Scalable and Unified Hardware to Compute Montgomery Inverse in GF(p) and GF(2^n)

A. Gutub, A. Tenca, E. Savas, and C. Koc

Information Security Laboratory
Electrical and Computer Engineering Department
Oregon State University, Corvallis, Oregon 97331

Workshop on Cryptographic Hardware and Embedded Systems

August 13 - 15, 2002

OREGON STATE UNIVERSITY

Presentation Outline

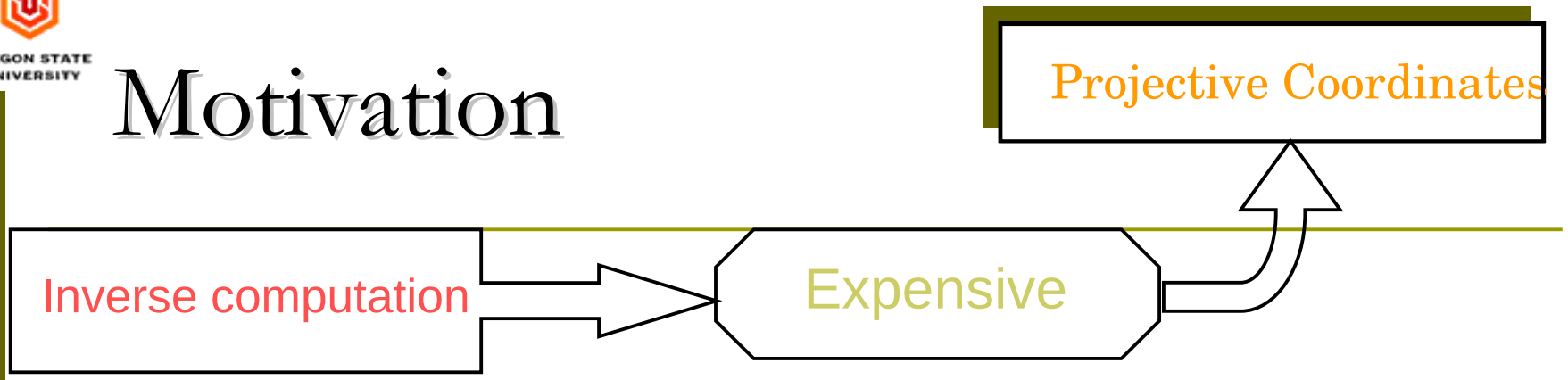
- Introduction
  - Motivation, related work
- GF(p) Montgomery inverse algorithm & hardware
  - Previous work
- GF(2^n) Montgomery inverse algorithm
  - AlmMonInv & CorPh phases
- The unified and scalable architecture implementation
- Area & speed comparisons
- Conclusions

OREGON STATE UNIVERSITY

Introduction

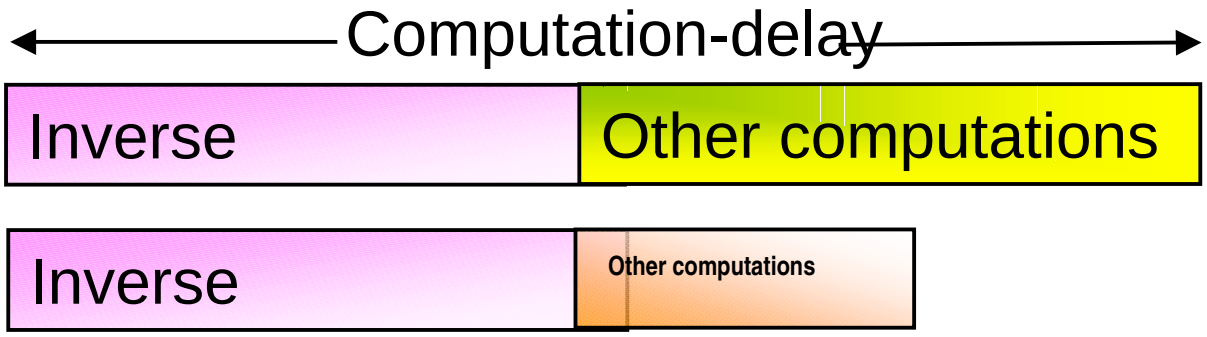
- Modular inverse is essential in public-key cryptography
  - a basic operation in the elliptic curve cryptography (ECC)
- This work is targeted mainly toward the ECC utilization
- ECC is frequently defined over finite fields GF(p) or GF(2^n)

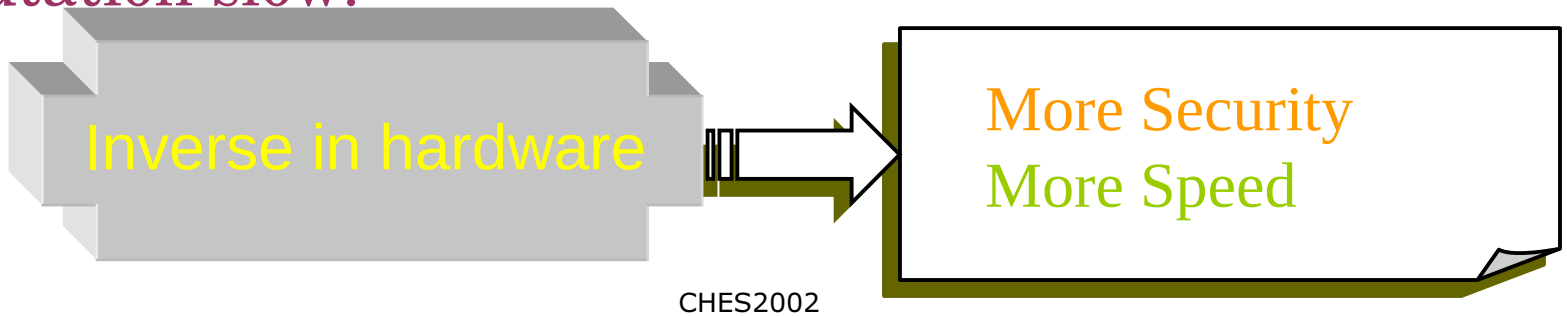
Motivation



But still inverse computation is needed once as a final stage

The inverse delay will dominate and make the whole computation slow.





