An Optimized Hardware Architecture for the Montgomery Multiplication Algorithm

Miaoqing Huang[1], Kris Gaj[2],
Soonhak Kwon[3], Tarek El-Ghazawi[1]

[1] The George Washington University, Washington, D.C., U.S.A.
[2] George Mason University, Fairfax, VA, U.S.A.
[3] Sungkyunkwan University, Suwon, Korea

Outline

- Motivation
- Classical Hardware Architecture for Montgomery Multiplication by Tenca and Koc from CHES 1999
- Our Optimized Hardware Architecture
- Conceptual Comparison
- Implementation Results
- Possible Extensions
- Conclusions

Motivation

- Fast modular multiplication required in multiple cryptographic transformations
  - RSA, DSA, Diffie-Hellman
  - Elliptic Curve Cryptosystems
  - ECM, p-1, Pollard's rho methods of factoring, etc.

- Montgomery Multiplication invented by Peter L. Montgomery in 1985 is most frequently used to implement repetitive sequence of modular multiplications in both software and hardware

- Montgomery Multiplication in hardware replaces division by a sequence of simple logic operations, conditional additions and right shifts

Montgomery Modular Multiplication (1)

$\mathbf{Z = X \cdot Y \bmod M}$ $\qquad$ X, Y, M – n-bit numbers

Integer domain $\qquad$ **Montgomery domain**

$$X \longrightarrow X' = X \cdot 2^n \bmod M$$

$$Y \longrightarrow Y' = Y \cdot 2^n \bmod M$$

$$\begin{aligned} Z' = \mathrm{MP}(X', Y', M) &= \\ &= X' \cdot Y' \cdot 2^{-n} \bmod M = \\ &= (X \cdot 2^n) \cdot (Y \cdot 2^n) \cdot 2^{-n} \bmod M = \\ &= X \cdot Y \cdot 2^n \bmod M \end{aligned}$$

$$Z = X \cdot Y \bmod M \longleftarrow Z' = Z \cdot 2^n \bmod M$$

Montgomery Modular Multiplication (2)

$$X \longrightarrow X'$$

$$X' = MP(X, 2^{2n} \bmod M, M) =$$
$$= X \cdot 2^{2n} \cdot 2^{-n} \bmod M = X \cdot 2^{n} \bmod M$$

$$Z \longleftarrow Z'$$

$$Z = MP(Z', 1, M) =$$
$$= (Z \cdot 2^{n}) \cdot 1 \cdot 2^{-n} \bmod M = Z \bmod M = Z$$

Montgomery Product

$S[0] = 0$

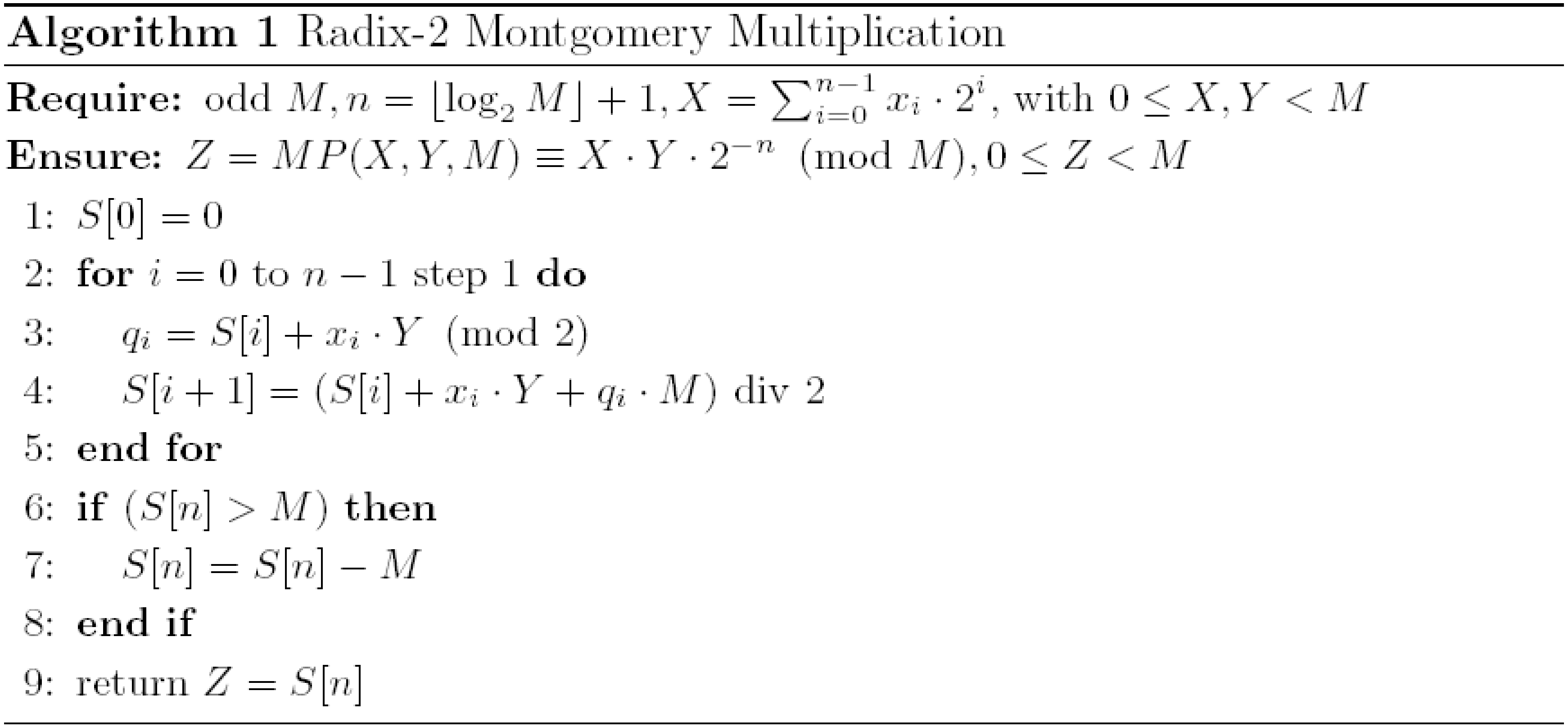
for $i=0$ to $n-1$

$$S[i+1] = \begin{cases} \dfrac{S[i]+x_i \cdot Y}{2} & \text{if} \quad q_i = S[i] + x_i \cdot Y \bmod 2 = 0 \\ \dfrac{S[i]+x_i \cdot Y + M}{2} & \text{if} \quad q_i = S[i] + x_i \cdot Y \bmod 2 = 1 \end{cases}$$

$Z = S[n]$

M assumed to be odd

Basic version of the Radix-2 Montgomery Multiplication Algorithm



Algorithm 1 Radix-2 Montgomery Multiplication

```
Require: odd $M, n = \lfloor \log_2 M \rfloor + 1, X = \sum_{i=0}^{n-1} x_i \cdot 2^i$, with $0 \leq X, Y < M$
Ensure: $Z = MP(X, Y, M) \equiv X \cdot Y \cdot 2^{-n} \pmod{M}, 0 \leq Z < M$
$S[0] = 0$
for $i = 0$ to $n - 1$ step 1 do
    $q_i = S[i] + x_i \cdot Y \pmod{2}$
    $S[i+1] = (S[i] + x_i \cdot Y + q_i \cdot M)$ div 2
end for
if $(S[n] > M)$ then
    $S[n] = S[n] - M$
end if
return $Z = S[n]$
```

Classical Design by Tenca & Koc
CHES 1999

Multiple Word Radix-2 Montgomery Multiplication algorithm (MWR2MM)

Main ideas:

Use of short precision words (w-bit each):

- Reduces broadcast problem in circuit implementation
- Word-oriented algorithm provides the support needed to develop scalable hardware units.

Operand Y(multiplicand) is scanned word-by-word,
operand X(multiplier) is scanned bit-by-bit.

