock_setup(void);
void GPIO_setup(void);
void AWU_setup(void);


void main(void)
{
    unsigned char s = 0x00;

    clock_setup();
    GPIO_setup();
    AWU_setup();
```

```c
	while(TRUE)
	{
		for(s = 0x00; s < 0x04; s++)
		{
			GPIO_WriteLow(GPIOD, GPIO_PIN_0);
			delay_ms(60);
			GPIO_WriteHigh(GPIOD, GPIO_PIN_0);
			delay_ms(60);
		}

		halt();
	};
}


void clock_setup(void)
{
	CLK_DeInit();

	CLK_HSECmd(DISABLE);

	CLK_LSICmd(ENABLE);
	while(CLK_GetFlagStatus(CLK_FLAG_LSIRDY) == FALSE);
	CLK_HSICmd(ENABLE);
	while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

	CLK_ClockSwitchCmd(ENABLE);
	CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
	CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

	CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
	DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

	CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, ENABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
	GPIO_DeInit(GPIOD);
	GPIO_Init(GPIOD, GPIO_PIN_0, GPIO_MODE_OUT_OD_HIZ_FAST);
}


void AWU_setup(void)
{
	AWU_IdleModeEnable();
	AWU_DeInit();
	AWU_LSICalibrationConfig(128000);
	AWU_Init(AWU_TIMEBASE_2S);
	AWU_Cmd(ENABLE);
```

```
        enableInterrupts();
}
```

Explanation

To keep things simple, again the on-board LED of Discovery board is used.

The main clock settings are not that important as AWU usually uses LSI as clock source. However, the AWU peripheral clock must be enabled.

```
CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV8);
CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI, DISABLE,
CLK_CURRENTCLOCKSTATE_ENABLE);

CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, ENABLE);
```

AWU setup is simple as shown below:

```
void AWU_setup(void)
{
        AWU_IdleModeEnable();
        AWU_DeInit();
        AWU_LSICalibrationConfig(128000);
        AWU_Init(AWU_TIMEBASE_2S);
        AWU_Cmd(ENABLE);
        enableInterrupts();
}
```

First, we enable idle mode. Then the AWU peripheral is deinitialized. Optionally LSI can be calibrated but in most cases, this is not needed and can be avoided. Initializing the AWU requires only the wake-up time. This time value can be some of the prefixed time values only and it dictates the time for wake up after the CPU has gone sleeping. Lastly, we have to enable the AWU unit.

With these settings, we will see that the on-board LED will blink briefly and then remain turned off for about 2 seconds. Since the AWU is set for 2 seconds, the CPU wakes up after 2 seconds from halted state and the process repeats over and over again.

```
for(s = 0x00; s < 0x04; s++)
{
        GPIO_WriteLow(GPIOD, GPIO_PIN_0);
        delay_ms(60);
        GPIO_WriteHigh(GPIOD, GPIO_PIN_0);
        delay_ms(60);
}

