Introduction to Synchronous Sequential Circuits

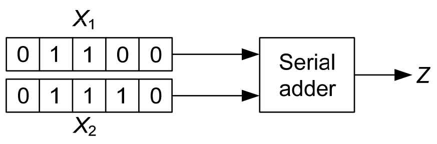
Zvi Kohavi and Niraj K. Jha

Sequential Circuits and Finite-state Machines

Sequential circuit: its outputs a function of external inputs as well as stored information

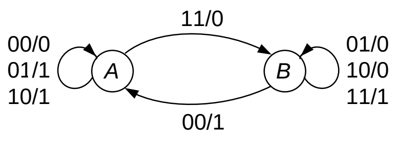
Finite-state machine (FSM): abstract model to describe the synchronous sequential machine and its spatial counterpart, the iterative network

Serial binary adder example: block diagram, addition process, state table and state diagram



$$\begin{array}{rcccccl} & t_5 & t_4 & t_3 & t_2 & t_1 & \\ & 0 & 1 & 1 & 0 & 0 & = X_1 \\ + & 0 & 1 & 1 & 1 & 0 & = X_2 \\ \hline & 1 & 1 & 0 & 1 & 0 & = Z \end{array}$$

	NS, z			
PS	$x_1x_2 = 00$	01	11	10
A	$A, 0$	$A, 1$	$B, 0$	$A, 1$
B	$A, 1$	$B, 0$	$B, 1$	$B, 0$



State Assignment

Device with two states capable of storing information: delay element with input Y and output y

- Two states: $y = 0$ and $y = 1$
- Since the present input value Y of the delay element is equal to its next output value: the input value is referred to as the next state of the delay
  - $Y(t) = y(t+1)$

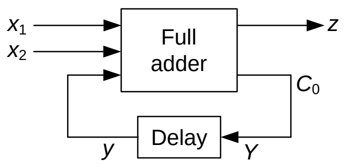
Example: assign state $y = 0$ to state A of the adder and $y = 1$ to B

- The value of y at t_i corresponds to the value of the carry generated at t_{i-1}
- Process of assigning the states of a physical device to the states of the serial adder: called state assignment
- Output value y: referred to as the state variable
- Transition/output table for the serial adder:

$$Y = x_1x_2 + x_1y + x_2y$$

$$z = x_1 \oplus x_2 \oplus y$$

	Next state Y				Output z			
y	x_1x_2 00	01	11	10	x_1x_2 00	01	11	10
0	0	0	1	0	0	1	0	1
1	0	1	1	1	1	0	1	0

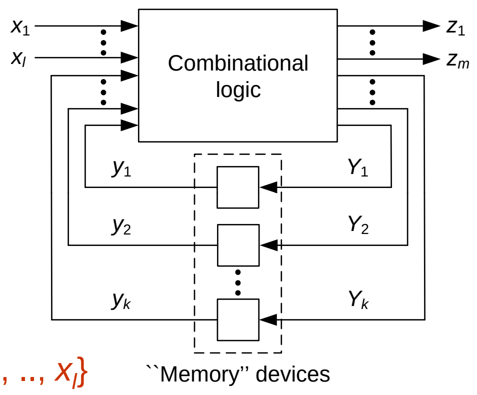


FSM: Definitions

FSMs: whose past histories can affect their future behavior in only a finite number of ways

- Serial adder: its response to the signals at time t is only a function of these signals and the value of the carry at t-1
  - Thus, its input histories can be grouped into just two classes: those resulting in a 1 carry and those resulting in a 0 carry at t
- Thus, every finite-state machine contains a finite number of memory devices: which store the information regarding the past input history

Synchronous Sequential Machines



Input variables: $\{x_1, x_2, .., x_l\}$

Input configuration, symbol, pattern or vector: ordered l-tuple of 0's and 1's

Input alphabet: set of $p = 2^l$ distinct input patterns

- Thus, input alphabet $I = \{I_1, I_2, .., I_p\}$
- Example: for two variables x_1 and x_2
  - $I = \{00, 01, 10, 11\}$

Output variables: $\{z_1, z_2, .., z_m\}$

Output configuration, symbol, pattern or vector: ordered m-tuple of 0's and 1's

Output alphabet: set of $q = 2^m$ distinct output patterns

- Thus, output alphabet $O = \{O_1, O_2, .., O_q\}$

Synchronous Sequential Machines (Contd.)

