Digitally-Adjustable Phrenic Nerve (DAPhNe) Stimulator System

Getting Started Guide

Created by:	Alexey Revinski
Revision:	1.0
Date:	18 September 2017

Table of Contents

Table of Abbreviations

BPM	Breaths-per-Minute
CAD	Computer-Aided Design
DAC	Digital-to-Analog Converter
DAPhNe	Digitally-Adjustable Phrenic Nerve (Stimulator)
DIP	Dual In-line Package (electrical component package)
EDA	Electronic Design Automation (software)
EMI	Electro-Magnetic Interference
GND	Ground
GUI	Graphical User Interface (software)
IAR	Ingenjörsfirman Anders Rundgren (software company)
IDE	Integrated Development Environment
LED	Light-Emitting Diode
MCU	Micro-Controller Unit (also PCB name)
NDEF	NFC Data Exchange Format
NFC	Near Field Communication
PC	Personal Computer
PCB	Printed Circuit Board
RAM	Random-Access Memory
SPDT	Single-Pole-Dual-Throw (switch type)
STIM	Stimulation circuit (PCB name)
STM8(L)	STMicroelectronics' 8-bit MCU line (L for low-power)
SWIM	Serial Wire Interface (debugging protocol)
USB	Universal Serial Bus
VBATT	Battery Voltage (electrical node/pin)

1 Introduction

This is a Getting Start Guide for future developers of the Digitally-Adjustable Phrenic Nerve (DAPhNe) Stimulator System, created by a team of undergraduate students at Northwestern University as part of their Biomedical Engineering capstone design project in 2016-2017 academic year.

This document was developed as a hardware and software guide to aid the next team working on the project. This document is **not** aimed to provide any physiological information that guided this project or any technical data about the device. It assumes that the reader has been already familiarized with the 2016-17 BME390 Project Reports and the DAPhNe Technical Reference v1.0 document.

2 Important Documents

The reader should familiarize himself with the following documents before setting up the project:

- Implantable Phrenic Nerve Stimulator System: Final Report (2016)
  - Outdated information from Fall of 2016, but a good discussion on the focus disease, state-of-the-art solutions to the problem, and physiological research done by the team up to that point
- Digitally Adjustable Phrenic Nerve (DAPhNe) Stimulator - Final Report: Quarter II (2017)
  - Mostly up-to-date technical information, with some reference to the animal model experiments planned for the initial prototype and performed in Spring/Summer of 2017
- DAPhNe Stimulator System Technical Reference: DAPhNe Stimulator Prototype v1.0, Rev 1.1
  - A complete technical information reference on the hardware and software of the DAPhNe Stimulator Prototype v1.0, which served in the abovementioned animal model experiments.

3 Hardware

The hardware of the prototype consists of multiple custom-designed and off-the-shelf parts, many broken-out on a DIP-style breadboard, and all incorporated into a single breadboard unit housed in a plastic case.

3.1 Checklist

The PCB hardware included in this prototype are the shown in Table 1.

Table 1: PCB Hardware

Name	Description
Custom DAPhNe MCU Board	Hand-assembled custom-designed PCB featuring an STM8L152 microcontroller and supporting circuitry; also, a plug-in hub for the ANT7-T-M24LR04 NFC receiver.
Custom DAPhNe STIM Board	Hand-assembled custom-designed PCB featuring a 10V boost converter and other stimulation-related components. Interfaces with DAPhNe MCU.
ANT7-T-M24LR04	Reference NFC antenna board featuring M24LR04 passive NFC memory tag from STMicroelectronics
PRT-10255 LiPower	3.3V buck-boost converter board by SparkFun electronics. Used to power the microcontroller and NFC memory
ST-LINK/V2	Debugging tool for STM8 and STM32 devices from ST, used in conjunction with IAR Embedded Workbench for STM8 to program firmware of the device
Custom LIR2450 Breakout Board	Hand-assembled board featuring a slide switch and push button to connect the battery to the circuit. Multiple terminal blocks provide easy access to the battery's terminals
STEVAL-ISB039V1R	A Qi-capable charging coil from STMicroelectronics; used in conjunction with STEVAL-ISB039V1T board for wireless charging of the battery.
CR95HF Demo Board	An NFC transceiver board used to communicate with the M24LR04 passive NFC memory. Capable of reading to and writing from M24LR04.s

Other hardware included in the hardware kit is either not used in the device prototype, or has been used in the past for development; there are PCB and breadboard passives and some residual active components. This document will not cover these components.

3.2 *Hardware Components: Bird's-Eye View*

Figure 1 depicts the current hardware prototype of the DAPhNe Stimulator.

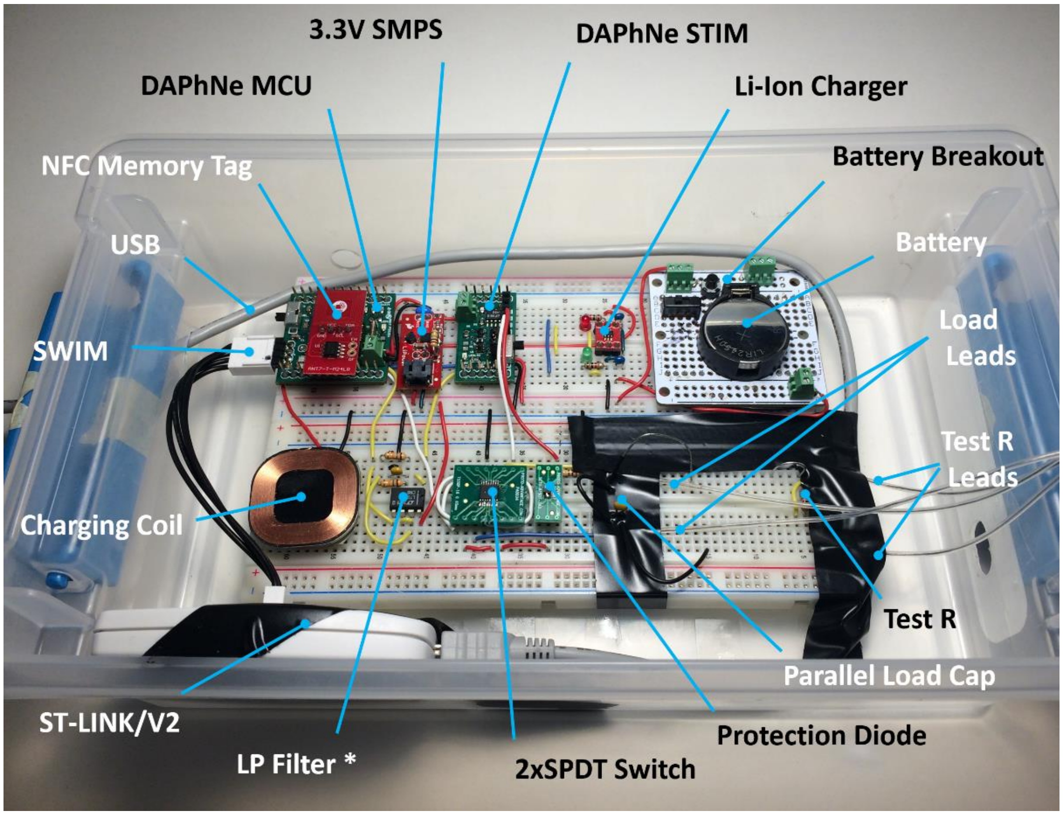


Figure 1: Prototype Overview

* Residual component; not needed for normal operation

Please refer to the DAPhNe Technical Reference document for more details on each of these components. This setup allows to quickly reprogram the device by merely plugging in the USB connector into a PC USB port. It also includes Avery test leads, which can be used to interface with Avery anode plates and electrodes, included in the passed-off hardware kit. There is a test resistor with leads that can be connected to the output load leads to quickly test operation of the device by measuring the voltage signal across the test resistor. The low pass filter circuit was added to smooth out certain spikes in the DAC output of the microcontroller; the effect was negligible, and the circuit can be taken out of the system. The wiring of the 2xSPDT switch and related nodes may have been flipped such that the polarity signal is held low for most of its duty cycle, rising to logic 1 only for the positive recharge pulses.

3.3 Powering the DAPhNe Boards

In the assembled DAPhNe Stimulator Prototype v1.0, powering the circuit is easy – there is a slide switch that connects the positive battery terminal to the positive power supply rail. When the slider is closest to the battery (Figure 2, left), the battery is connected to the rest of the circuit; when the slider is in the opposing position, the battery is disconnected (Figure 2, right).

