9-Sistemas Secuenciales

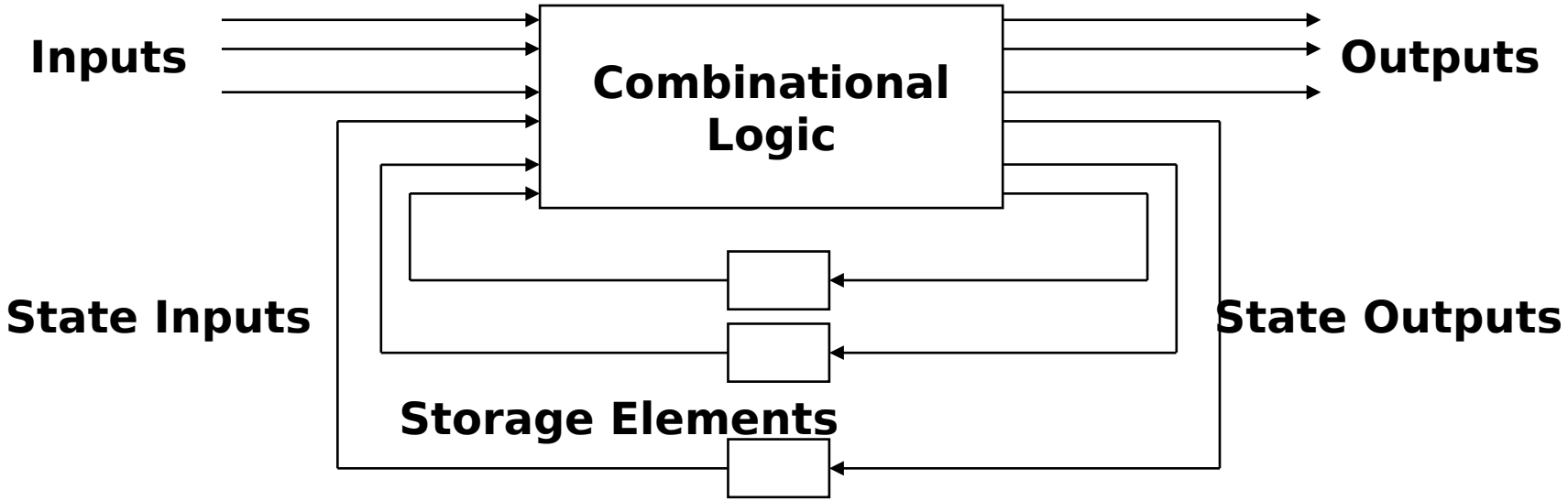
9.1 Máquinas de Estados Finitos

9.2 Mealy y Moore

9.3 Implementación en Verilog

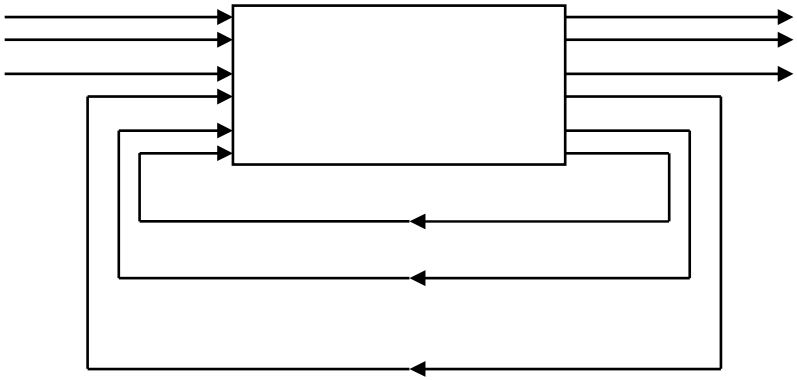
Abstracción

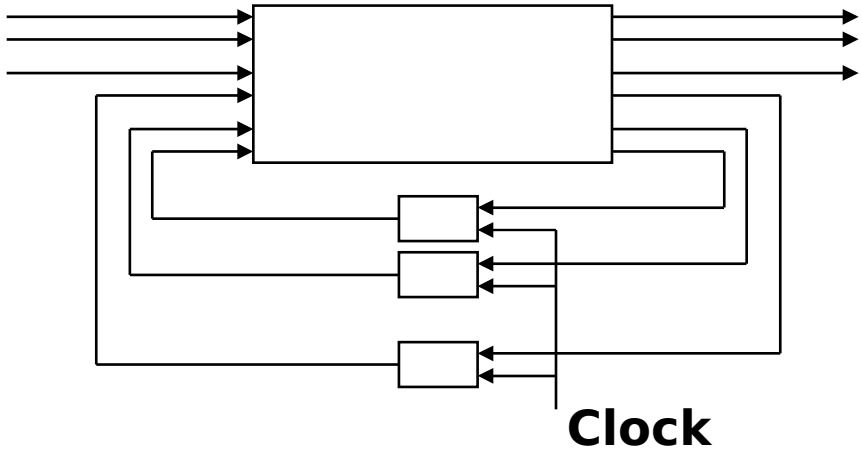
- Dividir circuito en lógica combinacional y estado (state)
- Localizar los enlaces de feedback (loops)
- Implementación de elementos de almacenamiento (storage elements) nos da diferentes formas de lógica secuencial



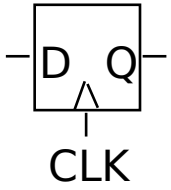
Formas de lógica secuencial

- Asincrónica – estados cambian cuandos los inputs cambian (elemento de almacenamiento pueden ser simples alambres of retardos)
- Sincrónica – cambios de estados ocurren en todos los elementos de almacenamiento al mismo tiempo (de acuerdo a una senal periódica – el reloj o clock)

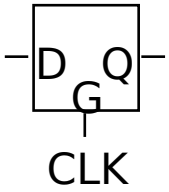




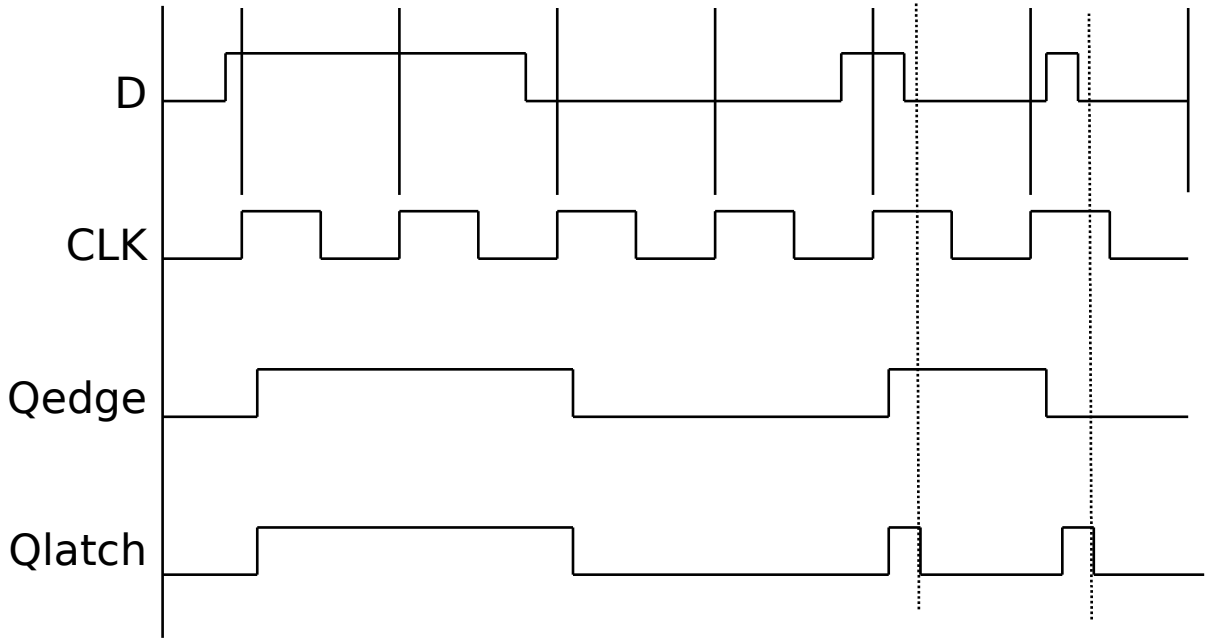
Elementos de almacenamiento: latches y flip-flops



positive edge-triggered flip-flop

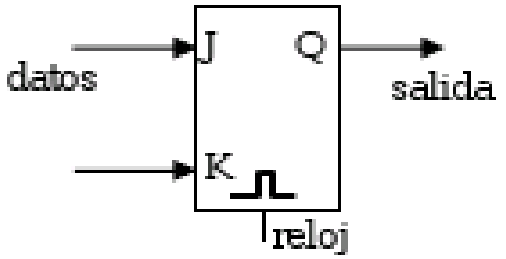


transparent (level-sensitive) latch

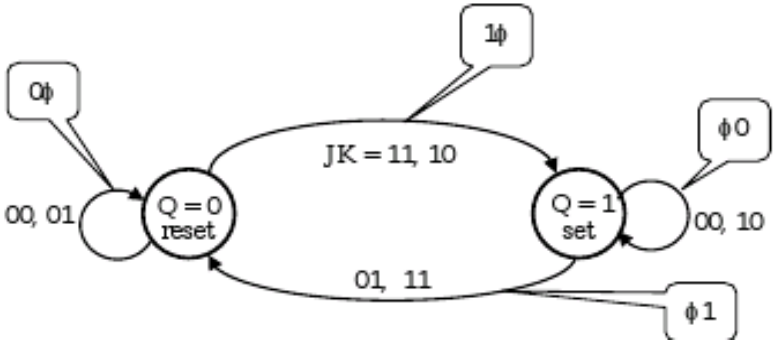


comportamiento no es el mismo si es que los inputs cambian cuando el clock esta alto

Flip Flop JK



- Diagrama de estados



- Tabla transiciones

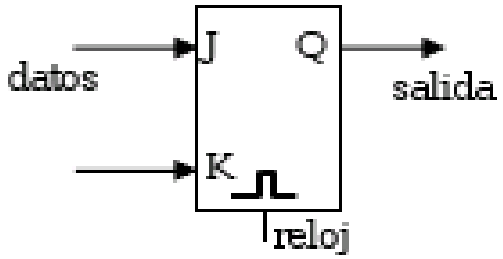
Q(k) \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

Q(k+1)

