Vietnam National University
Integrated Circuit Design Research and Education Center

Hardware Description Language
Verilog HDL

Multiple Clock Domains

D^+ Training Program

Lecturer: Đỗ Ngọc Quỳnh
E-mail:dnquynh@icdrec.edu.vn

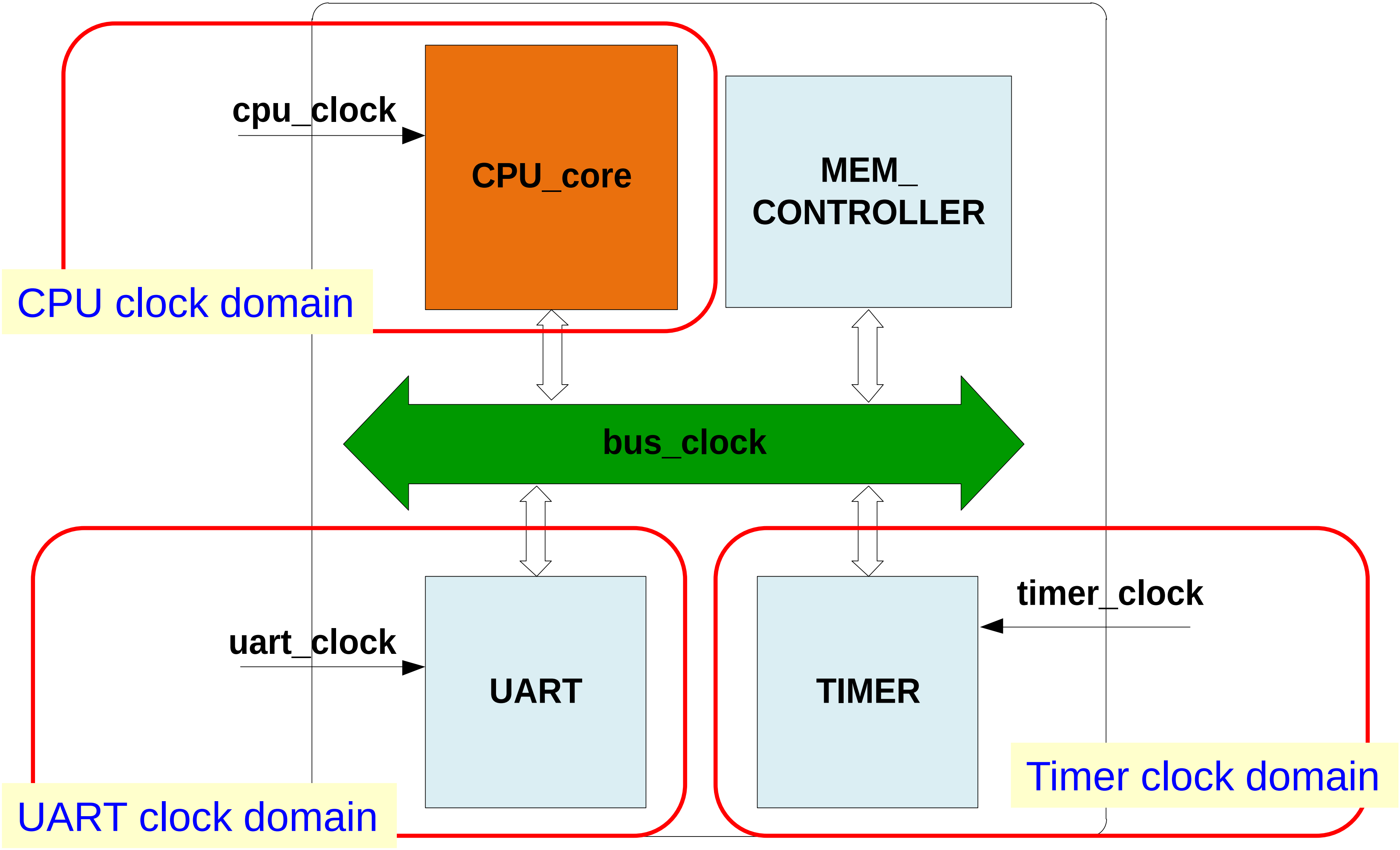
Contents

- Introduction
- Clock Domain Crossing Issues
- Synchronous clock domain crossings examples
- Exercise – Dual clock FIFO design

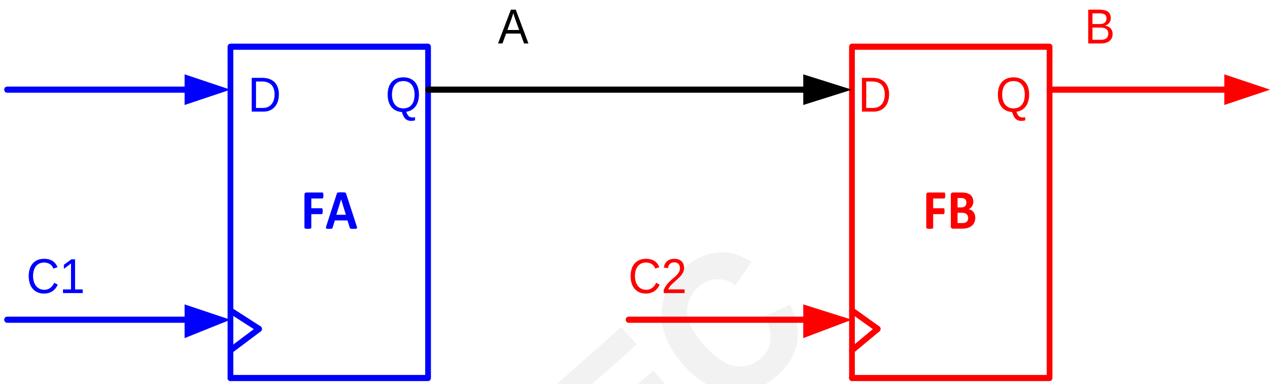
Multiple Clock Domains

INTRODUCTION

Introduction



Two clock domains

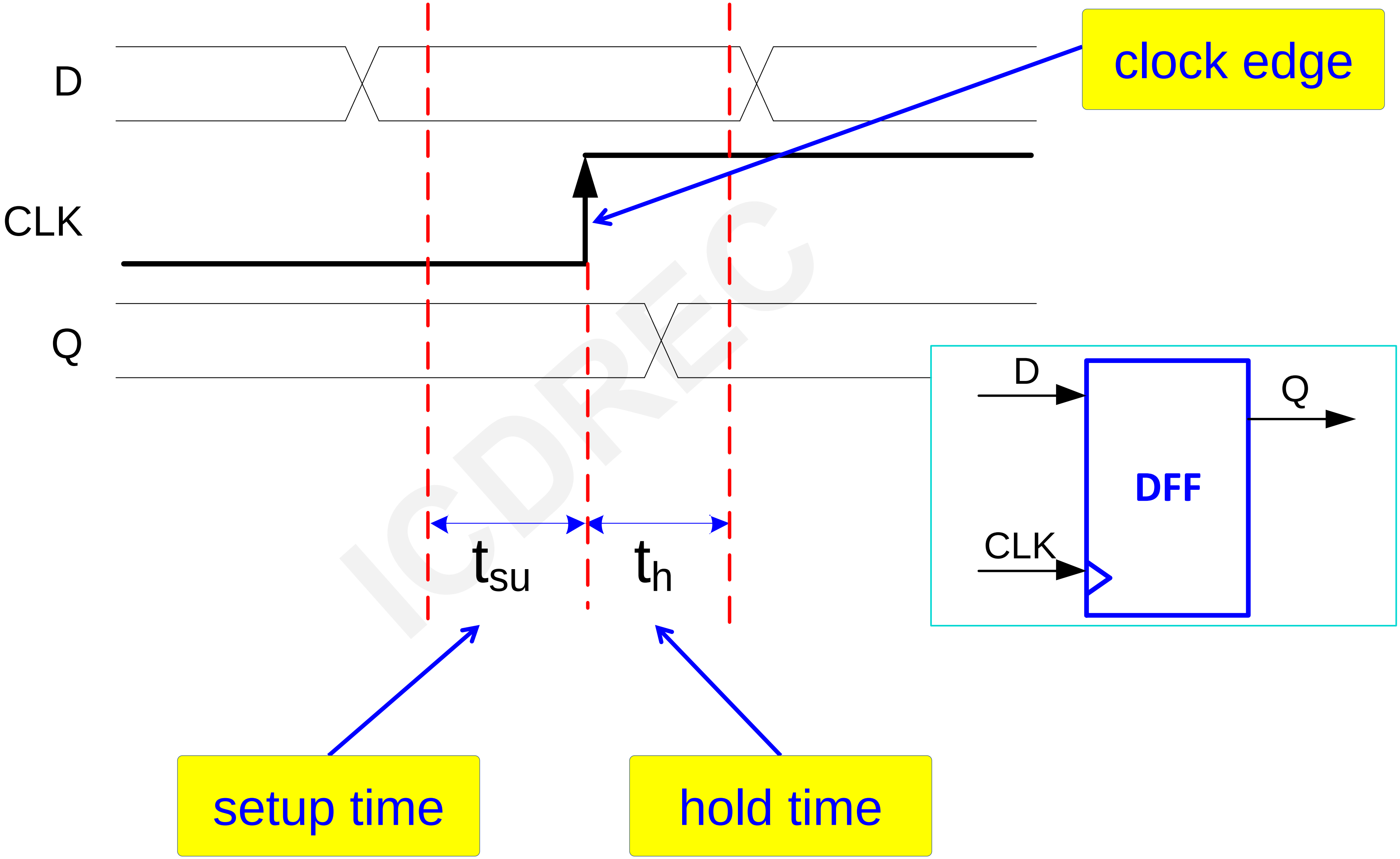


- Signal A is output of the C1 clock domain and be captured by the C2 clock domain.
- Depending on the relationship between the two clocks, there could be different types of problems in transferring data from the source clock to the destination clock.
- Along with that, the solutions to those problems can also be different.

Multiple Clock Domains

CLOCK DOMAIN CROSSING ISSUES

FF Setup & hold times (review)



