KU LEUVEN

BRUGES 9-12 SEPTEMBER ESSERC 2024

micas KU LEUVEN

# A 3.25 GHz Large-Integer Extended GCD Accelerator in 12 nm

Kavya Sreedhar[1], Gedeon Nyengele[1], Mark Horowitz[1], Christopher Torng[2]

[1]Stanford University, USA
[2]University of Southern California, USA

skavya@stanford.edu

IEEE SSCS Electron Devices Society CAS

# Cryptography relies on XGCD

BRUGES
9-12 SEPTEMBER
ESSERC 2024

- Many cryptography algorithms rely on fast extended GCD (XGCD)
  - Recent applications are dominated by large-integer XGCD
- XGCD computes Bézout coefficients $\boldsymbol{b_a}, \boldsymbol{b_b}$ satisfying Bézout's Identity

$$\boldsymbol{b_a}, \boldsymbol{b_b} : \boldsymbol{b_a} * a_0 + \boldsymbol{b_b} * b_0 = \gcd(a_0, b_0)$$

- We present the first chip for XGCD
  - Building from our carry-save-adder subtraction-based algorithm (CHES'22)

# Reducing inputs to find the GCD



- Algorithms use GCD-preserving transformations to find the GCD
- Iterations end when one of $a_i, b_i$ is zero and the other is the GCD
- Every cycle, either $a_i$ or $b_i$ is updated
  - When $a_i$ or $b_i$ is even, shift by one, two, or three
  - When $a_i$ and $b_i$ are odd, compute $(a_i + b_i)/4$ or $(a_i - b_i)/4$

# Reducing inputs to find the XGCD

XGCD: $b_a * a_0 + b_b * b_0 = \gcd(a_0, b_0)$

- Need to maintain two equations since either $a_i$ or $b_i$ will become zero
  - $u_i * a_0 + m_i * b_0 = a_i$
  - $y_i * a_0 + n_i * b_0 = b_i$
- Every cycle, either $a_i, u_i, m_i$ or $b_i, y_i, n_i$ is updated
- $u_i, m_i, y_i, n_i$ are updated each cycle in a way that ensures these equations hold
  - Divisibility of these variables may not match the divisibility of $a_i, b_i$

# Returning XGCD at the end

ESSERC 2024 BRUGES 9-12 SEPTEMBER

XGCD: $b_a * a_0 + b_b * b_0 = \mathrm{gcd}(a_0, b_0)$

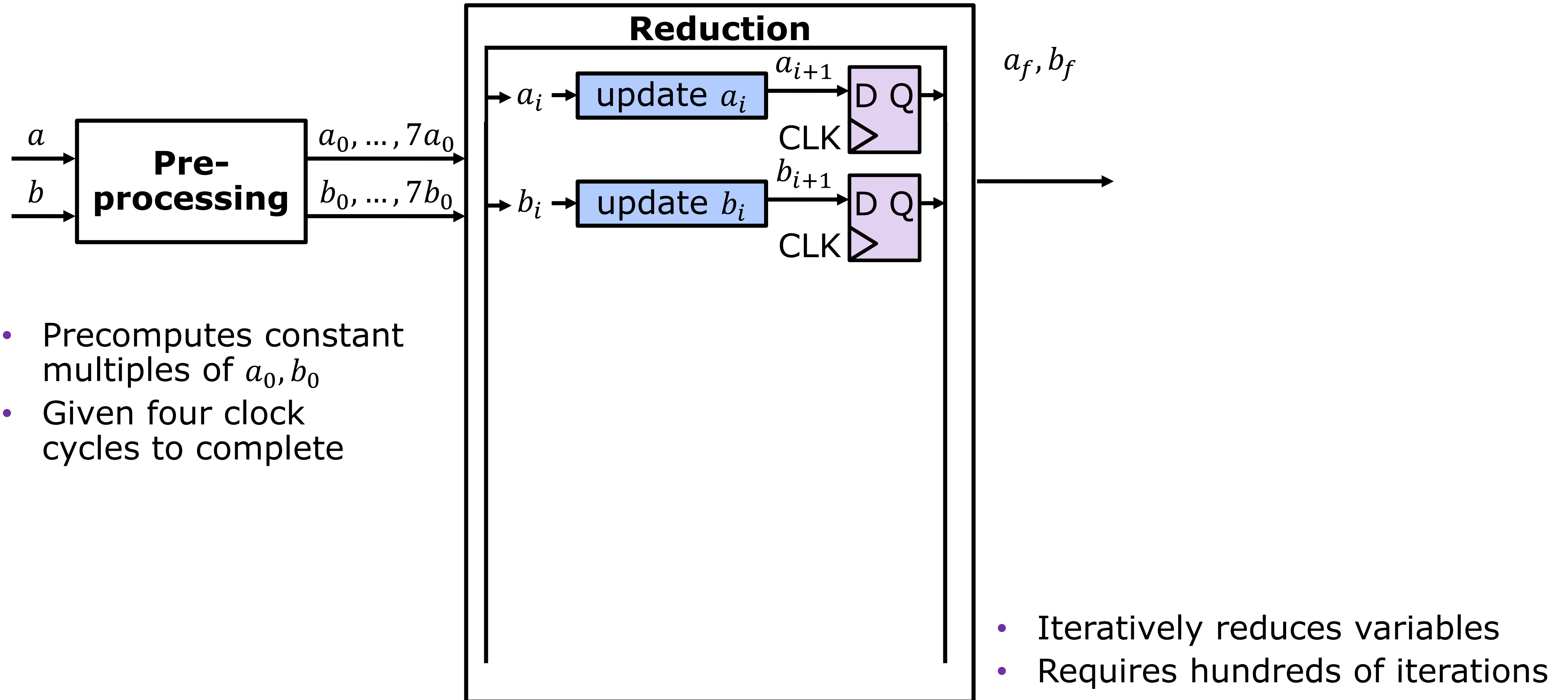
- At the end: one of $a_i, b_i$ is zero and the other is the GCD
- GCD: $\mathrm{gcd}(a_0, b_0) = a_i + b_i$
- Bézout coefficients: $b_a = u_f + y_f$; $b_b = m_f + n_f$

# Dataflow overview

- Precomputes constant multiples of $a_0, b_0$
- Given four clock cycles to complete

