

## ESE381 Embedded Microprocessor Systems Design II

Spring 2022 - K. Short  
revised January 15, 2022 10:02 am

### LABORATORY 01: Using the AVR128DB48 Curiosity Nano Microcontroller Board and Microchip Studio 7 for Firmware Development in C

To be performed the week beginning January 30th during your assigned laboratory section.

#### Prerequisite Reading and Microchip Studio Link

1. *AVR128DB48 Curiosity Nano Hardware User Guide*, Sections 1, 2 and 3 (pages 1 through 7) and Section 5 (pages 26 through 29). This document is available on Blackboard.
2. *Scope Fundamentals for EE Students*. (Blackboard: Course Documents > Oscilloscope Information). Read this if you need a review on using the oscilloscope.
3. *Microchip Studio 7 Download Link*. For this course you need Microchip Studio 7's IDE (Formerly Atmel Studio 7) on your personal computer. This is the same software used in ESE280, so you probably already have it. In the various documentation from Microchip you will see two different IDEs referenced, MPLAB X IDE and Microchip Studio. Their overall functionality is similar. We will only be using Microchip Studio in this course. You will also see references to MPLAB XC Compilers. These are cross compilers that can also be used with MPLAB X IDE, which we will not be using. The following link can be used to download Studio7.

<https://www.microchip.com/en-us/development-tools-tools-and-software/microchip-studio-for-avr-and-sam-devices>

Scroll down a bit on the page the link takes you to and you see the installer options. If you are using an internet connection, the first one should work for you.

#### Getting Started with Microchip Studio

Our Getting Started with Microchip Studio online training will guide you through all the major features of the IDE. This training is divided into sections, each of which includes an overview video and accompanying hands-on activities.

[Go to Microchip Studio Training](#)

[Downloads](#) [Documentation](#)

Windows (x86/x64)

Title	Date	Download
<a href="#">Microchip Studio for AVR and SAM Devices 7.0.2542 Web Installer</a>	01 Nov 2020	<a href="#">Download</a>

## Overview

Embedded system design involves both hardware and software design. The purpose of this laboratory is to provide you an initial experience in entering and compiling some simple C programs in Microchip Studio 7 and executing them on the AVR128DB48 Curiosity Nano microcontroller board. The AVR128DB48 is the target microcontroller used throughout this course. This laboratory assumes that you are already familiar with Atmel Studio 7 from ESE280. Microchip Studio 7 is a later and rebranded version of Atmel Studio 7. You need this newer version to support the AVR128DB48 microcontroller. The programs that you will be creating are C programs, not assembler, as in ESE280.

Another purpose of this laboratory is to refresh your knowledge on how to use an oscilloscope to measure periodic signals. These signals are generated at a pin of the microcontroller by running the example programs. These example programs introduce you to some basic hardware and C software concepts. These concepts are covered in detail in later lectures.

The purpose of the examples to be implemented this week are simply to generate an output that drives an LED under various scenarios.

## Design Tasks

In future laboratories, this section will consist of a set of design tasks that you have to perform prior to your scheduled laboratory section. These tasks typically require you to design some hardware that interfaces with the microcontroller and write C code that operates with that hardware. To accomplish this, you need to use OrCAD for schematic capture and Studio 7 for program development. When developing software as part of a design task, you emulate the software to verify that it is correct. This is done outside of and prior to your scheduled laboratory section.

This laboratory has only one simple design task, Task 2. You will be required to create a schematic and code for this task as your prelab assignment. However, for this particular laboratory you will not submit your schematic and code before your laboratory session as an actual prelab. Instead you will submit it at the end of your laboratory session.

You must download Studio 7 to your computer and compile Laboratory Task 1, prior to your scheduled laboratory section.

## Laboratory Tasks

Each laboratory task typically requires that you first build any required hardware that interfaces to the Curiosity microcontroller board for that task. Then you enter the program that you developed and verified, prior to your laboratory section, and download it to the microcontroller. You then run the program on the microcontroller and verify that it operates correctly with the hardware. When you have verified that your system works correctly, you must have a TA evaluate your system's performance. The TA will assign you a score for the task.

**Note: Do not connect the Curiosity board to the PC before powering ON the PC. Some PCs may try to boot from the Curiosity board's drive and may end up not booting.**

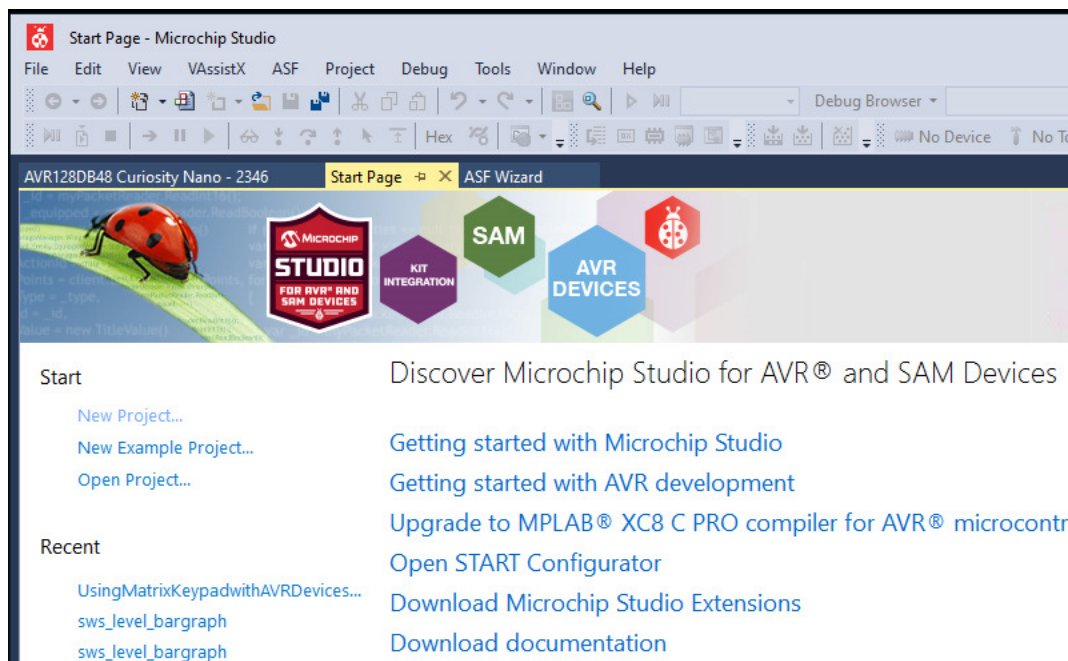
***Laboratory Task 1: Entering, Compiling, and Emulating the Program SW0\_controls\_LED0.***

