33rd IEEE International Symposium on Computer Arithmetic

ARITH 2026

Fulda, Germany. June 28 - July 1, 2026.

Enhancing Digit-Recurrence Floating-Point HUB Division

IEEE TRANSACTIONS ON COMPUTERS, VOL. 75, NO. 1, JANUARY 2026

Prof. Dr. Julio Villalba-Moreno
Dept. Computer Architecture
University of Malaga
SPAIN

ANDALUCIA TECH
Campus de Excelencia Internacional

Talk Outline

- Introduction and motivation
- HUB format
  - Floating point HUB numbers
  - Advantages and drawbacks
- Division under FP HUB
  - Digit recurrence algorithm
  - Data path bit-width and number of iterations
  - On-the-fly conversion
- Techniques to improve the digit-recurrent division in HUB
  - Operands scaling, prediction, high radix
- Evaluation
- Summary and conclusion

Introduction and motivation

Improve previous design

- Extension of techniques that improve the execution time of conventional floating-point division to the HUB format, such as operand scaling, prediction, and high radix.
- A thorough analysis of the on-the-fly conversion from redundant to conventional arithmetic in the HUB approach.
- Correction of a mistake in a previous HUB division paper [9].
- Application of the HUB representation to a recent high-radix design [2], proving that the HUB approach results in a simpler design due to rounding.
- Experimental results that corroborate the expected predictions on area, power and time.

[2] J. D. Bruguera, "Radix-64 floating-point divider," in *25th IEEE Symposium on Computer Arithmetic (ARITH 2018)*, June 2018, pp. 84–91.

[9] J. Villalba-Moreno, "Digit recurrence floating-point division under HUB format," in *23rd IEEE Symposium on Computer Arithmetic (ARITH 2016)*, July 2016, pp. 79–86.

HUB format

HUB format

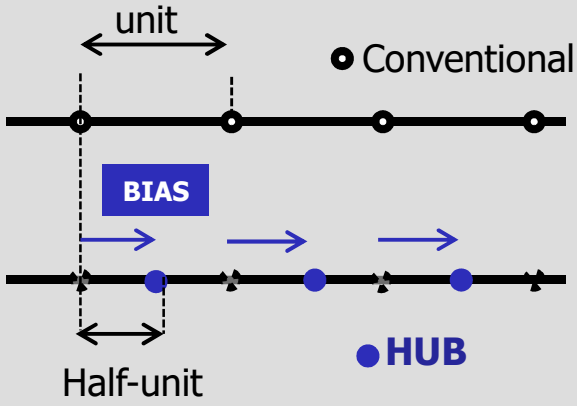
- **HUB** = **H**alf-**U**nit **B**iased format

Digit-vector

$$X = (X_{n-1}, X_{n-2}, \cdots X_1, X_0, X_{-1}, \cdots, X_{-f})$$

Conventional number in radix β

$$X = \left\lceil \sum_{i=-f}^{n-1} X_i \cdot \beta^i \right\rceil$$

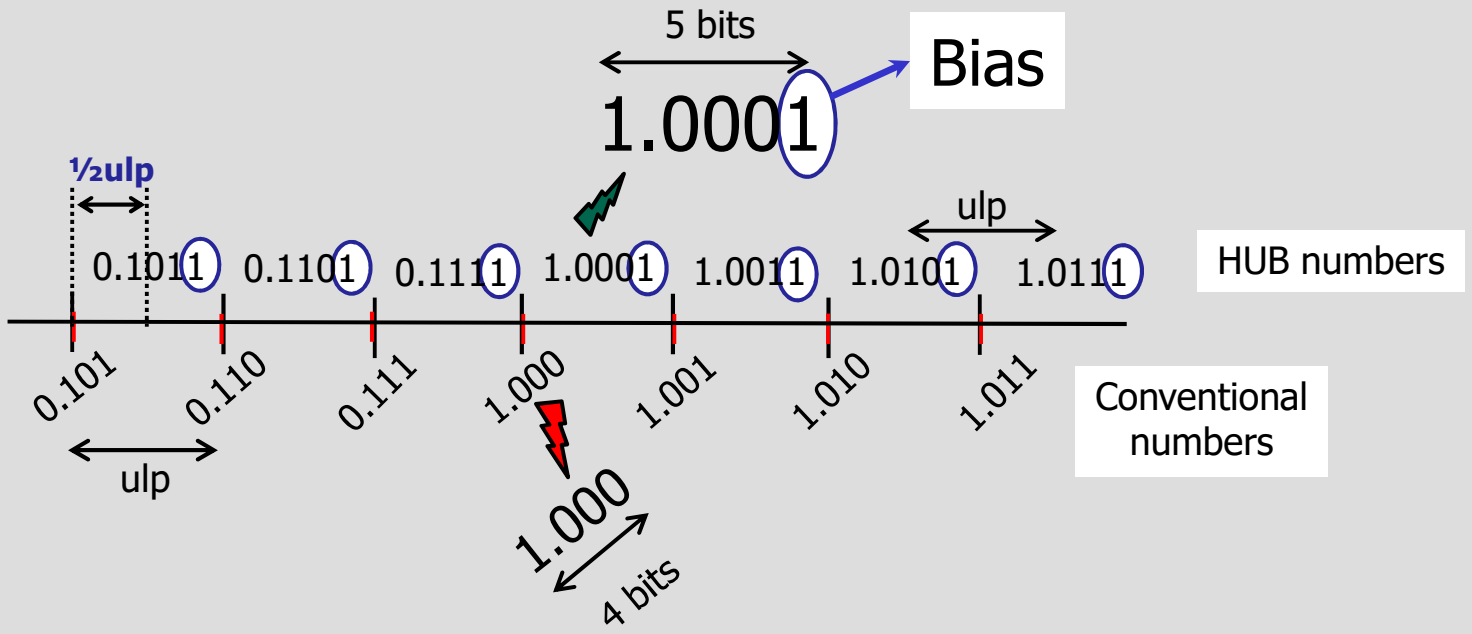


HUB number in radix β

$$X = \left\lceil \sum_{i=-f}^{n-1} X_i \cdot \beta^i \right\rceil + \underbrace{\frac{\beta}{2} \cdot \beta^{-f-1}}_{\text{BIAS}}$$

HUB format

- Example (4 bits, radix 2)



HUB format

- The extra LSB → ILSB Implicit Least Significant Bit

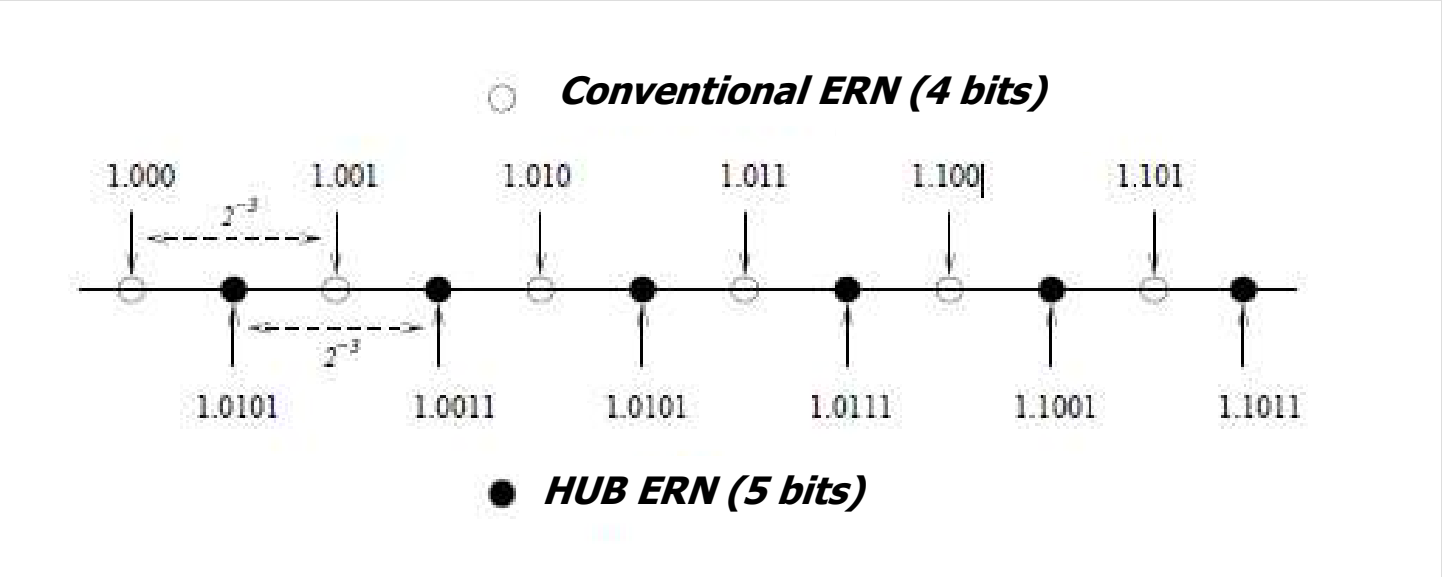
x x x . x x x x x x x x x x x x 1

ILSB

  - Value: 1
    - ➢ Implicit bit (ILSB)
      - Not stored
      - Not transmitted
      - Not needed for representation
    - ➢ Required when operating

HUB format

- Exactly representable numbers (ERN)



  - The set of ERN is different for both representations
  - Set(HUB) ∩ Set(Conventional) = Ø (Disjoint sets)
  - Disantance between two consequtive numbers is the same
  - The amount of ERN is the same
    - Both representations have the same precision

HUB format

- Floating point HUB number in radix-2 *(β=2)*

$$(S_x, M_x, E_x) \qquad x = (-1)^{S_x} M_x 2^{E_x}$$

E_x → Exponen (conventional)
M_x → Significand (HUB magnitude)
S_x → Sign (conventional)

