ESE381 Embedded Microprocessor Systems Design II

Spring 2024 - K. Short
revised January 15, 2024 1:11 pm

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 28th 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). (Brightspace: Course Documents > Lab Related Documents).

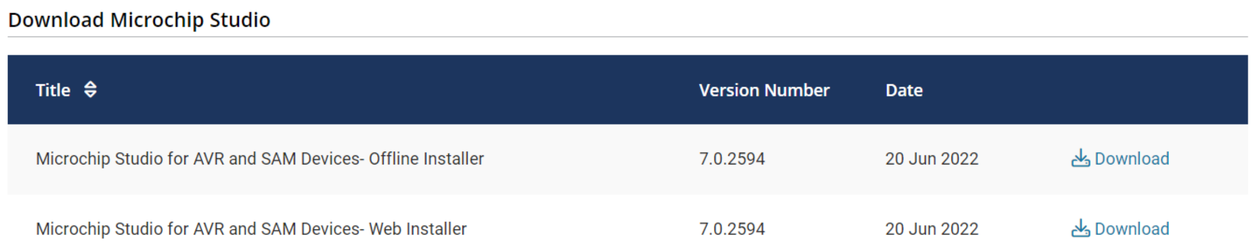
2. *Scope Fundamentals for EE Students.* (Brightspace: Course Documents > Lab Related Documents). Read this if you need a review on using the oscilloscope.

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

Click on the following link:

Or, scroll down a bit on the page and you will see the installer options. If you are using an internet connection, the second installer option should work for you.


Download Microchip Studio

Title	Version Number	Date	
Microchip Studio for AVR and SAM Devices- Offline Installer	7.0.2594	20 Jun 2022	Download
Microchip Studio for AVR and SAM Devices- Web Installer	7.0.2594	20 Jun 2022	Download

Overview

Embedded system design requires 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 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 for embedded systems 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 either simulate or 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.

If you do not have it already installed on your computer, you must download Studio 7 and compile and simulate Laboratory Task 1, prior to your scheduled laboratory section.

Laboratory Tasks

In this course 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.

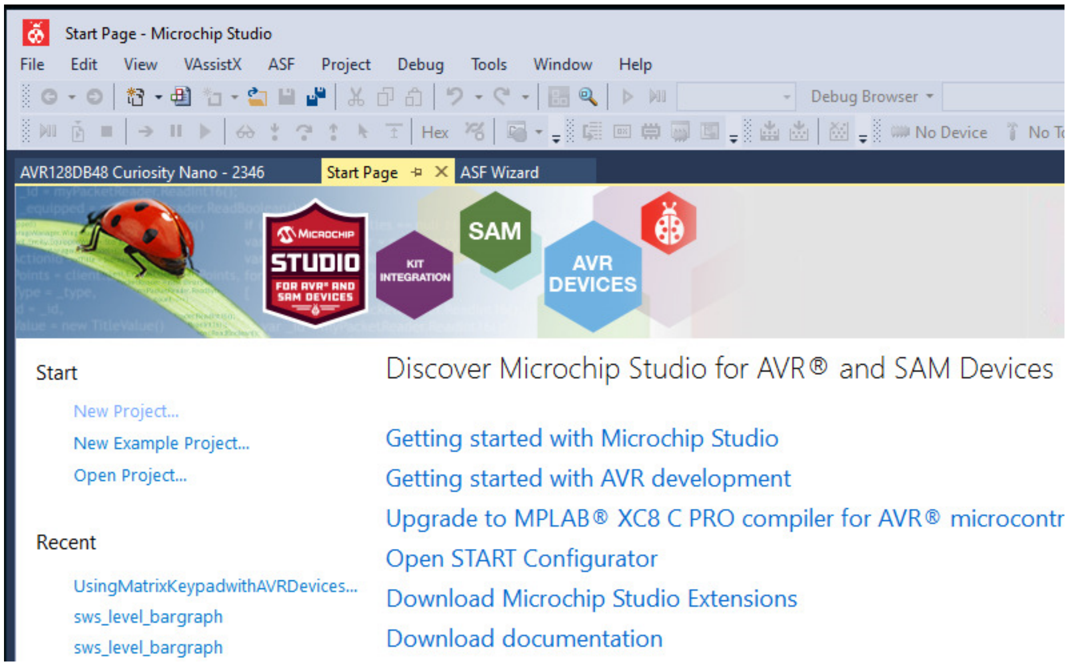
Note: Do not connect the Curiosity board to the PC before powering ON the PC. Some PCs may try to boot from the Curiosity board's drive and may end up not booting.

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 emulate this code.

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

Studio 7's start page appears.


Start Page - Microchip Studio
File Edit View VAssistX ASF Project Debug Tools Window Help
Debug Browser
Hex
No Device
AVR128DB48 Curiosity Nano - 2346
Start Page
ASF Wizard
STUDIO FOR AVR® AND SAM DEVICES
KIT INTEGRATION
SAM
AVR DEVICES

Start

New Project...
New Example Project...
Open Project...

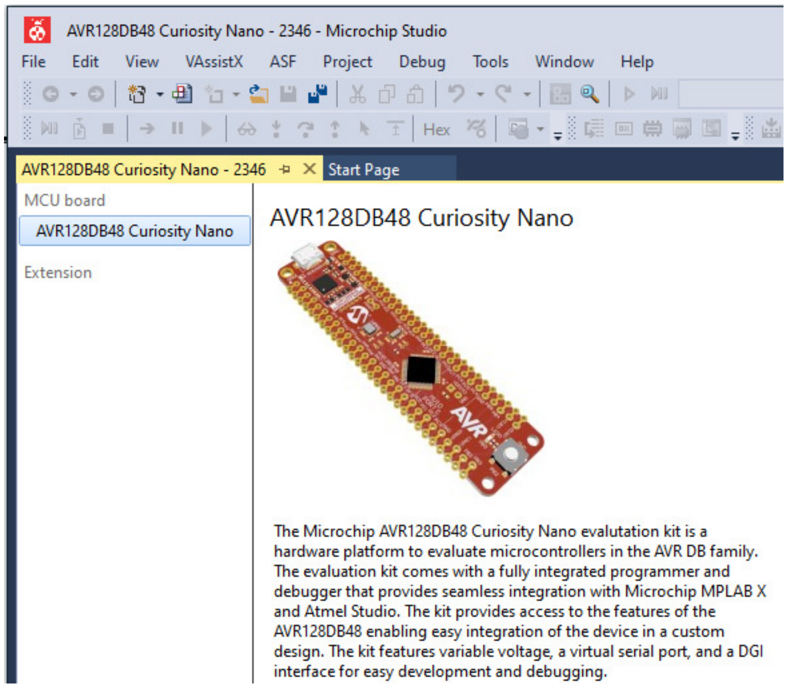
Recent

UsingMatrixKeypadwithAVRDevices...
sws_level_bargraph
sws_level_bargraph

Discover Microchip Studio for AVR® and SAM Devices

Getting started with Microchip Studio
Getting started with AVR development
Upgrade to MPLAB® XC8 C PRO compiler for AVR® microcontr
Open START Configurator
Download Microchip Studio Extensions
Download documentation

If you already have the Curiosity Nano plugged in the your PC's USB port before you click on the Microchip Studio 7 icon, a different page will appear with the AVR128DB48 Curiosity Nano shown.

