ECE 2300
Digital Logic & Computer Organization

Spring 2025

More Timing Analysis
Binary Arithmetic

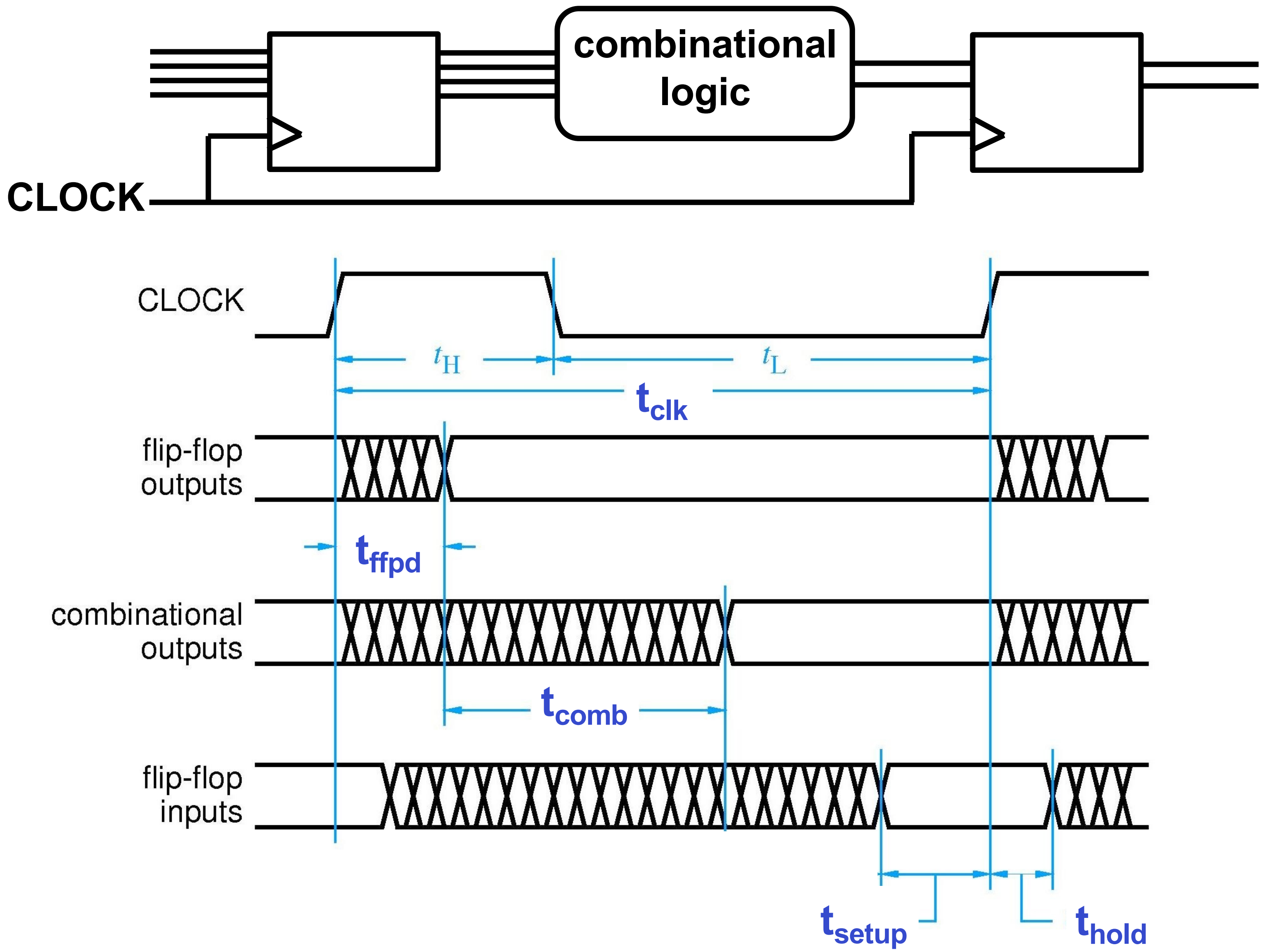
Cornell University

Announcements

- Lab 3 released; Form groups on CMS by Wed

- Lab 2b due tomorrow

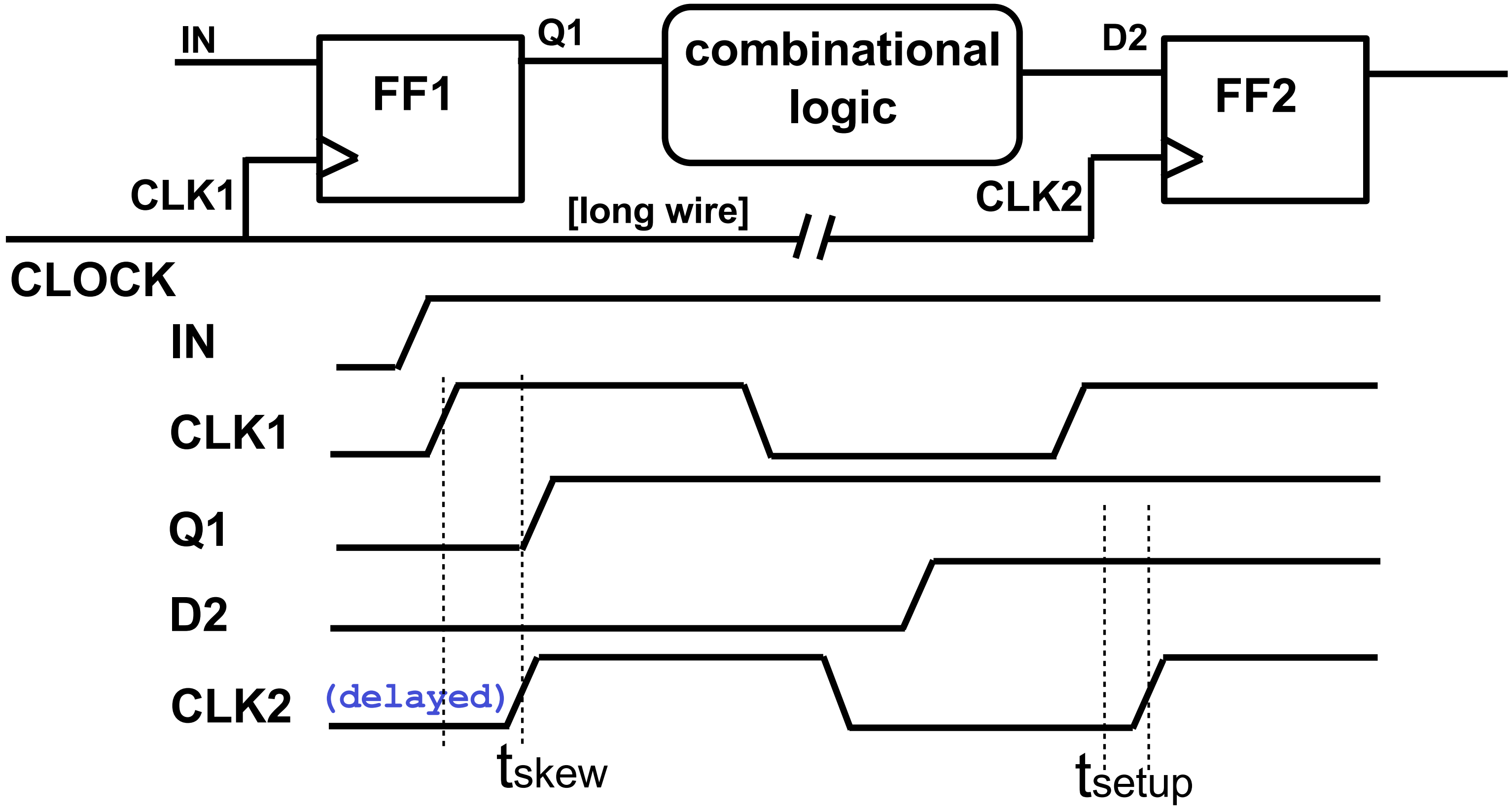
Review: Important Timing Parameters



Timing Analysis Discussion (1)

- To achieve a ***higher clock frequency*** (i.e., smaller cycle time), would you prefer
  1) A smaller or larger combinational delay?
  2) A wider or narrower setup time window?
  3) A wider or narrower hold time window?
  4) A positive or negative clock skew?

