RISC-V

System-on-Chip Design

Harris, Stine, Thompson, & Harris

Chapter 15:

Multiply & Divide

Chapter 15 :: Topics

Extensions: M (Multiply & Divide)

12.1 Principles

12.2 RISC-V Practices: M-extension

12.3 Test Plan

12.4 Wally Implementation

Principles

- **Signed/Unsigned Multiplication**
- **Carry-Save Adders**
- **Multiplication**
  - Array Multiplier
  - Signed Multiplication
  - Booth Encoding
  - Multiplier Synthesis
- **Division**
  - Radix-2 Division
  - Signed Division

Intro to Multiplication

- Multiplication of **XLEN** operands produces **2 XLEN** result.
- Unlike addition, multiplication **result** differs depending on operand **signs**.

Signed / Unsigned Multiplication

Example: XLEN = 4

- 1111 x 1111
- Signed x Signed:
  - 1111 = -1
  - -1 x -1 = 1: 0000 **0001**
- Unsigned x Unsigned:
  - 1111 = 15
  - 15 x 15 = 225: 1110 **0001**
- Signed x Unsigned:
  - 1111 = -1, 1111 = 15
  - -1 x 15 = -15: 1111 **0001**

The bottom XLEN (4) bits are the same independent of sign.

RISC-V Multiply Instructions

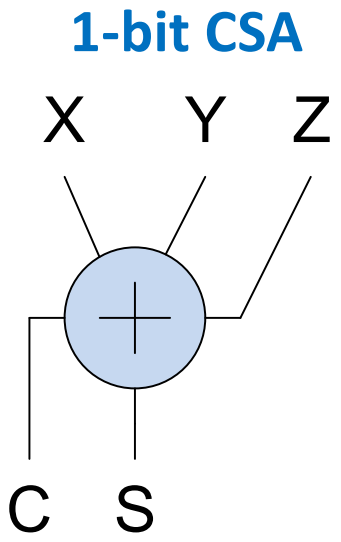
- **Lower XLEN bits:** `mul`
- **Upper XLEN bits:**
  - Signed/Signed: `mulh`
  - Unsigned: `mulhu`
  - Signed/Unsigned: `mulhsu`

Chapter 15: Multiply & Divide

Adders

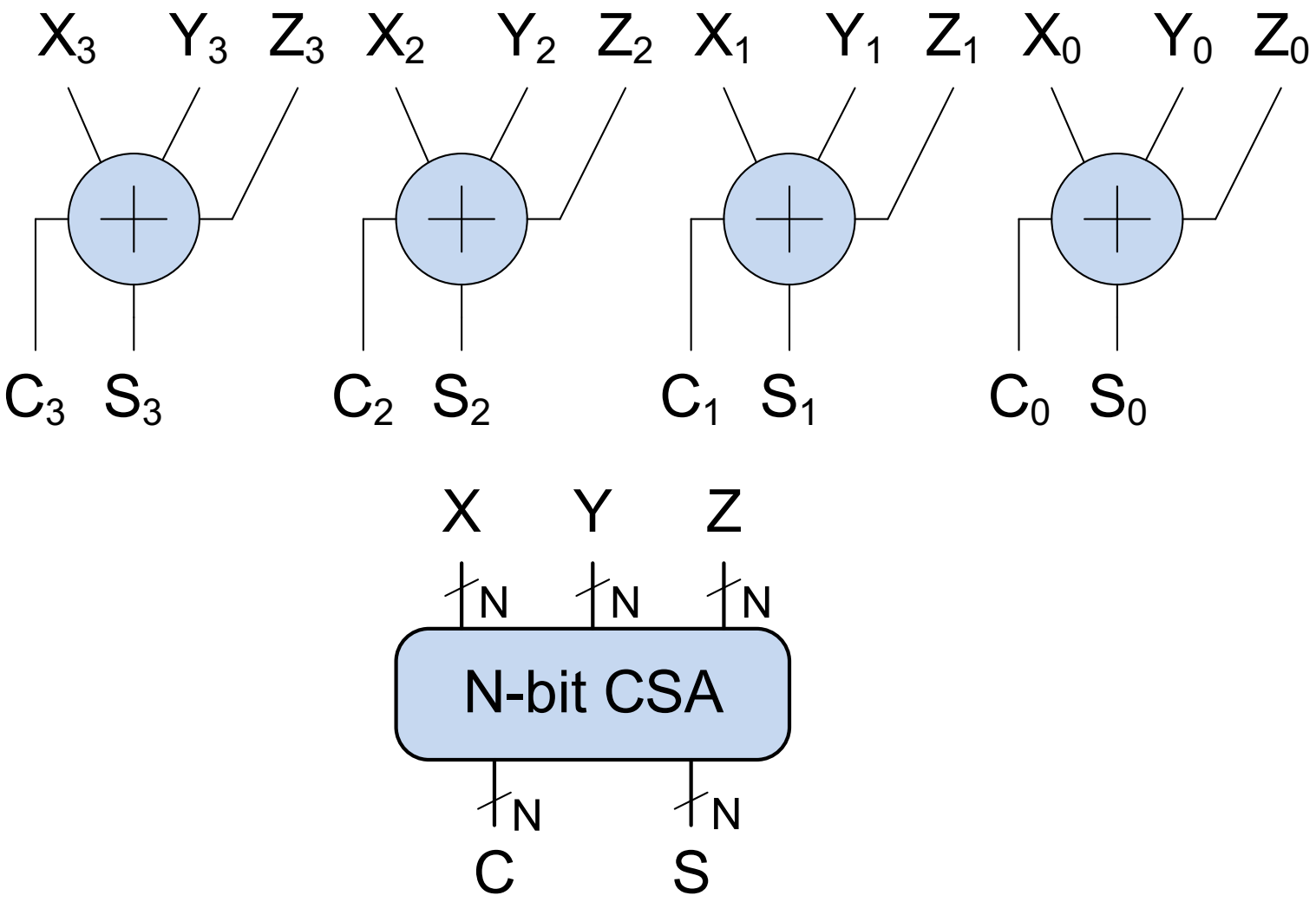
Carry-Save Adders (CSAs)

- **Used to add multiple words** (i.e., partial products in multipliers)



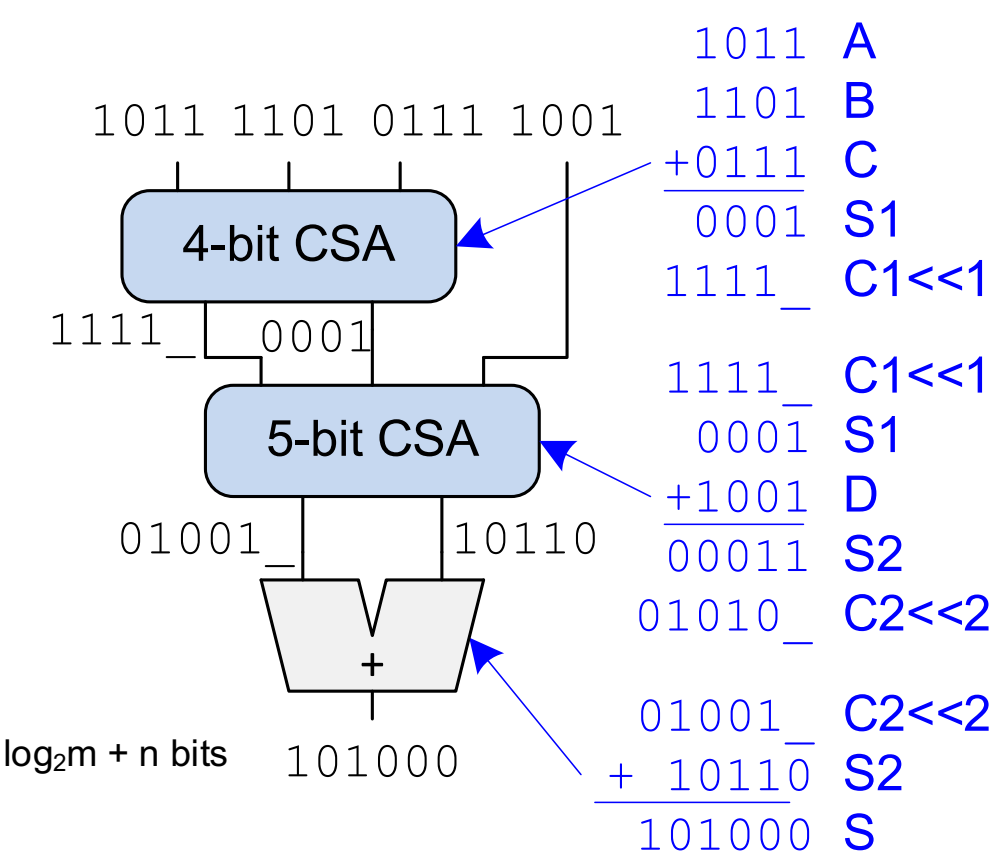
- **CSA operation:**
  - **3 inputs:** *X*, *Y*, *Z* (operands)
  - **2 outputs:** S (sum), C (carry out)
  - **Function:** {*C*, *S*} = *X* + *Y* + *Z*
    - Carry out (*C*) has double the weight of the sum (*S*)
    - To produce the actual sum, add 2*C* + *S* using regular carry-propagate adder (CPA)
  - Also called 3:2 adder

N-bit CSA

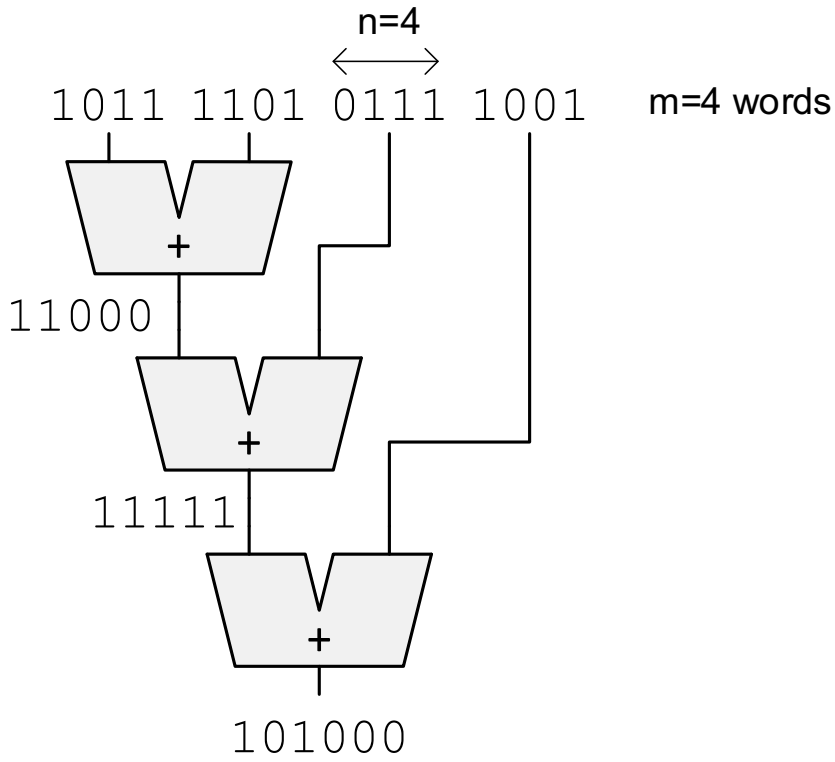

X3 Y3 Z3
X2 Y2 Z2
X1 Y1 Z1
X0 Y0 Z0
C3 S3
C2 S2
C1 S1
C0 S0
X Y Z
N N N
N-bit CSA
N N
C S

Addition of Four 4-bit Operands

CSA (Carry-Save Adder)

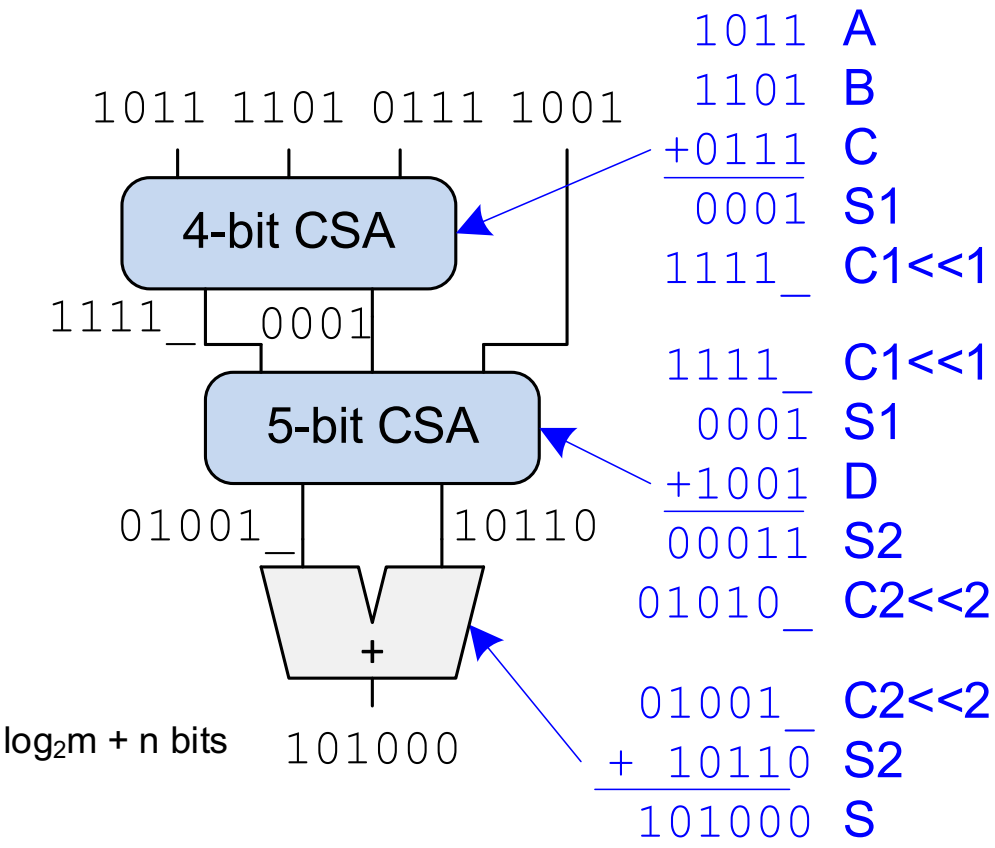


CPA (Carry-Propagate Adder)