Classical Design by Tenca & Koc
CHES 1999

Each word has w bits

Each operand has

n bits

e words

$$e = \left\lceil \frac{n+1}{w} \right\rceil$$

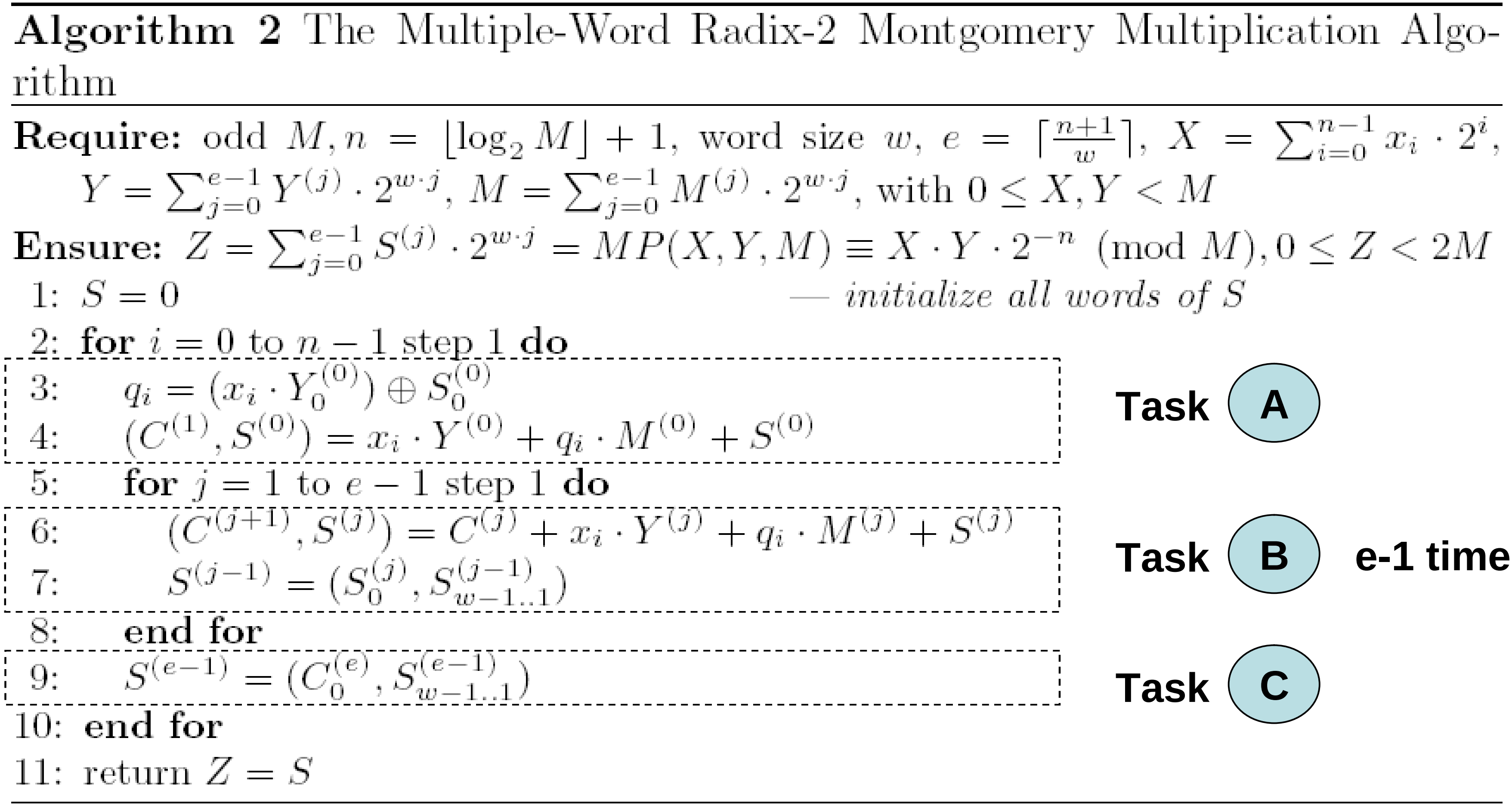
$$X = (x_{n-1}, \dots, x_1, x_0)$$

$$Y = (Y^{(e-1)}, \dots, Y^{(1)}, Y^{(0)})$$

$$M = (M^{(e-1)}, \dots, M^{(1)}, M^{(0)})$$

The bits are marked with subscripts, and the words are marked with superscripts.

MWR2MM
Multiple Word Radix-2 Montgomery Multiplication algorithm by Tenca and Koc



Algorithm 2 The Multiple-Word Radix-2 Montgomery Multiplication Algorithm

Require: odd $M, n = \lfloor \log_2 M \rfloor + 1$, word size w, $e = \lceil \frac{n+1}{w} \rceil$, $X = \sum_{i=0}^{n-1} x_i \cdot 2^i$, $Y = \sum_{j=0}^{e-1} Y^{(j)} \cdot 2^{w \cdot j}$, $M = \sum_{j=0}^{e-1} M^{(j)} \cdot 2^{w \cdot j}$, with $0 \leq X, Y < M$

Ensure: $Z = \sum_{j=0}^{e-1} S^{(j)} \cdot 2^{w \cdot j} = MP(X, Y, M) \equiv X \cdot Y \cdot 2^{-n} \pmod{M}, 0 \leq Z < 2M$

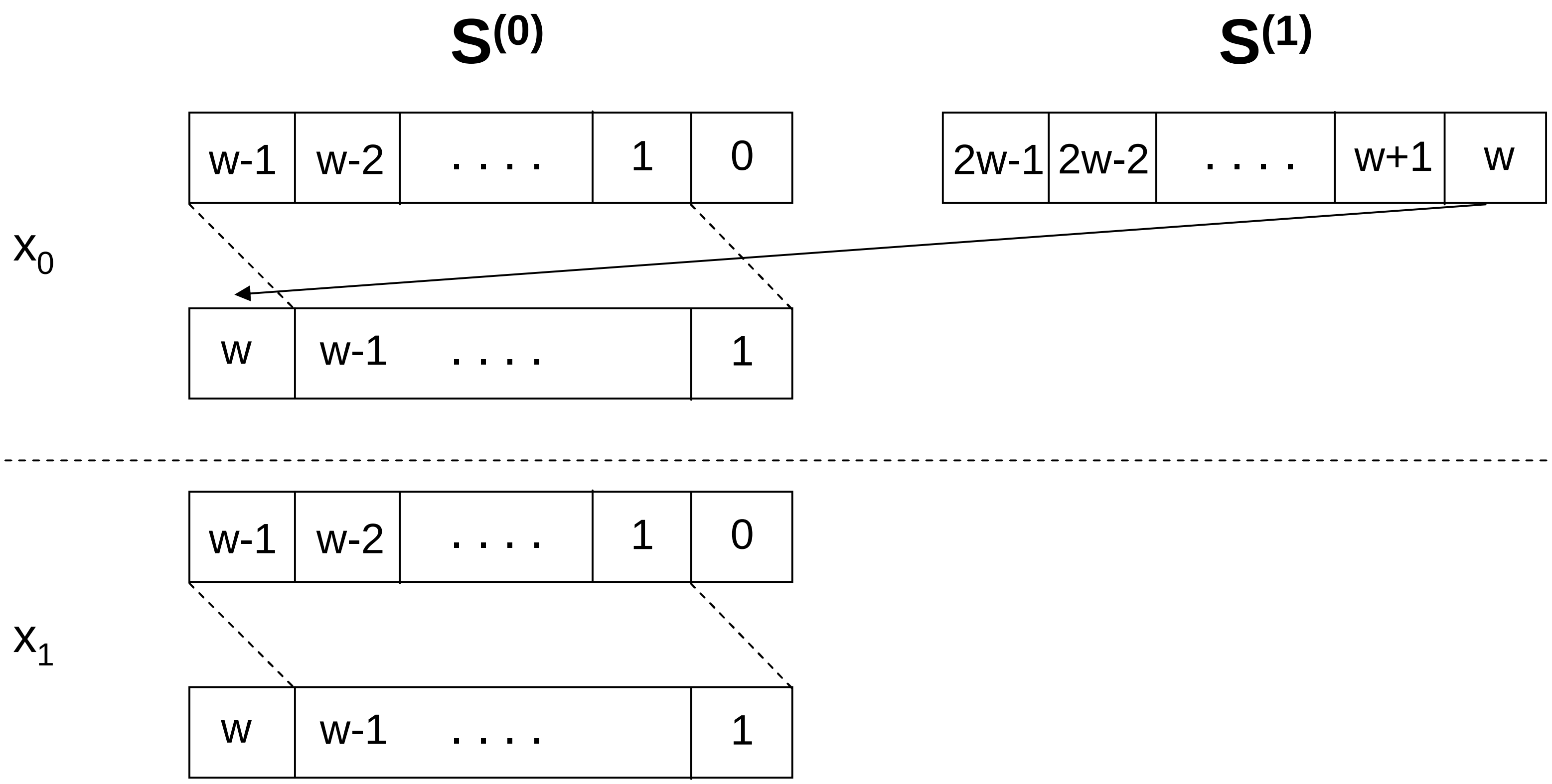
```
$S = 0$                                            — initialize all words of S
for $i = 0$ to $n - 1$ step 1 do
    $q_i = (x_i \cdot Y_0^{(0)}) \oplus S_0^{(0)}$
    $(C^{(1)}, S^{(0)}) = x_i \cdot Y^{(0)} + q_i \cdot M^{(0)} + S^{(0)}$
    for $j = 1$ to $e - 1$ step 1 do
        $(C^{(j+1)}, S^{(j)}) = C^{(j)} + x_i \cdot Y^{(j)} + q_i \cdot M^{(j)} + S^{(j)}$
        $S^{(j-1)} = (S_0^{(j)}, S_{w-1..1}^{(j-1)})$
    end for
    $S^{(e-1)} = (C_0^{(e)}, S_{w-1..1}^{(e-1)})$
end for
return $Z = S$
```

Task A

Task B e-1 times

Task C

Problem in Parallelizing Computations



$S^{(0)}$

$S^{(1)}$

w-1	w-2	. . . .	1	0

2w-1	2w-2	. . . .	w+1	w

x_0

w	w-1	1

w-1	w-2	. . . .	1	0

x_1

w	w-1	1

Data Dependency Graph by Tenca & Koc

- One PE is in charge of the computation of one column that corresponds to the updating of *S* with respect to one single bit x_i.

- The delay between two adjacent PEs is 2 clock cycles.