– Normalized HUB significand: $1 < M_x < 2$

Digit-vector

$$M_x = (M_{x_0}, M_{x_{-1}}, M_{x_{-2}}, \cdots, M_{x_{-f}})$$

$$M_x = \left[\sum_{i=0}^{f} M_{x_i} \cdot 2^{-i}\right] + \underbrace{2^{-f-1}}_{\text{BIAS}}$$

$$M_x = 1.M_{x_{-1}} M_{x_{-2}} \cdots M_{x_{-f}} 1$$

HUB format

- Normalized Floating point HUB number in radix-2

$$M_x = \left\lfloor \sum_{i=0}^{f} M_{x_i} \cdot 2^{-i} \right\rfloor + 2^{-f-1}$$

ILSB

$$M_x = 1.M_{x_{-1}} M_{x_{-2}} \cdots M_{x_{-f}}$$

Conventional

$$M_x = 1.M_{x_{-1}} M_{x_{-2}} \cdots M_{x_{-f}} 1$$

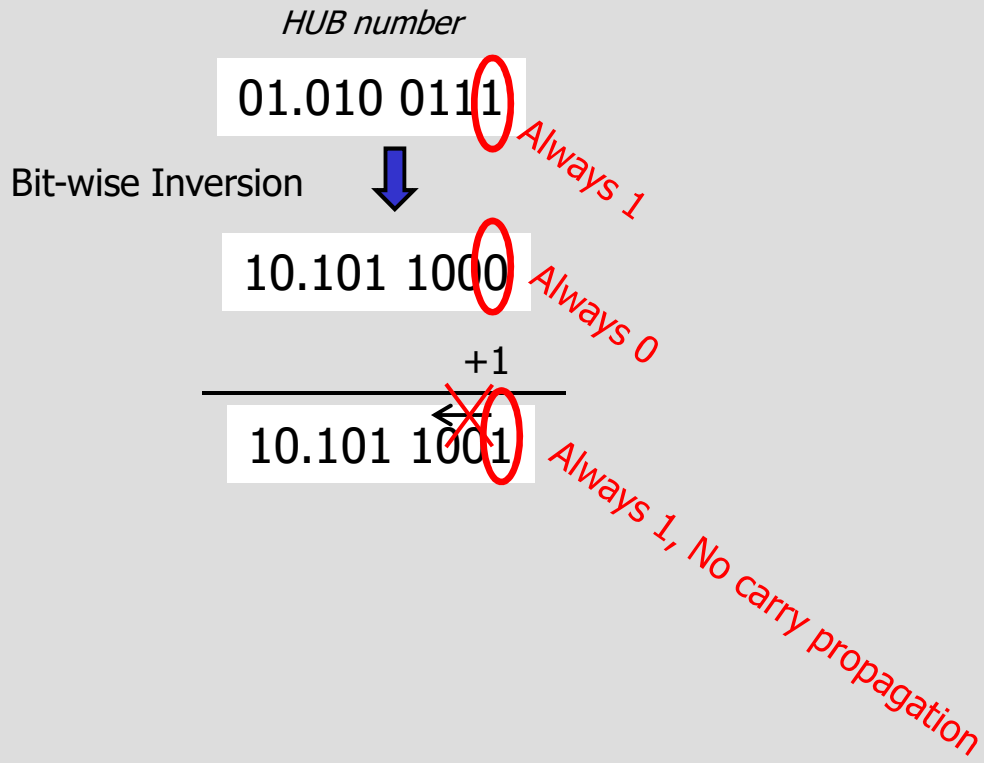
HUB

HUB format

- Advantages
  - Two's complement → bit-wise
  - Round to nearest → by truncation
  - No double rounding error
  - Simplicity
- Drawbacks
  - Not valid for integers
  - Other rounding modes require carry propagation
  - Not IEEE compliance

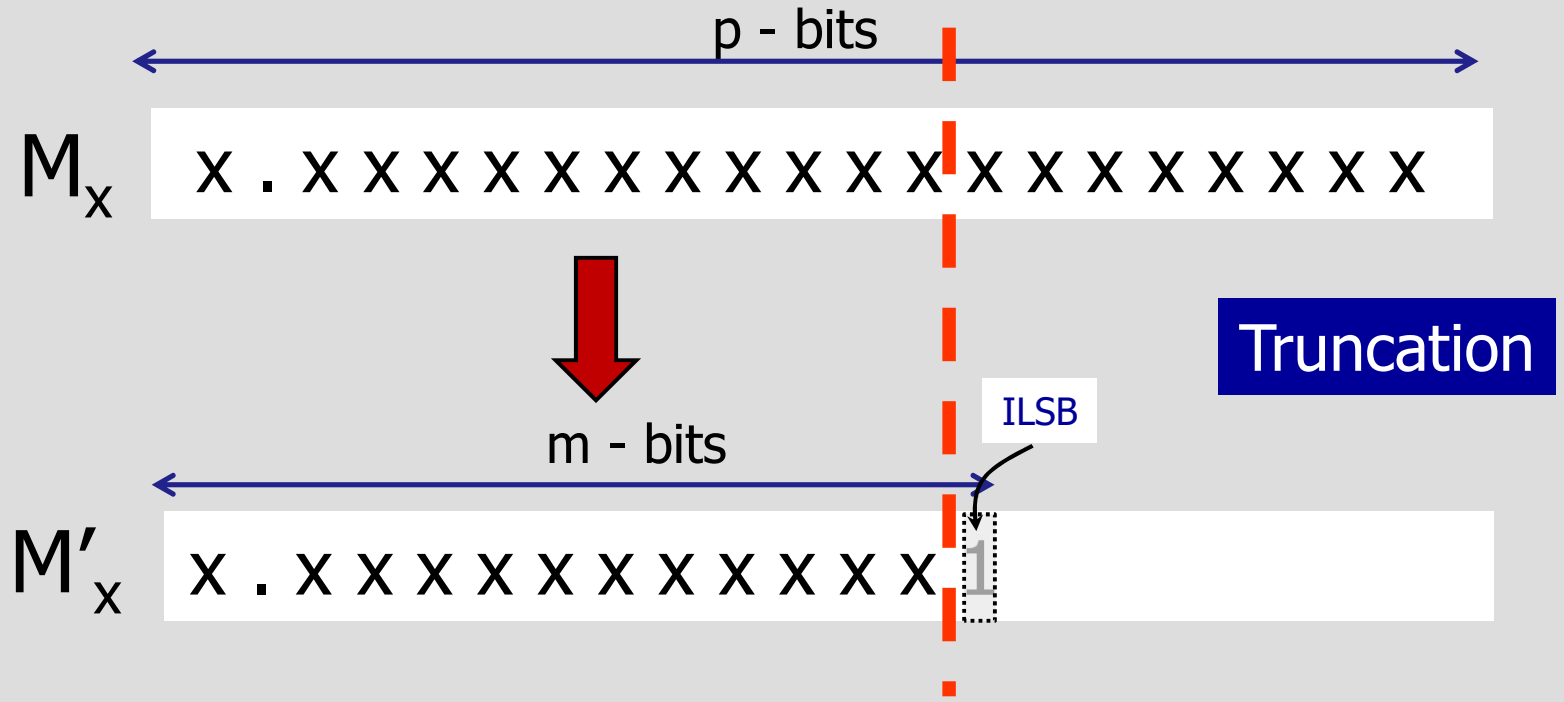
HUB format

- Two's complement of a HUB number
  - Invert the bits of the representative form (bit-wise)
    - » The ILSB=1 → No carry propagation



HUB format

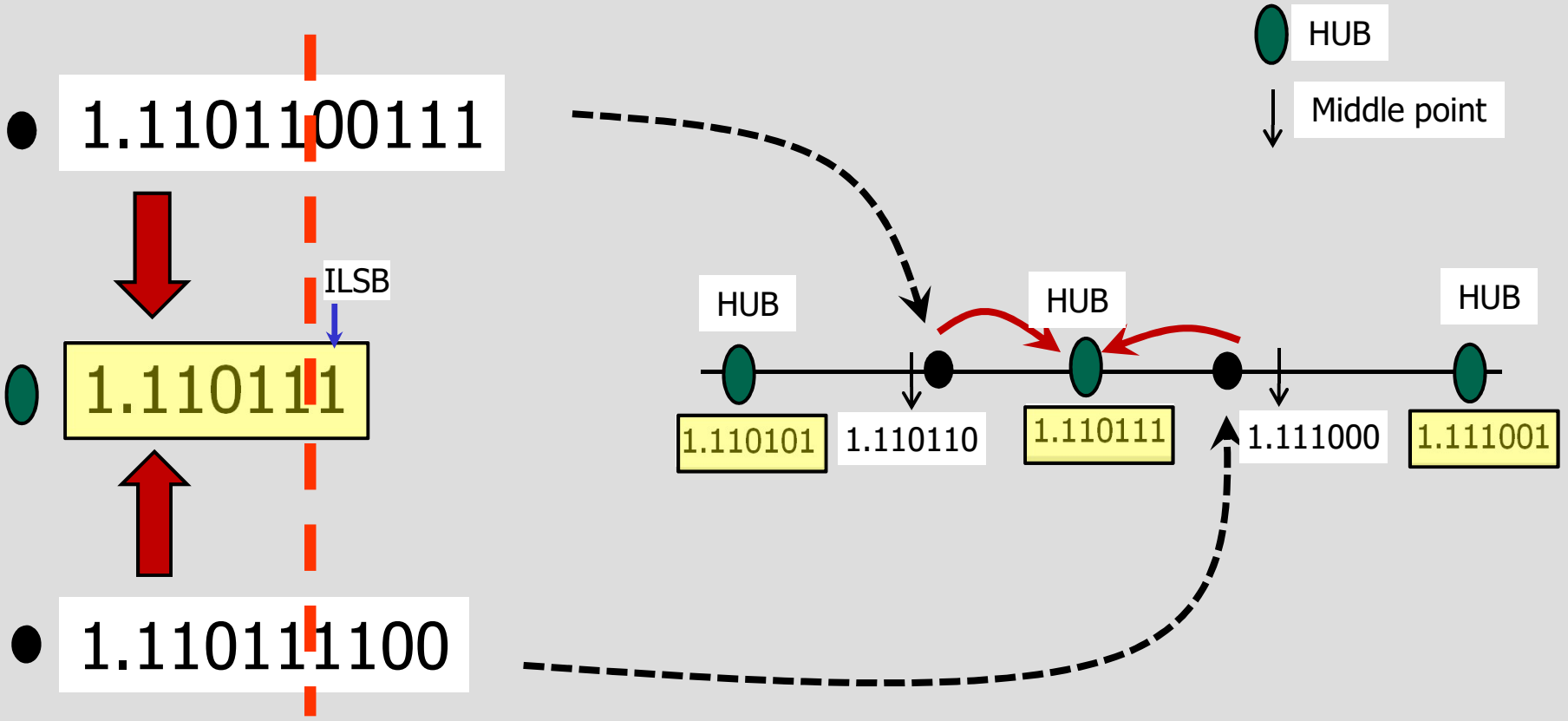
- Round to nearest of a HUB number: by truncation