- Ecuacion caraterística

$$Q(k+1) = J(k)\,\bar{Q}(k) + \bar{K}(k)\,Q(k)$$

Flip Flop JK



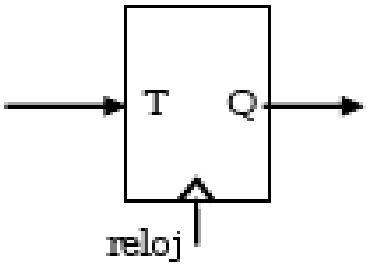
- Tabla Característica

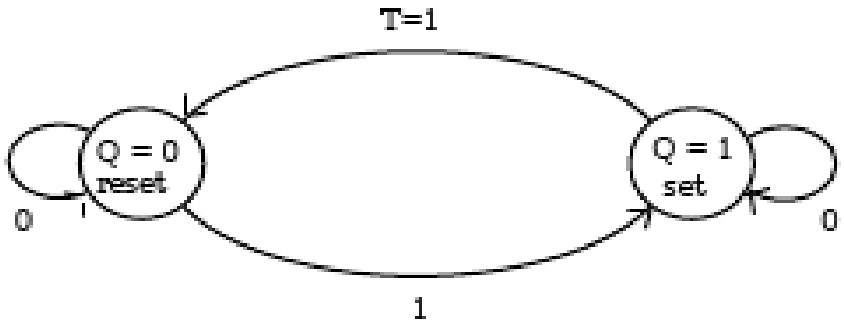
J	K	Q(k+1)
0	0	Q(k)
0	1	0
1	0	1
1	1	$\bar{Q}(k)$

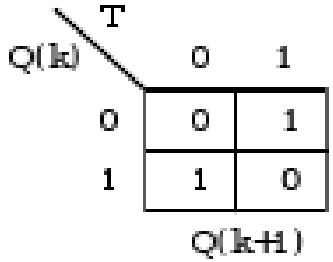
- Tabla de Excitaciones

Q(k) => Q(k+1)		J	K
0	0	0	ϕ
0	1	1	ϕ
1	0	ϕ	1
1	1	ϕ	0

Flip Flop T





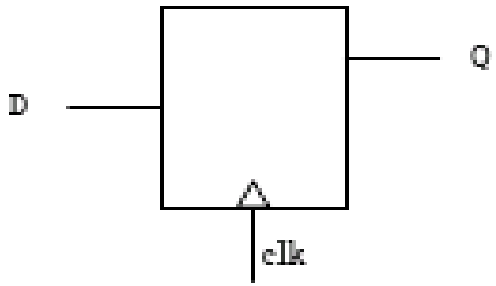


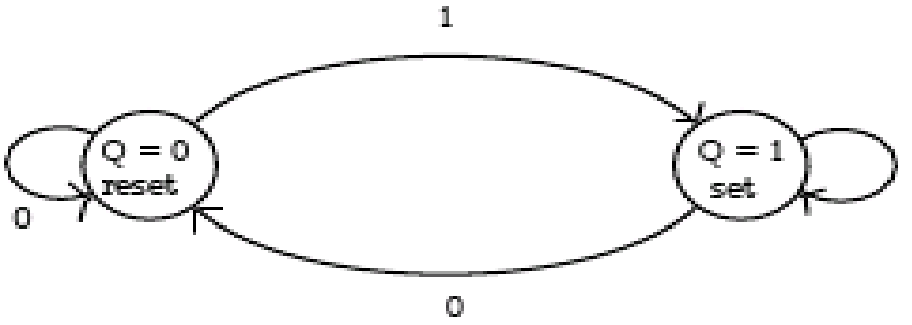
$$Q(k+1) = T(k)\overline{Q}(k) + \overline{T}(k)Q(k) = T(k) \oplus Q(k)$$

T	Q(k+1)
0	Q(k)
1	$\overline{Q}(k)$

Q(k)	-> Q(k+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

Flip Flop D





Q(k) \ D	0	1
0	0	1
1	0	1

Q(k+1)

Ecuación característica

$$Q(k+1) = D(k)$$

Tabla característica

D	Q(k+1)
0	0
1	1

Tabla de excitaciones

Q(k)	-> Q(k+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

Representaciones de máquinas de estados finitos (finite state machines)

- Estados: determinado por posibles valores en elementos de almacenamiento
- Transiciones: cambios de estado
- Reloj (clock): controla cuando los estados pueden cambiar al controlar elementos de almacenamiento

- Lógica secuencial
  - secuencia a través una serie de estados
  - basado en secuencia de valores de señales de input (x)

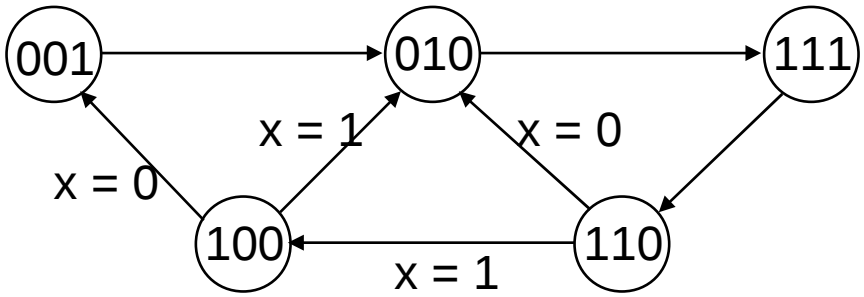
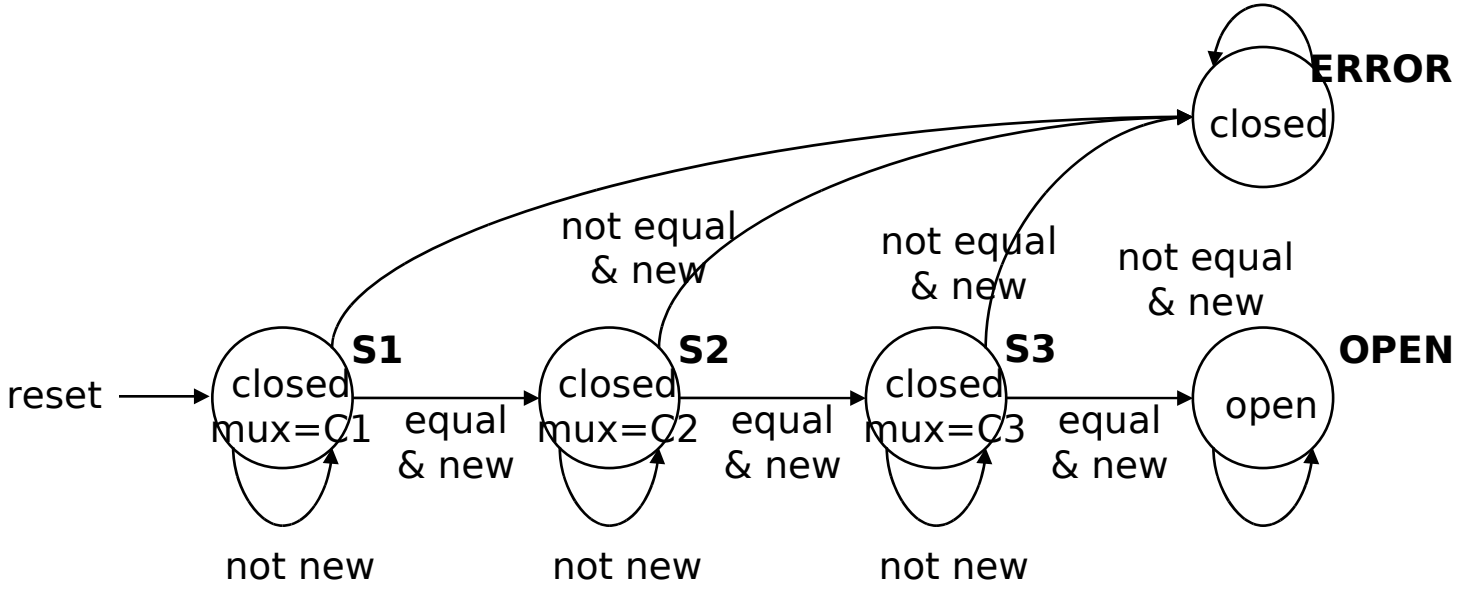


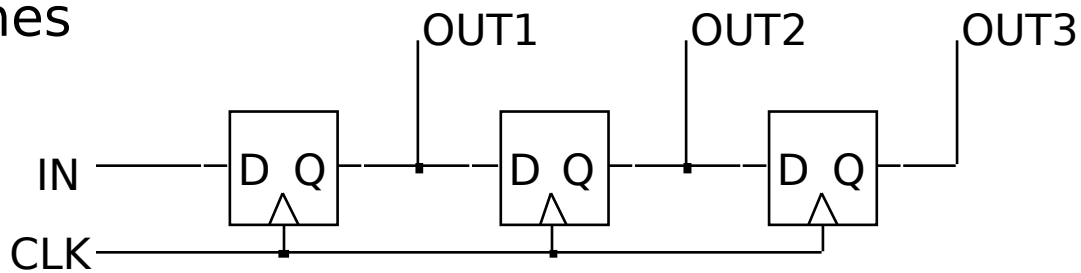
Diagrama de máquina de estados finitos

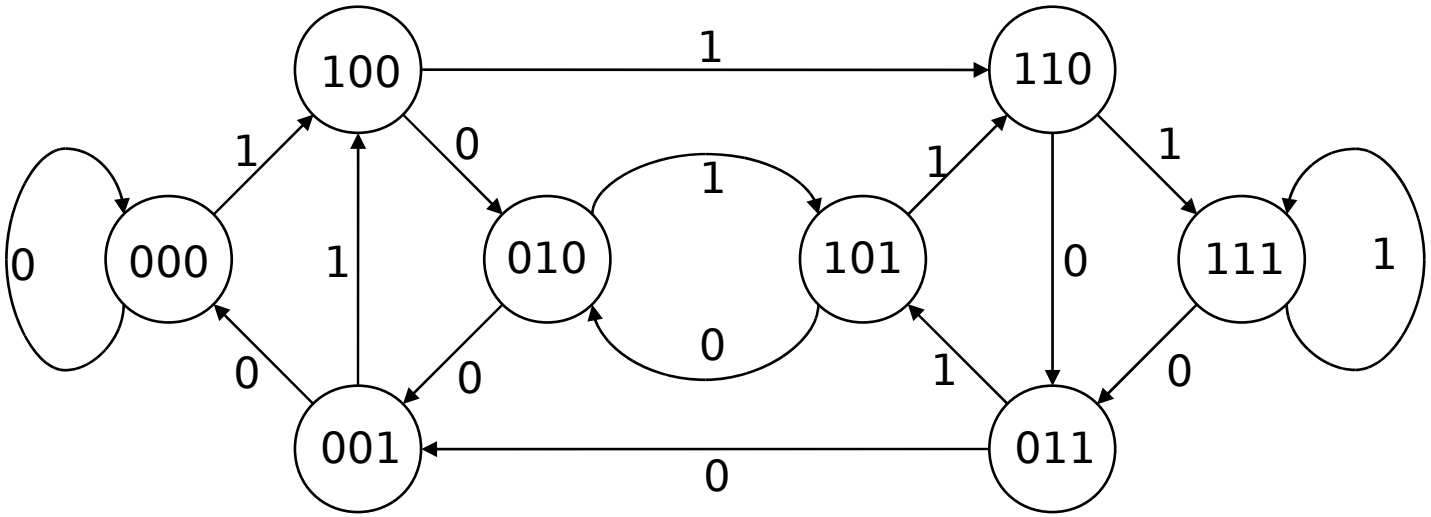
- Ejemplo: Candado de combinación
  - 5 estados
  - 5 auto-transiciones
  - 6 otras transiciones entre estados
  - 1 transición de reset (de todos los estados) al estado S1



