Lesson 102: Convergence Computing Method (CCM)

- Motivation
- CCM algorithm
- Numerical properties (CCM vs CORDIC)
- Project requirements

University of Victoria
British Columbia • Canada

Motivation

- Logarithm and exponential are widely used in signal processing
  - Determinant calculation requires multi-operand multiplication, which can be implemented by a *log-add-exp* approach
  - Compensation of non-linear effects and multiplicative noise in wireless communications requires *log* and *exp*
  - *Cepstral* (<u>spec</u>–tral → <u>ceps</u>–tral) processing in speech recognition applications

- Direct evaluation of *log* and *exp* is computationally demanding
  - It translates to a sequence of multiplications, additions, and memory look-up operations if the common Taylor series expansion is employed.

Convergence Computing Method (CCM)

- It is an iterative method for computing *log* & *exp* using only shifts and additions
  - **Cheap**: only shifts and additions are needed
  - **Sequential**: it is an iterative method

- It is similar to CORDIC

- It can be used to calculate square root, cube root, and higher-index roots

Calculation of Logarithm

- Calculate $\log M$, where $0.5 \leq M < 1.0$
  - In fixed-point representation, this means that M is a normalized fractional number (there are no zeros in front of the number except, maybe, a sign bit).
  - If M is not normalized, a left-shift operation will normalize it

- **Basic principle**: cyclic multiplication of M by a sequence of specially chosen factors A_i, as needed, until the product falls in a predefined range, $(1.0 - \Delta, 1.0\,]$

- The constant Δ specifies the precision of the computation

- Notation: P is the final product:

$$1 - \Delta \; < \; P \; \leq \; 1\,, \qquad \text{where} \;\; P = M \cdot \prod_{i=1}^{K} A_i$$

Calculation of Logarithm

- Take the logarithm of both sides:

$$\log M = \log P - \sum_{i=1}^{K} \log A_i \approx - \sum_{i=1}^{K} \log A_i$$

since $\log\ P \approx 0$ within the precision specified by Δ.

- **Main result**: $\log M$ is approximated as a sum of predefined constants, $-\log A_i$

- The factors A_i are either equal to 1 or of the form $1 + 2^{-i}$, such that a multiplication by A_i reduces to one addition and one shift

- The constants $\log(1 + 2^{-i})$ are precomputed and stored into a Look-Up Table

University of Victoria
British Columbia • Canada

Calculation of Binary Logarithm – Pseudocode

{$\log_2 M$ with K bits of precision}
for $i = 0$ to $K - 1$ **do**
    $\mathsf{LUT}(i) = \log_2(1 + 2^{-i})$ {calculate the table with $\log_2 A_i$}
end for
$f = 0$
for $i = 0$ to $K - 1$ **do**
    $\mu = M \cdot (1 + 2^{-i})$ {potential multiplication by A_i}
    $\phi = f - \mathsf{LUT}(i)$ {potential addition with $\log_2 A_i$}
    if $\mu \leq 1.0$ **then**
        $M = \mu$ {if product is less than 1 accept iteration,}
        $f = \phi$ {otherwise reject it (do nothing)}
    end if
end for
return f

Calculation of Exponential

- Calculate $\exp M$, where $0 \leq M < 1.0$
  - This means M is a positive and pure-fractional number
  - Correction arithmetic (e.g., shift operations) can always enforce that

- **Basic principle**: ciclic addition to M by a sequence of specially chosen summands B_i, as necessary, until the sum falls in a predefined range, $[0, \Delta)$

- The constant Δ specifies the precision of the computation

- Notation: S is the final sum:

$$0 \leq S < \Delta\,, \qquad \text{where } S = M - \sum_{i=1}^{K} B_i \quad \Rightarrow \quad M = S + \sum_{i=1}^{K} B_i$$

University of Victoria
British Columbia • Canada

Calculation of Exponential

- Take the exponential of both sides:

$$\exp M = \exp S \cdot \prod_{i=1}^{K} \exp B_i \approx - \prod_{i=1}^{K} \exp B_i$$

  since $\exp S \approx 1$ within the precision specified by Δ.

