Digital Circuits and Systems
Lecture 12 Metastability and Synchronization Failure

Tian Sheuan Chang

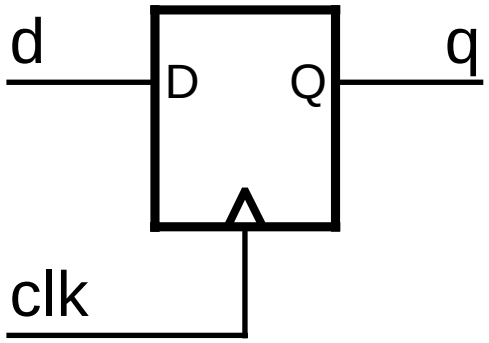
(or When Good Flip-Flops go Bad)為什麼DFF會壞掉

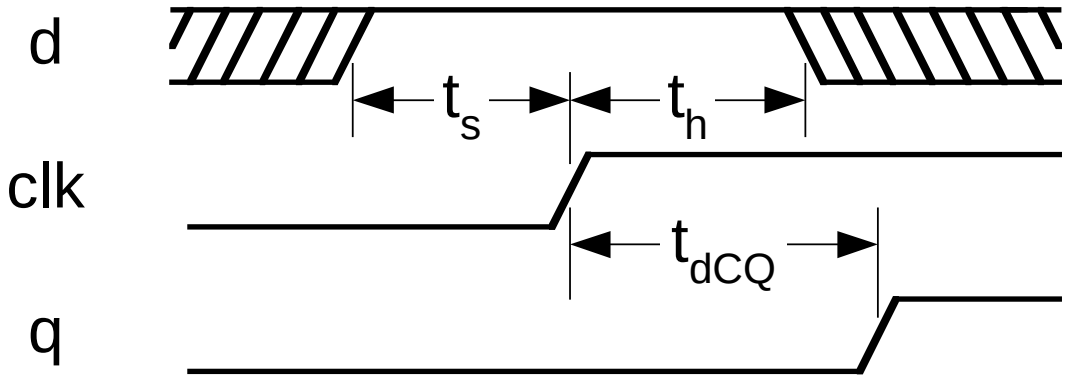
Outline [Dally Ch. 27/28]

- 壞掉的DFF
  - 當你違反setup time/hold time constraints時
- 對設計的影響與其解法
  - (何時最容易發生)

壞掉的DFF => 變成震盪器了
當你違反SETUP TIME/HOLD TIME CONSTRAINTS時

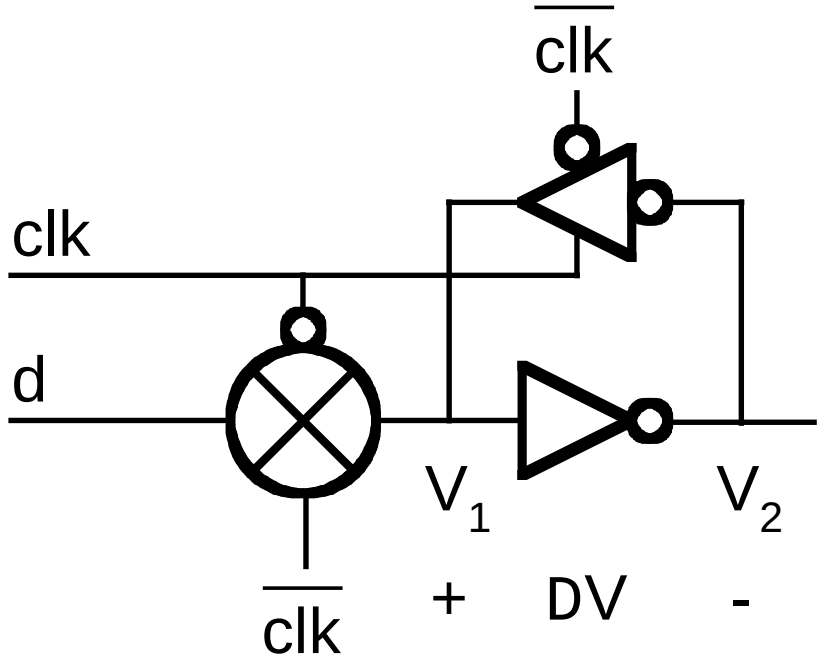
What happens when we violate setup and hold time constraints?

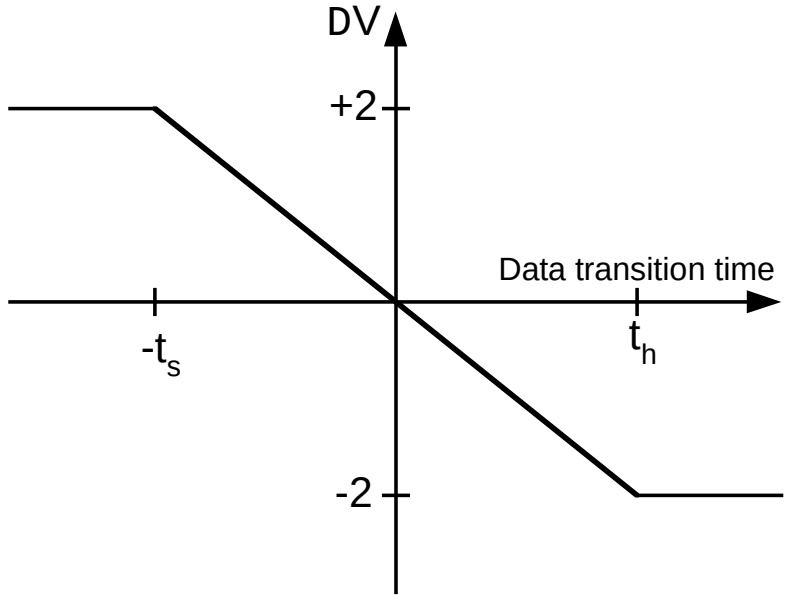




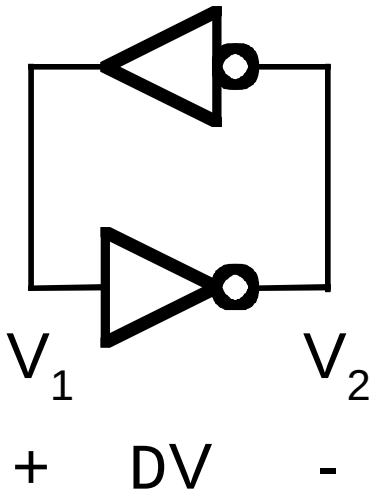
Look at structure of CMOS latch

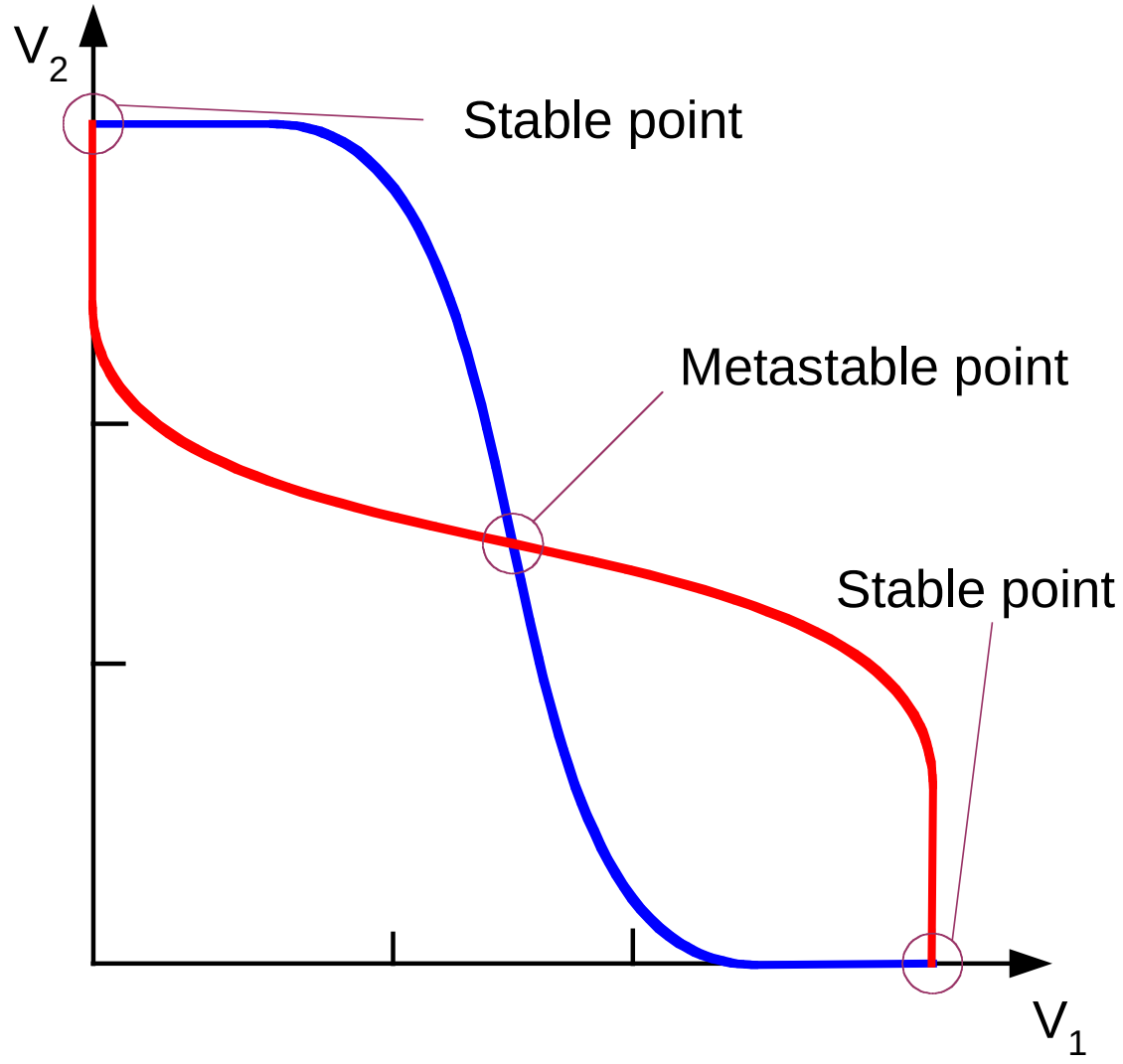
- Storage loop gets initialized with an ‘analog’ value
- Latch is a “time-to-voltage” converter



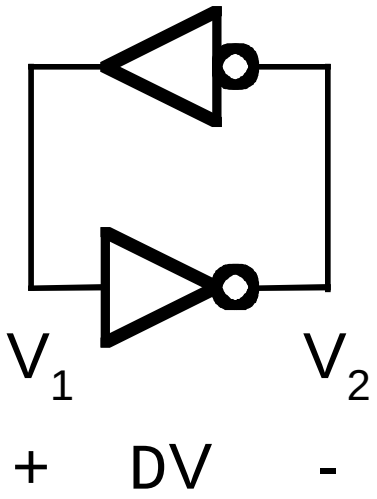


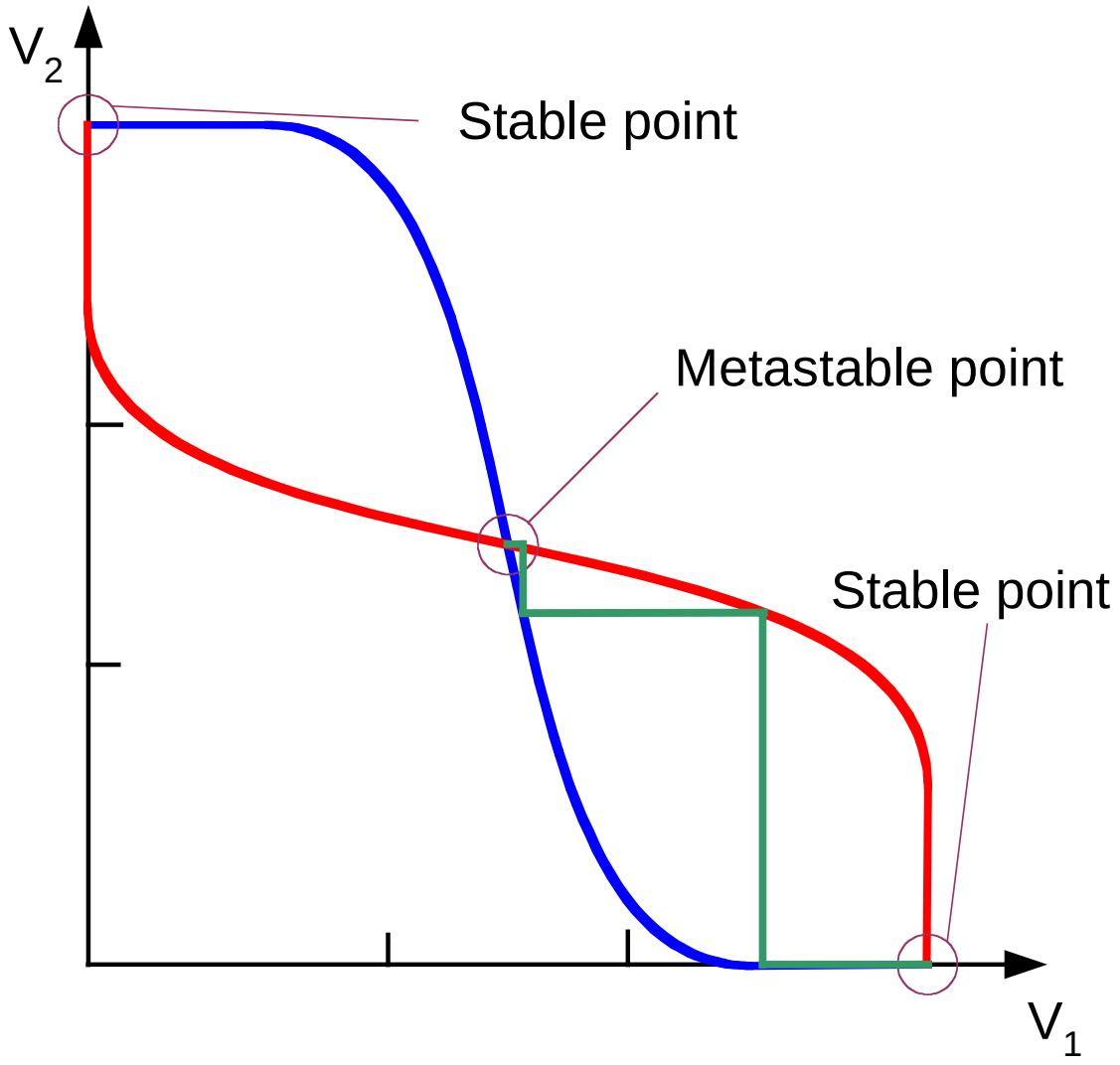
Storage loop has a metastable state between 0 and 1



