

CECS 412 INTRO TO EMBEDDED SYSTEMS
LAB 1
ATMEL Studio, AVR Assembly, ATmega328P(B)

Eugene Rockey
Copyright 2018, All Rights Reserved

Yellow highlight points out lab related action required from the student.

Green highlight points out report related action required from the student.

Blue highlight emphasizes certain terms and information.

This lab should be performed while referring to classroom notes, Xplained Mini User Manual, ATmega328P(B) Datasheet, AVR Instruction Manual, and the AVR Assembler Manual.

Lab 1 is designed to introduce and familiarize the student to and with ATMEL Studio, AVR Assembly, and the ATmega328P(B) Xplained Mini development board. Atmel Studio is an integrated development environment (IDE) used in CECS412 and ECE412 to develop embedded software for all its supported MCUs. In Lab 1, the student will use ATMEL Studio along with AVR assembly language programming to realize the fundamentals of programming the ATmega328P(B) MCU and some of its internal operations.

Assembly programming is the ultimate method of configuring and controlling digital hardware with the greatest efficiency and accuracy.

Assembly language is a low-level mnemonic language that is native to a specific target microprocessor or microcontroller (computer on a chip). Assembly source code is not hardware-independent like C source code or other higher-level languages. Microprocessors and microcontrollers execute their specific set of assembly instructions and therefore have their own unique assembly language. And since the target microcontroller for CECS412 and ECE412 is the 8-bit AVR ATmega328P(B), installed on the Xplained Mini evaluation board, its assembly instruction set and programming model will be two major learning objectives for lab 1. All documentation for the AVR assembly language is online at an Atmel website. And many third-party websites are available too. One very important document is the ATmega328P(B) datasheet since it details the chip's internal hardware, operations, and assembly instruction set.

<http://www.atmel.com/tools/MEGA328P-XMINI.aspx?tab=documents>

*Important: Always be aware of static electricity and take actions to prevent it from destroying electronic equipment. Use antistatic mats on the workbench. Use the antistatic wrist straps. 100% cotton clothing is usually has less static charge than synthetics. Rinsing your hands with warm water helps dissipate static charge. Hand lotion helps reduce static charge too. Washing and drying clothes using fabric softener reduces static buildup. **Keep the Xplained Mini board in its anti-static box when not in use. Do not allow the board to contact anything metal or conductive while powered on.**

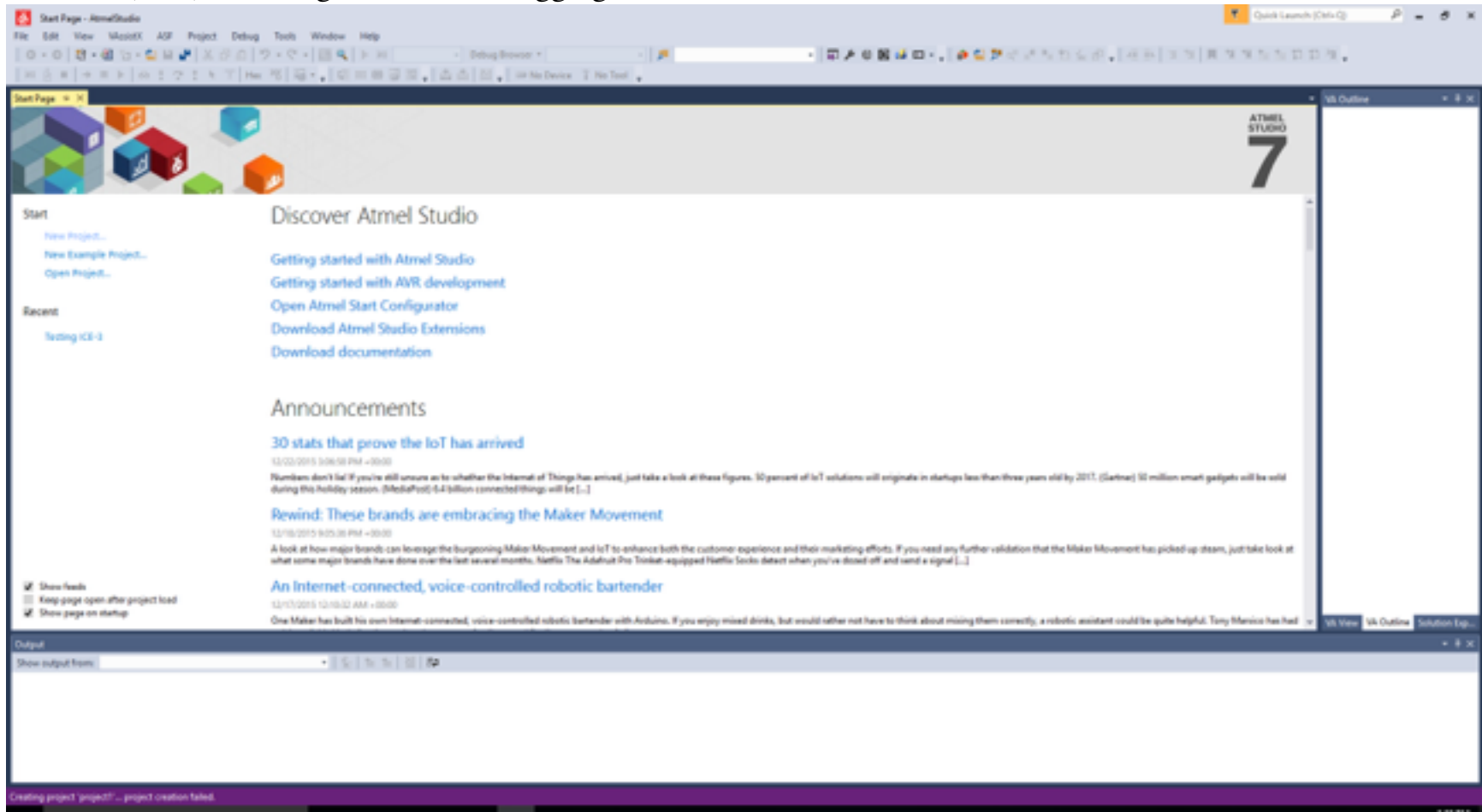
Part 1: Installation:

Install the latest Atmel Studio on your own compatible MS Windows PC or use a preinstalled lab computer (if available). The ECE domain may have special logon procedures for using its lab computers. If installing and setting up on your own PC, then use the following Atmel website (websites and their content may change over time). During or after installation, set the Atmel Studio UI to ADVANCED mode.

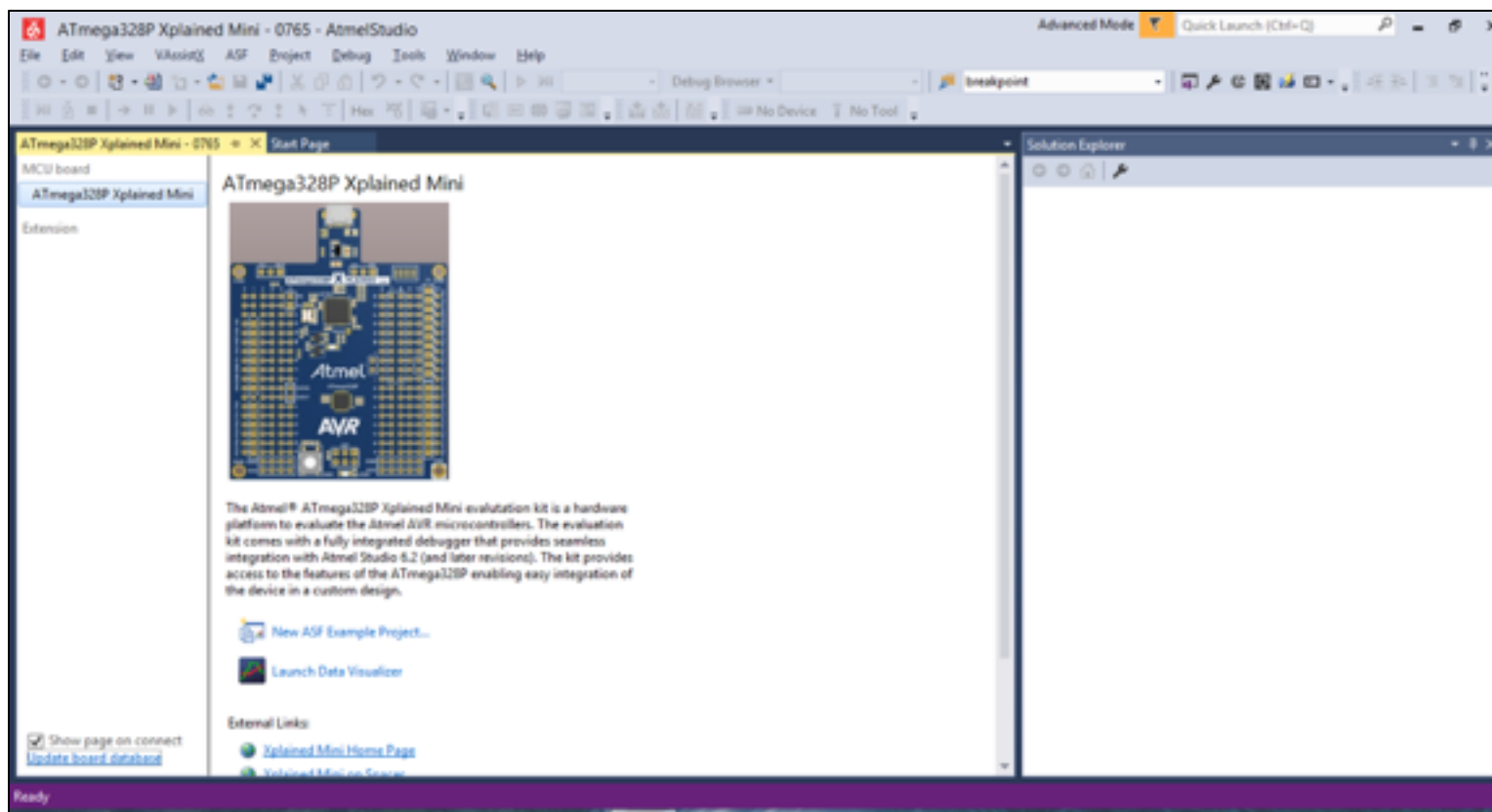
<http://www.atmel.com/tools/ATMELSTUDIO.aspx>