Ejemplo: Registro de corrimiento (shift register)

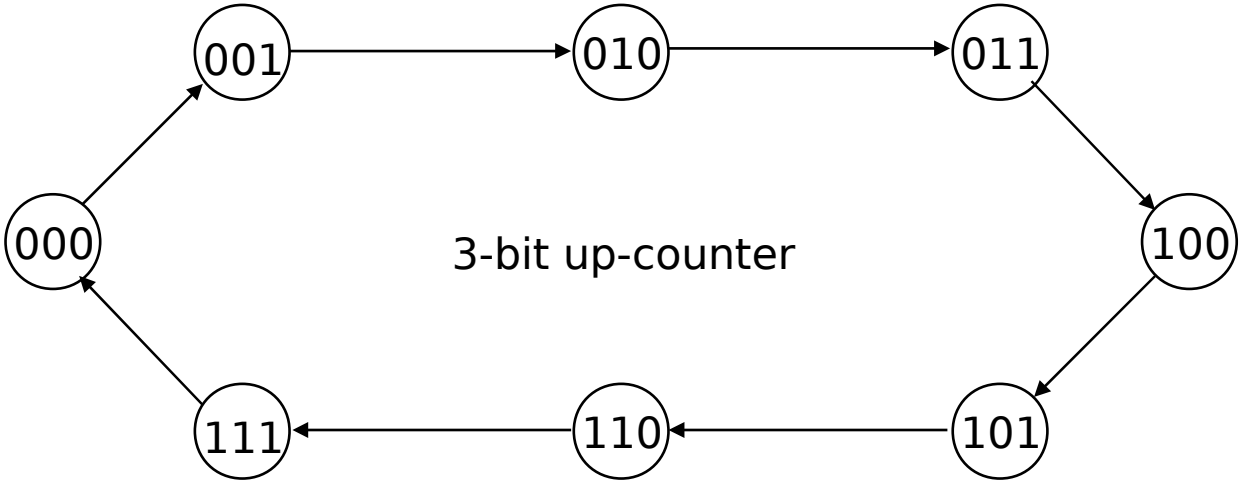
- Shift register
  - input mostrado en arcos de transiciones
  - valores de output mostrado en nodo de estado





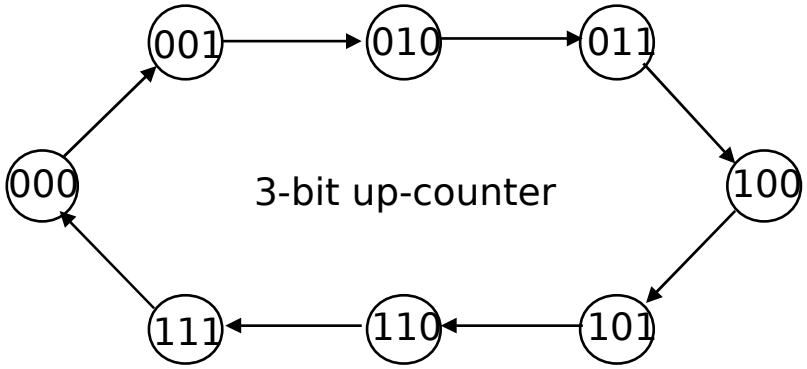
Ejemplo: Contadores

- Contadores
  - proceden a través de secuencia de estados bien definida en respuesta a enable.
- Muchos tipos de contadores: binario, BCD, código Gray
  - contador de subida de 3 bits: 000, 001, 010, 011, 100, 101, 110, 111, 000, ...
  - contador de bajada de 3-bits: 111, 110, 101, 100, 011, 010, 001, 000, 111, ...



Cómo convertir diagrama de estados a tabla de transiciones?

- Tabla de transiciones: forma tabular de diagrama de estados.
- Como una tabla de verdad (se especifican todos los outputs para las combinaciones de input).
- Ejemplo: contador



present state		next state	
0	000	001	1
1	001	010	2
2	010	011	3
3	011	100	4
4	100	101	5
5	101	110	6
6	110	111	7
7	111	000	0

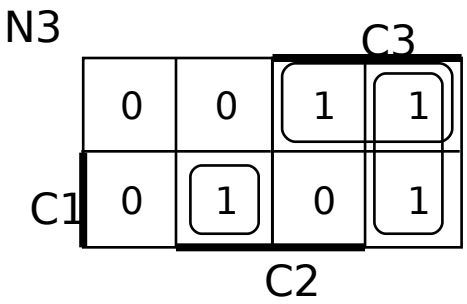
Implementación

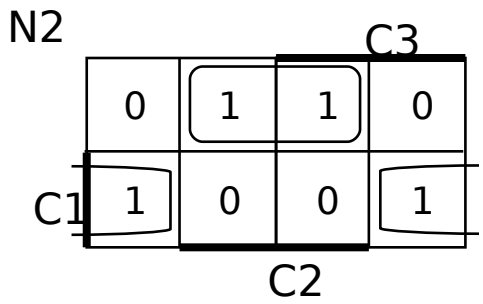
- Un flip-flop para cada bit de estado
- Lógica combinacional basada en codificación

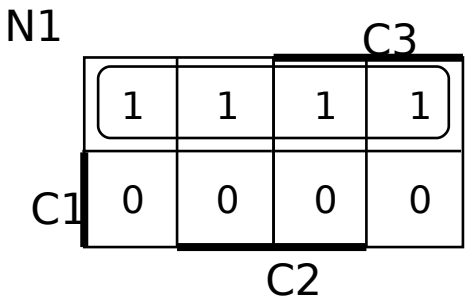
C3	C2	C1	N3	N2	N1
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

código en Verilog para mostrar que la función es un input a un D-FF

N1 <= C1'
N2 <= C1C2' + C1'C2
<= C1 xor C2
N3 <= C1C2C3' + C1'C3 + C2'C3
<= (C1C2)C3' + (C1' + C2')C3
<= (C1C2)C3' + (C1C2)'C3
<= (C1C2) xor C3

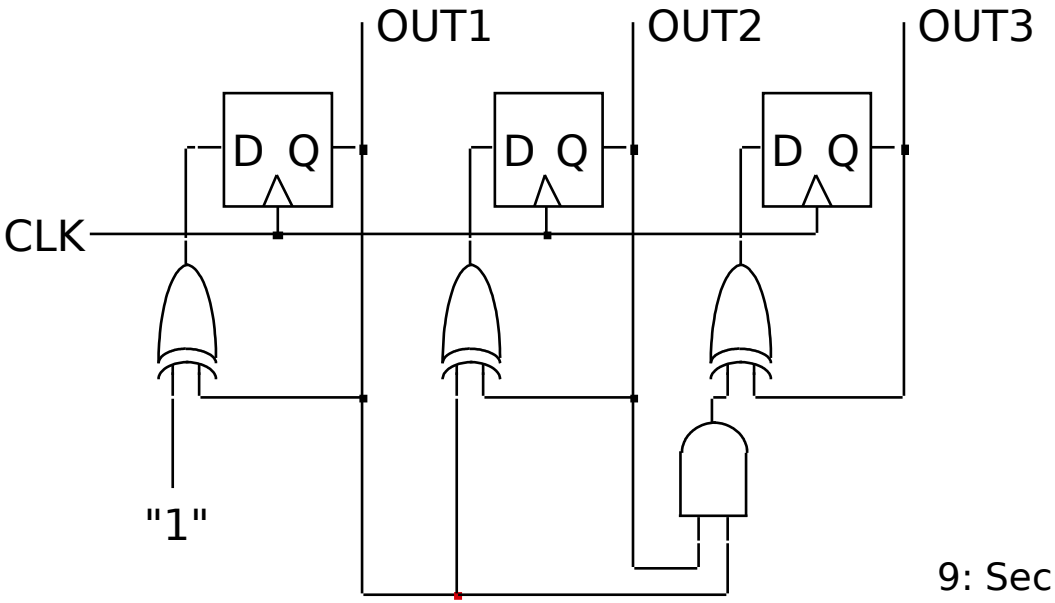






Implementación (cont)

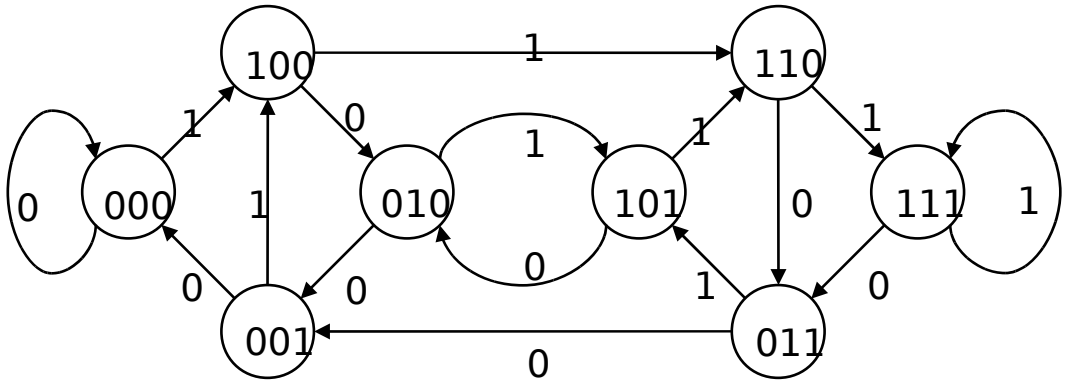
- Contador
  - 3 flip-flops para tener estado.
  - lógica para calcular próximo estado.
  - reloj controla cuando la memoria de los flip-flops cambia.
    - hay que esperar para que la lógica calcule nuevo valor
    - no hay que esperar mucho para no tener velocidad muy lenta



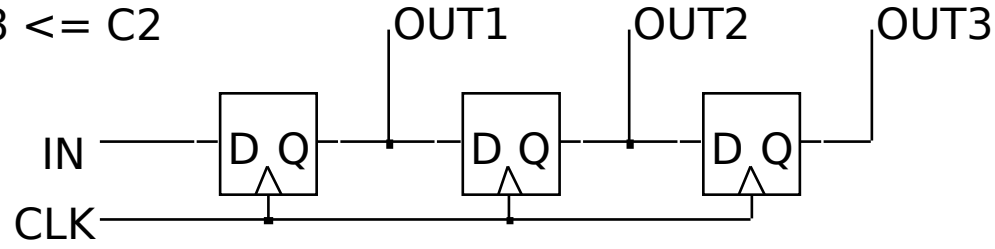
Implementación: Registro de corrimiento

- Input determina próximo estado

In	C1	C2	C3	N1	N2	N3
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	0	1	0
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	1	0	1
1	0	1	1	1	0	1
1	1	0	0	1	1	0
1	1	0	1	1	1	0
1	1	1	0	1	1	1
1	1	1	1	1	1	1

