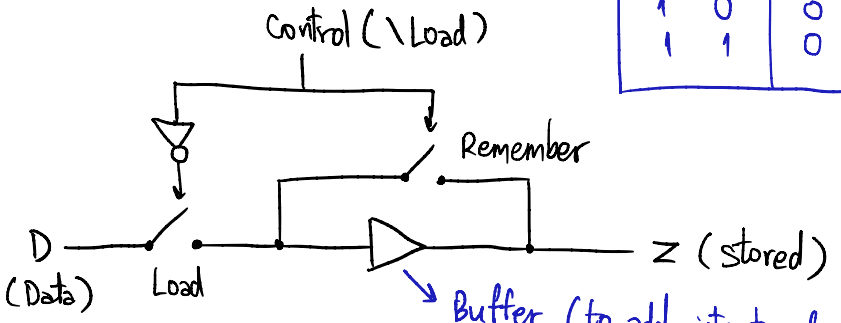
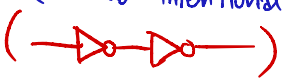
Sequential Logic Circuit

A	B	NOR	NAND
0	0	1	1
0	1	0	1
1	0	0	1
1	1	0	0

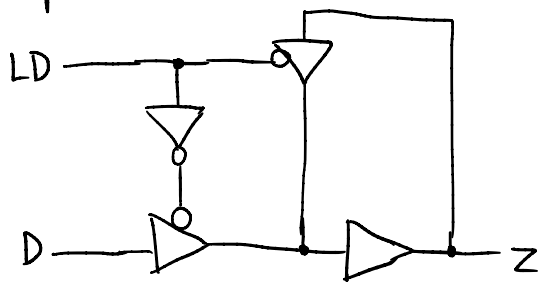
Static Memory Cell (MEM)



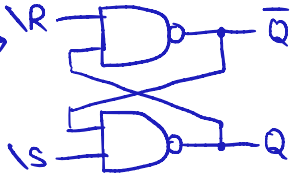
Buffer (to add intentional delay)



- Tri-state implementation



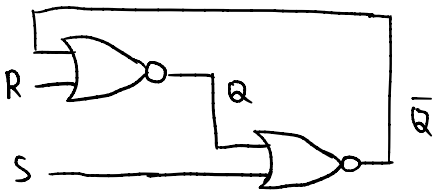
R-S Latch (Reset-Set)



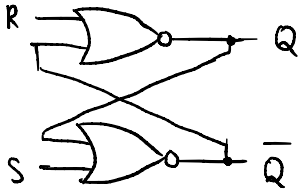
\R	\S	Q
0	0	Forbidden
0	1	0
1	0	1
1	1	Q

- $Q^+ = Q\bar{R} + S$

①



②

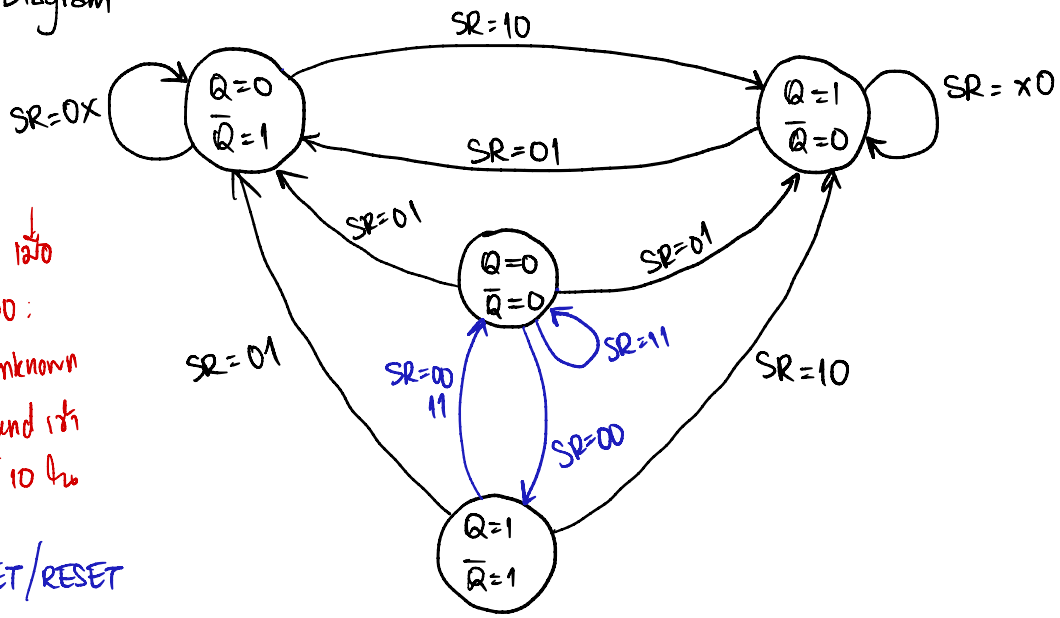


- Functional truth table :

R	S	Q
0	0	Q
0	1	1
1	0	0
1	1	Forbidden (Unstable)

→ เกิด Race condition

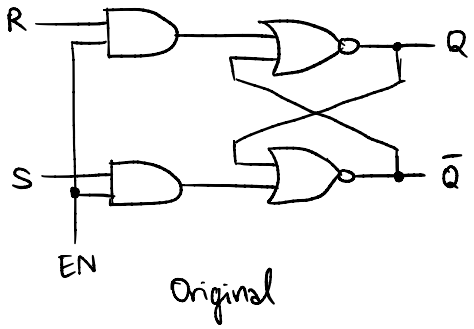
- State Diagram



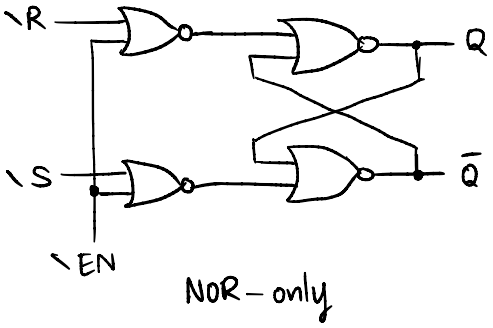
*SR Latch เมื่อเปลี่ยนสถานะ, SR=00 : Latch จะอยู่ใน Unknown State และจะ bound เข้า $Q\bar{Q}$ = 01 หรือ 10 ในช่วงเวลาขณะที่เข้าสู่วงจร SET/RESET

Level-triggered FF
(Level-sensitive latch) → output เปลี่ยนตาม Input ทันทีที่ EN ต่างจาก Flip Flop ที่ต้อง "อ่านค่า" ที่จุดเปลี่ยน CLK.

Gated RS Latch (Using EN Pin as Clock (CLK) Pin) → ทำต่อกัน clock จะเป็น "RS Flip Flop"

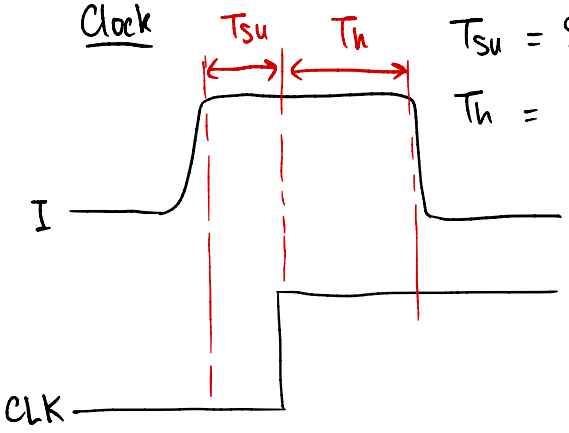


Original

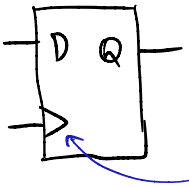


NOR-only

Clock



Tsu = Setup time
Th = Hold time
} Min. time, per specification.



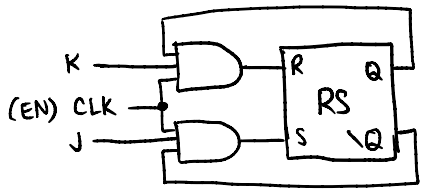
Symbol.

Racing Condition

Level-triggered FF
↳ Trigger ที่ค่าความจริง

Edge-triggered FF
↳ Trigger ที่ขอบ clock

JK Flip Flop to eliminate forbidden state



จะเกิด Racing 1-0-1-0 ตอน Toggle
แก้ด้วย Master-Slave เอามาดักด้วยตัว
ไม่ Enb (HOLD ไว้ช่วงเดียวกับ Clock ต่ำไป)

Characteristic Eqn.

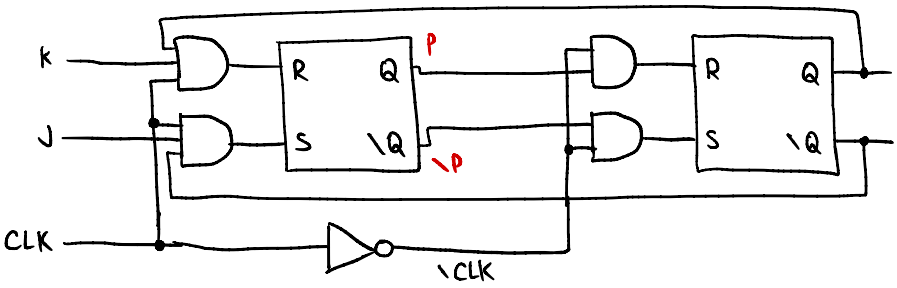
$$Q^+ = Q\bar{K} + \bar{Q}J$$

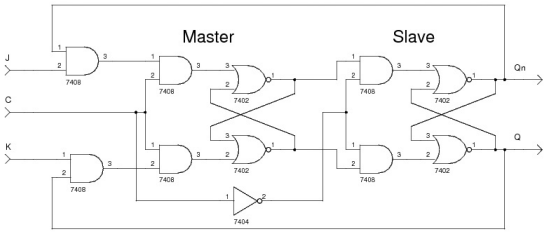
Func. TT

KJ-RS

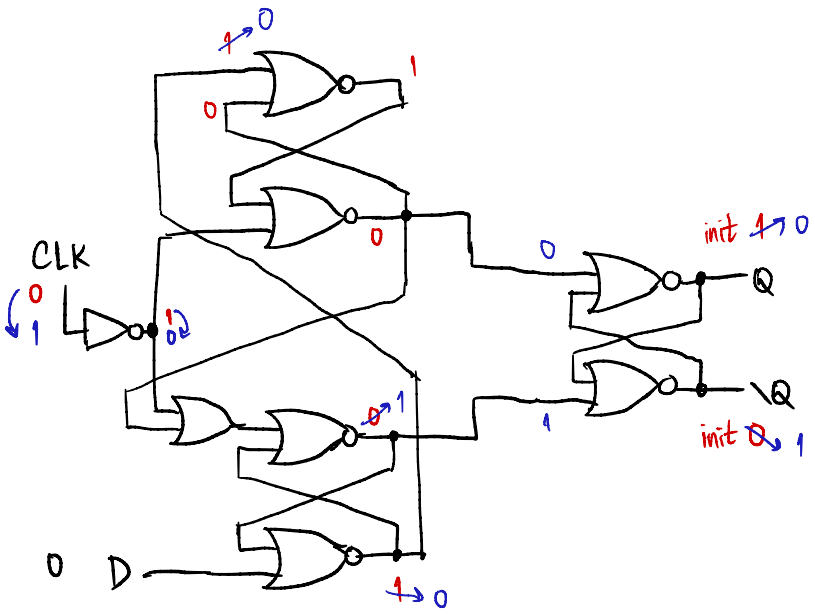
J	K	Q	Q^+	
0	0	0	0	HOLD
0	0	1	1	
0	1	0	0	RESET
0	1	1	0	
1	0	0	1	SET
1	0	1	1	
1	1	0	1	TOGGLE
1	1	1	0	