Part 2: Explore Atmel Studio:

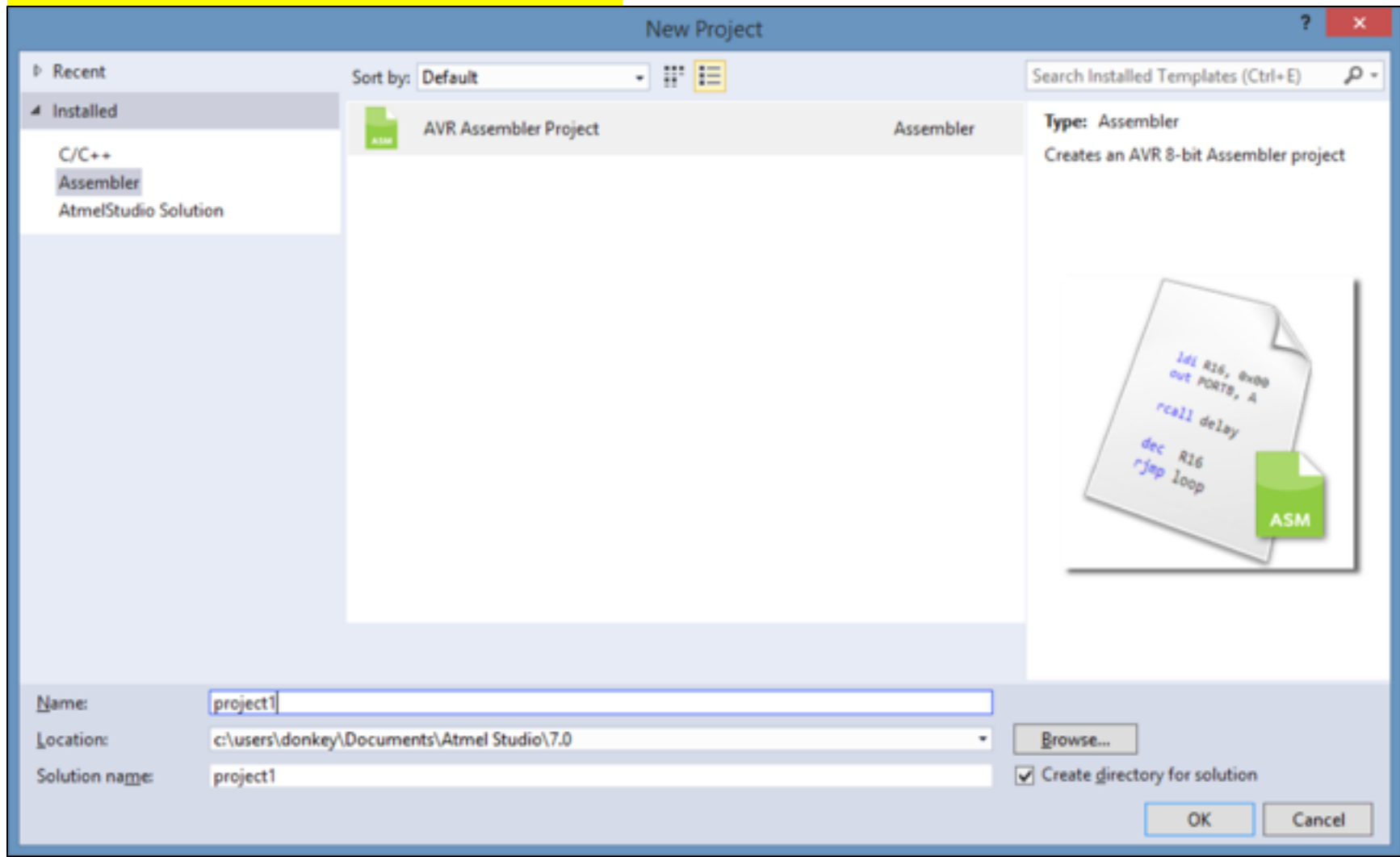
In this part, each student will explore some of the various components and menus of the Atmel Studio graphic user interface (GUI) including the Atmel debugging tools.



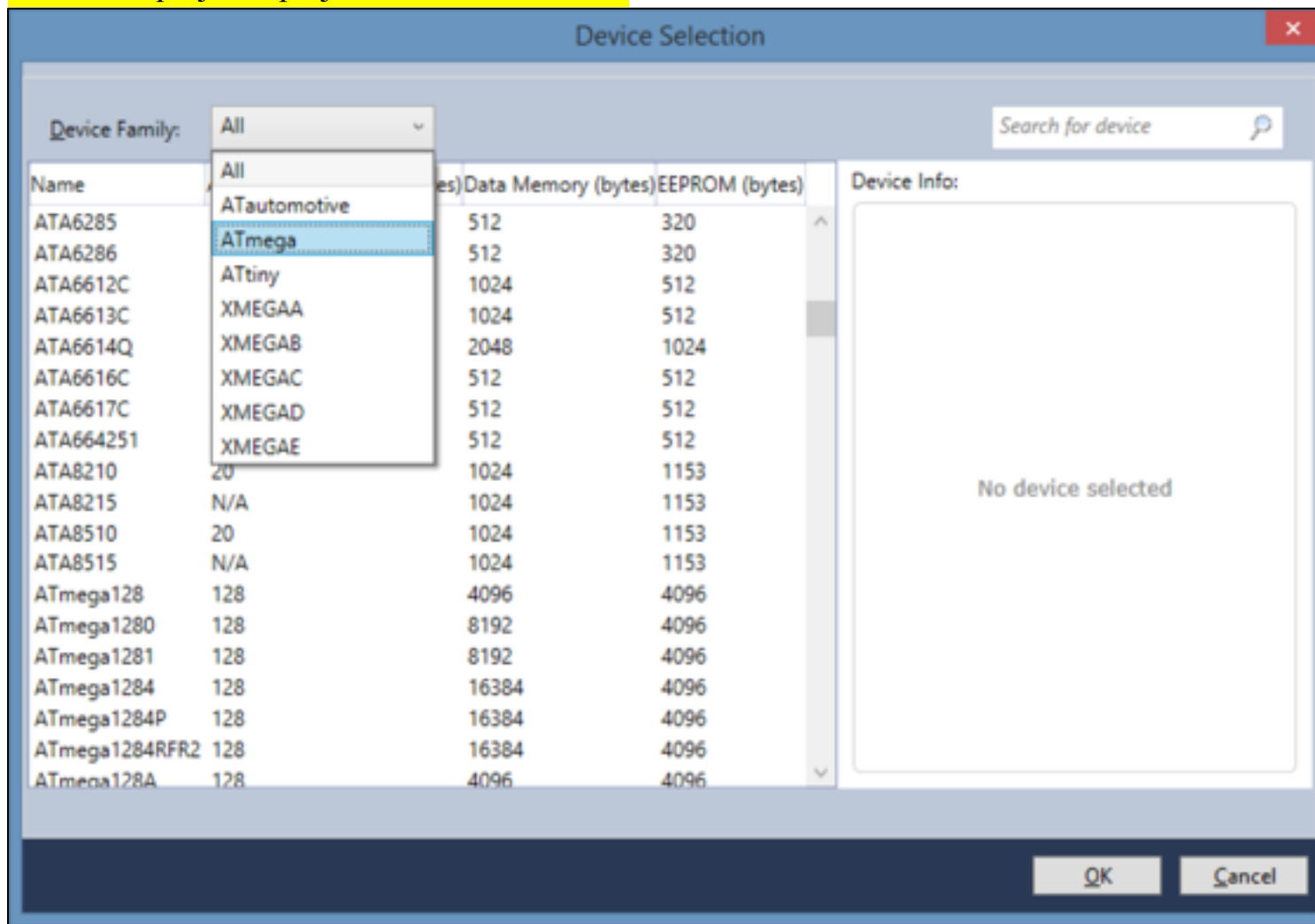
The Atmel menus and toolbars include standard MS Windows features as well as Atmel specific help, operations and tools for editing, programming, debugging, and interfacing to Atmel equipment. If the Xplained Mini board is connected then there should be a picture of it along with links to useful resources. Atmel's example projects provide great insight and reference for learning how to program a target device, explore these examples.



