ESE532:
System-on-a-Chip Architecture

Day 27: December 5, 2018

Representation and Precision

Today

- Fixed Point
- Errors from Limited Precision
- Precision Analysis / Interval Arithmetic
- Floating Point
  - If time permits

Message

- We must always calculate with limited precision
- Precision costs area (and energy)
  - Can economize area (and energy) by judiciously using just the precision we need
    - Something can do when building customized accelerator
  - Precision-cost tradeoff → design-space axis
- Can perform analysis on precision

Fixed Point

- Integer which interpret as a fraction
- Fixed-Point N.F
  - N bits
  - F bits below fraction (typically N>F)
  - Equivalently: meaning is Integer-value/2^F
    - F=0 → Integer

$$A = \sum_{i=0}^{N-1} a_i 2^{i-F}$$

Operator Sizes

Operator	LUTs	LUTs + DSPs
Double FP Add	712	681+3 DSPs
Single FP Add	370	219+2 DSPs
Fixed-Point Add (32)	16	
Fixed-Point Add (n)	n/2	
Double FP Multiply	2229	223+10 DSPs
Single FP Multiply	511	461+3 DSPs
Fixed Multiply (32x32)	1099	
Fixed Multiply (16x16)	283	1 DSP
Fixed Multiply (18x25)		1 DSP
Fixed Multiply (n)	~ n^2	

FP (Floating Point) sizes from: https://www.xilinx.com/support/documentation/ip_documentation/ru/floating-point.html

Observe

- Floating-Point operators are large compared to Fixed-Point
  - For similar precision
    - 712 vs. 32 for addition
- Double-precision Floating point operators are large
  - 2229 Multiply, 712 Add
    - Can quickly fill 50,000 LUT programmable logic

Fixed-Point Economy

- Can fit more logic (more parallelism) using modest fixed-point
  - At 16b: Multiply 283, Add 16
  - Vs. Double: 2229, 712
- But
  - How much precision do we need?
  - How do we determine?

Perfect Representation

- Start with Fixed-Point 16.8
- What do we need to
  - represent the result of an addition? (1a)
  - represent the result of a multiplication? (1b)

Sequence

- Across a sequence of operations
  - A, B, C start Fixed-Point 16.8
- Y=(A+B)*C
- Perfect representation for partial results up to Y?

Looping: Bound loop

```
res=0;
for(i=0;i<3;i++)
   res=res*x+a[i]
```

- Assume a[i], x start Fixed-Point 16.8
- Final precision needed for res?

Looping: Unbound

```
res=0;
for(i=0;i<len;i++)
   res=res*x+a[i]
```

- Assume a[i], x start Fixed-Point 16.8, len starts Integer 16
- Final precision needed for res?

Perfect Representation

- Start with Fixed-Point 16.8
- What do we need to
  - represent the result of a division? (1c)
    - E.g. 00000001.00000000/00000011.00000000

Conclude

- Cannot generally keep perfect precision
- Will typically need to decide how much precision we need and where

Errors from Limited Precision

Accept errors necessary.
How big are they?
How design to manage them?

What error introduce?

- **Absolute Error** – what level of error do we have in approximated value or a result
- Might be all we care about
  - Get answer to 1mV accuracy
  - …or 1 pixel accuracy

- 4.13742 – assume full
- 4.1374 – 0.00002
  - 10^{-5}
- 4.137 – 0.00042
  - 10^{-4}
- 4.14 – 0.00256
  - 10^{-3}
- 4.1 – 0.03742
  - 10^{-2}

What error introduce?

- **Relative Error** – error as percentage of intended result
- May be more relevant, particularly if trying to identify small values

- 4.13742 – 0.0003
  - Ideal: 4.13712
  - 2 decimal frac: 4.14-0.00
    - 4.14
    - (4.14-4.13712)/4.13712
    - 0.07% error
- 4.13742-4.13628
  - Ideal: 0.00114
  - 2 decimal fac: 4.14-4.14
    - 0.00
    - (0.00114-0)/0.00114
    - 100% error

Preclass 2

- Complete Table

Reduced Precision Calculation	Y	Y_1	Error (2 significant figures) Absolute $\lvert Y - Y_1 \rvert$	Relative $\lvert (Y - Y_1)/Y \rvert$
$Y_1 = A_1 + B_1$				
$Y_1 = A_1 \times B_1$				
$Y_1 = A_1/C_1$				
$Y_1 = A_1/D_1$				

Observe

- Add/Multiply relatively well behaved
- Must be very careful when
  - Division involved
  - Divisors can be small
    - Get approximated near zero

Precision Allocation

- Full precision can be too expensive
  - Non-sensical
- Limited precision introduces errors
  - May be smaller than we care about
- Determine minimal precision needed
  - …or where to spend precision…

Empirical Analysis

- Make guess at precisions
- Set precisions in calculation
- Simulate on data
- Evaluate results (absolute, relative error) compared to gold standard
  - Unlimited precision…or, at least, higher precision
    - Often standard is double-precision float
      - …but, as we'll, even that's a compromise
- Update precision guess and repeat

