ECEN301 Embedded Systems Lab 6
JTAG Debugging

Daniel Eisen 300447549

October 6, 2020

1 Objectives

So far we have only used the JTAG port for upload our code to the ARM device. But the port/interface and associated statemachine and capabilities have a far greater use. In this lab we investigated the direct register read/write feature, the use of step by step and breakpoint debugging and observing the affect of the ARM(+thumb) instruction sets optional suffix's on operational register flags.

2 Methodology

Register Reading and Stepping

To begin with we explored how the JTAG interface could enable us to explore the devices register values during code execution. Loading a previous project, programming the board and entering debug mode, from here the contents of the stack pointer, program counter, and Status register (as well as all other core registers) could be readily seen and by stepping the program through 1 line at a time the changes that the programs instructions have on these values can be witnessed in "real-time". The program counter can be seen to step through the program addresses, as well and the ADDS instruction setting status flags.

Inline Assembly and Status Flags

To further explore the CPSR register, new project is loaded. With this the ability to directly include assembly code inside a C program is explore by making use of `asm(...)`.
For example, for the GE and N flags to be affected during exaction the following assembly code was used.

```
asm("SUBS r0, r0, r1");
asm("SADD16 r0, r0, r1");
```

This SADD16 does not affect the N, Z, C, V, or Q flags, but will set GE if the result is $\geq$.
While the SUBS instruction due to the S suffix will affect the prior flags, and will set N as the result will be negative.

Breakpoints

Finally debug enable the setting of breakpoints midway through execution if stepping does not to happen until a certain point in the codes runtime. This can be done either graphically in the text editor, or with the debag CLI. Once the breakpoint is reaching, execution halts and we are able to step through and observe variable and register values.

3 Questions

1. ***What do you think this code does?***
   These lines of code are part of the setup for the volatile variables.
   `str lr, [sp, #-4]!` is a STORE with immediate offset (pre-indexed), storing the contents of the **lr** register in 4 places before the stack pointer.

   `80000498: E24DD00C sub sp, sp, #0xc` This subtracts 0xC from the stored in the stack pointer and stores the result in back in sp.

2. ***What is the difference in behaviour between the ADDS and ADD instructions?***
   Both instructions are the basic Add without carry operation. The difference being the optional suffix 'S', with this present the ADDS instruction (and any instruction with the S suffix) updates the N,Z,X,V flags in the Current Program Status Register (CPSR).

3. ***List the debug capabilities of the JTAG port.***
   The use of the JTAG port in debugging are:

   Traditional external connection testing of the chip. This can serve as a lower barrier to entry method of formfilling a similar role of a fully blown logic analyser in debugging the communions/connection between the device and its peripherals.

   Additionally, due to the JTAG's interface being able to directly read/write a devices registers it can also be used as a means of programming the flash, and enabling step by step debugging and breakpoints via the TAP controller/FSM.