# CREATING, SIMULATING, AND EMULATING IN *ATMEL STUDIO*

## INTRODUCTION

The purpose of this document is to present a quick tutorial on how to create an AVR® assembler project within *Atmel Studio*, how to simulate an assembly program with the software debugger built into *Atmel Studio*, and also how to emulate an assembly program with the debugger/programmer built into your µPAD.

Additionally, the appendix of this document identifies how to do the aforementioned using the C programming language. When using C and working across a network, you will need to perform additional steps also laid out in the last page of this document.

For more information on *Atmel Studio*, visit the *Atmel Studio User Guide*.

## REQUIRED MATERIALS

- *GPIO_Output.asm*
- µPAD v2.0 with USB A/B connector cable

## SUPPLEMENTAL MATERIALS

- µPAD v2.0 Schematic

## PROCEDURE

**NOTE:** This tutorial assumes that you already have *Atmel Studio* installed, and that you have set your workspace folder to a known location. See the *Atmel Studio Installation Instructions* posted on our course website to learn how to do so.

1. Open *Atmel Studio*, and create a new project by navigating to **File → New → Project**.
2. Under *Installed*, select **Assembler**, and then **AVR Assembler Project** (see *Figure 1*).
3. Browse to a desired location for which to save the file (using the *Location* textbox), and save the file with a meaningful name (using the *Name* textbox). For this tutorial, we will call the project "GPIO_Output". Leave everything else default, and create the project by clicking the *OK* button.
4. In the *Device Selection* window that automatically opens, select the correct device and click *OK*. For the entirety of this semester, we will be using the ATxmega128A1U (see *Figure 2*).

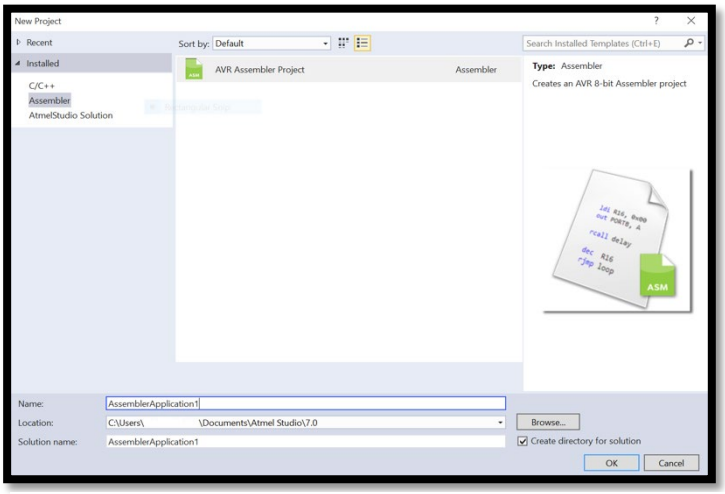


**Figure 1:** Choosing the project type

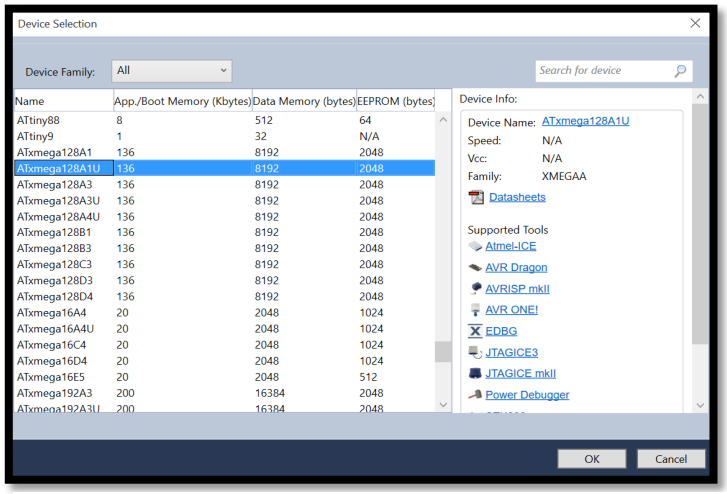


**Figure 2:** Choosing the device type

# CREATING, SIMULATING, AND EMULATING IN *ATMEL STUDIO*

You should now see a workspace similar to what is shown in *Figure 3*. From this point on, we will begin to use the code from the *GPIO_Output.asm* file mentioned above. This short program utilizes a GPIO port connected to the RGB LED package on your µPAD. A GPIO port can be defined in many capacities, though in the context of this tutorial, it suffices to relate a GPIO port with a group of physicals pins controlled by electrical signals, i.e., a low voltage signal corresponding to a binary '0', a higher voltage corresponding to a binary '1'.