Addition of Four 4-bit Operands

CSA

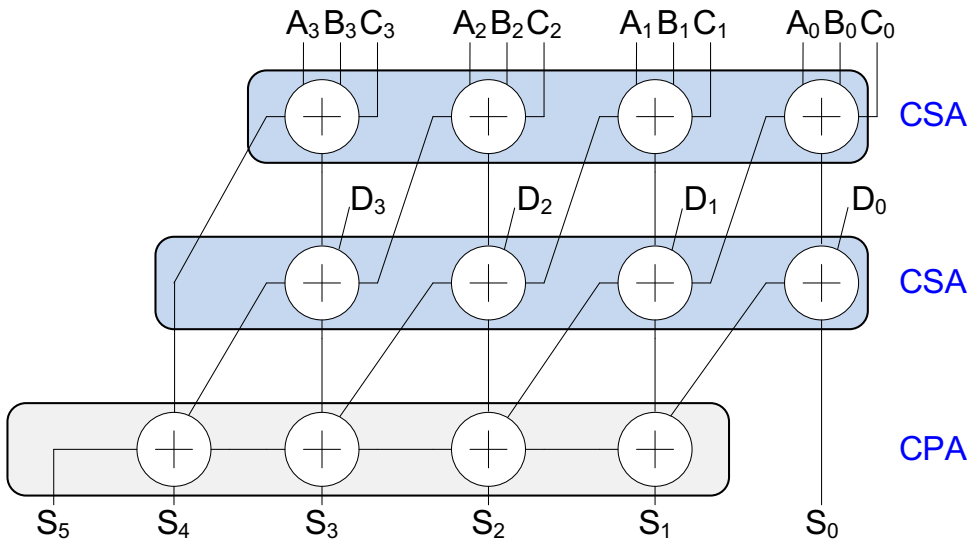



```
  1011  A
  1101  B
+ 0111  C
------
  0001  S1
 1111_  C1<<1

 1111_  C1<<1
  0001  S1
+ 1001  D
------
  0011  S2
 0101_  C2<<2

 0100_  C2<<2
+ 1011  S2
------
  1010  S
```

CSA Implementation



Chapter 15: Multiply & Divide

Multipliers

Multiplication

P* = *Y* x *X

- ***Y***: *M*-bit Multiplicand $\{y_{M-1}, y_{M-2}, \ldots y_1, y_0\}$
- ***X***: *N*-bit Multiplier $\{x_{M-1}, x_{M-2}, \ldots x_1, x_0\}$
- ***P***: *M*+*N*-bit product

$$P = \left(\sum_{j=0}^{M-1} y_j 2^j \right) \left(\sum_{i=0}^{N-1} x_i 2^i \right) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_i y_j 2^{i+j}$$

Multiplication

$P = Y \times X$

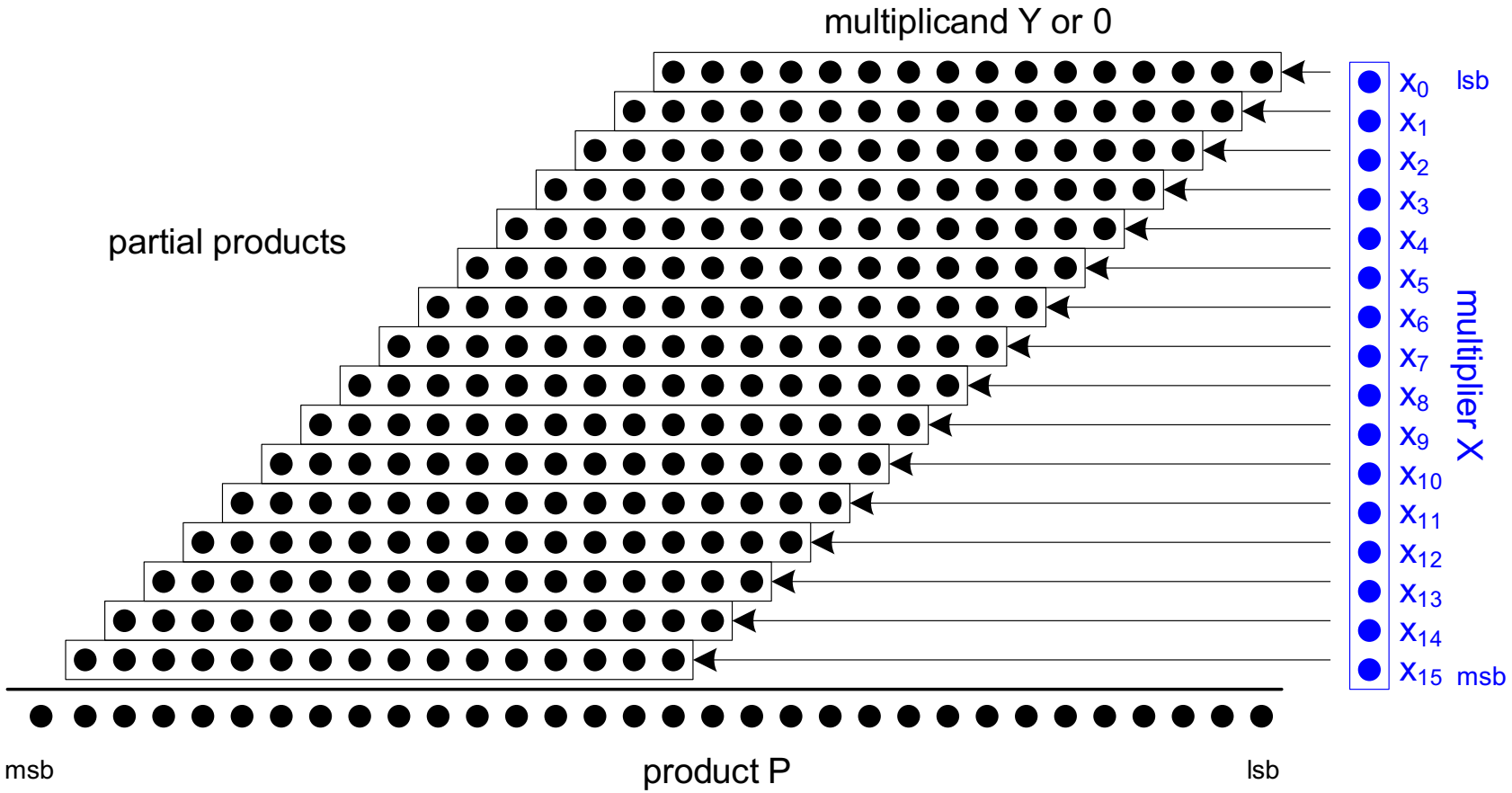
- ***Y***: *M*-bit Multiplicand $\{y_{M-1}, y_{M-2}, \ldots y_1, y_0\}$
- ***X***: *N*-bit Multiplier $\{x_{M-1}, x_{M-2}, \ldots x_1, x_0\}$
- ***P***: *M*+*N*-bit product

Example: 4x4 unsigned multiplication

				y_3	y_2	y_1	y_0	multiplicand	1011 : 11_{10}
				x_3	x_2	x_1	x_0	multiplier	x 0101 : 5_{10}
				x_0y_3	x_0y_2	x_0y_1	x_0y_0		1011
			x_1y_3	x_1y_2	x_1y_1	x_1y_0		partial products	0000
		x_2y_3	x_2y_2	x_2y_1	x_2y_0				1011
	x_3y_3	x_3y_2	x_3y_1	x_3y_0					+ 0000
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0	product	0110111 : 55_{10}

Multiplication: Dot Notation

Dot notation is useful for larger multiplications.