clock edge

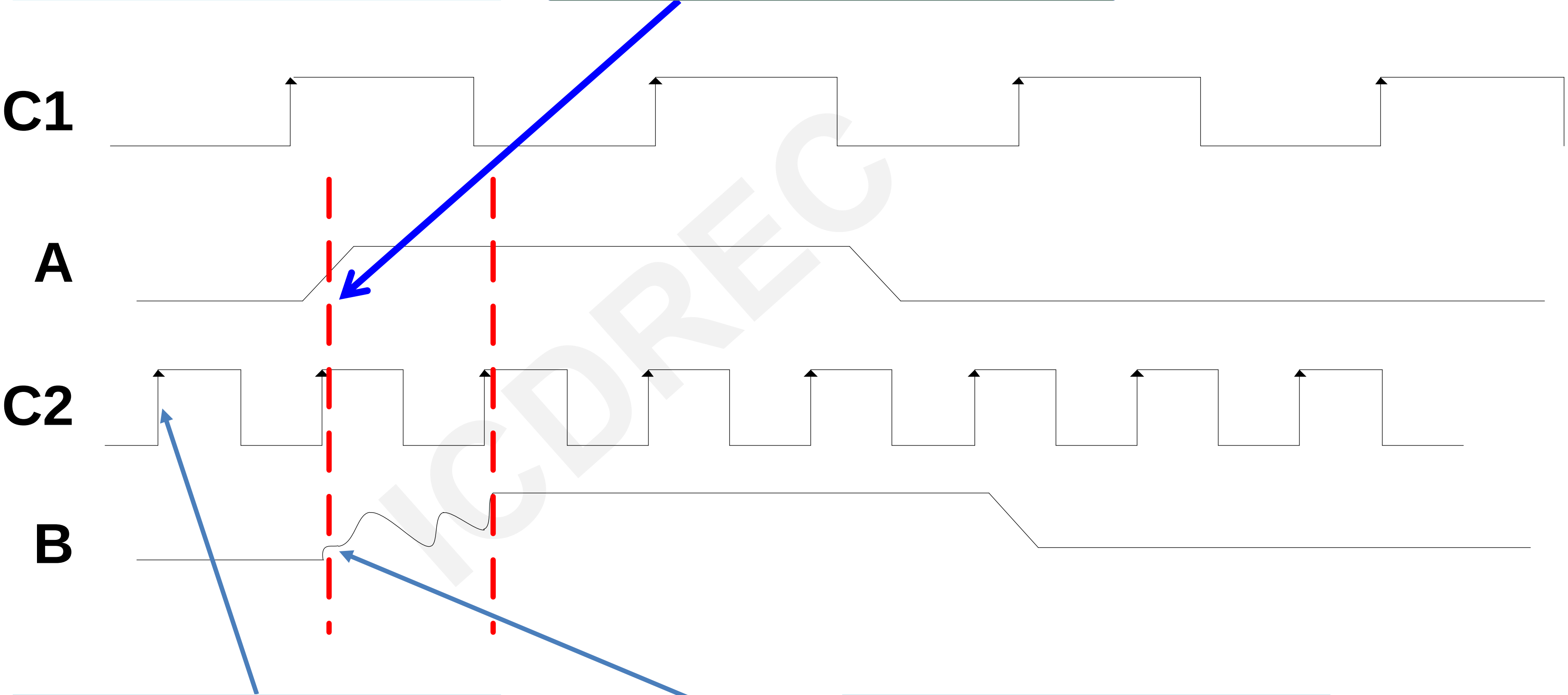
setup time

hold time

Data change at clock edge

A generated by C1

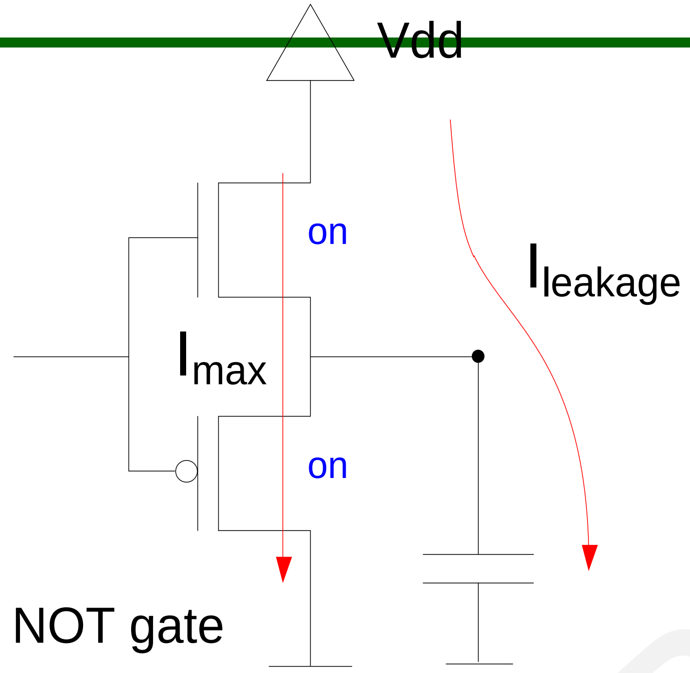
setup or hold violation

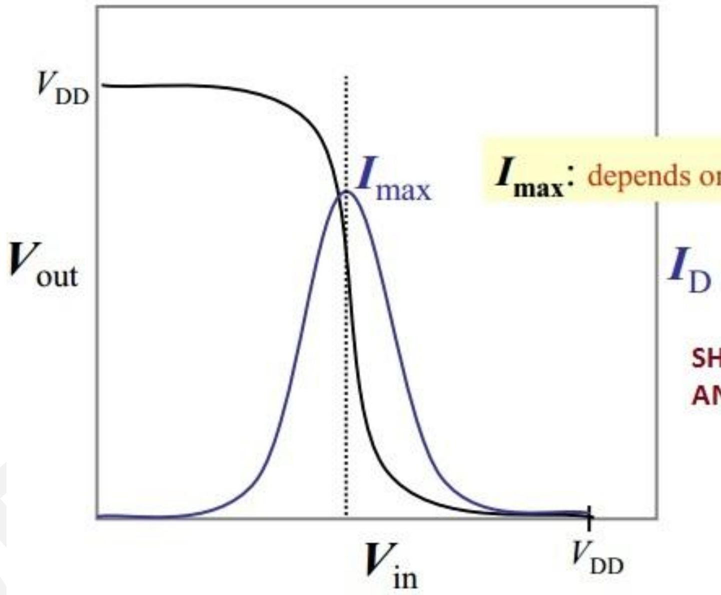


A sampled by C2

Clock signal is initially metastable

Metastability effect





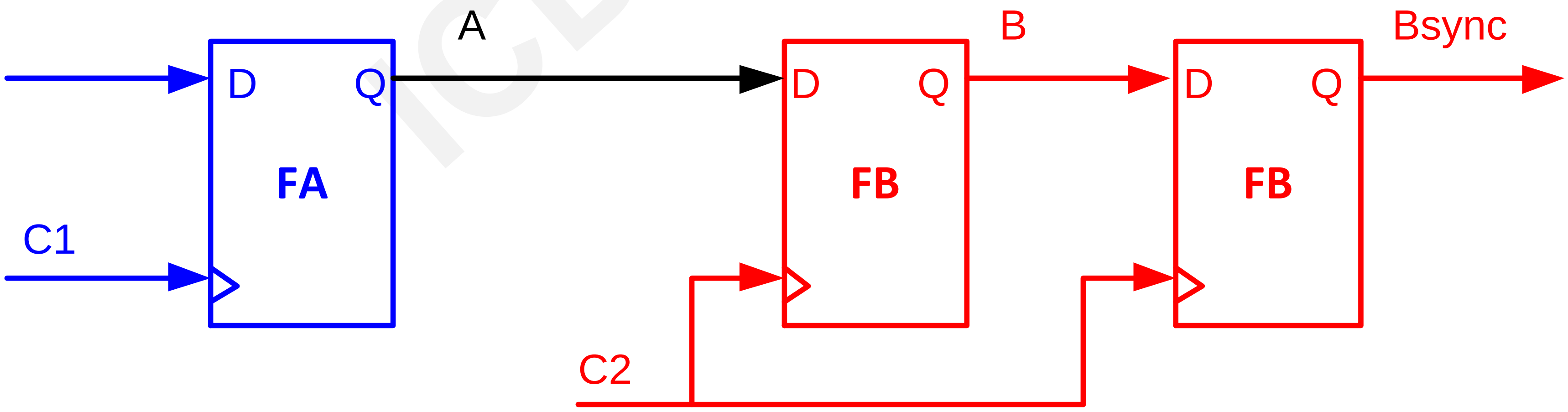
I_{max}: depends on saturation current of devices

SHORT CIRCUIT CURRENT IN AN INVERTER

- If the unstable data is fed to several other places in the design, it may lead to a high current flow and even chip burnout in the worst case.
- The design can enter into an unknown functional state, leading to functional issues in the design.
- The propagation delay could be high leading to timing issues.

Metastable solution

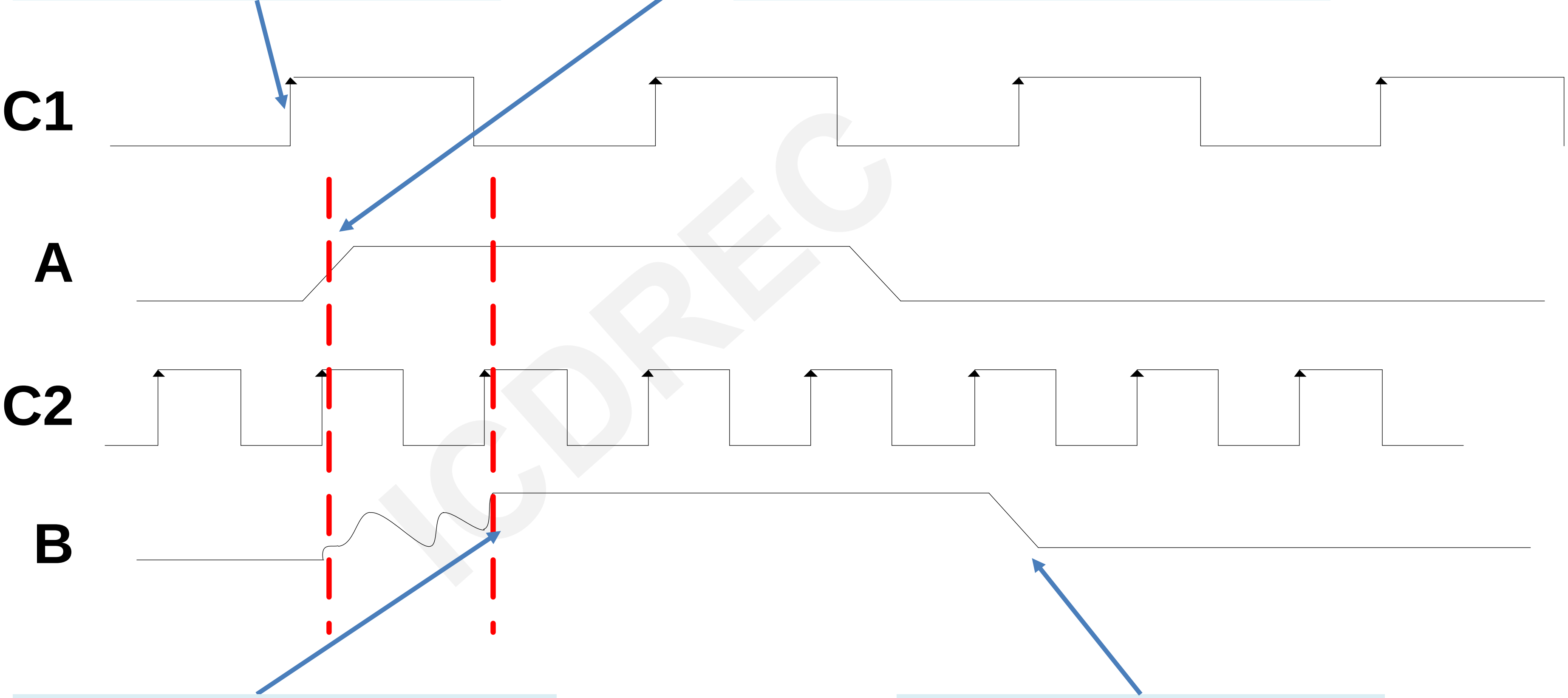
- Metastability problems can be avoided by adding special structures known as synchronizers in the destination domain. The synchronizers allow sufficient time for the oscillations to settle down and ensure that a stable output is obtained in the destination domain. A commonly used synchronizer is a multi-flop synchronizer.



