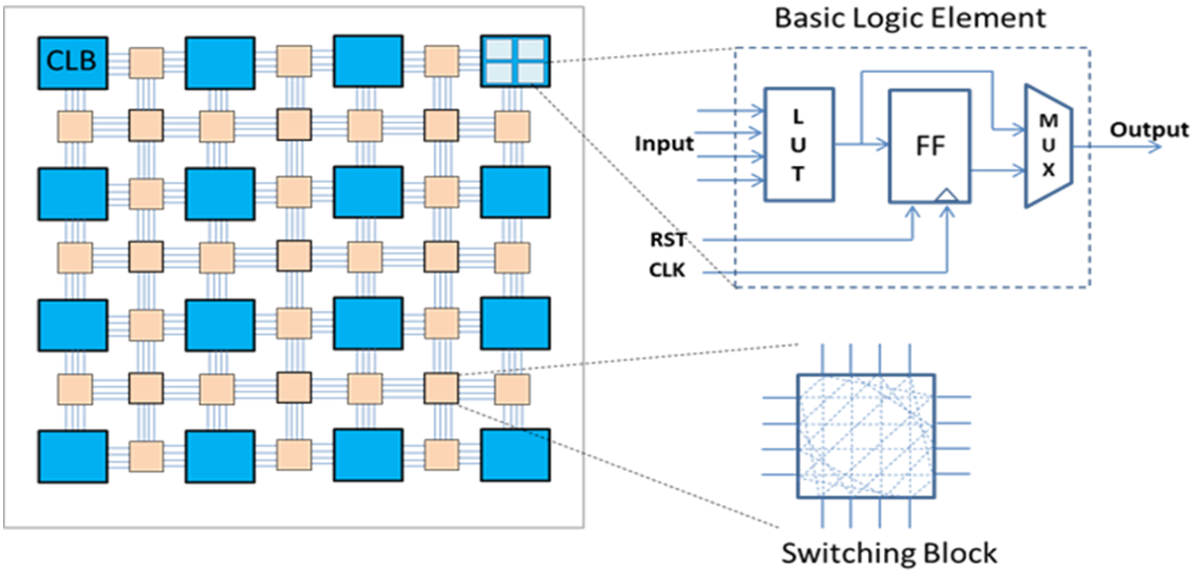
Intern - FPGA

2022/05/05 - FPGA Survey 1

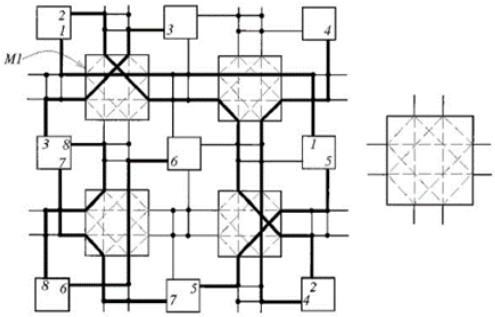
FPGA 資訊

FPGA 組成元素

1. 尋找表（LUT） -------
2. 多工器（MUX）　　| 邏輯單元（CLB）
3. 正反器（FF） --------
5. 線路（Wires）
6. 輸入輸出（I/O）
7. 其他嵌入 Block ------ Ex. DSP

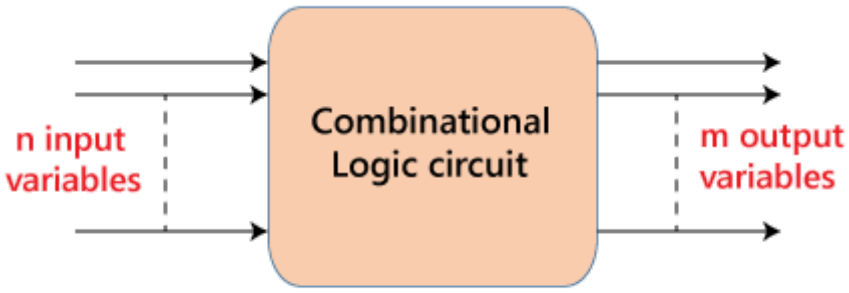


Routing Instance and an S Block



組合邏輯電路

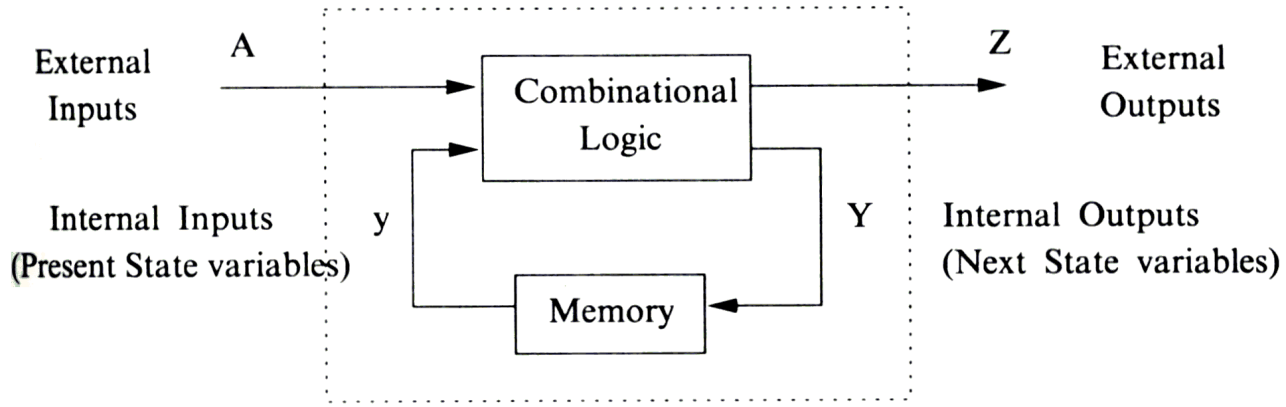
任一時刻的穩態輸出，僅僅與該時刻輸入變量的取值有關。意思是根據目前的 Input 產生對應的 Output，之間不受 Clock 影響。



Block diagram of a combinational circuit

循序邏輯電路

輸出會 Feedbake 回輸入，任一時刻的穩態輸出，與該時刻輸入變量還有前一刻輸入變量的取值有關。



HDL 資訊

Blocking & Non-Blocking

這是 Verilog 中兩種不同的給值的方式。Verilog 中的 Blocking 語句等同軟體語言一樣，是一行一行由上至下執行的，但 Non-blocking 則是同步執行的。

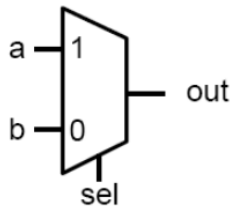
電路都使用 Blocking 的方式設計會造成電路串連太長，導致延遲太多時間。
電路都使用 Non-blocking 的方式設計會造成電路面積加大（成本提高），因為並行處理的輸出都要額外給予一個暫存器來儲存。

- 基本原則：

    1. 組合邏輯 assign 採用 Blocking，且必須搭配 wire。
    2. 循序邏輯無 Clock 的 always 區塊採用 Blocking，且必須搭配 reg。
    3. 循序邏輯有 Clock 的 always 區塊採用 Non-blocking，且必須搭配 reg。
    4. 一個 always 區塊中不能同時使用 Blocking 與 Non-blocking。

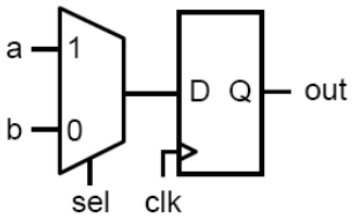
Combinational

```
module combinational(a, b, sel,
                     out);
  input a, b;
  input sel;
  output out;
  reg out;

  always @ (a or b or sel)
  begin
    if (sel) out = a;
    else out = b;
  end

endmodule
```



Sequential

```
module sequential(a, b, sel,
                  clk, out);
  input a, b;
  input sel, clk;
  output out;
  reg out;

  always @ (posedge clk)
  begin
    if (sel) out <= a;
    else out <= b;
  end

endmodule
```



- ***Blocking assignment:*** **evaluation and assignment are immediate**

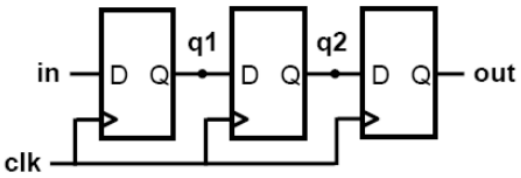
```
always @ (a or b or c)
begin
  x = a | b;          1. Evaluate a | b, assign result to x
  y = a ^ b ^ c;      2. Evaluate a^b^c, assign result to y
  z = b & ~c;         3. Evaluate b&(~c), assign result to z
end
```

- ***Nonblocking assignment:*** **all assignments deferred until all right-hand sides have been evaluated (end of simulation timestep)**

```
always @ (a or b or c)
begin
  x <= a | b;         1. Evaluate a | b but defer assignment of x
  y <= a ^ b ^ c;     2. Evaluate a^b^c  but defer assignment of y
  z <= b & ~c;        3. Evaluate b&(~c) but defer assignment of z
end                   4. Assign x, y, and z with their new values
```

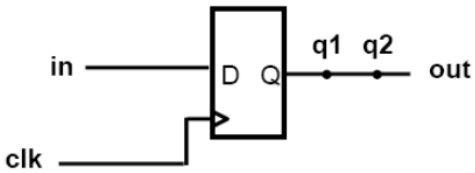
```
always @ (posedge clk)
begin
  q1 <= in;
  q2 <= q1;
  out <= q2;
end
```

"At each rising clock edge, *q1*, *q2*, and *out* simultaneously receive the old values of *in*, *q1*, and *q2*."




```
always @ (posedge clk)
begin
  q1 = in;
  q2 = q1;
  out = q2;
end
```

"At each rising clock edge, *q1* = *in*.
After that, *q2* = *q1* = *in*.
After that, *out* = *q2* = *q1* = *in*.
Therefore *out* = *in*."

