ECE 271 – Microcomputer Architecture and Applications Lecture 25

Vince Weaver

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

vincent.weaver@maine.edu

30 April 2019

Announcements

- Read Chapter 12
- Read Chapter 19
- Next class we will review for Final

ST visit

- Hopefully it was interesting
- Graphene and tape
- MEMS

Some last DAC/Music notes

- Good job on something cool
- Clicking noise in notes – when switching frequencies. How can avoid the offsets? Only change freq at zero-crossing

  Might not happen with twinkle
- Slow? You can adjust delay.
- Note run together? Need envelope, or pause between notes
- My demo: 3 channels

Midterm Update

- Almost done grading, will be handed back next class

Floating Point Comparison

- `float f = (5.0 - 1.0/7.0) + (1.0/7.0);`
- `if (f==5.0) printf("Five␣exactly.\n")`
- May work may not. Best way is to have some error (epsilon) and compare if absolute value less than a number.

Floating Point Rounding Rules

- Complex mess, can cause interesting issues
- Especially as in floating point there are extra bits usually kept around for accuracy, but they are rounded off when written out to memory and you have to fit exactly in 32 or 64 bits
- IEEE-754
  - Round to nearest

    What do we do if *exactly* in between?

    round to even

- Round toward zero (truncate)
- Round to $+$inf (round up)
- Round toward -inf (round down)

Floating Point Addition

- Shift smaller fraction to match larger one
- add or subtract based on sign bits
- normalize the sum
- round to appropriate bits
- detect overflow and underflow
- do example

Floating Point Multiplication

- Identify the sign
- add the exponents
- multiply the fractions (including leading hidden one)
- normalize the results

Transcendentals?

- Lookup tables?
- Newton's approximation (TODO put that here)
- cordic?

Quake Square Root Trick

- Fast inverse square root, popularized by use in Quake source code

```c
float InvSqrt(float x) {
    float xhalf=0.5f*x;
    int i=*(int *)&x;
    i=0x5f3759cff - (i>>1);
    x=*(float)&i;
    x=x*(1.5f-xhalf+x*x);
    return x;
}
```

Floating Point Drawbacks

- special hardware
- power hungry, if not commonly used
- chip area, expense
- back in day, special chip
- rounding issues
- money calcs. 1/10 only approximate. .0001100110011
- trouble near zero

Floating Point on Cortex-M4

- Optional on Cortex-M. Our boards do have it though.
- Thirty-two 32-bit registers S0 to S31
- Four special registers
  - CPACR – coprocessor access control reg
  - FPCCR – floating point context control
  - FPCAR – floating point context address
  - FPSCR – floating-point status and control
- The S registers can also be read as sixteen 64-bit registers D0 to D15. D0 contains S0 and S1, D1 contains S2 and

S3

- How does the ABI work?
  - When passing fp arguments put them in the registers.
  - Up to 16 32-bit or 8 64-bit can be passed
  - If you mix and match S/D then it gets complicated
  - What if you want to pass more? Goes on the stack
  - If result is fp return in S0/D0

IEEE 754 Standard

Floating Point hardware

- The FPU is disabled by default
- Have to enable CP10 and CP11
- Need to use memory barriers

Floating Point Status and Control Register (FPSCR)

- Has the N/Z/C/V bits set by the `VCMP` instructions Integer instructions cannot use these, have to cop it to the APSR using the `VMRS` instruction first
- Has control bits
  - Alt half-precision
  - Default NaN
  - Flush-to-zero
  - Rounding mode

- Has exception bits
  - Input Denormal
  - Inexact Cumulative
  - Underflow Cumulative
  - Overflow Cumulative
  - Division by Zero cumulative
  - Invalid Operation cumulative

Rounding Modes

- 00 Round to nearest (default)
- 01 Round to +infinity
- 10 Round to -infinity
- 11 Round to zero

Non-standard Modes

- Also supports some modes not in IEEE 754
  - Flush-to-zero
  - Default NaN
  - Alternative half-precision mode (16-bits?)

Exceptions

- Underflow/Overflow – when number is too small/big to represent
- Inexact exception – result lies between two floating point numbers, had to be rounded
- Invalid operation – things like 0 times infinity, infinity - infinity, sqrt(-1)
- Divide-by-zero
- Denormal (value to small, flushed to zero)

Floating Point Context Address Register (FPCAR)

- When get an interrupt, push the FP regs on the stack too
- Does “lazy stacking”, only saves them if the bit set

Instructions

- Load+Store
  - `VLDR.F32 Sd,[Rn]`
  - `VLDR.F64 Dd,[Rn]`
  - `VSTR` – store
  - `VLDM` – load multiple
  - `VPUSH` – push
  - `VPOP` – pop
  - `VMOV` – move immediate or SP/DP, also R

Instructions

- Math
  - `VADD.F32`
  - `VSUB`
  - `VDIV`
  - `VMUL`
  - `VNEG`
  - `VABS`
  - `VSQR`
  - `VMLA` – fused multiply add

- VCMP – compare – note goes to FP cmp register, need MVRS to move to integer flags registers

Misc other things

- Cortex M4 only supports single-precision floating point. What happens if you use a double? Emulated in software. (slow)
- What happens if you use FP in an IRQ routine? What could go wrong? Are the FP regs saved? What happens if you change the rounding
- Other ARM chips have fancier floating point, ARM Neon vector units

DMA

- Read Chapter 19
- Transfer data without the CPU being involved
  Why is this good? It's a bit slow. Load/store for every byte, CPU busy.
- Transfer data between peripherals and memory or memory to peripherals
- Examples
  - DAC – can queue up samples to play, and on timer interrupt the value loaded direct from MEM to DAC

w/o the CPU involved

- ADC – can read in sampled values to memory and CPU only has to deal with it once enough have built up

AMBA – Advanced Microcontroller Bus Architecture

- ARM standard that devices can connect to
  - ASB (Advanced System Bus) – high speed bus
  - AHB (Advanced high-perf bus) – used on Cortex M
  - APB (Advanced Peripheral Bus) – for low-speed devices
- Flow-through vs Fly-by DMA
- Cortex M has two DMA controllers, each 7 channels

With/Without DMA

- Without DMA
  - Busy wait until device is ready
  - Can also use interrupt – why might that be bad?
    High interrupt loads keep CPU from getting other work done, overhead of running handler each time
  - CPU loads value to register, stores out to device
- With DMA
  - DMA controller notified when device ready
  - Copies data in background

◦ Can optionally send an IRQ to CPU to let it know something happened

Programming DMA

- Channels in DMA controller hardcoded (sort of like GPIO pin assignments), have to select which one active
- There's a software and hardware priority for which takes precedence
- Programming registers
  - CMAR – channel memory address register
    Address of memory
  - CPAR – channel peripheral address register
    Address of device

- CNDTR – channel number of data register
  How much data to transfer
- CCR – channel configuration register
  direction, increment, circular, priority, interrupts

- Circular buffer
- DMA Interrupts
  - Half-transfer flag HT1F (half the data has been sent)
  - Transfer complete (TCIF)
  - Error (TEIF) if access memory it shouldn't
  - General (GIF) if any of above triggered
  - Clear these by writing 1 to the IFCR register