Formally: M'[0:m-2]=M[0:m-2]

HUB format

- Proposed rounding: round to nearest by truncation



The FPHUB format

- Floating point HUB format
  - Include special cases and 5 standardized formats
    - FPHUB has only one rounding mode: Round to nearest
    - Exponent is excess $2^{n_{exp}-1}$

$$(S_x, E_x, M_x) \longrightarrow x = (-1)^{S_x}(1 + M_x + 2^{-f-1})2^{E_x - 2^{n_{exp}-1}}$$

Table 1: FPHUB-defined formats

Format	Total bits	Exp. bits	Significand bits	Precision (bits)
FPHUB16	16	5	10 (12 with implicit bits)	11
FPHUB32	32	8	23 (25 with implicit bits)	24
FPHUB64	64	11	52 (54 with implicit bits)	53
FPHUB128	128	15	112 (114 with implicit bits)	113
FPHUB256	256	19	236 (238 with implicit bits)	237

The FPHUB format

- Floating point HUB format
  - Special cases

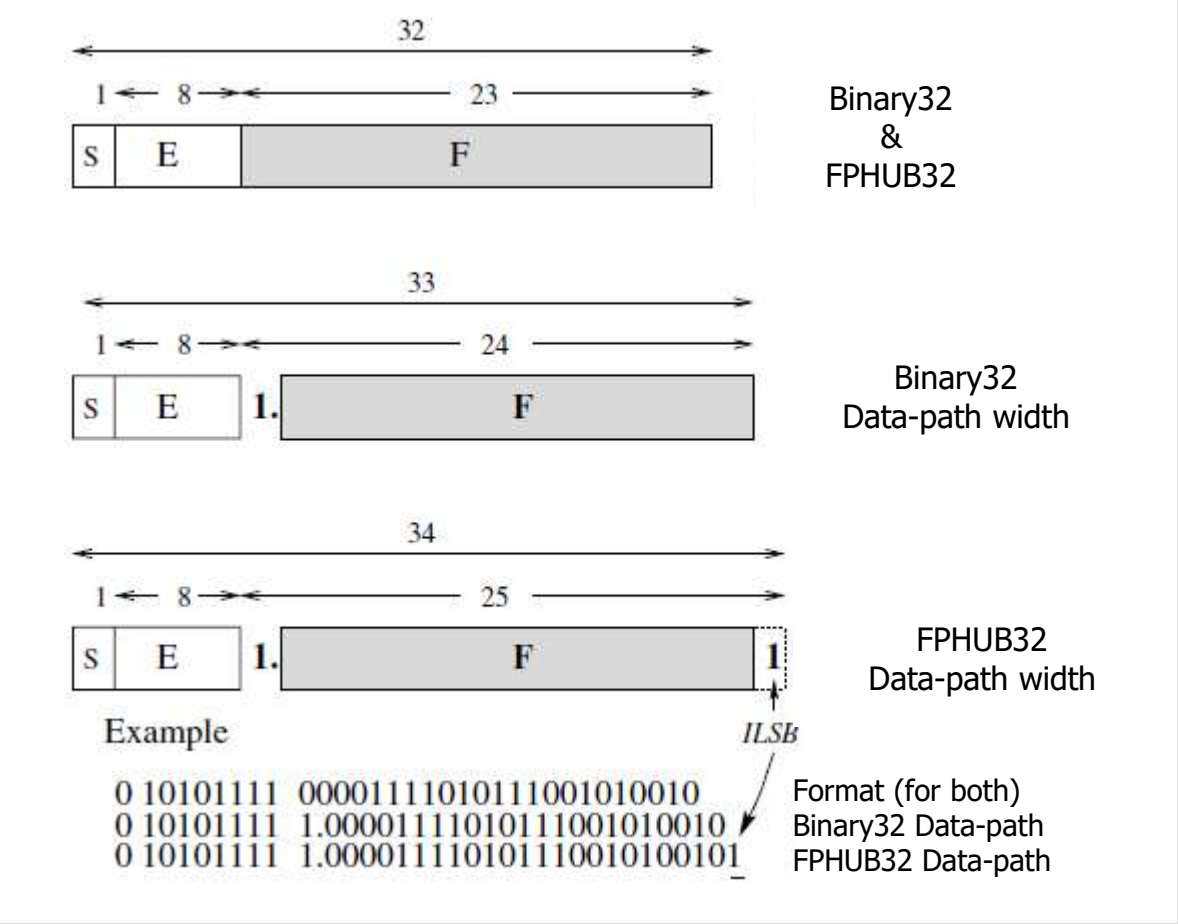
Case	Code	Example 23-fractional , 8-bit exponent
0	$(0,0,0)$	0 00000000 00000000000000000000000
0	$(1,0,0)$	1 00000000 00000000000000000000000
+1	$(0,128,0)$	0 10000000 00000000000000000000000
-1	$(1,128,0)$	1 10000000 00000000000000000000000
$+\infty$	$(0,255,2^{23}-1)$	0 11111111 11111111111111111111111
$-\infty$	$(1,255,2^{23}-1)$	1 11111111 11111111111111111111111

Table 2: Summary of special cases in FPHUB32

- $x + zero = x$
- $x * zero = zero$
- $x * one = x$
- $x \pm \infty = \pm\infty$
- $zero * one = zero$
- $x/one = x$
- $x/zero = \infty$ $(sign = sign(x))$
- $zero/x = zero$
- $zero/zero = +\infty$
- $x * \infty = \infty$ $(sign = sign(x))$
- $zero/\infty = \infty$
- $\infty/zero = \infty$
- $\infty * zero = \infty$

HUB format

IEEE binary32 and its HUB counterpart (FPHUB32)

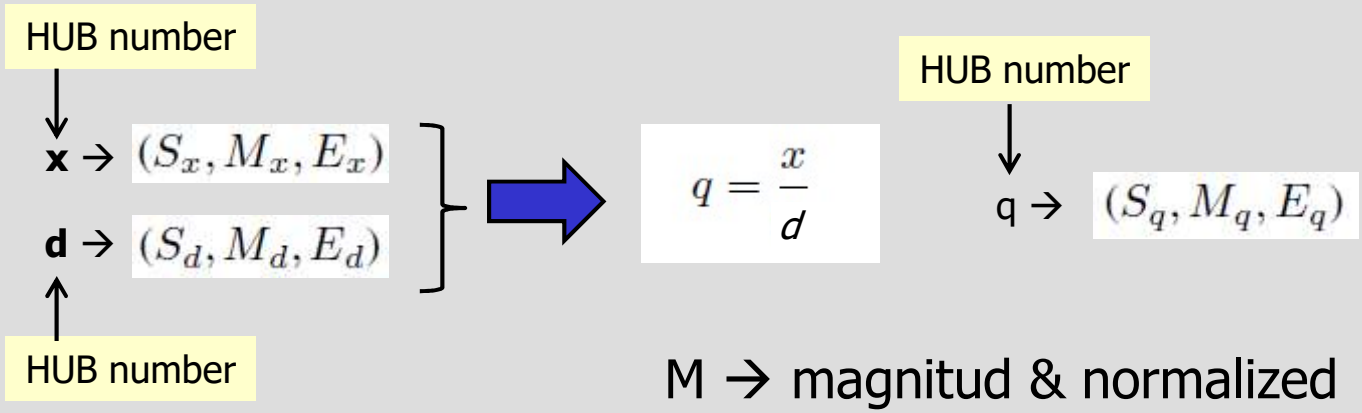


Floating point

- Efficiency of HUB format in Floating point implementations
  - Adder:
    - » Speed-up: 14%
    - » Area reduction: 38%
    - » Power reduction: 25% (single), 15% (double)
  - Multiplier
    - » Speedup: 17%
    - » Area reduction: 22%
    - » Power reduction: 2% (single), -2.6% (double)
  - Division
    - » Not actual implementation
  - Square root
    - » Not actual implementation
  - Fused Multiply–Add
    - » Speed-up: 16.7%
    - » Area reduction: 18%
    - » Power reduction: 14% (single)

Floating-point division for HUB

– Division of two FP HUB numbers



M → magnitud & normalized

Floating-point division for HUB

– Digit-recurrence algorithm [10]