Set of state variables: $\{y_1, y_2, .., y_k\}$

Present state: combination of values at the outputs of k memory elements

Set S of $n = 2^k$ k-tuples: entire set of states $S = \{S_1, S_2, .., S_n\}$

Next state: values of Y's

Synchronization achieved by means of clock pulses feeding the memory devices

Initial state: state of the machine before the application of an input sequence to it

Final state: state of the machine after the application of the input sequence

Memory Elements and Their Excitation Functions

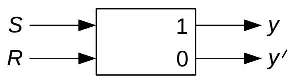
To generate the *Y*'s: memory devices must be supplied with appropriate input values

- Excitation functions: switching functions that describe the impact of x_i's and y_j's on the memory-element input
- Excitation table: its entries are the values of the memory-element inputs

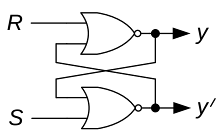
Most widely used memory elements: flip-flops, which are made of latches

- Latch: remains in one state indefinitely until an input signals directs it to do otherwise

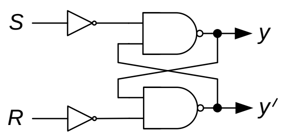
Set-reset of *SR* latch:



(*a*) Block diagram.



(*b*) NOR latch.



(*c*) NAND latch.

SR Latch (Contd.)

Excitation characteristics and requirements:

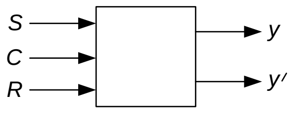
$y(t)$	$S(t)$	$R(t)$	$y(t+1)$
0	0	0	0
0	0	1	0
0	1	1	?
0	1	0	1
1	1	0	1
1	1	1	?
1	0	1	0
1	0	0	1

Circuit change		*Required value*	
From:	*To:*	*S*	*R*
0	0	0	–
0	1	1	0
1	0	0	1
1	1	–	0

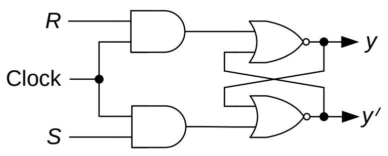
$$\begin{aligned} RS &= 0 \\ y(t+1) &= R'y(t) + S \end{aligned}$$

Clocked *SR* latch: all state changes synchronized to clock pulses

- Restrictions placed on the length and frequency of clock pulses: so that the circuit changes state no more than once for each clock pulse



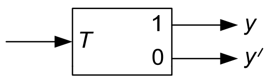
(*a*) Block diagram.



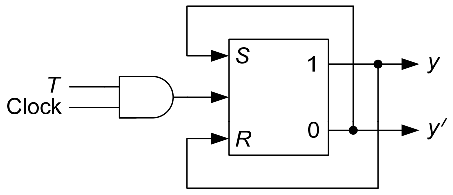
(*b*) Logic diagram.

Trigger or *T* Latch

Value 1 applied to its input triggers the latch to change state



(*a*) Block diagram.



(*b*) Deriving the T latch from the clocked SR latch.

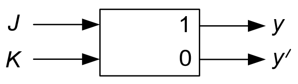
Excitations requirements:

Circuit change		*Required*
From:	*To:*	*value T*
0	0	0
0	1	1
1	0	1
1	1	0

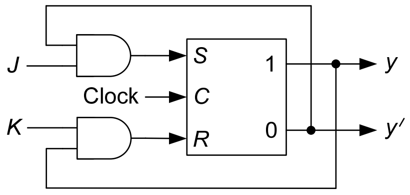
$$y(t+1) = Ty'(t) + T'y(t)$$
$$= T \oplus y(t)$$

The *JK* Latch

Unlike the *SR* latch, *J* = *K* = 1 is permitted: when it occurs, the latch acts like a trigger and switches to the complement state



(*a*) Block diagram.