□ Master-Slave JK Flip Flop





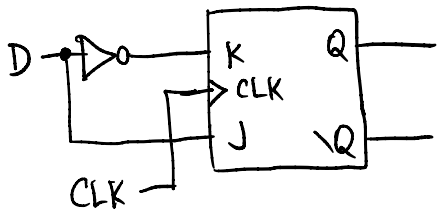
Edge-triggered FF



"Data"

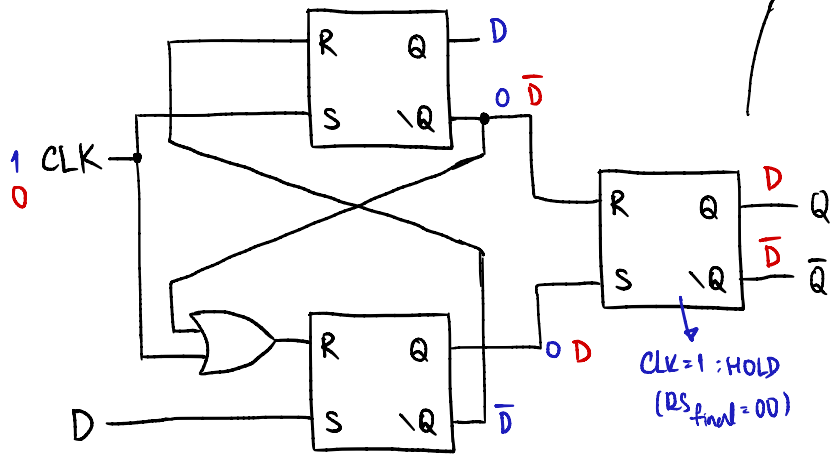
D Flip Flop : $Q^+ = D$

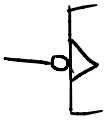
□ JK-FF implementation



JK: ไม่ค่อยดี (ไม่รู้ว่า output เปลี่ยนตาม clock ไหน)

□ Edge-triggered DFF





This is Negative edge (Falling edge)

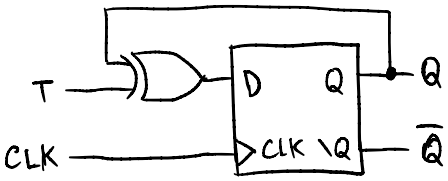
(CLK 1→0 : Data Latches (I. Sample)
then CLK=0 : Data Holds) (O. Valid)
When CLK=1 : Data Holds (Q Unchanged)

For Positive edge : CLK 0→1, 1
("Rising edge")

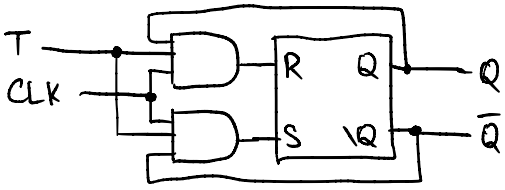
"Toggle"

T Flip Flop

□ สร้างจาก DFF, JK ได้



JK :



Excitation Table (Implementing FF from other types of FFs)

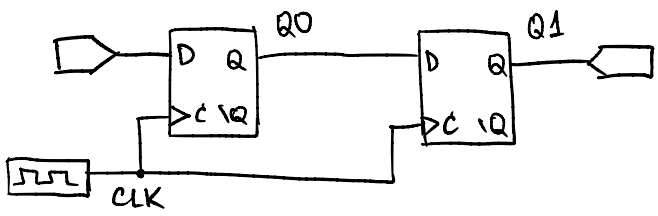
Q	Q^+	R	S	J	K	T	D
0	0	X	0	0	X	0	0
0	1	0	1	1	X	1	1
1	0	1	0	X	1	1	0
1	1	0	X	X	0	0	1

→ เอาไปทำ K-Map.

⊕ Presets and Clear Pin

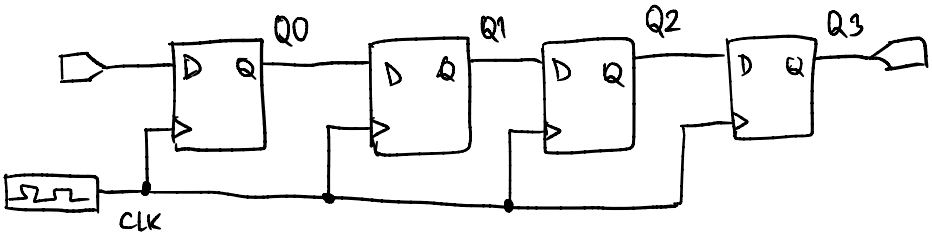
Shift Register

2-bit



Q1 เปลี่ยนตาม Q0 ในช่วง "ขาลง" (ช่วง CLK Rise/Fall)

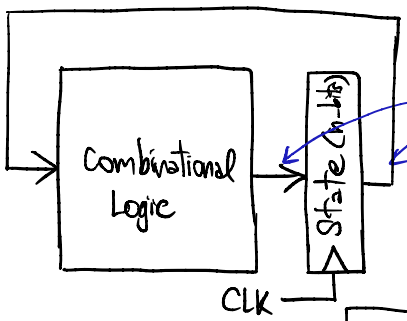
4-bit



Memory Implementation : Use latch > flip flop ∵ + less gates
+ less complex clocking

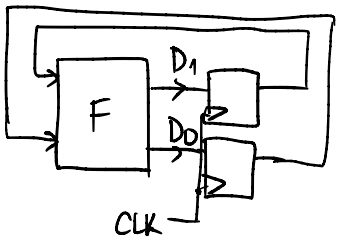
↳ อาศัย Narrow Timing ควบคุม Latch เพื่อ Update ค่า ไม่เหมือน Flip-Flop.
$(T < T_{CLK})$

Clocked Sequential System



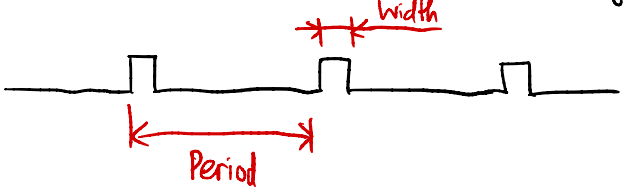
Bus(n) แทน 4 states ใช้ 2 bits: Q_1, Q_0.

e.g. 2-state system :



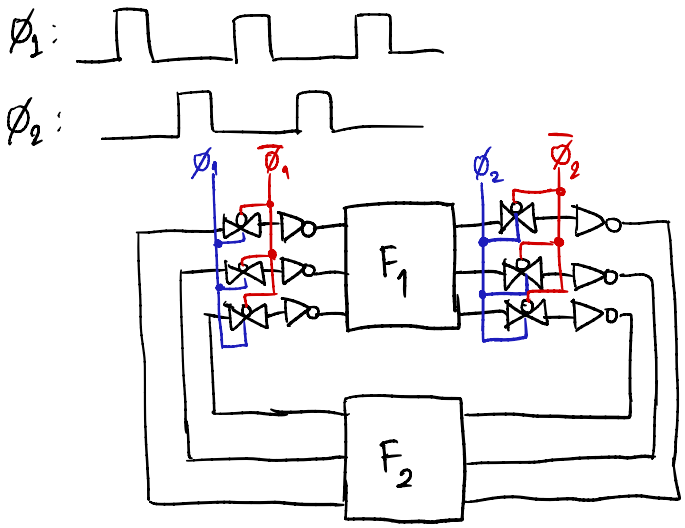
∴ CLK Width < Fastest propagation delay of F.

CLK Period > Slowest propagation delay of F.

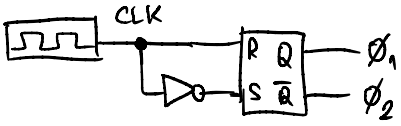


Two-Phase Non-Overlapped Clocking ("ช่วง CLK = 1 ไม่ทับซ้อนกัน")

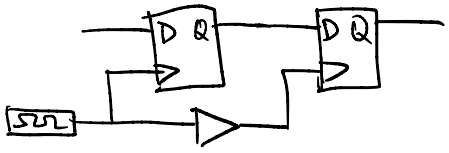
ก้อนซ้าย F ใช้ CLK1, ฝั่งขวา F ใช้ CLK2
Phases: (ϕ_1) (ϕ_2)



□ Generating 2-phase from 1 clock.