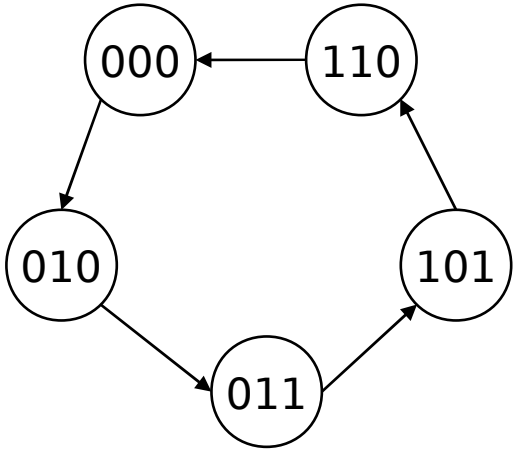


N1 <= In
N2 <= C1
N3 <= C2



Ejemplo: Contador más complejo

- Contador Complejo
  - repite 5 estados en secuencia
  - secuencia no es una representación numérica binaria
- Paso 1: hacer diagrama de transición de estados
  - contar secuencia: 000, 010, 011, 101, 110
- Paso 2: hacer tabla de transición de estados

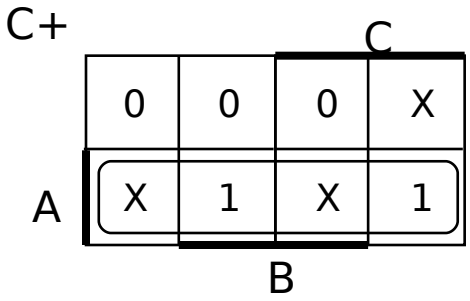


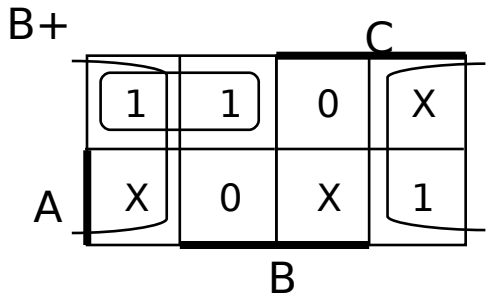
Present State			Next State		
C	B	A	C+	B+	A+
0	0	0	0	1	0
0	0	1	–	–	–
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	–	–	–
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	–	–	–

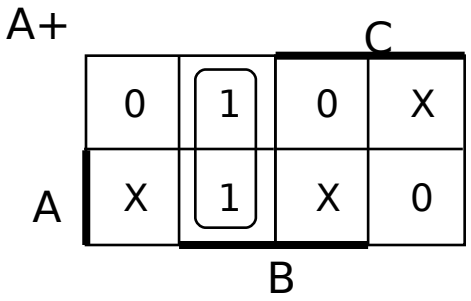
notar condiciones don't care por los estados no usados

Ejemplo: Contador más complejo (cont)

- Paso 3: mapas Karnaugh para próximas funciones







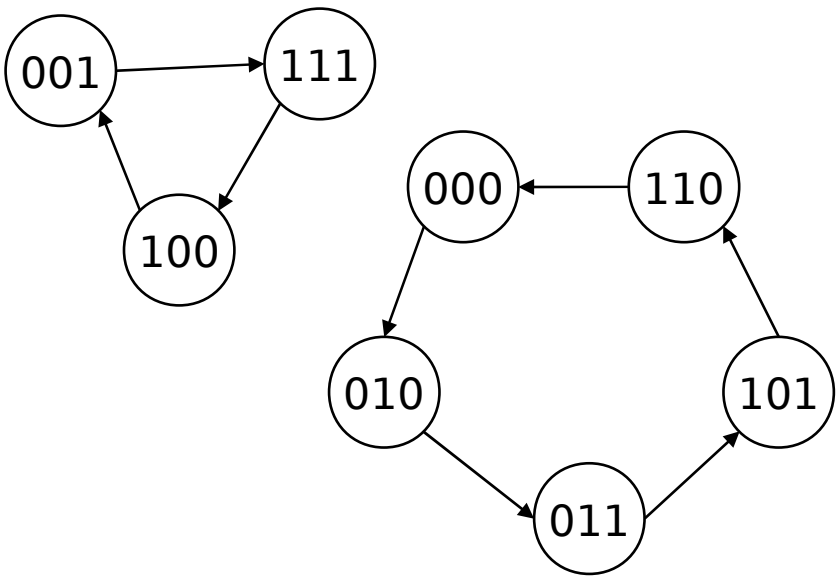
C+ <= A

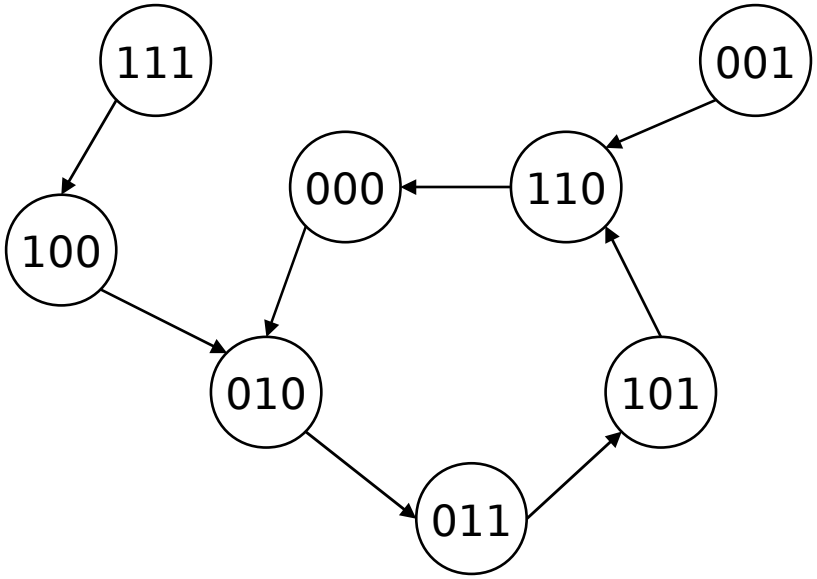
B+ <= B’ + A’C’

A+ <= BC’

Contadores con estados iniciales

- Estados iniciales
  - durante el inicio, el contador puede estar en un estado sin usar o inválido
  - el diseñador debe garantizar que eventualmente entre en un estado válido
  - diseñar para que estados inválidos transiciones a válidos





Contadores con estados iniciales (cont)

- Generar tabla de transición de estados con estados iniciales

C+

A \ B	0	1
0	0	0
1	1	1

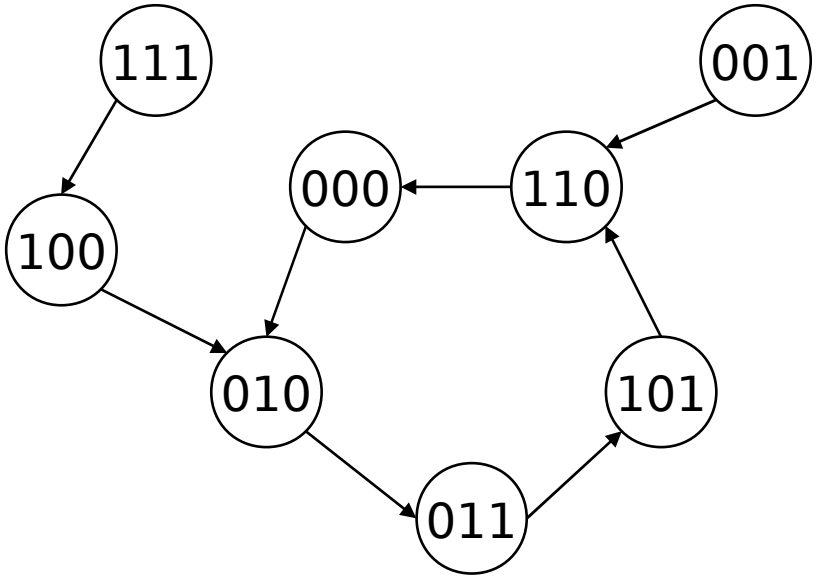
B+

A \ B	0	1
0	1	0
1	0	1

A+

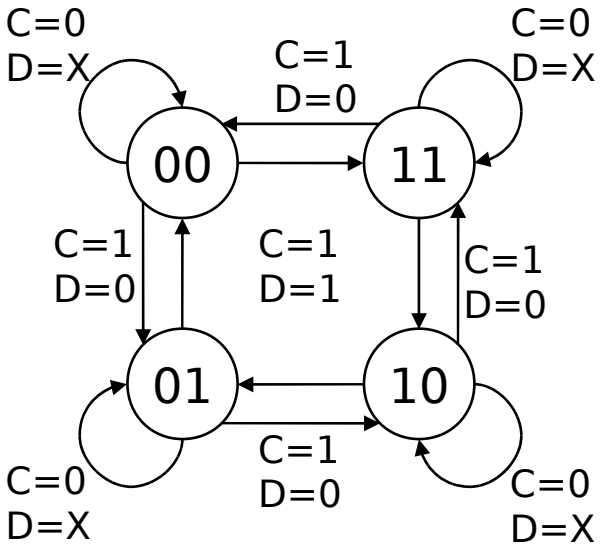
A \ B	0	1
0	0	1
1	0	1

Present State			Next State	
C	B	A	C+	B+	A+
0	0	0	0	1	0
0	0	1	1	1	0
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	1	0	0



Actividad

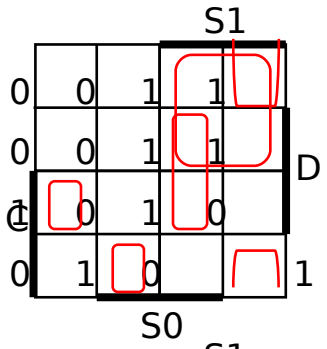
- Contador up-down de 2-bits (2 inputs)
  - dirección: D = 0 para up, D = 1 para down
  - cuenta: C = 0 para parar, C = 1 para contar



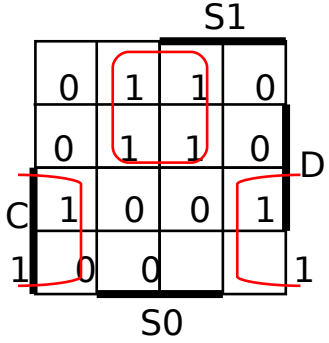
S1	S0	C	D	N1	N0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	1	1	0