# Dataflow overview

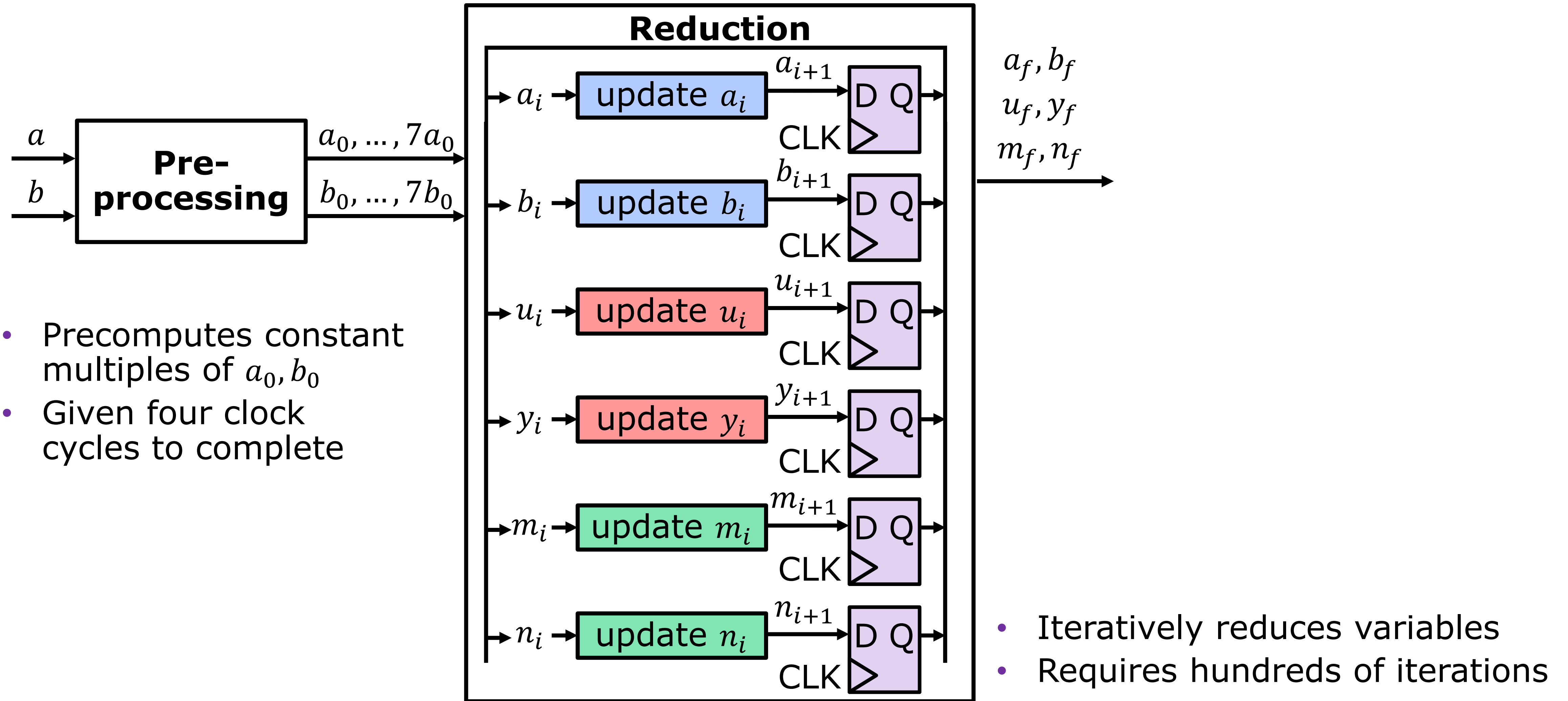
BRUGES 9-12 SEPTEMBER ESSERC 2024




- Precomputes constant multiples of $a_0, b_0$
- Given four clock cycles to complete

- Iteratively reduces variables
- Requires hundreds of iterations

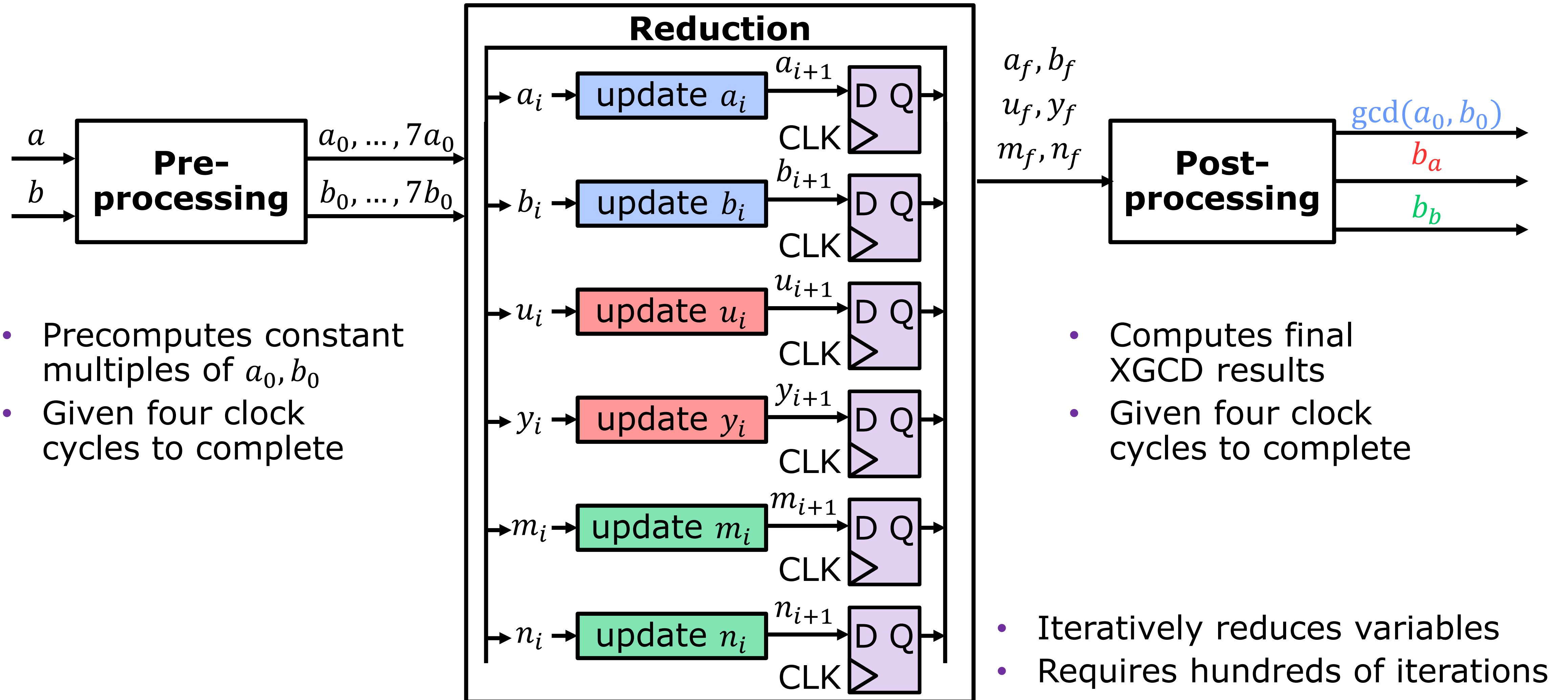
# Dataflow overview




- Precomputes constant multiples of $a_0, b_0$
- Given four clock cycles to complete

- Iteratively reduces variables
- Requires hundreds of iterations

# Dataflow overview




- Precomputes constant multiples of $a_0, b_0$
- Given four clock cycles to complete

- Computes final XGCD results
- Given four clock cycles to complete

- Iteratively reduces variables
- Requires hundreds of iterations

# Critical path



- There are 23 possible updates for each of the 4 Bézout variables
- Most complicated Bézout update to maintain Bézout equations

$$(u_i \pm y_i \pm k b_0) \gg 2$$

# Critical path

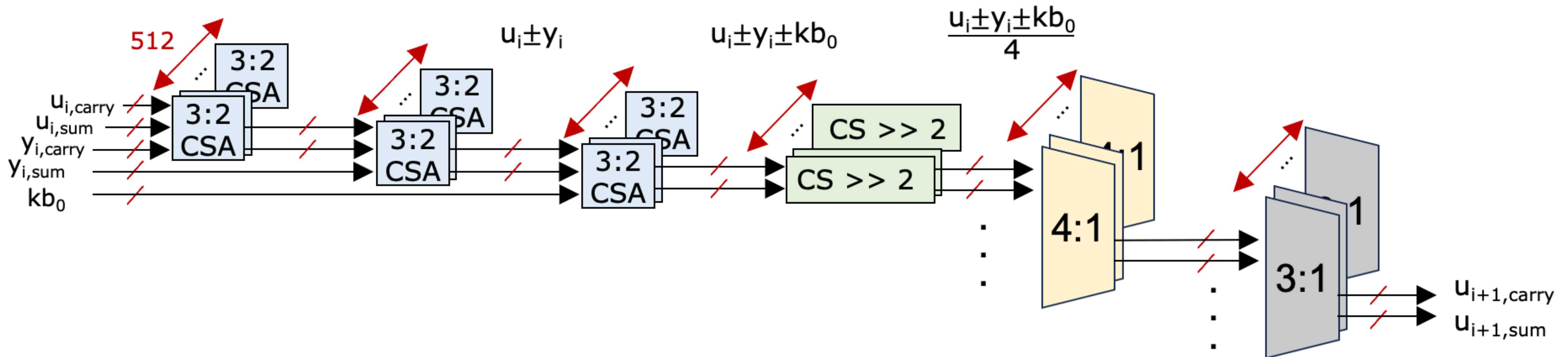
- There are 23 possible updates for each of the 4 Bézout variables
- Most complicated Bézout update to maintain Bézout equations

$$(u_i \pm y_i \pm kb_0) \gg 2$$

CS form    constant

- To accelerate this algorithm, we
  - Compute intermediates in carry-save (CS) form: no carry-propagation
  - Use late selects and precompute control signals to hide control delay