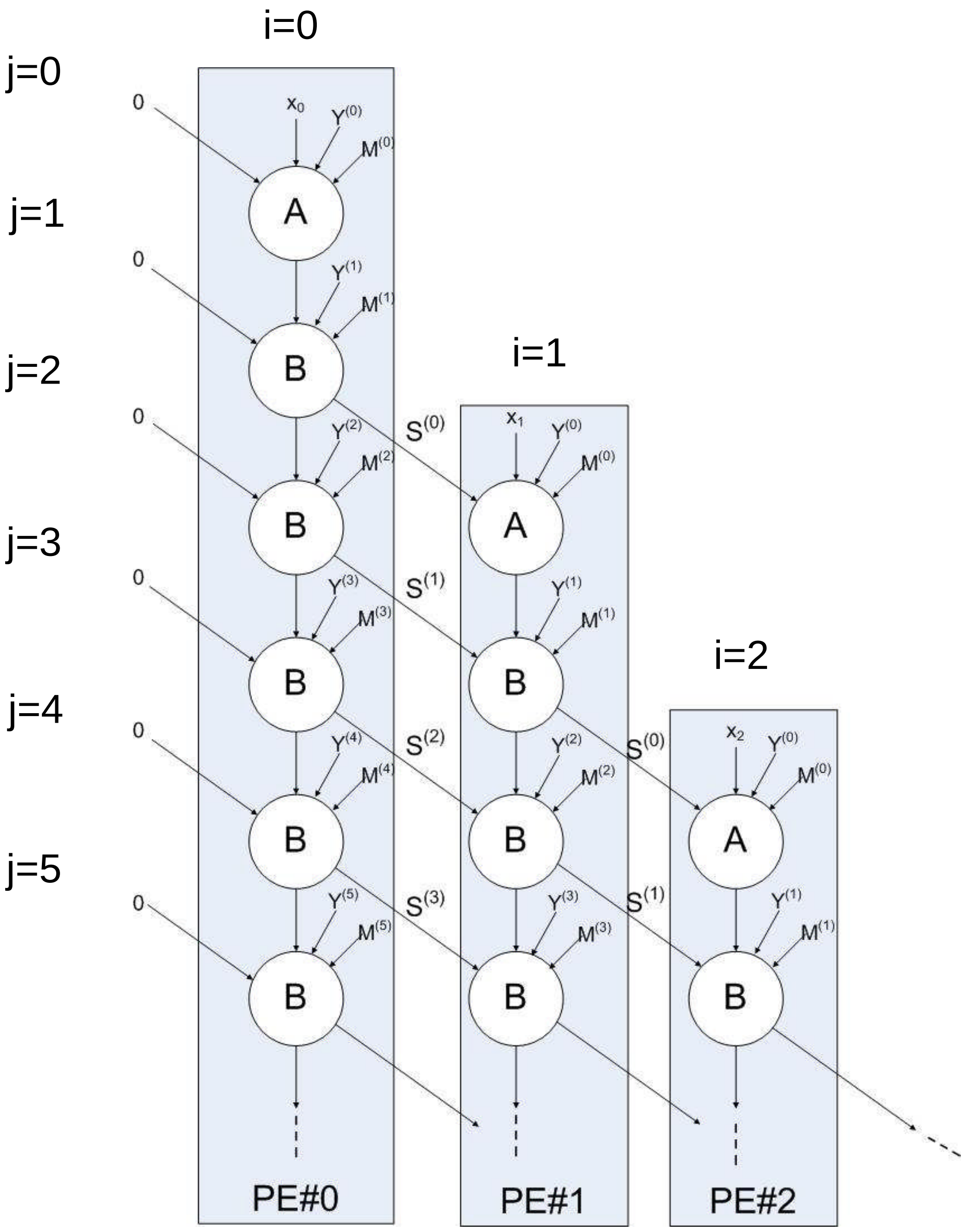
- The minimum computation time is

  $2 \bullet n+e-1$ clock cycles

- given

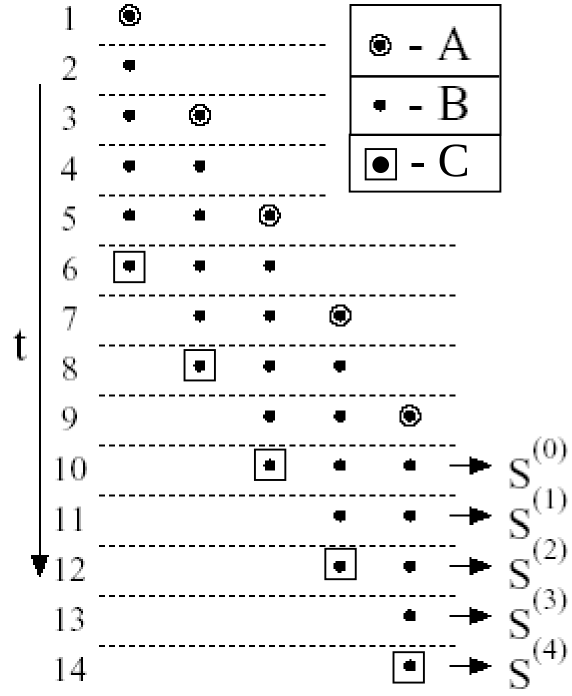
  $(e+1)/2$ PEs

  working in parallel.



Example of Operation of the Design by Tenca & Koc

Example of the computation executed for 5-bit operands with word-size
w = 1 bit



n = 5

w = 1

e = 5

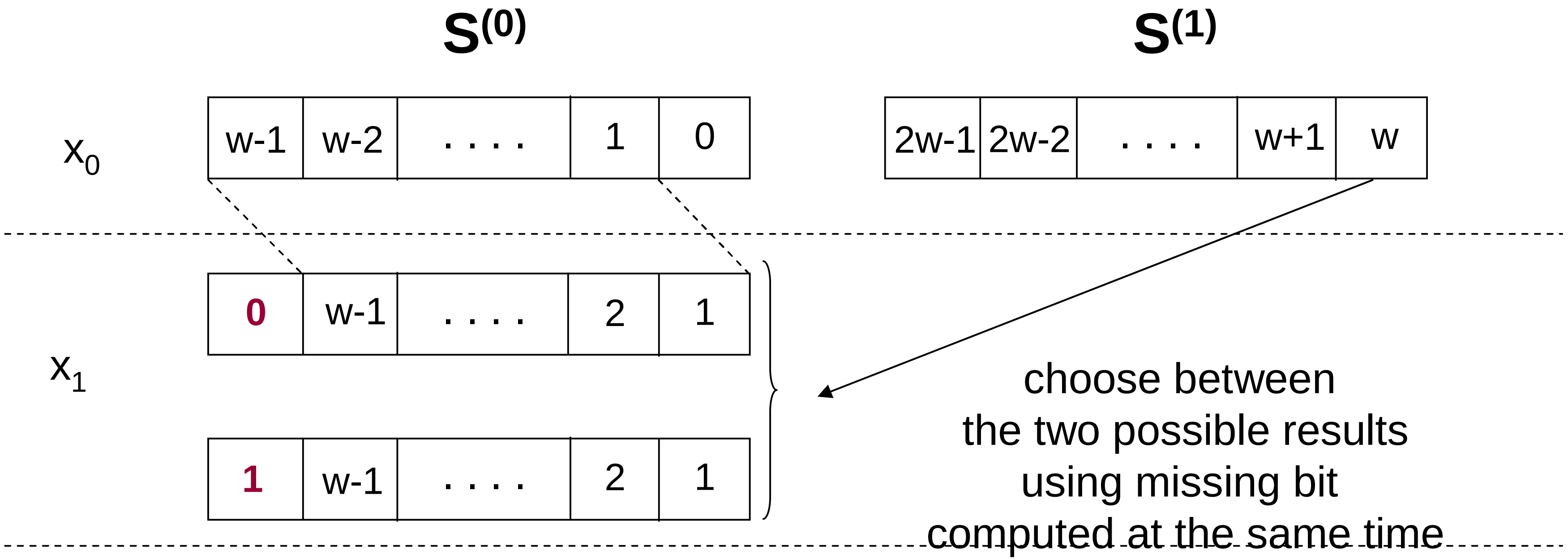
2n + e – 1 =
2·5 + 5 – 1 = 14 clock cycles

(e+1)/2 =
(5+1)/2 = 3 PEs
sufficient to perform all computations

Main Idea of the New Architecture

- In the architecture of Tenca & Koc
  - w-1 least significant bits of partial results $S^{(j)}$ are available one clock cycle before they are used
  - only one (most significant) bit is missing
- Let us compute a new partial result under two assumptions regarding the value of the most significant bit of $S^{(j)}$ and choose the correct value one clock cycle later

Idea for a Speed-up



choose between
the two possible results
using missing bit
computed at the same time

perform two computations
in parallel using two possible
values of the most-significant-bit