Previous Work

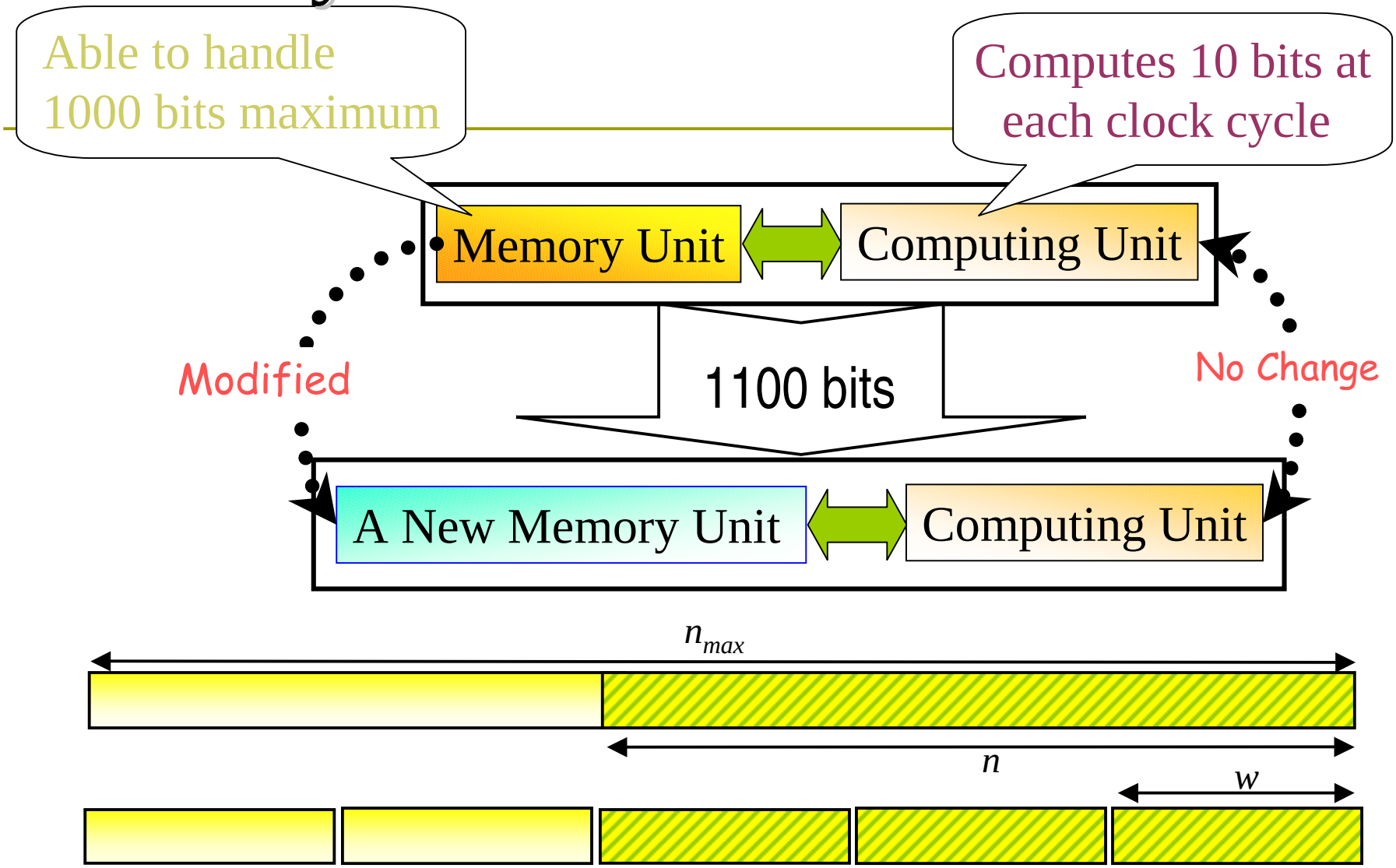
- Several designs for GF(2^n) only
  - Hasan (2001):➔ word-by-word design >> small area
- Takagi (1993): inverse algorithm with a redundant binary representation (GF(p))
  - large area and expensive data transformation.
- Goodman and Chandrakasan (2001): processor that performs inversion in both GF(p) and GF(2^n):
  - reconfigurable, designed for low power, large area.

OREGON STATE UNIVERSITY

Scalable Architecture

- Features:
  - Short design's longest path, independent of operand precision
  - Design area adjustable to available space (flexible)
  - Allows the computation with virtually infinite precision (limited only by memory).

Why Scalable Hardware?



OREGON STATE UNIVERSITY

Unified architecture

- *Definition*: An architecture is said to be unified when it is able to work with operands in both prime and binary extension fields (GF(p) and GF(2^n))

OREGON STATE UNIVERSITY

Modular Inverse (Extended Euclidean Alg.)

