CREATE, SIMULATE, DEBUG, AND RUN PROGRAMS IN *MICROCHIP (ATMEL) STUDIO*

Revision 0

INTRODUCTION

This tutorial covers the following:

- Creating an AVR® assembler project within *Microchip Studio*
- Simulating an assembly program with the software debugger built into *Microchip/Atmel Studio*
- Running an assembly program with the debugger/programmer built into your µPAD.

The appendix of this document identifies how to do the above with the C programming language. When using C across a network (which most of you will probably **NOT** do), you will need to perform the additional steps laid out in the last page of this document.

For more information on *Microchip/Atmel Studio*, visit the *Microchip/Atmel Studio 7 User Guide*.

KNOWN ISSUES

If a filename or path has a space or special symbol, Microchip Atmel Studio often will fail. Solution: Remove the space or special character.

REQUIRED MATERIALS

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

SUPPLEMENTAL MATERIALS

- µPAD v2.0 Schematic

PROCEDURE

This tutorial assumes that you already have *Microchip Studio* installed. If not, read the *Microchip Studio Installation Instructions*.

Creating the Project

1. Open *Microchip Studio* and create a new project via File → New → Project as shown in *Figure 1*.

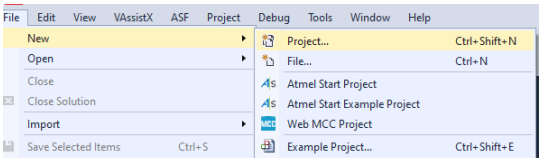

Figure 1: Navigating to Project Creation.

2. Under *Installed*, select Assembler, and then AVR Assembler Project as shown in *Figure 2*.

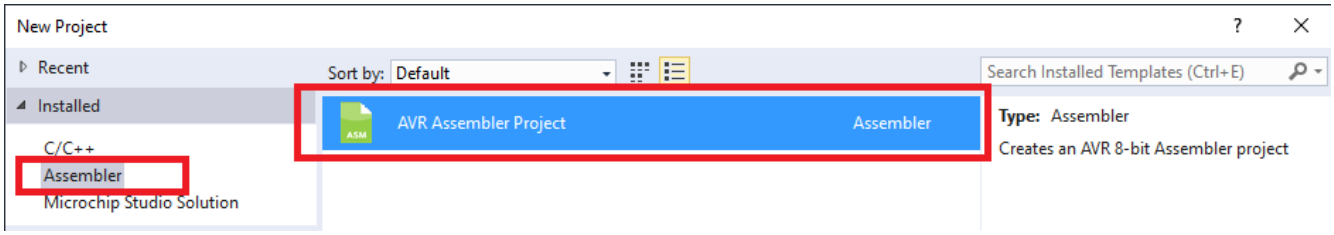

Figure 2: Selecting the Project Type.

3. Browse to a desired location for which to save the file (using the *Location* textbox) and save the file with a meaningful name as shown in *Figure 3*. For this tutorial, we will call the project "GPIO_Output".

Figure 3: Project Creation.

4. In the *Device Selection* window that automatically opens, select the correct device and click *OK*, as shown in Figure 4. For the entirety of this semester, we will be using the ATxmega128A1U. You should now see a workspace like what is shown in *Figure 5*.

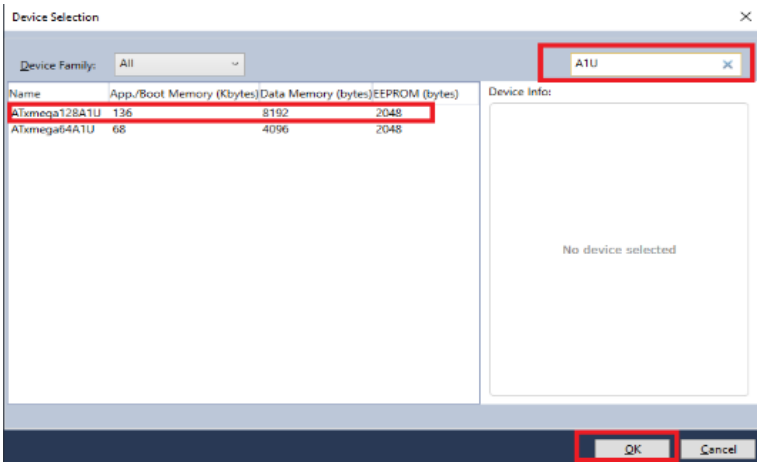

Figure 4: Selecting the Device Configuration.