# Critical path



# Critical path challenges



1. Cannot quickly compare large-integer values in CS form
2. Driving control signals is expensive
3. Shifting in CS form requires care

# Critical path challenges

1. **Cannot quickly compare large-integer values in CS form**
2. Driving control signals is expensive
3. Shifting in CS form requires care

# How to know if $a_i > b_i$

- When $a_i, b_i$ are odd, want to compare $a_i > b_i$

# How to know if $a_i > b_i$

- When $a_i, b_i$ are odd, want to compare $a_i > b_i$
- We track the approximate size of $a_i, b_i$
- Compute $\delta \approx \log_2 a_i - \log_2 b_i$, building from prior work
- $sign(\delta)$ approximates $sign(a_i - b_i)$

# Recall $a_i, b_i$ updates

- When $a_i$ or $b_i$ is even, divide by a power of two up until eight
  - $\delta$ is correctly updated by the number of bits reduced

# Recall $a_i, b_i$ updates

- When $a_i$ or $b_i$ is even, divide by a power of two up until eight
  - $\delta$ is correctly updated by the number of bits reduced
- When $a_i$ and $b_i$ are odd, compute $(a_i + b_i)/4$ or $(a_i - b_i)/4$
  - $\delta$ is conservatively updated by one
  - However, there can be more cancellation: $> 1$ bit could have been reduced
- As a result, $sign(\delta) \neq sign(a_i - b_i)$ 25% of the time

# Consequences of using $\delta$

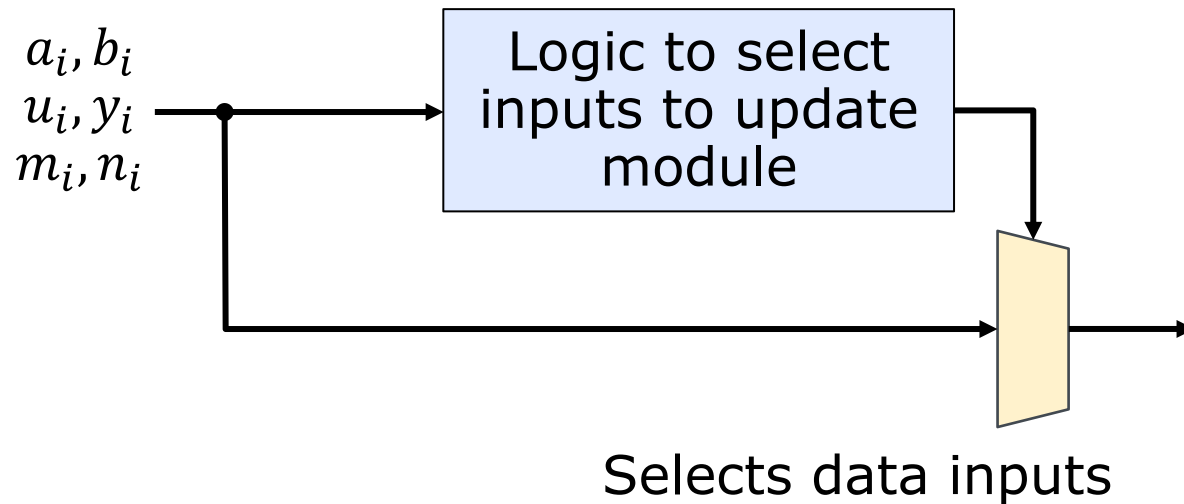
- Wrong variable can be updated, but algorithm still works
- Average cycle count increases by $15\%$
- Critical path delay is $4X$ shorter compared to using a slow full comparison
- For 512-bit inputs, updating $\delta$ requires a 10-bit carry-propagation adder

# Critical path challenges

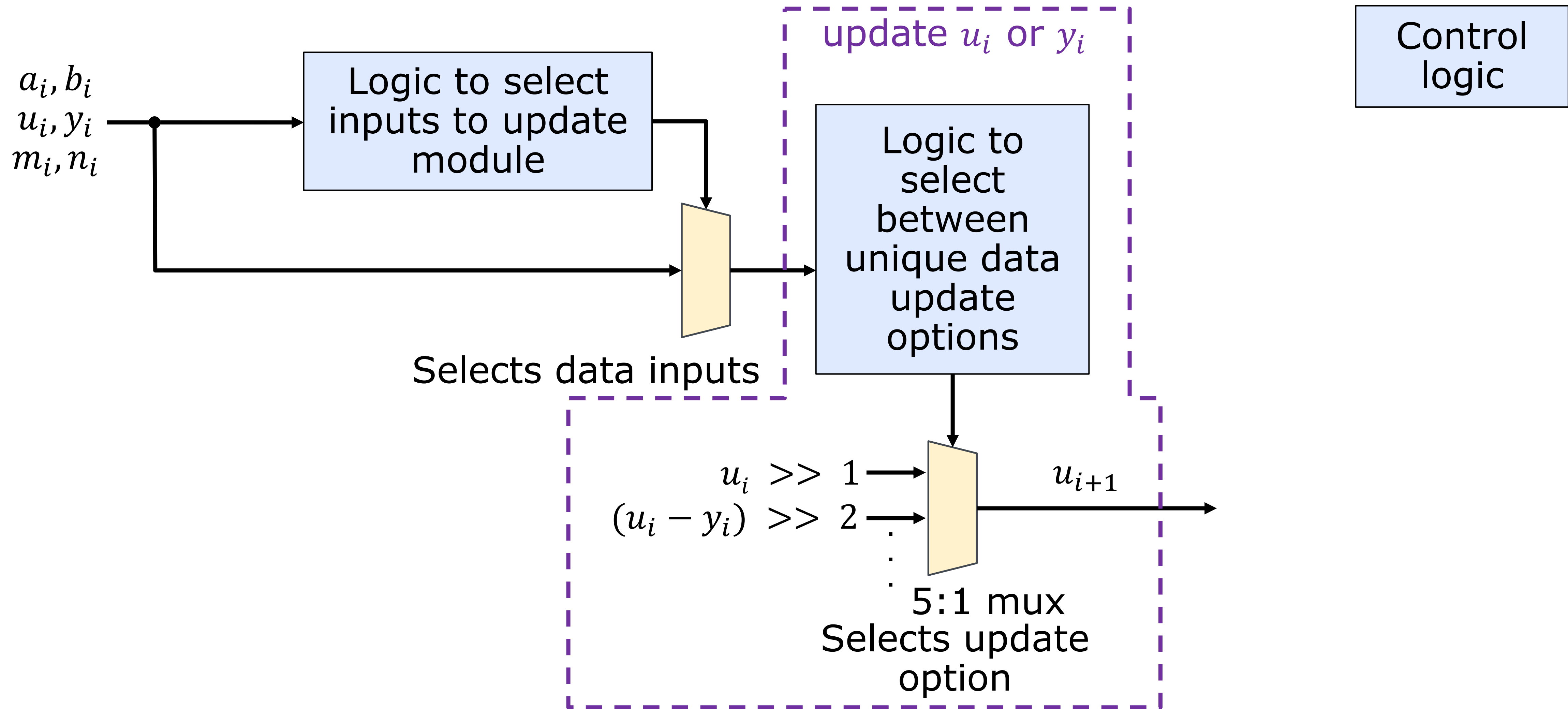
1. Cannot quickly compare large-integer values in CS form
2. **Driving control signals is expensive**
3. Shifting in CS form requires care