5. Copy the code from the accompanying *GPIO_Output.asm* file to your main program. Save the main program by navigating to **File → Save main.asm**, or by pressing Ctrl+S (i.e., Ctrl and then S) on your keyboard. Alternatively, save all aspects of the project by navigating to **File → Save All**, or by pressing Ctrl+Shift+S on your keyboard.
6. Build (and compile) the project solution by navigating to **Build → Build Solution** (or by using the function key F7). If your code has no errors, i.e., if it was copied correctly, the *Output* window at the bottom should include "Build succeeded."

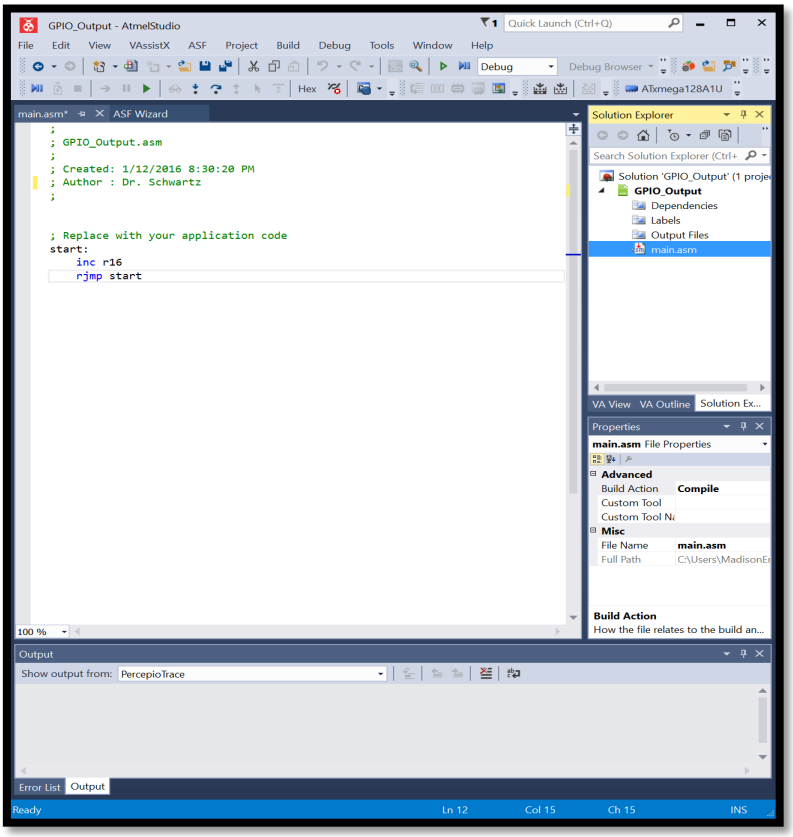


**Figure 3:** *Atmel Studio* application window, after creating project

**NOTE:** If your compiler detects any errors or warnings, you can double-click on an individual listing of either type and your cursor should be brought to the offending line (sometimes, your compiler cannot associate the error or warning with a specific line of code).

**NOTE:** If you would like to add line numbers in the code editor (which is highly recommended), navigate to **Tools → Options → Text Editor → All Languages** and select *Line Numbers*, under the *Settings* heading.

**NOTE:** Within the assembly file code editor, press Ctrl+Space on your keyboard to bring up a dialog box consisting of a list of assembly instructions for the currently chosen device. If any string of letters is typed into this dialog box, the listings shown will start with that specified string of letters. The internal program used for this function is sometimes known as *Intellisense*, *Autocomplete*, or *Auto Completion*.

