

8. Circuitos secuenciales síncronos

angel@uc.cl

IEE2713 Sistemas Digitales

Motivación

- En el capítulo anterior aprendimos acerca de los elementos capaces de almacenar bits
  - Latch y flip flop (de diferente tipo)
- También aprendimos algunos circuitos básicos secuenciales
  - Circuitos con memoria
  - La salida depende de la entrada y del “estado” del circuito
    - Ejemplos: contador, temporizador, SR, etc.
- En este capítulo aprenderemos cómo diseñar circuitos secuenciales de complejidad arbitraria
  - También se llaman **máquinas de estados finitos** o **FSM**



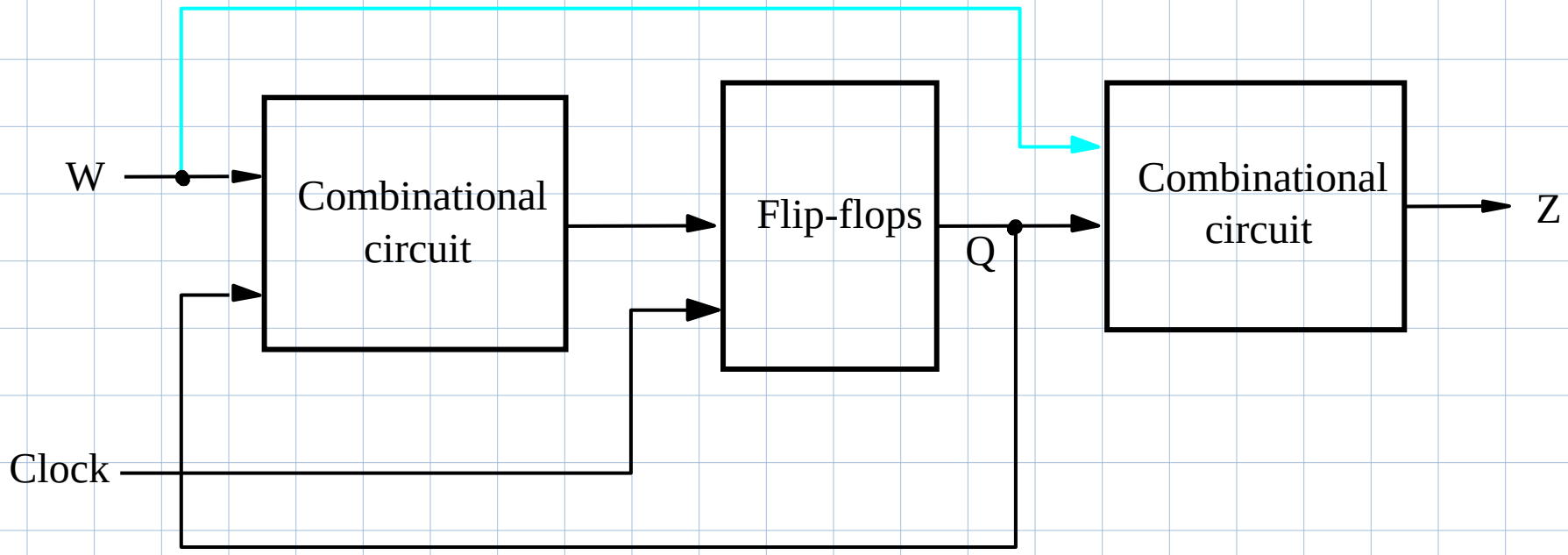
8.1 – 8.3

Introducción

angel@uc.cl

IEE2713 Sistemas Digitales

Esquema general de un circuito secuencial



8.1 Pasos básicos de diseño

(No espero que los entiendan a cabalidad ahora, sólo espero que comiencen a familiarizarse con ellos)

1. Determine las especificaciones deseadas del circuito
2. Determine los estados de la máquina y elabore un diagrama de estados
3. Elabore una tabla de estados a partir del diagrama de estados
4. Revise y minimice el número de estados según procedimiento de minimización que aprenderemos en 8.6
5. Determine el número de variables de estado necesarias para representar todos los estados, y asigne los estados
6. Elija el tipo de flip-flops que utilizará y determine las expresiones combinacionales del próximo estado y de las salidas
7. Implemente el circuito según las expresiones lógicas del punto anterior

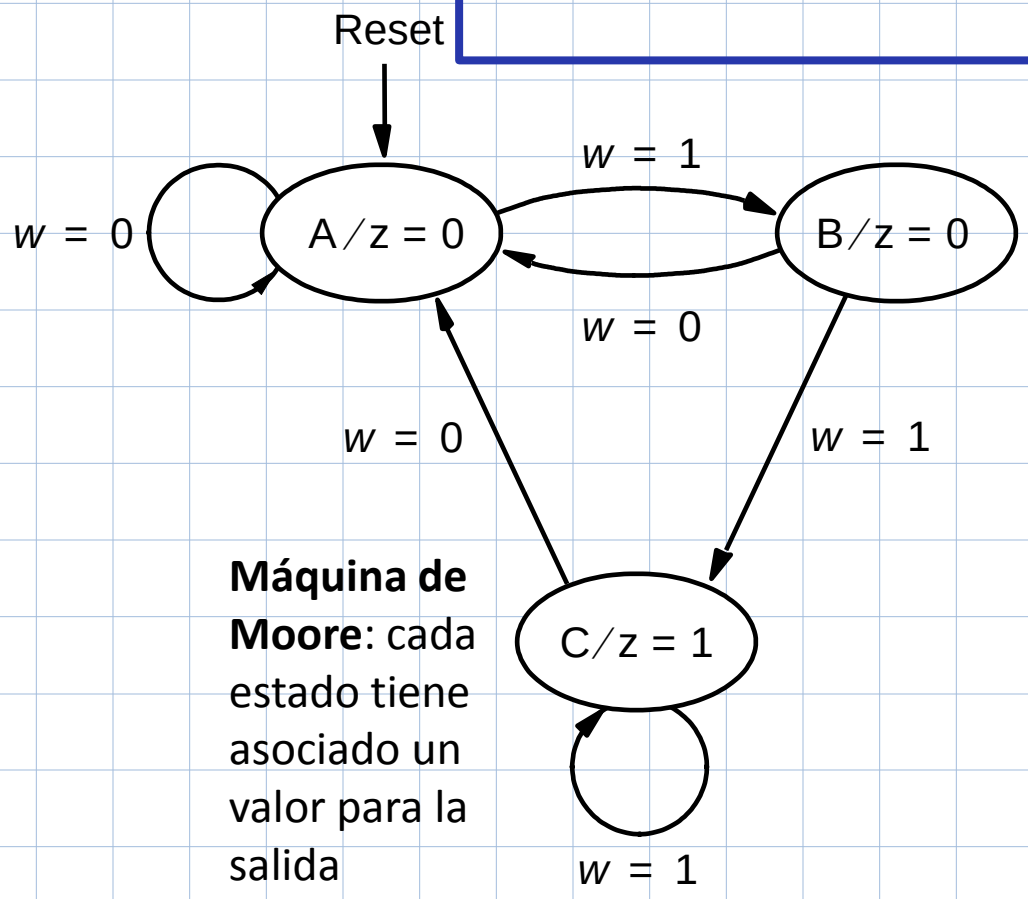
Veremos estos pasos con un ejemplo

- Especificaciones:
  - El circuito tiene una entrada w y una salida z
  - Los cambios ocurren en el flanco positivo del reloj
  - La salida z es 1 si durante dos ciclos precedentes de reloj la entrada w es 1; en cualquier otro caso, la salida z es 0
- A partir de las especificaciones, sabemos que:
  - z no depende sólo de w, sino también del estado anterior de w
    - El circuito es secuencial, tiene estados, tiene memoria
  - Podemos comprobarlo con la secuencia de entradas y salidas a continuación

Clockcycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0

8.1.1 Diagrama de estados

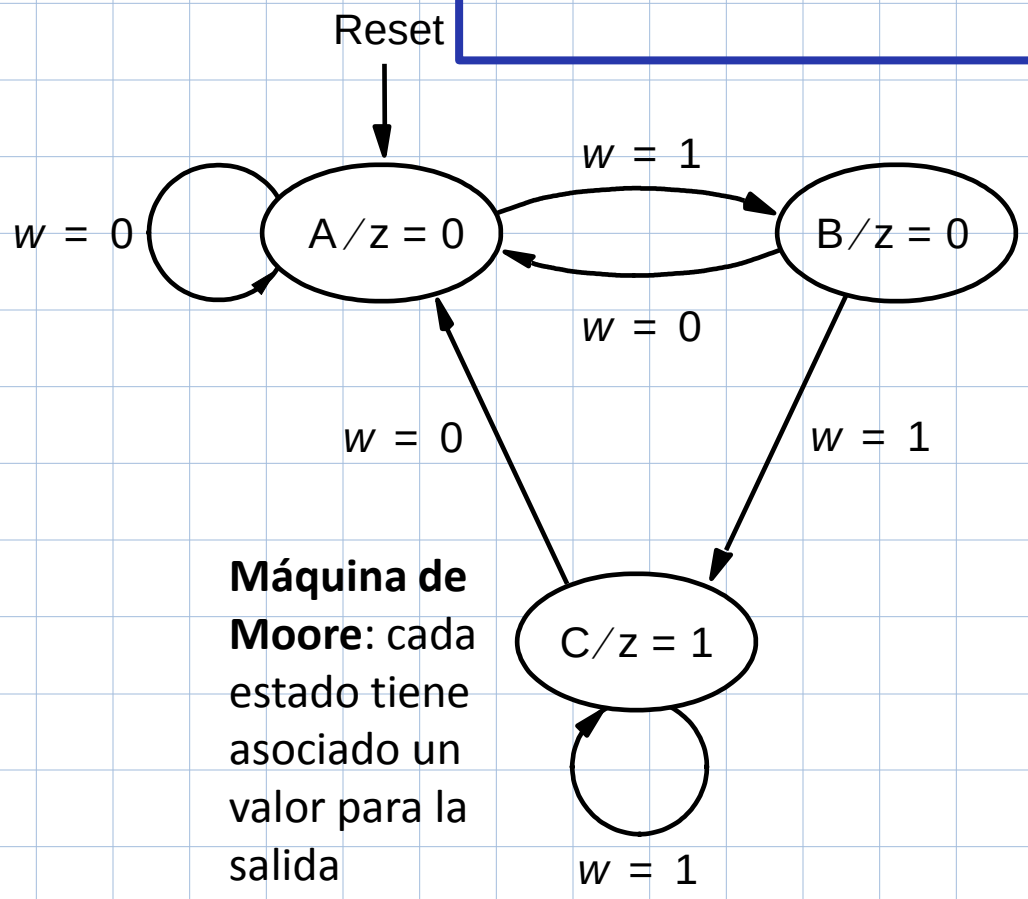
- Respondamos dos preguntas básicas:
  - ¿Cuántos estados tiene la FSM?
  - ¿Qué transiciones entre estados son posibles?
- No hay un procedimiento único, pero es más fácil si:
  - Elegimos un estado inicial
    - Es el estado al encender o resetear el circuito
    - En nuestro caso, le llamamos estado A
  - Tenemos otro estado, B, que indica que la entrada ya fue 1 una vez; z=0
  - Tenemos un tercer estado que indica que la entrada fue 1 dos veces seguidas; z=1



Máquina de Moore: cada estado tiene asociado un valor para la salida

8.1.1 Diagrama de estados

- El diagrama de estados tiene:
  - Nodos (círculos)
  - Transiciones entre estados (arcos)
- Cada nodo corresponde a un estado
  - En nuestro caso, A, B y C
- En el nodo indicamos el nombre del estado y la salida correspondiente
- En los arcos indicamos las entradas necesarias para movernos entre estados



Máquina de Moore: cada estado tiene asociado un valor para la salida

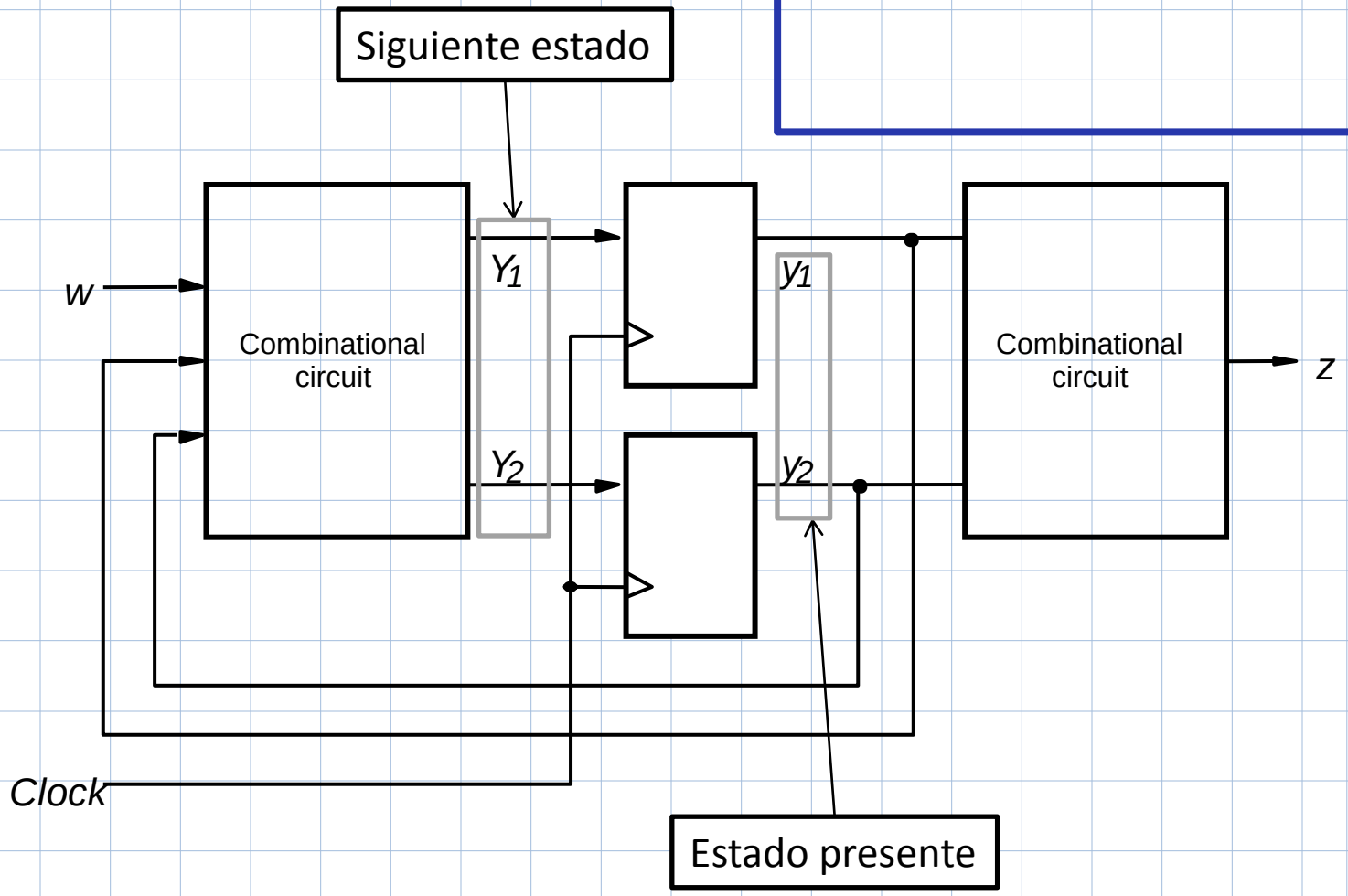
8.1.2 Tabla de estados

- El diagrama de estados describe el comportamiento de un circuito secuencial
- Para implementar el circuito es conveniente tabular los estados
- La tabla indica todas las transiciones desde cada *estado presente* hasta el *siguiente estado*, para diferentes entradas
  - La salida z es especificada para cada estado presente o actual
- Reset define el estado inicial

Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	A	B	0
B	A	C	0
C	A	C	1

8.1.3 Asignación de estados

- En la tabla de estados hay tres estados: A, B y C
- Cada estado está representado por una combinación de valores de las variables de estado
  - En este caso, y1 e y2
  - Las variables de estado son almacenadas en flip-flops
  - Dos flip-flops nos alcanzan para representar hasta 4 estados distintos
- El **siguiente estado** es función combinacional del **estado presente** y de la entrada w
- En este circuito, la salida z depende directamente del estado presente y puede ser generado por un circuito combinacional

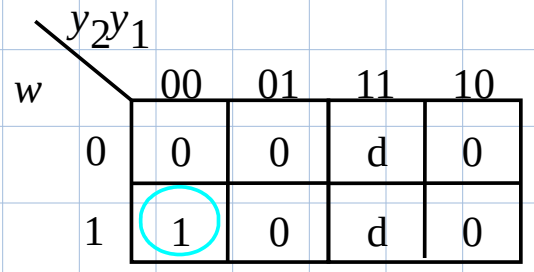


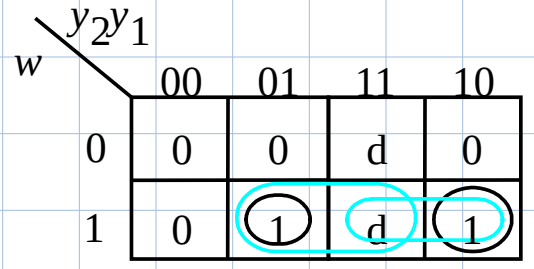
8.1.3 Asignación de estados

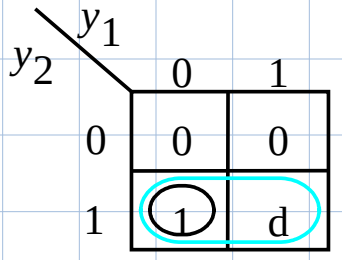
- Aquí tenemos una tabla de verdad que asigna:
  - el siguiente estado en función del estado presente y la entrada
  - La salida en función del estado presente
- Esta tabla de verdad da lugar a los circuitos **combinacionales** de la lámina anterior

	Present state	Next state		Output
		$w = 0$	$w = 1$	
	y_2y_1	Y_2Y_1	Y_2Y_1	z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