halt();
```

Demo

Video link: https://www.youtube.com/watch?v=xY2q8smQGUc.
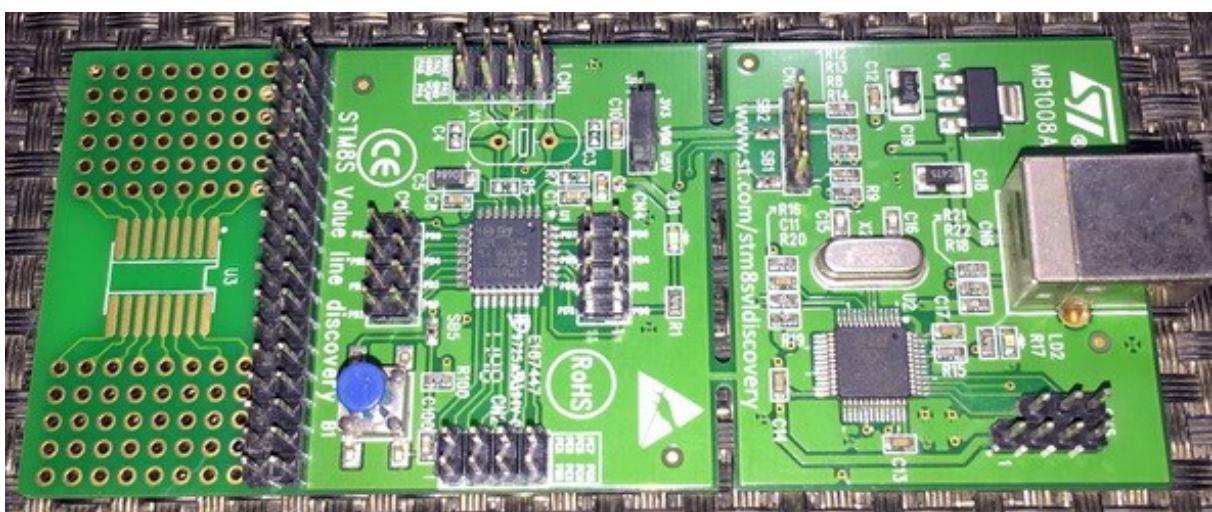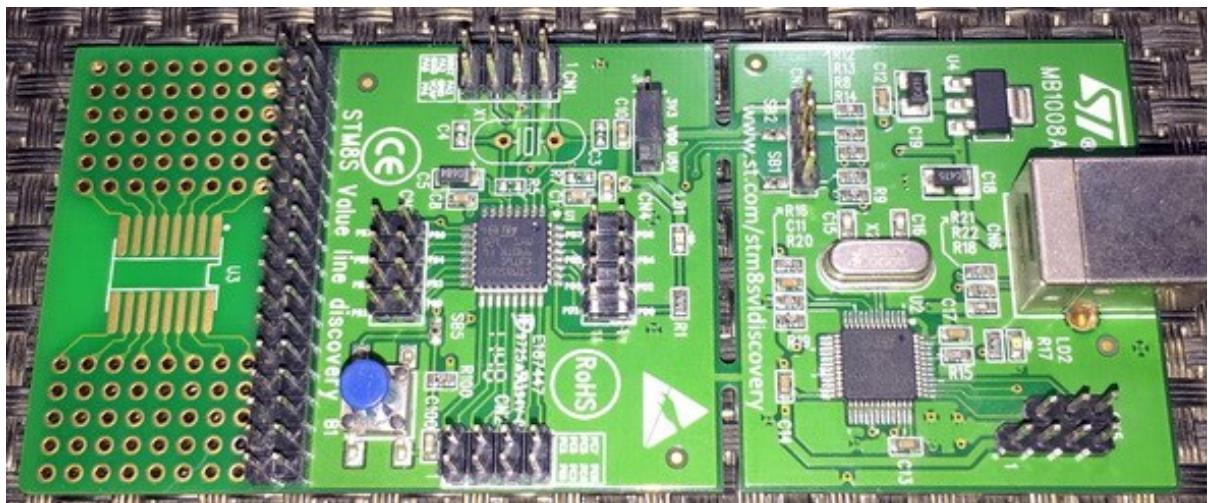
True Data EEPROM

There are several cases where we need to save some very important data in a microcontroller and have them retrieved later. For instance, we would like keep preferences like contrast, colour, etc. of our TV set fixed once set and would like the TV to remember them whenever we turn it on. Usually a micro has three types of memory – RAM, EEPROM and flash memory. EEPROM and flash memory are non-volatile memories, meaning they retain the data stored in them even when powered off. RAM, on the other hand, is a volatile memory. All these memories have different roles. RAM is mainly intended for fast processing and temporary storage. EEPROMs are intended for storing a small set of important data that are often needed to be recalled. Finally, there is flash memory to hold the application code. Thus, it is logical to store settings and calibration data in either EEPROM or flash memory.

STM8s have true data EEPROMs and flash memories. Both share same registers and controls. Shown below is the memory map of STM8S003K3 microcontroller:

You can find this info in your device's datasheet. Check the address ranges. This is important because if you try to write or read wrong locations, your application code will crash as it is an illegal operation.

Hardware Connection
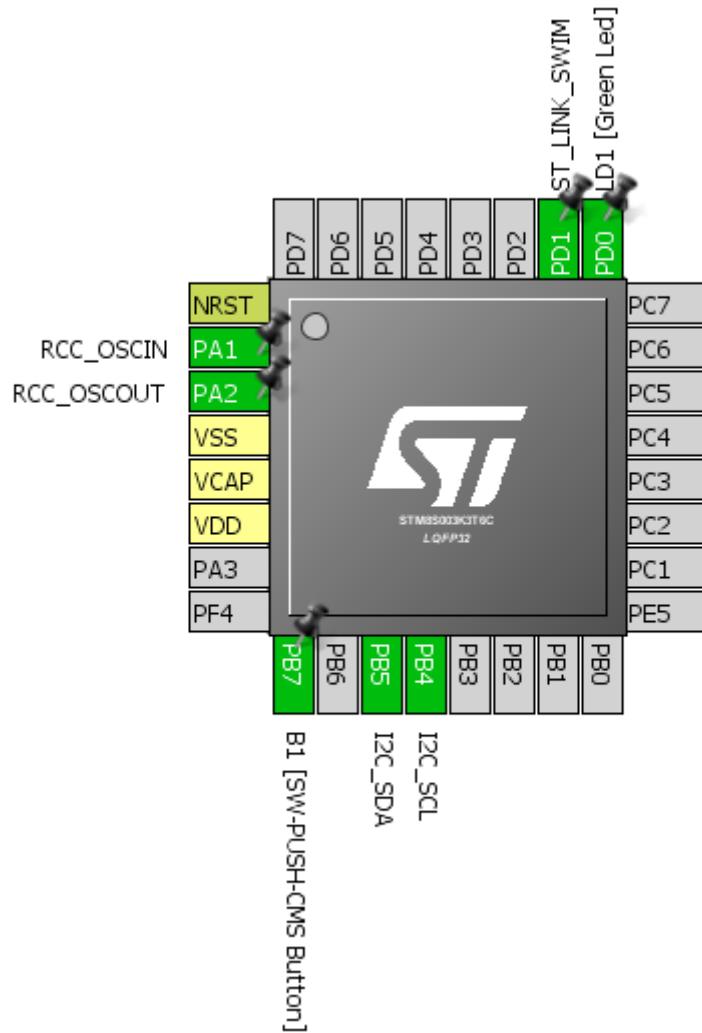
Code Example

```c
#include "STM8S.h"
#include "lcd.h"


#define Button_Port            GPIOB
#define Button_Pin             GPIO_PIN_7


unsigned char bl_state;
unsigned char data_value;


void clock_setup(void);
void GPIO_setup(void);
void Flash_setup(void);
void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned char value);
```

```c
void main()
{
        unsigned char value = 0x00;

        clock_setup();
        GPIO_setup();
        Flash_setup();

        LCD_init();
        LCD_clear_home();

        LCD_goto(0, 0);
        LCD_putstr("STM8 EEPROM Test");

        value = FLASH_ReadByte(0x4000);
        delay_ms(20);

        LCD_goto(0, 1);
        LCD_putstr("WR: ---");
        LCD_goto(9, 1);
        LCD_putstr("RD:");
        lcd_print(13, 1, value);
        delay_ms(2000);

        while(TRUE)
        {
                if(GPIO_ReadInputPin(Button_Port, Button_Pin) == FALSE)
                {
                        while(GPIO_ReadInputPin(Button_Port, Button_Pin) == FALSE);
                        FLASH_Unlock(FLASH_MEMTYPE_DATA);
                        FLASH_EraseByte(0x4000);
                        delay_ms(20);
                        FLASH_ProgramByte(0x4000, value);
                        delay_ms(20);
                        FLASH_Lock(FLASH_MEMTYPE_DATA);

                        lcd_print(13, 1, value);
                }

                delay_ms(20);
                lcd_print(4, 1, value);

                value++;
                delay_ms(200);
        };
}


void clock_setup(void)
{
        CLK_DeInit();

        CLK_HSECmd(DISABLE);
        CLK_LSICmd(DISABLE);
        CLK_HSICmd(ENABLE);
        while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

        CLK_ClockSwitchCmd(ENABLE);
```

```c
        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV4);

        CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

        CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI, DISABLE,
                              CLK_CURRENTCLOCKSTATE_ENABLE);

        CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, ENABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
        CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
        GPIO_DeInit(GPIOB);
        GPIO_Init(Button_Port, Button_Pin, GPIO_MODE_IN_FL_NO_IT);
}


void Flash_setup(void)
{
        FLASH_DeInit();
}


void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned char value)
{
        char chr = 0x00;

        chr = ((value / 100) + 0x30);
        LCD_goto(x_pos, y_pos);
        LCD_putchar(chr);

        chr = (((value / 10) % 10) + 0x30);
        LCD_goto((x_pos + 1), y_pos);
        LCD_putchar(chr);

        chr = ((value % 10) + 0x30);
        LCD_goto((x_pos + 2), y_pos);
        LCD_putchar(chr);
}
```

Explanation

In this demo, two-wire I2C LCD is used for displaying data. The code works by reading the last data stored in the EEPROM and incrementing a variable named *value*. Only one location (0x4000) is read and updated when the Discovery board's button is pressed.

Using the internal EEPROM is very easy. We need to deinitialize it first:

```c
void Flash_setup(void)
{
     FLASH_DeInit();
}
```

Reading a location is as follows:

```c
value = FLASH_ReadByte(0x4000);
```

We just have to set the location we wish to read.

To erase and to write, we need to unlock the memory type we wish to write using *Memory Access Security System (MASS)* key, write/erase the desired location and the lock it back. This lock feature ensures safety from data corruption and accidental writes.

