

User Guide

Re-programming PICO, a Single Use Negative Pressure Wound Therapy Device

Introduction

Smith & Nephew's PICO Negative Pressure Wound Therapy Device is an extremely useful tool for skin and deep wound regeneration. It is small and portable, consumes very little power, and has an extremely simple interface.

However, it is also a "single use" device; it can only be used for about a week of battery power in total, and then the device turns off on its own. While this is a great contamination precaution, this makes the device very expensive for every-day hospital and laboratory use. It is also programmed to achieve only a certain negative pressure range, which is not ideal for special situations.



This document provides information on how to modify a commercial PICO device such that it:

- ❖ Can be used indefinitely
- ❖ Is able to achieve a variety of different pressure ranges on demand
- ❖ Can be configured to operate on a variety of time scales
- ❖ Can be easily reprogrammed time after time using a simple debugging device

PICO uses the *STM8L151G4* microcontroller made by STMicroelectronics. Re-programming this device is done with a free IDE software, a standard peripheral library provided by ST, and an inexpensive debugging device. The device itself is physically modified for easier reprogramming, which allows for an easy plug-and-program interface not available with the original PICO device.

Included with this document are:

- ❖ Data Sheet for the STM8L15xx family of microcontrollers
- ❖ Reference Manual for the STM8L15xx family of microcontrollers
- ❖ ST-Link/V2 User Manual (Debugging Device)
- ❖ Application Note for the Omron 2SMPP-02 Pressure Sensor used by PICO
- ❖ Full Map of PICO Circuit Board
- ❖ STM8 standard peripheral library
- ❖ Proposed program used to reverse-engineer PICO.

1. Comparing Original and Re-programmed PICO

The original commercial PICO has certain operational sequences. When the device is first turned on, it operates the motor on DC and tries to achieve its negative pressure threshold of about -12.5 kPa. If it succeeds, it lets go until the pressure rises to about -9 kPa and then turns the motor back on. When operating normally, it flashes the green LED; if the battery is low – it turns on the appropriate red LED.

If it does not reach the low pressure threshold in 30 seconds, it first waits for 15 seconds. Then it turns on again, and if there is still a leak in the gauze, it turns off for an hour, flashing the “Leak” LED. The device turns back on either after an hour of waiting, or if the user presses the button. In both cases, it resets all of its variables and starts the cycle anew. If the device malfunctions and somehow starts pumping uncontrollably, PICO’s circuit automatically shuts off motor power supply when the pressure reaches about -22 kPa.

The re-programmed PICO has similar functionalities; in some cases the program extends the original capabilities, and in some cases, there are drawbacks. The following table summarizes and compares the two programs.

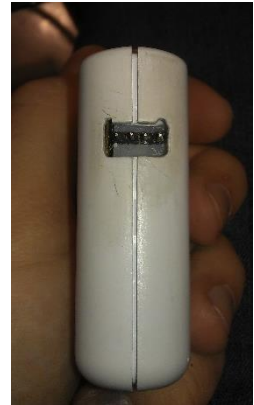
Function	Original PICO	Programmed PICO
Low Pressure Threshold	~ -12.5 kPa	Up to ~ -22 kPa
High Pressure Threshold	~ -9 kPa	Up to ~ -22 kPa
Automatic Shutoff Pressure	~ -22 kPa	~ -22 kPa unless the sensor resistance is changed
Trying to achieve LPT for...	30 s	programmable
Waiting after first failure for...	15 s	programmable
Waiting after second failure for...	1 hour	programmable
LEDs flash every...	2s	Default 2s; programmable
LEDs flash for...	~20ms	Default ~20ms; programmable
Error state	All LEDs flash	All LEDs flash (errors in User Controls)
Motor supply function (1 st try)	DC	DC
Motor supply function (otherwise)	PWM	DC
ADC Reading, Motor On	?(fast response)	Every 1ms
ADC Reading, Motor Off - every	1s	1s