8.1.4 Elección de flip-flops y derivación del siguiente estado y la salida







Ignoring don't cares

$$Y_1 = w\bar{y}_1\bar{y}_2$$

$$Y_2 = wy_1\bar{y}_2 + w\bar{y}_1y_2$$

$$z = y_1y_2$$

Using don't cares

$$Y_1 = w\bar{y}_1\bar{y}_2$$

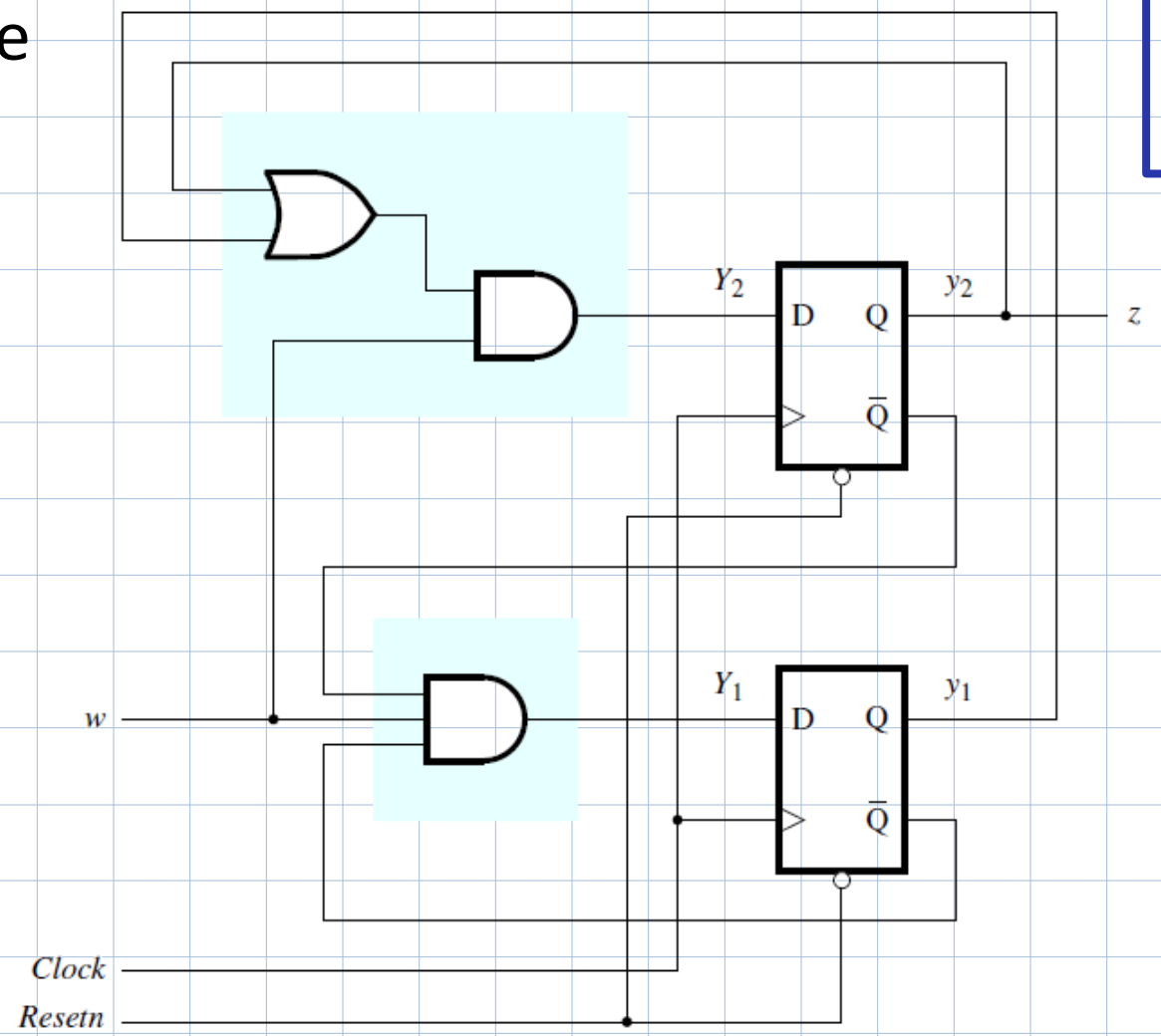
$$Y_2 = wy_1 + wy_2 = w(y_1 + y_2)$$

$$z = y_2$$

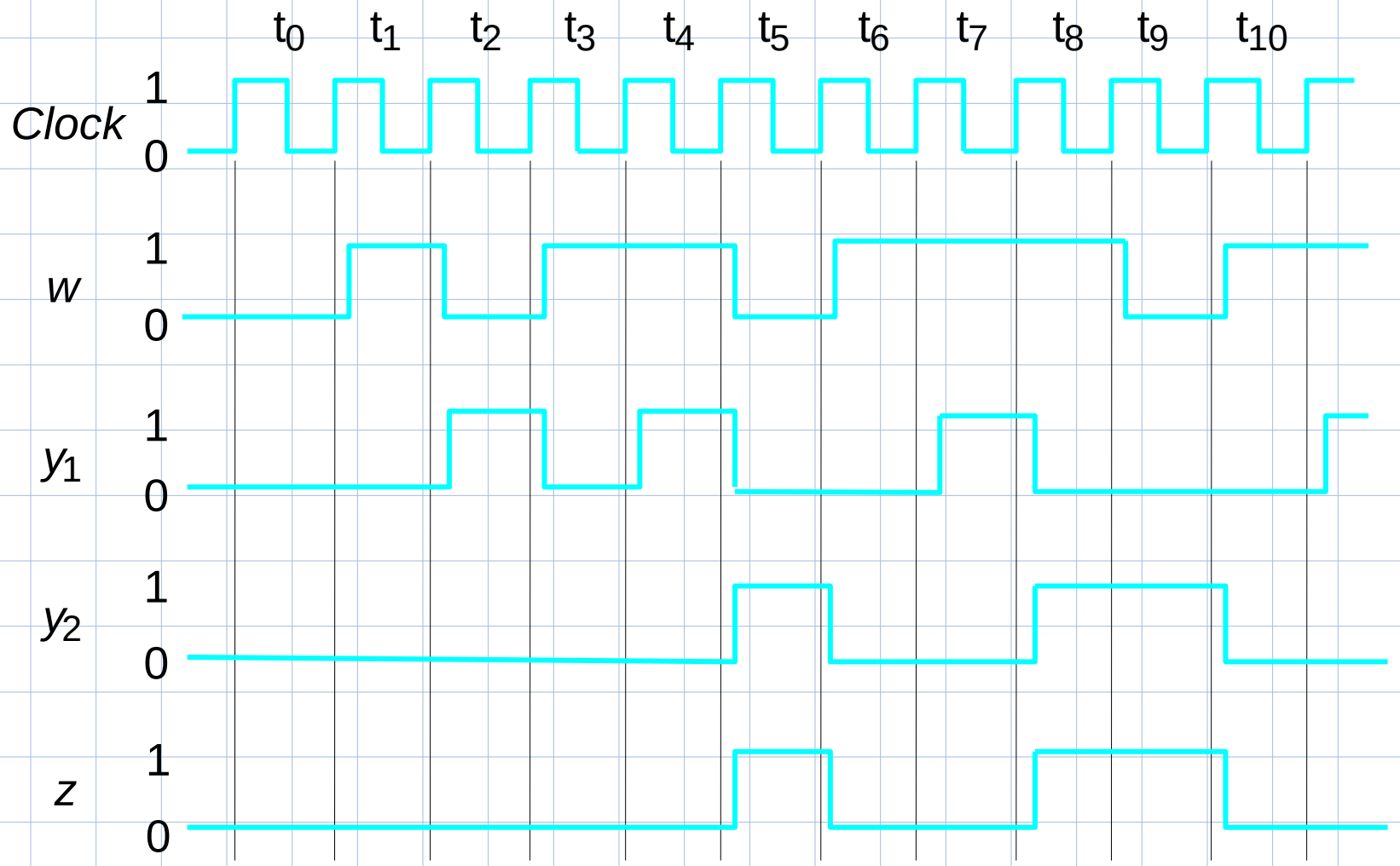
- Elegimos flip-flops D
  - Asignamos estados siguientes en forma directa
  - Usar otro tipo de flip-flops es menos intuitivo
- Usamos mapas de Karnaugh para determinar expresiones para lógica combinacional

8.1.4 Elección de flip-flops y derivación del siguiente estado y la salida

Circuito resultante



8.1.5 Diagrama de tiempo



- Notar operación de flip flops gatillados por flanco de subida
- Hay un pequeño retardo en w
  - Debido a lógica combinacional
  - Es bueno porque permite respetar tiempo de hold del flip-flop

8.1.6 Pasos de diseño - síntesis

(Ahora debieran entender bastante mejor de qué se trata)

1. Determine las especificaciones deseadas del circuito
2. Determine los estados de la máquina y elabore un diagrama de estados
3. Elabore una tabla de estados a partir del diagrama de estados
4. Revise y minimice el número de estados según procedimiento de minimización que aprenderemos en 8.6
5. Determine el número de variables de estado necesarias para representar todos los estados, y asigne los estados
6. Elija el tipo de flip-flops que utilizará y determine las expresiones combinacionales del próximo estado y de las salidas
7. Implemente el circuito según las expresiones lógicas del punto anterior

8.2 Problema de asignación de estados

- Existen diferentes formas de asignar los estados a una FSM
  - Unos mejores que otros
- Ejemplo: la asignación de estados de 5 láminas atrás parece razonable
  - El circuito resultante, 3 láminas atrás, parece sencillo y cumple
  - ¿Habrá una asignación de estados que resulte en un circuito más simple?
- Repitamos el ejemplo anterior con otra asignación de estados

	Present state	Next state		Output
		$w = 0$	$w = 1$	
	y_2y_1	Y_2Y_1	Y_2Y_1	z
A	00	00	01	0
B	01	00	11	0
C	11	00	11	1
	10	*dd*	*dd*	*d*

8.2 Problema de asignación de estados

- En este caso, si volvemos a utilizar flip-flops D, las expresiones para el siguiente estado y para la salida son:

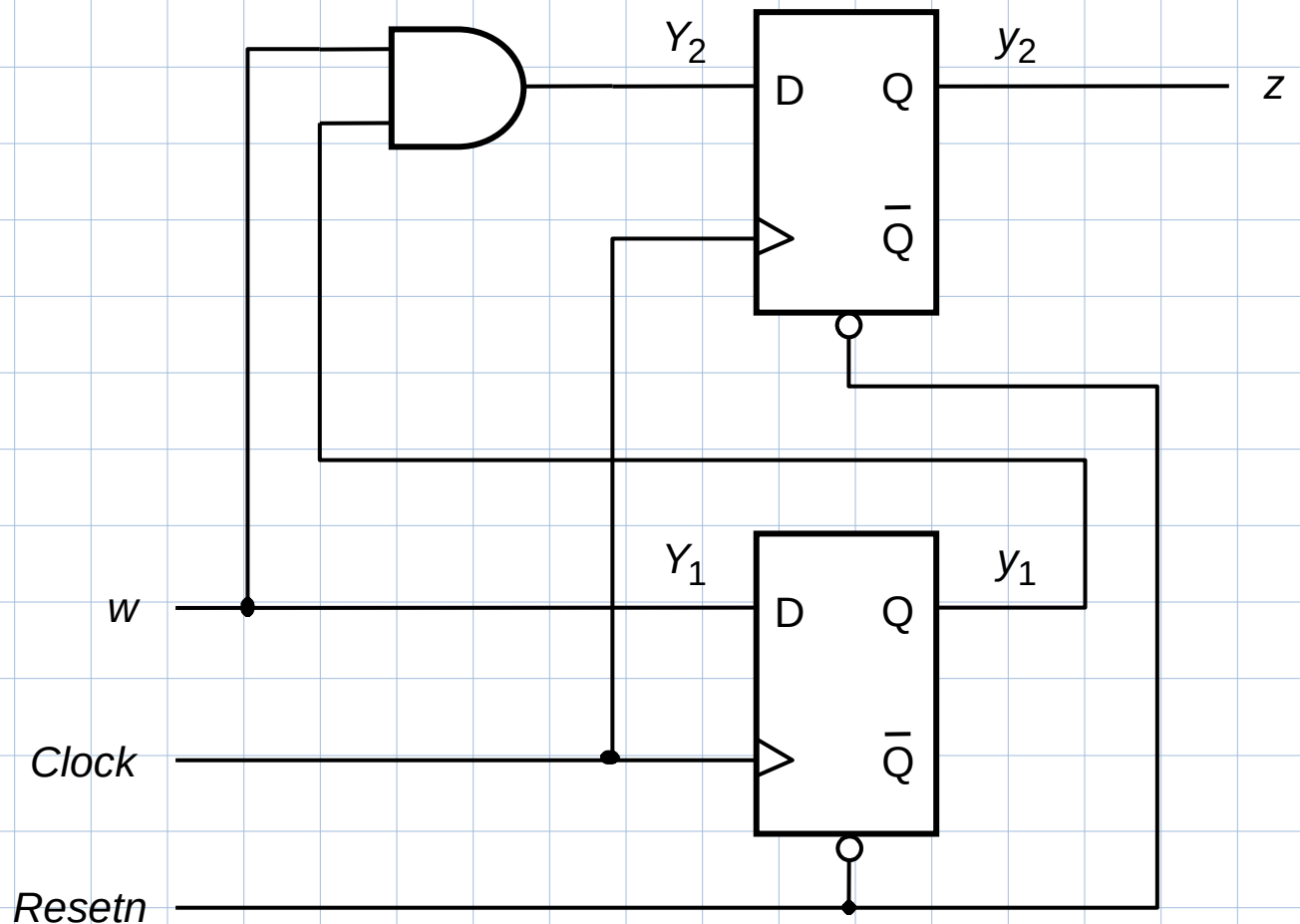
$$Y_1 = D_1 = w$$
$$Y_2 = D_2 = wy_1$$
$$z = y_2$$

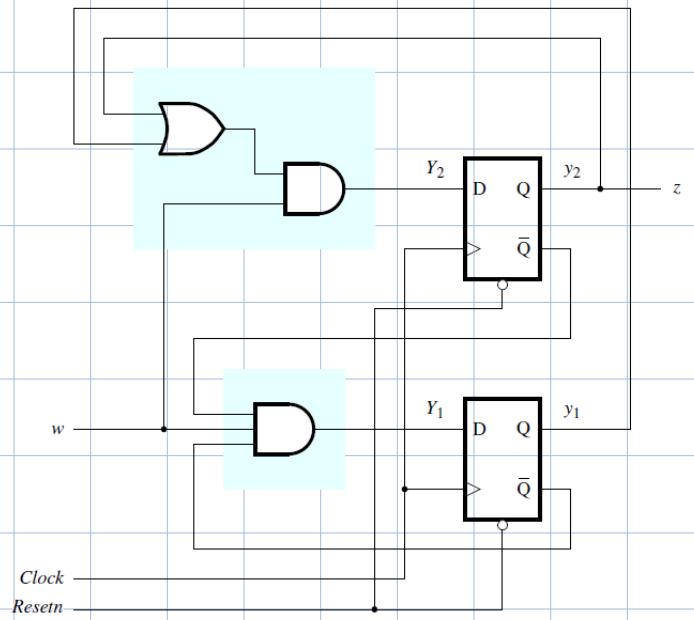
- El costo de este circuito es menor que el del ejemplo anterior (ver circuito en la siguiente página)
- En circuitos muy complicados, es difícil encontrar la asignación de estados óptima
  - Las herramientas de CAD lo hacen y sus algoritmos son secretos industriales

	Present state	Next state		Output
		$w = 0$	$w = 1$	
	y_2y_1	Y_2Y_1	Y_2Y_1	z
A	00	00	01	0
B	01	00	11	0
C	11	00	11	1
	10	*dd*	*dd*	*d*

8.2 Problema de asignación de estados

Circuito nuevo





Circuito antiguo

8.2.1 Codificación one-hot

- Una alternativa de asignación de estados es usar un flip flop por cada estado
- Las expresiones para el siguiente estado son:

$$Y_1 = \overline{w}$$
$$Y_2 = wy_1$$
$$Y_3 = w\overline{y}_1$$
$$z = y_3$$

- En caso de codificación one-hot, estas expresiones pueden ser obtenidas por inspección
  - No necesariamente son más simples…

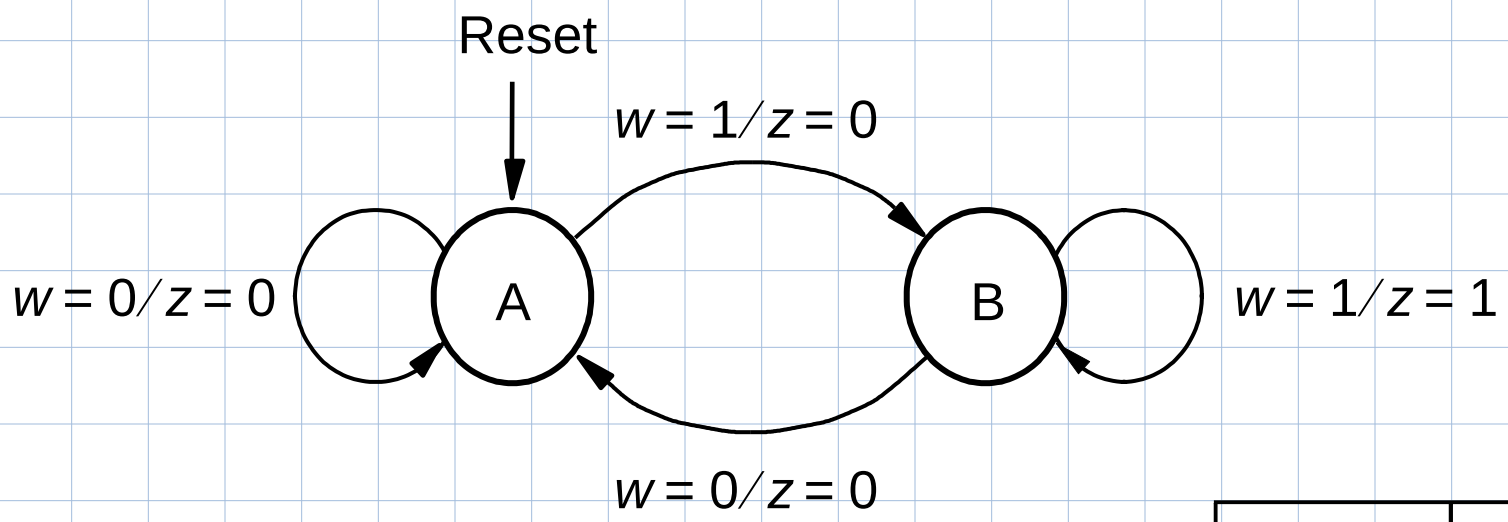
	Present state	Next state		Output
		$w = 0$	$w = 1$	
	$y_3y_2y_1$	$Y_3Y_2Y_1$	$Y_3Y_2Y_1$	z
A	0 0 1	0 0 1	0 1 0	0
B	0 1 0	0 0 1	1 0 0	0
C	1 0 0	0 0 1	1 0 0	1

8.3 Modelo de estados de Mealy

- En 8.1.1 vimos la **máquina de Moore**, en que cada estado tiene asociado un valor para la salida
- Existe otro concepto de máquina de estado, llamado **máquina de Mealy**, en que las salidas son función del estado **y también** de las entradas
- Las máquinas de Mealy son **más flexibles** en su diseño que las de Moore
- Haremos un ejemplo similar al anterior: queremos detectar dos entradas 1 consecutivas y la salida debiera activarse en el mismo ciclo en que la segunda entrada igual a 1 es detectada

Clock cycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	1	0	0	1	1	0	0

8.3 Modelo de estados de Mealy



- En este caso el diagrama de estados requiere sólo de 2 estados
- La notación puede ser confusa: en el ciclo presente, el valor de la salida corresponde a la etiqueta de la flecha que sale del nodo del estado presente

Present state	Next state		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
A	A	B	0	0
B	A	B	0	1