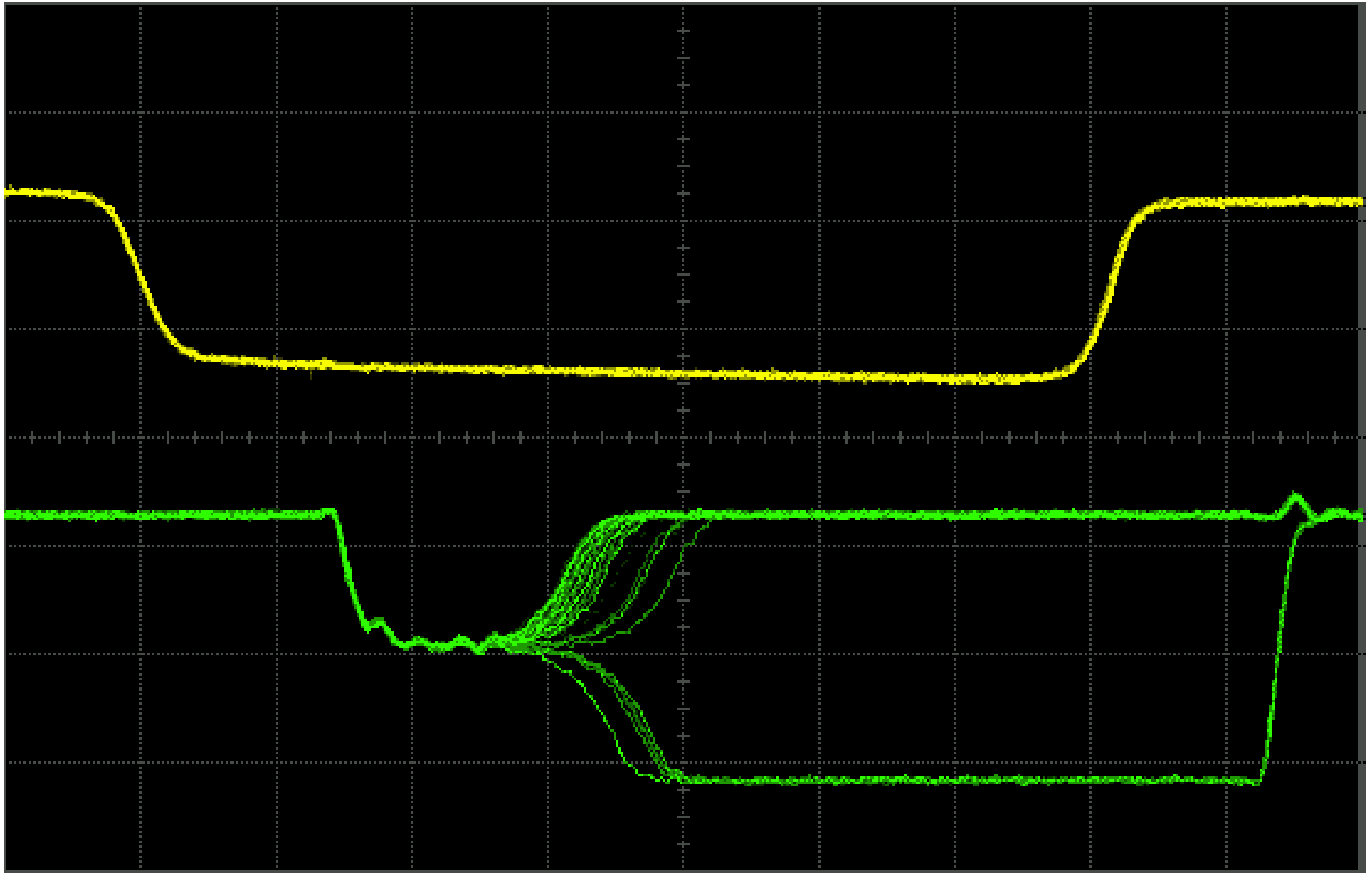


Dynamics of ΔV

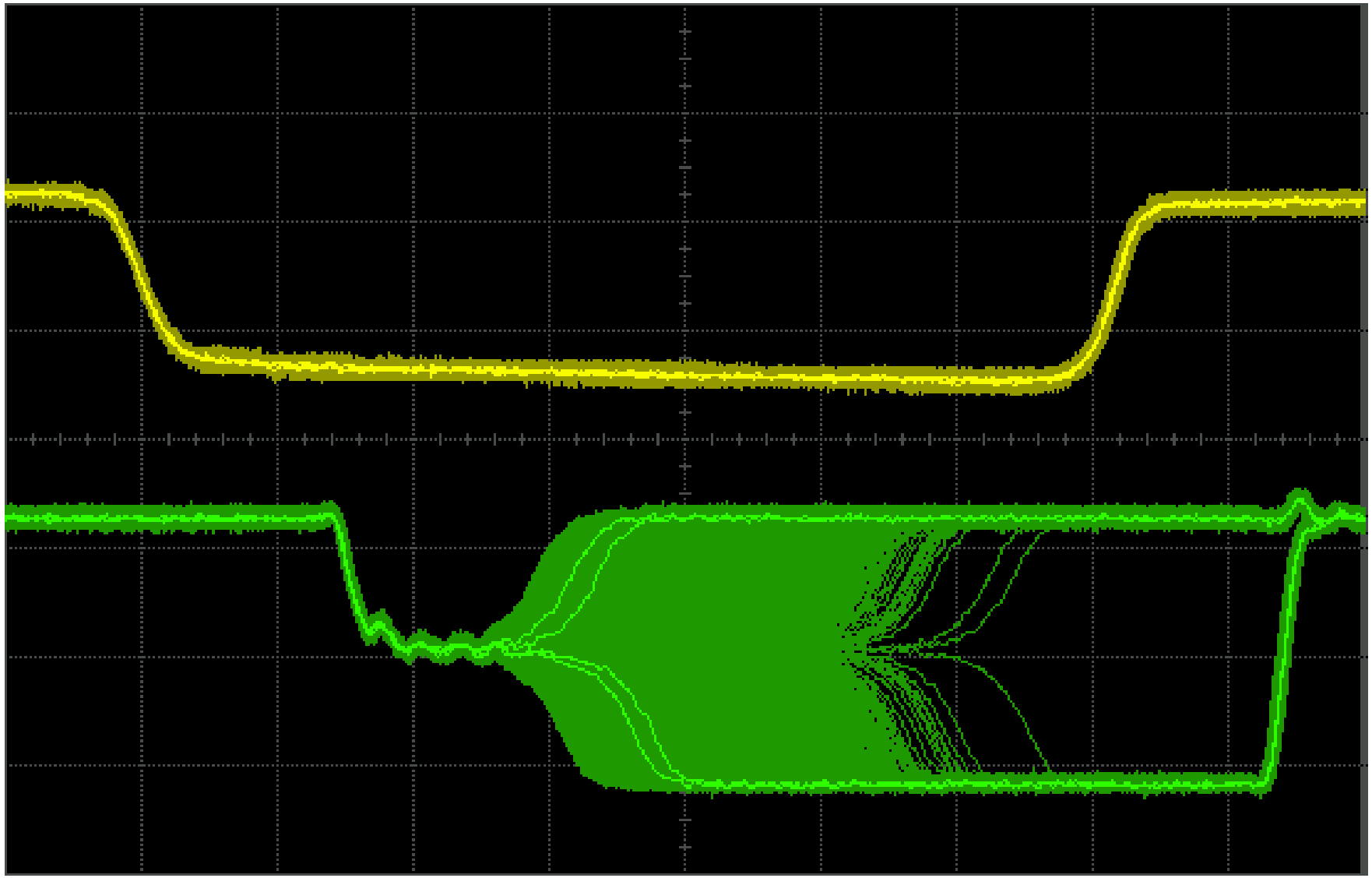




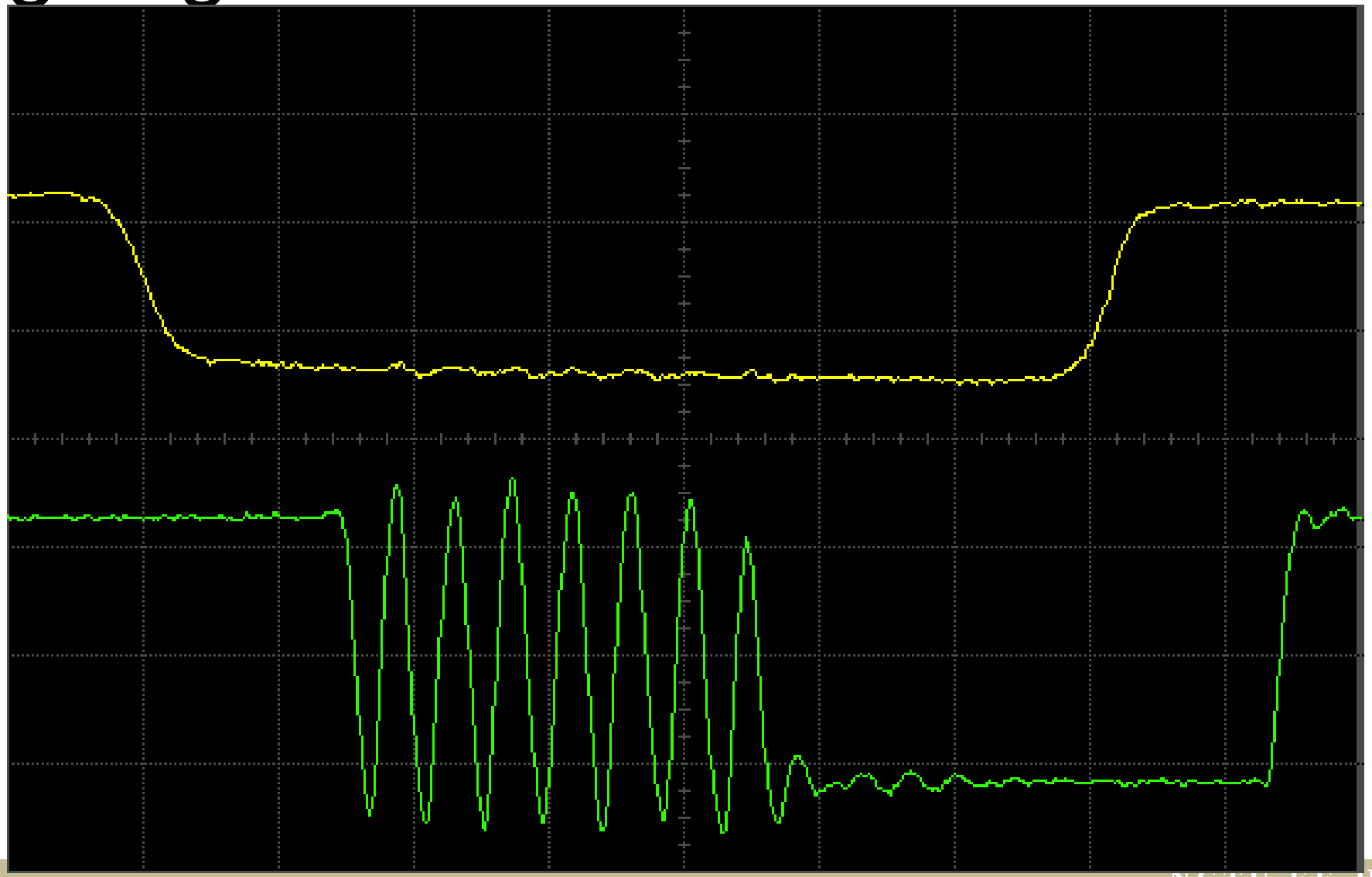
Metastable state of FF1 – 4007 Nand RS Latch