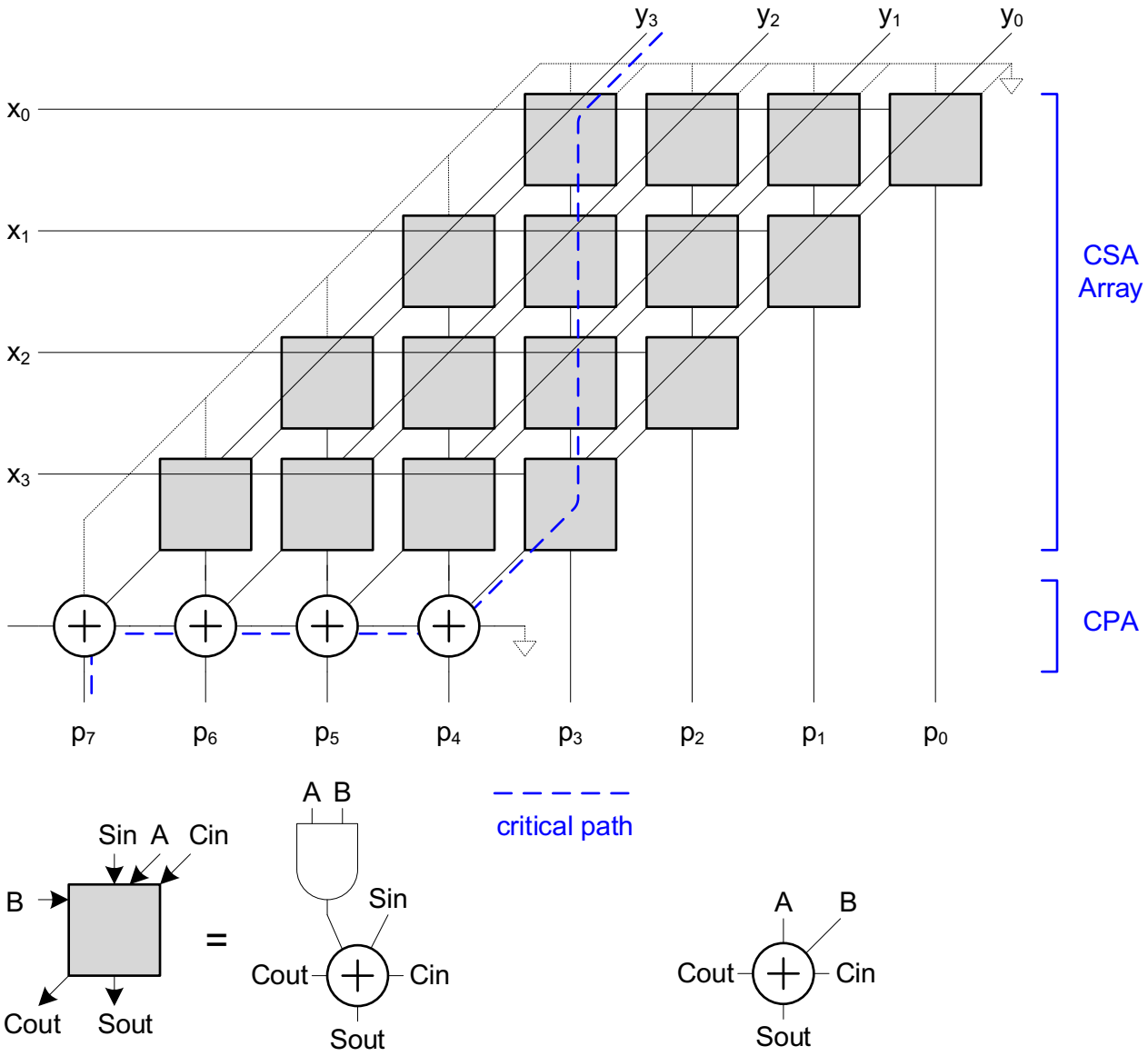


Example: 16 x 16 unsigned multiplier

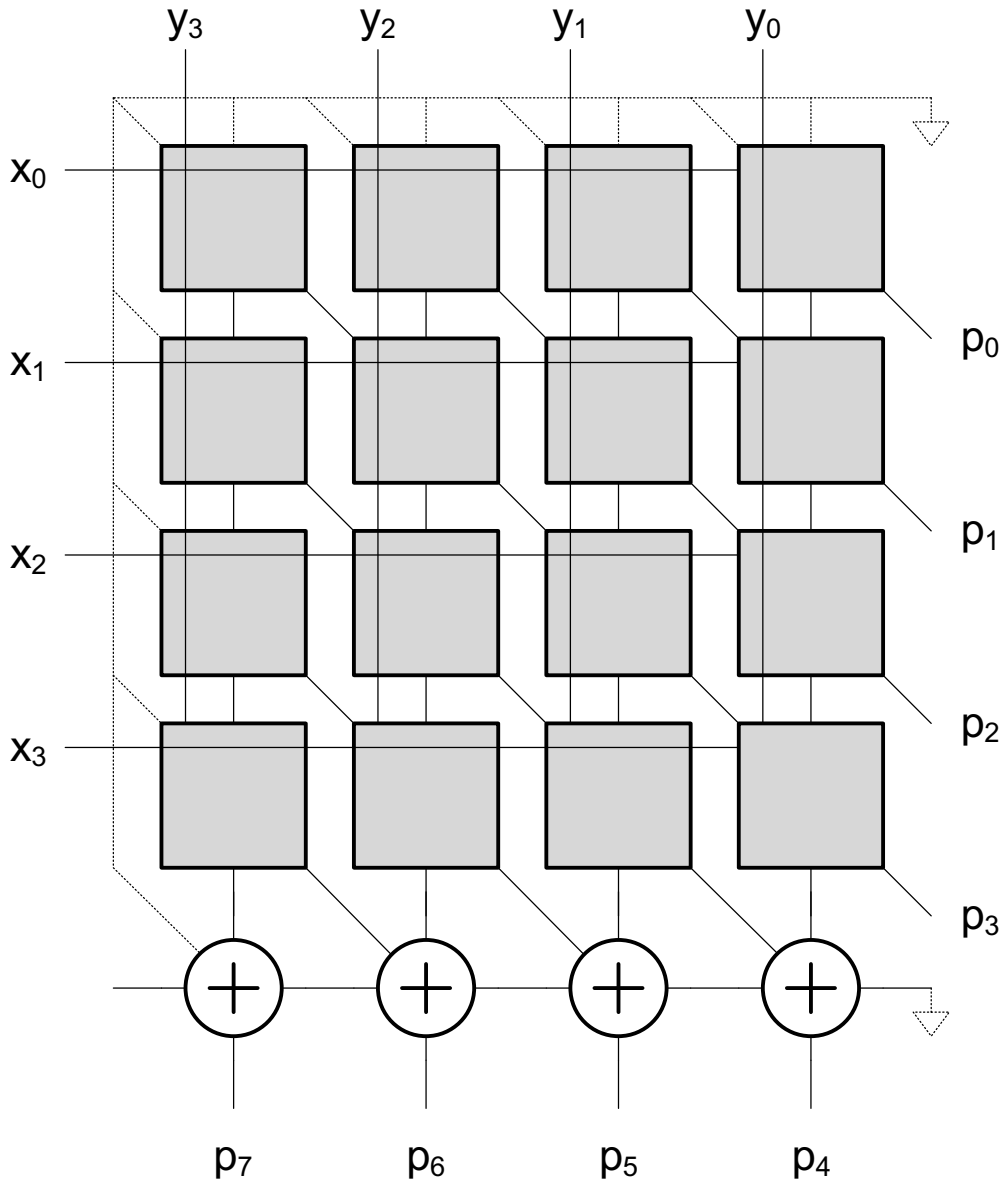
Chapter 15: Multiply & Divide

Array Multiplier

Array Multiplier



Array Multiplier: Square Format



Square format fits better on a chip

Chapter 15: Multiply & Divide

Signed Multiplication

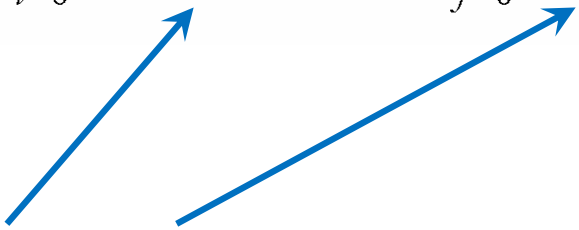
Signed Multiplication

Two of the partial products may be negative

- when x_{M-1} or x_{N-1} are 1 (not 0)

$$P = \left(-y_{M-1} 2^{M-1} + \sum_{j=0}^{M-2} y_j 2^j \right) \left(-x_{n-1} 2^{N-1} + \sum_{i=0}^{N-2} x_i 2^i \right)$$

$$= \sum_{i=0}^{N-2} \sum_{j=0}^{M-2} x_i y_j 2^{i+j} + x_{N-1} y_{M-1} 2^{M+N-2} - \left(\sum_{i=0}^{N-2} x_i y_{M-1} 2^{i+M-1} + \sum_{j=0}^{M-2} x_{N-1} y_j 2^{j+N-1} \right)$$



Partial products may be negative.

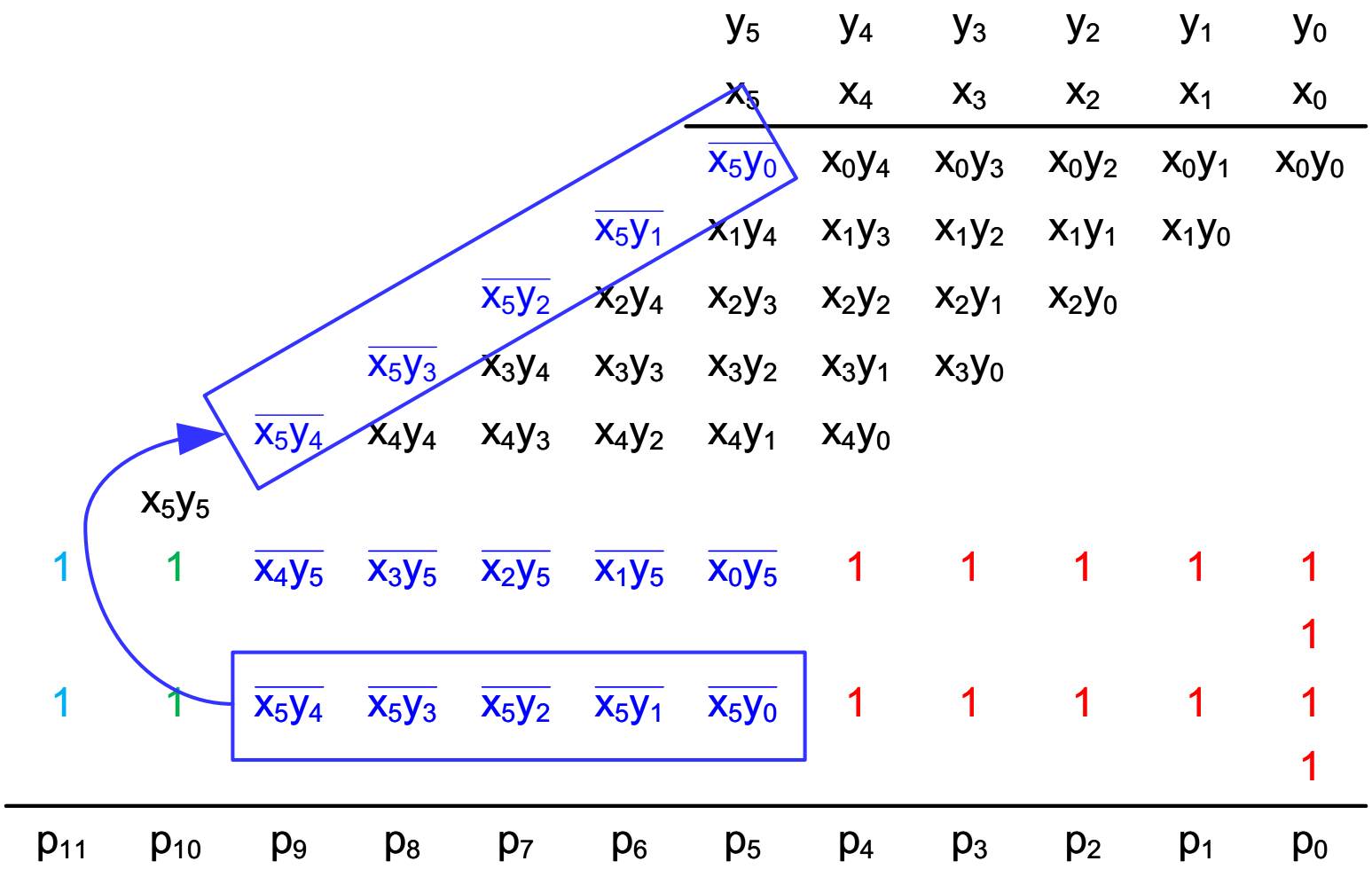
Signed Multiplication

$$\sum_{i=0}^{N-2}\sum_{j=0}^{M-2} x_i y_j 2^{i+j} + x_{N-1} y_{M-1} 2^{M+N-2} - \sum_{i=0}^{N-2} x_i y_{M-1} 2^{i+M-1} - \sum_{j=0}^{M-2} x_{N-1} y_j 2^{j+N-1}$$

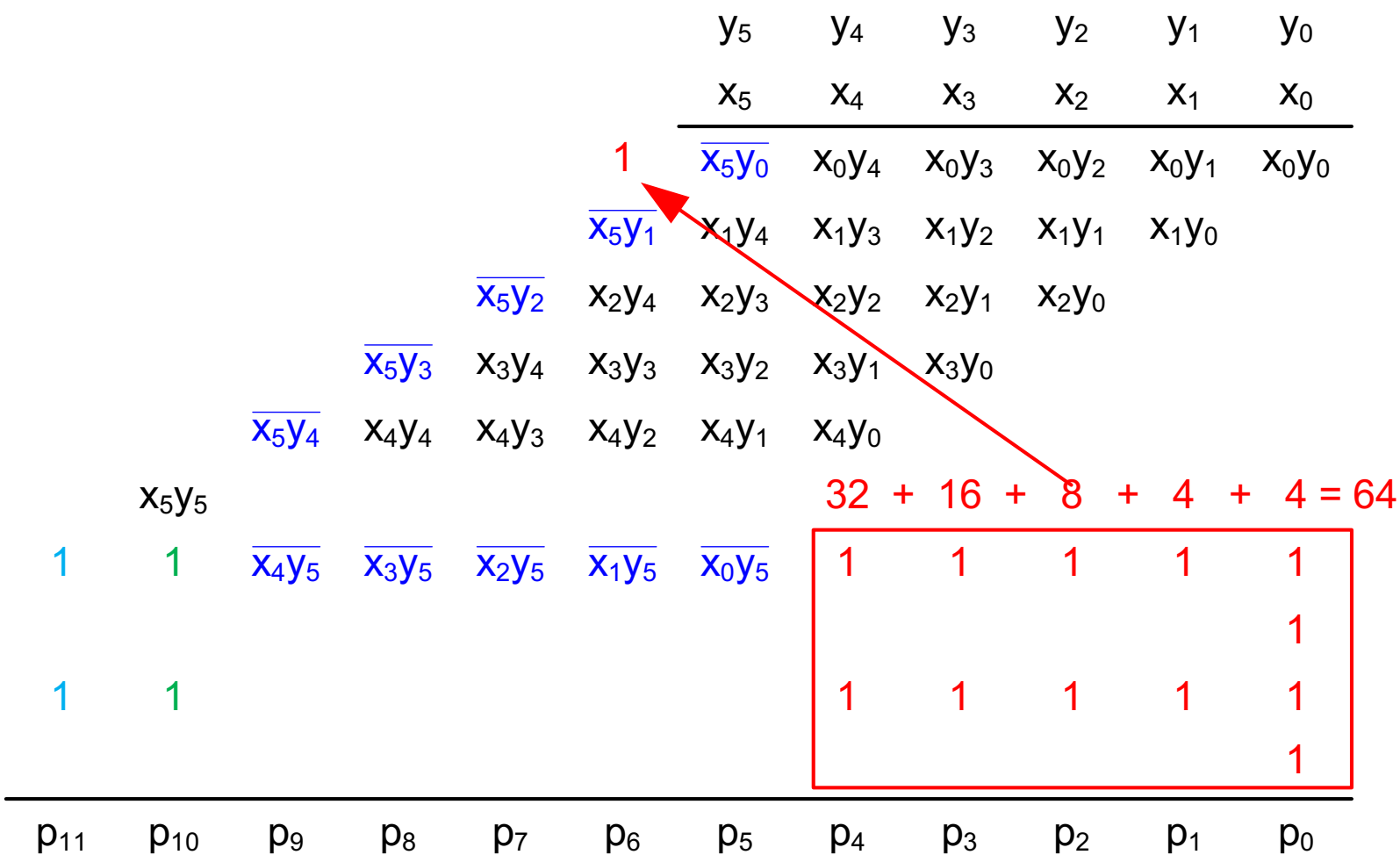
						y_5	y_4	y_3	y_2	y_1	y_0
						x_5	x_4	x_3	x_2	x_1	x_0
							x_0y_4	x_0y_3	x_0y_2	x_0y_1	x_0y_0
						x_1y_4	x_1y_3	x_1y_2	x_1y_1	x_1y_0	
					x_2y_4	x_2y_3	x_2y_2	x_2y_1	x_2y_0		
				x_3y_4	x_3y_3	x_3y_2	x_3y_1	x_3y_0			
			x_4y_4	x_4y_3	x_4y_2	x_4y_1	x_4y_0				
	x_5y_5										
1	1	$\overline{x_4y_5}$	$\overline{x_3y_5}$	$\overline{x_2y_5}$	$\overline{x_1y_5}$	$\overline{x_0y_5}$	1	1	1	1	1
											1
1	1	$\overline{x_5y_4}$	$\overline{x_5y_3}$	$\overline{x_5y_2}$	$\overline{x_5y_1}$	$\overline{x_5y_0}$	1	1	1	1	1
											1
p_{11}	p_{10}	p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

$$= \sum_{i=0}^{N-2}\sum_{j=0}^{M-2} x_i y_j 2^{i+j} + x_{N-1} y_{M-1} 2^{M+N-2} - \left(\sum_{i=0}^{N-2} x_i y_{M-1} 2^{i+M-1} + \sum_{j=0}^{M-2} x_{N-1} y_j 2^{j+N-1} \right)$$