Primary Advantage of the New Approach

- Reduction in the number of clock cycles

  from

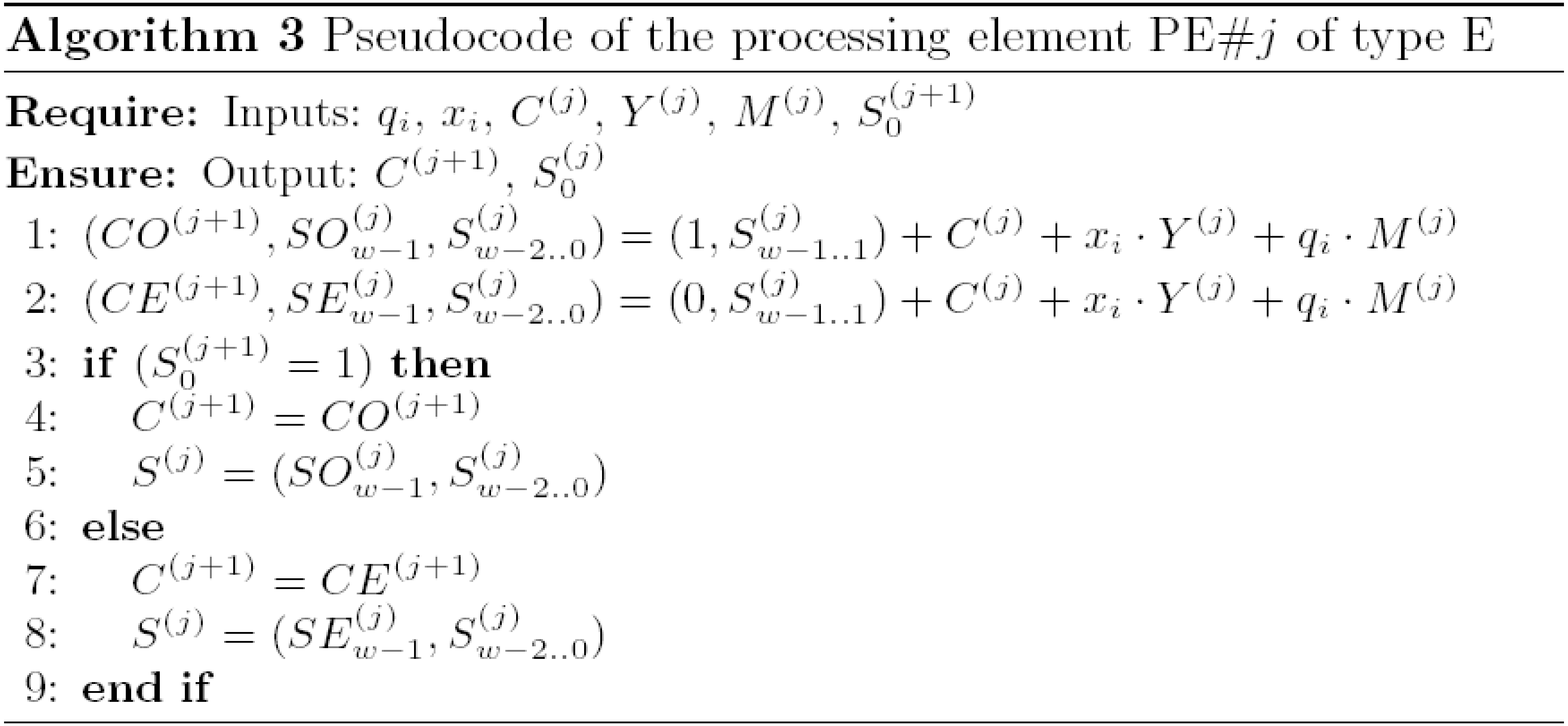
  $$2\,n + e - 1$$

  to

  $$n + e - 1$$

- Minimum penalty in terms of the area and clock period

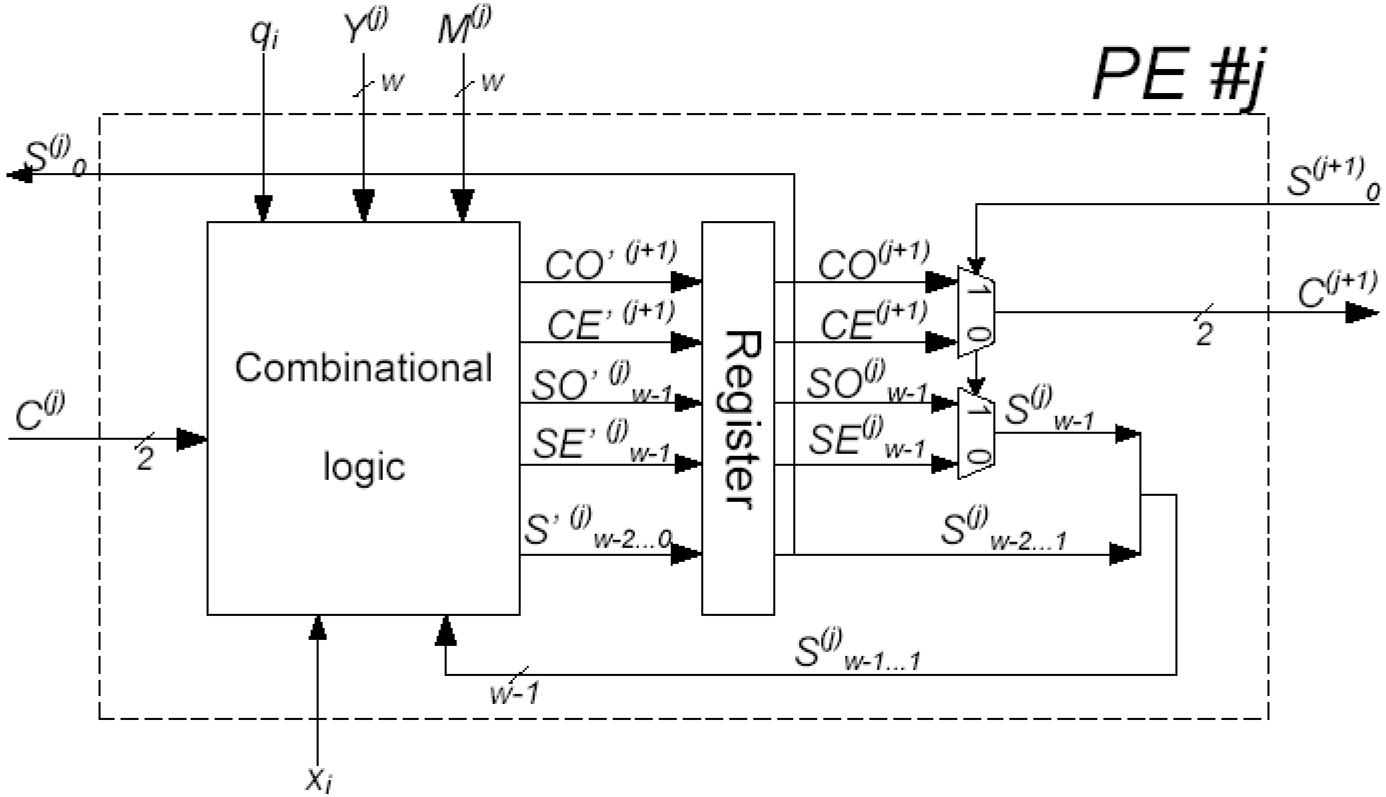
Pseudocode of the Main Processing Element



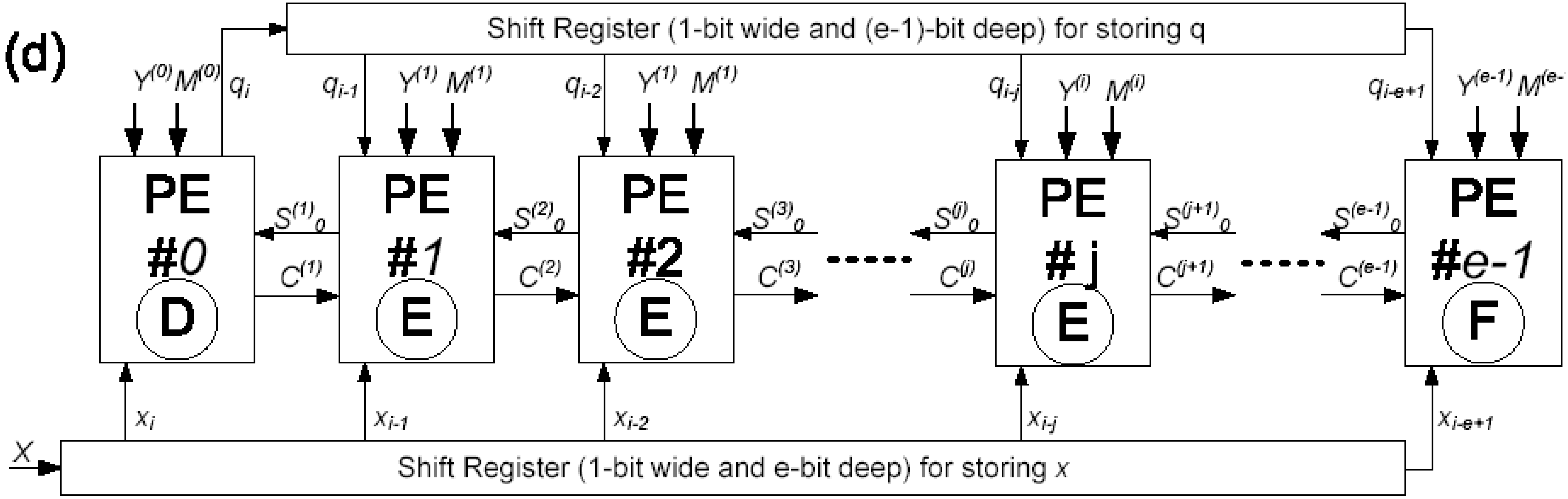
Algorithm 3 Pseudocode of the processing element PE#j of type E

```
Require: Inputs: $q_i$, $x_i$, $C^{(j)}$, $Y^{(j)}$, $M^{(j)}$, $S_0^{(j+1)}$
Ensure: Output: $C^{(j+1)}$, $S_0^{(j)}$
$(CO^{(j+1)}, SO_{w-1}^{(j)}, S_{w-2..0}^{(j)}) = (1, S_{w-1..1}^{(j)}) + C^{(j)} + x_i \cdot Y^{(j)} + q_i \cdot M^{(j)}$
$(CE^{(j+1)}, SE_{w-1}^{(j)}, S_{w-2..0}^{(j)}) = (0, S_{w-1..1}^{(j)}) + C^{(j)} + x_i \cdot Y^{(j)} + q_i \cdot M^{(j)}$
if $(S_0^{(j+1)} = 1)$ then
    $C^{(j+1)} = CO^{(j+1)}$
    $S^{(j)} = (SO_{w-1}^{(j)}, S_{w-2..0}^{(j)})$
else
    $C^{(j+1)} = CE^{(j+1)}$
    $S^{(j)} = (SE_{w-1}^{(j)}, S_{w-2..0}^{(j)})$
end if
```

