arm

Radix-64 Floating-Point Divider

Javier D. Bruguera

ARITH25
June 25-27, 2018

Overview

- Main features
  - General architecture
  - Performance
- Radix-64 digit-recurrence division
  - Overlapping of three radix-4 iterations
- Microarchitecture
  - Pre-processing
  - Digit iteration
    - Digit selection
    - Next remainder calculation
- Evaluation and comparison

Main Features

arm

Operands and Result

- Floating-point division, $q = \frac{x}{d}$,
- Normalized operands, $x, d \in [1,2)$, although …
- Subnormal operands are accepted → operand normalization before the digit iterations
- To simplify the rounding result is forced to be in $q \in [1,2)$
  - If result is $q \in [0.5,2)$ → rounding needs a *guard bit* and a *round bit*, and result can need a 1-bit left shift
  - $q \in [0.5,1)$ if $x < d$,
  - $\boldsymbol{q \in [1,2)}$ **if** $\boldsymbol{x \geq d}$
  - Early detection of $x < d \rightarrow q = \frac{2x}{d}, and\ q \in [1,2)$
  - Same mantissa as in x/d, but exponent needs to be decremented
- Support for double, single and half-precisions

arm

Digit-Recurrence Division Algorithm

- Radix-64, overlapping three radix-4 iteration: 6 bits of the result are obtained every cycle (6 bits/cycle)
  - Each radix-4 iteration gives 2 bits of the result
  - Each radix-4 iteration,
    1. Quotient digit selection, digits $\in \{-2, -1, \ 0, \ +1, +2\}$
    2. Remainder update
- Operands pre-scaling to have a simple quotient-digit selection function
  - If divisor is close enough to 1, the digit selection is independent on the divisor, depends only on the remainder
  - Dividend scaled as well to preserve the result
- The first quotient-digit (integer digit) can take values $\{+1, +2\}$ → simplified selection logic
  - In parallel with the pre-scaling

Early Termination Mode

- Occurs when
  - Any of the operands is NaN, ∞, 0
  - Division by a power of 2
- The result is not obtained in the digit calculation iterations
- Shorter latency

Latency

- Number of result bits: 1 integer bit + *n* fractional bits
  - Integer bit is obtained in parallel with the pre-scaling
  - Fractional bits include the guard bit, $n = 53\ (DP), 24\ (SP), 11\ (HP)$
- Number of digit cycles is (6 bit/cycle)
  - Half-precision: 2 digit cycles -> 6 radix-4 iterations -> 12 fractional bits
  - Single-precision: 4 digit cycles -> 12 radix-4 iterations -> 24 fractional bits
  - Double-precision: 9 digit cycles -> 27 radix-4 iterations -> 54 fractional bits
- Latency for normal operation (no subnormals, result normalized)
  - Half-precision: PSC – DGT –DGT – RND
  - Single precision: PSC – DGT – DGT – DGT – DGT – RND
  - Double-precision: PSC – DGT – DGT – DGT – DGT – DGT - DGT – DGT – DGT – DGT – RND
- Obtaining the first digit in parallel with the pre-scaling and forcing the result in $[1,2)$ contribute to save 1 cycle latency

arm

Digit-Recurrence Division

arm

Radix-r Digit-Recurrence Division

- Iterative algorithm
- Iteration *i* computes :
  - a radix-r quotient digit, q_{i+1},
  - a remainder, $rem[i+1]$
  - The remainder is used to get the next quotient digit
- $r = 4$
- Partial quotient before iteration *i*, $\mathrm{Q[i]} = \sum_{j=0}^{i} q_j \times 4^{-j}$
- At iteration *i*

$$q_{i+1} = SEL(4 \times \widehat{rem}[i])$$
$$rem[i+1] = 4 \times rem[i] - q_{i+1} \times d$$

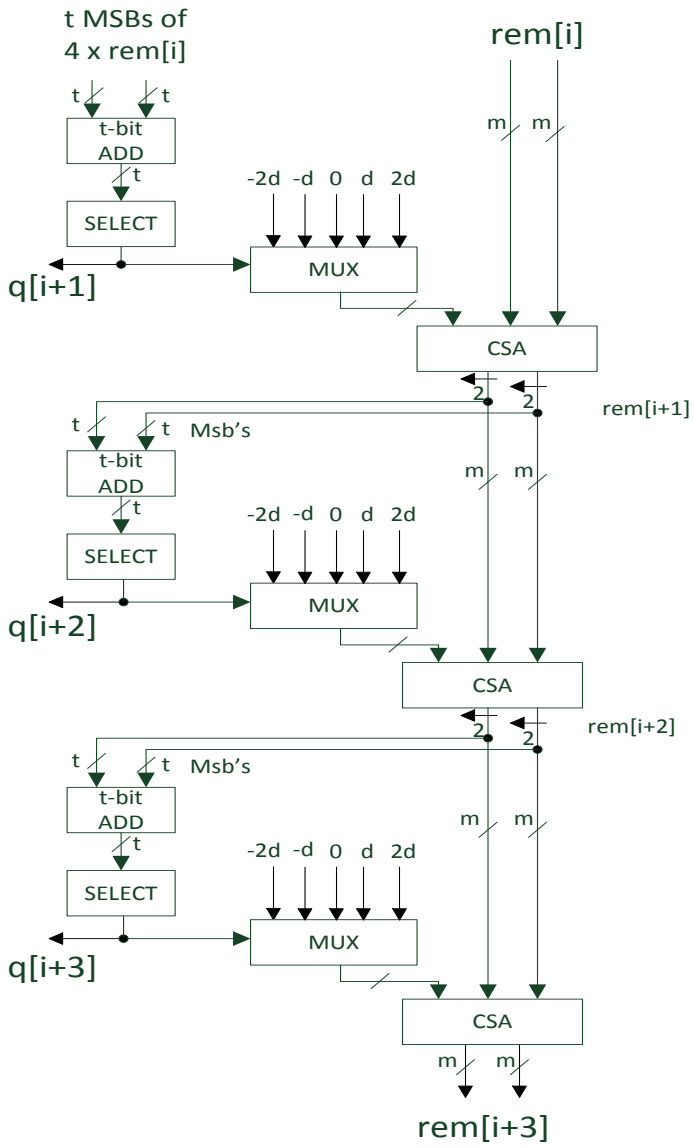
- Usually, $rem[i]$ in redundant carry-save or signed-digit representation
  - $\widehat{rem}[i]$ is an estimation of $rem[i]$ with few bits (**6 bits** in radix 4)

arm

Number of Iterations and Cycles

- Radix-4 iterations
  - Number of iterations is $it = \lceil n/\log_2 4 \rceil = \lceil n/2 \rceil$
  - For example, double-precision: $n = 53 \rightarrow it = 27$
- Radix-64 division, three radix-4 iterations per cycle
  - Number of cycles for normal division is $cycles = \lceil it/3 \rceil + 2$
  - 1 pre-scaling cycle, $\lceil it/3 \rceil$ digit cycles, 1 rounding cycle
  - For example, double-precision: $it = 27 \rightarrow cycles = 9 + 2 = 11$

Radix-64 Divider – Naive Implementation



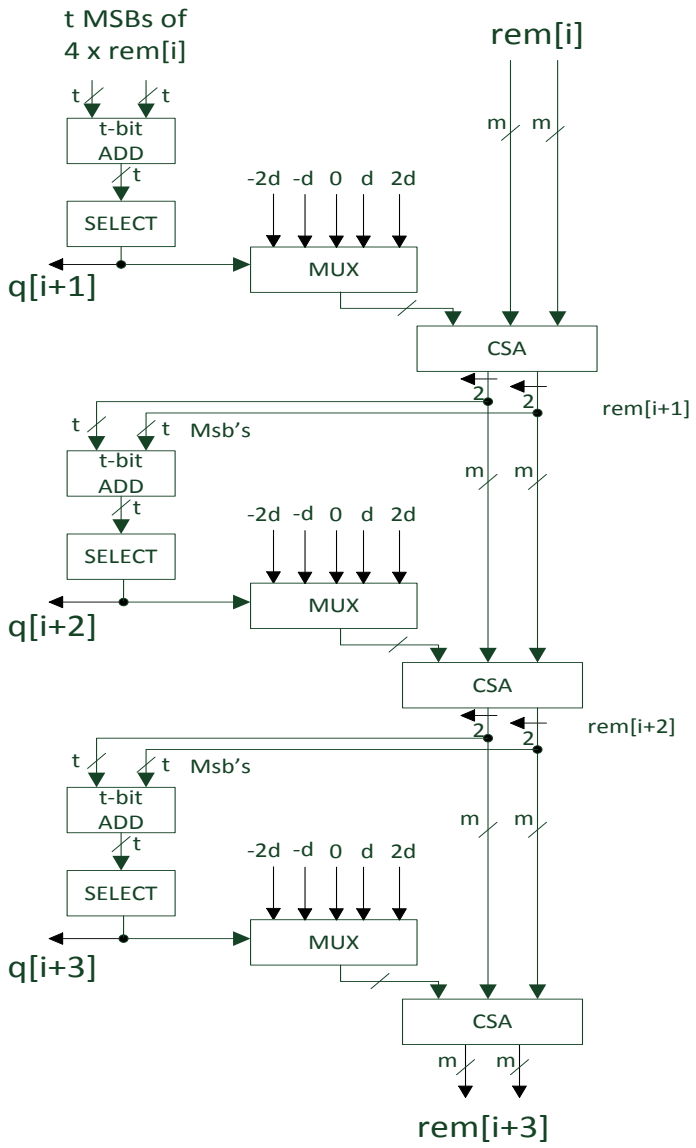
$$rem[i+1] = 4 \times rem[i] - q[i+1] \times d$$