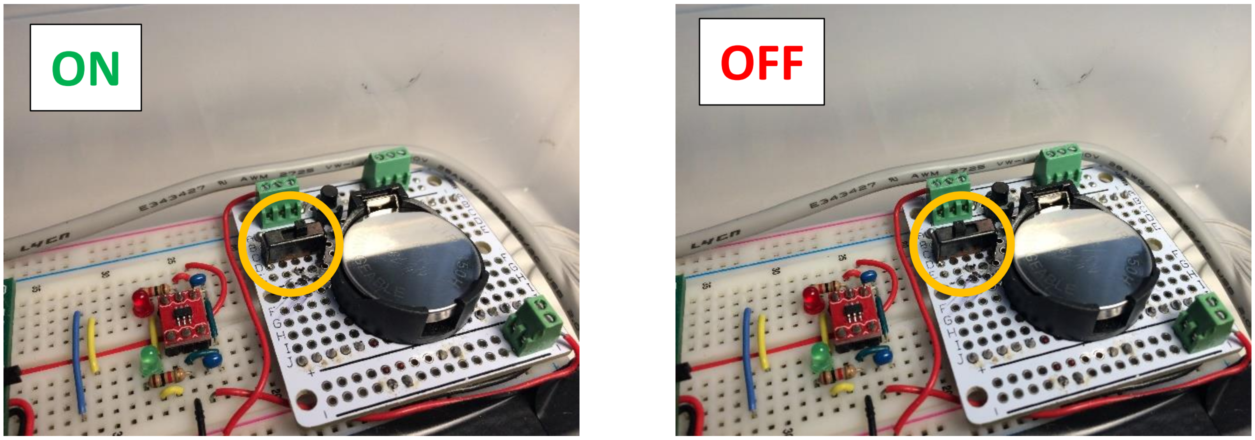


Figure 2: Battery ON/OFF (left/right) switch

Additionally, the battery board features a push-button that also connects the battery to the circuit when pushed. It can be used instead of the slide switch to power the circuit briefly.

Both DAPhNe MCU and DAPhNe STIM boards feature slide switches connecting and disconnecting power to the boards. The switches are in the ON position when, looking down from above, the slider is on the right of the switch (Figure 3, bottom left). Figure 3 shows the MCU board switch in the ON position.

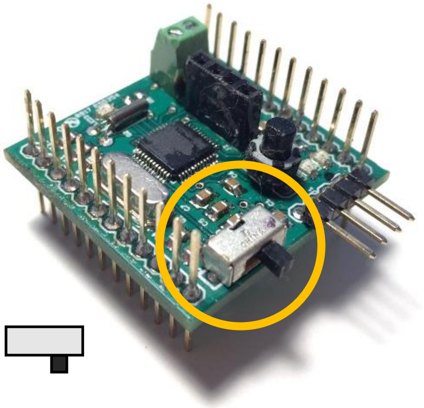


Figure 3: MCU board switch ON

The DAPhNe MCU board is powered by the LiPower 3.3V supply board from SparkFun Electronics. The LiPower board gets an unregulated supply directly from the battery. Alternatively, the MCU board can be powered by supplying regulated 3.3V on its VBATT terminal in the screw-in terminal block and turning the slide switch ON (Figure 4), or by supplying 3.3V directly to its 3.3V pin, in which case the switch will be bypassed. The DAPhNe STIM Board can be powered by supplying 3.3V to its VBATT pin, which is connected to the rest of the circuit by a similar slide switch. For pinout of the two boards, please refer to the associated EagleCAD board files.

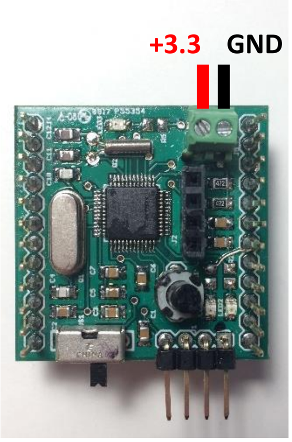


Figure 4: MCU board power terminal

Originally, both DAPhNe MCU and DAPhNe STIM have been designed with red LEDs that indicated whether the circuits have been powered on or not. The LEDs are still there, but for power consumption testing, their current-limiting resistors have been de-soldered from the boards, effectively disconnecting the LEDs from the circuits. Feel free to install these back for clear “powered on” visual indication.

3.4 Testing Device Output

To test the output, you need a multimeter and an oscilloscope; nScope will work fine for this. Do not power the circuit until Step 8.

1. Connect the load leads to test resistance leads.
2. With a multimeter, make sure the load leads are connected across the resistor.
3. Make sure both DAPhNe board switches are on.
4. Make sure the battery you are using is relatively fresh. It should supply above 3V; a good range is 3.6-3.8V. Although the design takes into account overdrawing the battery… don't.
5. You are about to connect an oscilloscope probe across the test resistor. **IF THE PROTOTYPE IS CONNECTED TO YOUR PC, MAKE SURE THE PC IS NOT PLUGGED INTO THE WALL SUPPLY BEFORE POWERING ON THE PROTOTYPE.**
   - The ground terminal of your oscilloscope probe may be directly earthed. So is your PC ground. If you connect the oscilloscope ground to any node other than PC's ground while the circuit is on, you will most likely fry the DAPhNe circuit. Or your PC. Or the $$$ oscilloscope…
   - Disconnecting the PC from the wall will make it "float" with respect to the earth, and then you can connect your oscilloscope probe to anywhere within the circuit.
6. Connect the oscilloscope probe across the test resistor. Most oscilloscope probe ground terminals end with an alligator clip; there is a wire breakout specifically included for this (Figure 5).
7. Set your oscilloscope screen to show 500µs interval and +/- 5V. Set the trigger to falling edge. Trigger continuously, not a single read.

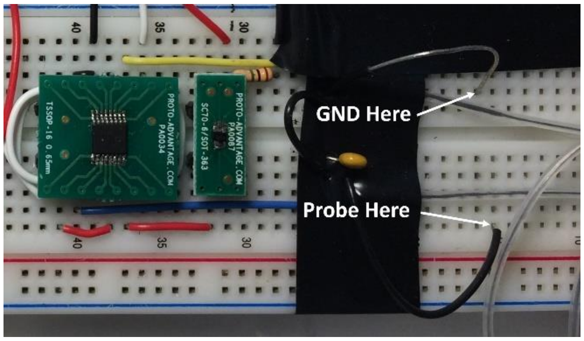


Figure 5: Test Probe Location

8. Power the prototype with the battery switch. You may want to install a simple "power on" LED for visual reminder that the prototype is powered on.

On your oscilloscope screen, you should see the signal such as the one shown in Figure 6 (captured with nScope). The signal edges may be very spiky – in that case, the load capacitor value needs to be increased. In tissue, these spikes are smoothed out by the natural body capacitive components.

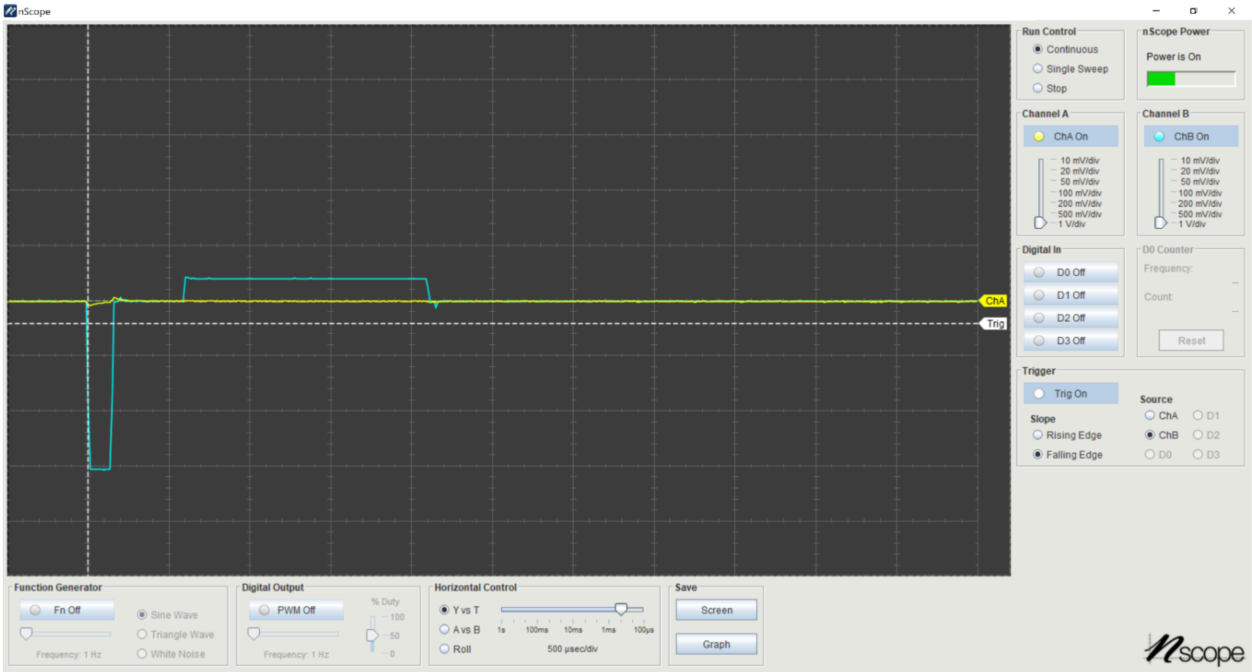


Figure 6: Test Result

If using nScope, connect the “Probe GND” node to nScope ground **(again, taking care that your PC is not connected to the wall)**, and use one of the analog channels to connect to the other side of the test resistor (second load lead). Connecting both load leads to different analog channels will not work, as the circuit will float with respect to PC ground.