Actividad (cont)

S1	S0	C	D	N1	N0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	1	1	0



N1 = C'S1
+ CDS0'S1' + DS0S1
+ CD'S0S1' + D'S0'S1



N0 = CS0' + C'S0

9-Sistemas Secuenciales

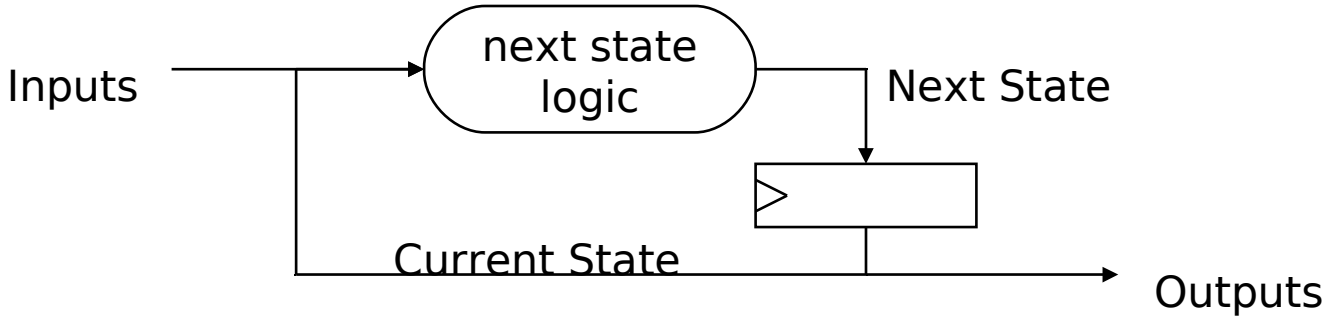
9.1 Máquinas de Estados Finitos

9.2 Mealy y Moore

9.3 Implementación en Verilog

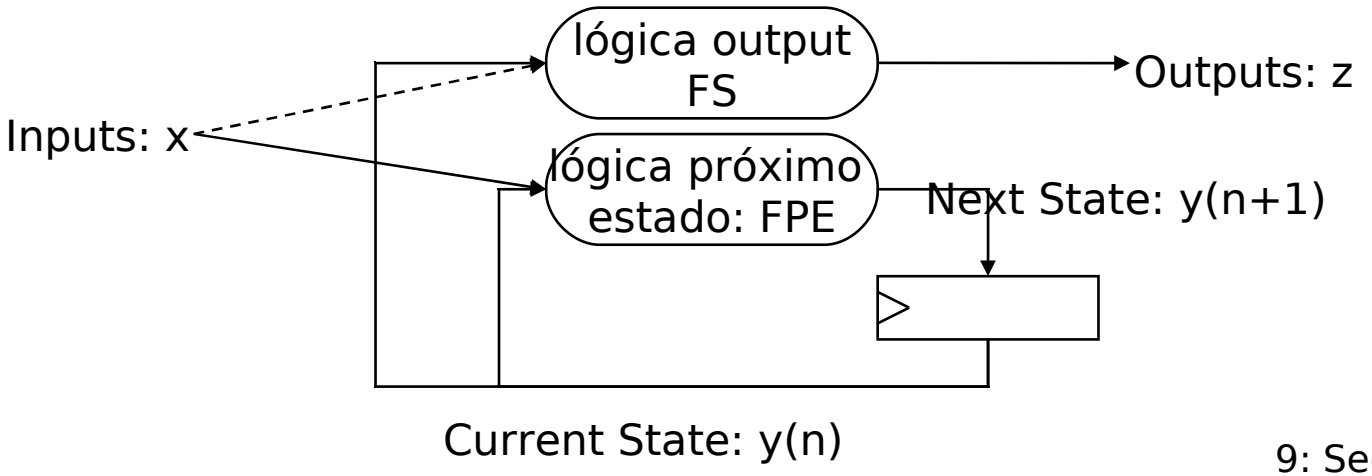
Modelo de Contador/registro de corrimiento

- Valores almacenados en registros representan estado del circuito
- Lógica combinacional calcula:
  - próximo estado
    - función de estados actuales e inputs
  - salidas (o outputs)
    - valores de flip-flops



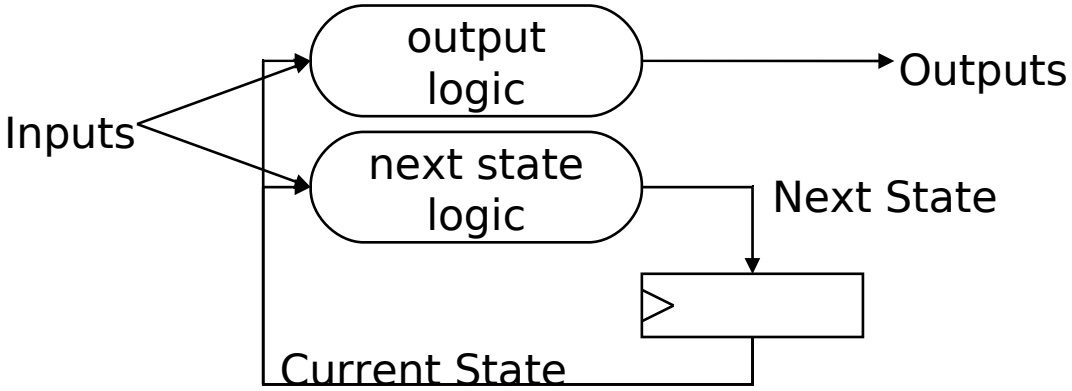
Modelo general

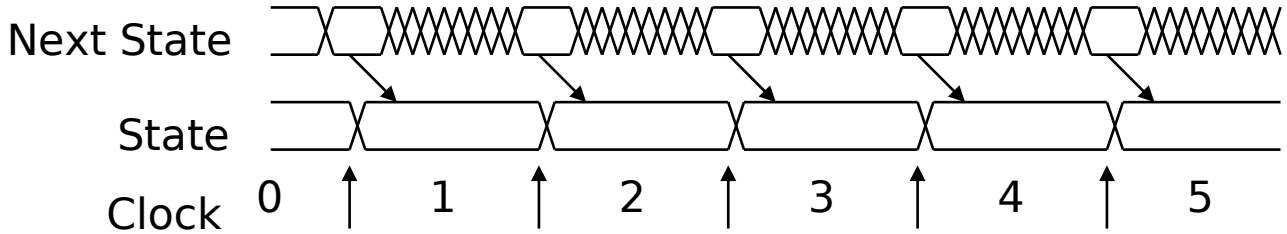
- Valores almacenados en registros representan el estado del circuito: y
- Lógica combinacional calcula
  - próximo estado: FPE (Función Próximo Estado)
    - función de estados actuales e inputs
  - outputs: FS (Función Salida)
    - Mealy: función de estado actual e inputs, z = F(y, x)
    - Moore: solo función de estado actual, z = F(y)



Modelo general (cont)

- Estados: $y_1, y_2, ..., y_k$
- Inputs: $x_1, x_2, ..., x_m$
- Outputs: $z_1, z_2, ..., z_n$
- Función transición: $FPE(y_i, x_j)$
- Función de output: $FS(y_i)$ or $FS(y_i, x_j)$



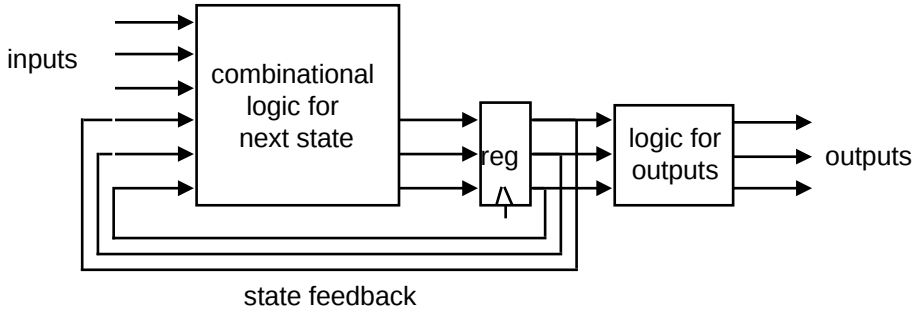


Máquinas Mealy vs Moore

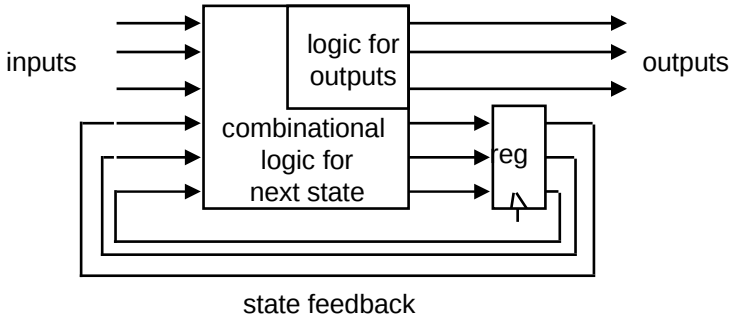
- Máquinas Mealy tienden a tener menos estados
  - outputs son diferentes en arcos (n^2) no en estados (n)
- Máquinas Moore
  - outputs cambian durante cambios del reloj (siempre un ciclo más tarde)
  - en máquinas Mealy, input puede causar cambios en output de inmediato cuando cambie lógica – puede causar problemas cuando se conectan múltiples máquinas
- Máquinas Mealy reaccionan mas rápido
  - reaccionan en el mismo ciclo – no tienen que esperar el reloj en algunos casos
  - en máquinas Moore – mas lógica puede ser necesaria para decodificar estado en outputs

Comparar máquinas Mealy y Moore (cont)

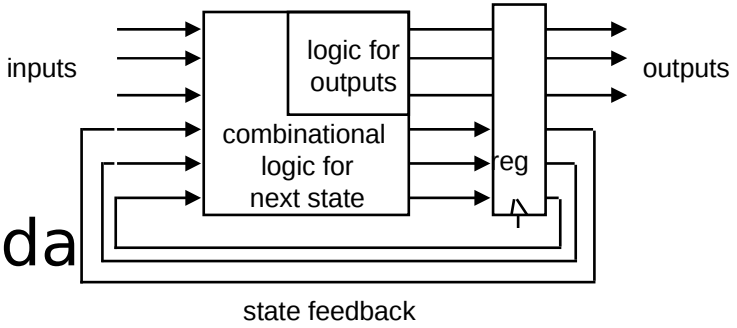
- Moore: z = F(y)



- Mealy: z = F(y, x)

