Workshop on Cryptographic Hardware and Embedded Systems 2002

Genus Two Hyperelliptic Curve Coprocessor

T.C. Clancy, Y. Liow, J.E. Webster, N. Boston

Illinois Center for Cryptography and Information Protection
Coordinated Science Laboratory
University of Illinois, Urbana-Champaign

Outline

- Hyperelliptic Curves
- Finite Field Implementation
- Polynomial Ring Implementation
- Point Adder and Doubler
- Point Multiplication
- Performance Results

Hyperelliptic Curves

- Defined by equation:

$$v^2 + H(u)v = F(u)$$

Polynomials

- $F(u), H(u) \in \mathrm{GF}(2^n)[u]$

Ring

Finite Field

- For genus g curve:
  - $\deg(F(u)) = 2g+1$
  - $\deg(H(u)) \leq g$

Implemented Curve

- Base Field: GF(2^{113})
- Genus $g = 2$

- Equation:

$$v^2 + uv = u^5 + u^2 + 1$$

- $F(u) = u^5 + u^2 + 1$
- $H(u) = u$

Jacobian of a Curve

Polynomials

- **Divisor:** Pair div(A,B) satisfying:

$$B^2 + H(u)B \equiv F(u) \pmod{A}$$

- **Reduced Divisor:** Divisor such that:
  - $\deg(A) \leq g$
  - $\deg(B) < \deg(A)$
- **Jacobian:** Set of all reduced divisors; *group* under binary operation defined in *Cantor's Algorithm*

Finite Field Implementation

- Using a polynomial basis
- Need the following Finite Field Units:
  - Addition
    - bitwise XOR
  - Multiplication
    - for each bit, AND → XOR → SHL → Reduce
  - Squaring
    - space out lower bits, reduce upper bits
  - Inversion
    - used [HHM00] method

Finite Field Results

Module	Cycles	Slices	Frequency
Addition	Combinatorial	No separate module created	
Multiplication	2 or 115	399	96 MHz
Squaring	2 or 59	186	124 MHz
Inversion	395 (avg)	1,631	98 MHz

Polynomial Implementation

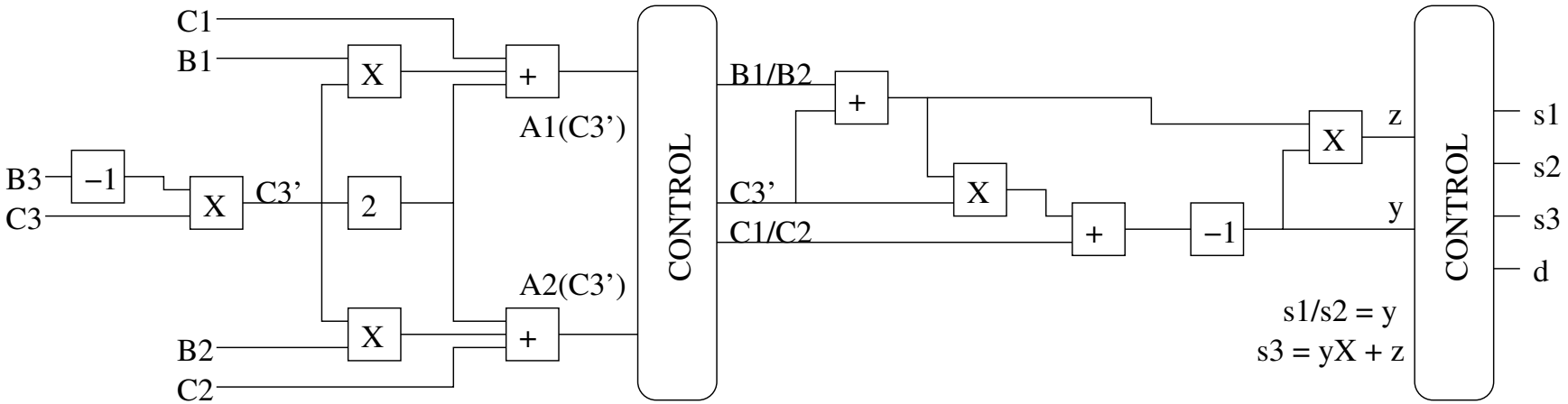
- Polynomials over GF(2^n), maximal degree 6
- Need the following Polynomial Units:
  - Addition
    - bitwise XOR
  - Multiplication, Squaring
    - same as FF, but over field elements, not bits
  - Division
    - straight division algorithm implementation
  - Greatest Common Divisor
    - Used a new method for computing GCD

New GCD

- Want:

$$d = \gcd(a_1, a_2, a_3) = s_1 a_1 + s_2 a_2 + s_3 a_3$$

- If $\deg(a_1, a_2, a_3) \neq (2,2,1)$ use EEA; else:

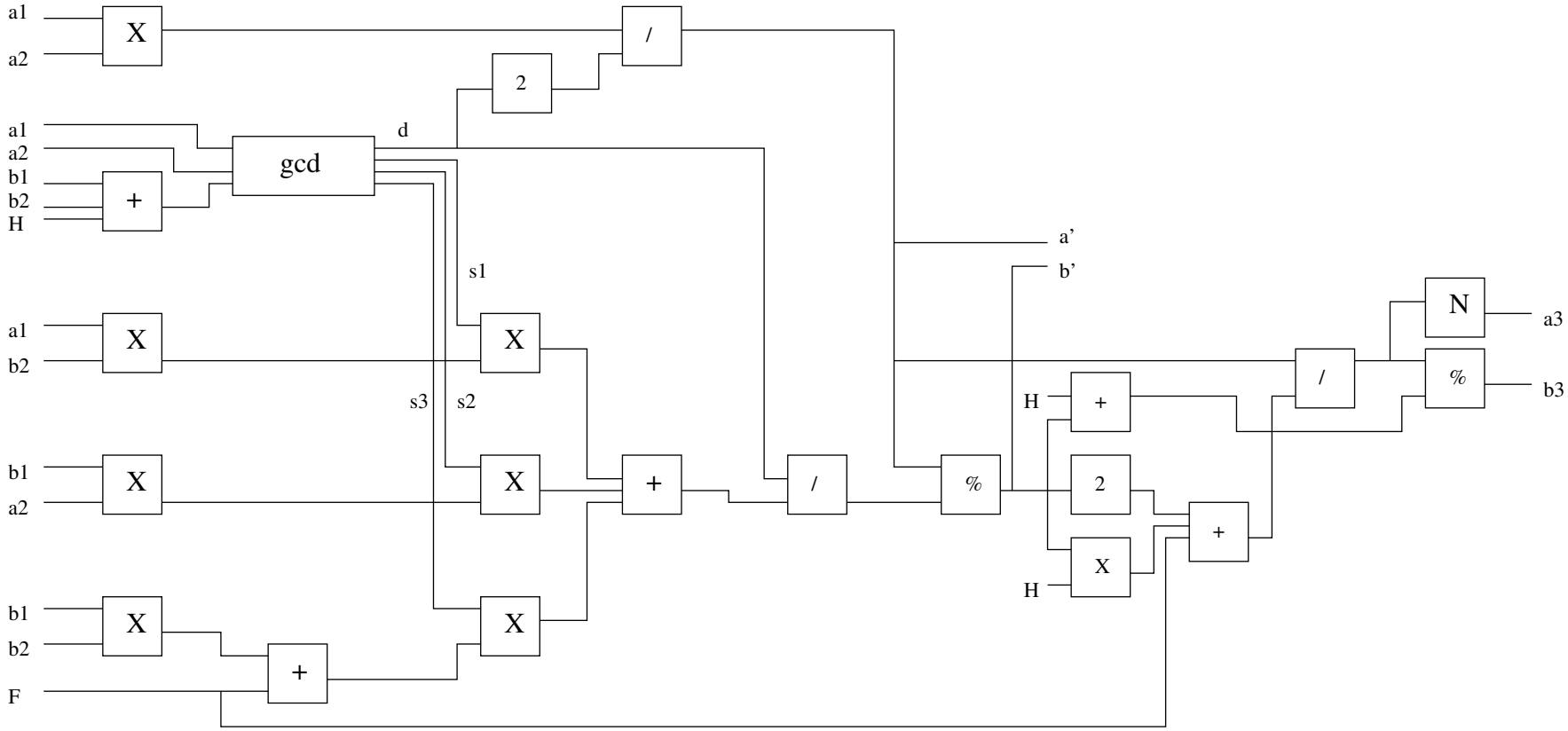


Polynomial Results

Module	Cycles	Slices	Frequency
Addition	Combinatorial	791	83 MHz
Multiplication	2 to 353	1,561	64 MHz
Squaring	2 or 59	515	55 MHz
Division	2 to 2,300	8,337	80 MHz
GCD	1,270 (avg)	3,515	96 MHz
Norm	615 (avg)	2,488	71 MHz

