Análisis y Síntesis

Introducción a los Sistemas Lógicos y Digitales

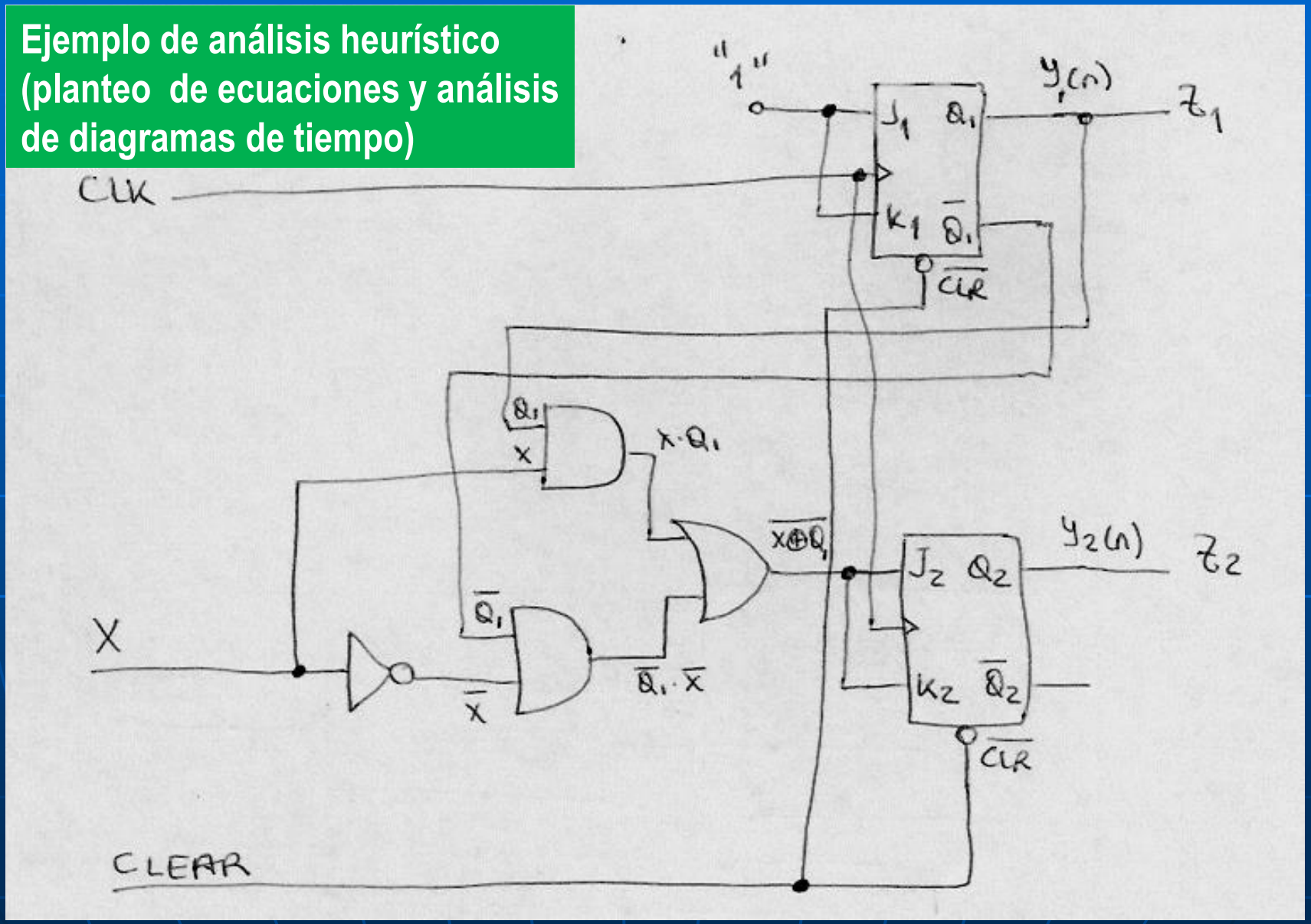
2016

Análisis y Síntesis

Métodos de análisis:

- Tabla de verdad.
- Heurístico.
- Diagramas de estado.
- Simulación.
- Test del hardware.
- etc.....

Ejemplo de análisis heurístico (planteo de ecuaciones y análisis de diagramas de tiempo)



Ecuaciones:

$J_1 = K_1 = 1$ $\quad J_2 = K_2 = x \cdot Q_1 + \overline{x} \cdot \overline{Q_1} = \overline{x \oplus Q_1}$

$x \cdot y_1(n) + \overline{x} \cdot \overline{y_1(n)} = \overline{x \oplus y_1(n)}$

Ecuación general del flip-flop JK: $y(n+1) = J \cdot \overline{y(n)} + \overline{K} \cdot y(n)$

$\left.\begin{matrix} x=0 \\ x=1 \end{matrix}\right\} y_1(n+1) = 1 \cdot \overline{y_1(n)} + 0 \cdot y_1(n) = \overline{y_1(n)}$

$x = 0 \Rightarrow J_2 = K_2 = \overline{0 \oplus Q_1} = \overline{Q_1} \Rightarrow y_2(n+1) = \overline{y_1(n)} \cdot \overline{y_2(n)} +$

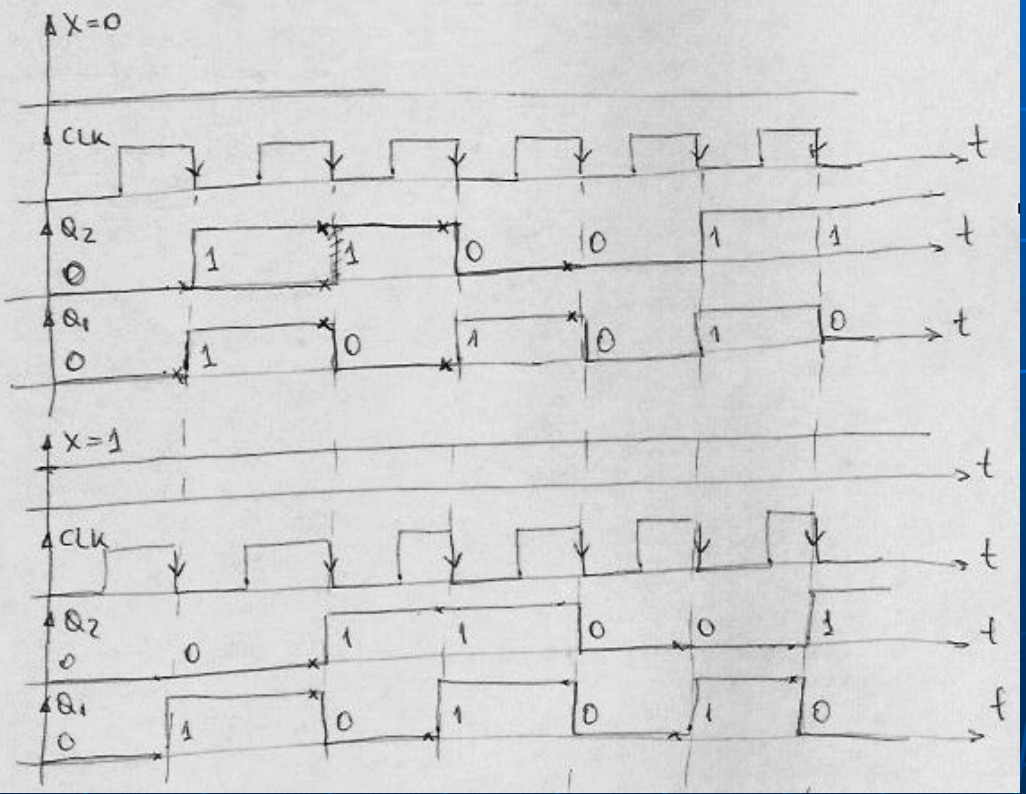
$y_2(n+1) = \overline{y_1(n) \oplus y_2(n)}$ $\qquad y_1(n) \cdot y_2(n)$

$x = 1 \Rightarrow J_2 = K_2 = \overline{1 \oplus Q_1} = Q_1 \Rightarrow y_2(n+1) = y_1(n) \cdot \overline{y_2(n)} +$

$y_2(n+1) = y_1(n) \oplus y_2(n)$ $\qquad \overline{y_1(n)} \cdot y_2(n)$

$$X=0 \begin{cases} y_1(n+1) = \overline{y_1(n)} \\ y_2(n+1) = \overline{y_1(n) \oplus y_2(n)} \end{cases}$$

$$X=1 \begin{cases} y_1(n+1) = \overline{y_1(n)} \\ y_2(n+1) = y_1(n) \oplus y_2(n) \end{cases}$$

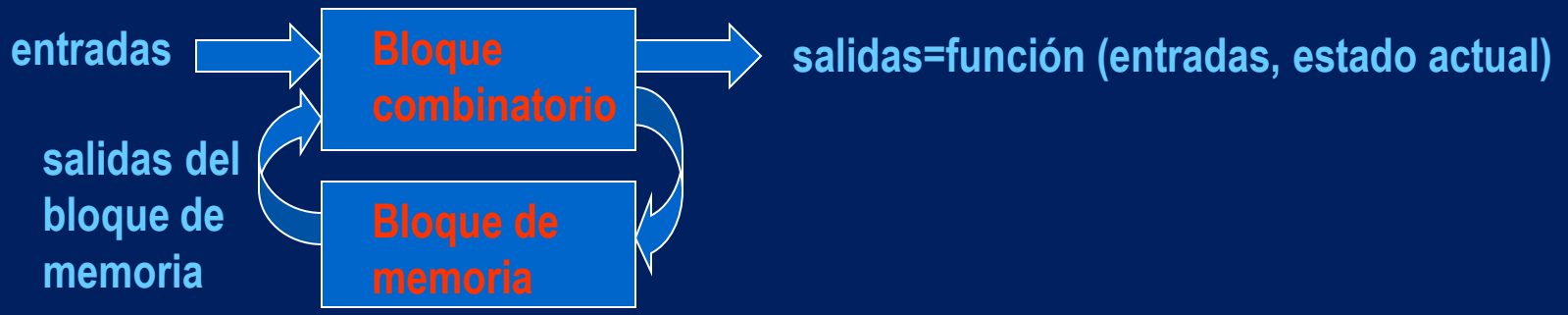


Suponiendo que la entrada X se mantiene constante, se observa que las salidas cambian de tal forma que si X=0, las mismas evolucionan como un contador binario regresivo.
Si en cambio, X=1, el modo de conteo es progresivo.

Máquinas de estado

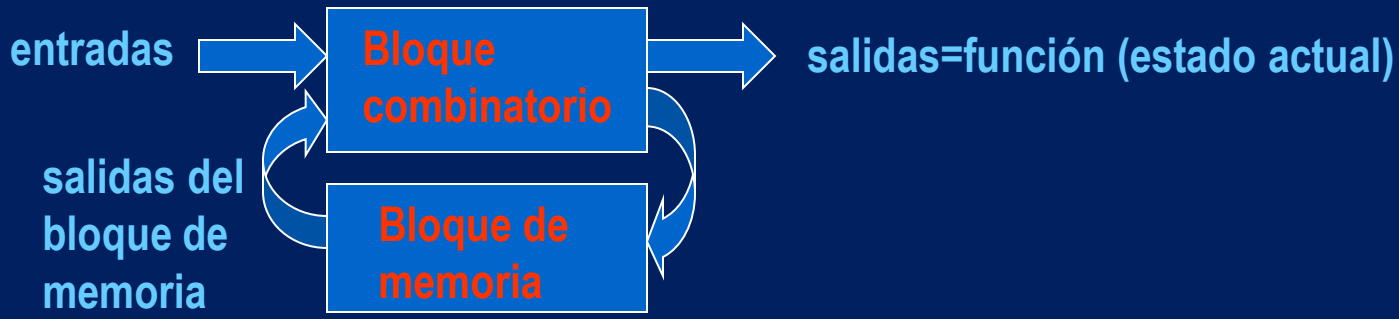
Modelo de Mealy

La(s) salida(s) depende(n) de la(s) entrada(s) y del estado actual.



Modelo de Moore

La(s) salida(s) depende(n) del del estado actual.



Se define estado a una dada combinación entre las salidas de los elementos de memoria (FFs) que conforman el circuito.