(*b*) Constructing the JK latch from the clocked SR latch.

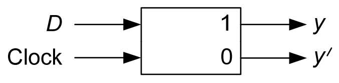
Excitation requirements:

Circuit change		*Required value*	
From:	*To:*	*J*	*K*
0	0	0	–
0	1	1	–
1	0	–	1
1	1	–	0

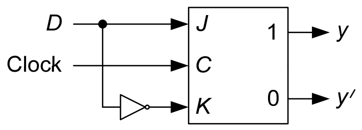
The *D* Latch

The next state of the *D* latch is equal to its present excitation:

$$y(t+1) = D(t)$$



(*a*) Block diagram.



(*b*) Transforming the JK latch to the D latch.

Clock Timing

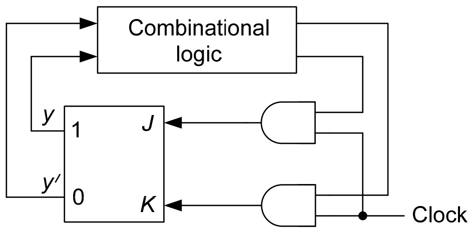
Clocked latch: changes state only in synchronization with the clock pulse and no more than once during each occurrence of the clock pulse

Duration of clock pulse: determined by circuit delays and signal propagation time through the latches

- Must be long enough to allow latch to change state, and
- Short enough so that the latch will not change state twice due to the same excitation

Excitation of a *JK* latch within a sequential circuit:

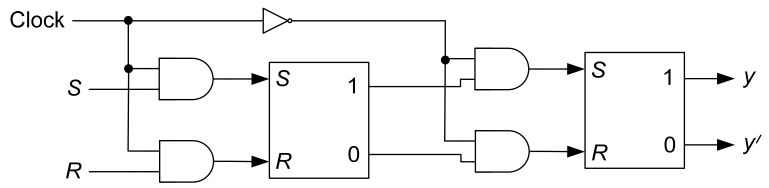
- Length of the clock pulse must allow the latch to generate the *y*'s
- But should not be present when the values of the *y*'s have propagated through the combinational circuit



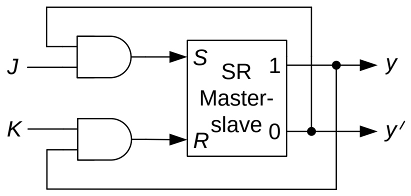
Master-slave Flip-flop

Master-slave flip-flop: a type of synchronous memory element that eliminates the timing problems by isolating its inputs from its outputs

Master-slave *SR* flip-flop:



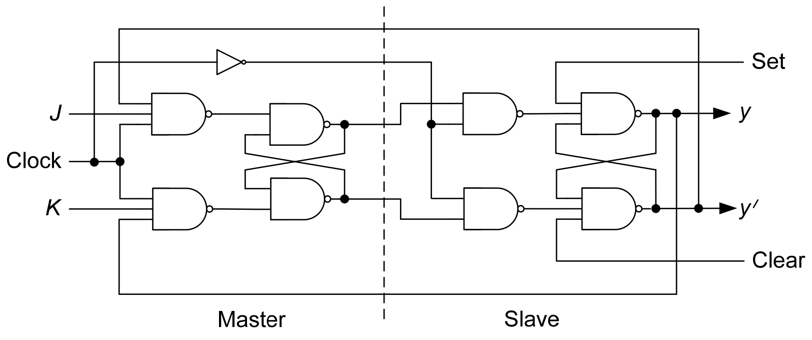
Master-slave *JK* flip-flop: since master-slave *SR* flip-flop suffers from the problem that both its inputs cannot be 1, it can be converted to a *JK* flip-flip