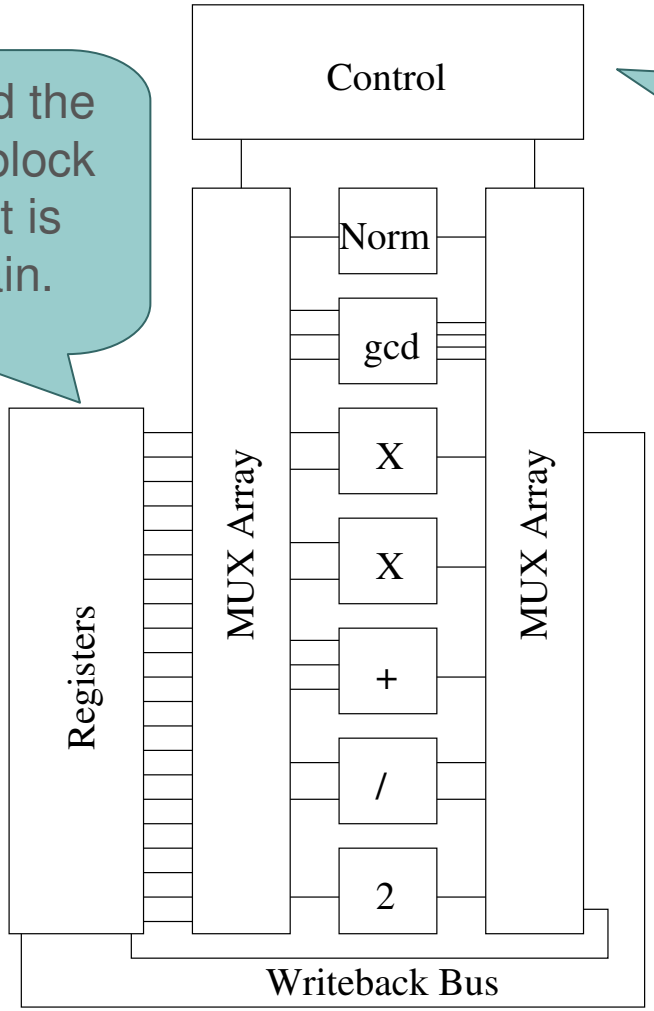
Point Addition

All blocks are **polynomial** blocks



More Efficient Architecture

Registers hold the computation block output until it is needed again.



Control unit schedules polynomial calculation blocks and channels data to implement Cantor's algorithm.

Implementation Results

Module	Cycles	Slices	Frequency
Adder	4,750	16,600	45 MHz
Doubler	4,050	15,100	45 MHz

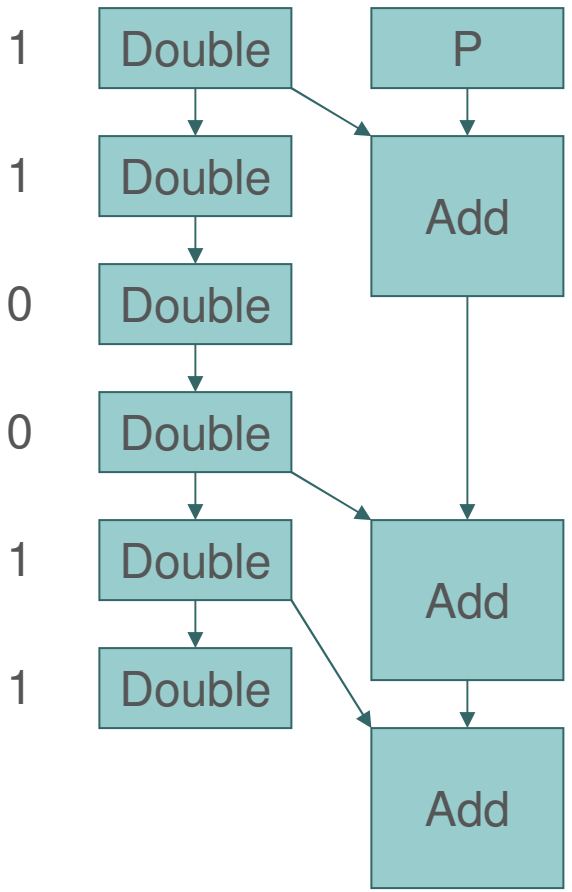
Point Multiplication

- Defined as: $kP = \sum_{i=1}^{k} P$
- Use distributive property
  - Express *k* in binary representation
  - Compute a basis by repeatedly doubling *P*
  - For each 1 in the binary representation of *k*, add the appropriate basis to compute *kP*

Theoretical Performance

- Point adder and point doubler operating in parallel
- Dependencies can allow the adder to get behind, but never ahead

Example, $k = 51_{10} = 110011_2$



General case: D/G/1 Queue

- Θ is an endomorphism of group G
- k_i be statistically independent, not equaling 0 with probability δ
- α be the time to perform group addition
- β be the time required to compute $(k_i \Theta^i P)$ given $(\Theta^{i-1} P)$
- $\delta\alpha < \beta$
- devices operating in parallel

General case: D/G/1 Queue

- Expected time to compute

$$\left(\sum_{i=0}^{n-1} k_i \Theta^i\right) P = \sum_{i=0}^{n-1} k_i (\Theta^i P)$$

- is bounded above by

$$\frac{\alpha^2 \delta (1-\delta)}{2(\beta - \alpha\delta)} + \alpha\delta + (n-1)\beta$$

Overall Results

- Point multiplication can be achieved in 10.1 milliseconds
- Smallest Xilinx FPGA capable of supporting the design is the Virtex II 2VP30, which has 30,816 slices