Data Loss

A generated by C1

A change very closely with C2 posedge

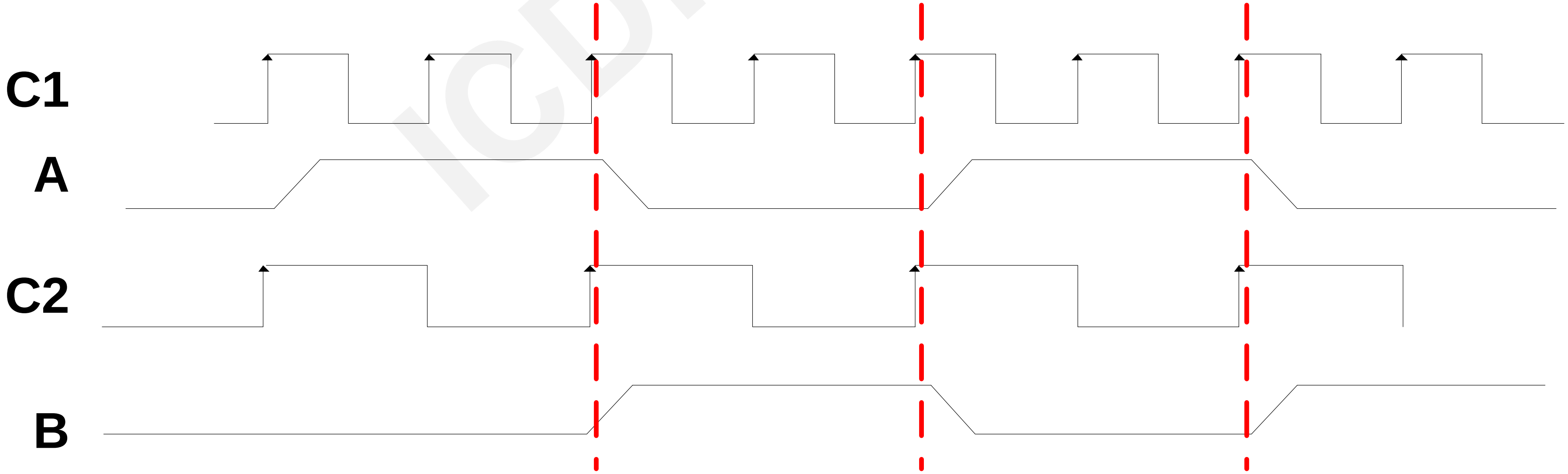


Data capture in 2nd clock cycle

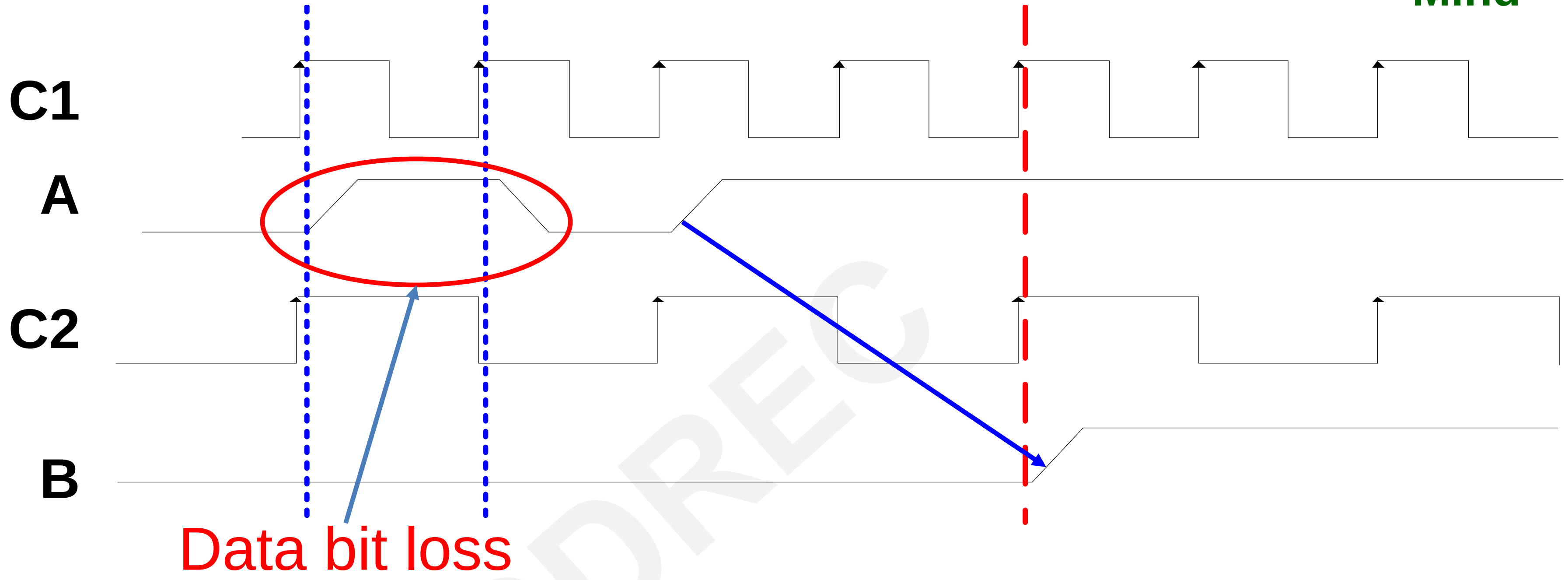
Data capture in 1nd clock cycle

No loss data example

- Clock C1 is twice as fast as the destination clock C2 and there is no phase difference between the two clocks.
- Assume that the input data sequence “A” generated on the positive edge of clock C1 is “00110011”. The data B captured on the positive edge of clock C2 will be “0101”.
- All the transitions on signal A are captured by B, the data is not lost.

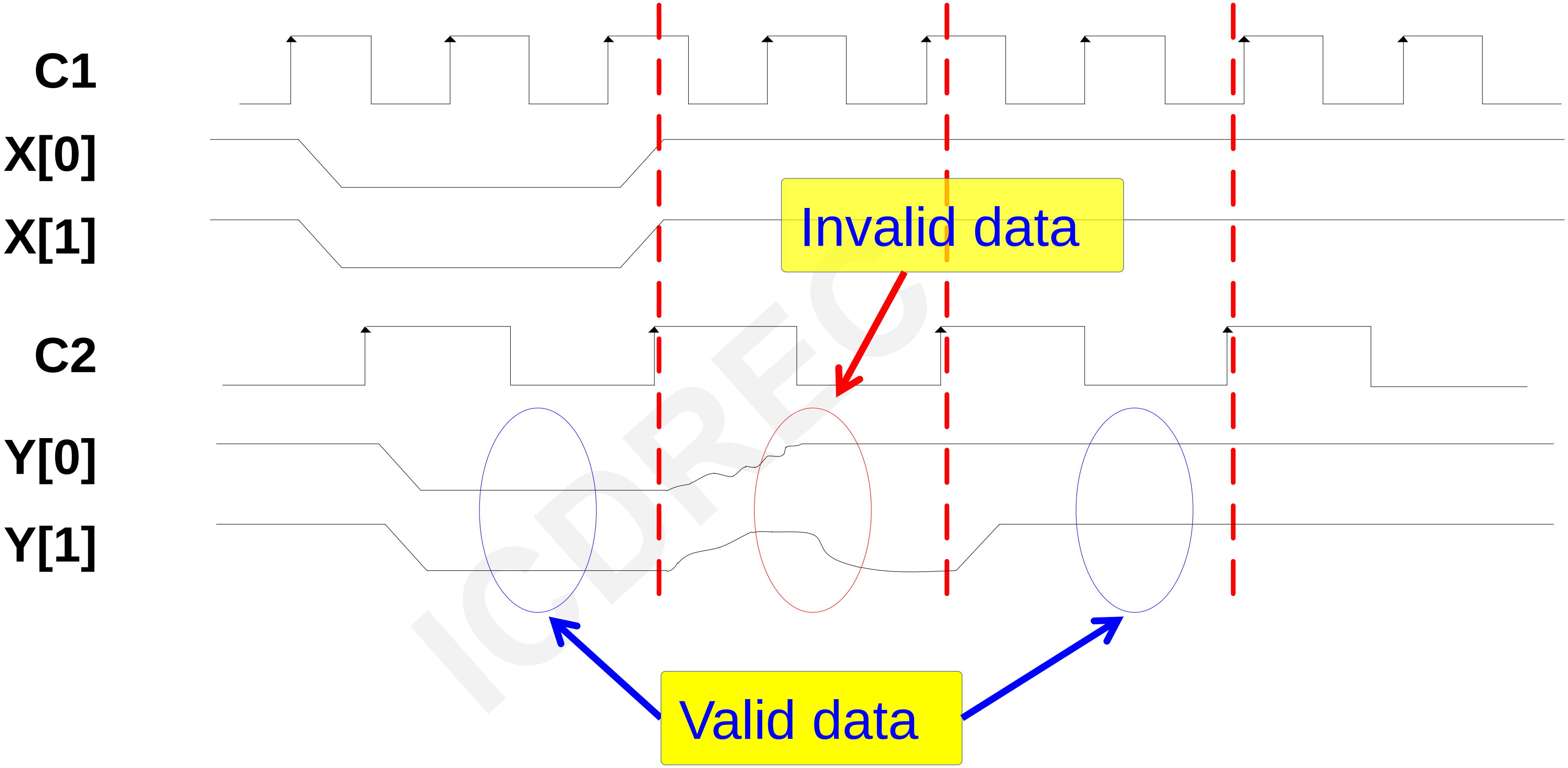


Data bit loss



- The data change in the C1 domain must held long enough to be properly captured in the C2 domain
- After every transition on A, at least one C2 clock edge should arrive where there is no setup or hold violation so that the source data is captured properly in the destination domain.

Data Incoherency



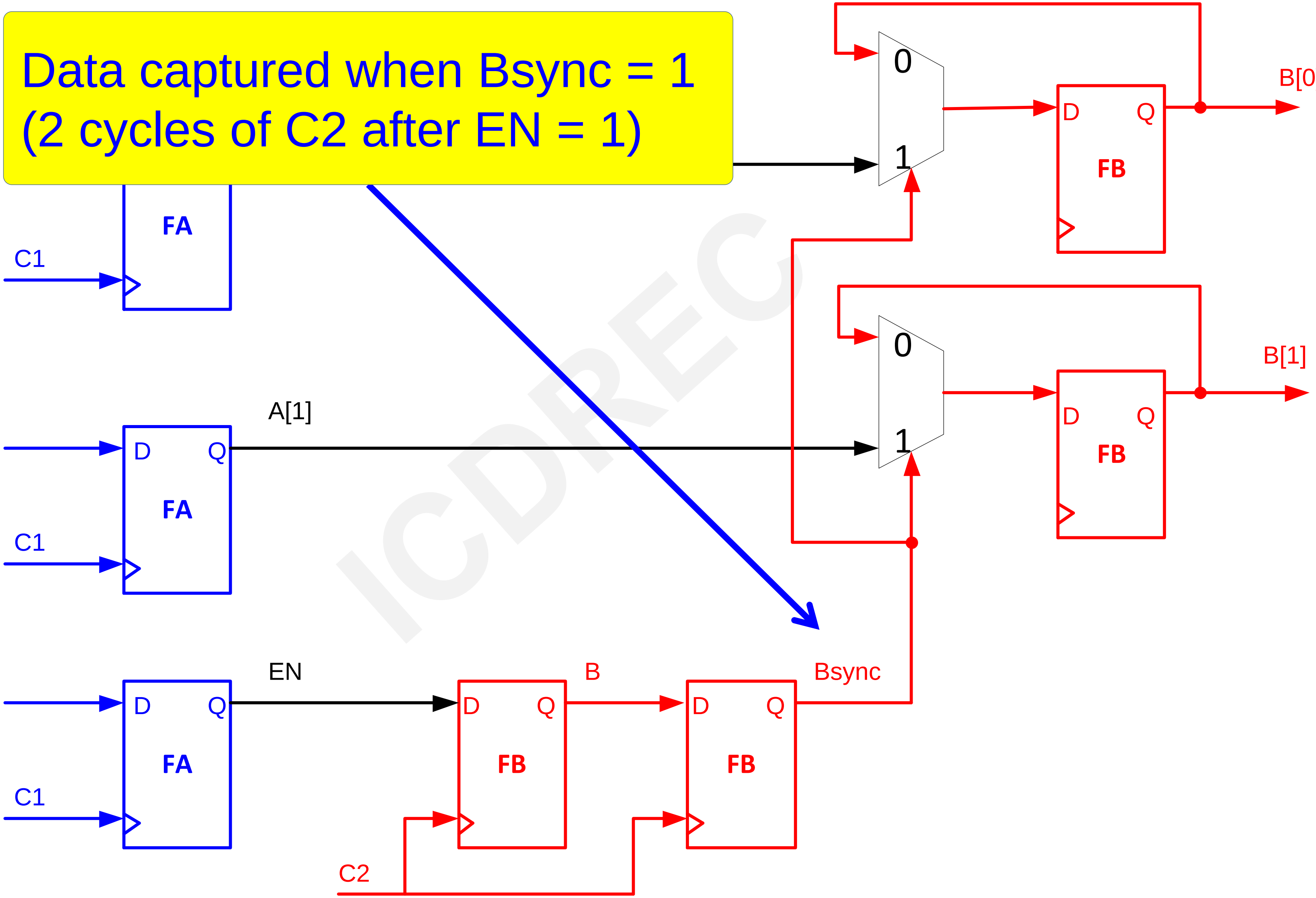
Invalid data

Valid data

- X[1:0] signals change from 00 to 11, but Y[1:0] signals change from 00 to 01 to 11

MUX recirculation technique

Data captured when Bsync = 1
(2 cycles of C2 after EN = 1)