4 Software

This project entailed development of firmware for an 8-bit microcontroller, PCB design, circuit simulation, and code version control. Listed below are tools needed and recommended for future development of the device. All of these are free of charge, unless specified by ($).

4.1 *Checklist*

4.1.1 Essential Tools

Below is a list of software directly used for development of the DAPhNe Stimulator System:

- **IAR Embedded Workbench for STM8**
  - Integrated Development Environment (IDE) used to generate firmware for the microcontroller. See instructions in Section 4.4 for installation instructions.
- **Autodesk EagleCAD**
  - 2-D EDA software for used to develop the DAPhNe PCBs. Great **tutorials** online!
- **STSW-M24LR011**
  - A GUI from STMicroelectronics used to control the CR95HF NFC transceiver board. See instructions on using this software in Section 4.7.3.

The following software is usually included with the IAR installation (no steps needed). However, if you find that IAR is unable to flash the microcontroller, you may need to install these separately:

- **STSW-LINK009**
  - Driver for the ST-LINK/V2 in-circuit debugging interface
- **STSW-LINK007**
  - Optional (but recommended) firmware upgrade for ST-LINK/V2

4.1.2 Optional Tools

These tools are not essential, but they greatly aided the development of the DAPhNe Stimulator System:

- **LTSpice**
  - SPICE simulator software – was used primarily to simulate the stimulation circuit behavior
- **Notepad ++**
  - A must-have text editing tool. Supports the overwhelming majority of text-based file types, and has numerous plug-ins for word-completion, file validation, etc.
- **nScope** ($)
  - Useful for quick circuit testing – contains decent low-end oscilloscope functionality, function-generator, and power supply; interfaces with a regular DIP-style breadboard.
- **Ultra Librarian Free Reader**, along with the Ultra Librarian **online part database**
  - Cloud-based PCB part database; the Free Reader can be used to convert between different types of component files. Nicely complements any EDA software.

4.2 Git

All developed source code, along with PCB EDA files, device datasheets, and reference documentation has been maintained and version-controlled using git; the project is contained in a remote repository on GitHub.com. To maintain the project, you need to install **git**. For those unfamiliar, this is a great version control tool and there are lots of great **tutorials** on it online.

4.2.1 Git Tools

The following tools are recommended to seamlessly maintain and version-control the developed code. Only one of the following is needed:

- **GitHub Desktop**
  - A simple git client developed to work specifically with GitHub
- **Tortoise Git**
  - A great Windows shell extension for git. Easy to use; very powerful version control tool

4.2.2 Getting Project Materials

The **remote repository** on GitHub.com contains the entirety of the relevant project materials.

IMPORTANT: This project repository is personal repository; to add to this, the repository may have to be taken down in near future. With this in mind…

If the repository still exists by the time you read this, you may fork it to develop it on your local machine – there are a lot of tutorials online on how to do this. By making a fork, you can develop the project and maintain it on your own local machine; in the future, if it is determined that the project can continue to be public, you may submit a pull request to me and we can work out how to merge the project into my public repo, or you may choose to make your own public repository. In the near future, attempts to push to my original personal repo will be ignored.

If the repository does not exist publicly by the time you read this, contact me via email at **alexeyrevinski2017@u.northwestern.edu** to get the project materials.

IMPORTANT: The information in the README.md file in the repo **may not contain most up-to-date information**. Most recent information on the project can be found in the DAPhNe Technical Reference document, as well as this document.

4.3 Project Structure

The project structure is simple – the folder names are self-explanatory. This section will only explain the structure of the MCU-related code project.

The MCU project resides in the `[DAPhNe_Folder]/MCU` folder. In there, the `Project` folder contains the IAR Embedded Workbench project, and the rest of the folders are source libraries from STMicroelectronics.

Going into the `Project` folder, the `EWSTM8` folder contains all of the IAR-specific files. The `inc` and `src` folders include all of the custom DAPhNe C header and source files, respectively.

IMPORTANT: Changing folder names and/or locations in the `[DAPhNe_Folder]/MCU` folder will break the build process. See Section 4.5.2 for more details.

4.4 Installing IAR EWSTM8 Kickstart Edition

IAR Embedded Workbench for STM8 is a professional integrated development environment tool that includes a complete C/C++ development toolchain – a C/C++ compiler, C/C++ libraries, an assembler, a linker, a code editor, project manager, and a variety of other tools. It is available in a free, code-size limited Kickstart edition.

In the Kickstart Edition for STM8 devices, the code size limit is 8 Kbytes; there is also no source code for runtime C libraries, no support for MISRA C, and no IAR C-STAT functionality, along with other restrictions. However, for this stage of development, the Kickstart edition is very sufficient. Alternatively, the developer may choose another toolchain to use with the source code included in the online repository.

To download IAR EWSTM8 Kickstart Edition, follow these steps:

1. Visit the **IAR EWSTM8** website.
2. Towards the bottom of the page, click on the **+** symbol to the right of "Download a free trial!".

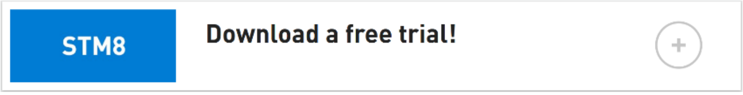


Figure 7: Free Trial Field

3. Click "Download Software" button on the left of the expanded field.
4. Run the executable file, if the process has not started automatically.
5. Installation wizard will run (Figure 8).
6. Click "Install IAR Embedded Workbench for STMicroelectronics STM8".

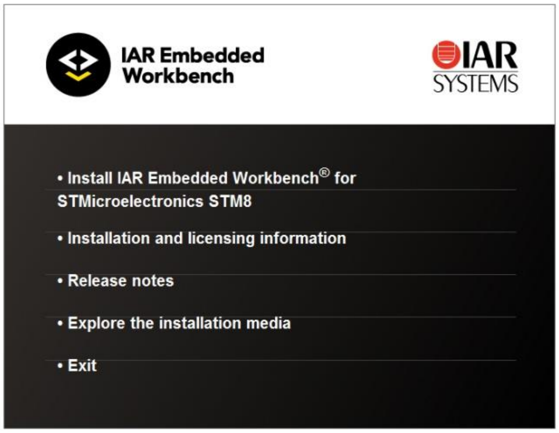
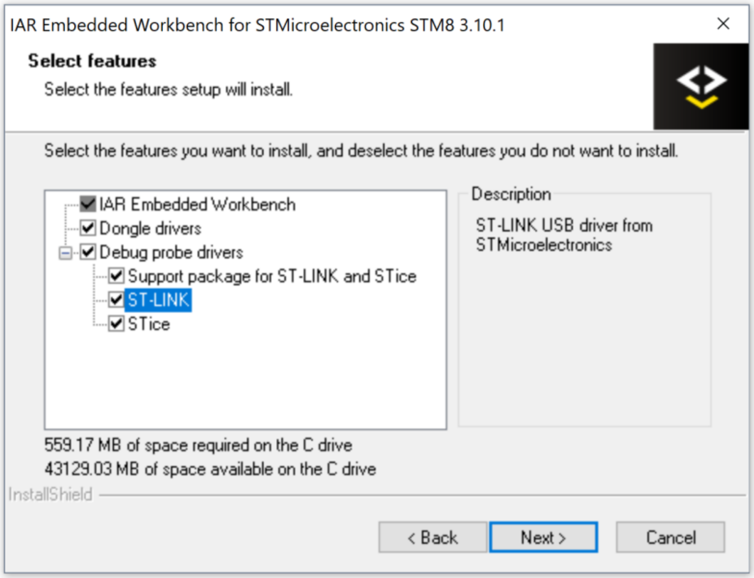

IAR Embedded Workbench
IAR SYSTEMS
• Install IAR Embedded Workbench® for STMicroelectronics STM8
• Installation and licensing information
• Release notes
• Explore the installation media
• Exit

Figure 8: IAR Installation Wizard

7. Click “Next” until the following window shows. Make sure that all components shown below are installed.



IAR Embedded Workbench for STMicroelectronics STM8 3.10.1

Select features

Select the features setup will install.