For this task, you create a project and enter the code for a program named SW0\_controls\_LED0. You then assemble and simulate this code.

On the PC desktop, click on the Microchip Studio 7 icon to launch Studio 7.

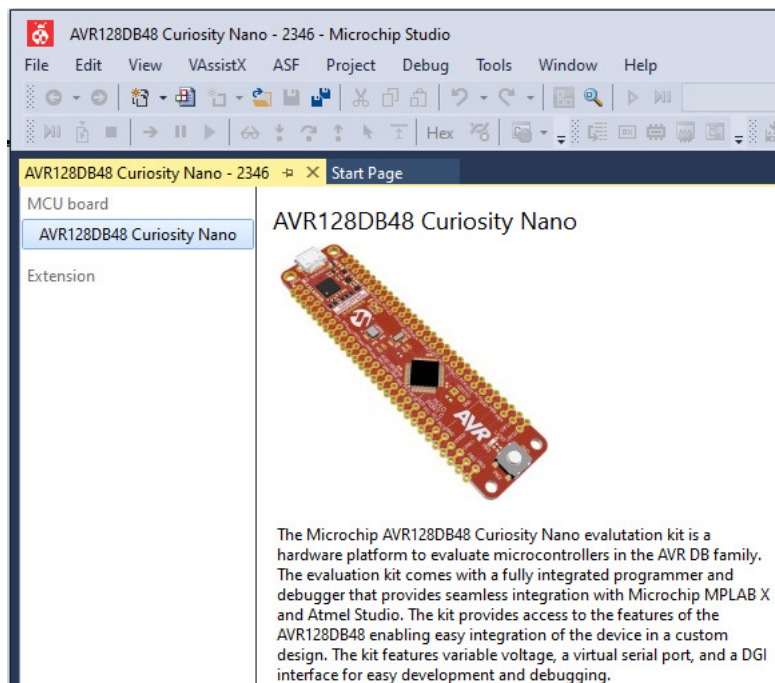


Studio 7's start page appears.



If you already have the Curiosity Nano plugged in the your PC's USB port, a different page will

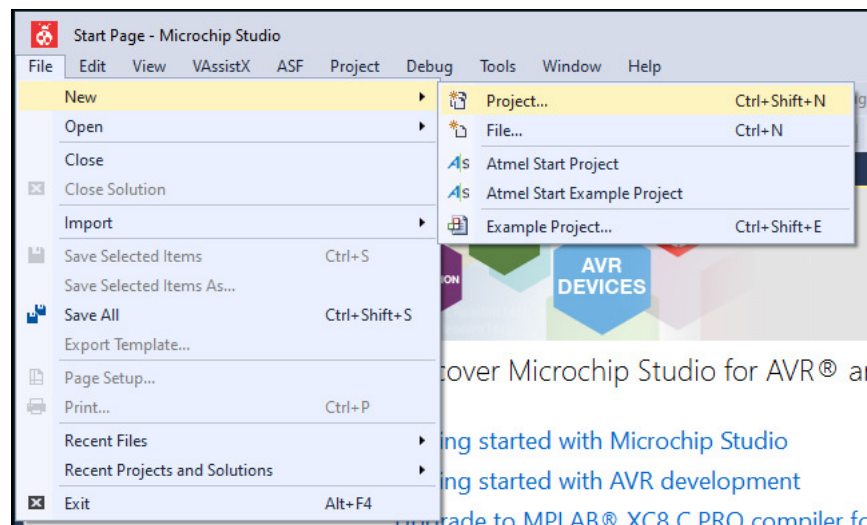
appear with the AVR128DB48 Curiosity Nano shown.