The File/Open/Project/Solution... menu path provides quick access to recently worked projects. Start a new assembler project using the File/New/Project... path.



In 'New Project' window, choose 'Assembler'. Look at the description pane and you will see that this action creates an 8-bit assembler project. Again, the ATmega328P(B) is a low power 8-bit MCU. Name this assembler project "project1" then click OK.



In the 'Device Selection' window, choose the 'Device Family' as 'ATmega' then scroll down and choose the target MCU or 'ATmega328P(B)'. Notice the application/boot (FLASH) memory, SRAM data memory, and the EEPROM memory. These memories, their purpose, and their locations, internal or external, are very important to know when writing assembly code for the MCU. In the AVR world, I/O locations, EEPROM locations, and SRAM locations all share a common memory map while FLASH locations have a separate memory map. A memory map specifies the hardware with respect to memory addresses.

The 'Device Info' box has very useful links to information about the target MCU, explore the [Datasheet](#) and [Device Page](#).

Device Selection

Device Family: ATmega

Search for device

Name	App./Boot Memory (Kbytes)	Data Memory (bytes)	EEPROM (bytes)
ATmega324PB	32	2048	1024
ATmega325	32	2048	1024
ATmega3250	32	2048	1024
ATmega3250A	32	2048	1024
ATmega3250P	32	2048	1024
ATmega3250PA	32	2048	1024
ATmega325A	32	2048	1024
ATmega325P	32	2048	1024
ATmega325PA	32	2048	1024
ATmega328	32	2048	1024
ATmega328P	32	2048	1024
ATmega328PB	32	2048	1024
ATmega329	32	2048	1024
ATmega3290	32	2048	1024
ATmega3290A	32	2048	1024
ATmega3290P	32	2048	1024
ATmega3290PA	32	2048	1024
ATmega329A	32	2048	1024
ATmega329P	32	2048	1024








Device Info:

Device Name: ATmega328P
 Speed: N/A
 Vcc: N/A
 Family: ATmega

[Datasheet \(Summary\)](#)

[Device Page](#)

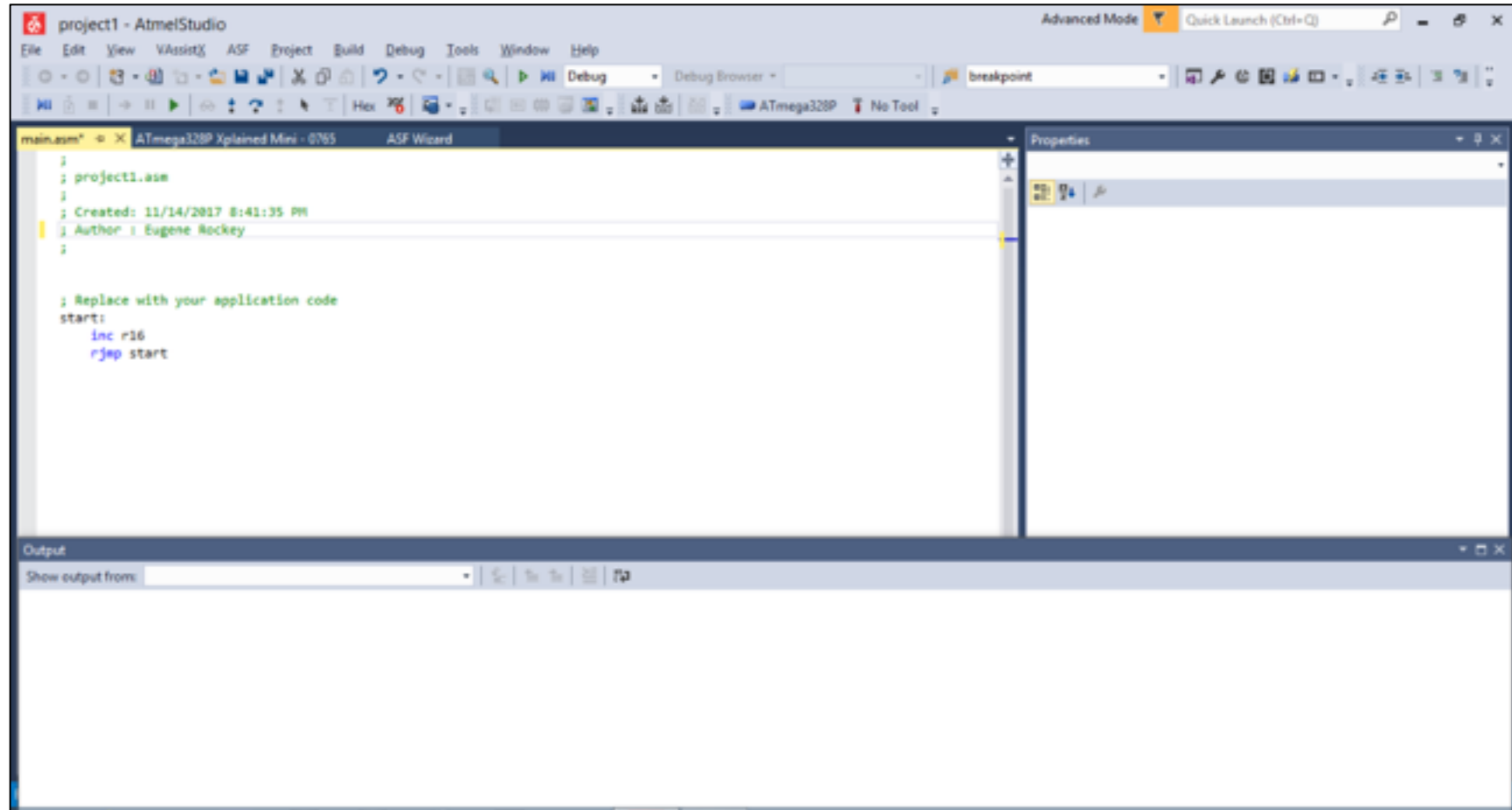
Supported Tools

 [Atmel-ICE](#)
 [AVR Dragon](#)
 [AVRISP mkII](#)
 [AVR ONE!](#)
 [EDBG](#)
 [EDBG MSD](#)
 [JTAGICE3](#)

OK

Cancel

Now click OK, the default main.asm code will be automatically produced in the editor window. The project assembly code may now be entered in main.asm overwriting the default code.



Enter the following lines in main.asm, make sure the four columns are aligned when entering assembly code.

```
/*
 * project1.asm
 *
 * Created: 1/18/2018 5:18:16 PM
 * Author: Eugene Rockey
 */
```

```

        .eseg                                ;EEPROM data segment, 0x0 to 0x3FF
        .org      0x0                        ;program the EEPROM using a .eep file
eevar:   .dw      0xfaff
msg:     .db      "HelloWorld",'\\n',0