arm

Radix-64 Divider Microarchitecture

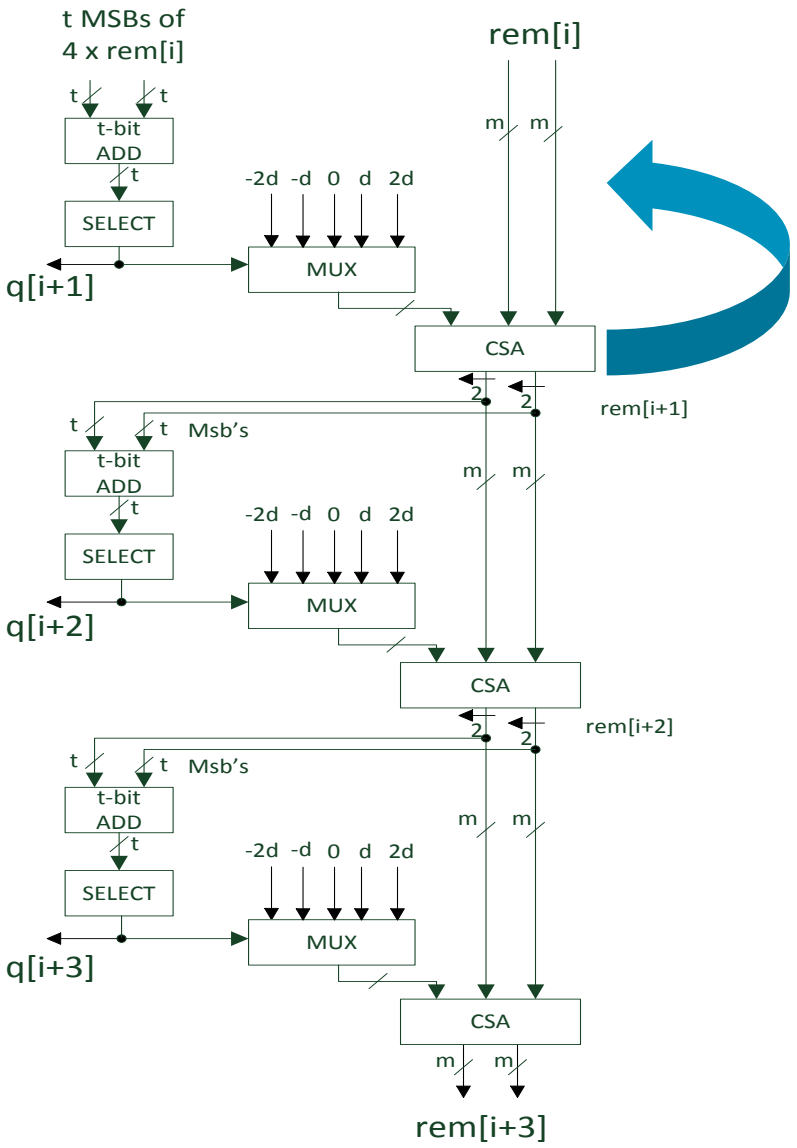
arm

Overlapping Radix-4 Iterations with Speculation/Replication



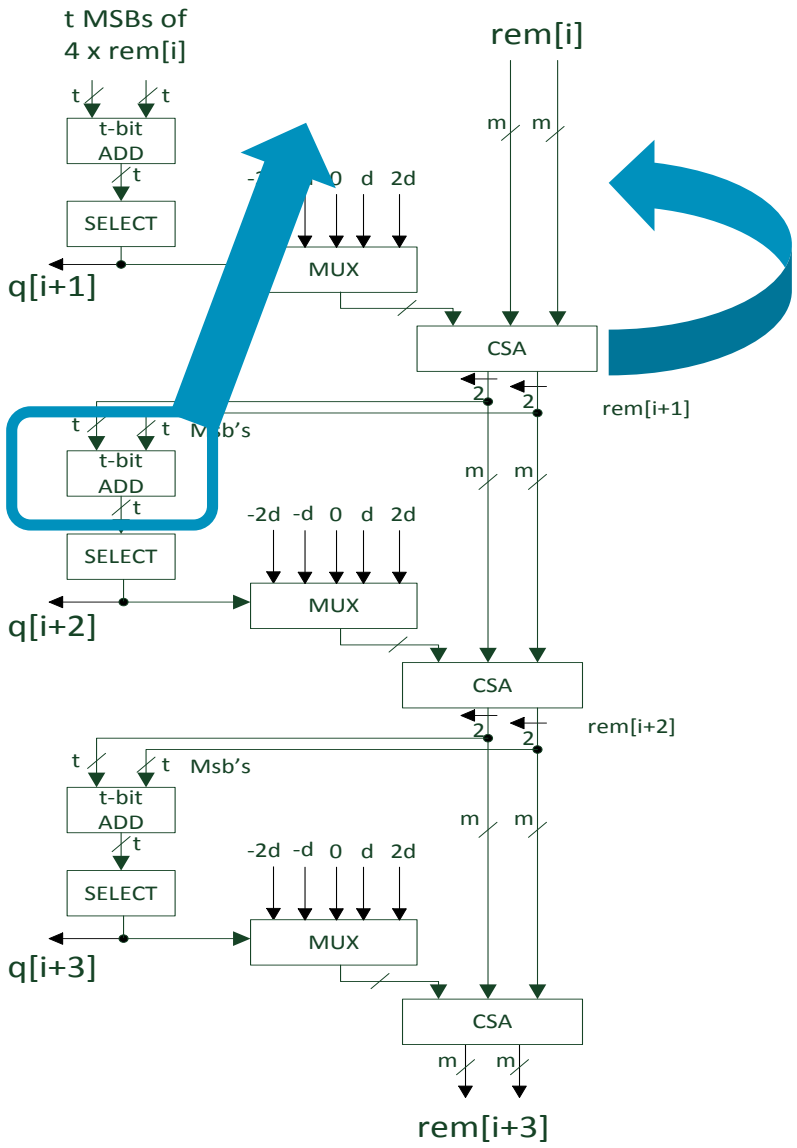
arm

Overlapping Radix-4 Iterations with Speculation/Replication



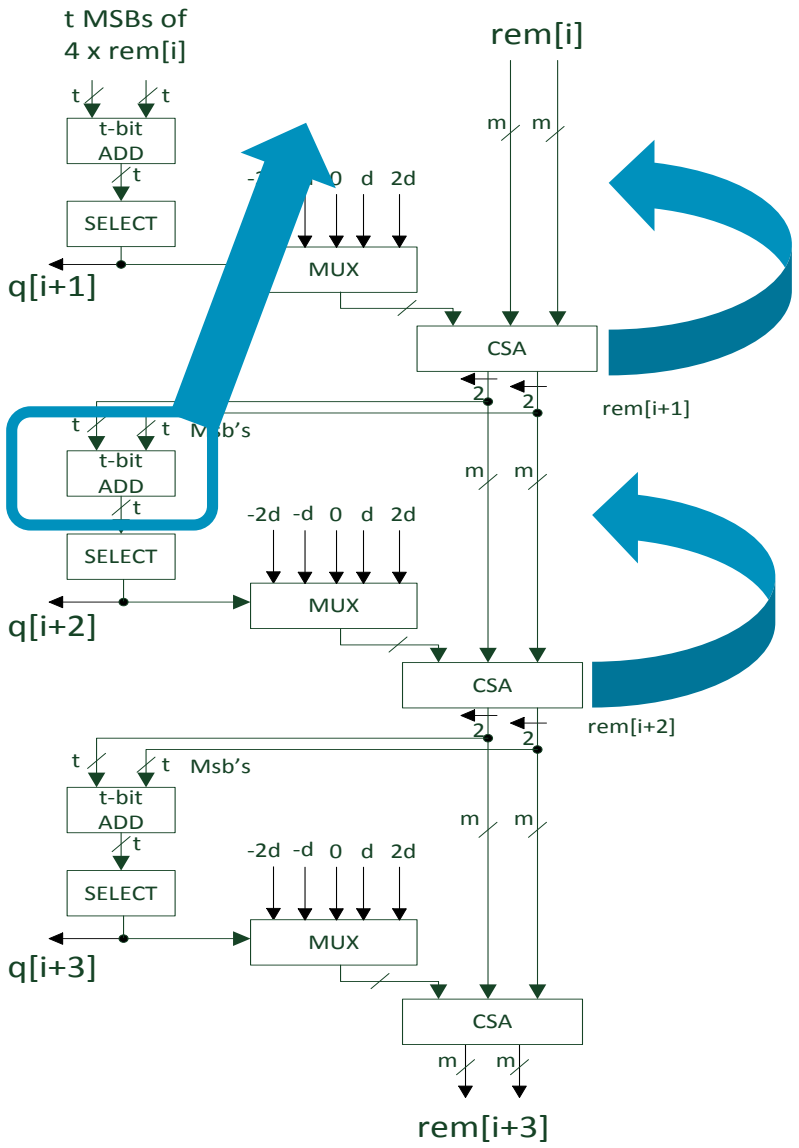
arm

Overlapping Radix-4 Iterations with Speculation/Replication



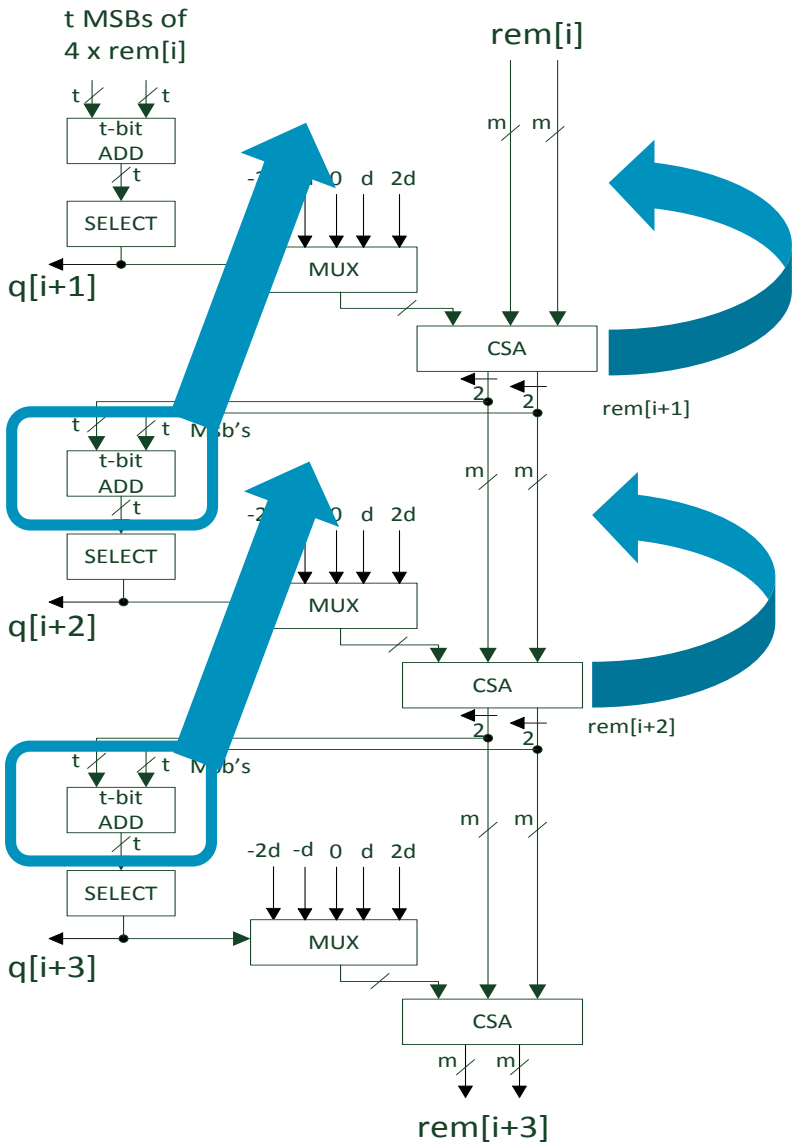
arm

