The Hitchhiker's Guide to EC-Adder HW Acceleration

Yuval Domb
yuval@ingonyama.com

Mission

- *Build a fully-pipelined BLS12-381 ECADDER*
- *Use Xilinx Virtex Ultrascale+*
- *Minimize DSP48 blocks*

One Slice	Two Slices
26Ux19U	34Ux28U
27Ux18U	35Ux27U
28Ux17U	36Ux26U

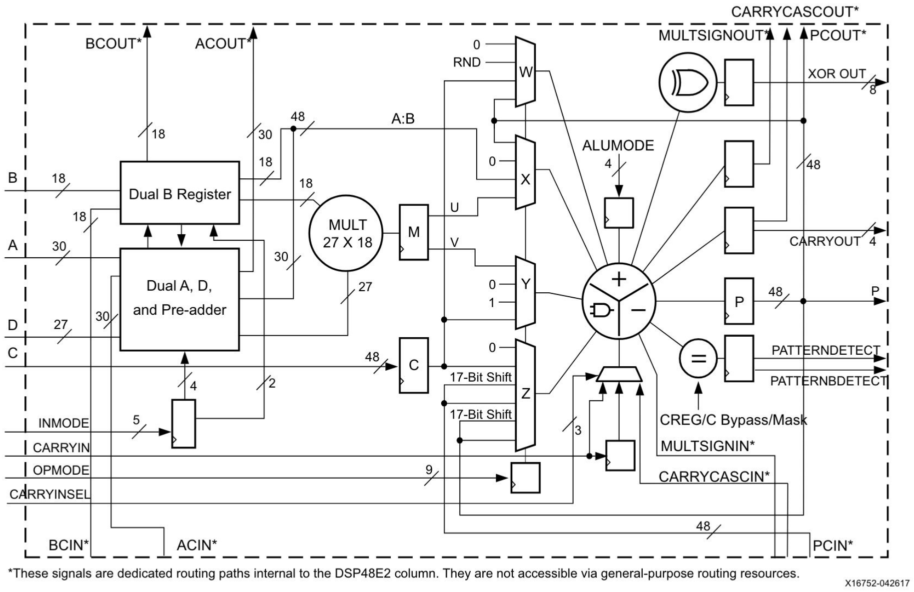


Figure 2-1: **Detailed DSP48E2 Functionality**

"UltraScale Architecture DSP Slice User Guide (UG579)", https://docs.amd.com/v/u/en-US/ug579-ultrascale-dsp

Elliptic Curve

EC Addition Formulas

Modular Multiplier Optimization

ECADDER Optimization

Toom-Cook Optimization

What is an Elliptic Curve?

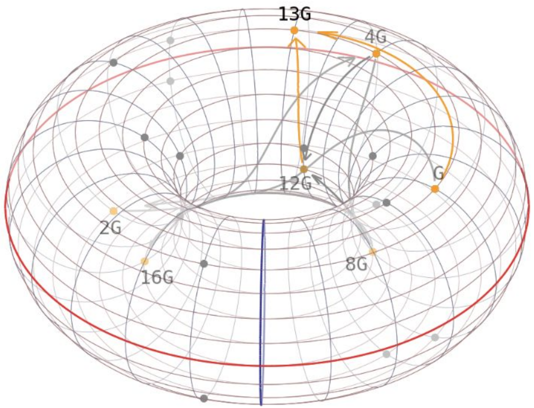
- *A smooth, projective, algebraic curve of genus 1*

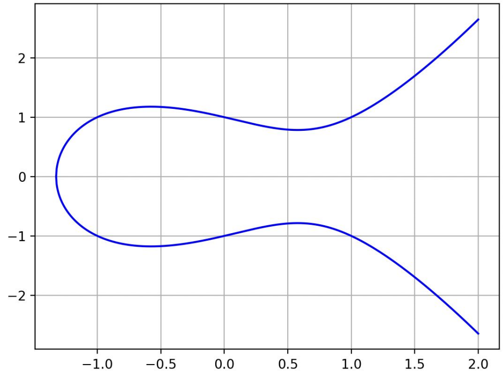
- *Short Weierstrass Form*

$$y^2 = x^3 + ax + b$$

$$4a^3 + 27b^2 \neq 0$$

$$a, b, x, y \in \mathbb{F}_{p^m}, \forall p > 3$$





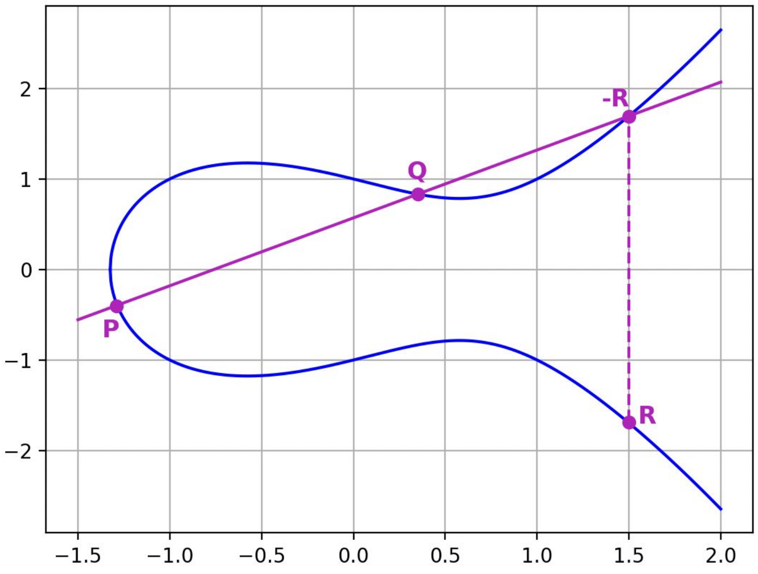
Dominik Joe Pantůček, “Elliptic curves: double and add ”, https://trustica.cz/2018/05/03/elliptic-curves-double-and-add/

EC Additive Group $E(\mathbb{F}_p)$

- *All points on curve + Point-at-Infinity* $\mathcal{O}$
- *Define addition geometrically*
- *Cyclic with Generator* G
- *Discrete-Log hardness*

$$n, G \rightarrow nG$$

$$n \not\leftarrow G, nG$$



EC Pairings

- *Distinct prime-sized subgroups, extensions, embeddings, twists...*
- *Pairing function - a bilinear map*

$$e : G_1 \times G_2 \to G_T$$

$$e(P, Q + R) = e(P, Q) \cdot e(P, R)$$

$$e(P + Q, R) = e(P, R) \cdot e(Q, R)$$

$$e(aP, bQ) = e(abP, Q) = e(P, abQ) = e(bP, aQ) = e(P, Q)^{ab}$$

Example: BLS12-381 Elliptic Curve

Barreto, Lynn, Scott

$$p = 0x1a0111ea397fe69a4b1ba7b6434bacd764774b84f38512bf6730d2a0f6b0f6241eabfffeb153ffffb9feffffffffaaab$$

$$r = 0x73eda753299d7d483339d80809a1d80553bda402fffe5bfeffffffff00000001$$

$$G_1 \subset E(\mathbb{F}_p), \quad E : y^2 = x^3 + 4$$

$$G_2 \subset E'(\mathbb{F}_{p^2}), \quad E' : y^2 = x^3 + 4(1+i)$$

$$G_T \subset \mathbb{F}_{p^{12}}$$

Ben Edgington, “BLS12-381 For The Rest Of Us”, https://hackmd.io/@benjaminion/bls12-381#BLS-digital-signatures

Example: BLS Digital Signatures

Boneh, Lynn, Shacham

- Secret key: $s \in \mathbb{F}_r$
- Public key: $[s]_1 \in G_1$
- Message: $m \in \mathbb{F}_r$
- Signature: $[sm]_2 \in G_2$
- Verification: $e([s]_1, [m]_2) = e([1]_1, [sm]_2)$