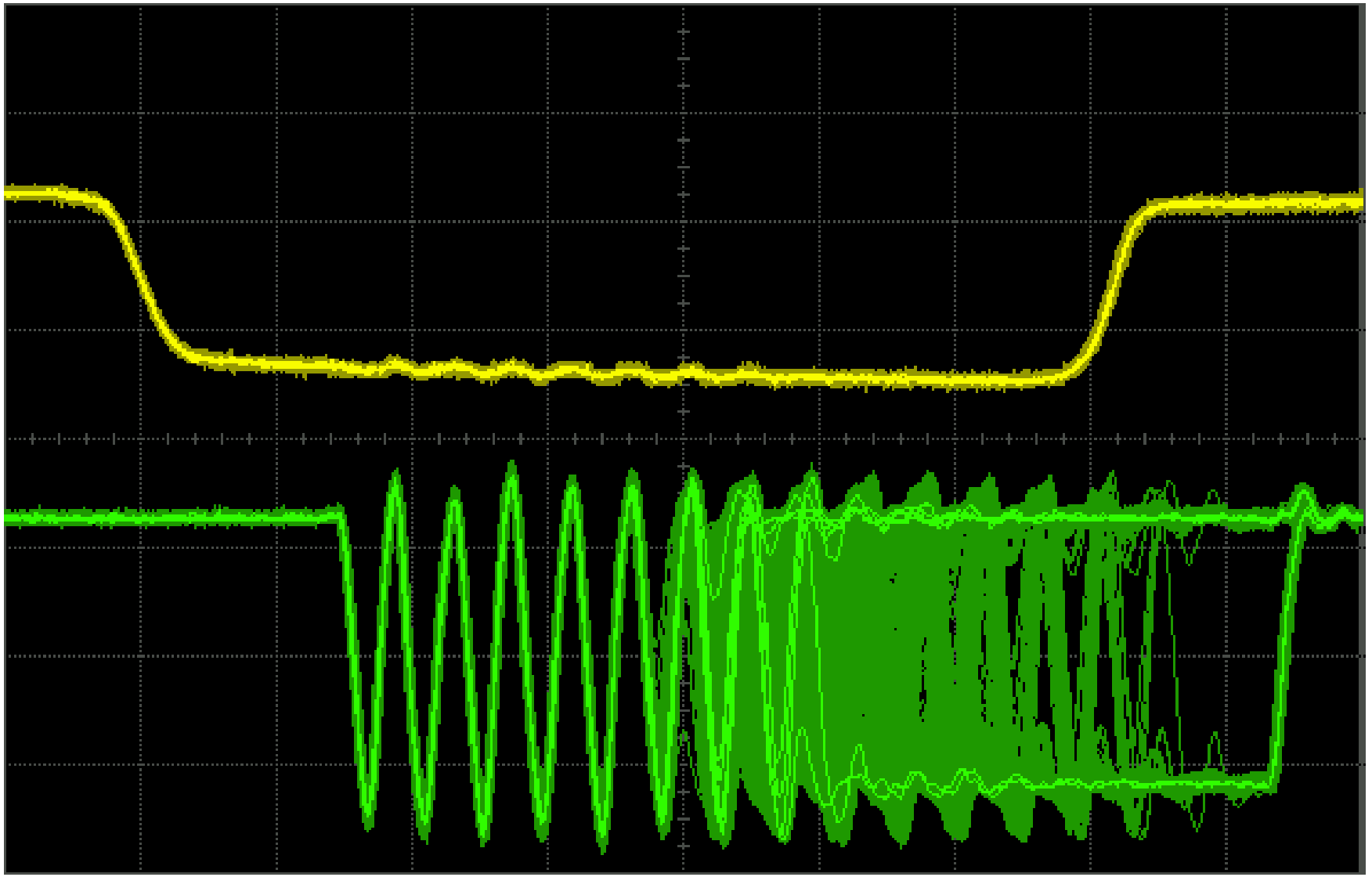
Over time the waveform fills in



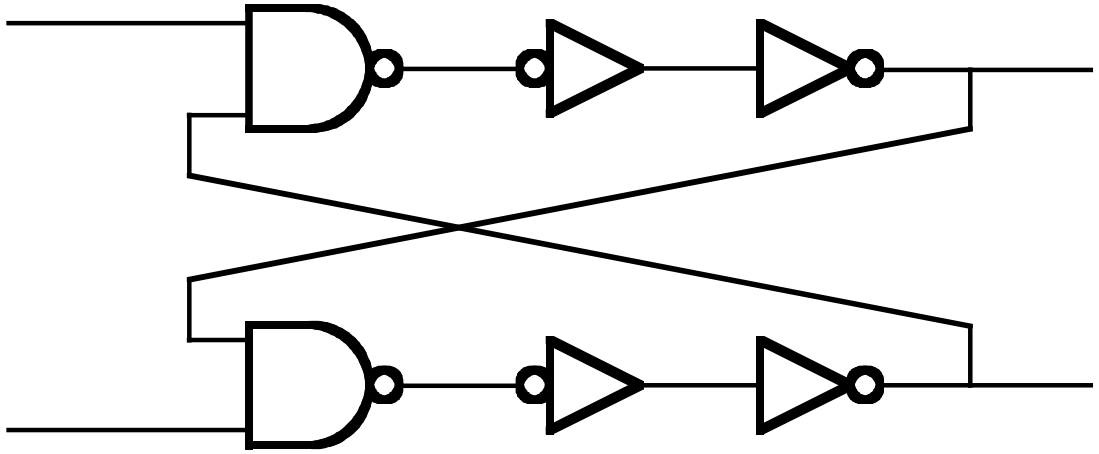
Metastable state of 4011 Nand RS Latch
What's going on here?



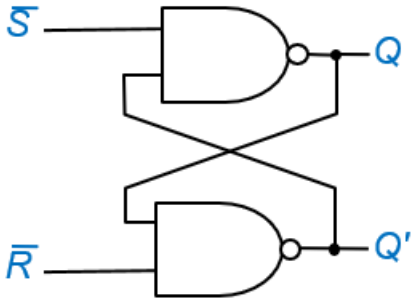
Over time this waveform fills in too

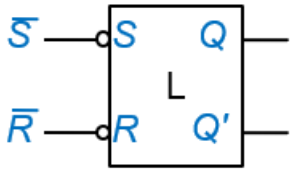


Actual circuit of 4011 Nand RS Latch



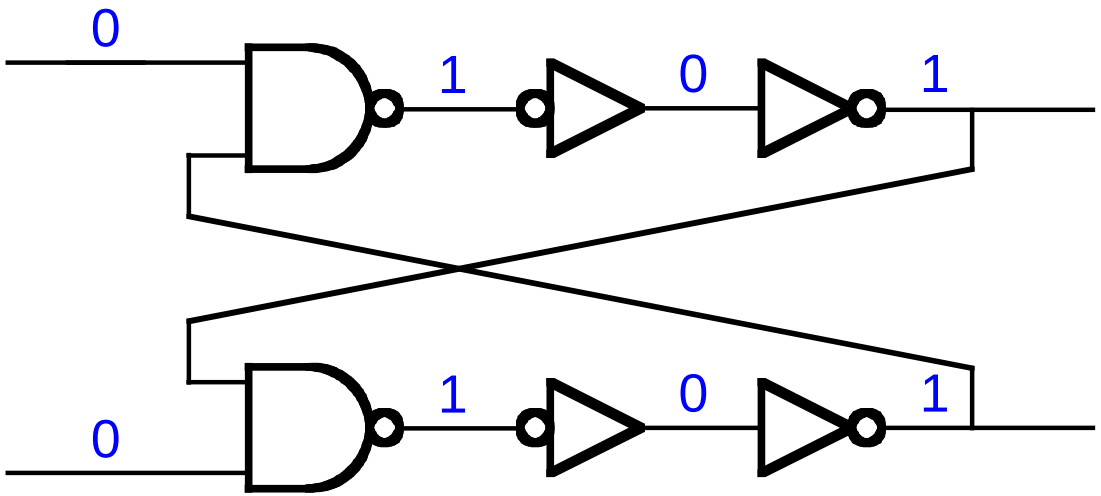
- $\bar{S}$-$\bar{R}$ latch: active-low inputs for *S* & *R*





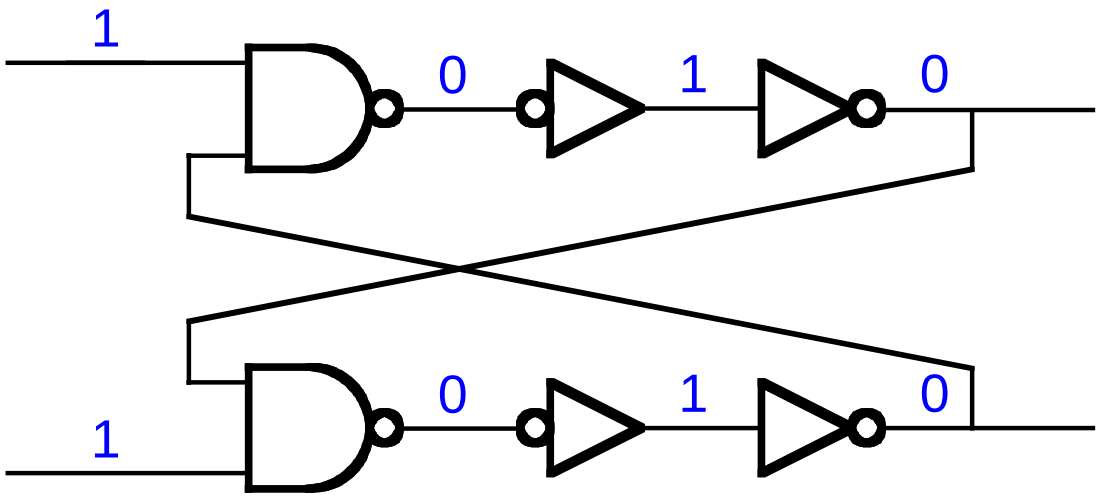
$\bar{S}$	$\bar{R}$	Q	Q^+	
1	1	0	0	Unchanged
1	1	1	1	
1	0	0	0	Reset to 0
1	0	1	0	
0	1	0	1	Set to 1
0	1	1	1	
0	0	0	–	Inputs not allowed
0	0	1	–	

Initial state when both inputs are low

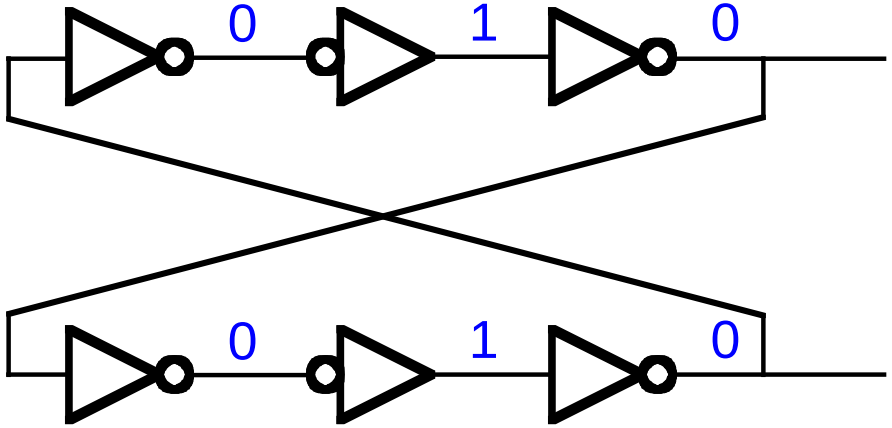


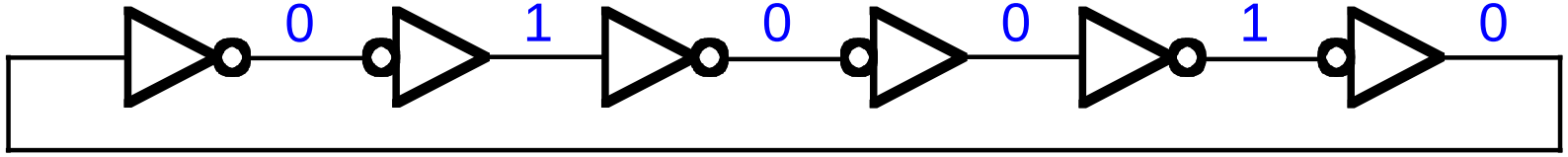
- When both inputs go high, it becomes a 6-stage ring oscillator