8.3 Modelo de estados de Mealy

- En este caso la asignación de estados es trivial
  - Usamos un único flip-flop, correspondiente a una única variable de estado y
- Usando un flip-flop D, las expresiones combinacionales resultantes son

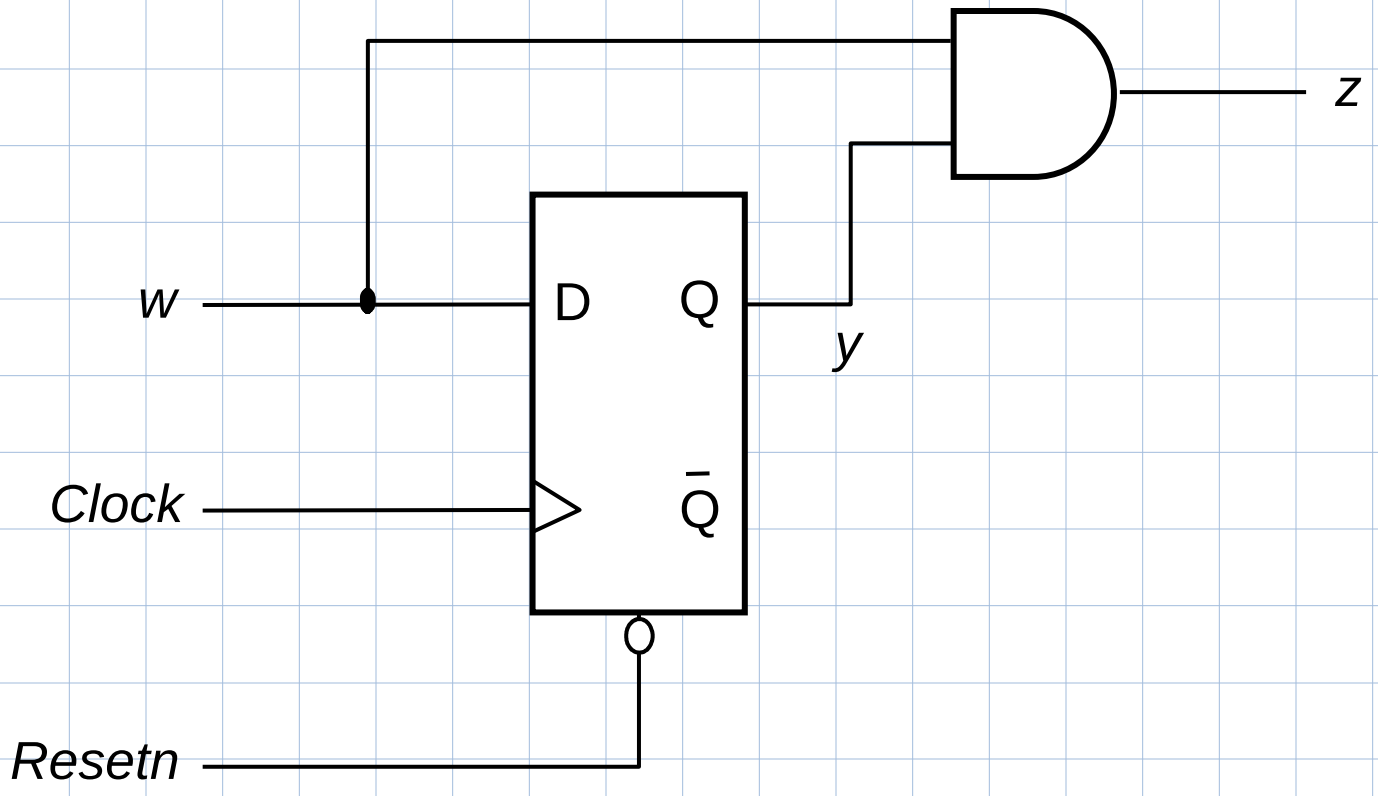
$$Y = D = w$$

$$z = wy$$

	Present state	Next state		Output	
		$w = 0$	$w = 1$	$w = 0$	$w = 1$
	y	Y	Y	z	z
A	0	0	1	0	0
B	1	0	1	0	1

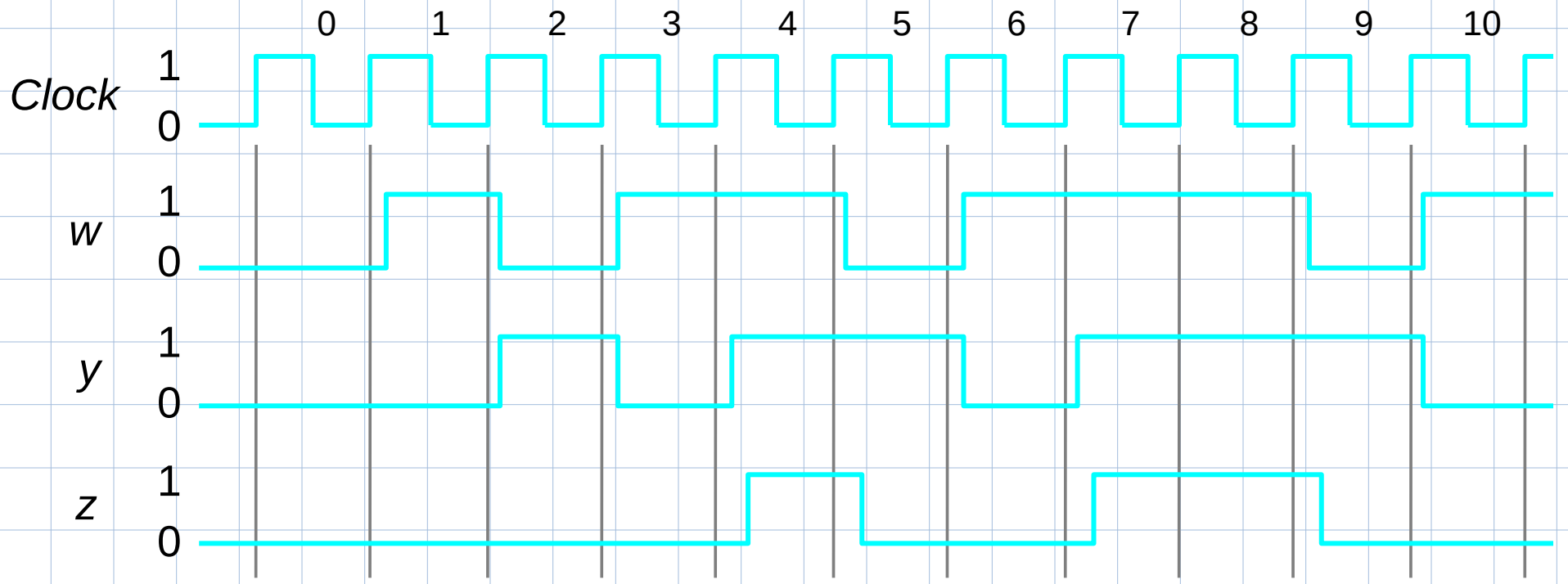
8.3 Modelo de estados de Mealy

Circuito resultante



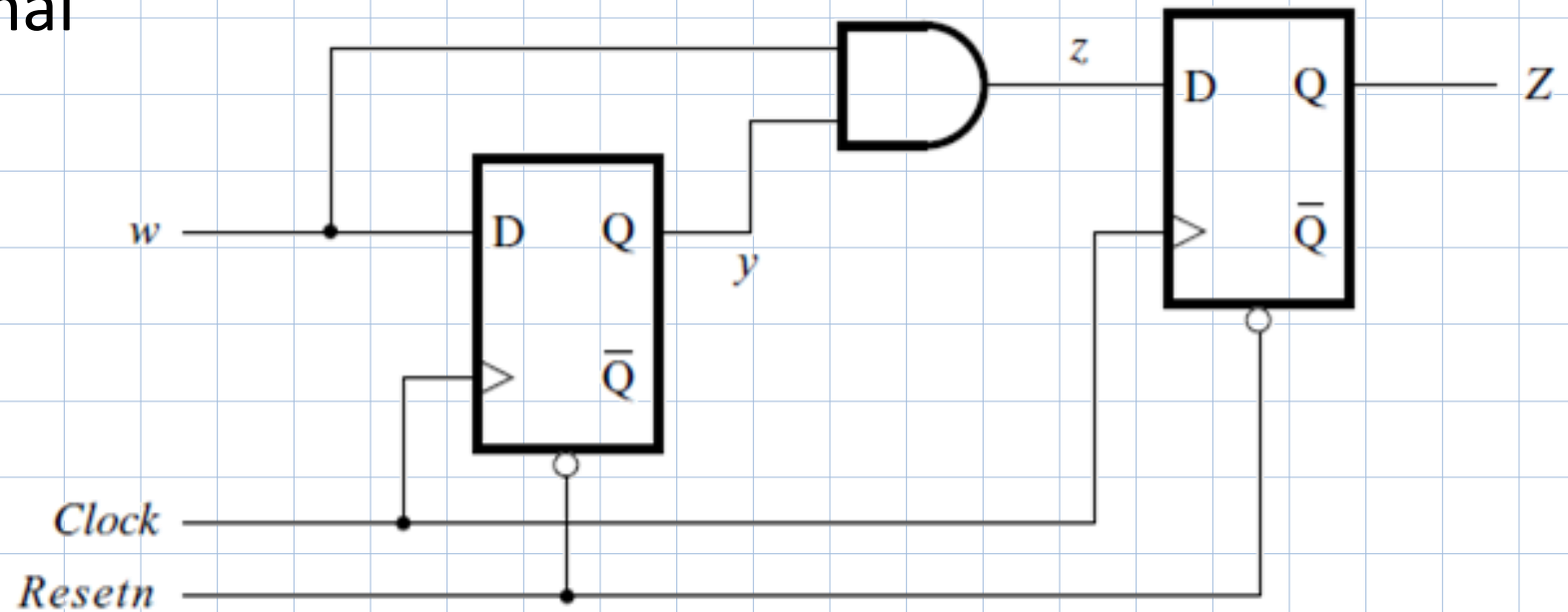
8.3 Modelo de estados de Mealy

Diagrama de tiempo



8.3 Modelo de estados de Mealy

- La mayor flexibilidad de las máquinas de Mealy por sobre las de Moore resulta en circuitos más simples
- Si quisiéramos lograr la misma funcionalidad del circuito anterior usando una FSM de Moore, tendríamos que agregar un flip-flop adicional





8.4 – 8.5.3

FSM en CAD y ejemplo

angel@uc.cl

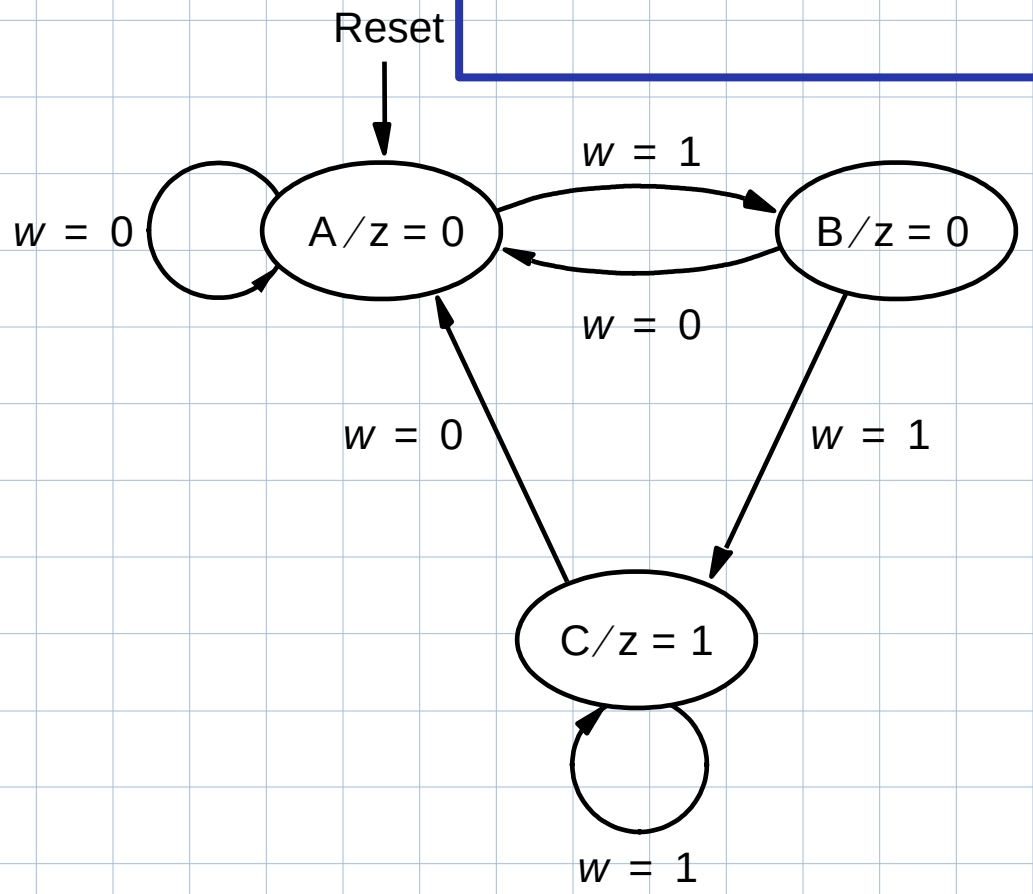
IEE2713 Sistemas Digitales

8.4 Diseño de FSMs usando CAD

- Las técnicas que vimos previamente son técnicas manuales
  - Igual es posible usarlas con CAD y Verilog, pero no es muy eficiente
- Hay herramientas de CAD diseñadas para trabajar con FSMs
  - Una opción es ingresar directamente el diagrama de estados
  - Otra alternativa es usar HDL directamente
- Veremos esto último a continuación

8.4.1 Verilog para FSMs tipo Moore

- No existe una forma estándar de describir FSMs en Verilog
- Una posibilidad es usar vectores para representar variables de estado presente (y) y estado futuro (Y)
  - Usamos parámetros para asociar estados a valores de variables de estado
- Las transiciones entre estados están especificadas en bloques always
  - Un bloque combinacional para la tabla
  - Un bloque secuencial para los FFs
- Es una máquina de Moore porque la salida z=1 va asociada al estado C



8.4.1 Verilog para FSMs tipo Moore

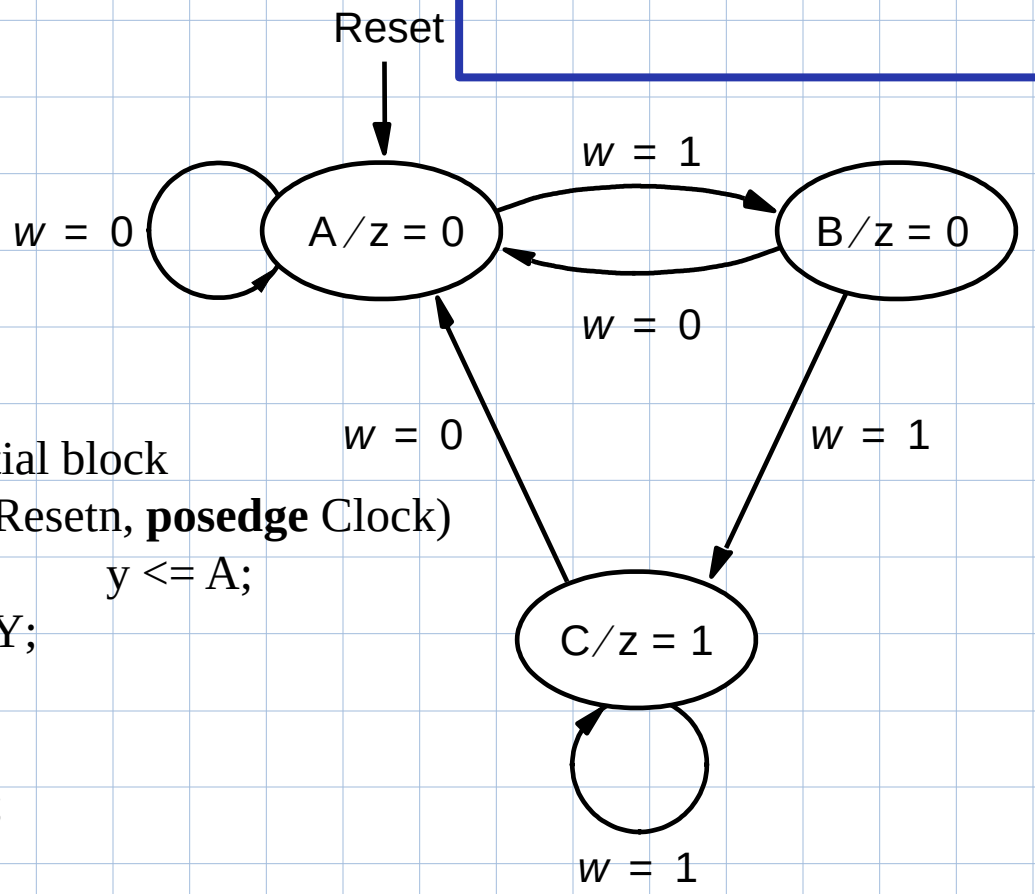
```
module simple (Clock, Resetn, w, z);
    input Clock, Resetn, w;
    output z;
    reg [2:1] y, Y;
    parameter [2:1] A = 2'b00, B = 2'b01, C = 2'b10;

    // Define the next state combinational circuit
    always @(w, y)
        case (y)
            A: if (w) Y = B;
               else   Y = A;
            B: if (w) Y = C;
               else   Y = A;
            C: if (w) Y = C;
               else   Y = A;
            default:  Y = 2'bxx;
        endcase

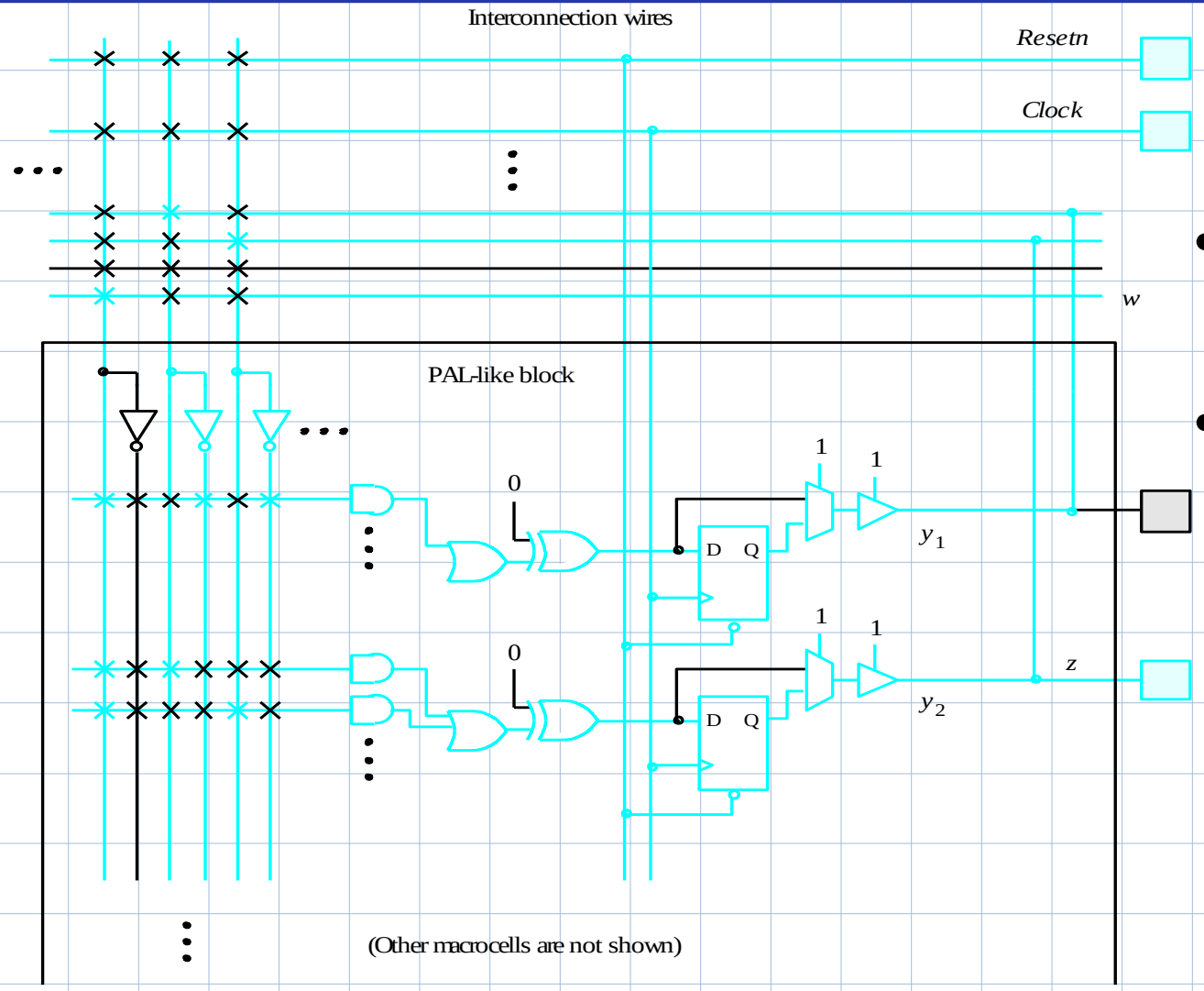
    // Define the sequential block
    always @(negedge Resetn, posedge Clock)
        if (Resetn = = 0)       y <= A;
        else                    y <= Y;

    // Define output
    assign z = (y = = C);

endmodule
```