Master-slave JK Flip-flop with Additional Inputs

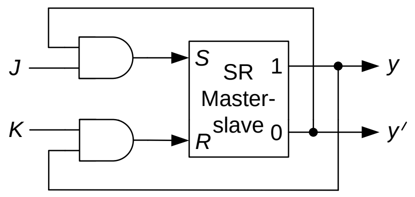
Direct set and clear inputs: override regular input signals and clock

- To set the slave output to 0: make set = 1 and clear = 0
- To set the slave output to 1: make set = 0 and clear = 1
- Assigning 0 to both set and clear: not allowed
- Assigning 1 to both set and clear: normal operation
- Useful in design of counters and shift registers



1’s Catching and 0’s Catching

SR and *JK* flip-flops suffer from 1’s catching and 0’s catching

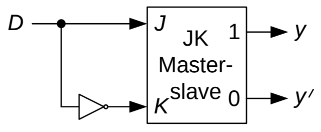


Master latch is transparent when the clock is high

- When the output of the slave latch is at 0 and the *J* input has a static-0 hazard (a transient glitch to 1) after the clock has gone high: then the master latch catches this set condition
  - It then passes the 1 to the slave latch when the clock goes low
- Similarly, when the output of the slave latch is at 1 and the *K* input has a static-0 hazard after the clock has gone high: then the master latch catches this reset condition
  - It then passes the 0 to the slave latch when the clock goes low

D flip-flop

Master-slave *D* flip-flop avoids the above problem: even when a static hazard occurs at the *D* input when the clock is high, the output of the master latch reverts to its old value when the glitch goes away



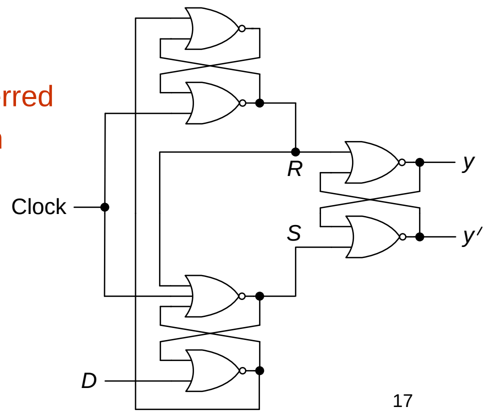
Edge-triggered Flip-flop

Positive (negative) edge-triggered *D* flip-flip: stores the value at the *D* input when the clock makes a 0 -> 1 (1 -> 0) transition

- Any change at the *D* input after the clock has made a transition does not have any effect on the value stored in the flip-flop

A negative edge-triggered *D* flip-flop:

- When the clock is high, the output of the bottommost (topmost) NOR gate is at *D'* (*D*), whereas the *S*-*R* inputs of the output latch are at 0, causing it to hold previous value
- When the clock goes low, the value from the bottommost (topmost) NOR gate gets transferred as *D* (*D'*) to the *S* (*R*) input of the output latch
  - Thus, output latch stores the value of *D*
- If there is a change in the value of the *D* input after the clock has made its transition, the bottommost NOR gate attains value 0
  - However, this cannot change the *SR* inputs of the output latch



Synthesis of Synchronous Sequential Circuits

Main steps:

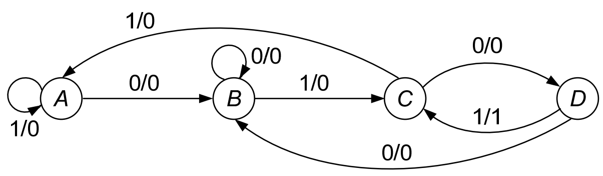
1. From a word description of the problem, form a state diagram or table
2. Check the table to determine if it contains any redundant states
   - If so, remove them (Chapter 10)
3. Select a state assignment and determine the type of memory elements
4. Derive transition and output tables
5. Derive an excitation table and obtain excitation and output functions from their respective tables
6. Draw a circuit diagram

Sequence Detector

One-input/one-output sequence detector: produces output value 1 every time sequence 0101 is detected, else 0