# Baseline with CSAs and $\delta$

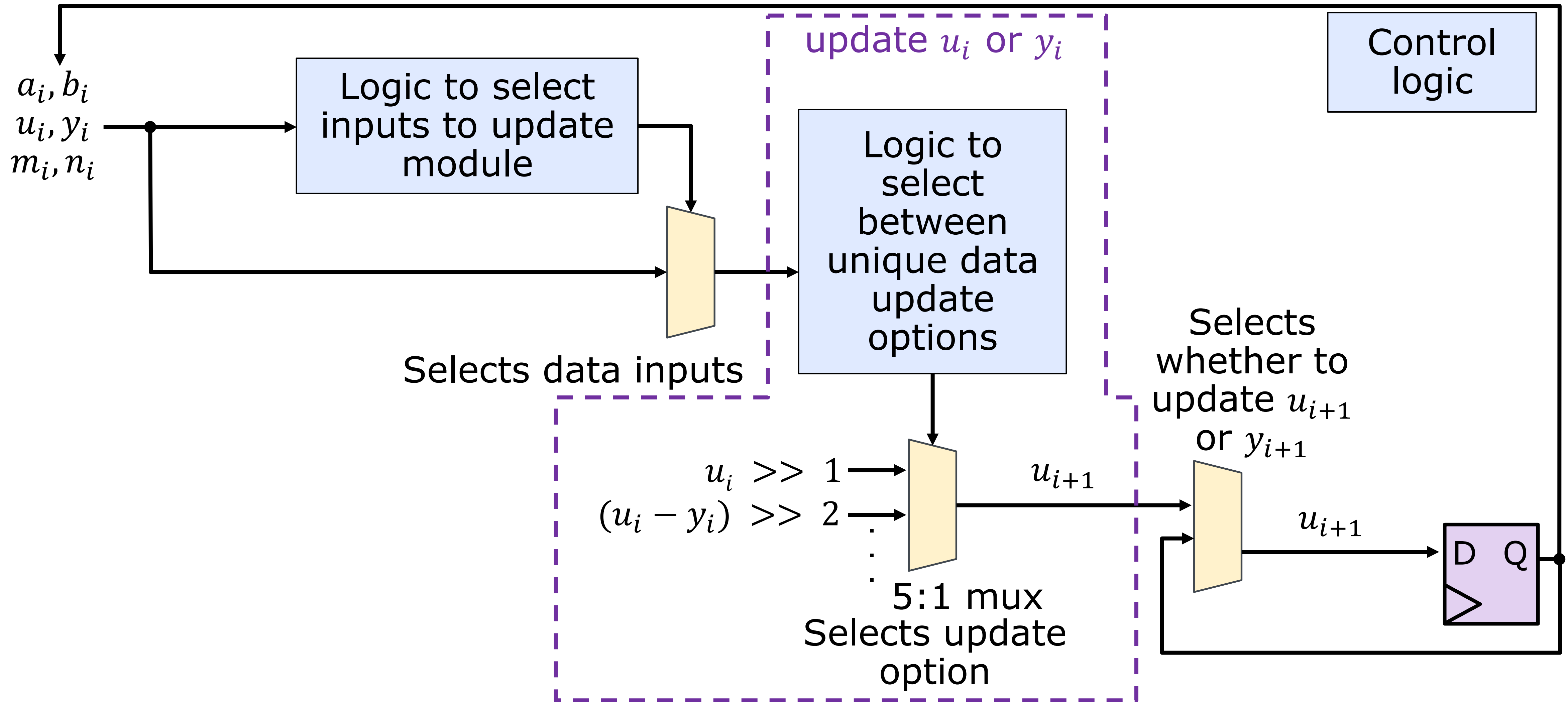
Control logic



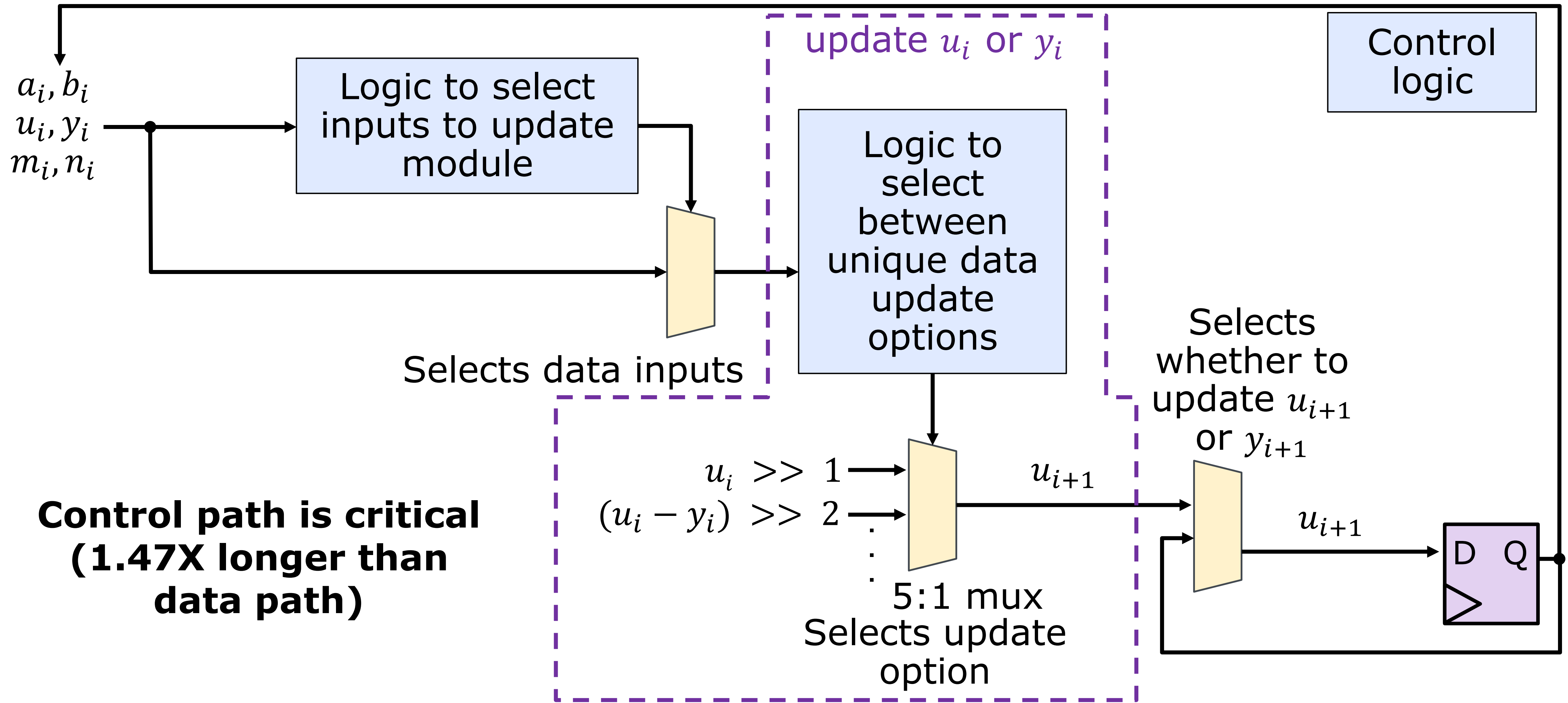
# Baseline with CSAs and $\delta$



# Baseline with CSAs and $\delta$



# Baseline with CSAs and $\delta$



# Using late selects

$a_i, b_i$
$u_i, y_i$
$m_i, n_i$

Logic to select inputs to update module

Selects data inputs

