ECE 375 Lab 3: Prelab

Eric Prather (932580666) – Winter 2019 – Lab 013 (Th 1000-1200)

Prelab Questions

1. What are some differences between the debugging mode and run mode of the AVR simulator? What do you think are some benefits of each mode?

In computer science, debug and run mode are two distinct sets of preprocessor directives and execution environments. Code compiled for a debug build will often not have certain subroutines condensed and comes with additional reflection data, whereas the final release tends to be fully compressed and optimized. Debug builds additionally possess additional metadata related to the program code allowing for advanced execution behaviors such as break-pointing and member value inspection. In C, common debugging software which uses this metadata include the Visual Studio Profiler and the GDB command line utility. The assembly equivalent of this, for the purposes of this class, is the AVR simulator's debug and release modes. The AVR debug mode allows the management of the program through metadata which, at a core level, can make inferences about and modify program state using an external set of metadata (not unlike the .NET framework reflection library) superimposed on the simulated hardware events. However, despite these clear advantages of debug mode, no debug compilation can ever *perfectly* represent a release build, so testing should always be done in the latter as well to prepare best for the deployment to the hardware.

2. What are breakpoints, and why are they useful when you are simulating your code?

Breakpoints are metadata flags that are used to correlate source-code lines with compiled machine code execution states. A developer can insert a breakpoint in program code and then a debug compiler will automatically generate the necessary metadata for that breakpoint to be observed in the execution environment. The most common use of breakpoints is to pause programs mid-execution so programmers can inspect the state of the program's stack and all of the data at certain pre-defined points while searching for the source of an error.

3. Explain what the I/O View and Processor windows are used for. Can you provide input to the simulation via these windows?
The IO windows show the status of the simulated IO pins and their associated registers (e.g. DDRX, PORTX, PINX). The processor window shows the state of the processor components. IO can be provided by the IO view. The simplest way to do this is by clicking on the bit checkboxes,, but you can also type in values in the value fields. Not all registers are immediately exposed through the I/O window.
The processor window is similar, but instead of displaying special external or IO registers on the board, it displays those on the inside of the processor. You can similarly provide values to the registers like with the I/O window.

4. The ATmega128 microcontroller features three different types of memory: data memory, program memory, and EEPROM. Which of these memory types can you access by using the Memory window of the simulator?

(a) Data memory only

(b) Program memory only

(c) Data and program memory

(d) EEPROM only

(e) All three types

The memory window of the simulator shows **(e) all three types** of memory. This is despite the fact that the program memory is not accessible by the microcontroller itself, so it does not make sense to show it in the debugger. Program memory wouldn't be useful to a debugger anyway. EEPROM is also not pertinent. It is static over the course of the simulation. Nevertheless, the inclusion of all of these types of memory is valuable for the sake of learning.