2. Physical Modifications

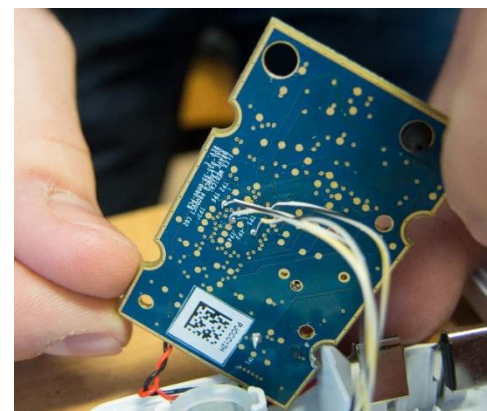
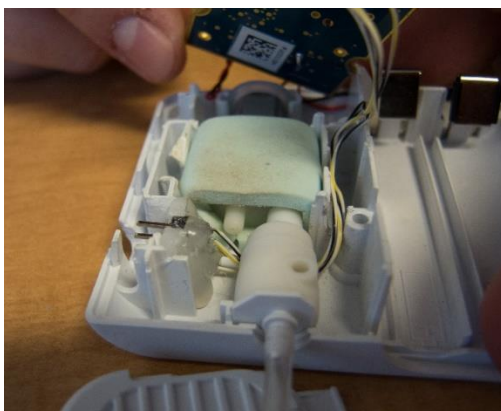
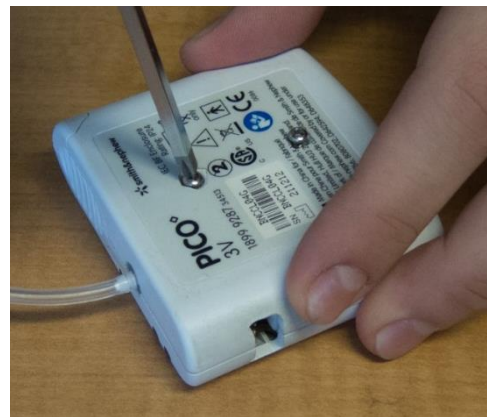
PICO's circuit board has explicit connections to the microcontroller's communication pins. The MCU can be reprogrammed using SWIM communication protocol (see *Section 5*), which operates via four pins: Supply Voltage, Data, Reset, and Ground (see *Map of PICO Circuit Board*).

To fit the debugger's generic four pin connector, the casing of the device has been modified to include a connector jack. Note that the jack is directional.

Inside the device, the four pin male connector secured between the walls of the casing by a super-adhesive is soldered to 28 gauge wires that, in turn, are soldered to the appropriate connections on the circuit board.



If a pressure threshold of less than about -22 kPa is desired, a resistor must be soldered across the V_{out+} and V_{out-} pins of the pressure sensor (see *2SMPP-02 Pressure Sensor* for pin positions). This fools the PICO's automatic shutoff circuit into accepting pressures lower than normal. The code variables must be changed accordingly (see *Section 8*; *stm8l15x_conf.h* in *PICO Program\Project*).



3. Software and Hardware

ST-Link/V2

The PICO Negative Pressure Wound Therapy device uses *STM8L151G4*, a ultra-low power microcontroller from STMicroelectronics. This MCU can be reprogrammed using a debugger device called *ST-Link/V2*, provided by ST. The USB driver for it and a firmware upgrade are available on the ST website.



ST-LINK/V2 in-circuit debugger/programmer kit:

http://www.st.com/web/catalog/tools/FM146/CL1984/SC720/SS1450/PF251168?s_searchtype=partnumber#

Driver for ST-LINK/V2 in-circuit debugger/programmer for STM8:

<http://www.st.com/web/en/catalog/tools/PF260219#>

(Optional) Firmware Upgrade for ST-LINK/V2 in-circuit debugger/programmer for STM8:

<http://www.st.com/web/en/catalog/tools/PF258194#>

IDE and Standard Peripheral Library

There are at least two IDEs usually used with this microcontroller: *STVD* (developed by ST), and *IAR Embedded Workbench for STM8*. The latter was used for reprogramming of the device. IAR's free trial minimal software (no time limit) is sufficient for the purposes of this project.

IAR Embedded Workbench for STM8:

<https://www.iar.com/iar-embedded-workbench/#!?architecture=STM8>

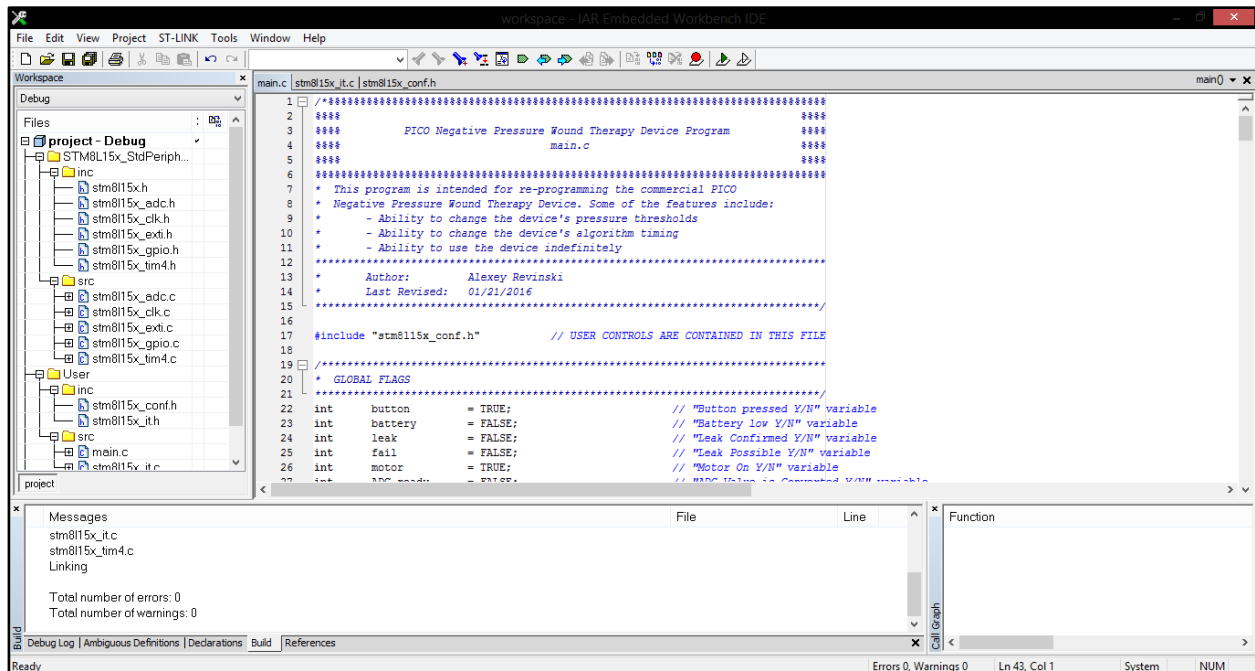
Along with installing the IAR software, the PICO program needs a peripheral function library to compile and download the program into the MCU. The library is available on the ST website, and a copy of it was also included in the folder *PICO Program*. The link is provided for reference.