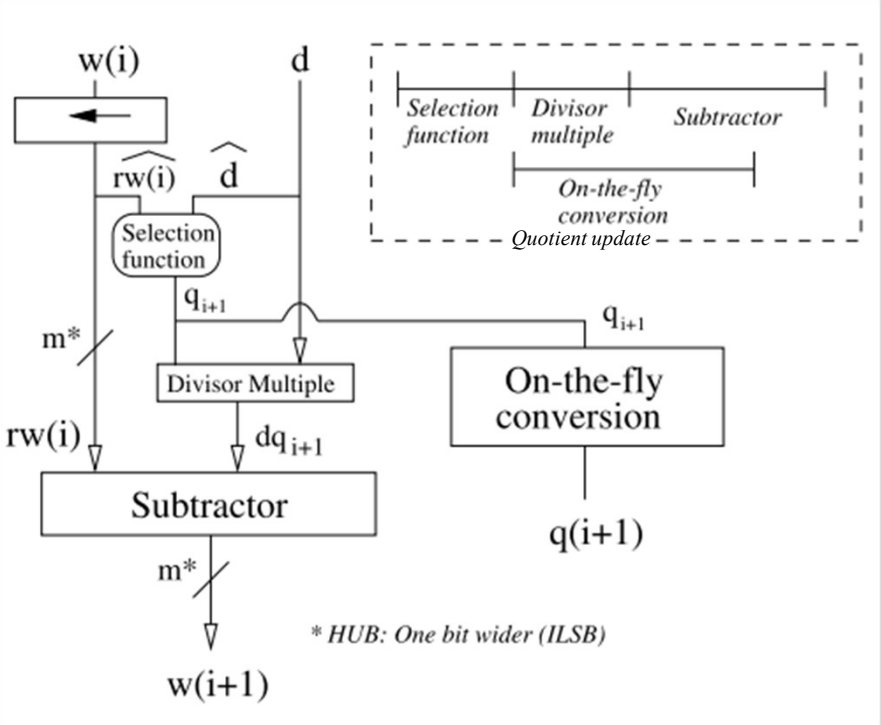
$$q(i) = q(0) + \sum_{j=1}^{i} q_j r^{-j}$$

$q_i \in [-a, a]$
$a \geq \lceil r/2 \rceil$

$$\rho = \frac{a}{r-1}, \qquad \frac{1}{2} < \rho \leq 1$$

Residual $w(i) = r^i(x - dq(i))$

Bound $|w(i)| \leq \rho \cdot d$



Recurrence $w(i+1) = rw(i) - dq_{i+1}$

Selection function: function of $\widehat{rw(i)}$ & $\widehat{d}$

Floating-point division for HUB

- Digit recurrence algorithm for FP HUB

M_x = 1.xxx…xxx1 $\quad 1 < M_x < 2$

M_d = 1.ddd…ddd1 $\quad 1 < M_d < 2$

$\Longrightarrow$ $M_q \in (½, 2)$ $\Longrightarrow$ Normalization required

- Initialization step
  - w(0) = x/2 if ρ =1 (maximum redundancy)
  - w(0) = x/4 if ρ <1
- Termination step
  - Correction if final negative residual (w(N)<0?)
  - Correction due to initialization (1 or 2 positions left shift)
  - Normalization if quotient ϵ (½<1)
  - **Rounding-to-nearest: by truncation / adding 1 ulp (conventional)**
    - » No overflow in HUB/overflow checking (conventional)
  - Check zero condition if exact quotient is needed

Floating-point division for HUB

- Data path bit-width for computing w(i+1)
  - m bits mantissa (m-1 in conventional)
  - 1 sign bit
  - Initial scaling: 1 bit (if ρ =1) or 2 bits (if ρ <1)
  - $\log_2 r$ bits due to r·w(i)

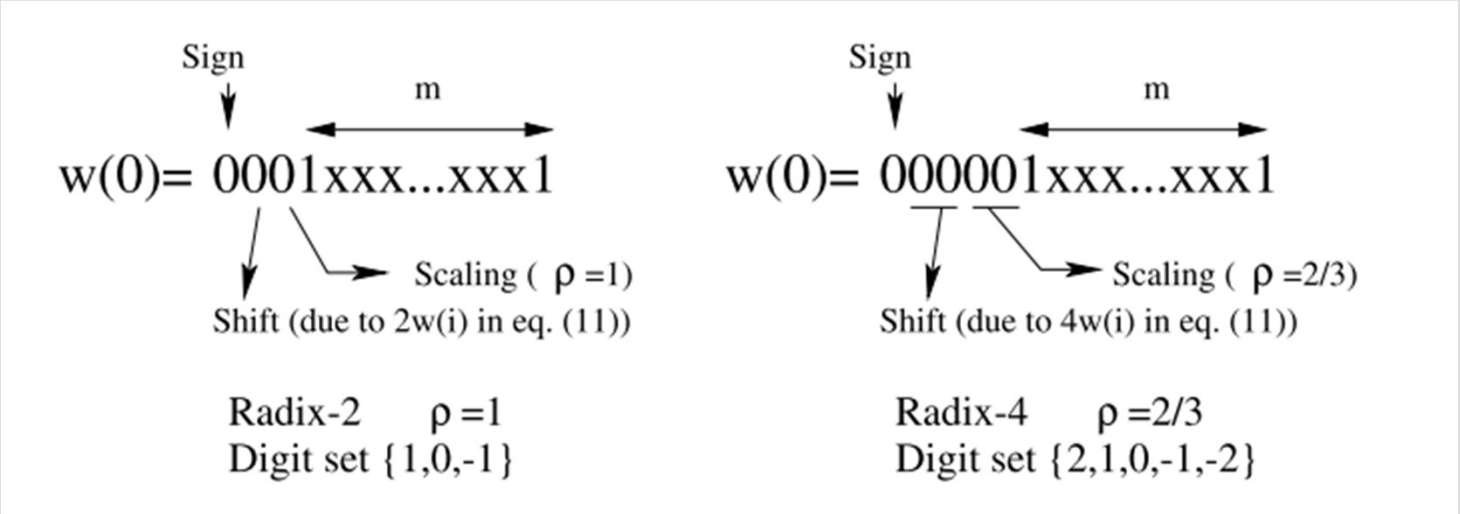
FPHUB

p= $m + 3 + \log_2 r - \lfloor \rho \rfloor$ bits

p= $m + 2 + \log_2 r - \lfloor \rho \rfloor$ bits

Conventional

- Example:



Conclusion: FPHUB uses a 1-bit wider datapath for computing the residual

Floating-point division for HUB

- Number of iterations N
  - Number of bits of the result (h) in HUB
    - *m* bits mantissa
    - 1 bit due normalization if q<1 (qϵ (½,2)), bit G
    - Initial scaling: 1 bit (if ρ =1) or 2 bits (if ρ <1), bit S

$$h = m + 1 + (2 - \lfloor \rho \rfloor) = m + 3 - \lfloor \rho \rfloor$$

  - Number of bits of the result (h) in conventional
    - *m-1* bits mantissa
    - 1 bit due normalization if q<1 (qϵ (½,2)), bit G
    - Initial scaling: 1 bit (if ρ =1) or 2 bits (if ρ <1), bit S
    - 1 rounding bit, bit R

$$h = (m - 1) + 1 + 1 + (2 - \lfloor \rho \rfloor) = m + 3 - \lfloor \rho \rfloor$$