Main Processing Element Type E

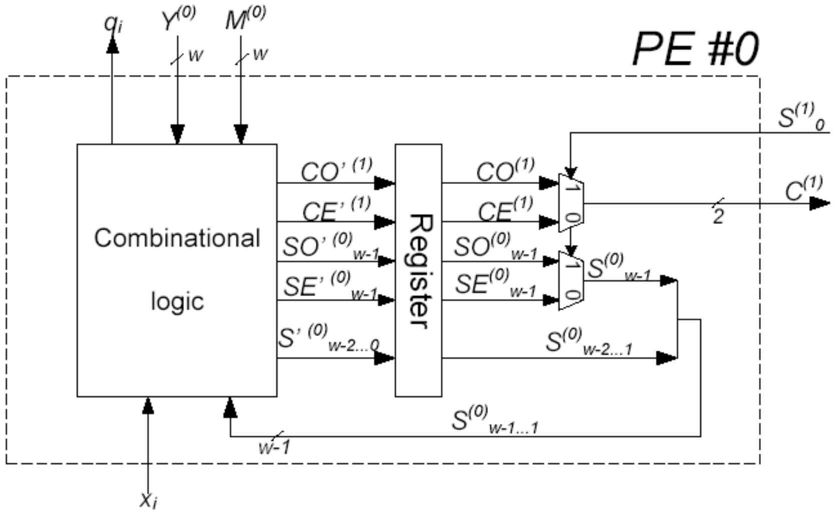


The Proposed Optimized Hardware Architecture

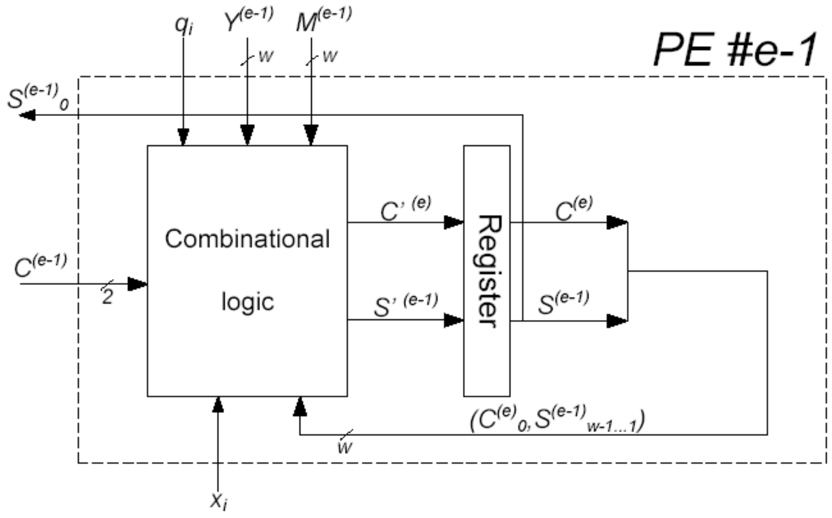

(d)
Shift Register (1-bit wide and (e-1)-bit deep) for storing q
$Y^{(0)}$ $M^{(0)}$ q_i
q_{i-1} $Y^{(1)}$ $M^{(1)}$
q_{i-2} $Y^{(1)}$ $M^{(1)}$
q_{i-j} $Y^{(j)}$ $M^{(j)}$
q_{i-e+1} $Y^{(e-1)}$ $M^{(e-}$
PE #0 D
PE #1 E
PE #2 E
PE # j E
PE #e-1 F
$S^{(1)}{}_0$ $C^{(1)}$
$S^{(2)}{}_0$ $C^{(2)}$
$S^{(3)}{}_0$ $C^{(3)}$
$S^{(j)}{}_0$ $C^{(j)}$
$S^{(j+1)}{}_0$ $C^{(j+1)}$
$S^{(e-1)}{}_0$ $C^{(e-1)}$
x_i
x_{i-1}
x_{i-2}
x_{i-j}
x_{i-e+1}
X
Shift Register (1-bit wide and e-bit deep) for storing x