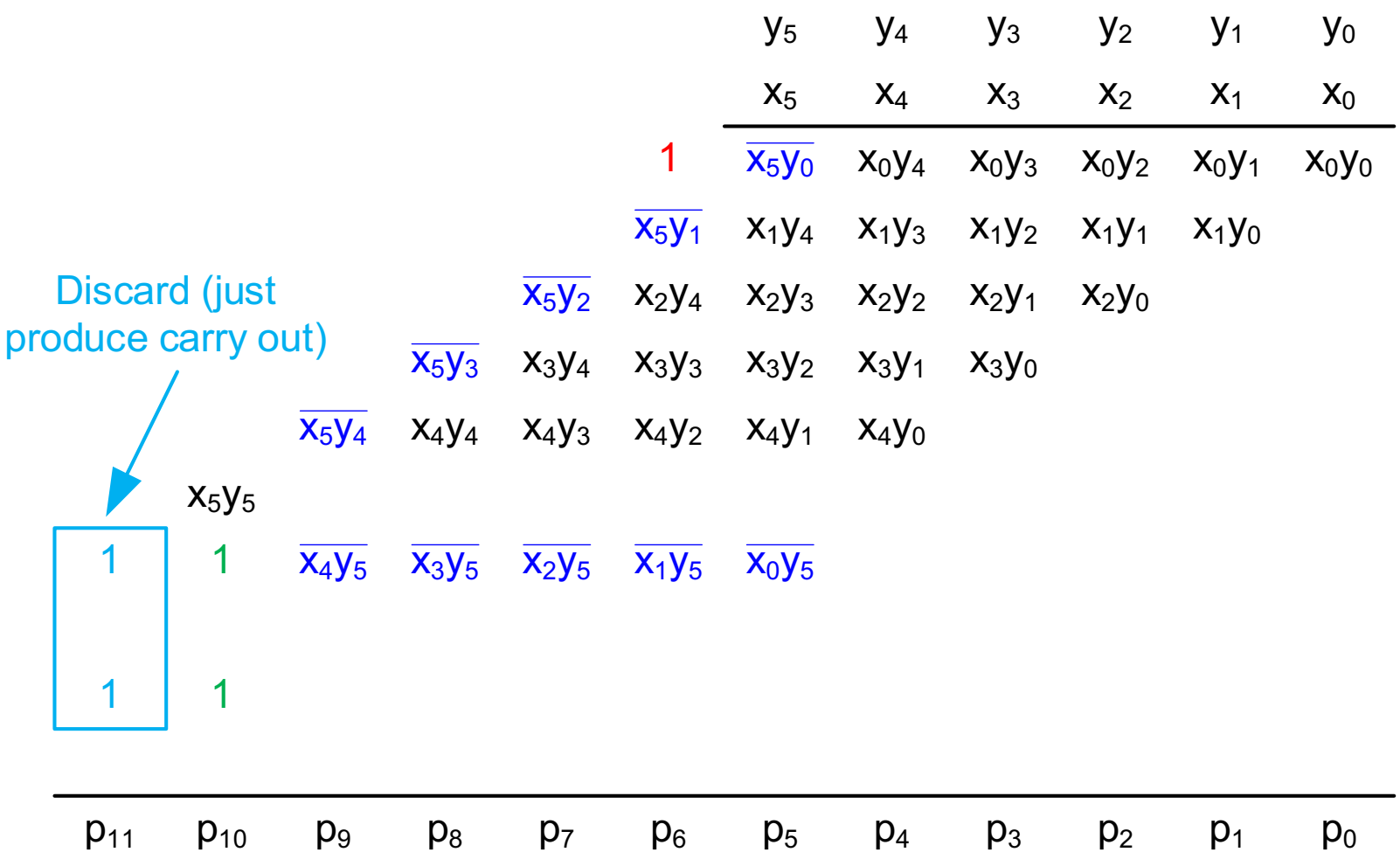
Simplified Partial Products



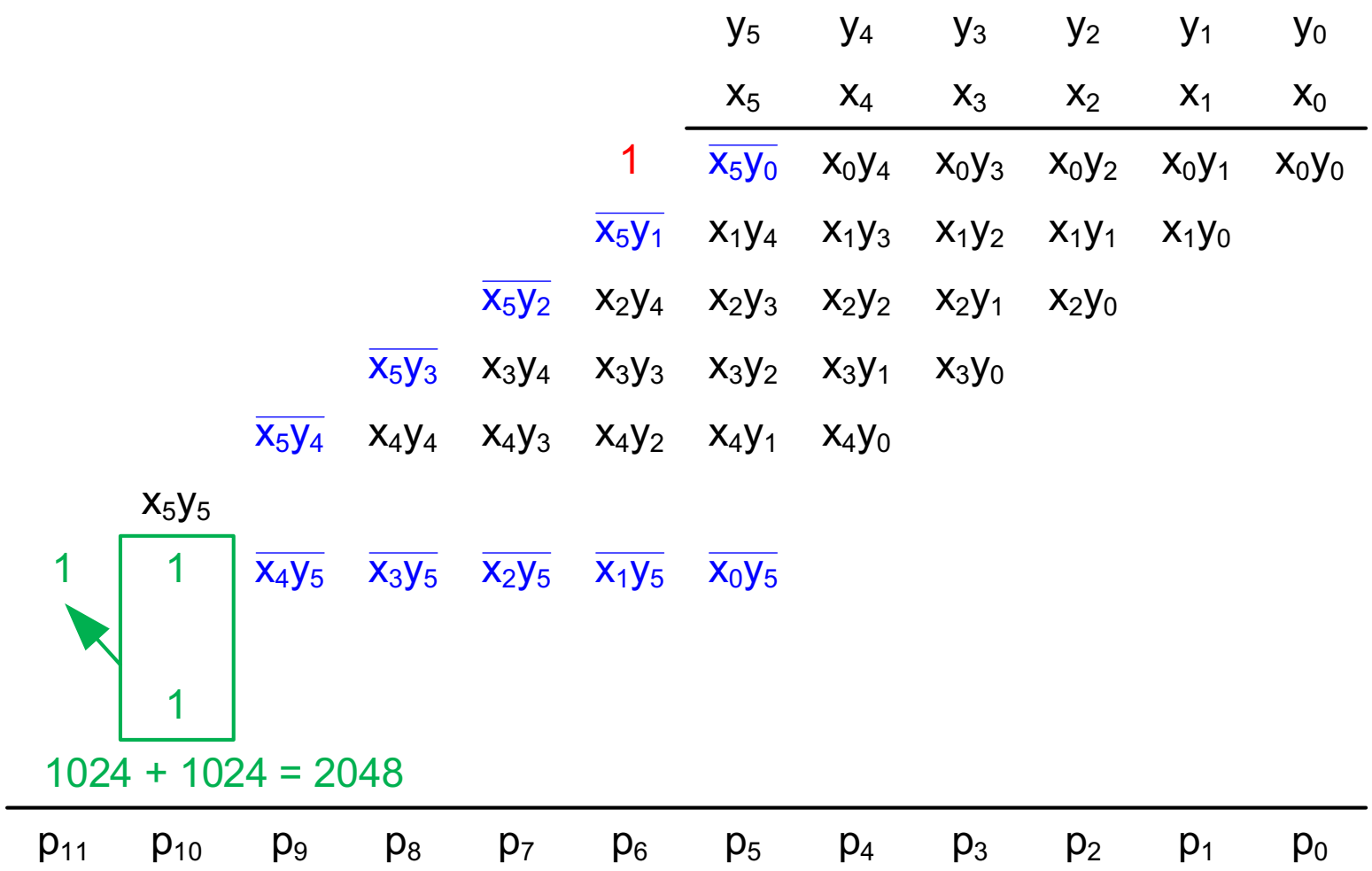
Simplified Partial Products, Cont'd



Simplified Partial Products, Cont'd



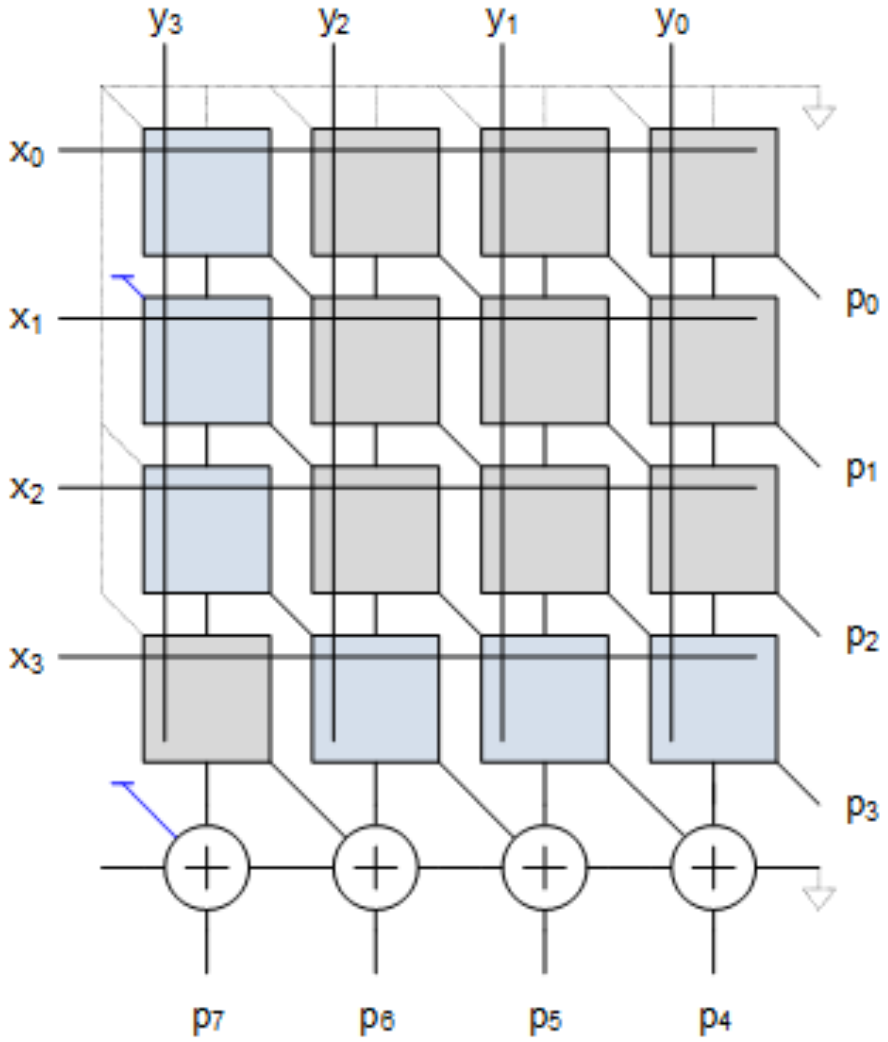
Simplified Partial Products, Cont'd

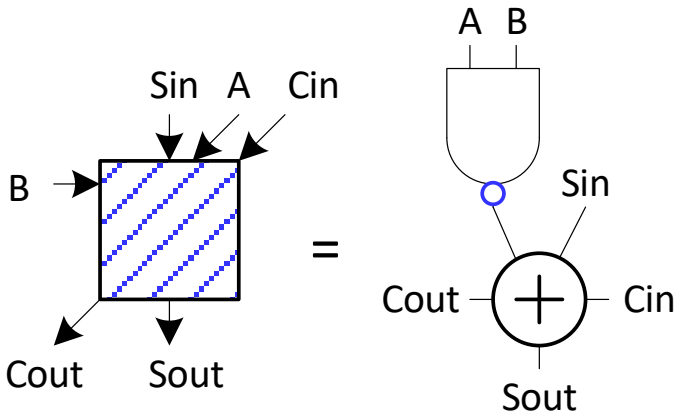


Simplified Partial Products, Cont'd

						y_5	y_4	y_3	y_2	y_1	y_0
						x_5	x_4	x_3	x_2	x_1	x_0
					1	$\overline{x_5y_0}$	x_0y_4	x_0y_3	x_0y_2	x_0y_1	x_0y_0
					$\overline{x_5y_1}$	x_1y_4	x_1y_3	x_1y_2	x_1y_1	x_1y_0	
				$\overline{x_5y_2}$	x_2y_4	x_2y_3	x_2y_2	x_2y_1	x_2y_0		
			$\overline{x_5y_3}$	x_3y_4	x_3y_3	x_3y_2	x_3y_1	x_3y_0			
		$\overline{x_5y_4}$	x_4y_4	x_4y_3	x_4y_2	x_4y_1	x_4y_0				
1	x_5y_5	$\overline{x_4y_5}$	$\overline{x_3y_5}$	$\overline{x_2y_5}$	$\overline{x_1y_5}$	$\overline{x_0y_5}$					
p_{11}	p_{10}	p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

Signed Multiplier





Chapter 15: Multiply & Divide

Booth Encoding

Multiplication Radix

- **Radix 2:** one partial product for each bit
  - **Partial product:**
    - $x_i = 0$: $P = 0$
    - $x_i = 1$: $P = Y$
- **Radix 4:** one partial product for every 2 bits
  - **Partial products:**
    - $\{x_{2i+1}, x_{2i}\} = 00$: $P = 0$
    - $\{x_{2i+1}, x_{2i}\} = 01$: $P = Y$
    - $\{x_{2i+1}, x_{2i}\} = 10$: $P = 2Y$
    - $\{x_{2i+1}, x_{2i}\} = 11$: $P = \mathbf{3Y}$

But 3*Y* is hard to produce.