Review: Positive Clock Skew



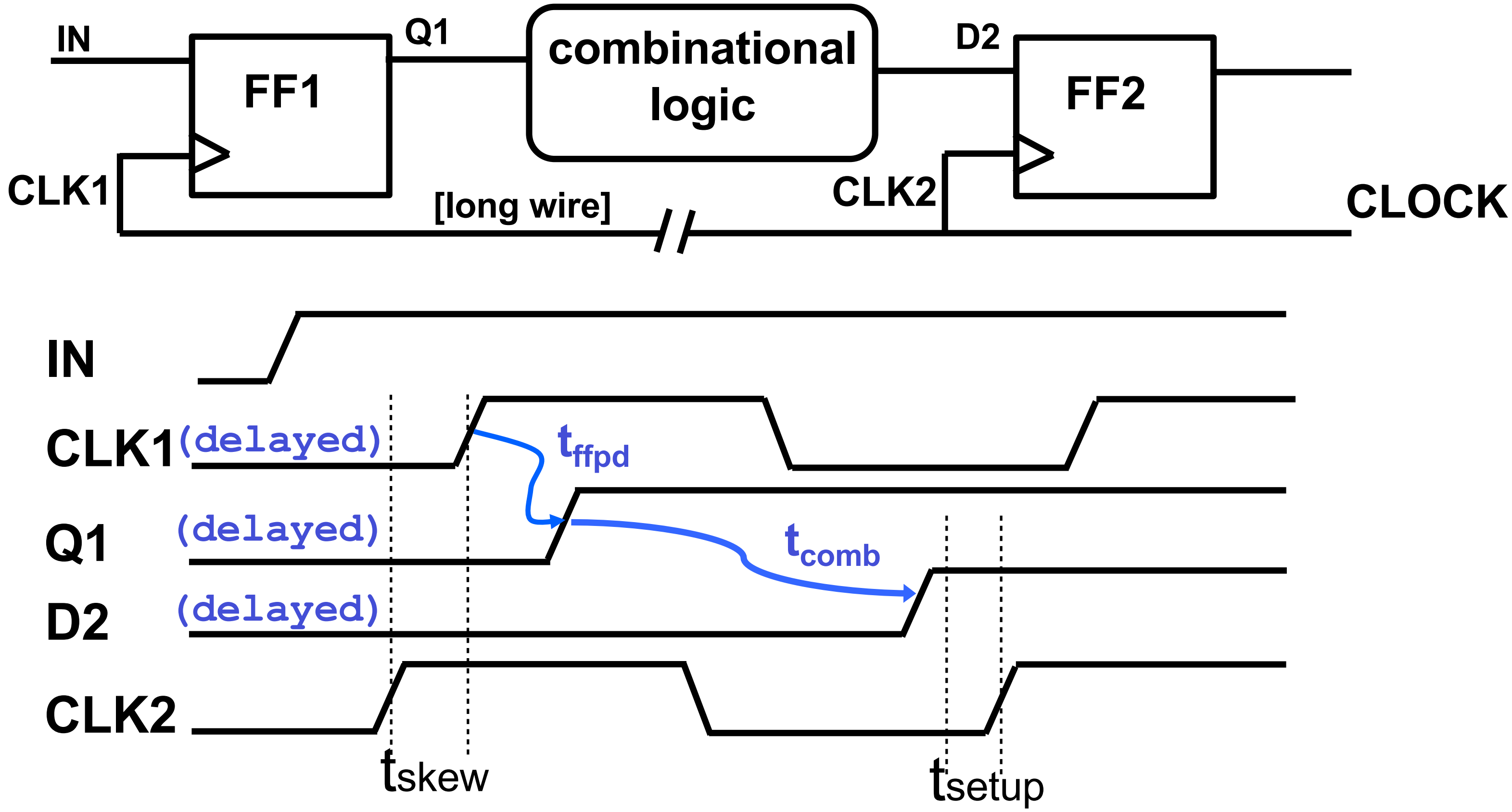
Receiving FF receives clock later than sending FF

$t_{ffpd(max)} + t_{comb(max)} + t_{setup} \leq t_{clk} + t_{skew(min)}$

➔ $t_{ffpd(max)} + t_{comb(max)} + (t_{setup} - t_{skew(min)}) \leq t_{clk}$

(beneficial skew: setup time window effectively narrowed)

Review: Negative Clock Skew



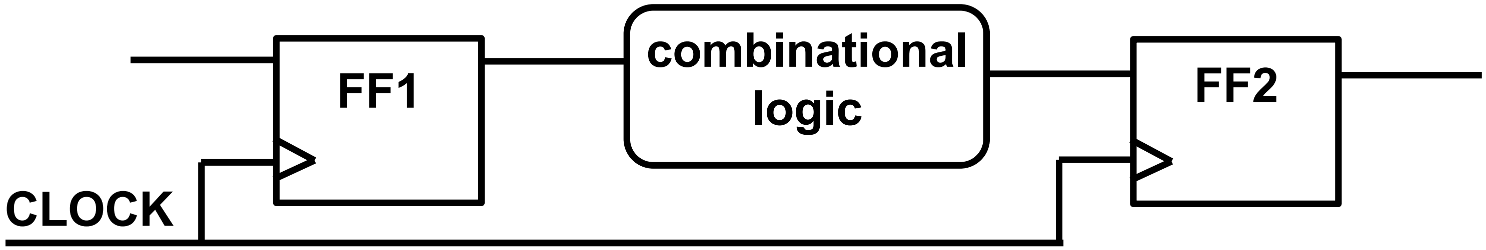
Receiving FF receives clock sooner than sending FF

$$t_{ffpd(max)} + t_{comb(max)} + t_{setup} \leq t_{clk} - t_{skew(max)}$$

➔ $$t_{ffpd(max)} + t_{comb(max)} + (t_{setup} + t_{skew(max)}) \leq t_{clk}$$

(harmful skew: setup time window effectively widened)