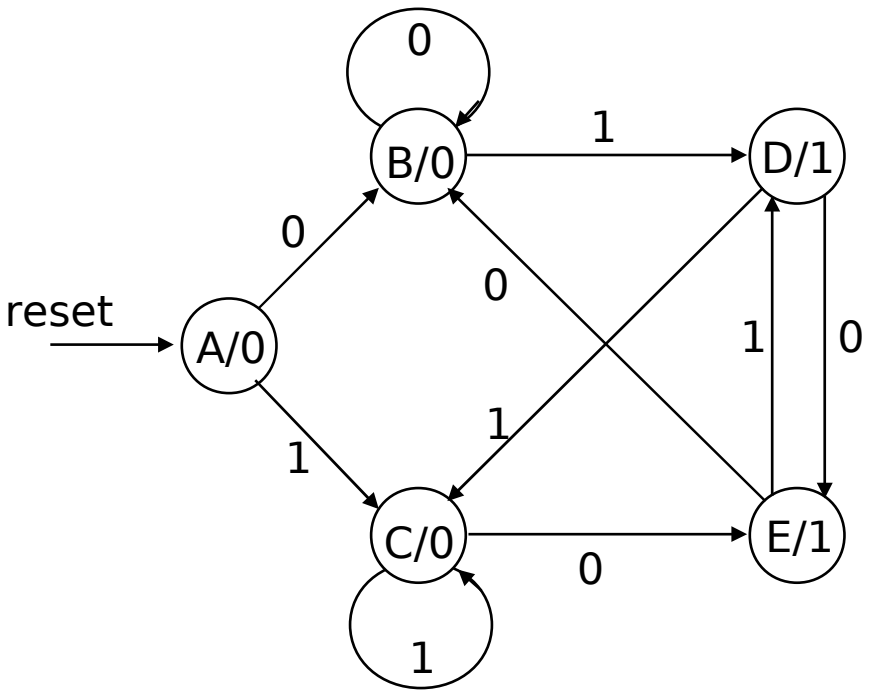


- Mealy Sincrónica
  z = F(y, x) con FF en salida



Especificar outputs para máquina Moore

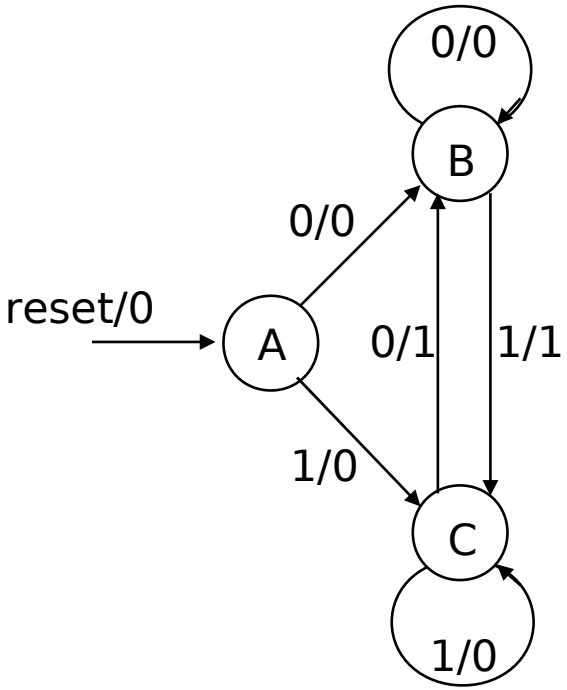
- Output es solo una función del estado
  - se especifica en nodos del diagrama de estado
  - Ejemplo: detector de secuencia para 01 o 10



reset	input	current state	next state	output
1	–	–	A	
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	D	0
0	0	C	E	0
0	1	C	C	0
0	0	D	E	1
0	1	D	C	1
0	0	E	B	1
0	1	E	D	1

Especificar outputs para máquina Mealy

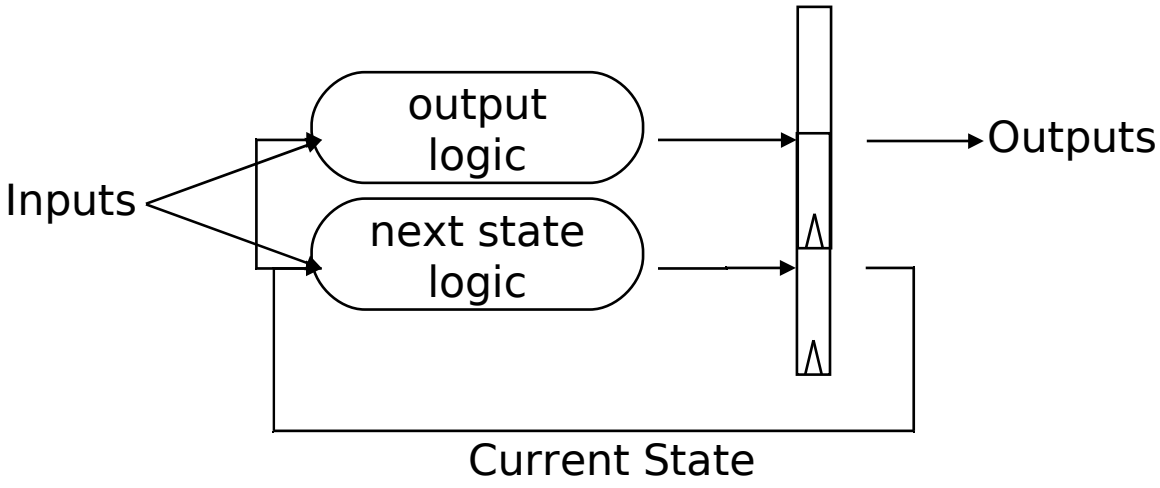
- Output es función de estados e inputs
  - especificar output en transición entre estados
  - Ejemplo: Detector de secuencia para 01 o 10



reset	input	current state	next state	output
1	–	–	A	0
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	C	1
0	0	C	B	1
0	1	C	C	0

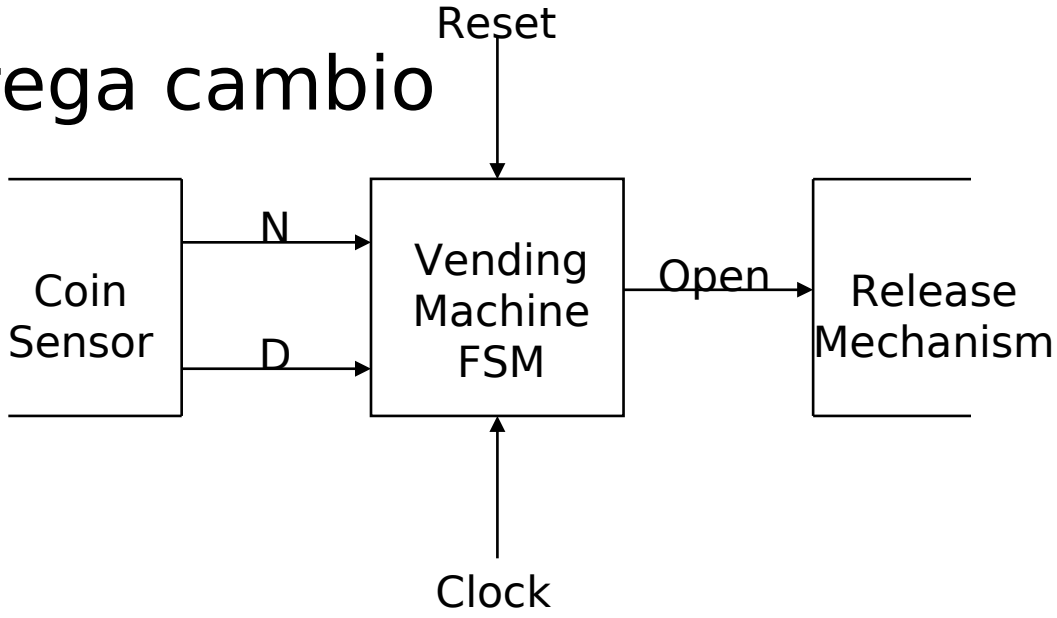
Máquina Mealy

- Máquina Mealy Sincrónica
  - estados y outputs con registros
  - evita outputs con ruidos (‘glitches’)
  - típicamente se implementa en PLDs



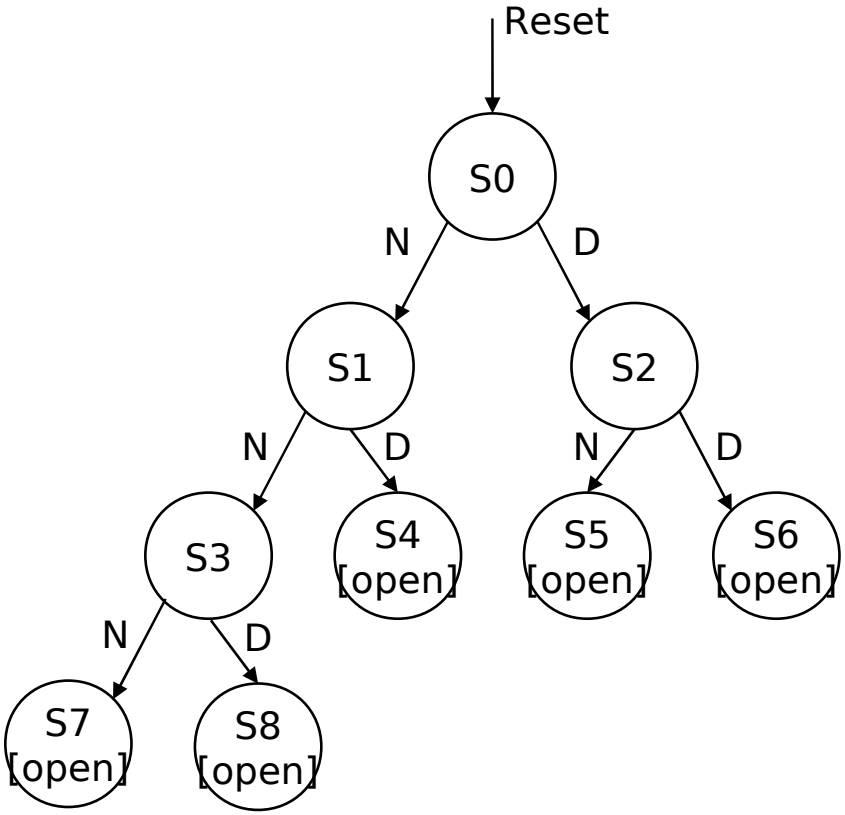
Ejemplo: máquinas de bebidas

- Entrega bebida después que 150 pesos son depositados
- Acepta diferentes monedas 50 (N), 100 (D)
- No entrega cambio