Overlapping Radix-4 Iterations with Speculation/Replication



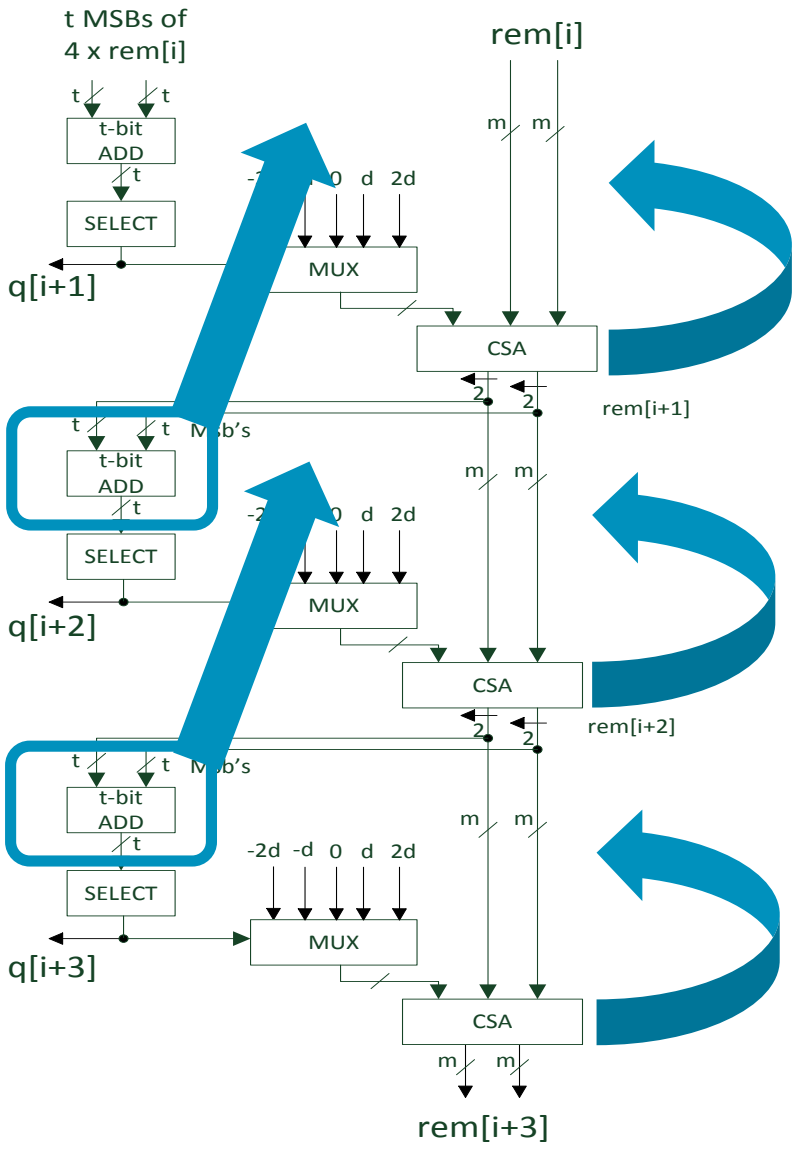
arm

Overlapping Radix-4 Iterations with Speculation/Replication



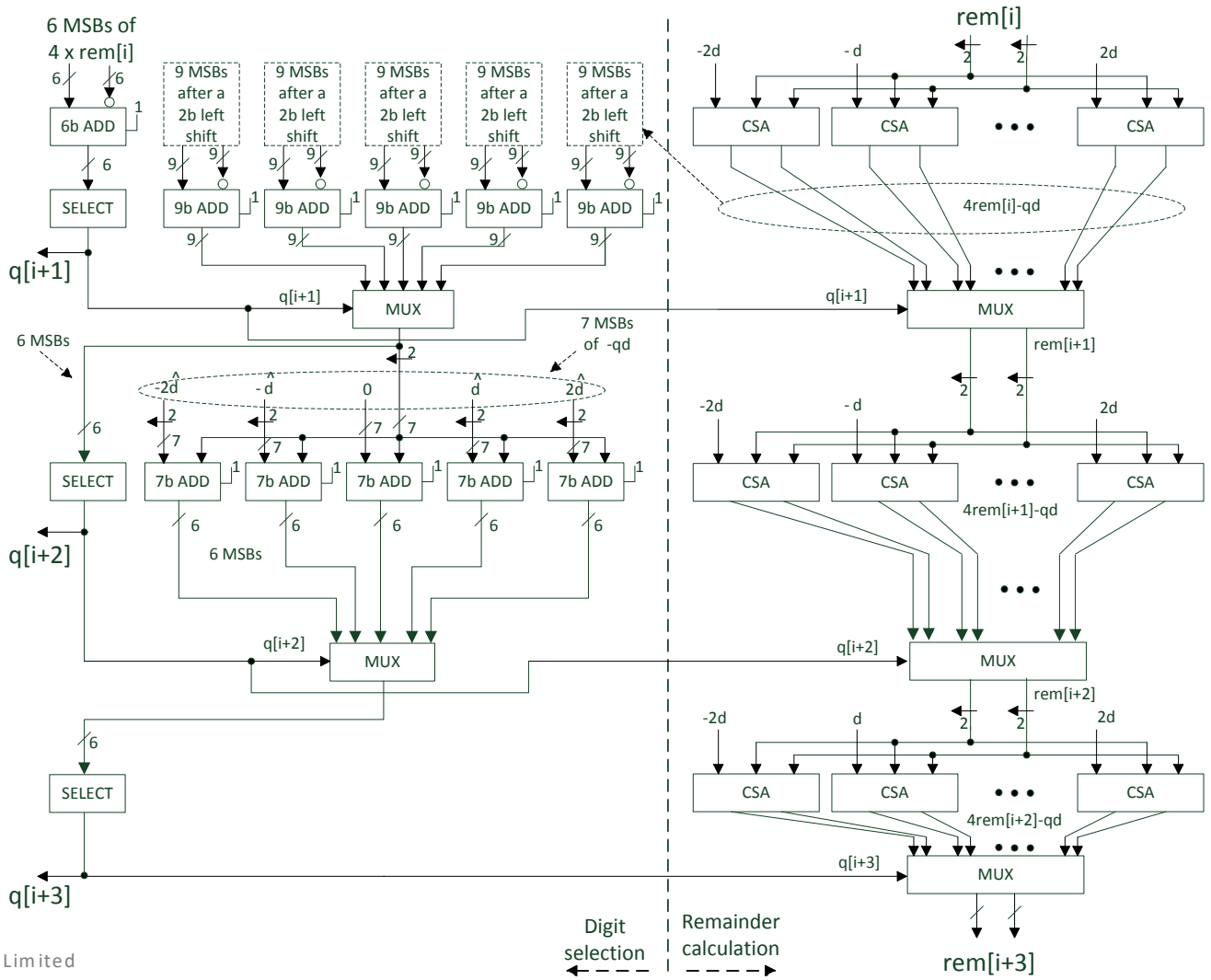
arm

Overlapping Radix-4 Iterations with Speculation/Replication

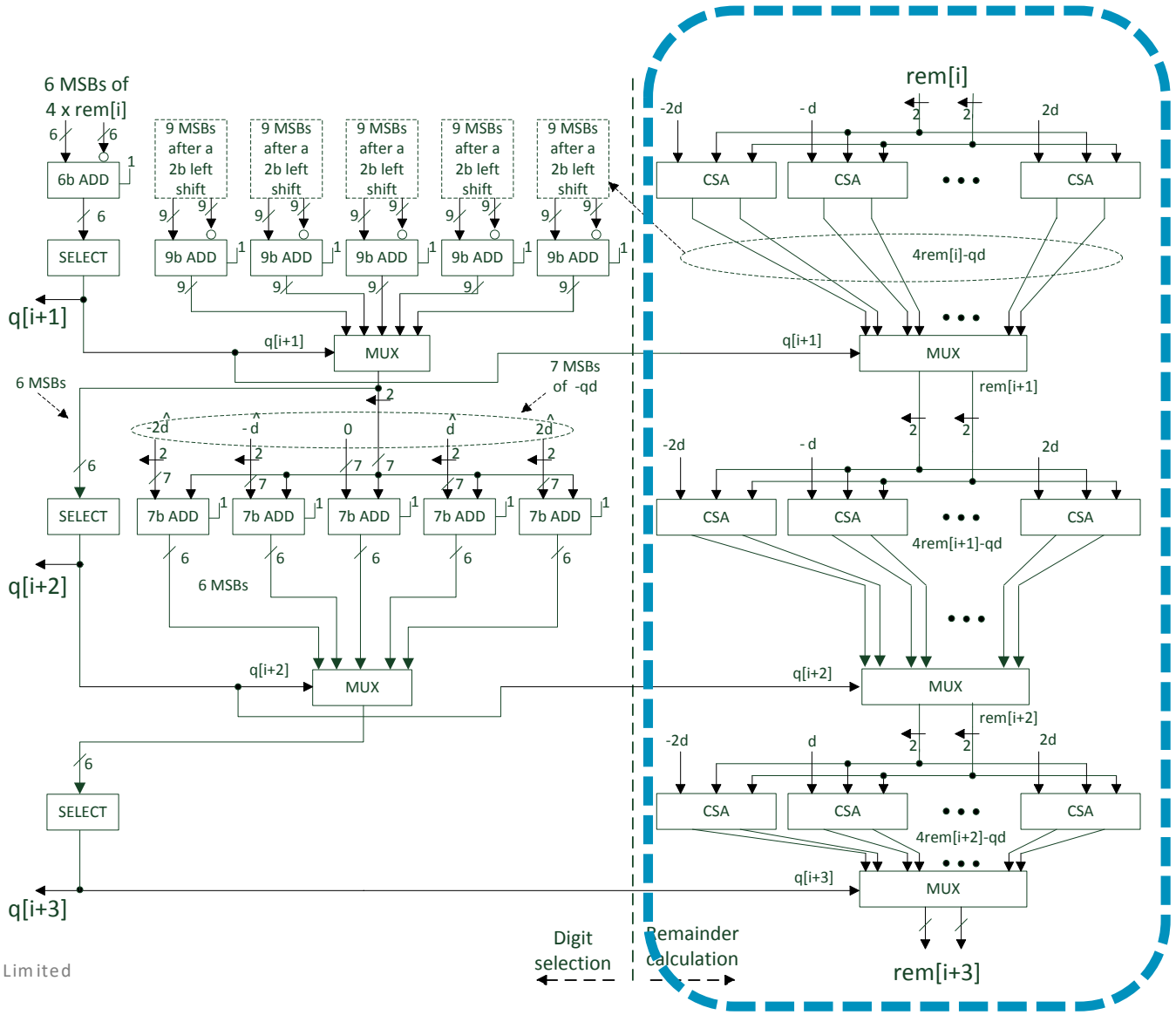


arm

Radix-64 Iteration



Radix-64 Iteration

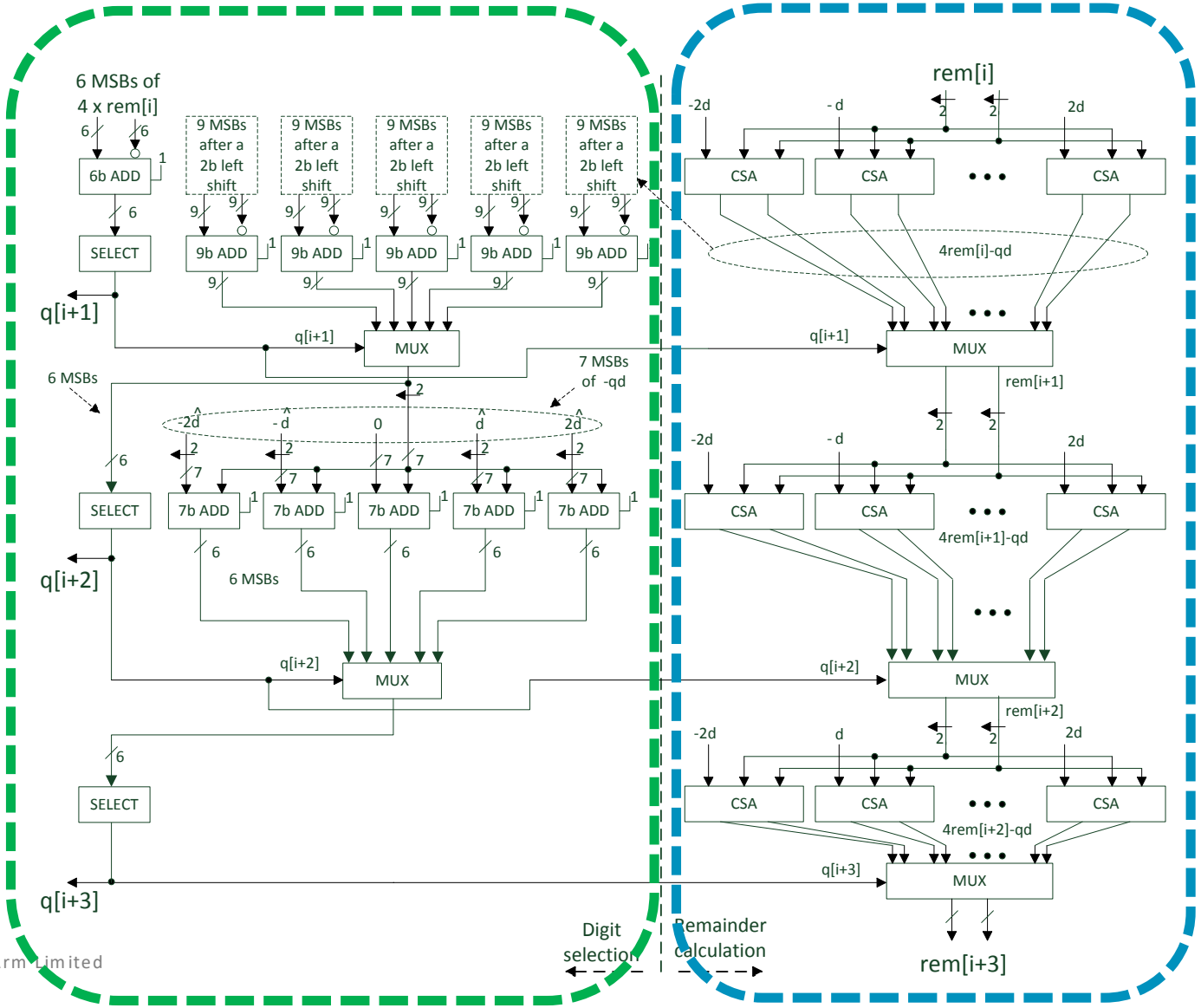