Recap: Avoiding Hold Time Violation

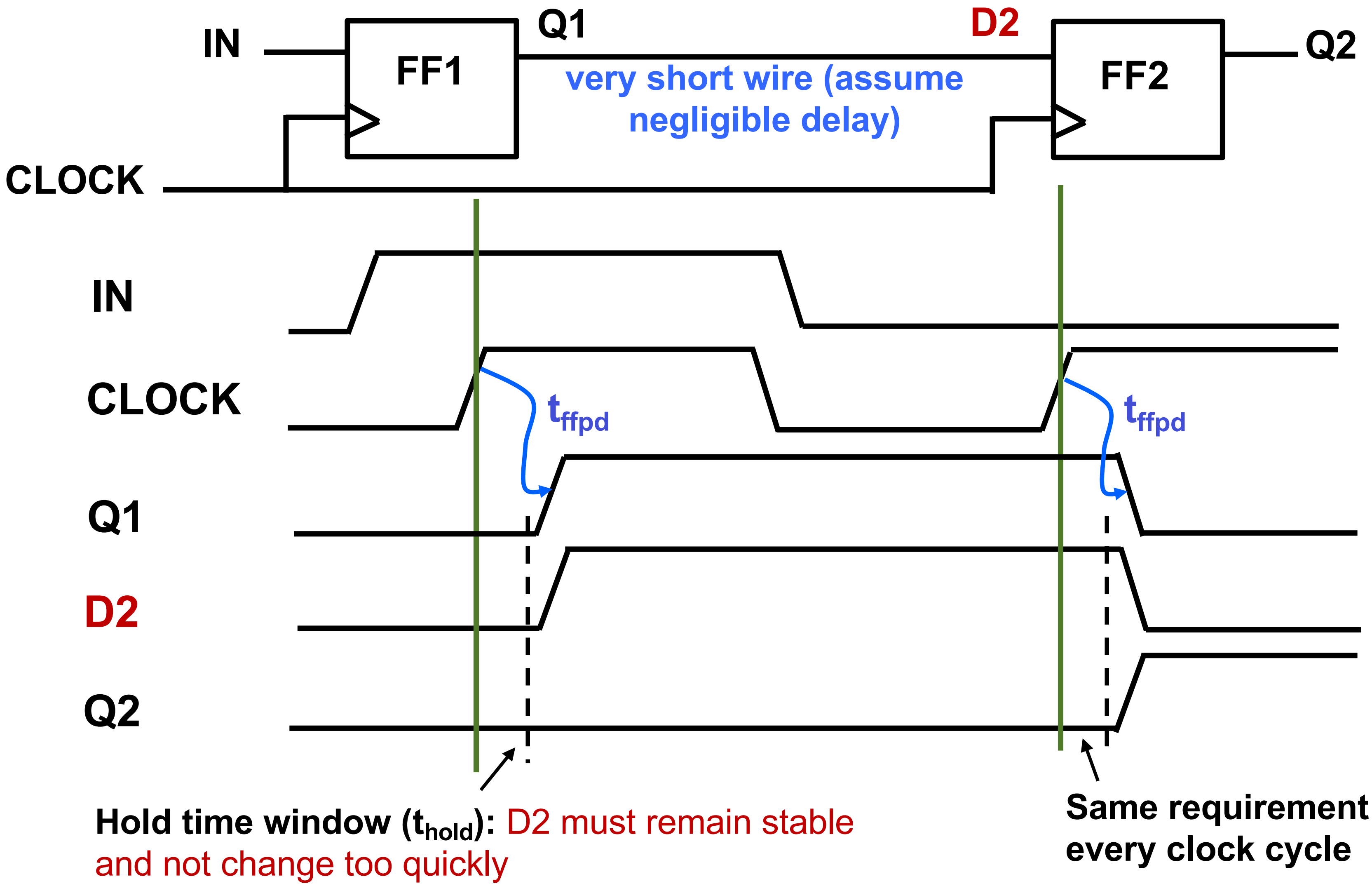


- FF input must remain stable after the triggering edge by at least t_{hold} amount of time
  - Otherwise, the receiving flip-flop may be contaminated with an unexpected value

- Need to consider minimum propagation delays (the shortest timing path) for hold time calculations

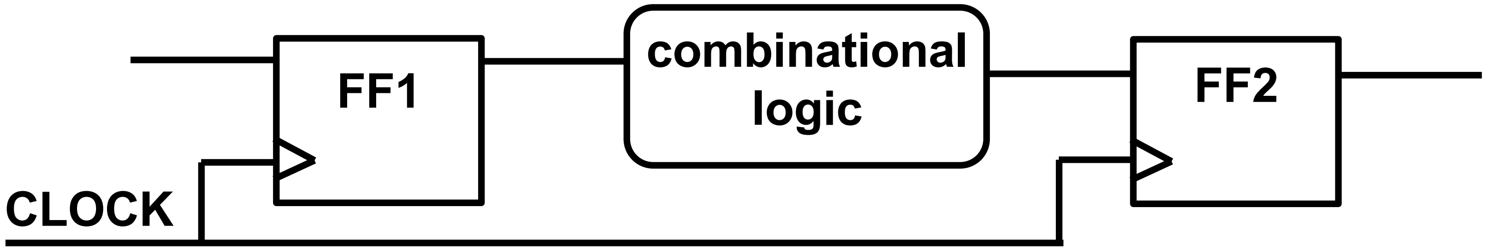
$$t_{ffpd(min)} + t_{comb(min)} \geq t_{hold}$$