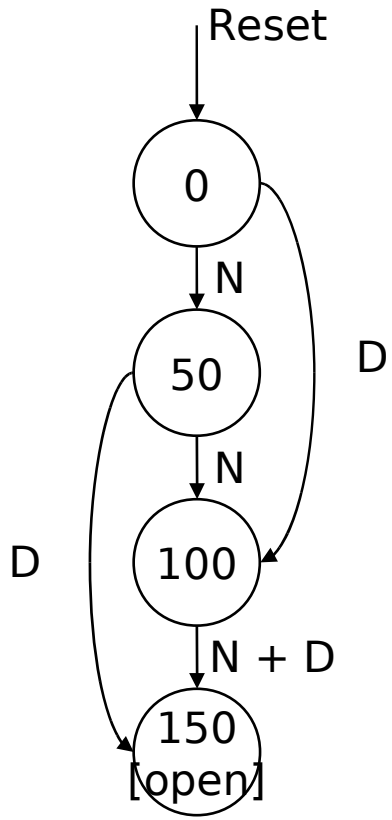
Ejemplo: máquinas de bebidas (cont)

- Representación abstracta
  - listar secuencias típicas:
    - tres de cincuenta
    - cincuenta, cien
    - cien, cincuenta
    - dos de cien
  - dibujar diagrama de estados:
    - inputs: N, D, reset
    - output: dar bebida (OPEN)
  - asumir:
    - N y D seteadas por un ciclo
    - cada estado tiene un auto estado para N = D = 0 (no hay moneda)



Ejemplo: máquinas de bebidas (cont)

- Minimizar número de estados – reusar estados si es posible



present state	inputs D	N	next state	output open
0	0	0	0	0
	0	1	50	0
	1	0	100	0
	1	1	–	–
50	0	0	50	0
	0	1	100	0
	1	0	150	0
	1	1	–	–
100	0	0	100	0
	0	1	150	0
	1	0	150	0
	1	1	–	–
150	–	–	150	1

tabla de estados simbólica

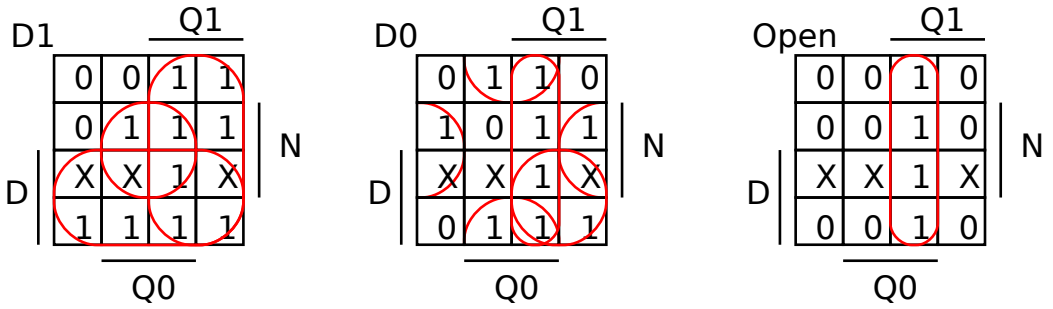
Ejemplo: máquinas de bebidas (cont)

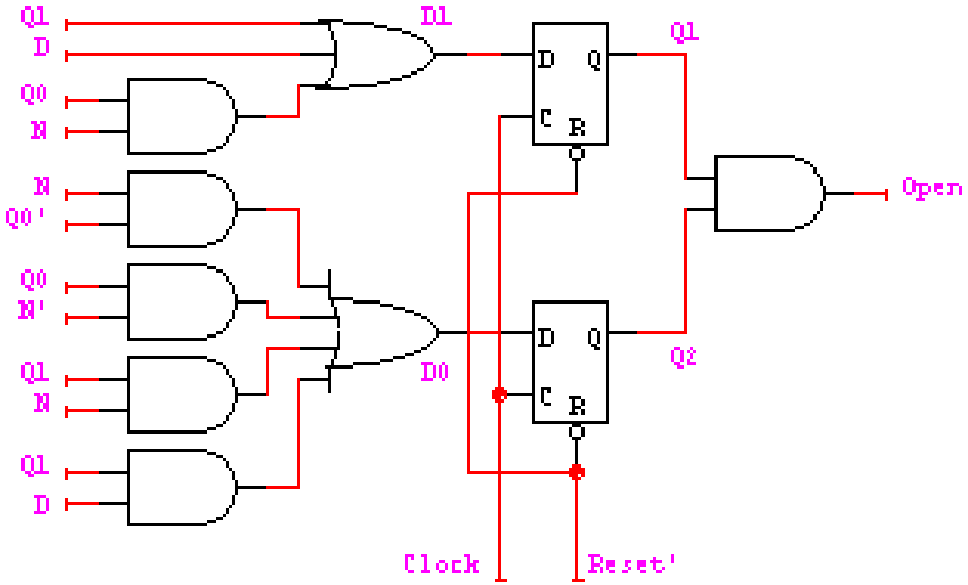
- Codificar estados

present state		inputs		next state		output
Q1	Q0	D	N	D1	D0	open
0	0	0	0	0	0	0
		0	1	0	1	0
		1	0	1	0	0
		1	1	–	–	–
0	1	0	0	0	1	0
		0	1	1	0	0
		1	0	1	1	0
		1	1	–	–	–
1	0	0	0	1	0	0
		0	1	1	1	0
		1	0	1	1	0
		1	1	–	–	–
1	1	–	–	1	1	1

Ejemplo: Implementación Moore

- Mapear la lógica





D1 = Q1 + D + Q0 N

D0 = Q0’ N + Q0 N’ + Q1 N + Q1 D

OPEN = Q1 Q0

Ejemplo: Implementación Moore (cont)

- Otra codificación (1 encendido)

present state Q3	Q2	Q1	Q0	inputs D	N	next state D3	D2	D1	D0	output open
0	0	0	1	0	0	0	0	0	1	0
				0	1	0	0	1	0	0
				1	0	0	1	0	0	0
				1	1	-	-	-	-	-
0	0	1	0	0	0	0	0	1	0	0
				0	1	0	1	0	0	0
				1	0	1	0	0	0	0
				1	1	-	-	-	-	-
0	1	0	0	0	0	0	1	0	0	0
				0	1	1	0	0	0	0
				1	0	1	0	0	0	0
				1	1	-	-	-	-	-
1	0	0	0	-	-	1	0	0	0	1

D0 = Q0 D’ N’

D1 = Q0 N + Q1 D’ N’

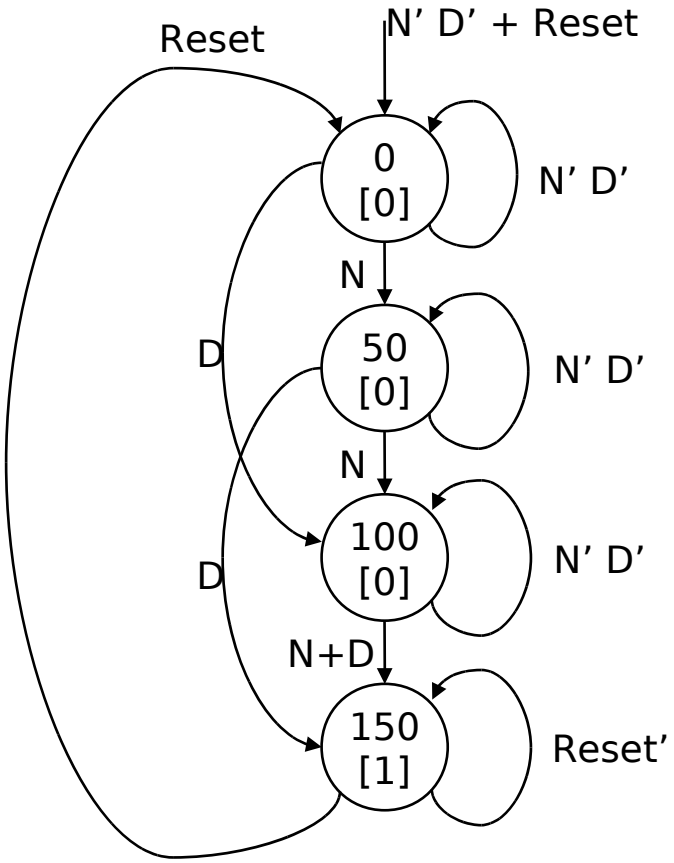
D2 = Q0 D + Q1 N + Q2 D’ N’

D3 = Q1 D + Q2 D + Q2 N + Q3

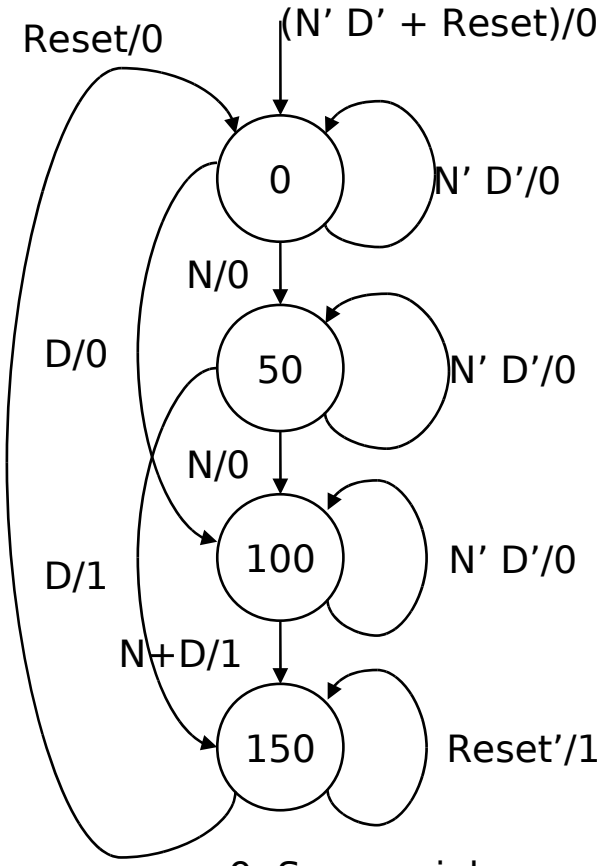
OPEN = Q3

Diagramas de Estados de Mealy y Moore

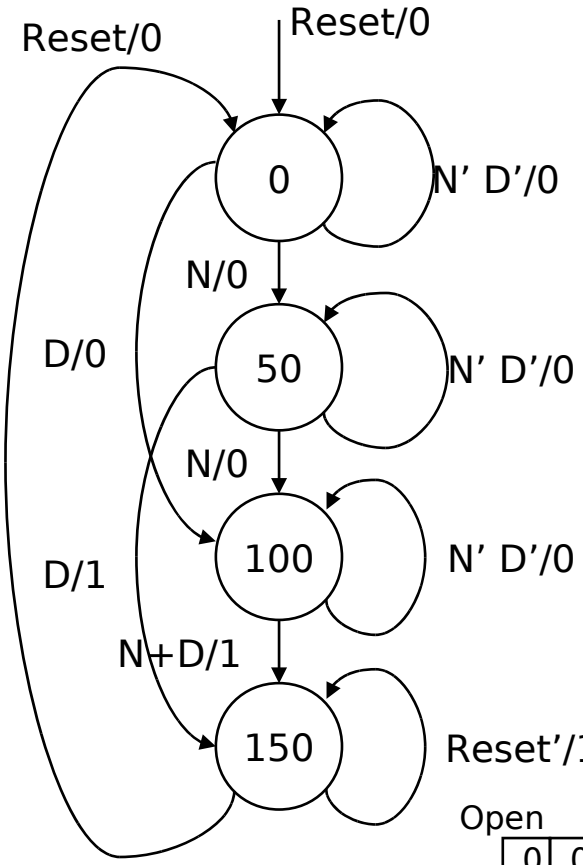
- Moore
  - outputs asociados con estados