Clock Skew Problem "Clock มาไม่พร้อมกันในแต่ละ Flip Flop (มี delay)"

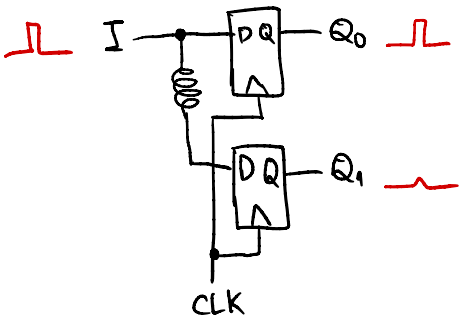


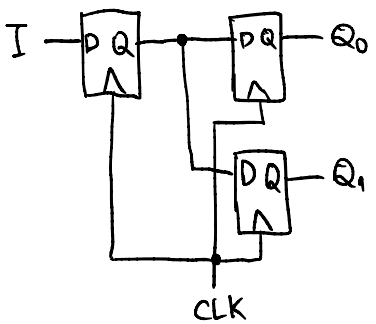
Handling Asynchronous Input

→ causes unpredictable output

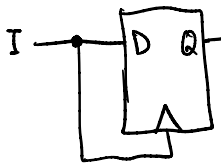
: ถ้าจะเชื่อมต่อกับ FF ต้องมี Input ที่ช่วงเวลา T_{su}, T_h การรับ Input ในวงจร จะต้องถูกใช้งานใน Synchronous Input ก่อน.

e.g. setup time violation





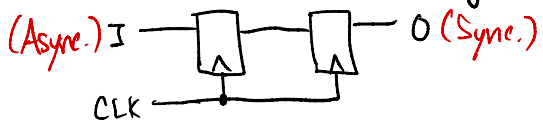
☐ Metastability : Synchronizer Failure



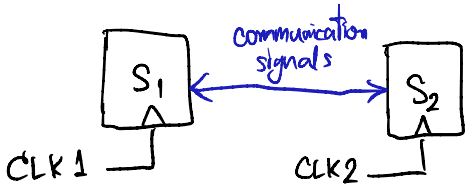
: Random Z value and time to steady.
(Can be either 0 or 1 at any time.)

Fixing : 1. Slow down clock
2. Use fastest logic in the synchronizer
3. Synchronizer cascading

} either of these.

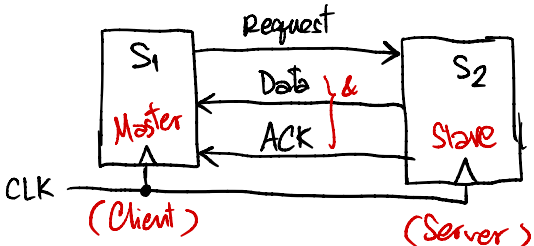


Self-Timed / Speed-independent circuits



SYNC

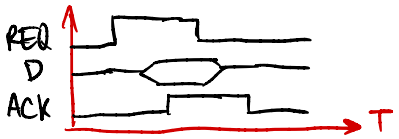
☐ REQ/ACK Signaling (Uses clock)



☐ REQ / WAIT (Uses clock) → "Data ยังไม่พร้อม" กรณีที่ M/S ไม่ได้ใช้ default clock ของกันและกัน data ควรเป็นตัว clock.

ASYNC

☐ 4-Cycle Signaling, RZ (No clock)



1. Master REQ (Start)
2. Slave ACK → Send Data
3. Master \REQ → Latches Data
4. Slave \ACK (Reset) : Everything = 0 ↔ Ready

1.) Master อ่านค่า ACK เรื่อยๆ (Polling)
2.) หรือ ใช้ Interrupt Pin ป้อนกลับมาที่ตัวมันเอง

☐ 2-Cycle Signaling, NRZ (No clock)

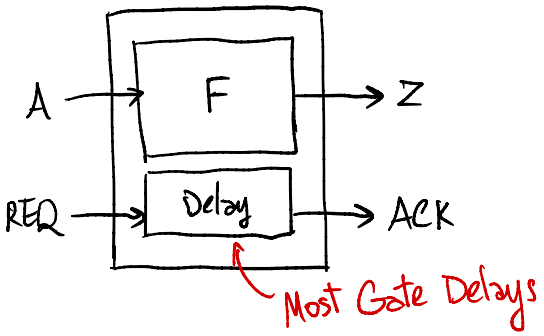
1. Master REQ
2. Slave ACK

} 1 รอบ send/latch Data.

1'. Master \REQ
2'. Slave \ACK

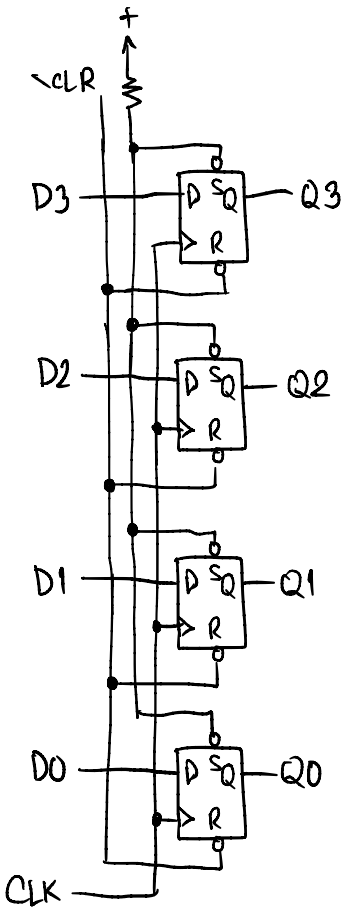
} อีก 1 รอบ send/latch Data.

Self-timed Circuits 은 worst case delay 가 Combinational logic

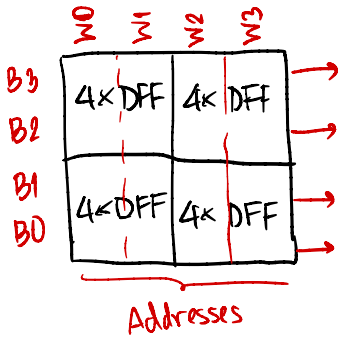


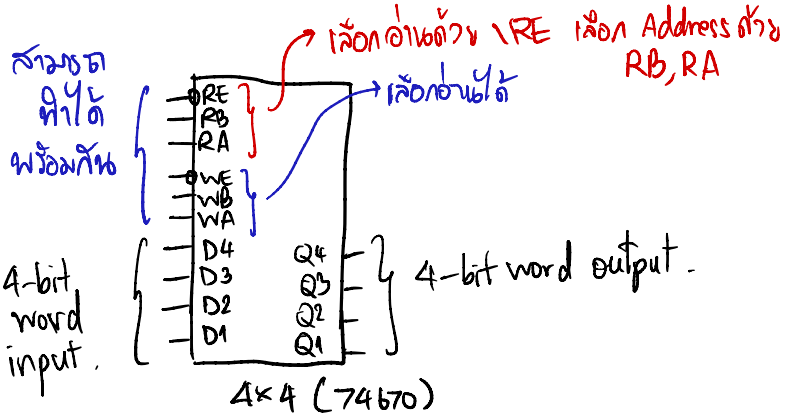
Registers

4-bit:



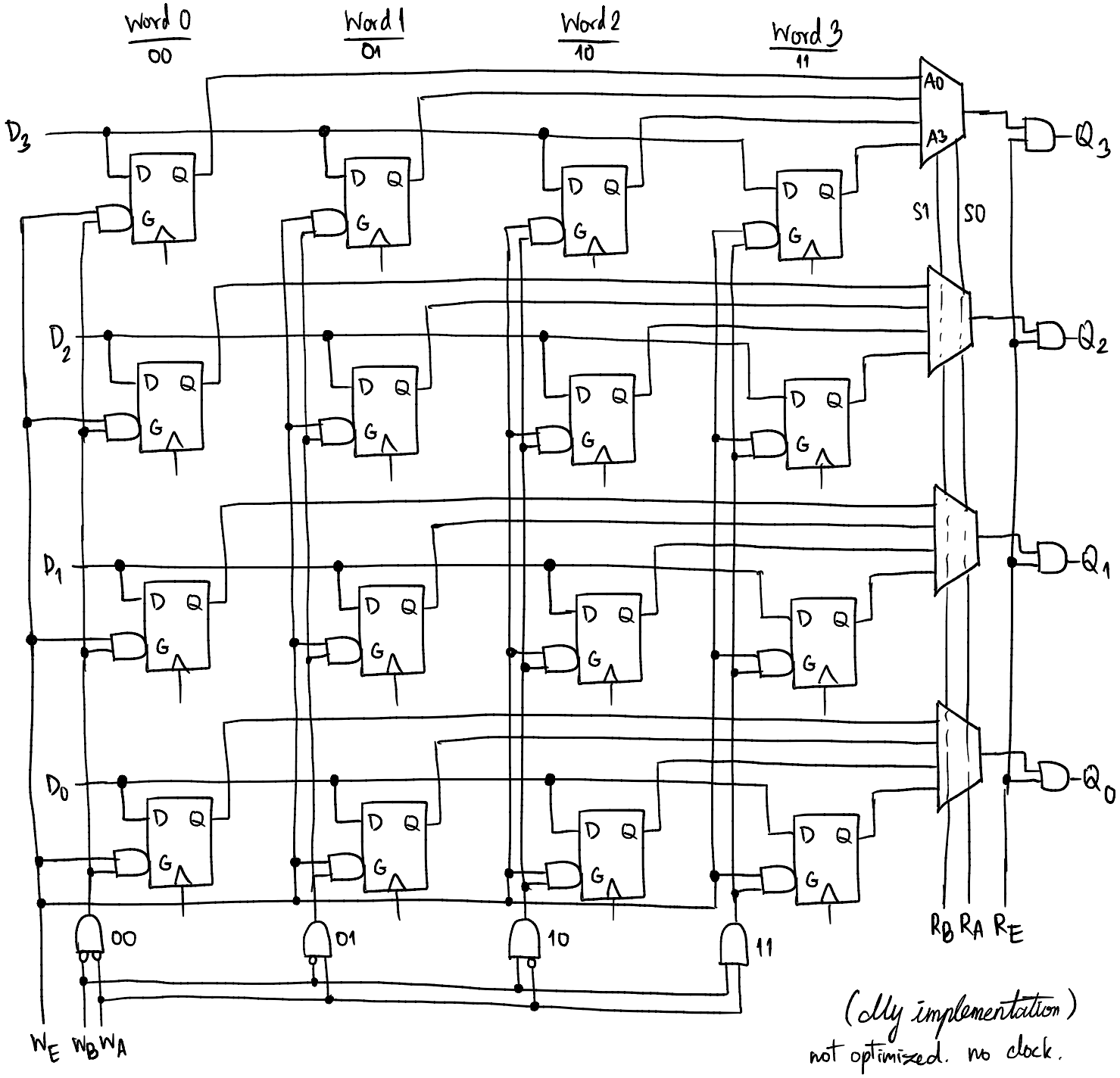
Register Files : คือ Register หลายตัวมาต่อกัน (2D)
หลักการเหมือนกับ ROM, RAM





4x4 Register File (ใช้ 16 x DFF)
Ref : 74670

(Address → Bits of words)



(My implementation)
not optimized. no clock.