Ben Edgington, “BLS12-381 For The Rest Of Us”, https://hackmd.io/@benjaminion/bls12-381#BLS-digital-signatures

Elliptic Curve

EC Addition Formulas

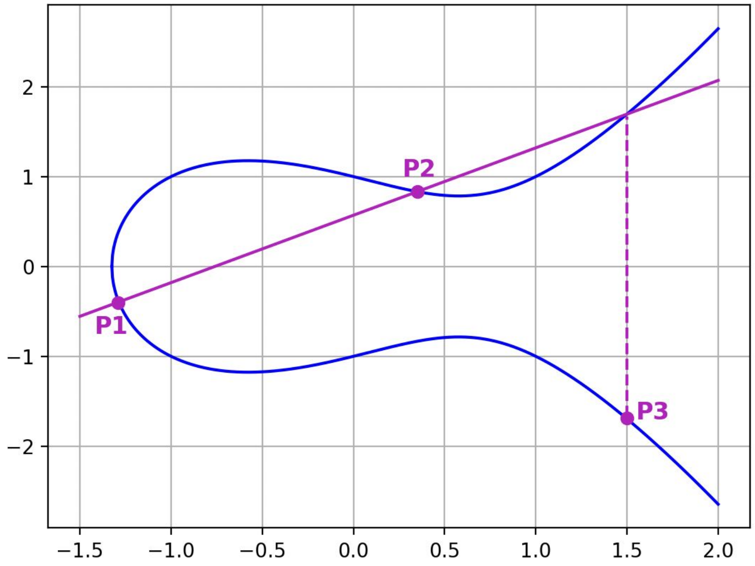
Modular Multiplier Optimization

ECADDER Optimization

Toom-Cook Optimization

Affine Addition (Geometric)

- *Formula:*
  - $\lambda = \frac{1}{x_2 - x_1}(y_2 - y_1)$
  - $x_3 = \lambda^2 - x_1 - x_2$
  - $y_3 = \lambda(x_1 - x_3) - y_1$
- *Cost:*
  - *3 multiplication*
  - *1 inversion*

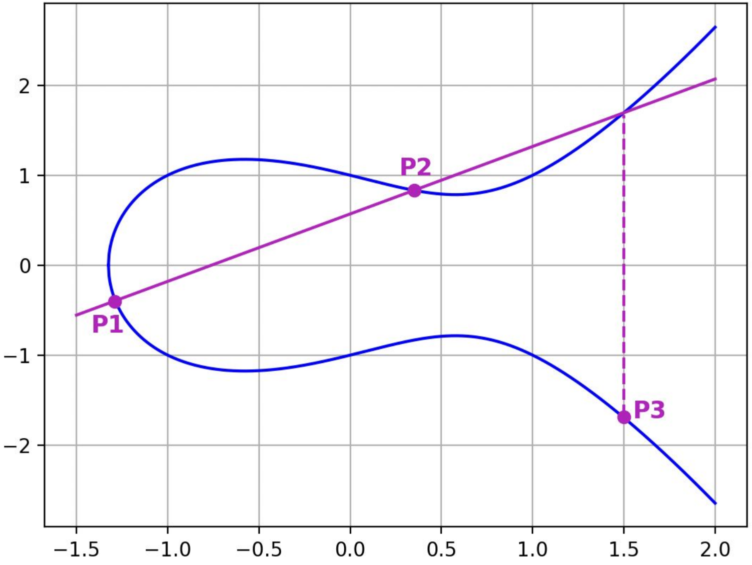


Batched Affine Addition

- *Batch many independent inversions*
  - *Calc iteratively:* $v_0, v_0 v_1, ..., \prod_0^m v_i$
  - *Invert:* $w = (\prod_0^m v_i)^{-1}$
  - *Calc recursively:* $\forall j \in [m:0]$

    $v_m^{-1} = w \prod_0^{m-1} v_i$

    $w \leftarrow w v_m$



- *Amortised Cost: 6 multiplication*
- *Requires: many independent additions, large memory for recursion*

Projective Addition

- Redundant representation: $(x, y, z) \rightarrow \left(\frac{x}{z}, \frac{y}{z}\right)$
- Complete formula (Bosma and Lenstra, 1995)

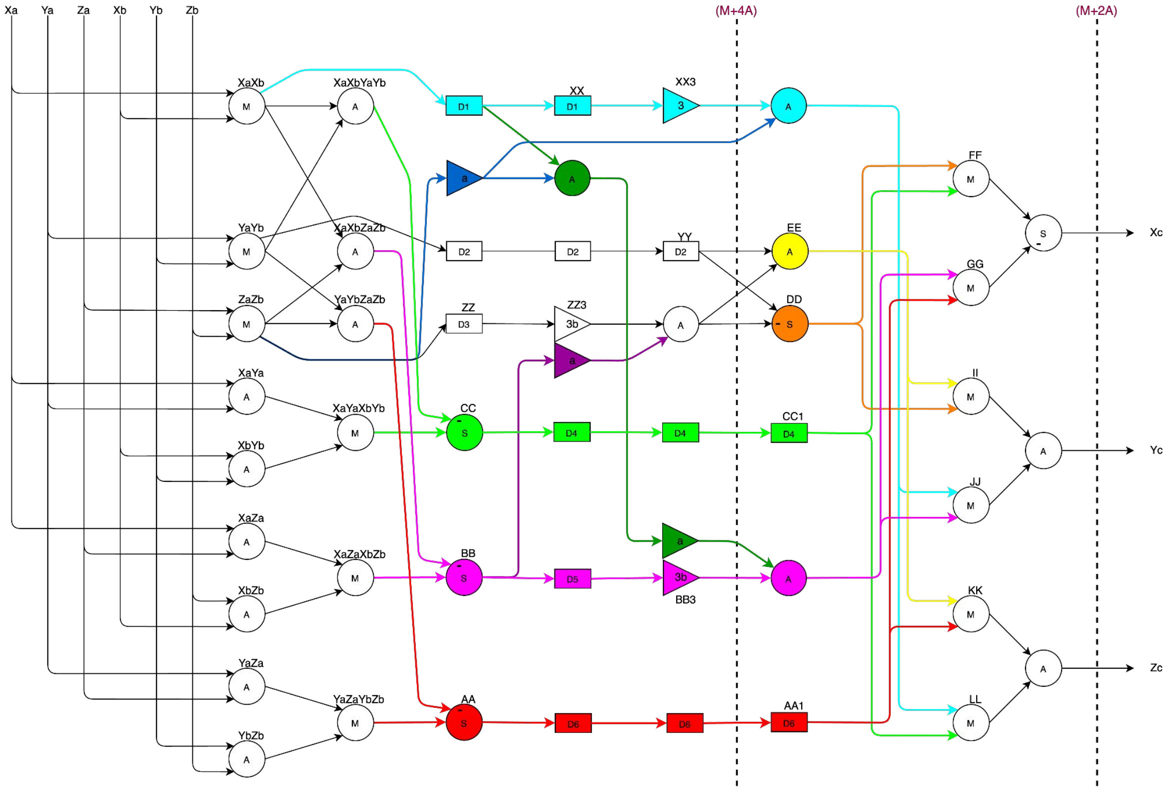
$$
\begin{aligned}
X_3 = &(X_1Y_2 + X_2Y_1)(Y_1Y_2 - a(X_1Z_2 + X_2Z_1) - 3bZ_1Z_2) \\
&- (Y_1Z_2 + Y_2Z_1)(aX_1X_2 + 3b(X_1Z_2 + X_2Z_1) - a^2Z_1Z_2), \\
Y_3 = &(3X_1X_2 + aZ_1Z_2)(aX_1X_2 + 3b(X_1Z_2 + X_2Z_1) - a^2Z_1Z_2) \\
&+ (Y_1Y_2 + a(X_1Z_2 + X_2Z_1) + 3bZ_1Z_2)(Y_1Y_2 - a(X_1Z_2 + X_2Z_1) - 3bZ_1Z_2), \\
Z_3 = &(Y_1Z_2 + Y_2Z_1)(Y_1Y_2 + a(X_1Z_2 + X_2Z_1) + 3bZ_1Z_2) \\
&+ (X_1Y_2 + X_2Y_1)(3X_1X_2 + aZ_1Z_2).
\end{aligned}
$$