Remainder update

arm

Radix-64 Iteration

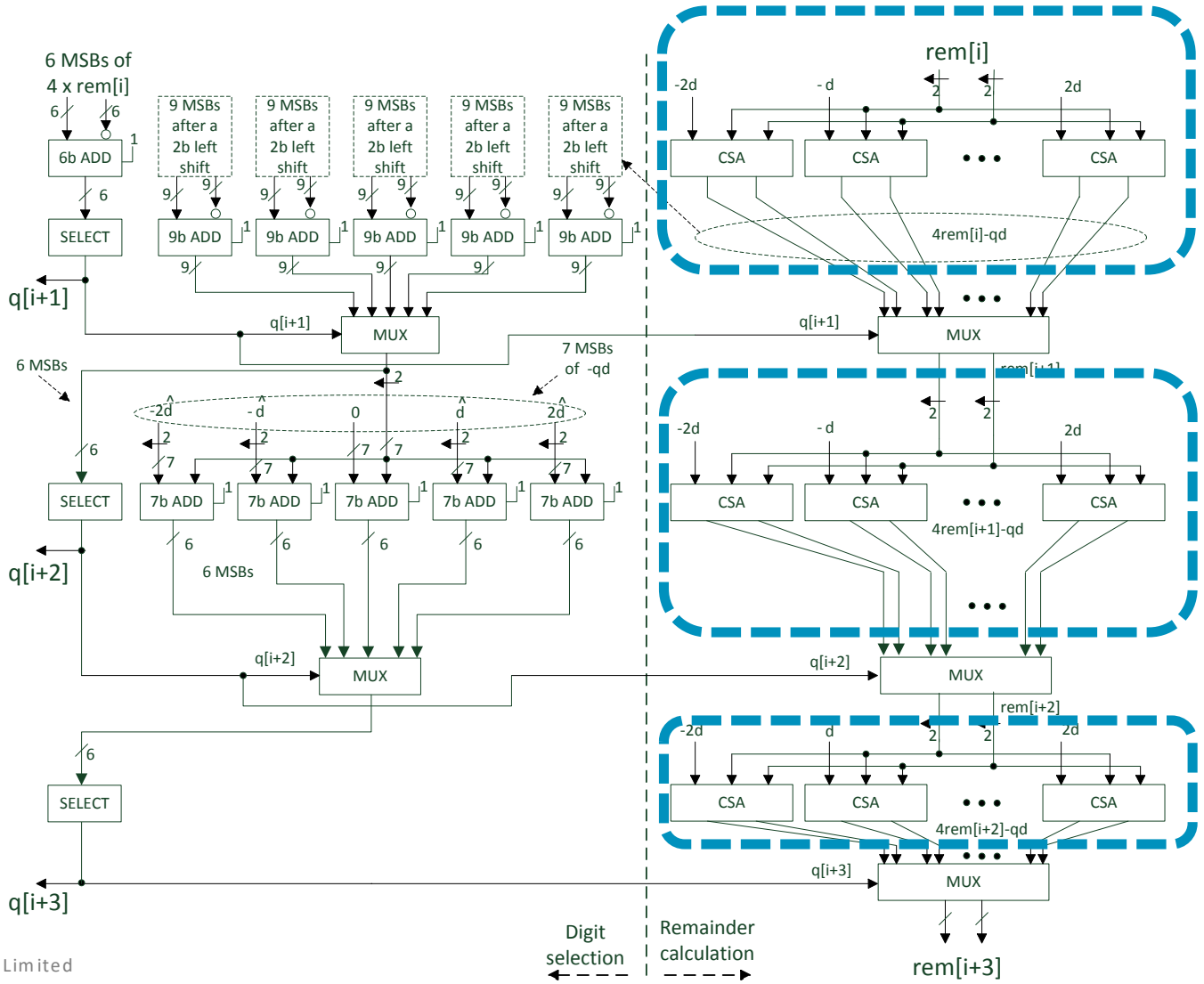
Digit Selection



Remainder update

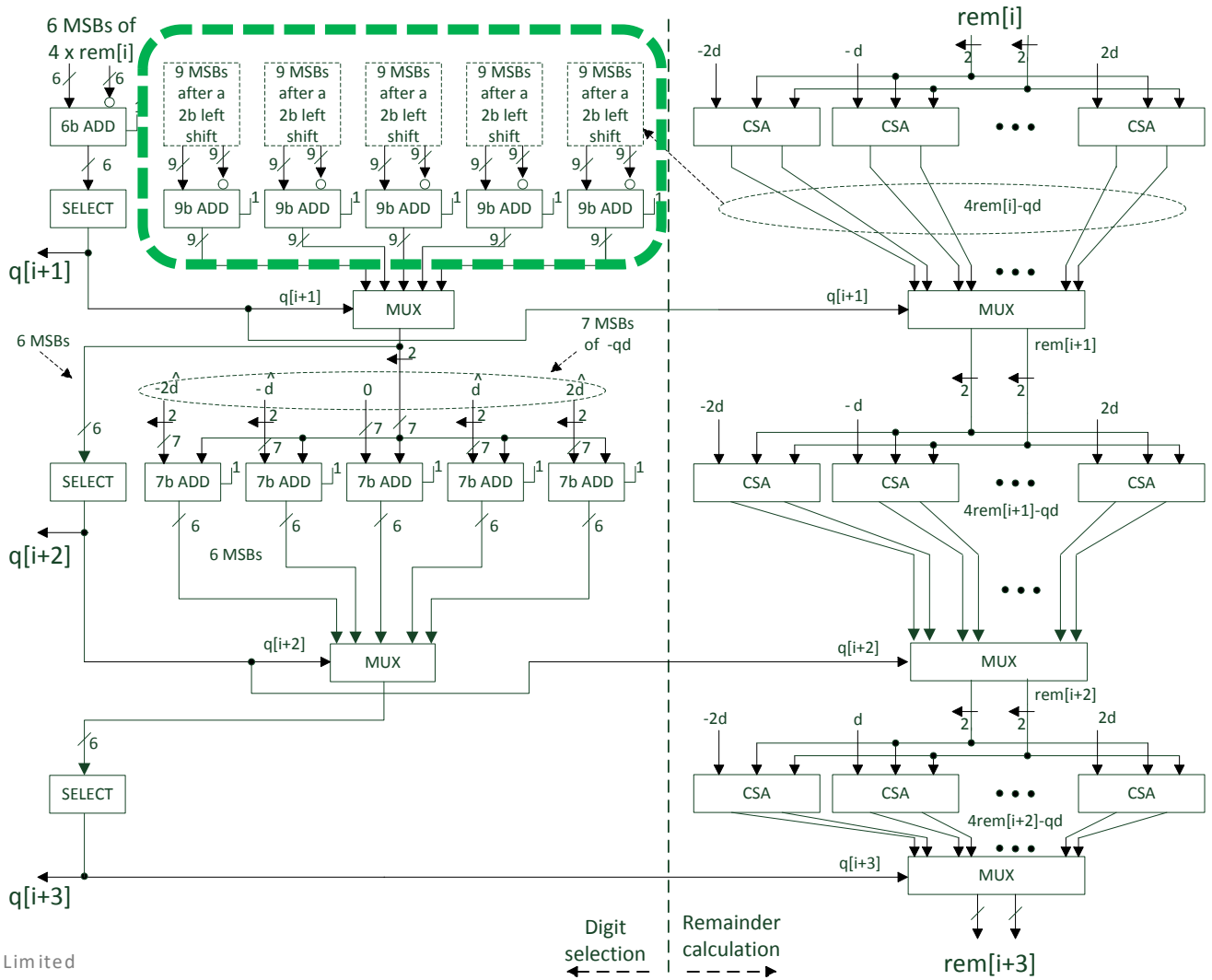
arm

Radix-64 Iteration



Radix-64 Iteration

Digit Selection

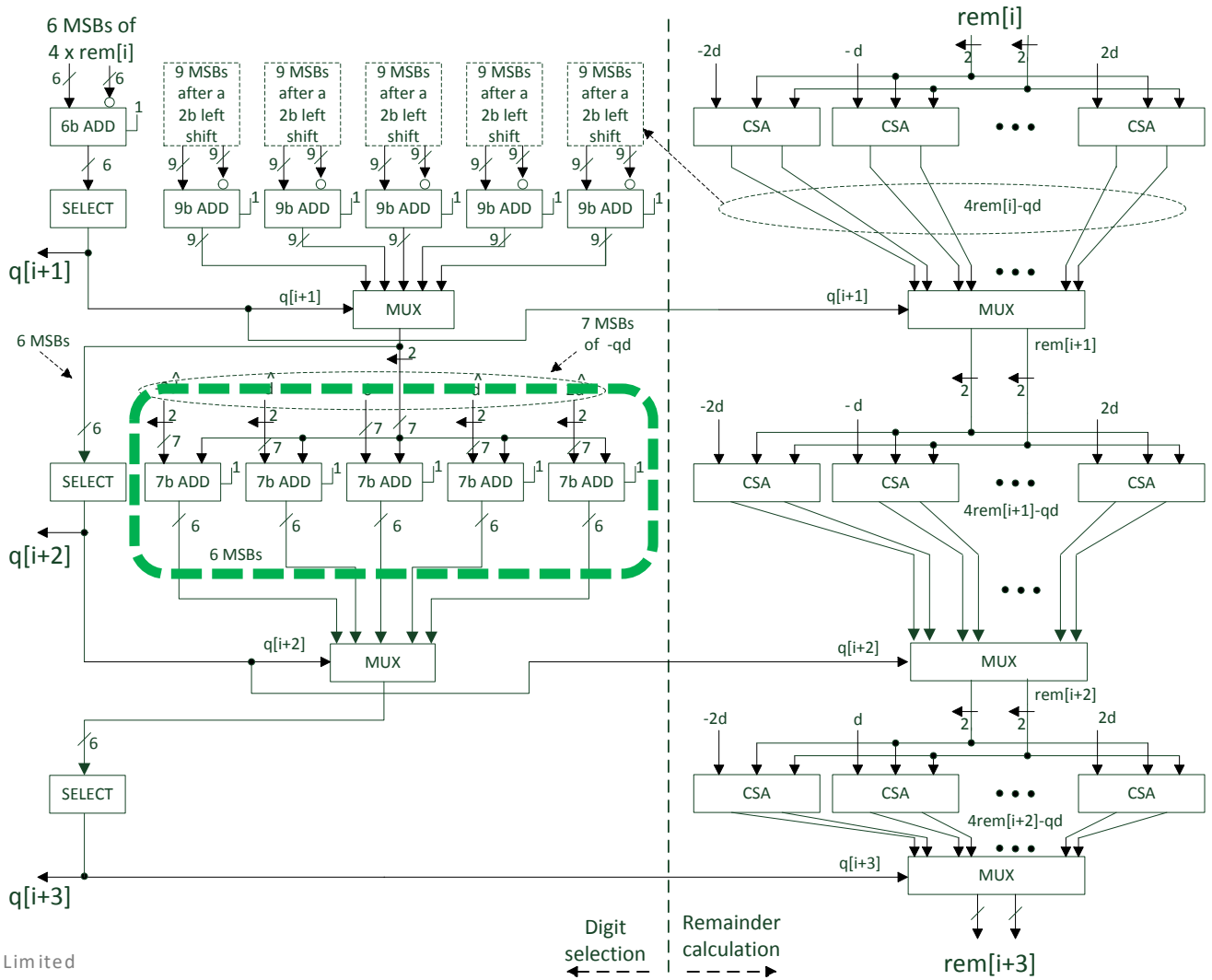


Remainder update

arm

Radix-64 Iteration

Digit Selection



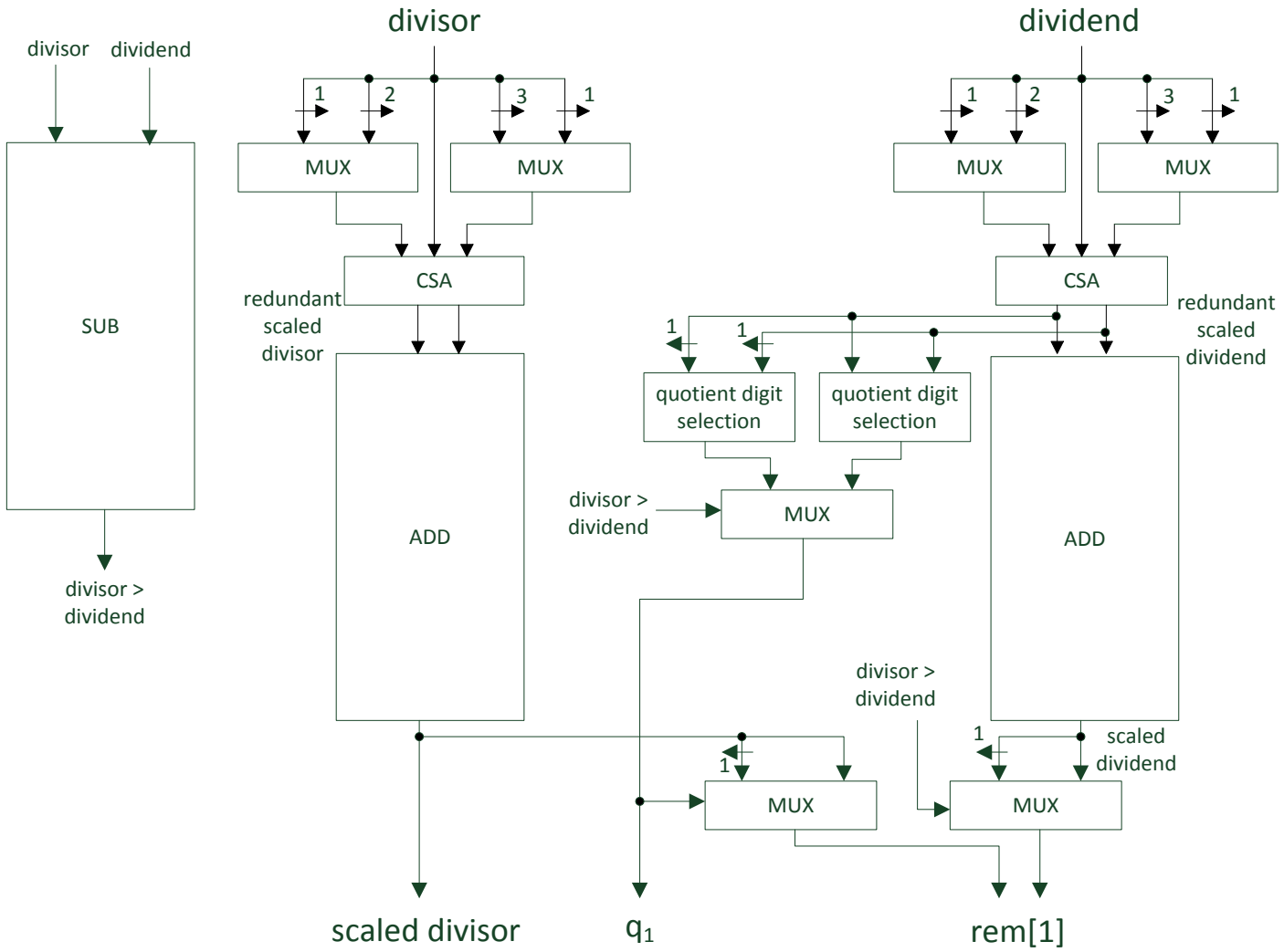
Remainder update

arm

First Cycle: Operands Pre-Scaling