L-Shift

DCBA → CBAX → BAXX

X ขึ้นด้วย LSI

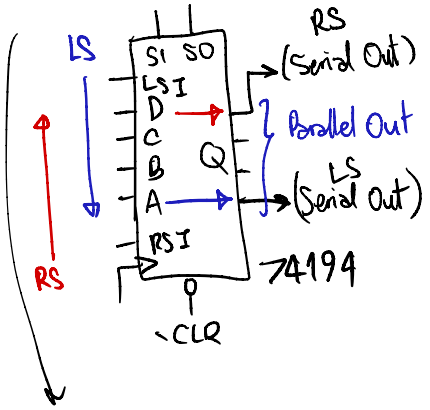
R-Shift

DCBA → XDCB → XXDC

ขึ้นด้วย RSI

Shift Register (Serial) LSI/RSI : Serial In , D,C,B,A : Parallel In.

4-bit : ABCD

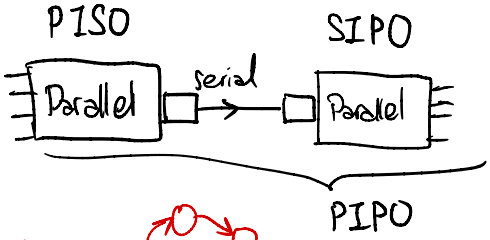


S1	S0	Function
1	1	Sync. Load (Parallel)
1	0	Left shift , D = LSI
0	1	Right shift , A = RSI
0	0	Hold

A B C D → B C D (LSI)

A B C D → (RSI) A B C

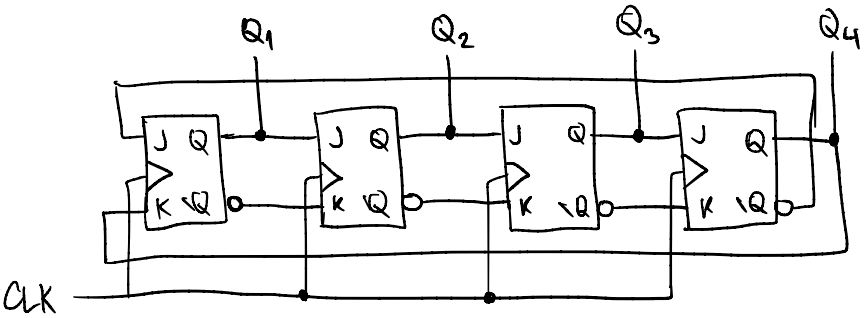
ใช้แปลง Serial - Parallel , Parallel - Serial ,



"Counter" คือ State ที่ Next State มีแบบเดียวและวนเป็นวง

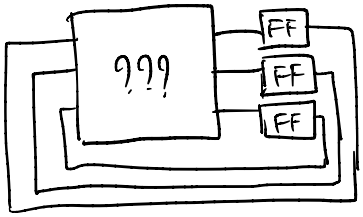
Counter Circuit (Sequential Increment / Decrement → "State")

"Mobius" Counter : ใช้หลักการของ Mobius strip (Johnson Counter)

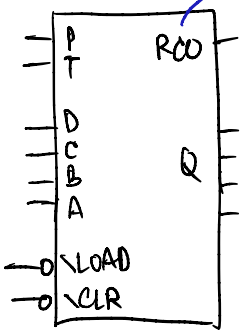


$Q_1Q_2Q_3Q_4$: 1000, 1100, 1110, 1111, 0111, 0011, 0001, 0000

การออกแบบวงจร Seq. เช่น 3-bit Counter:



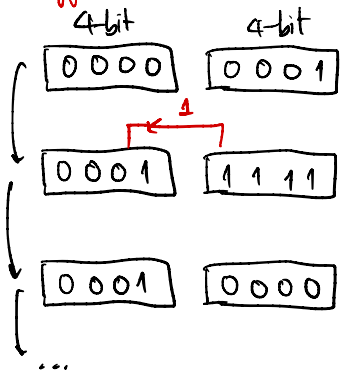
74163 Sync. counter



Ripple Carry Output ต่อ T ของ Higher-bit

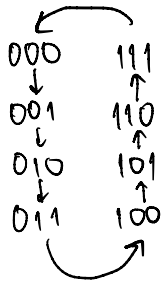
→ ส่ง 1 เมื่อ $Q_D:Q_A = 1111$ (ให้ Higher-bit counter นับ)

เป็นการส่ง Toggle ×1 clock



Choosing FF, Func. (ใช้สร้าง Finite-State Machine, FSM แบบใดก็ได้)

eg. 3-bit Upcounter

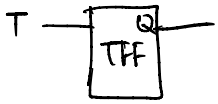


State transition table (ต่อจากหน้า)

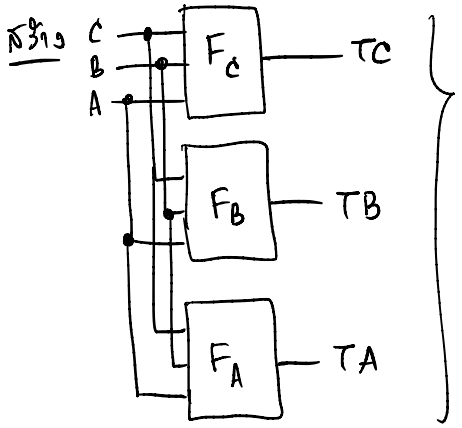
เลือกใช้ TFF ∴ มี period การเปลี่ยน

C	B	A	C^+	B^+	A^+	TC	TB	TA
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

ใช้ Excitation table ของ TFF



ของ C, B, A

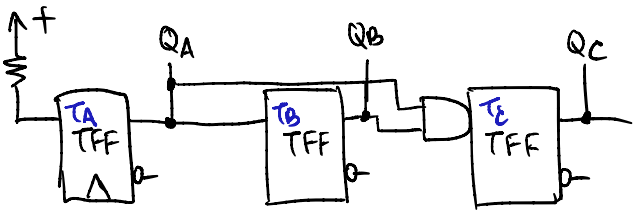


จะได้ $T_C = AB$

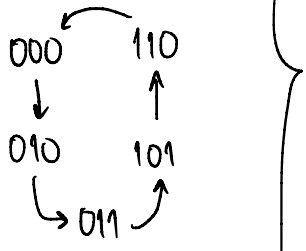
$T_B = A$

$T_A = 1$

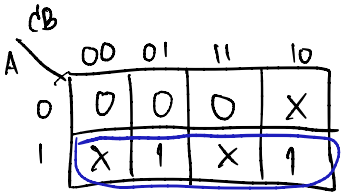
Result:



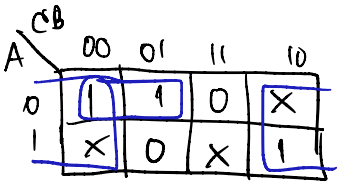
e.g.



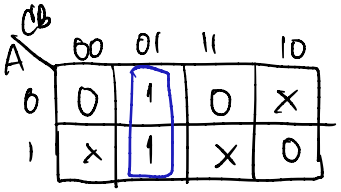
CBA	$C^+B^+A^+$	T_C T_B T_A
000	010	0 1 0
001	XXX	X X X
010	011	0 0 1
011	101	1 1 0
100	XXX	X X X
101	110	0 1 1
110	000	1 1 0
111	XXX	X X X



$C^+ = A$ (D_C)



$B^+ = \bar{B} + \bar{A}C$ (D_B)



$A^+ = \bar{C}B$ (D_A)

ใช้ DFF ได้

ทำ K-Map ได้ $T_C = A\bar{C} + \bar{A}C = A \oplus C$ ($D_C = A$)

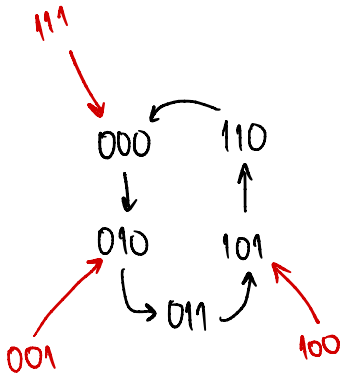
$T_B = A + \bar{B} + C$

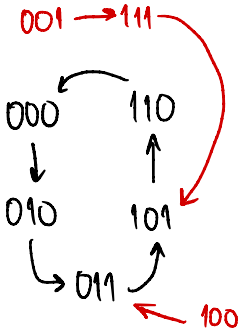
$T_A = \bar{A}B\bar{C} + \bar{B}C$

Using TFF

QQ^+	T
0 0	0
0 1	1
1 0	1
1 1	0

Guarantee Start-up State





ใช้ได้ดีกว่า

ใน Table, เปลี่ยนจาก XXX เป็น next state.

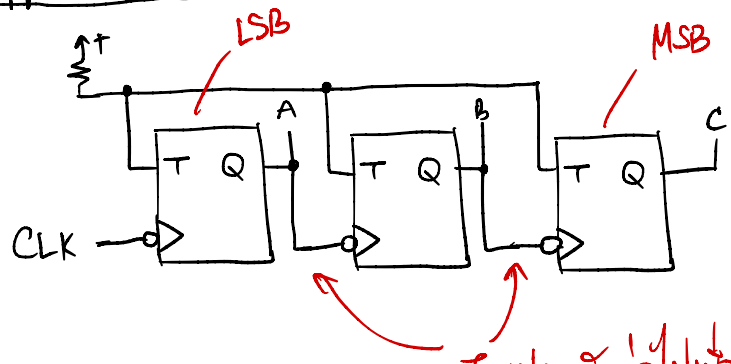
+ ถ้าใช้ FF แบบอื่น เช่น RS FF, JK FF ได้ แต่จะสร้างยากกว่า TFF, DFF
  - JK FF → เสี่ยงสูงเพราะ X เยอะ
  - DFF → ง่ายสุดเพราะใช้ next state เป็น input ได้เลย.