Figure 5: Microchip Studio Workspace.

Creating and Simulating the Program

We will be using the code from the *GPIO_Output.asm* file. This program utilizes a GPIO (General-Purpose Input/Output) port connected to the RGB LED package on your µPAD. As shown in *Figure 6*, these LEDs are connected to Port D, the GPIO port we will be utilizing throughout this program

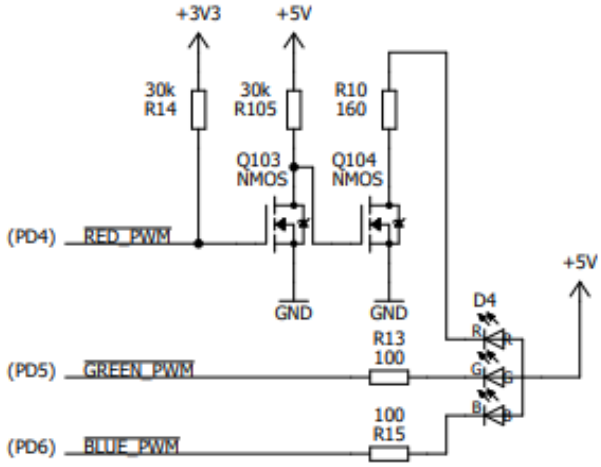

Figure 6: RGB LED package as shown on the uPad Schematic.

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'.