(Included in project folder) STM8L15x/16x/05x/AL3Lx/AL31x standard peripheral library:

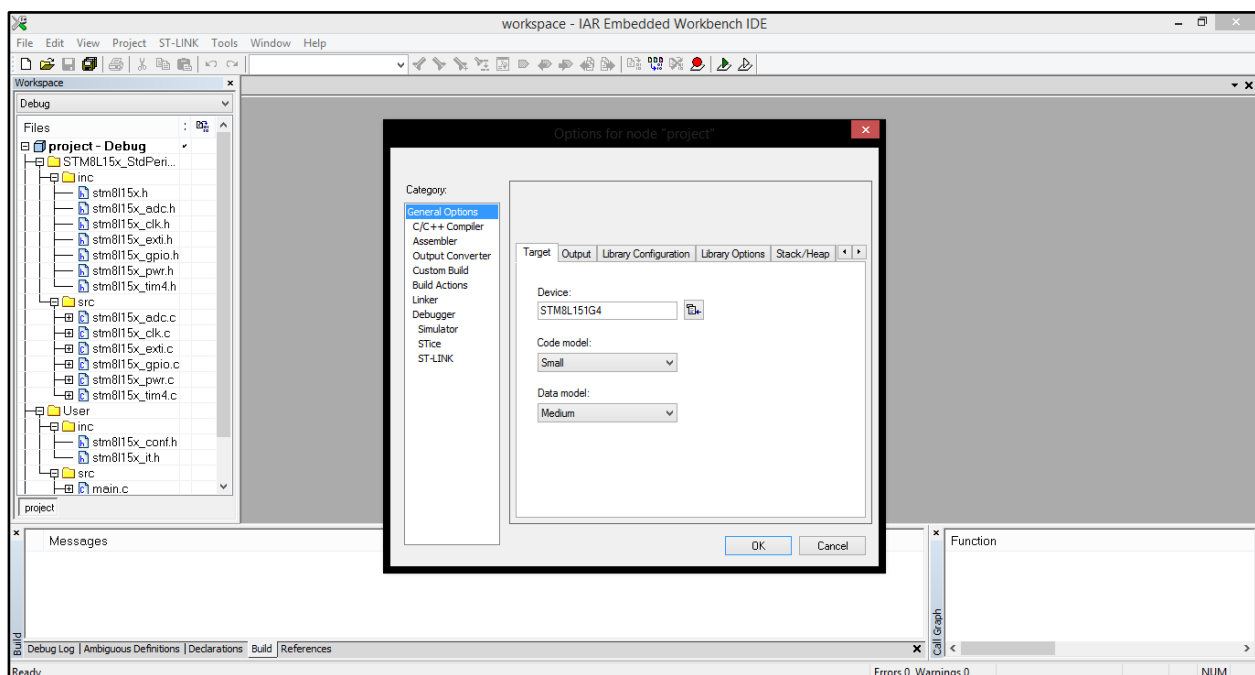
<http://www.st.com/web/en/catalog/tools/PF257956#>