Phase I

Input : $a \in [1, p-1]$ and p

Output : $r \in [1, p-1]$ and k, where $r = a^{-1}2^k \pmod{p}$ and $n \leq k \leq 2n$

```
1.    u := p, v := a, r := 0, and s := 1,
2.    k := 0
3.    while (v > 0)
4.        if u is even then u := u/2, s := 2s
5.        else if v is even then v := v/2, r := 2r
6.        else if u > v then u := (u - v)/2, r := r+s, s := 2s
7.        else v := (v - u)/2, s := s+r, r := 2r
8.        k := k + 1
9.    if r ≥ p then  r := r - p
10.   return r := p - r
```

OREGON STATE UNIVERSITY

Modular Inverse (Extended Euclidean Alg.)

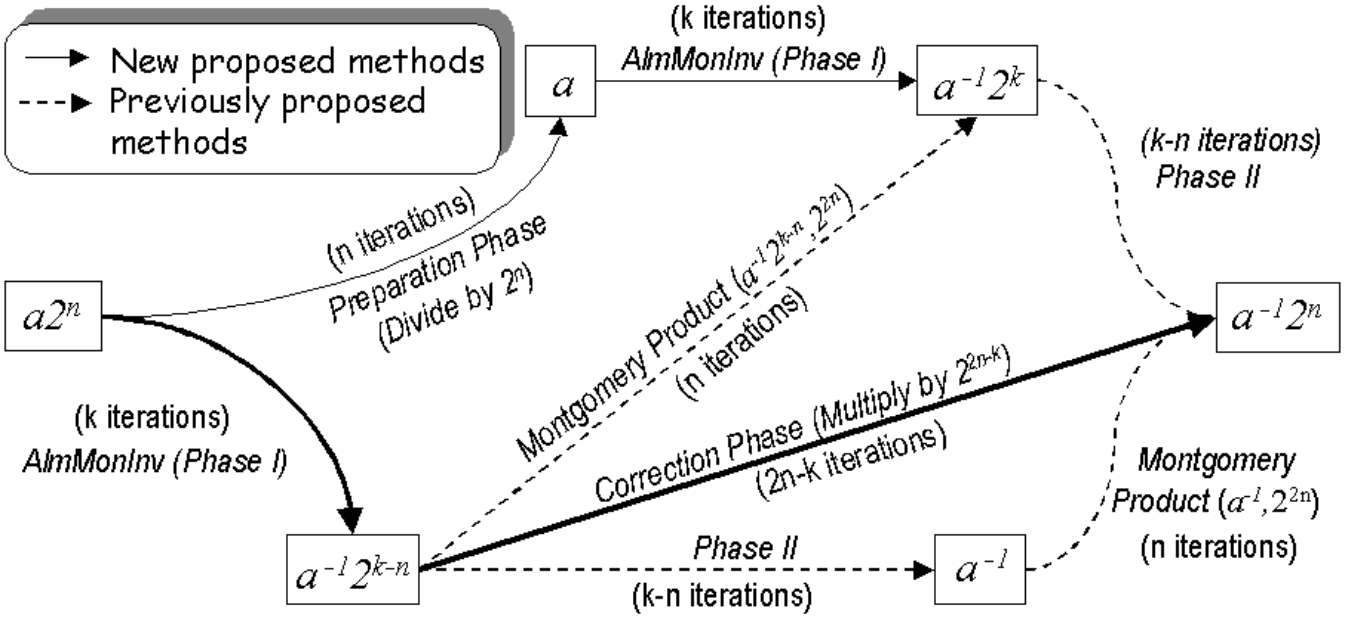
Phase II

Input : $r \in [1, p-1]$, p, and k (r and k from phase I)
Output : $x \in [1, p-1]$ where $x = a^{-1}2^n \pmod p$

```
11.     for i = 1 to k - n do
12.             if r is even then r := r/2
13.             else r := (r + p)/2
14.     return x := r
```

Montgomery Modular inverse

Based on Extended Euclidean Algorithm



Montgomery inverse hardware algorithm for GF(p)

Multi-bit shifting

MHW-Alg: GF(p) Multi-Bit Shifting AlmMonInv

Registers: u, v, r, s, x, y, z, and p (all registers hold n_{max} bits)
Input: $a2^m \in [1, p-1]$; Where p = modulus, and $m \geq n$ ($2^{n-1} \leq p < 2^n$)
Output: result$\in [1, p-1]$ & k;
Where result$=a^{-1}2^{k-m} \bmod p$ & $n<k<2n$

1. u = p; v = $a2^m$; r = 0; s = 1; x = 0; y = 0; z = 0; k = 0
2. if($u_2u_1u_0$=000)then{u=ShiftR(u,3);s=ShiftL(s,3);k=k+3};goto 8
2.1 if($u_2u_1u_0$=100)then{u=ShiftR(u,2);s=ShiftL(s,2);k=k+2};goto 8
2.2 if($u_2u_1u_0$=110)then{u=ShiftR(u,1);s=ShiftL(s,1)};goto 7
3. if($v_2v_1v_0$=000)then{v=ShiftR(v,3);r=ShiftL(r,3);k=k+3};goto 8
3.1 if($v_2v_1v_0$=100)then{v=ShiftR(v,2);r=ShiftL(r,2);k=k+2};goto 8
3.2 if($v_2v_1v_0$=110)then{v=ShiftR(v,1);r=ShiftL(r,1)};goto 8
4. x = Subtract (u, v); y = Subtract (v, u); z = Add (r, s)
5. if(x_{borrow}=0)then{u=ShiftR(x,1);v=z; s=ShiftL(s,1)};goto 7
6. s = z; v = ShiftR(y,1); r = ShiftL(r,1)
7. k = k + 1
8. if(v $\neq$ 0) go to step 2
9. x = Subtract (p, r); y = Subtract ($2p$, r)
10. if(x_{borrow} = 0)then{result=x}; else{ result = y}