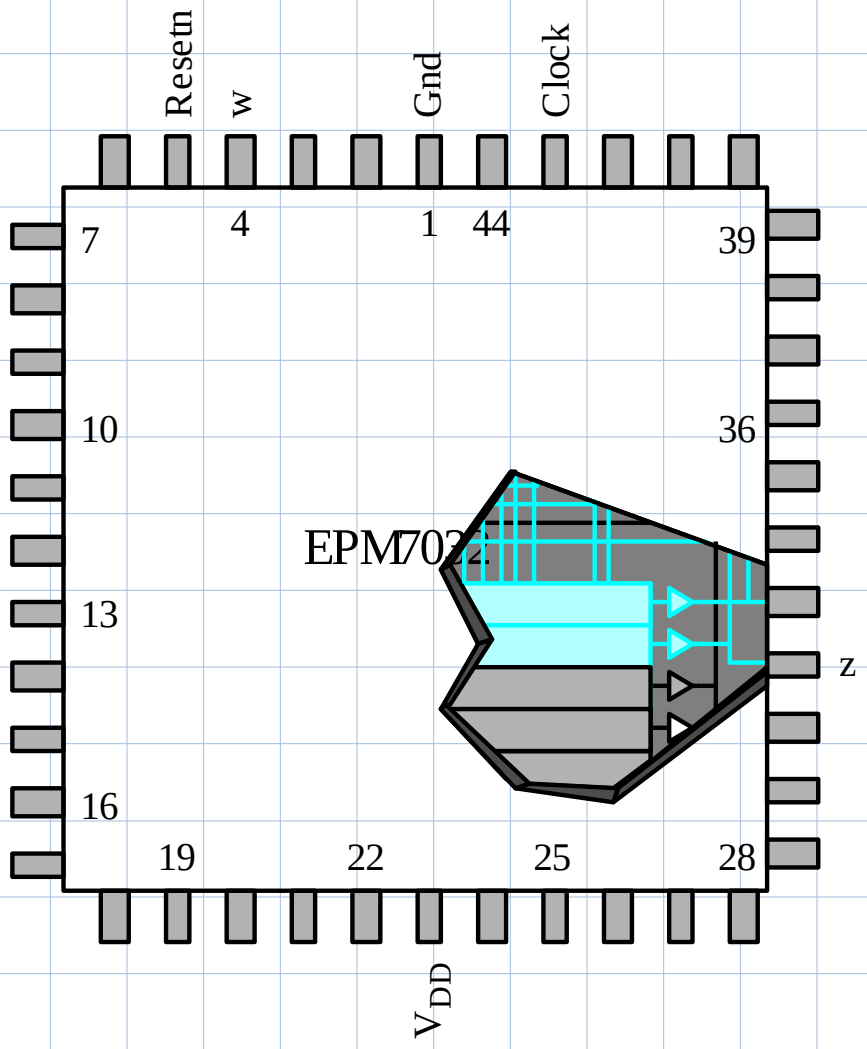


8.4.2 Síntesis del Verilog en CPLD



- La figura muestra parte de la implementación en CPLD
- El reloj y el reset son distribuidos a todo el chip mediante interconexiones globales (global wires)

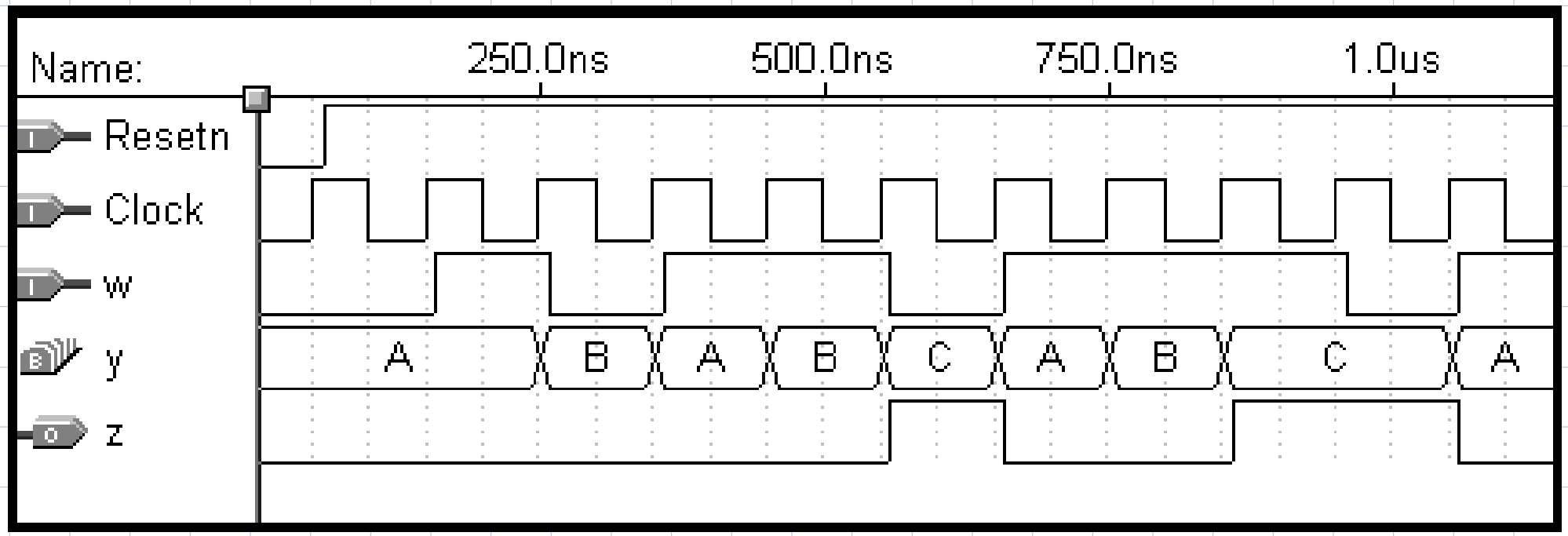
8.4.2 Síntesis del Verilog en CPLD



- La figura muestra una posible asignación de pines en un CPLD
- El circuito sólo utiliza una porción pequeña del CPLD, dos macroceldas

8.4.3 Simulación y pruebas del circuito

- Podemos realizar simulaciones y obtener diagramas de tiempo del funcionamiento
- Los estados están marcados con las letras A, B y C



8.4.4 Estilos alternativos del código

```
module simple (Clock, Resetn, w, z);
    input Clock, Resetn, w;
    output reg z;
    reg [2:1] y, Y;
    parameter [2:1] A = 2'b00, B = 2'b01, C = 2'b10;
    // Define the next state combinational circuit
    always @(w, y)
    begin
        case (y)
            A: if (w)  Y = B;
               else   Y = A;
            B: if (w)  Y = C;
              else    Y = A;
            C: if (w)  Y = C;
              else    Y = A;
            default:   Y = 2'bxx;
        endcase
        z = (y = = C);              //Define output
    end

    // Define the sequential block
    always @(negedge Resetn, posedge Clock)
        if (Resetn = = 0)  y <= A;
        else  y <= Y;

endmodule
```

En esta implementación alternativa especificamos z en el bloque always del circuito combinacional

8.4.4 Estilos alternativos del código

```
module simple (Clock, Resetn, w, z);
    input Clock, Resetn, w;
    output z;
    reg [2:1] y;
    parameter [2:1] A = 2'b00, B = 2'b01, C = 2'b10;

    // Define the sequential block

    always @(negedge Resetn, posedge Clock)
        if (Resetn = = 0)   y <= A;
        else
            case (y)
                A: if (w)   y <= B;
                   else     y <= A;
                B: if (w)   y <= C;
                   else     y <= A;
                C: if (w)   y <= C;
                   else     y <= A;
                default:    y <= 2'bxx;
            endcase

    // Define output
    assign z = (y = = C);

endmodule
```

- En esta tercera alternativa usamos un único bloque always
  - Combinamos flip-flop y máquina combinacional
- z es obtenido mediante asignación continua fuera del bloque always
  - Si fuera dentro del bloque always, usaría un FF adicional

8.4.5 Resumen de pasos de diseño con Verilog

- Hemos aprendido a diseñar máquinas secuenciales síncronas en Verilog
- La descripción en Verilog simplifica el trabajo, pero no puede reemplazar algunos pasos manuales en la definición de los estados
  - En el fondo, igual ayuda hacer un diagrama de estados de Mealy o de Moore

8.4.6 Especificación de la asignación de estados en Verilog

- La asignación de los estados afecta la complejidad del circuito
- El CAD podría encargarse de optimizar dicha complejidad
- En los ejemplos anteriores usamos parámetros para declarar los valores de las variables de estado en diferentes estados
- El compilador es capaz de reconocer descripciones de FSMs como las hemos visto en este capítulo, y puede tratar de optimizar la implementación
  - El diseñador puede activar o desactivar los algoritmos de manejo de FSMs

8.4.7 Especificación de una FSM de Mealy en Verilog

```
module mealy (Clock, Resetn, w, z);
   input Clock, Resetn, w;
   output z;
   reg y, Y, z;
   parameter A = 0, B = 1;

   // Define the next state and output combinational circuits
   always @(w or y)
      case (y)
         A: if (w)
            begin
               z = 0;
               Y = B;
            end
            else
            begin
               z = 0;
               Y = A;
            end
         B: if (w)
            begin
               z = 1;
               Y = B;
            end
            else
            begin
               z = 0;
               Y = A;
            end
      endcase

   // Define the sequential block
   always @(negedge Resetn or posedge Clock)
      if (Resetn == 0)  y <= A;
      else  y <= Y;

endmodule
```

- Especificamos las transiciones de estado de igual forma que en una FSM de Moore
- z está definido dentro del bloque case
  - Como z depende de w, la lista de sensibilidad debe incluir a w
- Existe un potencial problema con esta implementación de Mealy: si w llega en forma asíncrona, puede activar z erróneamente y producir un glitch – ver figura 8.38 en libro

8.5 Ejemplo de sumador serial

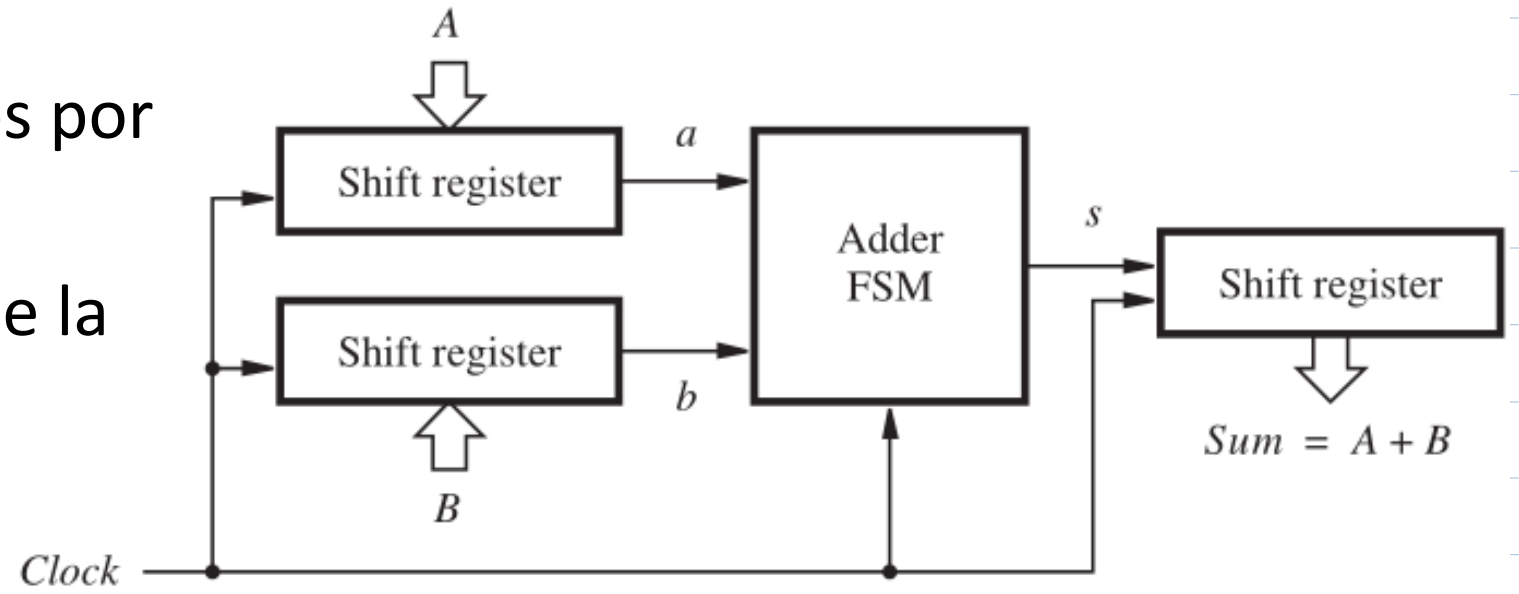
- En el capítulo 5 aprendimos sobre sumadores
  - Los sumadores rápidos son más complejos y caros en términos de bien raíz de silicio
- Si no nos importa tener un sumador lento, podemos usar un sumador serial
- Supongamos que queremos sumar dos números A + B = Sum, con

$$A = a_{n-1}a_{n-2}\ldots a_0 \qquad B = b_{n-1}b_{n-2}\ldots b_0 \qquad Sum = s_{n-1}s_{n-2}\ldots s_0$$

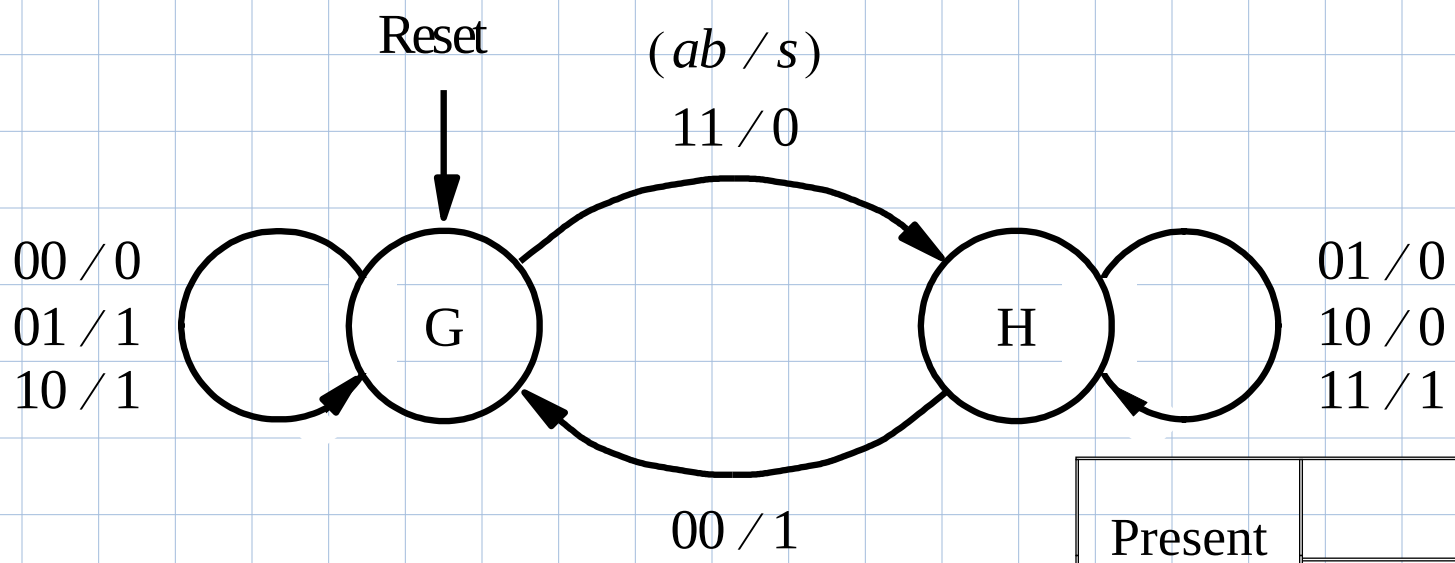
- Comenzaremos sumando $a_0 + b_0$, luego $a_1 + b_1$, y así sucesivamente
- Llevemos esto a un diagrama de bloques

8.5 Ejemplo de sumador serial

- Debemos partir cargando los sumandos en los registros de desplazamiento
- En cada ciclo de reloj sumamos un nuevo par de bits y almacenamos el resultado en el registro de desplazamiento Sum
- Usaremos FFs gatillados por flanco positivo
- Intentemos el diseño de la FSM a continuación



8.5.1 FSM de Mealy para el sumador serial



Sólo 2 estados, basta con 1 flip-flop

G: carry-in = 0
H: carry-in = 1

Present state	Next state				Output *s*			
	ab =00	01	10	11	00	01	10	11
G	G	G	G	H	0	1	1	0
H	G	H	H	H	1	0	0	1