+ หรือจะ Mix & Match ในวงจรก็ได้

T↔D Conversion

A	B	$(A \oplus B) \oplus B = A$
0	0	0
0	1	0
1	0	1
1	1	1

Ripple Counter

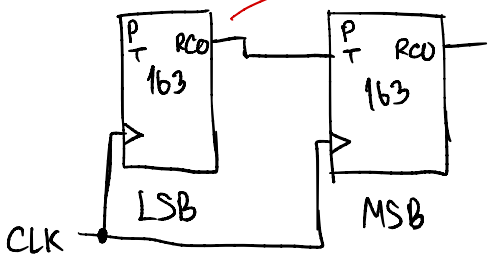


(CBA)

Toggle ตัวต่อไปเมื่อเปลี่ยนสถานะ: On Neg. Edge (1→0)

** ข้อเสีย - ทุกๆ การ Ripple จะมีการ Delay ทบไปเรื่อยๆ (Not sharp)
- แต่ละ bit เปลี่ยนไม่พร้อมกัน

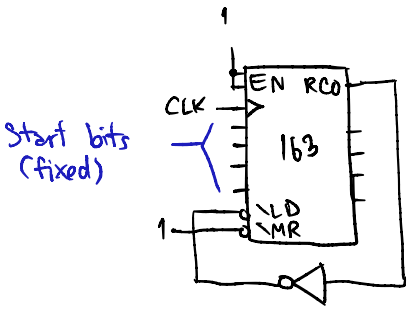
Cascaded Synchronous Counters with RCO



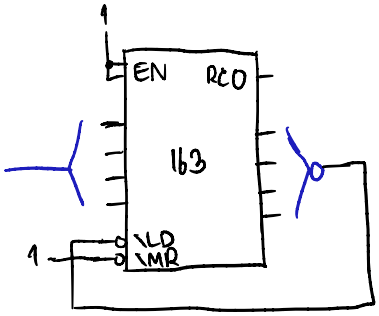
(RCO 0→1)
RCO จะถูก Assert ตาม DCBA ที่นับ (ไม่ใช่พร้อม Clock) เพื่อจะใช้ Assert Counter bit ถัดไป เพื่อรับ Clock.
หลังจากนั้น RCO จะเป็น 1→0

* ทุกวงจรที่เป็น Synchronous ต้องใช้จะอยู่เดียว 1 clock เสมอ

Offset Counter : Start bits

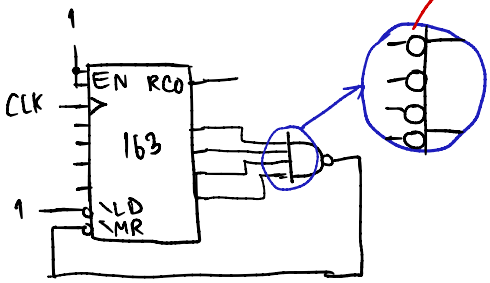


2-Way Limit



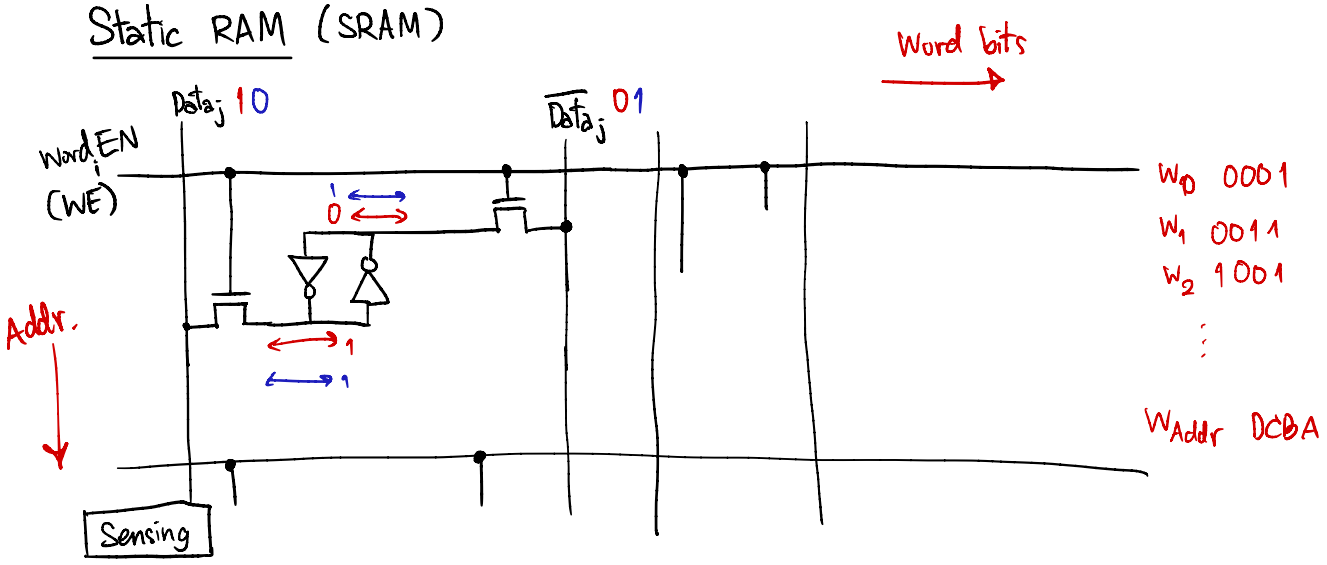
Limited Counter : End bits

Not เพราะเลือก Stopping (end) bits



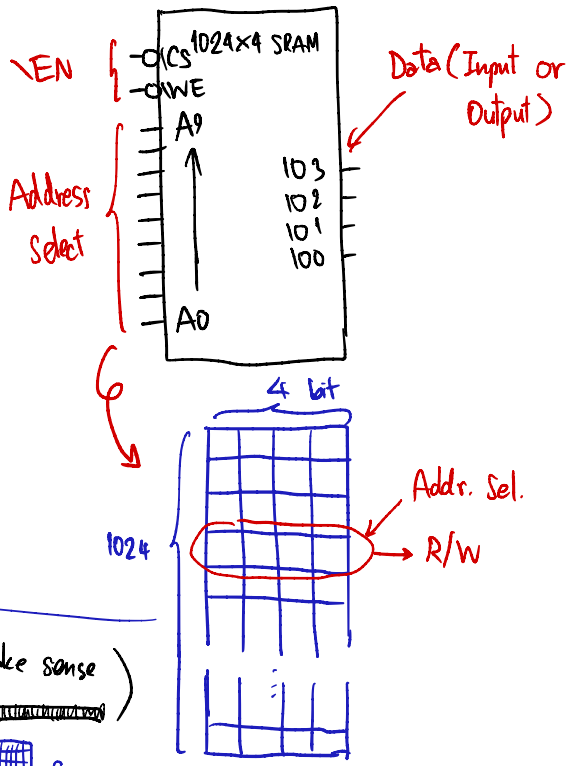
0101
0111
etc...

Static RAM (SRAM)

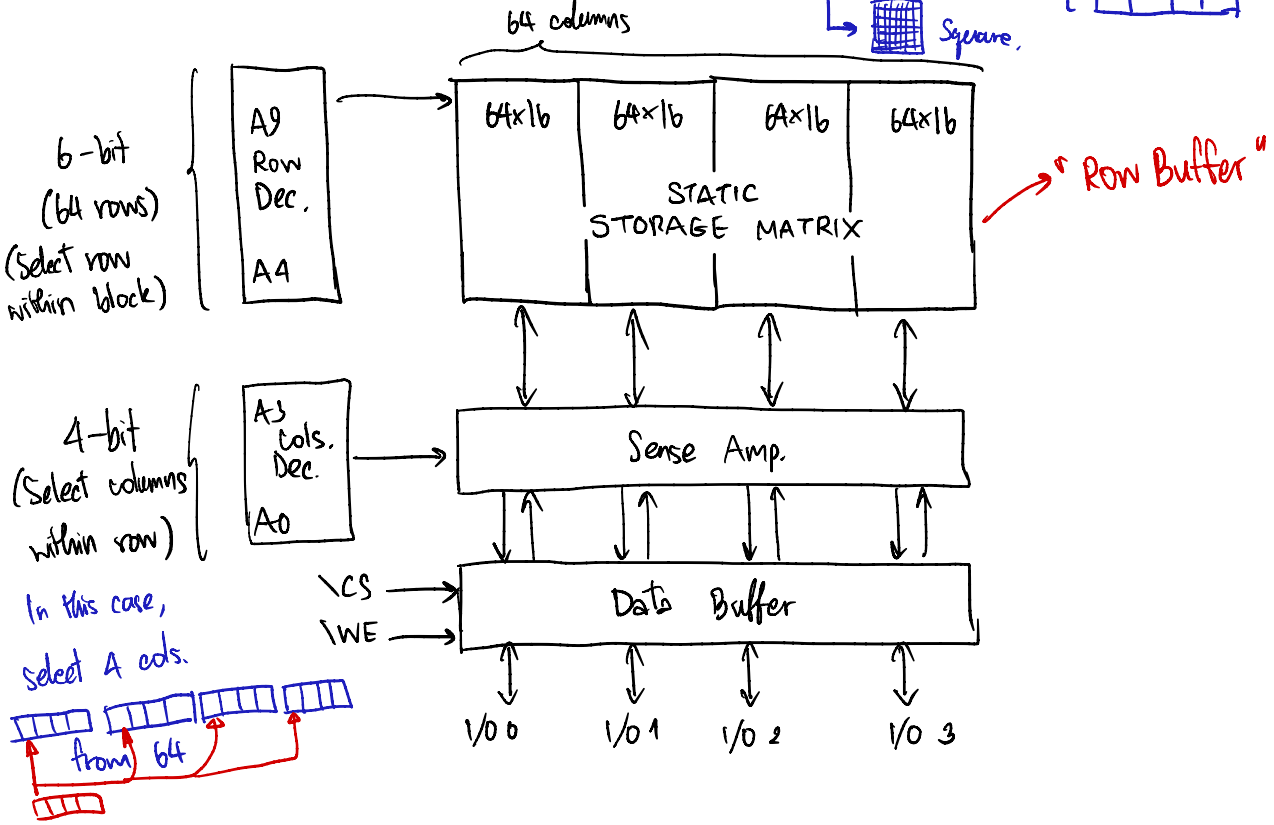


- ถ้าจะ Write : Data = \Data , WE = 1 (\WE = 0)
- ถ้าจะ Read : Data = $\overline{Data}$ = (0~1) (V_{RD})

เมื่อ \WE = 1 แล้วสัญญาณ share ไปยังส่วน (Sensing circuit) แล้ว Data (0~1) → Data ใน cell
$\overline{Data}$ (0~1) → \Data ใน cell

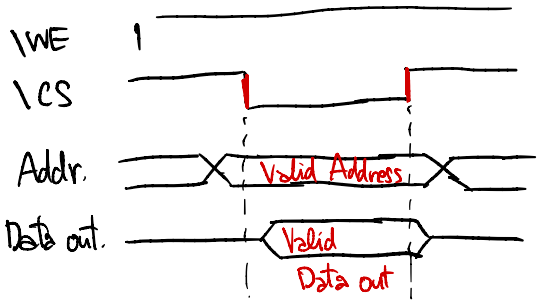


Designing sensible circuit (Real 1024 wouldn't make sense (long, thin)) → Square.