- Cost: 12 field multiplications
- Highly parallelizable

@ Karthik Inbasekar

Massolino et al., "Implementing Complete Formulas on Weierstrass Curves in Hardware", https://eprint.iacr.org/2016/1133.pdf

Projective Addition



Comparison of EC Addition Formulas

Type	Mults	Sqrs	Total
Batched Affine	5	1	6
Jacobian + Affine	7	4	11
Ext. Jacobian + Affine	8	2	10
Jacobian	11	5	16
Ext. Jacobian	12	2	14
Projective	12	0	12

"Short Weierstrass curves", https://www.hyperelliptic.org/EFD/g1p/auto-shortw.html

Elliptic Curve

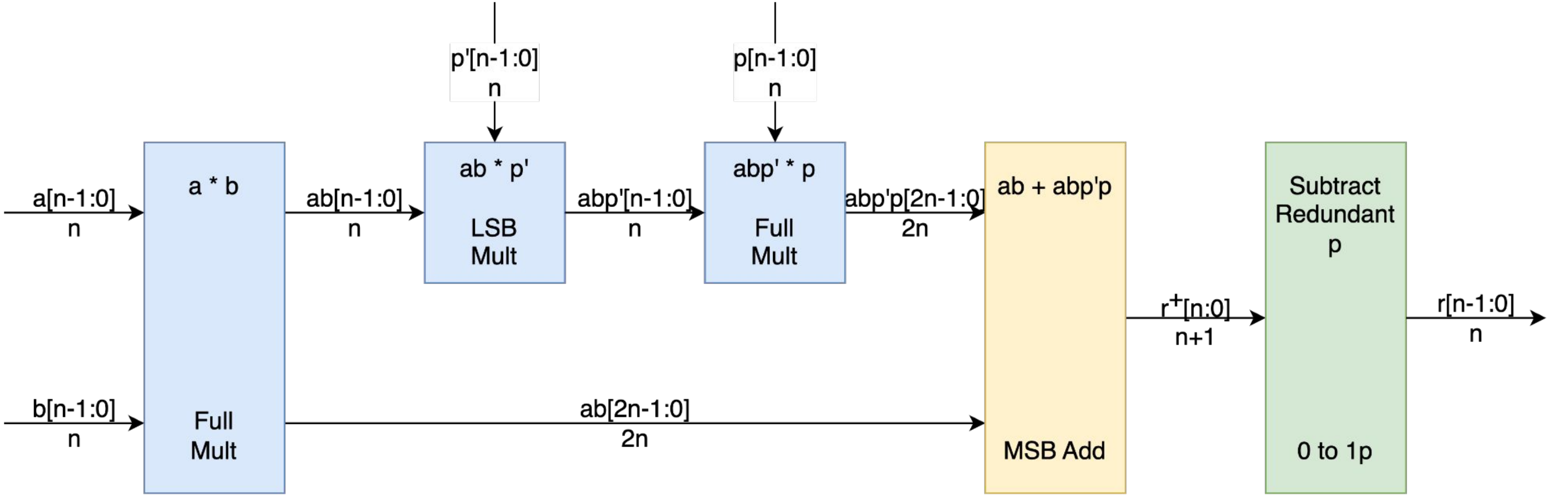
EC Addition Formulas

Modular Multiplier Optimization

ECADDER Optimization

Toom-Cook Optimization

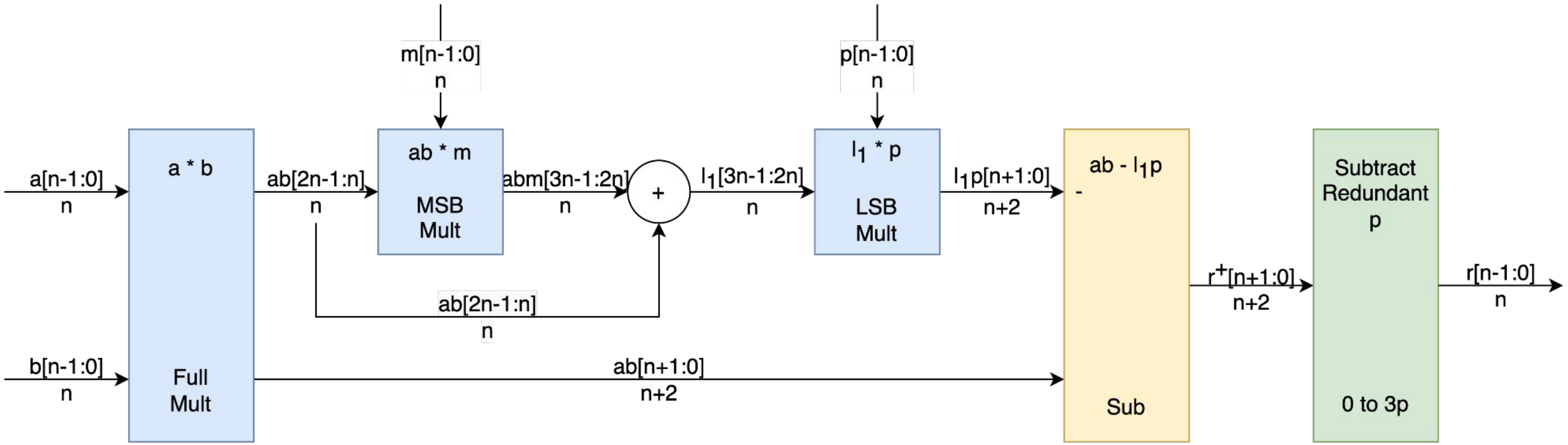
Montgomery Multiplier



- *Requires special transformation to Montgomery representation*
- *Good for multi-precision systems such as CPUs*

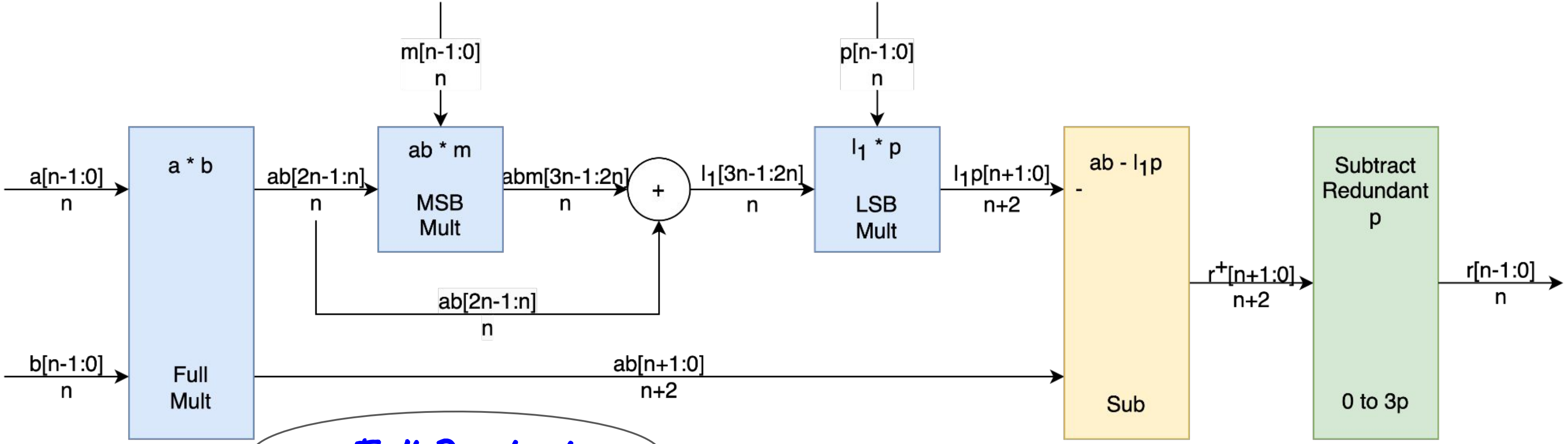
Joppe W. Bos and Peter L. Montgomery., "Montgomery Arithmetic from a Software Perspective", https://eprint.iacr.org/2017/1057.pdf