```c
FLASH_Unlock(FLASH_MEMTYPE_DATA);
FLASH_EraseByte(0x4000);
delay_ms(20);
FLASH_ProgramByte(0x4000, value);
delay_ms(20);
FLASH_Lock(FLASH_MEMTYPE_DATA);
```

Demo

Video link: https://www.youtube.com/watch?v=9gsMS9MkyJs.

Internal Flash Memory

Though flash memories are primarily intended for storing application codes, it is still possible to use them just like EEPROMs using ***In-Application Programming (IAP)***. However, it is important to check that by mistake, we don't write in those areas where application code reside. IAP can also be used for upgrading application firmware ***Over-The-Air (OTA)***.

Hardware Connection
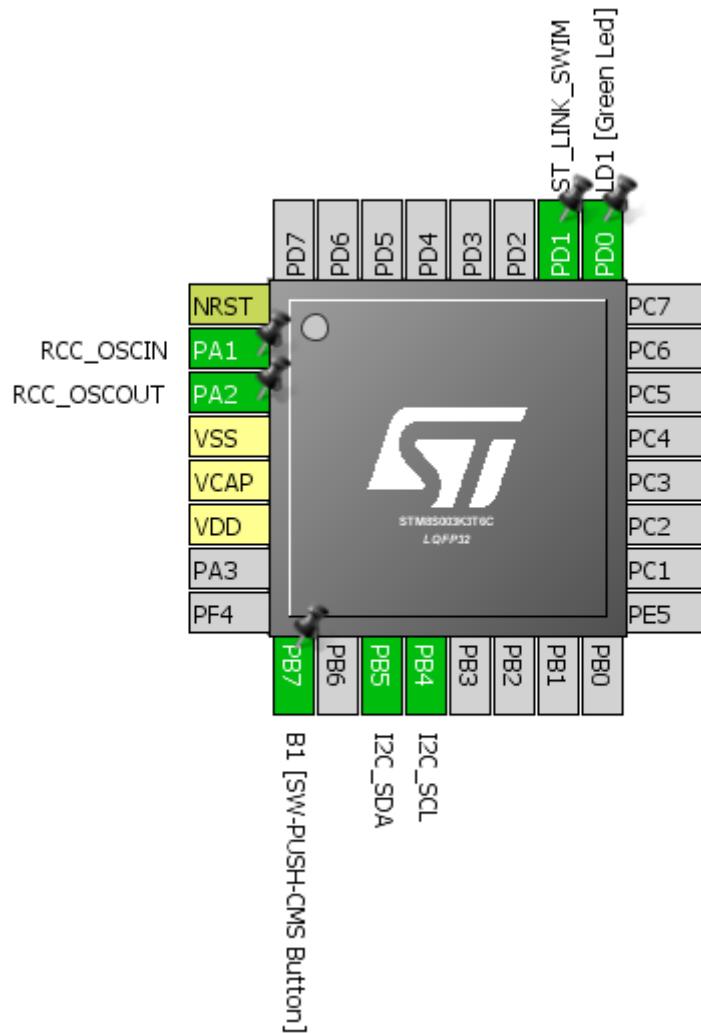
Code Example

```
#include "STM8S.h"
#include "lcd.h"


#define Button_Port            GPIOB
#define Button_Pin             GPIO_PIN_7
```

```c
unsigned char bl_state;
unsigned char data_value;


void clock_setup(void);
void GPIO_setup(void);
void Flash_setup(void);
void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned char value);


void main()
{
    unsigned char value = 0x00;

    clock_setup();
    GPIO_setup();
    Flash_setup();

    LCD_init();
    LCD_clear_home();

    LCD_goto(0, 0);
    LCD_putstr("STM8S Flash Test");

    value = FLASH_ReadByte(0x9FF0);
    delay_ms(20);

    LCD_goto(0, 1);
    LCD_putstr("WR: ---");
    LCD_goto(9, 1);
    LCD_putstr("RD:");
    lcd_print(13, 1, value);
    delay_ms(2000);

    while(TRUE)
    {
        if(GPIO_ReadInputPin(Button_Port, Button_Pin) == FALSE)
        {
            while(GPIO_ReadInputPin(Button_Port, Button_Pin) == FALSE);
            FLASH_Unlock(FLASH_MEMTYPE_PROG);
            FLASH_EraseByte(0x9FF0);
            delay_ms(20);
            FLASH_ProgramByte(0x9FF0, value);
            delay_ms(20);
            FLASH_Lock(FLASH_MEMTYPE_PROG);

            lcd_print(13, 1, value);
        }

        delay_ms(20);
        lcd_print(4, 1, value);

        value++;
        delay_ms(200);
    };
}
```

```c
void clock_setup(void)
{
	CLK_DeInit();

	CLK_HSECmd(DISABLE);
	CLK_LSICmd(DISABLE);
	CLK_HSICmd(ENABLE);
	while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

	CLK_ClockSwitchCmd(ENABLE);
	CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV4);

	CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

	CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI, DISABLE,
	                      CLK_CURRENTCLOCKSTATE_ENABLE);

	CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, ENABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
	CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}


void GPIO_setup(void)
{
	GPIO_DeInit(GPIOB);
	GPIO_Init(Button_Port, Button_Pin, GPIO_MODE_IN_FL_NO_IT);
}


void Flash_setup(void)
{
	FLASH_DeInit();
}


void lcd_print(unsigned char x_pos, unsigned char y_pos, unsigned char value)
{
	char chr = 0x00;

	chr = ((value / 100) + 0x30);
	LCD_goto(x_pos, y_pos);
	LCD_putchar(chr);

	chr = (((value / 10) % 10) + 0x30);
	LCD_goto((x_pos + 1), y_pos);
	LCD_putchar(chr);

	chr = ((value % 10) + 0x30);
	LCD_goto((x_pos + 2), y_pos);
	LCD_putchar(chr);
}
```

Explanation

This example is same as the EEPROM example. The differences are the read, write and erase instructions.

```
value = FLASH_ReadByte(0x9FF0);

….