  Oscillation is metastable

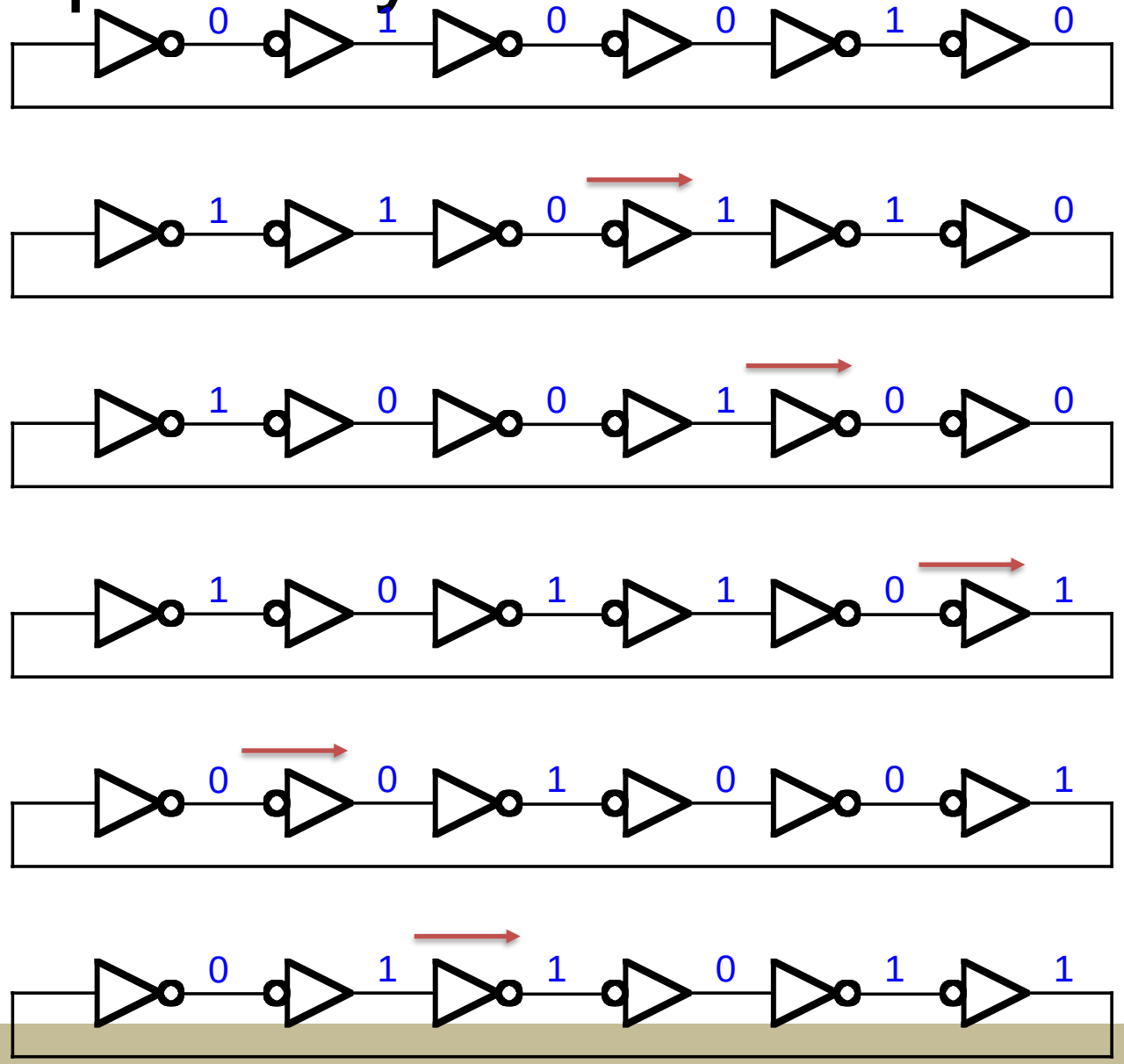


Oscillation is metastable



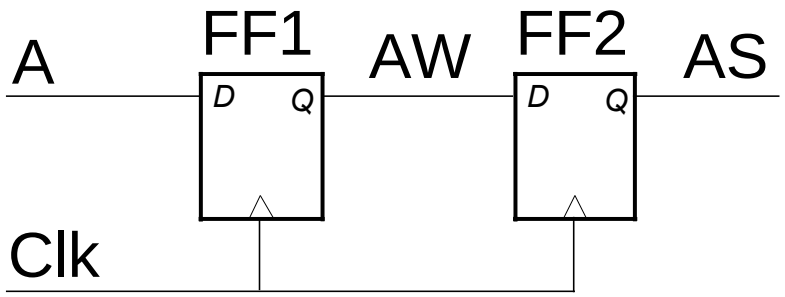


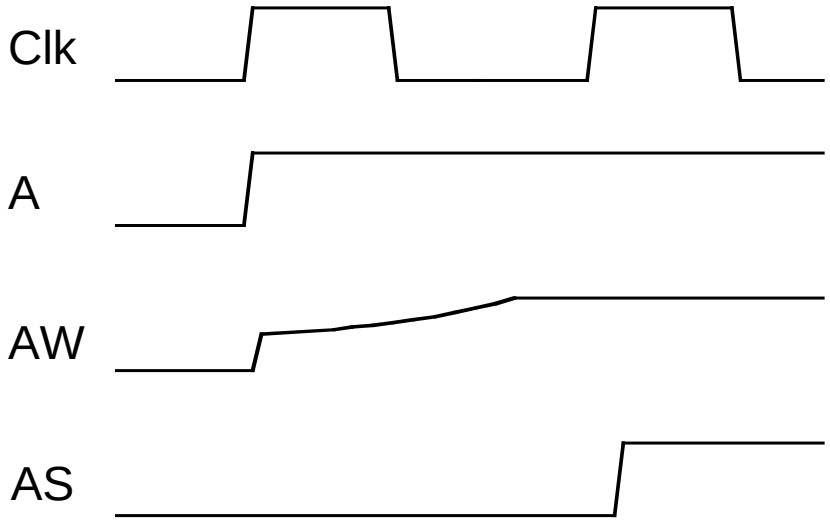
If delays are balanced, ring sequences through six states repeatedly



對設計的影響
(何時最容易發生)

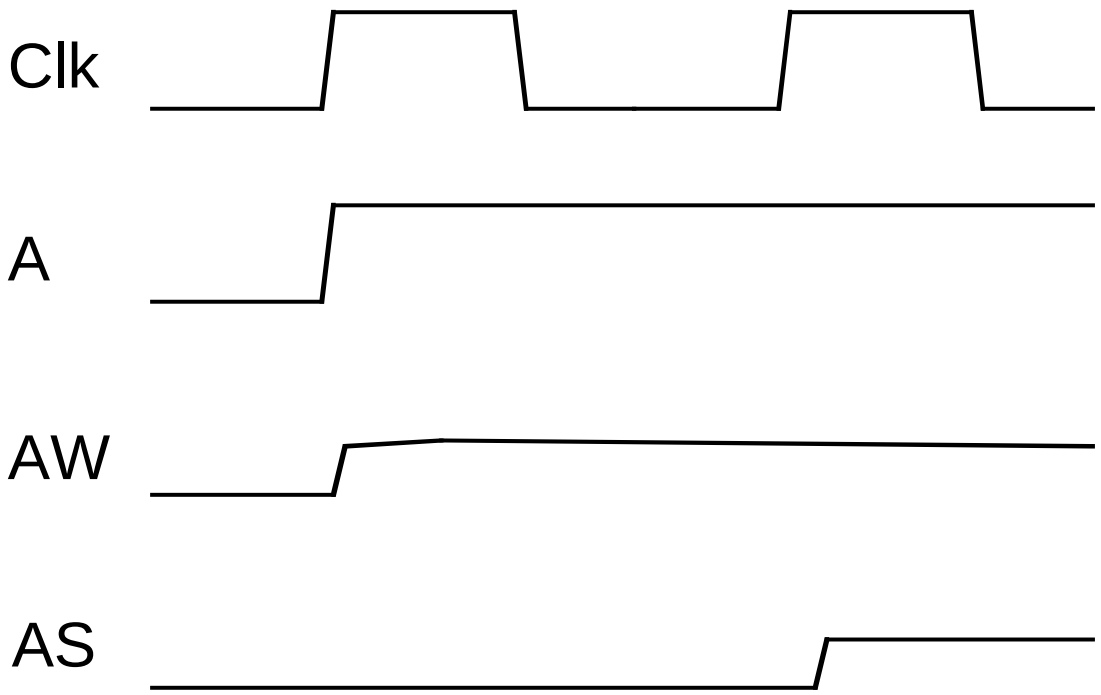
A Brute-Force Synchronizer





(c) 2005-2012 W. J. Dally

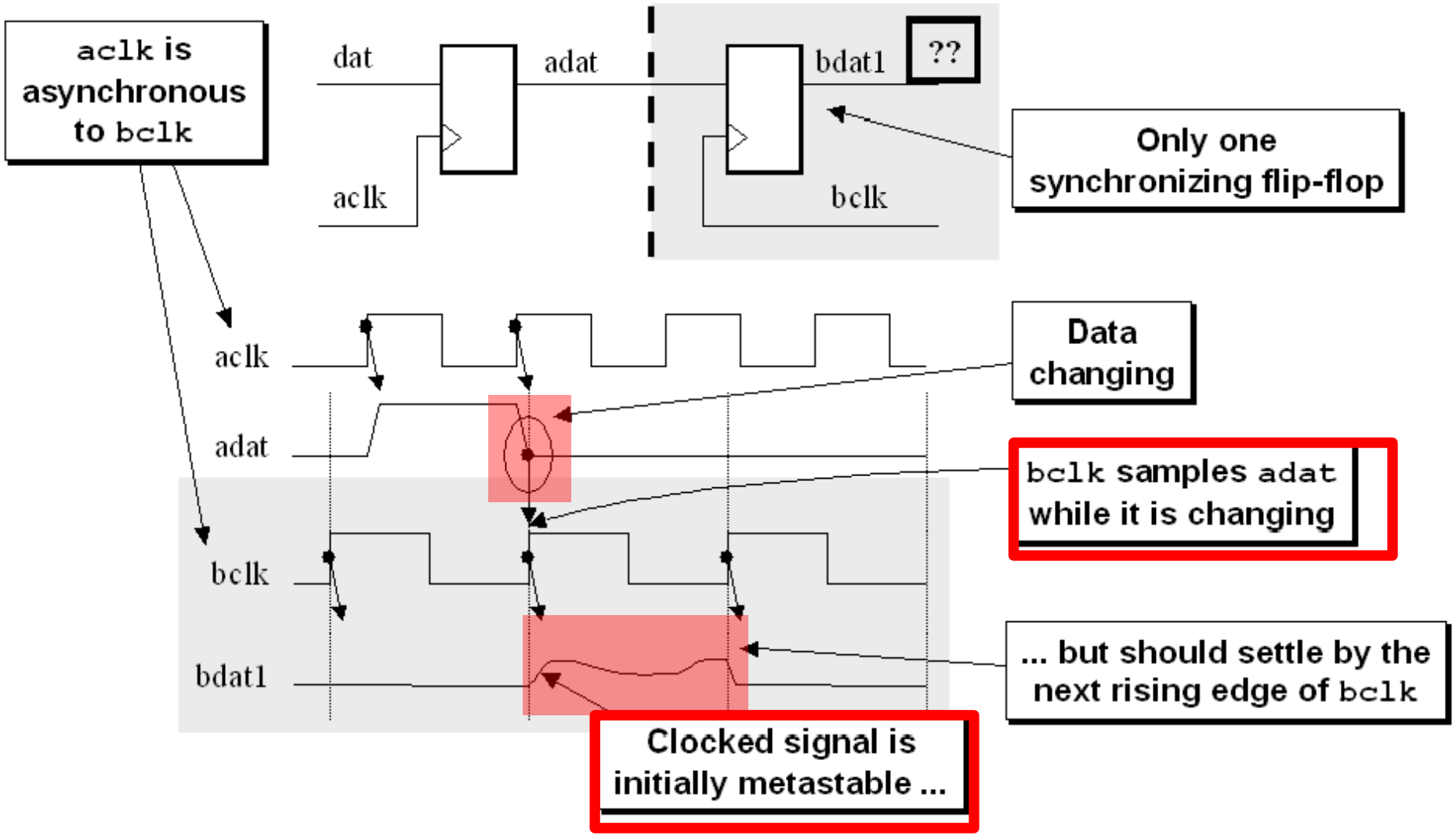
What if AW is *still* in a metastable state when FF2 is clocked?