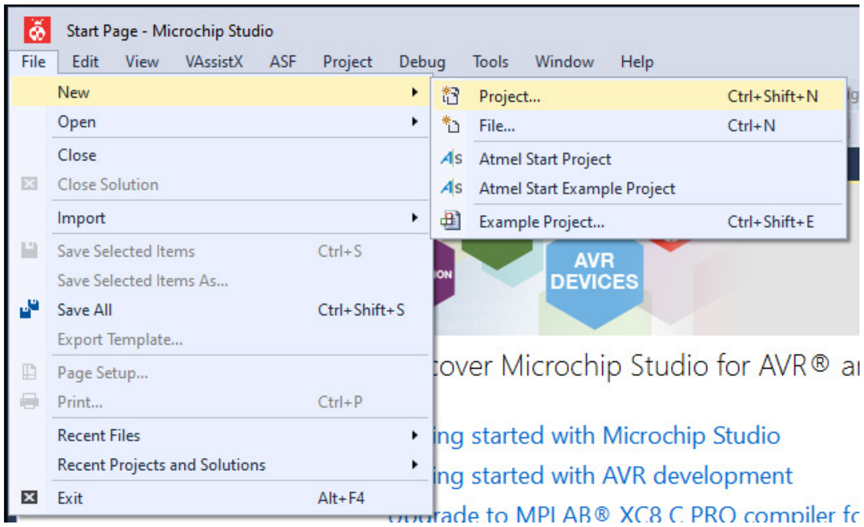

AVR128DB48 Curiosity Nano - 2346 - Microchip Studio
File Edit View VAssistX ASF Project Debug Tools Window Help
AVR128DB48 Curiosity Nano - 2346
Start Page
MCU board
AVR128DB48 Curiosity Nano
Extension
AVR128DB48 Curiosity Nano

The Microchip AVR128DB48 Curiosity Nano evalutation kit is a hardware platform to evaluate microcontrollers in the AVR DB family. The evaluation kit comes with a fully integrated programmer and debugger that provides seamless integration with Microchip MPLAB X and Atmel Studio. The kit provides access to the features of the AVR128DB48 enabling easy integration of the device in a custom design. The kit features variable voltage, a virtual serial port, and a DGI interface for easy development and debugging.

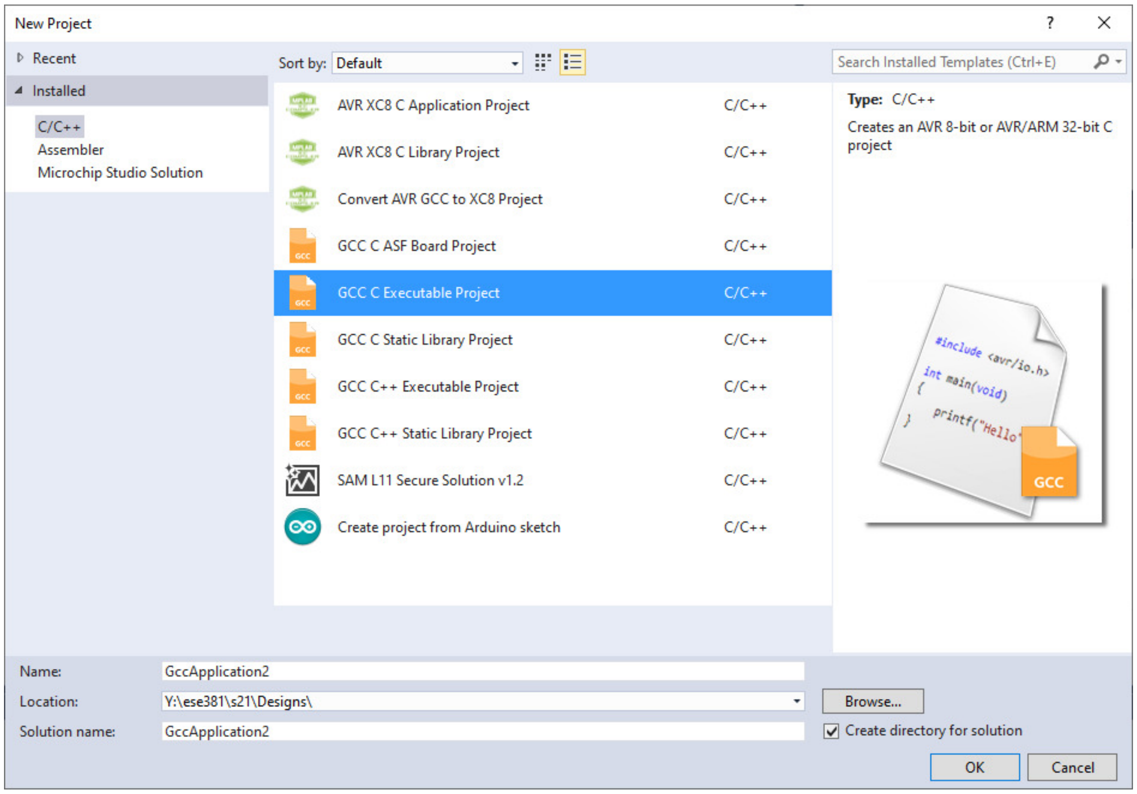
Just click the Start Page tab to switch to the Start page.

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

Next, click File > New > Project.


Start Page - Microchip Studio
File Edit View VAssistX ASF Project Debug Tools Window Help
New
Open
Close
Close Solution
Import
Save Selected Items Ctrl+S
Save Selected Items As...
Save All Ctrl+Shift+S
Export Template...
Page Setup...
Print... Ctrl+P
Recent Files
Recent Projects and Solutions
Exit Alt+F4
Project... Ctrl+Shift+N
File... Ctrl+N
Atmel Start Project
Atmel Start Example Project
Example Project... Ctrl+Shift+E
AVR DEVICES
over Microchip Studio for AVR® a
ing started with Microchip Studio
ing started with AVR development

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



New Project

Sort by: Default

Search Installed Templates (Ctrl+E)

AVR XC8 C Application Project	C/C++
AVR XC8 C Library Project	C/C++
Convert AVR GCC to XC8 Project	C/C++
GCC C ASF Board Project	C/C++
GCC C Executable Project	C/C++
GCC C Static Library Project	C/C++
GCC C++ Executable Project	C/C++
GCC C++ Static Library Project	C/C++
SAM L11 Secure Solution v1.2	C/C++
Create project from Arduino sketch	C/C++