Example: Hold Time Constraint



$$t_{ffpd(min)} + t_{comb(min)} = t_{ffpd(min)} + 0 \geq t_{hold}$$

Example: Hold Time Calculations



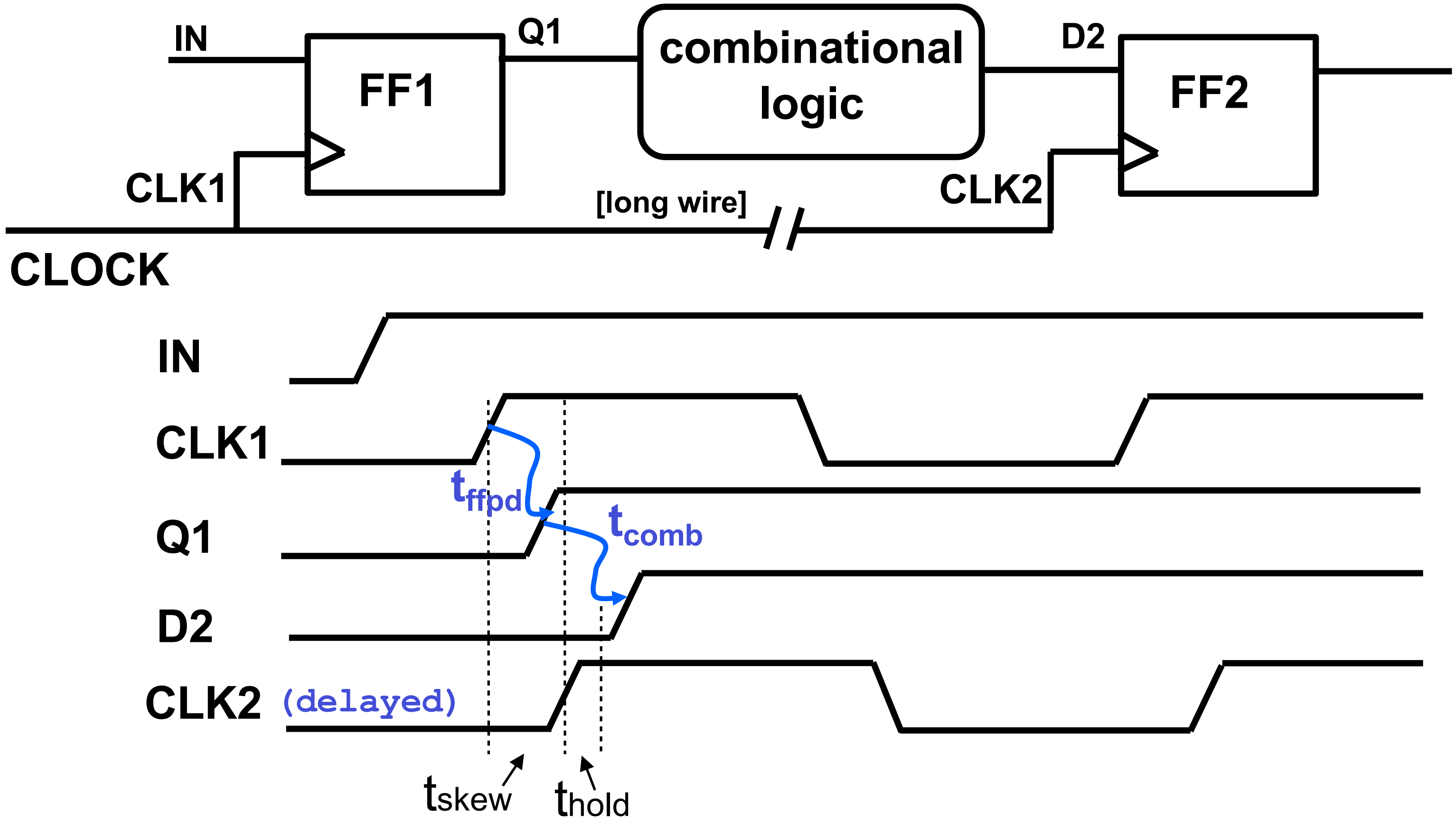
	Prop Delay (ns)		Setup Time (ns)	Hold Time (ns)
	min	max		
FF	1	2	3	2
Comb	2	7	-	-

- Hold time at FF2 met?