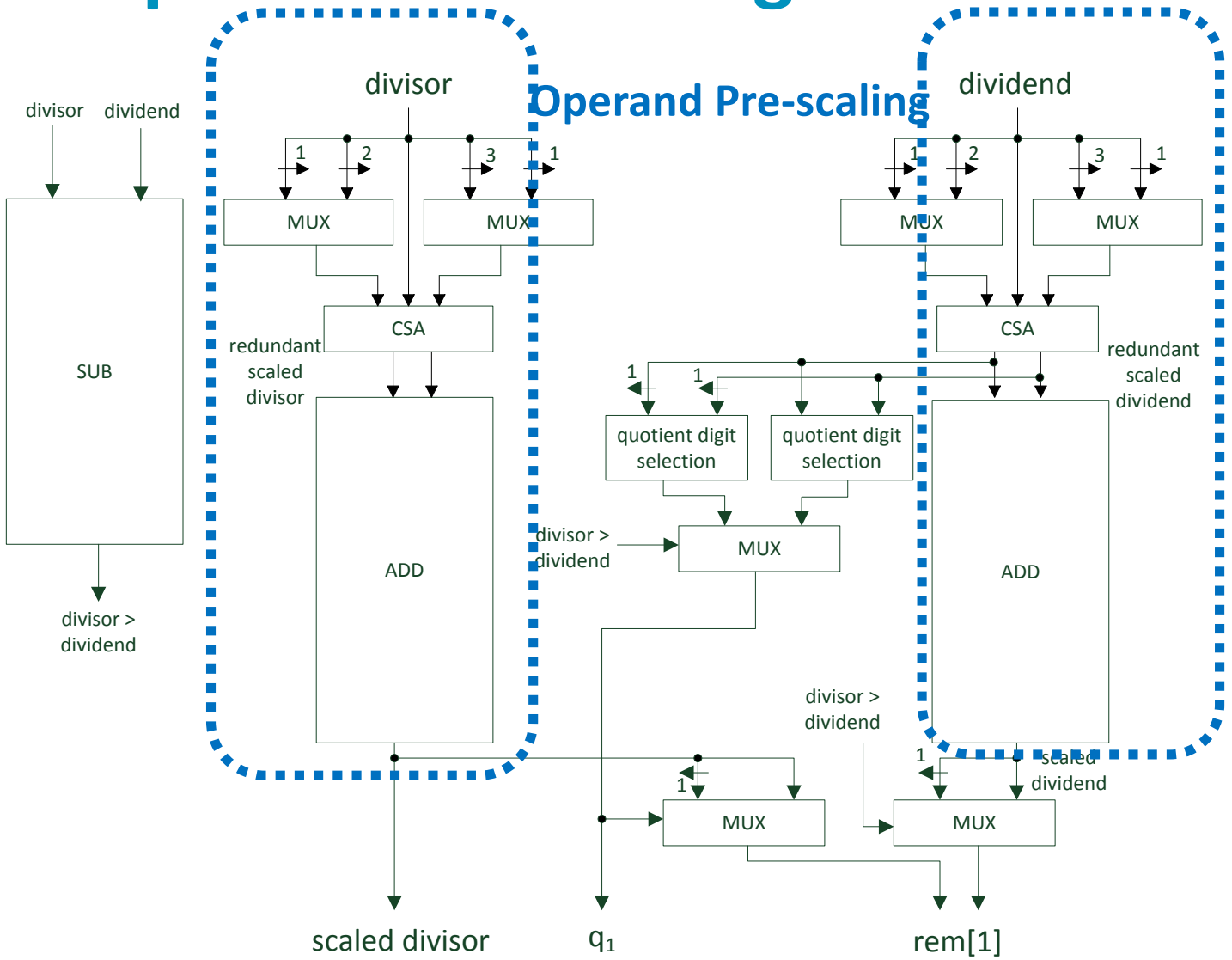
1. Scaling of divisor and dividend
   - Divisor close to 1 to have a simpler digit selection function
   - Digit selection function depends only on the remainder, it does not depend on the divisor
   - Dividend is scaled as well to preserve the result
2. Operand comparison
   - If $x < d$ the dividend is left-shifted by 1 bit, $q = 2x/d$
   - Result in $[1,2)$
   - Makes rounding easier: only a guard bit, there is not a rounding bit
3. Integer quotient-digit calculation
   - Result is in $[1,2)$, then integer digit is $\{+1, +2\}$
   - Simplified selection function
   - Replicated for $x > d$ and $x \leq d$
4. Initial remainder
   - The scaled divisor and scaled dividend are used for the initial redundant remainder
   - Includes 1-bit left shift if $x < d$