Barrett Multiplier ✓



- *Works for any modulo reduction*
- *Uses native number representation*
- *Provides many opportunities for general-purpose hardware optimization*

Barrett Multiplier ✓



Full Product

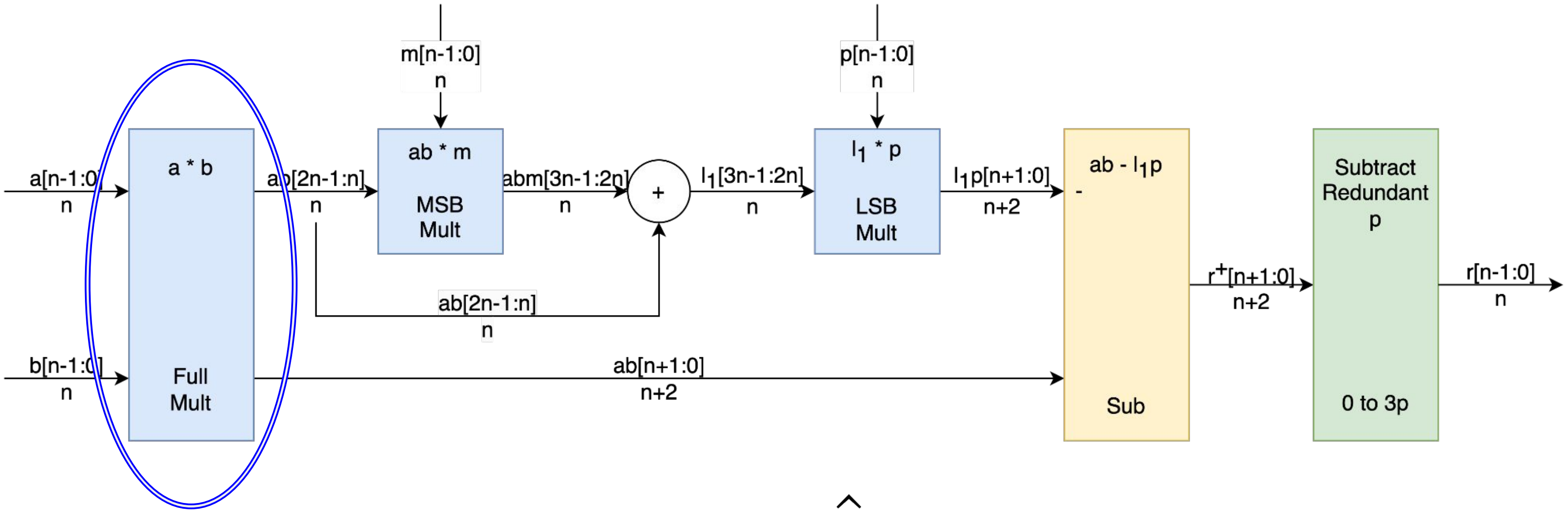
Redundancy

Remainder

$$r = ab - \hat{l}q - \lambda q$$

Quotient Approximation

Barrett Optimization 1 - Karatsuba



$$r = ab - \hat{l}q - \lambda q$$

Barrett Optimization 1 - Karatsuba

- Apply initially to ab full-multiplier
- General concept:

$$\begin{aligned} ab &= (a_h 2^{\frac{n}{2}} + a_l)(b_h 2^{\frac{n}{2}} + b_l) \\ &= a_h b_h 2^n + ((a_h + a_l)(b_h + b_l) - a_h b_h - a_l b_l) 2^{\frac{n}{2}} + a_l b_l \end{aligned}$$

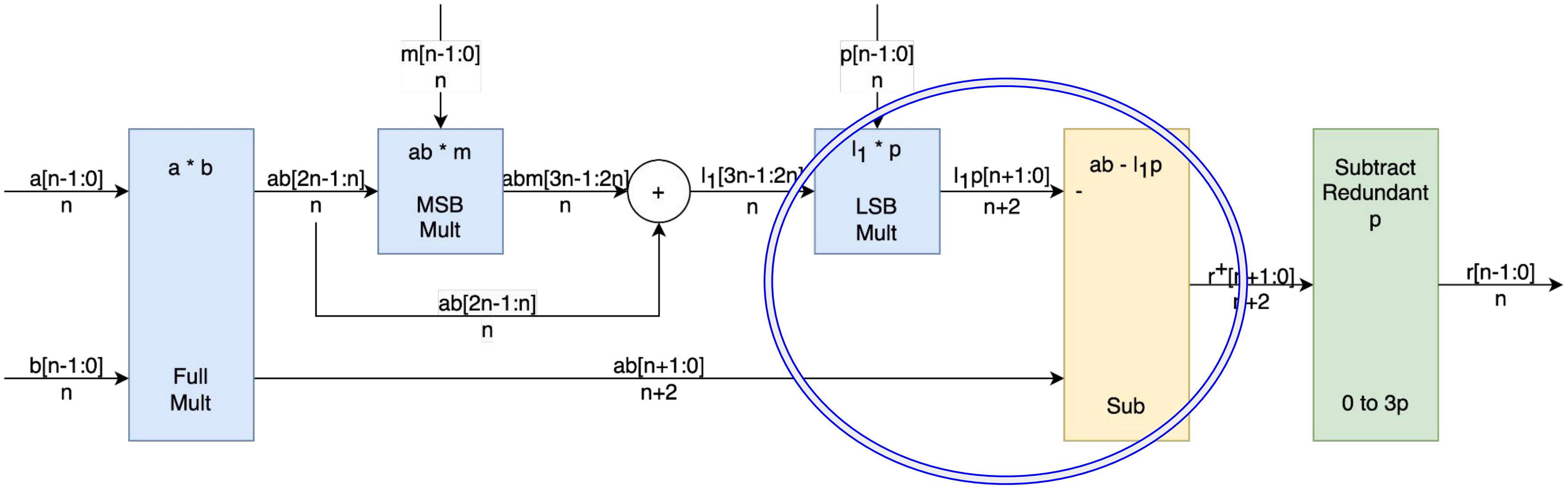
- Uses 3 instead of 4 half-length multipliers
- Can be applied recursively

“Karatsuba algorithm”, https://en.wikipedia.org/wiki/Karatsuba_algorithm

BLS12-381: Karatsuba Full Mult

Base Mult	Strategy	Calculation	Result
19Ux19U	Schoolbook	ceil(381/19)^2	441 slices
27Ux27U	Schoolbook	2 * ceil(381/27)^2	450 slices
24Ux24U	Karatsuba	u = ceil(381/24) = 16 2 * 3^log2(u)	**162** slices