Timing Analysis Discussion (2)

- To *avoid hold time violation*, would you prefer
  1) A smaller or larger combinational delay?
  2) A wider or narrower setup time window?
  3) A wider or narrower hold time window?
  4) A positive or negative clock skew?

Hold Time With Positive Clock Skew



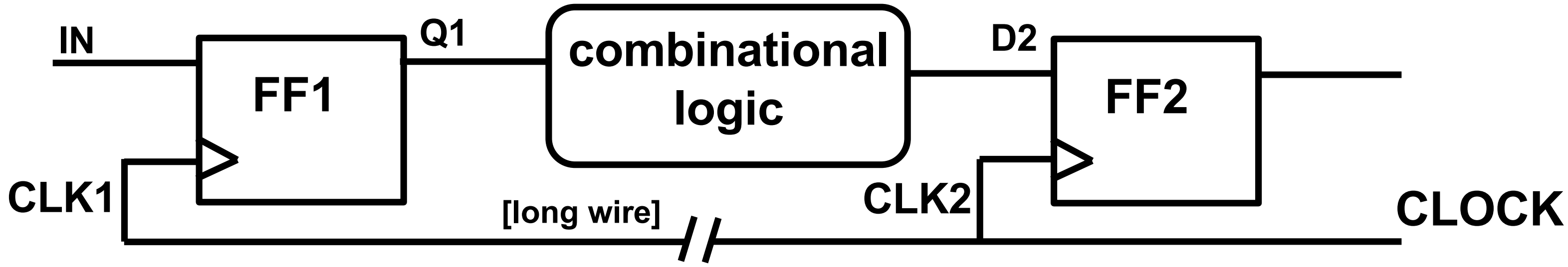
Receiving FF receives clock later than sending FF

$$t_{ffpd(min)} + t_{comb(min)} \geq t_{hold} + t_{skew(max)}$$

(hold time window
effectively widened)

Harmful skew for meeting hold time constraint

Hold Time With Negative Clock Skew



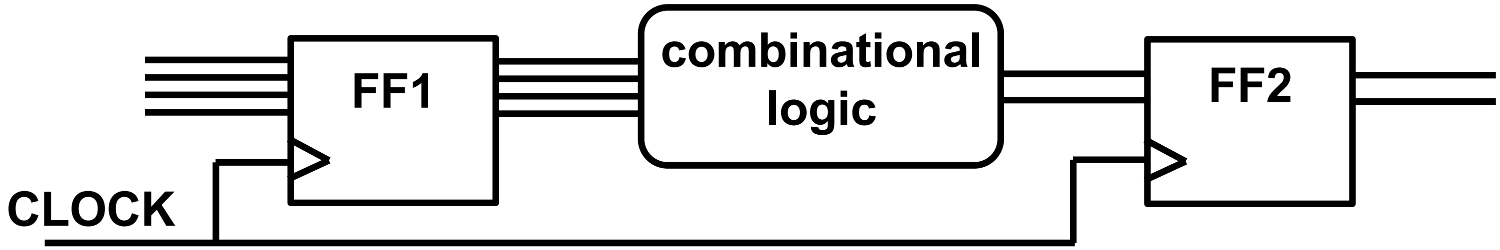
What if receiving FF receives clock sooner than sending FF?

$$t_{ffpd(min)} + t_{comb(min)} \geq t_{hold} - t_{skew(min)}$$

(hold time window effectively narrowed)

Beneficial skew for meeting hold time constraint

Example: Hold Time Analysis with Clock Skew

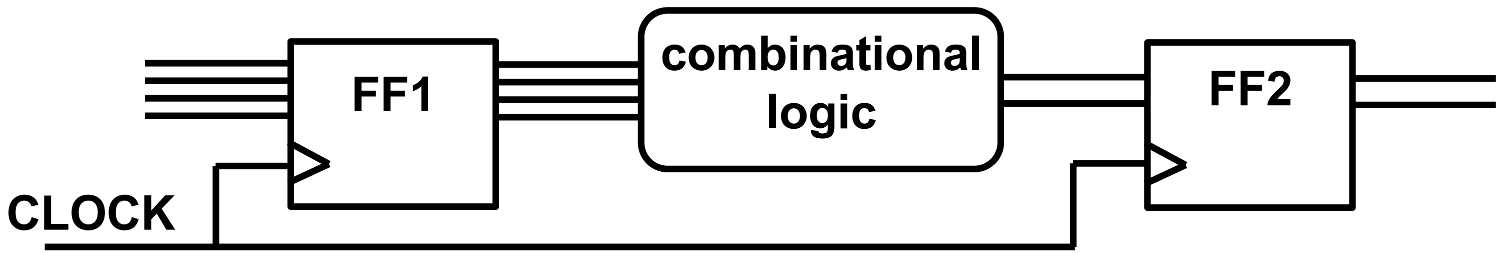


Clock may arrive at FF2 <u>up to 2ns</u> later than FF1

	Prop Delay (ns)		Setup Time (ns)	Hold Time (ns)
	min	max		
FF	1	3	3	2
Comb	3	7	-	-

- Hold time at FF2 met?