4. Configuring IAR Embedded Workbench

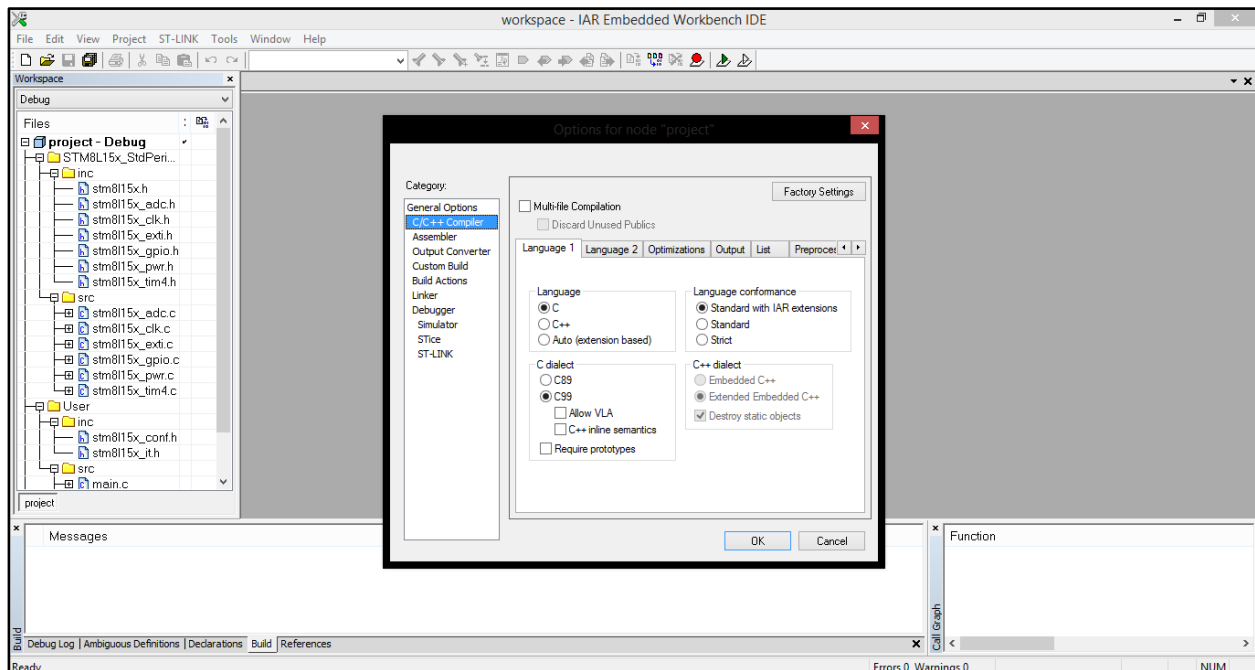
Once IAR Embedded Workbench has been installed, open the IAR software. On the top left, select *File>Open>Workspace...*; then, go to ...*PICO Program\Project\EWSTM8* and open the file “*workspace.eww*”. The following workspace should appear:



Next, select *Project>Options>General Options*. In the “Device” field, choose STM8L151G4.

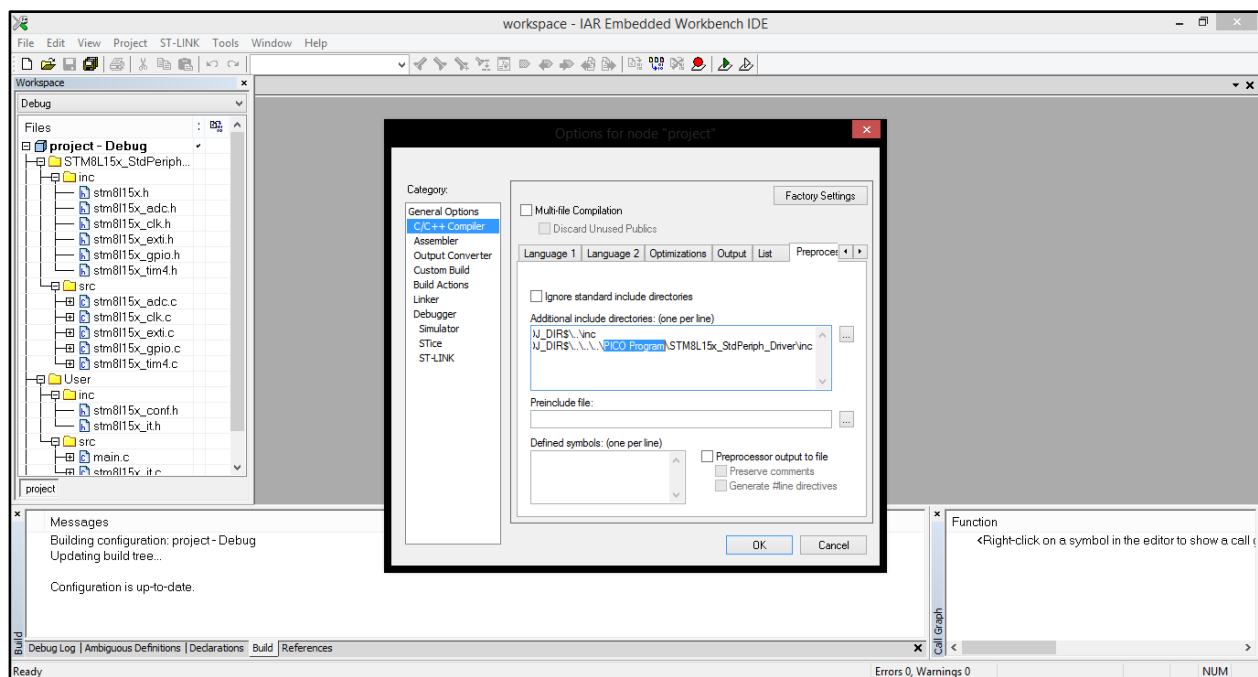


Select *C/C++ Compiler*. Parameters below should be set by default; if not, change them as shown.

