Assignment 2 Feedback

Winter 2014

Common Problems: Coding

- **Code Documentation**: Some documentation appears to be written by people who have never read someone else's code (as a learning experience)
- **Coding style**: See comment above
- **Magic numbers**: what do they mean?
- Parts 1 and 2 as a single file with no comment about what was different in the two (i.e. just code for Part 2).
- Results don't match the submitted code ☹
- Results are obviously wrong ☹ ☹

Common Problems: Technical

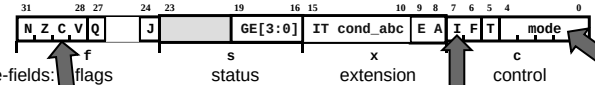
- **Timer configuration**: Not getting the intended/desired frequencies
  - System clock, timer clock, interrupt rate (10 ms)
- **Pre-scaling execution timer** to a much slower rate
  - E.G. processor @ 80 MHz but timer @ 10 KHz
    - 80 MHz: 80Mtick = 12.5 nanoseconds
    - 10 KHz: 10Ktick = 100 microseconds = 8,000 x 80MTick
      - Accuracy in measuring overhead?
      - ISR overhead ~40+ cycles: 1 10Ktick = ~ 200 x ISR
- "Volatile" used "everywhere"

Interrupt Function

- No one declared their ISR (part 2) as an interrupt function ☹ (in CCS: interrupt, in Keil: __irq)

  interrupt void myISRFunction ( ) { …
- Well … OK … not strictly needed for Cortex-M4 😐
- ARM processor has "modes" of operation
  - I hinted at this in the discussion of Assignment 1: SysTick timer requires privileged access …
- ARM processor bank-switches registers when changes mode (i.e. into IRQ mode to service interrupt)
  - Must change mode back to original mode when leaving ISR

ARMv7 Program Status Register (PSR)



Byte-fields: flags status extension control

- **Condition code flags**
  - N = **N**egative result from ALU
  - Z = **Z**ero result from ALU
  - C = ALU operation **C**arried out
  - V = ALU operation o**V**erflowed
- IT bits: Controls special conditional execution of Thumb2
- E bit: endianness of load/store
- A bit: disable special type of Abort (Data)
- GE bits: SIMD status (later)
- Interrupt Disable bits.
  - **I = 1: Disables the IRQ.**
  - **F** = 1: Disables the FIQ.
- T bit:
  - T = 0: Processor in **ARM state**
  - T = 1: Processor in **Thumb state**
- **Mode bits**
  - **Specify the processor mode**

Two PSR registers: Current PSR (**CPSR**) , Saved PSR (**SPSR**)

Register Set & Modes

SP : Stack Pointer
LR : Link Register
PC: Program Counter
CPSR: **Current** Program Status Register
SPSR: **Saved** Program Status Register

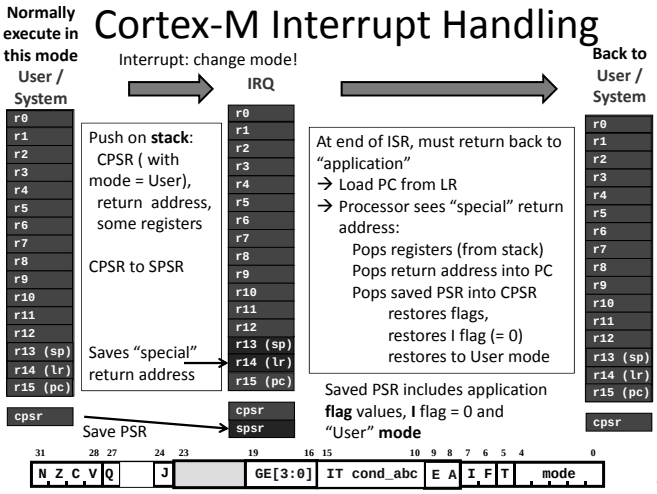
Current Visible Registers

User / System	Abort Mode	FIQ	IRQ	SVC	Undefined
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
r8	r8	r8	r8	r8	r8
r9	r9	r9	r9	r9	r9
r10	r10	r10	r10	r10	r10
r11	r11	r11	r11	r11	r11
r12	r12	r12	r12	r12	r12
r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)
r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)
r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)
Current PSR: cpsr	cpsr	cpsr	cpsr	cpsr	cpsr
	Saved PSR: spsr	spsr	spsr	spsr	spsr

Banked out Registers

NB: Cortex-M4 has some subtle differences

Cortex-M Interrupt Handling



Is the Cortex-M Style of Interrupt Handling "Common" ?

- Saving state on stack → YES (PSR, return address)
- "Special" return address → **NO**
- Most would include a special "return from interrupt" instruction
  - Restores PSR as well as return address
- In C: "interrupt" (or "__irq") is needed to tell the compiler to do anything special to turn a function into an ISR
  - E.G. "return from interrupt" instead of return from function