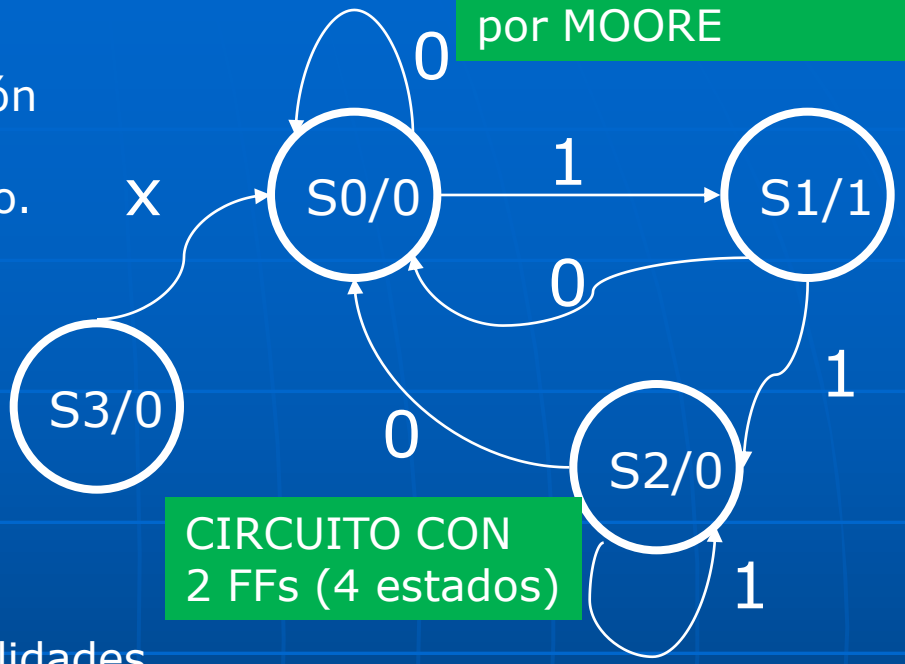
En Moore una salida NO puede cambiar su valor mientras esté en un dado estado.

En Mealy, en cambio, puede hacerlo.

En estos dos ejemplos hay una entrada entonces de cada estado hay dos posibilidades de acción.
Si hay dos entradas serán 4 y así sucesivamente.

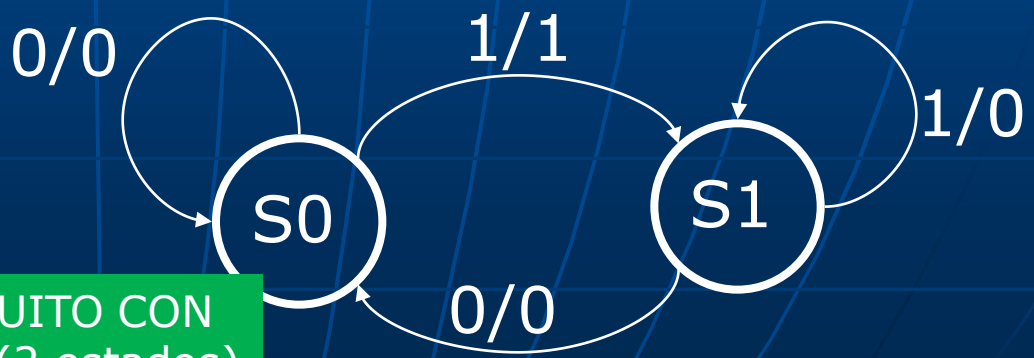
La referencia temporal en estos diagramas está implícita:
Para pasar de un estado a otro DEBE llegar un flanco de RELOJ.

Modelo de descripción por MOORE



CIRCUITO CON
2 FFs (4 estados)

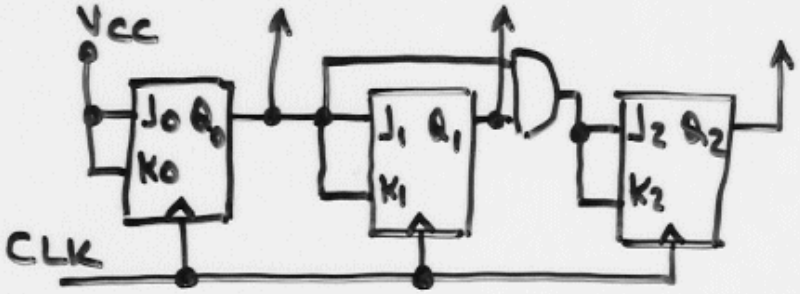
Modelo de descripción por MEALY



CIRCUITO CON
1 FF (2 estados)

Ejemplo de análisis por diagrama de estado

Analizar el siguiente circuito sincrónico



Ecuación de un FF tipo JK: $Q^{n+1} = \overline{Q^n} \cdot J + Q^n \cdot \overline{K}$

FF_0: $Q_0^{n+1} = \overline{Q_0^n} \cdot J_0 + Q_0^n \cdot \overline{K_0} = \overline{Q_0^n} \cdot J_0 + Q_0^n \cdot \overline{J_0} = Q_0^n \oplus J_0$

$J_0 = K_0 = 1 \Rightarrow Q_0^{n+1} = \overline{Q_0^n}$

FF_1: $Q_1^{n+1} = \overline{Q_1^n} \cdot J_1 + Q_1^n \cdot \overline{K_1} = \overline{Q_1^n} \cdot J_1 + Q_1^n \cdot \overline{J_1} = Q_1^n \oplus J_1$

$J_1 = K_1 = Q_0^n \Rightarrow Q_1^{n+1} = Q_1^n \oplus Q_0^n$

Si $Q_0^n = 0 \rightarrow Q_1^{n+1} = Q_1^n$

Si $Q_0^n = 1 \rightarrow Q_1^{n+1} = \overline{Q_1^n}$

$FF_2 : Q_2^{n+1} = \overline{Q_2^n} \cdot J_2 + Q_2^n \cdot \overline{K_2} = \overline{Q_2^n} \cdot J_2 + Q_2^n \cdot \overline{J_2} = Q_2^n \oplus J_2$

$J_2 = K_2 = Q_0^n \cdot Q_1^n$

$Q_2^{n+1} = Q_2^n \oplus Q_0^n \cdot Q_1^n \Rightarrow$

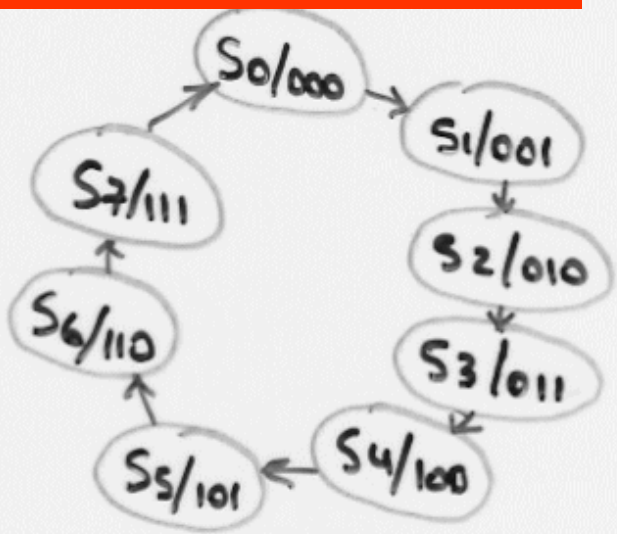
Si $Q_1^n ; Q_0^n \neq 11 \rightarrow Q_2^{n+1} = Q_2^n$

si $Q_1^n ; Q_0^n = 11 \rightarrow Q_2^{n+1} = \overline{Q_2^n}$

Tabla de estados

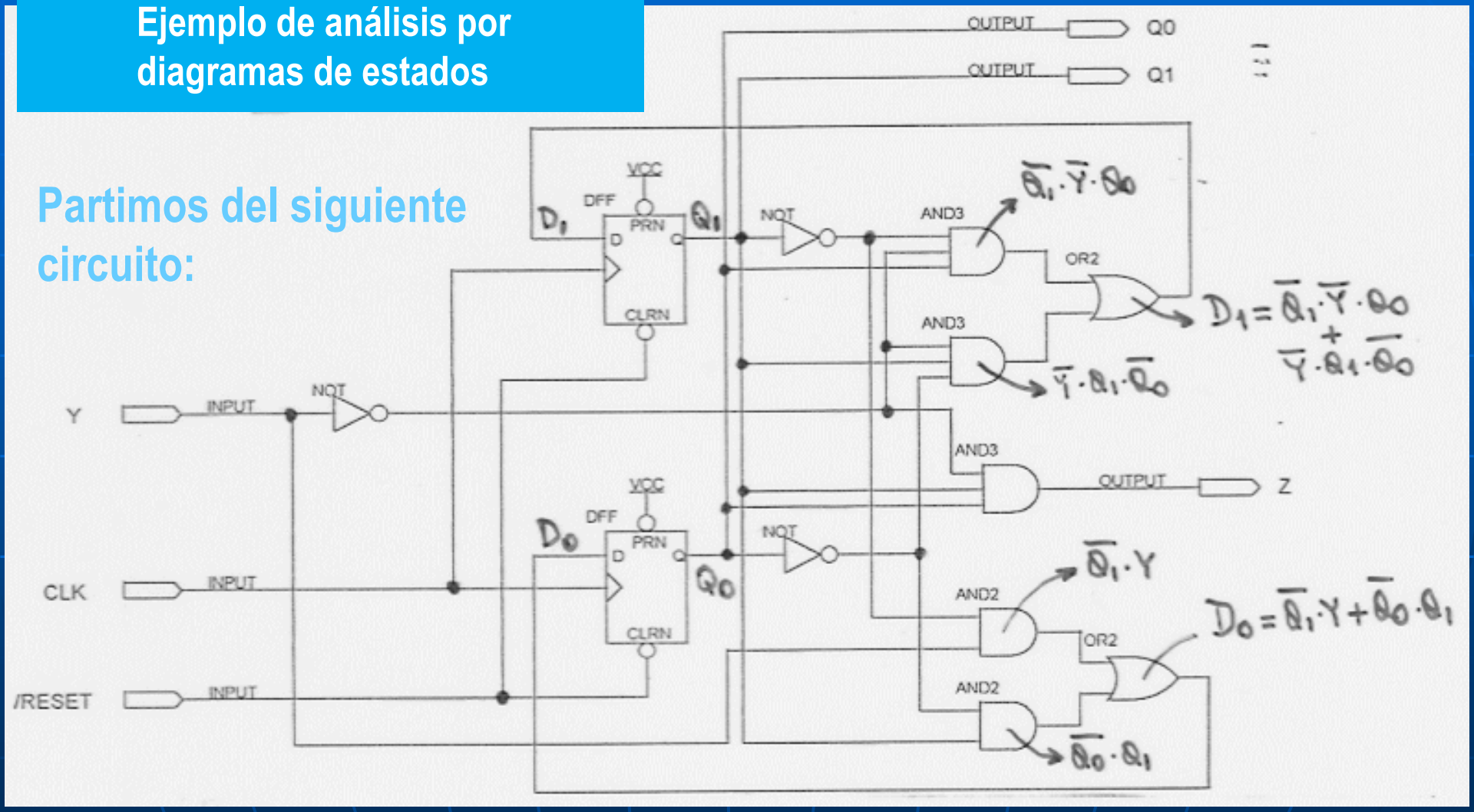
	Q_2^n	Q_1^n	Q_0^n	Q_2^{n+1}	Q_1^{n+1}	Q_0^{n+1}
S_0	0	0	0	0	0	1
S_1	0	0	1	0	1	0
S_2	0	1	0	0	1	1
S_3	0	1	1	1	0	0
S_4	1	0	0	1	0	1
S_5	1	0	1	1	1	0
S_6	1	1	0	1	1	1
S_7	1	1	1	0	0	0

El circuito es un contador binario progresivo de 3 bits



Ejemplo de análisis por diagramas de estados

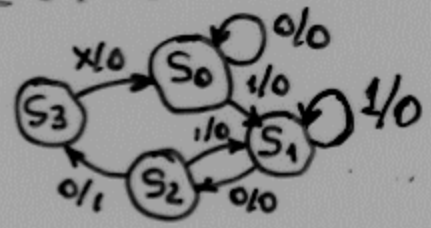
Partimos del siguiente circuito:



$Z = \overline{Y} \cdot Q_1 \cdot Q_0$; $D_0 = \overline{Q_1} \cdot Y + \overline{Q_0} \cdot Q_1$; $D_1 = \overline{Q_1} \cdot \overline{Y} \cdot Q_0 + \overline{Y} \, Q_1 \overline{Q_0} = \overline{Y} \cdot (Q_1 \oplus Q_0)$