- Necesitamos dos estados, G y H, determinados por el valor del carry
- La salida s depende de las entradas y del estado

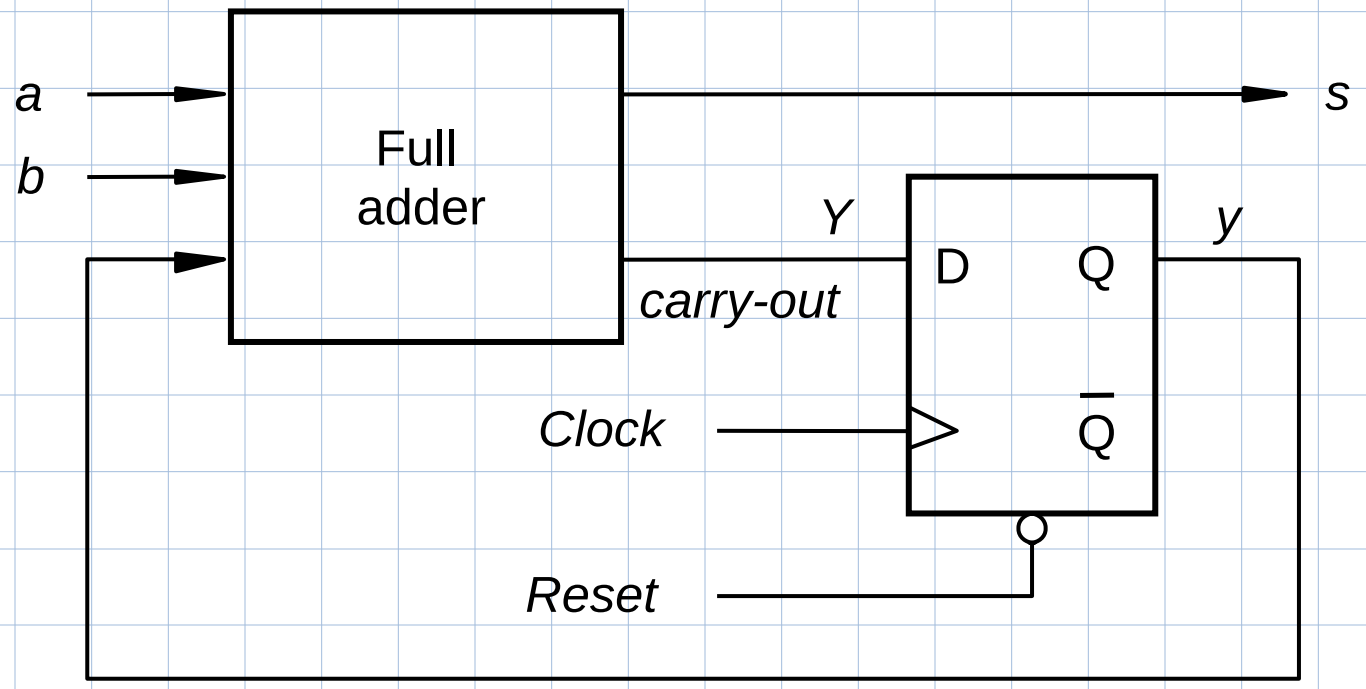
8.5.1 FSM de Mealy para el sumador serial

- La asignación de estados nos lleva a las siguientes ecuaciones para el estado siguiente y la salida:

$$Y = ab + ay + by$$

$$s = a \oplus b \oplus y$$

- En este caso, y es el carry-in, Y es el carry-out, y s es la suma



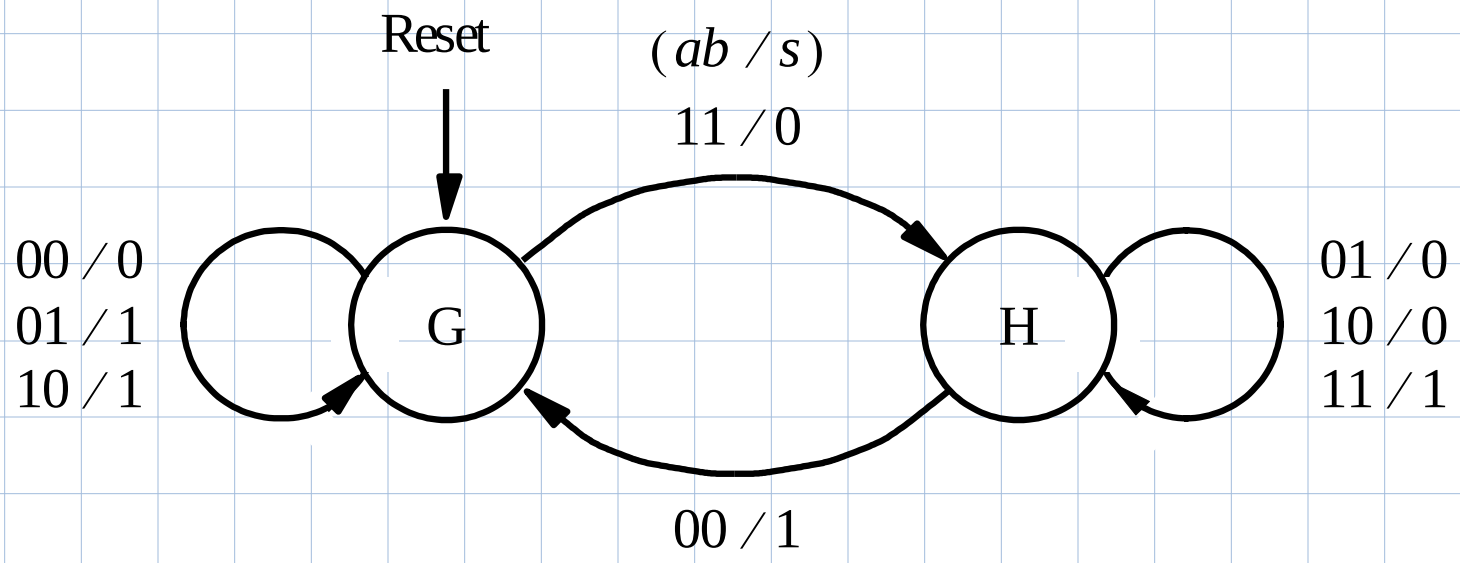
8.5.1 FSM de Mealy para el sumador serial

- La asignación de estados nos lleva a las siguientes ecuaciones para el estado siguiente y la salida:

- En este caso, y es el carry-in, Y es el carry-out, y s es la suma

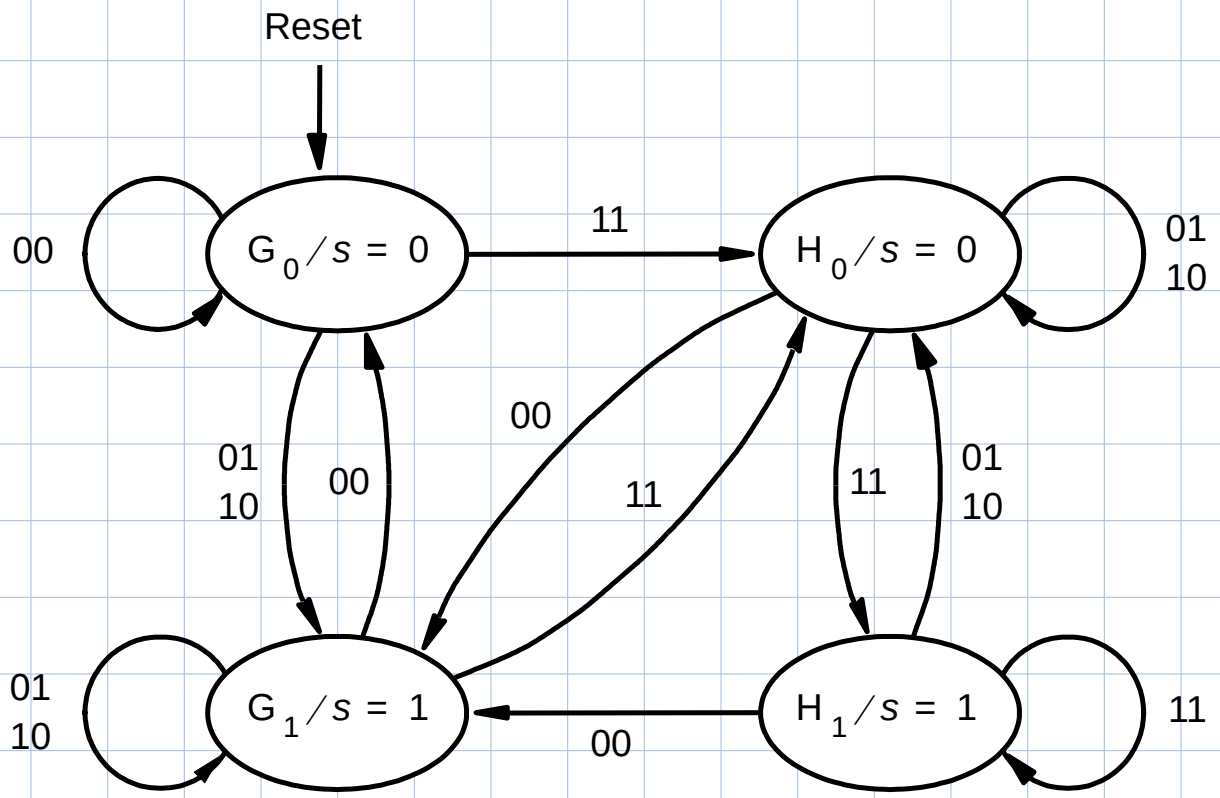
8.5.2 FSM de Moore para el sumador serial

- Intentaremos un diseño para la FSM del sumador empleando máquina de estados de Moore
- Dado que en Moore la salida va asociada al estado, tenemos que aumentar el número de estados, porque en G y H se admiten ambos tipos de salidas



Máquina de Mealy

8.5.2 FSM de Moore para el sumador serial



Present state	Nextstate				Output
	ab =00	01	10	11	*s*
G_0	G_0	G_1	G_1	H_0	0
G_1	G_0	G_1	G_1	H_0	1
H_0	G_1	H_0	H_0	H_1	0
H_1	G_1	H_0	H_0	H_1	1

8.5.2 FSM de Moore para el sumador serial

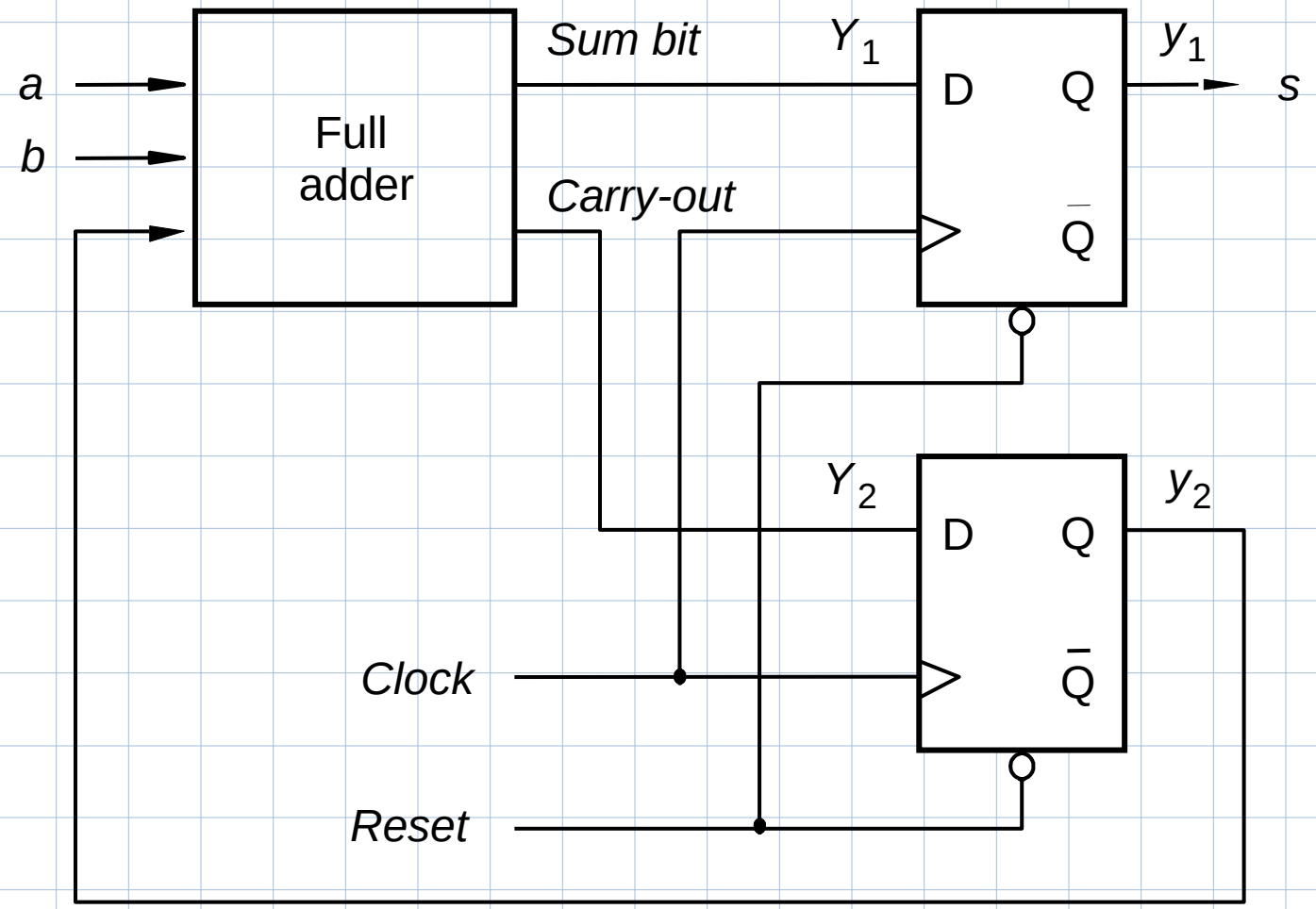
- La tabla muestra la asignación de estados
- Las expresiones para el siguiente estado y la salida son:

$$Y_1 = a \oplus b \oplus y_2$$
$$Y_2 = ab + ay_2 + by_2$$
$$s = y_1$$

- Similar a la máquina de Mealy, pero aquí la salida *s* pasa por un FF extra y se retrasa un ciclo

Present state y_2y_1	Nextstate Y_2Y_1				Output s
	ab =00	01	10	11	
00	0 0	01	0 1	10	0
01	0 0	01	0 1	10	1
10	0 1	10	1 0	11	0
11	0 1	10	1 0	11	1

8.5.2 FSM de Moore para el sumador serial – circuito resultante



8.5.3 Código verilog para el sumador serial

```
module shiftrne (R, L, E, w, Clock, Q);
    parameter n = 8;
    input [n-1:0] R;
    input L, E, w, Clock;
    output reg [n-1:0] Q;
    integer k;

    always @(posedge Clock)
        if (L)
            Q <= R;
        else if (E)
        begin
            for (k = n-1; k > 0; k = k-1)
                Q[k-1] <= Q[k];
            Q[n-1] <= w;
        end

endmodule
```

- Comenzaremos con el shift register, que usamos en ambas implementaciones
- Este SR tiene entrada de enable

8.5.3 Código verilog para el sumador serial 1/3

```
module serial_adder (A, B, Reset, Clock, Sum);
   input [7:0] A, B;
   input Reset, Clock;
   output wire [7:0] Sum;
   reg [3:0] Count;
   reg s, y, Y;
   wire [7:0] QA, QB;
   wire Run;
   parameter G = 1'b0, H = 1'b1;

   shiftrne shift_A (A, Reset, 1'b1, 1'b0, Clock, QA);
   shiftrne shift_B  (B, Reset, 1'b1, 1'b0, Clock, QB);
   shiftrne shift_Sum  (8'b0, Reset, Run, s, Clock, Sum);
```

8.5.3 Código verilog para el sumador serial 2/3

```
// Adder FSM
// Output and next state combinational circuit
always @(QA, QB, y)
    case (y)
        G: begin
            s = QA[0] ^ QB[0];
            if (QA[0] & QB[0])   Y = H;
             else                Y = G;
                end
        H: begin
            s = QA[0] ~^ QB[0];
            if (~QA[0] & ~QB[0])   Y = G;
            else    Y = H;
             end
        default: Y = G;
     endcase
```

8.5.3 Código verilog para el sumador serial 3/3

```
// Sequential block
always @(posedge Clock)
    if (Reset) y <= G;
    else     y <= Y;

// Control the shifting process
always @(posedge Clock)
    if (Reset)                  Count = 8;
    else if (Run)               Count = Count - 1;
assign Run = |Count;

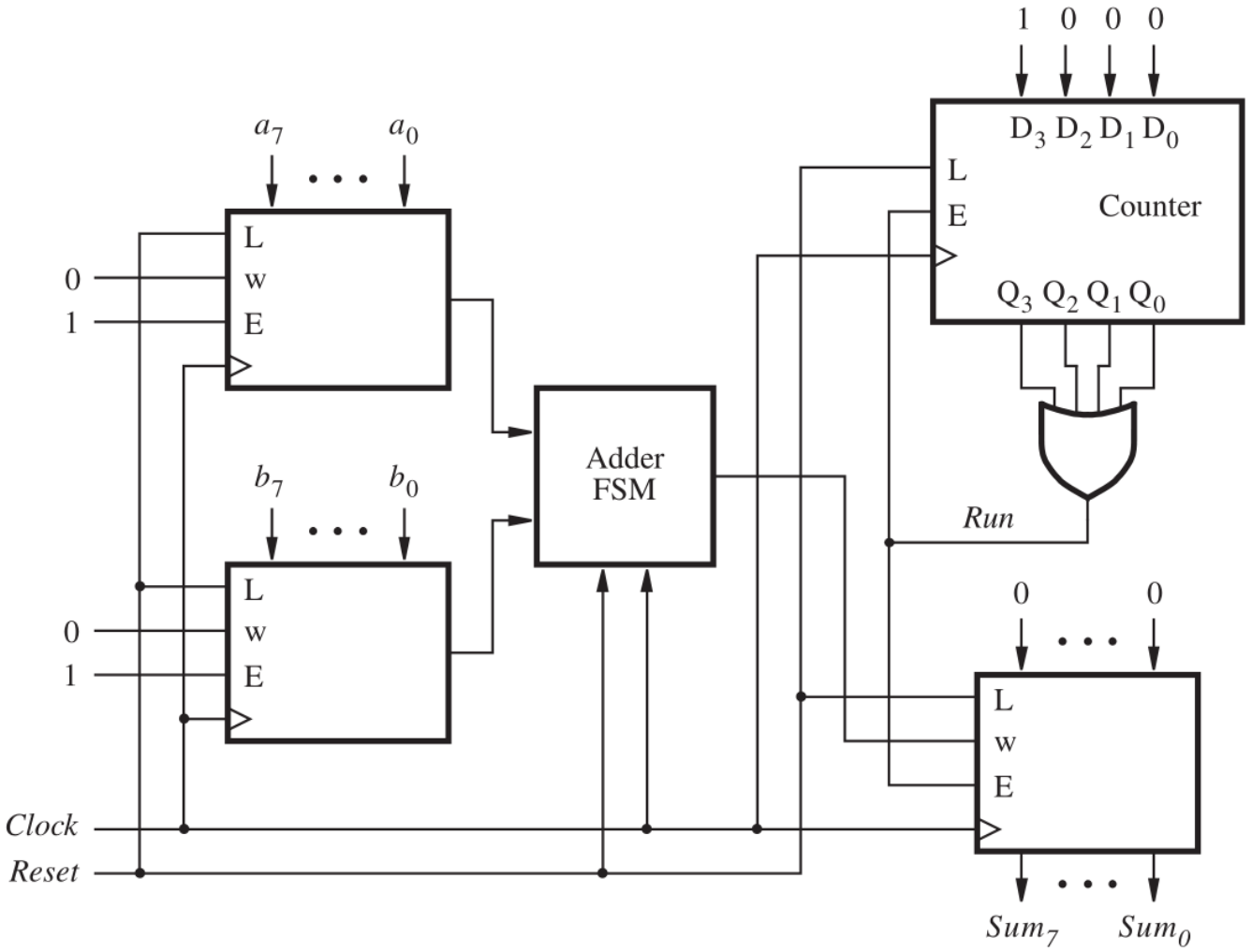
endmodule
```

Cargamos en el contador el número de bits que queremos sumar

El contador reduce la cuenta hasta 0

Usamos máquina de Mealy en este ejemplo

8.5.3 Código verilog para el sumador serial – Circuito resultante





8.6 – 8.7.4

Minimización de estados y ejemplo

angel@uc.cl

IEE2713 Sistemas Digitales

8.6 Minimización de estados

- Para máquinas de estado finitas más complejas, no es fácil determinar el número mínimo de estados requeridos
- Nos interesa utilizar menos estados para:
  - Reducir la cantidad de flip-flops necesarios para describir los estados
  - Simplificar los circuitos combinacionales
- Se dice que dos estados son *equivalentes*, si y solo si para cada posible **secuencia de entradas**, la **secuencia de salida** será exactamente la misma
- El CAD puede detectar estados equivalentes
- A veces es tmás fácil determinar si dos estados **no son** equivalentes

8.6.1 Procedimiento para la minimización de la partición

- Este método se enfoca en identificar los estados que **no** son equivalentes
- Para ello, el conjunto de estados es separado en particiones con subconjuntos de estados de manera que los estados de un subconjunto no son equivalentes a los de otro subconjunto
- Consideremos la partición inicial de la tabla
  - P1 = (ABCDEFG)
- Propondremos una segunda partición de acuerdo a estados que tienen la misma salida:
  - P2 = (ABD)(CEFG)
- Ahora hay que evaluar si los estados entre paréntesis son equivalentes. Si lo son, los consideraremos como un solo estado

¿Es una FSM de Moore o Mealy?

