Phil Lenox
Devlin Junker
Lab 4 Write-up

JTAG is a protocol for interfacing with the lowest levels of a processor, or digital logic device. JTAG has many uses, it can be used to load program into memory on embedded systems, where resources are too limited to be able to compile and build programs on the native environment. Similarly, JTAG is often used as the protocol for writing to FPGAs, CPLDs and other programmable digital logic. One of the neatest feature JTAG offers however is the ability to view every register on the CPU, and each memory address of a system at a given point in time. This is impossible, or at least difficult to the point where it isn't worth it to pursue, without the use of a debugger external to the system being debugged. JTAG allows the external controller, in this case our PC, to interrupt the execution of instruction on the processor, and on a very low level, direct how the program on the debugged system progresses. When the debugged system is halted, we are able to read all of the registers, including state registers. This is an invaluable tool for lower level programming, when an on board debugger is impossible, and sometimes higher level languages as well.

We had issues using openOCD and GDB to debug the program, however the program we were planning on testing was the following:

```
.section .text
._start: .global start
        .global main

        B main

main:
        MOV r0, #1 ; A = 1
        MOV r1, #2 ; B = 2
        MOV r2, #1 ; C = 1
        MOV r3, #3 ; D = 2
        MOV r4, #1 ; E = 1

        ADD r0, r0, r1
        ADD r0, r0, r2
        MUL r2, r3, r4
        SUB r0, r0, r2

        BX lr
```

When we tried to debug a program, we had a number of issues. We first tried using GDB on the Beaglebone to step through the program, and then examine the registers using the openocd telnet, however that did not work. We also compiled an arm-specific version of gdb, which we ran on our computer and tried connecting to the Beaglebone with. That was also a dead end, as we were unable to load and execute our program on the Beaglebone due to memory access errors. We had read online that these errors were caused by an older version of GDB, however our arm-specific gdb was version 7.2, so we don't think that that was the error.

However, after playing with the JTAG commands, if we had been able to open a program successfully, the *reg* command allows us to view and set register values at a given time. For our simple example program, we could modify the values of the registers before the arithmetic operations in our program. It would also be possible to force the processor to jump to other parts of the code, either by using the step command to direct where the program goes next, or by directly changing the PC register value, both of which have the strong likelihood of completely breaking the program. GDB can also be used to control the program flow, setting breakpoints and stepping through the lines of code, while openOCD can be used to halt the execution and read or set register values. Typically GDB can only view values at specific memory addresses, but with openOCD, the registers can be read too. By connecting GDB to openOCD, the openOCD commands can be executed with the GDB instance and provide an all in one debugging and analysis tool.