1. 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.
2. Build the project solution by navigating to **Build → Build Solution** (or by using the function key F7). If your code has no errors, the *Output* window at the bottom should include "Build succeeded" as shown in *Figure 7*.

```
Done building target "Build" in project "GPIO_Output.asmproj".
Done building project "GPIO_Output.asmproj".

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

Figure 7: Build Success.

NOTE: We suggest enabling line numbers in the code editor. Navigate to **Tools → Options → Text Editor → All Languages** and select *Line Numbers*, under the *Settings* heading, as shown in *Figure 8*.

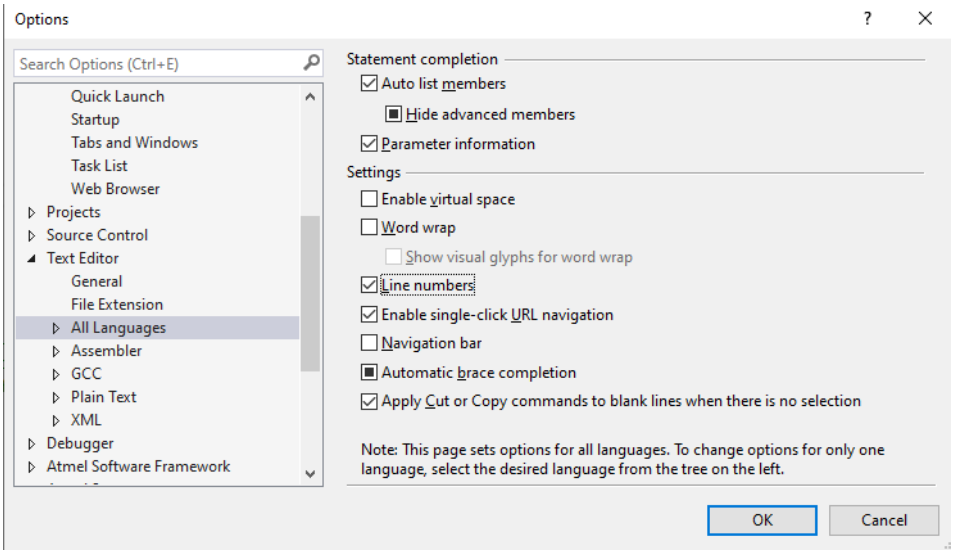

Figure 8: Enabling Line Numbers.

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: 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*.

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*.

3. Click the target selector icon as shown in *Figure 9*. This will open the project *Tool* window. You can also navigate here via *Project* → *<Project_Name> Properties…*

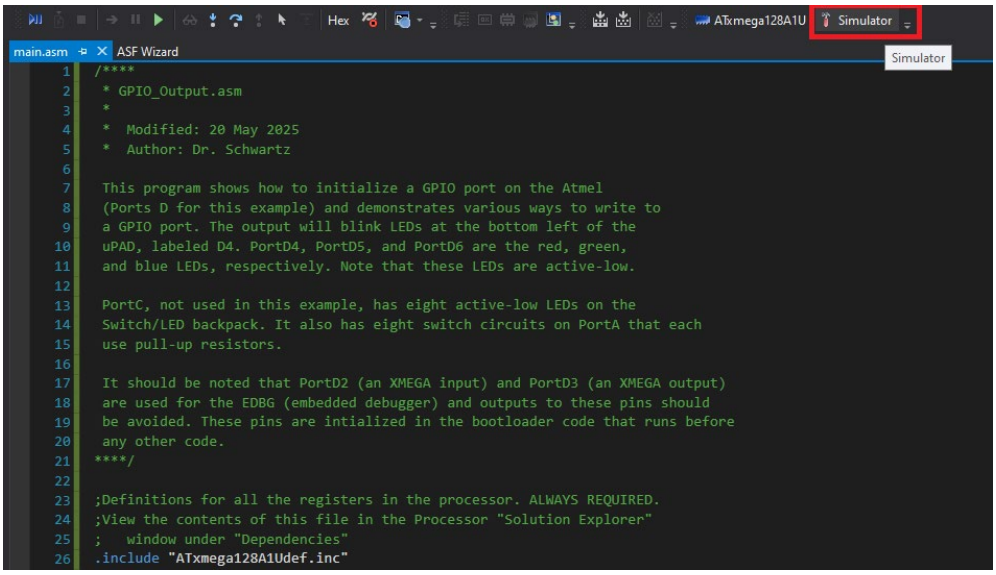

Figure 9: Target Selector.

4. Under *Selected debugger/programmer*, use the dropdown menu to select *Simulator* as shown in *Figure 10*. The target selector icon should now display *Simulator*. Finalize these changes by saving the project (**File → Save All**) and then close the project properties window.