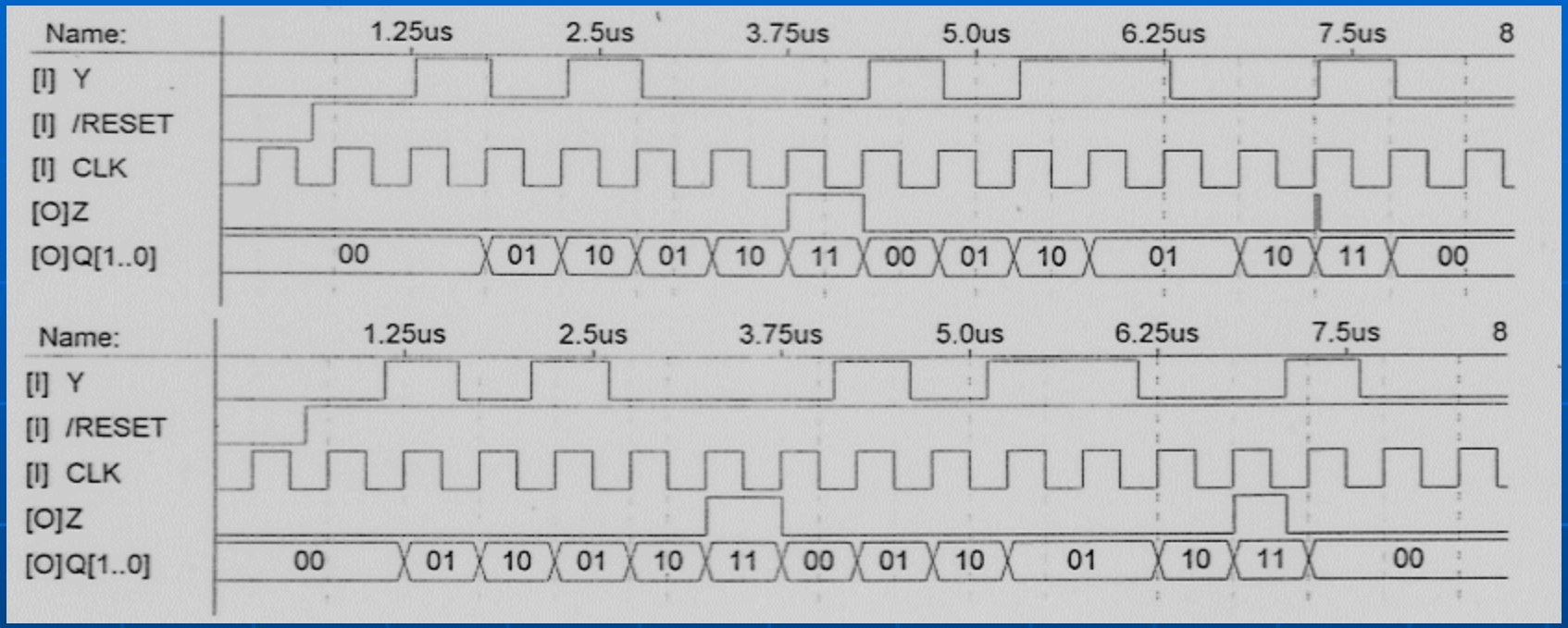
$y=0 \Rightarrow \begin{cases} D_0 = \overline{Q_0} \cdot Q_1 \\ D_1 = Q_0 \oplus Q_1 \end{cases}$; $y=1 \Rightarrow \begin{cases} D_0 = \overline{Q_1} + \overline{Q_0} \cdot Q_1 \\ D_1 = 0 \end{cases}$

	Q_1	Q_0	D_1	D_0	D_1	D_0	Z	
S_0	0	0	0	0	0	1	0	0
S_1	0	1	1	0	0	1	0	0
S_2	1	0	1	1	0	1	0	0
S_3	1	1	0	0	0	0	1	0
			$y=0$		$y=1$		0	1



Este circuito puede ser un detector de secuencia cíclico de una entrada serie tal que si la secuencia es "100" sacará un 1 por la salida, caso contrario quedará siempre en 0.
Además, el siguiente bit después de detectar la secuencia correcta es descartado.....

Simulación del circuito anterior



En este ejemplo, se realiza una posterior simulación por software con entrada por esquemático (MaxPlus-II) con lo cual puede ayudar a estudiar el comportamiento del circuito ya sea para estudiar un circuito desconocido como también para verificar el comportamiento de uno que se está diseñando.

Análisis y Síntesis

Métodos de síntesis:

Heurístico basado en:

- Tabla de verdad.
- Ecuaciones lógicas.
- Diagramas de Karnaugh.

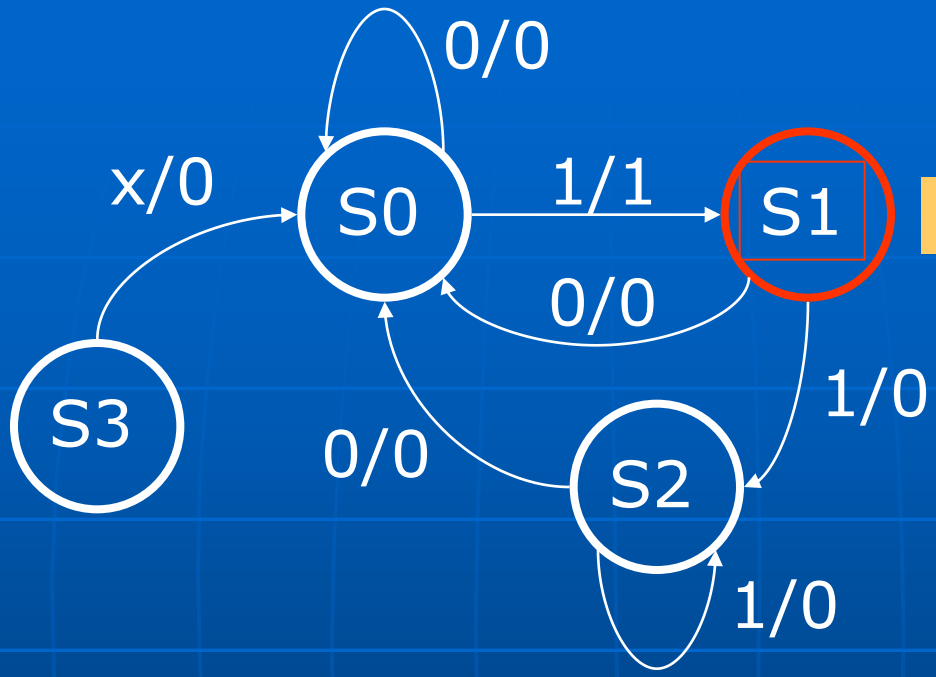
Diagramas de estados. → Síntesis con lógica standard.
Síntesis por descripción en HDL.

Algoritmos de síntesis.

Diseño basado en Lenguaje de Descripción del Hardware (HDL)

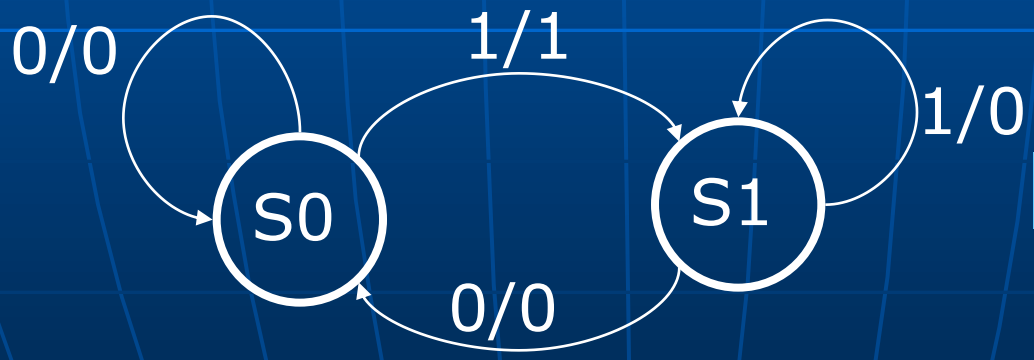
etc.....

REDUNDANCIA



ESTADO REDUNDANTE

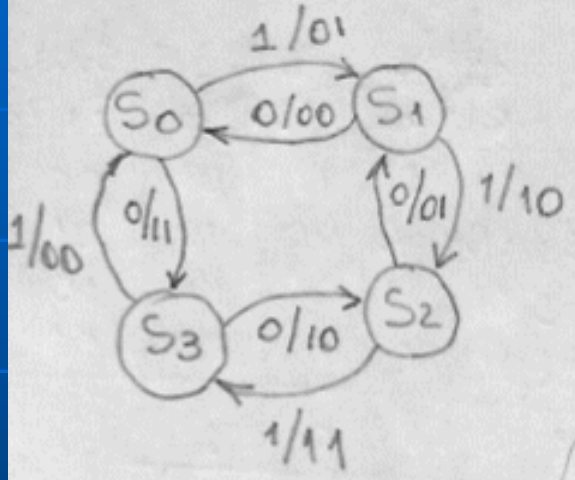
SE NECESITAN 2 FF



SE NECESITA 1 FF

Contador binario progresivo-regresivo de 2 bits

Contador binario de 2 bits progresivo - regresivo MEALY



			Z_1	Z_0		
E.A	E.S		Q_1	Q_0	D_1	D_0
00≡S0	S_3	S_1	11	01	11	01
01≡S1	S_0	S_2	00	10	00	10
10≡S2	S_1	S_3	01	11	01	11
11≡S3	S_2	S_0	10	00	10	00
	0	1	0	1	0	1
	X		X		X	

4 estados ⇒ 2 FF's

COINCIDEN SALIDAS CON ESTADOS PARA ESTE CASO PARTICULAR

Tablas de transición para generar el siguiente estado

TABLA DE TRANSICION
FF "D"

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

TABLA DE TRANSICION
FF "JK"

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Tablas de transición para generar el siguiente estado

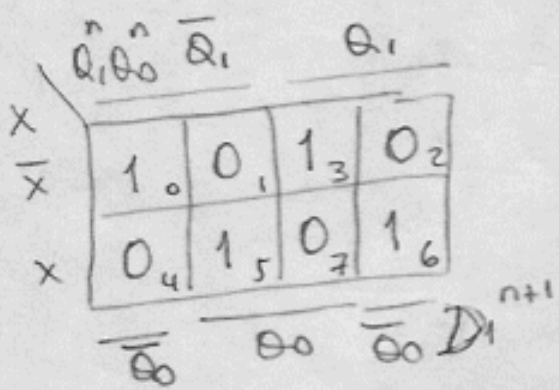
TABLA DE TRANSICION
FF "D"

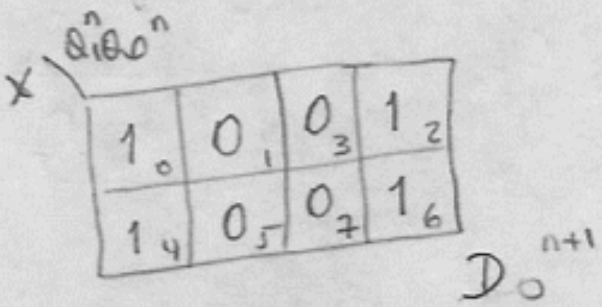
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

TABLA DE TRANSICION
FF "JK"

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Método para lógica standard





$$D_0 = \overline{Q}_0$$

$$D_1 = \overline{x}\cdot\overline{Q}_1\overline{Q}_0 + \overline{x}\cdot Q_1Q_0 + x\cdot\overline{Q}_1Q_0 + x\cdot Q_1\overline{Q}_0$$