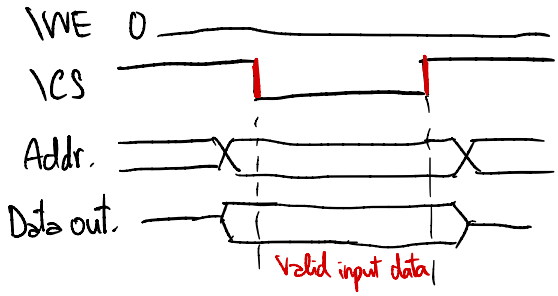
SRAM Timing

Read

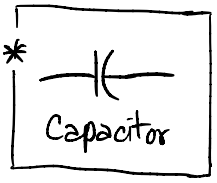


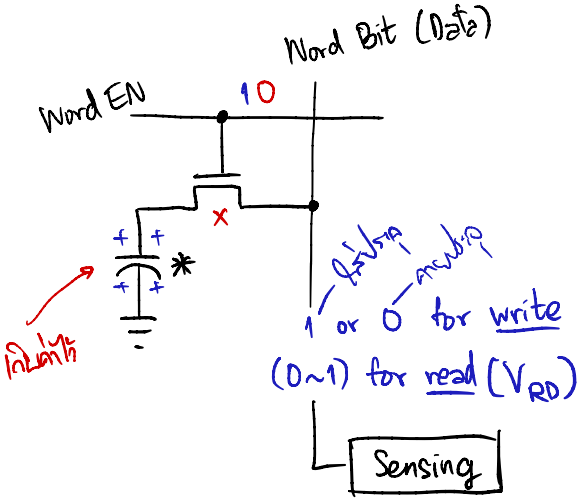
ต้องพร้อมก่อนสั้ง

Write



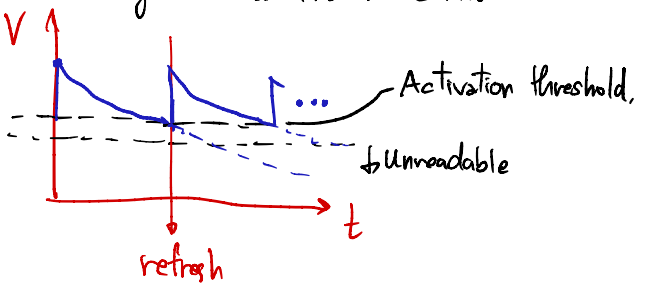
Dynamic RAM (DRAM)



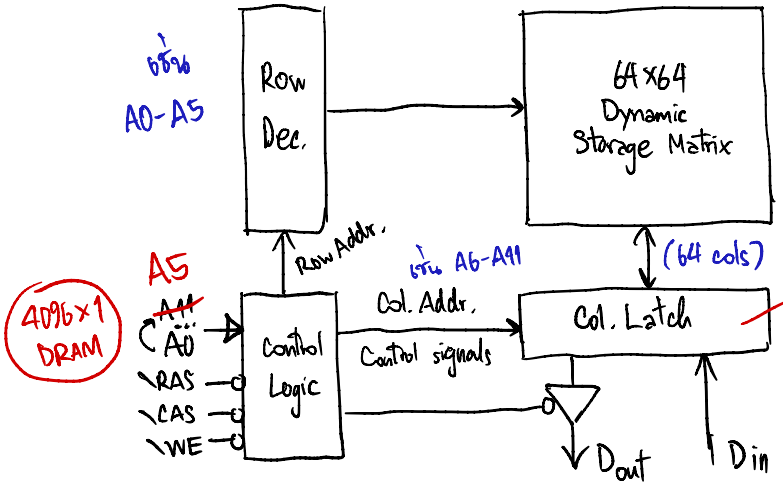


- Destructive Read out. (เกิดจากการคายประจุ)
- Need for Refresh Cycle
  (Capacitor storage decays over time)
  due to leakage of current.

การแก้ไขคือ ต้องทำ Read แล้ว Write คืน

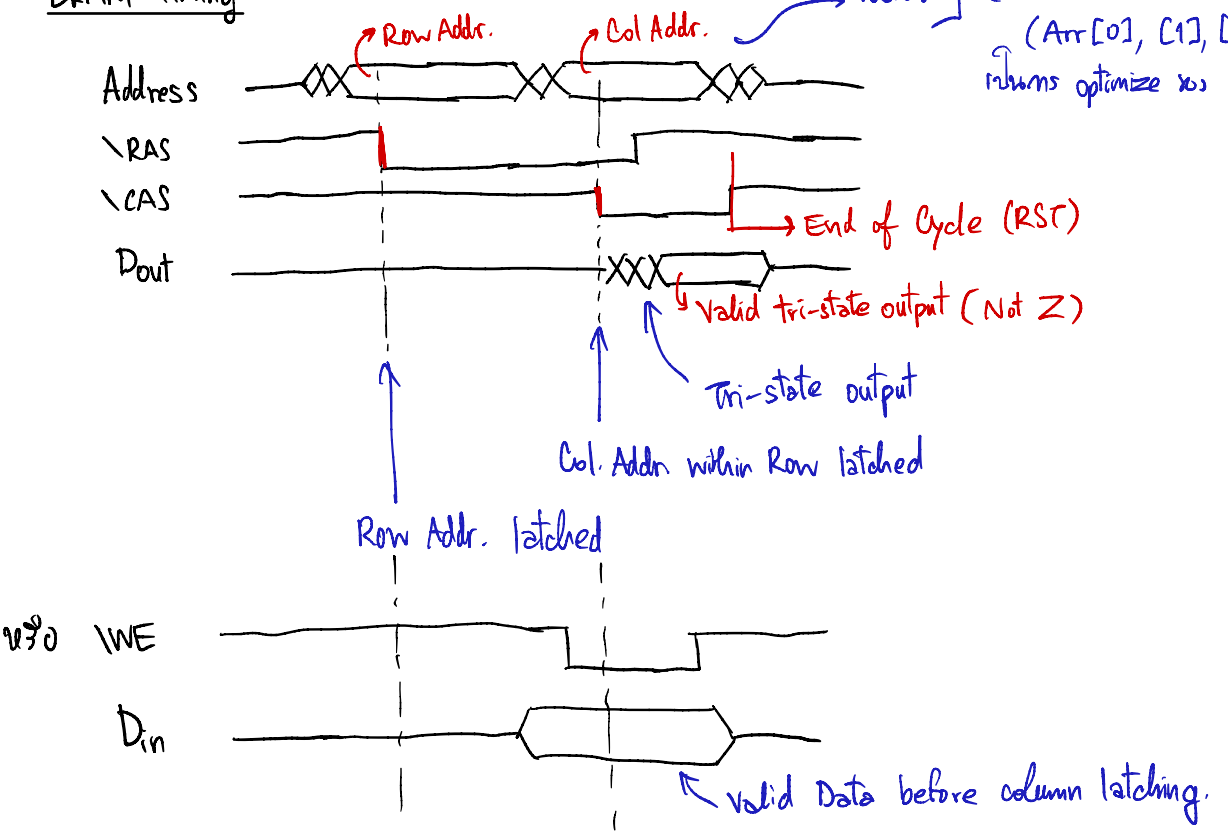


Circuit of DRAM



+ ลด # input จาก 12 → 6 pin
โดยการ input 2 ครั้งแยะ ด้วย \RAS, \CAS
" Row Address Strobe,
Column Address Strobe "

DRAM Timing



→ ระหว่าง Burst read ใช้ (Arr[0], [1], [2], [3], ...) เรียกว่า optimize ของ controller.

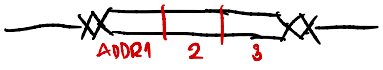
- Nibble.

RAM Refresh

→ คือ Memory คือ Row pointer ไว้

DRAM Modes

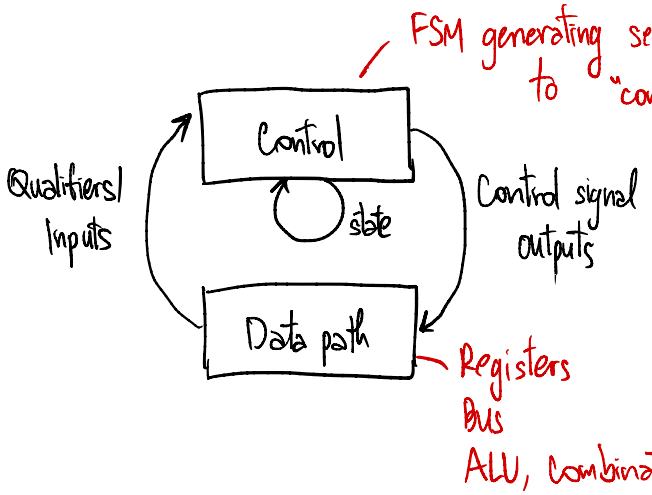
- Page Mode : RAS, CAS, CAS, ..., CAS; RAS, CAS, ... → อ่าน consecutive words in the same "page"
  - Increases throughput
  - Reduce latency
- Static Column : เหมือน Page แต่ ไม่ต้องปิด/เปิด CAS, เปลี่ยน Addr. ได้เลย