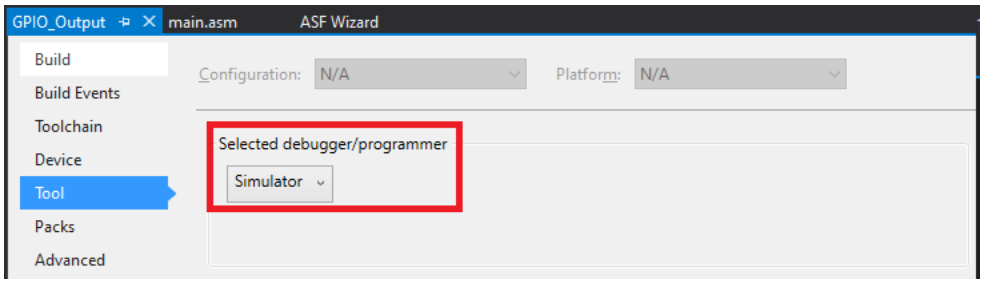

Figure 10: Selecting the Debugger.

Before executing the program, we will place a breakpoint. Breakpoints halt your program immediately *before* executing the instruction specified at the breakpoint.

Note: Later in the course, certain functions cannot be debugged via breakpoints and must be handled with alternative methods (like LED indication).

5. Place a breakpoint in your code editor on the first assembly instruction within your *main* program: `ldi R16, BIT456`, by using your mouse to click the pane to the far left of the instruction. A red dot will appear next to this specified instruction, as shown in *Figure 11*.

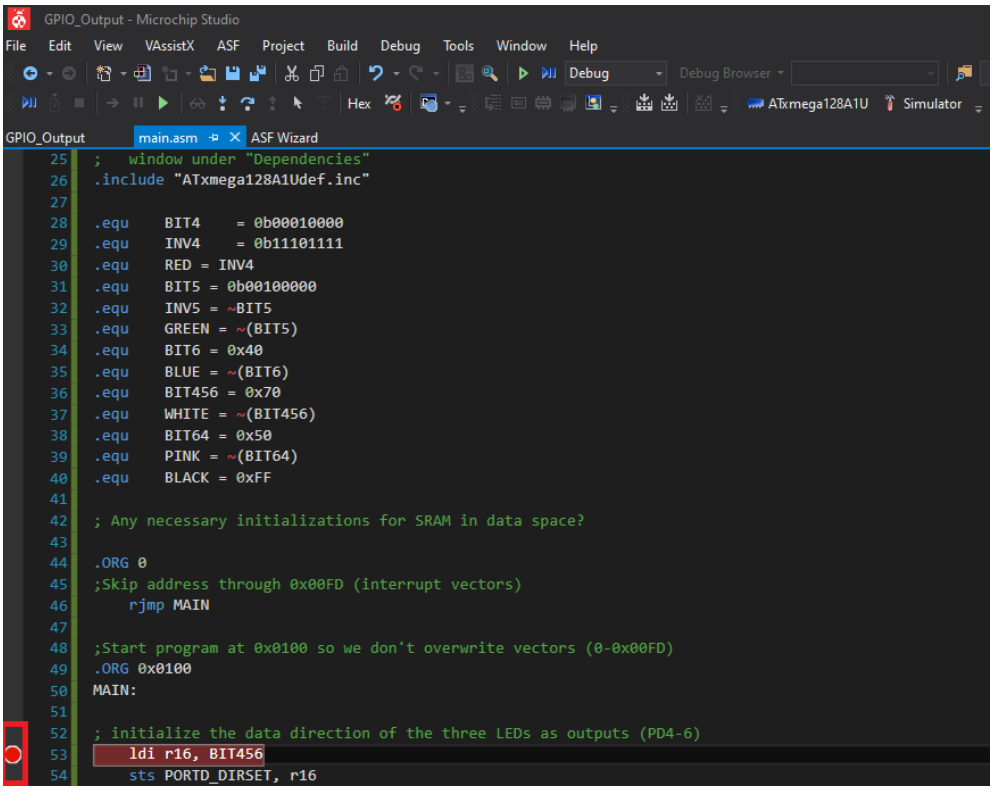

Figure 11: Breakpoint at first instruction of the given program, along with the debug Continue icon.

6. Start debugging the main program. To start debugging, select the *Start Debugging* icon shown in *Figure 12*, navigate to **Debug → Continue,** or simply press **F5** on your keyboard. The break pointed line of code will be highlighted, following a yellow arrow in the gray pane where the breakpoint resides. This indicates the next instruction to be executed.

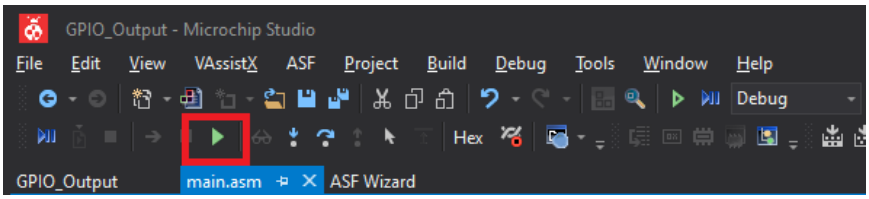

Figure 12: Starting Simulation.

7. While debugging, you can view any of the processor's registers, I/O ports, memory locations, etc.; a very powerful tool when writing embedded software. Open the *I/O* view by navigating to **Debug → Windows → I/O**, as shown in *Figure 13*.