Type: C/C++

Creates an AVR 8-bit or AVR/ARM 32-bit C project

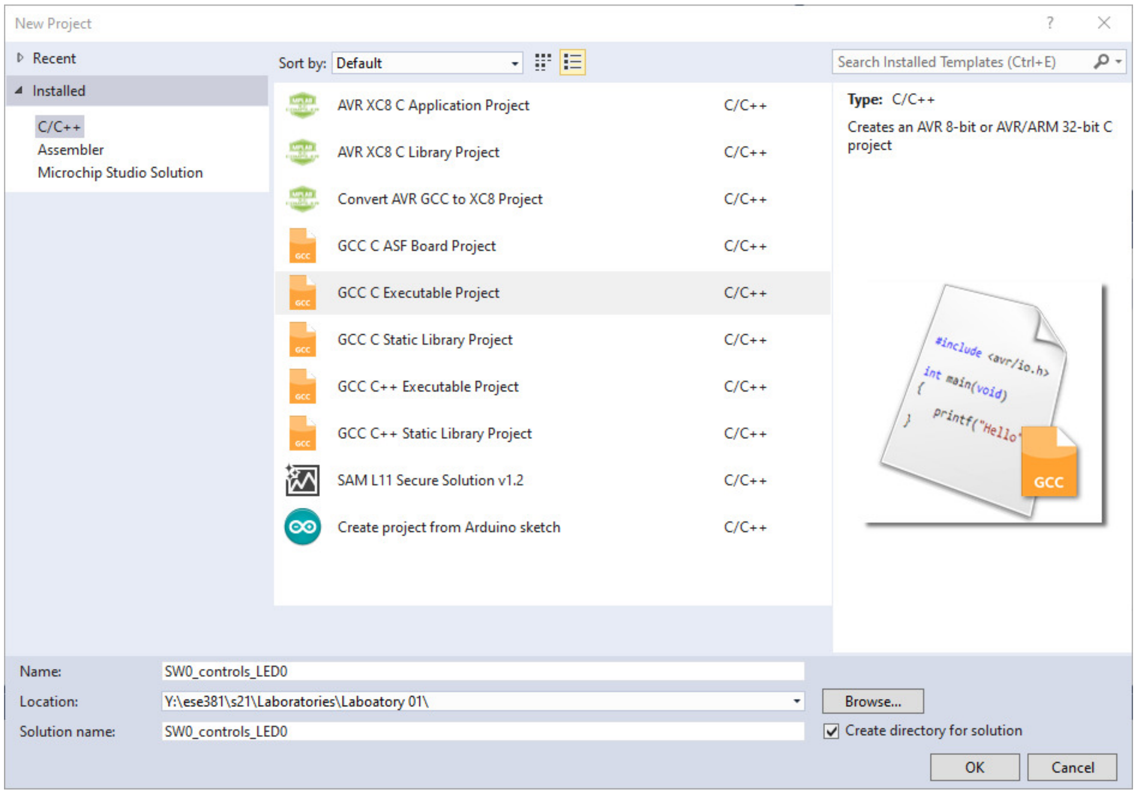
Name: GccApplication2

Location: Y:\ese381\s21\Designs\

Solution name: GccApplication2

☑ Create directory for solution

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.



New Project

Recent

Installed

C/C++

Assembler

Microchip Studio Solution

Sort by: Default

Search Installed Templates (Ctrl+E)

AVR XC8 C Application Project	C/C++
AVR XC8 C Library Project	C/C++
Convert AVR GCC to XC8 Project	C/C++
GCC C ASF Board Project	C/C++
GCC C Executable Project	C/C++
GCC C Static Library Project	C/C++
GCC C++ Executable Project	C/C++
GCC C++ Static Library Project	C/C++
SAM L11 Secure Solution v1.2	C/C++
Create project from Arduino sketch	C/C++

Type: C/C++

Creates an AVR 8-bit or AVR/ARM 32-bit C project

Name: SW0_controls_LED0

Location: Y:\ese381\s21\Laboratories\Laboatory 01\

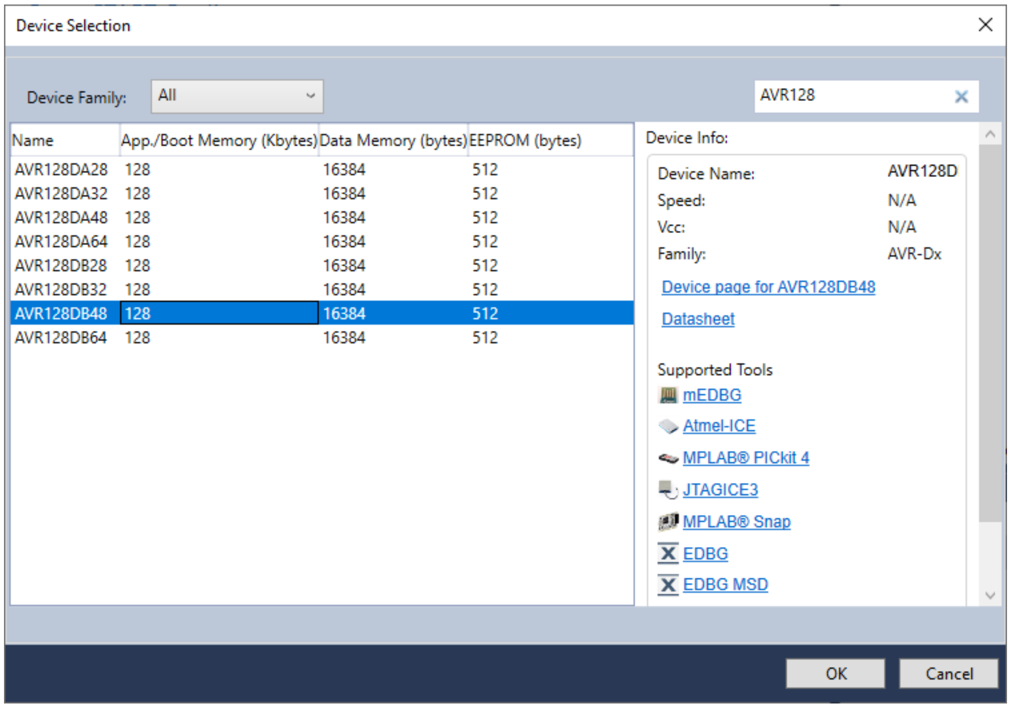
Solution name: SW0_controls_LED0

Browse...

✓ Create directory for solution

OK Cancel

The Device Selection dialog opens. For Device Family, select AVR128 or type AVR128 in the search box on the top right of the dialog. This narrows down the devices to select from to those in this family. Select AVR128DB48, then click OK. Alternatively, you can type AVR128DB48 into the search box and select the single choice that appears and then click OK.



Device Selection

Device Family: All

AVR128

Name	App./Boot Memory (Kbytes)	Data Memory (bytes)	EEPROM (bytes)
AVR128DA28	128	16384	512
AVR128DA32	128	16384	512
AVR128DA48	128	16384	512
AVR128DA64	128	16384	512
AVR128DB28	128	16384	512
AVR128DB32	128	16384	512
AVR128DB48	128	16384	512
AVR128DB64	128	16384	512

Device Info:

Device Name:	AVR128D
Speed:	N/A
Vcc:	N/A
Family:	AVR-Dx

Device page for AVR128DB48

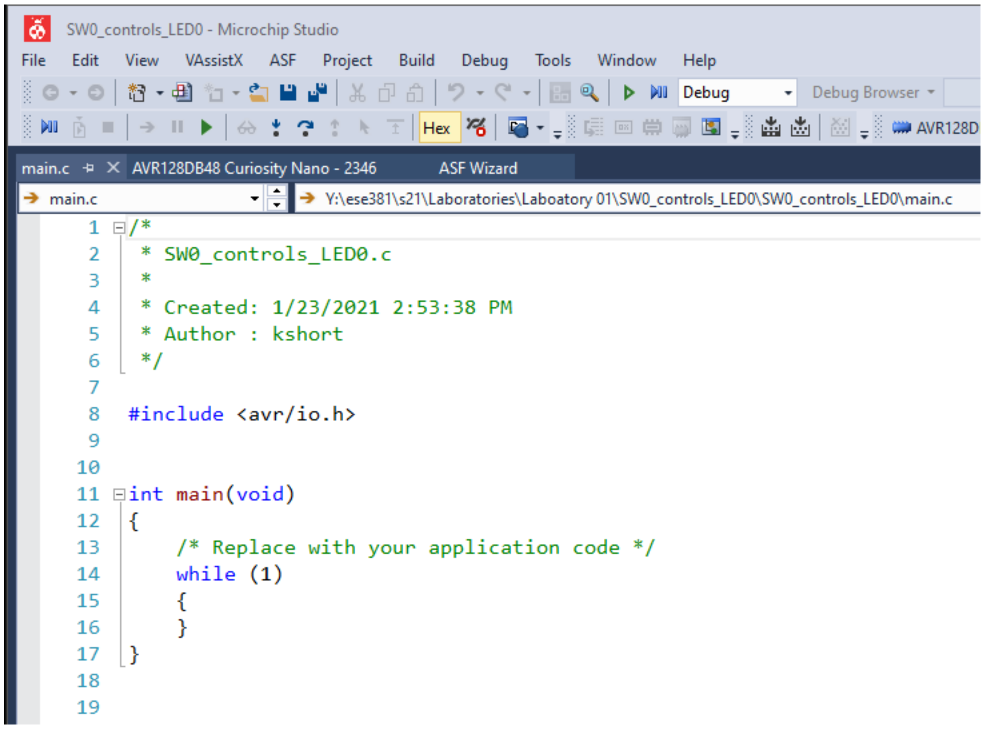
Datasheet

Supported Tools

- mEDBG
- Atmel-ICE
- MPLAB® PICkit 4
- JTAGICE3
- MPLAB® Snap
- EDBG
- EDBG MSD

OK Cancel

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



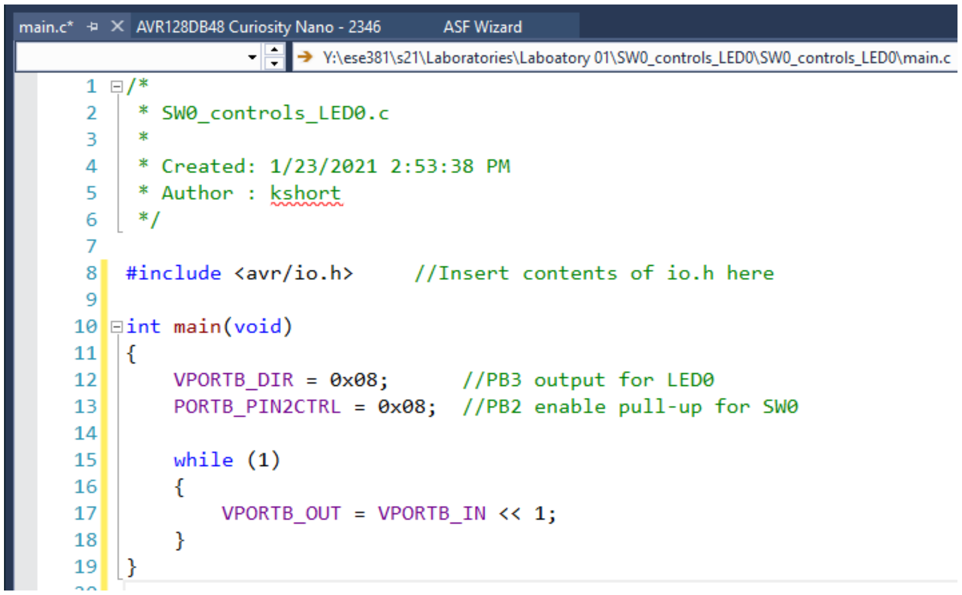
SW0_controls_LED0 - Microchip Studio
File Edit View VAssistX ASF Project Build Debug Tools Window Help
Debug
Debug Browser
Hex
AVR128D
main.c
AVR128DB48 Curiosity Nano - 2346
ASF Wizard

main.c

Y:\ese381\s21\Laboratories\Laboatory 01\SW0_controls_LED0\SW0_controls_LED0\main.c

```
/*
 * SW0_controls_LED0.c
 *
 * Created: 1/23/2021 2:53:38 PM
 * Author : kshort
 */

#include <avr/io.h>


int main(void)
{
    /* Replace with your application code */
    while (1)
    {
    }
}


```

Type in the code for the program.



main.c* AVR128DB48 Curiosity Nano - 2346 ASF Wizard

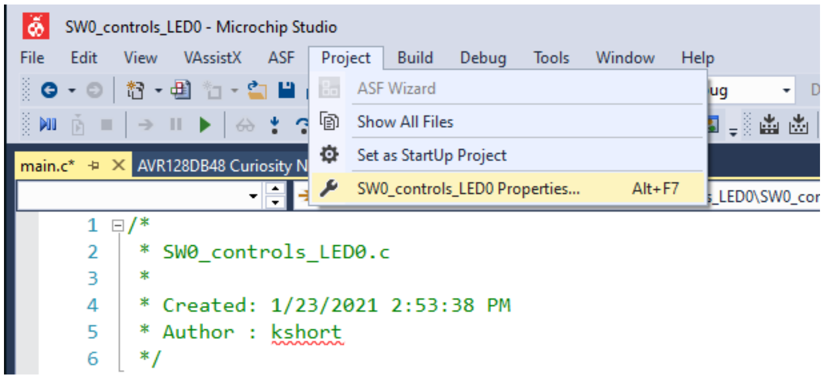
Y:\ese381\s21\Laboratories\Laboatory 01\SW0_controls_LED0\SW0_controls_LED0\main.c

```
/*
 * SW0_controls_LED0.c
 *
 * Created: 1/23/2021 2:53:38 PM
 * Author : kshort
 */

#include <avr/io.h>      //Insert contents of io.h here

int main(void)
{
    VPORTB_DIR = 0x08;      //PB3 output for LED0
    PORTB_PIN2CTRL = 0x08;  //PB2 enable pull-up for SW0

    while (1)
    {
        VPORTB_OUT = VPORTB_IN << 1;
    }
}
```

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.

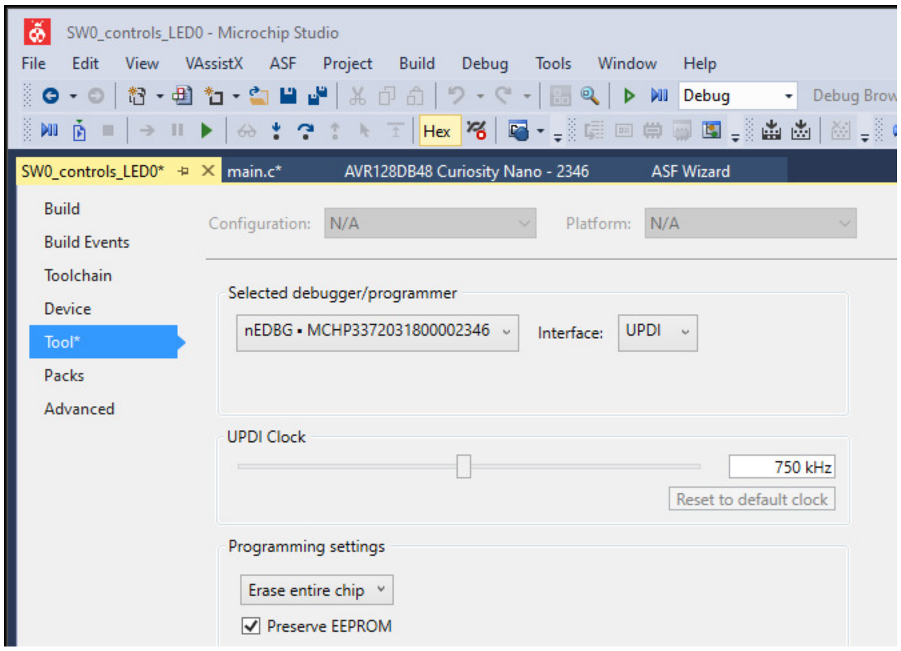
The microcontroller that you are developing code for is called the target microcontroller. In your case the target microcontroller is the AVR128DB48. The emulator lets you evaluate and debug the operation of a code running on the actual target microcontroller and interacting with any circuitry externally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

In the menu, go to Project and at the bottom of the drop-down menu and select SW0_controls_LED0 Properties.


SW0_controls_LED0 - Microchip Studio
File Edit View VAssistX ASF Project Build Debug Tools Window Help
ASF Wizard
Show All Files
Set as StartUp Project
SW0_controls_LED0 Properties... Alt+F7
main.c* AVR128DB48 Curiosity N

```
/*
 * SW0_controls_LED0.c
 *
 * Created: 1/23/2021 2:53:38 PM
 * Author : kshort
 */
```

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.