MHW-Alg2:GF(p) Multi-Bit Shifting CorPh

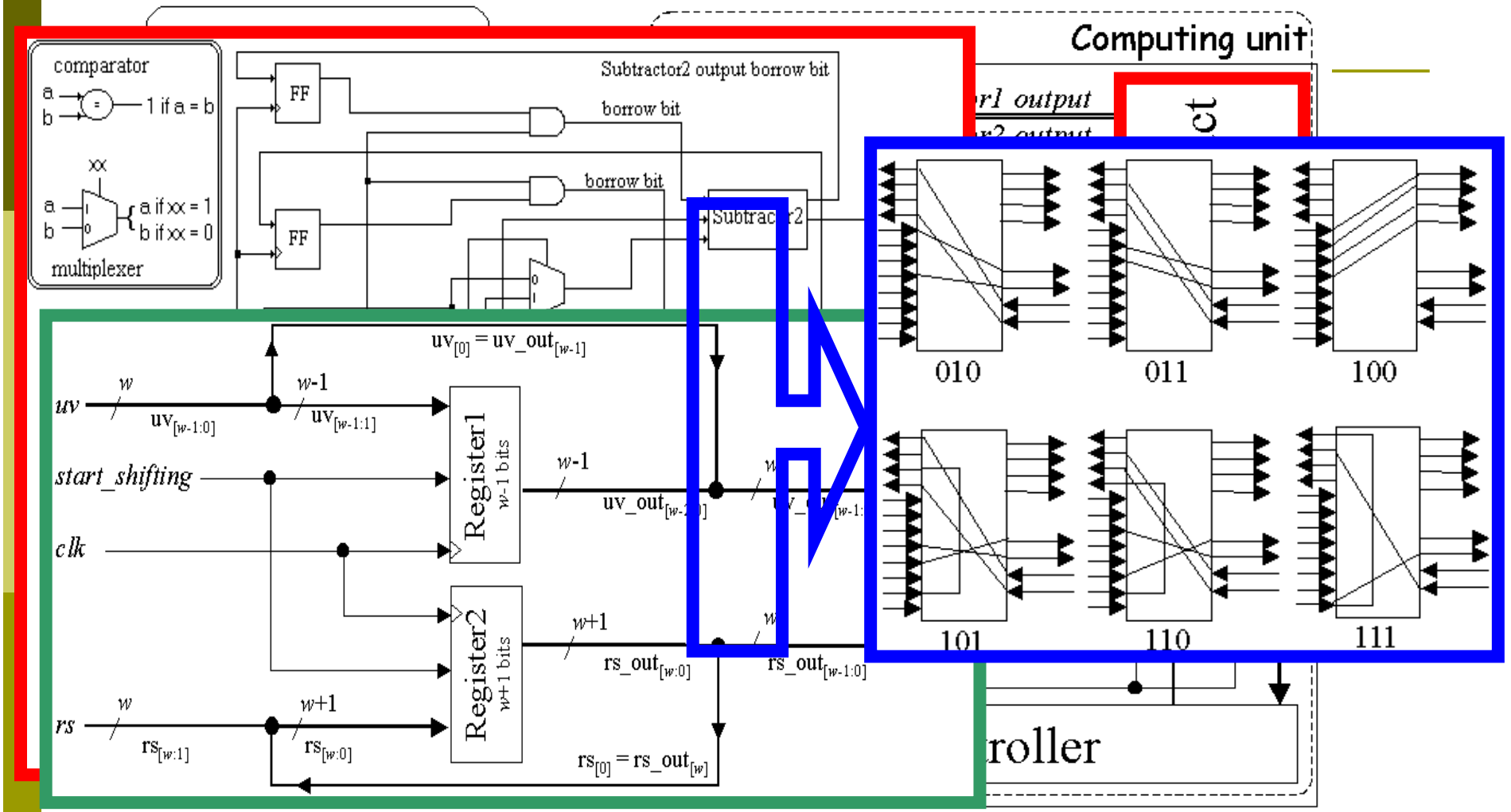
Registers: r, u, v, x, y, z, and p (all registers hold n_{max} bits)
Input: r, p, n, k;
Where r ($r=a^{-1}2^{k-m} \bmod p$)&k from *MHW-Alg*
Output: result; Where result = $a^{-1}2^m \pmod p$.

11. j = $2m$-k; x = 0; y = 0; z = 0
12. v = $2p$; u = $3p$
13. While $j > 0$
14. if(j =1) then {r = ShiftL(r,1); j=j-1}
15. else {r = ShiftL(r,2); j=j-2}
16. x= Subtract(r,p);y=Subtract(r,v);z=Subtract(r,u)
17. if(z_{borrow} = 0) then {r = z}
18. else if(y_{borrow} = 0) then {r = y}
19. else if(x_{borrow} = 0) then {r = x}
20. result = r