~~update $u_i$ or $y_i$~~
update $u_i$

Logic to select between unique data update options

Control logic

$u_i >> 1$
$(u_i - y_i) >> 2$

$u_{i+1}$

5:1 mux
Selects update option

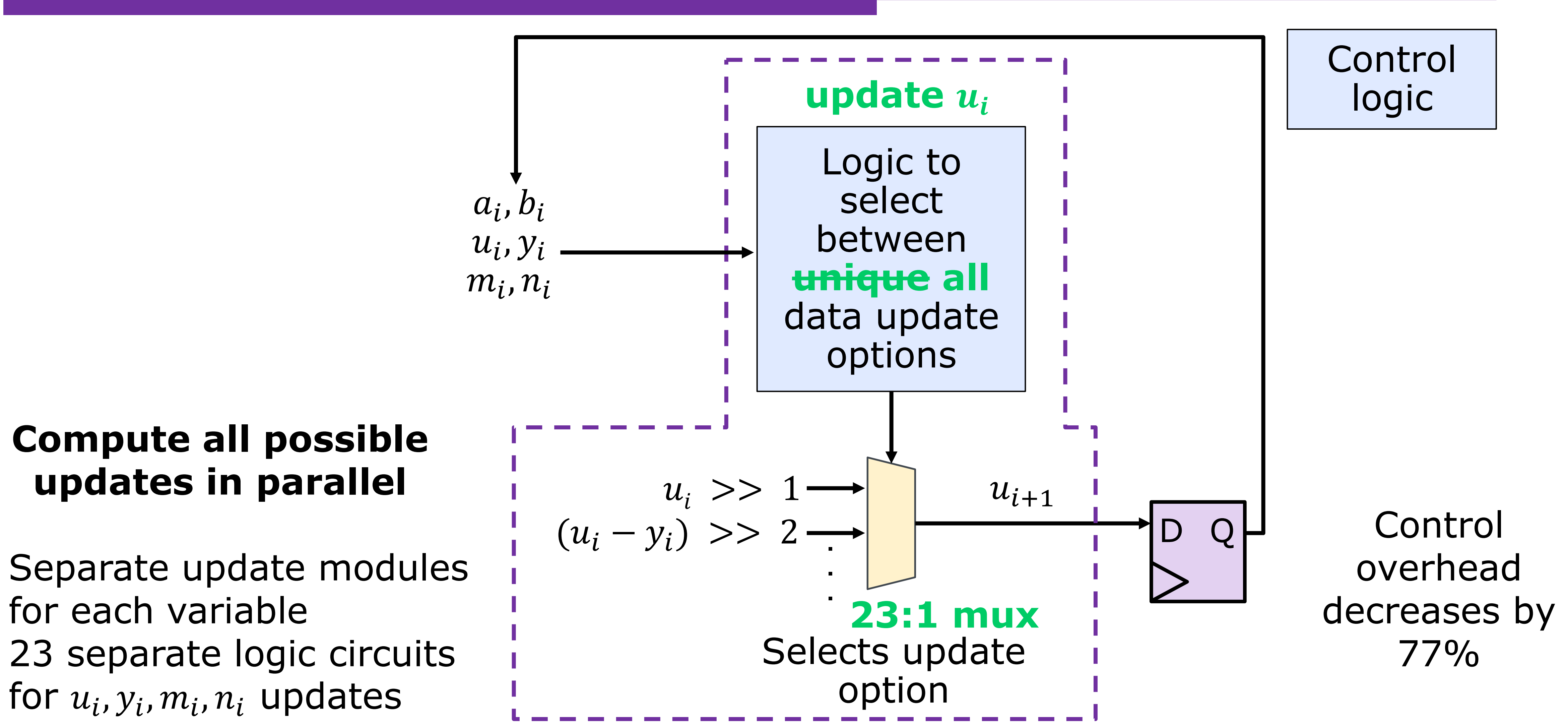
Selects whether to update $u_{i+1}$ or $y_{i+1}$

D Q

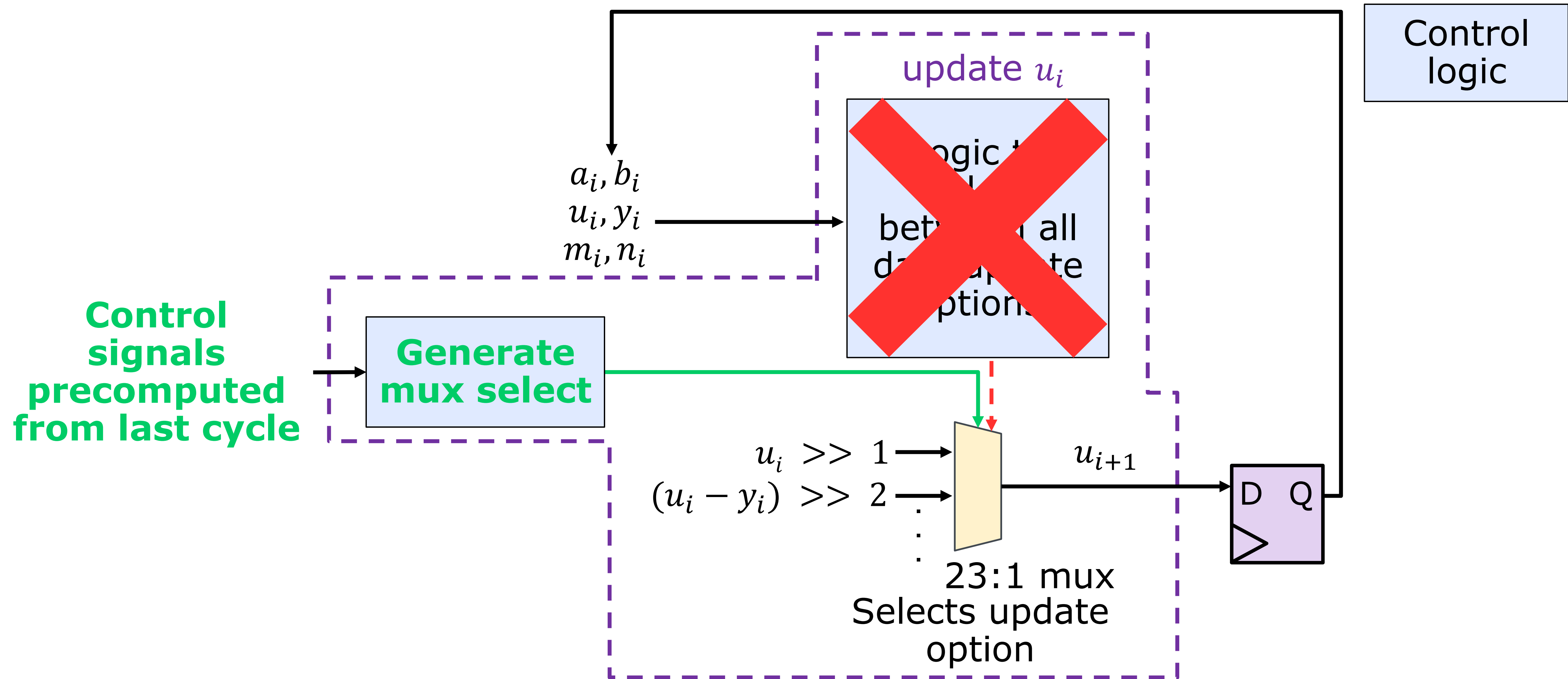
**Compute all possible updates in parallel**