FLASH_Unlock(FLASH_MEMTYPE_PROG);
FLASH_EraseByte(0x9FF0);
delay_ms(20);
FLASH_ProgramByte(0x9FF0, value);
delay_ms(20);
FLASH_Lock(FLASH_MEMTYPE_PROG);
```

The same MASS keying technique is needed and everything is same because both memories are controlled using the same set of registers. A few things must be noted while using data EEPROM and program flash:

- The memory type needs to be specified - FLASH_MEMTYPE_PROG when using program/flash memory and FLASH_MEMTYPE_DATA when using the internal true data EEPROM.

- Apply *Read-Out-Protection (ROP)* with *ROP* configuration bit if you want to protect your code from unfriendly eyes.
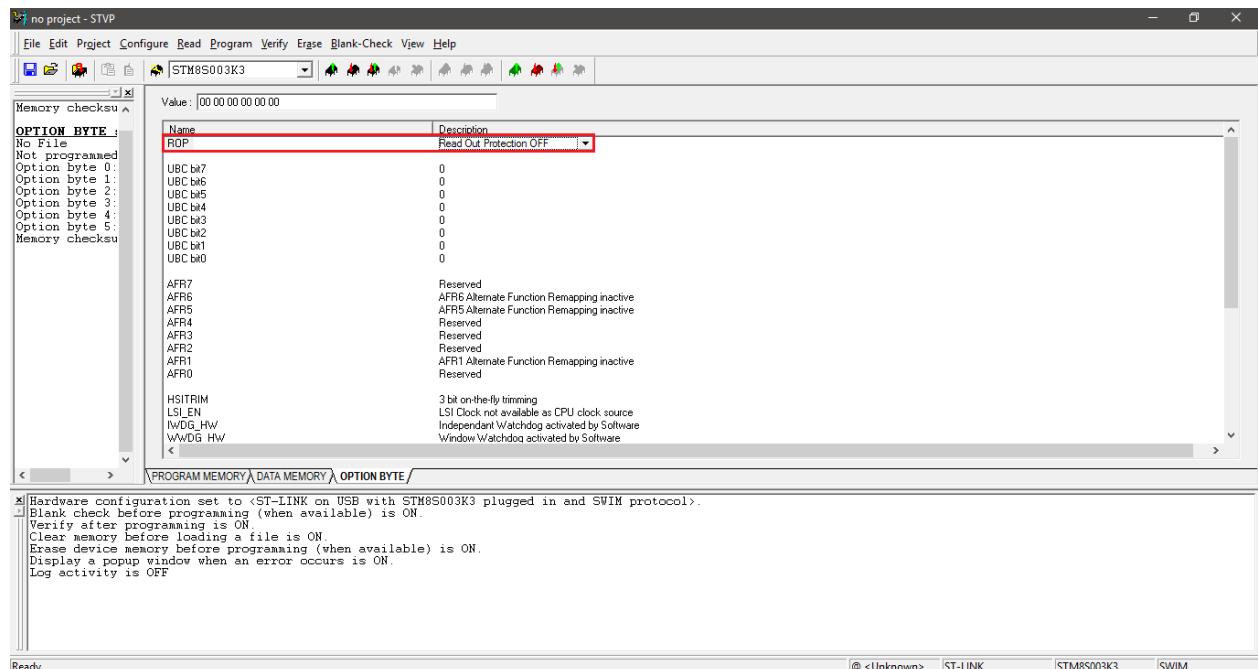

- Avoid frequent writes/erases to increase memory life cycles. If needed to write/erase data frequently use wear-leveling technique. Read operations do not wear memories.

- For clock speeds up to 16 MHz, it is not needed to wait after a write/erase operation. I still insist waiting for a short while before executing next instructions.

- The read, write and erase location must not be outside specified boundary limits or else the application will surely crash or perform improperly. Check device datasheet for memory locations. A much easier and faster way is to use ST Visual Programmer.
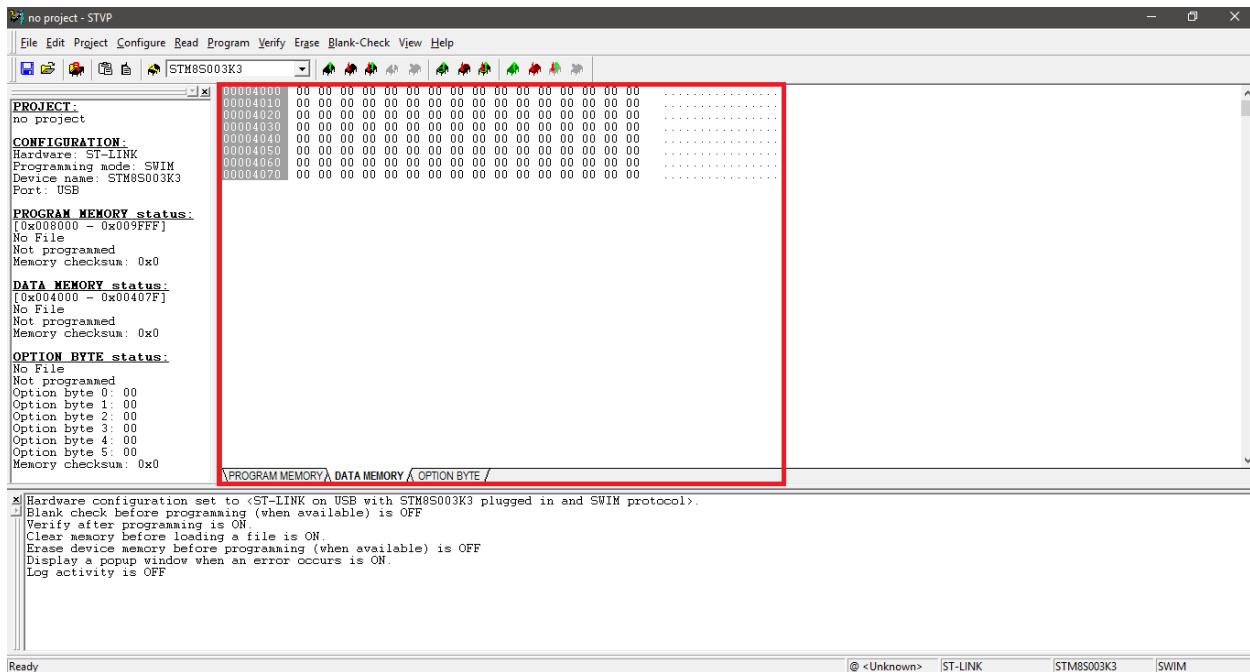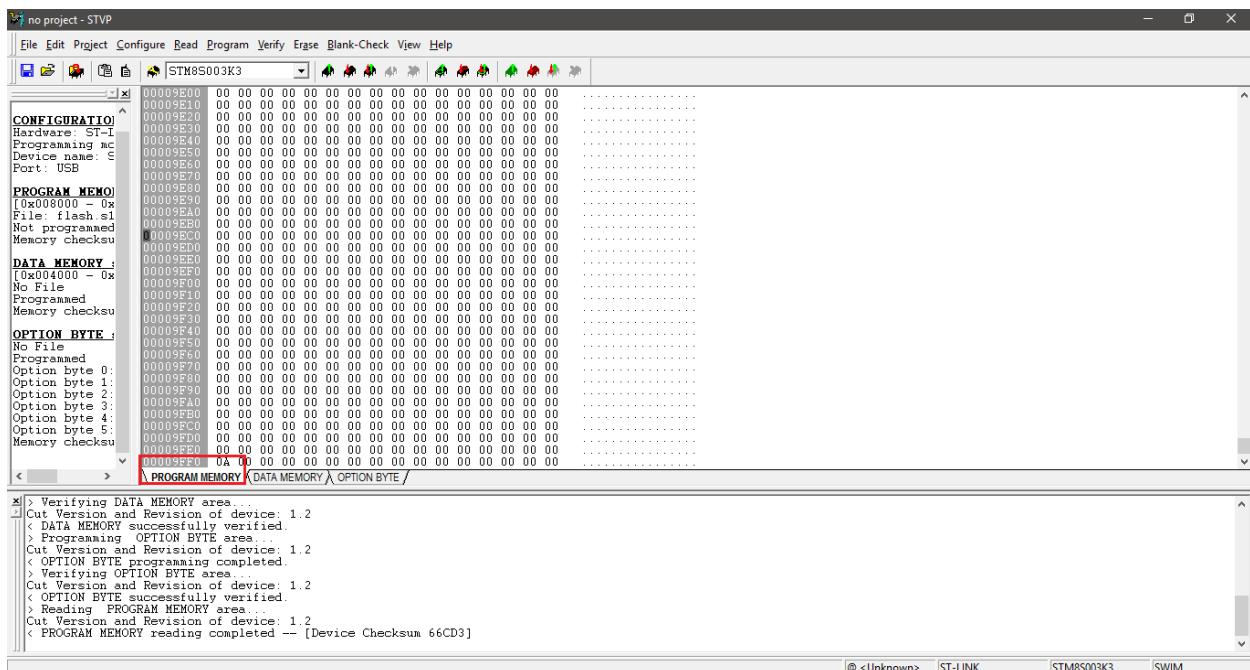
- Power source must not be fluctuating.

- By definition a block is a set of bytes and a page is a set of blocks. Most modern memories do not allow byte-level write/erase. However, both memory types of STM8s allow byte-level read, write and erase. By the way, it is wise to use page and block writes/erases to enhance memory life cycle and for faster operations.

- It is also possible to set configuration bytes using IAP.

Demo

Video link: https://youtu.be/N-AUuSuAQyA.
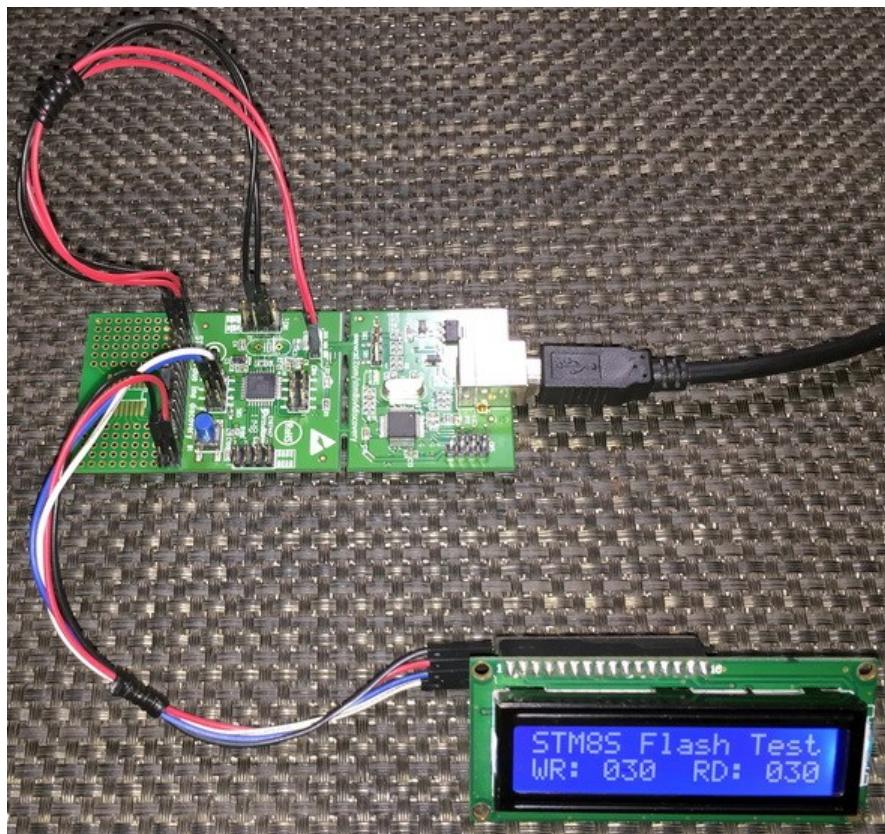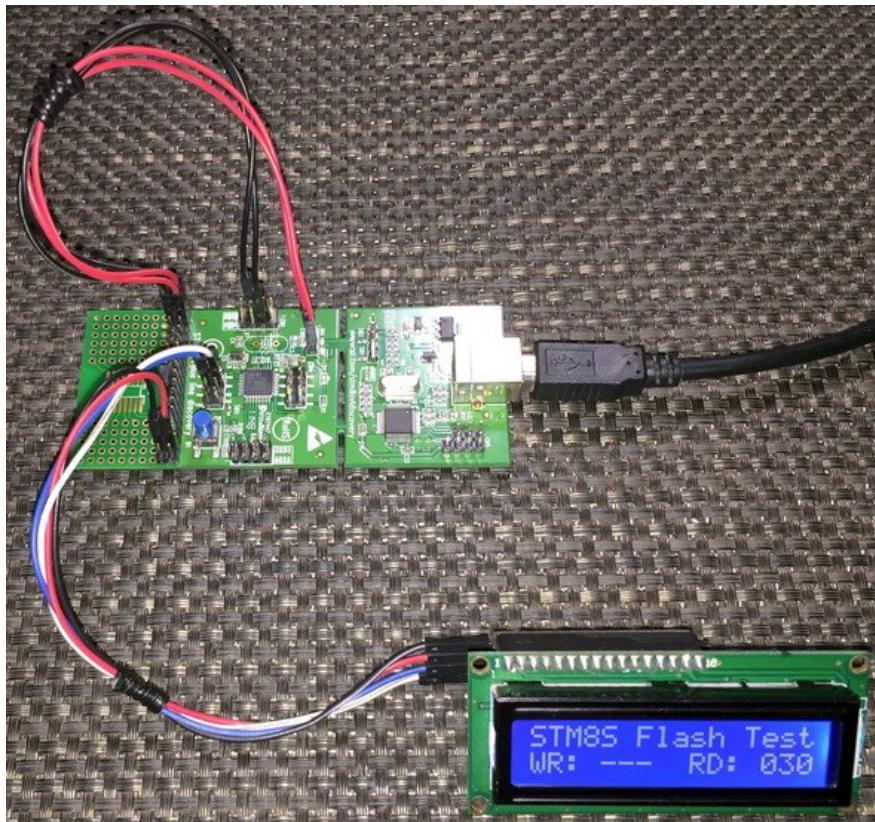
Some Useful Tips

When using a new compiler, I evaluate some certain things. For instance, how do I include my own written library files, interrupt management, what conventions I must follow and what dos and don'ts must be observed.

Creation & Addition of libraries