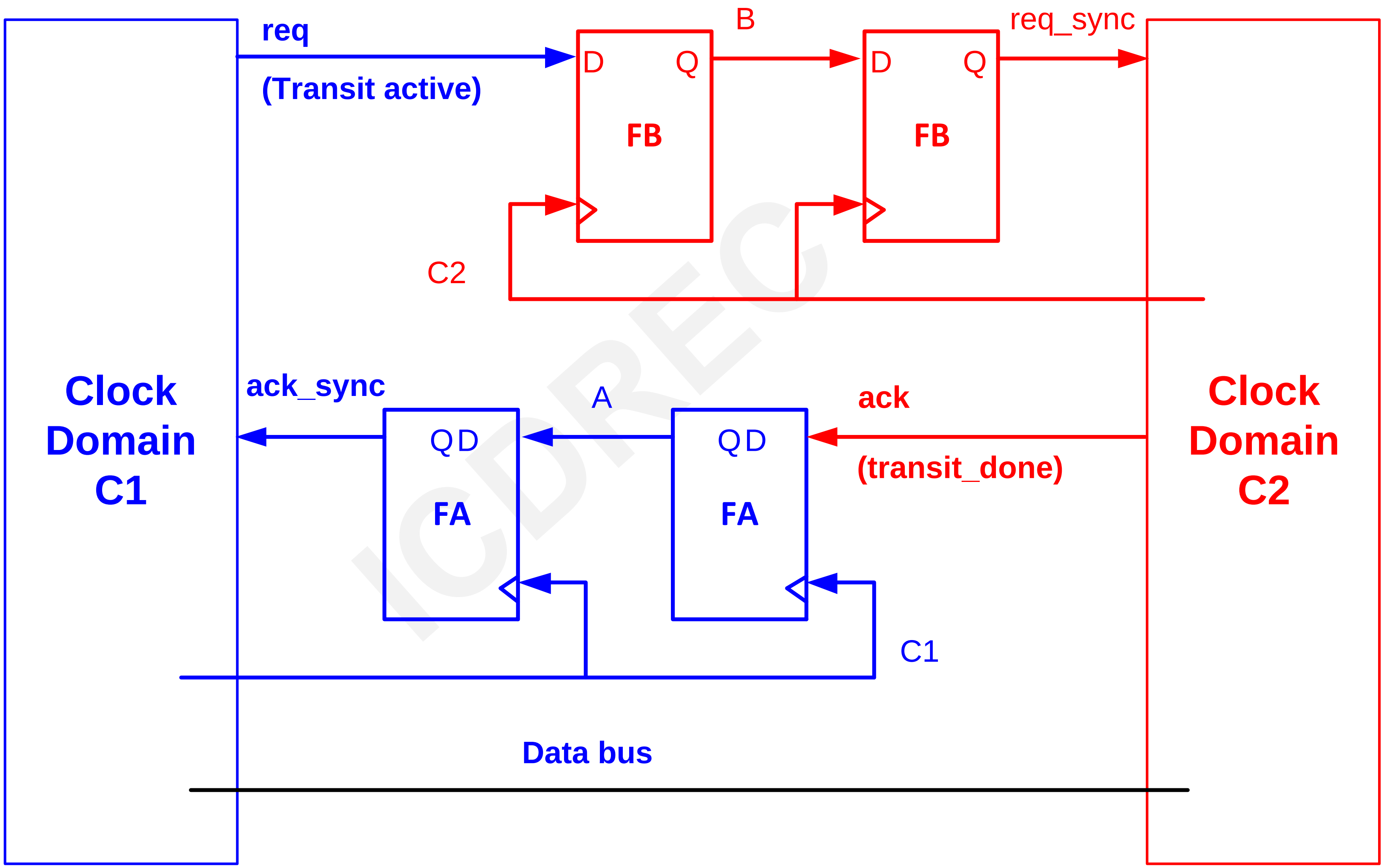


Gray Encoding to Avoid Data Incoherence

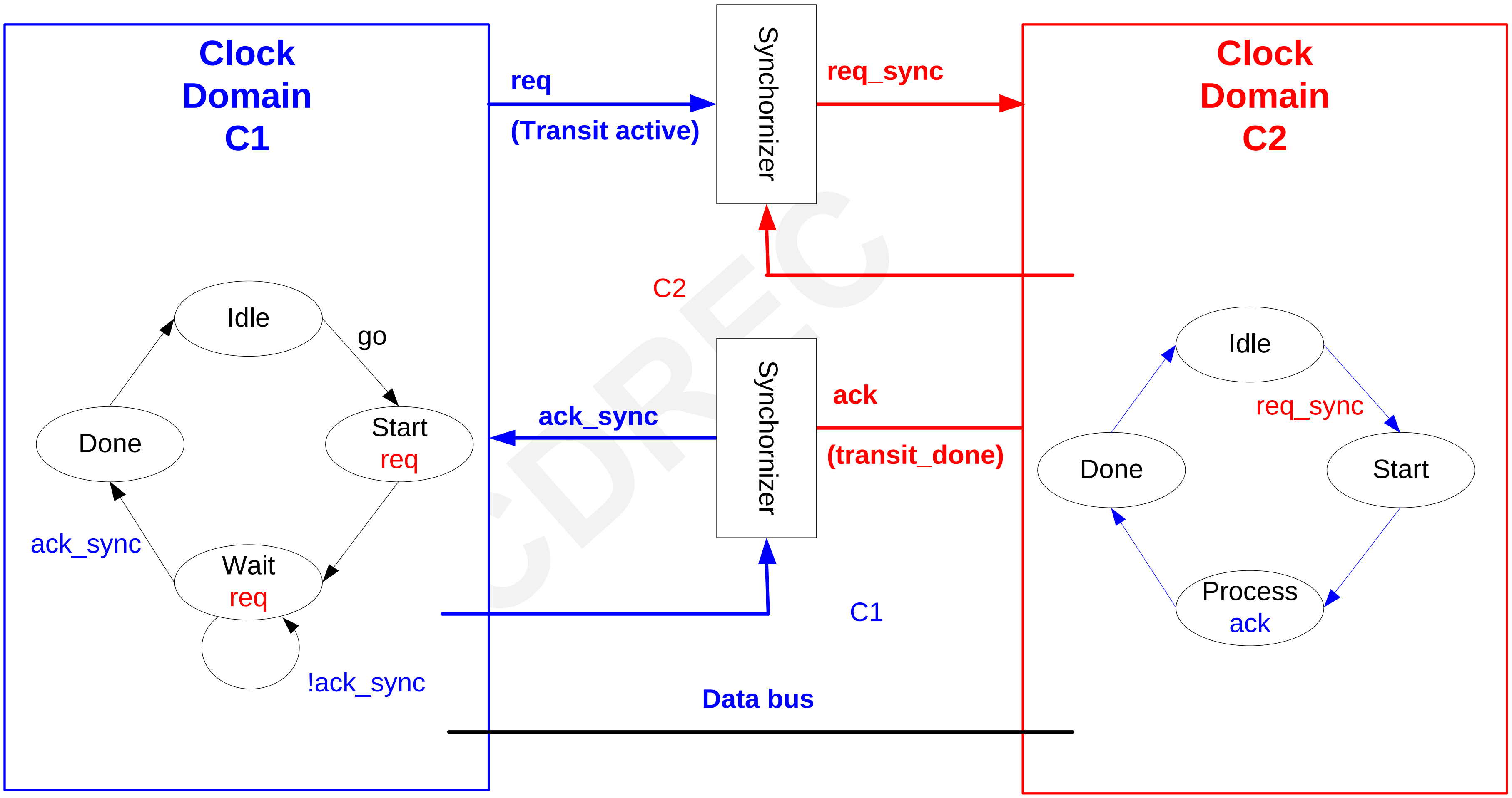
Decimal	Bin [3:0]	Gray [3:0]
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

For vector control signals (multi-bit signals, such as address buses), the usual solution is to use a Gray code when crossing a clock domain boundary. A Gray code ensures that only a single bit changes as the bus counts up or down.

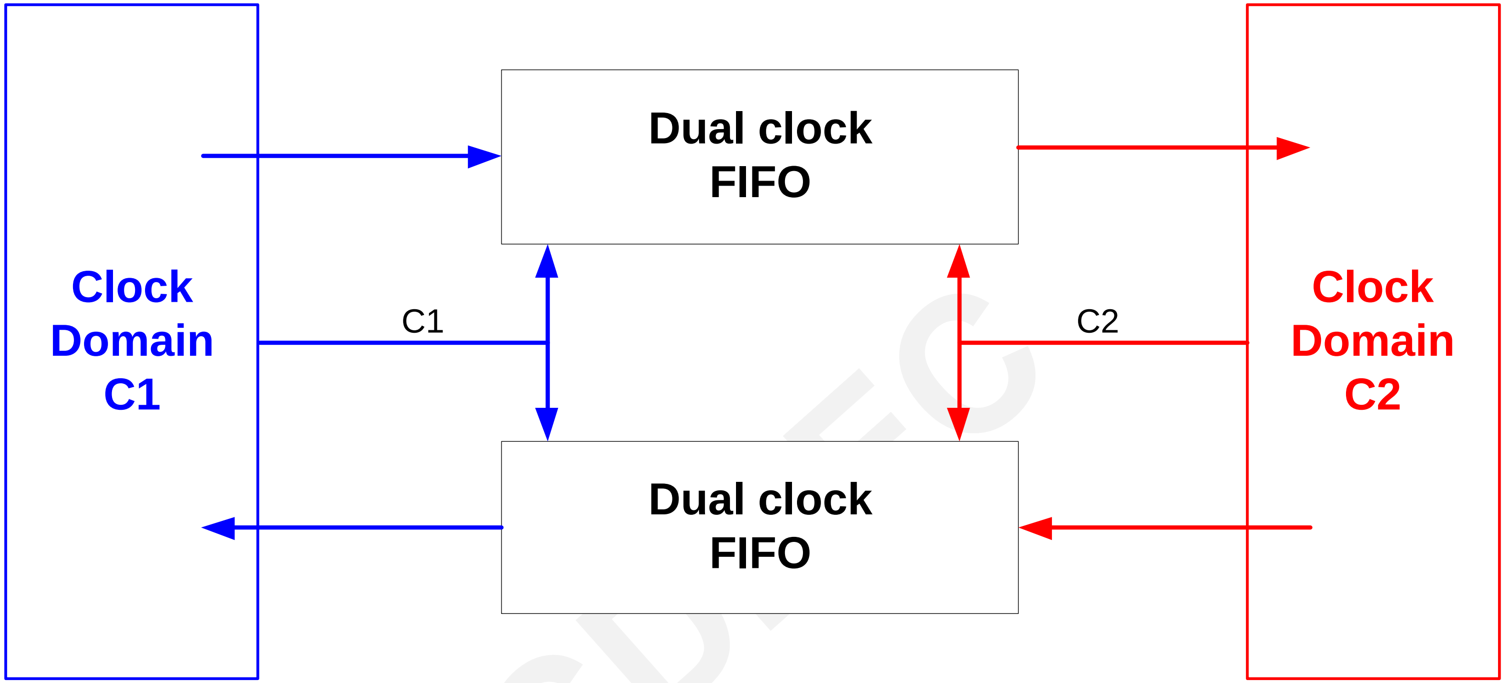
Handshaking Data between Clock Domains (1/2)



Handshaking Data between Clock Domains (2/2)



Using FIFO for Passing Data between Clock Domains

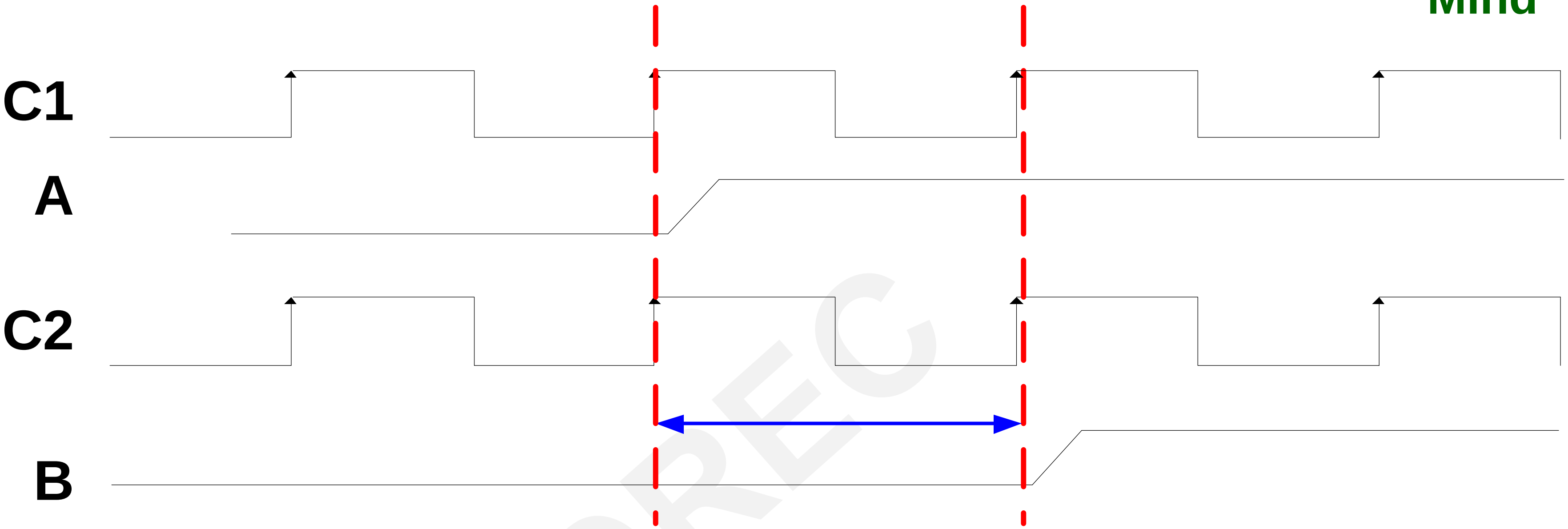


- One of the most popular methods of passing data
- A dual port memory is used for the FIFO storage.
- One port is controlled by the sender, the other port is controlled by the receiver.
- Two control signals are used to indicate if the FIFO is empty, full