訊號根本不穩定
會抓錯值

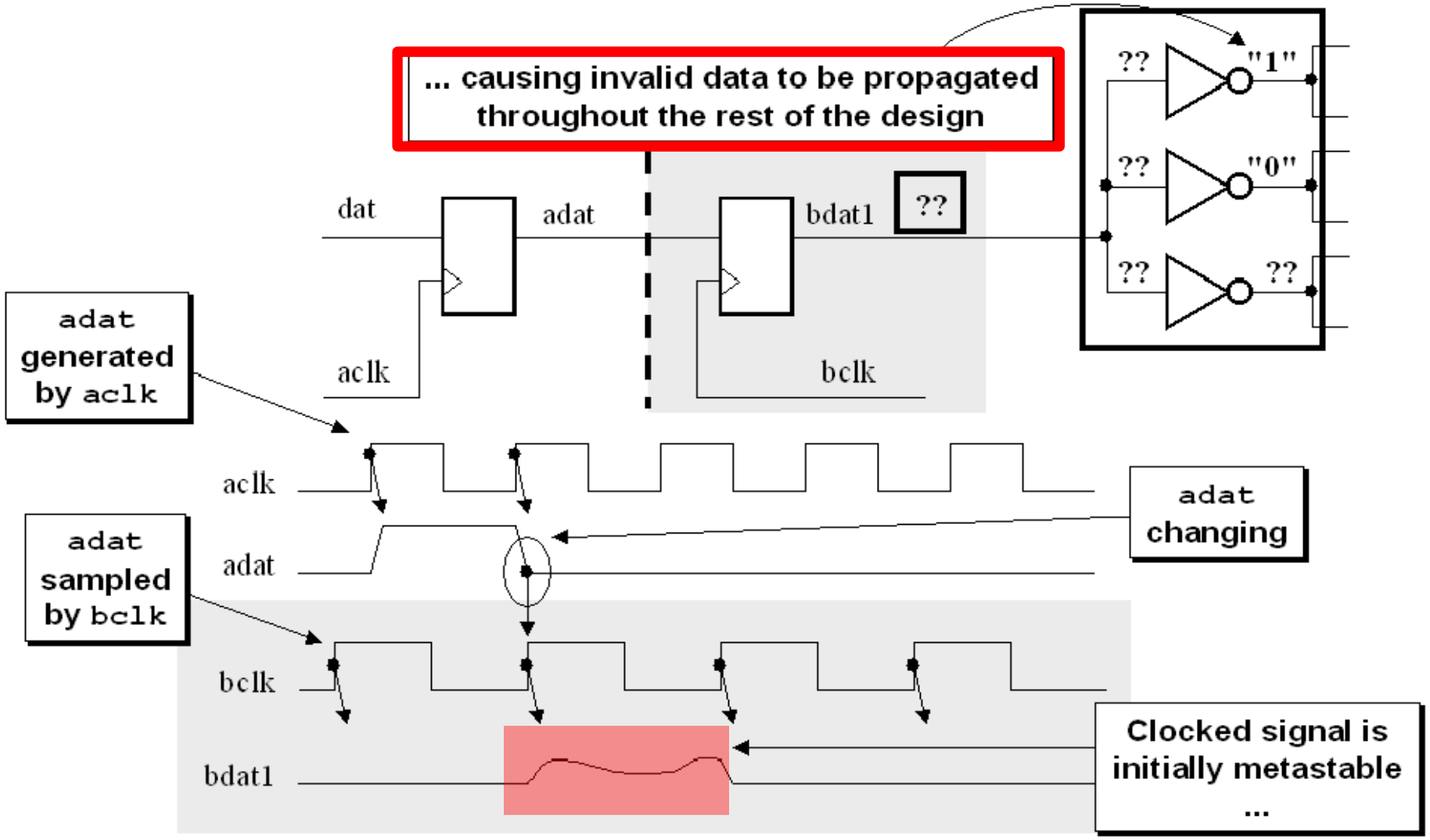
(c) 2005-2012 W. J. Dally

訊號跨越不同clock domain時，最容易發生

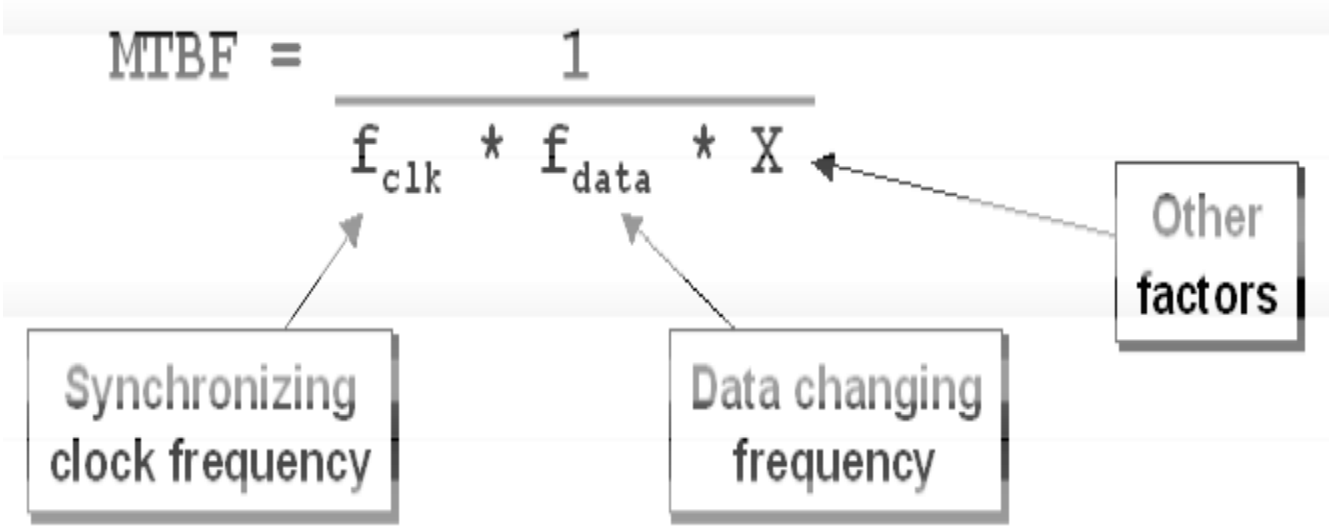


http://www.sunburst-design.com/papers/CummingsSNUG2008Boston_CDC.pdf

Metastable bdat1 output propagating invalid data throughout the design



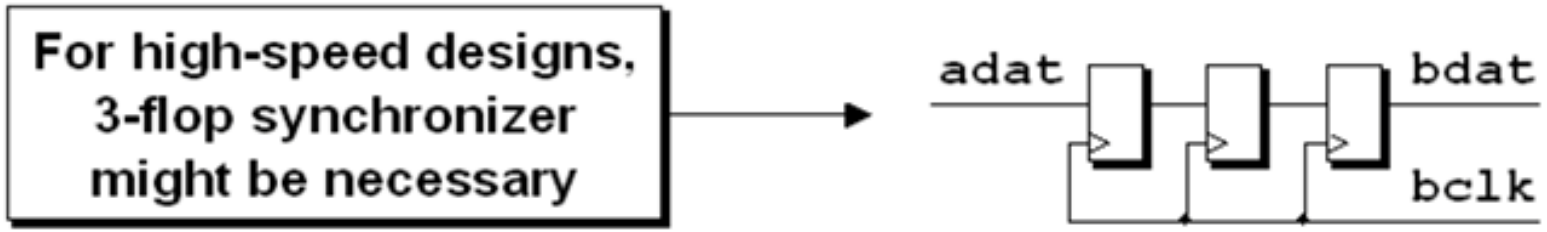
發生機率MTBF (mean time between failure)



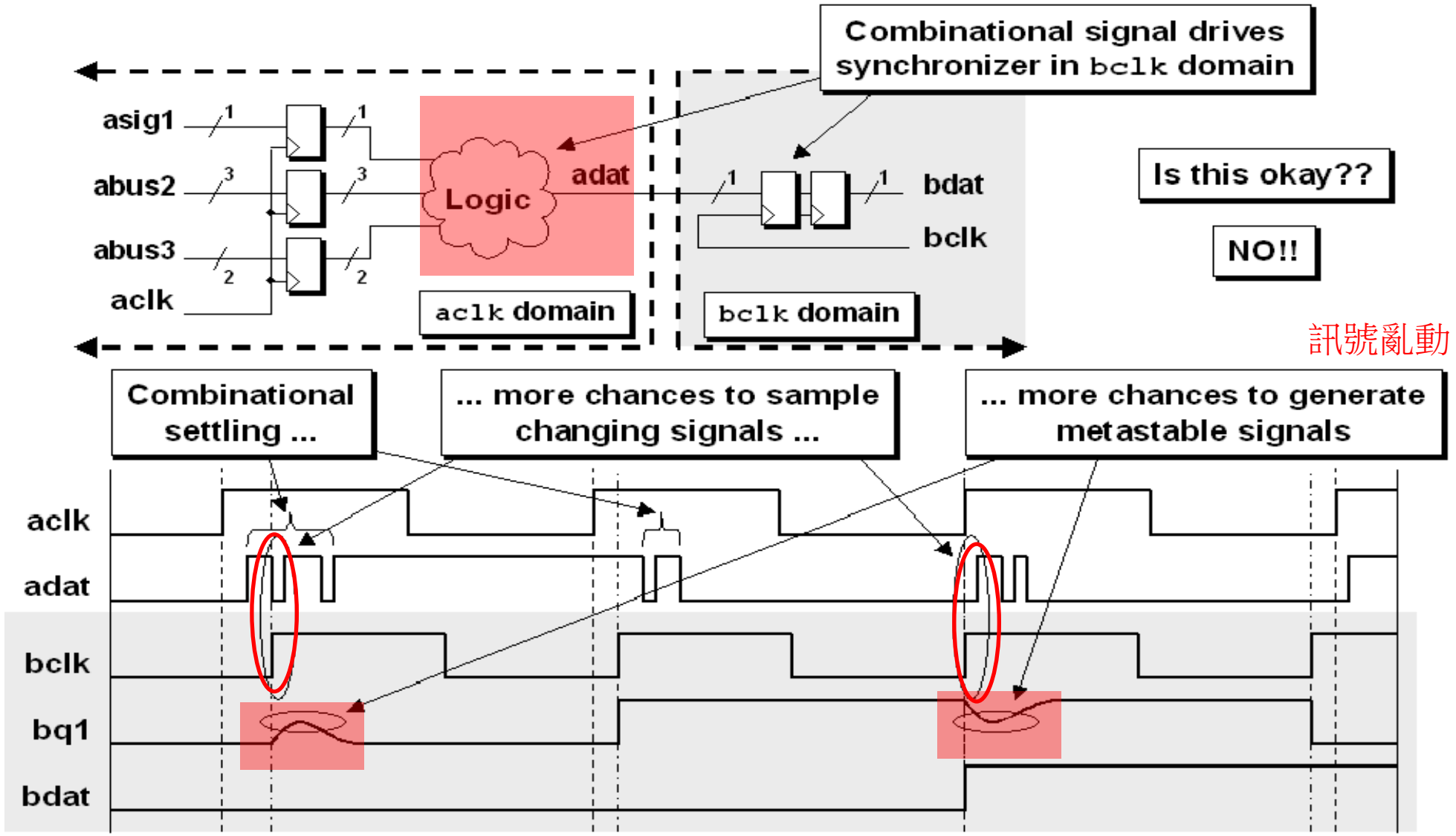
failures occur more frequently (shorter MTBF) in higher speed designs, or when the sampled data changes more frequently.
越高頻或訊號變化越快，越容易產生錯誤