Select the features you want to install, and deselect the features you do not want to install.

- ☑ IAR Embedded Workbench
- ☑ Dongle drivers
- ☑ Debug probe drivers
  - ☑ Support package for ST-LINK and STice
  - ☑ ST-LINK
  - ☑ STice

Description

ST-LINK USB driver from STMicroelectronics

559.17 MB of space required on the C drive
43129.03 MB of space available on the C drive

InstallShield

Figure 9: Features to include

8. After the product is installed, start IAR.

9. Since you do not have a license yet, a License Wizard will start and require a license key.

10. Select “Register with IAR Systems to get an evaluation license”, and follow instructions for registering online.

11. After completing registration and email verification, proceed with the License Wizard.

12. You are now able to use the Kickstart Edition of IAR Embedded Workbench for STM8.

4.5 Setting up the IAR EWSTM8 Project

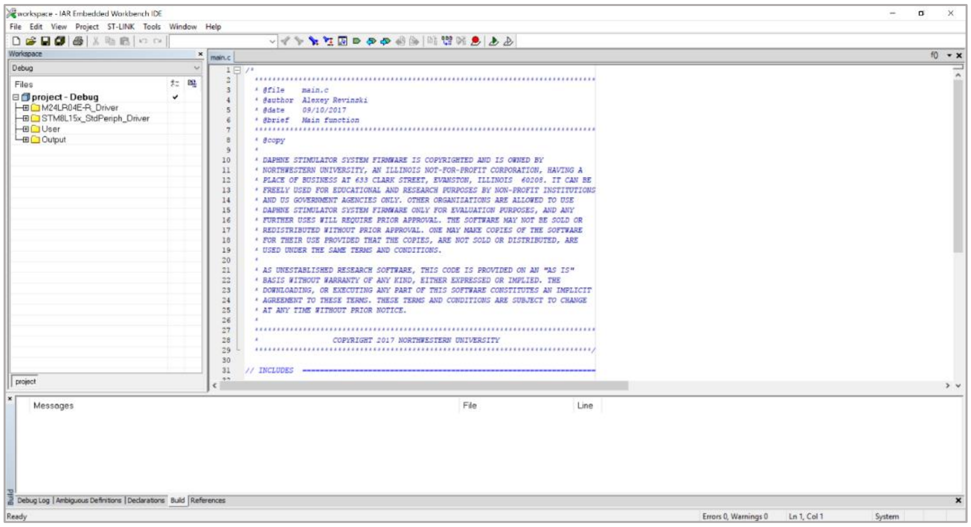
This section will give an overview on opening the IAR project, checking its settings, and building it.

4.5.1 Opening the Project

To open the STM8 project in IAR:

1. Open IAR EWSTM8
2. Click *File* → *Open* → *Workspace…*
3. Select "`workspace.eww`" file in: `[DAPhNe_Folder]/MCU/Project/EWSTM8`
4. Click "Open".

The environment should change to the window shown in Figure 10.



```
/*
  ****************************************************************************
  * @file    main.c
  * @author  Alexey Revinski
  * @date    09/10/2017
  * @brief   Main function
  ****************************************************************************
  * @copy
  *
  * DAPHNE STIMULATOR SYSTEM FIRMWARE IS COPYRIGHTED AND IS OWNED BY
  * NORTHWESTERN UNIVERSITY, AN ILLINOIS NOT-FOR-PROFIT CORPORATION, HAVING A
  * PLACE OF BUSINESS AT 633 CLARK STREET, EVANSTON, ILLINOIS  60208. IT CAN BE
  * FREELY USED FOR EDUCATIONAL AND RESEARCH PURPOSES BY NON-PROFIT INSTITUTIONS
  * AND US GOVERNMENT AGENCIES ONLY. OTHER ORGANIZATIONS ARE ALLOWED TO USE
  * DAPHNE STIMULATOR SYSTEM FIRMWARE ONLY FOR EVALUATION PURPOSES, AND ANY
  * FURTHER USES WILL REQUIRE PRIOR APPROVAL. THE SOFTWARE MAY NOT BE SOLD OR
  * REDISTRIBUTED WITHOUT PRIOR APPROVAL. ONE MAY MAKE COPIES OF THE SOFTWARE
  * FOR THEIR USE PROVIDED THAT THE COPIES, ARE NOT SOLD OR DISTRIBUTED, ARE
  * USED UNDER THE SAME TERMS AND CONDITIONS.
  *
  * AS UNESTABLISHED RESEARCH SOFTWARE, THIS CODE IS PROVIDED ON AN "AS IS"
  * BASIS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED. THE
  * DOWNLOADING, OR EXECUTING ANY PART OF THIS SOFTWARE CONSTITUTES AN IMPLICIT
  * AGREEMENT TO THESE TERMS. THESE TERMS AND CONDITIONS ARE SUBJECT TO CHANGE
  * AT ANY TIME WITHOUT PRIOR NOTICE.
  *
  ****************************************************************************
  *                 COPYRIGHT 2017 NORTHWESTERN UNIVERSITY
  ****************************************************************************/

// INCLUDES  -----------------------------------------------------------------
```

Figure 10: Opened project workspace

The center-right field is the code editor field. Here, you can modify existing code or create new files. The top left field is the workspace field. Although it looks like a folder directory, **it is not**. This is merely an organizational tool – the IAR "folders", called groups, are not actually file system directories. The bottom field is a log field – it contains a debug log, build log, etc.

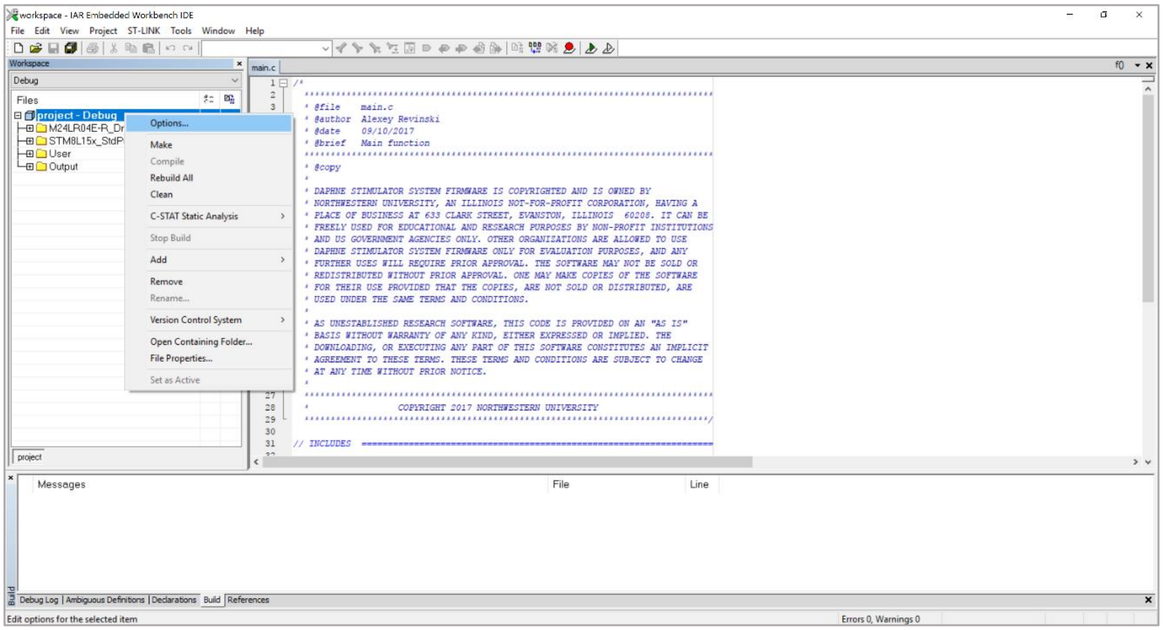
All of the developed DAPhNe Stimulator source code is located in the group `User`. The STMicroelectronics peripheral library for STM8L15x devices is contained in group `STM8L15x_StdPeriph_Driver`. Finally, code pertaining to the M24LR04 NFC memory is contained in the respective group as well. Files that end with "`_User`" have been originally part of a third-party solution, but have been modified to fit the requirements of this application.

The `Output` group contains – you guessed it – the output of the build process. The `project.out` file contains the object file dependencies – do not modify these files yourself after the build process, unless you know exactly what you are doing. There is also an optionally-generated linker map file in the output folder. This file is very useful to determine memory usage – for both Flash and RAM.