$$N = \left\lceil \frac{h}{\log_2 r} \right\rceil$$

Floating-point division for HUB

- Number of iterations N
  - Number of bits of the result (h) in HUB

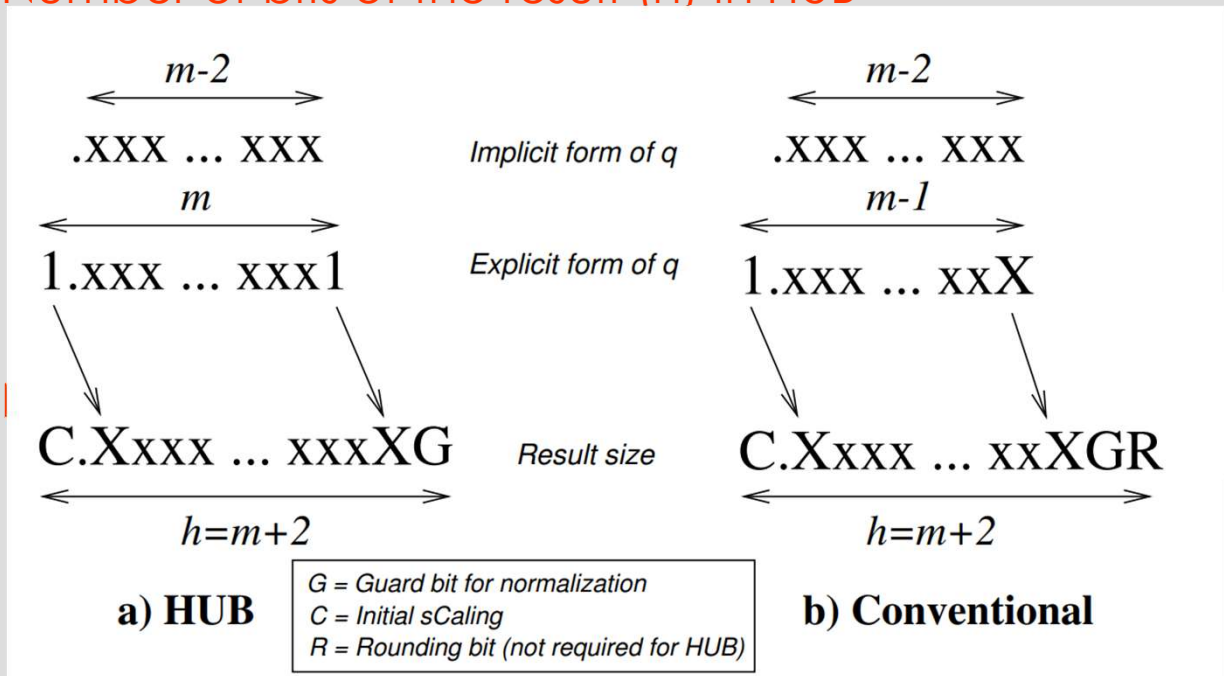


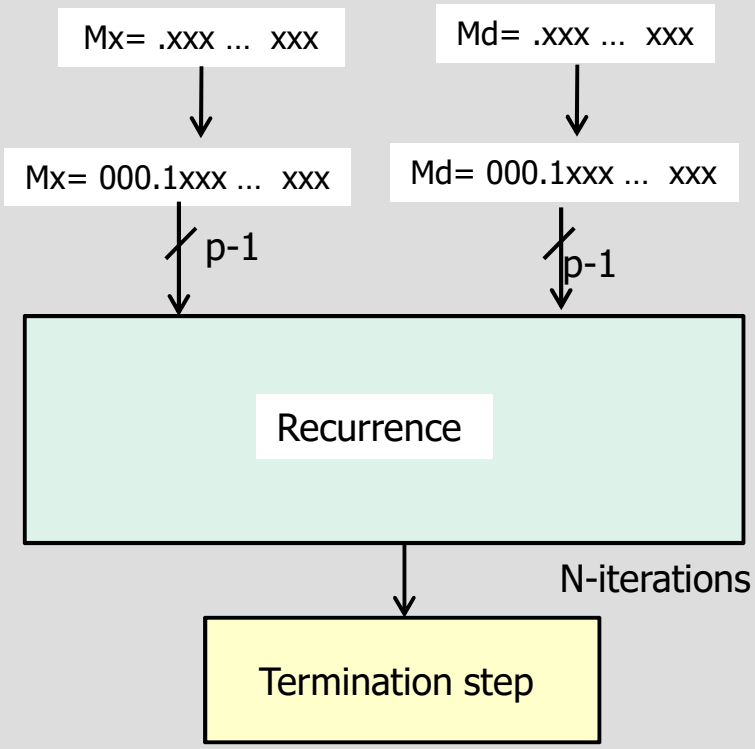
Fig. 5. Size of the result h (required to calculate the number of iterations of the recurrence) for HUB and IEEE for the same precision and $\rho = 1$ ($h = m + 3 - \lfloor \rho \rfloor = m + 2$)

$$N = \left\lceil \frac{h}{\log_2 r} \right\rceil$$

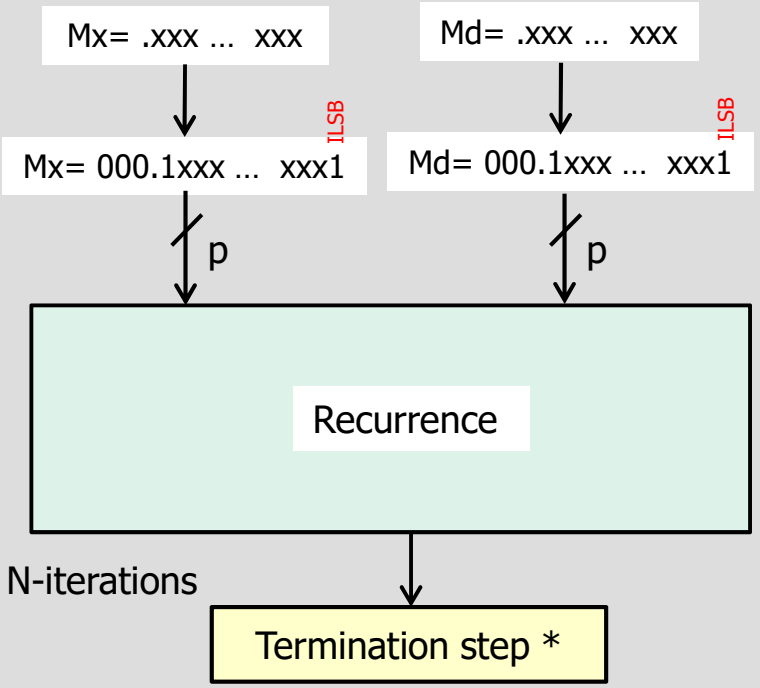
$h =$

Conclusion: same number of iterations for HUB and for its conventional counterpart

Floating-point division for HUB



Conventional with round to nearest



HUB with round to nearest (truncation)

On-the-fly conversion and rounding

- On-the-fly conversion
  - Overlapping the final conversion from redundant to conventional with the iterations
  - Rounding, correction and normalization is integrated in the on-the-fly conversion of the last digit

On-the-fly conversion and rounding

– On-the-fly conversion (conventional & HUB)

Q(i) → *i* MSD of the converted quotient (digit vector)

$$Q(i) = \sum_{j=1}^{i} q_i r^{-i}$$

To avoid carry propagation:

$$QD(i) = Q(i) - r^{-i}$$

On-the-fly algorithm: Concatenation

$$Q(i+1) = \begin{cases} (Q(i) \parallel q_{i+1}) & \text{if } q_{i+1} \geq 0 \\ (QD(i) \parallel (r - |q_{i+1}|)) & \text{if } q_{i+1} < 0 \end{cases}$$

$$QD(i+1) = \begin{cases} (Q(i) \parallel q_{i+1} - 1) & \text{if } q_{i+1} > 0 \\ (QD(i) \parallel (r - 1 - |q_{i+1}|)) & \text{if } q_{i+1} \leq 0 \end{cases}$$

Last digit computation q_N

Conventional

- Round to nearest → addition of 1 ulp after regular iterations
  - Last digit out of range of the digit set [-a,a] if $a \geq r-2$
  - $q_N = a \rightarrow q_N+1 \notin [-a,a]$

$$q_N^* = q_N - sign + 1 + norm$$

- A new form *QR(i)* is defined to combine the correction, normalization and rounding:

$$QR(i) = Q(i) + r^{-i}$$

$$QR(i+1) = \begin{cases} (QR(i) \parallel 0) & \text{if } q_{i+1} = r-1 \\ (Q(i) \parallel q_{i+1}+1) & \text{if } -1 \leq q_{i+1} \leq r-2 \\ (QD(i) \parallel (r+1-|q_{i+1}|)) & \text{if } q_{i+1} < -1 \end{cases}$$

$$q_N^* \in \{-a, a+1\} \qquad u = q_N^* \bmod r$$

$$Q(N)_{rounded} = \begin{cases} QR(N-1) \parallel u & \text{if } q_N^* \geq r \\ Q(N-1) \parallel u & \text{if } 0 \leq q_N^* \leq r-1 \\ QD(N-1) \parallel u & \text{if } q_N^* < r \end{cases} \quad (27)$$

HUB

- Round to nearest → by truncation after regular iterations
  - Last digit inside range always

$$q_{N_{HUB}}^* = q_N - sign$$

- New form NOT required

$$Q(N)_{rounded} = \begin{cases} Q(N) & \text{if } sign = 0 \\ QD(N) & \text{if } sign = 1 \end{cases} \quad (30)$$

$$Q(N)_{normalized} = \begin{cases} Q(N)[0:m-2] & \text{if } Q(N)[0] = 1 \\ Q(N)[1:m-1] & \text{if } Q(N)[0] = 0 \end{cases} \quad (28)$$

On-the-fly conversion and rounding

– On-the-fly conversion for HUB

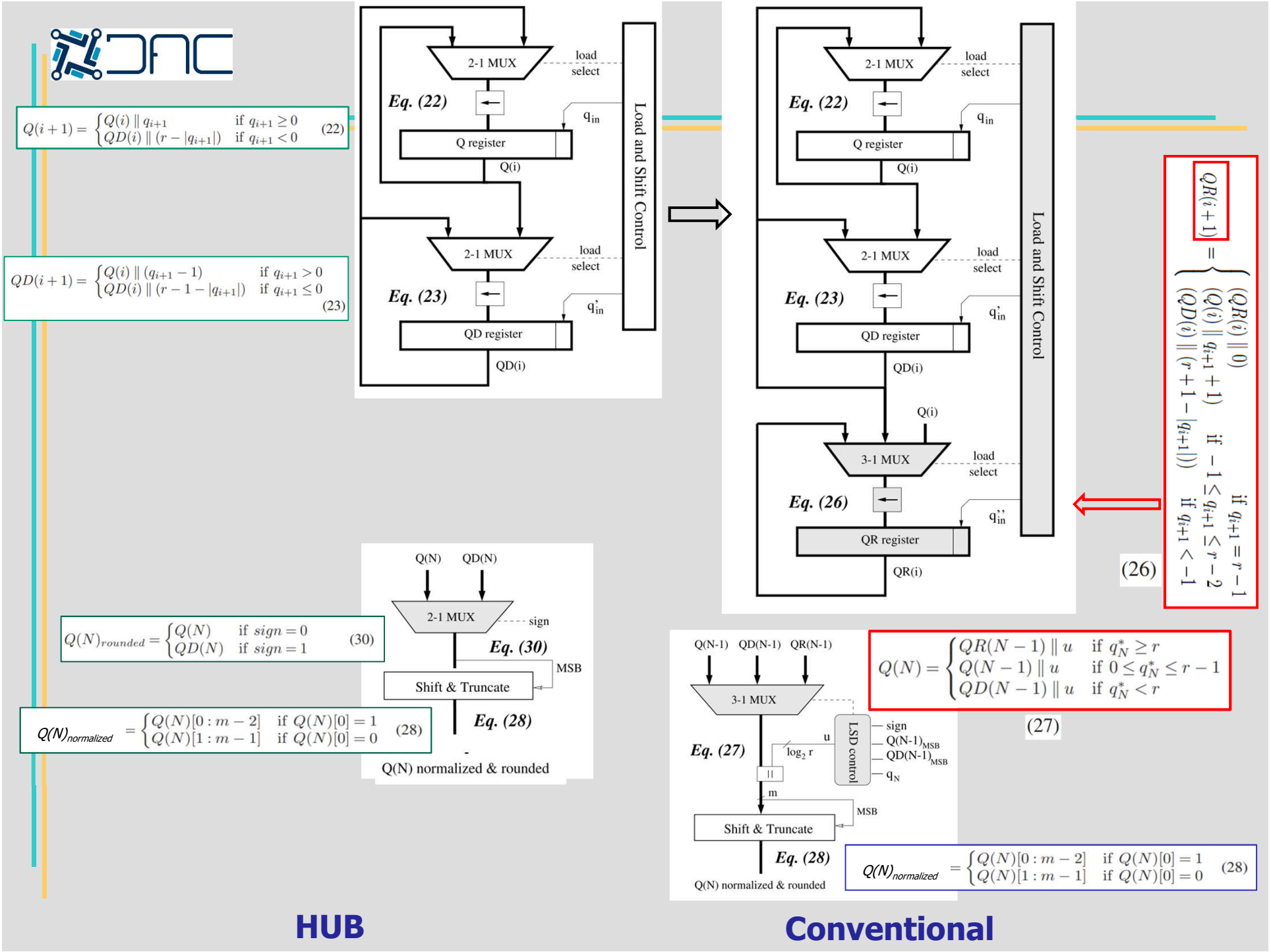
$$Q(i+1) = \begin{cases} (Q(i) \parallel q_{i+1}) & \text{if } q_{i+1} \geq 0 \\ (QD(i) \parallel (r - |q_{i+1}|)) & \text{if } q_{i+1} < 0 \end{cases}$$