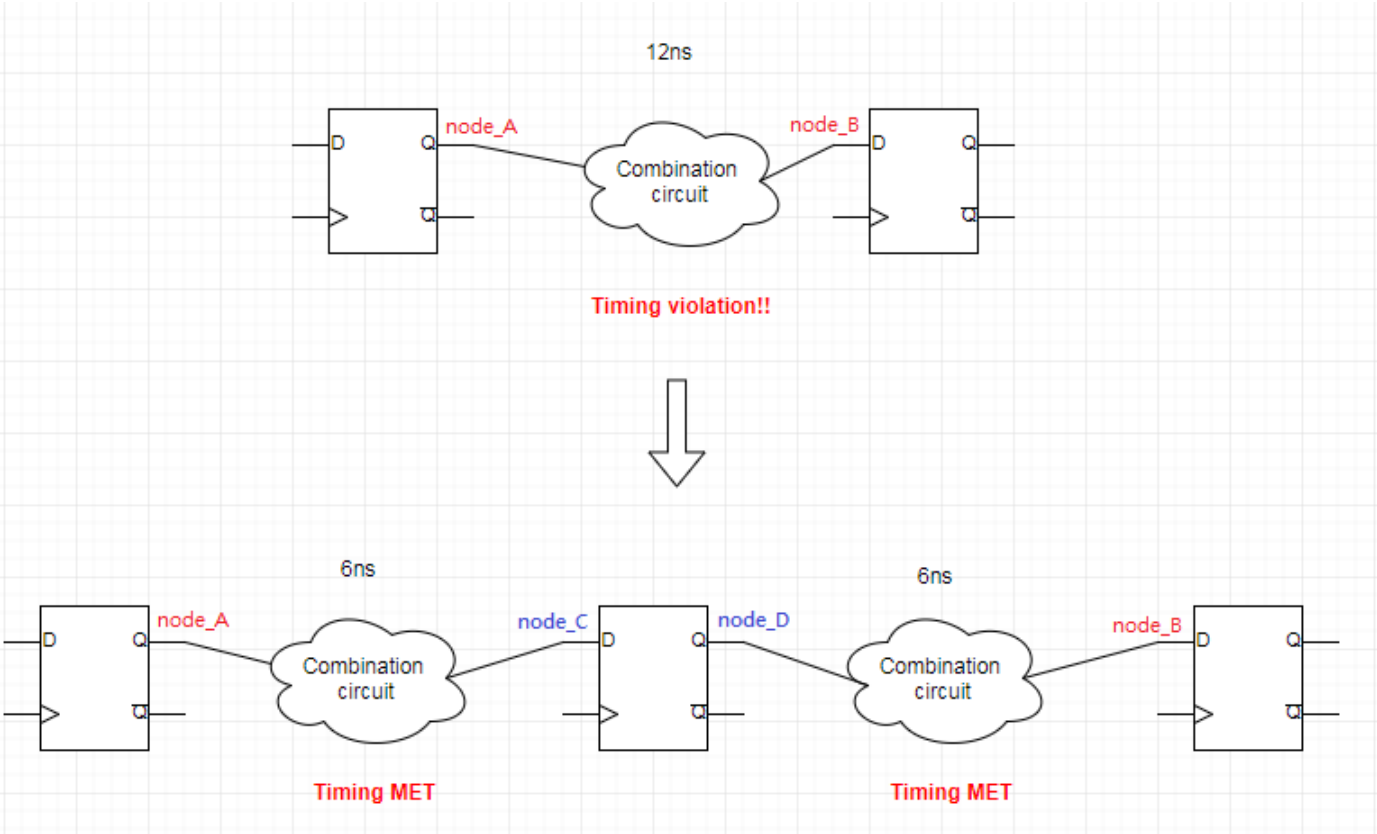


時序資訊

Setup Time & Hold Time

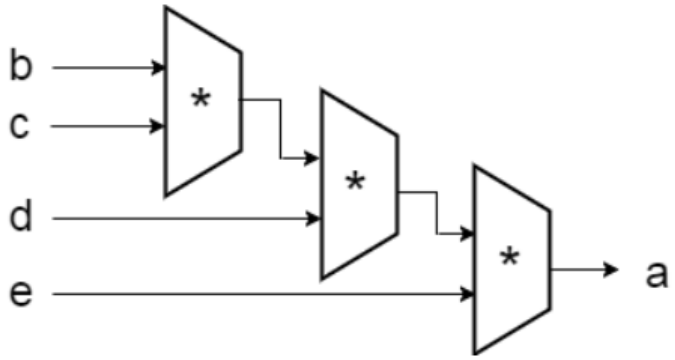
Setup time：Clock 上升前，存進暫存器前需維持一段穩定的時間，才能保證存進暫存器的值沒有問題，這段需維持穩定的時間就稱為 Setup time，遇到 Setup time violation 的話，最簡單的方法就是根據 Violation 的資訊找出有問題的 Path，然後多加一層 Register 進去。

會遇到 Timing 問題的通常就像下圖，Combination 運算太過複雜，一個 Clock 做不完邏輯運算，所以拆成兩個 Clock 去運算，原本的邏輯運算需要 12 ns，如果 Run 在 100MHz (10 ns) Clock rate 的板子下就會有 Timing violation，所以找到問題的 Path，在中間多加一個 Register 進去。



上述有效解決 Timing 問題，但會影響 Performance，原本預計一個 Clock 就東西突然要變成兩個 Clock，效率突然降兩倍，為此嘗試用 Pipeline 解決 Timing violation。

假設 Combination circuit 運算是 a <= bcde;

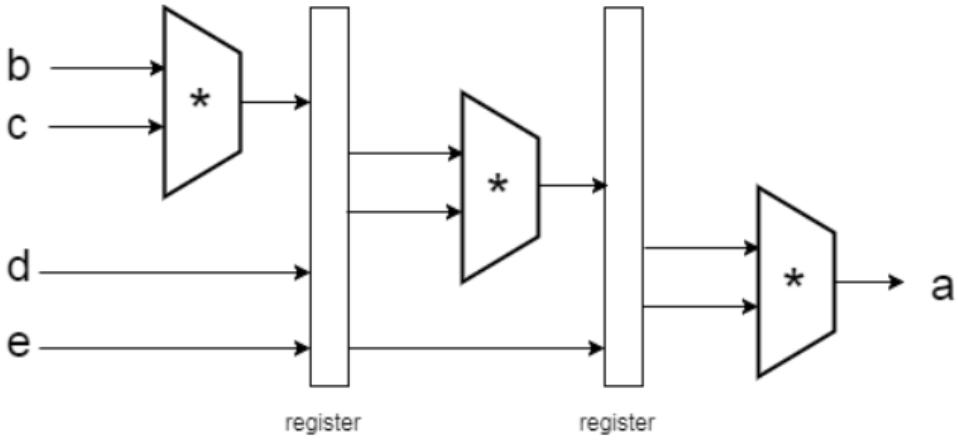


一般

□□□□ | □□□□ | □□□□
指令 1　　指令 2　　指令 3

如果單指令所需時間是 10 + 15 + 10 + 5 = 40 ns
總共執行時間為 40 * 3 = 120 ns

Run 在 1000 MHz (100 ns) Clock rate 的板子下有問題



Pipeline

□□□□　　指令 1
　□□□□　　指令 2
　　□□□□　　指令 3

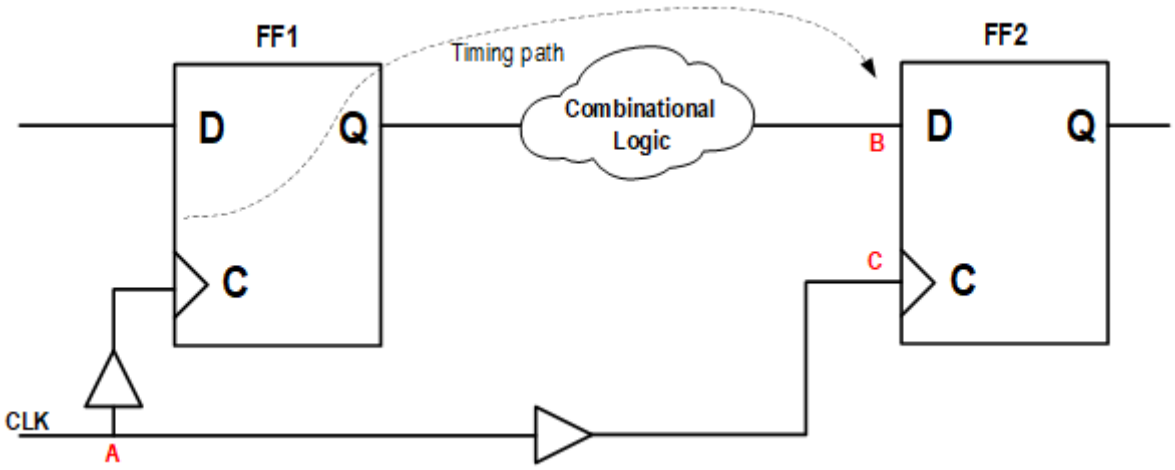
單個 Clock cycle 所執行時間 max{10, 15, 10, 5} = 15 ns
共執行了 6 個 Clock cycle，執行時間為 15 * 6 = 90 ns

可以 Run 在 1000 MHz (100 ns) Clock rate 的板子下

Pipeline 能維持一個 Cycle 就算出結果，但相對要多花額外的資源去儲存前一筆的資料，用 Pipeline 除了加速以外，資源也要納入考量，若花了很多資源去加速一個不常用到的電路是挺浪費的。

Hold time：clock 上升後，暫存器的值需穩定一段時間，才能保證傳到下一層時的值是正確的，這段穩定的時間就稱為 Hold time，遇到 Hold time violation 時，可以加幾個 buffer 緩衝。

首先 CLK（A）觸發後，有兩條路徑會同時開賽跑，一條從 FF1-C 跑到 FF1-Q 再跑到 FF2-D 這條路徑（虛線）的時間假設為 T1，另一條直通到 FF2-C 假設為 T2，如果 T2 比 T1 快，那 FF2 的 D 資料就會因為還沒有穩定直接被採樣。