4.5.2 Validating Project Settings

In order to successfully build the project, the project's settings must be set correctly. While the project should build without errors right away, make sure that the correct options are set.

First, right-click on the project's root directory in the workspace and select Options (Figure 11).



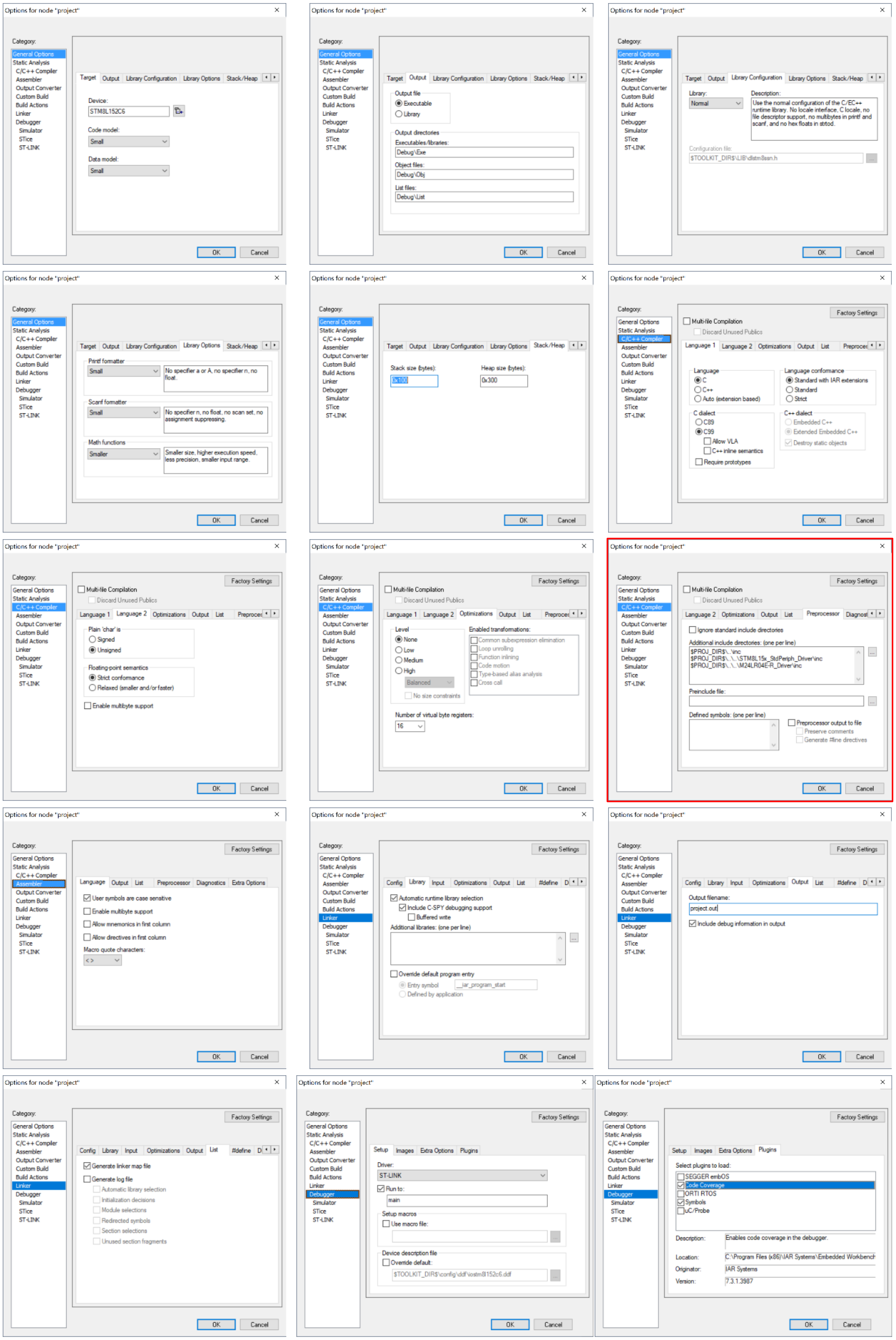
```
/*
 *****************************************************************************
 * @file    main.c
 * @author  Alexey Revinski
 * @date    09/10/2017
 * @brief   Main function
 *****************************************************************************
 * @copy
 *
 * DAPHNE STIMULATOR SYSTEM FIRMWARE IS COPYRIGHTED AND IS OWNED BY
 * NORTHWESTERN UNIVERSITY, AN ILLINOIS NOT-FOR-PROFIT CORPORATION, HAVING A
 * PLACE OF BUSINESS AT 633 CLARK STREET, EVANSTON, ILLINOIS  60208. IT CAN BE
 * FREELY USED FOR EDUCATIONAL AND RESEARCH PURPOSES BY NON-PROFIT INSTITUTIONS
 * AND US GOVERNMENT AGENCIES ONLY. OTHER ORGANIZATIONS ARE ALLOWED TO USE
 * DAPHNE STIMULATOR SYSTEM FIRMWARE ONLY FOR EVALUATION PURPOSES, AND ANY
 * FURTHER USES WILL REQUIRE PRIOR APPROVAL. THE SOFTWARE MAY NOT BE SOLD OR
 * REDISTRIBUTED WITHOUT PRIOR APPROVAL. ONE MAY MAKE COPIES OF THE SOFTWARE
 * FOR THEIR USE PROVIDED THAT THE COPIES, ARE NOT SOLD OR DISTRIBUTED, ARE
 * USED UNDER THE SAME TERMS AND CONDITIONS.
 *
 * AS UNESTABLISHED RESEARCH SOFTWARE, THIS CODE IS PROVIDED ON AN "AS IS"
 * BASIS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED. THE
 * DOWNLOADING, OR EXECUTING ANY PART OF THIS SOFTWARE CONSTITUTES AN IMPLICIT
 * AGREEMENT TO THESE TERMS. THESE TERMS AND CONDITIONS ARE SUBJECT TO CHANGE
 * AT ANY TIME WITHOUT PRIOR NOTICE.
 *
 *****************************************************************************
 *                   COPYRIGHT 2017 NORTHWESTERN UNIVERSITY
 *****************************************************************************/

// INCLUDES  =================================================================
```

Figure 11: Selecting project options

The following page (Figure 12) shows the options that should be selected in various tabs in the "Options" window. Other tabs are not important for building the project correctly.

IMPORTANT: The IAR Embedded Workbench has been configured to use relative paths to find various source files. These paths are found in *Project* → *Options* → *C/C++ Compiler* → *Preprocessor* tab (outlined with red in Figure 12). Should you rename or change folder location within `[DAPhNe_Folder]/MCU`, change the default paths in this tab as well.



General Options – Target:
Device: STM8L152C6
Code model: Small
Data model: Small

General Options – Output:
Output file: ● Executable ○ Library
Output directories
Executables/libraries: Debug\Exe
Object files: Debug\Obj
List files: Debug\List

General Options – Library Configuration:
Library: Normal
Description: Use the normal configuration of the C/EC++ runtime library. No locale interface, C locale, no file descriptor support, no multibytes in printf and scanf, and no hex floats in strtod.
Configuration file: $TOOLKIT_DIR$\LIB\dlstm8ssn.h

General Options – Library Options:
Printf formatter: Small – No specifier a or A, no specifier n, no float.
Scanf formatter: Small – No specifier n, no float, no scan set, no assignment suppressing.
Math functions: Smaller – Smaller size, higher execution speed, less precision, smaller input range.