Empirical Analysis

- Make guess at precisions
- Set precisions in calculation
- Simulate on data
- Evaluate results compared to gold standard

Demands Care

- Test coverage
  - Adequate set of test data to trigger worst-case errors?
- Requires some understanding of calculation
- Shouldn't be entirely black box

Vivado HLS Support

- Has libraries to support
  - Arbitrary precision integers
  - Arbitrary precision fixed point
- For
  - Simulation
  - Synthesis
- UG902 – Vivado HLS User Guide
  - Chapter 2: Abritrary Precision Data Type Library

Precision Analysis

Precision Analysis

- Can analyze worst-case error impacts from limited precision
- Give results not sensitive to test set
- Give guidance on where to allocate precision
- …can be automated

Limit Precision Inputs

- Typically start with limited precision
  - A/D only sample to 12b
    - Real-world had more precise value, but didn't capture
  - Quantized data stored in representation
    - Sound samples, DCT frequency coefficients
- We start with error
  - What does that mean about values we calculate?

Interval Analysis

- Treat every value as an interval arrange
- Model effects of operations on range of results
- A=(A.H, A.L) e.g. read 37 (37.49,37.51)
- A+B=(A.H+B.H,A.L+B.L)
- Assuming Positive A, B and interval not cross 0, what is range for:
  - A*B
  - A/B

Interval Analysis

- Treat every value as an interval arrange
- Model effects of operations on range of results
- A=(A.H, A.L)
- A+B=(A.H+B.H,A.L+B.L)
- Positive A, B and B interval not cross 0
  - A*B=(A.H*B.H,A.L*B.L)
  - A/B=(A.H/B.L,A.L/B.H)

Interval Analysis

With ranges that may cross zero…

- A*B=(max(A.H*B.H,A.H*B.L,A.L*B.H,A.L*B.L), min(A.H*B.H,A.H*B.L,A.L*B.H,A.L*B.L))
- A/B … more complicated
  - If B.H positive, B.L negative, can become infinity

Limited Precision

- A=(A.H,A.L)=(A+ε,A-ε)=A±ε
- E.g. if rounded to Fixed Point 16.8
  - ε may be 2^{-9}

Preclass 2

- Complete Table

Calculation	Result Range (report like A)
$Y = 1 + A$	$A_8 + 1 \pm \epsilon$
$Y = A + B$	
$Y = 2A$	
$Y = A \times B$	

Multiplication

- (A±ε)(B±ε)
- A*B ± ((A+B) ε + ε^2)
- A and B can be MAXVAL, MINVAL
  - Assume symmetric (MAXVAL=-MINVAL)
- A*B ± (2*MAXVAL* ε + ε^2)
  - Probably reasonable to drop ε^2
- A*B ± 2*MAXVAL* ε

Multiply Range Example

- Recall: Fixed-Point 16.8 multiply
  - Full precision result: 32.16 (preclass 1b)
- What is error interval for result of 16.8 multiply?

Range Example

- Error Interval for 16.8 fixed-point multiply
- For 16.8, ε_0=1/512, maxval=256
- From preclass 3 symbolic answer
  - A*B ± 2*MAXVAL* ε
  - A*B ± 2*256*(1/512)
  - Or A*B±1

Observe

- Full precision may keep bits well below the error in the calculation
  - E.g. 32.16 result, keeping 16b below 2^0 term
  - Entire fraction is below the error in the calculation
    - A*B±1

Rounding

- Rounding introduces an error
- Round Nearest A to Fixed-Point N.8
  $\varepsilon=2^{-9}$
- As does truncation, floor, ceil…
  - But asymetric interval

Compute and Round Error

- Error range if round 32.16 result to 18.2?
  - (from 16.8 multiply)
  - Hint:
    - How much from calculation?
    - How much additional from rounding?

Compute and Round Error

- Rounding 32.16 to 18.2 introduces a quantization error of 2^{-3}
- Our 32.16 multiply result was ±1
  - Add an addition ±1/8
  - Total error: ±9/8

Result

- Dropping (rounding) bits may not increase error range (much)

Symbolic

- If work through computation symbolically, can generate equation for error
- Each rounding (precision drop) adds some ε_i precision

Approach

- At each step compute interval
- Keep track of
  - maxval
  - ε

Symbolic Example

- Start A, B, C with $maxval_0$, ε_0
- Y=(A+B)*C;
- A+B $maxval_1=2maxval_0$, $\varepsilon_2=2\varepsilon_0+\varepsilon_1$
  - ε_1 for round after this operation
- (A+B)*C
  - $maxval_2=maxval_1*maxval_0$
  - $\varepsilon_4=\varepsilon_2*maxval_0+\varepsilon_0*maxval_1+\varepsilon_3$
  - ε_3 for round at this operation

Result Precision

- After series of operations, may have expression like:
  - $Y\pm(\varepsilon_3+(\varepsilon_1+4*\varepsilon_0)*maxval_0)$
- If looking for result with precision $\pm\varepsilon_{res}$