1. Separate update modules for each variable

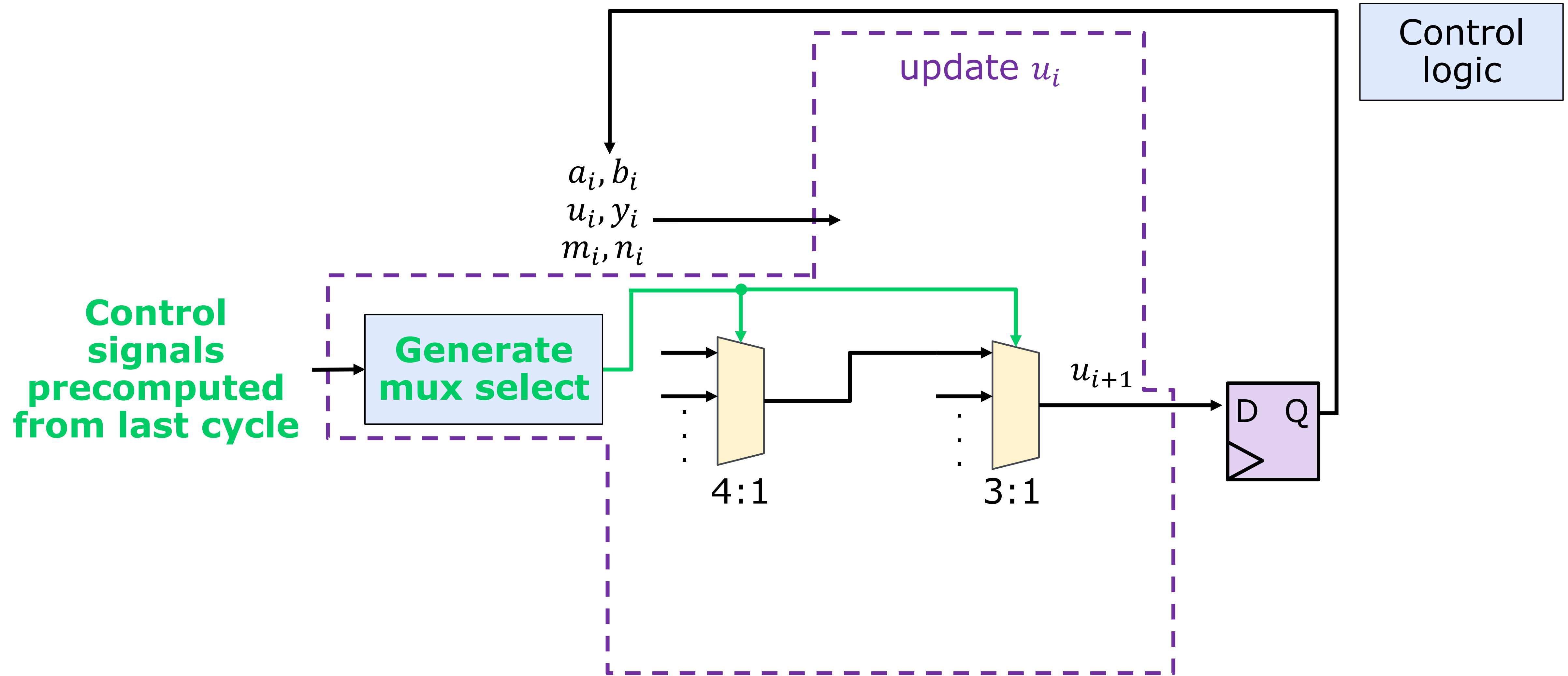
# Using late selects



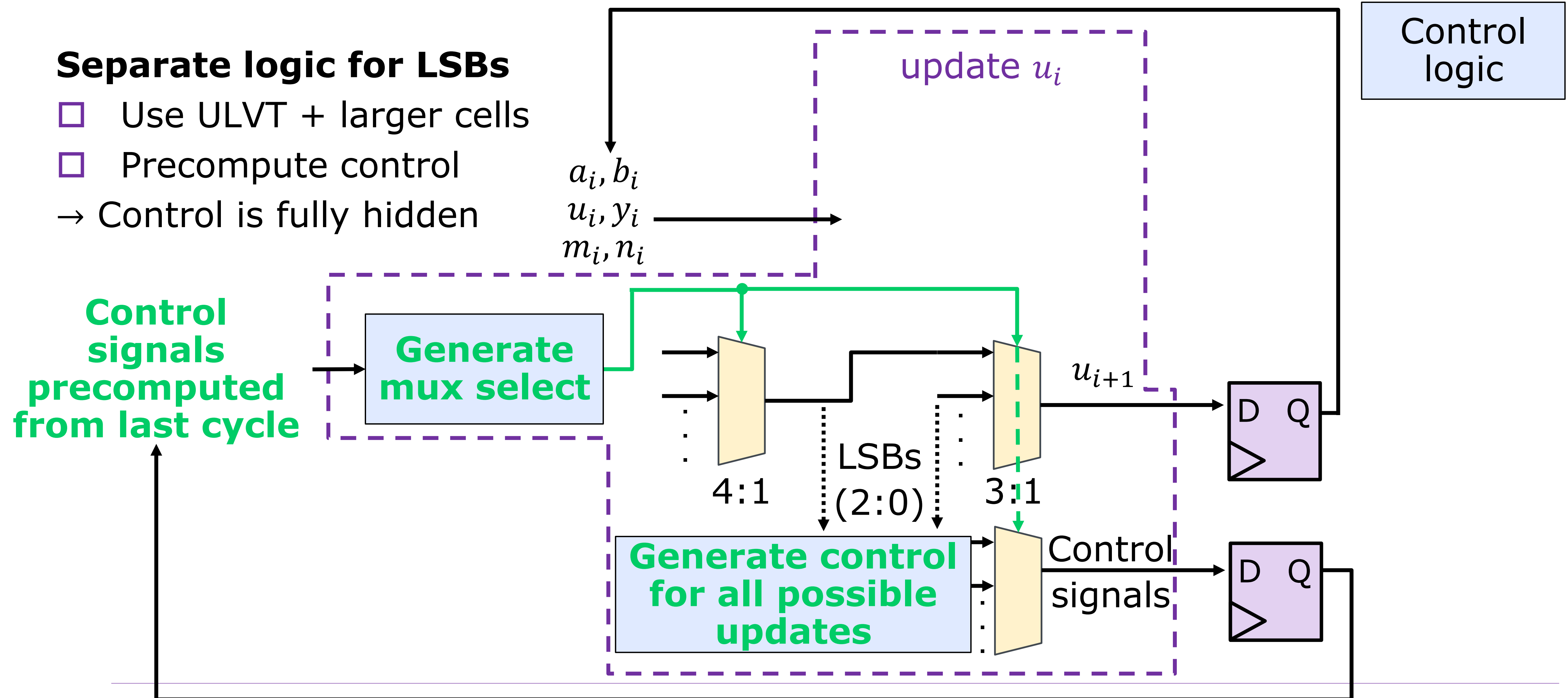
# Precomputing control



# Precomputing control



# Precomputing control

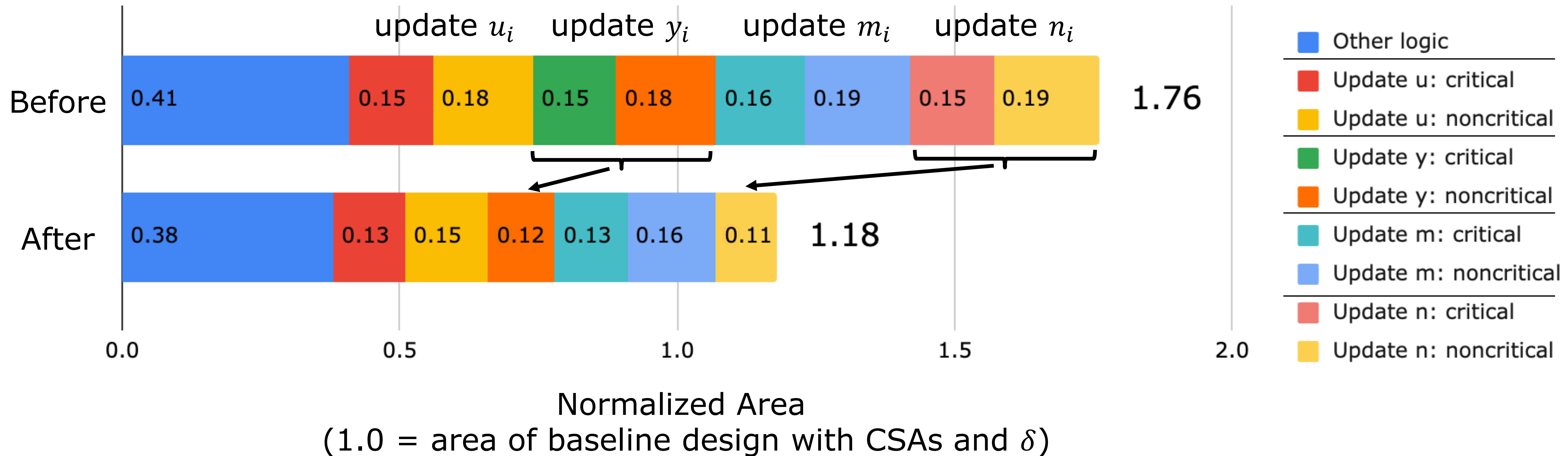


# Minimizing area overhead

- We observe that in the XGCD algorithm
  - Different variables must have the same divisibility
  - Some sum variable updates are the same
  - Some difference variable updates only differ by a sign

# Minimizing area overhead

BRUGES 9-12 SEPTEMBER ESSERC 2024



# Critical path challenges

1. Cannot quickly compare large-integer values in CS form
2. Driving control signals is expensive
3. **Shifting in CS form requires care**

# Shifting in CS form

BRUGES
9-12 SEPTEMBER
ESSERC 2024

- The problem
  - Our hardware has negative numbers → must preserve sign when shifting
  - Shifted result can be off by one when both *carry* and *sum* are odd