Barrett Optimization 2 - LSB Mult



$$r = ab - \hat{l}q - \lambda q$$

Barrett Optimization 2 - LSB Mult

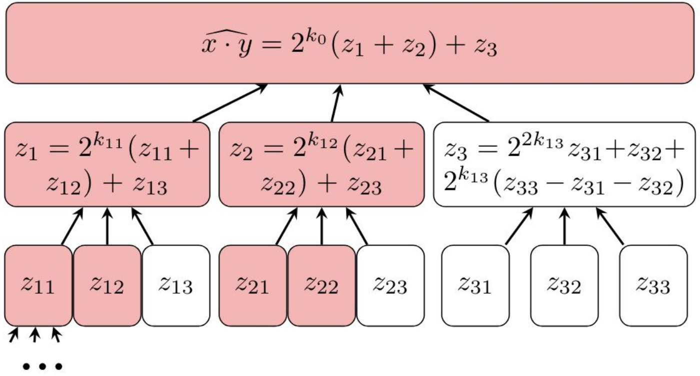
- *Calculate the remainder:* $r = ab \mod q = ab - lq$
- *By approximating* $l - \lambda \leq \hat{l} \leq l$ *we get:* $r + \lambda q = ab - \hat{l}q$
- *Using long subtraction for the case* $\lambda = 0$ *:*

$$\begin{array}{cccccc} & ab[2n-1] & \ldots & ab[n] & ab[n-1] & \ldots & ab[0] \\ - & \hat{l}q[2n-1] & \ldots & \hat{l}q[n] & \hat{l}q[n-1] & \ldots & \hat{l}q[0] \\ \hline & 0 & \ldots & 0 & r[n-1] & \ldots & r[0] \end{array}$$

- *<u>Conclusion</u>: Only need to calculate approximately* $n + log_2(\lambda)$ *LSBs of* $\hat{l}q$

Yuval Domb, "Fast Modular Multiplication", https://github.com/ingonyama-zk/papers/blob/main/modular_multiplication.pdf

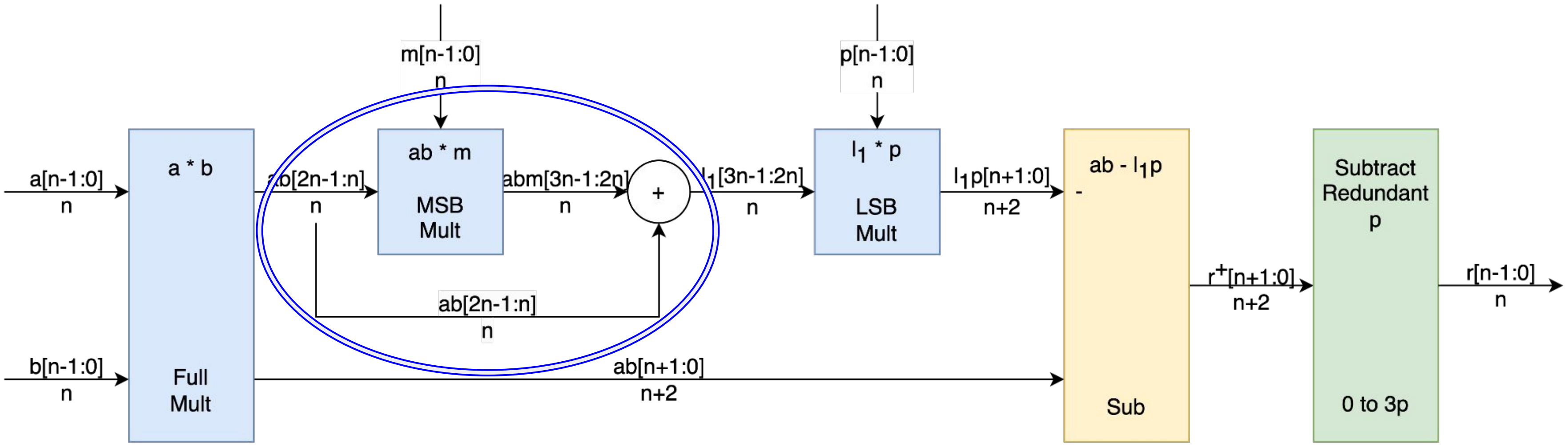
BLS12-381: Karatsuba LSB Mult with Pruning



- *Using 24Ux24U multipliers with recursive Karatsuba: <120 slices*

Charles. F. Xavier, "PipeMSM: Hardware Acceleration for Multi-Scalar Multiplication", https://eprint.iacr.org/2022/999.pdf

Barrett Optimization 3 - MSB Mult



$$r = ab - \hat{l}q - \lambda q$$

Barrett Optimization 3 - MSB Mult

$$\hat{l}_0 = \left\lfloor \frac{abm}{2^{2n}} \right\rfloor$$

$$e(\hat{l}_0) < 1$$

- Since $m \approx \frac{1}{q}$ cannot be represented with fixed precision:

$$\hat{l}_1 = \left\lfloor \left\lfloor \frac{ab}{2^n} \right\rfloor \cdot \frac{m}{2^n} \right\rfloor$$

$$e(\hat{l}_1) < 3$$

- By taking just the MSBs of the result:

Since:

$$\begin{aligned} \frac{abm}{2^{2n}} &= \frac{ab[2n-1:n] \cdot m}{2^n} + \frac{ab[n-1:0] \cdot m}{2^{2n}} \\ &< \frac{ab[2n-1:n] \cdot m}{2^n} + 2 \end{aligned}$$

Yuval Domb, “Fast Modular Multiplication”, https://github.com/ingonyama-zk/papers/blob/main/modular_multiplication.pdf

Barrett Optimization 3 - Karatsuba MSB Mult

- *By approximating the MSB multiplier using Karatsuba pruning*

$$e(\hat{l_1}) < 3 + \left\lceil \frac{\Delta^{\{k_{ij}\}}}{2^n} \right\rceil$$

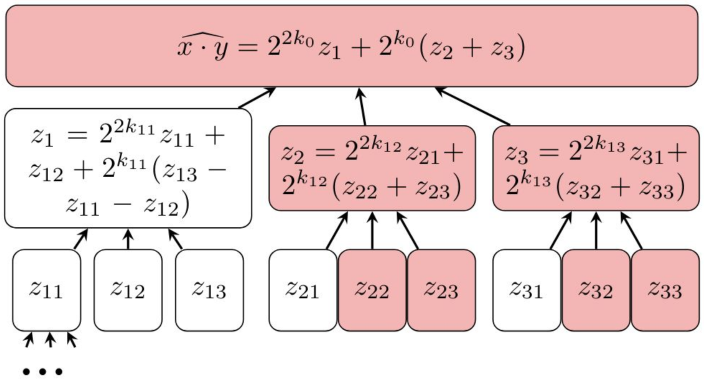
Since:

$$0 \leq \Delta(x \cdot y) < 2^{2k_0} + 2^{k_0}(\Delta(z_1) + \Delta(z_2)) <$$
$$2^{2k_0} + 2^{k_0}(2^{2k_{12}} + 2^{2k_{13}} + 2^{k_{12}}(\Delta(z_{22}) + \Delta(z_{23})) +$$
$$2^{k_{13}}(\Delta(z_{32}) + \Delta(z_{33}))) < \cdots = \Delta^{\{k_{ij}\}}$$

@ Dmytro (Dima) Tymokhanov

Charles. F. Xavier, "PipeMSM: Hardware Acceleration for Multi-Scalar Multiplication", https://eprint.iacr.org/2022/999.pdf