```

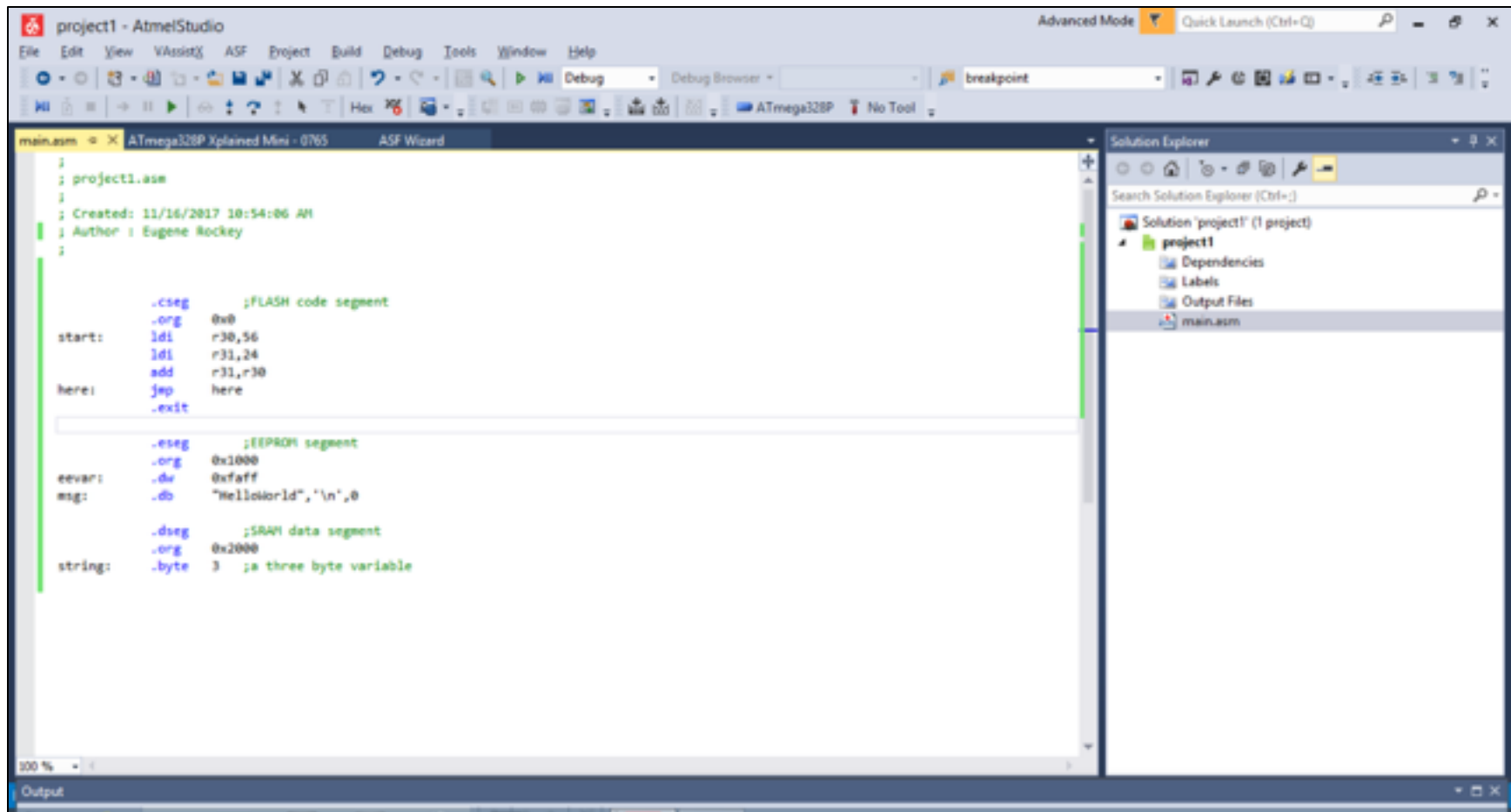
```

        .dseg                      ;SRAM data segment, 0x100 to 0x8FF
        .org      0x100
string: .byte      3               ;a three byte var

```

```
.cseg                                ;FLASH code segment, 0x0 to 0x3FFF-Loader
.org      0x0
start:    ldi       r30,56
          ldi       r31,24
          add       r31,r30
here:     jmp       here
          .exit      ;make .exit the very last line
```

This code demonstrates which type of memory is being targeted according to the assembler directives `.cseg`, `.eseg`, and `.dseg`. Assembler directives are identified by the period character. In your report, research and discuss the directives found in this example code including `.cseg`, `.eseg`, `.dseg`, `.org`, `.exit`, `.dw`, `.db`, and `.byte`. Refer to the AVR Assembler Manual.



```

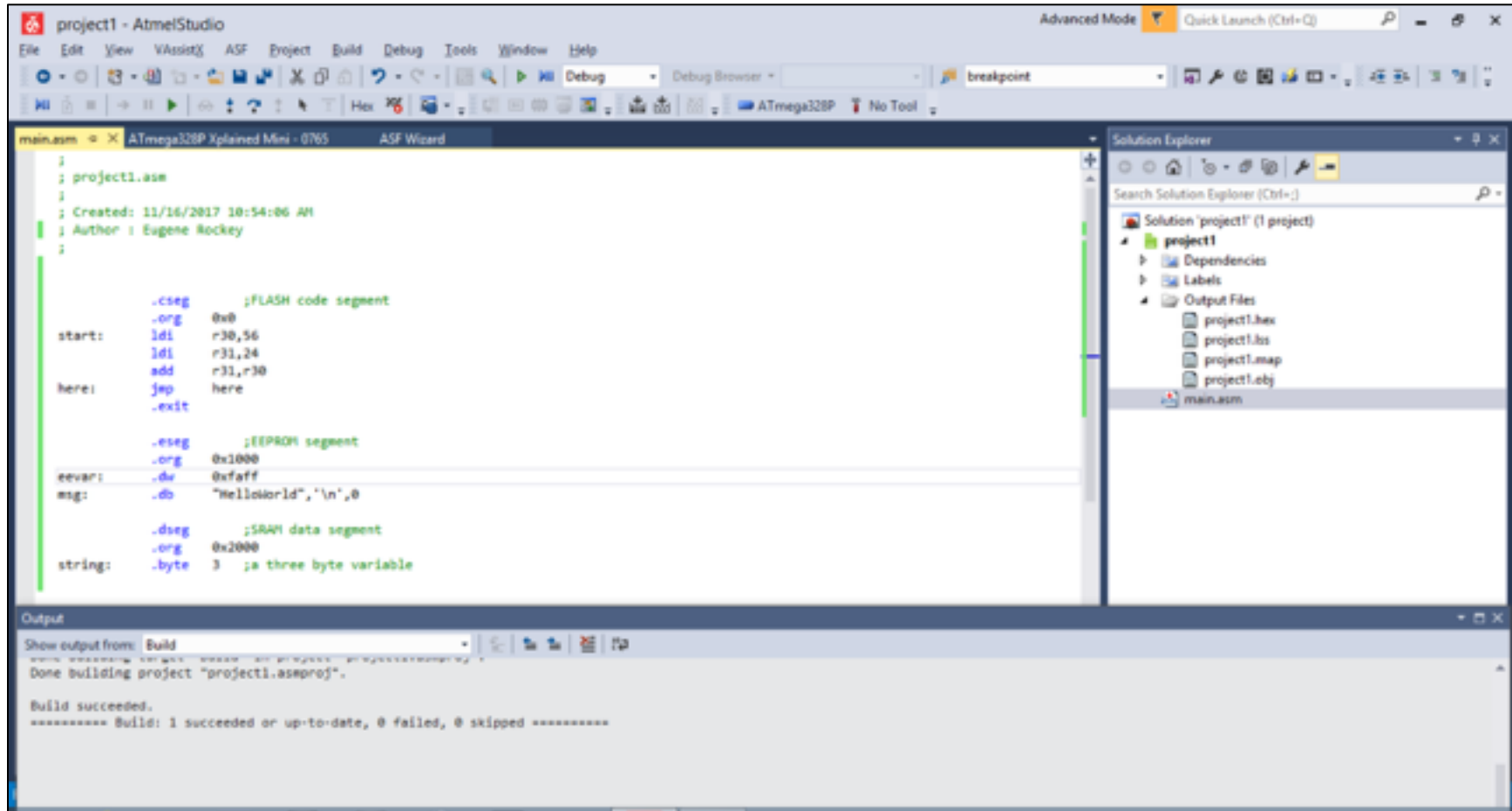
; project1.asm
;
; Created: 11/16/2017 10:54:06 AM
; Author : Eugene Rockey
;
;
; FLASH code segment
.cseg
.org 0x0
start:
    ldi r30,56
    ldi r31,24
    add r31,r30
here:
    jmp here
    .exit

; EEPROM segment
.eseg
.org 0x1000
eevar:
    .dw 0xffff
msg:
    .db "HelloWorld","\n",0

; SRAM data segment
.dseg
.org 0x2000
string:
    .byte 3 ;a three byte variable