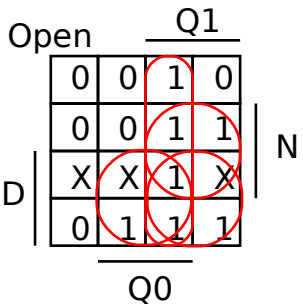
- Mealy
  - outputs asociados con transiciones



Ejemplo: Implementación Mealy



present state		inputs		next state		output
Q1	Q0	D	N	D1	D0	open
0	0	0	0	0	0	0
		0	1	0	1	0
		1	0	1	0	0
		1	1	–	–	–
0	1	0	0	0	1	0
		0	1	1	0	0
		1	0	1	1	1
		1	1	–	–	–
1	0	0	0	1	0	0
		0	1	1	1	1
		1	0	1	1	1
		1	1	–	–	–
1	1	–	–	1	1	1



D0 = Q0'N + Q0N' + Q1N + Q1D

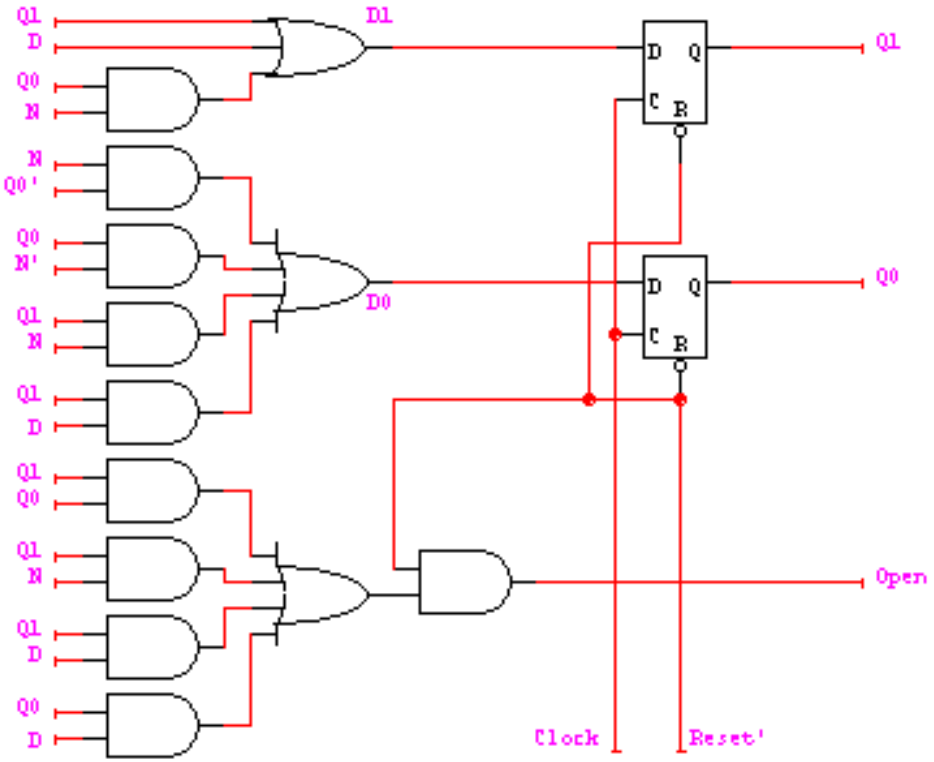
D1 = Q1 + D + Q0N

OPEN = Q1Q0 + Q1N + Q1D + Q0D

Ejemplo: Implementación Mealy (cont)

D0 = Q0’N + Q0N’ + Q1N + Q1D
D1 = Q1 + D + Q0N
OPEN = Q1Q0 + Q1N + Q1D + Q0

hay que asegurar que OPEN es 0 cuando hay reset – con compuerta



9-Sistemas Secuenciales

9.1 Maquinas de Estados Finitos

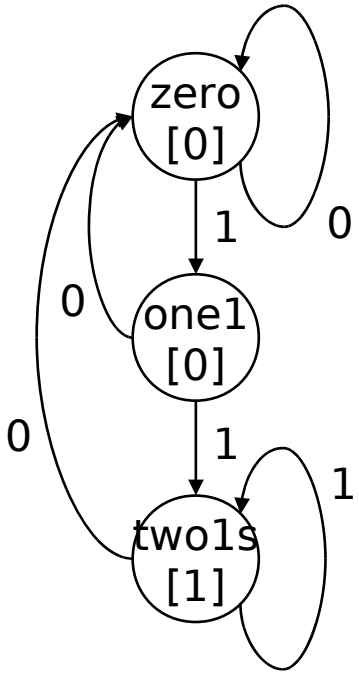
9.2 Mealy y Moore FSMs

9.3 Implementación en Verilog

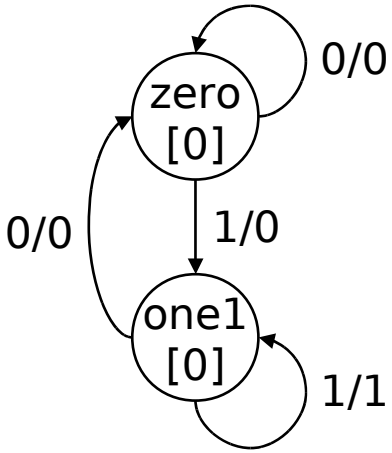
Ejemplo: reducir string de 1s en 1

- Eliminar un 1 de cada string de 1s en el input

Moore



Mealy

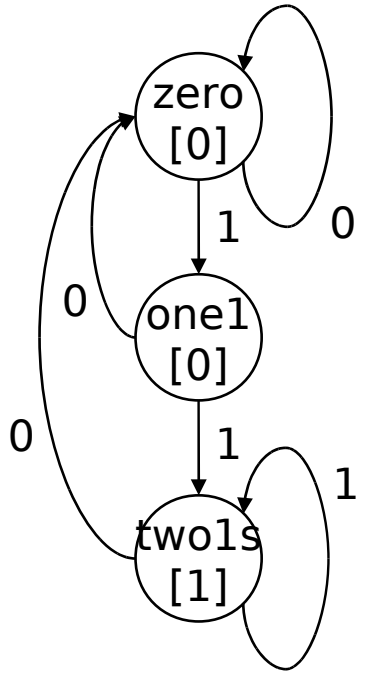


Ejemplo: reducir string de 1s en 1

- Verilog: Máquina de Moore

asignar estados

```
module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;

  parameter zero  = 2'b00;
  parameter one1  = 2'b01;
  parameter two1s = 2'b10;


  reg out;
  reg [2:1] state;       // state variables
  reg [2:1] next_state;

  always @(posedge clk)
    if (reset) state = zero;
    else       state = next_state;
```



Ejemplo: reducir string de 1s en 1 (cont)

```
always @(in or state)

  case (state)
    zero:
   // last input was a zero
    begin
      if (in) next_state = one1;
      else    next_state = zero;
    end
     one1:
   // we've seen one 1
    begin
      if (in) next_state = two1s;
      else    next_state = zero;
    end
     two1s:
   // we've seen at least 2 ones
    begin
      if (in) next_state = two1s;
      else    next_state = zero;
    end
  endcase
```

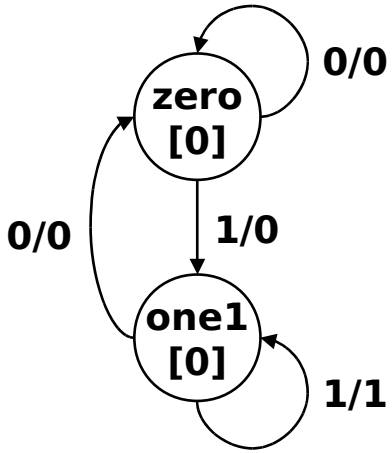
hay que incluir todas las señales que son determinan el estado

el output solo depende del estado

```
always @(state)
  case (state)
    zero: out = 0;
    one1: out = 0;
   two1s: out = 1;
  endcase

endmodule
```

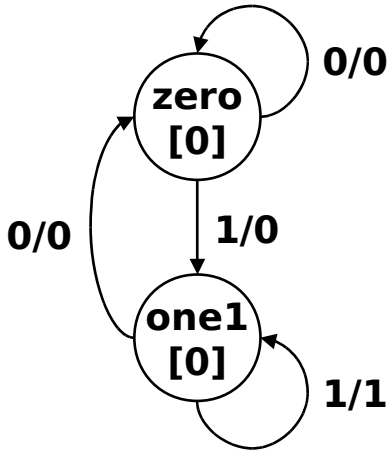
Verilog para MEF Mealy

```
module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;
  reg out;
  reg state;  // state variables
  reg next_state;

  always @(posedge clk)
    if (reset) state = zero;
    else       state = next_state;

  always @(in or state)
    case (state)
      zero:               // last input was a zero
      begin
        out = 0;
        if (in) next_state = one;
        else    next_state = zero;
      end
      one:                // we've seen one 1
      if (in)
      begin
          next_state = one; out = 1;
      end
      else
      begin
          next_state = zero; out = 0;
      end
    endcase
endmodule
```



Verilog para MEF Mealy (otra versión)

```
module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;
  reg out;
  reg state; // state variables

  always @(posedge clk)
    if (reset) state = zero;
    else
     case (state)
      zero:      // last input was a zero
     begin
       out = 0;
       if (in) state = one;
       else    state = zero;
     end
      one:       // we've seen one 1
     if (in)
     begin
        state = one; out = 1;
     end
     else
     begin
        state = zero; out = 0;
     end
    endcase
endmodule
```



Resumen MEFs

- Modelos para representar circuitos secuenciales
  - abstracción de elementos secuenciales
  - máquinas de estados finitos y diagramas de estados
  - Mealy, Moore y maquinas sincrónicas Mealy
- Procedimiento de diseño usando MEFs
  - generar diagrama de estados
  - generar tabla de transiciones de estados
  - determinar funciones de próximo estado y output
  - implementar lógica combinacional
- HDLs