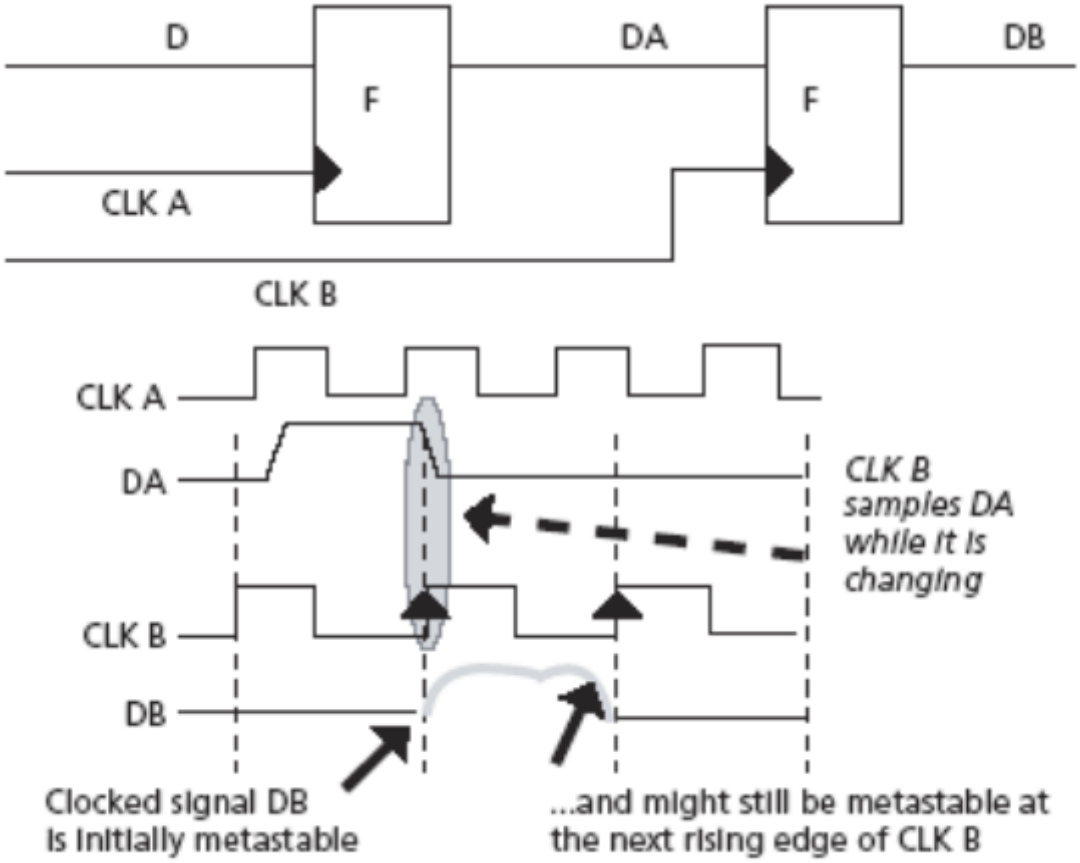
Clock Domain 資訊

Clock Domain

由某一頻率 Clock 所驅動的電路為一個 Clock Domain。

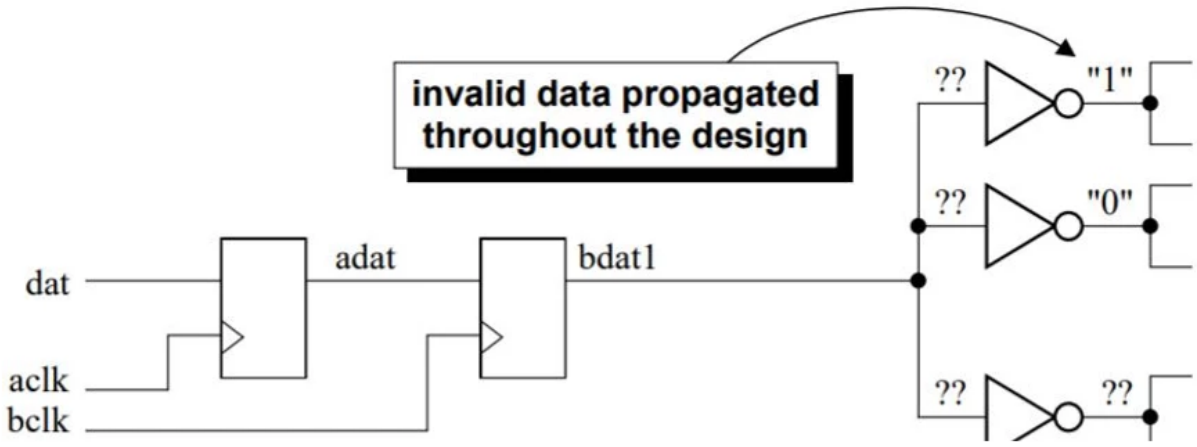
Metastability

有兩個 FF（FF-A、FF-B），分別由兩個不同的 CLK 所驅動（不一定誰的頻率快誰慢），當 DA 隨著 CLK A 產生一個 Cycle 訊號，CLK B 需要在一個 Cycle 內將訊號鎖進 FF，但由於兩個 Clk Frequency 不同，當 DA 變為 0 的瞬間，CLK B 的 Posiedge trigger 正要把 DA 的值存進 DFF-B，由於 DA 的訊號驟降讓 DFF-B 沒有足夠的準備時間來完成鎖值，此時就會發生 Metastable（亞穩態，或稱準穩態）。



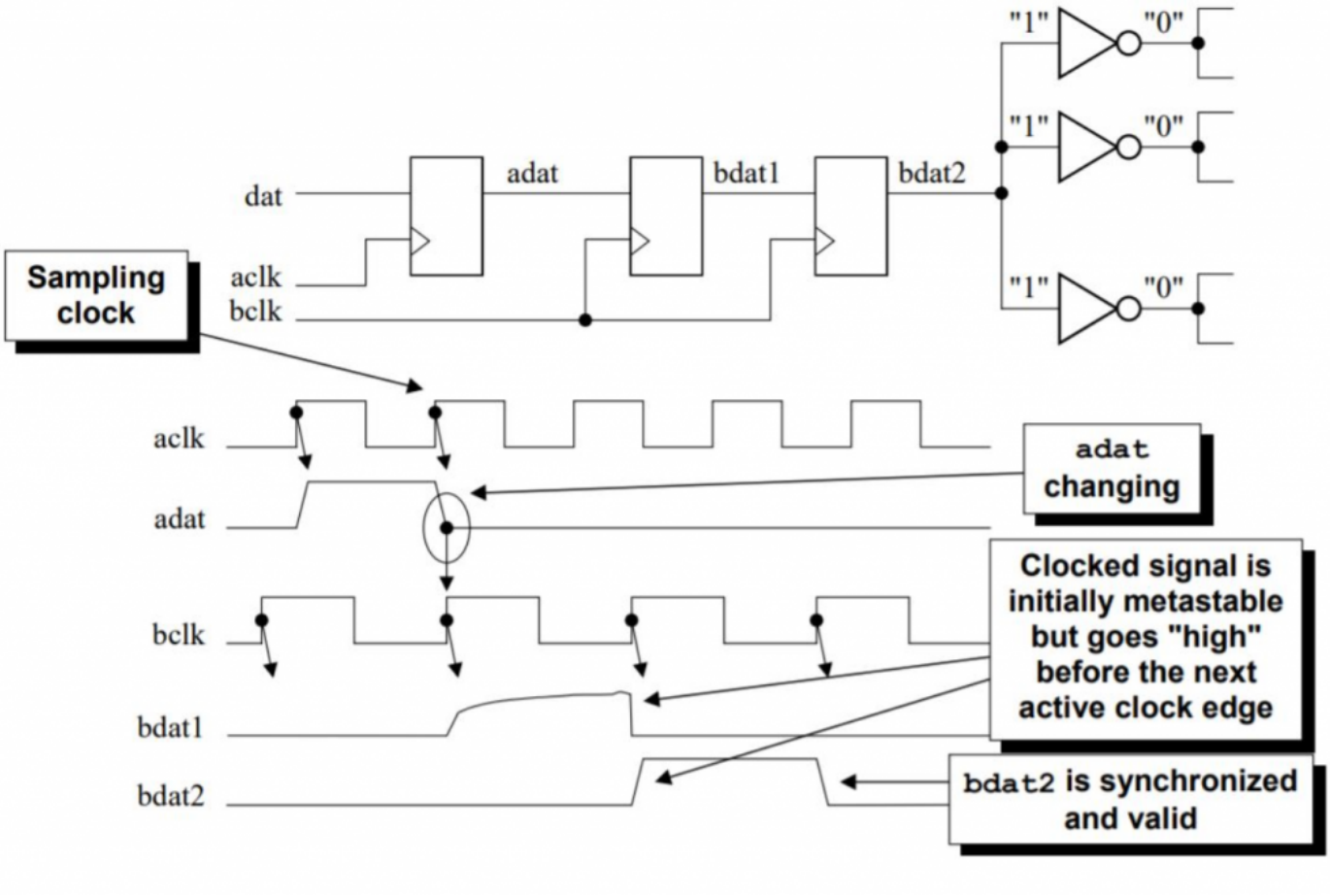
沒有足夠的準備時間的意思是？

正反器由許多 CMOS（互補式金屬氧化物半導體）所組成，CMOS 要存值就需要足夠的"充電"時間（Setup time）。



當 bdat1 處於 Metastable 時，後面如果分接到三個 Not gate，不能保證每個 Not gate 的輸出會是高電位還是低電位，因為輸入處於 Metastable，輸入的電流不穩定，有的 Gate 認為接收到的是高電位，有的卻認為接收到低電位，當 bdat1 接到多個 Combination circuit 時，電路的功能就可能會出錯，造成系統故障。

那怎麼解決？bdat1 後面不要接上任何組合電路，再接一級由 bclk 驅動的 FF，這就是 Two Flip-Flop Synchronizer。處理 Metastable 的最基本的步驟就是遵守 Two Flip-Flop 方法。



CDC 設計

提高平均故障間隔（MTBF）從而降低亞穩態發生的概率，處理方法可以分為兩大類：單 Bit 訊號 CDC 處理、多 Bit 訊號 CDC 處理。多 Bit 訊號的傳遞不光只有亞穩態這一個問題，還可能會因為多個訊號之間由於工藝、PCB 佈局等因素導致的訊號傳輸延時（Skew），從而導致訊號被漏採或者錯採。反正就是要想辦法"同步"。

常見的同步方法有：

1. m-FF based Synchronizers.
2. MUX based Synchronizers.
3. Handshake Synchronizer.
4. Toggle Synchronizers.

m-FF based synchronizers 使用條件：

1. source clock domain is slower than destination clock domain.
2. clock domain crossing (CDC) is on a control signal (either single bit or multibit).