```

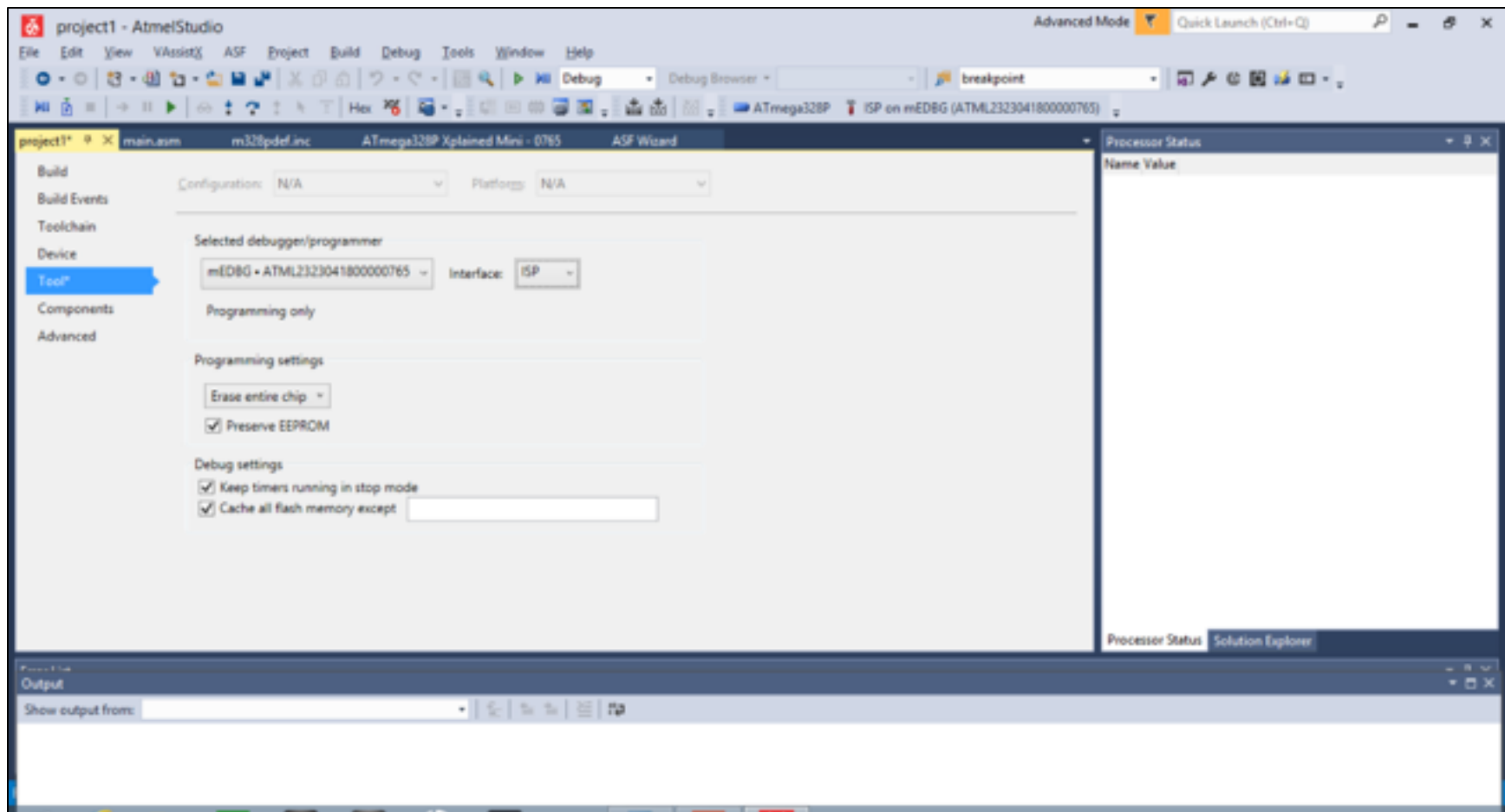
Under the 'Build' menu, click 'Build Solution'. Notice in the Solution Explorer pane that output files are generated. Generated files include a .hex, .lss, .map and .obj file. In your report, research and discuss each of these file types: .hex, .lss, .map, and .obj.



After BUILDing in Atmel Studio, there are two options for executing and debugging the project code, either by using the Xplained Mini board or the built-in simulator. The simulator can be used if the board is not available, however; at this time, it seems the simulator can only simulate the CPU execution, internal memories, and registers. The simulator does not simulate peripherals nor any operations outside of the MCU core. So at times, it may be quicker to use the simulator but other times it may be necessary to connect the board.

Option 1: Using the Xplained Mini board (study this option until you have the Xplained Mini kit):

With the board connected to the computer, click 'Build Solution' again. Click the hammer icon in the tool bar and select mEDBG and ISP as the 'Selected debugger/programmer' and 'Interface'. The MCU must first be programmed before the code can be executed and or debugged by the on-board mEDBG chip. The mEDBG chip is a supporting MCU providing the virtual communication interface and in-circuit debugging operations between the AT-mega328P(B) and Atmel Studio.



Click the 'Tools' menu then 'Device Programming'. In the 'Device Programming' window click 'Apply' then 'Read' then 'Memories' then 'Program'. After programming is complete, close the 'Device Programming' window.

mEDBG (ATML2323041800000765) - Device Programming



Tool	Device	Interface	Device signature	Target Voltage
mEDBG	ATmega328P	ISP	0x1E950F	5.0 V

Apply

Read

Read



Interface settings

Tool information

Tool settings

Device information

Oscillator calibration

Memories

Fuses

Lock bits

Production file

Device

Erase Chip

Erase now

Flash (32 KB)

c:\users\donkey\Documents\Atmel Studio\7.0\project1\project1\Debug\project1.hex

☒ Erase device before programming

☒ Verify Flash after programming

☐ Advanced

Program

Verify

Read...

EEPROM (1 KB)

☒ Verify EEPROM after programming

☐ Advanced

Program

Verify

Read...

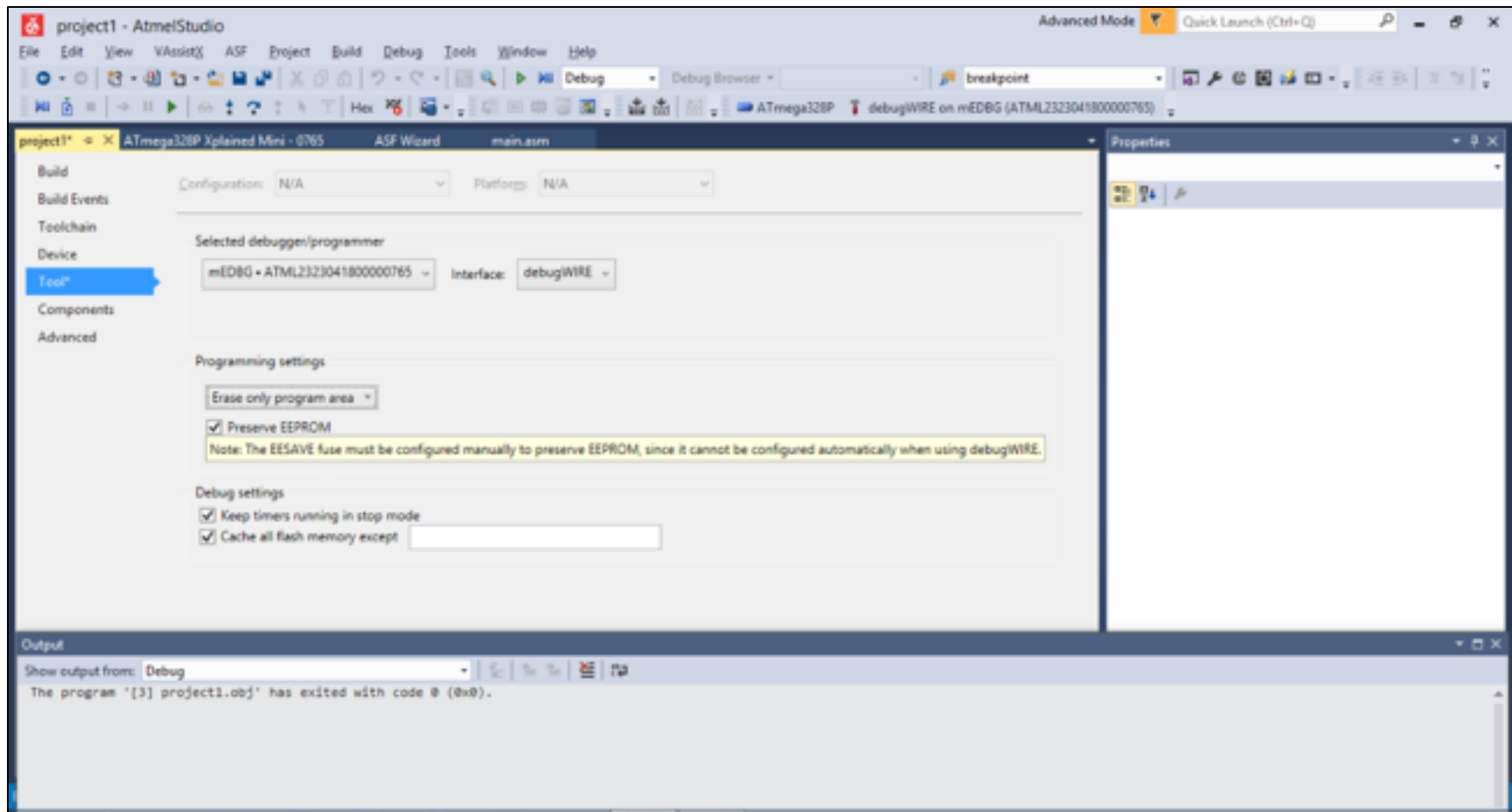
Erasing device... OK
Programming Flash...OK
Verifying Flash...OK