General Options – Stack/Heap:
Stack size (bytes): 0x100
Heap size (bytes): 0x300

C/C++ Compiler – Language 1:
□ Multi-file Compilation
□ Discard Unused Publics
Language: ● C ○ C++ ○ Auto (extension based)
Language conformance: ● Standard with IAR extensions ○ Standard ○ Strict
C dialect: ○ C89 ● C99 □ Allow VLA □ C++ inline semantics
C++ dialect: ○ Embedded C++ ● Extended Embedded C++ ☑ Destroy static objects
□ Require prototypes

C/C++ Compiler – Language 2:
Plain 'char' is: ○ Signed ● Unsigned
Floating-point semantics: ● Strict conformance ○ Relaxed (smaller and/or faster)
□ Enable multibyte support

C/C++ Compiler – Optimizations:
Level: ● None ○ Low ○ Medium ○ High (Balanced) □ No size constraints
Enabled transformations: □ Common subexpression elimination □ Loop unrolling □ Function inlining □ Code motion □ Type-based alias analysis □ Cross call
Number of virtual byte registers: 16

C/C++ Compiler – Preprocessor:
□ Ignore standard include directories
Additional include directories: (one per line)
$PROJ_DIR$\..\inc
$PROJ_DIR$\..\..\STM8L15x_StdPeriph_Driver\inc
$PROJ_DIR$\..\..\M24LR04E-R_Driver\inc
Preinclude file:
Defined symbols: (one per line)
□ Preprocessor output to file □ Preserve comments □ Generate #line directives

Assembler – Language:
☑ User symbols are case sensitive
□ Enable multibyte support
□ Allow mnemonics in first column
□ Allow directives in first column
Macro quote characters: <>

Linker – Library:
☑ Automatic runtime library selection
☑ Include C-SPY debugging support
□ Buffered write
Additional libraries: (one per line)
□ Override default program entry
● Entry symbol __iar_program_start
○ Defined by application

Linker – Output:
Output filename: project.out
☑ Include debug information in output

Linker – List:
☑ Generate linker map file
□ Generate log file
□ Automatic library selection
□ Initialization decisions
□ Module selections
□ Redirected symbols
□ Section selections
□ Unused section fragments

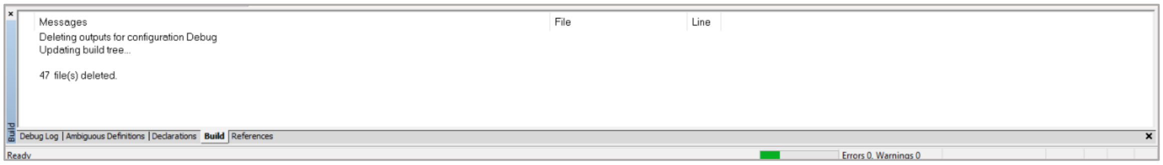
Debugger – Setup:
Driver: ST-LINK
☑ Run to: main
Setup macros: □ Use macro file:
Device description file: □ Override default: $TOOLKIT_DIR$\config\ddf\iostm8l152c6.ddf

Debugger – Plugins:
Select plugins to load:
□ SEGGER embOS
☑ Code Coverage
□ ORTI RTOS
☑ Symbols
□ uC/Probe
Description: Enables code coverage in the debugger.
Location: C:\Program Files (x86)\IAR Systems\Embedded Workbench
Originator: IAR Systems
Version: 7.3.1.3987

Figure 12: Various project options – only the most important ones

4.5.3 Building the Project

One the project settings have been confirmed, IAR's build process can be invoked by pressing F7, or, alternatively, the button on the top toolbar. For a first-time build, however, clean the project output files by selecting *Project*→*Clean*. Figure 13 shows the log upon cleaning the project.


Build
Debug Log | Ambiguous Definitions | Declarations | Build | References
Ready

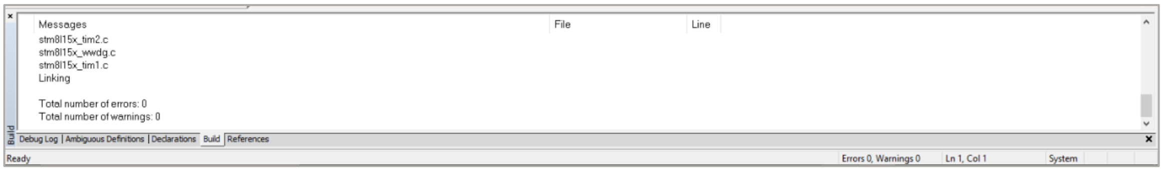
```
Messages                                        File          Line
Deleting outputs for configuration Debug
Updating build tree...

47 file(s) deleted.
```

```
Errors 0, Warnings 0
```

Figure 13: Clean log

Then, build the project using F7 or . If you have configured the project correctly, the log should display no errors or warnings.


Build
Debug Log | Ambiguous Definitions | Declarations | Build | References
Ready
Ln 1, Col 1
System

```
Messages                                        File          Line
stm8l15x_tim2.c
stm8l15x_wwdg.c
stm8l15x_tim1.c
Linking

Total number of errors: 0
Total number of warnings: 0
```

```
Errors 0, Warnings 0
```

Figure 14: Build process log

4.5.4 Project Test Status

My last minor changes to the project (formatting) have been made on 09/18/2017, and building successfully, resulting in proper device operation. If any errors arise, the problems are in your project settings or your specific platform compatibility. Check your drivers, C libraries, etc.

4.6 Updating STM8L Firmware

The DAPhNe MCU board should already be flashed with the latest firmware upon reception. However, if the board needs to be reprogrammed, follow these next steps.

4.6.1 Necessary Hardware

To flash the chip successfully, you need the following hardware, shown in Figure 15:

1. DAPhNe MCU Board (**A**)
2. ST-LINK/V2 debugging module (**B**)
3. Mini-USB-B-to-USB-A wire (**C**)
4. 4-pin SWIM Connector (**D**)

The board needs to be powered – either by a battery through the LiPower Board, or by another 3.3V supply. The ST-LINK/V2 utility does not provide power to the device on its own, and the flash process will not be successful if the board is not powered by an external supply.

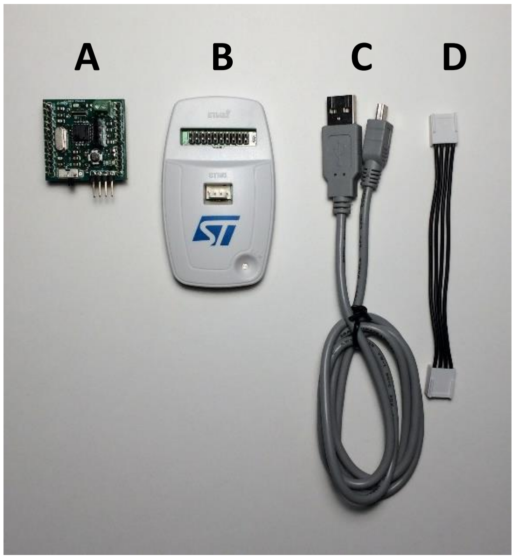


Figure 15: Debugging materials

The assembled setup shown in Figure 16 already has all of the above hardware in proper configuration; you only need to connect the USB connector to your PC to flash the device. Section 4.6.2 is for reference only.

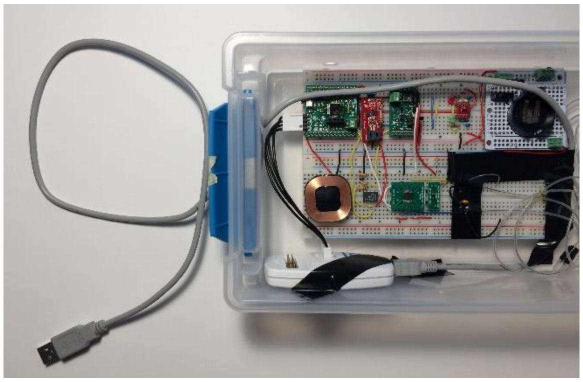


Figure 16: Assembled prototype

4.6.2 Connecting to DAPhNe MCU Board

To connect to DAPhNe MCU:

1. Connect the USB-A connector of the Mini-USB-B-to-USB-A wire to your PC
2. Connect the other side to the Mini-USB-B port of the ST-LINK/V2 debugging module
3. Connect the 4-pin SWIM connector to the ST-LINK/V2 debugging module. Take care to connect it so that the two plastic guide rails of the connector hug the plastic lip of the male connector of the ST-LINK/V2 (Figure 17)
4. Connect the other side of the 4-pin SWIM connector wire to DAPhNe MCU. As in the left part of Figure 18, make sure the two guide rails of the connector point upwards, and not down, as in the example on the right. Wrong polarity may damage the board and/or the debugger.