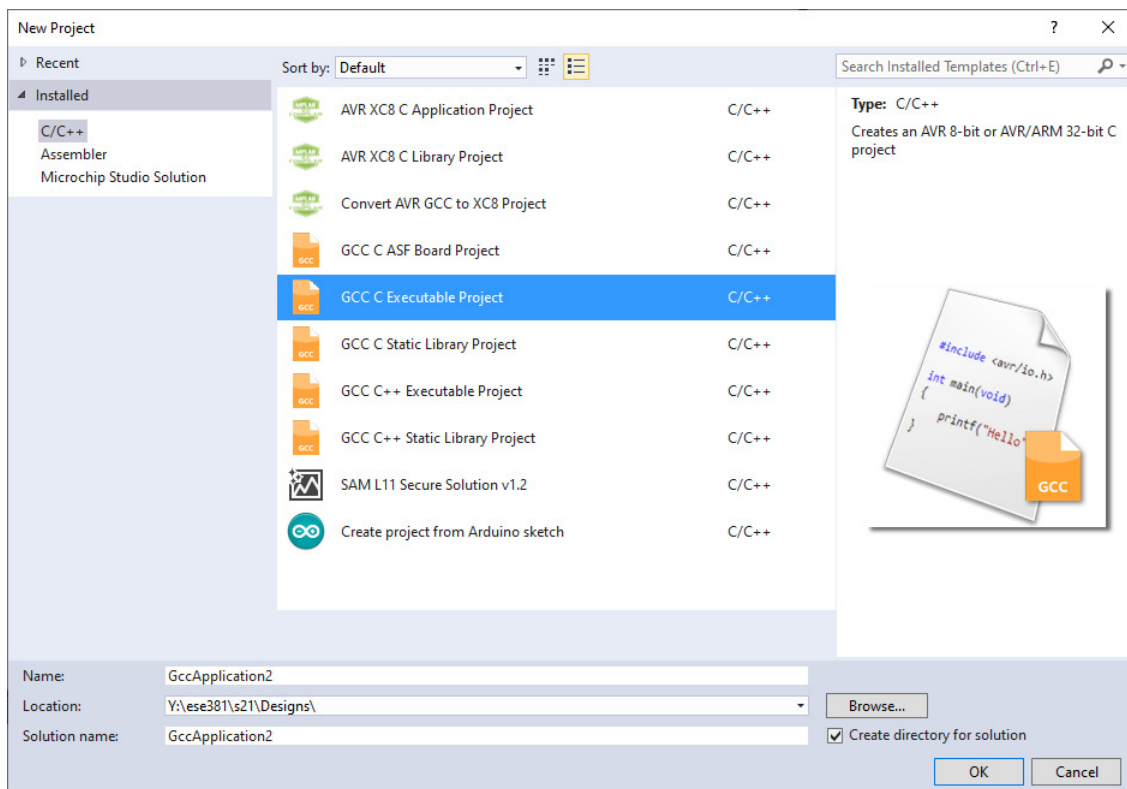
Just click the Start Page tab to switch to the Start page.

If Studio opens with a project already loaded, go to the File menu and select Close Solution.

Next, click File > New > Project.

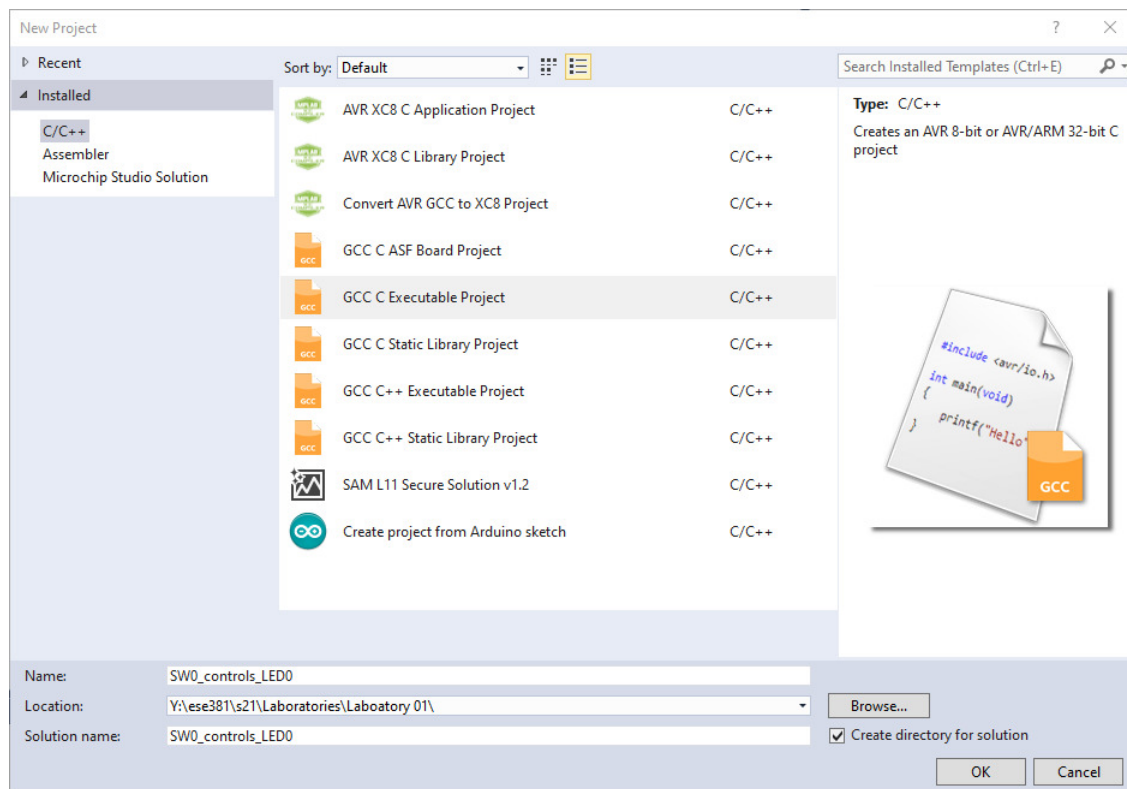


The Project Wizard opens. We want to create a GCC C Executable Project.