- **Result**: $\exp M$ is approximated as a product of predefined constants, $\exp B_i$

- The factors B_i are either equal to 0 or of the form $\log(1 + 2^{-i})$, such that a multiplication by $\exp B_i$ reduces to one addition and one shift

- The constants $\log(1 + 2^{-i})$ are precomputed and stored into a Look-Up Table

Calculation of Base-2 Exponential – Pseudocode

{2^M with K bits of precision}
for $i = 0$ to $K - 1$ **do**
    LUT$(i) = \log_2(1 + 2^{-i})$ {calculate the table with B_i}
end for
$f = 1.0$
for $i = 0$ to $K - 1$ **do**
    $\mu = M -$ LUT(i) {potential addition with B_i}
    $\phi = f \cdot (1 + 2^{-i})$ {potential multiplication by 2^{B_i}}
    if $\mu \geq 0$ **then**
        $M = \mu$ {if sum is greater than 0 accept iteration,}
        $f = \phi$ {otherwise reject it (do nothing)}
    end if
end for
return f

Calculation of Square Root

- Calculate $\sqrt{M}$, where $1.0 \leq M < 4.0$
  - In fixed-point representation, this means that M has two bits for the whole part, whereas the remaining bits specify the fractional part.
  - If this is not the case, shift operation will fix the representation.

- **Basic principle**: cyclic multiplication by a sequence of specially chosen factors A_i^2, as needed, until the product falls in a predefined range, $(M - \Delta, M\,]$

- The constant Δ specifies the precision of the computation

- After K iterations:

$$M - \Delta < \prod_{i=1}^{K} A_i^2 \leq M\,, \qquad \text{and} \qquad \sqrt{M - \Delta} < \prod_{i=1}^{K} A_i \leq \sqrt{M}$$

Calculation of Square Root

- The factors A_i are either equal to 1 or of the form $1 + 2^{-i}$, such that a multiplication by A_i reduces to one addition and one shift.

- A multiplication by A_i^2 reduces to two additions and two shift operations.

- In a similar fashion, it is possible to calculate the cubic root (and, in general, a root of any order).

- The logarithm, exponential, square root, and cubic root can be calculated by using additions and shift operations only.

- Recall that this is a sequential algorithm.

Calculation of Square Root – Pseudocode

$\{\sqrt{M}$ with K bits of precision$\}$
$f = 1.0$
$f\_sqrt = 1.0$
for $i = 0$ to $K - 1$ **do**
    $\mu = f \cdot (1 + 2^{-i}) \cdot (1 + 2^{-i})$ $\{$potential multiplication by $A_i^2\}$
    $\mu\_sqrt = f\_sqrt \cdot (1 + 2^{-i})$ $\{$potential multiplication by $A_i\}$
    if $\mu \leq M$ **then**
        $f = \mu$ $\{$if product is less than M accept iteration,$\}$
        $f\_sqrt = \mu\_sqrt$ $\{$otherwise reject it (do nothing)$\}$
    end if
end for
return f_sqrt

CCM – project requirements

- Build the testbench:
  - values for M to calculate $\log$ and $\exp$
- Implement the CCM algorithm using integer arithmetic
  - software (write C routines)
  - horizontal firmware with two and three issue slots
  - custom hardware (write VHDL/Verilog)
- Define a new instruction that will return the transcendental function
  - You must comply with the ARM architecture (you can have at most two arguments and one result per instruction call)

CCM – project requirements

- Rewrite the high-level code and instantiate the new instruction
  - Use assembly inlining
- Estimate
  - the performance improvement of hardware-based solution versus software-based solution
  - the performance improvement of a 2-issue slot firmware-based solution versus software-based solution
- Estimate the penalty in terms of number of gates for the hardware solution

Questions, feedback

?