SW0_controls_LED0 - Microchip Studio
File Edit View VAssistX ASF Project Build Debug Tools Window Help
SW0_controls_LED0* main.c* AVR128DB48 Curiosity Nano - 2346 ASF Wizard
Build
Build Events
Toolchain
Device
Tool*
Packs
Advanced
Reset to default clock

Configuration: N/A Platform: N/A

Selected debugger/programmer: nEDBG • MCHP3372031800002346 Interface: UPDI

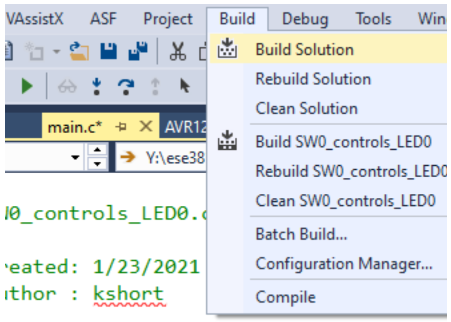
UPDI Clock: 750 kHz

Programming settings: Erase entire chip

[✓] Preserve EEPROM

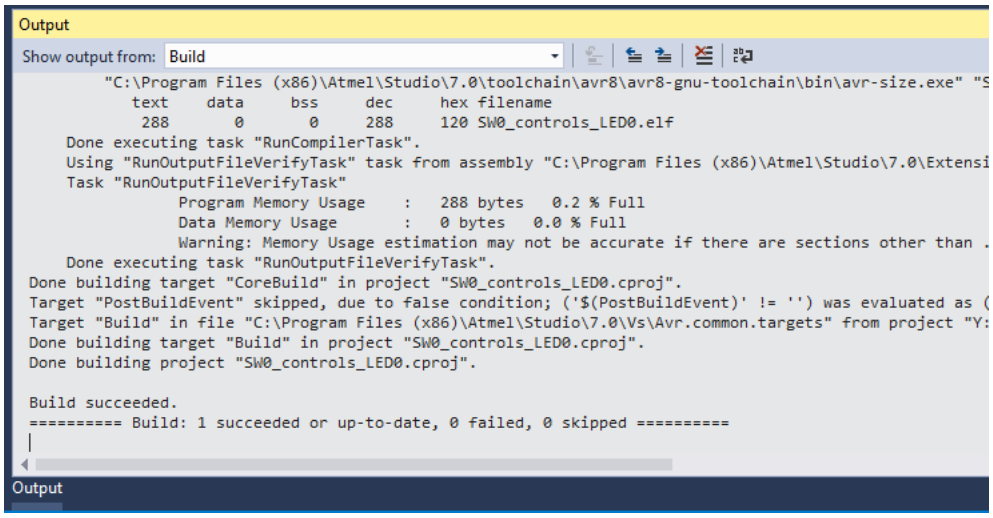
Next click on the main.c tab to select the editor window.

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



```
I0_controls_LED0.c
eated: 1/23/2021
thor : kshort
```

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.

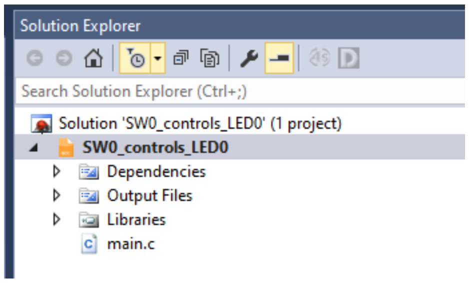


```
        "C:\Program Files (x86)\Atmel\Studio\7.0\toolchain\avr8\avr8-gnu-toolchain\bin\avr-size.exe" "S
           text    data     bss     dec     hex filename
            288       0       0     288     120 SW0_controls_LED0.elf
    Done executing task "RunCompilerTask".
    Using "RunOutputFileVerifyTask" 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 ==========
```

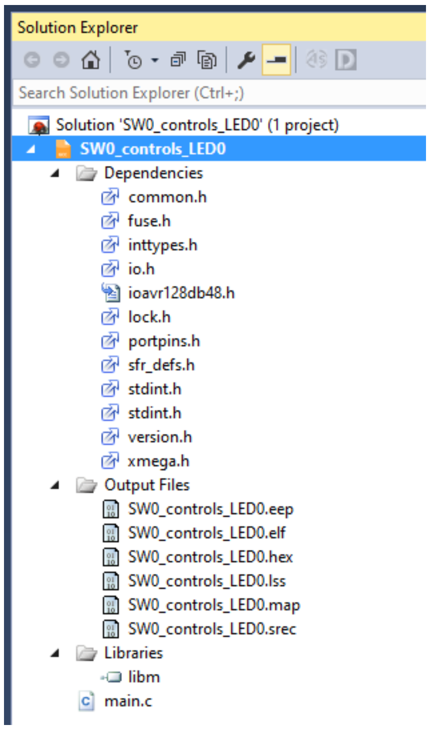
If errors or warnings are displayed, check for typos in your typed-in copy of the program and correct them. After doing so, again select Build Solution from the Build menu to compile the program again.

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.


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'SW0_controls_LED0' (1 project)
SW0_controls_LED0
Dependencies
Output Files
Libraries
main.c

Expand all of the paths under the project name.

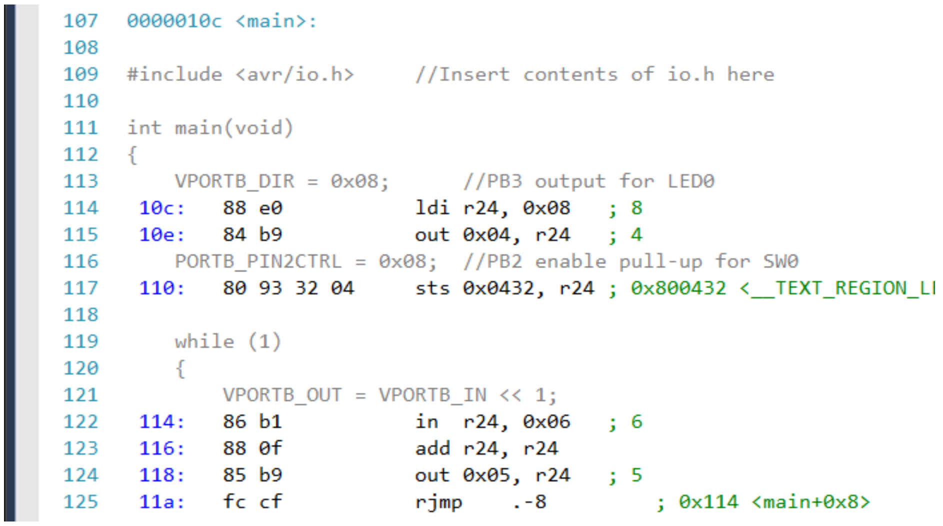


Solution 'SW0_controls_LED0' (1 project)
SW0_controls_LED0
Dependencies
common.h
fuse.h
inttypes.h
io.h
ioavr128db48.h
lock.h
portpins.h
sfr_defs.h
stdint.h
stdint.h
version.h
xmega.h
Output Files
SW0_controls_LED0.eep
SW0_controls_LED0.elf
SW0_controls_LED0.hex
SW0_controls_LED0.lss
SW0_controls_LED0.map
SW0_controls_LED0.srec
Libraries
libm
main.c

The files listed under Dependences 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 C source 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.