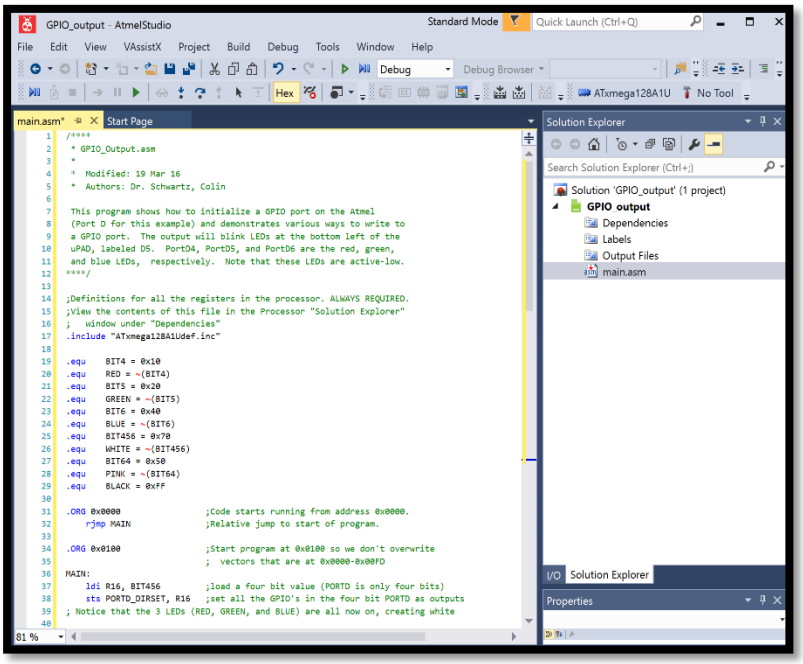


**Figure 4:** *Atmel Studio* application window, after copying contents of *GPIO_Output.asm*

# CREATING, SIMULATING, AND EMULATING IN *ATMEL STUDIO*

Before we can execute the program, we must select the appropriate debugging tool in *Atmel Studio*. For simulation, we will select the software simulator built into *Atmel Studio*, and for emulation, we will select the programmer/debugger built into your µPAD.

7. To perform a simulation of the program, we must first select the simulator debugging tool. Click the target selector icon (listed *No Tool* by default), as pointed to by the red arrow in *Figure 5*. When you select this icon, the project *Tool* window shown in *Figure 6* will appear. (Alternatively, the project *Tool* window can be opened by navigating to **Project → *<Project_Name>* Properties**, and then selecting the *Tool* option.) Under *Selected debugger/programmer*, use the dropdown menu to select *Simulator*. The target selector icon should now display *Simulator* (as also shown in *Figure 6*). Finalize these changes by saving the project (**File → Save All**), and then close the project properties window.