At some point in working with any microcontroller, you'll need two basic libraries more than anything else. These are LCD and delay libraries. LCDs are great tools for quickly projecting or presenting data apart from debugging a code with a debugger. Similarly, time-wasting delay loops help us slow down things at our liking. Humans are not as fast as machines. Delays can be avoided in many novel ways but delays keep things simple and so are necessities in some areas.

The Standard Peripheral Library only provides libraries for hardware peripherals and surely not for anything else. It is also practically impossible to provide library for all hardware on available on the planet. Thus, whenever when we will be needing new hardware integrations with STM8s, we will have to code and tag our libraries with our projects. So how can we do so?

Earlier in this article I discussed about Alphanumeric LCDs and delays. If you check the datasheet of such LCDs, you'll find initialization sequences in some while in others you may also find ready-made codes. These sequences are needed to be translated in code just like what we do with I2C or SPI-based devices. Shown below is such an example:
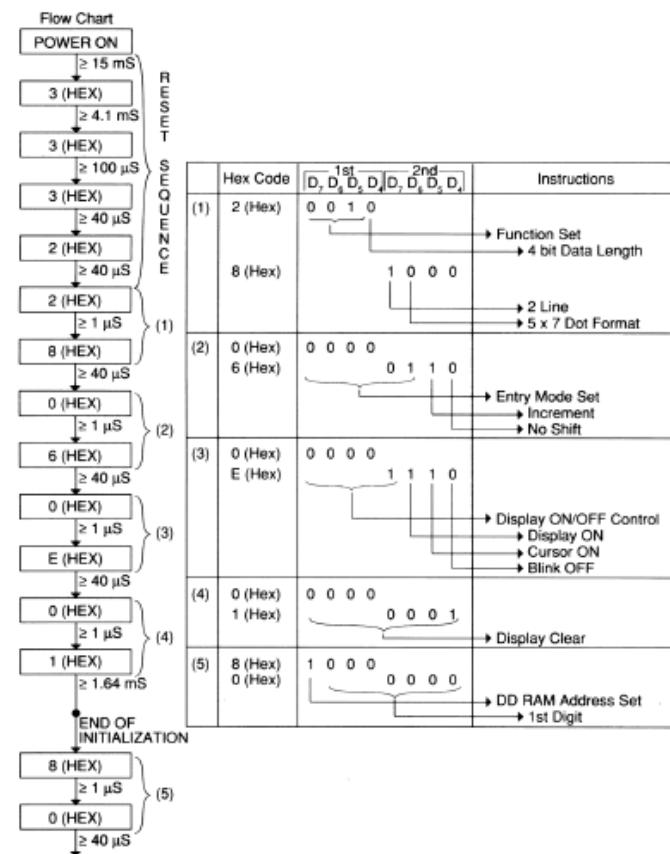
Creating new libraries is simple. Just need to follow the following steps:

- There should be a header file and a source file for every new module. For example, **lcd.h** and **lcd.c**.

- Every header file should start with the inclusion of **stm8s.h** header file (*#include "stm8s.h"*). This header is needed because it allows the access to the internal hardware modules available in a STM8 micro. For example, we will need access to GPIOs to develop our LCD library.

- A good practice is that the header files only contain function prototypes, definitions, constants, enumerations and global variables.

- The corresponding source file must only include its header file in beginning.

- The source file should contain the body of codes for all functions declared in the header file.

- When one library is dependent on the functions of another's, the one that will be required in the new library must be included first. For example, we will need delay library in coding the LCD library because there are **delay_ms** functions in some parts of the LCD library and so delay library should be included first. This should be the systematic order:

```
#include "stm8s_delay.h"
#include "lcd.h"
```

You can include these files at the bottom part of the **stm8s_conf.h** header file complying with right precedence as shown below:

```
84      #include "stm8s_uart3.h"
85    #endif /* STM8S208 || STM8S207 || STM8AF52Ax || STM8AF62Ax */
86    #if defined(STM8AF622x)
87      #include "stm8s_uart4.h"
88    #endif /* (STM8AF622x) */
89    #include "stm8s_wwdg.h"
90
91
92    /* YOUR HEADER FILES */   <=========
93
94
95    /* Exported types ----------------------------------------------------------*/
96    /* Exported constants ------------------------------------------------------*/
97    /* Uncomment the line below to expanse the "assert_param" macro in the
98       Standard Peripheral Library drivers code */
99    //#define USE_FULL_ASSERT    (1)
100
101    /* Exported macro ----------------------------------------------------------*/
102    #ifdef  USE_FULL_ASSERT
103
104    /**
105     * @brief  The assert_param macro is used for function's parameters check.
106     * @param expr: If expr is false, it calls assert_failed function
107     *    which reports the name of the source file and the source
108     *    line number of the call that failed.
109     *    If expr is true, it returns no value.
110     * @retval : None
111     */
```

Alternatively, you can add them after the first line **#include "stm8s.h"** in your main source code.

Peripheral Clock Configurations

In most codes revealed so far, I made clock configurations every time. The reasons behind so are

- Selection of right clock source.

- Adjustment of peripheral and system clocks as per requirement. Again, it is mainly intended to balance off both power consumption and overall performance.

- Disabling any unused hardware. This reduces power consumption and help us avoid certain hardware conflicts.