Booth Encoding

- **Radix 4:**
  - **Partial products:**
    - $\{x_{2i+1}, x_{2i}\} = 00$: $P = 0$
    - $\{x_{2i+1}, x_{2i}\} = 01$: $P = Y$
    - $\{x_{2i+1}, x_{2i}\} = 10$: $P = 2Y =$ **$-2Y$ $(+ 4Y)$**
    - $\{x_{2i+1}, x_{2i}\} = 11$: $P = 3Y =$ **$-Y$ $(+ 4Y)$**

Instead:
Subtract: $-2Y$ and $-Y$
Then, next pair of bits adds **$4Y$**

Booth Encoding

- **Radix 4:**
  - **Partial products:**
    - $\{x_{2i+1}, x_{2i}\} = 00$: $P =$ **0**
    - $\{x_{2i+1}, x_{2i}\} = 01$: $P =$ ***Y***
    - $\{x_{2i+1}, x_{2i}\} = 10$: $P =$ ***-2Y***
    - $\{x_{2i+1}, x_{2i}\} = 11$: $P =$ ***-Y***

Subtract: $-2Y$ and $-Y$ (for 10 and 11 cases)
Then, next pair of bits adds ***4Y***

So, if msb of current pair is 1 ($x_{2i+1} = 1$), then will add ***4Y*** to the next pair.

Booth Encoding

- **Radix 4:**
  - **Partial products, considering msb of previous pair:**
    - $\{x_{2i+1}, x_{2i}, x_{2i-1}\} = 000$: $P = 0$
    - $\{x_{2i+1}, x_{2i}, x_{2i-1}\} = 010$: $P = Y$
    - $\{x_{2i+1}, x_{2i}, x_{2i-1}\} = 100$: $P = -2Y$
    - $\{x_{2i+1}, x_{2i}, x_{2i-1}\} = 110$: $P = -Y$
    - $\{x_{2i+1}, x_{2i}, x_{2i-1}\} = 001$: $P = Y + 0 = Y$
    - $\{x_{2i+1}, x_{2i}, x_{2i-1}\} = 011$: $P = Y + Y = 2Y$
    - $\{x_{2i+1}, x_{2i}, x_{2i-1}\} = 101$: $P = Y + -2Y = -Y$
    - $\{x_{2i+1}, x_{2i}, x_{2i-1}\} = 111$: $P = Y + -Y = 0$

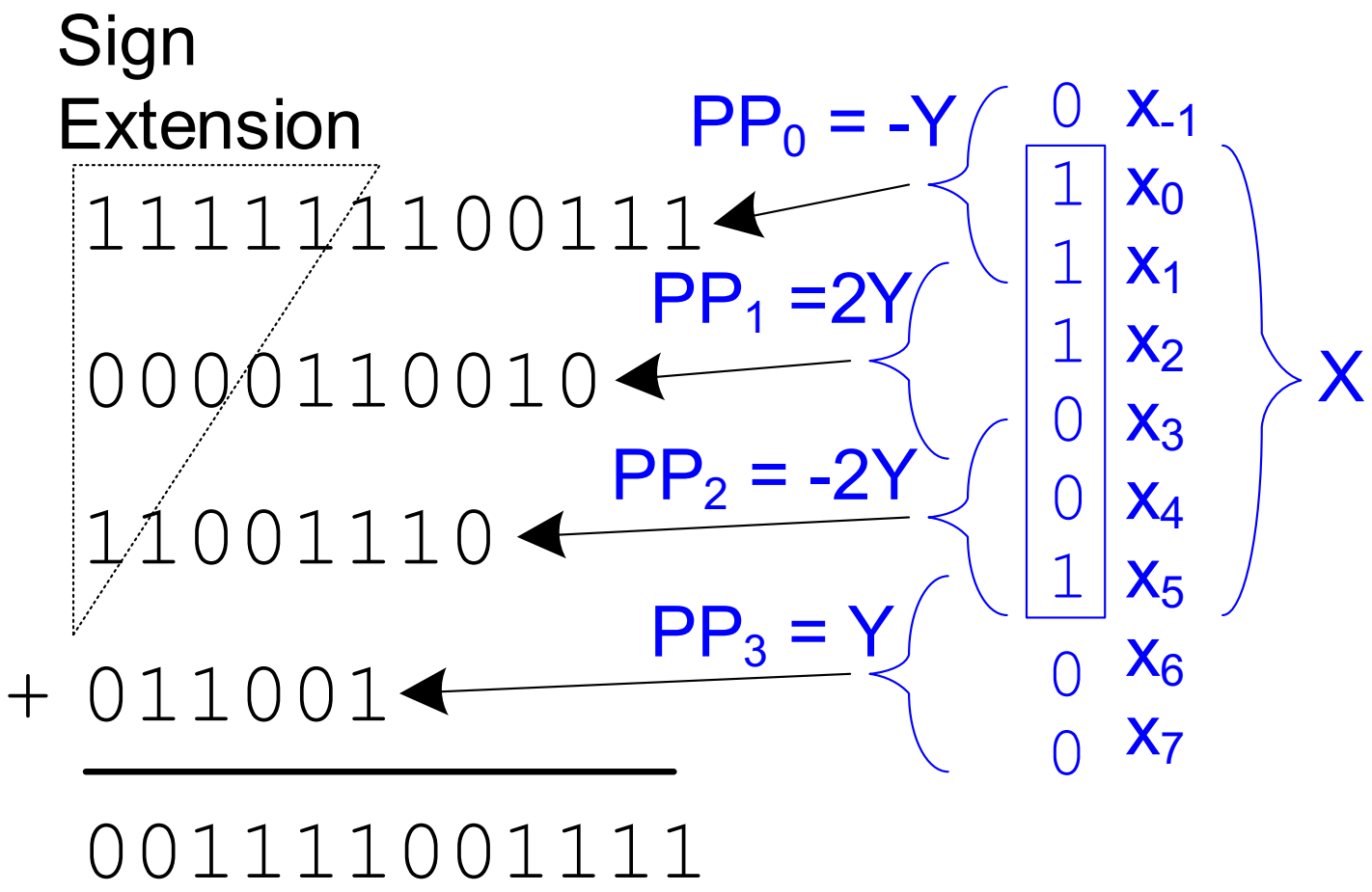
Adding Y in current pair is like adding $4Y$ in previous pair (because of left shift by 2 bits)

Booth Encoding

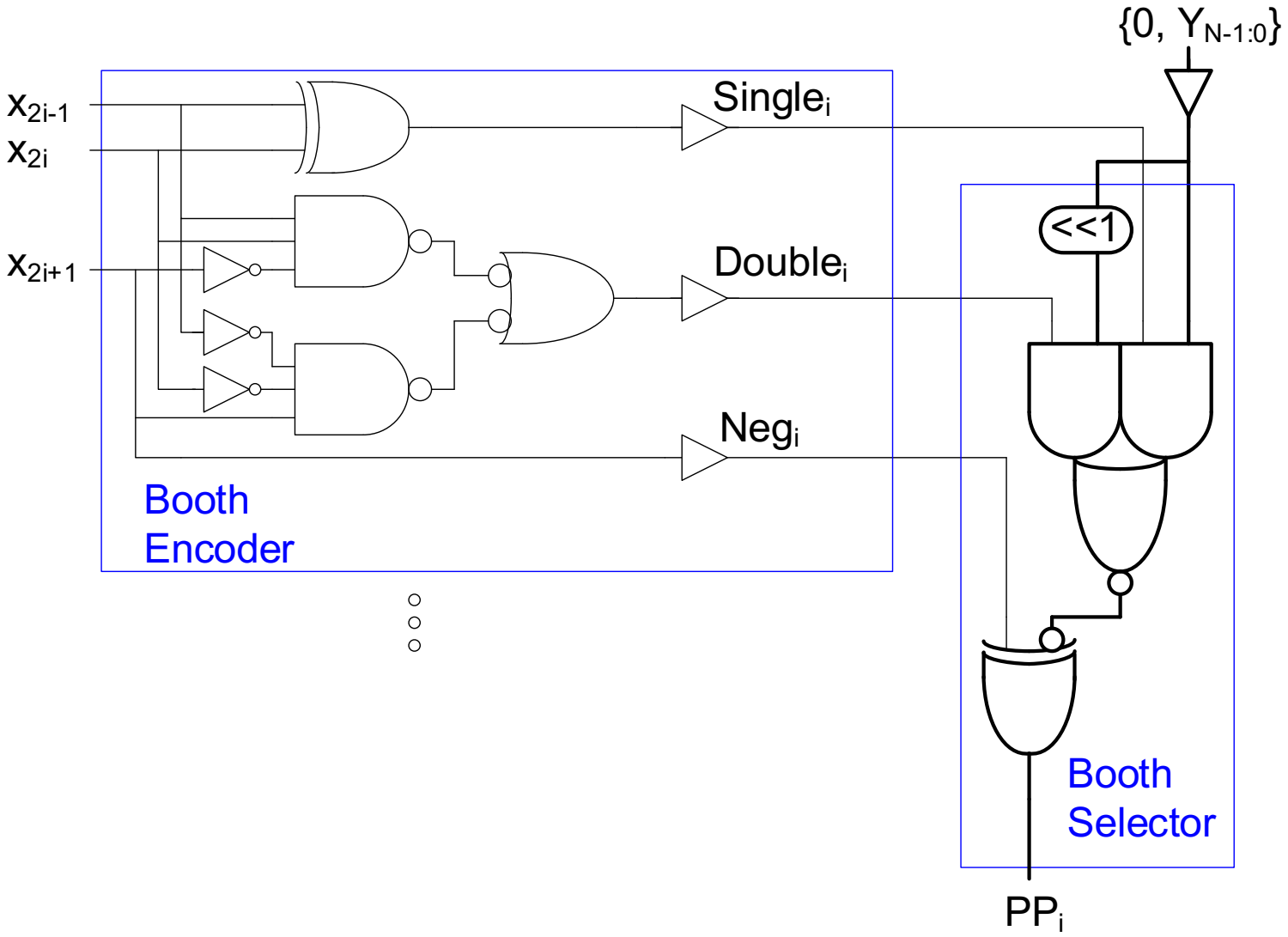
Inputs			Partial Products	Booth Selects		
x_{2i+1}	x_{2i}	x_{2i-1}	PP_i	$Single_i$	$Double_i$	Neg_i
0	0	0	0	0	0	0
0	1	0	Y	1	0	0
1	0	0	$-2Y$	0	1	1
1	1	0	$-Y$	1	0	1
0	0	**1**	Y (= ***Y*** + 0)	1	0	0
0	1	**1**	2Y (= ***Y*** + Y)	0	1	0
1	0	**1**	$-Y$ (= ***Y*** - $2Y$)	1	0	1
1	1	**1**	0 (= ***Y*** - Y = -0)	0	0	1

Booth Encoding Example

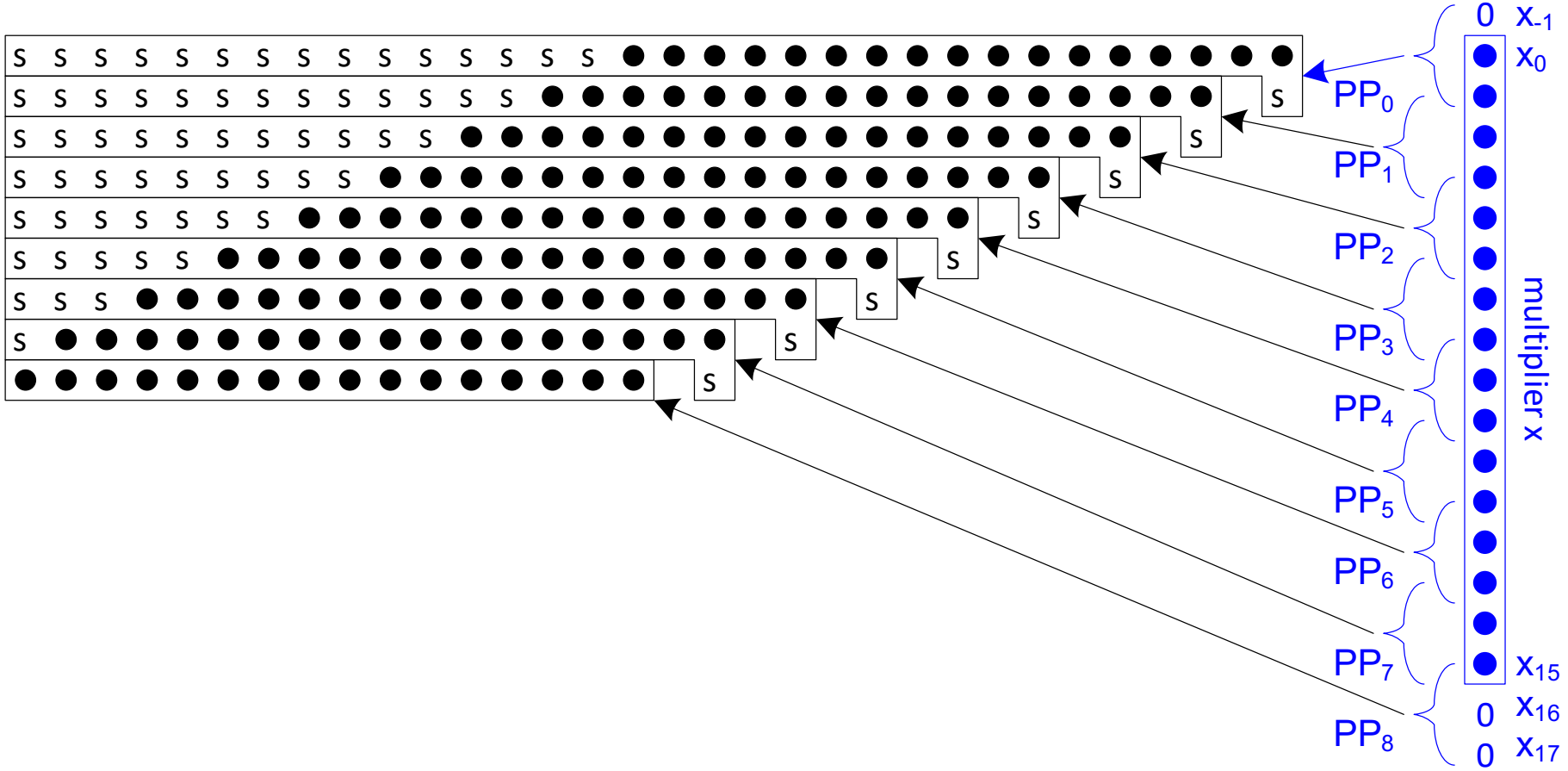
100111 x 011001 = 001111001111 (39 x 25 = 975)