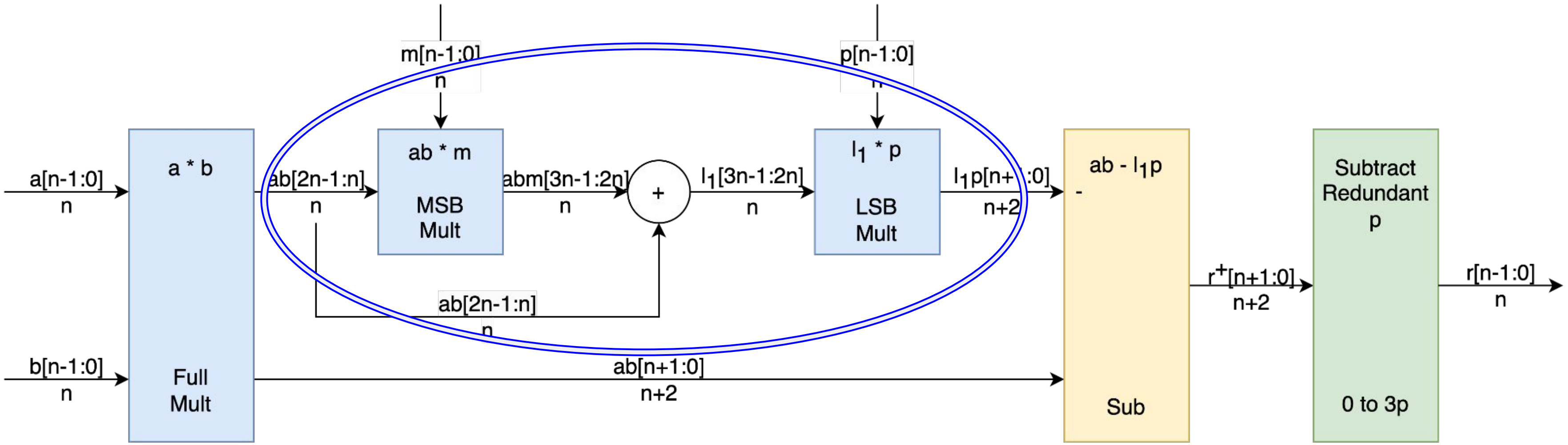
BLS12-381: Karatsuba MSB Mult with Pruning



- *Using 24Ux24U multipliers with recursive Karatsuba: <120 slices*

Charles. F. Xavier, "PipeMSM: Hardware Acceleration for Multi-Scalar Multiplication", https://eprint.iacr.org/2022/999.pdf

Barrett Optimization 4 - Constant Mult and CSD



$$r = ab - \hat{l}q - \lambda q$$

Barrett Optimization 4 - Constant Mult and CSD

- Redundant Signed Digit (RSD) representation uses {-1,0,1} “bits”
- Canonical Signed Digit (CSD)
    - RSD with minimal number of non-zero terms
    - Canonical form
- Example: 7 = [0,1,1,1]=[1,0,0,-1]
- A CSD form of a typical n-bits number has n/3 non-zeros

Charles. F. Xavier, “PipeMSM: Hardware Acceleration for Multi-Scalar Multiplication”, https://eprint.iacr.org/2022/999.pdf

Barrett Optimization 4 - Constant Mult and CSD

- A constant multiplier is equivalent to k additions where k is the number of non-zeros terms in the constant
- For CSD the average k is n/3 but sometimes it's much less, especially for small n
- Strategy:
  - Trade the "CSD-light-weight" Karatsuba leaf multipliers with additions
  - When constant part has many leading zeros use one-slice multipliers

Charles. F. Xavier, "PipeMSM: Hardware Acceleration for Multi-Scalar Multiplication", https://eprint.iacr.org/2022/999.pdf

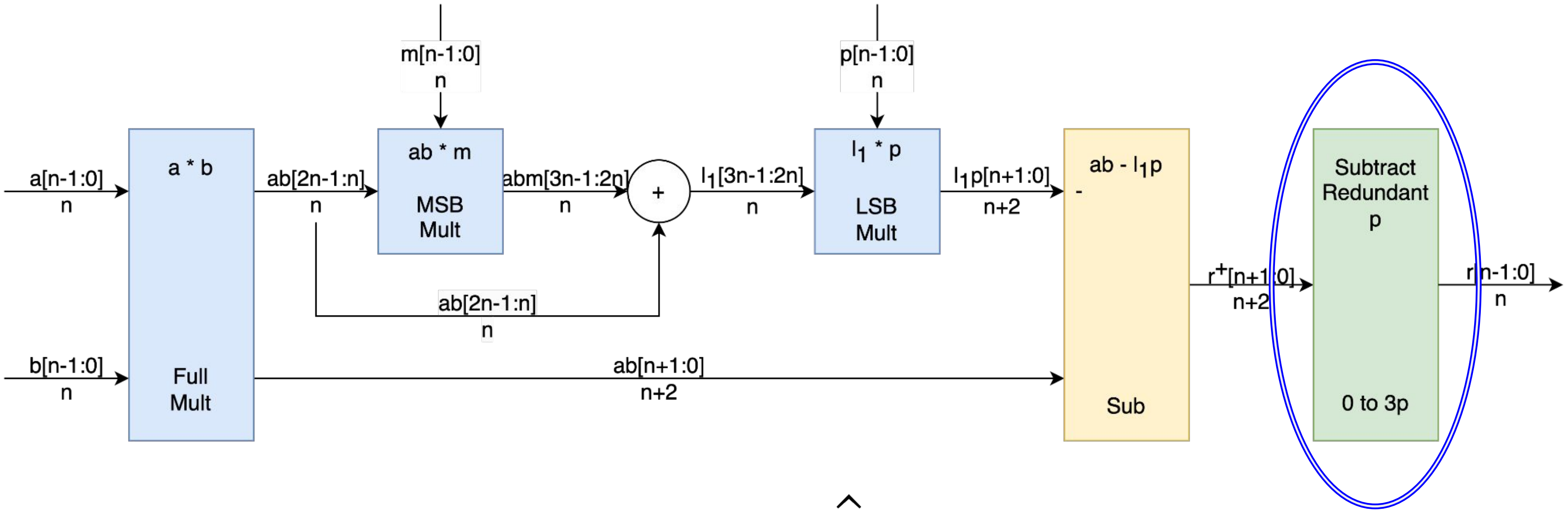
BLS12-381: Constant Mult and CSD

- Replace multipliers by additions when "CSD-weight" does no exceed 4
- Use one-slice multipliers when possible (sufficiently small constants)
- And extensive "under-the-hood" optimization...
- MSB constant mult: 62 slices
- LSB constant mult: 30 slices

@ Hadar Sackstein

Charles. F. Xavier, "PipeMSM: Hardware Acceleration for Multi-Scalar Multiplication", https://eprint.iacr.org/2022/999.pdf