$$QD(i+1) = \begin{cases} (Q(i) \parallel q_{i+1} - 1) & \text{if } q_{i+1} > 0 \\ (QD(i) \parallel (r - 1 - |q_{i+1}|)) & \text{if } q_{i+1} \leq 0 \end{cases}$$

– Round to nearest is carried out by truncation → No carry propagation → no addition of 1 ulp after regular iterations
  » Last digit is always inside the range of the digit set [-a,a]
– The form *QR(i)* is not required any more

$$QR(i) = Q(i) + r^{-i}$$

$$QR(i+1) = \begin{cases} (QR(i) \parallel 0) & \text{if } q_{i+1} = r - 1 \\ (Q(i) \parallel q_{i+1} - 1) & \text{if } -1 \leq q_{i+1} \leq r - 2 \\ (QD(i) \parallel (r + 1 - |q_{i+1}|)) & \text{if } q_{i+1} < -1 \end{cases}$$


$Q(i+1) = \begin{cases} Q(i) \parallel q_{i+1} & \text{if } q_{i+1} \geq 0 \\ QD(i) \parallel (r - |q_{i+1}|) & \text{if } q_{i+1} < 0 \end{cases}$ (22)
$QD(i+1) = \begin{cases} Q(i) \parallel (q_{i+1} - 1) & \text{if } q_{i+1} > 0 \\ QD(i) \parallel (r - 1 - |q_{i+1}|) & \text{if } q_{i+1} \leq 0 \end{cases}$ (23)
2-1 MUX
load select
Eq. (22)
q_{in}
Q register
Q(i)
2-1 MUX
load select
Eq. (23)
q'_{in}
QD register
QD(i)
Load and Shift Control
2-1 MUX
load select
Eq. (22)
q_{in}
Q register
Q(i)
2-1 MUX
load select
Eq. (23)
q'_{in}
QD register
QD(i)
Q(i)
3-1 MUX
load select
Eq. (26)
q''_{in}
QR register
QR(i)
Load and Shift Control
$QR(i+1) = \begin{cases} (QR(i) \parallel 0) & \text{if } q_{i+1} = r - 1 \\ (Q(i) \parallel q_{i+1} + 1) & \text{if } -1 \leq q_{i+1} \leq r - 2 \\ (QD(i) \parallel (r + 1 - |q_{i+1}|)) & \text{if } q_{i+1} < -1 \end{cases}$
(26)
Q(N) QD(N)
2-1 MUX
sign
Eq. (30)
MSB
Shift & Truncate
Eq. (28)
Q(N) normalized & rounded
$Q(N)_{rounded} = \begin{cases} Q(N) & \text{if } sign = 0 \\ QD(N) & \text{if } sign = 1 \end{cases}$ (30)
$Q(N)_{normalized} = \begin{cases} Q(N)[0:m-2] & \text{if } Q(N)[0] = 1 \\ Q(N)[1:m-1] & \text{if } Q(N)[0] = 0 \end{cases}$ (28)
Q(N-1) QD(N-1) QR(N-1)
3-1 MUX
Eq. (27)
u
$\log_2 r$
m
LSD control
sign
$Q(N-1)_{MSB}$
$QD(N-1)_{MSB}$
q_N
MSB
Shift & Truncate
Eq. (28)
Q(N) normalized & rounded
$Q(N) = \begin{cases} QR(N-1) \parallel u & \text{if } q^*_N \geq r \\ Q(N-1) \parallel u & \text{if } 0 \leq q^*_N \leq r-1 \\ QD(N-1) \parallel u & \text{if } q^*_N < r \end{cases}$
(27)
$Q(N)_{normalized} = \begin{cases} Q(N)[0:m-2] & \text{if } Q(N)[0] = 1 \\ Q(N)[1:m-1] & \text{if } Q(N)[0] = 0 \end{cases}$ (28)
HUB
Conventional

HUB division

- Conclusion
  - The residual datapath *w(i)*, is one bit wider in HUB
    - » Higher area and power in HUB
  - Rounding is simpler in HUB
    - » Lower area and power in HUB
  - On-the-fly conversion is simpler in HUB
    - » Lower area and power in HUB

An actual implementation is needed to determine whether the increase in datapath width in HUB is compensated for by the simpler rounding and on-the-fly conversion.

Improving the HUB division

- Is it possible to apply advanced division techniques to the HUB format?
  - Scaling of the operands
  - Prediction/speculation
  - High radix

Improving the HUB division

- Scaling of the operands
  - Objetive: selection function depending only of some MSBs of the residual [12] by prescaling of the divisor close to 1

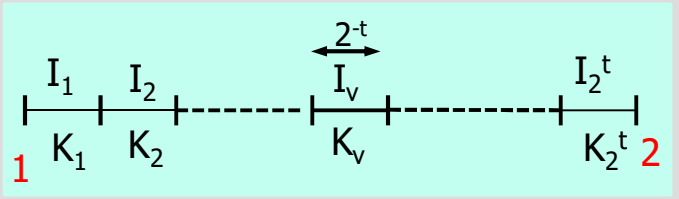
Selection function: sel($\widehat{rw(i)}$ & $\widehat{d}$) → sel($\widehat{rw(i)}$)

Dividend and divisor scaled by ***K***

$$1 - \alpha \leq d_s = K \cdot d \leq 1 + \beta$$

- The divisor range (d∈[1,2)) is partitioned into intervals $\mathbf{I_v}$ of length $\mathbf{2^{-t}}$, each associated with a different scaling factor.
- K_v Scale factor of Interval I_v

d ∈ [1,2) →



$$I_v = \left[2 - \frac{v}{2^t}, 2 - \frac{v-1}{2^t}\right) \quad 1 \leq v \leq 2^t$$

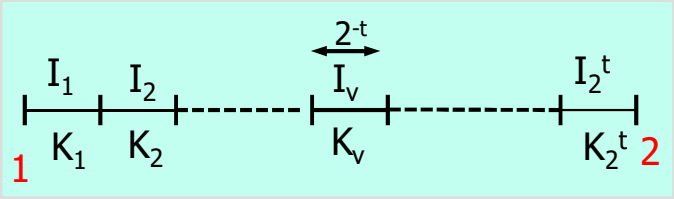
$$(1-\alpha)\frac{2^t}{2^{t+1}-v} \leq K_v \leq (1+\beta)\frac{2^t}{2^{t+1}+1-v}$$

$$2^t \geq \frac{(1-\alpha)}{\alpha+\beta}$$

Improving the HUB division

– Scaling of the operands in HUB

Conventional → $d \in [1,2)$ →



HUB → $d_{HUB} \in (1,2)$ →

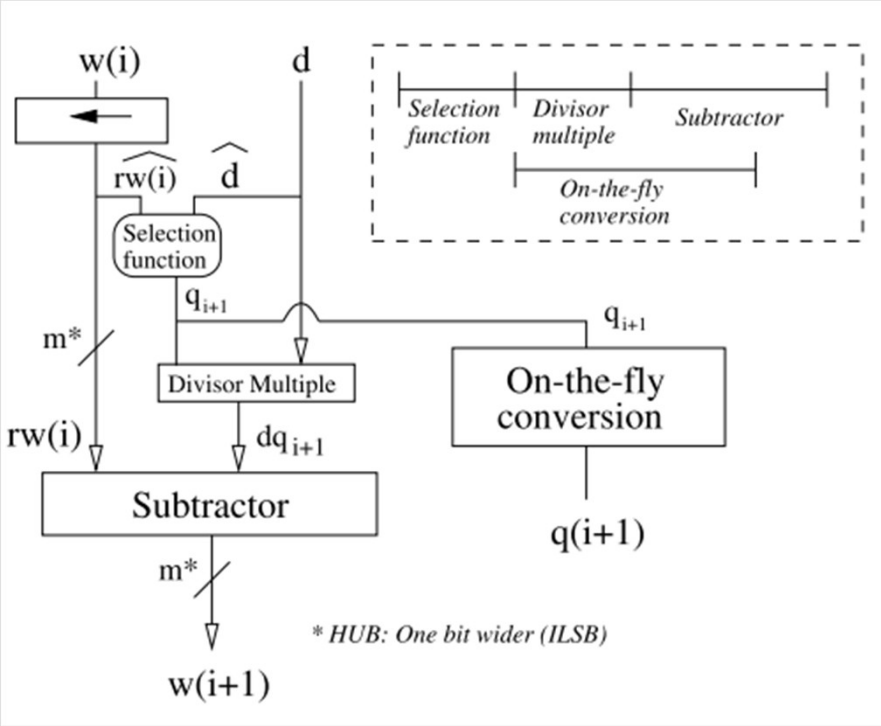
$$(1,2)_{(HUB)} \subset [1,2)_{(conventional)}$$