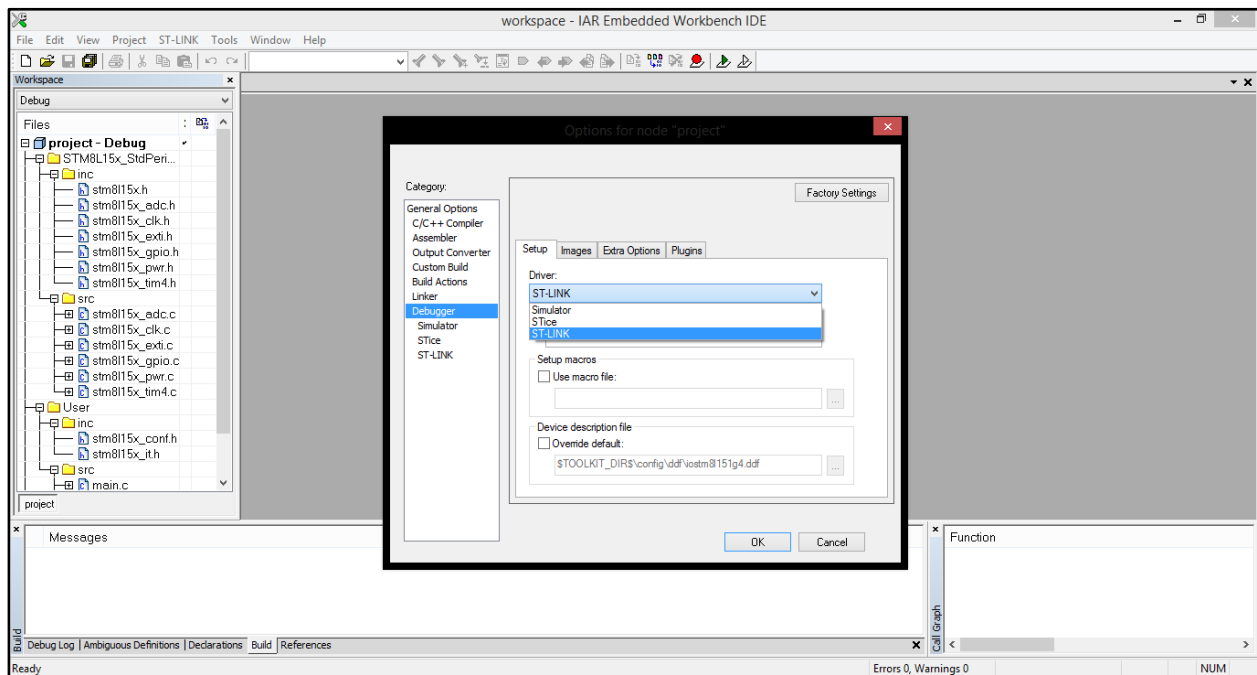


In the same category, select the *Preprocessor* tab. The project uses a standard peripheral library provided by STMicroelectronics. The second directory should be as shown below. If you rename the folder *PICO Program*, your directory should be:

`$PROJ_DIR$\\..\\..\\{NEW FOLDER}\\STM8L15x_StdPeriph_Driver\\inc`



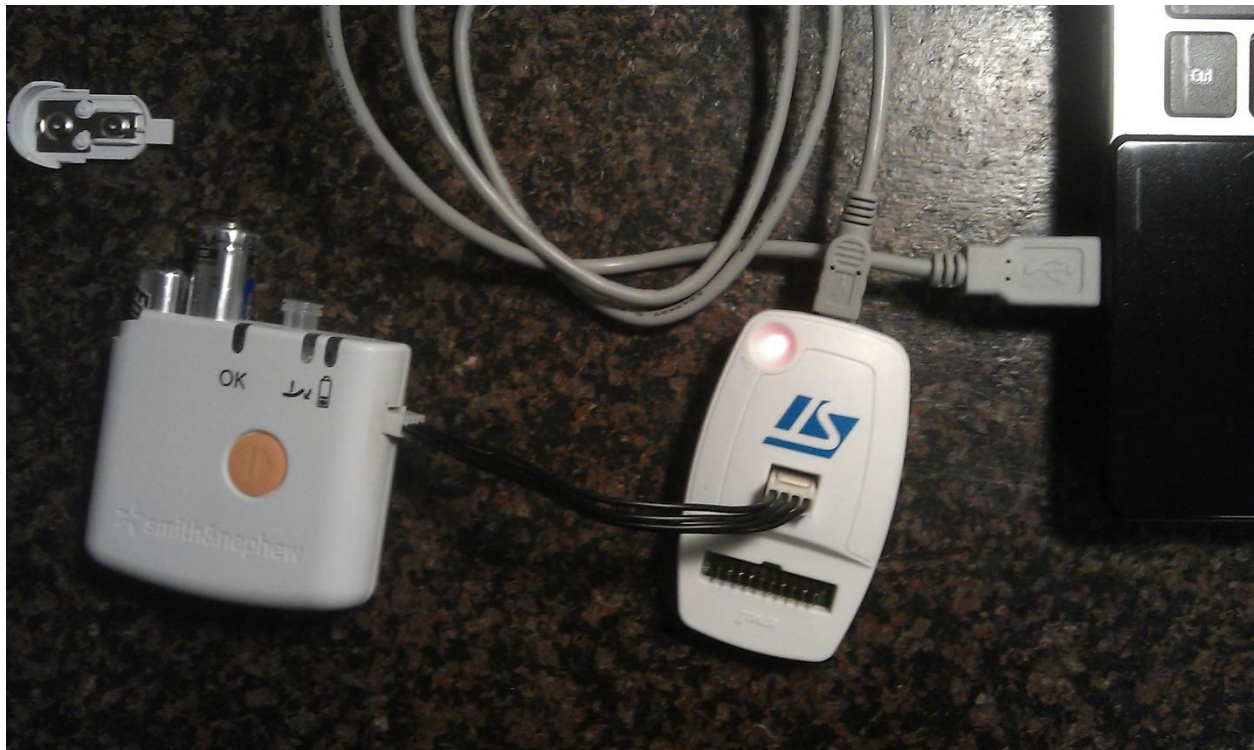
Finally, select the category *Debugger*, and select ST-LINK as the driver. Otherwise, the software will be writing the program to a virtual simulator. Click OK. The workspace is ready to go!



5. Setting up PICO and ST-Link/V2

The debugger uses SWIM communication protocol to talk to the microcontroller and interacts with a computer using USB 2.0/3.0. To get PICO ready for re-programming, **insert two AA batteries** into the device and make appropriate connections as shown. The generic four pin connectors (between PICO and *ST-Link/V2*) used by the SWIM communication are directional, so make sure that they are inserted correctly.

The red LED on *ST-Link/V2* should turn on, and PICO is ready to accept a new program.



6. Getting to Know IAR Embedded Workbench

Open the file “workspace.eww” in ... \PICO Program\Project\EWSTM8.

The workspace in editing mode consists of four fields: *Text Editor*, *Files*, *Messages*, and *Function*. Files can be viewed in the text editor by double clicking them in the *Files* field. Pressing F7 will trigger the making process; after a successful build, *Messages* field should report that there were no errors or warnings, as shown in the figure below. Debugging breakpoints can be set in the text editor field by clicking on the left margin of the field next to line numbers.

The top bar contains the buttons:



Make (F7)