Verifying Flash...OK

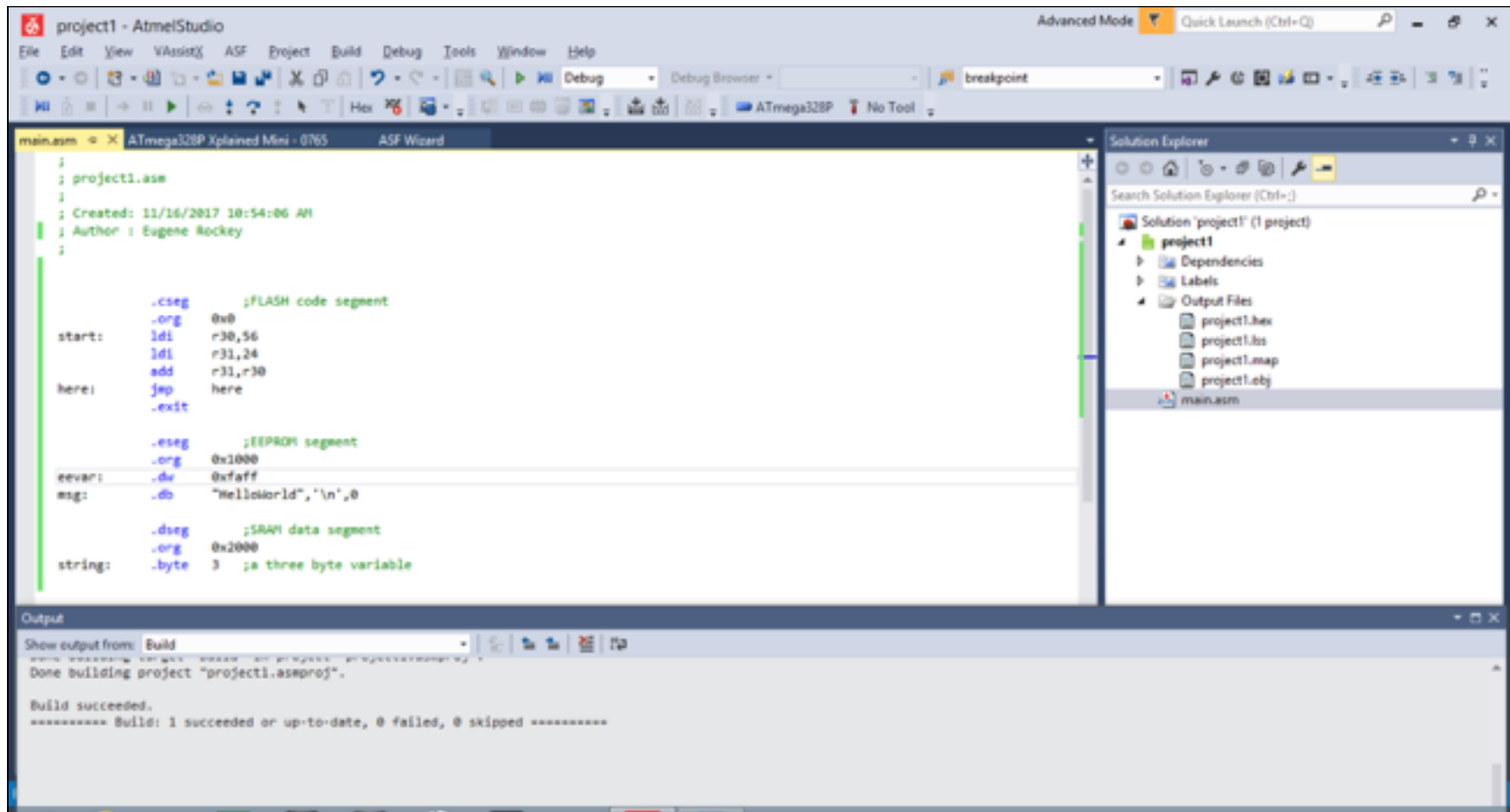
Close

Switch to Debug mode...

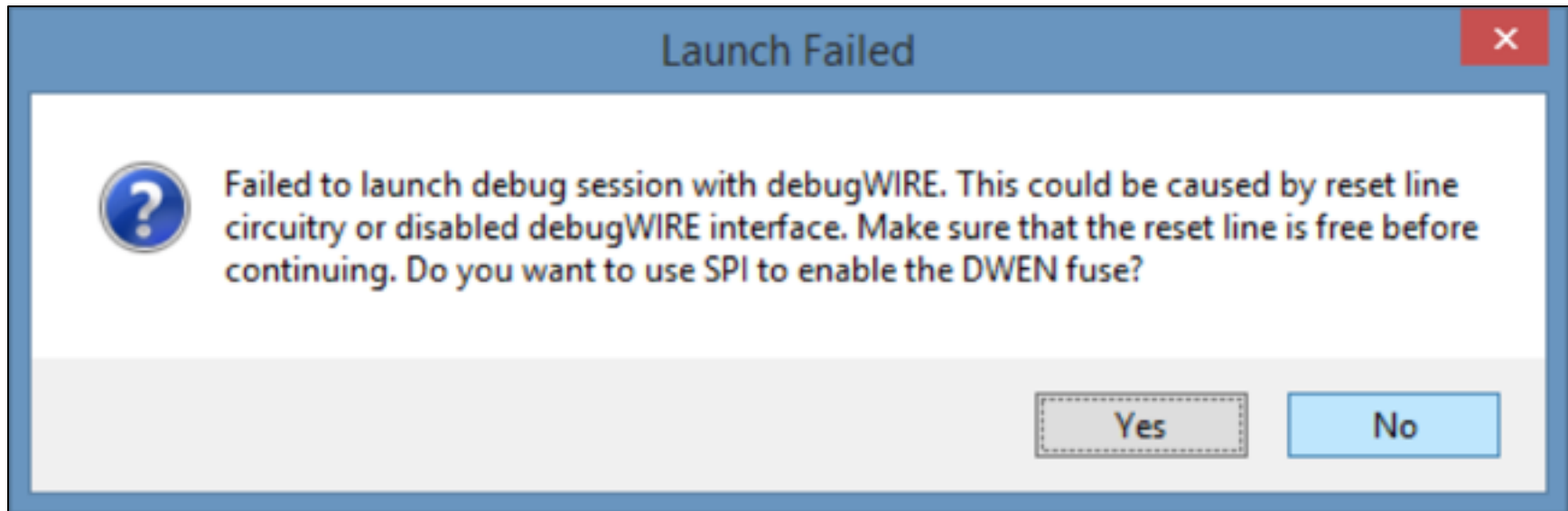
Now that the board has been programmed, change the interface from ISP to debugWIRE in the project1 tab or in the toolbar.



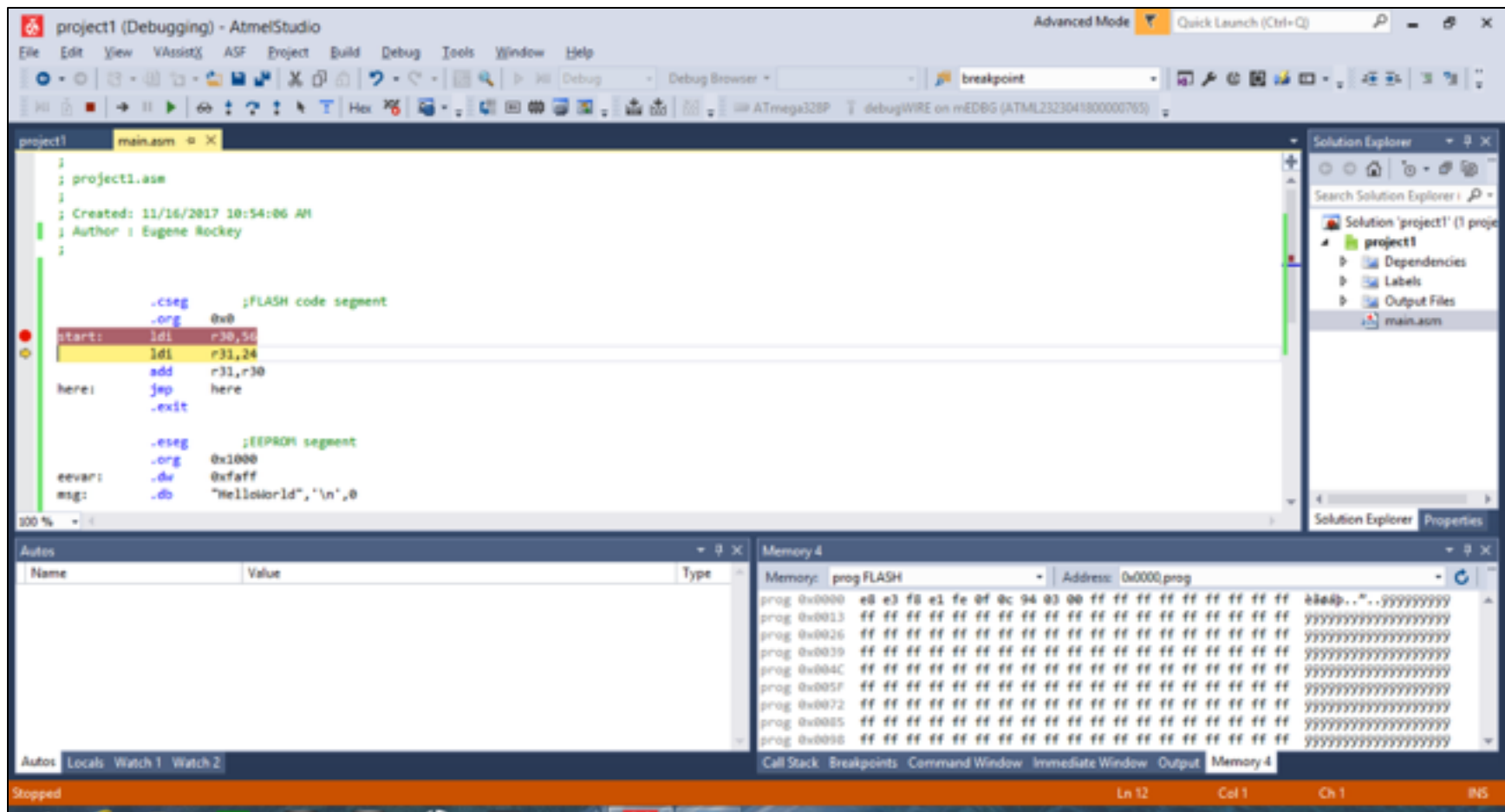
Return to the editor/code window or main.asm tab where the project1 assembly source code is entered.