Present state	Next state $w = 0$	Next state $w = 1$	Output z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

8.6.1 Procedimiento para la minimización de la partición

- Analizamos (ABD):
  - Cuando w = 0: Los estados siguientes de (ABD) son (BDB)
    - como estos están en el mismo bloque, entonces los estados A, B y D podrían ser equivalentes
  - Cuando w = 1: Los estados siguientes son (CFG)
    - como estos también están en el mismo bloque, los estados A, B y D podrían ser equivalentes
- Analizamos (CEFG):
  - Cuando w = 0: Los estados siguientes son (FFEF), los cuales están en el mismo bloque
  - Cuando w = 1: Los estados siguientes son (ECDG)
    - Como cada estado no está en el mismo bloque, entonces (CEFG) no son equivalentes y debemos buscar otra partición
- Probamos con una nueva partición:
  - P3 = (ABD)(CEG)(F)

P2 = (ABD)(CEFG)

Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

8.6.1 Procedimiento para la minimización de la partición

- Repitiendo el mismo proceso para P3, nos damos cuenta que ahora los estados siguientes del bloque (ABD) no están en la misma partición
- Particionando nuevamente:
  - P4 = (AD)(B)(CEG)(F)
- Realizando el mismo análisis para P4 nos damos cuenta que cumplimos con los requisitos que sus estados siguientes tanto para w = 0 como para w = 1 se encuentran en los mismos bloques

P3 = (ABD)(CEG)(F)

Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

8.6.1 Procedimiento para la minimización de la partición

- Finalmente, de P4 concluimos que los estados A y D son equivalentes, al igual que C con E y G
- De esta forma, logramos reducir el número de estados de 7 a 4, disminuyendo en 1 la cantidad de flip-flops necesarios para representar todos los estados
- Esto nos lleva a una nueva tabla con menos estados
- En la ayudantía se verá un ejercicio más completo y aplicado creando una máquina expendedora (ejemplo 8.7 del libro)

P4 = (AD)(B)(CEG)(F)

Present state	Nextstate		Output z
	w = 0	w = 1	
A	B	C	1
B	A	F	1
C	F	C	0
F	C	A	0

8.6.2 FSMs con especificación incompleta

- El método de partición funciona bien cuando están bien definidas todas las entradas y estados
- Se dice que una FSM tiene especificación incompleta cuando una o más entradas en la tabla de estados no están especificadas (don’t cares)
- Es importante notar que este proceso de minimización **no garantiza** una implementación menos compleja
  - A veces es más importante la asignación de estados que la minimización de éstos

8.6.2 FSMs con especificación incompleta - ejemplo

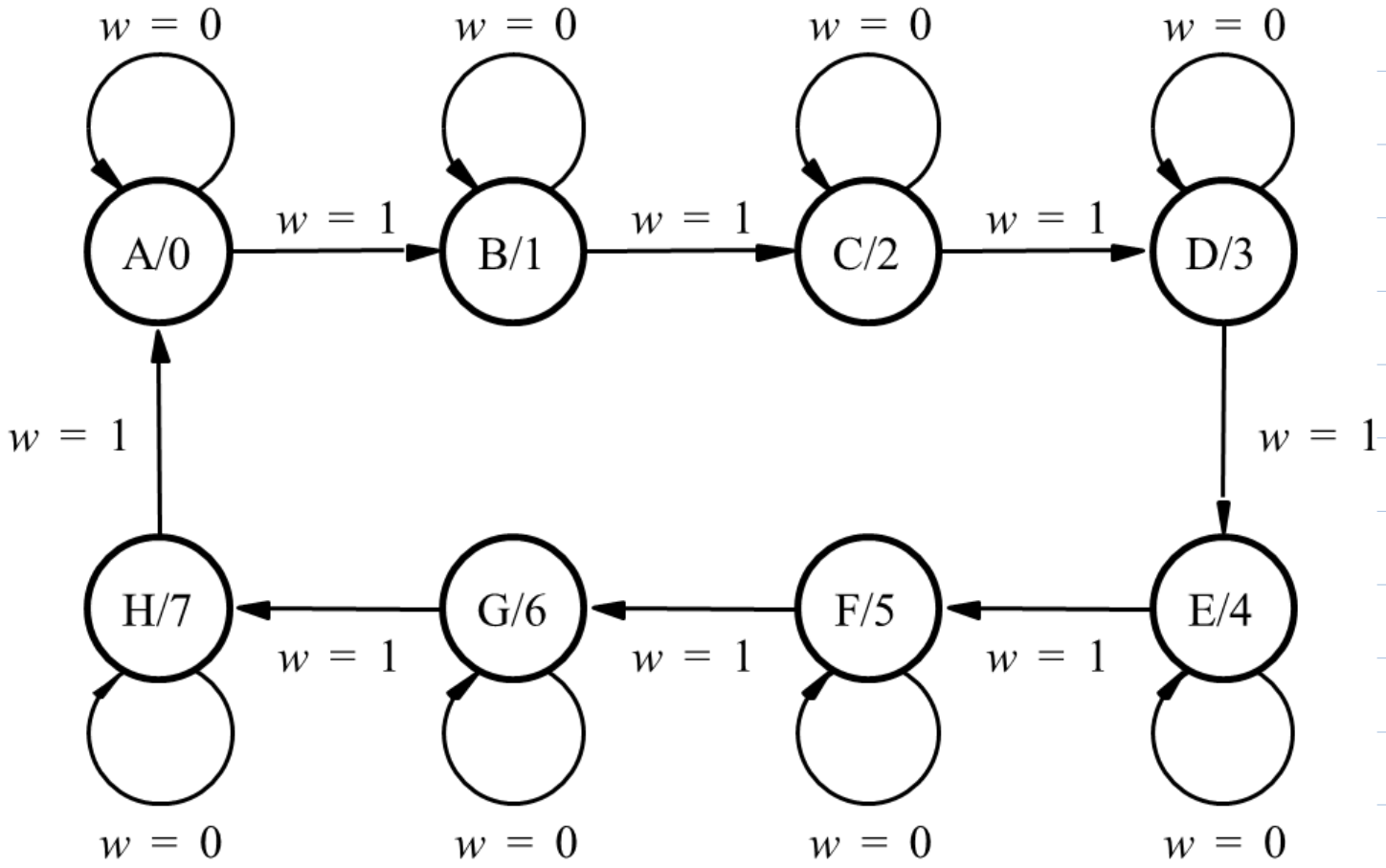
- En la tabla, notamos que hay estados siguientes y salidas que no están definidas
- Por ejemplo, si realizamos el proceso de partición asumiendo que las salidas no definidas son 0:
  - P1 = (A)(B)(D)(G)(CE)(F)
- En cambio, si asumimos que las salidas son 1:
  - P1 = (AD)(B)(CEG)(F)
- En el segundo caso, reducimos la cantidad de estados a 4, eliminando un flip-flop por estado

Present state	Next state		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
A	B	C	0	0
B	D	–	0	–
C	F	E	0	1
D	B	G	0	0
E	F	C	0	1
F	E	D	0	1
G	F	–	0	–

8.7 Diseño de un contador usando la aproximación del circuito secuencial

- Ya vimos como funciona un contador, ahora vamos a diseñar uno usando los procedimientos aprendidos hasta ahora
- Requisitos:
  - Secuencia de la cuenta: 0, 1, 2, …, 6, 7, 0, 1, …
  - Entrada w
    - Mientras w = 1 el contador incrementa en cada ciclo de clock
    - Cuando w = 0 la cuenta se pausa

8.7.1 Diagrama de estados y tabla de estados para un contador modulo-8



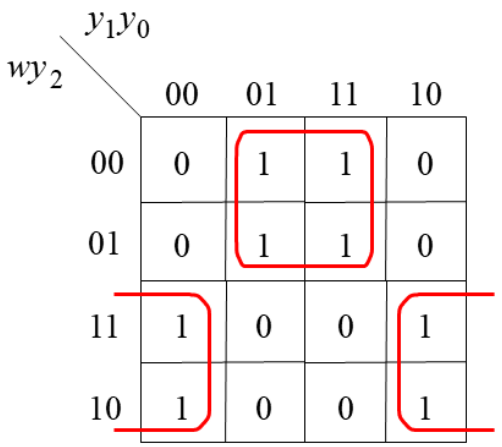
Present state	Next state		Output
	$w = 0$	$w = 1$	
A	A	B	0
B	B	C	1
C	C	D	2
D	D	E	3
E	E	F	4
F	F	G	5
G	G	H	6
H	H	A	7

8.7.2 Asignación de estados

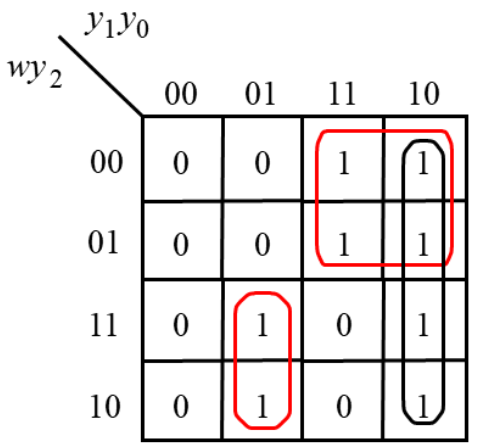
A partir de la tabla, enumeramos los estados para continuar con los mapas de Karnaugh:

	Present state $y_2y_1y_0$	Next state $w = 0$ $Y_2Y_1Y_0$	Next state $w = 1$ $Y_2Y_1Y_0$	Count $z_2z_1z_0$
A	000	000	001	000
B	001	001	010	001
C	010	010	011	010
D	011	011	100	011
E	100	100	101	100
F	101	101	110	101
G	110	110	111	110
H	111	111	000	111

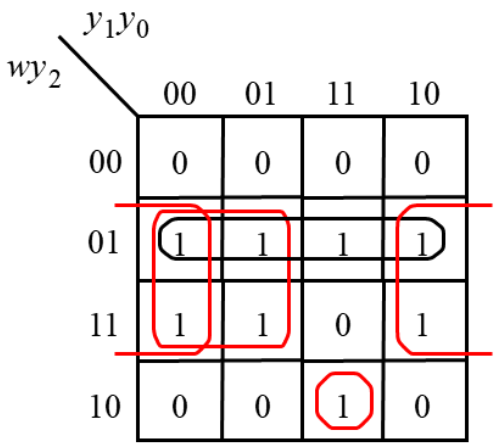
8.7.3 Implementación con flip-flops D



$$Y_0 = \bar{w}y_0 + w\bar{y}_0$$



$$Y_1 = \bar{w}y_1 + y_1\bar{y}_0 + wy_0\bar{y}_1$$



$$Y_2 = \bar{w}y_2 + \bar{y}_0y_2 + \bar{y}_1y_2 + wy_0y_1\bar{y}_2$$

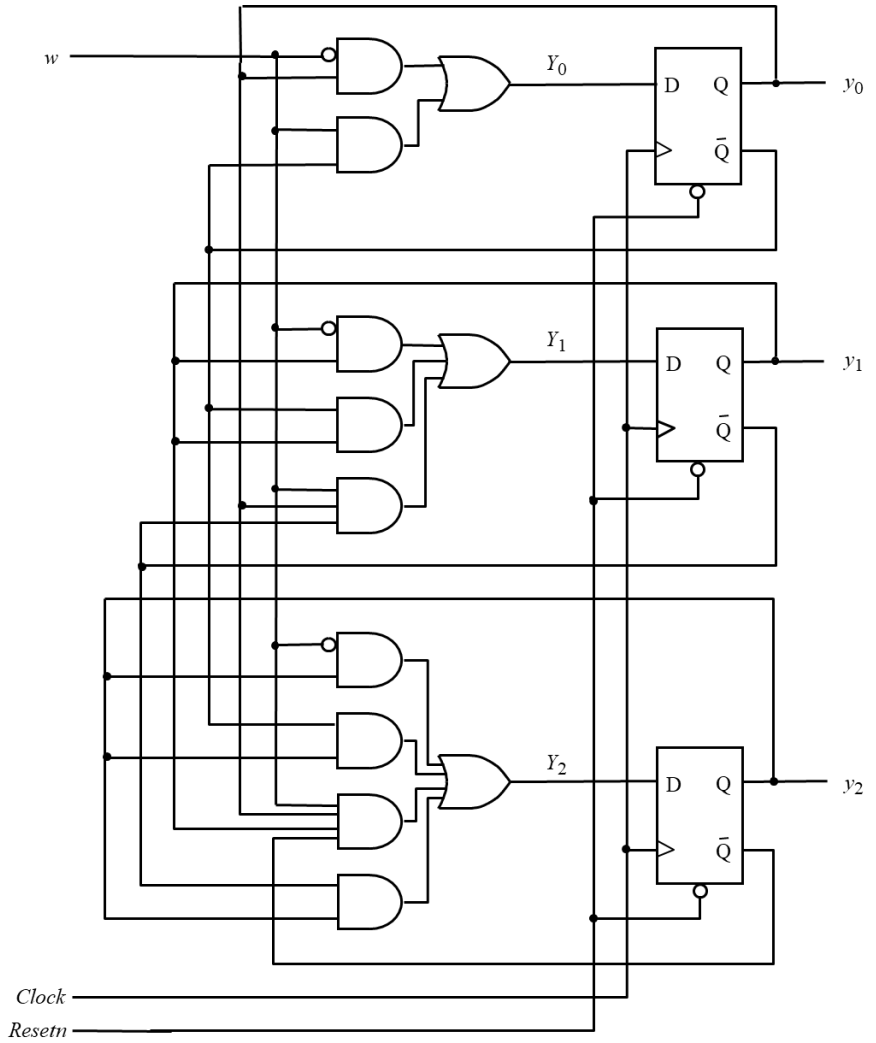
$$D_0 = Y_0 = \overline{w}y_0 + w\overline{y}_0$$

$$D_1 = Y_1 = \overline{w}y_1 + y_1\overline{y}_0 + wy_0\overline{y}_1$$

$$D_2 = Y_2 = \overline{w}y_2 + \overline{y}_0y_2 + \overline{y}_1y_2 + wy_0y_1\overline{y}_2$$

8.7.3 Implementación con flip-flops D

- Presten especial atención a la complejidad del circuito y la cantidad de compuertas utilizadas
- A continuación vamos a realizar el mismo procedimiento pero ahora con flip-flops JK



8.7.3 Implementación con flip-flops D

Para implementar un contador con más bits, podemos ver un patrón al simplificar las ecuaciones obtenidas de los mapas de Karnaugh:

$$D_0 = Y_0 = \overline{w}y_0 + w\overline{y}_0$$
$$D_1 = Y_1 = \overline{w}y_1 + y_1\overline{y}_0 + wy_0\overline{y}_1$$
$$D_2 = Y_2 = \overline{w}y_2 + \overline{y}_0y_2 + \overline{y}_1y_2 + wy_0y_1\overline{y}_2$$

No hay un patrón claro

Álgebra

$$= (\overline{w} + \overline{y}_0 + \overline{y}_1)y_2 + wy_0y_1\overline{y}_2$$
$$= \overline{wy_0y}_1y_2 + wy_0y_1\overline{y}_2$$
$$= wy_0y_1 \oplus y_2$$

Sí hay un patrón claro

Paréntesis: no es tan difícil diseñar un contador a mano, pero piensen en lo fácil que es hacerlo con Verilog

```
module updowncount (R, Clock, L, E, up_down, Q);
        parameter n = 8;
        input [n-1:0] R;
        input Clock, L, E, up_down;
        output reg [n-1:0] Q;
        always @(posedge Clock)
                if (L)
                        Q <= R;
                else if (E)
                        Q <= Q + (up_down ? 1 : -1);
endmodule
```

8.7.4 Implementación con flip-flops JK

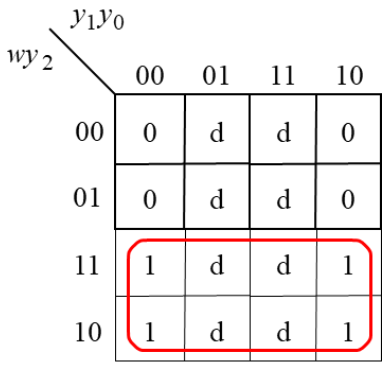
Con flip-flops JK, temenos una alternativa que produce más salidas y mapas-K, pero simplifica el circuito final (muchos don’t cares)