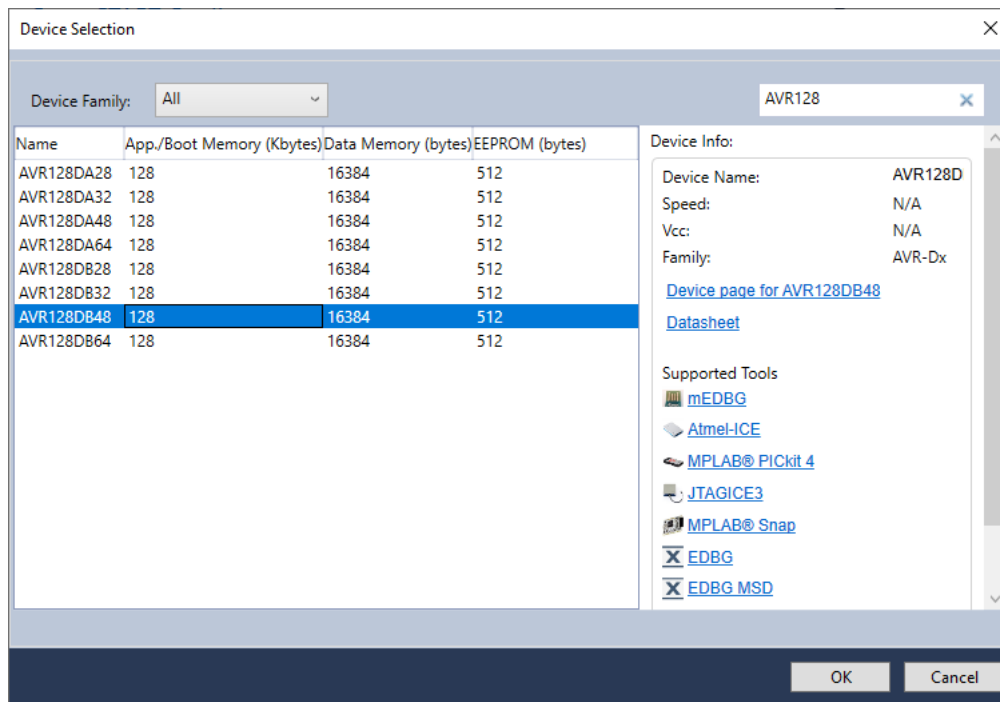


Under Installed (on the left side), select C/C++. In the middle portion of the window select GCC C Executable Project. In the Name textbox type SW0\_controls\_LED0. For Location, browse to the folder on your thumb drive where you want this project to reside. Make sure the check box,

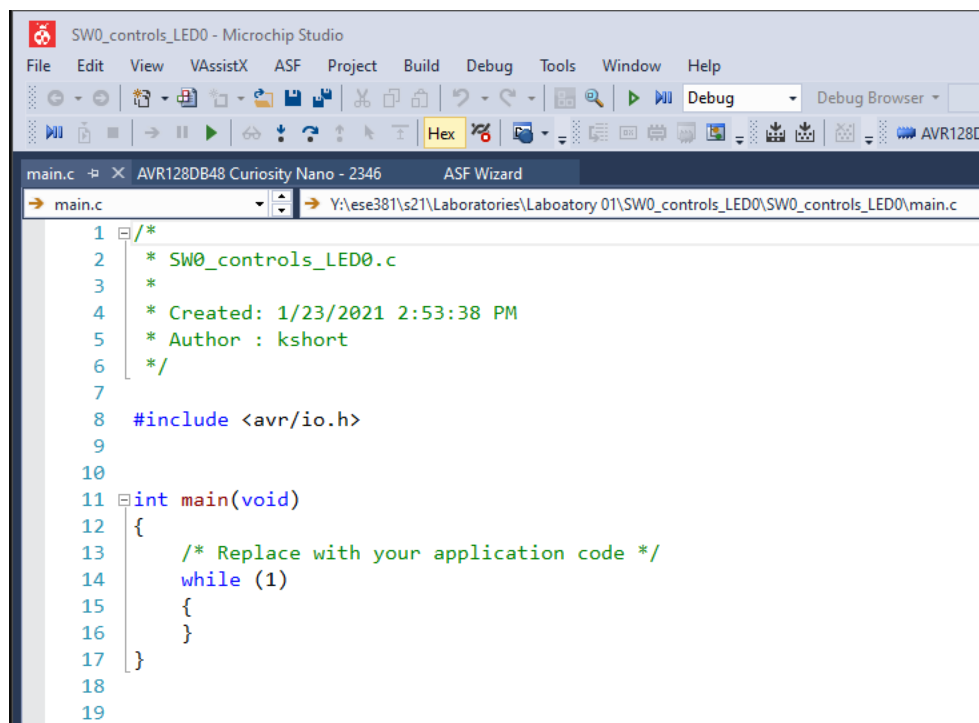
on the bottom right, for **Create directory for solution** is checked. Click OK.



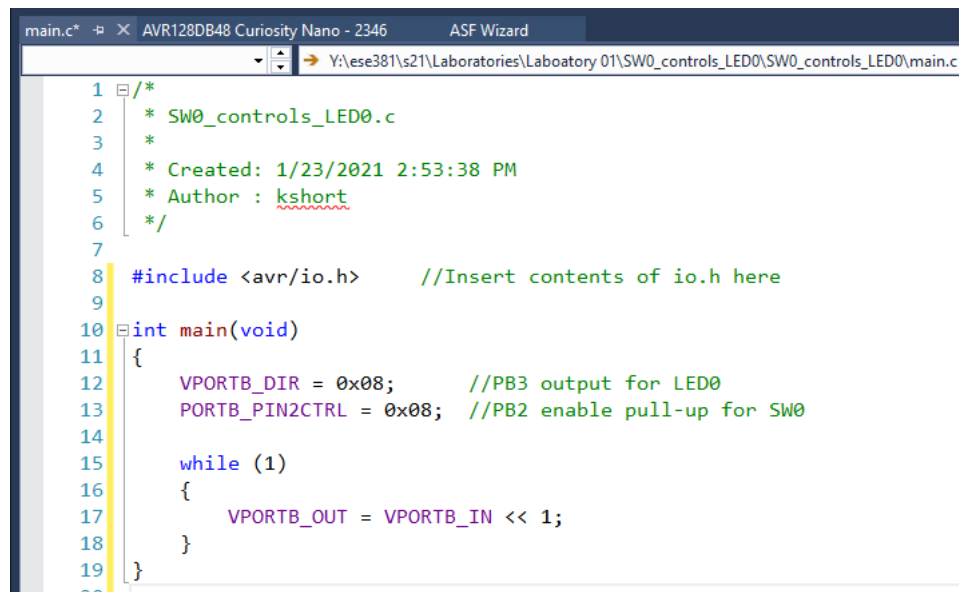
The Device Selection dialog opens. For Device Family, select AVR128 or type AVR128 in the search box on the type right. This narrows down the devices to select from to those in this family. Select AVR128DB48, then click OK.



The Project window opens with the source code editor open. A comment block appears in the source code along with a simple program template.