The First and the Last Processing Elements

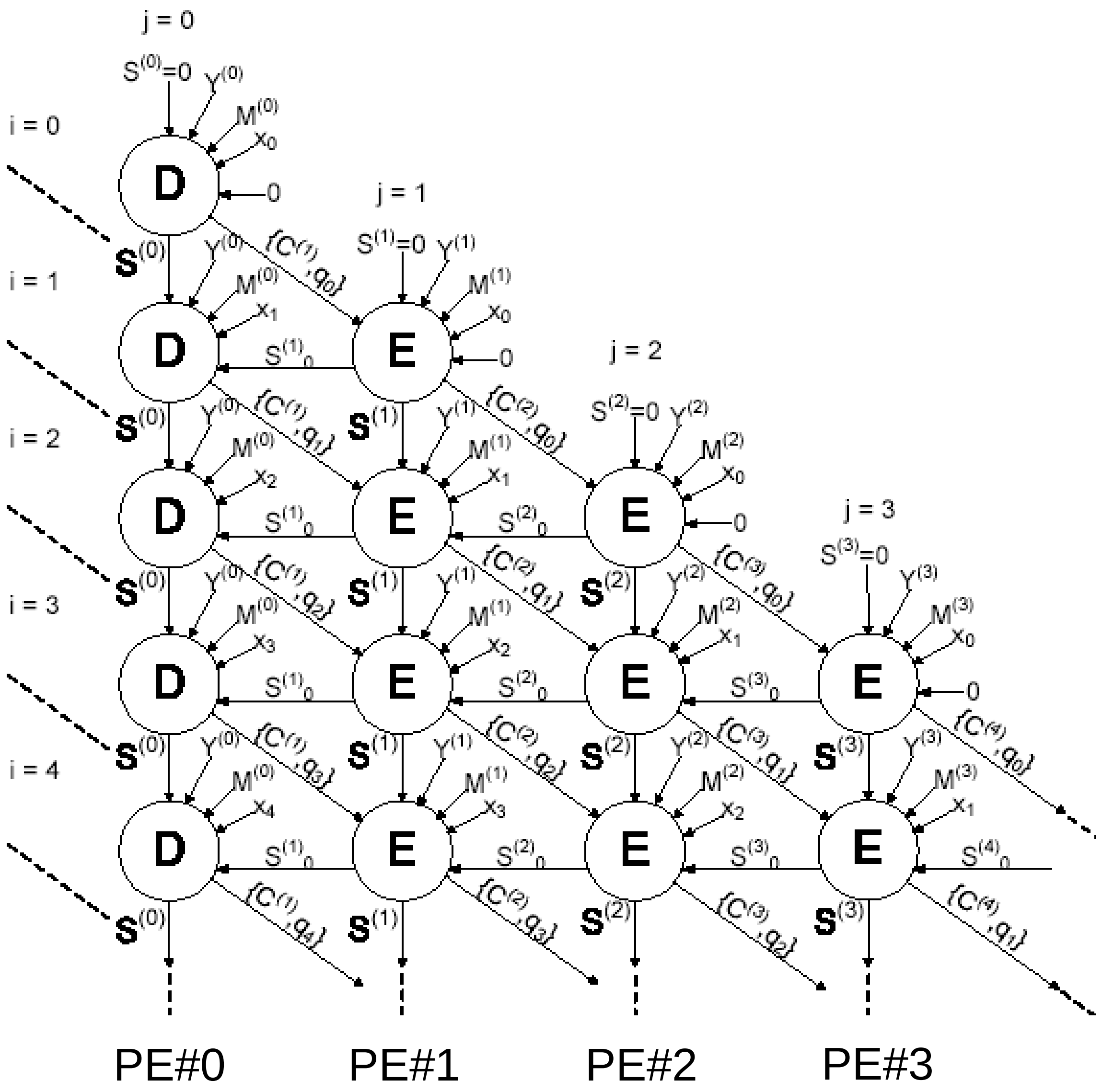
Type D



Type F



Data Dependency Graph of the Proposed New Architecture



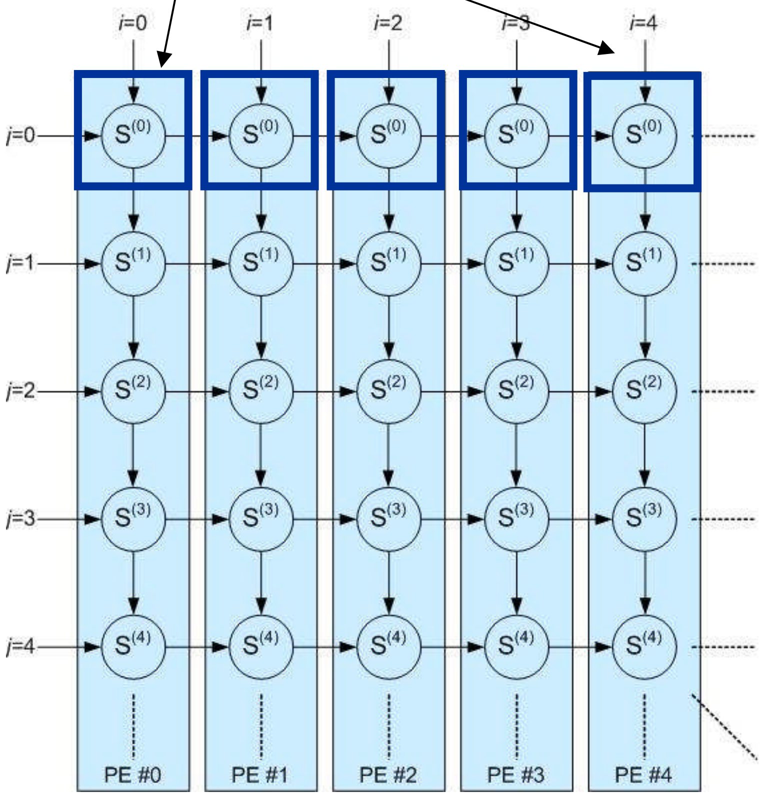
The Overall Computation Pattern

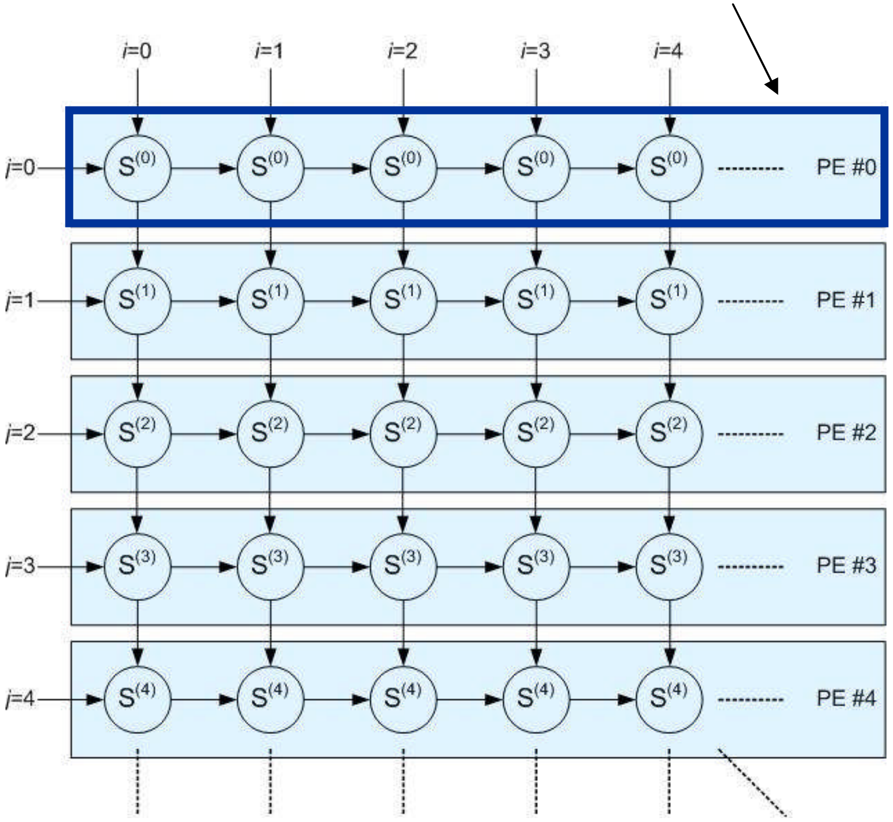
Special state of each PE

vs.