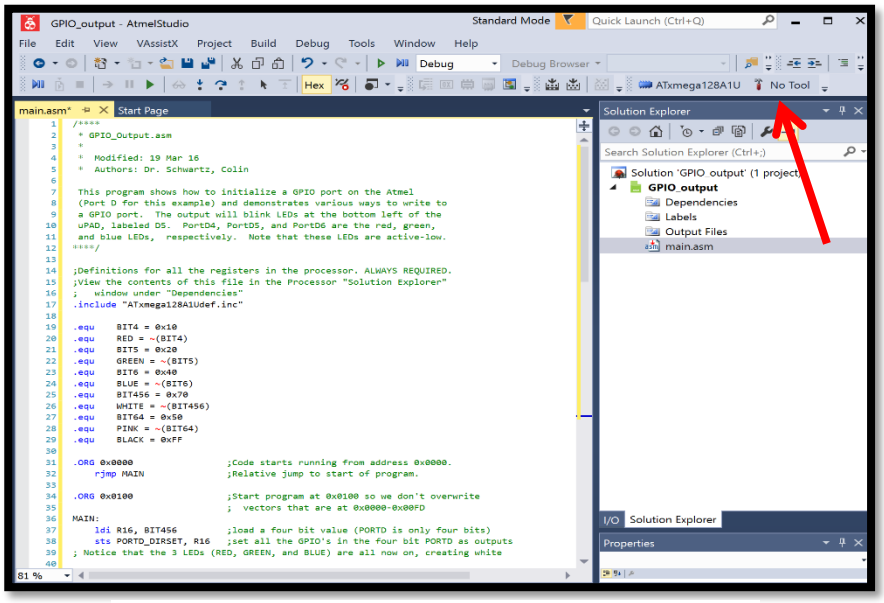


**Figure 5:** Target selector icon

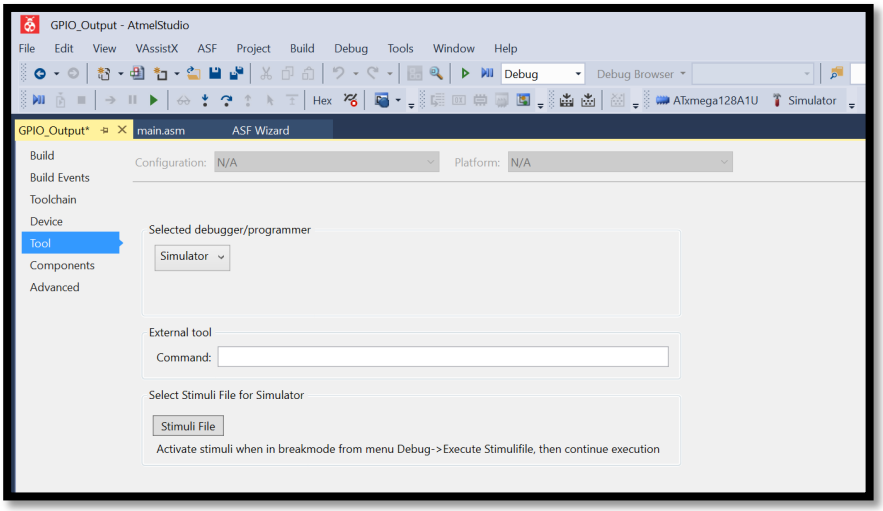


**Figure 6:** Project *Tool* window

# CREATING, SIMULATING, AND EMULATING IN *ATMEL STUDIO*

Before executing the program, we will place a breakpoint. Breakpoints are used to intentionally halt your program at a specific point of execution for debugging purposes, and are optional. A given breakpoint will halt your program immediately *before* executing the instruction specified at the breakpoint.

8. Place a breakpoint in your code editor on the first assembly instruction within your *main* program, i.e., `ldi R16, BIT456`, by using your mouse to click the gray pane to the far left of the instruction. A red dot will appear next to this specified instruction, as shown in *Figure 7*.