- Compile, Assemble, and Link project files



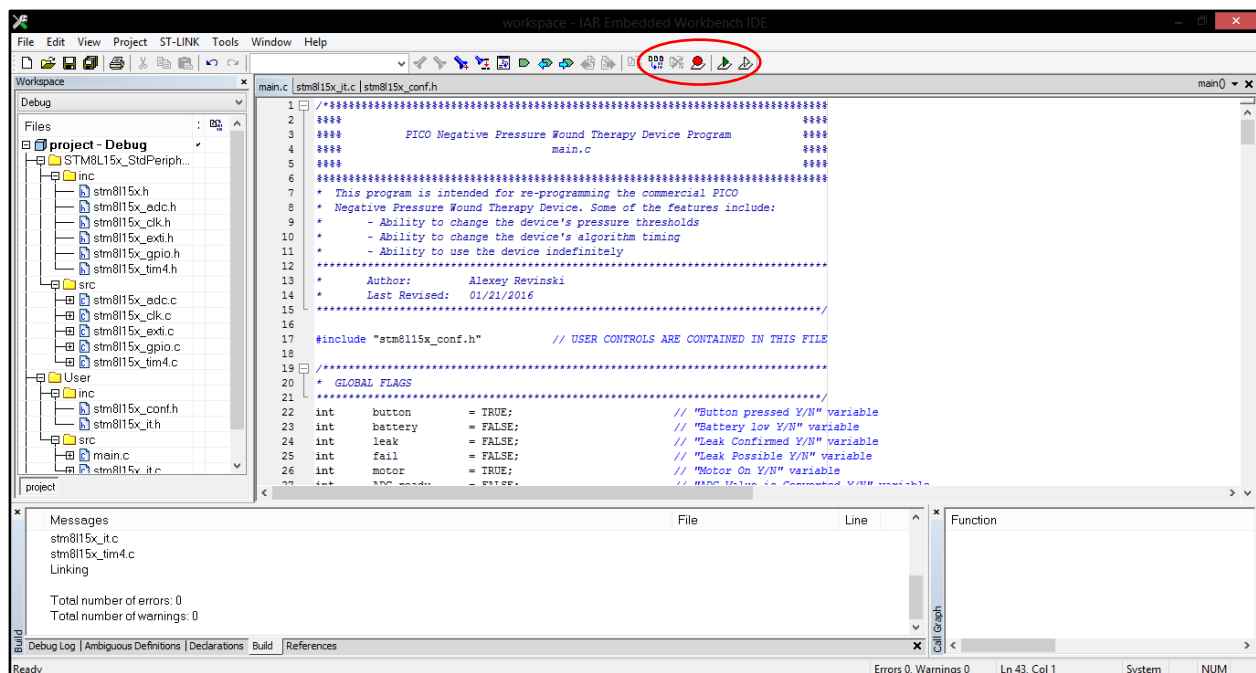
Debug and Download

- Compile, Assemble, Link, Download, and starts Debugging Mode



Debug w/o Download

- Compile, Assemble, Link, Download; remains in Editing Mode



As it is, the program should be able to fully compile and download into PICO’s microcontroller. However, before clicking *Make*, make sure that all of the files shown above are present in the file tree. If not, right-click on the folder that they should be contained in, click *Add > Add Files...* and select the appropriate files in the respective folder in *PICO Program\{FOLDER}*.


Connect PICO to the USB port (see *Section 5*), making sure that the device is powered by two AA batteries. *ST-Link/V2* debugger should have a red LED on.

The device is ready to accept the program!

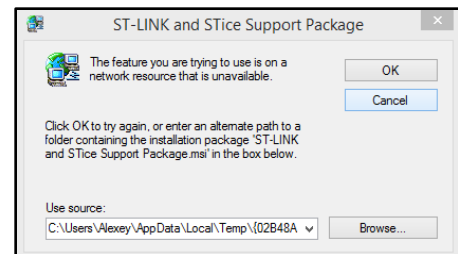
7. Downloading the Program


NOTE:

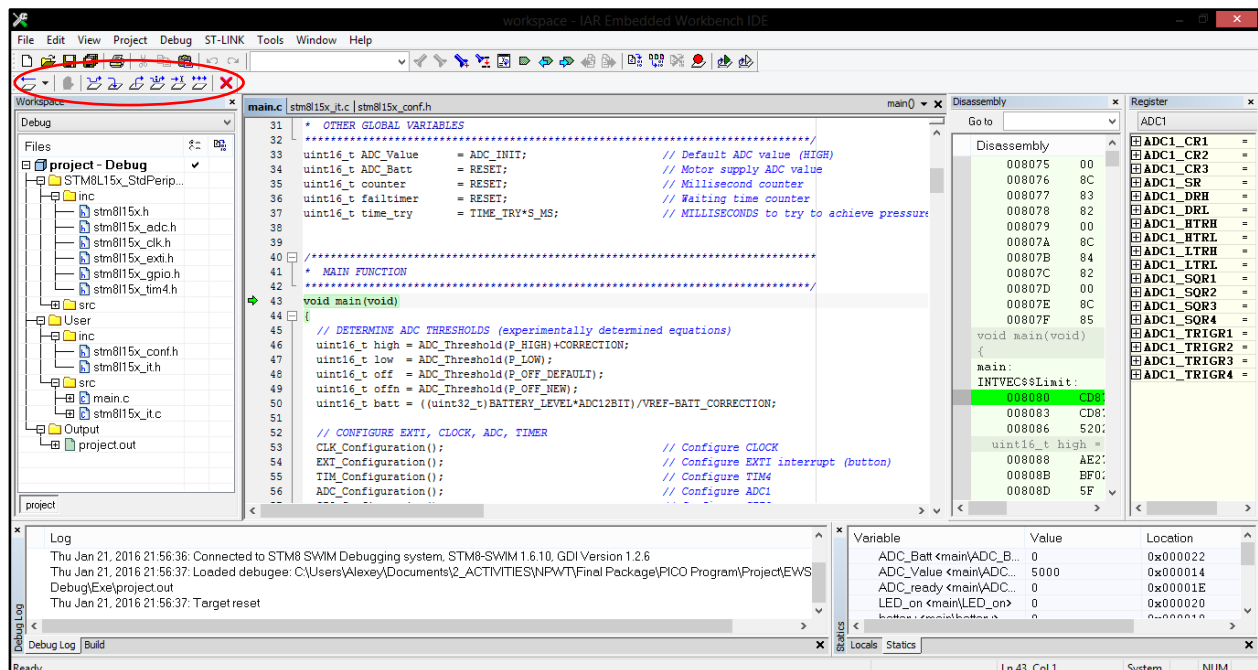
Because the *PICO Program* folder was most likely moved into a new directory and the project scripts use previous unknown directories, the project first needs to be purged of any output files. To perform the operation, select **Project>Clean**. Now, as the program will be re-built, new paths will be taken into account.