那設計要怎麼做?single bit case

超高頻設計，就用三個DFF串接



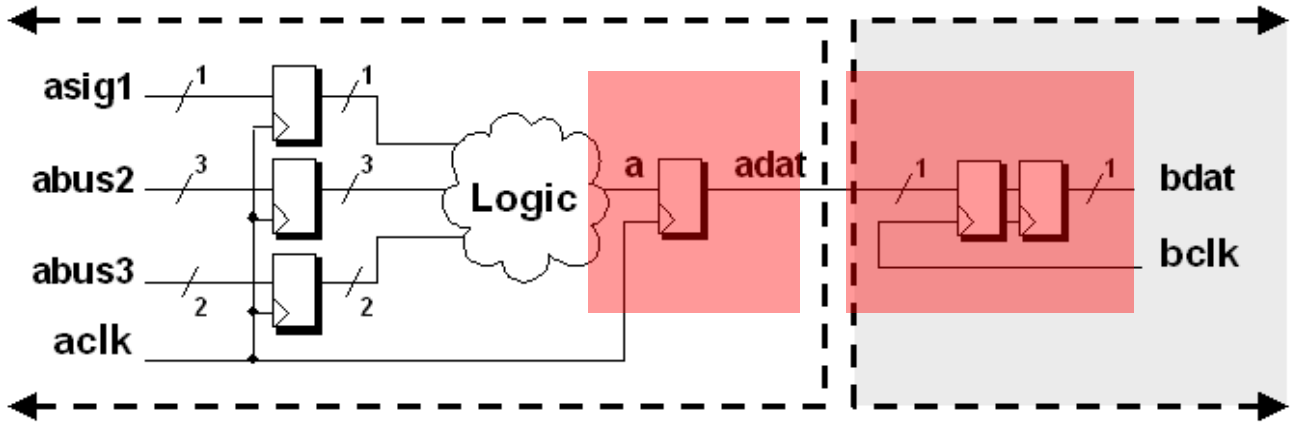
不良示範: 沒經過register就送過去



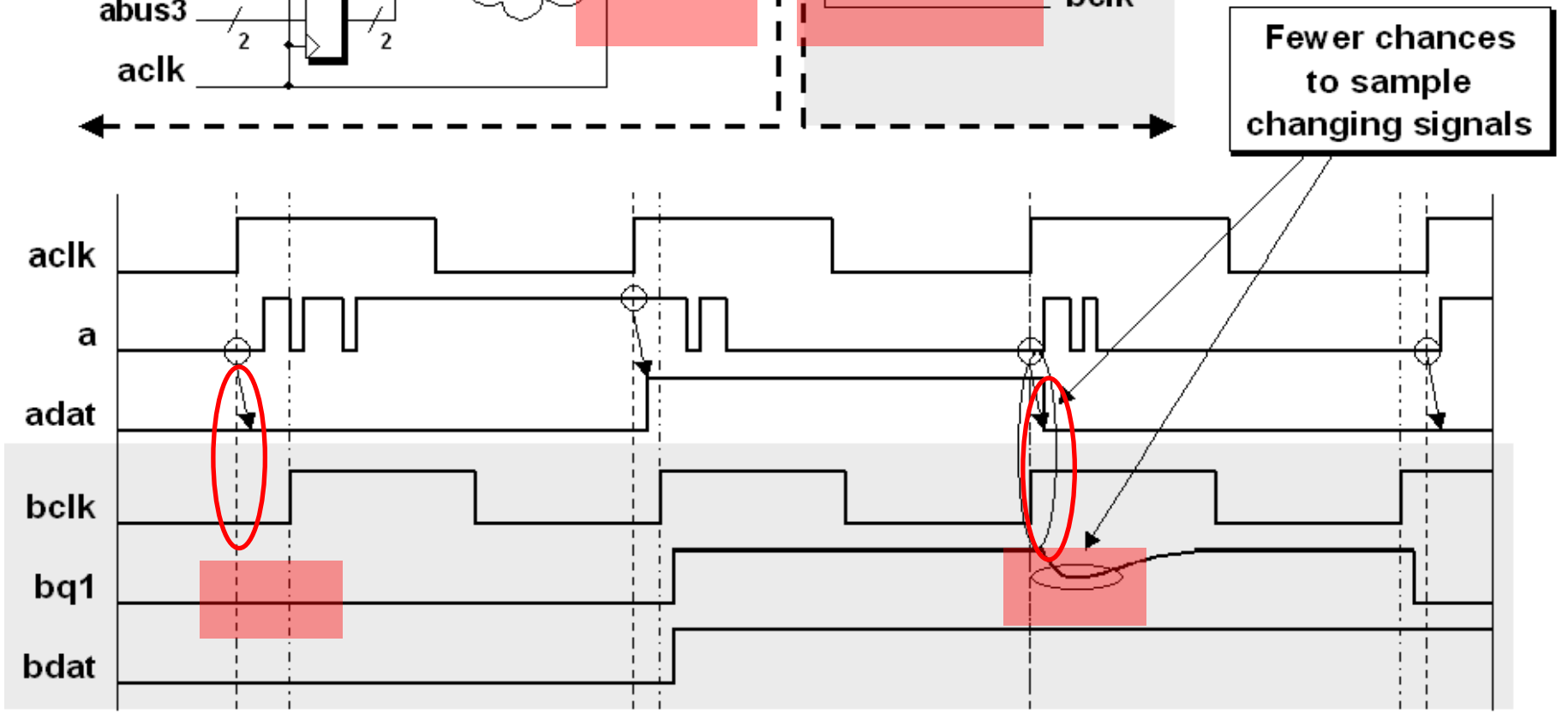
訊號亂動，抓錯機率高

優良設計

發送端經DFF　　接收端用兩顆串接

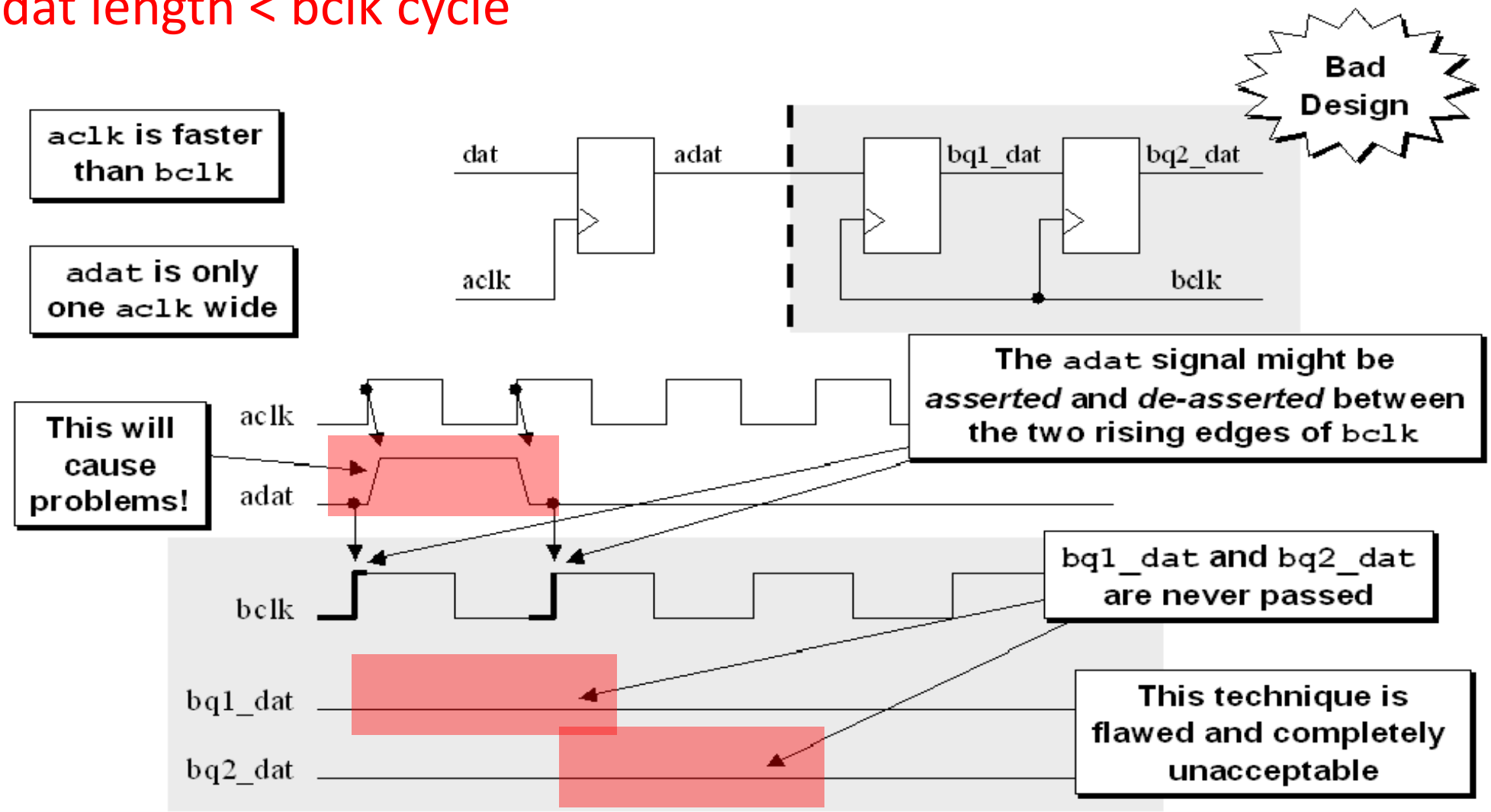


訊號較穩，抓錯機率小



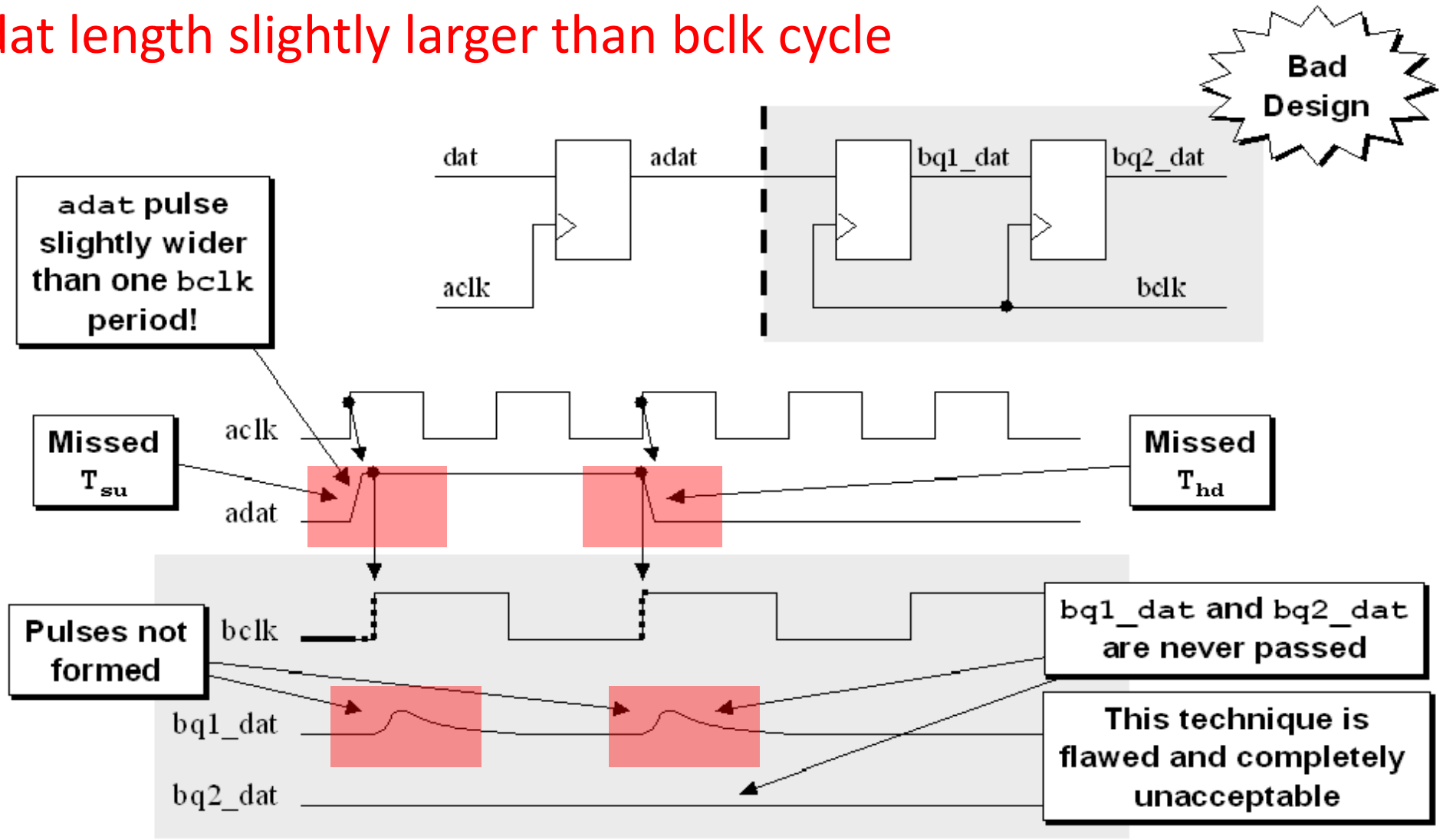
發送端訊號要怎麼給: 不能太短，會被錯過

adat length < bclk cycle

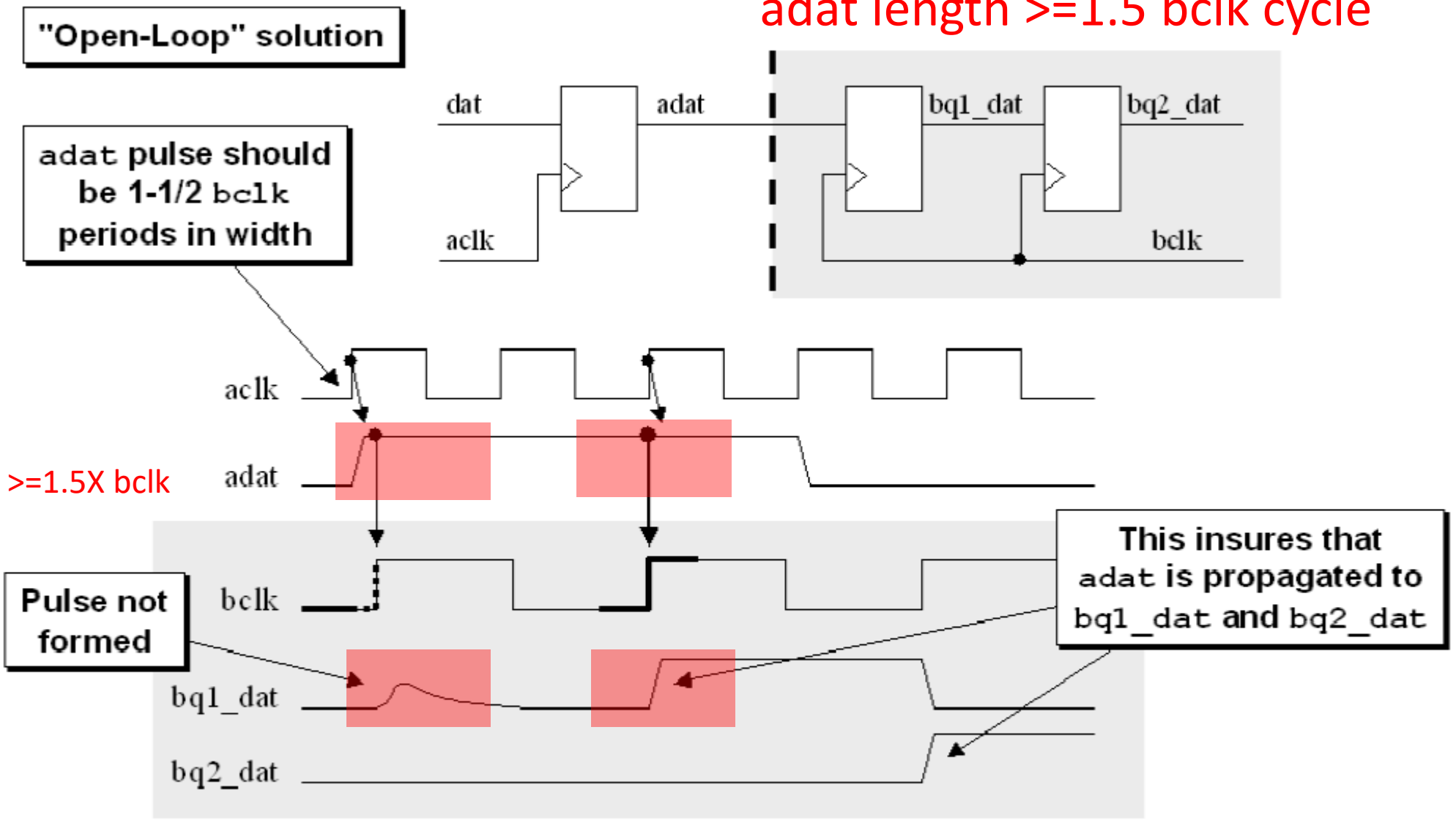


發送端訊號要怎麼給: 不夠長，也會被錯過