	Present state $y_2y_1y_0$	Flip-flop inputs							Count $z_2z_1z_0$
		$w = 0$				$w = 1$			
		$Y_2Y_1Y_0$	J_2K_2	J_1K_1	J_0K_0	$Y_2Y_1Y_0$	J_2K_2	J_1K_1	J_0K_0	
A	000	000	0d	0d	0d	001	0d	0d	1d	000
B	001	001	0d	0d	d0	010	0d	1d	d1	001
C	010	010	0d	d0	0d	011	0d	d0	1d	010
D	011	011	0d	d0	d0	100	1d	d1	d1	011
E	100	100	d0	0d	0d	101	d0	0d	1d	100
F	101	101	d0	0d	d0	110	d0	1d	d1	101
G	110	110	d0	d0	0d	111	d0	d0	1d	110
H	111	111	d0	d0	d0	000	d1	d1	d1	111

J	K	Q (t+1)
0	0	Q (t)
0	1	0
1	0	1
1	1	$\overline{Q}$ (t)

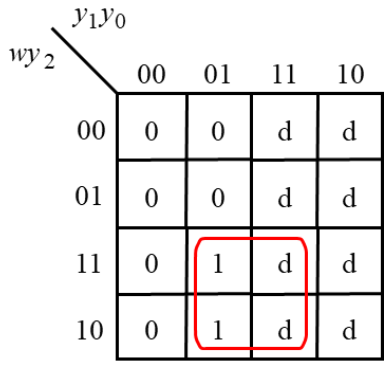
8.7.4 Implementación con flip-flops JK

Los mapas-K resultan mucho más simples, a costa de ser más



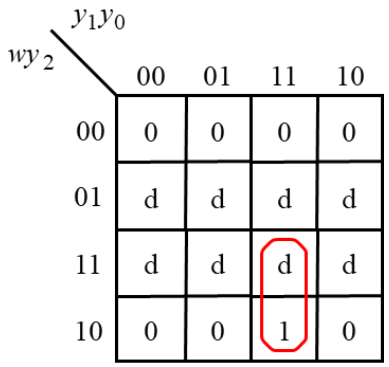
wy_2 \ y_1y_0	00	01	11	10
00	0	d	d	0
01	0	d	d	0
11	1	d	d	1
10	1	d	d	1

$J_0 = w$



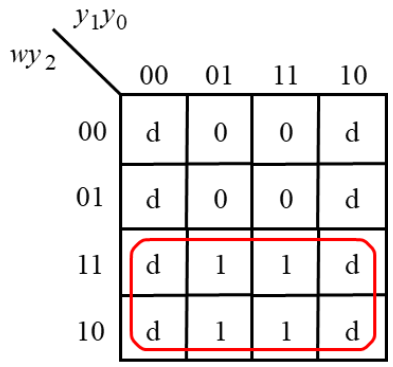
wy_2 \ y_1y_0	00	01	11	10
00	0	0	d	d
01	0	0	d	d
11	0	1	d	d
10	0	1	d	d

$J_1 = wy_0$



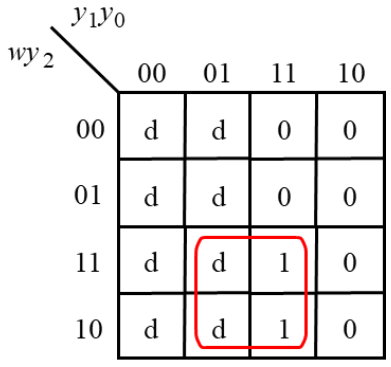
wy_2 \ y_1y_0	00	01	11	10
00	0	0	0	0
01	d	d	d	d
11	d	d	d	d
10	0	0	1	0

$J_2 = wy_0y_1$



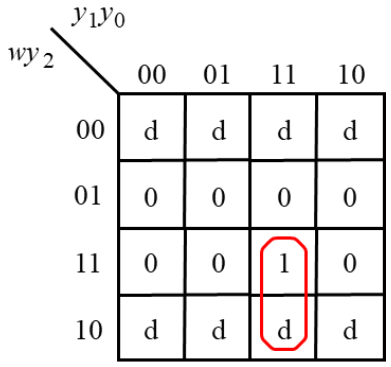
wy_2 \ y_1y_0	00	01	11	10
00	d	0	0	d
01	d	0	0	d
11	d	1	1	d
10	d	1	1	d

$K_0 = w$



wy_2 \ y_1y_0	00	01	11	10
00	d	d	0	0
01	d	d	0	0
11	d	d	1	0
10	d	d	1	0

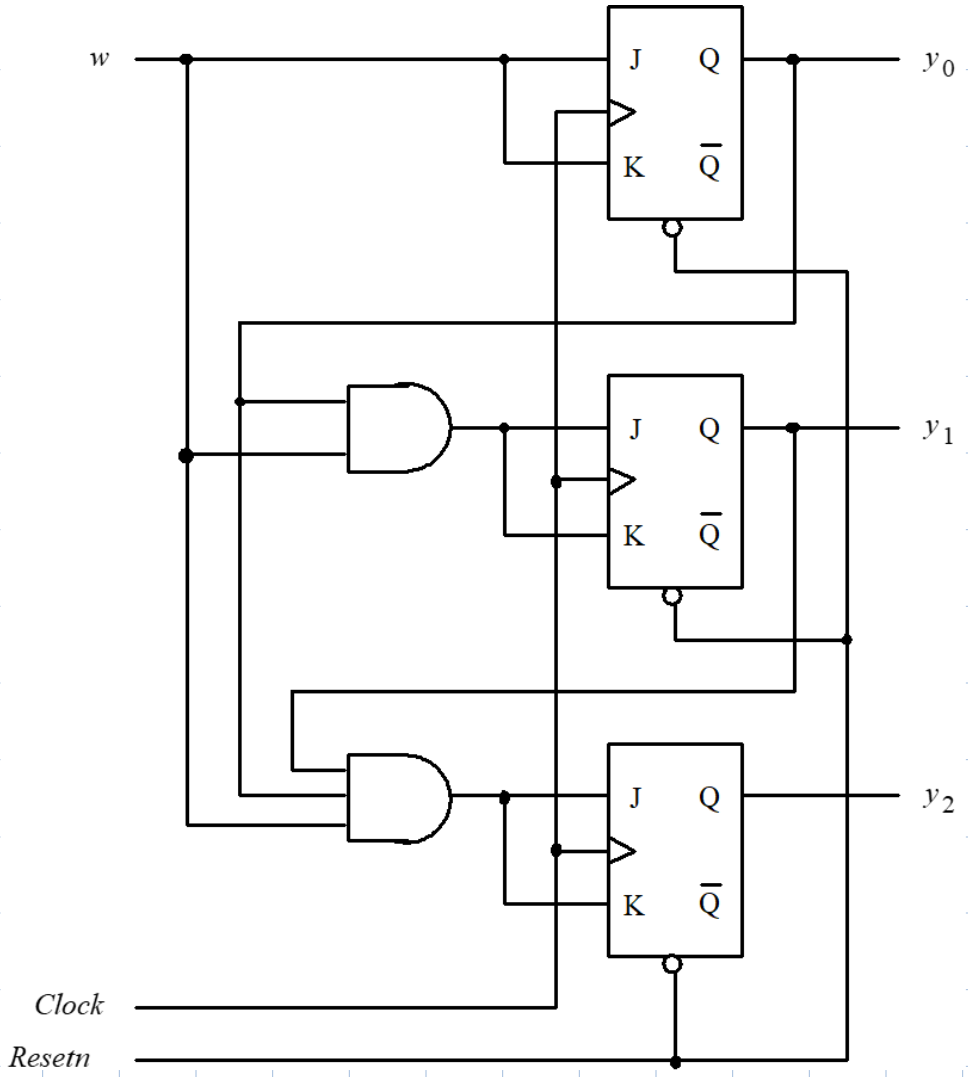
$K_1 = wy_0$



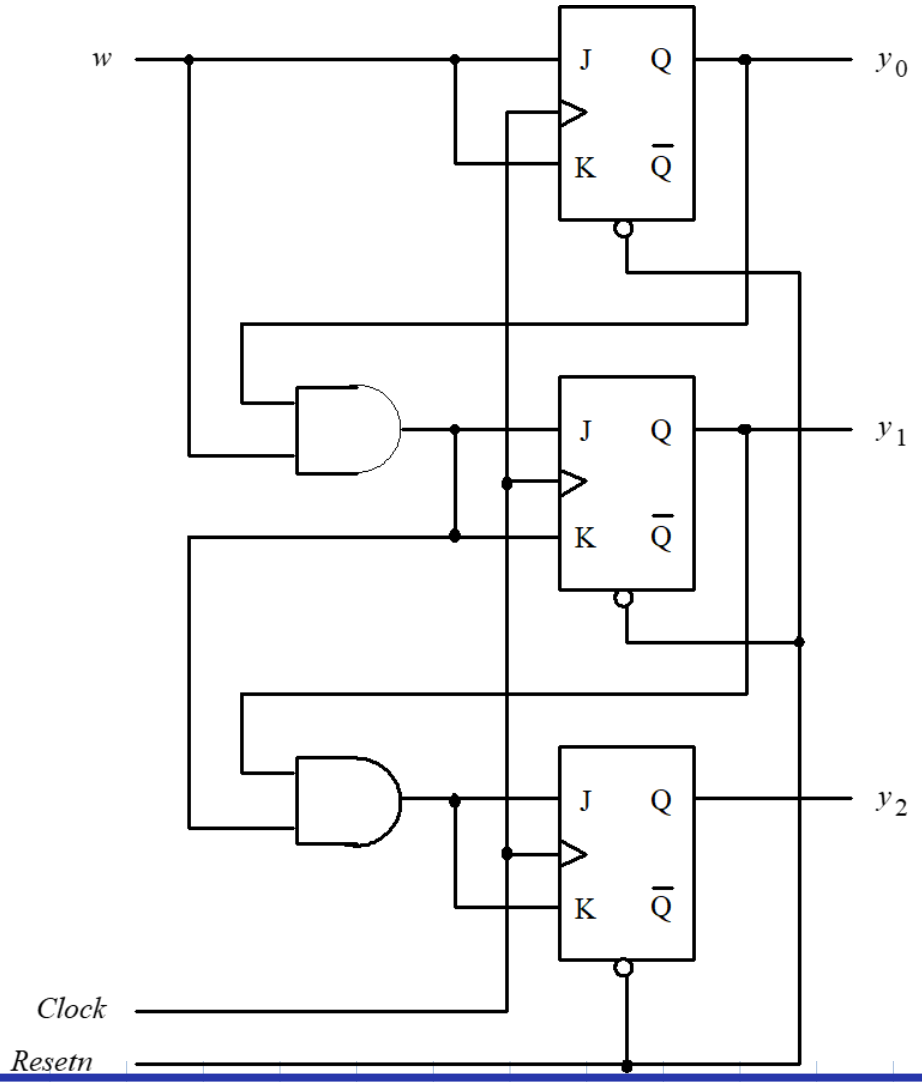
wy_2 \ y_1y_0	00	01	11	10
00	d	d	d	d
01	0	0	0	0
11	0	0	1	0
10	d	d	d	d

$K_2 = wy_0y_1$

8.7.4 Implementación con flip-flops JK



8.7.4 Implementación con flip-flops JK Factorizada



- Al igual que con flip-flops D podemos obtener una versión factorizada según las siguientes ecuaciones

$$J_2 = K_2 = (wy_0)y_1 \qquad = J_1 y_1$$

$$J_n = K_n = (wy_0 \cdots y_{n-2})y_{n-1} = J_{n-1} y_{n-1}$$

- Recordar que cuando las entradas J y K de un flip-flop están conectadas, éste se comporta como un flip-flop T



8.7.5 – 8.12

Ejemplos

angel@uc.cl

IEE2713 Sistemas Digitales

8.7.5 Ejemplo: un contador diferente

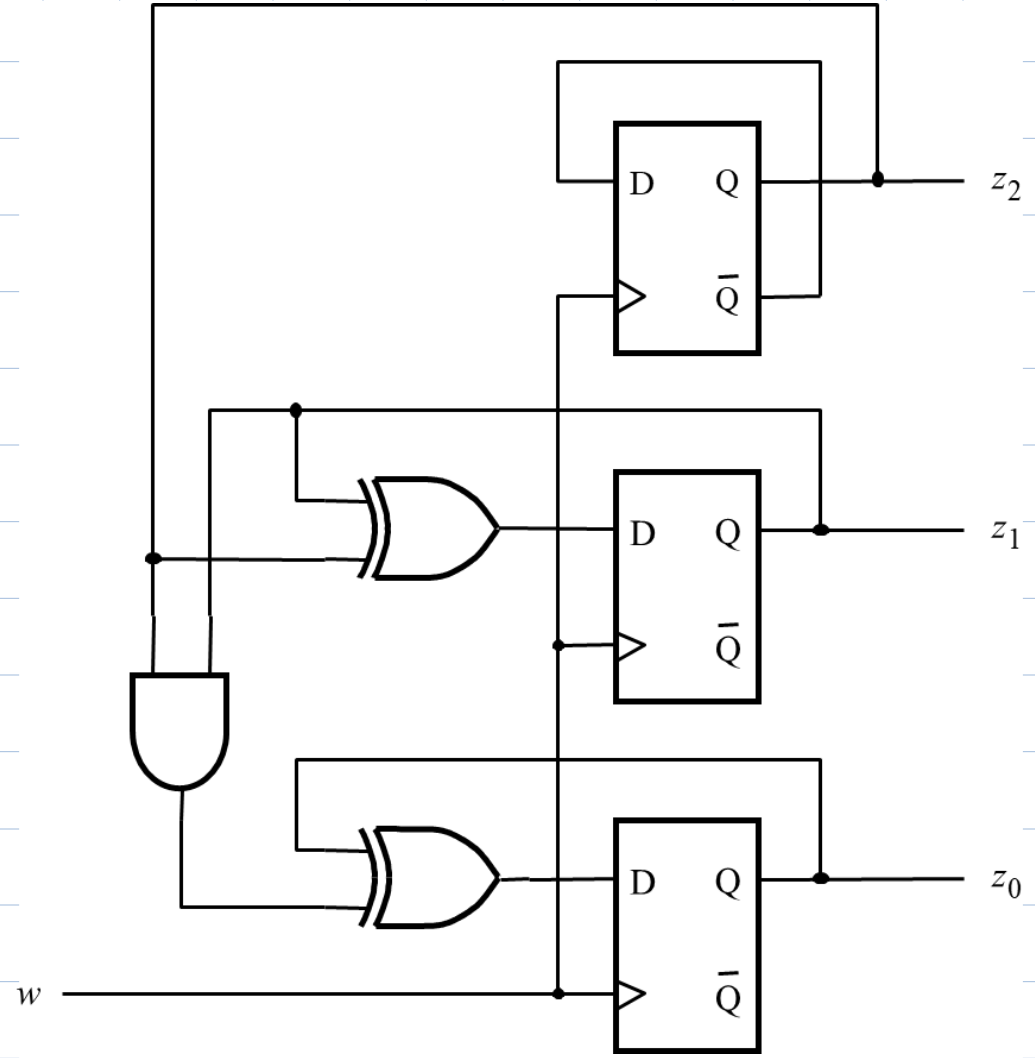
- Consideremos un contador que cuenta pulsos en entrada w, según la secuencia
  - 0, 4, 2, 6, 1, 5, 3, 7, 0, 4, …
  - Como queremos contar la cantidad de pulsos w, tiene sentido usarlo como clock, y el enable siempre en 1
- Usando las tablas, llegamos a las siguientes ecuaciones para el siguiente estado:

$$
\begin{aligned}
D_2 &= Y_2 = \bar{y}_2 \\
D_1 &= Y_1 = y_1 \oplus y_2 \\
D_0 &= Y_0 = y_0\bar{y}_1 + y_0\bar{y}_2 + \bar{y}_0 y_1 y_2 \\
&= y_0(\bar{y}_1 + \bar{y}_2) + \bar{y}_0 y_1 y_2 \\
&= y_0 \oplus y_1 y_2
\end{aligned}
$$

Present state	Next state	Output $z_2z_1z_0$
A	B	000
B	C	100
C	D	010
D	E	110
E	F	001
F	G	101
G	H	011
H	A	111

Present state $y_2y_1y_0$	Next state $Y_2Y_1Y_0$	Output $z_2z_1z_0$
000	100	000
100	010	100
010	110	010
110	001	110
001	101	001
101	011	101
011	111	011
111	000	111

8.7.5 Ejemplo: un contador diferente



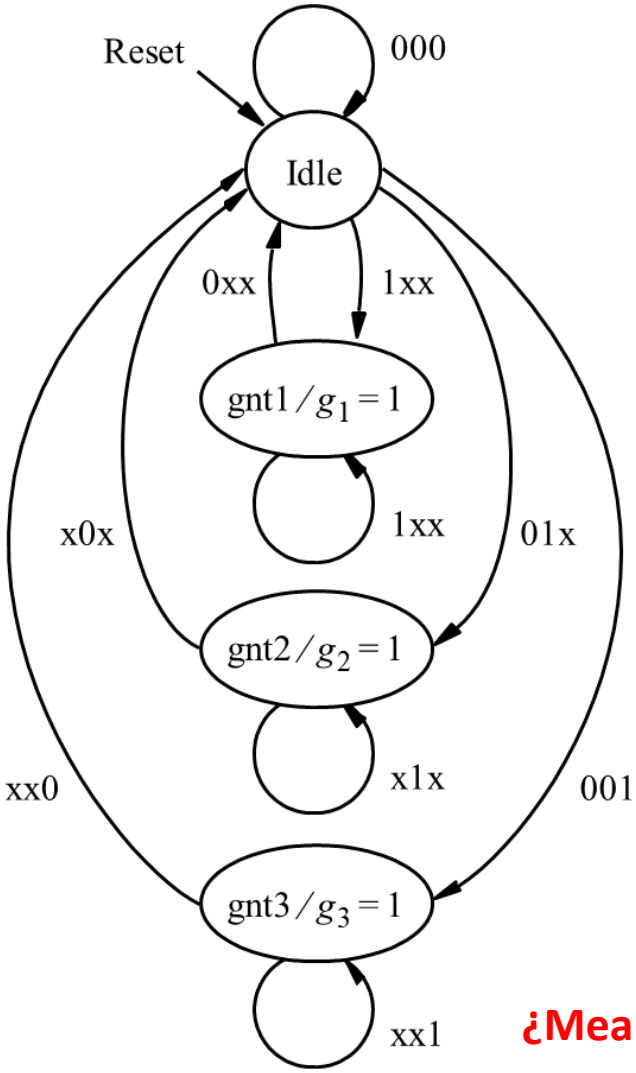
Circuito resultante

- Este circuito cuenta en orden incorrecto porque expresa la cuenta desde LSB hasta MSB (al revés de lo usual)
- Por eso es similar a uno de los contadores vistos en el capítulo anterior, pero con el orden de los bits de salida alterado

8.8 Una FSM como circuito arbitrador

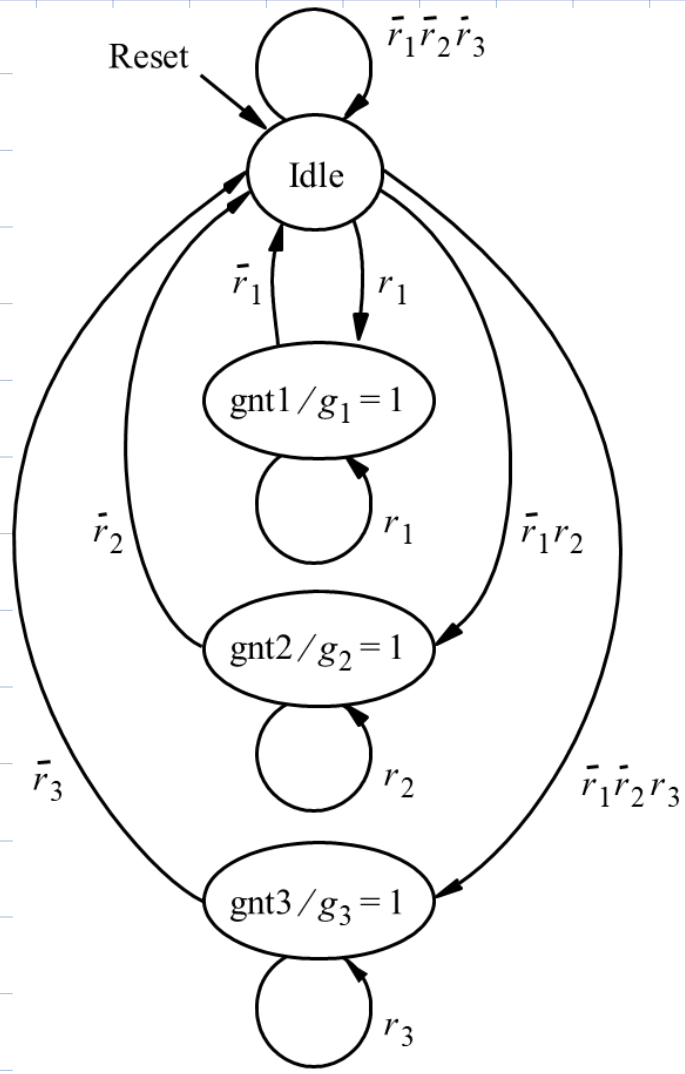
- Vamos a diseñar un circuito que administra el acceso a un recurso que puede ser solicitado por **tres** dispositivos
- Supuestos:
  - Cada dispositivo indica con una señal cuándo quiere utilizar el recurso
  - Se concede la solicitud siempre y cuando el recurso no esté siendo utilizado por otro dispositivo
  - El uso del recurso por un dispositivo se mantiene mientras su señal se mantenga activa
- Vamos a llamar a las entradas **requests** (r1, r2, r3), y a las salidas, **grants** (g1, g2, g3)
- El dispositivo 1 tiene la mayor prioridad en caso de hacer solicitudes al mismo tiempo

8.8 Una FSM como circuito arbitrador



¿Mealy o Moore?

