ECE 271 – Microcomputer Architecture and Applications Lecture 15

Vince Weaver

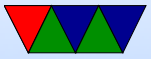
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

10 March 2022

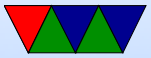
Announcements

- Read Chapter 11
- Midterm will be graded soon, not before break though



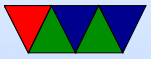
Lab #7 Preview

- Create an accurate delay function using a system timer and interrupts using C
- Blink an LED using this new delay function



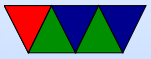
Simple Timer – Theory

- Load a value (count) into a register
- Start it counting
- Every clock cycle it decrements by 1
- When it hits 0 you are done
- If you have a 1MHz clock, how many times would you need to count for a 1ms delay?
- 1MHz is 1us, so 1000



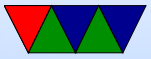
Complex Timers

- Don't worry, we'll get to much more advanced timer topics eventually



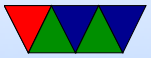
Checking for I/O

- How do you know when a timer count hits 0?
- How can you know if a joystick button is pressed?
- etc?
- What we've done so far is called polling



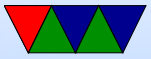
Polling

- To see if some sort of task is done, just regularly check the value
- Usually check it in a tight loop
- Downsides to polling
  - Wastes power/resources, constantly checking
  - Hard to do other things while waiting
  - Can check less frequently, but then latency to when you notice things changed
- Is there an alternative?



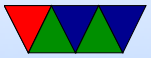
Interrupts

- When some sort of event happens, it "interrupts" what the CPU is doing
- Package analogy
  - Waiting for package
  - Can check porch every 30s. Makes it hard to get work done
  - Instead can wait for doorbell to ring. Drop what you're doing to get it, get the package, but then resume what you were doing

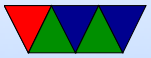


Types of Interrupts

- Internal to CPU / Exceptions
  - Timer
  - Illegal instruction
  - Illegal memory access
- Something your code does
  - System call
- Something external hardware does
  - Keyboard press
  - Network packet comes in

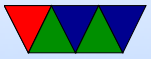


- Close lid of laptop
- Sound device wants more input



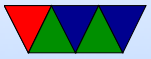
Interrupt Handling

- What happens when an interrupt comes in?
- Jumps to an "interrupt handler" (also known as an "interrupt service routine (ISR)' that takes over
- Problem: how do we get back to our original code?
- Problem: can the handler use any registers?
- Problem: can it save registers on the stack? (does the stack pointer have to be valid or have room on it?)



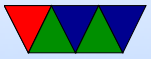
Interrupt Handling

- Cortex M does this differently from other ARM processors
- This throws me because I am used to doing this on Raspberry Pis
- In an ideal world an Operating System will abstract this away, as systems do thing differently



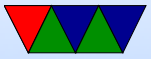
Interrupt Vectors / Routines

- Some systems have only a single interrupt handler
- Some systems interrupt jumps directly to fixed address (say 0xfffffffe) and starts executing code there
- Other systems have vectored interrupts, where there's a table of addresses, and depending which interrupt happened you look up the address in the table and jump to it
- Cortex-M uses multiple vectored interrupts



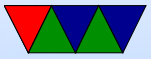
Cortex-M Interrupt Numbers

- On Cortex M interrupts are numbered -15 to 240
- -15 to -1 are internal processor interrupts, 0 to 240 are external
- -15 is Reset, (at address 4) which is called at boot or press black button
- -1 is SystemTick (used in lab)
- Various hardware are the upper ones



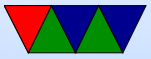
Interrupt Vectors

- Cortex-M has a set of vectors at bottom of memory (you can move it around with VTOR register)
- Add 15 to source, then multiply by 4 (Remember, 32-bits) then offset from 4 (to skip stack)
- Grab that value, and that's the address to jump to
- Address 0 is the value that gets put in the stack pointer at boot
- Address 4 through first 1k) are interrupt vectors



Interrupt Stacking/Unstacking

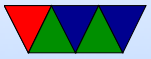
- Before jumping to handler, CPU saves R0,R1,R2,R3 and R12,LR,PSR,PC to the stack.
- Which stack?
  There are actually *two* stack pointers, the MSP (main stack pointer) and PSP (process stack pointer)
  I think for this class we can assume it's always going to the MSP
- This could be worse, non cortex-M ARM processors have like 7 different modes each with it's own set of banked



registers

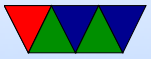
- At end, restores these all. Also clears the I flag
- Return by BX LR, the processor knows to do extra stuff if you are in interrupt handler mode

  How can it tell? Deep down it's actually putting a special magic value like 0xfffffffXX in LR and returning to that (it has bits to say which stack to use, etc) but that's not important for this class



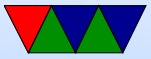
Interrupt Handler

- On Cortex-M can just be a plain C function (this isn't always true, other architectures require some assembly)



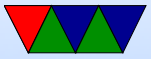
Interrupt Priority

- Can interrupt an interrupt. Why?
- Real time systems
- Non-maskable interrupt?



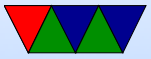
Enabling an Interrupt

- Setup handler
- Make sure vector points to it
- Enable the interrupt for the device you want in ISER
- Globally enable interrupts for the processor

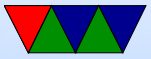


Handling an Interrupt

- Handler called.
- Save regs? Not on Cortex-M, does it for you
- Figure out what interrupt happened (Cortex-M a vector for each so not a problem?)
- "ACK" the interrupt, tell the hardware we are handling it
- Do whatever we want. Should this take a while?
- Exit interrupt and return to where we were. Cortex-M will clear interrupt flag and restore regs from special

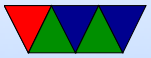


stack for is

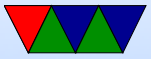


Programming the Interrupt Hardware

- Cortex-M has Nested Vectored Interrupt Controller (NVIC)
- 256 interrupts
- Each has 6 bits (spread across different regs)
  - Enable (ISER0..ISER7)
  - Disable (ICER0..ICER7)
  - Pending (ISPR)
  - Un-pending (ICPR)
  - Active (IABR)

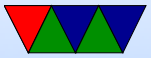


- SW trigger (STIR)



Lab#7 Preview

- Setup a timer that operates at 1ms (1000Hz)
- Setup an interrupt handler that runs at 1000Hz, that decrements a value you set down to zero.
- Create a delay function that uses this timer to do "exact" timing.
  Set the value to 1000, then spin waiting for it to be zero.
- Use it to blink LED exactly
- Measure this with oscilloscope (know how to, taking 214?)



Apple II Timer/Interrupt Demo