Barrett Optimization 5 - Conditional Subtraction



$$r = ab - \hat{l}q - \lambda q$$

@ Tony Wu

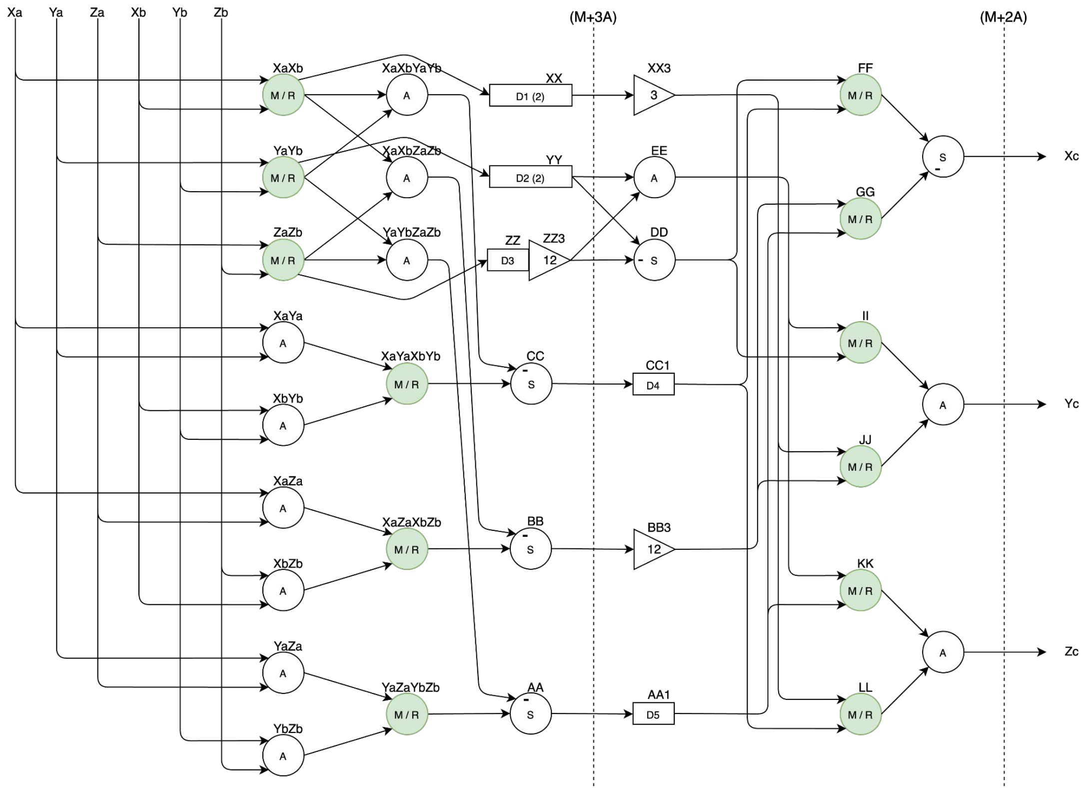
Barrett Optimization 5 - Conditional Subtraction

- *Inspired by Niall Emmart's work on MSM (see slide 26)*
- *Observation: The conditional subtraction of redundant copies of* q *depends only on the inaccuracy of the quotient estimate* $\hat{l}$ *and is bounded, regardless of the size of the product* ab *(Reduction accuracy is independent of full product size).*
- *Conclusion:*
  - *Plan the reduction for a larger product, say 384x384.*
  - *Design the reduction such that* $r + \lambda q$ *does not exceed 384 bits.*

Niall Emmart, "A deep dive into optimizing Multi-Scalar Multiplication", https://cbr.stanford.edu/seminarTalks/slides_20230526_niall_emmart.pptx

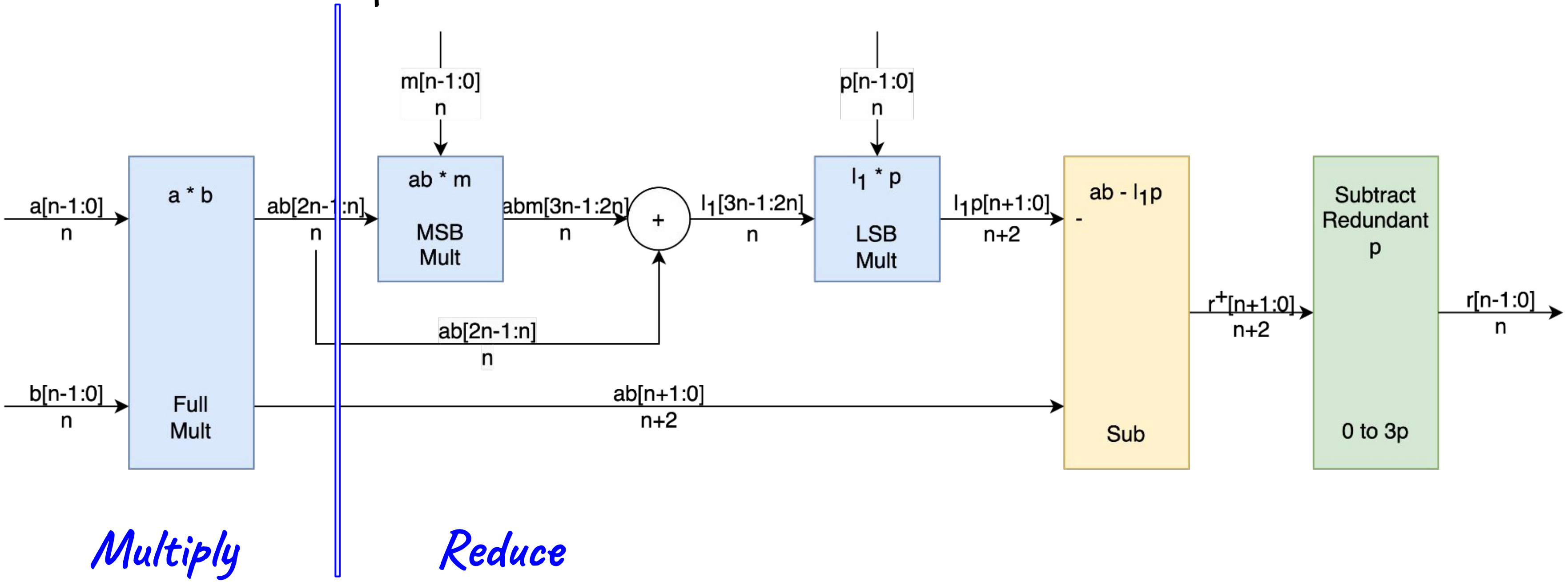
Elliptic Curve	
EC Addition Formulas	5292
Modular Multiplier Optimization	**3048**
ECADDER Optimization	
Toom-Cook Optimization	

Projective ECADDER for BLS12-381 (a=0, b=4)

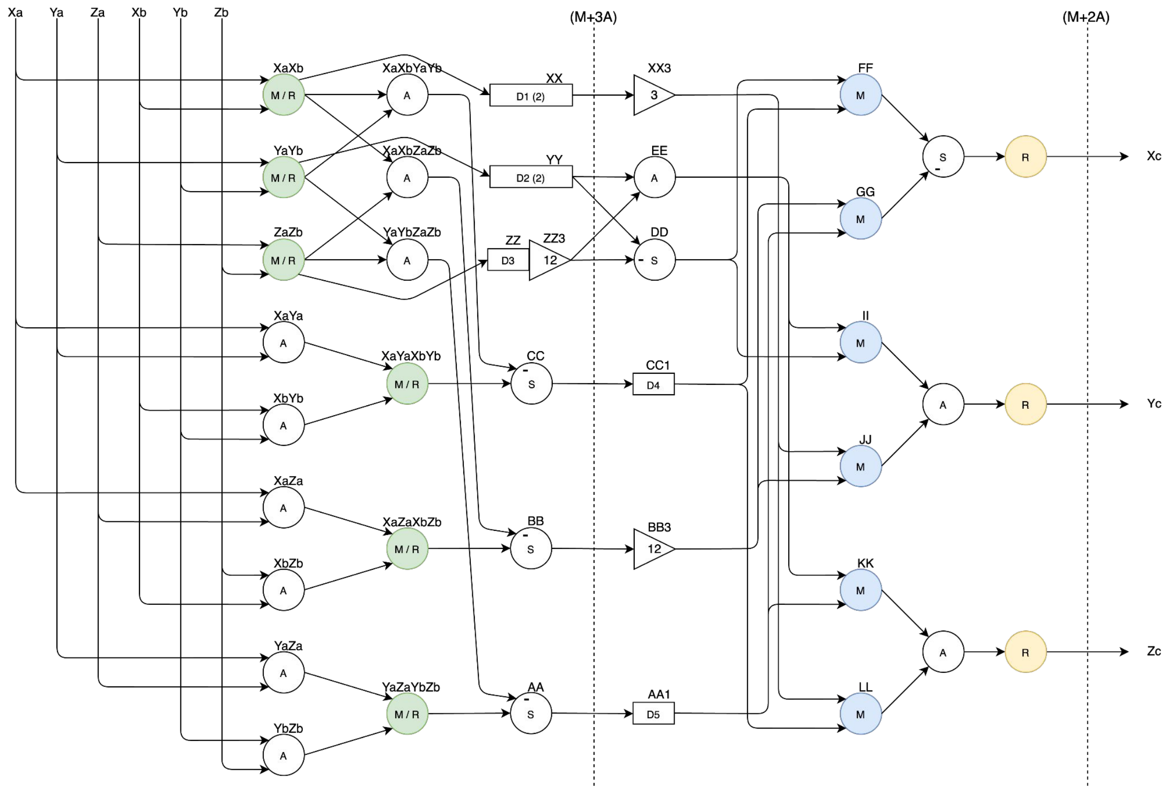


- 12 Multiply (162)
- 12 Reduce (92)
- **3048 slices**

Barrett Multiplier



Projective ECADDER - Multiply/Reduce Separation



- Separate Multiply and Reduce
- Reduce input increased by 1 bit
- 12 Multiply (162)
- 9 Reduce (92)
- **2772 slices**