Conclusion: Operand scaling is applicable to HUB

Improving the HUB division

- Speculation (by replication) of the quotient-digits and residual
  - Quotient-digit selection function operates concurrently with the residual computation [2, 17-20]

- Estimating the residual and digit quotient by selecting the MSBs of the residual
- HUB number behaves like a conventional number during intermediate operations → no influence whether HUB or not



Conclusion: Prediction is applicable to HUB

Improving the HUB division

- High radix
  - Based on radix-2 and/or radix-4
    - Non-overlapped
      - » multiple low-radix stages to form a high-radix stage
    - Overlapped
      - » overlap of the quotient-digit selection of consecutive radix-2 (or radix-4) stages.
    - Prescaling and selection by truncation/rounding
      - » The quotient digit is produced by truncation or rounding of the shifted residual to the left of the radix point
        - » Multiplier required

    → It is independent of whether the initial value comes from a HUB number or not

Conclusion: High radix is applicable to HUB

RADIX-64 HUB DIVISION

- [2] J. D. Bruguera, "Radix-64 floating-point divider," in *Proc. 25th IEEE Symp. Comput. Arithmetic (ARITH)*, Jun. 2018, pp. 84–91 (see also [15,[16],[17])
  - Design complying IEEE
    - Subnormals, several rounding modes
  - HUB desgin
    - No subnormals, only round-to-nearest mode
      - We focus on normal numbers

RADIX-64 FP DIVISION of [2]

- Rounding process is simplified by forzing the result in the Interval [1,2) (no normalization required)
  - x,d ϵ [1,2) → Result q ϵ [0.5,2)
    - If x ≥ d → Result q ϵ [1,2)
      - Early detection of the condition x<d to forze x>d by left-shifting x by 1 bit
        → q=2·x/d → exponent adjustment
        → q ϵ [1,2) (Normalization not required)

      → Applicable to HUB operands
- Radix-64 by overlapping iterations
  - Linking three consecutive radix-4 iterations per cycle (6 quotient bits per cycle)
    - Use prediction and speculation on the residual
      - Digit set q {±2, ±1, 0}, a=2, ρ=2/3
      - Scaling of the operands (selection function does not depend on the divisor)
      - The first quotient digit (q_0=+1 or +2) obtained in parallel with the scaling
        - » One iteration is saved in sigle precisión
      - Classic digit-recurrence división algorithm in radix-4

      → Applicable to HUB (residual datapath is one bit wider, as shown before)

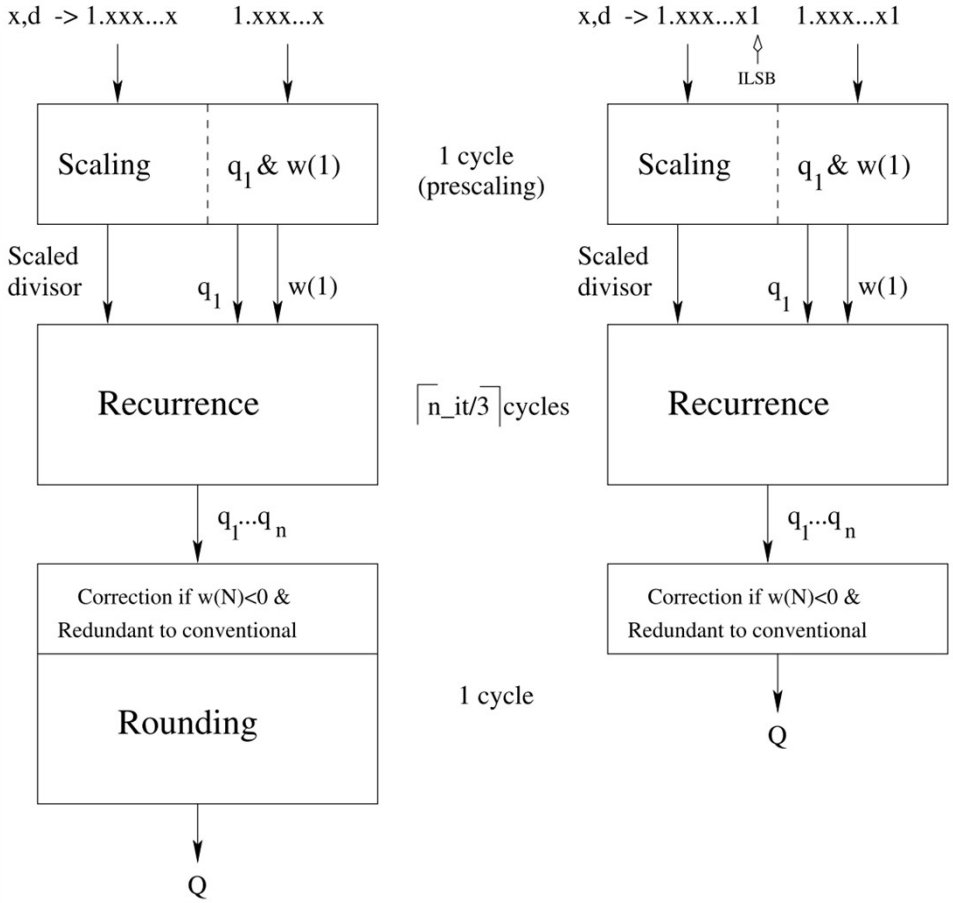
RADIX-64 FP DIVISION

- Number of iterations

$n\_it = \lceil h/2 \rceil$ h → num. bits in the result

Number of cycles (3 iterations per cycle)

$n\_cycles = \lceil n\_it/3 \rceil + 2$

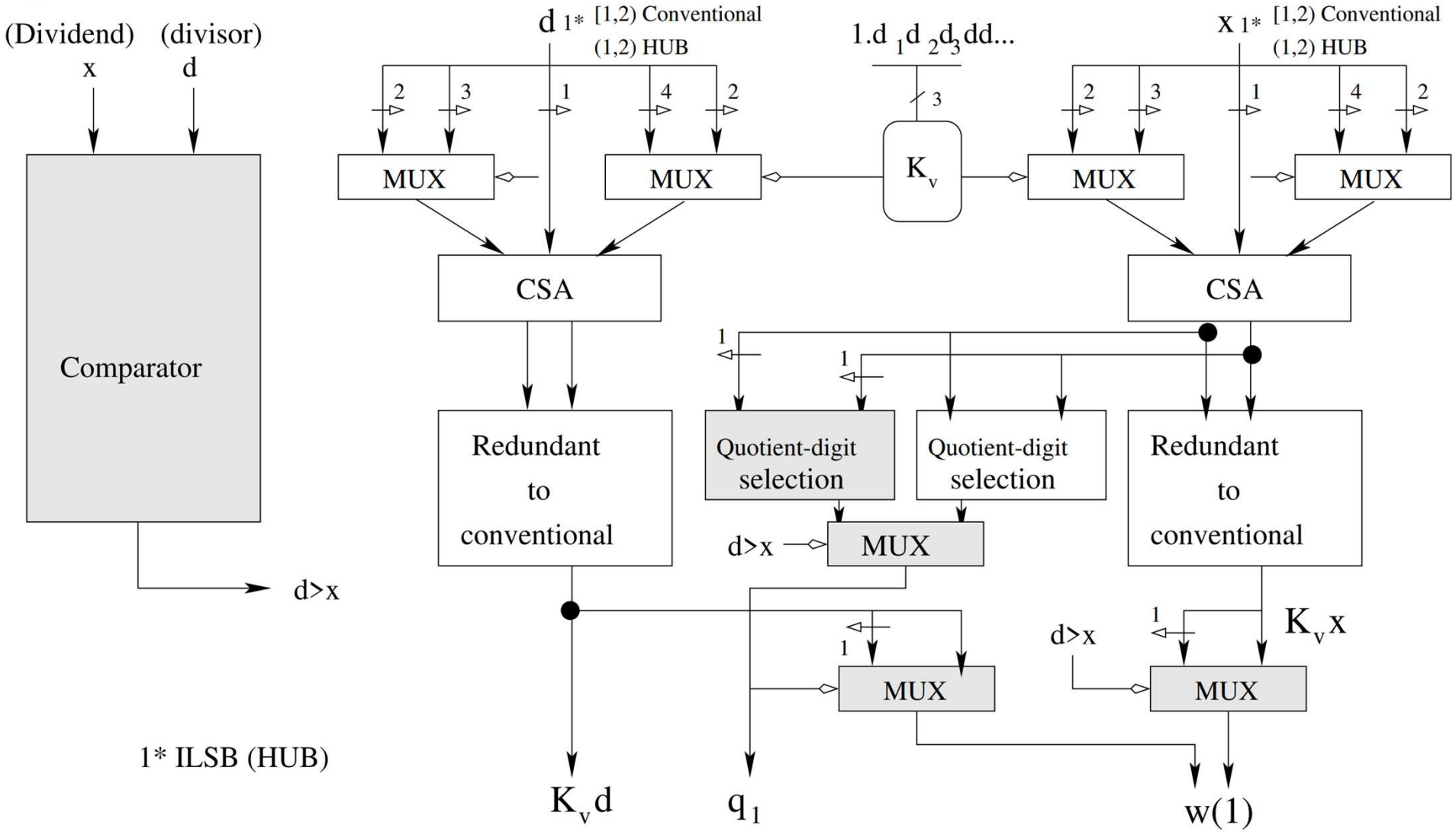


a) Block diagram in [2] *b) Block diagram in HUB*

	h	n_it	n_cycles
binary16	11	6	4
binary32	24	12	6
binary64	53	27	11

** On-the-fly conversión not used since it result in a worse cycle time [2]*

RADIX-64 FP DIVISION of [2]



Prescaling stage for [2] and for HUB.