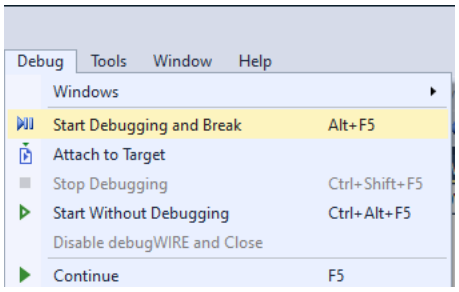
```
107  0000010c <main>:
108
109  #include <avr/io.h>     //Insert contents of io.h here
110
111  int main(void)
112  {
113      VPORTB_DIR = 0x08;      //PB3 output for LED0
114   10c:    88 e0          ldi r24, 0x08   ; 8
115   10e:    84 b9          out 0x04, r24   ; 4
116      PORTB_PIN2CTRL = 0x08;  //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>
```

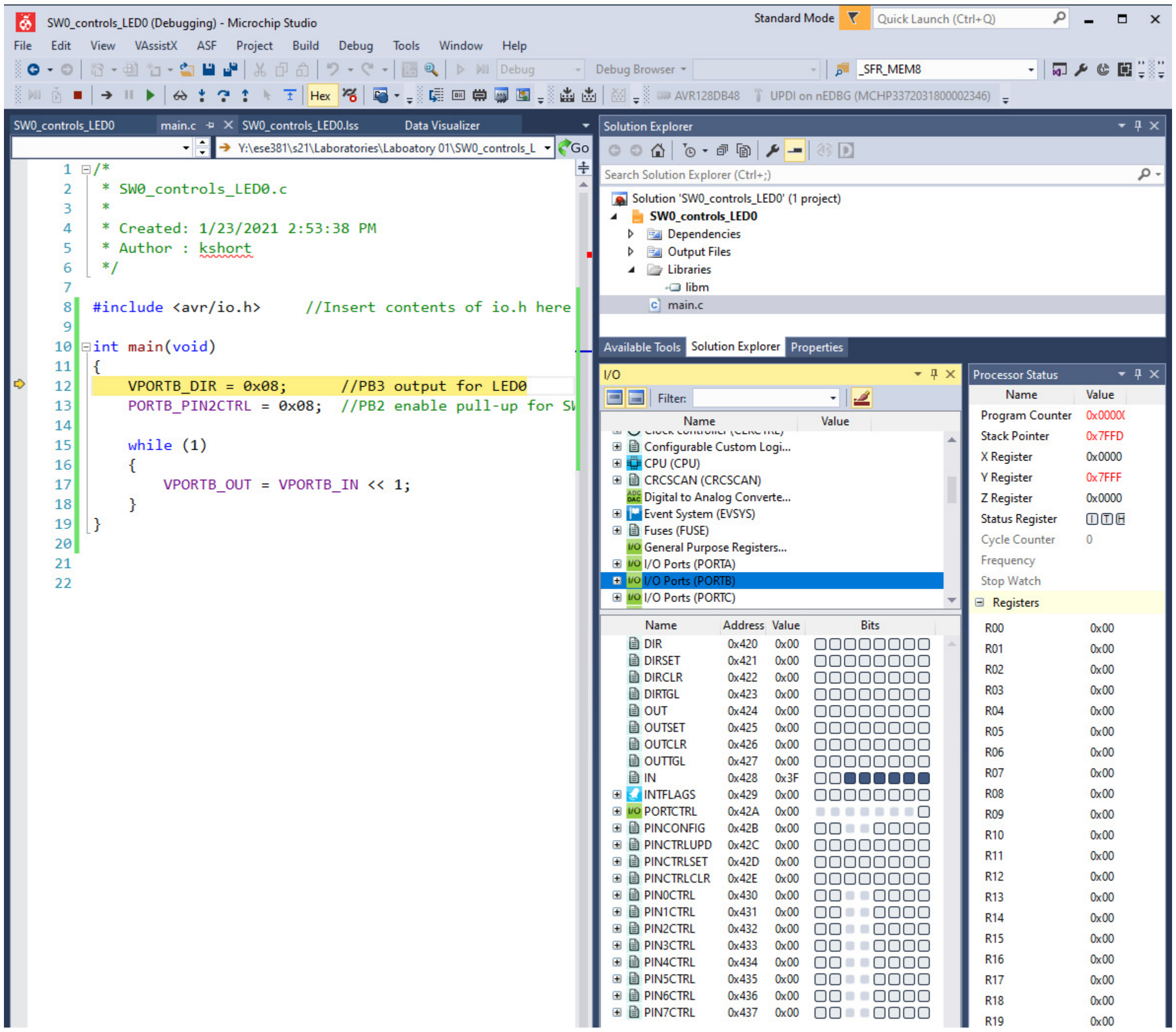
The first column of numbers in this file are the line numbers in the file. The second column in this portion of the file gives the address of the machine code in hexadecimal. The third column gives the actual machine code in hexadecimal. To the right of each line of machine code is the assembly language 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** code for your program code and **exit** code for your program code, if an exit occurs.

Note: **When you are asked to write a program as part of your laboratory, you are usually required to submit a PDF copy of your C source code 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.


Debug
Tools
Window
Help
Windows
Start Debugging and Break Alt+F5
Attach to Target
Stop Debugging Ctrl+Shift+F5
Start Without Debugging Ctrl+Alt+F5
Disable debugWIRE and Close
Continue F5


SW0_controls_LED0 (Debugging) - Microchip Studio
Standard Mode
Quick Launch (Ctrl+Q)
File Edit View VAssistX ASF Project Build Debug Tools Window Help
Debug
Debug Browser
_SFR_MEM8
Hex
AVR128DB48
UPDI on nEDBG (MCHP3372031800002346)
SW0_controls_LED0
main.c
SW0_controls_LED0.lss
Data Visualizer
Y:\ese381\s21\Laboratories\Laboatory 01\SW0_controls_L
Go
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'SW0_controls_LED0' (1 project)
SW0_controls_LED0
Dependencies
Output Files
Libraries
libm
main.c
Available Tools
Solution Explorer
Properties

```c
/*
 * SW0_controls_LED0.c
 *
 * Created: 1/23/2021 2:53:38 PM
 * Author : kshort
 */

#include <avr/io.h>     //Insert contents of io.h here

int main(void)
{
    VPORTB_DIR = 0x08;       //PB3 output for LED0
    PORTB_PIN2CTRL = 0x08;  //PB2 enable pull-up for SW
    
    while (1)
    {
        VPORTB_OUT = VPORTB_IN << 1;
    }
}
```

I/O

Filter:

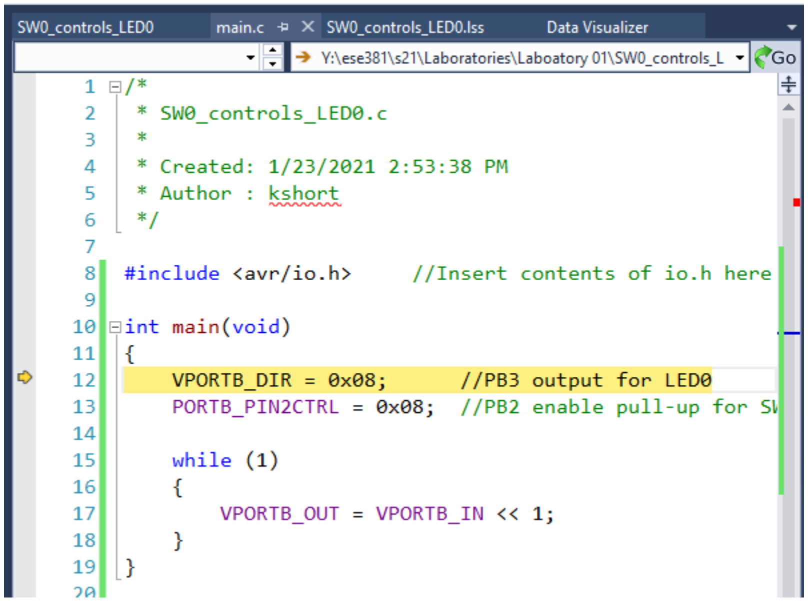
Name	Value
Clock Controller (CLKCTRL)	
Configurable Custom Logi...	
CPU (CPU)	
CRCSCAN (CRCSCAN)	
Digital to Analog Converte...	
Event System (EVSYS)	
Fuses (FUSE)	
General Purpose Registers...	
I/O Ports (PORTA)	
I/O Ports (PORTB)	
I/O Ports (PORTC)	