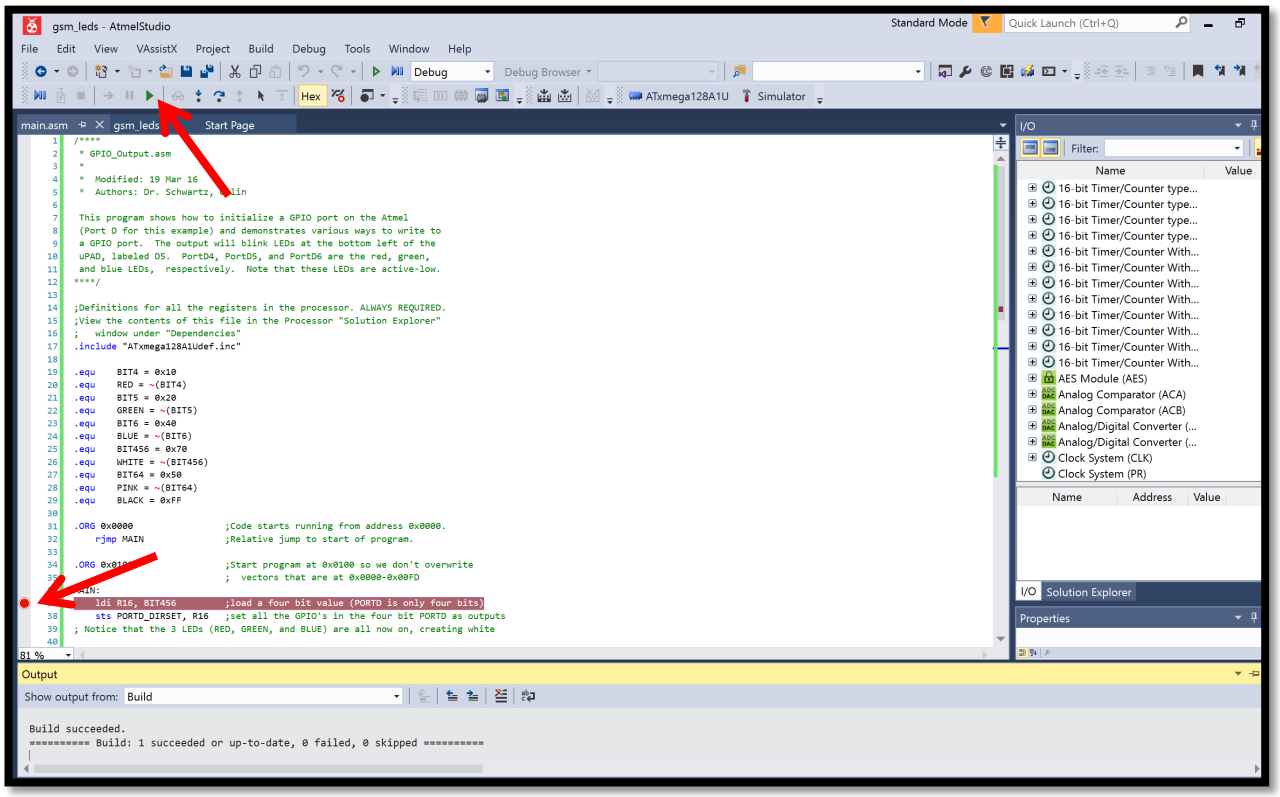


**Figure 7:** Breakpoint at first instruction of the given program, along with the debug Continue icon.

To start the program simulation, i.e., to start debugging, you can either select the *Start Debugging* icon (also known as the *Continue* icon) specified by the green arrow near the top of the *Atmel Studio* window (see the topmost red arrow in the *Figure 7*), or navigate to **Debug → Continue,** or even simply press **F5** on your keyboard.

9. Start debugging the main program. The program should stop execution at the breakpoint you placed, and the specified line of code should be highlighted yellow, following a yellow arrow in the gray pane where the breakpoint resides. (The yellow arrow, along with the yellow highlight, indicates the next instruction to be executed.)

While debugging, you can view any of the processor's registers, I/O ports, memory locations, etc. To do so is extremely helpful when writing any embedded software. Since we are using the I/O port *PORTD* in this example program, in addition to internal GPIO registers, we will explore the *I/O* and *Processor Status* views of the debugger.

10. Open the *I/O* view by navigating to **Debug → Windows → I/O**, as shown in *Figure 8*. Open the *Processor Status* view by navigating to **Debug → Windows → Processor Status**.