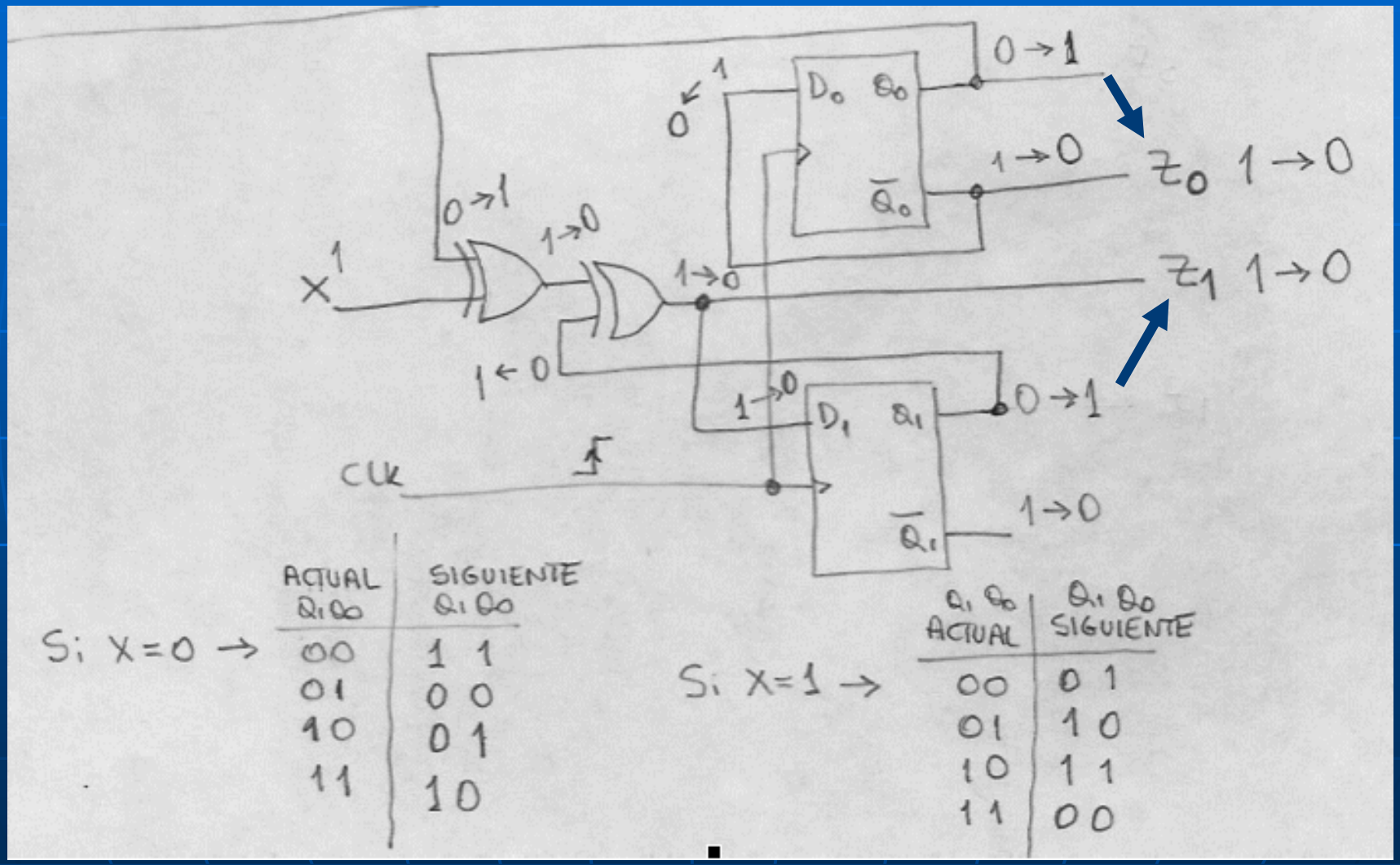
$$D_1 = \overline{x}\cdot\underbrace{(\overline{Q}_1\overline{Q}_0 + Q_1Q_0)}_{Q_1\odot Q_0} + x\cdot\underbrace{(\overline{Q}_1Q_0 + Q_1\overline{Q}_0)}_{Q_1\oplus Q_0} = \overline{x}\cdot\overline{Q_1\oplus Q_0} + x\cdot(Q_1\oplus Q_0)$$

$$\overline{D_1} = x\cdot\underbrace{(\overline{Q}_1\overline{Q}_0 + Q_1Q_0)}_{\overline{Q_1\oplus Q_0}} + \overline{x}\cdot\underbrace{(\overline{Q}_1Q_0 + Q_1\overline{Q}_0)}_{Q_1\oplus Q_0} = x\oplus Q_1\oplus Q_0 \rightarrow$$

$$D_1 = \overline{x\oplus Q_1\oplus Q_0}$$

Contador binario progresivo-regresivo de 2 bits

Contador binario progresivo-regresivo de 2 bits



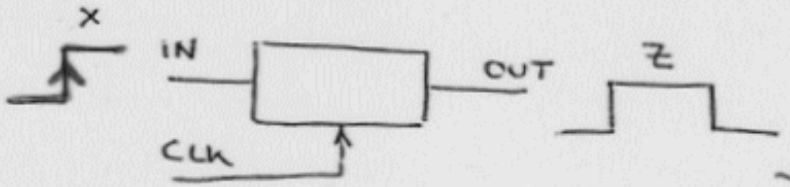
Si X=0 →

ACTUAL $Q_1 Q_0$	SIGUIENTE $Q_1 Q_0$
00	1 1
01	0 0
10	0 1
11	1 0

Si X=1 →

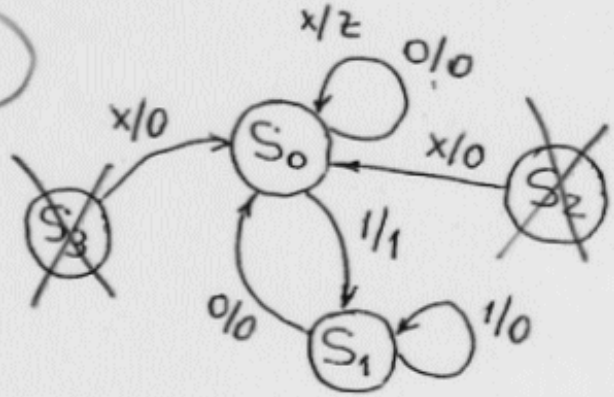
$Q_1 Q_0$ ACTUAL	$Q_1 Q_0$ SIGUIENTE
00	0 1
01	1 0
10	1 1
11	0 0

Monoestable con máquina de estados



Detecta ⊓

Mealy:



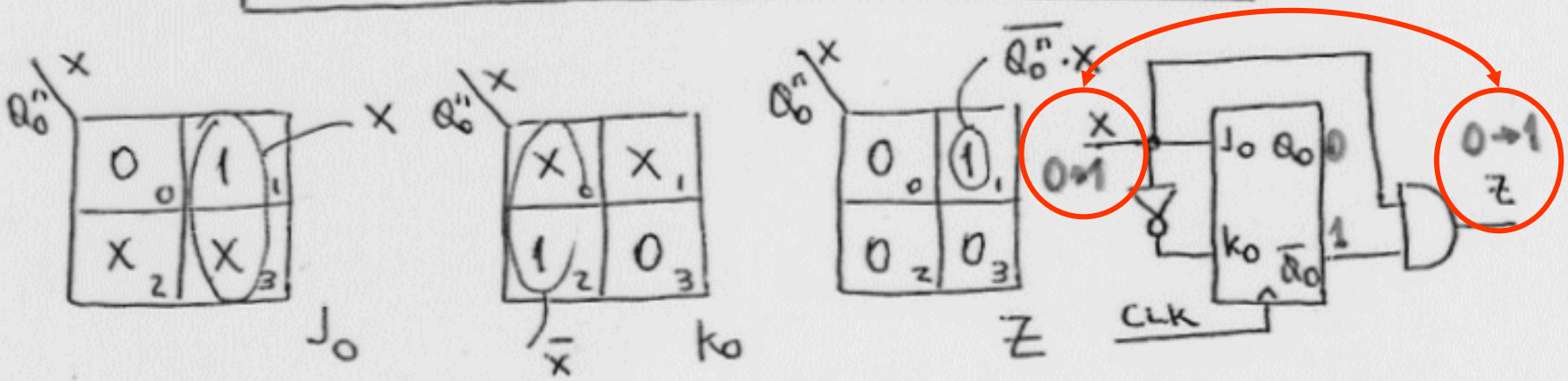
Q_n	Q_{n+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Tabla de transición del flip-flop JK

Método para lógica standard

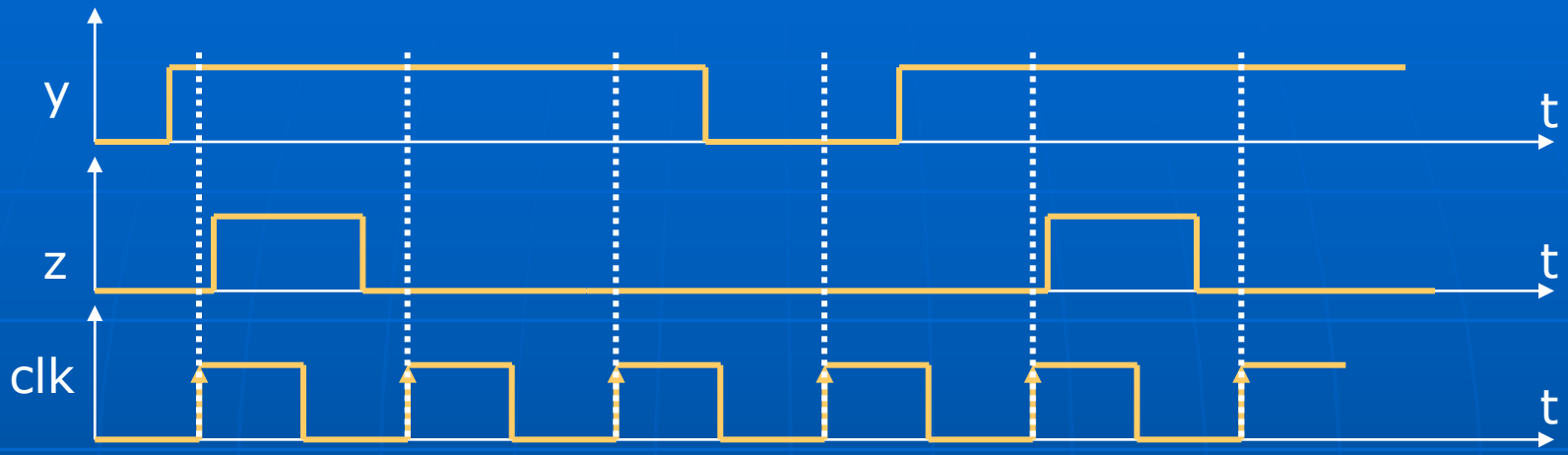
Tabla de excitación

Est.	Q_0^n	Q_0^{n+1}	J_0	K_0	Z	Q_0^{n+1}	J_0	K_0	Z
		$X=0$				$X=1$			
S_0	0	0	0	x	0	1	1	X	1
S_1	1	0	x	1	0	1	X	0	0



Si la entrada en S0 cambia de 0 a 1, la salida sigue a la entrada sin importar si el reloj habilita el cambio.
ESTO PUEDE TRAER PROBLEMAS EN DISEÑOS MAS COMPLEJOS....!!!!!!!!!!!!!

Diseño de circuito monoestable disparado por flanco ascendente



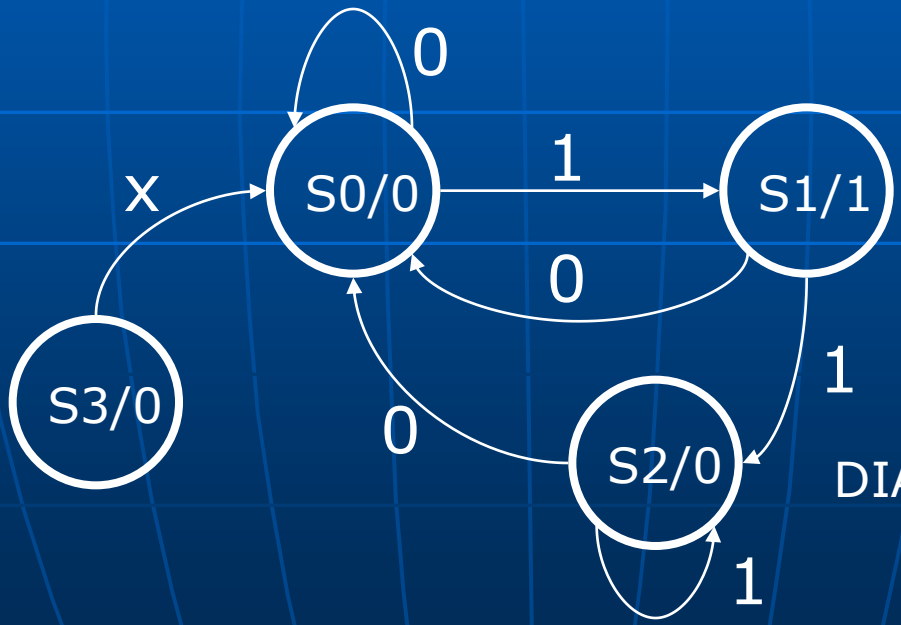


DIAGRAMA DE ESTADOS

Síntesis por Moore

Método para lógica standard

TABLA DE EXCITACIÓN

ESTADO ACTUAL / SALIDAS ACTUALES			ESTADO SIGUIENTE / SALIDAS SIGUIENTES				ENTRADAS ACTUALES			
	Q_{1n}	Q_{0n}	Q_1	$Q_{0(n+1)}$	Q_1	$Q_{0(n+1)}$	D_1	$D_{0(n+1)}$	D_1	$D_{0(n+1)}$
S_0	0	0	0	0	0	1	0	0	0	1
S_1	0	1	0	0	1	0	0	0	1	0
S_2	1	0	0	0	1	0	0	0	1	0
S_3	1	1	0	0	0	0	0	0	0	0
			0		0	1	0		1	
			y				y			