First Cycle: Operands Pre-Scaling



arm

First Cycle: Operands Pre-Scaling



$$M \times d \in [1 - 1/64, 1 + 1/8]$$

Scaling factor

$$M = 1 + b \times 2^{-3}$$
$$0 \le b \le 8, b \ne 7$$

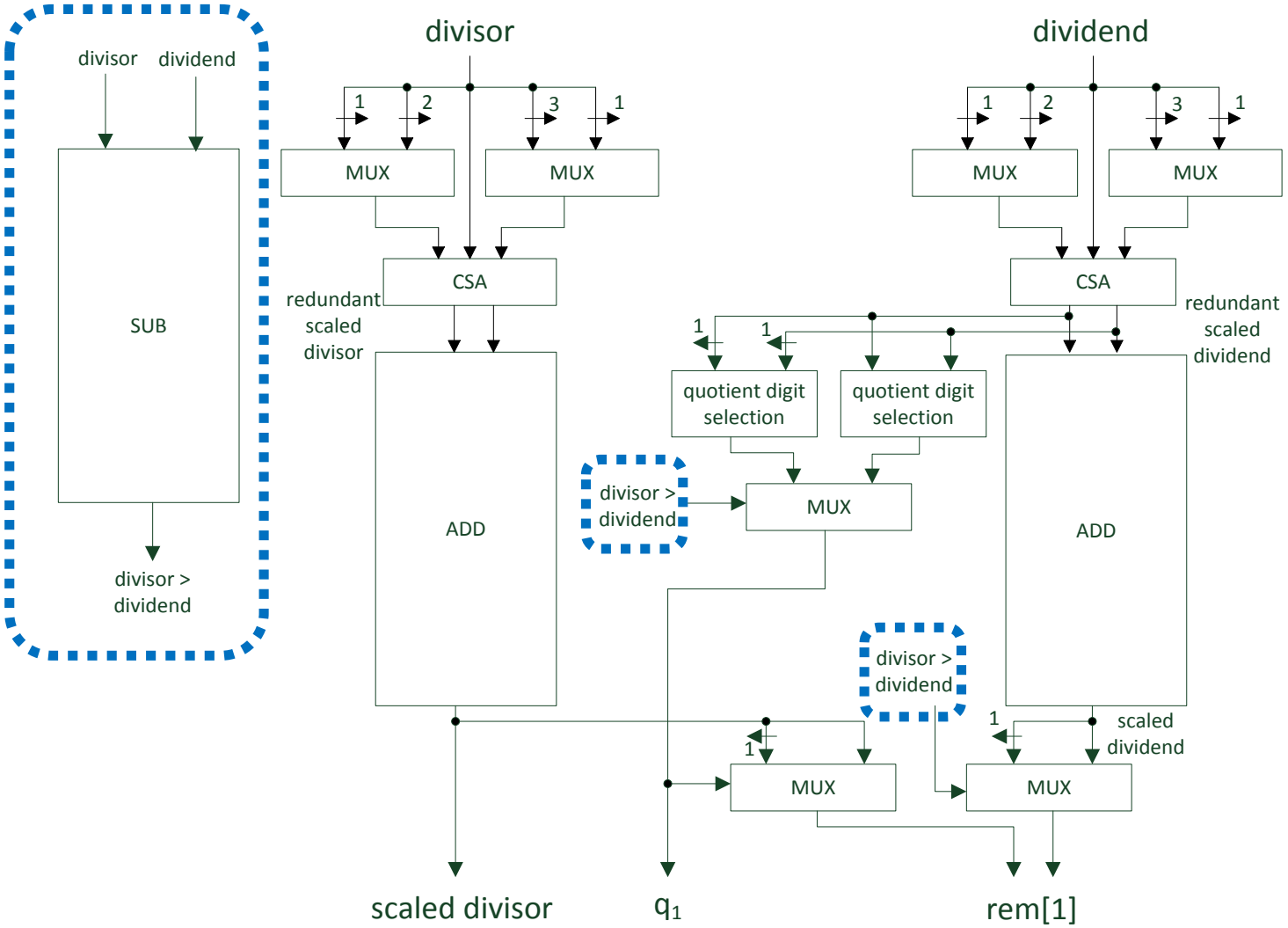
1.xxx	M
000	1+1/2+1/2
001	1+1/4+1/2
010	1+1/2+1/8
011	1+1/2+0
100	1+1/4+1/8
101	1+1/4+0
110	1+0+1/8
111	1+01/8