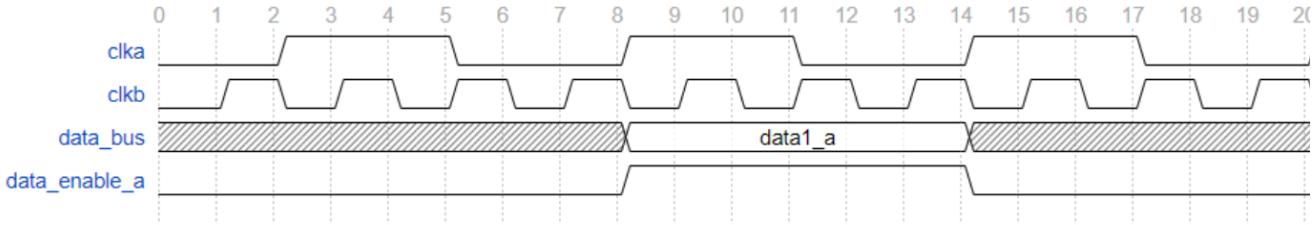
概念與圖示如上所示。

MUX based synchronizers 使用條件：

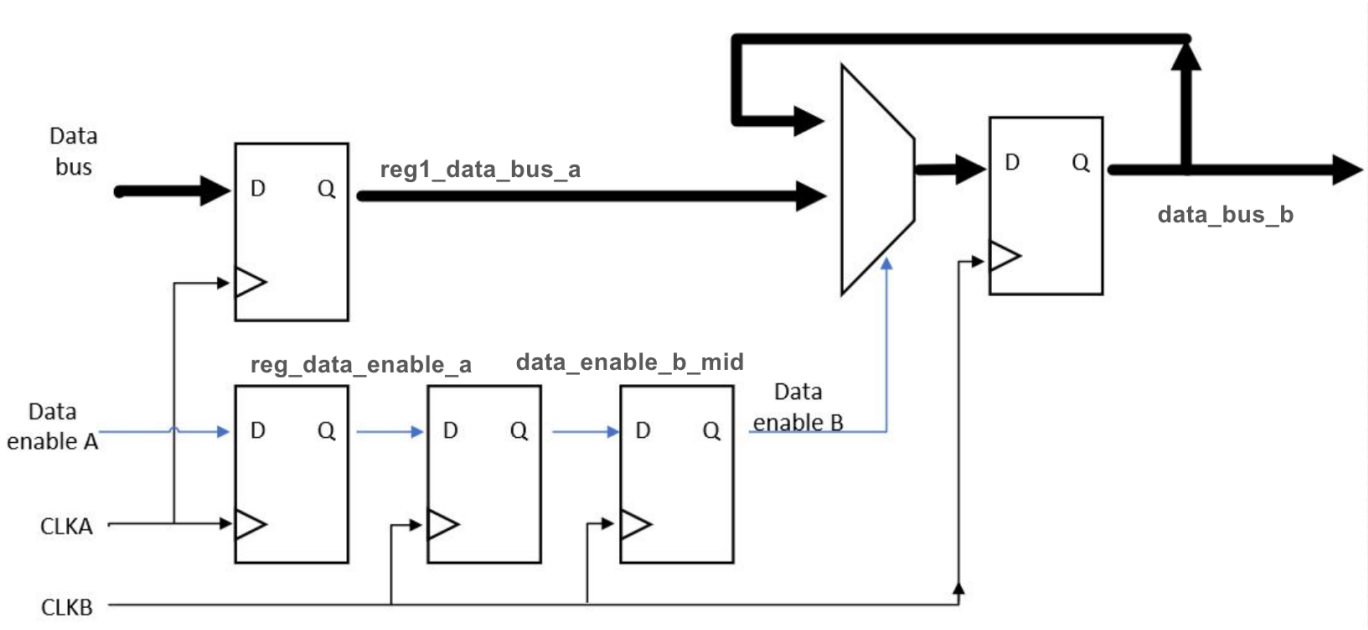
1. The source clock domain is slower than destination clock domain.
2. Clock domain crossing (CDC) is on a data bus.

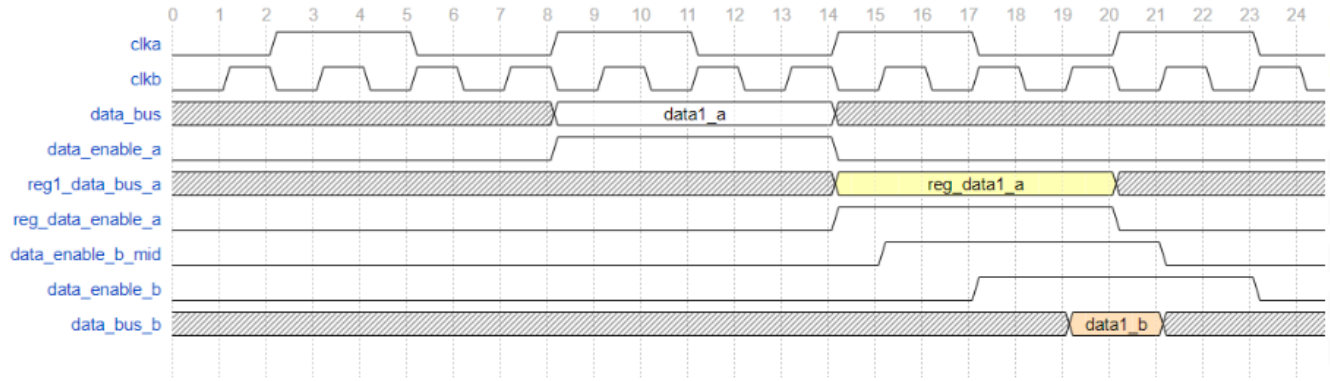
3. Data is stable for at least m+1 clock cycles.

MUX Synchronizers 要求被同步的資料，跟隨一個 Enable Singal。



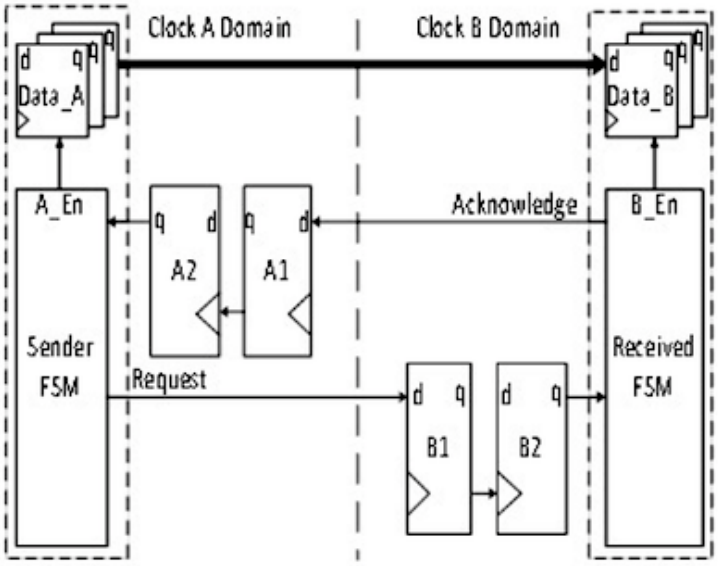
想要將 Data bus 從 CLKA 轉到 CLKB，通過 Data bus 的 valid 訊號，即為 Data enable A；將 Data enable A 使用 Two Flip-Flop Synchronizer 跨到 CLKB，也就是 Data enable B ，並且使用 Data enable B 當作最右邊 DFF 的 Flip-Flop Enable（在圖中使用 MUX 示意）。由於 Data enable A 的時序等同於 Data bus，跨到 Data enable B 時也就保證了 data bus 穿過 DFF 的訊號已經穩定，即可拿來鎖入最後一級 DFF，最後一級 DFF 的 Q 即是 Data bus 存在於 CLKB Domain 的穩定信號。

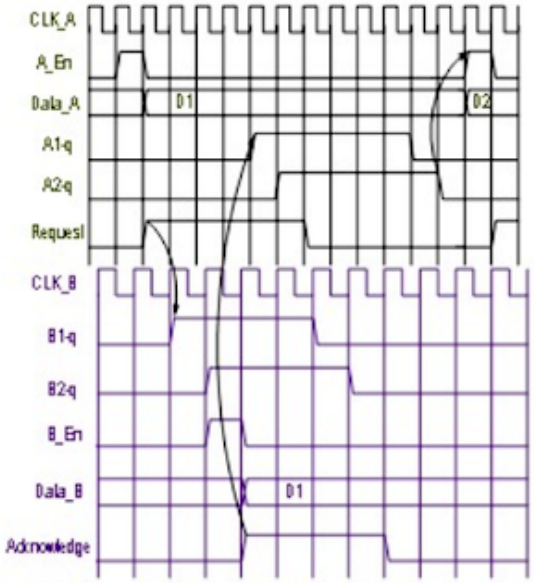




Handshake Synchronizer 使用條件：

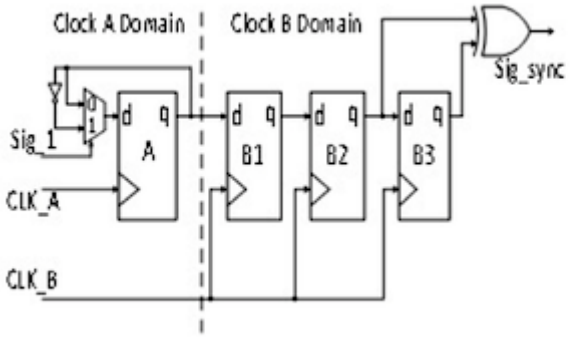
- Extension of mux based synchronization with an additional acknowledgment line.

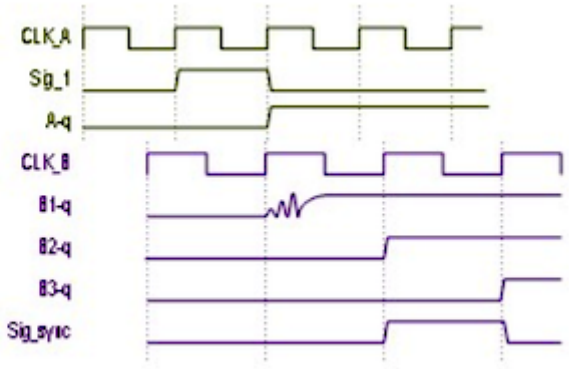




Toggle synchronizers 使用條件：

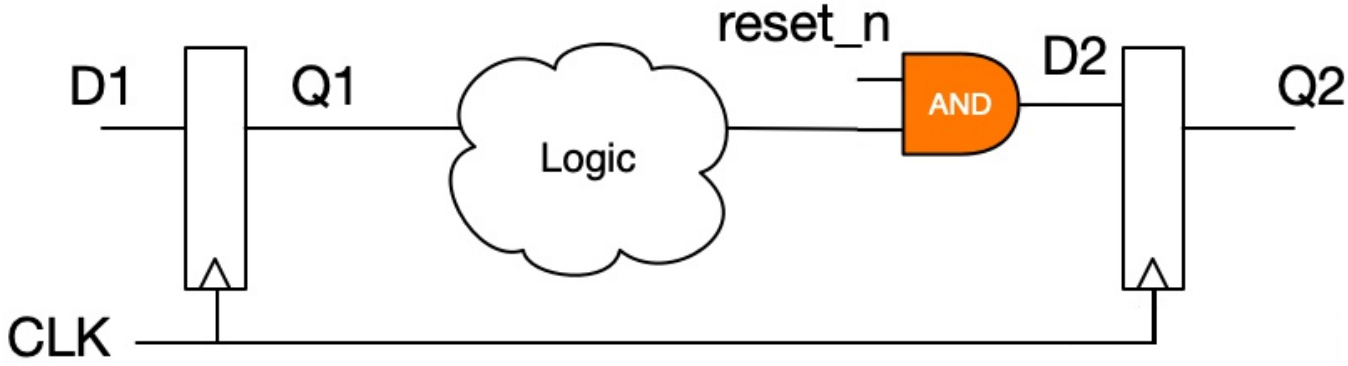
1. source clock domain is faster than destination clock domain.
2. Clock domain crossing (CDC)is on a single bit.