One special PE type

simpler structure of each PE



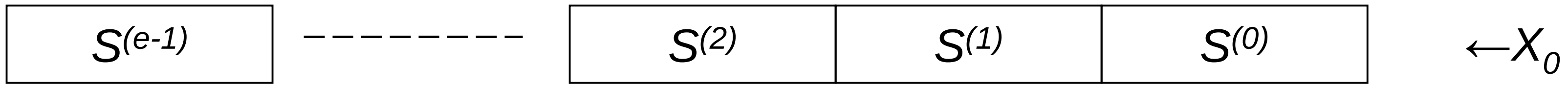


Tenca & Koc, CHES 1999

Our new proposed architecture

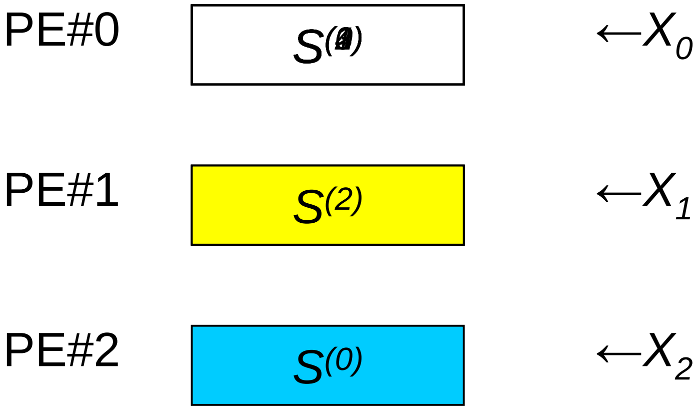
Demonstration of Computations

- Sequential



$S^{(e-1)}$ - - - - - - - - - $S^{(2)}$ $S^{(1)}$ $S^{(0)}$ ← X_0

- Tenca & Koç’s proposal



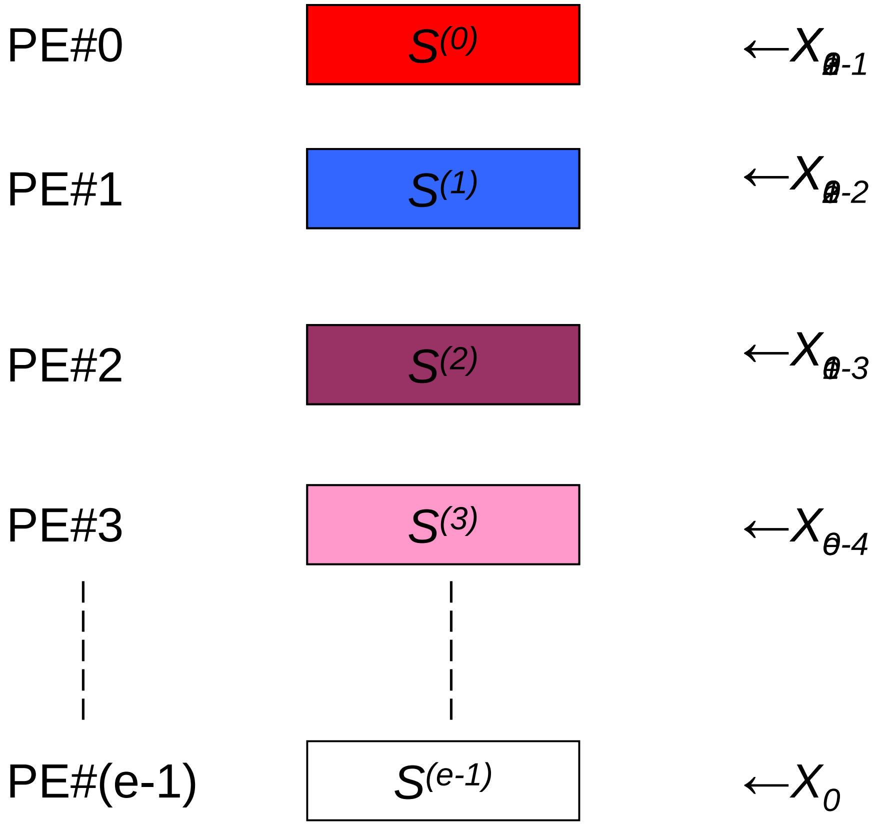
PE#0 S([illegible]) ← X_0

PE#1 $S^{(2)}$ ← X_1

PE#2 $S^{(0)}$ ← X_2

Demonstration of Computations *(cont.)*

- **The proposed optimized architecture**



PE#0 $S^{(0)}$ $\leftarrow X_{e-1}$

PE#1 $S^{(1)}$ $\leftarrow X_{e-2}$

PE#2 $S^{(2)}$ $\leftarrow X_{e-3}$

PE#3 $S^{(3)}$ $\leftarrow X_{e-4}$

PE#(e-1) $S^{(e-1)}$ $\leftarrow X_0$

Related Work

- Tenca, A. and Koç, Ç.K.: A scalable architecture for Montgomery multiplication, CHES 1999, LNCS, vol.1717, pp.94--108, Springer, Heidelberg, 1999
  - The original paper proposing the MWR2MM algorithm
- Tenca, A., Todorov, G., and Koç. Ç.K.: High-radix design of a scalable modular multiplier, CHES 2001, LNCS, vol.2162, pp.185--201, Springer, Heidelberg, 2001
  - Scan several bits of *X* one time instead of one bit
- Harris, D., Krishnamurthy, R., Anders, M., Mathew, S. and Hsu, S.: An Improved Unified Scalable Radix-2 Montgomery Multiplier, ARITH 17, pp.172-178, 2005
  - Left shift *Y* and *M* instead of right shift $S^{(i)}$
- Michalski, E. A. and Buell, D. A.: A scalable architecture for RSA cryptography on large FPGAs, FPL 2006, pp.145--152, 2006
  - FPGA-specific, assumes the use of of built-in multipliers
- McIvor, C., McLoone, M. and McCanny, J.V.: Modified Montgomery Modular Multiplication and RSA Exponentiation Techniques, IEE Proceedings – Computers & Digital Techniques, vol.151, no.6, pp.402-408, 2004
  - Use carry-save addition on full-size n-bit operands

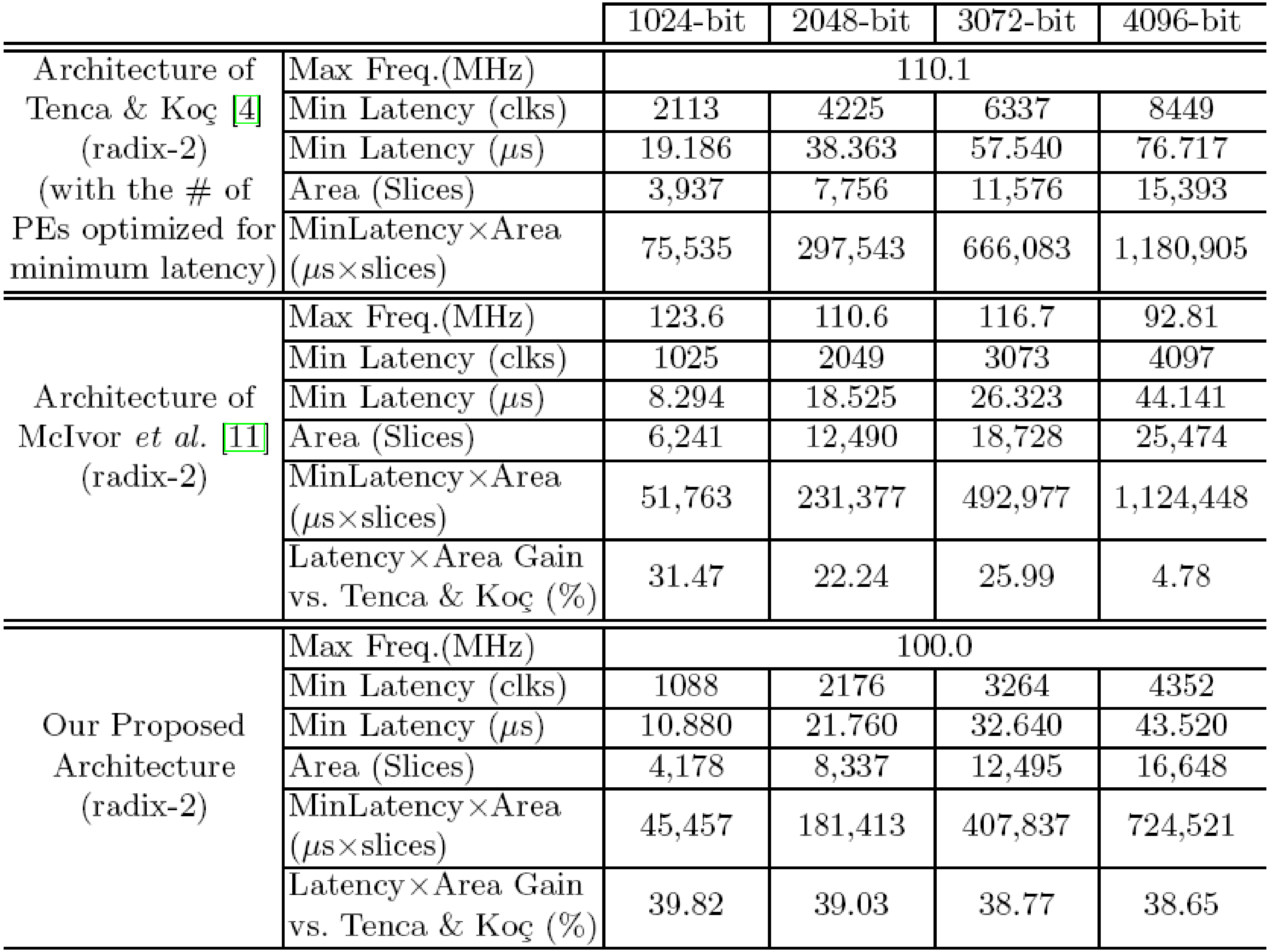
Implementation, Verification, and Experimental Testing

New architecture and two previous architectures:

- Modeled in Verilog HDL and/or VHDL
- Functionally verified by comparison with a reference software implementation of the Montgomery Multiplication
- Implemented using Xilinx Virtex-II 6000-4 FPGA
- Experimentally tested using SRC 6 reconfigurable computer based on microprocessors and FPGAs with the 100 MHz maximum clock frequency for the FPGA part