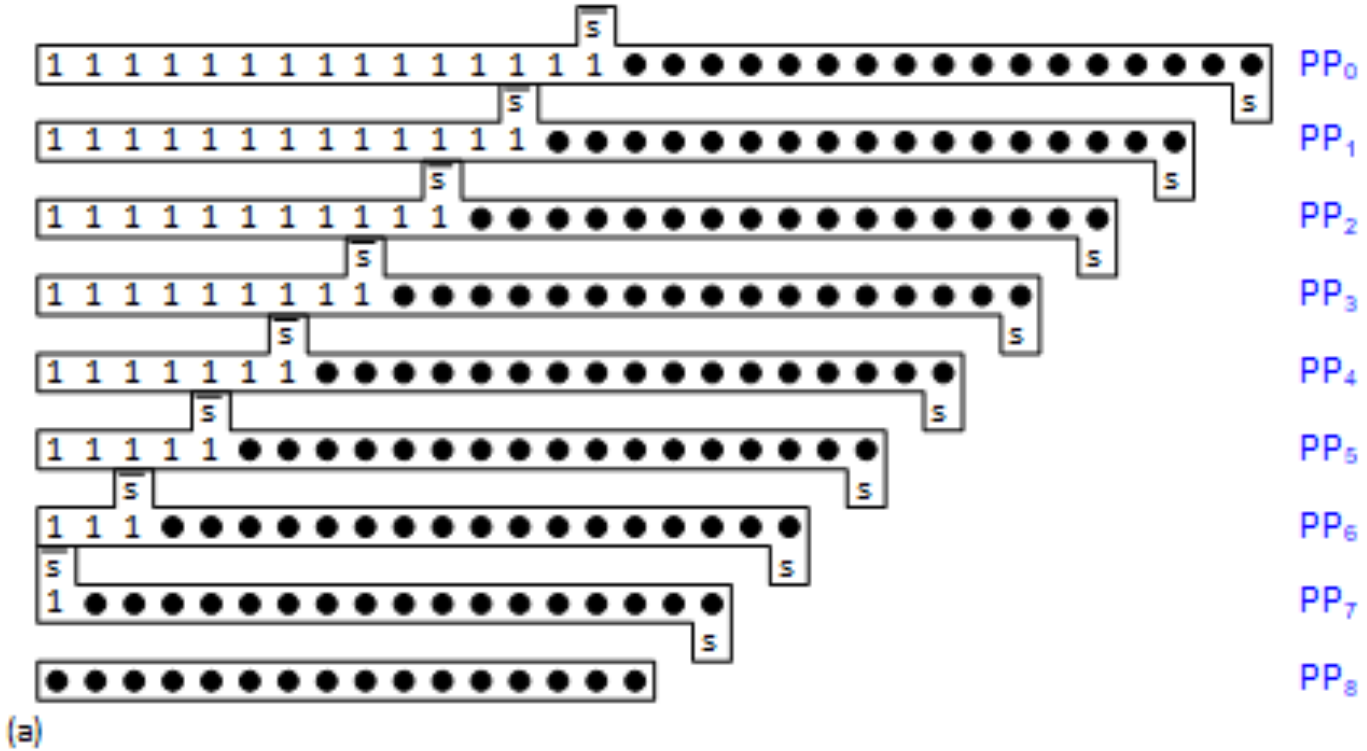
Booth Encoding Hardware



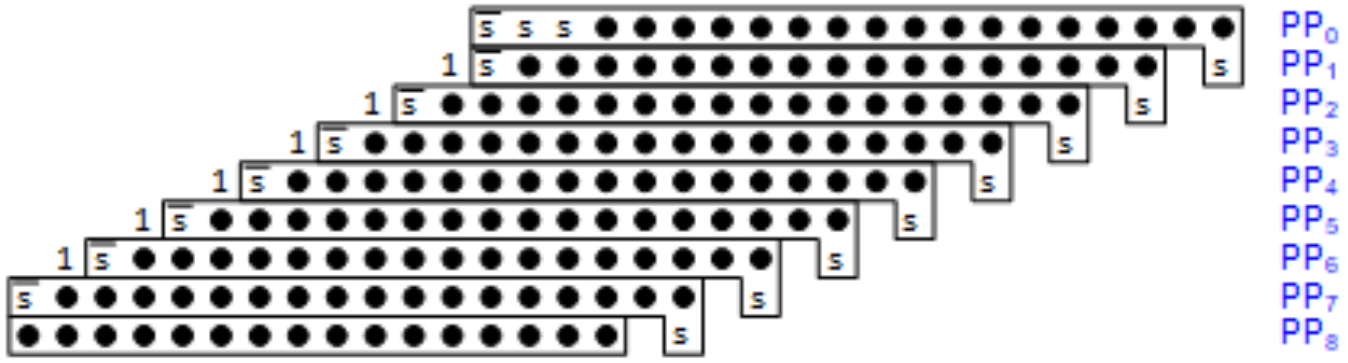
Booth Encoding w/ Sign Extension



Simplified Sign Extension



(a)



(b)

Multiplier synthesis

```
assign p = x * y;

// Synthesis tool optimizes
```

Chapter 15: Multiply & Divide

Division

Intro to Division

X/D = Q + REM/D

or

X = Q x D + REM

$$Q = \sum_{i=0}^{N-1} q_i \, 2^{N-1-i} \qquad X = \sum_{i=0}^{N-1} x_i \, 2^{N-1-i}$$

- **Dividend:** X
- **Divisor:** Y
- **Quotient:** Q
- **Remainder:** REM (same sign as X)

Division Algorithm

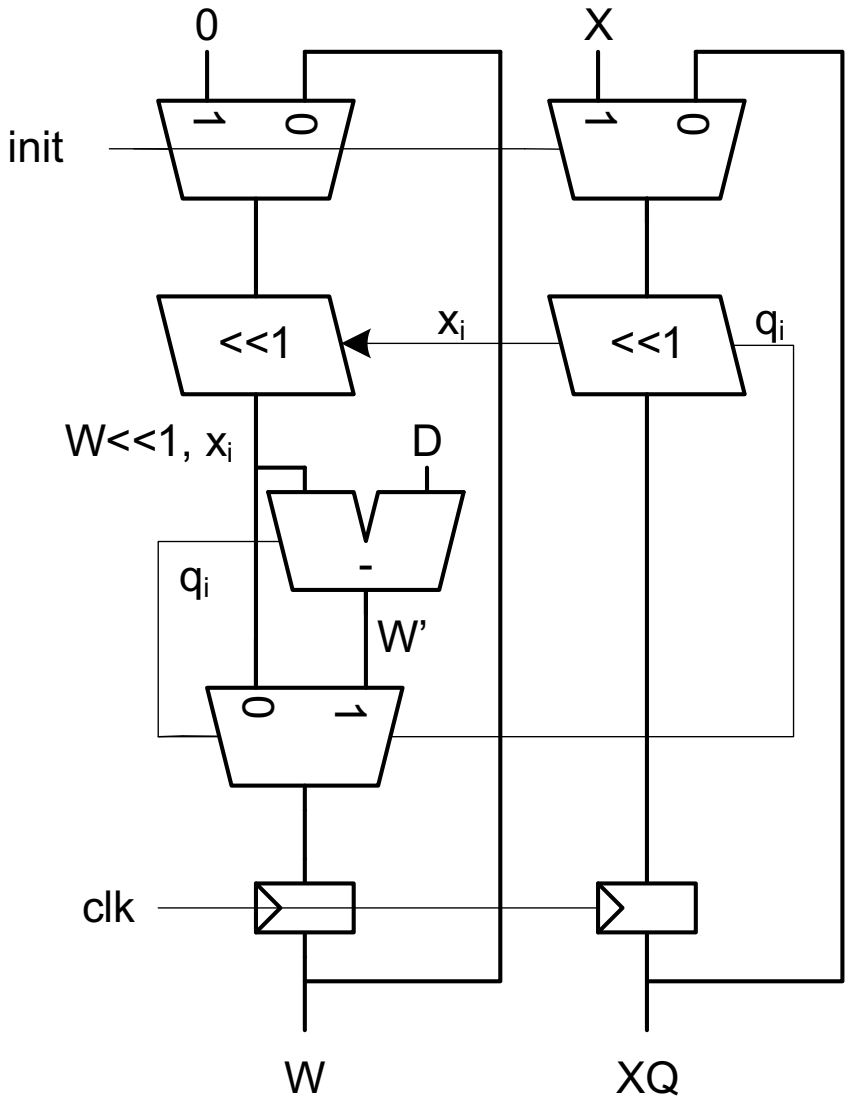
$$Q = \sum_{i=0}^{N-1} q_i \, 2^{N-1-i} \qquad X = \sum_{i=0}^{N-1} x_i \, 2^{N-1-i}$$

For division:
Using big-endian bit numbering (bit 0 is msb and bit *N*-1 is lsb)

$W = 0$
for $i = 0$ to N-1
$W = \{W << 1, x_i\}$
$W' = W - D$
if ($W' \geq 0$) $q_i = 1, W = W'$
else $q_i = 0$

At end of algorithm:
Q = quotient
W = remainder

Radix-2 Unsigned Integer Divider



Unsigned Integer Division Example

i	W<<1, x_i	q_i	W output	XQ output
init			0000	1101
0	0001	**0**	0001	101**0**
1	0011	**1**	0001	01**01**
2	0010	**1**	0000	1**011**
3	0001	**0**	0001	**0110**

$W = 0$

for $i = 0$ to N-1

$W = \{W << 1, x_i\}$

$W' = W - D$

if ($W' \geq 0$) $q_i = 1$, $W = W'$

else $q_i = 0$

Signed Division

Examples:

X`	D	Q	REM
13	2	6	1
13	-2	-6	1
-13	2	-6	-1
-13	-2	6	-1

Signed division:

- Take absolute value of inputs
- Perform unsigned integer division
- If signs of *X* and *D* differ, negate *Q*
- If the sign of *X* is negative, negate *REM*

Chapter 15: Multiply & Divide

RISC-V Practices

M-Extension

op	funct3	funct7	Type	Instruction	Description	Operation
0110011 (51)	000	0000001	R	`mul rd, rs1, rs2`	multiply	`rd = (rs1 * rs2)`$_{XLEN-1:0}$
0110011 (51)	001	0000001	R	`mulh rd, rs1, rs2`	multiply high signed signed	`rd = (rs1 * rs2)`$_{2*XLEN-1:XLEN}$
0110011 (51)	010	0000001	R	`mulhsu rd, rs1, rs2`	multiply high signed unsigned	`rd = (rs1 * rs2)`$_{2*XLEN-1:XLEN}$
0110011 (51)	011	0000001	R	`mulhu rd, rs1, rs2`	multiply high unsigned unsigned	`rd = (rs1 * rs2)`$_{2*XLEN-1:XLEN}$
0110011 (51)	100	0000001	R	`div rd, rs1, rs2`	divide (signed)	`rd = rs1 / rs2`
0110011 (51)	101	0000001	R	`divu rd, rs1, rs2`	divide unsigned	`rd = rs1 / rs2`
0110011 (51)	110	0000001	R	`rem rd, rs1, rs2`	remainder (signed)	`rd = rs1 % rs2`
0110011 (51)	111	0000001	R	`remu rd, rs1, rs2`	remainder unsigned	`rd = rs1 % rs2`

RV64M adds `mulw`, `divw`, `remw`, `divuw`, `remuw` with op = 59. These operate on only the lower 32 bits of a register.