Multi-precision operators

ISVLSI-2002

The scalable hardware



OREGON STATE UNIVERSITY

GF(2^n) Features

- $a(x)=a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+ \ldots +a_2x^2+a_1x+a_0$, where $a_i \in$ GF(2)
  - $a=(a_{n-1}\ a_{n-2}\ \ldots\ a_2\ a_1\ a_0)$
- { + & - } in GF(p) $\equiv \oplus$ in GF(2^n)
- $p(x)=x^n+p_{n-1}x^{n-1}+p_{n-2}x^{n-2}+ \ldots +p_2x^2+p_1x+p_0$
  - $p=(1\ p_{n-1}\ p_{n-2}\ \ldots\ p_2\ p_1\ p_0)$
- $||a(x)|| = ||p(x)||$ (degree of $a(x)$ = degree of $p(x)$) $\Rightarrow a=p \oplus a$
- *Normal Subtraction (carry propagate) for degree testing*

OREGON STATE UNIVERSITY

GF(2^n) Montgomery inverse

GF(2^n) AlmMonInv Algorithm

Input: $a2^m \in GF(2^n)$ & p; (p=irreducible polynomial & $m \geq n$)
Output: result $\in [1, p-1]$ & k (result $= a^{-1}2^{k-m} \bmod p$ & $n < k < 2n$)

1. u = p; v = $a2^m$; r = 0; s = 1; k = 0
2. While (v > 0)
3. if $u_0 = 0$ then {u = u/2; s = 2s}
4. else if $v_0 = 0$ then {v = v/2; r = 2r}
5. else if u > v then {u = (u⊕v)/2; r = r⊕s; s = 2s}
6. else {v = (u⊕v)/2; s = r⊕s; r = 2r}
7. $k=k+1$
8. if $r \geq 2^{n+1}$ ($\|r\| > \|p\|$) then {result = $2p \oplus r$}
9. else if $r \geq 2^n$ ($\|r\| = \|p\|$) then {result = $p \oplus r$}
10. else result = r

GF(2^n) CorPh Algorithm

Input: r, p, m, & cowherd r & k from *AlmMonInv*
Output: result; Where result = $a^{-1}2^m \pmod p$

11. $j = 2m-k$
12. While $j > 0$
13. $r = 2r$
14. if $r \geq 2^n$ ($\|r\| = \|p\|$) then {r = $p \oplus r$}
15. $j = j-1$
16. result = r

Montgomery inverse hardware algorithm for GF(2^n)

MHW-Alg3:GF(2^n) Multi-Bit Shifting AlmMonInv

Registers: u, v, r, s, x, y, z, & p (all registers hold n_{max} bits)
Input: $a2^m, 2^n \in [1,p\text{-}1]$ (p=irreducible polynomial & $m \geq n$)
Output: result$\in[1, p\text{-}1]$ & k (result=$a^{-1}2^{k-m} mod\ p$ & $n<k<2n$)

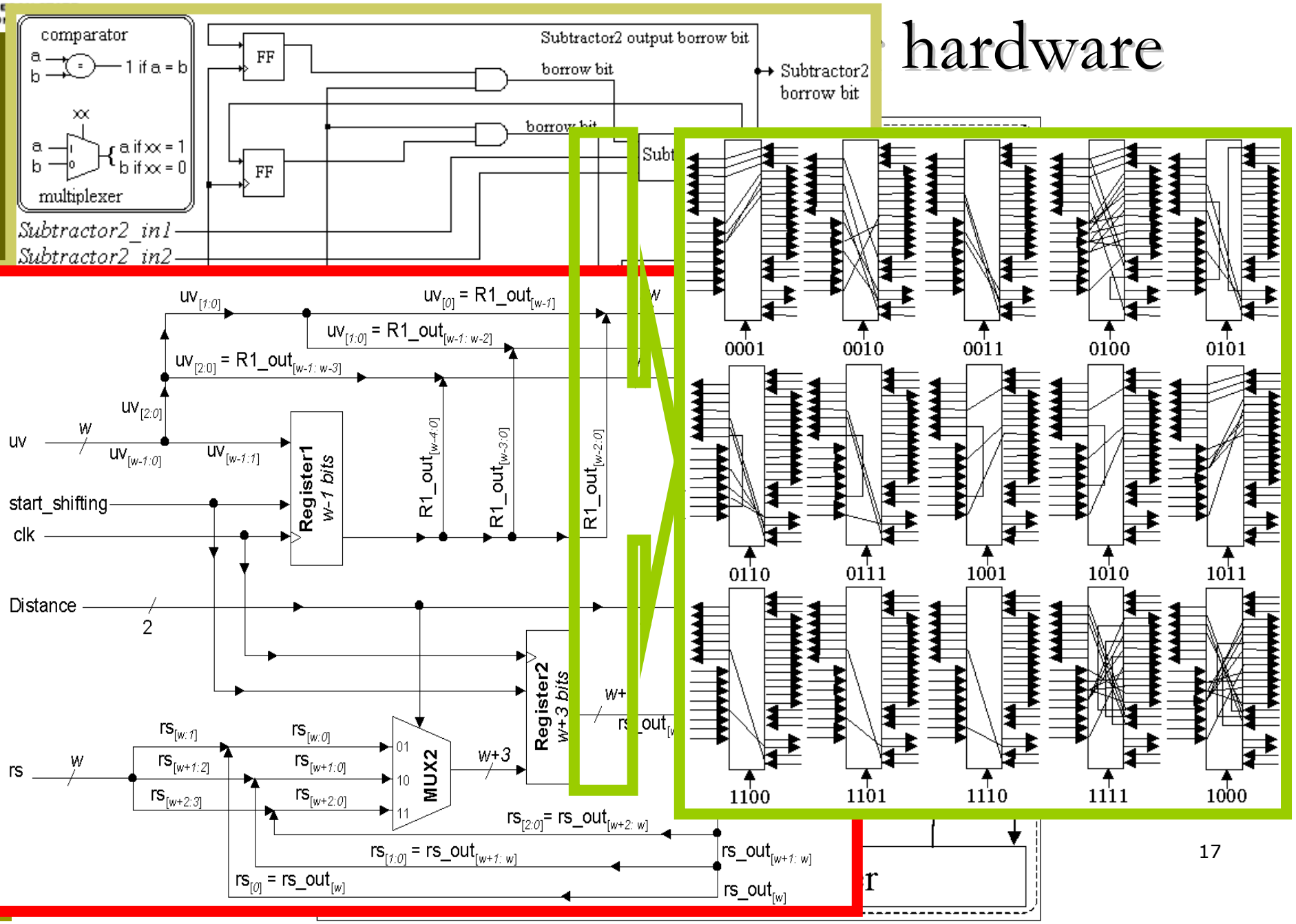
1. u = p; v = $a2^m$; r = 0; s = 1; x = 0; y = 2^n; z = 0; k = 0
2. if($u_2u_1u_0$=000)then{u=ShiftR(u,3);s=ShiftL(s,3);k=k+3};goto 8
2.1. if($u_2u_1u_0$=100)then{u=ShiftR(u,2);s=ShiftL(s,2);k=k+2};goto 8
2.2. if($u_2u_1u_0$=110)then{u=ShiftR(u,1);s=ShiftL(s,1)};goto 7
3. if($v_2v_1v_0$=000)then{v=ShiftR(v,3);r=ShiftL(r,3);k=k+3};goto 8
3.1. if($v_2v_1v_0$=100)then{v=ShiftR(v,2);r=ShiftL(r,2);k=k+2};goto 8
3.2. if($v_2v_1v_0$=110)then{v=ShiftR(v,1);r=ShiftL(r,1)};goto 8
4. S1 = Subtract (u, v); x = v⊕u; z = r⊕s
5. if($S1_{borrow}$=0)then{u=ShiftR(x,1); r=z; s=ShiftL(s,1)};goto 7
6. s = z; v = ShiftR(x,1); r = ShiftL(r,1)
7. k = k + 1
8. if(v ≠ 0) go to step 2
9. x = p⊕r ; z = $2p$⊕r ; S1 = Subtract (y,x); S2 = Subtract (y,z)
10. if($S1_{borrow}$=0)then{result=x}
10.1 else if($S2_{borrow}$=0)then{result=z}
10.2 else {result = r}