8.8 Una FSM como circuito arbitrador



Este estilo de etiquetado es más intuitivo de entender y nos lleva más rápidamente a expresiones de álgebra booleana

8.8.1 Implementación del circuito arbitrador

```
module arbiter (r, Resetn, Clock, g);
    input [1:3] r;
    input Resetn, Clock;
    output wire [1:3] g;
    reg [2:1] y, Y;
    parameter Idle = 2'b00, gnt1 = 2'b01, gnt2 = 2'b10, gnt3 = 2'b11;

    // Next state combinational circuit
    always @(r, y)
        case (y)
            Idle: casex (r)
                        3'b000: Y = Idle;
                        3'b1xx: Y = gnt1;
                        3'b01x: Y = gnt2;
                        3'b001: Y = gnt3;
                        default: Y = Idle;
                    endcase
            gnt1: if (r[1])  Y = gnt1;
                    else  Y = Idle;
            gnt2: if (r[2])  Y = gnt2;
                    else  Y = Idle;
            gnt3: if (r[3])  Y = gnt3;
                    else  Y = Idle;
            default: Y = Idle;
        endcase

    // Sequential block
    always @(posedge Clock)
        if (Resetn == 0)    y <= Idle;
        else     y <= Y;

    //  Define output
    assign g[1] = (y == gnt1);
    assign g[2] = (y == gnt2);
    assign g[3] = (y == gnt3);

endmodule
```

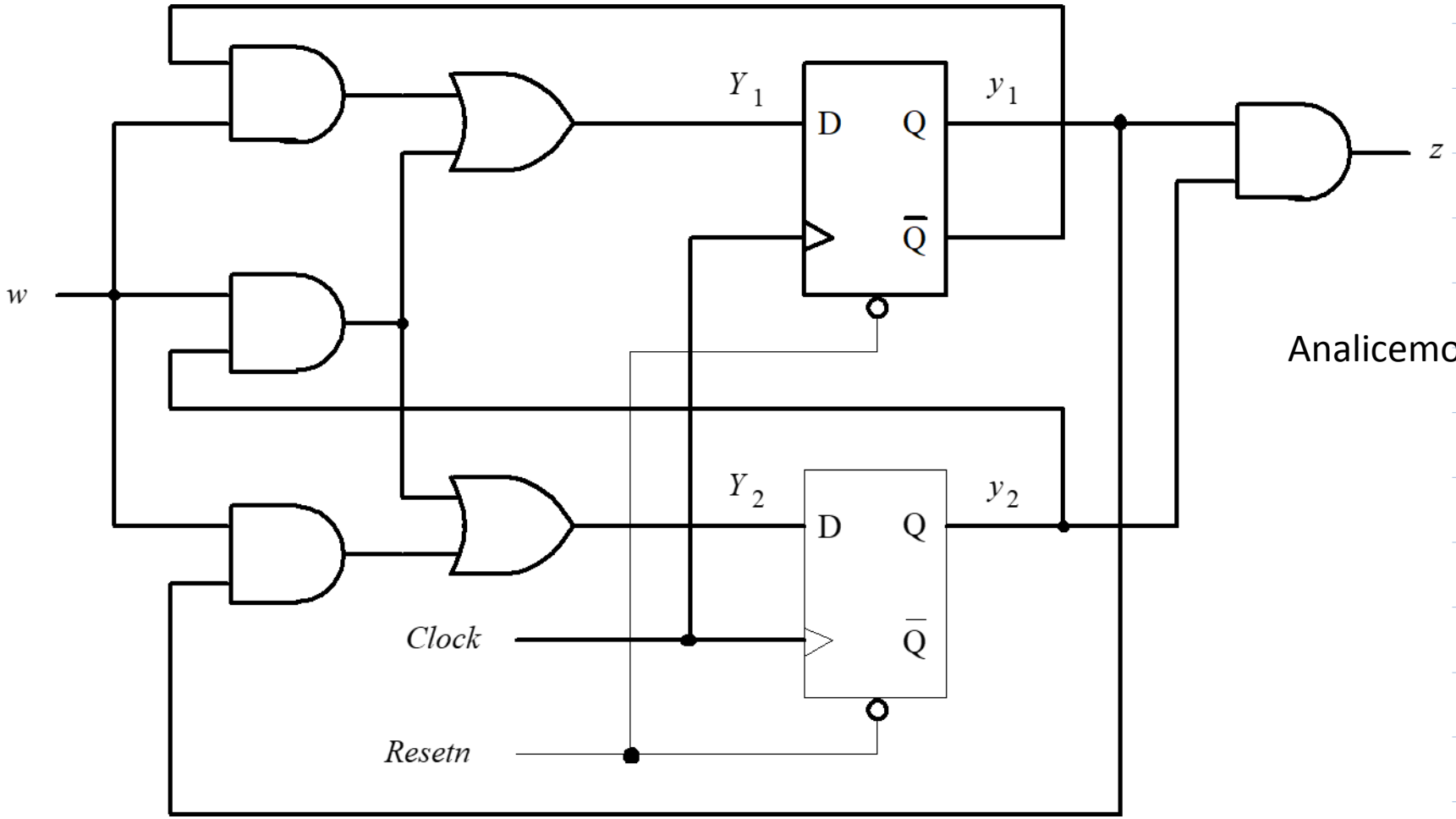
8.8.2 Minimización del retardo de salida de una FSM

- Una FSM puede tener diferentes retardos dependiendo de su implementación
  - Por ejemplo, una implementación one-hot del arbitrador requiere más flip-flops, pero menos lógica combinacional después de los flip-flops
  - Esto hace que una implementación one-hot pueda ser **más rápida** que una implementación codificada en forma binaria
- Los diseñadores de circuitos secuenciales realizan este tipo de optimización para cumplir con las especificaciones de diseño

8.9 Análisis de circuitos secuenciales síncronos

- Además de diseñar circuitos síncronos, como en los ejemplos anteriores, también es importante aprender a analizar circuitos existentes
  - Esto es mucho más sencillo que el proceso de diseño
- La manera más fácil es realizar el proceso inverso al diseño
  - Las salidas de los flip-flops representan las variables del estado **actual**
  - Las entradas determinan el estado **siguiente**
- Usando esta información podemos construir la tabla de estados
  - Luego, a partir de la tabla llegamos al diagrama de estados, lo que nos permite conocer el comportamiento complete del ciruito

8.9 Análisis de circuitos secuenciales síncronos



Analicemos este ejemplo

8.9 Análisis de circuitos secuenciales síncronos

$$Y_1 = w\bar{y}_1 + wy_2$$
$$Y_2 = wy_1 + wy_2$$
$$z = y_1 y_2$$

- Como hay dos flip-flops, tenemos como máximo 4 estados en la FSM
- Es buena idea es asumir un estado inicial para los flip-flops $y_1 = y_2 = 0$
- De las ecuaciones derivamos las siguientes tablas
  - La interpretación de éstas es el de un circuito que "detecta" un patrón de tres 1s consecutivos, ya que al tercer 1, la salida se vuelve z = 1 y no se reinicia hasta que la entrada se vuelva cero

Present state y_2y_1	Next State Y_2Y_1 w = 0	Next State Y_2Y_1 w = 1	Output z
0 0	0 0	01	0
0 1	0 0	10	0
1 0	0 0	11	0
1 1	0 0	11	1

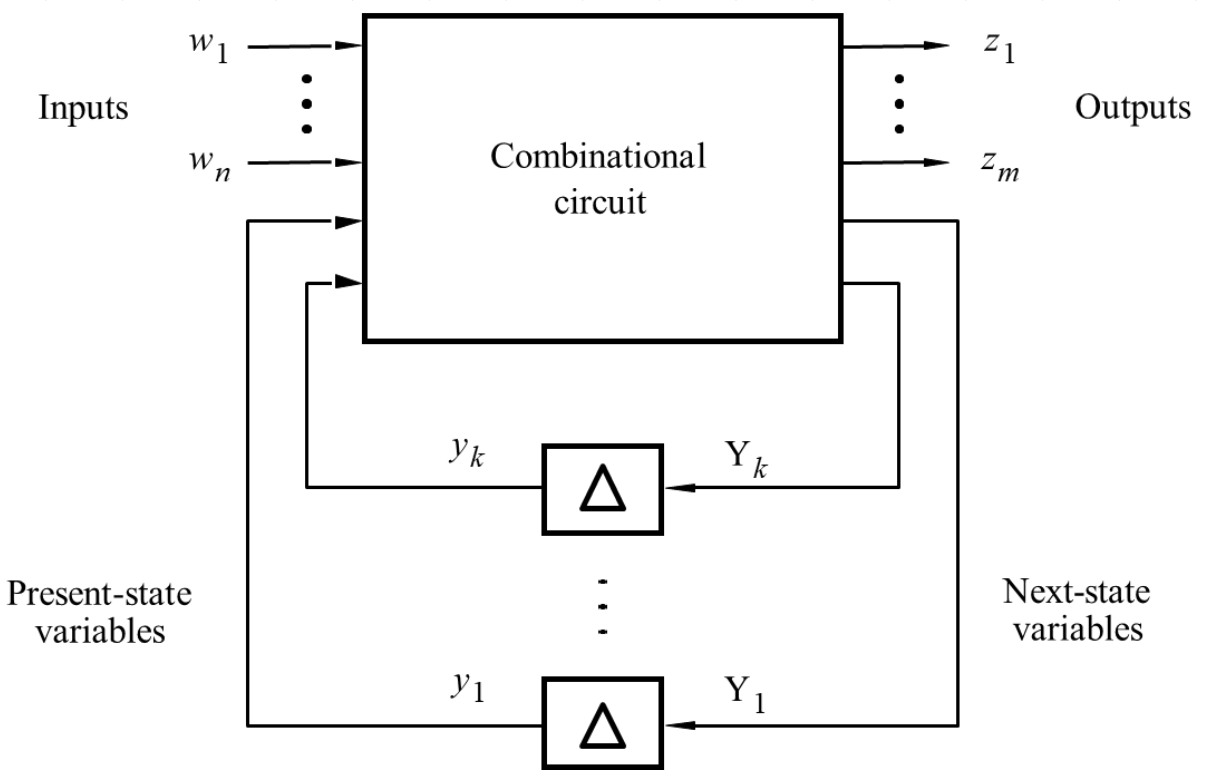
Present state	Next state w = 0	Next state w = 1	Output z
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

8.11 Modelo formal de circuitos secuenciales

- Por el momento hemos hecho un análisis descriptivo y más intuitivo
- Sin embargo, también es posible realizar un análisis formal
- Consideremos:
  - N entradas: $W = \{w_1, w_2, \ldots, w_n\}$
  - M salidas: $Z = \{z_1, z_2, \ldots, z_m\}$
  - K Variables del estado actual: $y = \{y_1, y_2, \ldots, y_k\}$
  - K Variables del estado siguiente: $Y = \{Y_1, Y_2, \ldots, Y_k\}$
  - 2^k Estados: $S = \{S_1, S_2, \ldots, S_{2^k}\}$

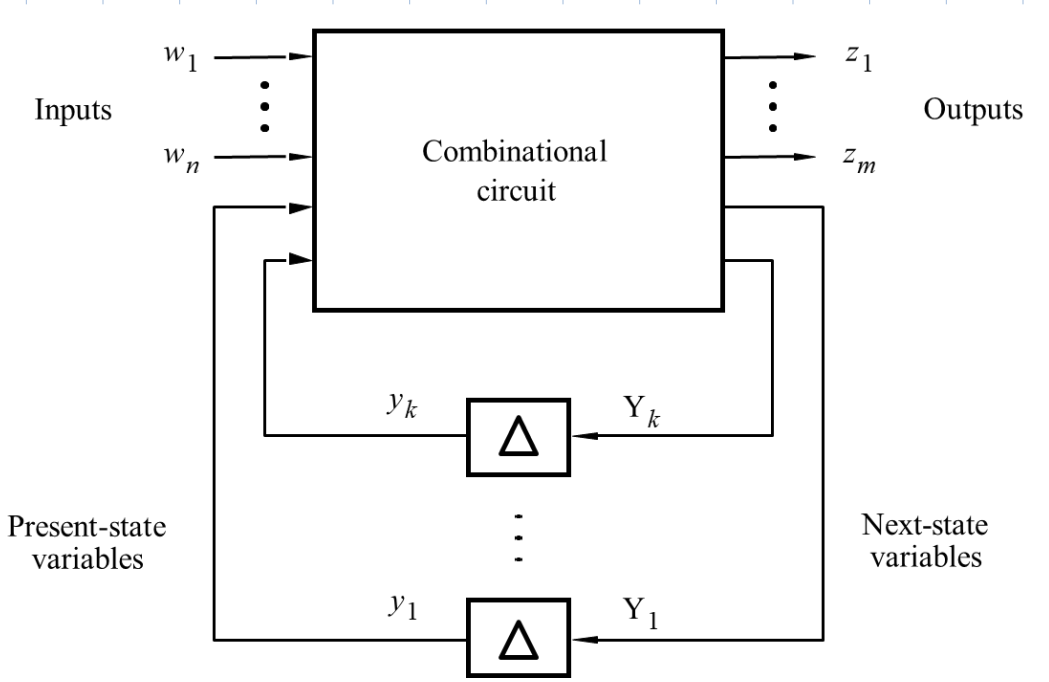
8.11 Modelo formal de circuitos secuenciales

- Un circuito secuencial se puede representar de manera general según la imagen
- Los elementos Δ representan un retardo (delay) que asegura que *y* tomará los valores de *Y* luego de un retardo Δ
- En circuitos secuenciales estos retardos corresponden a los flip-flops

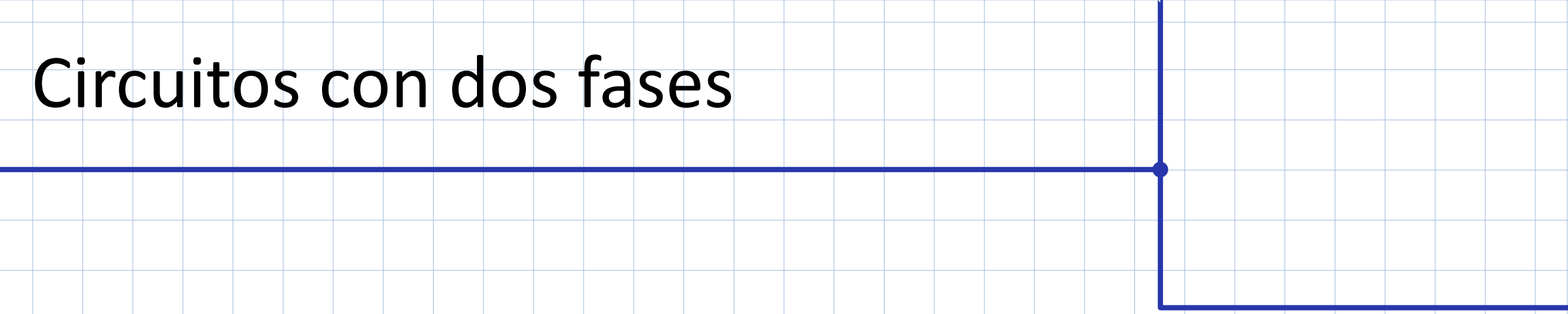


8.11 Modelo formal de circuitos secuenciales

- Un circuito secuencial M se puede definir formalmente como:
  - M = (W, Z, S, φ, λ)
- En donde:
  - W, Z, S son conjuntos finitos, no vacíos de entradas, salidas y estados, respectivamente
  - φ es la función de transición de estados: S(t+1) = φ[W(t), S(t)]
  - λ es la función de los outputs.
  - Para una máquina de Moore: λ(t) = λ[S(t)]
  - Para una máquina de Mealy: λ(t) = λ[W(t), S(t)]
  - Para nuestro caso asumimos que el tiempo entre t y t+1 es un ciclo de clock



Circuitos con dos fases



8.12 Resumen

- La existencia de loops cerrados y delays en un circuito secuencial permite un comportamiento caracterizado por un conjunto de estados
- Los valores presentes de las entradas ya no son el único factor determinante de este comportamiento, ya que una secuencia diferente de entradas puede causar un comportamiento diferente al pasar por distintos estados
- Los retardos de propagación son importantes de considerar
- El proceso de diseño asume que el período de reloj es más largo que el tiempo de propagación más largo del circuito para que éste funcione
- Estos circuitos síncronos son **muy** utilizados en diseños prácticos y son diseñados con herramientas CAD