- Example: 010101 -> 000101

State diagram and state table:



	NS, z	
PS	$x = 0$	$x = 1$
A	$B, 0$	$A, 0$
B	$B, 0$	$C, 0$
C	$D, 0$	$A, 0$
D	$B, 0$	$C, 1$

Transition and output tables:

	Y_1Y_2		z	
y_1y_2	$x = 0$	$x = 1$	$x = 0$	$x = 1$
$A \rightarrow 00$	01	00	0	0
$B \rightarrow 01$	01	11	0	0
$C \rightarrow 11$	10	00	0	0
$D \rightarrow 10$	01	11	0	1

Sequence Detector (Contd.)

Excitation and output maps:

$z = xy_1y_2'$
$y_1 = x'y_1y_2 + xy_1'y_2 + xy_1y_2'$
$y_2 = y_1y_2' + x'y_1' + y_1'y_2$

$y_1y_2 \backslash x$	0	1
00		
01		
11		
10		1

(*a*) *z* map.

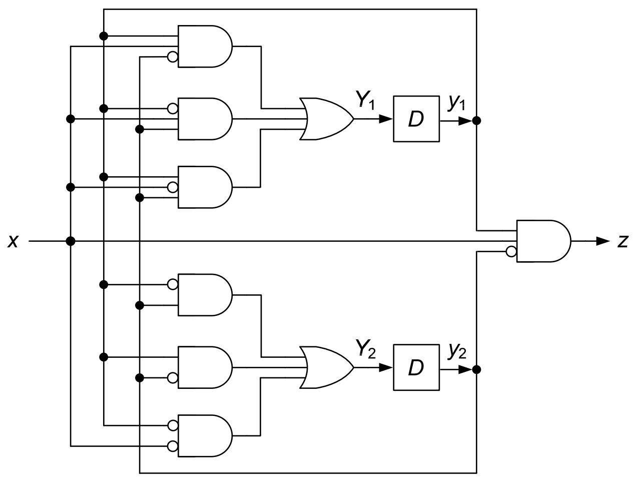
$y_1y_2 \backslash x$	0	1
00		
01		1
11	1	
10		1

(*b*) Y_1 map.

$y_1y_2 \backslash x$	0	1
00	1	
01	1	1
11		
10	1	1

(*c*) Y_2 map.

Logic diagram:



Sequence Detector (Contd.)

Another state assignment:

y_1y_2	Y_1Y_2 $x=0$	Y_1Y_2 $x=1$	z $x=0$	z $x=1$
$A \rightarrow 00$	01	00	0	0
$B \rightarrow 01$	01	10	0	0
$C \rightarrow 10$	11	00	0	0
$D \rightarrow 11$	01	10	0	1

$z = xy_1y_2$

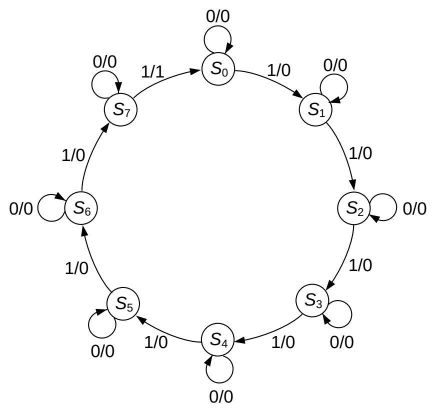
$Y_1 = x'y_1y_2' + xy_2$

$Y_2 = x'$

Binary Counter

One-input/one-output modulo-8 binary counter: produces output value 1 for every eighth input 1 value

State diagram and state table:



PS	NS $x=0$	NS $x=1$	$Output$ $x=0$	$Output$ $x=1$
S_0	S_0	S_1	0	0
S_1	S_1	S_2	0	0
S_2	S_2	S_3	0	0
S_3	S_3	S_4	0	0
S_4	S_4	S_5	0	0
S_5	S_5	S_6	0	0
S_6	S_6	S_7	0	0
S_7	S_7	S_0	0	1