Type in the code for the program.

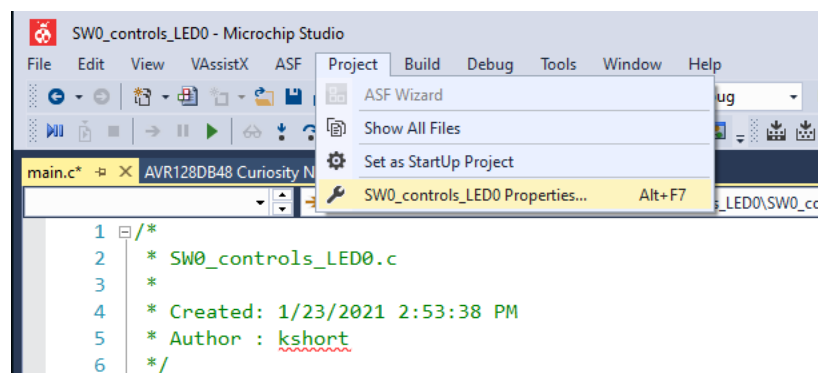


```
1  /*
2  * SW0_controls_LED0.c
3  *
4  * Created: 1/23/2021 2:53:38 PM
5  * Author : kshort
6  */
7
8  #include <avr/io.h>    //Insert contents of io.h here
9
10 int main(void)
11 {
12     VPORTB_DIR = 0x08;    //PB3 output for LED0
13     PORTB_PIN2CTRL = 0x08; //PB2 enable pull-up for SW0
14
15     while (1)
16     {
17         VPORTB_OUT = VPORTB_IN << 1;
18     }
19 }
```

To execute an AVR128DB48 program in Studio 7, you must have an actual AVR128DB48 microcontroller and a hardware debugger. A debugger or in-circuit emulator was traditionally a hardware module that connected the PC to the target microcontroller and its associated physical circuitry. The Curiosity board has logic on the board that, along with logic built into the AVR128DB48, eliminates the need for a separate ICE module.

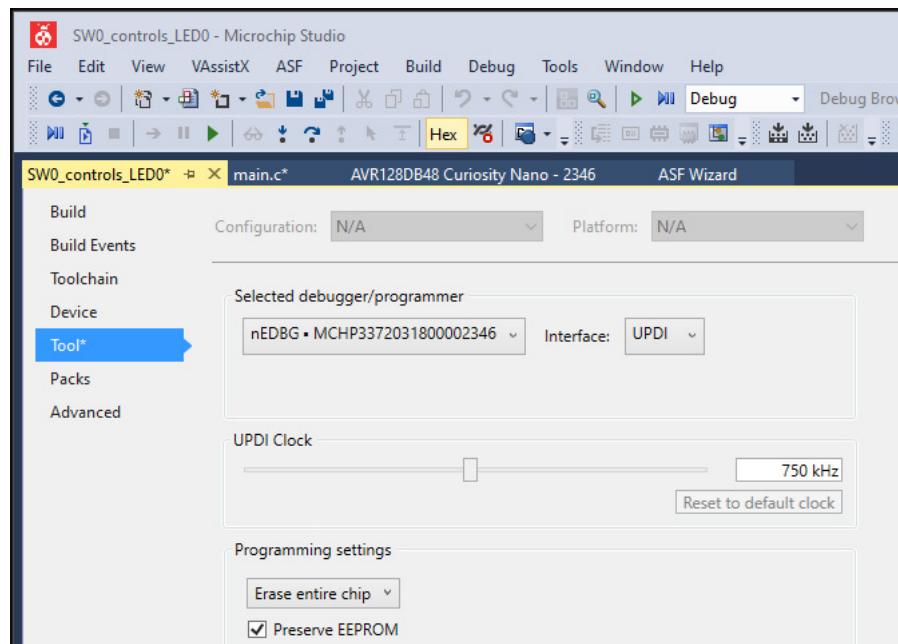
The microcontroller that you are developing your program for is called the target microcontroller. In your case the target microcontroller is the AVR128DB48. The emulator lets you evaluate and debug the operation of a program running on the actual target microcontroller and interacting with any circuitry externally connected to it.

In the menu, go to Project and at the bottom of the drop-down menu and select SW0\_controls\_LED0 Properties.



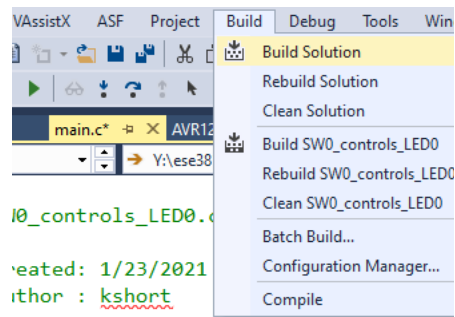
The Project properties window opens. Select the Tool tab on the left. Then open the drop-down selection box under **Selected debugger/programmer**. Select the hardware debugger on the Curiosity Nano. Its identifier will start with nEDBG followed by a serial number.



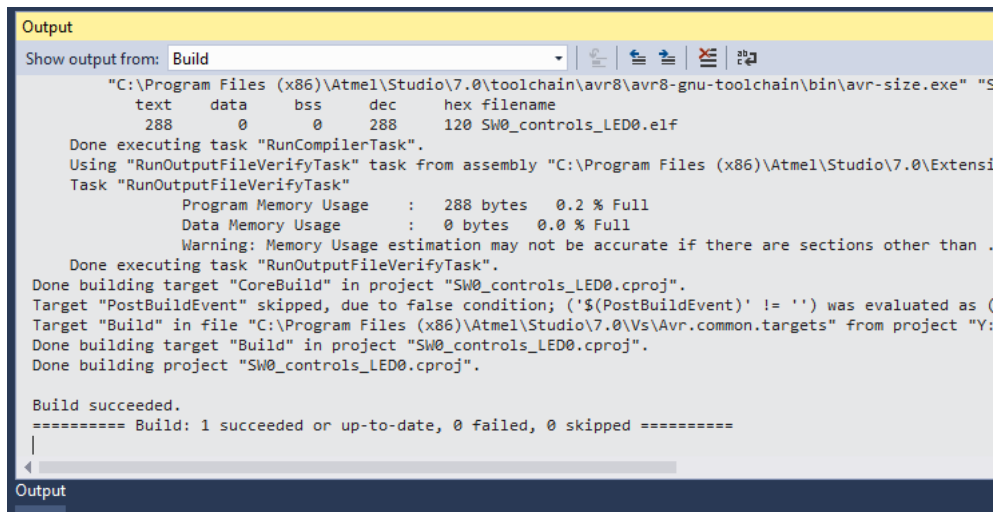


Next click on the main.c tab to select the editor window.