MHW-Alg4:GF(2^n) Multi-Bit Shifting CorPh

Registers: r, u, v, s, x, y, z, & p (all registers hold n_{max} bits)
Input: $r, p, m, 2^n$ & k;
Where r ($r=a^{-1}2^{k-m} mod\ p$)& k from *HW-Alg3*
Output: result; Where result = $a^{-1}2^m$ *(mod p)*.

11. j = *2m-k-1*; x = 0; y = 0; z = 0
12. v = $2p$; u = $3p$; s = 2^n
13. While $j > 0$
14. if j =*1* then {r = ShiftL(r,1); j=*j-1*}
15. else {r = ShiftL(r,2); j=*j-2*}
16. x = p⊕r ; y = u⊕r ; z = u⊕r
16.1 S1=Subtract(s,x);S2=Subtract(s,y);S3=Subtract(s,z)
17. if ($S3_{borrow}$ = 0) then {r = z}
18. else if ($S2_{borrow}$ = 0) then {r = y }
19. else if ($S1_{borrow}$ = 0) then {r = x}
20. result = r

hardware


comparator
1 if a = b
multiplexer
a if xx = 1
b if xx = 0
FF
Subtractor2 output borrow bit
borrow bit
Subtractor2 borrow bit
Subtractor2_in1
Subtractor2_in2
uv
start_shifting
clk
Distance
rs
Register1 w-1 bits
Register2 w+3 bits
MUX2
0001
0010
0011
0100
0101
0110
0111
1001
1010
1011
1100
1101
1110
1111
1000