```c
void clock_setup(void)
{
    CLK_DeInit();

    CLK_HSECmd(DISABLE);
    CLK_LSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_LSIRDY) == FALSE);
    CLK_HSICmd(ENABLE);
    while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);

    CLK_ClockSwitchCmd(ENABLE);
    CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
    CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

    CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
    DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);

    CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
    CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
}
```

The following lines select clock sources:

```c
CLK_DeInit();

CLK_HSECmd(DISABLE);
CLK_LSICmd(ENABLE);
while(CLK_GetFlagStatus(CLK_FLAG_LSIRDY) == FALSE);
CLK_HSICmd(ENABLE);
while(CLK_GetFlagStatus(CLK_FLAG_HSIRDY) == FALSE);
```

What these lines do are enabling/disabling clock sources and wait for the sources to stabilize.

Then the following lines select clock prescalers and switching:

```
CLK_ClockSwitchCmd(ENABLE);
CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);

CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSI,
DISABLE, CLK_CURRENTCLOCKSTATE_ENABLE);
```

Finally, the last segment enables/disables peripheral clocks:

```
CLK_PeripheralClockConfig(CLK_PERIPHERAL_SPI, DISABLE);
CLK_PeripheralClockConfig(CLK_PERIPHERAL_I2C, DISABLE);
CLK_PeripheralClockConfig(CLK_PERIPHERAL_ADC, DISABLE);
CLK_PeripheralClockConfig(CLK_PERIPHERAL_AWU, DISABLE);
CLK_PeripheralClockConfig(CLK_PERIPHERAL_UART1, DISABLE);
CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER1, DISABLE);
CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER2, DISABLE);
CLK_PeripheralClockConfig(CLK_PERIPHERAL_TIMER4, DISABLE);
```

This segment is very important and should always be rechecked. Different chips have different internal hardware peripheral and so this segment will be different. For instance, STM8S105 has no UART1 module but it has UART2 instead. Research which hardware are available in your target micro and then code this segment. I ended up with several wasted hours of finding trouble in various cases only to find that I didn't enable the required hardware. Check the device datasheet or STM8CubeMX as discussed previously.

Configuring Similar Stuffs Quickly

Sometimes you may end up doing the same stuff over and over again while you could have done it simply with one or two lines of code. For example, in the LCD library, the GPIOs that connect with the LCD share the same configurations. All are fast push-pull outputs.

```
GPIO_Init(LCD_PORT, LCD_RS, GPIO_MODE_OUT_PP_HIGH_FAST);
GPIO_Init(LCD_PORT, LCD_EN, GPIO_MODE_OUT_PP_HIGH_FAST);
GPIO_Init(LCD_PORT, LCD_DB4, GPIO_MODE_OUT_PP_HIGH_FAST);
GPIO_Init(LCD_PORT, LCD_DB5, GPIO_MODE_OUT_PP_HIGH_FAST);
GPIO_Init(LCD_PORT, LCD_DB6, GPIO_MODE_OUT_PP_HIGH_FAST);
GPIO_Init(LCD_PORT, LCD_DB7, GPIO_MODE_OUT_PP_HIGH_FAST);
```

This can be done in a more simplistic manner with just one line of code:

```
GPIO_Init(LCD_PORT, ((GPIO_Pin_TypeDef)(LCD_RS | LCD_EN | LCD_DB4 | LCD_DB5 |
LCD_DB6 | LCD_DB7)), GPIO_MODE_OUT_PP_HIGH_FAST);
```

As you can see it is just a bunch of logical *OR* operation. The same method is applicable for other peripherals that share the same initialization function.

Some Stuffs About Cosmic C and SPL

- Functions, variables and definitions in Cosmic C are case sensitive and so be careful about this.

- Try to avoid polling methods. Try to use interrupt-based ones but make sure that there is no interrupt-within-interrupt case or otherwise your code may behave erratically. Best is to attach interrupts for important tasks like timing-related jobs, ADC conversions and communications. It is up to your design requirements and choices.

- **CTRL + Space** or code assist is a great helper. Use it to the fullest.

- Functions with no arguments must not have empty argument areas. For example, you cannot write:

```
void setup ();
```

You should write it as:

```
void setup (void);
```

- Definitions and constants can be declared as with any C compiler.

- Wherever there are flags, you need to be careful. You should check and clear flags even if it is cleared by hardware. For instance, when reading ADC, the ADC **End-Of-Conversion (EOC)** flag is automatically cleared but still in the code you should check and clear it.

```
ADC1_ClearFlag(ADC1_FLAG_EOC);
```

Flags are so important that unless you check and clear them appropriately, you may not get the right result from your code. Personally, I didn't care much until I got myself into trouble.

- You can mix assembly codes with your C code to enhance performance and optimization. However, you need to have sound knowledge of the assembly instructions. This is a rare requirement. The delay library, for instance, uses no operation assembly instruction to achieve delays. This is written as shown:

```
_asm ("nop");
```

- Empty loops and blank conditions are ignored by the compiler as a part of code optimization.

- Long ago, Atmel (now Microchip) published a document regarding ways to efficiently optimize C coding. This document holds true for most microcontrollers. For example, in that document it is stated that a decrementing **do-while** loop is much more faster and code efficient than an incrementing **do-while** loop. You can apply the methods presented there and other similar ideas with STM8 microcontrollers too. The document can be downloaded from here:

  http://www.atmel.com/images/doc8453.pdf.

  There are other similar documents from other manufacturers that are similarly helpful.

Microchip's *Tips & Tricks* documents are some that are can be mentioned in this list. These documents not only discuss software tricks and tips but also hardware ones. I would also like to suggest readers to go through app notes and other related literature.

If you are designing your own PCB that will house a STM8 microcontroller, you must follow recommended PCB layouts and design considerations. There are tons of docs on this field that will surely help you to design the best PCBs.

- Don't mess with configuration (fuse) bits unless needed. Most of the times you will never have to deal with them. *AFRs* are used when remapping is needed or to enable special GPIO functionality. These will be of most importance.

- Though I don't feel it as a necessity and don't recommend it, you can avoid using the SPL and still code by raw register level access. For example, you can blink a LED with the following code:

```c
#include "stm8s.h"


void main (void)
{
    GPIOD->DDR |= 0x01;
    GPIOD->CR1 |= 0x01;

    for(;;)
    {
        GPIOD->ODR ^= (1 << 0);
        delay_ms(100);
    };
}
```

As you can see it is both tedious and meaningless unless you comment every single line. However, there will times when such raw coding will become necessary as they offer speed and less code size compared to SPL.

- Bitwise and logic operations are useful. Not only they are fast, they just deal with the designated bits only. SPL (*stm8s.h* header file) has support for such operations but it is still better to know them. Here are some common operations:

```c
//For setting a bit of a register
#define bit_set(reg, bit_val)                    reg |= (1 << bit_val)

//For clearing a bit of a register
#define bit_clr(reg, bit_val)                    reg &= (~(1 << bit_val))