Go to the Build menu and select Build Solutions (or use function key F7).



The program you entered is compiled using the AVR GCC compiler in Studio. The last line in the Output window at the bottom of the screen should indicate that there are no errors or warnings.



```
Output
Show output from: Build

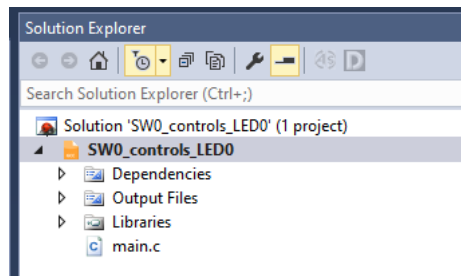
"C:\Program Files (x86)\Atmel\Studio\7.0\toolchain\avr8-gnu-toolchain\bin\avr-size.exe" "S
text    data    bss    dec    hex filename
288      0      0    288    120 SW0_controls_LED0.elf
Done executing task "RunCompilerTask".
Using "RunOutputFileVerifyTask" task from assembly "C:\Program Files (x86)\Atmel\Studio\7.0\Extensi
Task "RunOutputFileVerifyTask"
    Program Memory Usage : 288 bytes 0.2 % Full
    Data Memory Usage   : 0 bytes 0.0 % Full
Warning: Memory Usage estimation may not be accurate if there are sections other than .
Done executing task "RunOutputFileVerifyTask".
Done building target "CoreBuild" in project "SW0_controls_LED0.cproj".
Target "PostBuildEvent" skipped, due to false condition; ('$(PostBuildEvent)' != '') was evaluated as (
Target "Build" in file "C:\Program Files (x86)\Atmel\Studio\7.0\Vs\Avr.common.targets" from project "Y:
Done building target "Build" in project "SW0_controls_LED0.cproj".
Done building project "SW0_controls_LED0.cproj".

Build succeeded.
===== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped =====
|
Output
```

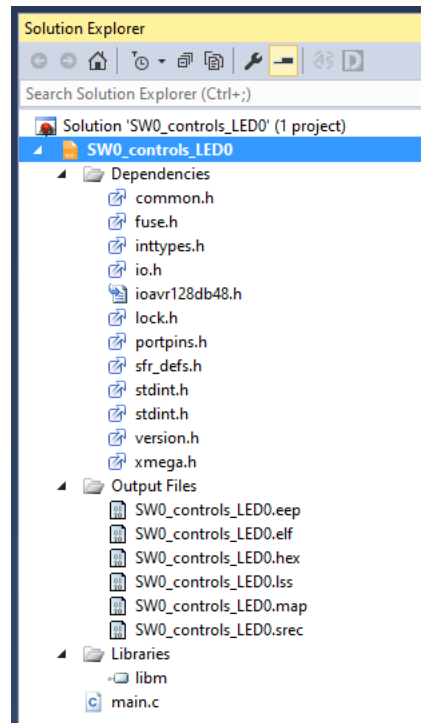
If errors or warnings are displayed, check for typos in your typed-in copy of the program and correct them. After doing so, again select Build Solution from the Build menu to compile the program again.

Note that the Output window also shows how much memory this program required. It required 288 bytes of Flash memory (Program Memory Usage). Note that this task could be done with far fewer bytes in assembly language. Probably less than 10 bytes.

To the right of the source code editor window is the Solution Explorer window.



Expand all of the paths under the project name.



The files listed under Dependencies are the header files required to compile this program. They were included as a result of the `#include <avr/io.h>` statement in the program. Under Output Files are all the files created by the compilation. Under Libraries is the library used.

You can double click on any of the files to examine their contents. Double click on the `SW0_controls_LED0.lss` file.

The later portion of this file contains a listing of the assembly code generated by the compilation. In that portion of the file, the source C statements are listed as comments. Each of these comments is followed by a list of assembly instructions that the C statement, referenced in the comment, was compiled to. The portion of the file that corresponds to the statements in the C source program is

shown below.

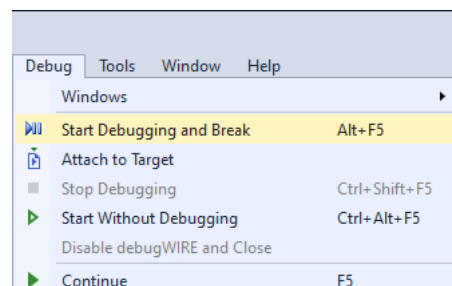
```
107 0000010c <main>:
108
109 #include <avr/io.h>    //Insert contents of io.h here
110
111 int main(void)
112 {
113     VPORTB_DIR = 0x08;    //PB3 output for LED0
114 10c: 88 e0      ldi r24, 0x08    ; 8
115 10e: 84 b9      out 0x04, r24    ; 4
116     PORTB_PIN2CTRL = 0x08; //PB2 enable pull-up for SW0
117 110: 80 93 32 04  sts 0x0432, r24 ; 0x800432 <__TEXT_REGION_LI
118
119     while (1)
120     {
121         VPORTB_OUT = VPORTB_IN << 1;
122 114: 86 b1      in r24, 0x06    ; 6
123 116: 88 0f      add r24, r24
124 118: 85 b9      out 0x05, r24    ; 5
125 11a: fc cf      rjmp  .-8      ; 0x114 <main+0x8>
```