Síntesis por Moore

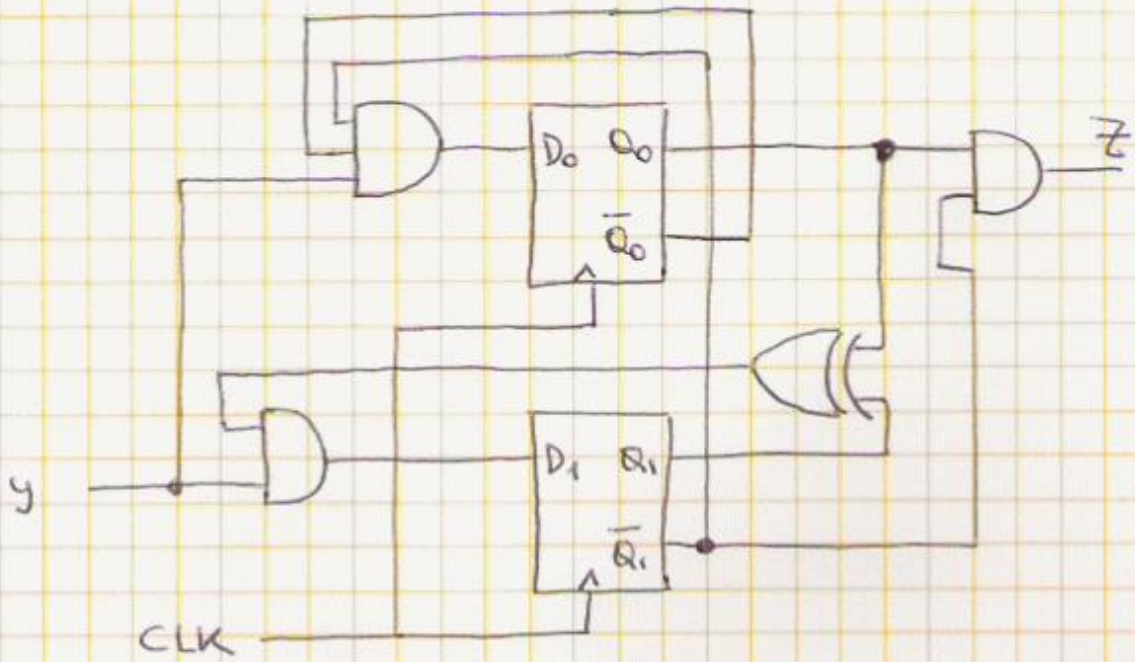
y \ Q_1Q_0	00	01	11	10
0	00	00	00	00
1	01	10	00	10

D_1D_0

$$D_1 = y \cdot (\bar{Q}_1 Q_0 + Q_1 \bar{Q}_0) = y \cdot (Q_0 \oplus Q_1)$$

$$D_0 = y \cdot \bar{Q}_1 \cdot \bar{Q}_0$$

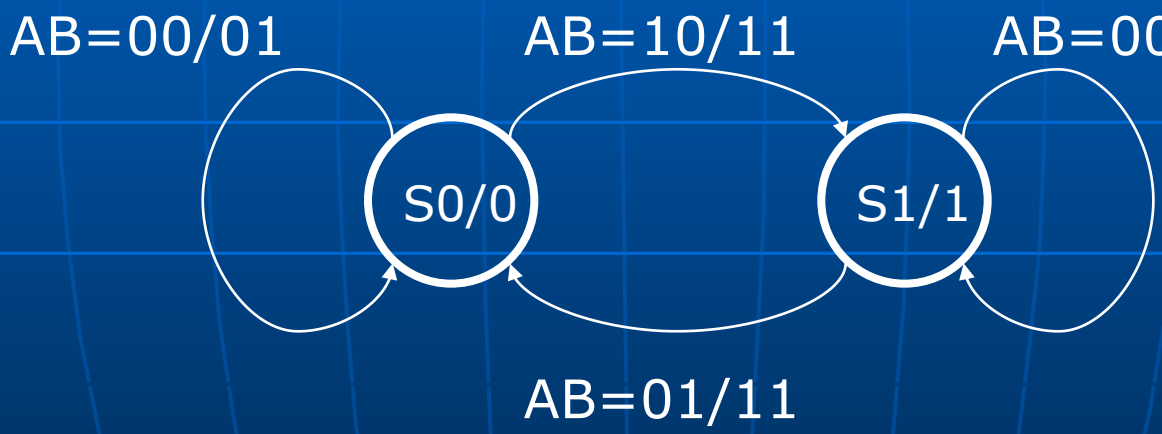
$$Z = \overline{Q_1 \cdot Q_0}$$



Por Moore el mismo problema generó un hardware un poco mas complejo pero con la salida sin depender de la entrada, salvo en cada flanco de reloj.

Síntesis por Moore

Diseñar en base a un flip-flop JK un circuito que responda con el siguiente diagrama de estados:



Método para lógica standard

TABLA DE EXCITACIÓN

ESTADO ACTUAL	ESTADO SIGUIENTE				ENTRADAS ACTUALES J_0K_0			
	00	01	10	11	00	01	10	11
0	0	0	1	1	0X	0X	1X	1X
1	1	0	1	0	X0	X1	X0	X1

AB

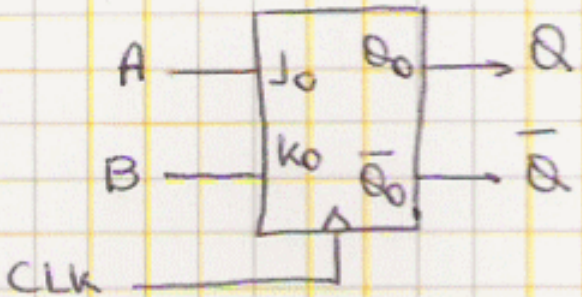
AB

Q_{0_n} \ AB	$\bar{A}$ 00	$\bar{A}$ 01	A 11	A 10
$\bar{Q}_0$ 0	0X	0X	1X	1X
Q_0 1	X0	X1	X1	X0
	$\bar{B}$	B		$\bar{B}$

$J_0 = A$

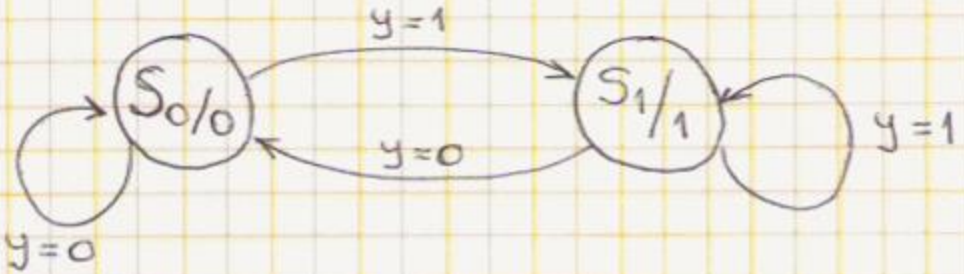
$K_0 = B$

$\Rightarrow$



El diagrama de estados corresponde a un flip-flop "JK“..!!!

Síntesis por Moore



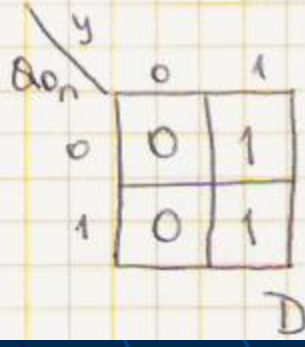
ESTADO ACTUAL	ESTADO SIGUIENTE / SALIDA
	y
0	0 1
1	0 1

Método para lógica standard

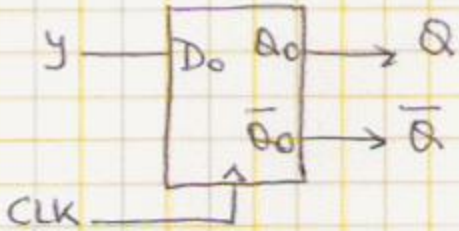
Implementar con FF "D"

TABLA DE EXCITACIÓN

ESTADO ACTUAL	ESTADO SIGUIENTE y	ENTRADA ACTUAL D_0 y
0	0 1	0 1
1	0 1	0 1



$D_0 = y$



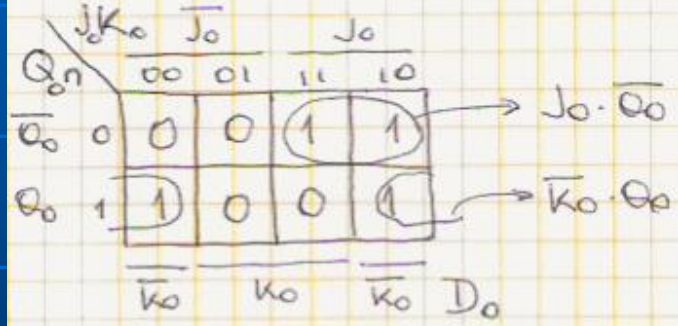
El diagrama de estados era el de un FF "D"

Método para lógica standard

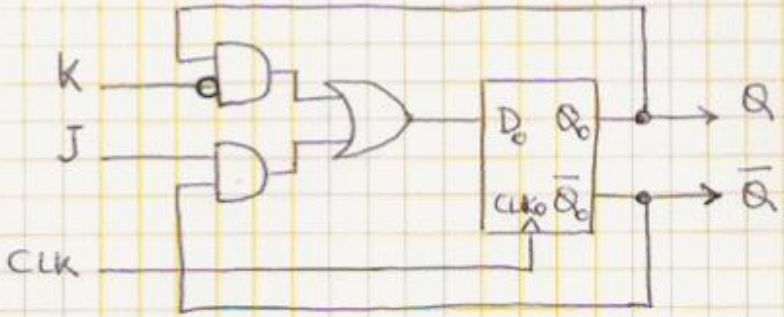
Diseño de JK con tipo D (MOORE)

TABLA DE EXCITACIÓN

ESTADO ACTUAL Q_{0n}	ESTADO SIGUIENTE Q_{0n+1}				ENTRADAS ACTUAL D_0			
0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0
	00	01	10	11	00	01	10	11
		J_0K_0				J_0K_0		



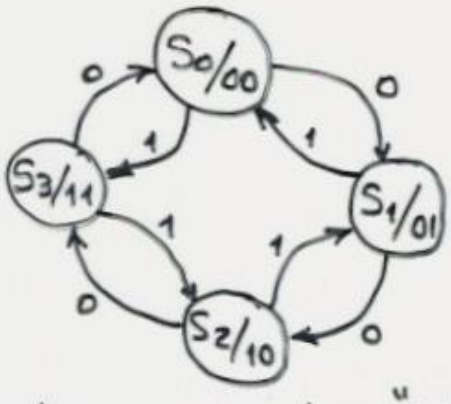
$$D_0 = J_0 \cdot \overline{Q_0} + \overline{K_0} \cdot Q_0$$



Síntesis por Moore

Diseño de contador progresivo-regresivo con FF "D" y "JK"

Método para lógica standard



Implementar con FF tipo "D" y "JK"

ESTADO	$Q_1^n Q_0^n$	$Q_1^{n+1} Q_0^{n+1}$		$D_1 D_0$		J_1	K_1	J_0	K_0	J_1	K_1	J_0	K_0
S_0	00	01	11	01	11	0	X	1	X	1	X	1	X
S_1	01	10	00	10	00	1	X	X	1	0	X	X	1
S_2	10	11	01	11	01	X	0	1	X	X	1	1	X
S_3	11	00	10	00	10	X	1	X	1	X	0	X	1
		y=0	y=1	y=0	y=1	y=0				y=1			

FF_1 "D" FF_0 "JK"

Tabla de transición FF tipo "D"

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Tabla de transición FF tipo "JK"

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Método para lógica standard

Sintetizar un contador progresivo-regresivo binario de 2 bits con MÁQUINA DE MOORE [CONTINUACIÓN]