Click the 'Debug' menu then 'Start Debugging and Break'. A 'Launch Failed' dialog will appear asking to enable the DWEN fuse, click YES.



Wait a few moments then the code should begin to execute but then break or stop to allow for further debugging options. A red circle with a yellow arrow will be in the breakpoint field pointing to the first line of code ready to execute when the user clicks the 'Step Into' debug command.



Users can place red breakpoint dots on different lines of code to stop execution at strategic points in the code and for analysis. Stop debugging by clicking the 'Stop Debugging' command either under the 'Debug' menu or in the DEBUG toolbar. Start debugging by clicking one of the 'Start Debugging' commands. Note that the code can also be executed in normal runtime by clicking the 'Start Without Debugging' command either under the 'Debug' menu

or in the DEBUG toolbar.

Return to program mode...

To return Atmel to program mode, you must disable the debug mode by clicking the 'Disable debugWIRE and Close' command under the 'Debug' menu. If the 'Disable debugWIRE and Close' label in the 'Debug' menu is not active and the system is still in debug mode then activate the 'Disable debugWIRE and Close' label by clicking the 'Start Debugging and Break' command then go back and click the 'Disable debugWIRE and Close' command.

VERY IMPORTANT: the DWEN fuse must be enabled to use the debugWIRE interface for in-circuit debugging of the ATmega328P(B) MCU and the DWEN fuse must be disabled to use the ISP interface for programming the ATmega328P(B) MCU.

Atmel Studio is in DEBUG mode (debugWIRE interface, DWEN fuse enabled) if you cannot program the ATmega328P(B) chip.

Atmel Studio is in PROGRAM mode (ISP interface, DWEN fuse disabled) if you cannot execute and or debug the ATmega328P(B) chip.

Switching between the PROGRAM and DEBUG modes with the board connected can be tricky at first and takes some practice to fully understand. It is basically a repetitive cycle as described here. Unfortunately, at this time, the program and debug modes cannot be active at the same time.

Option 2: The Simulator:

By using the built in simulator, the student will be able to run and debug assembly code instructions as long as they operate within the realm of the simulated CPU, memories, and registers, otherwise; an assembly error may occur. This option is very useful since it allows the student to program and debug code without the Xplained Mini board connected. In the project1 tab, click on 'Selected debugger/programmer' and select 'Simulator'. Go to the toolbar and click the 'Start Debugging' icon. Again, main.asm will begin execution at the first instruction but then stop to wait for the next debug command from the user. Again, notice that the line of targeted code has now changed color and is pointed to by a yellow arrow inside a red dot.

Important: if you're planning to debug using the Xplained Mini board, the code must be compiled/assembled and programmed into the target device first. This is because the code is actually executed in the target processor. And any future changes to the code means the code must again be assembled and programmed into the target device before debugging, otherwise; you may experience problems due to different code being in the Atmel editor versus the target device.

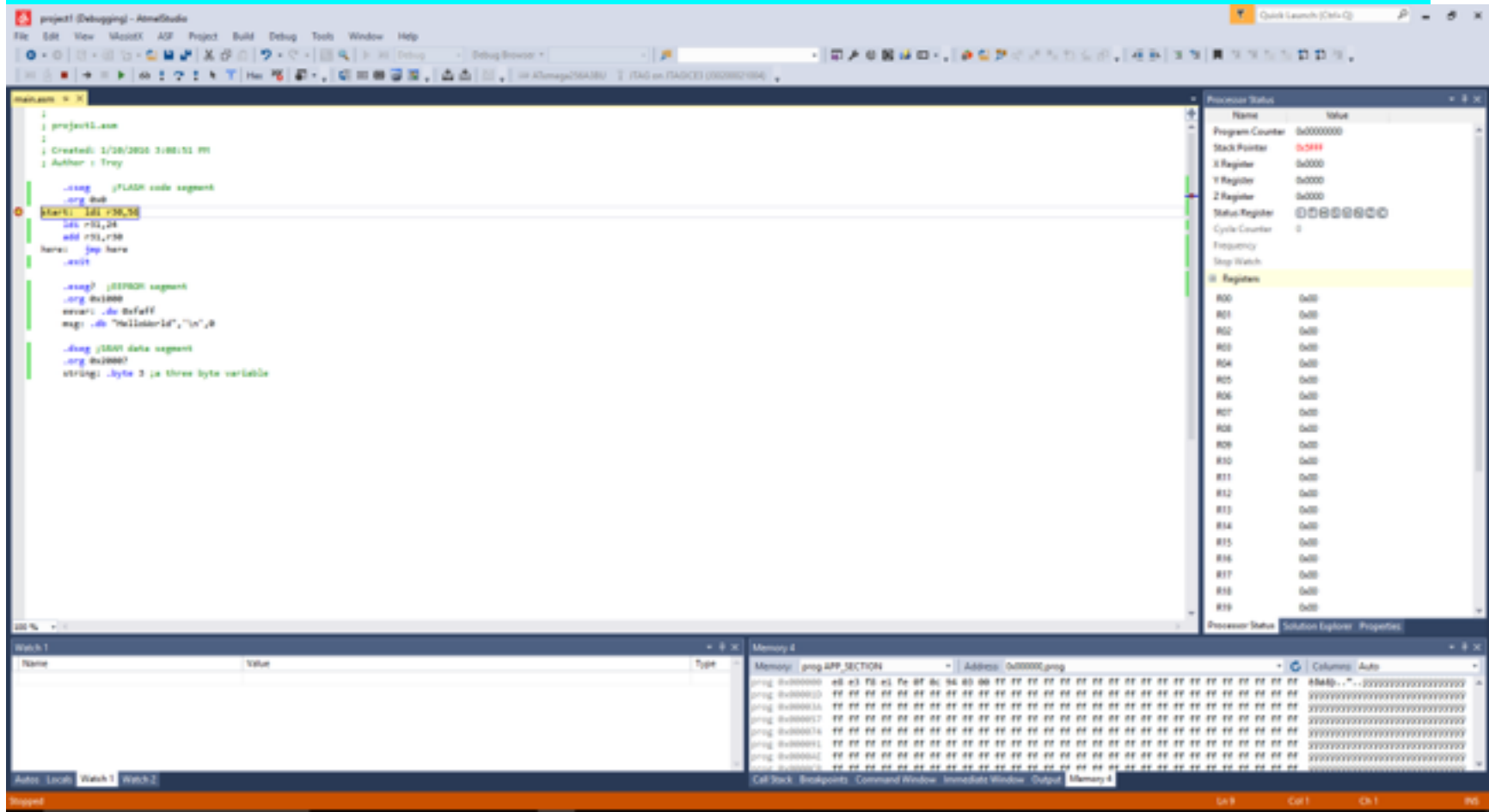
Continue to explore the many features of Atmel Studio to prepare yourself to work future labs and the final project. The [youtube.com](https://www.youtube.com) website has many helpful ATmega328P(B) and Atmel Studio videos.

On with the show...