adat length slightly larger than bclk cycle

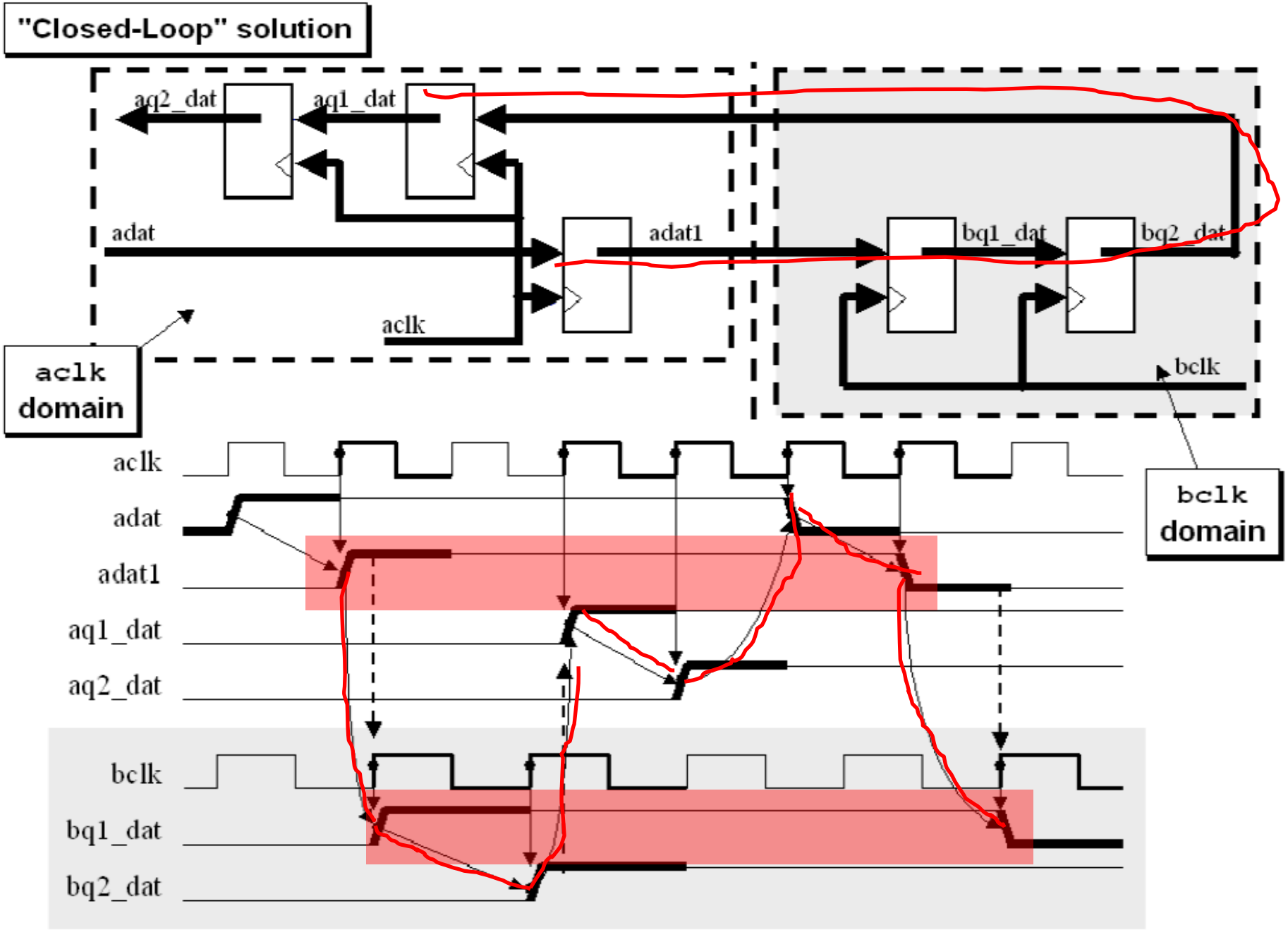


adat length >=1.5 bclk cycle



訊號要夠長，>=1.5X bclk

Figure 10 - Lengthened pulse to guarantee that the control signal will be sampled



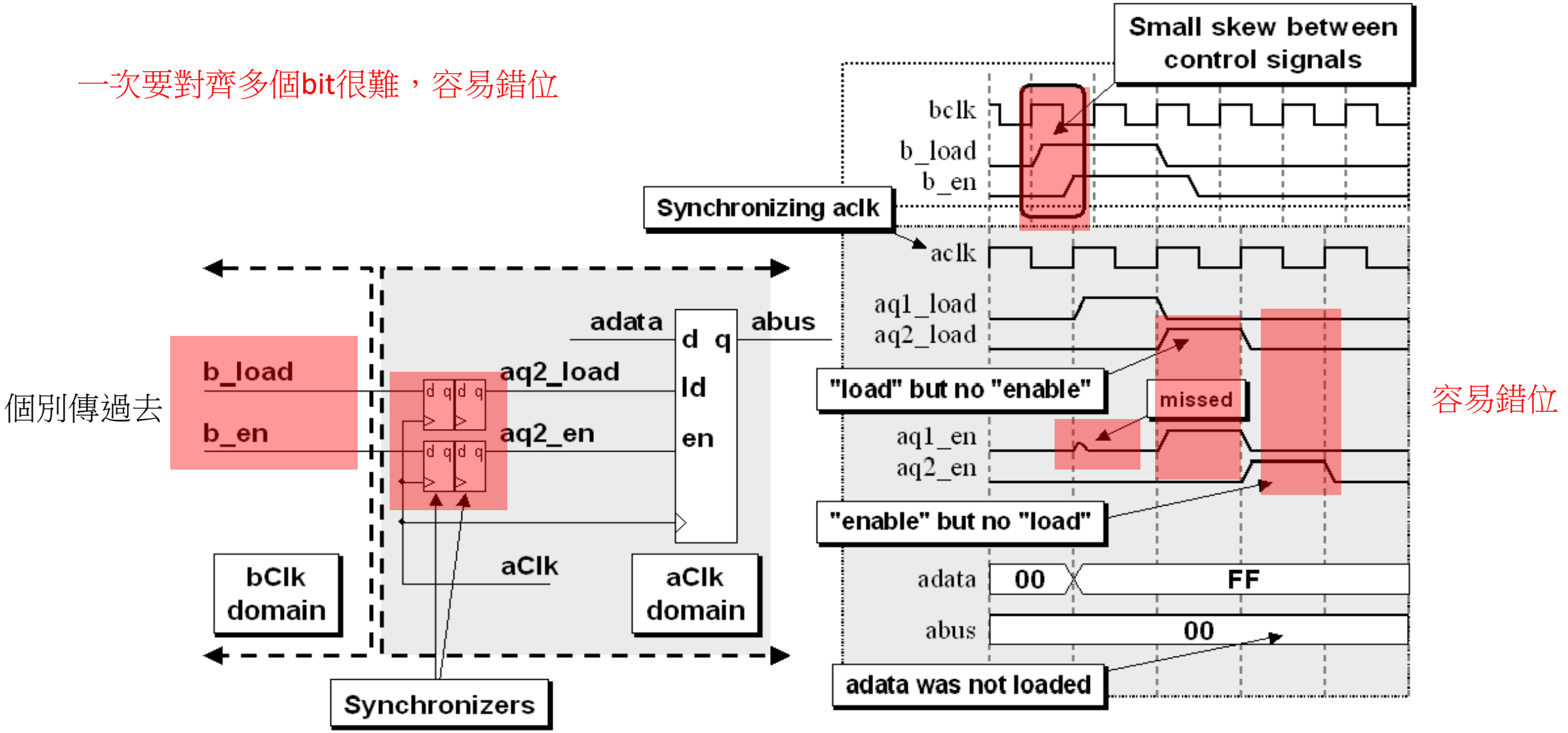
要寄有回條的掛號信

接收端回傳一個訊號，
告訴發送端已經收到，
發送端再關掉訊號

Figure 11 - Signal with feedback to acknowledge receipt

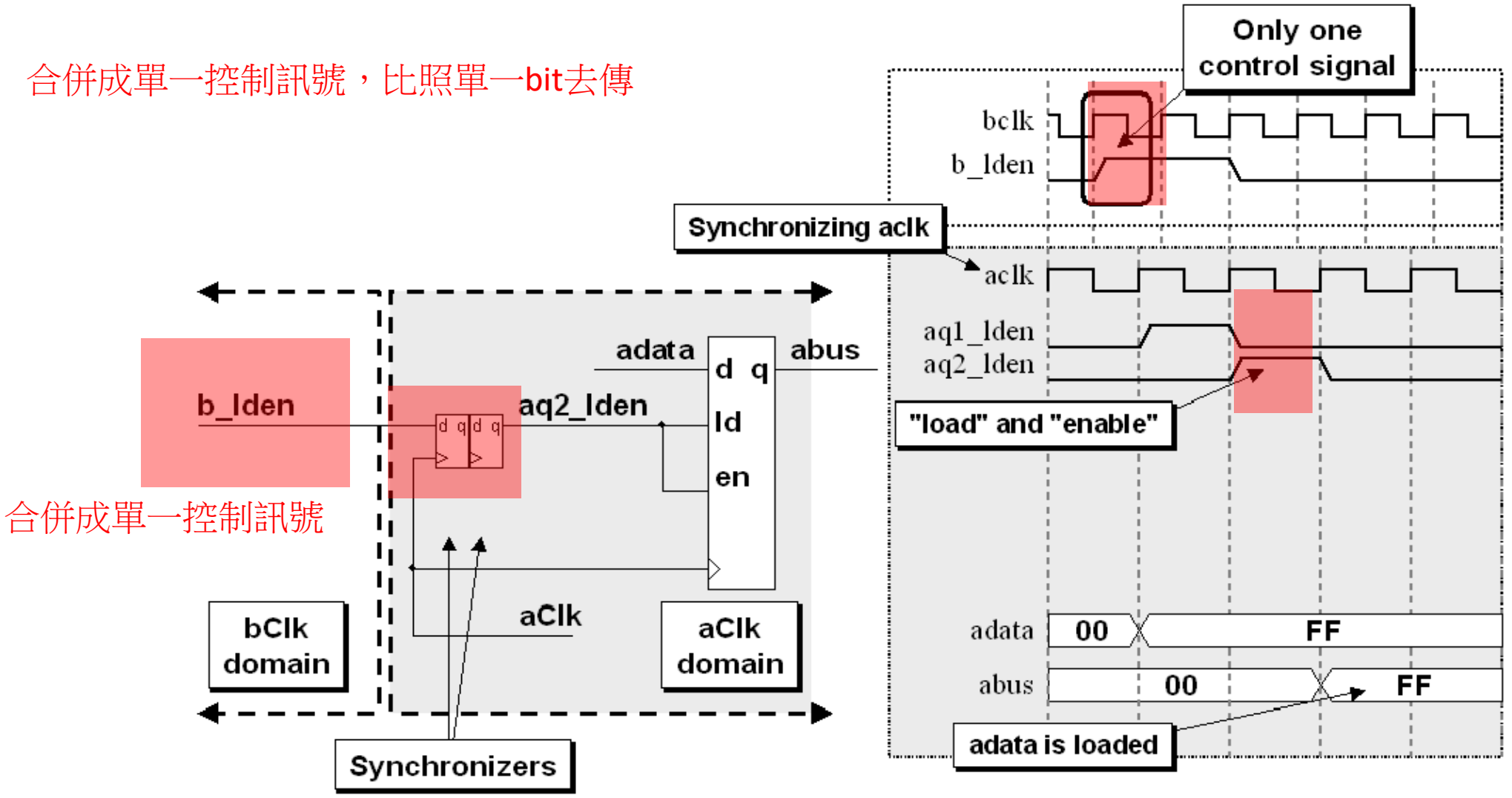
那設計要怎麼做?multi-bit case (control sig)