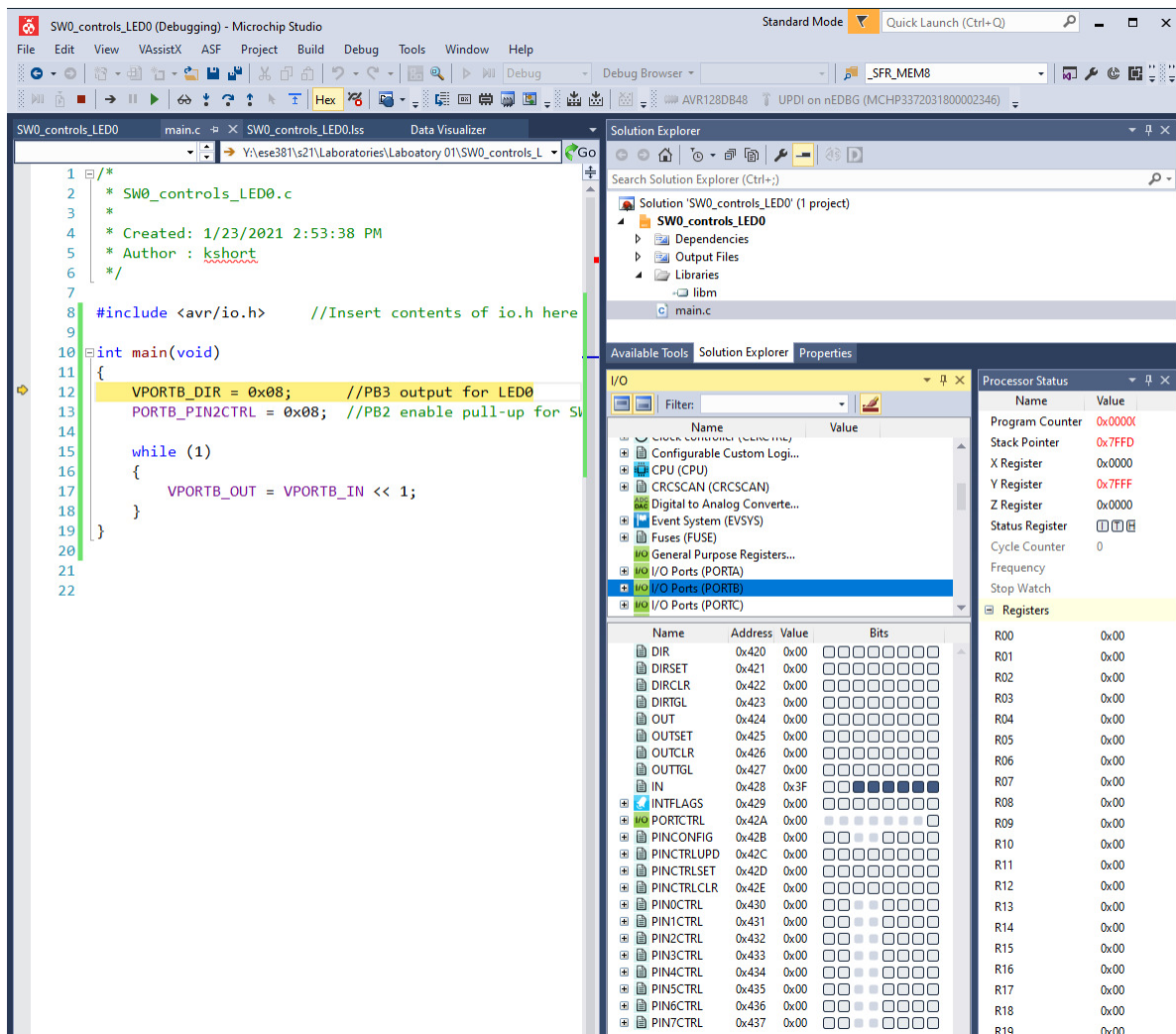
The first column of numbers in this file are the line numbers in the file. The second column in this portion of the file gives the address of the machine code in hexadecimal. The third column gives the actual machine code in hexadecimal. To the right of each line of machine code is the assembly language source code from which the machine code was created.

In the file there assembly instructions before this portion of the file and after it. This is required code that is automatically generated by the compiler to provide the startup for your program code and exit of your program code, if an exit occurs.

**Note: When you are asked to write a program as part of your laboratory, you are required to submit a PDF copy of your source file (the file with the extension .c) as part of your prelab.** If you have an Adobe PDF or CutePDF Writer printer installed on your computer, you can simply select File > Print and choose one of these two as your printer to create a PDF file.

Click on the main.asm tab to display the editor window with your program. Next, go to the Debug menu and select Start Debugging and Break.





A yellow arrow appears next to the first instruction in the program. This arrow indicates the **next** instruction to be executed, which, at this point, is actually the first instruction in the source program.

```

1  /*
2   * SW0_controls_LED0.c
3   *
4   * Created: 1/23/2021 2:53:38 PM
5   * Author : kshort
6   */
7
8  #include <avr/io.h>    //Insert contents of io.h here
9
10 int main(void)
11 {
12     VPORTB_DIR = 0x08;    //PB3 output for LED0
13     PORTB_PIN2CTRL = 0x08; //PB2 enable pull-up for SW0
14
15     while (1)
16     {
17         VPORTB_OUT = VPORTB_IN << 1;
18     }
19 }

```

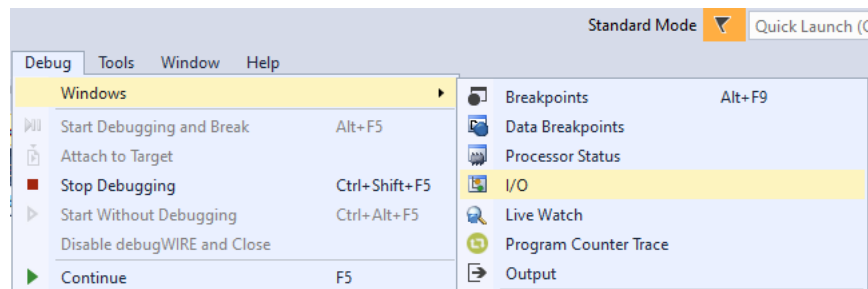
If the Processor Status window is not showing, from the Debug menu select Windows > Processor Status to open the Processor Status window.