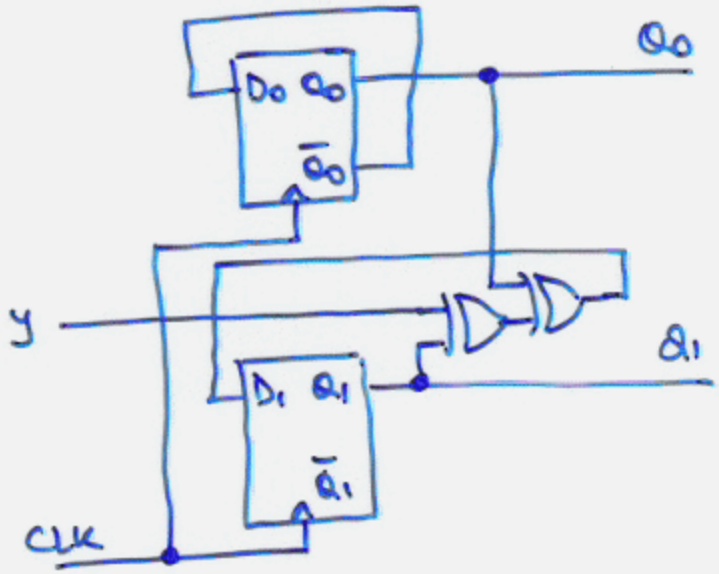
IMPLEMENTACION CON FFs tipo "D"

y \ Q_1Q_0	00	01	11	10
0	1	0	0	1
1	1	0	0	1

$D_0 = \bar{Q}_0$

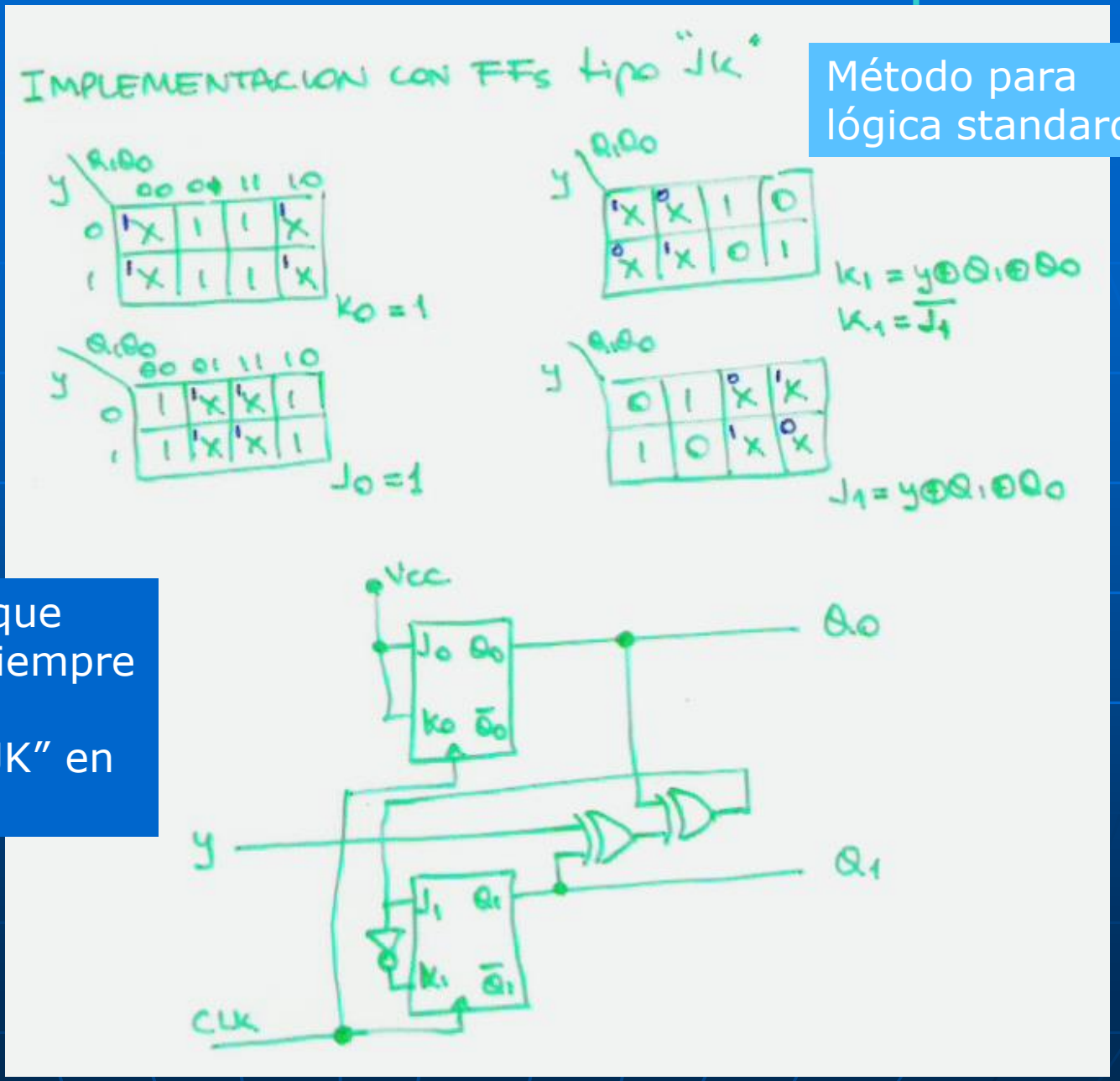
y \ Q_1Q_0	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$D_1 = y \oplus Q_1 \oplus Q_0$



Síntesis por Moore

Método para lógica standard



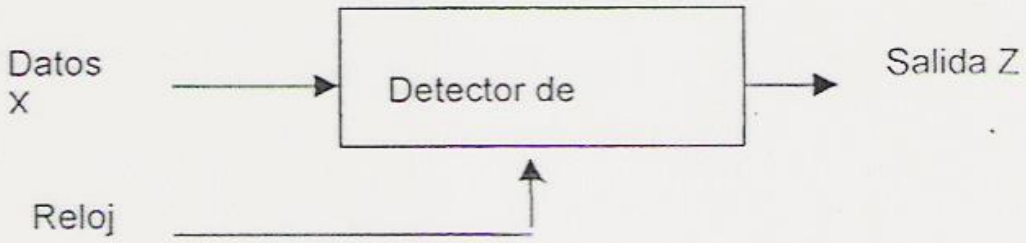
Este es un ejemplo que demuestra que No siempre se usa menos lógica cuando se emplea "JK" en vez de "D"

Método de síntesis por descripción de diagrama de estado empleando la máquina de Moore.

Diseño de un circuito detector de secuencia.

La salida Z, tiene por misión detectar cuando una dada combinación de bits es detectada, por ejemplo indicándolo con un nivel lógico alto y permaneciendo en bajo para cualquier otra combinación de bits de entrada.

El diagrama en bloques de tal circuito se muestra en la siguiente figura



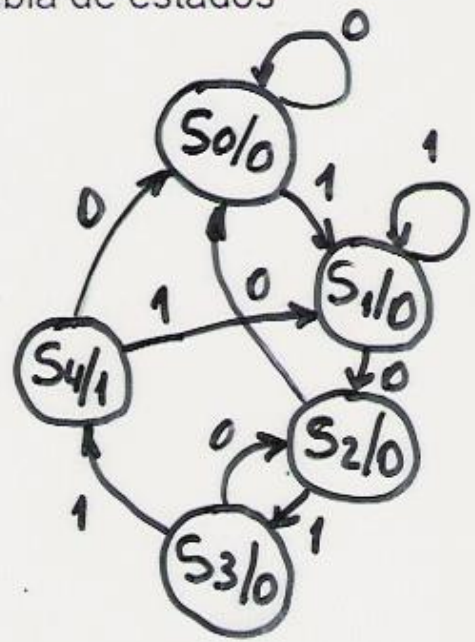
Las especificaciones para el mismo son las siguientes:

Detectar una secuencia de 4 bits cuya trama sea 1011, debiéndo poner a la salida Z en 1 para indicar dicha condición, caso contrario Z deberá permanecer en nivel lógico bajo.
Se permite solapamiento en la trama de bits
Ejemplo:

X	10000010110111111011011000000
Z	00000000010010000001001000000

Paso 1: Plantear el problema realizando el diagrama de estados y/o tabla de estados del circuito especificado.

Estado actual	Salida actual	Estado siguiente	
		X=0	X=1
S0	0	S0	S1
S1	0	S2	S1
S2	0	S0	S3
S3	0	S2	S4
S4	1	S0	S1



Paso 2: Asignación de estados.

S0 ⇒ Q2 Q1 Q0 = 000
S1 ⇒ Q2 Q1 Q0 = 001
S2 ⇒ Q2 Q1 Q0 = 010
S3 ⇒ Q2 Q1 Q0 = 011
S4 ⇒ Q2 Q1 Q0 = 100

Método para lógica standard

Paso 3: Realizar la tabla de excitación del circuito:

Estado actual	Salida FFs actual	Salida Z actual	Estado siguiente / Salidas FFs		Entradas FFs	
			X=0	X=1	X=0	X=1
S0	000	0	S0 / 000	S1 / 001	?	?
S1	001	0	S2 / 010	S1 / 001	?	?
S2	010	0	S0 / 000	S3 / 011	?	?
S3	011	0	S2 / 010	S4 / 100	?	?
S4	100	1	S0 / 000	S1 / 001	?	?
S5	101	?	?	?	?	?
S6	110	?	?	?	?	?
S7	111	?	?	?	?	?

Como regla general para emplear un dado tipo de flip-flop, se debe plantear la denominada tabla de transición del mismo:

Qactual	Qsiguiente	Dactual
0	0	0
0	1	1
1	0	0
1	1	1

La tabla quedaría provisoriamente:

Estado actual	Salida FFs actual	Salida Z actual	Estado siguiente / Salidas FFs		Entradas FFs D2D1D0	
			X=0	X=1	X=0	X=1
S0	000	0	S0 / 000	S1 / 001	000	001
S1	001	0	S2 / 010	S1 / 001	010	001
S2	010	0	S0 / 000	S3 / 011	000	011
S3	011	0	S2 / 000	S4 / 100	000	100
S4	100	1	S0 / 000	S1 / 001	000	001
S5	101	?	?	?	?	?
S6	110	?	?	?	?	?
S7	111	?	?	?	?	?

En cambio si la elección es emplear flip-flops tipo JK se deberían tener en cuenta las dos entradas JK por flip-flop, es decir, un total de seis variables: J0, K0, J1, K1, J2 y K2.

La tabla de transición de este tipo de flip-flop es la siguiente:

Q actual	Q siguiente	J actual	K actual
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Si se desea emplear flip-flop tipo JK la tabla de transición anterior quedaría de la siguiente manera:

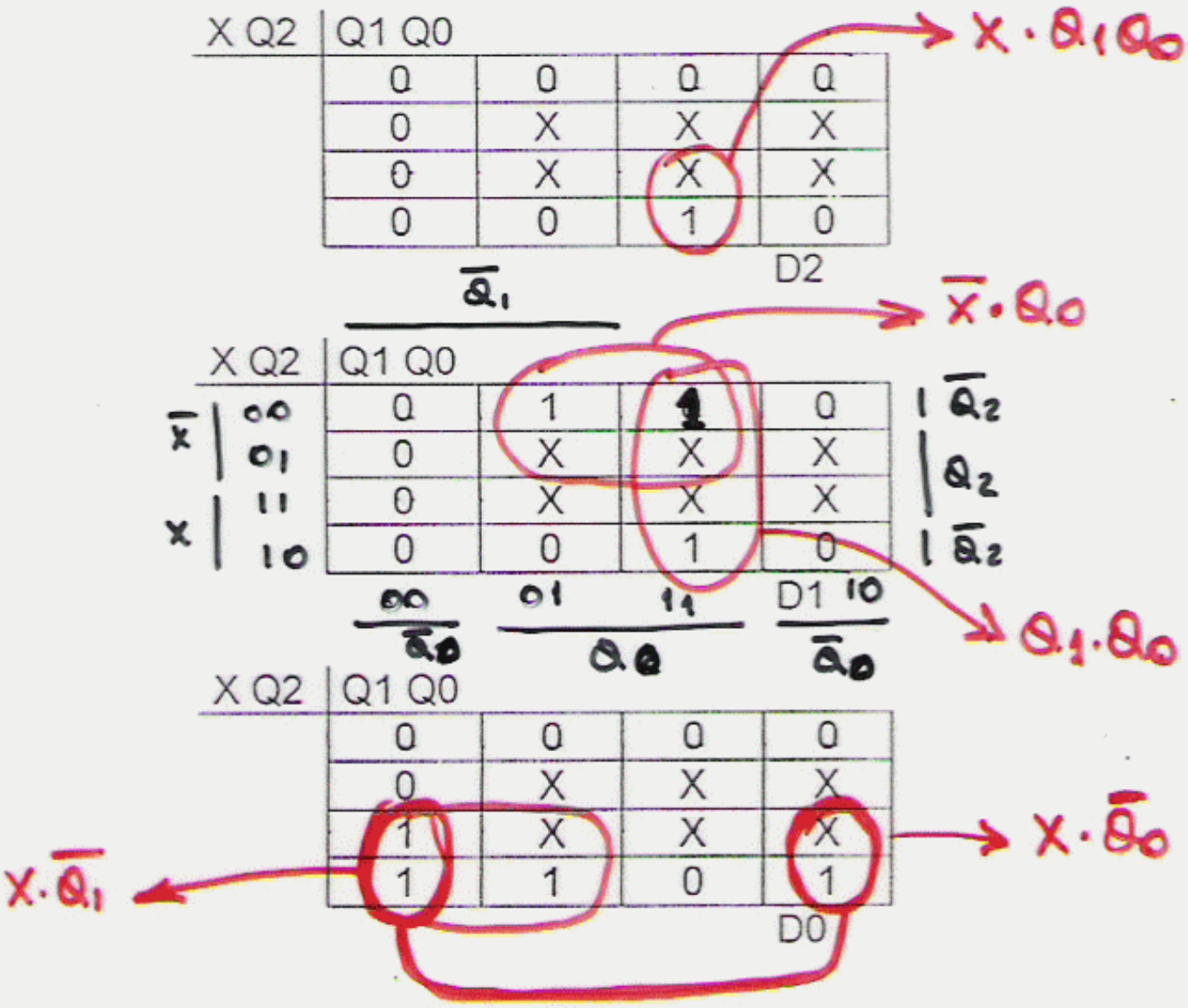
Estado actual	Salida FFs actual`	Salida Z actual	Estado siguiente / Salidas FFs		Entradas FFs J2K2 J1K1 J0K0	Entradas FFs J2K2 J1K1 J0K0
			X=0	X=1	X=0	X=1
S0	000	0	S0 / 000	S1 / 001	0X 0X 0X	0X 0X 1X
S1	001	0	S2 / 010	S1 / 001	0X 1X X1	0X 0X X0
S2	010	0	S0 / 000	S3 / 011	0X X1 0X	0X X0 1X
S3	011	0	S2 / 010	S4 / 100	0X X0 X1	1X X1 X1
S4	100	1	S0 / 000	S1 / 001	X1 0X 0X	X1 0X 1X
S5	101	?	?	?	?	?
S6	110	?	?	?	?	?
S7	111	?	?	?	?	?