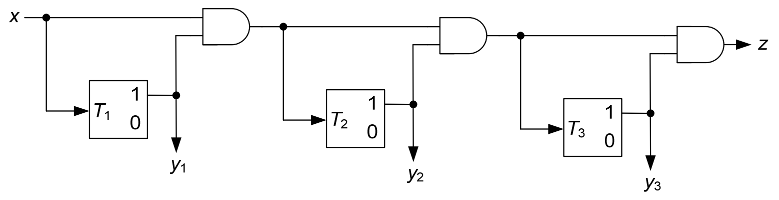
Binary Counter (Contd.)

Transition and output tables:

PS	NS		z	
$y_3y_2y_1$	$x=0$	$x=1$	$x=0$	$x=1$
000	000	001	0	0
001	001	010	0	0
010	010	011	0	0
011	011	100	0	0
100	100	101	0	0
101	101	110	0	0
110	110	111	0	0
111	111	000	0	1

Excitation table for *T* flip-flops and logic diagram:

	$T_3T_2T_1$	
$y_3y_2y_1$	$x=0$	$x=1$
000	000	001
001	000	011
010	000	001
011	000	111
100	000	001
101	000	011
110	000	001
111	000	111



$T_1 = x$

$T_2 = xy_1$

$T_3 = xy_1y_2$

$z = xy_1y_2y_3$

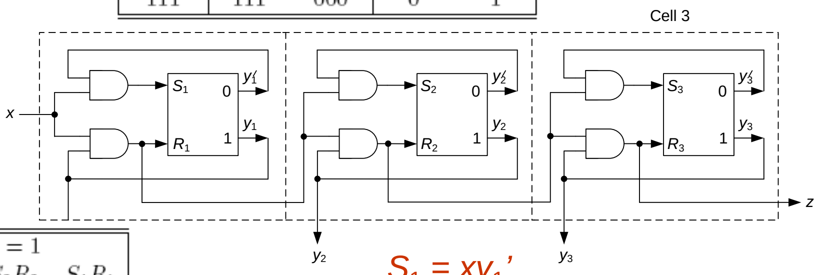
Implementing the Counter with *SR* Flip-flops

Transition and output tables:

PS	NS		z	
$y_3y_2y_1$	$x=0$	$x=1$	$x=0$	$x=1$
000	000	001	0	0
001	001	010	0	0
010	010	011	0	0
011	011	100	0	0
100	100	101	0	0
101	101	110	0	0
110	110	111	0	0
111	111	000	0	1

Excitation table for *SR* flip-flops and logic diagram:

- Trivially extensible to modulo-16 counter



	$x=0$			$x=1$		
$y_3y_2y_1$	S_3R_3	S_2R_2	S_1R_1	S_3R_3	S_2R_2	S_1R_1
000	0–	0–	0–	0–	0–	10
001	0–	0–	–0	0–	10	01
010	0–	–0	0–	0–	–0	10
011	0–	–0	–0	10	01	01
100	–0	0–	0–	–0	0–	10
101	–0	0–	–0	–0	10	01
110	–0	–0	0–	–0	–0	10
111	–0	–0	–0	01	01	01

$S_1 = xy_1'$

$R_1 = xy_1$

$S_2 = xy_1y_2'$

$R_2 = xy_1y_2$

$S_3 = xy_1y_2y_3'$

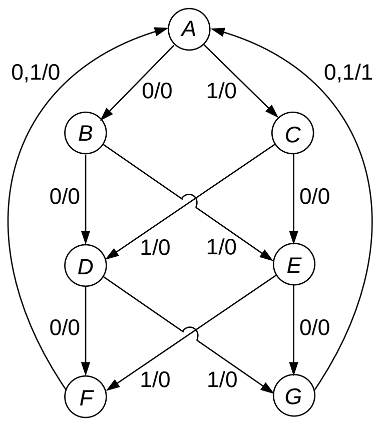
$R_3 = z = xy_1y_2y_3$

Parity-bit Generator

Serial parity-bit generator: receives coded messages and adds a parity bit to every *m*-bit message