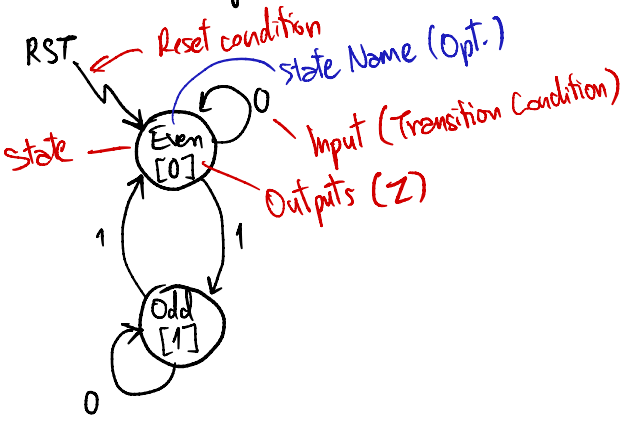
- Nibble Mode : Page Mode ที่ สามารถ CAS ต่อเนื่องได้ (Sometimes, คนมักจะเรียกสั้น row)

Finite State Machine Design

Computer Hardware : Data path + Control



e.g. Odd parity checker



State Name (Symbolic) → binary (Encoded)

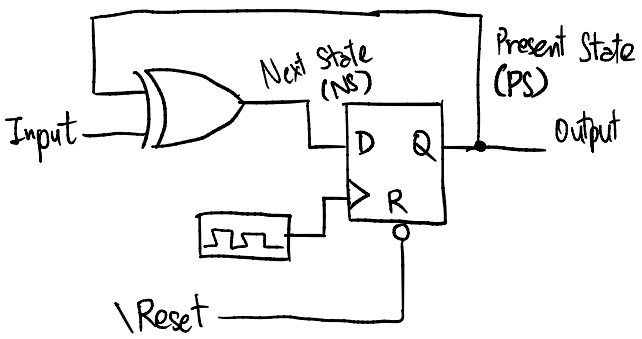
Symbolic :

S	I	S+/Z
Even	0	→ Even/0
Even	1	→ Odd/1
⋮		

Encoded:

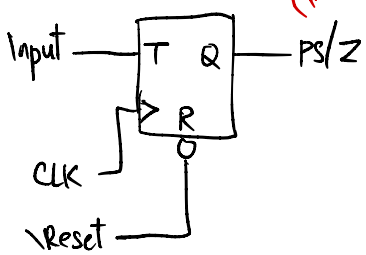
S	I	S+/Z
0	0	0/0
0	1	1/1

Using DFF



Using TFF

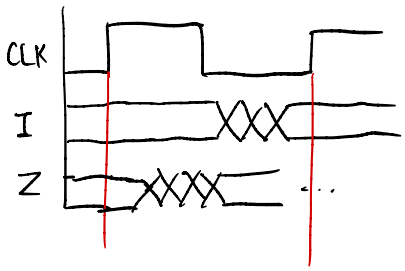
(หรือ PS = Z)



Timing Design : - Clocking Event : Pos. Edge, Neg. Edge, Level-sensitive

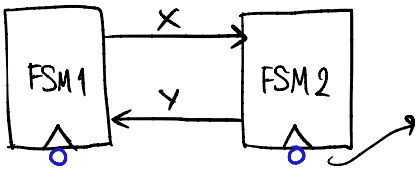
- T_{su}, T_h Consideration : Input stable BEFORE clocking event

e.g. Pos. Edge



Using multiple / partitioning FSM

- Communicating FSM

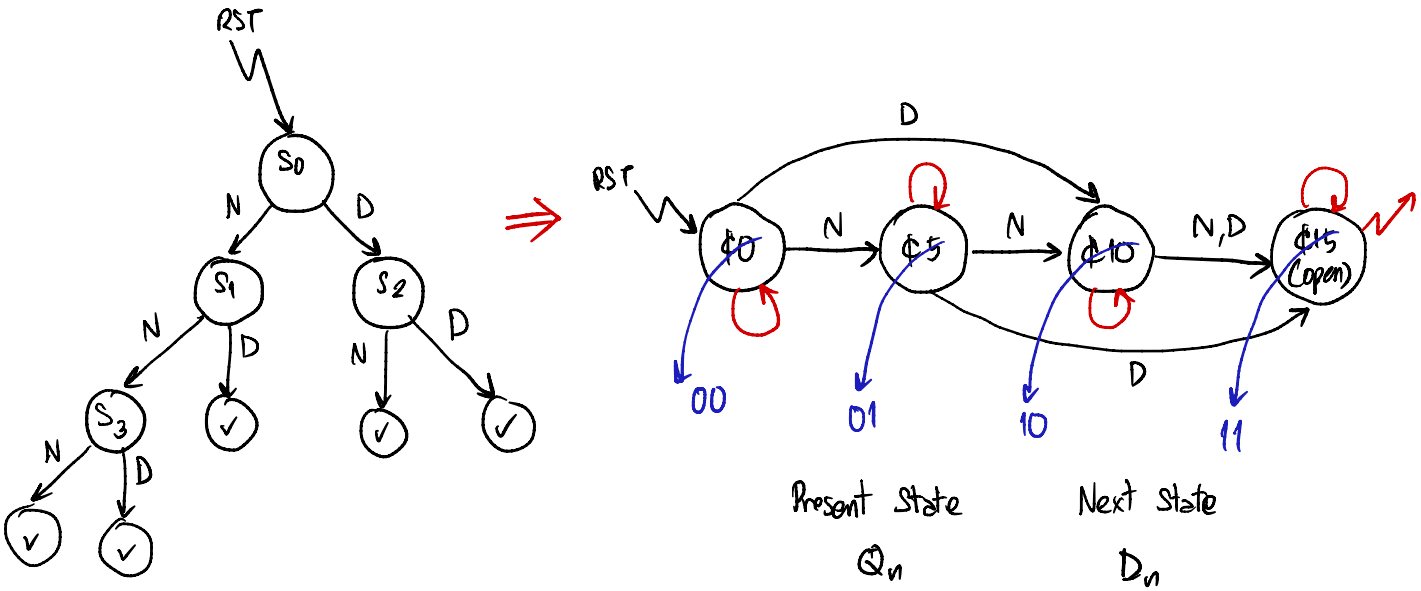


ใช้ Edge Trigger ทั้งหมด

- State Minimization , Design Approach

Abstract Design → Encode → Choose FF → Implement

e.g. Vending Machine รับเหรียญ (Overflow, no changes)

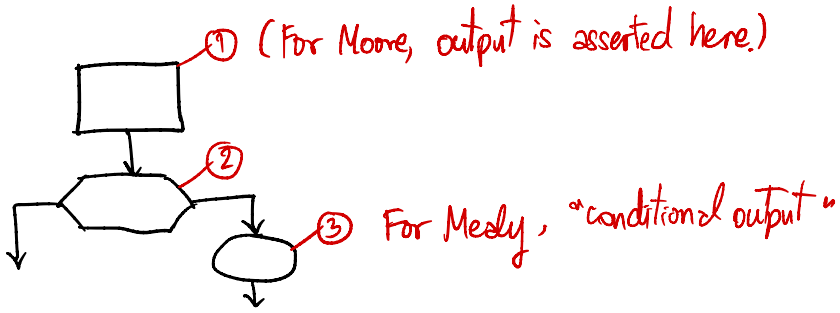


Alternative Representation : ASM Chart

Hardware Descriptive Language : VHDL, Verilog

→ ASM Notation

- State Box ①
- Decision Box ②
- Output Box ③



+ See examples in DIG LO LAB, ASM Slides

FSM State Reduction

- Row Matching Method : รวม case ที่ Q^+ และ Z เหมือนกัน

e.g.

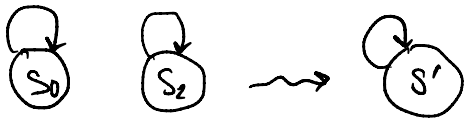
Input Seq.	Q	Q^+ x=0	Q^+ x=1	Z x=0	Z x=1	
011	S_{10}	S_0	S_0	1	0	(011 or 101)
101	S_{12}	S_0	S_0	1	0	
000	S_7	S_0	S_0	0	0	not (011 or 101)
001	S_8					
010	S_9					
100	S_{11}					
110	S_{13}					
111	S_{14}					

Problem : might not be most reduced state

e.g. Odd-parity checker

Q	Q^+ x=0	Q^+ x=1	Z
S_0	S_0	S_1	0
S_1	S_1	S_2	1
S_2	S_2	S_1	0

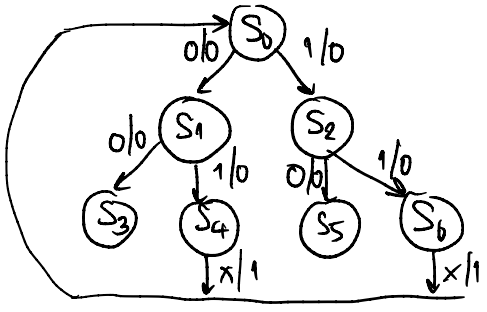
Should merge



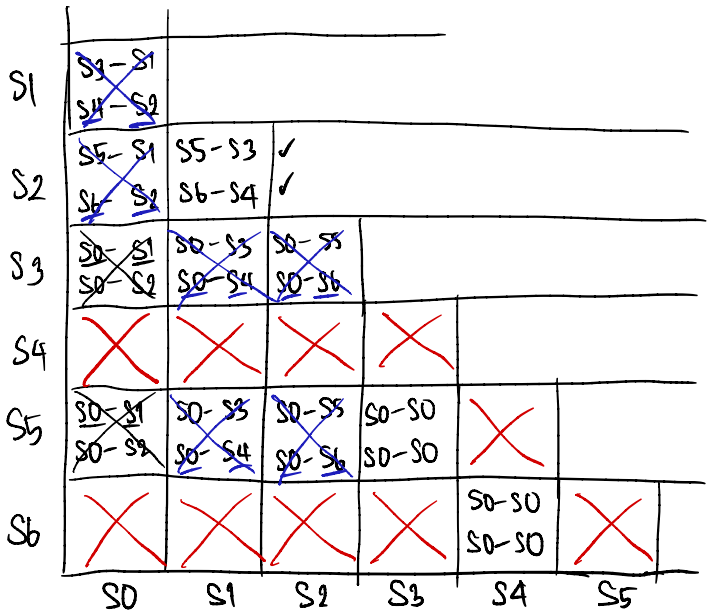
- Implication Method : หา most reduced states โดย

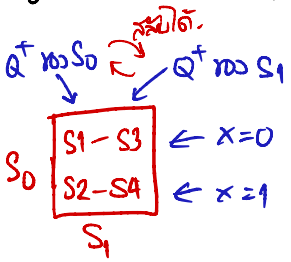
e.g. Output 1 เมื่อ serial seq. เป็น 010 หรือ 110

Seq.	Q	$Q^+(0)$	$Q^+(1)$	Z(0)	Z(1)
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_0	S_0	0	0
01	S_4			1	0
10	S_5			0	0
11	S_6			1	0



→ ทำตารางความสัมพันธ์ S_i, S_j (S_j, S_i เหมือน S_i, S_j ทำแค่ tri lower)





Seq.	Q	$Q^+(0)$	$Q^+(1)$	z(0)	z(1)
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_0	S_0	0	0
01	S_4			1	0
10	S_5			0	0
11	S_6			1	0

1. Cross out คู่ที่ไม่น่าจะรวมได้ (output ไม่ตรงกันแน่ๆ)

(X)

2. Cross out คู่ A-B ที่โดน cross ไปใน 1.

(X) เช่น S2-S4 ในช่อง S1 □ S_0

(ไล่จากบนลงล่าง, ซ้ายไปขวา) (↓↓↓)

↓ →

3. ทำซ้ำ cross out คู่ที่โดน cross ไปใน 2.