Example: Hold Time Analysis with Clock Skew



Clock may arrive at FF2 up to 2ns later than FF1

	Prop Delay (ns)		Setup Time (ns)	Hold Time (ns)
	min	max		
FF	1	3	3	2
Comb	3	7	-	-

- Hold time at FF2 met?

$t_{ffpd(min)} + t_{comb(min)} >= t_{hold} + t_{skew(max)}$

$1 + 3 >= 2 + 2$

The hold time constraint is met

Course Content

- Binary numbers and logic gates
- Boolean algebra and combinational logic
- Sequential logic and state machines
- Clocking and timing analysis

- Binary arithmetic
- Memories

- Instruction set architecture
- Processor organization
- Caches and virtual memory
- Input/output

Unsigned Binary Integers

- An n-bit unsigned number represents 2^n integer values
  - Range is from 0 to 2^n-1
- For the unsigned binary number $b_{n-1}b_{n-2}\ldots b_1b_0$, the decimal number is

$$D = \sum_{i=0}^{n-1} b_i \bullet 2^i$$

2^2	2^1	2^0	value
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Unsigned Binary Addition

- **Just like base-10**
  - **Add from right to left, propagating carry**

	carry	
10010 (18)	10010 (18)	01111 (15)
+ 01001 (9)	+ 01011 (11)	+ 00011 (3)
10010 (27)	(29)	10010 (18)

Signed Magnitude Representation

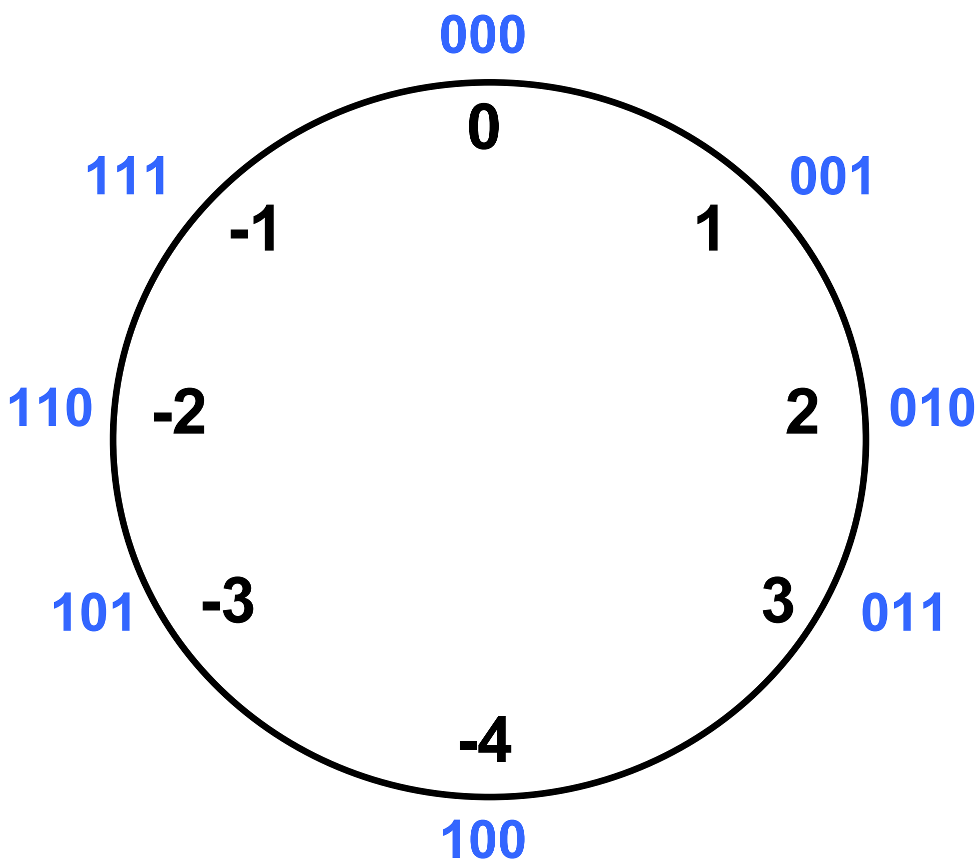
- Most significant bit is used as a sign bit
  - Sign bit of 0 for positive (001 = 1)
  - Sign bit of 1 for negative (101 = -1)
- *Range* is from $-(2^{n-1}-1)$ to $(2^{n-1}-1)$ for an n-bit number

- Two representations for zero (+0 and -0)

- Does ordinary binary addition still work?

$$
\begin{array}{rrl}
 & 001 & (1) \\
+ & 101 & (-1) \\
\hline
 & 110 & (\text{not } 0)
\end{array}
$$

