ECE 271 – Microcomputer Architecture and Applications
Lecture 16

Vince Weaver

http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

26 March 2019

Announcements

- Read Chapter 11
- Midterm graded
- Lab #7 is happening

Hand back Midterms

Lab #7 Notes

- Access the timer, in C

Interrupt Review

- Cortex-M has 255 interrupts. -1 to -15 built-in, 0-240 external
- When something triggers interrupt (traditionally pull a line low) stops execution and jumps to interrupt handler
- With vector interrupt handler, a vector each with an address for each handler, look up in table from interrupt number and jump
- First thing you need to do is save registers so we can use them. Cortex-M does this for you, saves R0,R1,R2,R3,

R12,PSR,LR,PC
(Note, saves on the ”MSP – Main Stack Pointer”. To confuse things there’s also a special ”PSP – Process stack pointer” but that’s possibly only used if you’re writing an OS)

- Then your code runs in a handler, which is much like a C function
- You may need to “ACK” the interrupt, let the hardware know you are handling things so it can stop asserting the IRQ line
- Do whatever you need to do

- Return. Can return just like a regular return (some architectures require a special return-from-irq instruction... not Cortex-M though)

  Cortex-M does weird stuff with Link Register – special value with `FFFF` in high bits that indicates we are returning from an IRQ handler and that the return value is on the stack (more info on this in the textbook/manual)
- Ideally the main code running on the processor doesn't even notice an interrupt happened

Setting up/enabling Interrupts

- Note this and SysTick described in Cortex-M4 Devices Generic User Guide DUI0553.pdf not in the STM32L4 manual
- Interrupt Set Enable Register – (ISER0–ISER7) note, this is like the BSRR register, 1 means enable, 0 means do nothing
- Interrupt Clear Enable Register (ICER0–ICER7)
- Setting/clearing. Bitmask, so 32-bits

```
word_offset=irq_num>>5;   // why?
bit_offset=irq_num&0x1f;  // why not % 32
NVIC->ISER[word_offset]=(1<<bit_offset);
```

- Assembly – note byte vs word addressing

```
// irq to enable in r0
ldr r4,=NVIC_BASE

lsr r1,r0,#5    // get word_offset into r1
lsl r1,r1,#2    // change to byte offset, mulx4
add r1,r1,#NVIC_ISER0

and r2,r0,#0x1f // get bit offset in r2
mov r3,#1
lsl r3,r3,r2

str r3,[r4,r1]
```

Setting Priority

- SHP (system handler priority)
- Byte array in the SCB (System Control Block)

```
SCB->SHP[(((uint8_t)irq)&0xf)-4]=(priority<<4)&0xff);
```

- For the external ones, there's the IP (interrupt priority register) in the NVIC structure.

```
NVIC->IP[irq]=(priority<<4)&0xff;
```

Global Interrupt Enable/Disable

- CPS (change processor state) instruction – pseudo instruction that sets the PRIMASK (priority mask) register
- CPSID i – disable interrupts
- CPSID f – disable fault handlers
- CPSIE i
- CPSID f
- Can also set priority mask manually to disable interrupts above a certain level. Need MSR instruction as it's a

special register

- The way to do this is the `CPSIE I` assembly language instruction.
- Can we do this in C? We'll have to use inline assembly.
  - On Keil, you can do this:

```
__asm("CPSIE␣i");
```

  - On Linux it will look like:

```
asm volatile ( "cpsie␣i" );
```

NMI – Non-maskable Interrupts

- An interrupt that cannot be stopped
- What are the useful for?
- Watchdog timers?
- Hacking, performance counters?