//For toggling a bit of a register
#define bit_tgl(reg, bit_val)                    reg ^= (1 << bit_val)

//For extracting the bit state of a register
#define get_bit(reg, bit_val)                    (reg & (1 << bit_val))

//For extracting the states of masked bits of a register
#define get_reg(reg, msk)                        (reg & msk)
```

Unlocking a Locked STM8 Chip

If you have accidentally locked a STM8 chip by setting the **Readout Protection** configuration bit and no longer able to use it, you can unlock it easily.

When you lock a chip, the programmer interface will give you a warning notification. If you retry to reprogram/erase a locked chip you'll get an error like this:
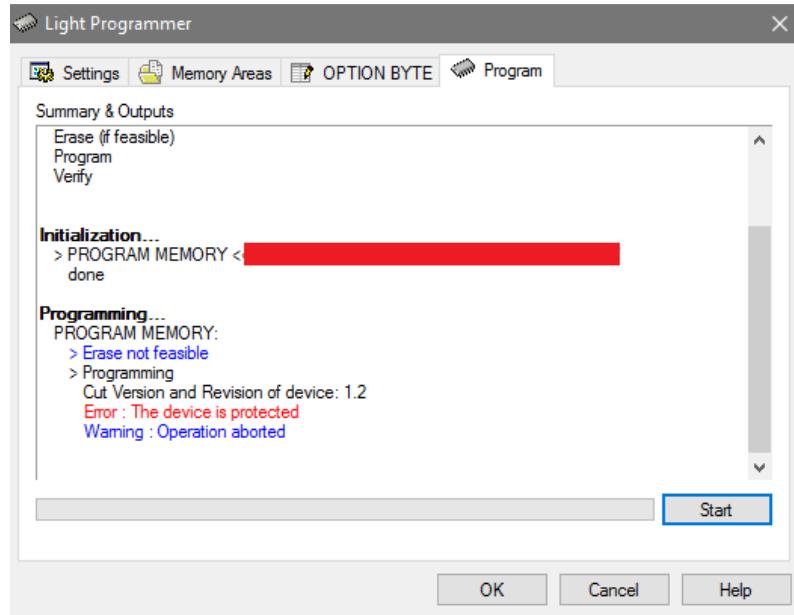
No matter what you do, you won't be able to use it.

To unlock, go to the light programmer interface of STVD and check the **Unlock Device** checkbox as shown below:
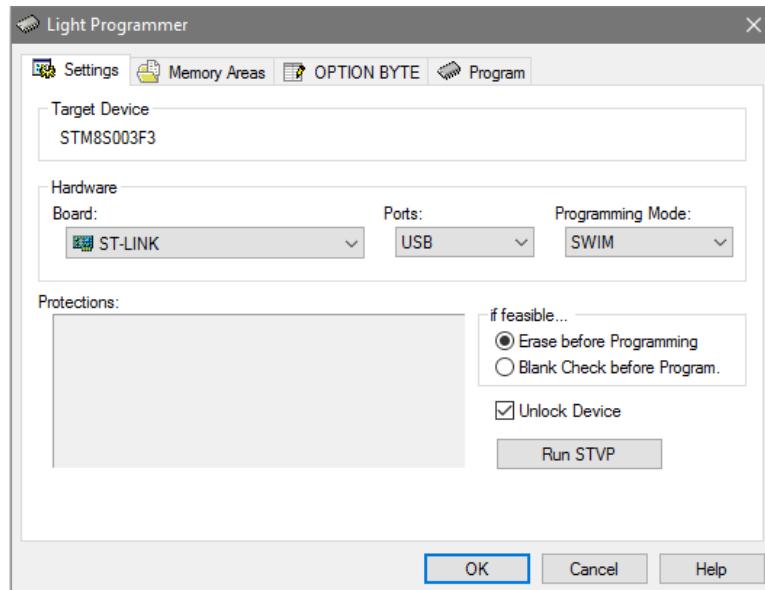
Also select **Erase before Programming** radio button because it is highly likely that your target chip is not empty. Now once you retry to reprogram, it will get unlocked.

Mastering C Language

You need not to be a C whiz to work with microcontrollers but certain things will surely help you to resolve some critical problems with simple codes. You must check supported data types whenever you begin working in a new development environment and should always use unsigned-signed designations to avoid unnecessary mistakes. Likewise, variable size is also important. Pointer, structures, unions and arrays are helpful features of C-language. You must learn how to use and apply them successfully. Without these you can still work but things will look really dirty. When coding for a new work, you must try to settle what you wish from your system and how should it behave. There should be an organized workflow and thereby your code will automatically be formulated in a state-of-machine algorithm or as a real-time system. You must try to avoid delays and loops wherever possible. Try to avoid polling and use interrupt-based systems. This will make your device behave in real-time with zero latency. However, you must be careful in handling interrupts because interrupts within interrupts will cause your system to crash miserably. Functions make things modular and thus easy to modify or debug. Repeated tasks should be placed in functions. A blinking LED code may look simple and stupid but sometimes very useful for testing stuffs. Some basic knowledge on mathematics and algorithms are also requirements for becoming a good embedded-system specialist.

Epilogue

In the end, I would like to share that my tiny raw-level knowledge and experiences with STM32s (http://embedded-lab.com/blog/stm32-tutorials/) earlier paid off handsomely. Due to that I was able to compiler this article decently and quickly. Personally, I feel that whosoever knows STM8 micros well will master STM32s and vice-versa because except the cores all hardware in both architectures are not just similar but same sometimes. This is for the first time I have admired STM's SPL. My experiences with STM32 SPL was not well as so I decided to go on my own. However, this time things were different. Things were joyful and less difficult.

I would like to thank a few people who influenced me in composing this article:

- Firstly, my friends and acquaintances. I wanted to help them out and that drove me to dig things deep.

- **Mr. Ben Ryves** (benryves.com/tutorials/stm8s-discovery/).
  Though his methods are different than mine, his article guided me a lot in the beginning. It is perhaps the most popular site for tutorials on STM8s and well organized. He used STM8S105 Discovery.

- **Mark Stevens** (http://blog.mark-stevens.co.uk)
  His tutorials on STM8 are not based on SPL. He showed stuffs with raw-level register access and with IAR compiler. Still his blog is informative.

- http://www.emcu.it/. This Italian site was helpful in getting some early info.

- STMicroelectronics team for releasing the STM8CubeMX during my writeup.

- Cosmic team for freeing up their C compiler.

I spent nearly three months straight putting together all these things and at present, I must say that I have great expectations from STM8 microcontrollers.

Happy coding.

Author: Shawon M. Shahryiar
https://www.facebook.com/groups/microarena
https://www.facebook.com/MicroArena   *24.04.2017*