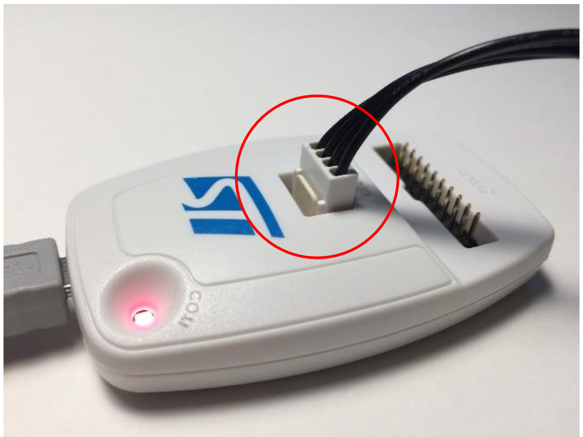


Figure 17: Connecting SWIM to ST-LINK/V2

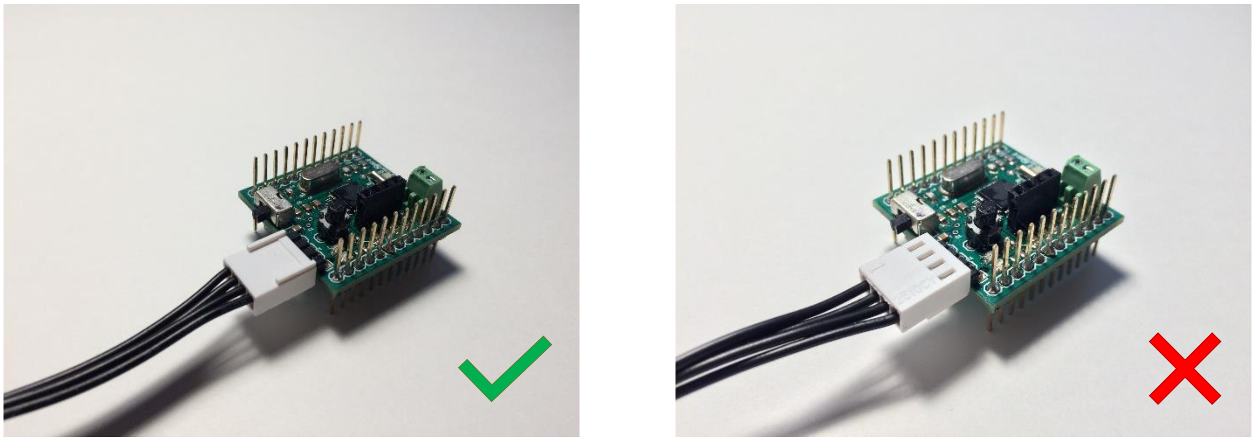


Figure 18: (Left) Correct way to connect SWIM to DAPhNe MCU; (right) wrong connection

The resulting assembly is shown in Figure 19.

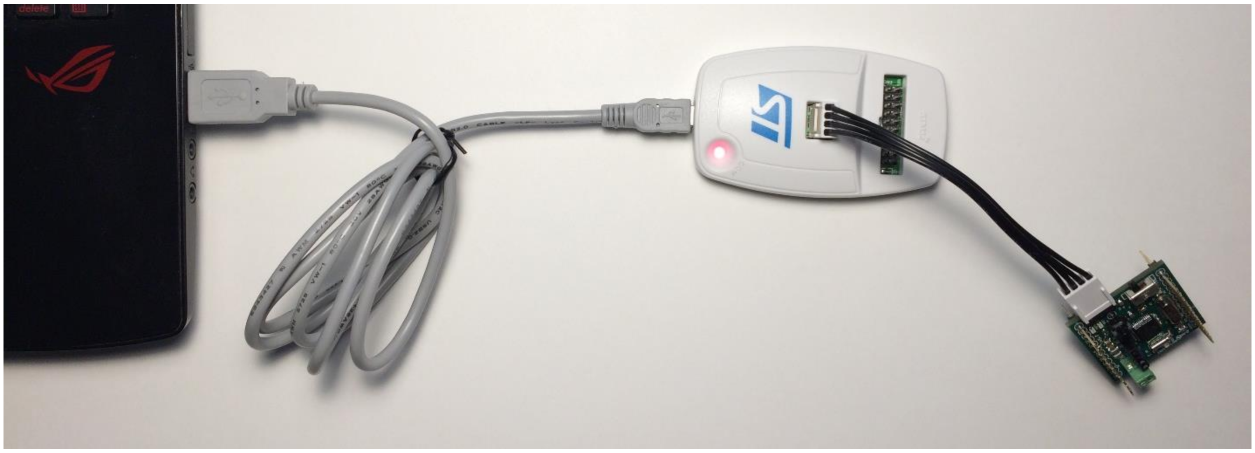
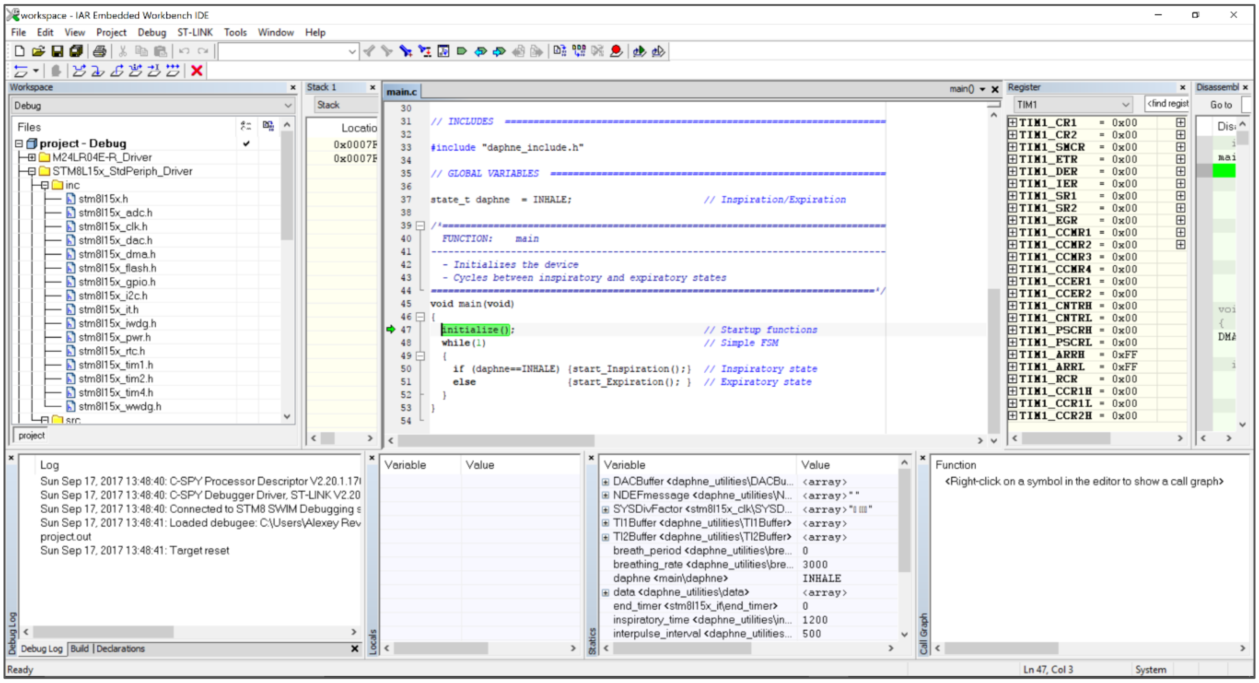


Figure 19: Fully assembled debugging interface (without the power supply to DAPhNe MCU)

4.6.3 Flashing the Microcontroller

After building the project in IAR, **powering the board**, and connecting the debugging interface, simply press ▶ to download the new firmware to the microcontroller. This will also start the debugging session, shown in Figure 20. If you get to this screen without any error messages, the chip has been successfully re-programmed, and you are ready to debug or develop further.



```
// INCLUDES  ==================================================================

#include "daphne_include.h"

// GLOBAL VARIABLES  ==========================================================

state_t daphne  = INHALE;                          // Inspiration/Expiration

/*============================================================================
  FUNCTION:    main
------------------------------------------------------------------------------
  - Initializes the device
  - Cycles between inspiratory and expiratory states
============================================================================*/
void main(void)
{
  initialize();                                    // Startup functions
  while(1)                                         // Simple FSM
  {
    if (daphne==INHALE) {start_Inspiration();}     // Inspiratory state
    else                {start_Expiration(); }     // Expiratory state
  }
}
```

```
TIM1_CR1    = 0x00
TIM1_CR2    = 0x00
TIM1_SMCR   = 0x00
TIM1_ETR    = 0x00
TIM1_DER    = 0x00
TIM1_IER    = 0x00
TIM1_SR1    = 0x00
TIM1_SR2    = 0x00
TIM1_EGR    = 0x00
TIM1_CCMR1  = 0x00
TIM1_CCMR2  = 0x00
TIM1_CCMR3  = 0x00
TIM1_CCMR4  = 0x00
TIM1_CCER1  = 0x00
TIM1_CCER2  = 0x00
TIM1_CNTRH  = 0x00
TIM1_CNTRL  = 0x00
TIM1_PSCRH  = 0x00
TIM1_PSCRL  = 0x00
TIM1_ARRH   = 0xFF
TIM1_ARRL   = 0xFF
TIM1_RCR    = 0x00
TIM1_CCR1H  = 0x00
TIM1_CCR1L  = 0x00
TIM1_CCR2H  = 0x00
```

```
Log
Sun Sep 17, 2017 13:48:40: C-SPY Processor Descriptor V2.20.1.17
Sun Sep 17, 2017 13:48:40: C-SPY Debugger Driver, ST-LINK V2.20
Sun Sep 17, 2017 13:48:40: Connected to STM8 SWIM Debugging s
Sun Sep 17, 2017 13:48:41: Loaded debugee: C:\Users\Alexey Rev
project.out
Sun Sep 17, 2017 13:48:41: Target reset
```