# CREATING, SIMULATING, AND EMULATING IN *ATMEL STUDIO*

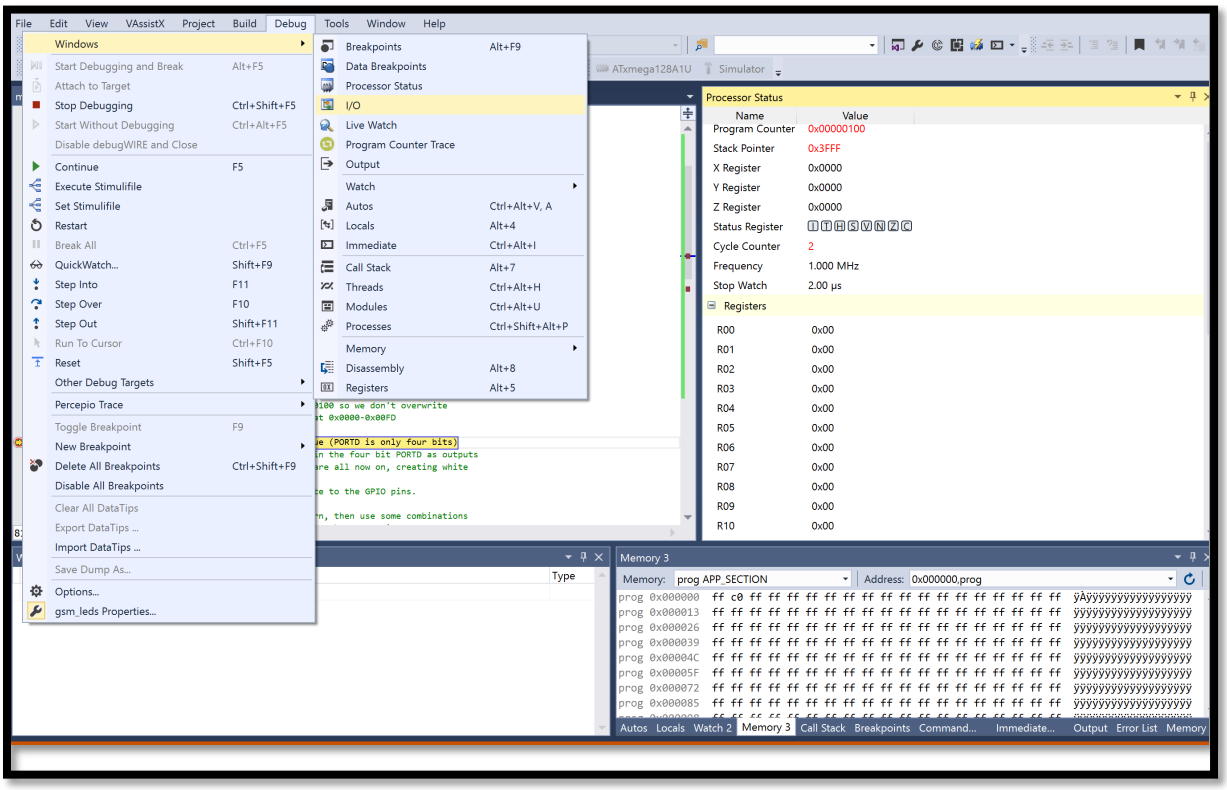


**Figure 8:** Selecting the I/O Debug view

**NOTE:** When simulating, if you would like the simulated clock frequency to match the clock frequency of the actual µPAD board (if it does not already match), i.e., the actual clock speed when you emulate, select the value listed next to *Frequency* in the *Processor Status* window, and enter the oscillator frequency that you have chosen for your device (e.g., 2.000 MHz, the default ATXMEGA128A1U clock frequency).

Now, we will utilize the I/O view to view all of registers within PORTD, starting at the point of execution specified by our chosen breakpoint.

11. Within the I/O view, filter for and select *I/O Port Configuration (PORTD)*. (To search for this, you may type something as simple as "portd" in the *Filter* textbox, as shown in *Figure 9*.) After selecting the correct port configuration, you will be able to view all of the registers associated with PORTD.

To execute the next instruction within your program, you can click the *Step Into* icon (as pointed to by the leftmost red arrow in *Figure 9*), or press **F11** on your keyboard. (You may also step through each instruction by navigating to **Debug → Step Into**).

12. Step through the program code, identifying changes that occur to the registers within PORTD, as well as to the microcontroller's internal GPIO registers.

**NOTE:** Two other useful debug stepping features are *Step Over* and *Step Out*, as pointed to by the second-rightmost and rightmost red arrows in *Figure 9*, respectively. *Step Over* will always execute the next instruction in the current procedure frame as a single unit, i.e., if the next instruction to be executed consists of a procedure call, the entire procedure will be executed in a single step. *Step Out* executes the remaining lines of a function in which the current execution point lies.