Now, in debug mode, examine the PROCESSOR, WATCH and MEMORY views or windowpanes by clicking the 'Processor Status' icon in the toolbar. A window should appear that will give you critical information about the CPU's registers including the program counter. The 'Processor Status' pane is used to monitor the AVR CPU registers and their content as the code is executed line-by-line. The 'Watch' pane is used to monitor global variables and their values as the code is executed line-by-line. 'Watch' can be found by selecting

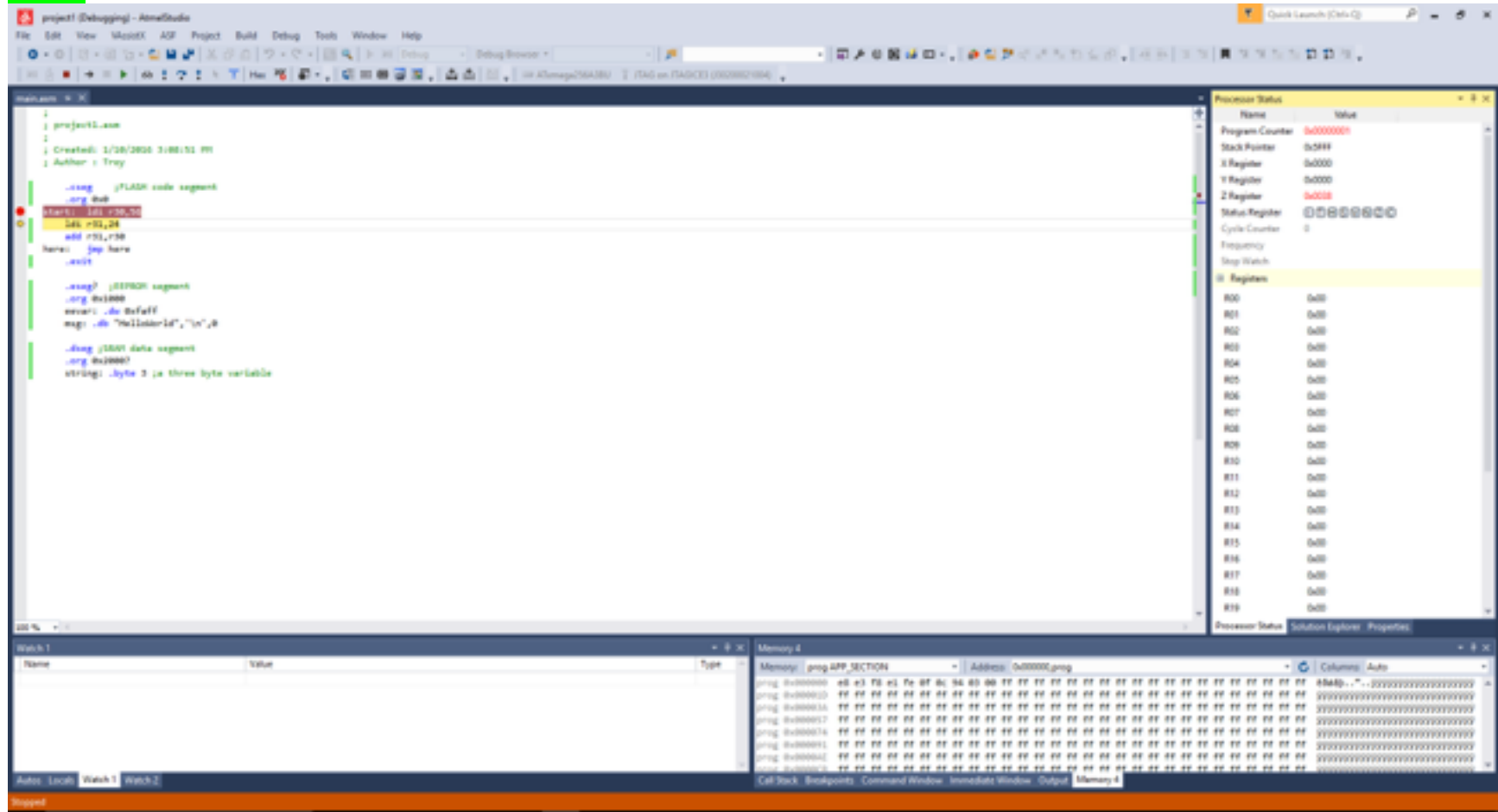
Debug/Windows/Watch/Watch1. The 'Memory' pane is used to view the data stored at addresses of various types of internal memory which includes FLASH, EEPROM, and SRAM.

If the 'Start Debugging', now 'Continue', icon is clicked a second time then the program will run normally, at least until another breakpoint or the program counter has reached the end of the program.



Click the 'Step Into' icon to execute the current line of code, this updates the program counter in the 'Pro-

the instruction `ldi r30,56` cause register r30 to contain the value 0x38 ? – answer and discuss this in your report.



Notice the program counter has been incremented to the address 0x00000001, why has the program counter been incremented from 0x00000000 to 0x00000001? In your report, discuss what the program counter has done here. Continue to 'Step Into' the entire program, line--by--line, until the last line of

code labeled here: is reached. After which, click on the program counter value and change it to 0 then click 'Step Into'. In your report, discuss what happens to the yellow arrow in the breakpoint field. In your report, research and discuss all the debug tools available in Atmel Studio.

Part 3: Explore AVR Assembler:

A combination of class notes and Atmel references are suggested for this portion of the project. In this part, you will program several instructions using various categories of the AVR 328P(B) instruction set in order to understand each instruction's function and purpose in a practical assembly program. Memory and I/O locations and the core processor's addressing modes and registers are demonstrated in conjunction with the instructions. AVR assembler directives are also included. For every unique assembly instruction encounter from this point to the end of the document, research and discuss its addressing mode in your formal report.

Begin with project1 main.asm already open in Atmel Studio. Enter and 'Step Into' the following lines of code to examine the behavior of math and logic and data transfer instructions. List the following code in your formal report under the SOFTWARE heading and fill in the 'operation-flags comment here' with the operation of the instruction and which status flags are affected by the instruction.

```
/*
 * project1.asm
 *
 * Created: 1/18/2018 5:18:16 PM
 * Author: Eugene Rockey
 */
//Arithmetic, logic, Data Transfer Instructions
    .cseg                ;FLASH code segment
    .org    0x0
```

```

start:    ldi    r26,0x00    ;operation-flags comment here
          ldi    r27,0x01    ;operation-flags comment here
          ldi    r30,56      ;operation-flags comment here
          ldi    r31,24      ;operation-flags comment here
          add    r31,r30      ;operation-flags comment here
          sub    r31,r30      ;operation-flags comment here
          and    r30,r31      ;operation-flags comment here
          mul    r30,r31      ;operation-flags comment here
          st     X,r30        ;operation-flags comment here
          clr    r30          ;operation-flags comment here
          ser    r31          ;operation-flags comment here
here:     jmp    here        ;operation-flags comment here
          .exit

```

After building and stepping through the code, examine the X register and the SRAM memory location 0x100. What value is stored at 0x100, how was it generated in the first place, how did it get stored there? Tip: Look at what specifically is happening with every line of instruction in this code, refer to the AVR Instruction Manual. And what changes, if any, did you notice about the status register bits, including their colors, while stepping through the code? In your report, discuss and answer all of these questions.


```

/*
* project1.asm
*
* Created: 1/18/2018 5:18:16 PM
* Author: Eugene Rockey
*/
//Conditional Branch, MCU instructions

        .cseg                ;FLASH code segment
        .org 0x0
start:   ldi    r26,0x00
        ldi    r27,0x01
        ldi    r30,56
        ldi    r31,24
        sub    r31,r30
        brmi   here          ;boolean test comment here
        st     X,r30
        nop                    ;operation-flags comment here
        clr    r30             ;operation-flags comment here
here:    breq   here          ;boolean test comment here
        .exit

```

After building and stepping through the code, discuss what has happened to the status register bits after the SUB instruction has executed. In your report, discuss what the BRMI instruction did because of the status register bits. In your report, discuss the NOP instruction and why BREQ instruction is branching to here:.

Enter the following lines of code to examine the behavior of bit instructions. List the following code in your formal report under the SOFTWARE heading and fill in the ‘ operation-flags comment here’ with the operation of the instruction and which status flags which can be affected by the instruction.

```
/*
 * project1.asm
 *
 * Created: 1/18/2018 5:18:16 PM
 * Author: Eugene Rockey
 */
//Bit Instructions

        .cseg                ;FLASH code segment
        .org    0x0
start:   ldi    r26,0x00
        ldi    r27,0x01
        ldi    r30,0xAC
        lsl    r30            ;operation-flags comment here
        lsr    r30            ;operation-flags comment here
        asr    r30            ;operation-flags comment here
        bset   2              ;operation-flags comment here
        bclr   2              ;operation-flags comment here
        brmi   here
        st     X+,r30         ;operation-flags comment here
        st     X+,r30         ;operation-flags comment here
        st     X+,r30         ;operation-flags comment here
here:    jmp    here
        .exit
```

Examine every line of execution while monitoring the processor registers and in particular the status register and SRAM location 0x100. In your report, discuss what specifically is happening with the LSL, LSR, ASR, BSET, and BCLR instructions. In your report, discuss why the BRMI did not branch and what value gets stored at SRAM location 0x100, 0x101, and 0x102. How did the value get generated in the first place? In your report, discuss what the instruction “ST X+,R30” did specifically.

Remember to rename your report as specified in the ‘readme’ document and proof read the .pdf file before submitting it to the instructor via Blackboard.