- Multiply, divide, and remainder instructions
- Operate on XLEN-bit words
- Treated as signed or unsigned values
- RV64M adds operations using the lower 32 bits of the registers

Chapter 15: Multiply & Divide

Test Plan

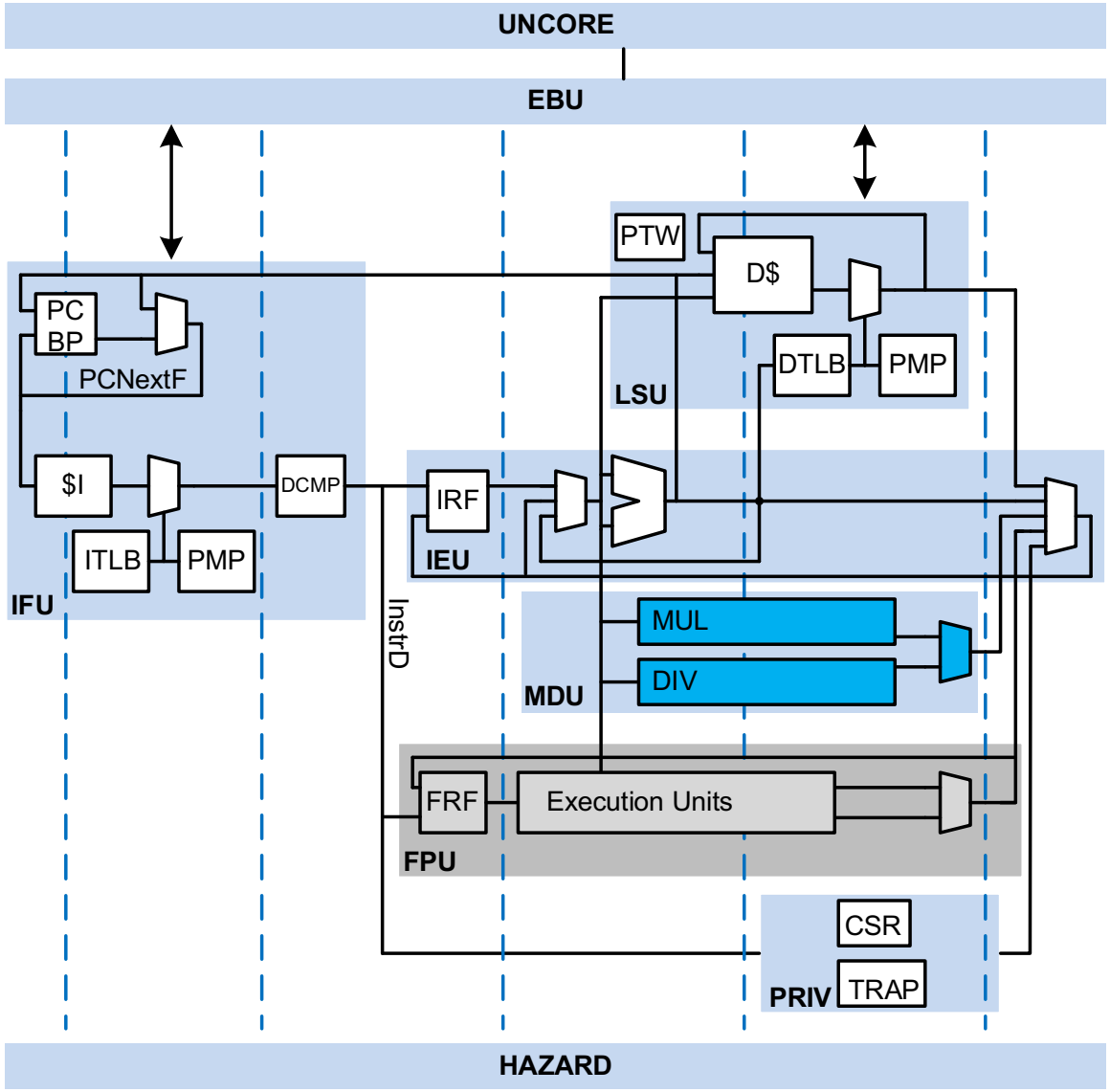
Test Plan: riscv-arch-test

Instruction	RV32M Tests	RV64M Tests
div	587	675
divu	722	821
divuw		820
divw		667
mul	583	680
mulh	583	673
mulhsu	645	740
mulhu	719	811
mulw		670
rem	582	662
remu	720	819
remuw		815
remw		672
Total	**5141**	**9525**

Chapter 15: Multiply & Divide

Wally Implementation

Wally Multiply/Divide Unit (MDU)



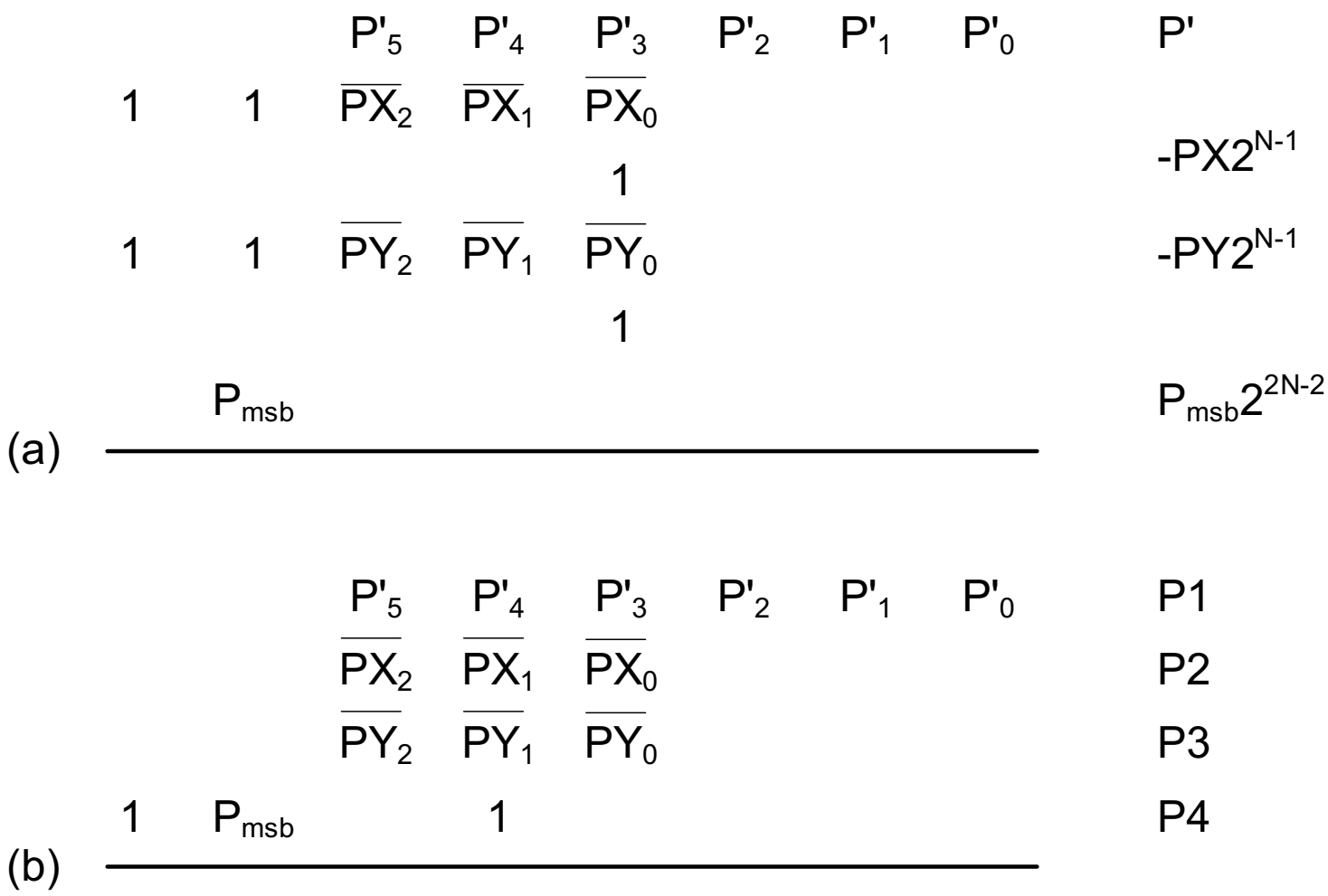
Multiplication: Partial Products

Term	Definition	Comments
P'	$X' \times Y'$	N-1 × N-1 bit multiplication
PX	$Y_{n-1} \times X'$	N-1 bit partial product
PY	$X_{n-1} \times Y'$	N-1 bit partial product
P_{msb}	$X_{n-1} \times Y_{n-1}$	1-bit partial product

Signed & Unsigned Multiplication

Flavor	*X*	*Y*
`mulhu`	Unsigned	Unsigned
`mulh`	Signed	Signed
`mulhsu`	Signed	Unsigned

Pre-Adding 1s in Dot Diagram

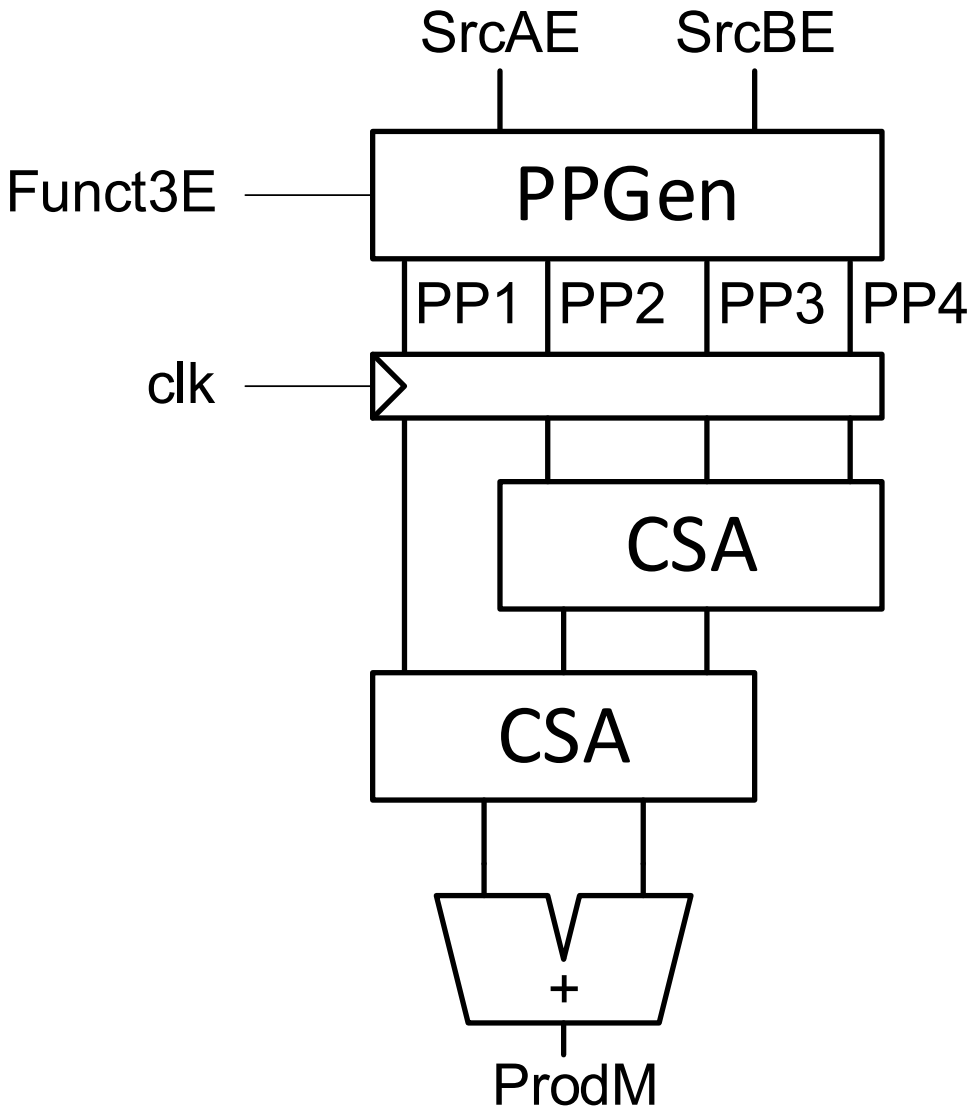


Signed & Unsigned Multiplication

Flavor	*P1*	*P2*	*P3*	*P4*
`mulhu`	P'	$(PX)2^{N-1}$	$(PY)2^{N-1}$	$P_{msb}2^{2N-2}$
`mulh`	P'	$(\overline{PX})2^{N-1}$	$(\overline{PY})2^{N-1}$	$P_{msb}2^{2N-2} + 2^{2N-1} + 2^{N}$
`mulhsu`	P'			<exercise to reader>