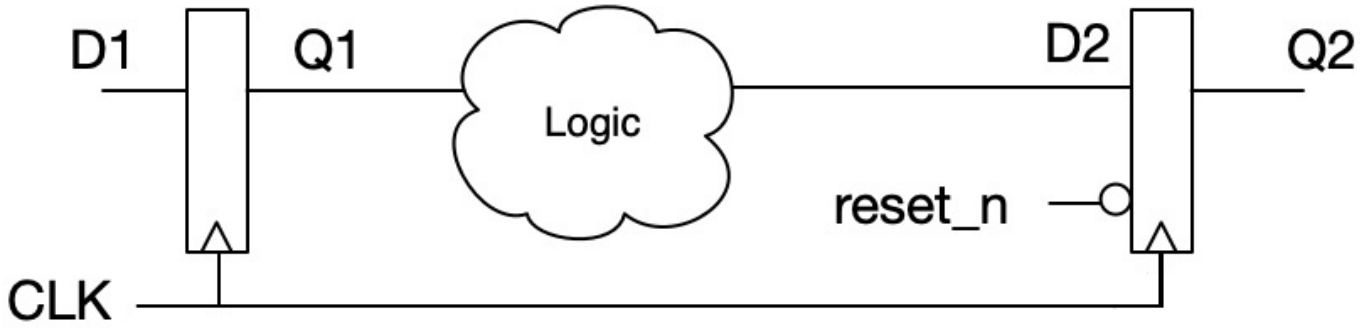
Asynchronous Reset Synchronous Release

同步 Reset (Synchronous Reset)：當 Reset 訊號為 Active 的時候，Register 在下一個Clk Pulse 到來之後被重設，Clk Pulse 到來之前 Register 保持其之前的值。



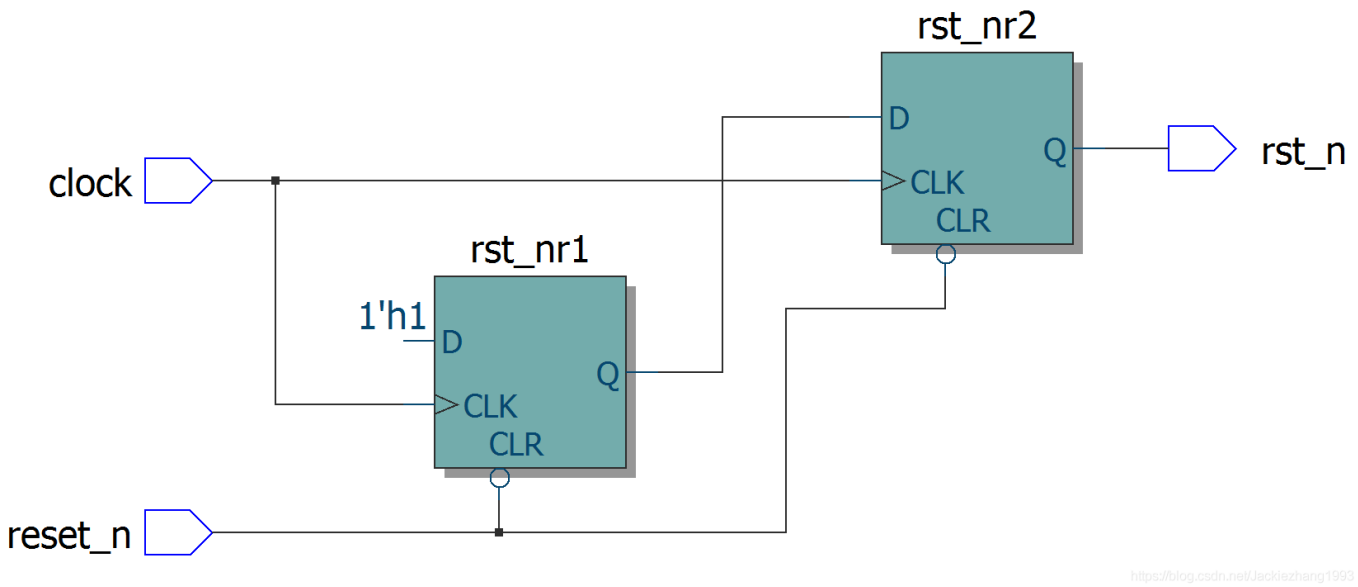
```
always @(posedge clk) begin
  if (!reset_n) begin
     q2 <= 1'b0;
  end
  else begin
     q2 <= ...
  end
end
```

非同步 Reset (Asynchronous Reset)：當 Reset 訊號為 Active 的時候，Register 立刻被重設，與 Clk Pulse 到來與否沒有關係。




```
always @(posedge clk or negedge reset_n) begin
  if (!reset_n) begin
     q2 <= 1'b0;
  end
  else begin
     q2 <= ...
  end
end
```

非同步重設，同步釋放（Synchronized Asynchronous Reset）：STA 時要保證訊號的傳輸滿足 Setup Time & Hold Time，避免 Metastability Sample。還有 Reset 訊號時到來時 Reset 和 Release 也必須滿足 Setup Time & Hold Time。一般採用非同步重設，同步釋放，如圖所示（只適用於沒有 PLL 的系統）。在 reset_n 到來的時候不受 clock 的同步，而是在 reset_n 釋放的時候受到 clock 的同步。



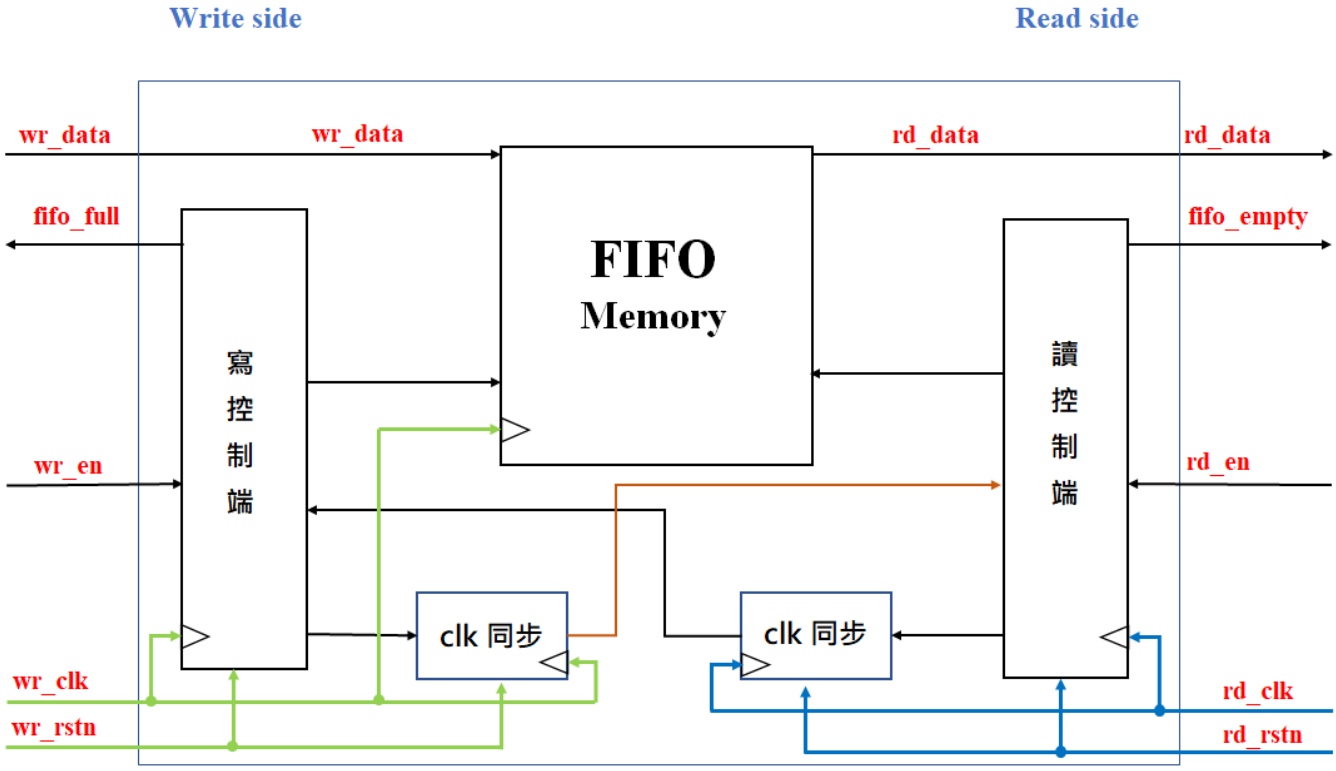

```
module sync_async_reset(clock, reset_n, rst_n);

    input clock, reset_n;
    output rst_n;
    reg rst_nr1, rst_nr2;

    always @(posedge clk or negedge reset_n) begin
        if (!reset_n) begin
            rst_nr1 <= 1'b0; // Asynchronous Reset
            rst_nr2 <= 1'b0;
        end
        else begin
            rst_nr1 <= 1'b1; // Synchronous Release
            rst_nr2 <= rst_nr1;
        end
    end

    assign rst_n = rst_nr2;

endmodule
```

非同步 FIFO

一種 Data Cache，沒有外部讀寫 address，使用起來簡單，缺點是只能順序寫入與讀出資料，其資料地址由內部讀寫指標自動加 1 完成，不能像一般儲存器可以由 address 決定讀取或寫入某個指定的地址。同步 FIFO 是指讀寫為同一個 Clk。在 Clk Pulse 來臨時同時發生讀寫操作。非同步 FIFO 是指讀寫 Clk 不一致，讀寫 Clk 是互相獨立的。