Click  to build the executables and download the program into PICO's MCU. This will start the debugging mode.

When you download the program for the first time since the beginning of the IAR session, you may get an error message. The *ST-Link/V2* software is often trying to download an unnecessary (and unavailable...) support package for *ST-Link/V2* that the device does not need. Click Cancel, and then OK. The program will enter debugging mode.



The new toolbar that appears on the left contains debugging step tools (see below). Clicking  will run the program until a breakpoint. In absence of breakpoints, the program can be physically tested real-time.



To exit, click . The program has been downloaded, and PICO is now ready for user testing.

8. Understanding the Program Files

The project consists of two user-defined .c files, one .h file, and a group of included standard peripheral function files. The three user defined/modified files are:

- ❖ *stm8l15x_conf.h* - Preprocessor commands, instructions, user control variables
- ❖ *stm8l15x_it.c* - Interrupt service routines
- ❖ *main.c* - Main function and helper functions

User Control File: **stm8l15x_conf.h**

This is the file from which the behavior of PICO is controlled. First, it contains a section on which pins of the STM8L151G4 microcontroller the device uses and indicates their default functions. For detailed descriptions of each function, please reference the *STM8L151xx Datasheet* file included in the *PICO Program* folder. Document named *Map of PICO Circuit Board* contains information on spatial arrangement of the pins.

PICO has certain default pressure thresholds and time-dependend operation sequences (see *Section 1*). These are provided in the *stm8l15x_conf.h* file for convenience.

Default PICO pressure thresholds are followed by the *Instructions* section. PICO can be controlled with 8 variables that are defined as constants at compile-time. This means the program needs to be re-built and re-downloaded each time these “User Control” variables are modified.

These variables are:

- ❖ `PRESSURE_UNITS` - Selects units of pressure used for other variables
- ❖ `P_HIGH` - Selects the high pressure threshold
- ❖ `P_LOW` - Selects the low pressure threshold
- ❖ `RESISTOR_ADDED` - Modifies calculations if pressure sensor resistance is changed
- ❖ `TIME_TRY` - If PICO cannot achieve the low pressure threshold within this time period, the gauze may have an air leak
- ❖ `TIME_FAIL` - The time PICO waits after first try before turning back on
- ❖ `TIME_LEAK` - The time PICO waits after a leak has been determined
- ❖ `BATTERY_LEVEL` - Motor power supply voltage below which the device cannot work properly (cannot achieve low pressure threshold).

NOTE: Only modify the variable `RESISTOR_ADDED` if the pressure resistance has indeed been changed. Otherwise, the device will not target the correct pressure thresholds and may cause unexpected damage to the patient’s tissues.

Next are experimental constants, time constants, macros, and other variables that the user should generally not change unless an alternative device functionality is needed.

ISR File: stm8l15x_it.c

This is a re-programmed version of the ST-provided Interrupt Service Routine library file. The new PICO program only uses two interrupts: a button interrupt, and a timer interrupt.

The button ISR is used to de-bounce the button and change a software flag indicating that the button has been pressed. The timer-based ISR is used to turn on and off PICO's LEDs based on what the device is doing (normal operation vs. leak vs. battery is low vs. erroneous program variables).

Main Function File: main.c

This file contains various software flags, time counters, the main function, and the helper functions. These functions are:

- ❖ Pressure_Units - Modifies equation constants based on units used
- ❖ ADC_Threshold - Calculates ADC equivalent of the pressure thresholds
- ❖ GIO_Configuration - Configures all used and unused input/output pins
- ❖ ADC_Configuration - Configures ADC resolution, pins, and capture mode
- ❖ CLK_Configuration - Configures system and peripheral clocks
- ❖ EXT_Configuration - Configures external interrupt source (button pin)
- ❖ TIM_Configuration - Configures timer frequency and mode
- ❖ wait_s - Enables the device to enter waiting mode as the leak is identified

First, the main function calculates the ADC equivalents of the user-identified pressure and battery level thresholds. Then it configures the device's I/O, ADC, Clocks, interrupts, and the timer and enters an infinite loop.

There, it first checks if the User Control constants contain appropriate values. If not, the device enters halt mode and flashes all three LEDs.

If the device has been programmed well, it enters automatic halt mode and waits for the user to turn on the device. When the button has been pressed, CPU initializes appropriate variables and enters the regular sequence of operations: achieve pressure, let go, wait if pressure not achieved, etc.

When the motor of the device is on, it constantly checks the pressure levels. Once the negative pressure threshold has been achieved, the device turns on ADC and pressure sensor circuitry only every second for reduction of power consumption.

Each second, it also checks for battery level. Once the battery gets to a dangerously low level, the device flashes the "Battery Low" LED.

For information on physical functionality of the program, refer to *Section 1*.