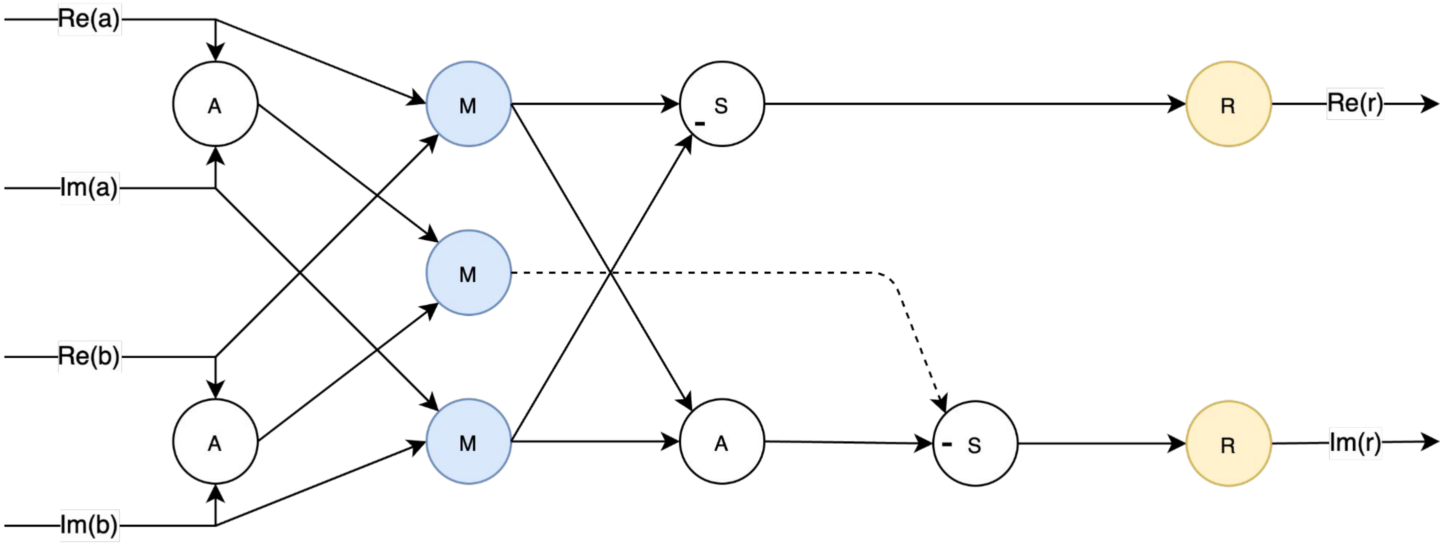
Sidenote: G2 Field Multiplication

- *Reminder:* $G_2 \subset E'(\mathbb{F}_{p^2}), \quad E' : y^2 = x^3 + 4(1+i)$
- *General concept (Karatsuba):*

$$\begin{aligned} ab &= (a_r + ia_i)(b_r + ib_i) \\ &= a_r b_r - a_i b_i + i((a_r + a_i)(b_r + b_i) - a_r b_r - a_i b_i) \end{aligned}$$

- *Multiply: 3 base-field multipliers*
- *Reduce: 2 base-field reductions*

Sidenote: G2 Field Multiplication



Elliptic Curve

EC Addition Formulas	5292
Modular Multiplier Optimization	3048
ECADDER Optimization	**2772**
Toom-Cook Optimization	

Toom-Cook Multiplication

- A generalization of Karatsuba
- Represent multiplicands as polynomials sampled at 2^k where $n = mk$

$$a = a_{m-1}2^{(m-1)k} + ... + a_1 2^k + a_0$$
$$a(x) = a_{m-1}x^{m-1} + ... + a_1 x + a_0$$
$$\Rightarrow a = a(x = 2^k)$$

- Calculate polynomial product in a suitable evaluations domain
- Transform product-polynomial back to coefficients domain and sample at 2^k

“Toom-cook multiplication”, https://en.wikipedia.org/wiki/Toom-Cook_multiplication

Toom-Cook Multiplication (e.g. n=3k)

- *Represent as polynomial product:*

$$p(x) = (a_2x^2 + a_1x + a_0)(b_2x^2 + b_1x + b_0)$$

- *Evaluate over domain* $\{0, 1, -1, -2, \infty\}$ *of length 5:*

$$\begin{bmatrix} p(0) \\ p(1) \\ p(-1) \\ p(-2) \\ p(\infty) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & -2 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \odot \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & -2 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

Element-wise Multiplication

“Toom-cook multiplication”, https://en.wikipedia.org/wiki/Toom-Cook_multiplication

Toom-Cook Multiplication (e.g. n=3k)

- *The product polynomial follows:*

$$\begin{bmatrix} p(0) \\ p(1) \\ p(-1) \\ p(-2) \\ p(\infty) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & -2 & 4 & -8 & 16 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}$$

- *Inverting the above results in:*

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 3/2 & 1 & -3 & 1/2 & -6 \\ -3 & 3/2 & 3/2 & 0 & -3 \\ -3/2 & 1/2 & 3/2 & -1/2 & 6 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} p(0) \\ p(1) \\ p(-1) \\ p(-2) \\ p(\infty) \end{bmatrix}$$

"Toom-cook multiplication", https://en.wikipedia.org/wiki/Toom-Cook_multiplication

Toom-Cook Multiplication Implications

- Pro: Number of multiplications equals order of resulting product-polynomial
- Pro: Like Karatsuba, can be applied recursively
- Pro: Extendable to asymmetric multi-precision schemes
- Pro: Extendable to products of more than 2 arguments
- Con: Involves many small constant multiplications
- Con: Requires division ☹

BLS12-381: Toom Cook

Mult	381/Width	Full Mult	MSB+LSB	Total
19Ux19U	20.1 < 24 = 2*3*4	3*5*7 = 105	60	1800
24Ux24U	15.9 < 16 = 2^4	2(3^4) = 162	92	2772
27Ux27U	14.1 < 15 = 3*5	2(5*9) = 90	52	**1548**

Elliptic Curve

EC Addition Formulas	5292
Modular Multiplier Optimization	3048
ECADDER Optimization	2772
Toom-Cook Optimization	**1548**

Ingonyama's Aleo IP uses TC ECADDER

	Aleo IP	RTX 4090	RTX 3090	RTX 3080
Proofs / joule	2000	35.6	21.9	18.8
Proofs per sec / mm2	2000	26.3	11.1	9.6
Proofs / second	6000 x N	16000	7000	6000
Power (W)	3 x N	450	320	320

"Product Announcement: Aleo IP Core", https://www.ingonyama.com/blog/product-announcement-aleo-ip-core

Thank You

yuval@ingonyama.com