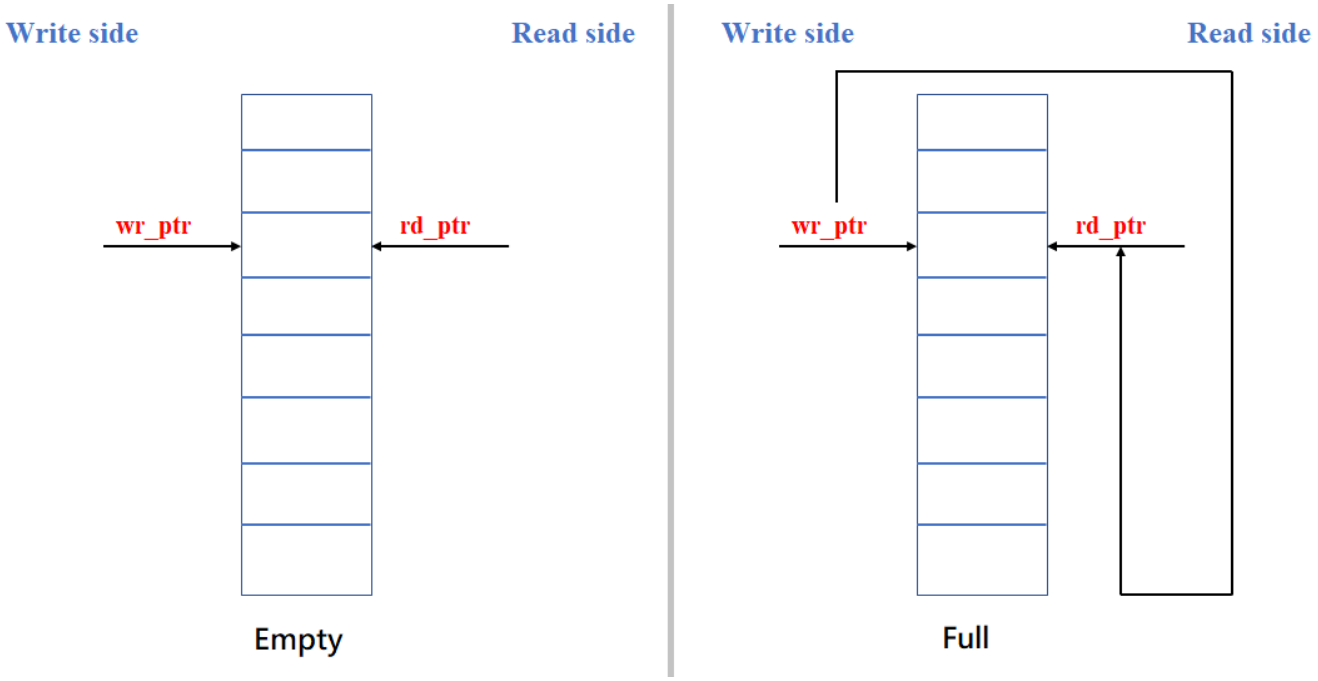
非同步 FIFO 主要由五部分組成：寫控制端、讀控制端、FIFO Memory 和兩個時脈同步端。寫控制端用於判斷是否可以寫入資料、讀控制端用於判斷是否可以讀取資料、FIFO Memory 用於儲存資料、兩個時脈同步端用於將讀寫 Clock 進行同步處理。

簡化圖：

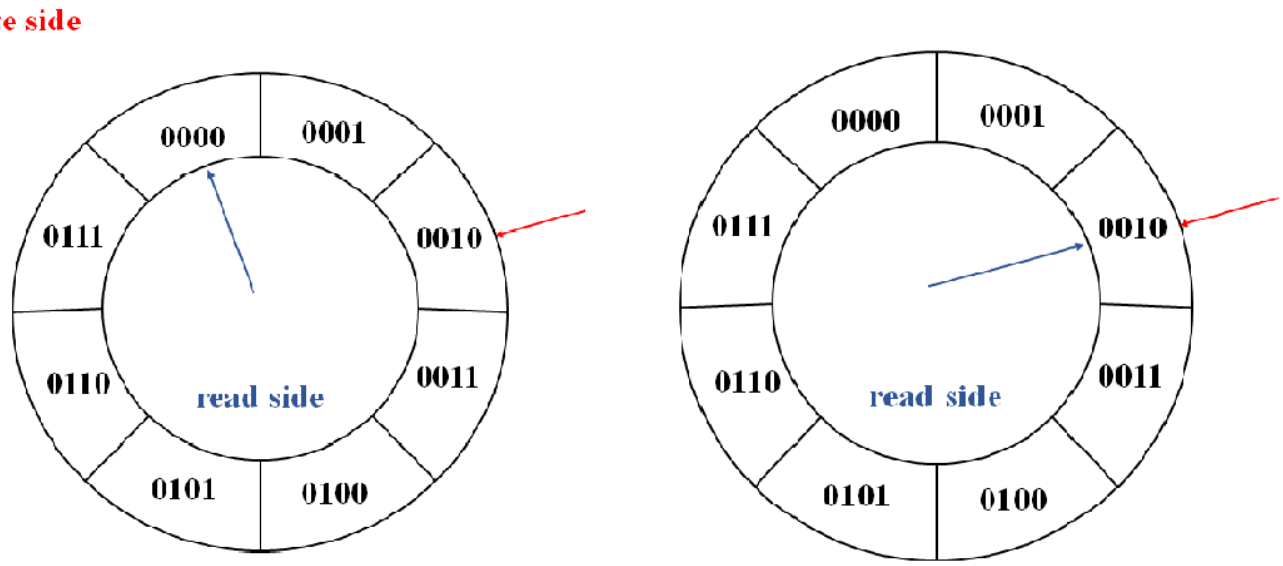


判斷能否寫入或讀取資料關鍵在於：

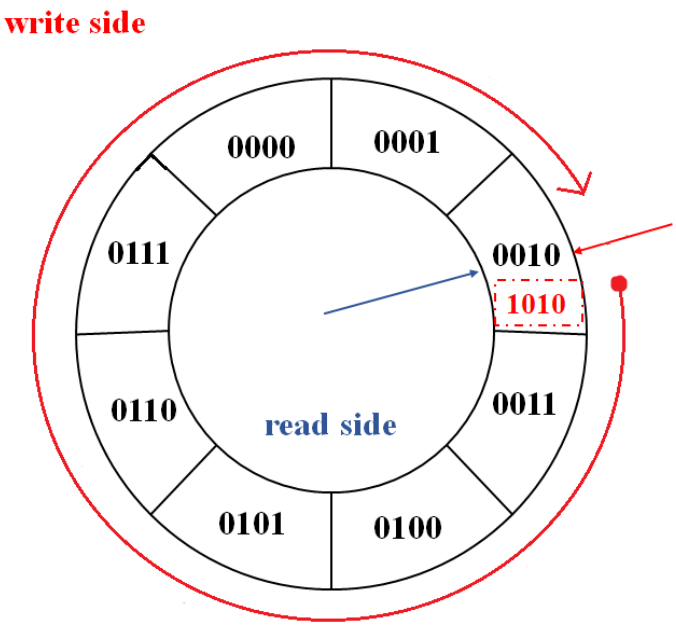
- Write 時，Write enable 有效且 FIFO 不是滿的。
- Read 時，Read enable 有效且 FIFO 不是空的。



Empty 可以理解為 rd_ptr 追上 wr_ptr，Full 可以理解為 wr_ptr 再次追上 rd_ptr。在非同步 FIFO 中，讀寫是在不同的 Clock 下進行的，因此在比較 rd_ptr 跟 wr_ptr 之前，應該先同步 Clock。而在同步之前，應該先將二進制 Address 轉換為 Gray code。



假設內圈為讀，外圈為寫，Empty 是 rd/wr_ptr 指向同一個地址，就像這樣 ↑ 。此時，rd/wr_ptr 完全相同，就以 0010 為例，0010 的 Gray code 為 0011，可以看出對於 Empty，無論是二進制還是 Gray code 所有 Bit 都相同



十進碼	二進碼	格雷碼
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

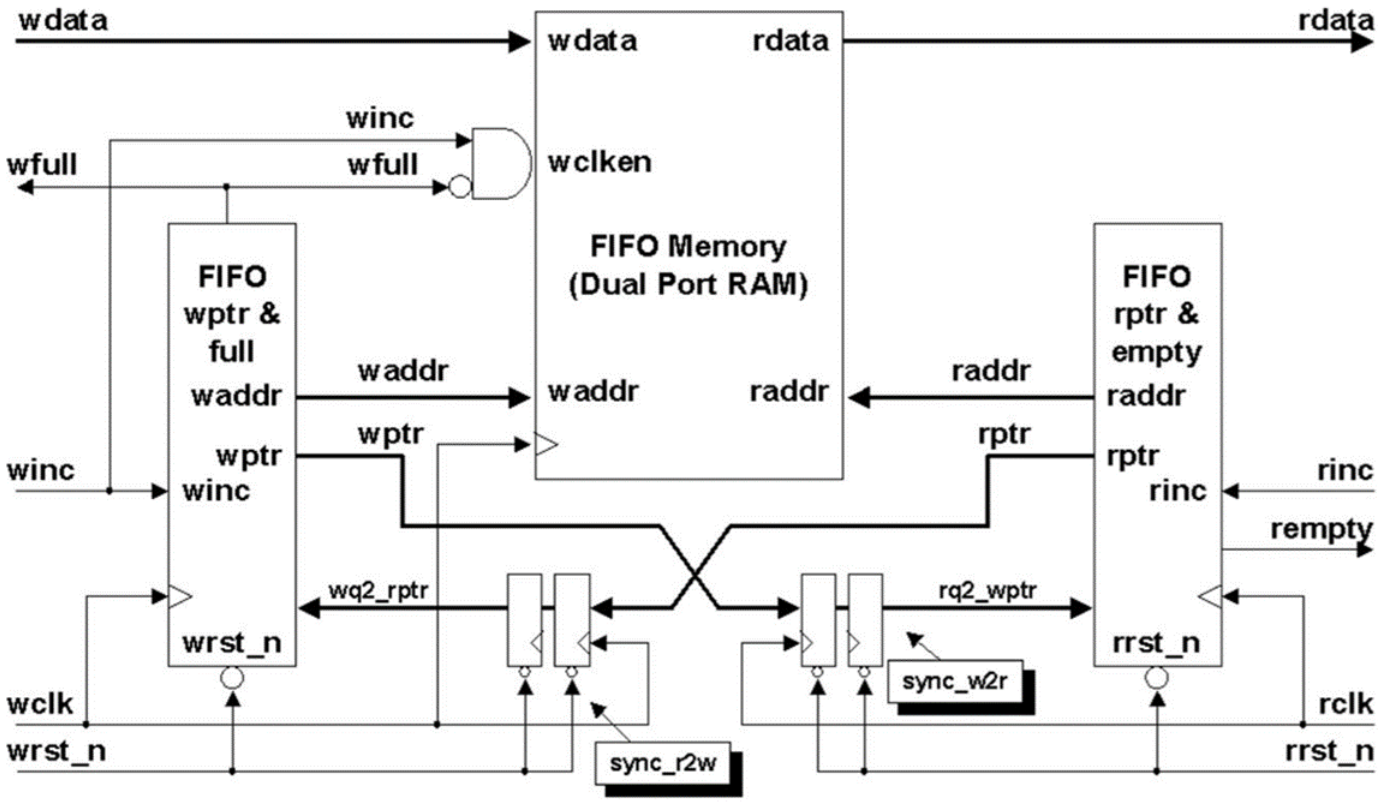
走了一圈之後，1010 的 Gray code 是 1111， 0010 的 Gray code 是 0011，對比兩個 Gray code 可以發現，此時高兩位相反，低兩位相同，這便是 Gray code 下寫滿的判斷條件。

因此可以總結出判斷 Empty 跟 Full 可以透過 Gray code：

- Empty 時，rd_ptr 跟 wr_ptr 的 Gray code 一模一樣。
- Full 時，rd_ptr 跟 wr_ptr 的 Gray code 差 2 個 Bits（高兩位相反，低兩位相同）。