RADIX-64 FP DIVISION of [2]

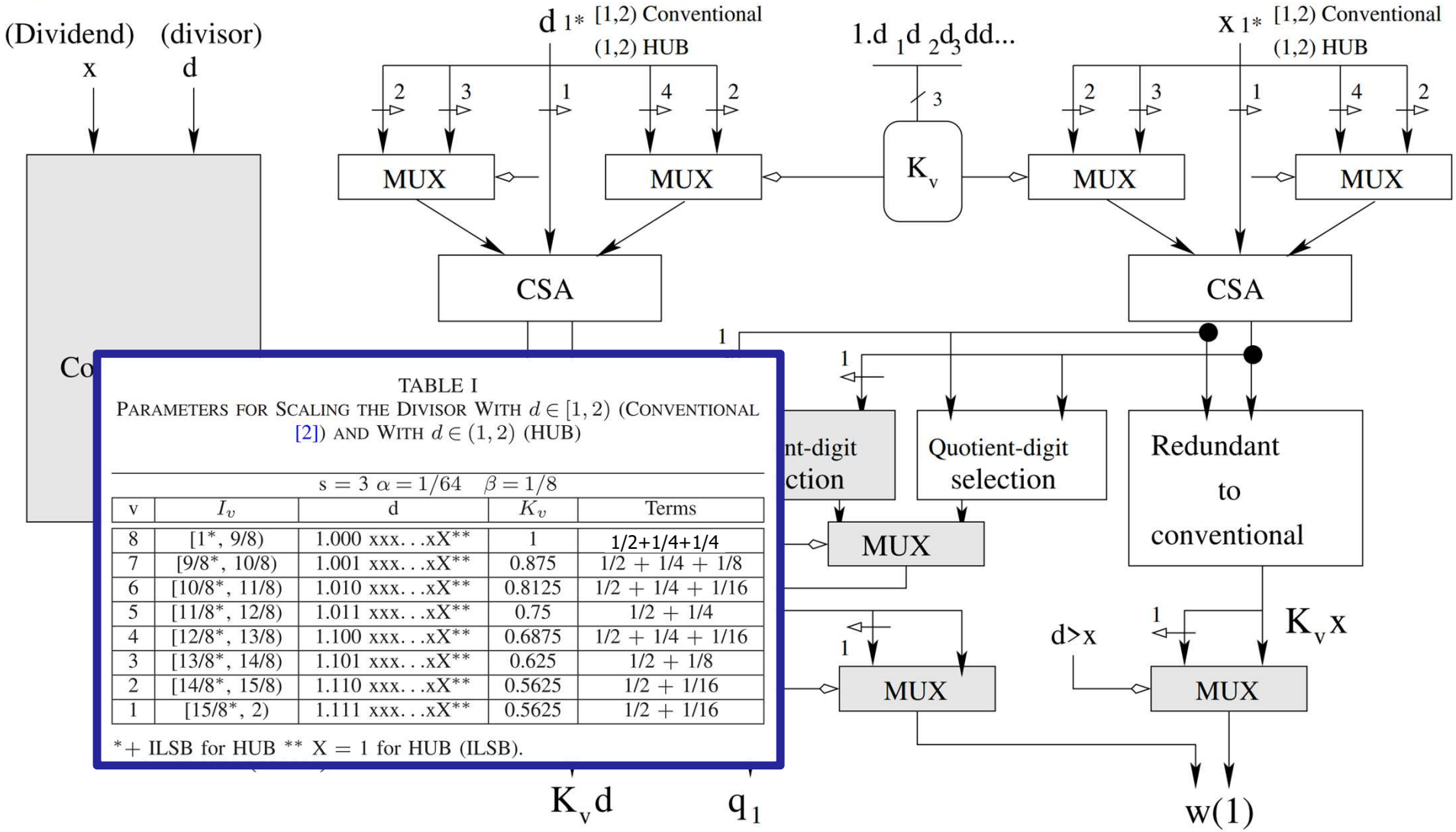


TABLE I
PARAMETERS FOR SCALING THE DIVISOR WITH $d \in [1, 2)$ (CONVENTIONAL [2]) AND WITH $d \in (1, 2)$ (HUB)

s = 3 $\alpha = 1/64$ $\beta = 1/8$

v	I_v	d	K_v	Terms
8	[1*, 9/8)	1.000 xxx...xX**	1	1/2+1/4+1/4
7	[9/8*, 10/8)	1.001 xxx...xX**	0.875	1/2 + 1/4 + 1/8
6	[10/8*, 11/8)	1.010 xxx...xX**	0.8125	1/2 + 1/4 + 1/16
5	[11/8*, 12/8)	1.011 xxx...xX**	0.75	1/2 + 1/4
4	[12/8*, 13/8)	1.100 xxx...xX**	0.6875	1/2 + 1/4 + 1/16
3	[13/8*, 14/8)	1.101 xxx...xX**	0.625	1/2 + 1/8
2	[14/8*, 15/8)	1.110 xxx...xX**	0.5625	1/2 + 1/16
1	[15/8*, 2)	1.111 xxx...xX**	0.5625	1/2 + 1/16

*+ ILSB for HUB ** X = 1 for HUB (ILSB).

Prescaling stage for [2] and for HUB.

Evaluation

- Implementation: Synopsys Design Compiler 28 nm TSMC standard library
- Comparison of architectures based on conventional representation versus their HUB counterparts for the same precision
  - First design: Conventional
    - Radix-2 sequential architecture
    - Residual in carry-save
    - Digit set {±1,0}
    - On-the-fly conversion
    - Round-to-nearest
  - Second design: HUB counterpart
    - Radix-2 sequential architecture
    - Residual in carry-save
    - Digit set {±1,0}
    - On-the-fly conversion
    - Round-to-nearest
  - Third design: Conventional
    - Radix-4 sequential architecture
    - Residual non-redundat
    - Digit set {±2,±1,0}
    - Operands scaling
    - Round-to-nearest
  - Four design: HUB counterpart
    - Radix-4 sequential architecture
    - Residual non-redundat
    - Digit set {±2,±1,0}
    - Operands scaling
    - Round-to-nearest

Evaluation

Implementation: Synopsys Design Compiler 28 nm TSMC standard library

Radix-2 redundant residual, on-the-fly conversion ($\rho = 1$)								Radix-4 non-redundant residual, x & d scaling ($\rho = 2/3$)						
Design	Area	%	Power	%	Delay	%	It*	Area	%	Power	%	Delay	%	It**
16-Conv.	665.02		53.01		3.5		13+1	1923.26		171.4		2.04		2+7+2
16-HUB	631.51	-5.04	49.73	-6.19	3.5	0	13+1	1817.05	-5.52	167.1	-2.51	1.96	-3.92	2+7+1
32-Conv.	1223.46		99.98		6.75		26+1	3348.2		327.4		3.45		2+14+2
32-HUB	1113.33	-9.00	91.61	-8.37	6.75	0	26+1	3354.8	0.19	325.1	-0.7	3.23	-6.38	2+14+1
64-Conv.	2335.28		189.3		14		55+1	6680.52		713.7		6.69		2+28+2
64-HUB	2198,82	-5.84	181.9	-3.91	14	0	55+1	6528.94	-2.27	713.1	-0.08	6.60	-1.35	2+28+1

* Num. of iterations: X+1 $\rightarrow$ X recurrence + 1 on-the-fly end ** 2+X+1+1 $\rightarrow$ 2 scaling + X recurrence + 1 correction + 1 rounding

TABLE II

AREA (μm^2), POWER (μW^2), AND DELAY (ns) FOR THE RADIX-2 AND RADIX- 4 DESIGNS FOR 16, 32, AND 64 BITS

Radix-2

- HUB Data path residual is one bit wider
- Rounding hardware is eliminated or simplifier
- The same number of iterations
- Conclusion:

  the extra bit in the residual data path (due to ILSB) is compensated by the area (power) savings from a simpler on- the-fly conversion and rounding

Radix-4

- HUB Data path residual is one bit wider
- Rounding hardware is eliminated or simplifier
- One less iteration in HUB (rounding)
- Conclusion:

  the extra bit in the residual data path (due to ILSB) is compensated by the area (power) savings from a simpler rounding and one less iteration

Summary and Conclusion

- HUB supports all major digit-recurrence division optimization techniques
  - Operand prescaling, speculation, high-radix division, and on-the-fly conversion remain fully compatible with HUB
- We have shown that HUB can be applied to a state-of-the-art radix-64 design
  - Round-to-nearest is performed by truncation, eliminating the final rounding stage.
- Implementation results confirm the expected benefits
  - Moderate reductions in area and power while maintaining comparable or even improved delay despite the extra ILSB bit in the datapath
- Future work: HUB-based floating-point square root

THANK YOU!

QUESTIONS?