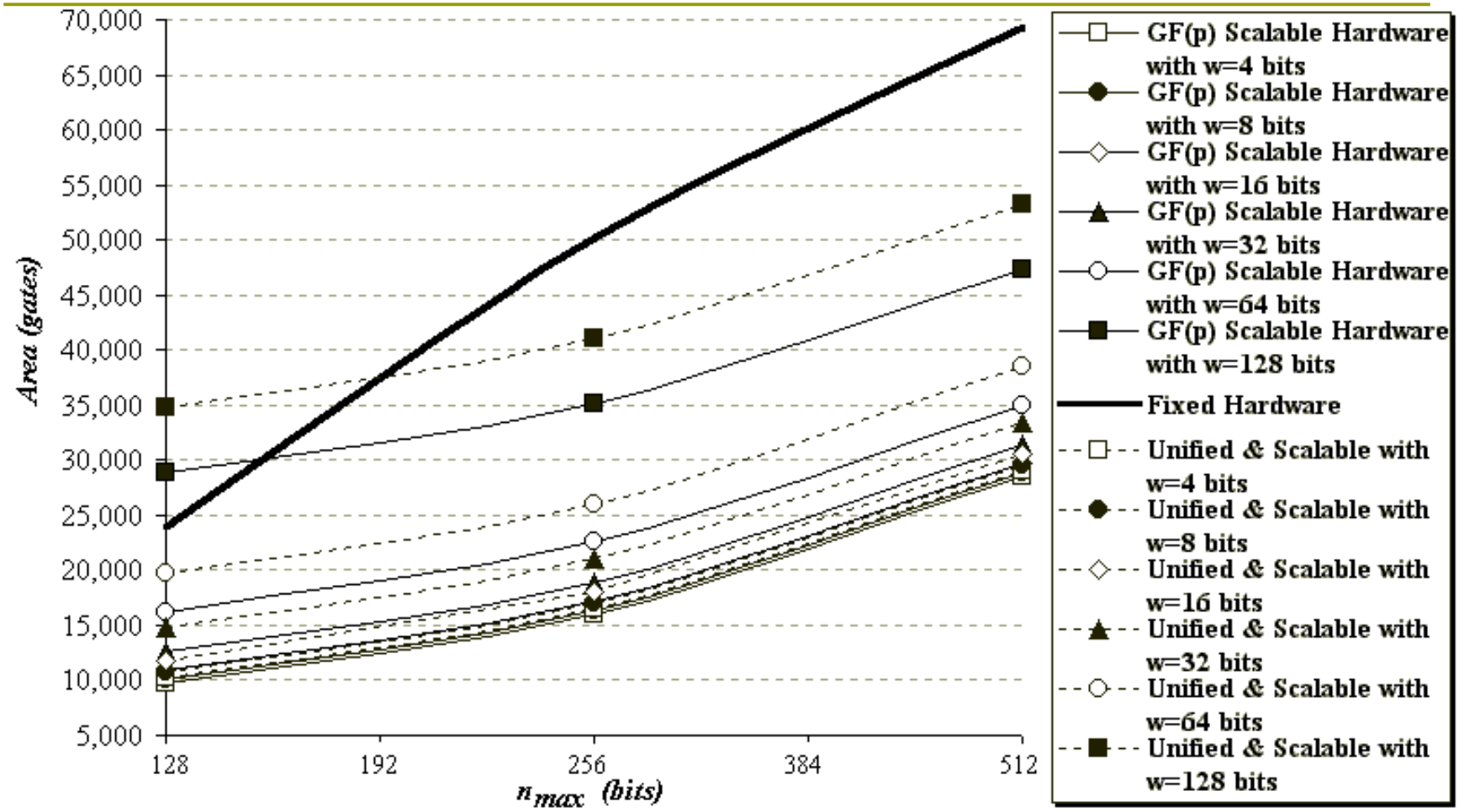
OREGON STATE UNIVERSITY

Experimental Results

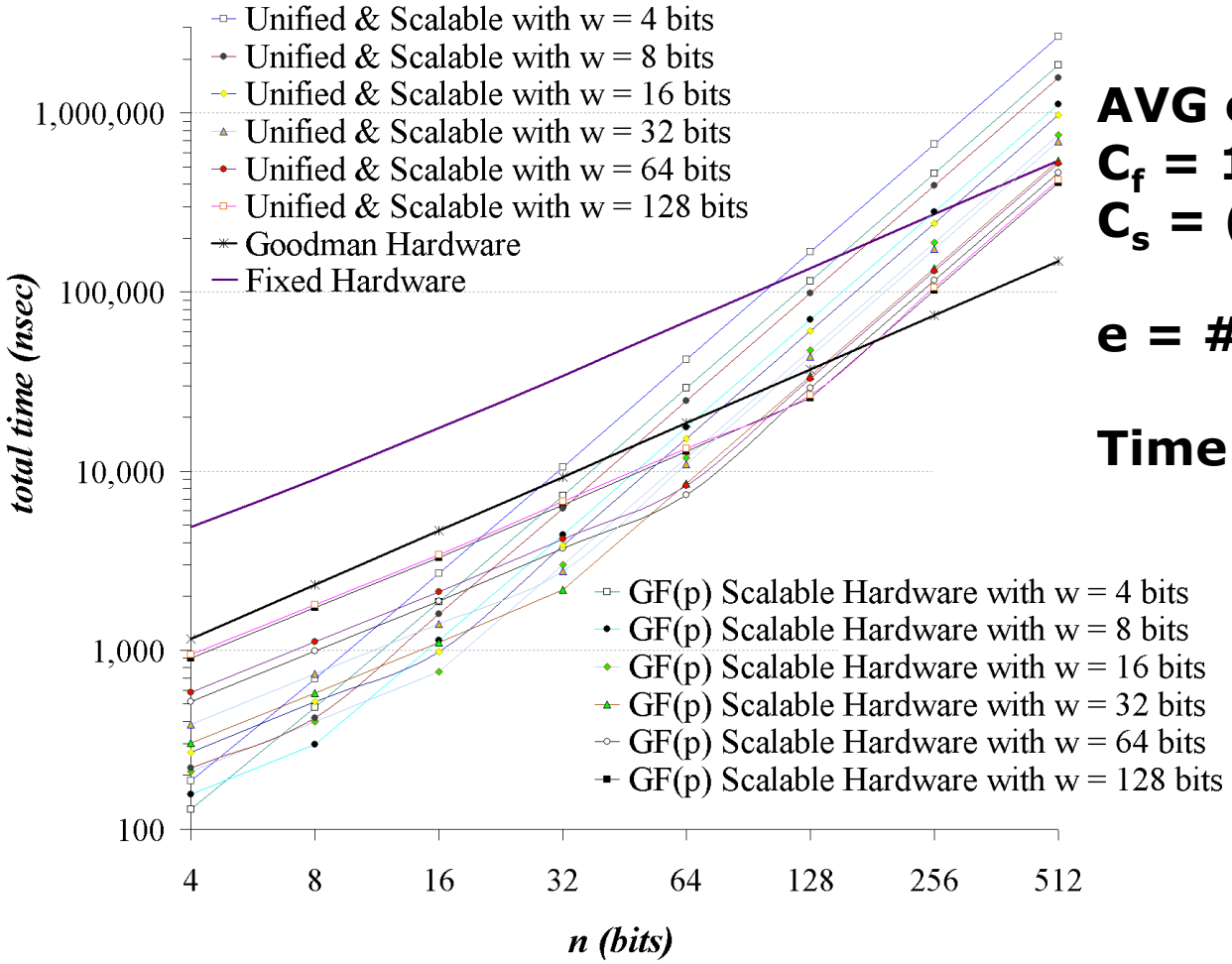
- VHDL ➔ functional simulation
- Maple – verification.
- Leonardo (Mentor Graphics) – synthesis
  - 0.5 Micron CMOS technology - ASIC Design Kit (ADK)
  - VHDL code compiled to obtain estimates for:
    - ✓Area ➔ the number of gates
    - ✓Clock Period➔Longest path delay (nanoseconds)