一次要對齊多個bit很難，容易錯位



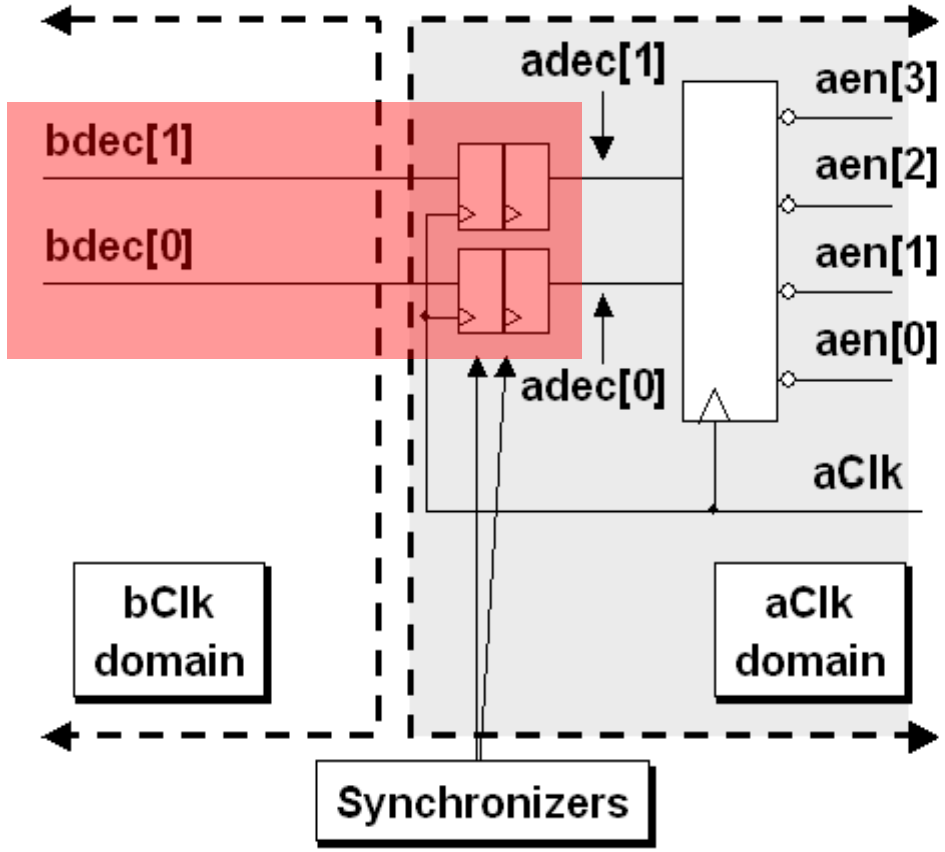
那設計要怎麼做?multi-bit case (control sig)

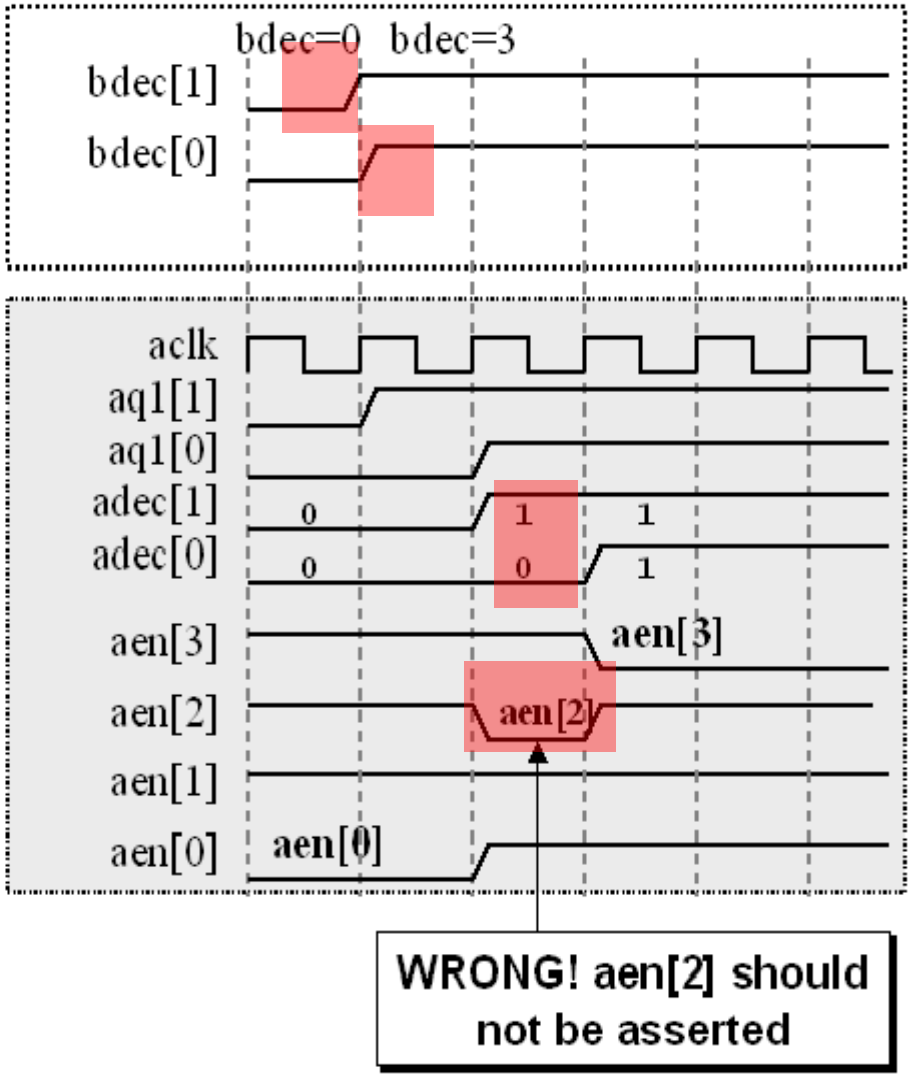
合併成單一控制訊號，比照單一bit去傳



多個data bit 要傳，直接傳會出錯

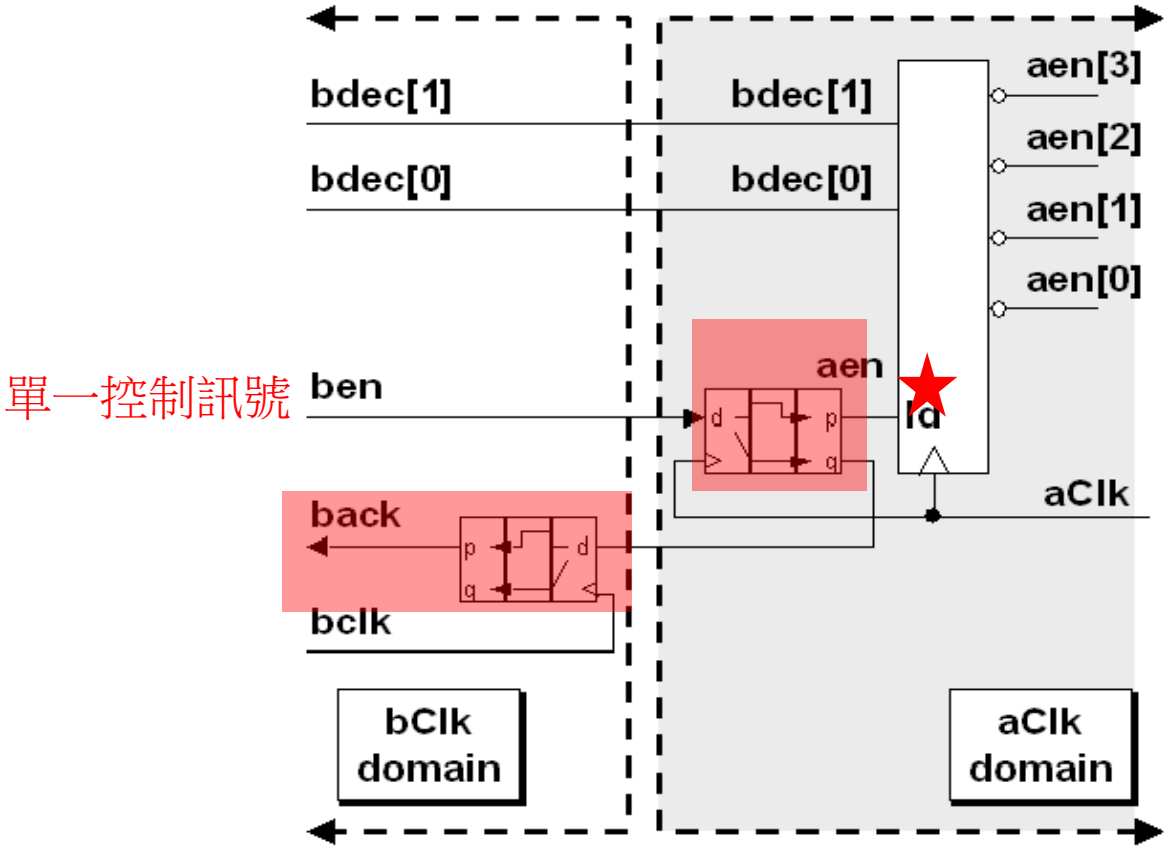
One hot active low address decoder

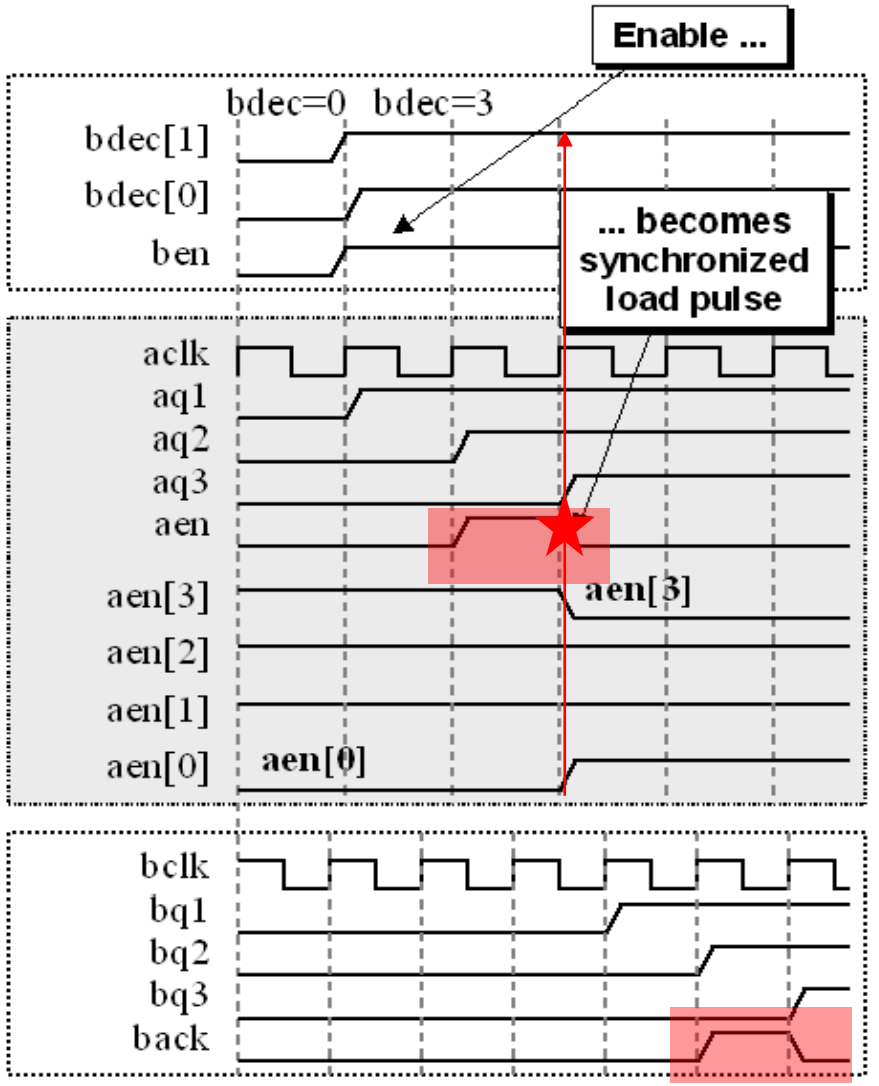




用單一控制訊號過同步器，再去控制data

One hot active low address decoder





傳counter 值

Warning: The Surgeon General has determined that passing binary-coded and one-hot signals through a brute-force synchronizer can be hazardous to your circuits.

(c) 2005-2012 W. J. Dally

Gray Code 是個好主意，一次只有一個bit變

Solution:

Use a Gray code counter

For all but the MSB:

```
next_b[i] = (b[i-1] & !(|b[i-2:0])) ? !xor(b[n-1:i+1]) : b[i];
```

For the MSB:

```
next_b[i] = (|b[i-2:0]) ? b[i] : b[i-1] ;
```

Can we use this Gray code for the head and tail pointers of our FIFO?

(c) 2005-2012 W. J. Dally

```
module GrayCount4(clk, rst, out) ;
  input clk, rst ;
  output [3:0] out ;
  wire [3:0] out, next ;

  DFF #(4) count(clk, next, out) ;

  assign next[0] = !rst & !(out[1]^out[2]^out[3]) ;
  assign next[1] = !rst & (out[0] ? !(out[2]^out[3]) : out[1]) ;
  assign next[2] = !rst & ((out[1] & !out[0]) ? !out[3] : out[2]) ;
  assign next[3] = !rst & (!(|out[1:0]) ? out[2] : out[3]) ;
endmodule
```

(c) 2005-2012 W. J. Dally

```
module gray2bin #(parameter SIZE = 4)
(output logic [SIZE-1:0] bin,
input  logic [SIZE-1:0] gray);

always_comb
  for (int i=0; i<SIZE; i++) bin[i] = ^(gray>>i);
endmodule
```

```
module bin2gray #(parameter SIZE = 4)
(output logic [SIZE-1:0] gray,
input  logic [SIZE-1:0] bin);

  assign gray = (bin>>1) ^ bin;
endmodule
```

```
# xxxx
# 0000
# 0001
# 0011
# 0010
# 0110
# 0111
# 0101
# 0100
# 1100
# 1101
# 1111
# 1110
# 1010
# 1011
# 1001
# 1000
# 0000
# 0001
# 0011
# 0010
```