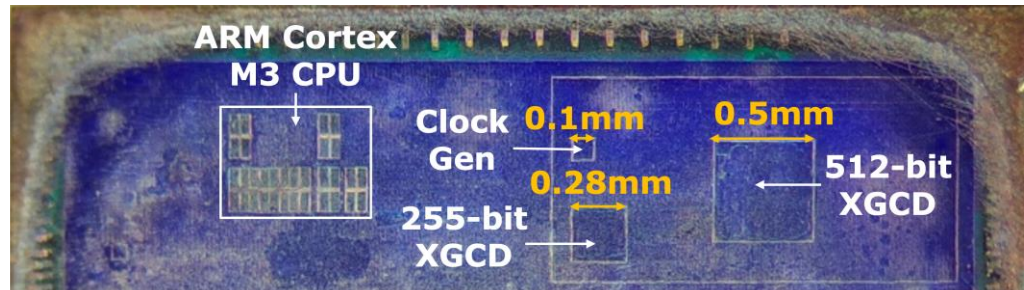
# Shifting in CS form

BRUGES
9-12 SEPTEMBER
ESSERC 2024

- The problem
  - Our hardware has negative numbers → must preserve sign when shifting
  - Shifted result can be off by one when both *carry* and *sum* are odd
- The solution
  - Build from prior work to specialize logic to minimize delay of signed logic shifting
  - Use a half adder to insert shifted out carry
    - Make detecting when to apply this correction cheap

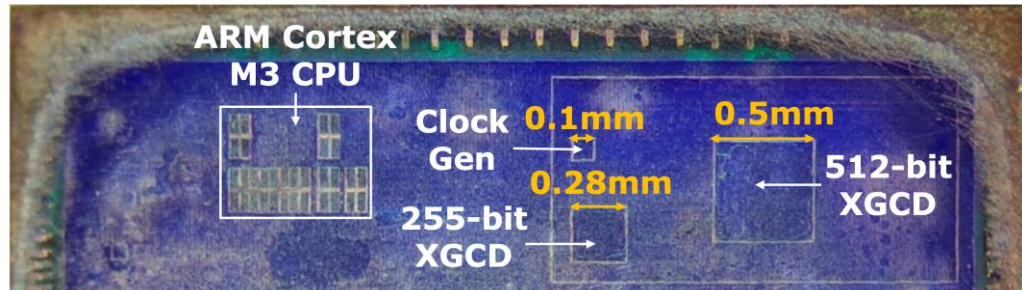
# Die photo and chip specs





# Die photo and chip specs



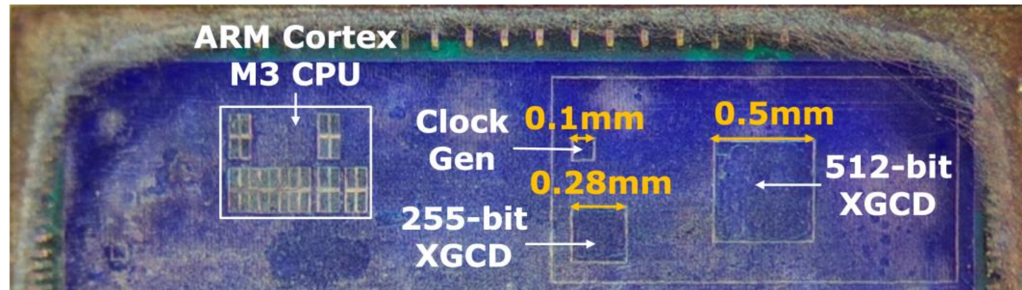



Execution modes:

- Fast
- Constant time
- Debugging

# Die photo and chip specs





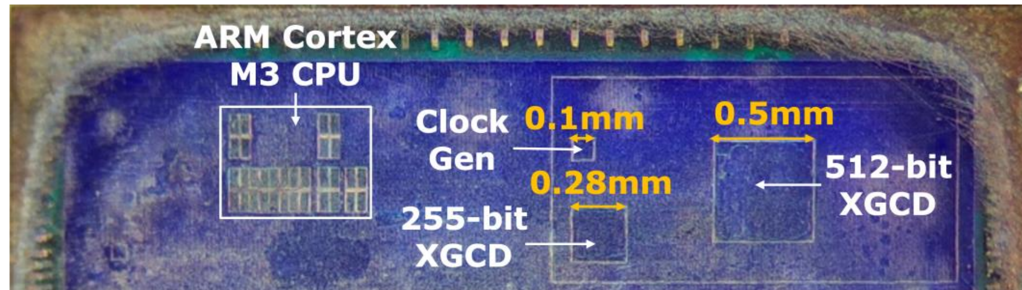

Execution modes:

- Fast
- Constant time
- Debugging

| | 512-bit XGCD | 255-bit XGCD |
|---|---|---|
| Technology | GF 12nm FinFET | |
| Voltage | 0.9V | |
| Input Bitwidth | 512 | 255 |
| **Max Clock Frequency (GHz)** | **3.25** | **3.25** |