OREGON STATE UNIVERSITY

Area Comparison



Speed Comparison (n_{max}=512 bits)



AVG cycles
C_f = 1.53n
C_s = (2.4n+1)e

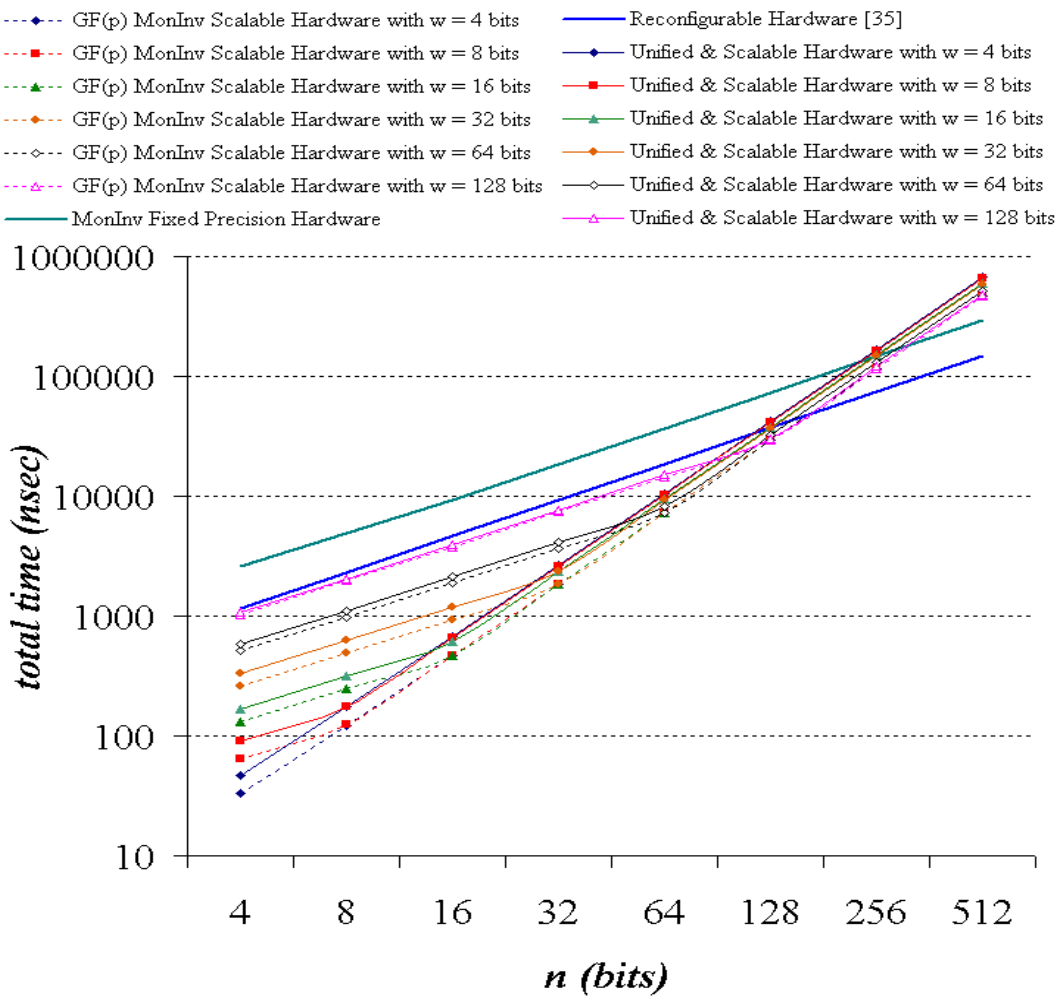
e = #words

Time=C*cycle_time

OREGON STATE UNIVERSITY

Speed Comparison (n_{max}=512 bits)

Technology independent



OREGON STATE UNIVERSITY

Conclusions

- A scalable and unified architecture that operates in both GF(p) and GF(2^n) fields was proposed.
- Adjusted a GF(2^n) MonInv algorithm to include the multi-bit shifting method making it very similar to a previously proposed GF(p) inversion hw design.
- A comparison of the scalable & unified design with a reconfigurable hardware shows that the scalable design saves a lot of area and operates at comparable speed.
- Scalable/unified design has similar or better performance than a fixed-precision design, with significantly less area.
- Small extra cost to add unified design feature to previously proposed design for GF(p) only. (around 8.4%).

Note

- Figures in the paper included in the proceedings -> difficult to read
  - Contact tenca@ece.orst.edu

THANK YOU