Multiple Clock Domains

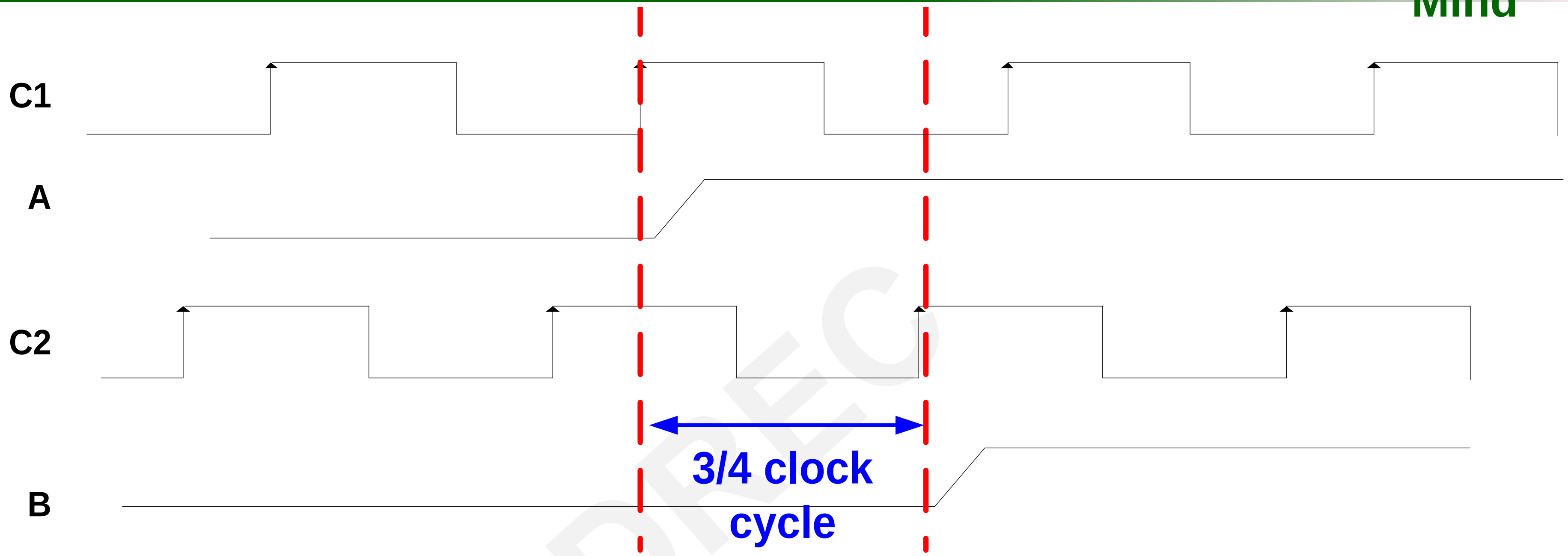
SYNCHRONOUS CLOCK DOMAIN CROSSINGS EXAMPLES

Same frequency, same phase clocks



- The clocks C1 and C2 have the same frequency and 0 phase difference.
- The clocks C1 and C2 are identical and generated from the same root clock, the data transfer from C1 to C2 is essentially not a clock domain crossing.
- For all practical purposes, this is the case of a single clock design.

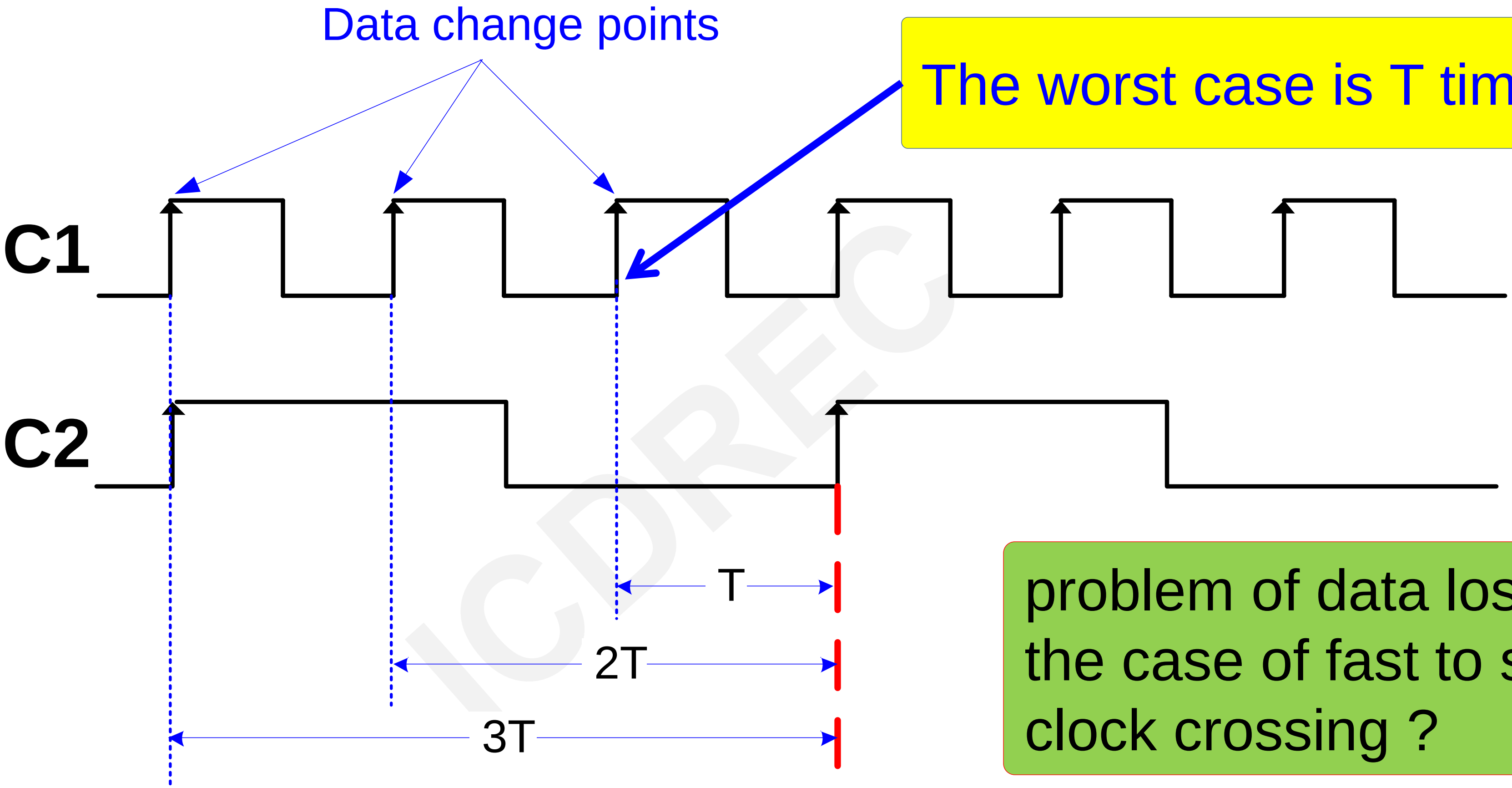
Same frequency and constant phase difference



- These are the clocks having the same time period but a constant phase difference. A typical example is the use of a clock and its inverted clock.
- Another example is a clock which is phase shifted from its parent clock, for example by T/4 where T is the time period of the clocks.

Integer multiple clocks ($T_{C2} = 3T_{C1}$)

The worst case is T time



problem of data loss in the case of fast to slow clock crossing ?

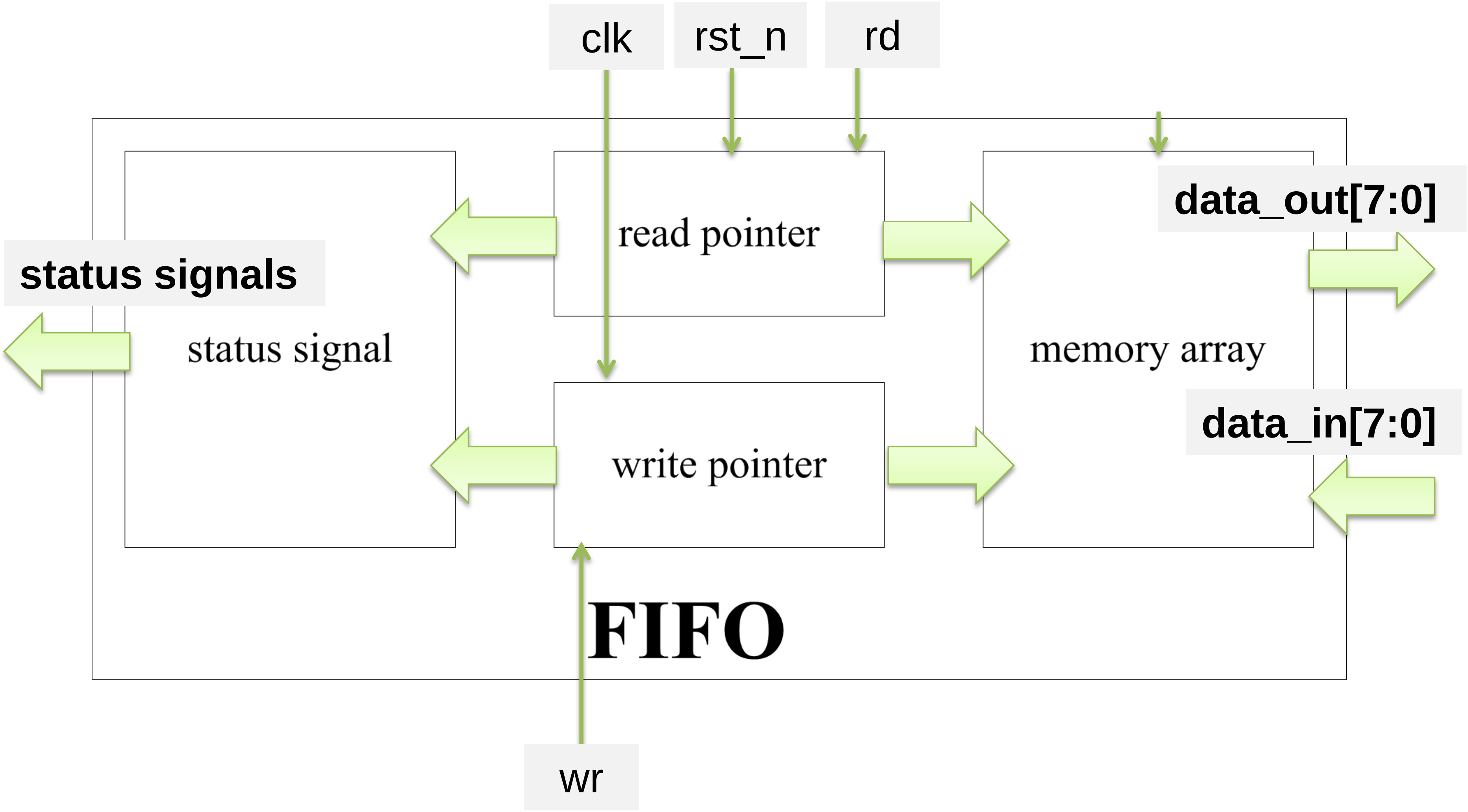
T, 2T, or 3T time available for data capture depending on data change point