(X) ไปเรื่อยๆ จนหมด

∴ เหลือคู่ช่อง S2 □ S_1, S5 □ S_3, S6 □ S_4

→ S1S2, S3S5, S4S6 ✱✱

กลับมา State table!

Seq.	Q	$Q^+(0)$	$Q^+(1)$	z(0)	z(1)
Reset	S_0	S_1	S_2	0	0
0	~~S_1~~ → S_1'	~~S_3~~ S_3'	~~S_4~~ S_4'	0	0
1	S_2	S_5	S_6	0	0
00	~~S_3~~ → S_3'	S_0	S_0	0	0
01	~~S_4~~ → S_4'			1	0
10	S_5			0	0
11	S_6			1	0

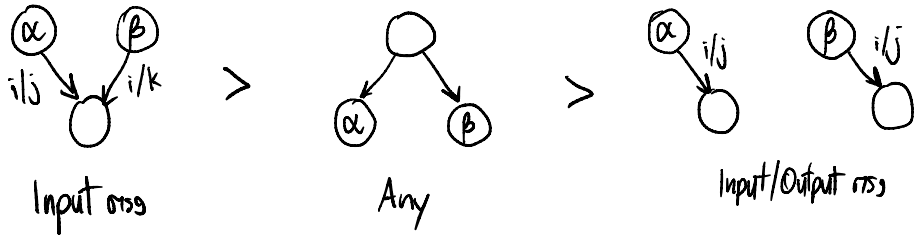
เหลือ

Reset	S_0	S_1	S_2	0	0
0 or 1	S_1'	S_3'	S_4'	0	0
00 or 10	S_3'	S_0	S_0	0	0
01 or 11	S_4'	S_0	S_0	1	0

- State Assignment (State Encoding)

  ↳ ให้ State ที่อยู่ติดกัน ต่างกันน้อย bit น้อยที่สุด (อาจใช้ K-map ช่วยดู "ติด" กัน)

Priority :



Input ตรง Any Input/Output ตรง

- Encode แบบ "One Hot Encodings"

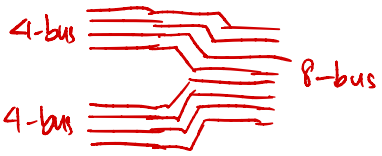
  ↳ #states = #bits

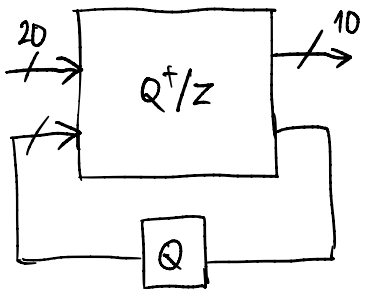
การเลือก Flip Flop

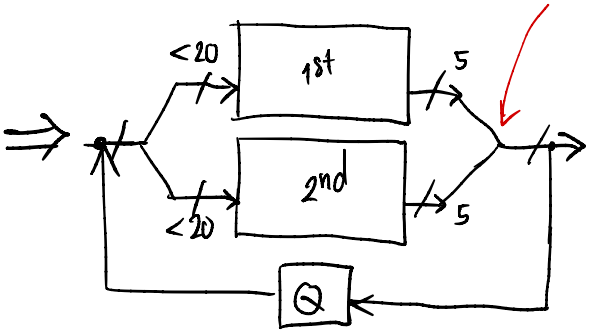
J-K : #gate น้อย, #สัญญาณเยอะ

D : implement ง่ายสุด

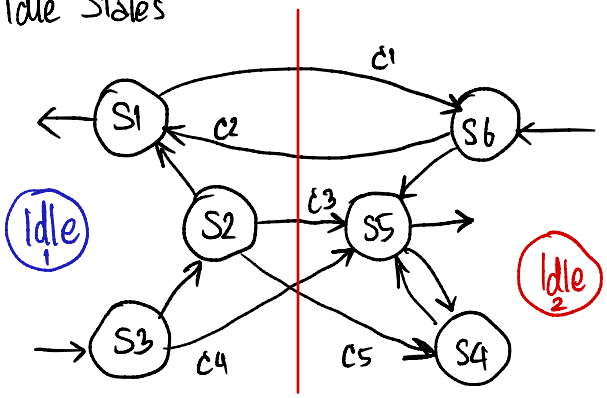
FSM Partitioning : Limited I/O, limited programmable IC, ...







→ Idle States

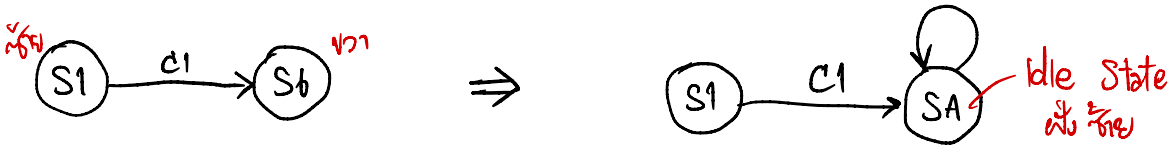


Before : 6 states

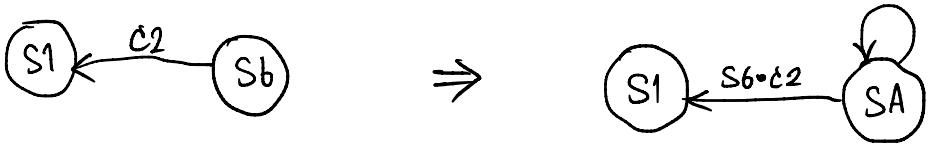
- แบ่ง FSM เป็น $FSM_1, FSM_2, \ldots, FSM_n$ ให้ Idle state เพื่อให้ FSM_i ไม่ต้องวิ่งไปหา state ที่ไม่อยู่ใน FSM_i
- เช่น อยู่ S5 ใน FSM ขวา, ถ้าใช้ FSM ซ้าย ต้อง Idle ก่อน

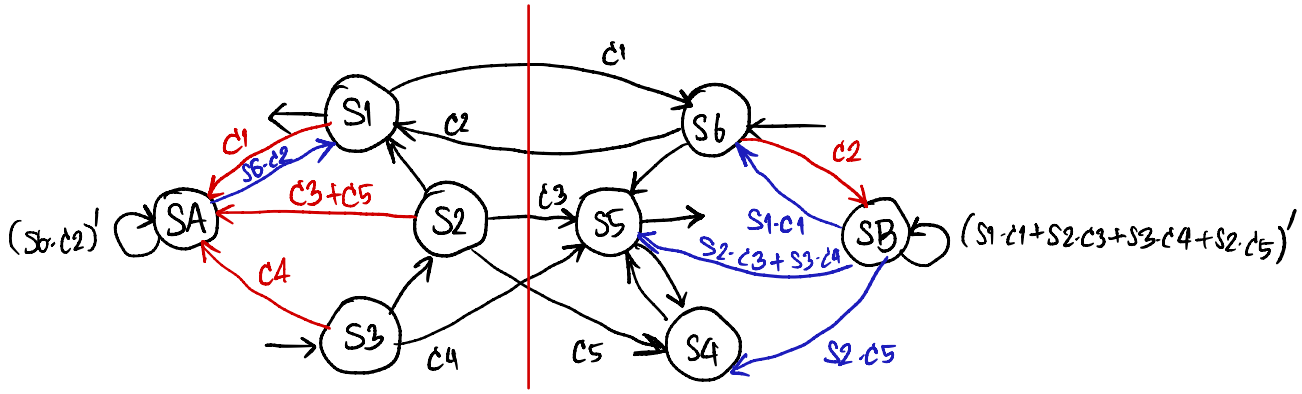
Rules for Partitioning

1. Source Transformation

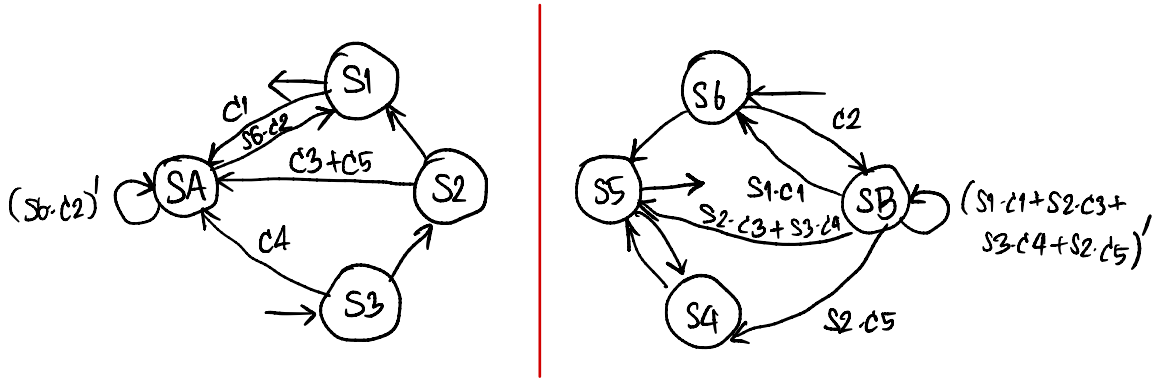


2. Destination Transformation





Idle States Transform



After : 2 FSM , 4+4 states