# CREATING, SIMULATING, AND EMULATING IN *ATMEL STUDIO*

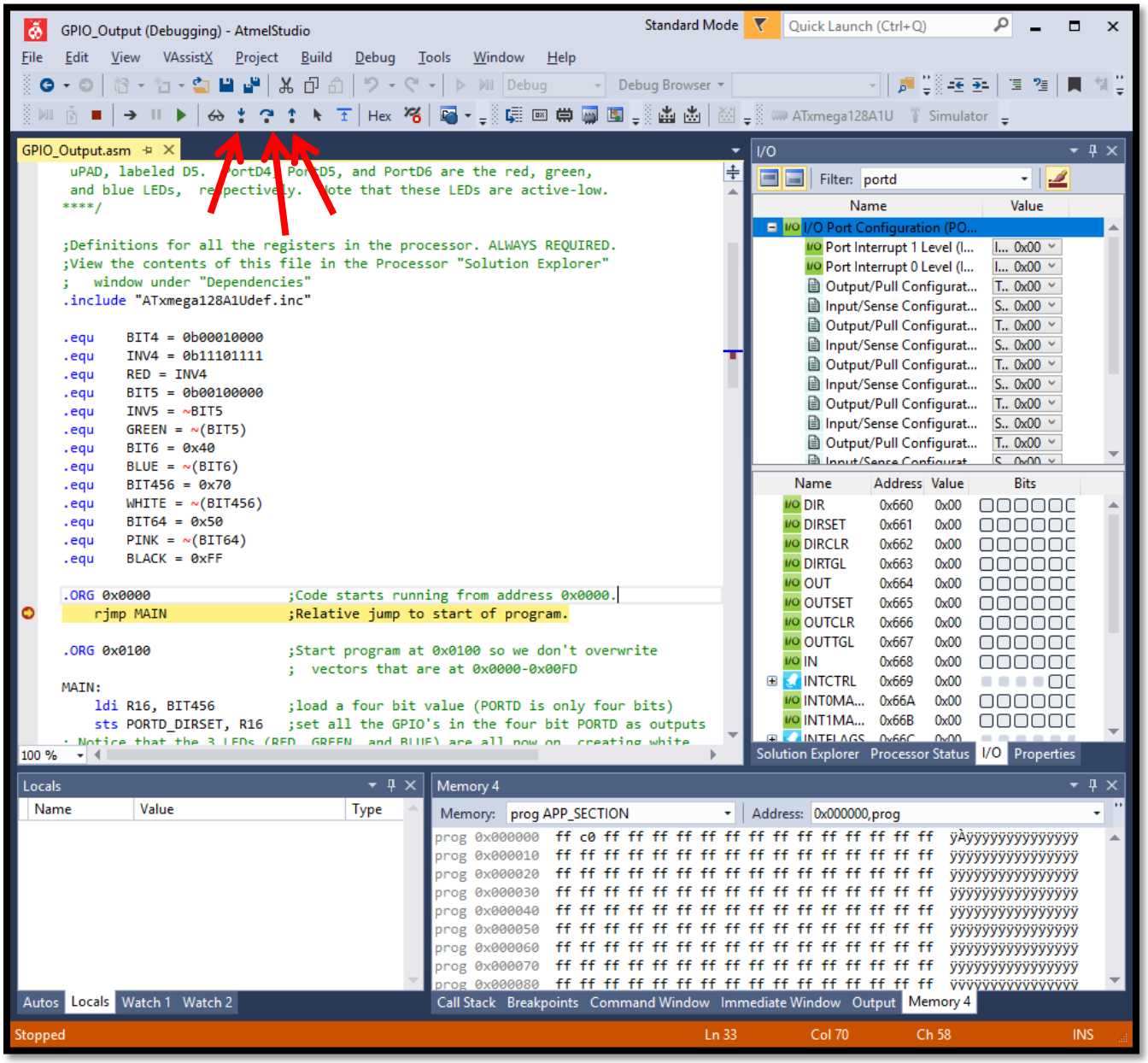


**Figure 9:** *Step Into, Step Over, and Step Out* debug icons

We will now stop debugging and begin to emulate the program on the µPAD. To stop debugging, you can navigate to **Debug → Stop Debugging,** or click on the *Stop Debugging* icon, i.e., the red square, in the toolbar.

13. Connect the µPAD to your computer with your USB A/B connector cable. Select the on-board Atmel **E**mbedded **Debug**ger (**EDBG**) as the *Selected debugger/programmer*, within the project *Tool* menu, as done in Step 7. Also verify that the *Interface* is chosen to be *PDI* (as shown in *Figure 10*). Finalize these changes by saving the project.

**NOTE:** The PDI clock frequency is only representative of the programmer/debugger, and need not be the same value as the clock frequency of the processor. It is recommended to *not* change the default value.