Another Way to Encode Signed Binary Numbers



Two's Complement Representation (2's C)

- A (slightly) different positional encoding: **MSB has weight -2^{n-1}**
  - n is the bitwidth
  - For the 2's C binary number $b_{n-1}b_{n-2}\ldots b_1b_0$, the decimal is

$$D = -b_{n-1} \bullet 2^{n-1} + \sum_{i=0}^{n-2} b_i \bullet 2^i$$

- Range of an n-bit number: -2^{n-1} through $2^{n-1}-1$
  - Positive numbers and zero are same as unsigned binary representation
  - Most negative number (namely, -2^{n-1}) has no positive counterpart

-2^2	2^1	2^0	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	-4
1	0	1	-3
1	1	0	-2
1	1	1	-1

Two's Complement Addition

- Procedure for addition is the same as unsigned addition regardless of the signs of the numbers

$$
\begin{array}{rrl}
 & \mathbf{001} & (1) \\
+ & \underline{\mathbf{111}} & (-1) \\
 & \mathbf{000} & (0)
\end{array}
$$

Negating a 2'C Number

- To get two's complement negative notation of an integer
  - Flip every bit first
  - Then add one

```
    001   (1)
    110   (1's comp)
+     1
    ---
    111   (-1)
```

```
    01001   (9)
    10110   (1's comp)
+       1
    -----
    10111   (-9)
```

$$-X = (X' + 1)$$

2's C Negation Shortcut

- To get -X
  - Copy bits from right to left up to and including the first "1"
  - Flip remaining bits to the left

```
  011010000
  100101111   (1's comp)
+         1
  ---------
  100110000
```

```
0110|10000
(flip) | (copy)
1001|10000
```

Converting Binary (2's C) to Decimal

1. If MSB = 1, take two's complement to get a positive number
2. Add powers of 2 for bit positions that have a "1"
3. If original number was negative, add a minus sign

$$
\begin{aligned}
X &= 11100110_{two} \\
-X &= 00011010 \\
&= 2^4+2^3+2^1 = 16+8+2 \\
&= 26_{ten} \\
X &= -26_{ten}
\end{aligned}
$$

Assuming 8-bit 2's complement numbers

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Converting Decimal to Binary (2's C)

First Method: *Division*

1. **Change to nonnegative decimal number**
2. **Divide by two – remainder is least significant bit**
3. **Keep dividing by two until answer is zero, recording remainders from right (LSB) to left**
4. **<u>Append a zero as the MSB</u>; if original number X was negative, return X'+1**

$X = 104_{ten}$

104/2	=	52 r0	*bit 0 = 0*
52/2	=	26 r0	*bit 1 = 0*
26/2	=	13 r0	*bit 2 = 0*
13/2	=	6 r1	*bit 3 = 1*
6/2	=	3 r0	*bit 4 = 0*
3/2	=	1 r1	*bit 5 = 1*
1/2	=	0 r1	*bit 6 = 1*

$X = 01101000_{two}$

Converting Decimal to Binary (2's C)

Second Method: *Subtract Powers of Two*

1. Change to nonnegative decimal number
2. Subtract largest power of two
   less than or equal to number
3. Put a one in the corresponding bit position
4. Keep subtracting until result is zero
5. <u>Append a zero as MSB</u>;
   if original was X negative, return X'+1

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

$X = 104_{ten}$

104 - 64	=	40	*bit 6 = 1*
40 - 32	=	8	*bit 5 = 1*
8 - 8	=	0	*bit 3 = 1*

$X = 01101000_{two}$

Fixed Size Representation

- Microprocessors usually represent numbers as fixed size n-bit values
- Result of adding two n-bit integers is stored as n bits
- Integers are typically 32 or 64 bits (*words*)
  - 4 or 8 *bytes* (1 *byte* = 8 bits)

Fixed Size Addition

- Examples with n = 4

	2	0010
+	3	0011
	5	0101

	2	0010
+	-3	1101
	-1	1111

	-2	1110
+	6	0110
	4	0100

	-2	1110
+	-6	1010
	-8	1000

	7	0111
+	6	0110
	-3	1101

	-7	1001
+	-4	1100
	5	0101

Something went wrong!

Overflow

- If operands are too big, sum cannot be represented as *n*-bit 2's complement number

01000 (8)		11000	(-8)
+ 01001	(9)	+ 10111	(-9)
10001	(-15)	01111	(+15)

- Overflow occurs if
  - Signs of both operands are the same, and
  - Sign of sum is different
- Another test (easy to do in hardware)
  - Carry into MSB does not equal carry out

Exercise: Would Overflow Occur?

$$\begin{array}{r} 011100 \\ +\ 010101 \\ \hline \end{array}$$

Next Class

More Binary Arithmetic

ALU

(H&H 5.1-5.2.4)