Name	Value
Program Counter	0x0000
Stack Pointer	0x7FFD
X Register	0x0000
Y Register	0x7FFF
Z Register	0x0000
Status Register	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
Cycle Counter	0
Frequency	
Stop Watch	
<b>Registers</b>	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00

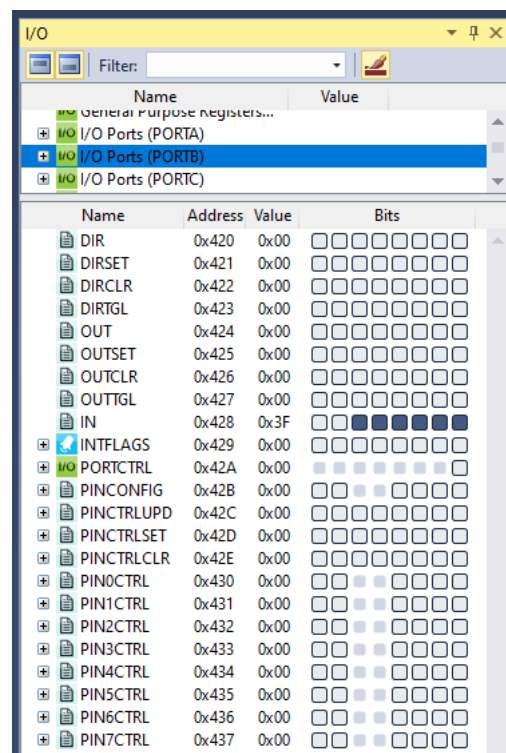
This window shows all the AVR128DB48's CPU registers, including the general purpose registers. It also has three fields that don't correspond to actual AVR128DB48 registers: Cycle Counter, Frequency, and Stop Watch. These are grayed out because they are only usable when you have selected simulator as your debugger, and we do not have that choice for the AVR128DB48.

To see the values of the registers associated with the I/O ports, go to the Debug menu and select

Windows > I/O.



The I/O window opens. Our current program works with the registers associated with PORTB, so you are interested in the values in those registers. Click on PORTB. The bottom portion of the I/O View window displays the bits of the registers associated with PORTB.



Since you have selected PORTB, the registers in the bottom portion of the I/O window are those associated with PORTB. For example, the register DIR is actually PORTB\_DIR, which is the same physical register as VPORTB\_DIR.

You can run and debug your C program now just as you did with your assembly language programs in ESE280. Run this program and verify that it works.

If your program runs correctly, stop debugging and modify your program by commenting out the statement that enables the pull-up resistor on the input pin.

Rerun your program without the pull-up resistor enabled. Does the system operate the same? Note particularly if it takes more time for LED0 to turn OFF when you release the pushbutton.

If there is a difference, we might think that connecting a scope to PB2 (SW0) of the Curiosity board and to ground would provide some information that might explain the difference. Connect your oscilloscope to pin PB2 and to ground. Note that you must configure the oscilloscope to catch the single event of the pushbutton's release.

Does this system without a pull-up enabled on PB2 operate the same with the oscilloscope connected? Does the scope show the change in the signal at PB2 when the pushbutton is released? You will need to answer these questions at the end of the laboratory.

**Have a TA verify that your software and circuit performs as required. Demo the waveforms for PB3 with and without the pull-up enabled. Get the TA's signature.**

### ***Laboratory Task 2: Two Pushbuttons Control an LED***

Two pushbuttons are to be used to control when an LED is turned ON. One pushbutton must be connected to PA2 and the other to PA5. The LED must be connected to PB5. The only external resistor to be used is the current limiting resistor for the LED. The only time the LED is to be ON is when the pushbutton connected to PA2 is not pressed and the pushbutton connected to PA5 is pressed.

Using OrCAD draw a schematic showing the pushbutton circuitry and the LED circuitry and to which pins of the Curiosity board these circuits connect. Do not show a symbol for the Curiosity board.

Using the program that was provided for Task 1 as a starting point, write this program. Name the program `PB1_and_PB2_control_LED`. Only use the DIR, IN, OUT, and PINxCTRL port registers in your program. You are being limited to these registers in this task for a purpose, so you must comply with this restriction.

**Submit your schematic and C source program. Have a TA verify that your circuit performs as required and get the TA's signature.**

### ***Laboratory Task 3: Using Emulation to Execute the C Program `blink_LED0`***

The program that follows blinks LED0. Create a project named `blink_LED0`. Compile and run the program. The program is supposed to blink LED0 at a 1 Hz rate. Make sure you compile this program with the optimization set at -Og. Use the oscilloscope to measure the period of the waveform at PB3 to confirm this.

**Have a TA verify that your software and circuit performs as required. Demo the measurement of the period of the waveform at PB3. Get the TA's signature.**



```

#include <avr/io.h>
#define F_CPU 4000000 //Frequency used by delay macros.
#include <util/delay.h> //Header for delay macros and functions.

int main(void)
{
    PORTB.DIR |= PIN3_bm;

    while (1)
    {
        PORTB.OUT |= PIN3_bm;
        _delay_ms(500);
        PORTB.OUT &= ~PIN3_bm;
        _delay_ms(500);
    }
}

```