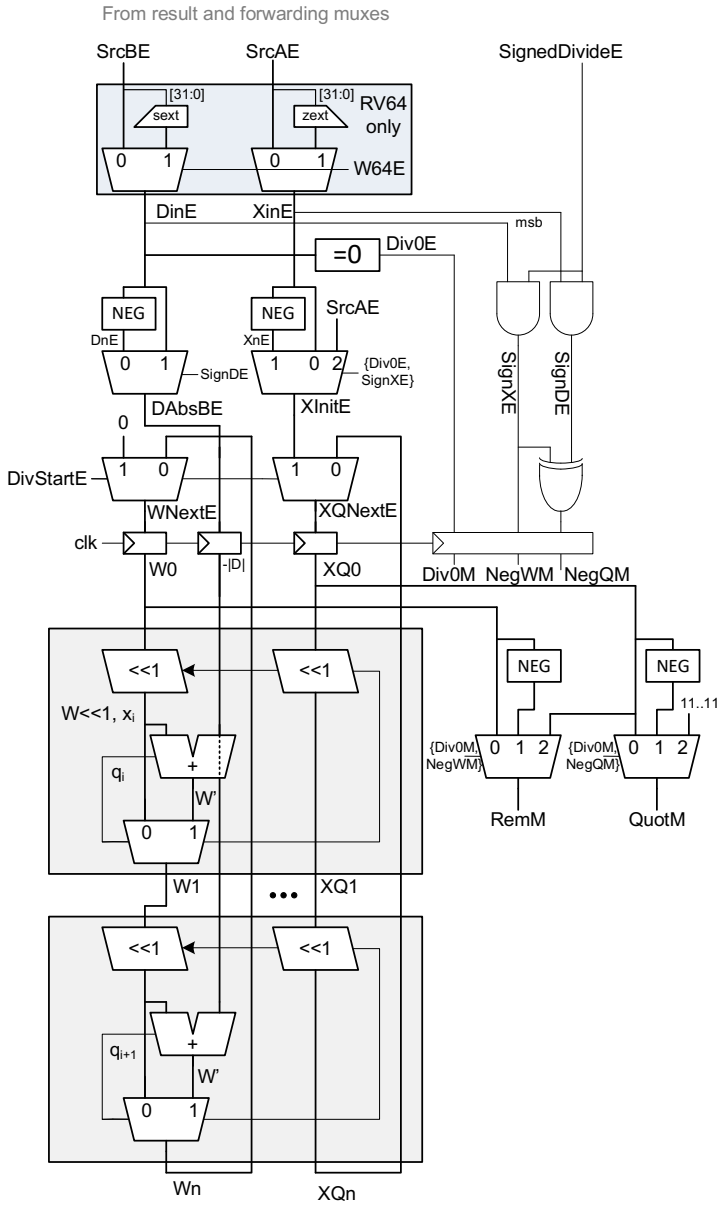
Wally Integer Multiplier Unit

`mul`

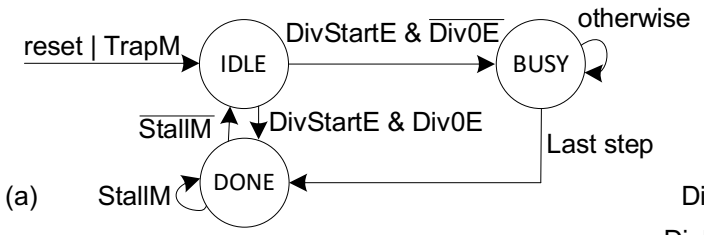


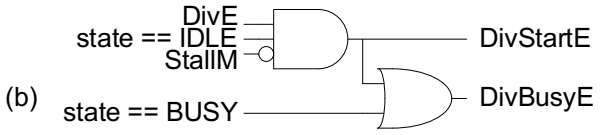
Wally Integer Divider Unit

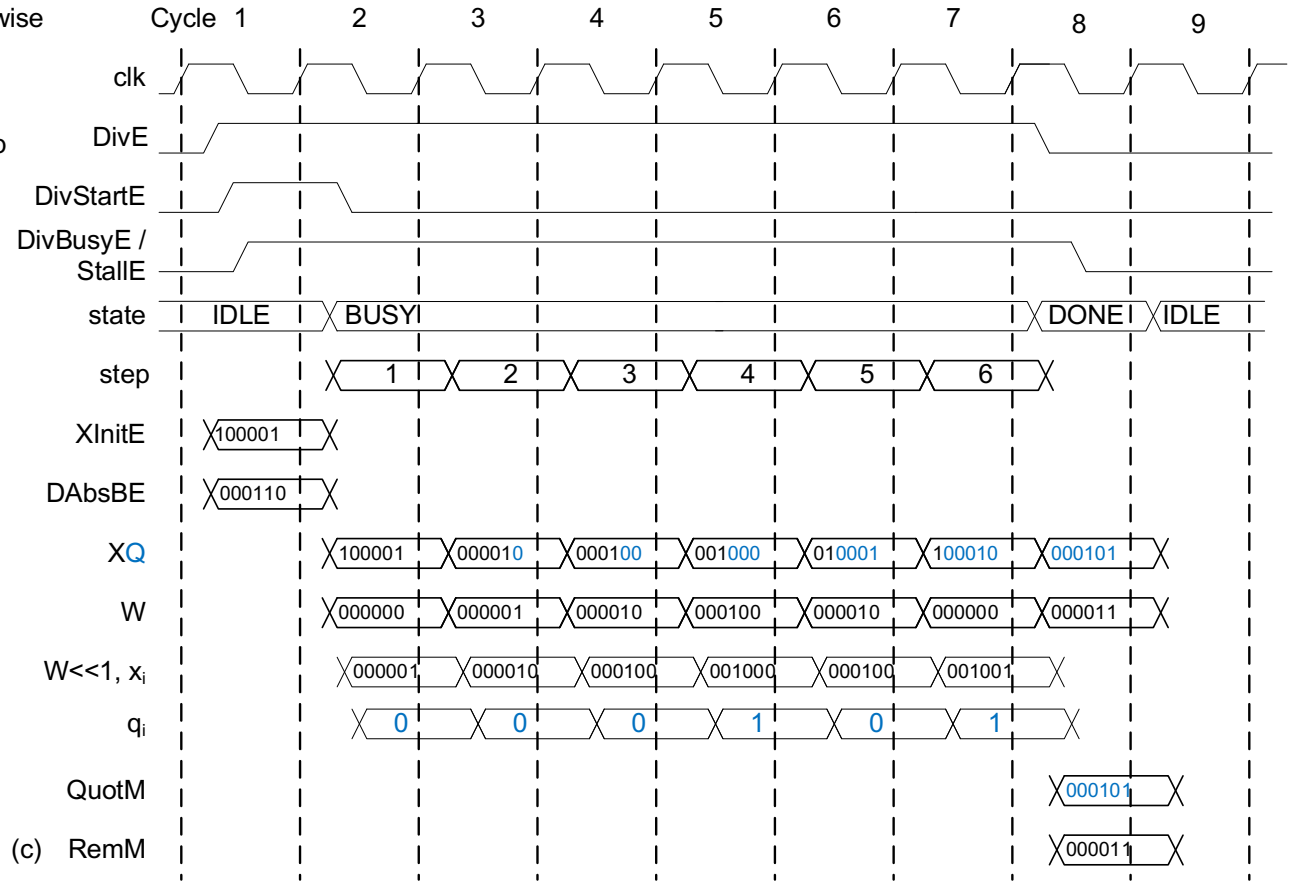
div



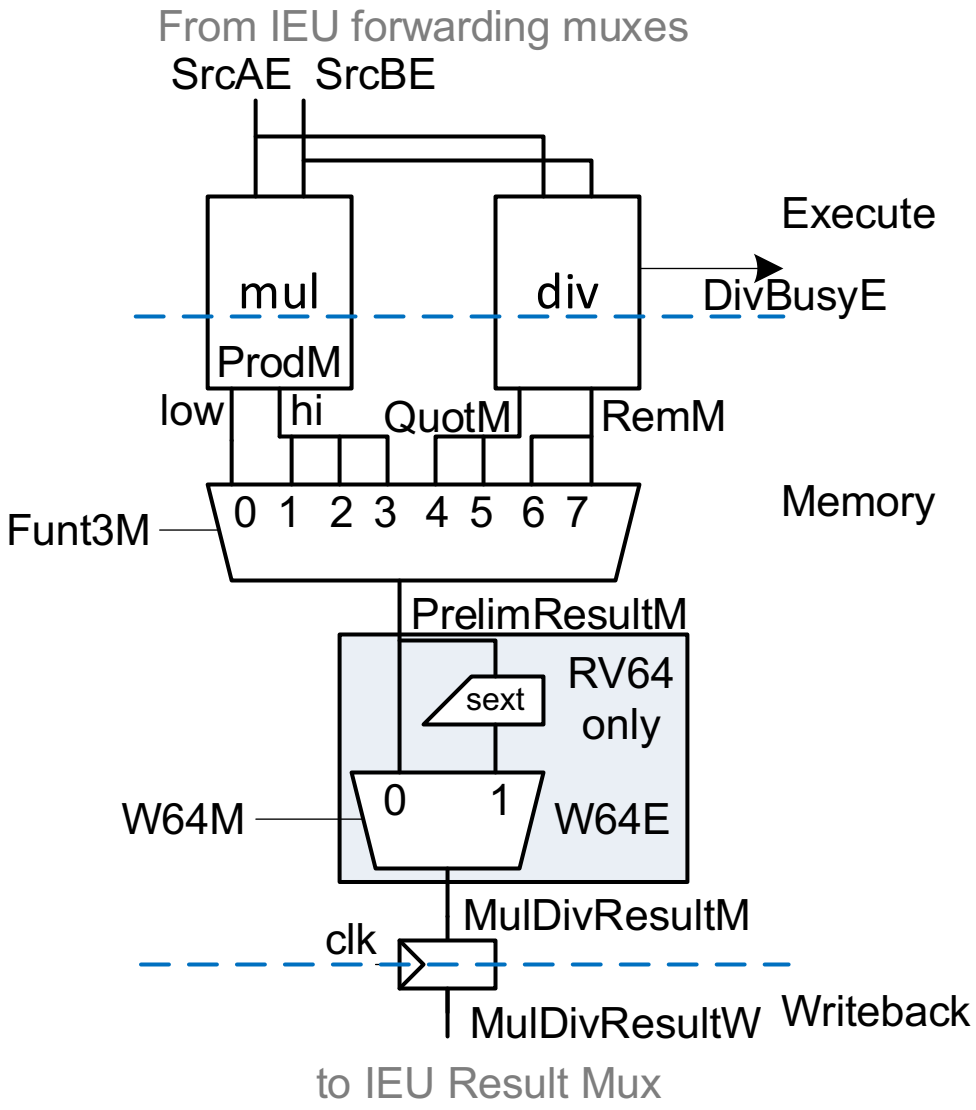
Integer Divider Timing



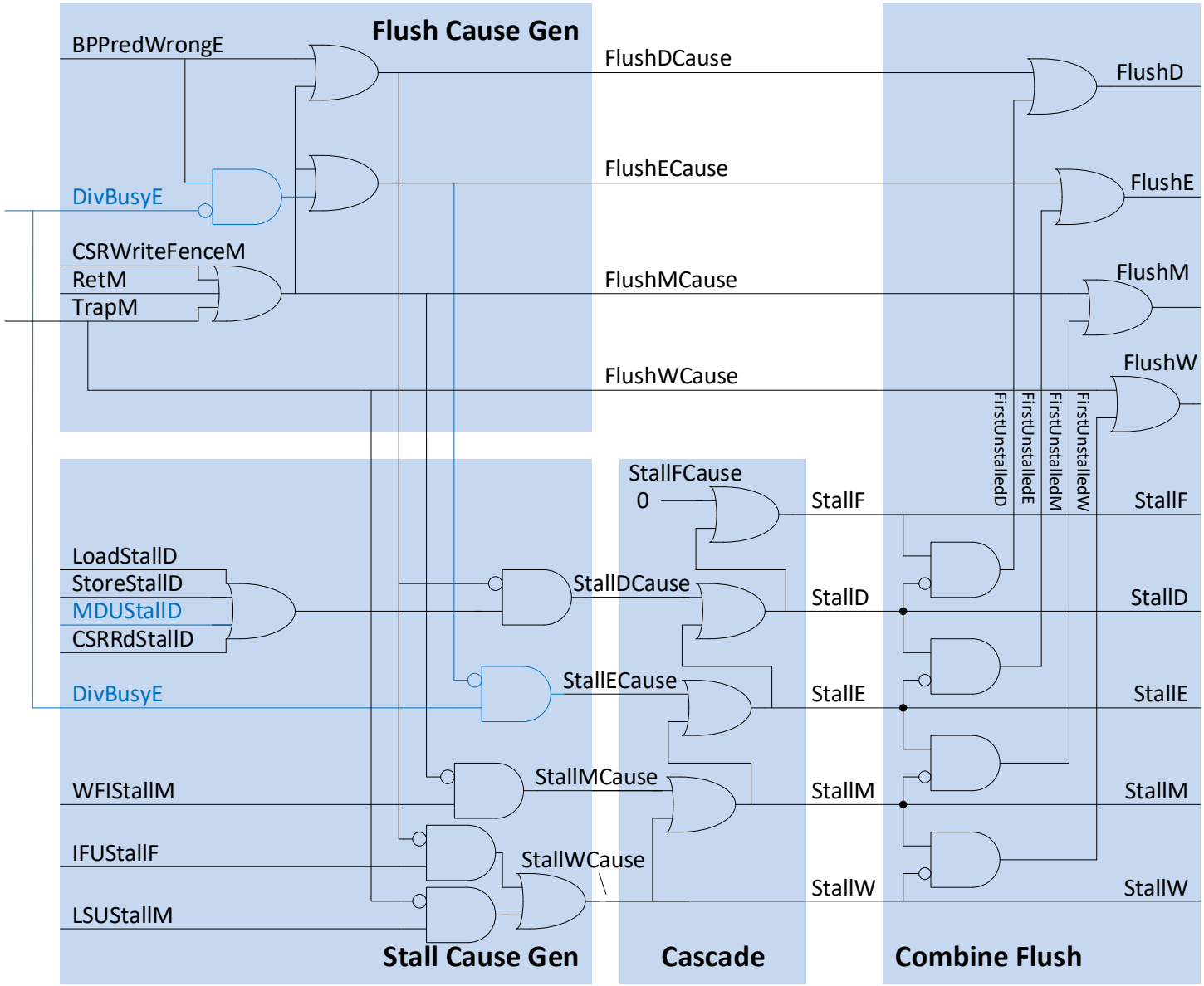




MDU


From IEU forwarding muxes
SrcAE
SrcBE
mul
div
Execute
DivBusyE
ProdM
low
hi
QuotM
RemM
0 1 2 3 4 5 6 7
Funt3M
Memory
PrelimResultM
sext
RV64
only
0
1
W64M
W64E
MulDivResultM
clk
MulDivResultW
Writeback
to IEU Result Mux

Hazard Unit


Flush Cause Gen
BPPredWrongE
DivBusyE
CSRWriteFenceM
RetM
TrapM
FlushDCause
FlushECause
FlushMCause
FlushWCause
FlushD
FlushE
FlushM
FlushW
FirstUnstalledD
FirstUnstalledE
FirstUnstalledM
FirstUnstalledW
StallFCause
0
StallF
StallD
StallE
StallM
StallW
LoadStallD
StoreStallD
MDUStallD
CSRRdStallD
DivBusyE
WFIStallM
IFUStallF
LSUStallM
StallDCause
StallECause
StallMCause
StallWCause
Stall Cause Gen
Cascade
Combine Flush

About these Notes

RISC-V System-on-Chip Design Lecture Notes