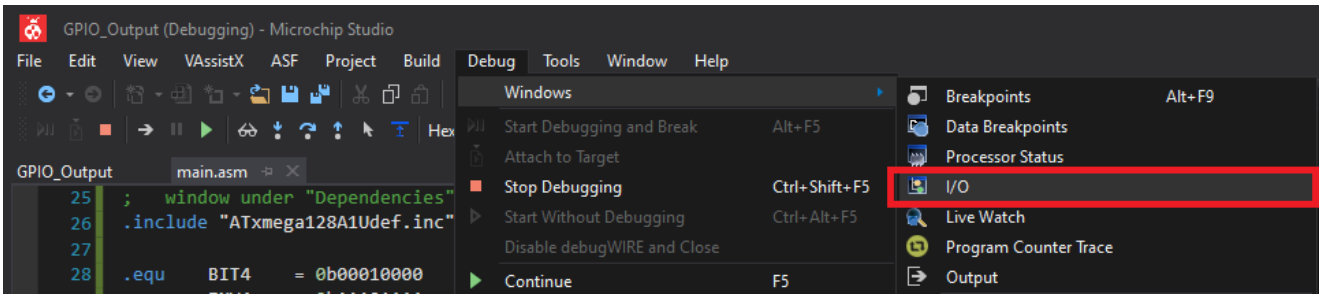

Figure 13: Opening the I/O view.

8. Open the *Processor Status* view by navigating to **Debug → Windows → Processor Status**. Your screen should now resemble *Figure 14*. It is likely available by default.

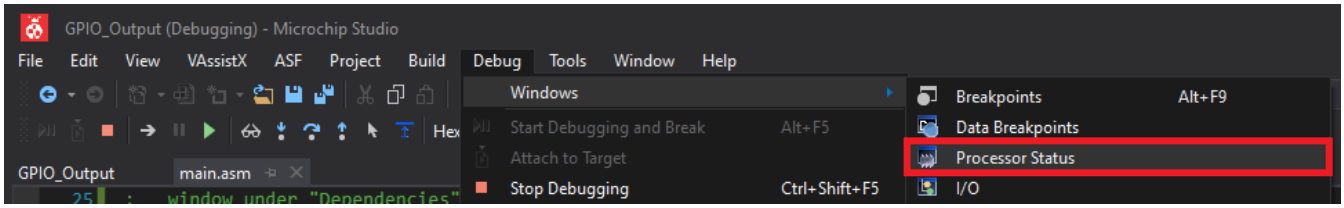

Figure 14: Opening the Processor Status Window.

9. *Figure 15* shows a standard debug view with the four generally most useful (and the default) windows shown: *Program File*, *Processor Status*, *Watch*, and *Memory*.

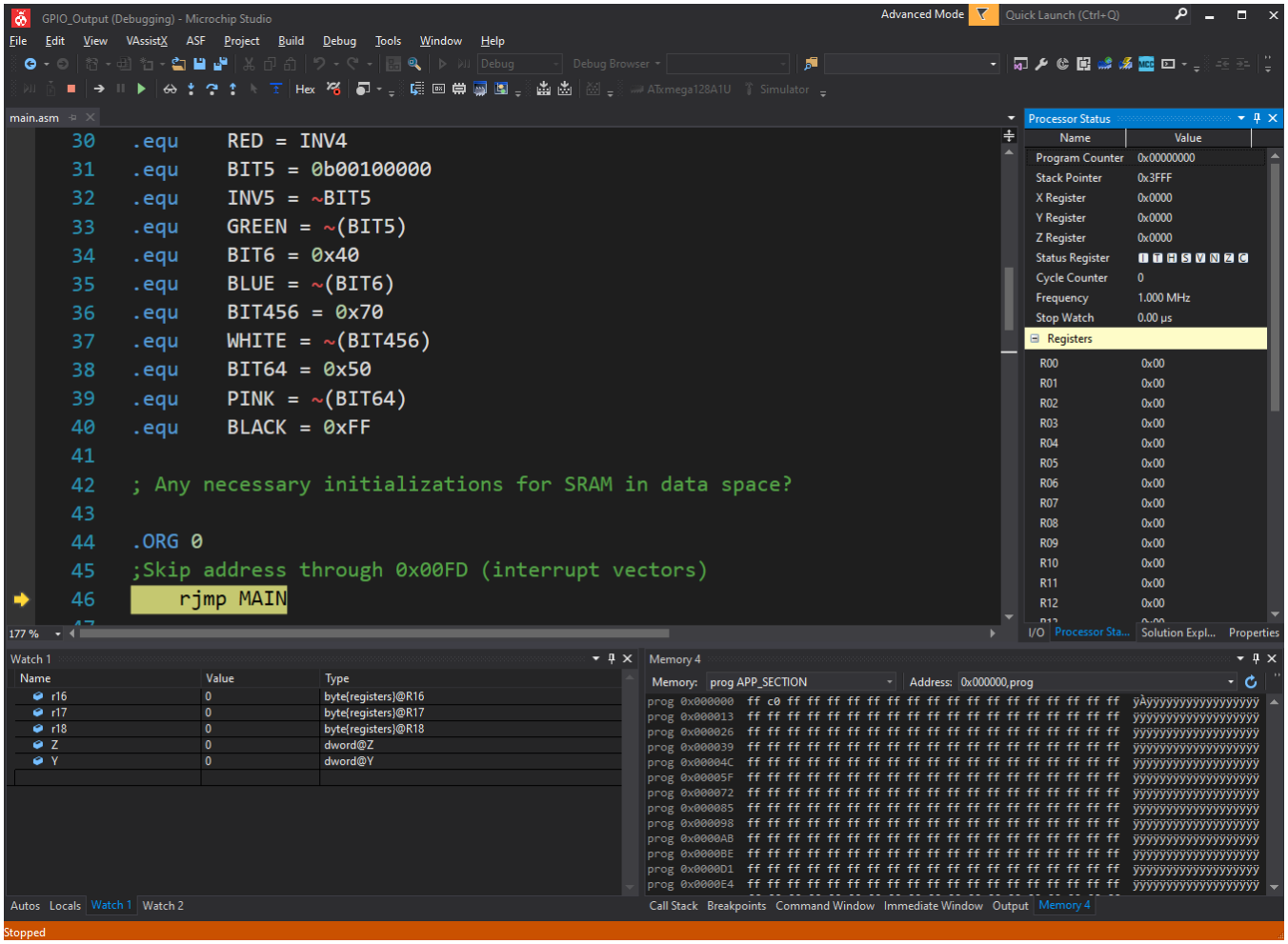

Figure 15: Standard Debug View.

NOTE: When simulating, if you would like the simulated clock frequency to match the clock frequency of the actual µPAD board, select the value listed next to *Frequency* in the *Processor Status* window, and enter the oscillator frequency that you have chosen for your device.

10. 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 16*.)

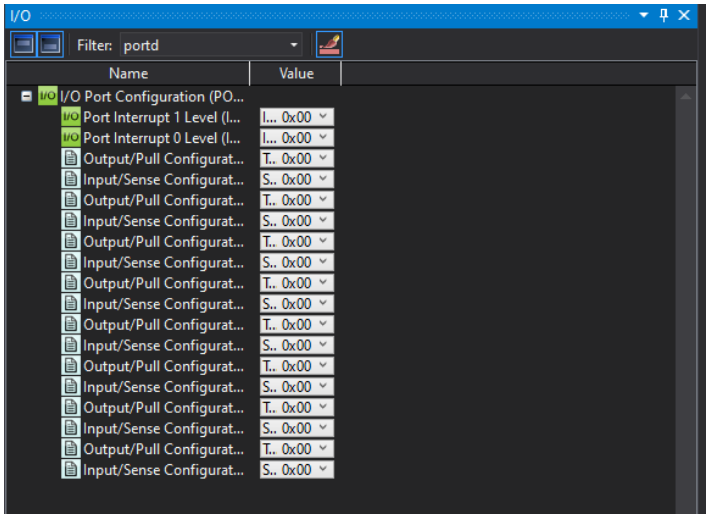

Figure 16: Filtering I/O View.

11. To execute the next instruction within your program, you can click the *Step Into* icon as shown in *Figure 17*. Step through the program code, identifying changes that occur to the registers within PORTD, as well as to the microcontroller's internal GPIO registers.

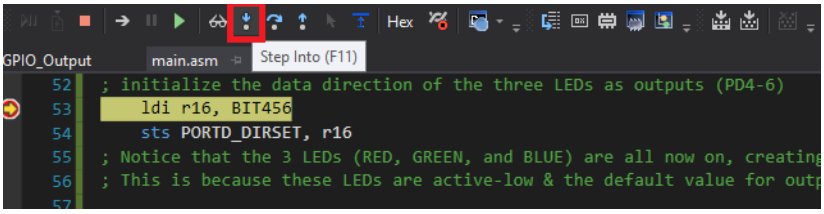

Figure 17: Step Controls for Program Debugging.