- Assume *m* = 3 and even parity

State diagram and state table:



PS	NS		z	
$y_1y_2y_3$	$x=0$	$x=1$	$x=0$	$x=1$
$A \rightarrow 000$	B	C	0	0
$B \rightarrow 010$	D	E	0	0
$C \rightarrow 011$	E	D	0	0
$D \rightarrow 110$	F	G	0	0
$E \rightarrow 111$	G	F	0	0
$F \rightarrow 100$	A	A	0	0
$G \rightarrow 101$	A	A	1	1

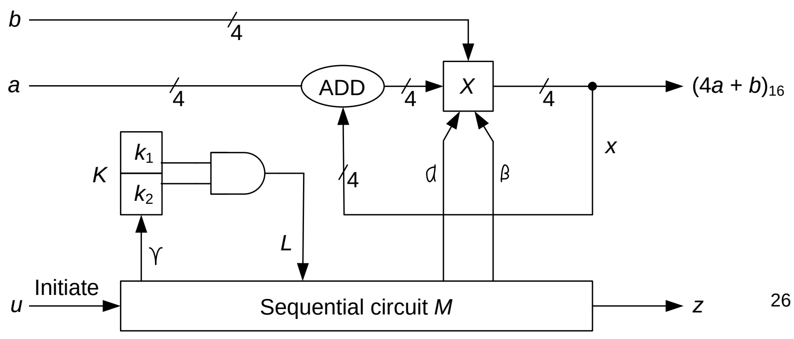
$J_1 = y_2$
$K_1 = y_2'$
$J_2 = y_1'$
$K_2 = y_1$
$J_3 = xy_1' + xy_2$
$K_3 = x + y_2'$
$z = y_2'y_3$

Sequential Circuit as a Control Element

Control element: streamlines computation by providing appropriate control signals

Example: digital system that computes the value of (4*a* + *b*) modulo 16

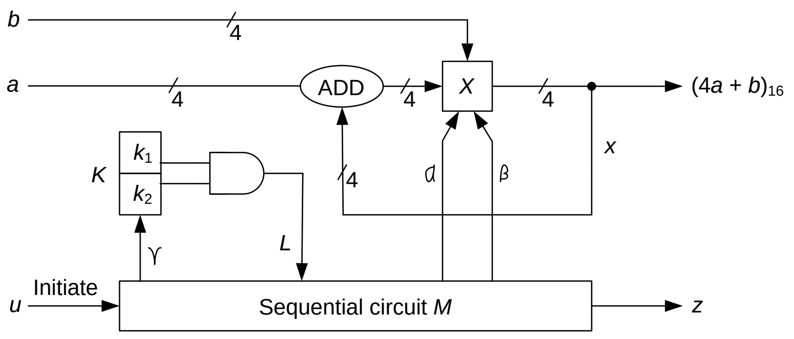
- *a*, *b*: four-bit binary number
- *X*: register containing four flip-flops
- *x*: number stored in *X*
- Register can be loaded with: either *b* or *a* + *x*
- Addition performed by: a four-bit parallel adder
- *K*: modulo-4 binary counter, whose output *L* equals 1 whenever the count is 3 modulo 4



Example (Contd.)

Sequential circuit *M*:

- Input *u*: initiates computation
- Input *L*: gives the count of *K*
- Outputs: α, β, γ, *z*
- When $\alpha = 1$: contents of *b* transferred to *X*
- When $\beta = 1$: values of *x* and *a* added and transferred back to *X*
- When $\gamma = 1$: count of *K* increased by 1
- *z* = 1: whenever final result available in *X*