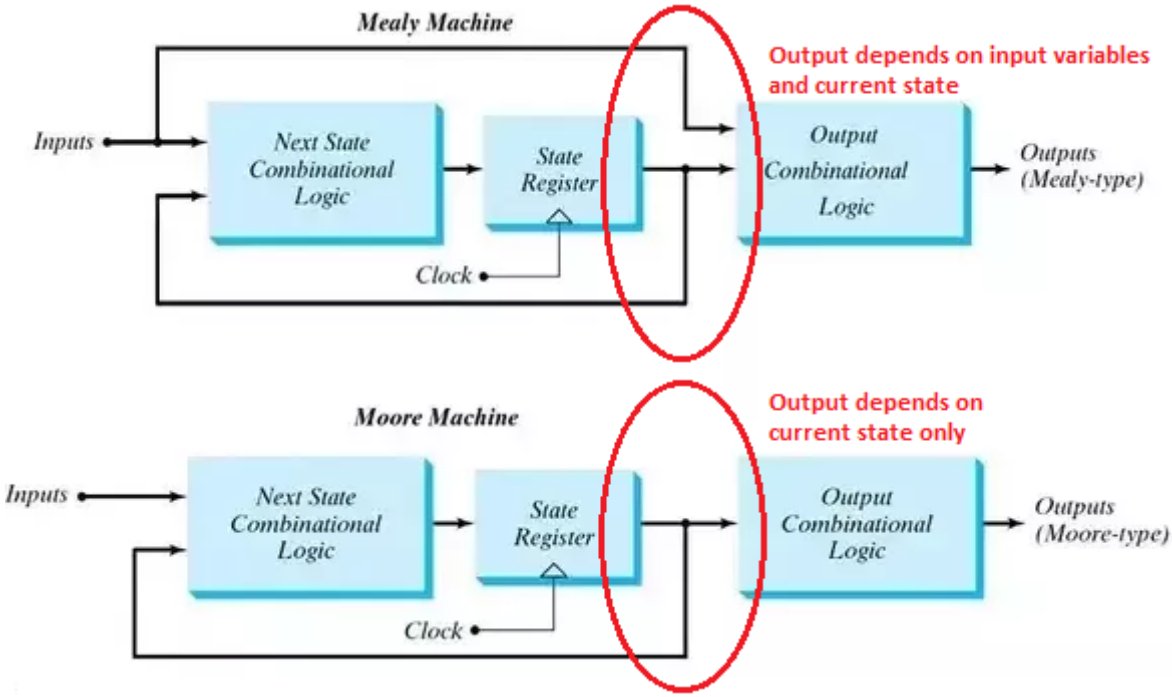
```
// Full
    always @(posedge wr_clk or negedge wr_rstn) begin
        if(!wr_rstn)
            fifo_full <= 0; // Write reset
        else if((wr_ptr[$clog2(DEPTH)] != rd_ptr[$clog2(DEPTH)])
            && (wr_ptr[$clog2(DEPTH) - 1] != rd_ptr[$clog2(DEPTH) - 1])
            && (wr_ptr[$clog2(DEPTH) - 2 : 0] == rd_ptr[$clog2(DEPTH) - 2 : 0]))
            fifo_full <= 1;
        else
            fifo_full <= 0;
    end

// Empty
    always @ (posedge rd_clk or negedge rd_rstn) begin
        if(!rd_rstn)
            fifo_empty <= 0; // Read reset
        else if(wr_ptr[$clog2(DEPTH) : 0] == rd_ptr[$clog2(DEPTH) : 0])
            fifo_empty <= 1;
        else
            fifo_empty <= 0;
    end
```

非同步 FIFO 架構圖：



FSM

Moore：出只取決於當前狀態的狀態機。
Mealy：輸出不光取決於當前狀態，還與輸入有關係的狀態機。



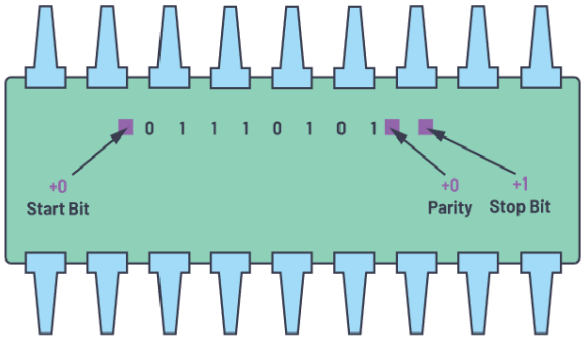
Inrerface

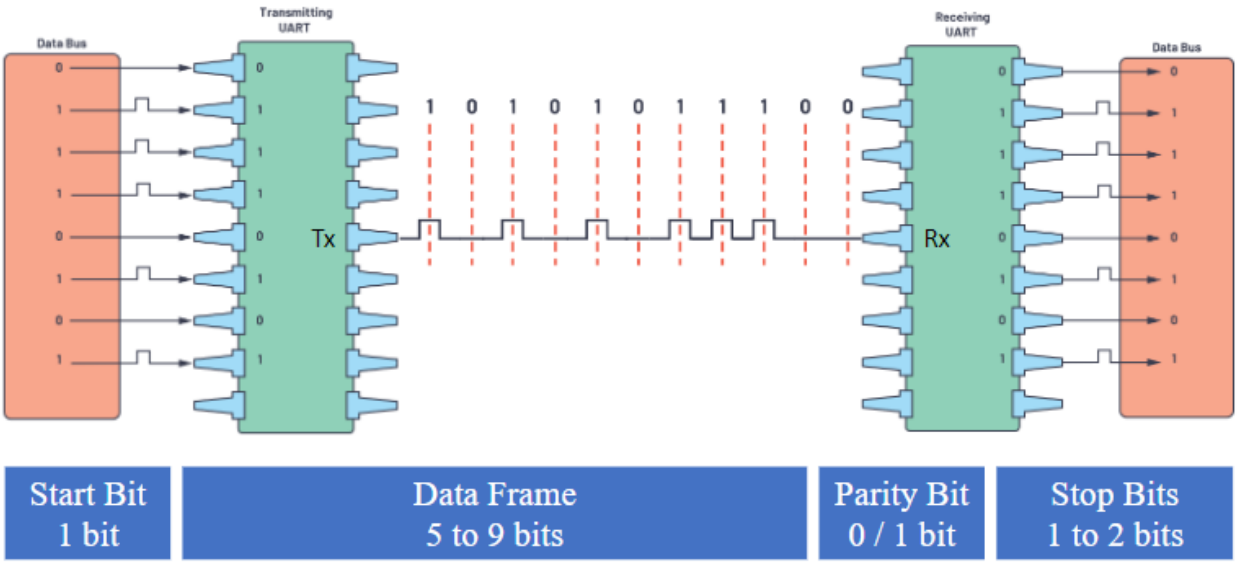
UART 非同步串列通訊埠

要開始資料傳輸，Transmitting UART 會將傳輸線從"高電壓拉到低電壓"並保持 1 個 Clock cycle。當 Receiving UART 檢測到高到低電壓躍遷時，便開始以串列傳輸速率對應的頻率讀取 Data Frame 中的 Bit。

透過 Parity Bit，Receiving UART 判斷傳輸期間是否有資料發生改變。電磁輻射、不一致的串列傳輸速率或長距離資料傳輸都可能改變資料位元。

為了表示資料封包結束，Transmitting UART 將資料傳輸線從低電壓驅動到高電壓並保持 1 到 2 Bits 時間。

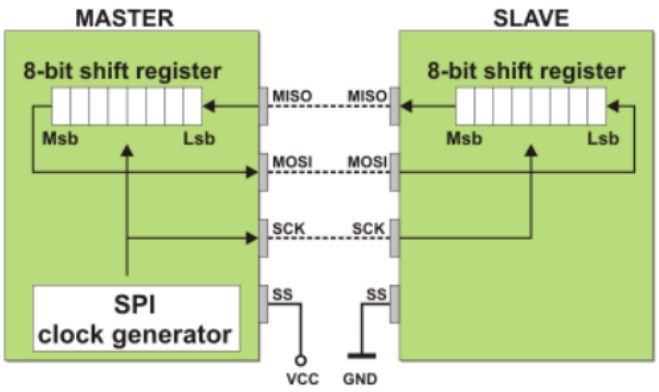


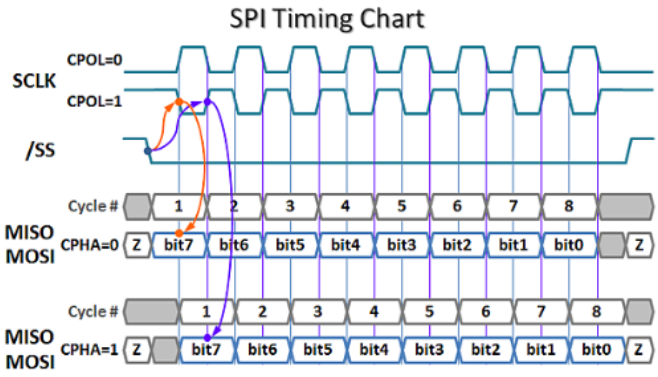


SPI 高速同步串列介面（全雙工）

SPI Slave 晶片一般只支援一種工作模式，所以通常 Master 必需遷就 Slave 把工作模式設成和 Slave 一致，才能正常運作（二個內部都是 Shift Register 所以 MOSI 和 MISO 的時序需求一樣）。

1. 判別 SCLK 的極性（Polarity），極性指 SPI 不工作時, SCLK 停留在高電位還是低電位。CPOL = 0 是 SCLK 在不工作時停留在低電位, CPOL = 1 則是停留在高電位.
2. Slave 晶片是在 SCLK 的 Negative Edge Trigger 栓鎖資料, 還是在 SCLK 的 Positive Edge Trigger。要讓對方栓鎖資料，我們就必需把資料 Hold 住，保持穩定，所以 Slave 晶片在 SCLK 的 Negative 栓鎖資料, 相對的 Master 就必需要在之前的 Positive 就送出資料，反之亦然。
3. CPHA 的定義並不是依 Positive/Negative Edge Trigger。CPHA 設定的是栓鎖資料的時機是在 /SS 訊號下降之後，SCLK 的奇數次變化（CPHA = 0），還是偶數次變化（CPHA = 1）。
4. 所以 CPHA 配合 CPOL（SCLK 的極性）設定，組合起來一共有 4 種工作模式。

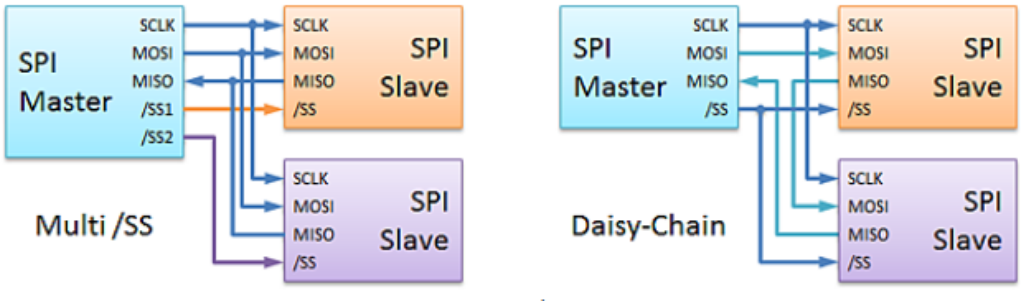




SPI 接腳名稱及意義

接腳名稱	中文	說明
MOSI	主出從入	master 數據輸出, slave 數據輸入
MISO	主入從出	master 數據輸入, slave 數據輸出
SCLK	時脈訊號	時脈信號, 由 master 產生並控制
/SS	晶片致能	slave 選擇信號, 由 master 控制. slave 只有在 /SS 信號為低電位時, 才會對 master 的操作指令有反應.