NOTE: Two other useful debug stepping features are *Step Over* and *Step Out*. *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.

We will now stop debugging and begin to running 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.

Running the Program in Hardware

1. 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 3 of the Simulation section. Also verify that the *Interface* is chosen to be *PDI* (as shown in *Figure 18*). Finalize these changes by saving the project.

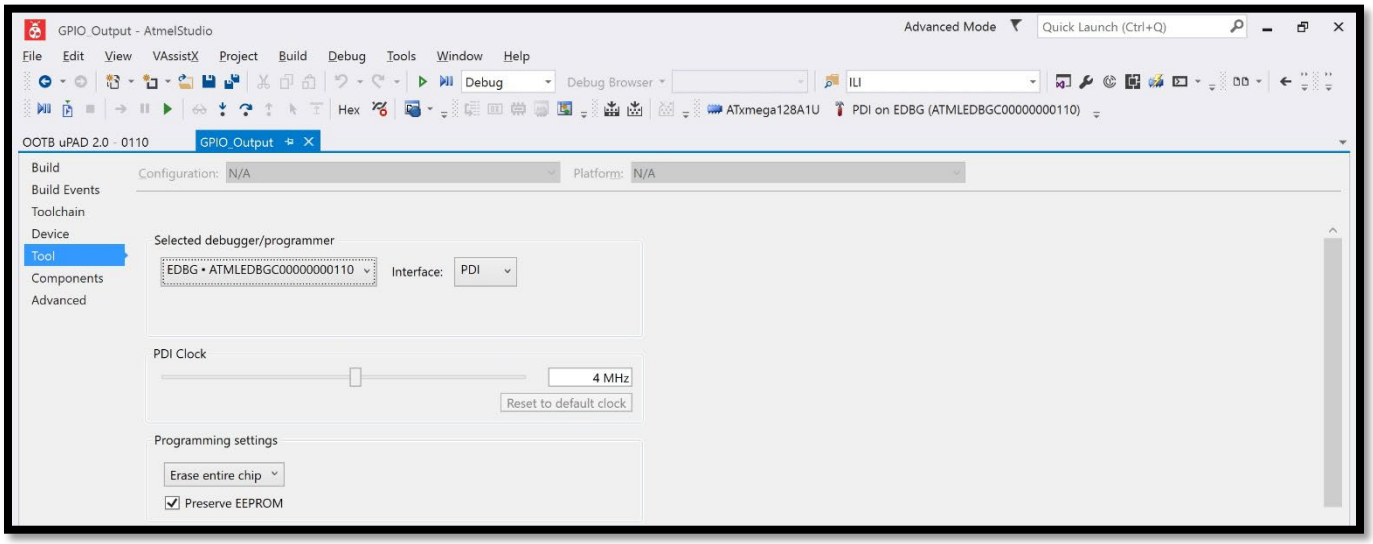

Figure 18: Selecting the on-board µPAD debugger.

NOTE: The PDI (Program and Debug Interface) clock frequency is only representative of the programmer/debugger, not the microprocessor. It is recommended to leave the default value.

2. If the EDBG device does not appear as shown in *Figure 18*, there are a few possiblities to correct this problem.
   - If you are CONVINCED that you properly followed the install directions, 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 Microchip Studio. The procedure can be found at https://www.microchip.com/forums/m1087470.aspx.
3. Repeat step 6 in the previous section. The code will now be executed on Microchip hardware rather than through software on your computer. 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.

NOTES ABOUT MEMORY WINDOW