Name	Address	Value	Bits
DIR	0x420	0x00	☐☐☐☐☐☐☐☐
DIRSET	0x421	0x00	☐☐☐☐☐☐☐☐
DIRCLR	0x422	0x00	☐☐☐☐☐☐☐☐
DIRTGL	0x423	0x00	☐☐☐☐☐☐☐☐
OUT	0x424	0x00	☐☐☐☐☐☐☐☐
OUTSET	0x425	0x00	☐☐☐☐☐☐☐☐
OUTCLR	0x426	0x00	☐☐☐☐☐☐☐☐
OUTTGL	0x427	0x00	☐☐☐☐☐☐☐☐
IN	0x428	0x3F	☐☐■■■■■■
INTFLAGS	0x429	0x00	☐☐☐☐☐☐☐☐
PORTCTRL	0x42A	0x00	·······☐
PINCONFIG	0x42B	0x00	☐☐··☐☐☐☐
PINCTRLUPD	0x42C	0x00	☐☐☐☐☐☐☐☐
PINCTRLSET	0x42D	0x00	☐☐☐☐☐☐☐☐
PINCTRLCLR	0x42E	0x00	☐☐☐☐☐☐☐☐
PIN0CTRL	0x430	0x00	☐☐··☐☐☐☐
PIN1CTRL	0x431	0x00	☐☐··☐☐☐☐
PIN2CTRL	0x432	0x00	☐☐··☐☐☐☐
PIN3CTRL	0x433	0x00	☐☐··☐☐☐☐
PIN4CTRL	0x434	0x00	☐☐··☐☐☐☐
PIN5CTRL	0x435	0x00	☐☐··☐☐☐☐
PIN6CTRL	0x436	0x00	☐☐··☐☐☐☐
PIN7CTRL	0x437	0x00	☐☐··☐☐☐☐

Processor Status

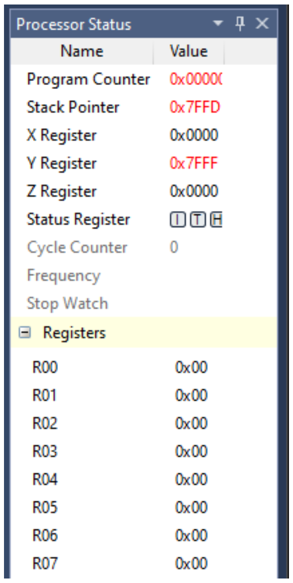
Name	Value
Program Counter	0x0000(
Stack Pointer	0x7FFD
X Register	0x0000
Y Register	0x7FFF
Z Register	0x0000
Status Register	I T H
Cycle Counter	0
Frequency	
Stop Watch	
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x00
R17	0x00
R18	0x00
R19	0x00

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. Note: even though this is where you think of your program as starting, the actual startup code has been automatically executed before this.



```
/*
 * SW0_controls_LED0.c
 *
 * Created: 1/23/2021 2:53:38 PM
 * Author : kshort
 */

#include <avr/io.h>     //Insert contents of io.h here

int main(void)
{
    VPORTB_DIR = 0x08;        //PB3 output for LED0
    PORTB_PIN2CTRL = 0x08;   //PB2 enable pull-up for SW

    while (1)
    {
        VPORTB_OUT = VPORTB_IN << 1;
    }
}
```

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

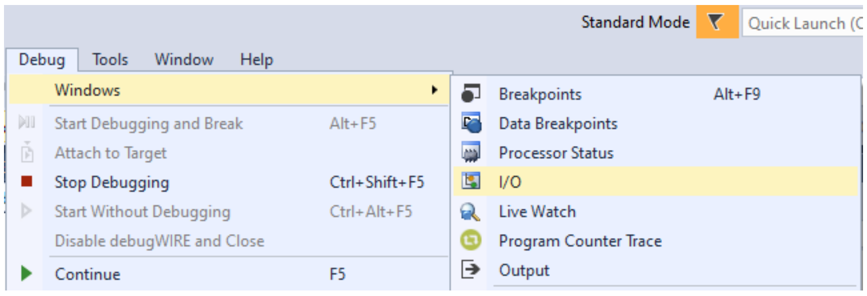


Name	Value
Program Counter	0x0000(
Stack Pointer	0x7FFD
X Register	0x0000
Y Register	0x7FFF
Z Register	0x0000
Status Register	I T H
Cycle Counter	0
Frequency	
Stop Watch	
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00

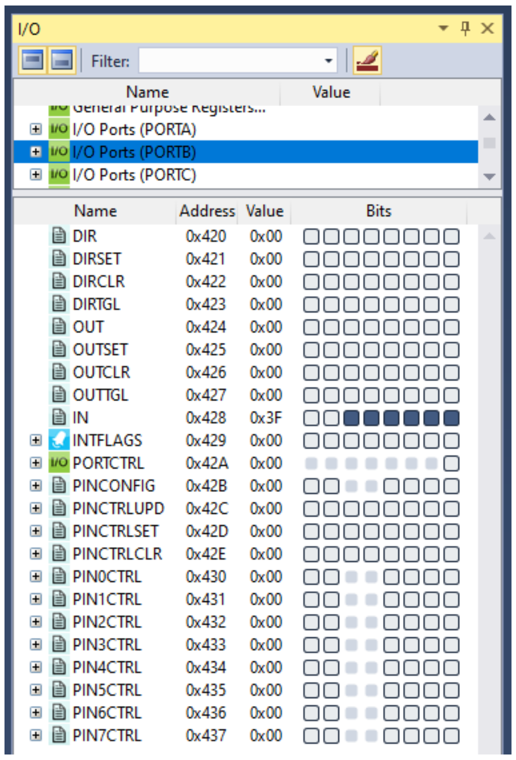
This window shows all the AVR128DB48's CPU registers, including the general purpose registers. It also has three fields that don't correspond to actual AVR128DB48 registers: Cycle Counter, Frequency, and Stop Watch. These are grayed out because they are only usable when you have selected simulator as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



Name	Value
General Purpose Registers...	
I/O Ports (PORTA)	
I/O Ports (PORTB)	
I/O Ports (PORTC)	

Name	Address	Value	Bits
DIR	0x420	0x00	☐☐☐☐☐☐☐☐
DIRSET	0x421	0x00	☐☐☐☐☐☐☐☐
DIRCLR	0x422	0x00	☐☐☐☐☐☐☐☐
DIRTGL	0x423	0x00	☐☐☐☐☐☐☐☐
OUT	0x424	0x00	☐☐☐☐☐☐☐☐
OUTSET	0x425	0x00	☐☐☐☐☐☐☐☐
OUTCLR	0x426	0x00	☐☐☐☐☐☐☐☐
OUTTGL	0x427	0x00	☐☐☐☐☐☐☐☐
IN	0x428	0x3F	☐☐■■■■■■
INTFLAGS	0x429	0x00	☐☐☐☐☐☐☐☐
PORTCTRL	0x42A	0x00	·······☐
PINCONFIG	0x42B	0x00	☐☐··☐☐☐☐
PINCTRLUPD	0x42C	0x00	☐☐☐☐☐☐☐☐
PINCTRLSET	0x42D	0x00	☐☐☐☐☐☐☐☐
PINCTRLCLR	0x42E	0x00	☐☐☐☐☐☐☐☐
PIN0CTRL	0x430	0x00	☐☐··☐☐☐☐
PIN1CTRL	0x431	0x00	☐☐··☐☐☐☐
PIN2CTRL	0x432	0x00	☐☐··☐☐☐☐
PIN3CTRL	0x433	0x00	☐☐··☐☐☐☐
PIN4CTRL	0x434	0x00	☐☐··☐☐☐☐
PIN5CTRL	0x435	0x00	☐☐··☐☐☐☐
PIN6CTRL	0x436	0x00	☐☐··☐☐☐☐
PIN7CTRL	0x437	0x00	☐☐··☐☐☐☐

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, which was actually used in the program.