Questions and Discussion

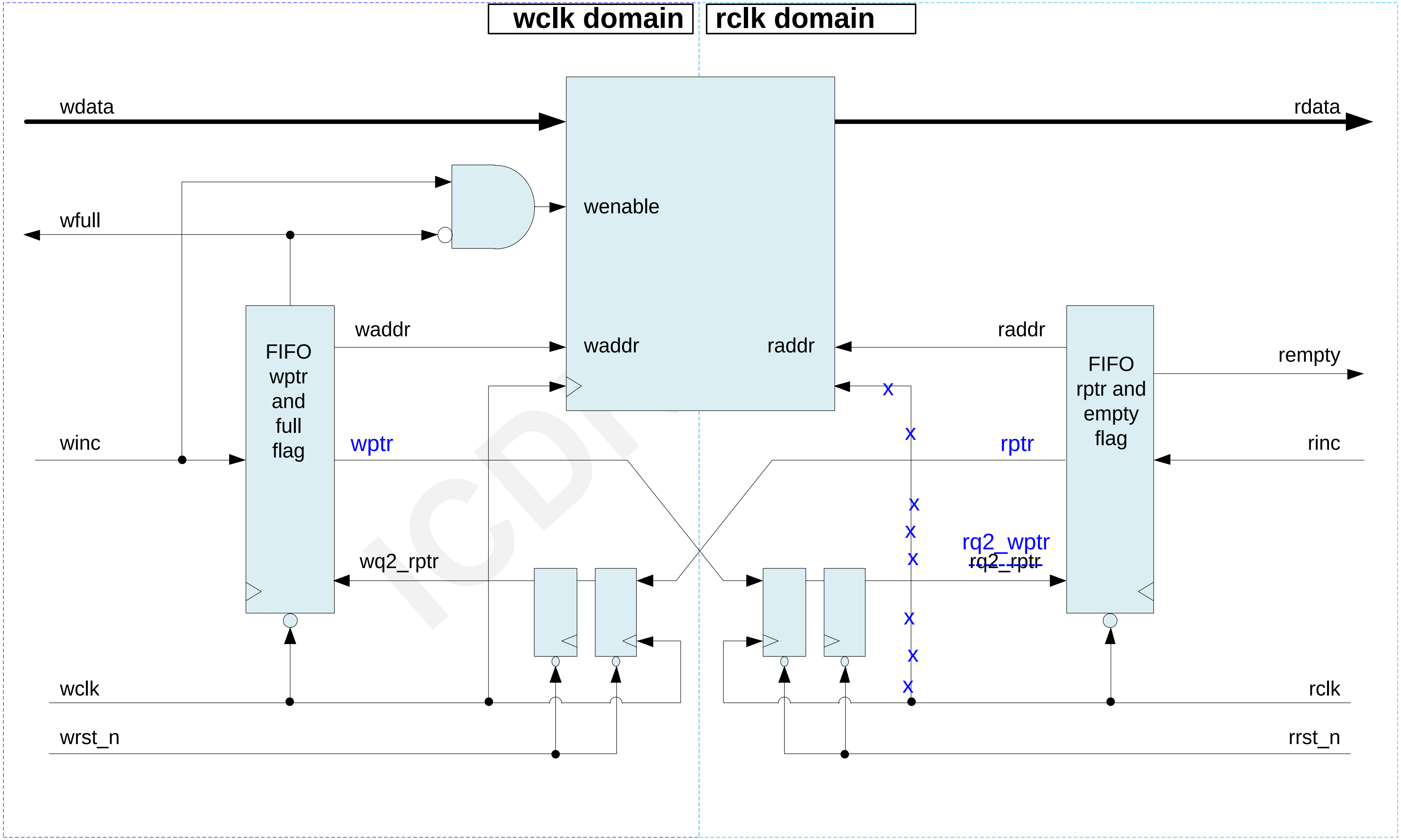
Multiple Clock Domains

EXERCISE – DUAL CLOCK FIFO DESIGN

Single clock FIFO



Dual clock FIFO



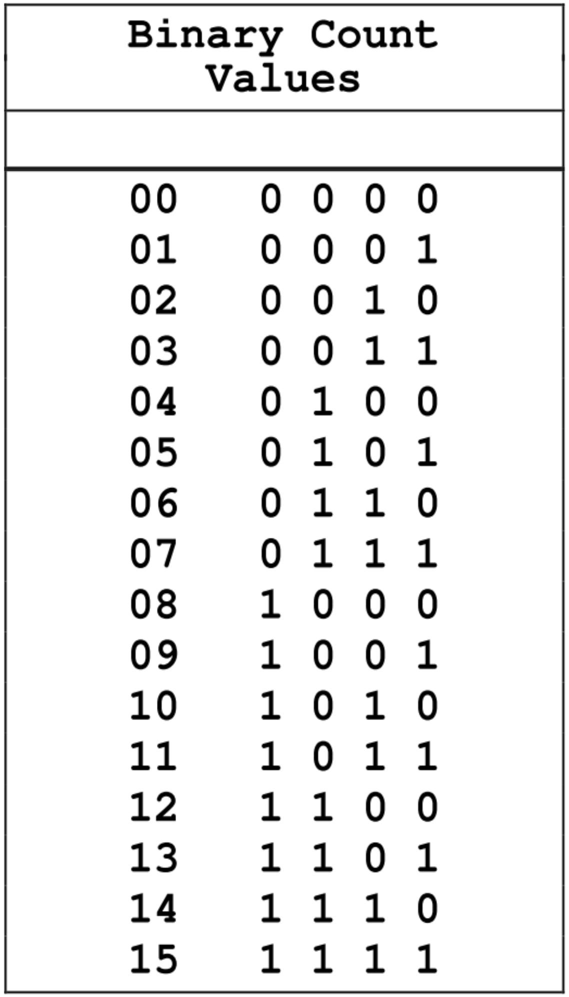
Problem

- Upgrade the single clock FIFO in subject 2 to dual clock FIFO
- Two clock domains are asynchronous
- Using Gray code technique for read & write pointers
- Checking RTL code with Leda
- Simulating the design with ModelSim

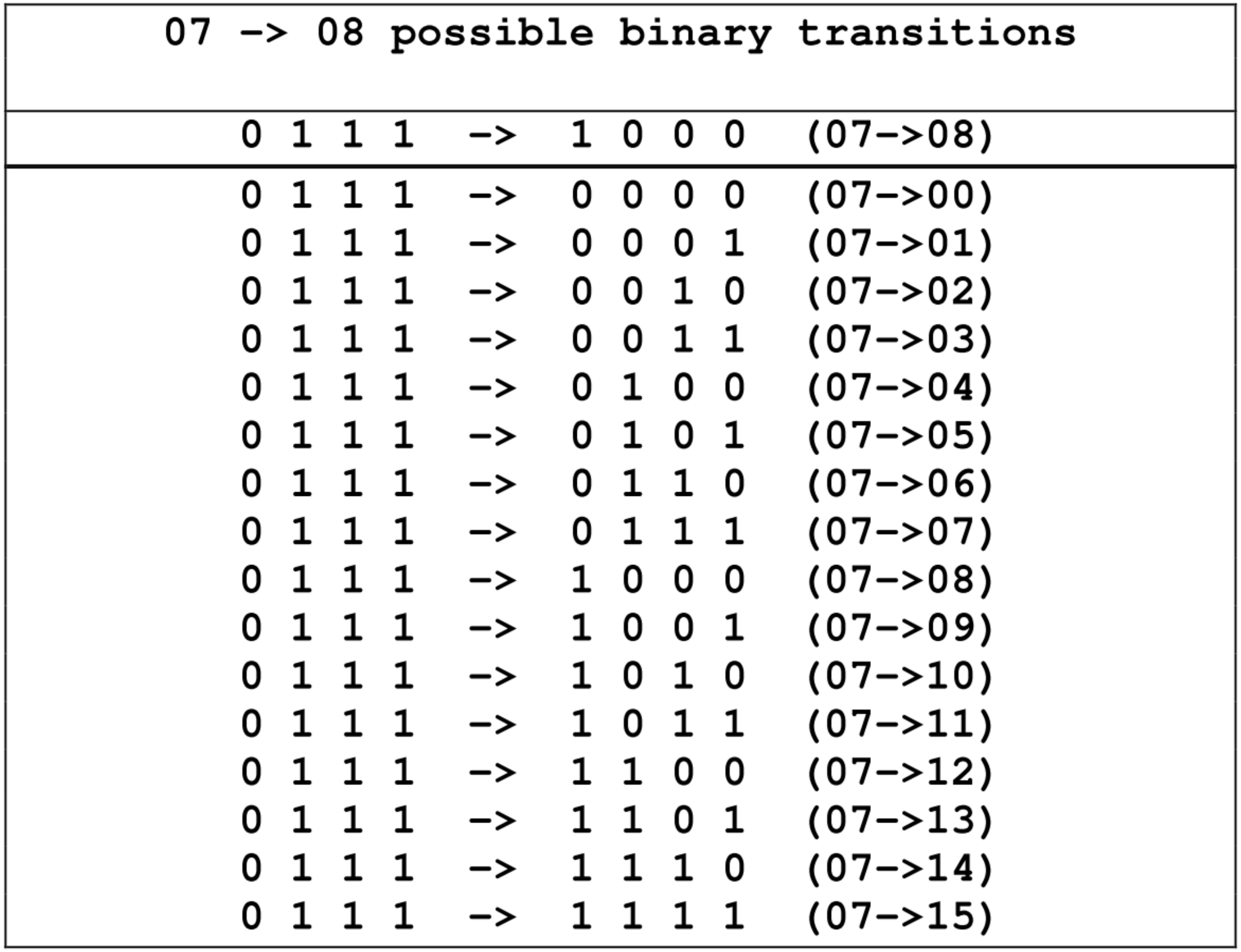
-Parameter NUM_LEVEL is the depth of dual clock FIFO. The valid values are 8,16,32.

-The data width is 8 bits.

Binary count values sampled in mid-transition

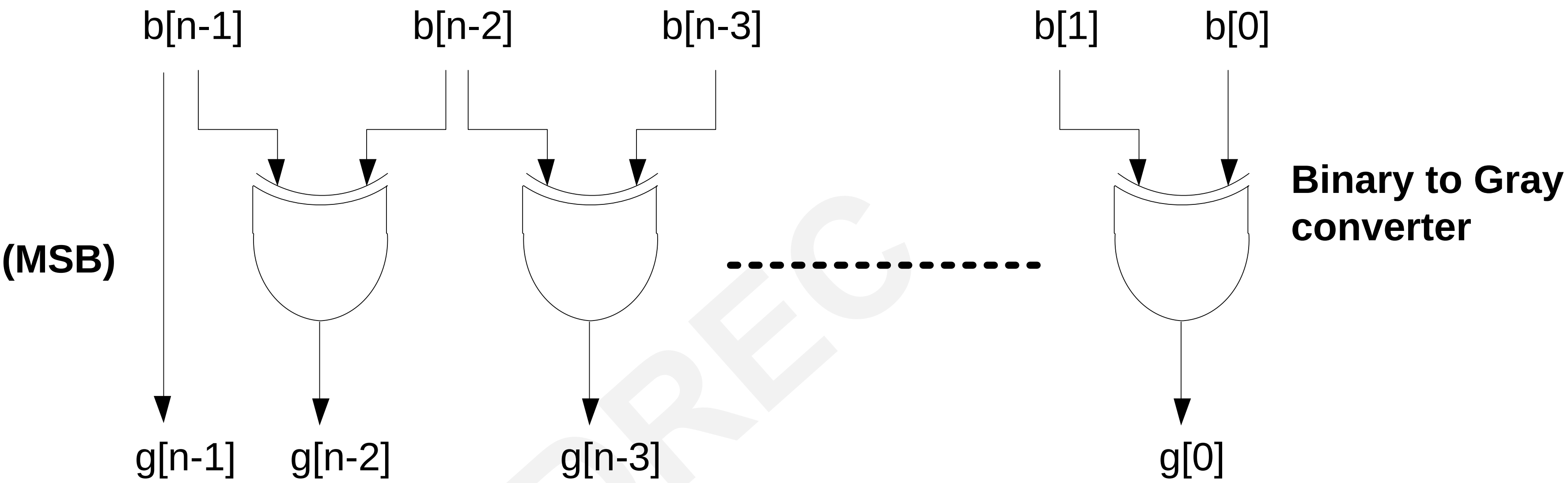


Binary Count Values	
00	0 0 0 0
01	0 0 0 1
02	0 0 1 0
03	0 0 1 1
04	0 1 0 0
05	0 1 0 1
06	0 1 1 0
07	0 1 1 1
08	1 0 0 0
09	1 0 0 1
10	1 0 1 0
11	1 0 1 1
12	1 1 0 0
13	1 1 0 1
14	1 1 1 0
15	1 1 1 1