# CREATING, SIMULATING, AND EMULATING IN *ATMEL STUDIO*

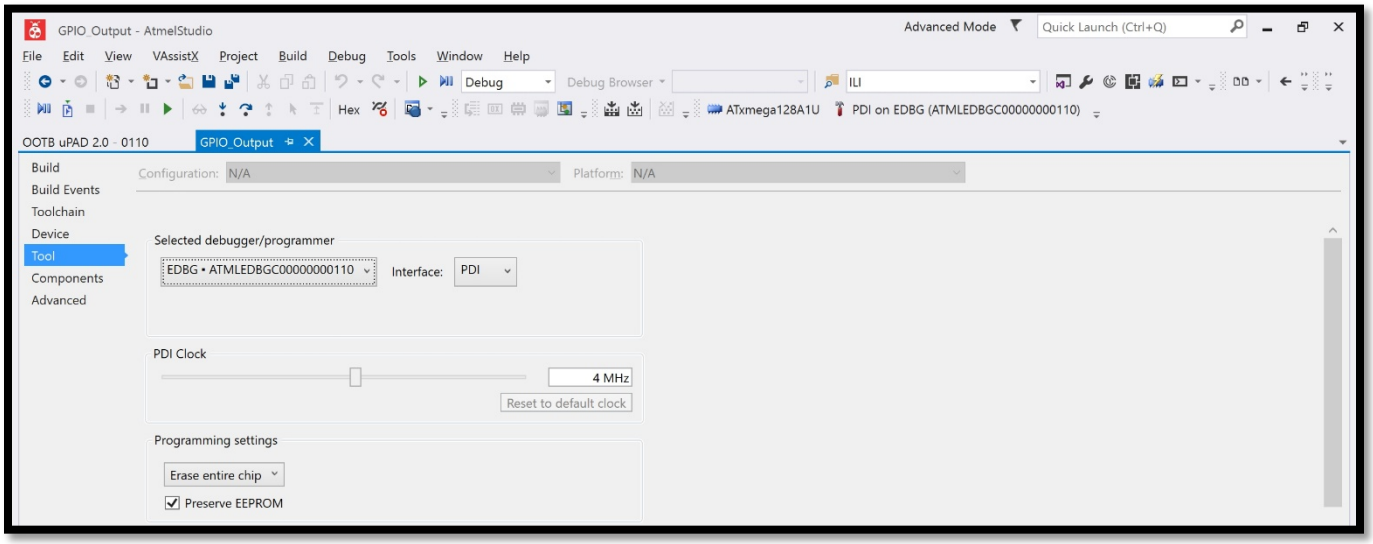


**Figure 10:** Selecting the on-board µPAD debugger

14. If the EDBG device does not appear as shown in Figure 10, there are a few possiblities to correct this problem. Try the first one below; if this does not work then try the second one.
    - If you are CONVINCED that you properly followed the install directions (and the previous tweet did not fix it), uninstall it, turn off your antivirus, reinstall it, & turn on antivirus.
    - Use the Device Manager to remove some extra drivers that altered the USB function through Atmel Studio. The procedure can be found at https://www.microchip.com/forums/m1087470.aspx.
15. Repeat steps 9-12. The code will now be executing in hardware rather than in software; once more, this is known as emulation. Upon stepping through the program, you should see the RGB LED package at the bottom left of your µPAD (labeled D4) reflect the changes made to PORTD within the program. These LEDs are connected to PORTD of your processor, as shown in the µPAD v2.0 Schematic.

# CREATING, SIMULATING, AND EMULATING IN *ATMEL STUDIO*

## APPENDIX A: SPECIAL CONSIDERATIONS FOR PROJECTS USING C

*Atmel Studio* cannot work accoss networks without using a network drive. Below are instructions on how to create a network drive, for Windows 10 and versions prior.

### Setting Up A Network Drive For *Atmel Studio*

**Windows 10**:

1. Type **Windows-E** (i.e., hold down the windows key and then type E) to open and Windows Exporer
2. In Windows Explorer, select **This PC**
3. Select **Computer → Map network drive**
4. Select a drive letter, e.g., **Z:**
5. Put the path to the folder that you want to use, e.g., \\mil.ufl.edu\tebow\3744\labs\
6. Select **Reconnect at sign-in**
7. Select **Finish**

**Pre-Windows 10**:

1. Type **Windows-E** (i.e., hold down the windows key and then type E) to open and Windows Exporer
2. In Windows Explorer, select **This PC**
3. Select **Map network drive**
4. Select a drive letter, e.g., **Z:**
5. Put the path to the folder that you want to use, e.g., \\mil.ufl.edu\tebow\3744\labs\
6. Select **Reconnect at logon**
7. Select **Finish**

**Procedure to Create, Simulate, and Emulate in C**

1. Open *Atmel Studio,* and create a new project by navigating to **File → New → Project**.
2. Under *Installed*, under the **C/C++** subheading, select **GCC C Executable Project**.
3. Find the path to the proper location with the *Location* textbox. If necessary, put in the network drive (e.g., Z:), and then the correct folder name.
4. Create the project by clicking *OK*, and perform everything else as specified in the above tutorial. The only new limitation might be that to be able to completely step through a C program, the compiler's optimization level may need to be changed. (Optimization allows your compiler to interpret your written C code and attempt to generate more efficient assembly/machine code, i.e., some C statements written in your program might be removed by the compiler if any level of optimization is enabled.)
    - To turn off optimization, first navigate to **Project → <*Project_Name*> Properties**. Within the project properties window, select the **Toolchain** subheading. Under **AVR/GNU C Compiler** in the *Toolchain*, select **Optimization**. Within the *Optimization* window, select the drop-down box for *Optimization Level,* and choose *None (-O0)*. In newer versions of *Atmel Studio*, there is an *Optimize debugging experience (-Og)* option.
        - Don't forget to optimization back on (*Optimize (-O1)*, by default) when you are finished debugging.

**NOTE:** Details for each optimization level specified by the AVR/GNU C Compiler can be found in the ***Atmel Studio User Guide***.