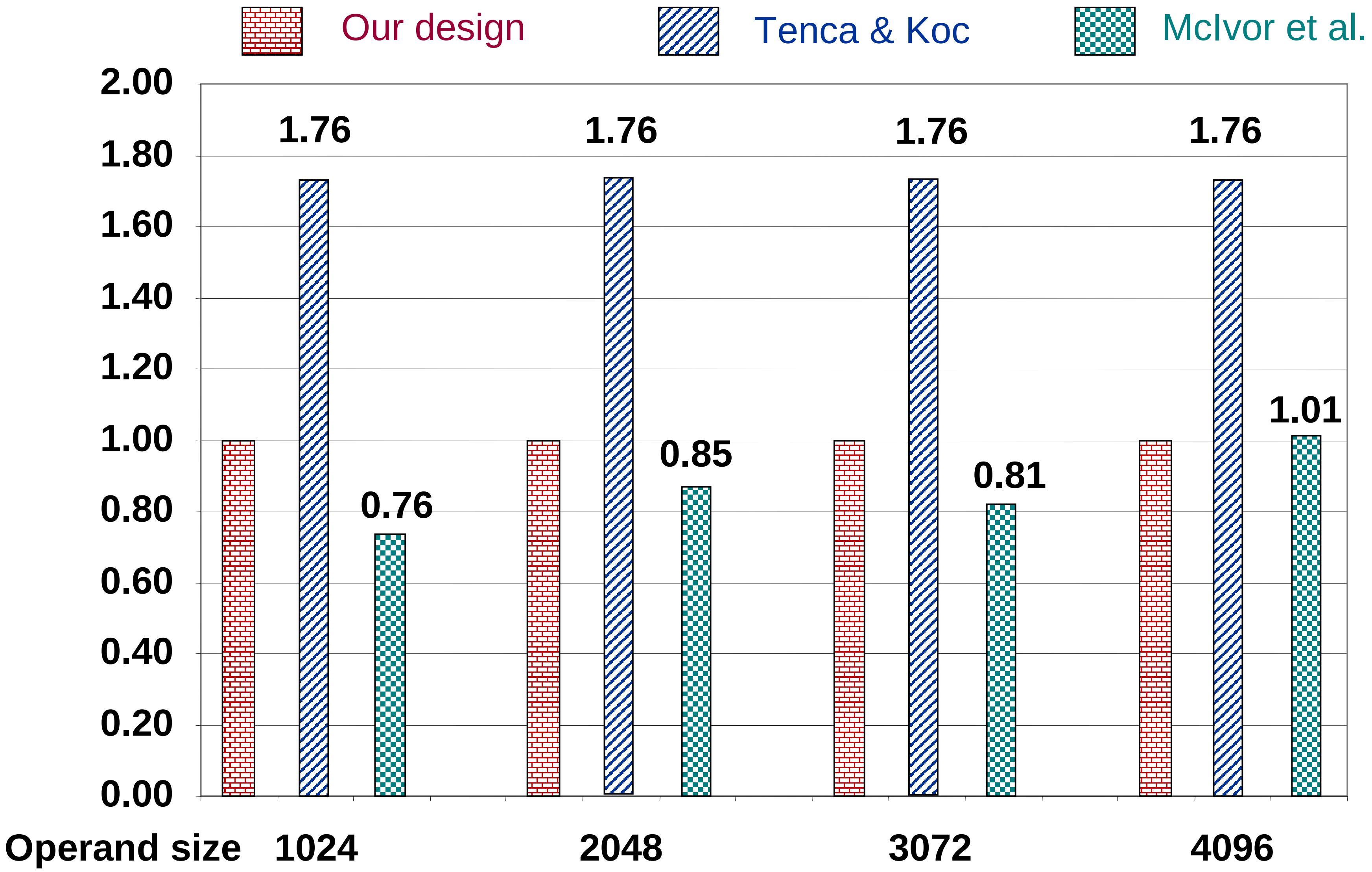
Implementation Results

Test platform: Xilinx Virtex-II 6000 FF1517-4

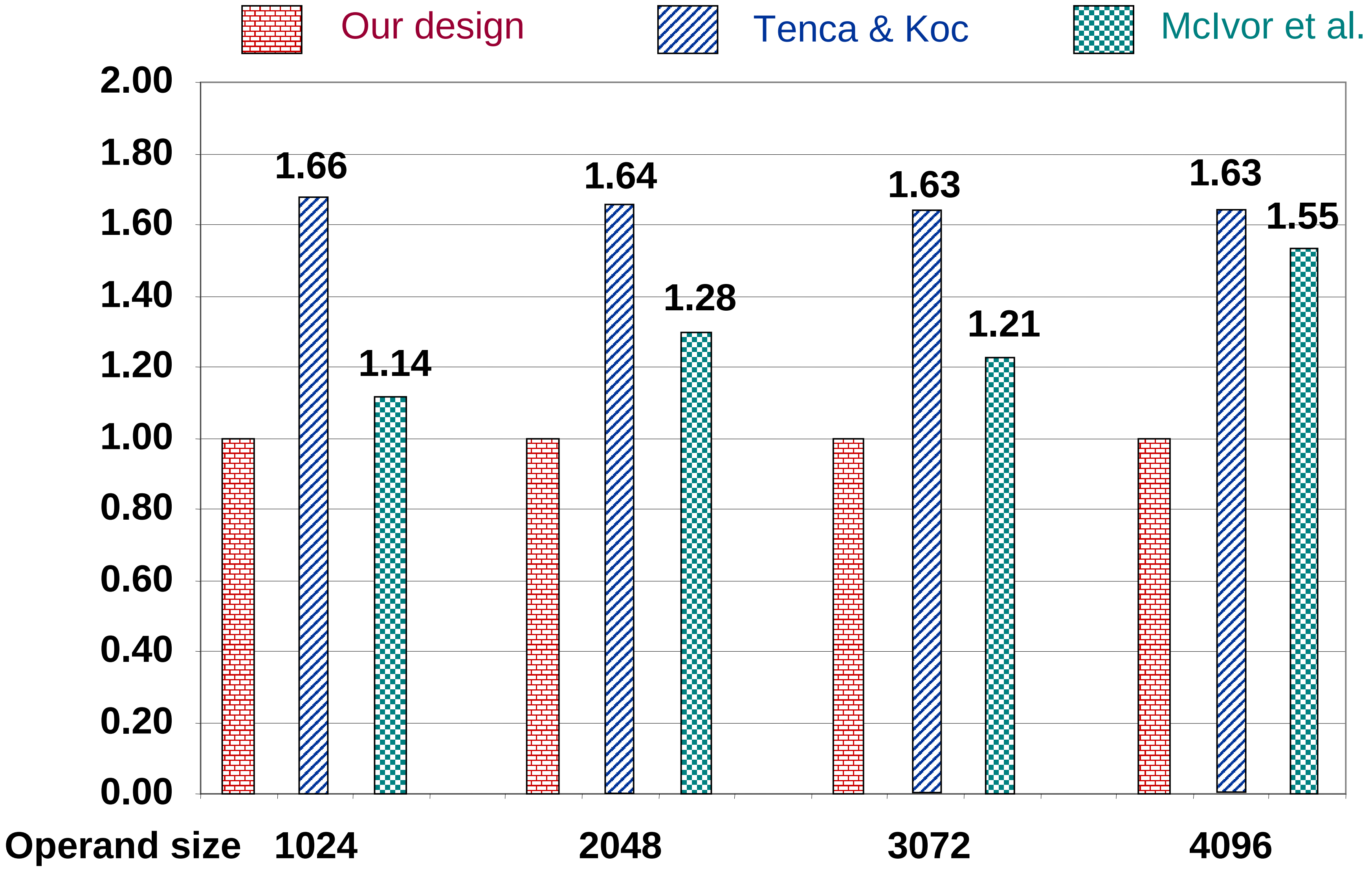


		1024-bit	2048-bit	3072-bit	4096-bit
Architecture of Tenca & Koç [4] (radix-2) (with the # of PEs optimized for minimum latency)	Max Freq.(MHz)	110.1			
	Min Latency (clks)	2113	4225	6337	8449
	Min Latency (μs)	19.186	38.363	57.540	76.717
	Area (Slices)	3,937	7,756	11,576	15,393
	MinLatency×Area (μs×slices)	75,535	297,543	666,083	1,180,905
Architecture of McIvor *et al.* [11] (radix-2)	Max Freq.(MHz)	123.6	110.6	116.7	92.81
	Min Latency (clks)	1025	2049	3073	4097
	Min Latency (μs)	8.294	18.525	26.323	44.141
	Area (Slices)	6,241	12,490	18,728	25,474
	MinLatency×Area (μs×slices)	51,763	231,377	492,977	1,124,448
	Latency×Area Gain vs. Tenca & Koç (%)	31.47	22.24	25.99	4.78
Our Proposed Architecture (radix-2)	Max Freq.(MHz)	100.0			
	Min Latency (clks)	1088	2176	3264	4352
	Min Latency (μs)	10.880	21.760	32.640	43.520
	Area (Slices)	4,178	8,337	12,495	16,648
	MinLatency×Area (μs×slices)	45,457	181,413	407,837	724,521
	Latency×Area Gain vs. Tenca & Koç (%)	39.82	39.03	38.77	38.65

Normalized Latency
New Architecture vs. Previous Architectures



Normalized Product Latency Times Area
New Architecture vs. Previous Architectures



Radix-4 Architecture

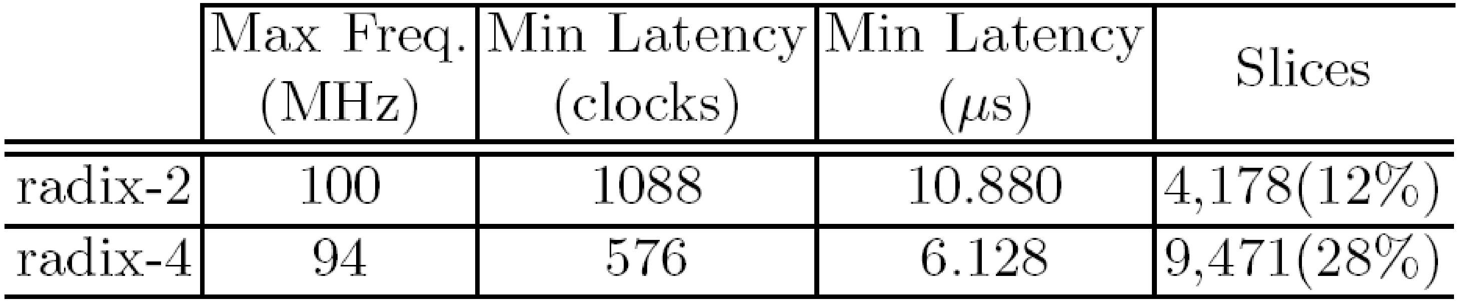
- The same optimization concept can be applied to a radix-4 implementation
- Two bits of X processed in each iteration
- Latency in clock cycles reduced by a factor of almost 2
- However, the clock period and the area increase
- Radix higher than 4 not viable because of

  the large area and the requirement of multiplication

  of words by digits in the range from 0 to radix-1

Comparison between radix-2 and radix-4 versions of the proposed architecture (n=1024, w=16)

Xilinx Virtex-II 6000 FF1517-4 FPGA



	Max Freq. (MHz)	Min Latency (clocks)	Min Latency (μs)	Slices
radix-2	100	1088	10.880	4,178(12%)
radix-4	94	576	6.128	9,471(28%)

$$\frac{\text{Latency * area [for radix 4]}}{\text{Latency * area [for radix 2]}} = \mathbf{1.28}$$

Conclusions

- New optimized architecture for the word-based Montgomery Multiplier

Compared to the classical design by Tenca & Koc:

- Minimum latency smaller by a factor of about **1.8**, in terms of both clock cycles and absolute time units
- Comparable circuit area for minimum latency
- Improvement in terms of the product of latency times area by a factor of about **1.6**
- Reduced scalability (fixed vs. variable number of processing elements required for the given operand size)

  [to be fixed in the new architecture under development]

Compared to the newer design by McIvor et al.:

- Area smaller by about 50%
- Improvement in terms of the product of latency times area by a factor between **1.14** and **1.55**
- Similar scalability

Thank you!

Questions???