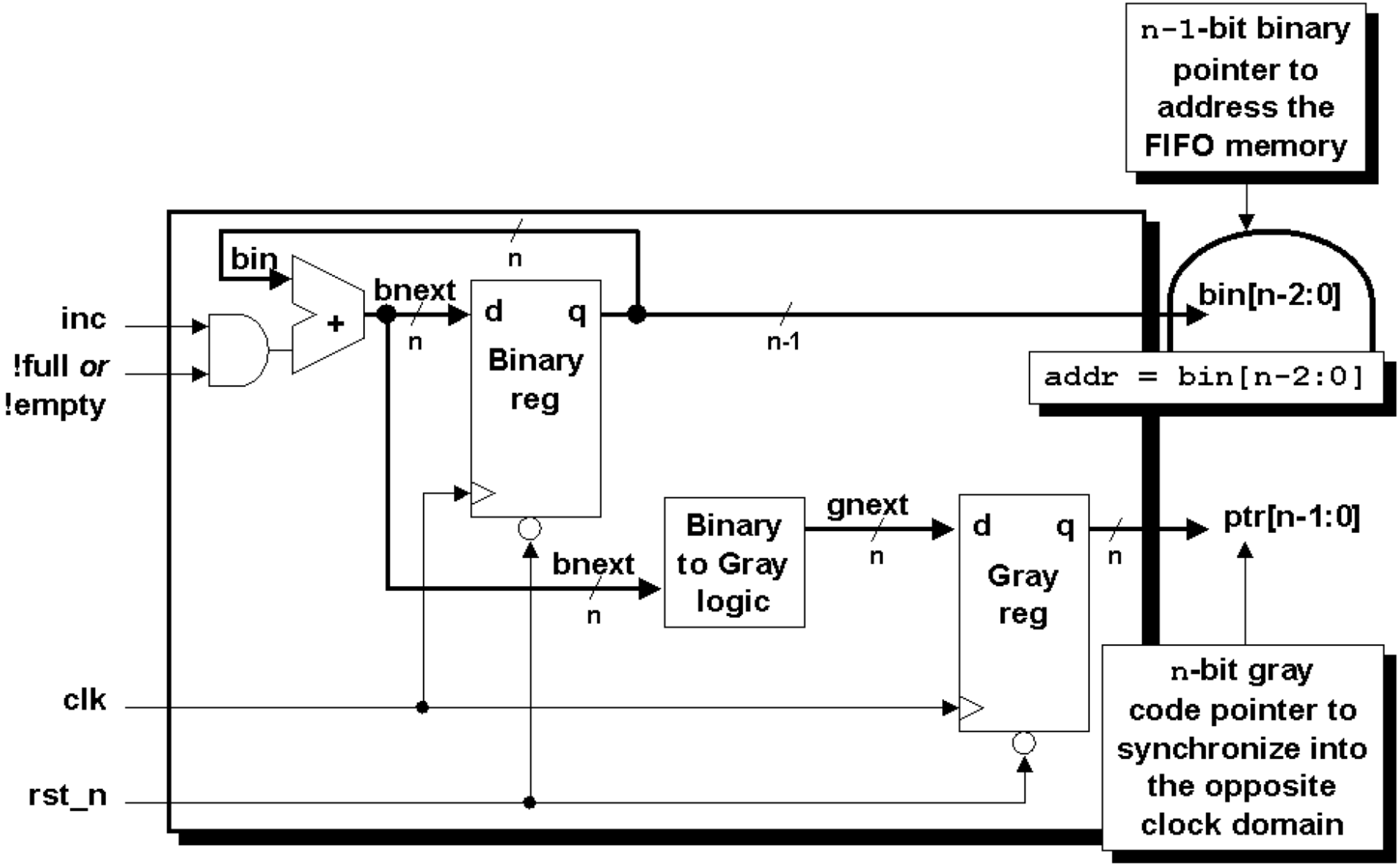


07 -> 08 possible binary transitions			
0 1 1 1	->	1 0 0 0	(07->08)
0 1 1 1	->	0 0 0 0	(07->00)
0 1 1 1	->	0 0 0 1	(07->01)
0 1 1 1	->	0 0 1 0	(07->02)
0 1 1 1	->	0 0 1 1	(07->03)
0 1 1 1	->	0 1 0 0	(07->04)
0 1 1 1	->	0 1 0 1	(07->05)
0 1 1 1	->	0 1 1 0	(07->06)
0 1 1 1	->	0 1 1 1	(07->07)
0 1 1 1	->	1 0 0 0	(07->08)
0 1 1 1	->	1 0 0 1	(07->09)
0 1 1 1	->	1 0 1 0	(07->10)
0 1 1 1	->	1 0 1 1	(07->11)
0 1 1 1	->	1 1 0 0	(07->12)
0 1 1 1	->	1 1 0 1	(07->13)
0 1 1 1	->	1 1 1 0	(07->14)
0 1 1 1	->	1 1 1 1	(07->15)

Binary-to-Gray



```
gray[0] = bin[0] ^ bin[1];
gray[1] = bin[1] ^ bin[2];
gray[2] = bin[2] ^ bin[3];
gray[3] = bin[3] ^  1'b0 ; // same as gray[3] = bin[3];
```

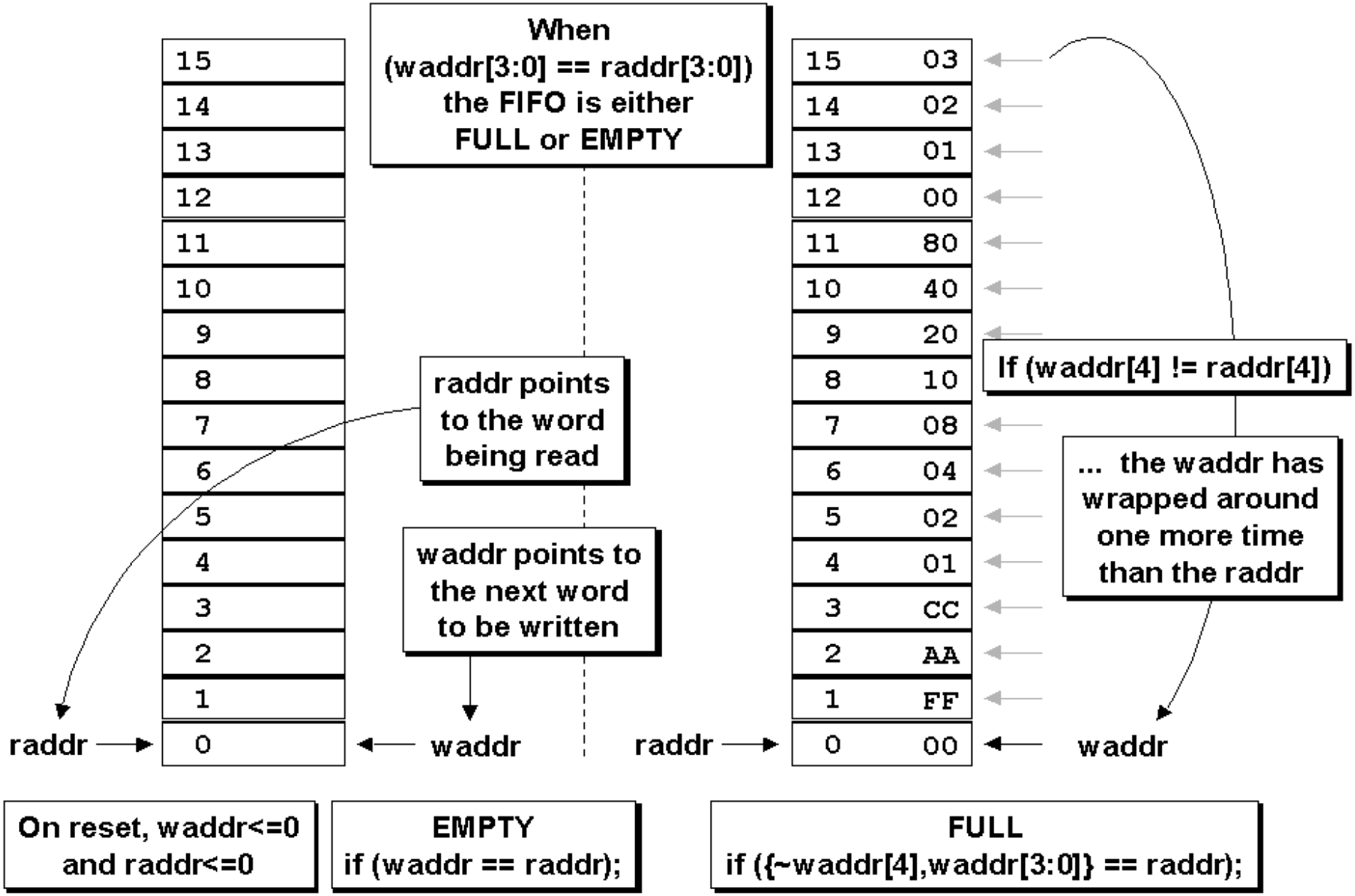


n-1-bit binary pointer to address the FIFO memory

addr = bin[n-2:0]

n-bit gray code pointer to synchronize into the opposite clock domain

FIFO full & empty with Binary



When (waddr[3:0] == raddr[3:0]) the FIFO is either FULL or EMPTY

15	
14	
13	
12	
11	
10	
9	
8	
7	
6	
5	
4	
3	
2	
1	
0	

raddr points to the word being read

waddr points to the next word to be written

raddr → 0 ← waddr

15	03
14	02
13	01
12	00
11	80
10	40
9	20
8	10
7	08
6	04
5	02
4	01
3	CC
2	AA
1	FF
0	00

raddr → 0 ← waddr

If (waddr[4] != raddr[4])

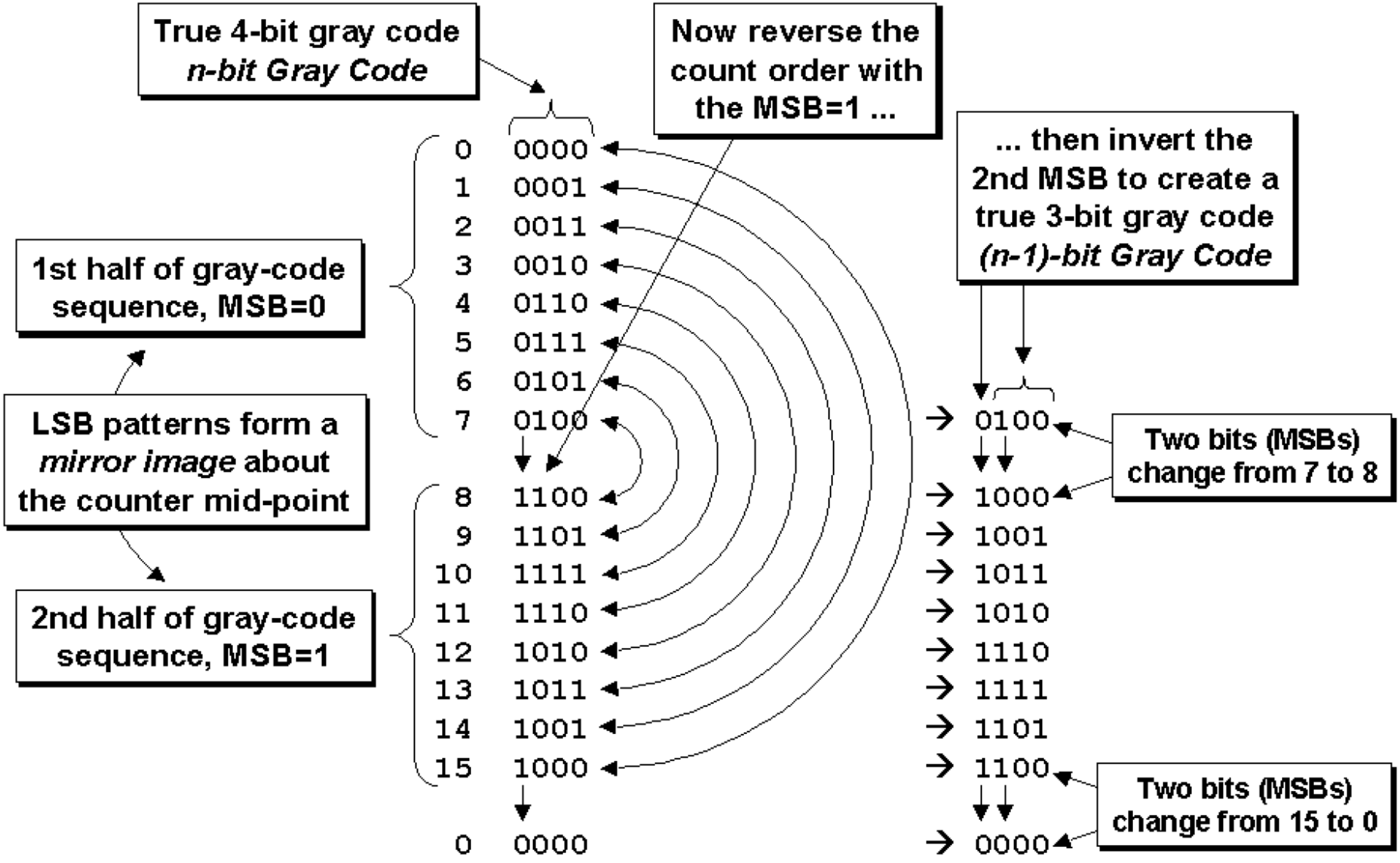
... the waddr has wrapped around one more time than the raddr

On reset, waddr<=0 and raddr<=0

EMPTY
if (waddr == raddr);

FULL
if ({~waddr[4],waddr[3:0]} == raddr);

Counter with different size problem



True 4-bit gray code
n-bit Gray Code

Now reverse the count order with the MSB=1 ...