**Design performance is independent of bitwidth with CSAs**

# Die photo and chip specs



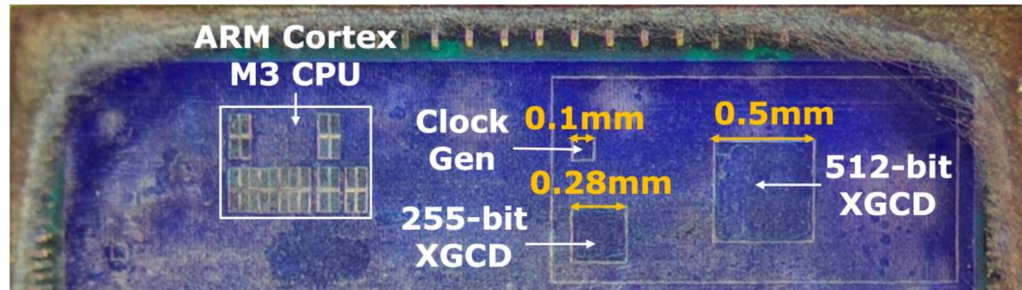

Execution modes:

- Fast
- Constant time
- Debugging

| | 512-bit XGCD | 255-bit XGCD |
|---|---|---|
| Technology | GF 12nm FinFET | |
| Voltage | 0.9V | |
| Input Bitwidth | 512 | 255 |
| Max Clock Frequency (GHz) | 3.25 | 3.25 |
| **Total Area ($mm^2$)** | **0.25** | **0.076** |

**The 255-bit XGCD unit was optimized for constant-time applications**

# Die photo and chip specs

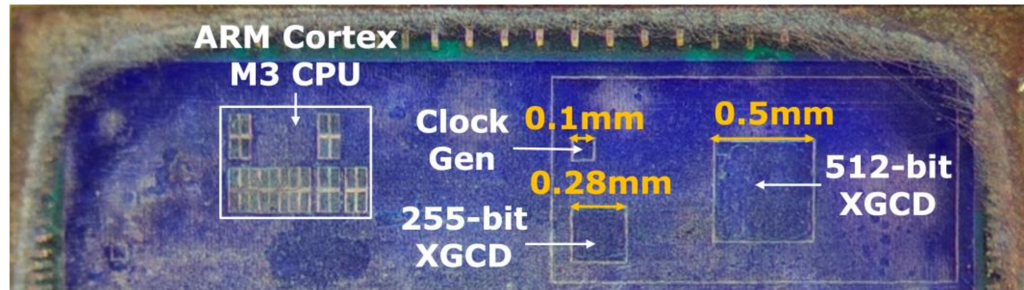



Execution modes:

- ☐ Fast
- ☐ Constant time
- ☐ Debugging

| | **512-bit XGCD** | **255-bit XGCD** |
|---|---|---|
| Technology | GF 12nm FinFET | |
| Voltage | 0.9V | |
| Input Bitwidth | 512 | 255 |
| Max Clock Frequency (GHz) | 3.25 | 3.25 |
| Total Area ($mm^2$) | 0.25 | 0.076 |
| **Average Execution Time (ns)** | | **119** |
| **Constant-time Execution Time (ns)** | | **119** |

# Die photo and chip specs




Execution modes:
- ☐ Fast
- ☐ Constant time
- ☐ Debugging

| | 512-bit XGCD | 255-bit XGCD |
|---|---|---|
| Technology | GF 12nm FinFET | |
| Voltage | 0.9V | |
| Input Bitwidth | 512 | 255 |
| Max Clock Frequency (GHz) | 3.25 | 3.25 |
| Total Area (mm$^2$) | 0.25 | 0.076 |
| **Average Execution Time (ns)** | **176** | **119** |
| **Constant-time Execution Time (ns)** | **239** | **119** |

# Non-constant-time XGCD

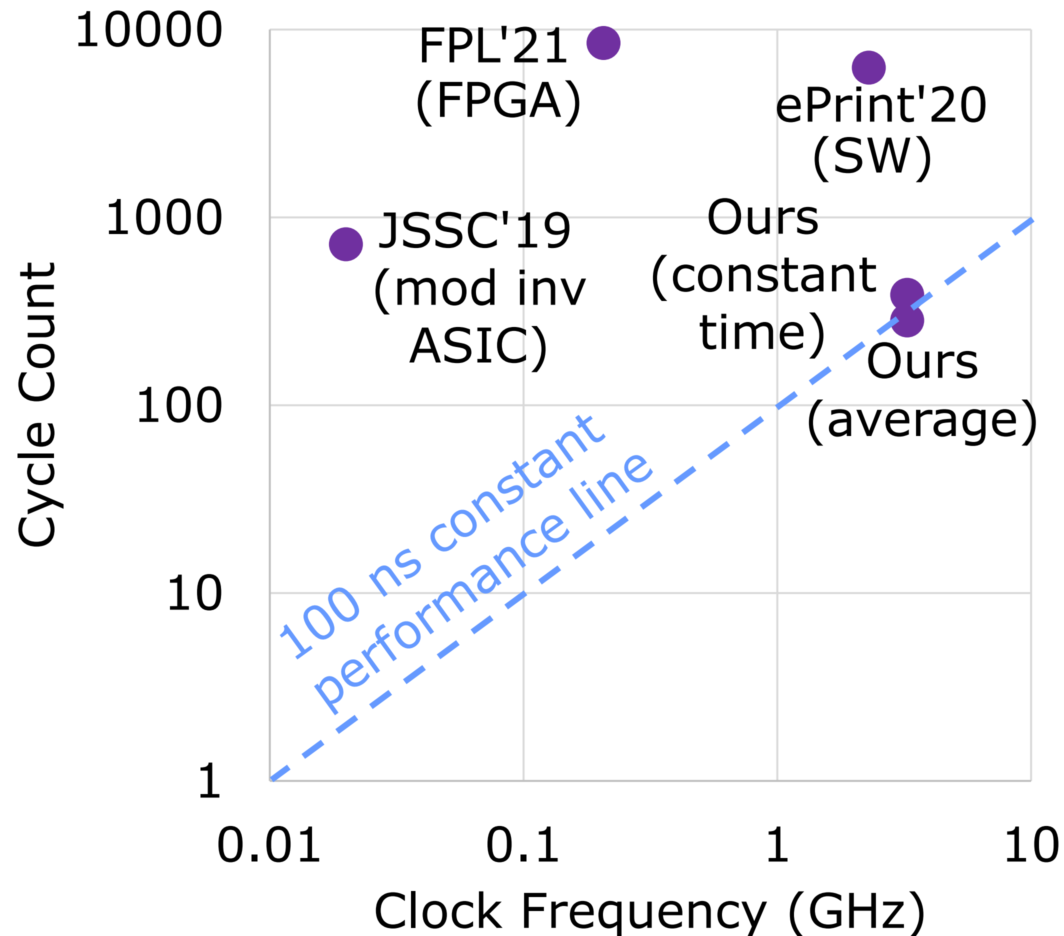


- Can easily scale performance of our design to the bitwidth in prior work
  - All non-constant-time prior work used division algorithms

- 18X faster than prior 1024-bit HW simulations (APCCAS'20, ISCAS'21)
- 19X faster than a chip with a 4096-bit mod inv unit (ISSCC'23)

- Area comparisons with bitwidth and technology scaling
  - 8.5X to 14X smaller than APCCAS'20
  - 2X to 3.4X smaller than ISCAS'21

# Constant-time XGCD




- JSSC’19, FPL’21 use division algos
- 255-bit constant-time XGCD speedups
  - 303X faster than JSSC’19
  - 344X faster than FPL’21
  - 23X faster than ePrint’20
- Our fast average execution is 1.4X faster than our constant-time XGCD

# Key contributions

- We present the first ASIC for XGCD, with a constant-time config
  - Validates the advantage of subtraction-based XGCD algorithms in HW
- We address challenges with implementing CSAs in practice
  - Avoid large-integer comparisons for control
  - Use late selects and precompute control to minimize control delay
  - Contribute efficient circuitry for shifting in CS form
- Our chip is 23X faster than SW, 18X faster than HW sims, and 303X faster than prior chips for modular inversion

Paper

Personal Website