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 ON is when the pushbutton connected to PA2 and the pushbutton connected to PA5 are simultaneously 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.

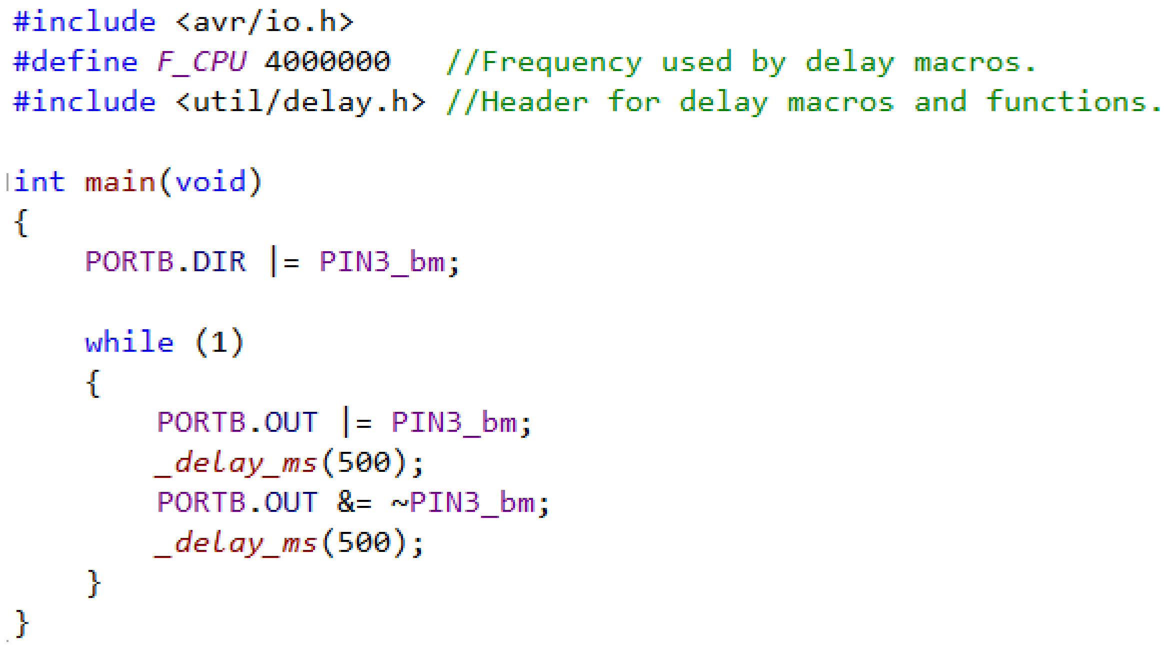
Using the program that was provided for Task 1 as a starting point, write this program. Name the program `PB1_and PB2_control_LED`. Only use the DIR, IN, OUT, and PINxCTRL port registers in your program. You are being limited to these registers for this task for a purpose, so you must comply with this restriction. It does not matter how your program changes outputs of VPORTB other than PB5.

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

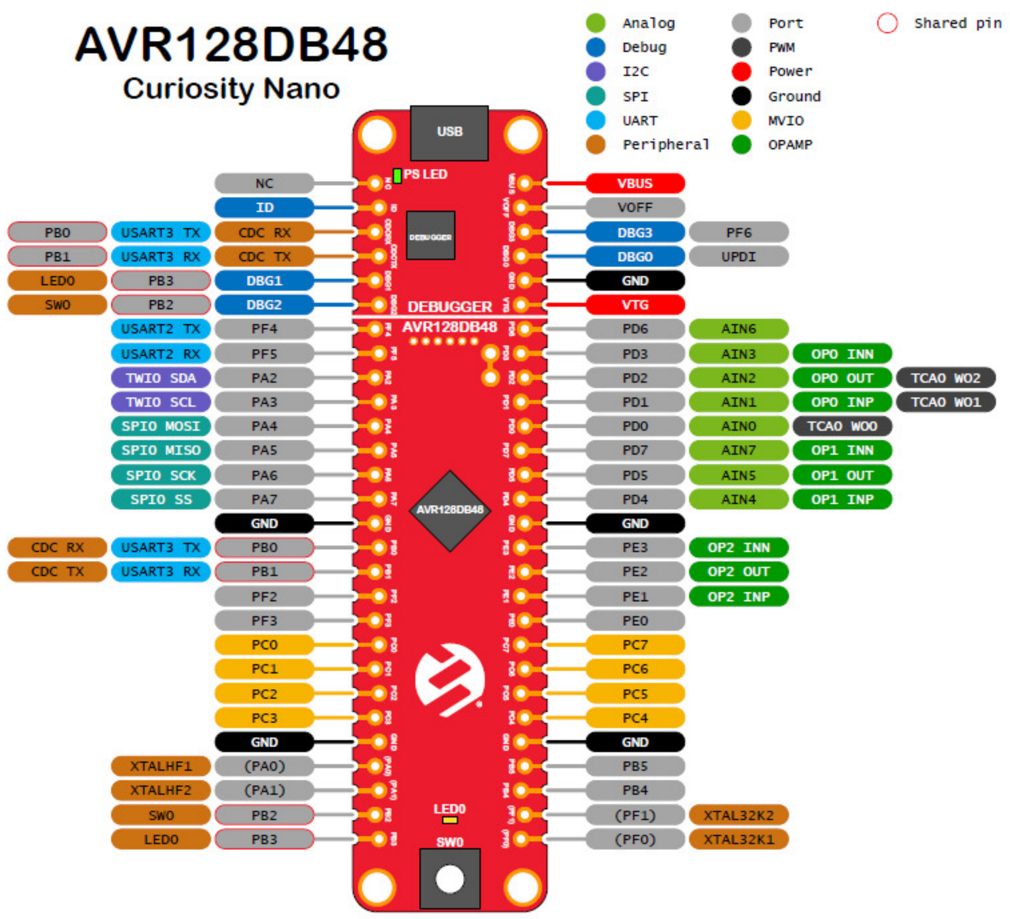
Laboratory Task 3: Using Emulation to Execute the C Program blink_LED0

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 to -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);
    }
}
```



Questions

Provide a brief written answer for each of the following questions. Submit your answers at the end of your assigned laboratory section.

1. For Task 1, without the oscilloscope connected, does the program produce the same result without the internal pull-up resistor enabled as it does with the pull-up enabled? Explain your answer.

2. When you connect the oscilloscope and run the program of Task 1 without the pull-up enabled, does the system operate the same as when the oscilloscope is not connected? Explain your answer.

3. Give an explanation for why the system from Task 1, without the pull-up enabled and without the oscilloscope connected, operates the way it does.

4. How accurate was the 1 Hz signal at PB3 for Task 3? Compute the percent error for the signal.

5. If your 1 Hz signal for Task 3 was reasonably accurate, what can you conclude about the CPU's clock frequency?