... then invert the 2nd MSB to create a true 3-bit gray code
(n-1)-bit Gray Code

1st half of gray-code sequence, MSB=0

LSB patterns form a *mirror image* about the counter mid-point

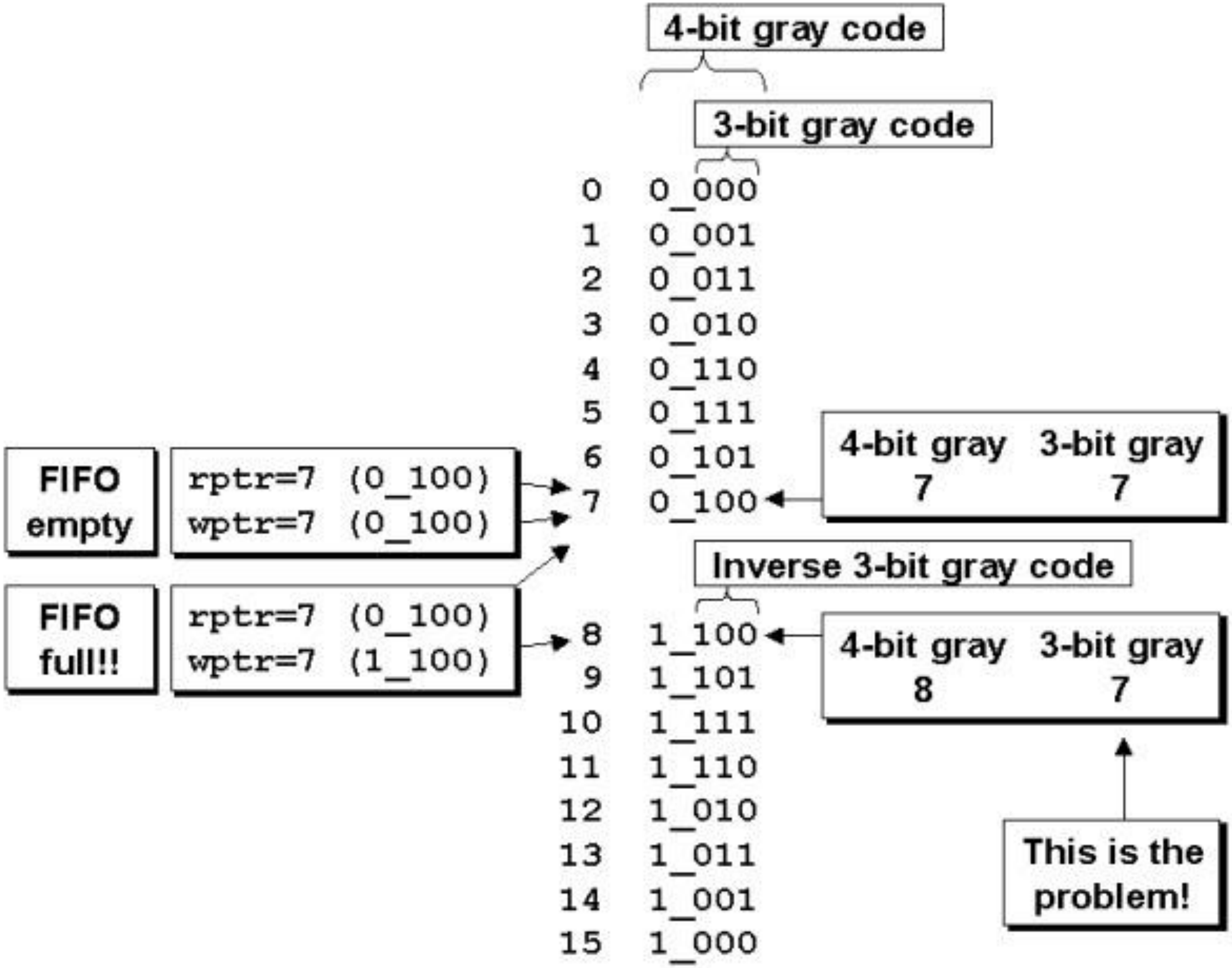
2nd half of gray-code sequence, MSB=1

Two bits (MSBs) change from 7 to 8

Two bits (MSBs) change from 15 to 0

0	0000	
1	0001	
2	0011	
3	0010	
4	0110	
5	0111	
6	0101	
7	0100	0100
8	1100	1000
9	1101	1001
10	1111	1011
11	1110	1010
12	1010	1110
13	1011	1111
14	1001	1101
15	1000	1100
0	0000	0000

Full & empty flags with Gray



4-bit gray code

3-bit gray code

0	0_000
1	0_001
2	0_011
3	0_010
4	0_110
5	0_111
6	0_101
7	0_100
8	1_100
9	1_101
10	1_111
11	1_110
12	1_010
13	1_011
14	1_001
15	1_000

Inverse 3-bit gray code

FIFO empty

```
rptr=7  (0_100)
wptr=7  (0_100)
```

FIFO full!!

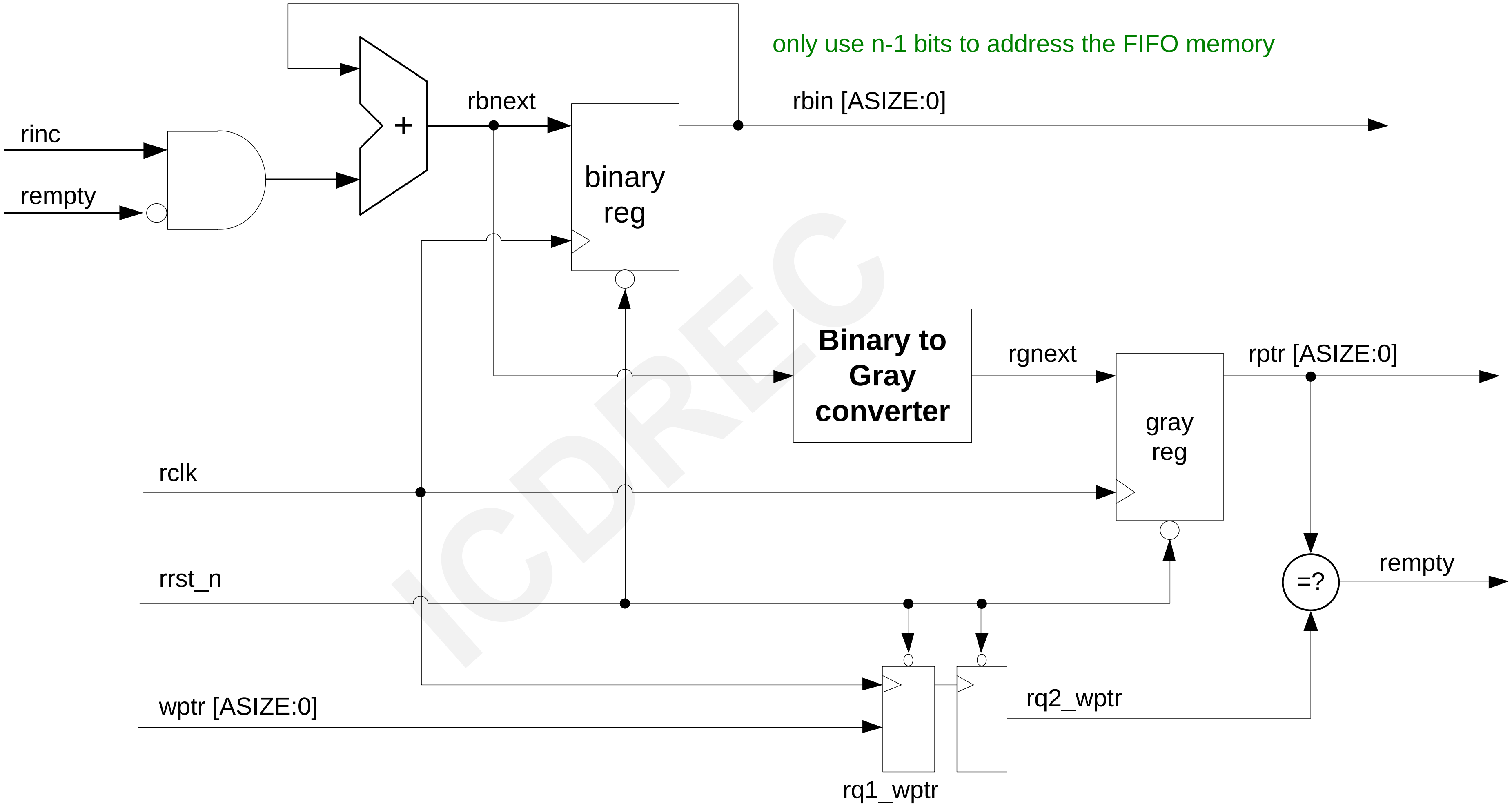
```
rptr=7  (0_100)
wptr=7  (1_100)
```

4-bit gray	3-bit gray
7	7

4-bit gray	3-bit gray
8	7

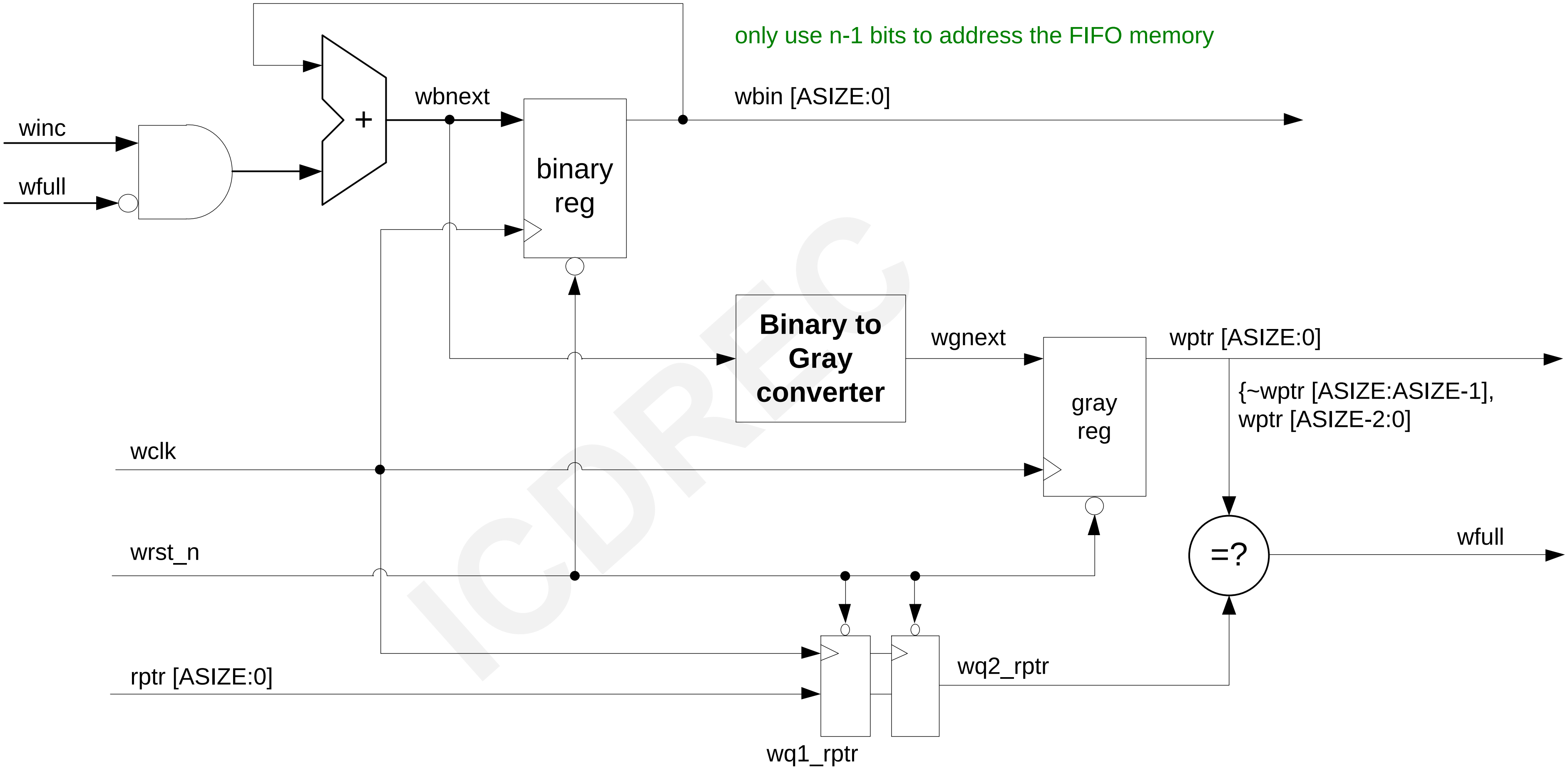
This is the problem!

FIFO empty flag



rptr [ASIZE:0] = rq2_wptr [ASIZE:0]

FIFO full flag



{~wptr [ASIZE:ASIZE-1], wptr [ASIZE-2:0] == wq2_rptr

ICDREC
SigmaK3
SM01-25
Made in VIETNAM

VN8-01
DESIGNED BY ICDREC
VIET NAM

VN16-32
DESIGNED BY ICDREC
VIET NAM

Thank You!