SPI Master 介接多個 Slave 的二種接法：

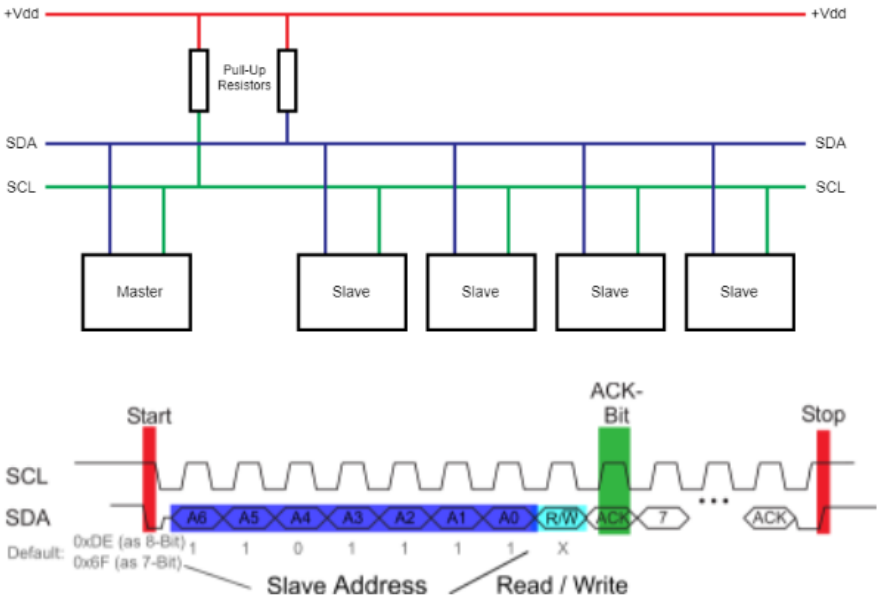


使用 SPI，最麻煩一件事是確認周邊晶片的工作模式。

Mode # of Model A	Mode # of Model B	C_{POL}	C_{PHA}	資料栓鎖時機		MISO MOSI 時序圖
1	0	C_{POL} =0	C_{PHA} =0	奇數次	上升緣	上半組
0	1	C_{POL} =0	C_{PHA} =1	偶數次	下降緣	下半組
3	2	C_{POL} =1	C_{PHA} =0	奇數次	下降緣	上半組
2	3	C_{POL} =1	C_{PHA} =1	偶數次	上升緣	下半組
Model A: For ARM-based and Microchip PIC Model B: 其他						

I^2C（半雙工）

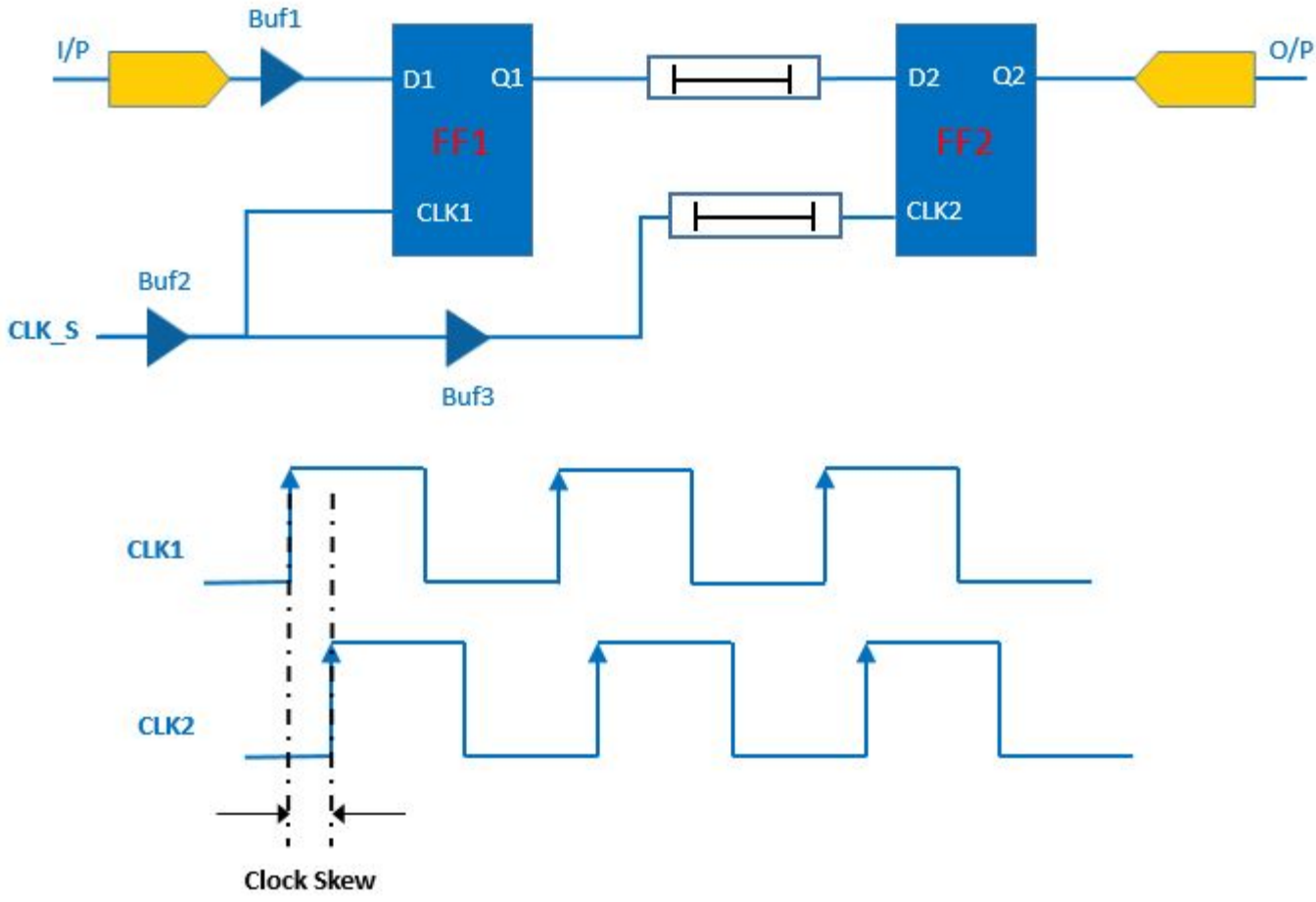
I^2C Bus 協定在傳輸的內容上，除了 Start（啟始）和 Stop（結束）二個訊號之外，所有的訊號傳輸固定 8 bits（1 Byte）為一組，MSB（Most Significant Bit）先送出。發送端在每組（8 bits）訊號送出後，需讀取接收端所回應的一個 ACK bit 或者 NACK bit（看 Spec or Datasheet 決定）。發送端不一定是 Master。例如: 讀取資料時，發送端為 Slave。



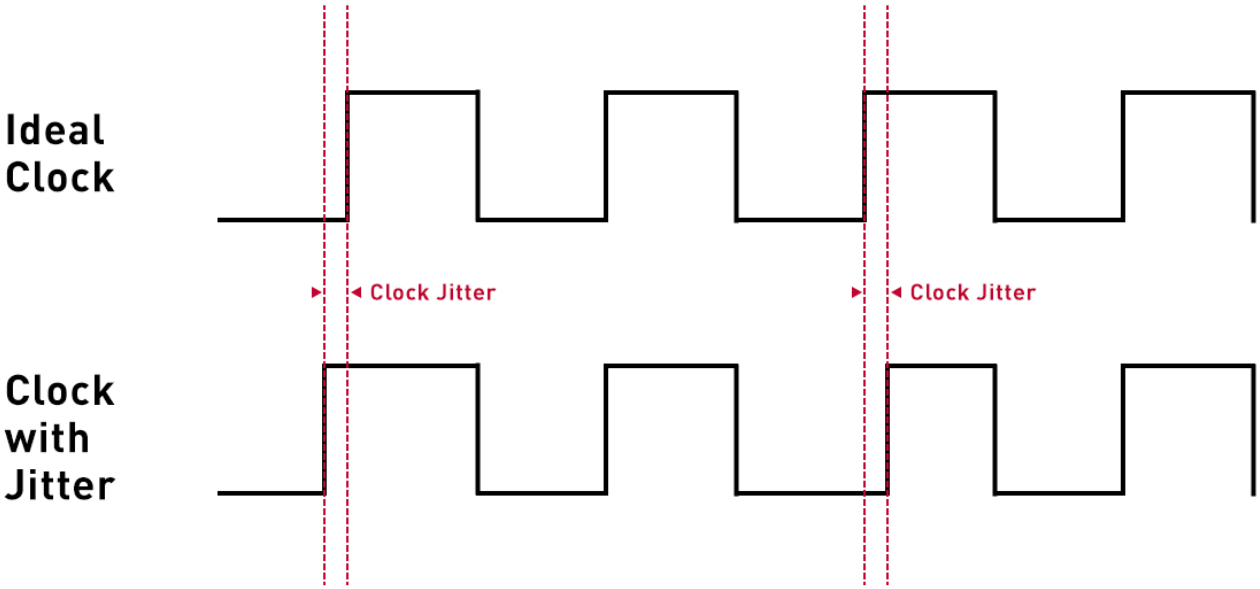
其他

Clock Jitter 與 Clock Skew 區別

簡言之，Skew 通常是 Clock 相位上的不確定，而 Jitter 是指 Clock 頻率上的不確定。Clock 到達不同 Register 所經歷路徑的驅動和負載的不同，Clk Pulse 的位置有所差異，是 Skew。



而由於振盪器本身穩定性，電源以及溫度變化等原因造成了 Clock 頻率的變化，就是 Jitter。



tags: Intern