- The memory window can show either internal Program Memory and internal RAM (Data Memory). To view Program Memory, select ***prog APP_SECTION*** immediately following the ***Memory:*** and select ***data INTERNAL_SRAM*** for viewing RAM in Data Memory. Immediately, following ***Address:*** put the first address that you would like to view (without changing the ***,prog*** or ***,data*** that will already be to the right of the address).
- Note that it is a window to look at individual bytes. Therefore, when viewing Program Memory, the specified Program Memory address has been shifted one bit to the left with the least significant bit of 0 corresponding to the least significant byte of the Program Memory word and the least significant bit of 1 corresponding to the most significant byte of the Program Memory word. This is because program memory is *word* addressed.
- The memory window does **NOT** work for external addresses! Do **NOT** depend on it.

APPENDIX A: SPECIAL CONSIDERATIONS FOR PROJECTS USING C

Microchip 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 *Microchip Studio*

Windows 10 and 11:

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\4744\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\4744\labs\
6. Select **Reconnect at logon**
7. Select **Finish**

Procedure to Create, Simulate, and Run Programs in C

1. Open *Microchip 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)*. There is now also a *Debugging* compiler option; for this you can leave it alone.

NOTE: Details for each optimization level specified by the AVR/GNU C Compiler can be found in the ***Microchip/Atmel Studio 7 User Guide***.