arm

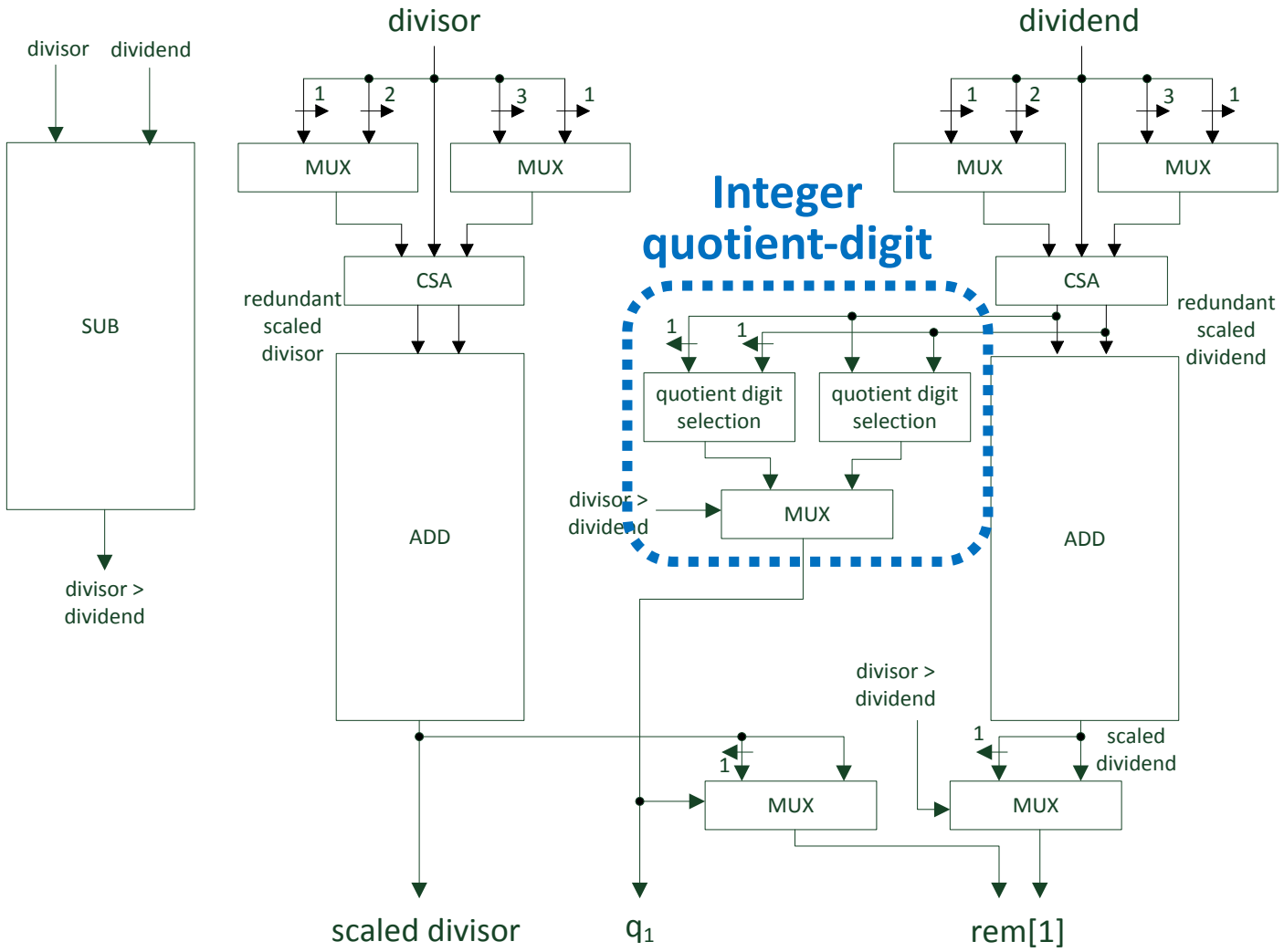
First Cycle: Operands Pre-Scaling

Operand Comparison

To save time, the non-scaled operands are compared

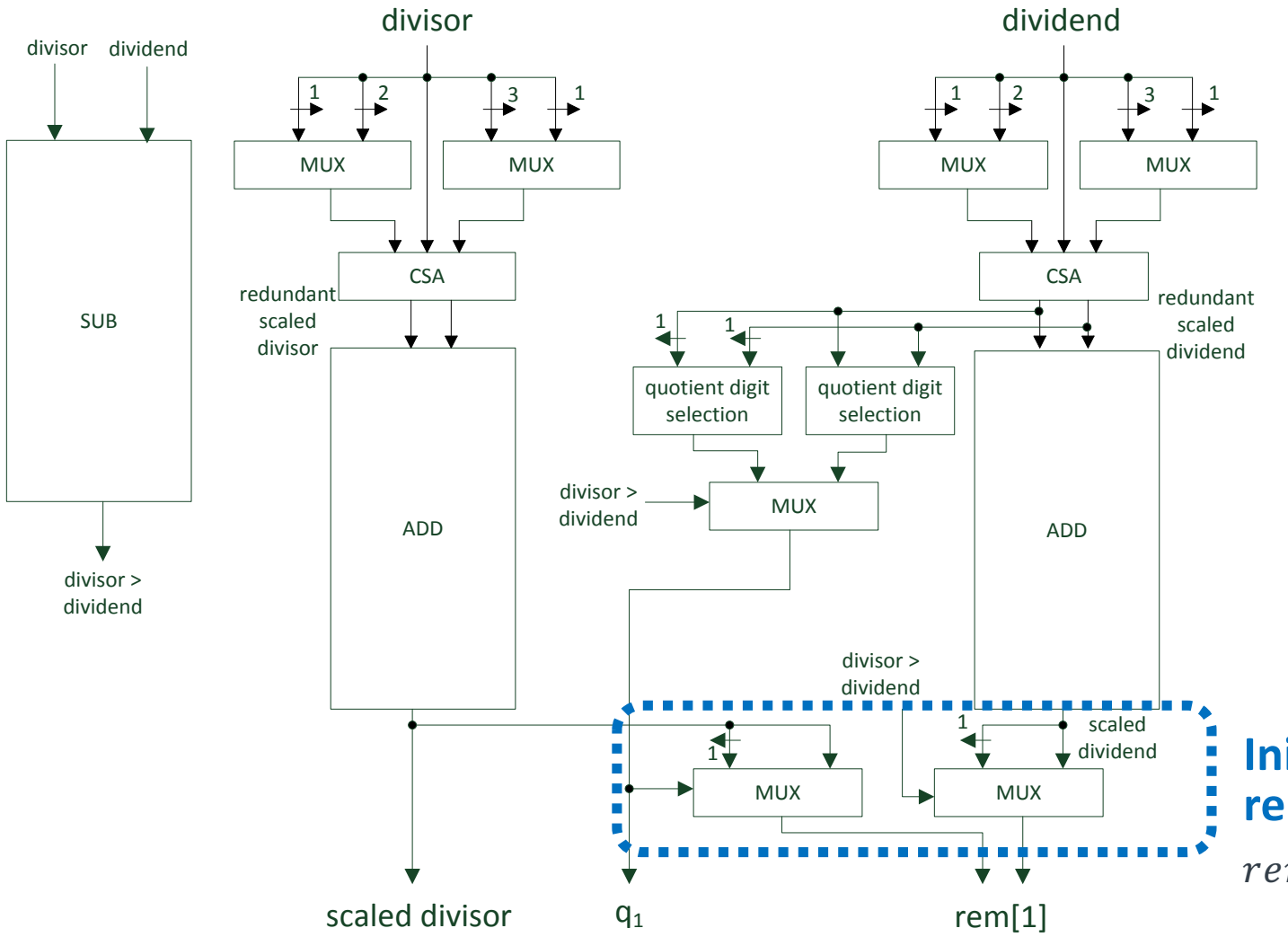


First Cycle: Operands Pre-Scaling



arm

First Cycle: Operands Pre-Scaling



Initial remainder

$$rem[1] = Mx - q_1 \times Md$$

arm

Evaluation

arm

Evaluation and Comparison

- Evaluation
  - Latency
  - Area
- Comparison with other recent processors
  - AMD K7 multiplicative division algorithm
  - AMD Jaguar multiplicative division algorithm
  - IBM z13 radix-4 digit-recurrence division algorithm
  - HAL Sparc multiplicative division algorithm
  - Intel 2018 (*lake) radix-1024 digit-recurrence division algorithm

* No information about the microarchitecture, just some notes with the radix and SP/DP latencies

Latency

	Double precision	Single precision	Half precision
Regular input, normalized result	11	6	4
Regular input, subnormal result	12	7	5
One subnormal input, normalized result	13	8	6
One subnormal input, subnormal result	14	9	7
Two subnormal input, normalized result	14	9	7

- Example: Single precision
  - Regular input, norm result: PSC – DGT – DGT – DGT – DGT – RND1
  - Regular input, subnorm result: PSC – DGT – DGT – DGT – DGT – RND1 – RND2
  - 1 subnormal input, norm result: UNP – NM – PSC – DGT – DGT – DGT – DGT – RND1
  - 1 subnormal input, subnorm result: UNP – NM – PSC – DGT – DGT – DGT – DGT – RND1 – RND2
  - 2 subnormal input, norm result: UNP – NM – NM – PSC – DGT – DGT – DGT – DGT – RND1

PSC - > pre-scaling, UNP -> unpacking, DGT -> digit iteration, NM -> normalization, RND1,2 -> rounding

arm

Latency - Comparison

	Algorithm	Half-precision	Single-precision	Double-precision
AMD K7	multiplicative	N/A	16	20
AMD Jaguar	multiplicative	N/A	14	19
IBM z13	Radix-4	N/A	23	37
HAL Sparc	multiplicative	N/A	16	19
Intel 2018 (lake)	Radix-1024	N/A	10	13
ARM	Radix-64	4	6	11

- Latencies include pre-processing (unpacking, pre-scaling), iteration cycles, and post-processing (rounding) and assuming normalized input/output
- Divider based on multiplicative algorithms:
  - Latency limited by the latency of multiplication or multiply-and-accumulate
  - Can be significative
- Radix-1024 divider (10 bit/cycle)
  - Pre-processing (probably) needs several cycles

arm

Area

- Large area
- Radix-64 iteration
  - fifteen 58-bit CSA (5 58-bit CSA per radix-4 iteration)
  - five 9-bit adder
  - five 7-bit adder
  - Selection of three quotient-digits
  - Muxes
- Pre-scaling
  - Three 58-bit adder
  - Two 58-bit CSA
  - Reduced selection logic
  - Muxes
- Rounding, Normalization

	DGT	Pre-proc	Total
58-bit adders	--	3	3
Small adders	10	2	12
58-bit CSA	15	2	17
Selection logic	3	2	5
58-bit 4-to-1 mux	6	--	6
58-bit 2-to-1 mux	--	6	6
narrow muxes	2	1	3

arm

Area - Comparison

- Multiplicative algorithms
  - Modest area, smaller than in the radix-64 divider
  - Reusing the existing FP multipliers
  - Look-Up table for initial seed
  - Muxes
- Radix-4 algorithm
  - Redundant remainder: 116-bit sum word, 28-bit carry word
  - 6 most-significant bits of the remainder are non-redundant
  - Area:
    - 3-to-2 CSA,
    - 2 small CPA,
    - digit selection table
- Radix-1024 algorithm
  - Large area (probably)
  - Pre-scaling: $1/d$ approximation, 2 multiplier
  - Iteration: small adder for digit selection, rectangular multiplier for remainder update

Conclusions

arm

Conclusions

- Radix-64 floating-point divider
  - 6 bits/cycle
  - Overlapping of three radix-4 iterations
  - Pre-scaling to have a simple selection function
  - Result in $[1,2)$
  - First iteration in parallel with the pre-scaling
- Low latency fp divider
  - 11, 6, and 4 cycles for double, single, and half-precision
  - Additional cycles in case of subnormal input/output
- Smaller latency than other implementations, although area is larger
- Integer division could be easily integrated
- Shared logic with floating-point square-root

arm

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
תודה

Floating-Point Division

- Iterative digit-recurrence algorithm
  - In a radix-r algorithm each iteration computes a radix-r digit of the quotient.
  - A radix-r digit represents $\log_2 r$ bits of the quotient
- Number of iterations depends on the result precision and on the radix
- Several cycles
  - Unpacking
  - Pre-scaling
  - Normalization (1 cycle per subnormal input)
  - Digit calculation (several cycles)
  - Rounding (2 rounding cycles if quotient is subnormal)