



# Building & Using Libraries for Output

April 19<sup>th</sup>, 2016

 The work in this sheet has to be finished in the current week!

In this and the upcoming exercises, you will create several drivers and modules for the peripherals existent on the SES board. Those will be reused by later (also graded) exercises to create more and more complex systems. To enable reuse of all the basic functionality, you will put your drivers (starting with LED in this exercise) into a library which will be referenced by upcoming projects as needed. Additionally, you will learn to include and use precompiled libraries for UART and LCD.

 Even though this exercise is not graded, the created library is part of the later submission and the grading, so coding style is important!

## Task 2.1 : An SES library

Create a new project named `ses`, as target choose "AVR cross target **static library**". Projects which use the library have to set their include path (to find the header files) and link the library created by this project, but more on that later – for now, let's create a driver to access the LEDs.

As you will need the sources of your library for the graded exercises later, you should commit your `ses` project to your SVN repository now. Instructions can be found in the AVR Eclipse Tutorial. Please make sure that you only commit source and header files (NOT compiled object files, the library file or the whole Debug/Release folder). Make also sure that your driver library ends up at

<https://svn.ti5.tu-harburg.de/courses/ses/2016/teamX/ses> where X is your team's number.


 **Keep the structure of your `ses` directory flat, i.e., do not use subdirectories there!**

## Task 2.2 : Writing an LED Device Driver

In this lab, we'll concentrate on LEDs and buttons. As you have to reuse your driver library in future exercises, be careful to make everything work properly. As a first step, download and extract the *sheet2\_templates.zip* archive from the Stud.IP page.

In the Eclipse tutorial one LED was accessed using bit operations. Obviously, this is not very handy. Furthermore, this way of using LEDs is no good solution from a software-engineering point of view at all. As a result, we will develop a driver for using the LEDs. The driver consists of three parts: the header files *ses\_common.h*, *ses\_led.h* and the source file *ses\_led.c*. These files should be copied your newly created `ses` library project.

The given header files already define all constants, e.g., the port and the LED pins. Furthermore, the header files contain all function declarations and the desired functionality of the functions. In this exercise you have to implement the given function prototypes! Empty functions are already present in *ses\_led.c*.

 Use the macro `DDR_REGISTER(x)` from *ses\_common.h* to calculate the data direction register for a given port `x`. This macro helps to avoid redundancy.



The LEDs are *low active* (a logical *high* will turn them off)



For each operation ensure that all other bits remain unchanged!

### Task 2.3 : A Simple Traffic Light

Testing is an important part of the software development process. For this reason, you will write a very basic traffic light application using your newly implemented LED library. The traffic light is supposed to abide by the following cyclic scheme:

Step	LEDs	Timing
1	Green	5.0 s
2	Yellow	0.5 s
3	Red	2.0 s
4	Red and Yellow	0.3 s
5(1)	Green	5.0 s
...		

As advertised, you now need to use the library created in the former exercise. First, create a new project for your traffic light application. The project has to end up at

[https://svn.ti5.tu-harburg.de/courses/ses/2016/teamX/task\\_2](https://svn.ti5.tu-harburg.de/courses/ses/2016/teamX/task_2) where X is your team's number.

The new project has to know the location of the header files of your library and also needs to link to the compiled library. To achieve this with Eclipse, right click your new project and navigate to

**Properties->C/C++Build->Settings->Tool Settings;**

there, use AVR Compiler/Directories for the include paths and AVR Linker/Libraries for the linking:

- Under **AVR Compiler->Directories** click the add symbol, choose Workspace and select your library project's folder
- Under **AVR C Linker->Libraries** click the add symbol for libraries and type `ses`; then click the add symbol for library paths, choose workspace and select the `ses/Debug` directory

**Make sure that the settings are made for the configurations (Debug/Release) you currently use and remember this later if you change anything!** Additionally, to make code navigation and resolution of headers available, add your library project to the list of referenced projects (**Project Properties->Project References**) and to make sure changes in the library trigger a rebuild of projects using it, add the active configuration of your `ses` project to the referenced projects in **Project Properties->C/C++ General->Paths and Symbols->References** tab.



To avoid having to setup all this again and again, it is a good idea to create an empty template project with all those settings made and create a new project by copying and renaming this empty template project. **Be careful to not copy a project which is under version control, because the versioning info will be copied, too, which is definitely not what you want!**

In your new traffic light project, create a `main.c` file and implement the traffic light functionality (don't forget to include the `ses_led.h` header) directly within a loop in `int main(void)`. After this you can compile and flash to test the program. You can restart your program using the **Reset** button on the SES Development Board.

## Task 2.4 : Linking UART and LCD Libraries

- Copy the *libuart.a*, *liblcd.a*, *ses\_uart.h* and *ses\_lcd.h* to your ses library project (do not check in the libraries to your repository, however).
- Open the properties view of the traffic light project (right click on project name -> *properties*).
- Open *C/C++ Build -> Settings -> AVR C Linker -> Libraries* and add *uart* and *lcd* to the field *Libraries* (prefix *lib* and the suffix *.a* are resolved automatically); also, on the same page, add the path of the libraries to the search path (-L option).

## Task 2.5 : Start-up Messages

- Include *ses\_lcd.h* and *ses\_uart.h* in the *main.c*.
- Add the lines `uart_init(57600);` and `lcd_init();` to the initialization section of the *main.c*.
- To print a message to UART and LCD, use the statements `fprintf(uartout, "START\n");` and `fprintf(lcdout, "START");`, respectively.

After compiling and flashing, the microcontroller sends the message via the UART to the PC. However, we need a program to receive the data. If not already done, you have to install the driver for the USB-FTDI cable. Have a look into the Tutorial for reference. Now, you can install a terminal program, e.g. *cutecom* for Ubuntu or *Termite* ([http://www.compuphase.com/software\\_termite.htm](http://www.compuphase.com/software_termite.htm)) for Windows. Start the terminal program, select a baudrate of 57600 and select the assigned COM port for the FTDI cable. All other initial settings can remain (8N1 no handshake). Try to connect to the port. If everything works you should see output messages when you press the reset button.

- 🔗 The UART and LCD libraries define FILE descriptors, which can be used together with `fprintf`. If you want to use `printf`, then add the line `stdout=uartout;` right after the initialization of the UART. Of course this only works for one of UART and LCD at the same time.

## Task 2.6 : Printing the State of the Traffic Light

- Create a function with the signature `void printState(const char* state)` that prints out the character string via UART and LCD.
- Use this function after each state change to signal the current state of the traffic light.

- 🔗 By using `fprintf(uartout, "%s\n", state);` you can include the character string in the output.
- 🔗 Have a look into the *ses\_lcd.h* for additional functions that will help you to write the state to the correct position to the LCD.

## Task 2.7 : Using the UART Input (Optional)

- 🔗 If you have no time left to finish this task during this week, please skip it to catch up in the next week. You can also receive bytes from the PC. For this purpose, use the function `uart_getc()`. Program a simple pedestrian traffic light, i.e. only switch the light from green to red if a specific character was received over the UART.