Variable	Value

Variable	Value
⊞ DACBuffer <daphne_utilities\DACBu...	<array>
⊞ NDEFmessage <daphne_utilities\N...	<array>" "
⊞ SYSDivFactor <stm8l15x_clk\SYSD...	<array>"▯ ▯▯▯"
⊞ TI1Buffer <daphne_utilities\TI1Buffer>	<array>
⊞ TI2Buffer <daphne_utilities\TI2Buffer>	<array>
breath_period <daphne_utilities\bre...	0
breathing_rate <daphne_utilities\bre...	3000
daphne <main\daphne>	INHALE
⊞ data <daphne_utilities\data>	<array>
end_timer <stm8l15x_it\end_timer>	0
inspiratory_time <daphne_utilities\in...	1200
interpulse_interval <daphne_utilities...	500

Function
<Right-click on a symbol in the editor to show a call graph>

Figure 20: Debugging session

If the program is not able to flash the chip, the problem is in hardware, unless IAR tells you explicitly that it is missing device drivers. If it does, check out the “Essential Tools” checklist. Otherwise, ask yourself:

1. Did I connect the SWIM connector correctly?
2. Did I power the boards with the battery switch?
3. Is the DAPhNe MCU slide switch in the ON position?
4. Is the ST-LINK/V2 powered on? If the red/green LED on the debugger is not shining, either the debugger is shot or your USB port is faulty.
5. Does the battery supply adequate voltage? You may need get a fresh one.
6. Does the LiPower supply 3.3V? If battery is okay and there is no 3.3V output from LiPower, the little board may have seen its last day.

4.7 Interacting with NFC Memory

The prototype uses an NFC memory tag reference board form STMicroelectronics – ANT7-T-M24LR04. The DAPhNe MCU board was designed to be a plug-in port for this memory tag.

4.7.1 Connecting NFC Memory to DAPhNe MCU

Figure 21 shows the proper way to connect the memory to the MCU board. Notice that the antenna portion of the memory tag board sits right above the microcontroller. If the board is plugged in the opposite way, the memory chip can get damaged. As a side note, having the NFC antenna right over the microcontroller is not good design for intra-circuit EMI. Although there haven't been issues with it so far, it will need to be addressed in the next prototype – the tag doesn't even have to be in the vicinity of the microcontroller, other than for stable I2C communication.

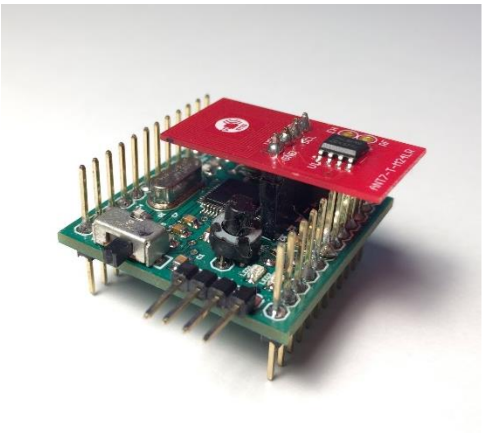


Figure 21: NFC memory tag position on the DAPhNe MCU

4.7.2 Reprogramming the NFC Memory

The M24LR04 is a great NFC solution to passive NFC communication. It uses energy harvesting to power itself when an external controller communicates with it. This means that the chip's memory can be read/written without the M24LR04 chip being explicitly powered – it uses energy from the RF signals coming from the transceiver.

The transceiver in this application, for now, is the CR95HF Demo board by STMicroelectronics. To use it, you will need a USB-B-to-USB-A type connector wire (<u>not included in the passed-off hardware – you will need to find your own</u>). Connect the USB-B male connector to the board, and the USB-A connector to your PC. This powers the CR95HF board; when the board's antenna is brought close to the NFC tag, the board will flash a red LED, signifying that it is sensing an NFC tag in the vicinity.

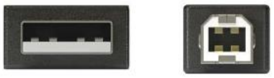


Figure 22: USB-A (left), USB-B (right)

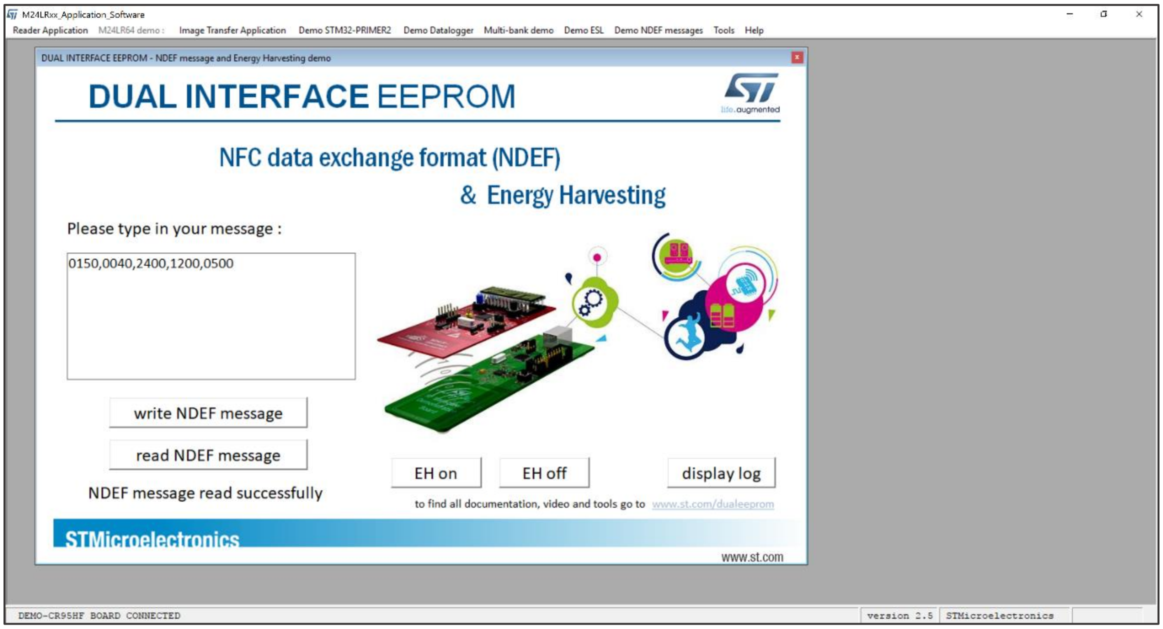
4.7.3 Using NFC Demo Software

When the CR95HF board is connected to PC, you can use it to reprogram the NFC memory tag using demo software from STMicroelectronics (see the software checklist above).

To read data from the memory,

1. Open M24LRxx Application Software
2. From the drop-down menu, select CR95HF DEMO KIT
3. Click OK
4. Select *Demo NDEF Messages* → *show Demo NDEF and Energy Harvesting*
5. Put the CR95HF Board's antenna in the immediate vicinity of the memory tag
6. Click *read NDEF message* button.

The text field should show something similar to Figure 23. To write to the memory, simply change some of the read values, and click *write NDEF message* button.



Please type in your message :

0150,0040,2400,1200,0500

NDEF message read successfully

DEMO-CR95HF BOARD CONNECTED

Figure 23: Read Operation Result

4.7.4 DAPhNe Data Protocol

At this stage of development, no specific reliable protocol has been developed. So far, the transmitted data is just a single string, containing five 4-digit comma-separated values. For proper functionality of the DAPhNe stimulator, each value must consist of 4 digits (if the value is “15”, the representation is 0015), and the values must be comma-separated. Upon programming, the device assumes new settings on the very next breathing cycle.

The five values are summarized in Table 2.

Table 2: DAPhNe Data Protocol

	Value	**Unit**
1	Pulse Width	µs
2	Pulse Magnitude	0.01 mA
3	Breathing Rate	0.01 BPM
4	Inspiratory Time	0.01 s
5	Interpulse Interval*	µs

* The period of time between the negative and positive pulses. This value is not used in current firmware – it is hardcoded to 3 times the pulse width

Pulse magnitude, breathing rate, and inspiratory time values are defined such that the values can have 0.01 precision in the firmware. So, a 24.56 BPM breathing rate will be represented as 2456, and current of 5.89 mA will be represented as 0589, etc.

As a side note, the device has a hardcoded “minimum” that it assumes if the tag is not there at the beginning of the operation. Although hot-plugging is not recommended, the tag may be plugged-in at run time, and the MCU will assume the values stored in the tag. If the tag is taken out in the middle of operation, the device continues with the last known mode settings.