  $\varepsilon_{res}\geq(\varepsilon_3+(\varepsilon_1+4*\varepsilon_0)*maxval_0)$

Result Precision

- Fixed Precision12.0 = Round(val)
  - E.g. A/D output
  - Only need to know val to ε=1/2
- Fixed Precision 12.0 = Round(val/4)
  - E.g. Quantized value stored in file
  - Only need to know val to ε=2
  - Start 13.8, $maxval_0$=32
  - $\varepsilon_{res} \geq (\varepsilon_3+(\varepsilon_1+4*\varepsilon_0)*maxval_0)=(\varepsilon_3+(\varepsilon_1+4*\varepsilon_0)*32)$
    - What epsilons might solve?
      - Hint: try budget half unit for each ε_3, ε_1 term

Optimize Precision Allocation

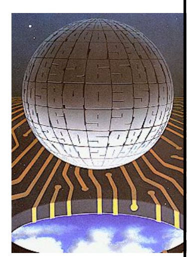
- $\varepsilon_{res} \geq (\varepsilon_3+\varepsilon_1+4*\varepsilon_0*32)$
  - Maybe: ε_0=1/256, ε_1=1/2, ε_3=1/64
  - 1/256 – 7 bit fraction, 1/64 – 5 bit, ½ -- no fraction
- More generally
  - Combine with area model and look at expense of providing each ε_i
  - Round to 12.7, Fixed 12.7 add, round to 11.5, Fixed 11.5 multiply, round to 12.0
    - 12/2 add, 12/2 round, 11*11 for multiply ~ 133 LUTs
- Try pick ε_i to meet ε_{res} while minimizing area

Tools

- Tools can automate interval calculations to verify precision
  - E.g. Gappa++
  - https://bitbucket.org/mlinderm/gappa

Floating Point

(time permit)



Robert Tinney
(Byte circa 1980)

Floating-Point vs. Fixed-Point

- Floating-Point (esp. double-precision) is a **big hammer** solution
  - Trades hardware/energy for programmer attention to needs
  - Standards have been well thought out so works over wide range
    - One size fits all (…almost)
  - Most cases it is more than needed
- Cost/energy sensitive designs will ask what's necessary and tune accordingly

Floating Point

- Leading 0s aren't that useful
- Can represent more compactly by counting them
  - Only need log bits to count the zeros
- Represent value as $v=1.m * 2^{(exp-offset)}$
  - Mantissa (m)
  - Exponent (exp)
- Like Scientific Notation

Floating Point

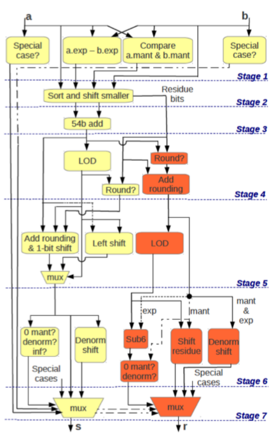
- Floating Point means
  - Move the datapath to the interesting/significant part of computation
  - Don't represent leading zeros
  - Don't represent less significant bits
    - Even if they are well above 1
      - In the integer portion

Standard Floating Point

- Double Precision (64b)
  - 1 bit sign
  - 11 bit exponent
    - Offset 1023 represents 1023 to -1022
  - 53 bit mantissa (52b + implicit 1)
- Single Precision (32b)
  - 1 bit sign
  - 8 bit exponent (offset 127)
  - 24 bit mantissa (23b + implicit 1)

Expensive FP Add

- Recall
  - 712 LUTs double
  - 370 single
- Double:
  - 54b add one stage of 7 in pipeline
  - Done in 27 6-LUTs



Floating Point Multiply

Double 2229, Single 461

- Don't need to sort, pre-shift
- m=A.m*B.m (53x53 multiply dominates)
- e=A.e+B.e
- Still have shifting, rounding at end

Dynamic Range

- Floating Point has very wide dynamic range
- Can deal with significant piece being anywhere in 1023 to -1022
- For fixed-point to cover same range
  - Fixed Point 2046.1022
  - Add 1024 LUTs
  - Multiply ~ 4M LUTs
- When need dynamic range, FP economical

Customization

- Can customize Floating Point on FPGA or custom silicon
  - Mantissa bits
  - Exponent bits
- Fewer bits when need less precision or range to save area
- More bits if need greater precision or range

Floating Point

Not free of precision problems

- $((1+2^{100})-2^{99})-2^{99}=0$
- $1+(2^{100}+(-2^{99}-2^{99}))=1$

Floating-Point Analysis

- Can do similar analysis on floating point
  - …and there are tools to help
  - Including Gappa++

Big Ideas

- We must always calculate with limited precision
- Precision costs area (and energy)
  - Can economize area (and energy) by judiciously using just the precision we need
  - Precision-cost tradeoff
    - Design-space axes
- Can perform analysis on precision

Admin

- Project Due Friday
  - Report individual
  - Elf, bitstream, decoder – one per group
  - Code – everyone turn in, but same across group
- Return boards Monday in Class
- Exam following Friday (12/14)
  - Towne 303 (here), 9am