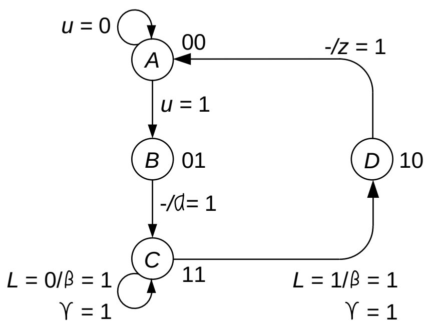


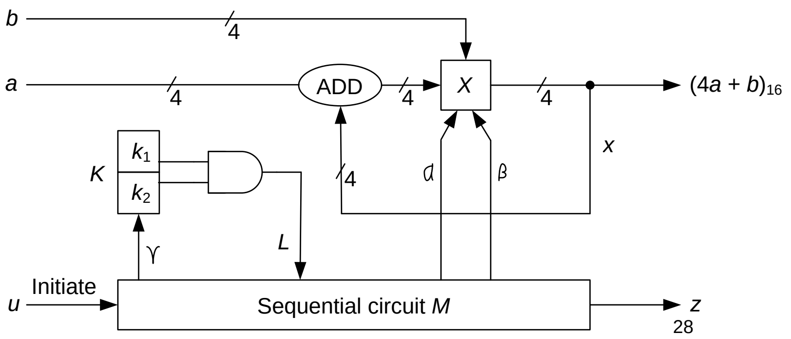
Example (Contd.)

Sequential circuit *M*:

- *K, u, z*: initially at 0
- When *u* = 1: computation starts by setting $\alpha = 1$
  - Causes *b* to be loaded into *X*
- To add *a* to *x*: set $\beta = 1$ and $\gamma = 1$ to keep track of the number of times *a* has been added to *x*
- After four such additions: *z* = 1 and the computation is complete
- At this point: *K* = 0 to be ready for the next computation

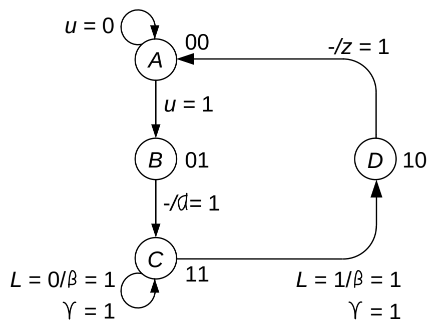
State diagram:





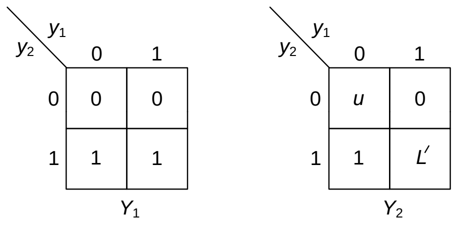
Example (Contd.)

State assignment, transition table, maps and logic diagram:



PS y_1y_2	*NS* Y_1Y_2
00	0u
01	11
11	1L'
10	00

(*a*) Transition table.



(*b*) Maps for Y_1 and Y_2.

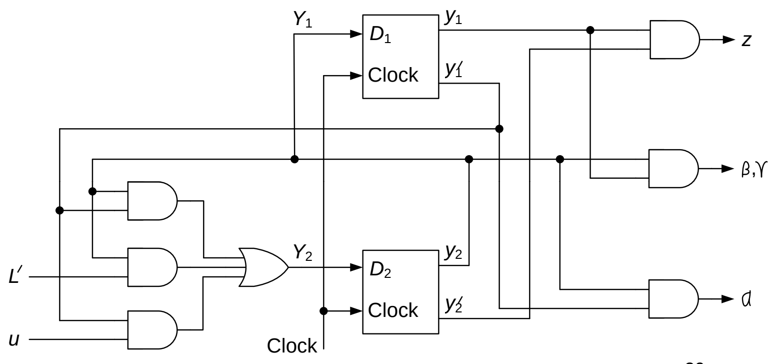
$\alpha = y_1'y_2$

$\beta = \gamma = y_1y_2$

$z = y_1y_2'$

$Y_1 = y_2$

$Y_2 = y_1'y_2 + uy_1' + L'y_2$



(*c*) Logic diagram.