Continuando con la síntesisde implementación con flip-flops tipo D

Estado actual	Salida FFs actual	Salida Z actual	Estado siguiente / Salidas FFs		Entradas FFs D2D1D0	
			X=0	X=1	X=0	X=1
S0	000	0	S0 / 000	S1 / 001	000	001
S1	001	0	S2 / 010	S1 / 001	010	001
S2	010	0	S0 / 000	S3 / 011	000	011
S3	011	0	S2 / 010	S4 / 100	010	100
S4	100	1	S0 / 000	S1 / 001	000	001
S5	101	X	X/XXX	X/XXX	XXX	XXX
S6	110	X	X/XXX	X/XXX	XXX	XXX
S7	111	X	X/XXX	X/XXX	XXX	XXX

Síntesis por Moore

Lo que sigue es sintetizar las funciones de D2, D1 y D0:



Análisis y Síntesis

De los Karnaugh sacamos las siguientes expresiones:

$$D2 = X\ Q0\ Q1$$

$$D1 = /X\ \ Q0 + Q1\ Q0$$

$$D0 = X\ (\ /Q0 + /Q1)$$

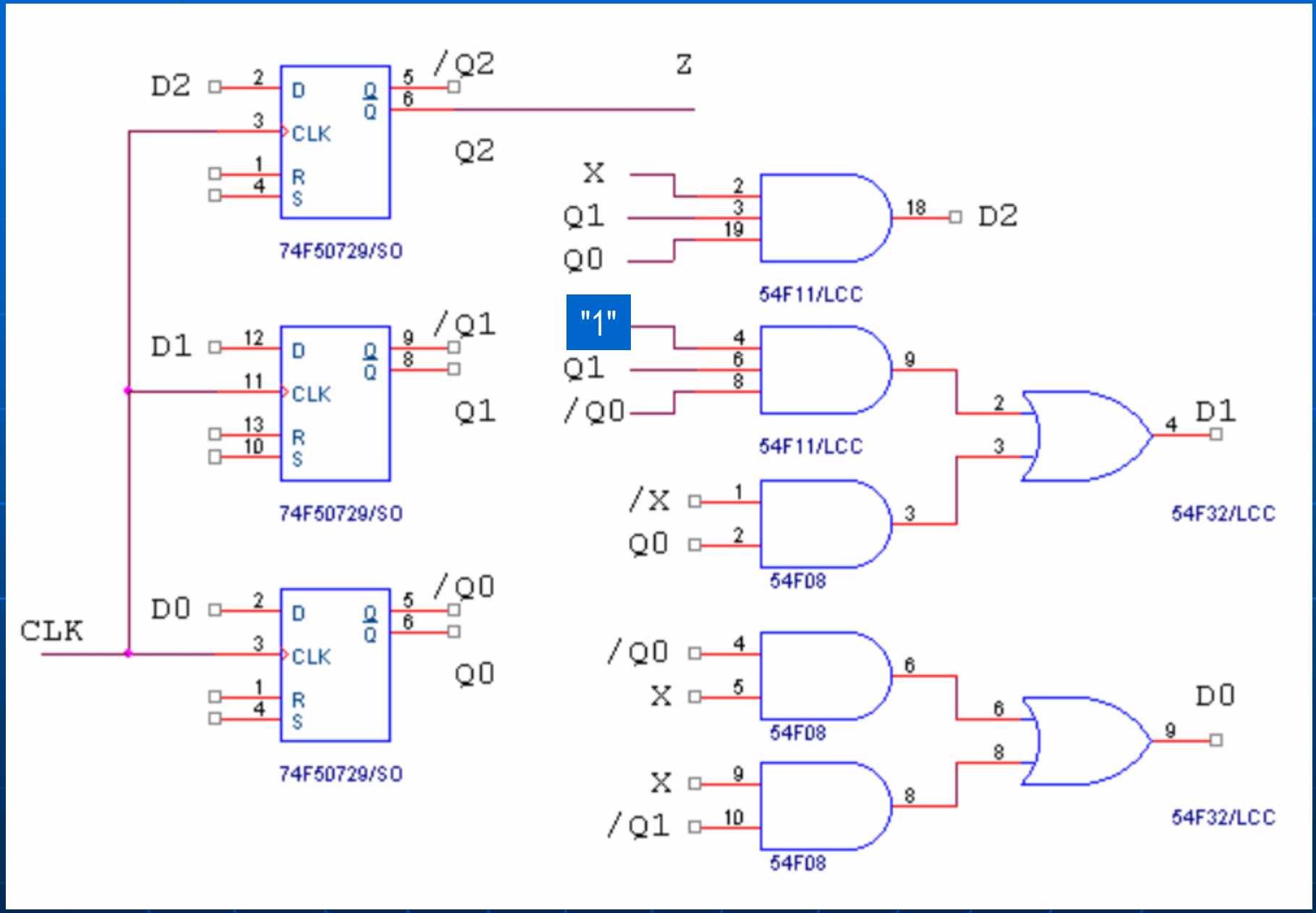
Por otro lado la salida Z, que sólo depende de las salidas de los flip-flops, tenemos:

Q2	Q1 Q0			
	0	0	0	0
	1	X	X	X
				Z

De aquí se desprende que Z vale:

$$Z = Q2$$

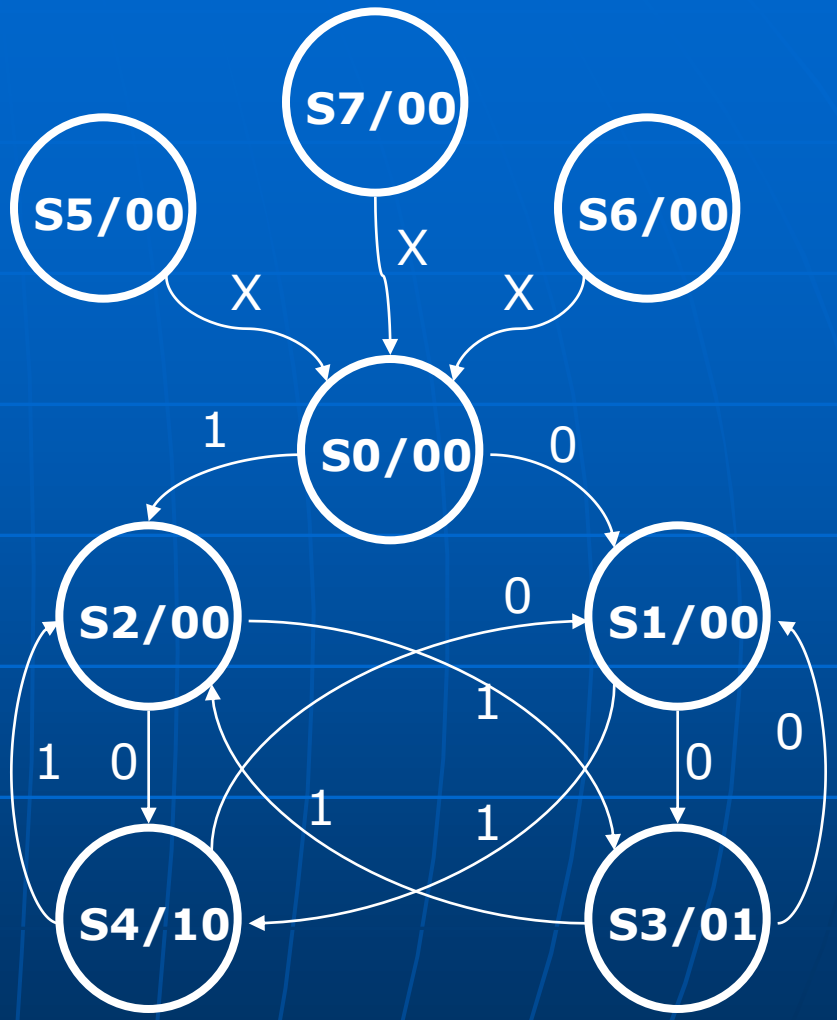
Análisis y Síntesis



Síntesis por Moore

Detector de paridad par en formato serie de 2 bits de magnitud

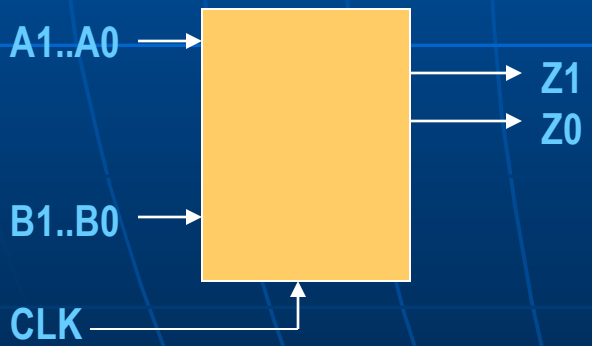
(BITS ENTRANTES)		Z_1	Z_0
D_1	D_0	(SALIDAS)	
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1
trabajado		0	0

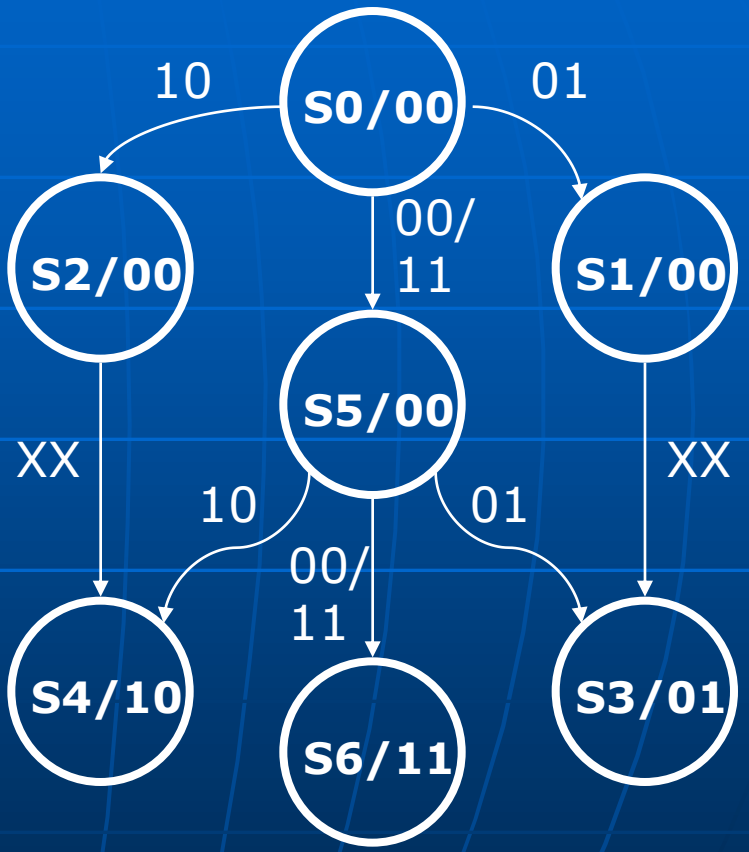


Diseño de comparador de magnitud serie de dos números sin signo (A y B) donde se transmite el bit mas significativo primero

Las salidas Z1Z0 deben cumplir con la siguiente tabla:

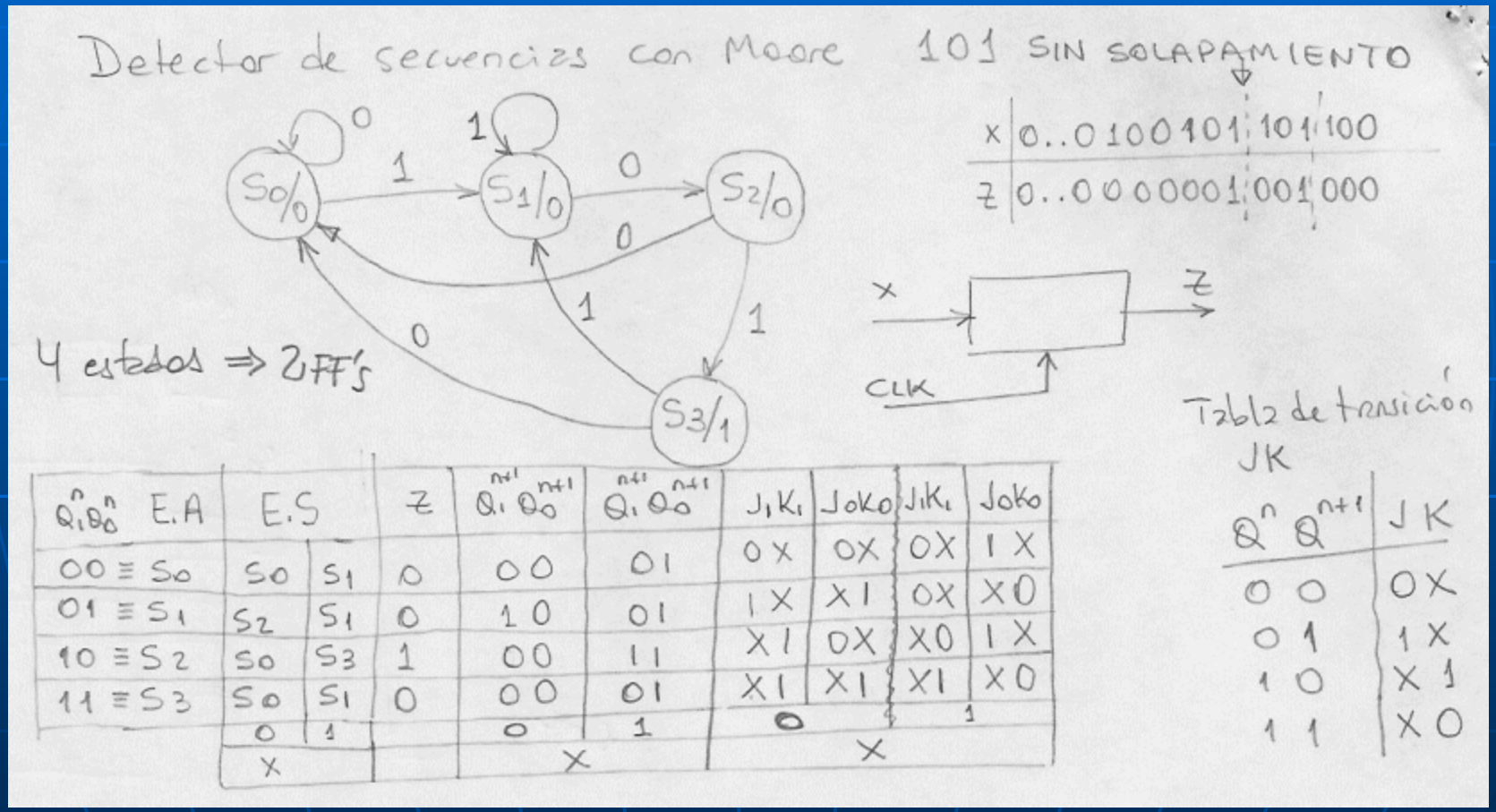
Si A>B => Z1Z0=10
Si A<B => Z1Z0=01
Si A=B => Z1Z0=11
Si se está comparando => Z1Z0=00





Método para lógica standard

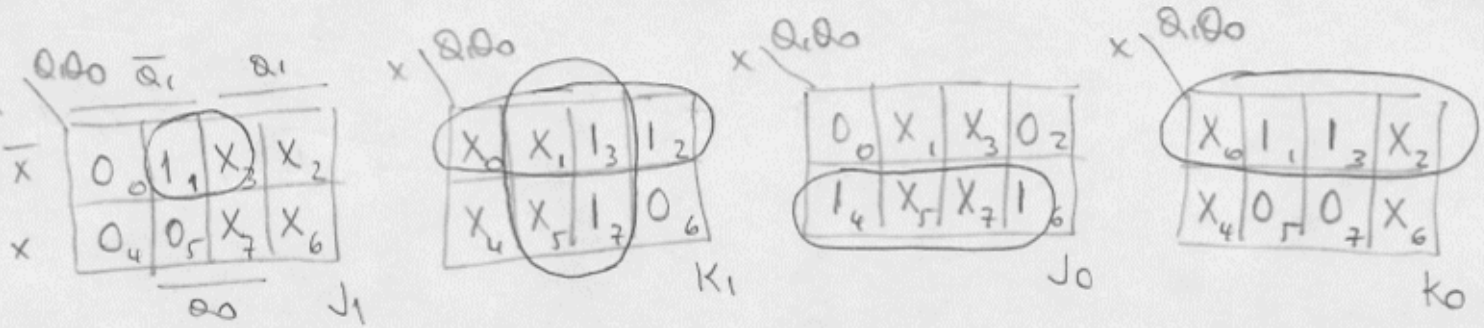
Detector de la secuencia "101" sin solapamiento



Q_1Q_0 E.A	E.S		z	$Q_1^{n+1}Q_0^{n+1}$		J_1K_1	J_0K_0	J_1K_1	J_0K_0
00 ≡ S0	S0	S1	0	00	01	0X	0X	0X	1X
01 ≡ S1	S2	S1	0	10	01	1X	X1	0X	X0
10 ≡ S2	S0	S3	1	00	11	X1	0X	X0	1X
11 ≡ S3	S0	S1	0	00	01	X1	X1	X1	X0
	0	1		0	1	0		1	
	X			X		X			

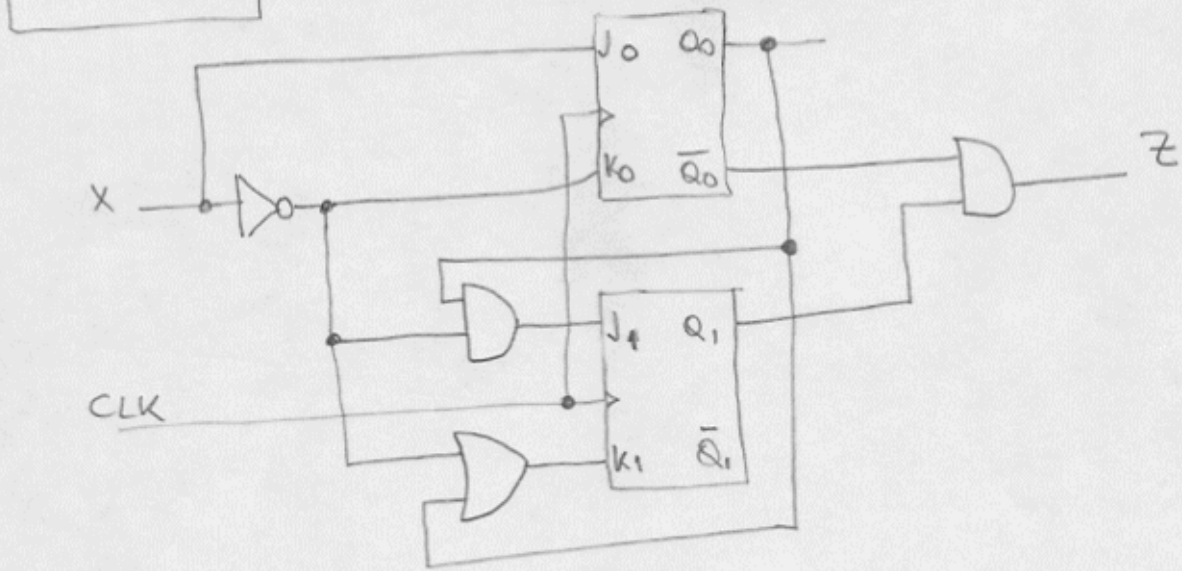
$Q\ Q^{n+1}$	J K
0 0	0X
0 1	1X
1 0	X1
1 1	X0

Detector de la secuencia "101" sin solapamiento



$J_1 = \bar{x} \cdot Q_0$ $\quad K_1 = \bar{x} + Q_0$ $\quad J_0 = x$ $\quad K_0 = \bar{x}$

$z = Q_1 \cdot \bar{Q}_0$

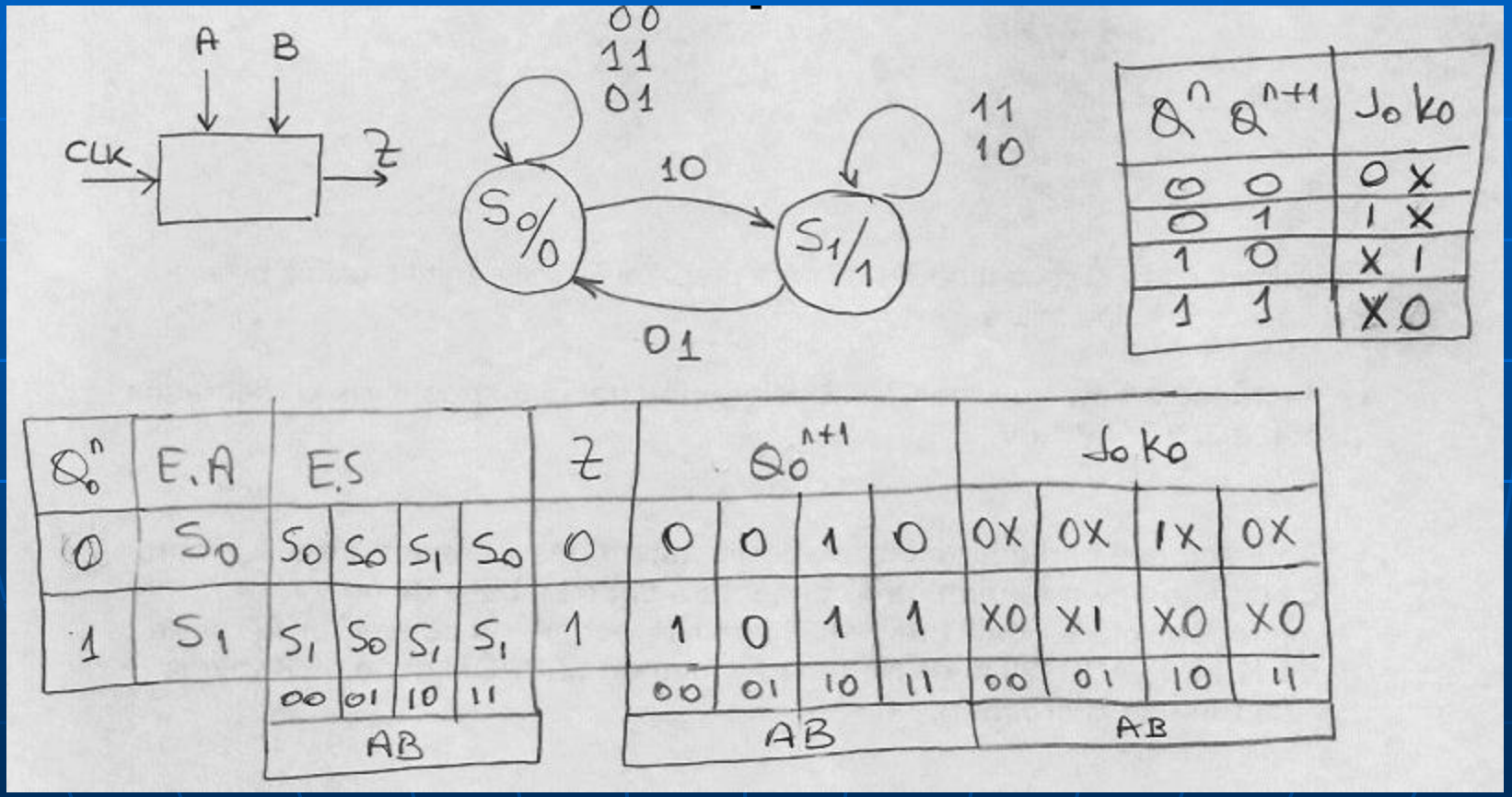


El flip-flop JK_0 funciona como tipo D copiando a X.

Síntesis por Moore

Método para lógica standard

Comando de un motor con dos pulsadores A y B

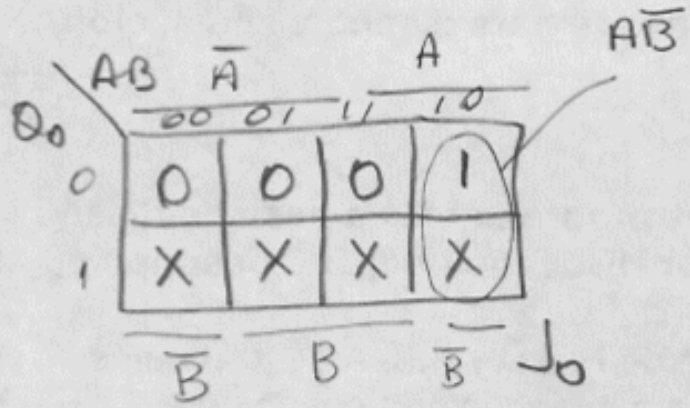


Q^n	Q^{n+1}	J_0 K_0
0	0	0 X
0	1	1 X
1	0	X 1
1	1	X 0

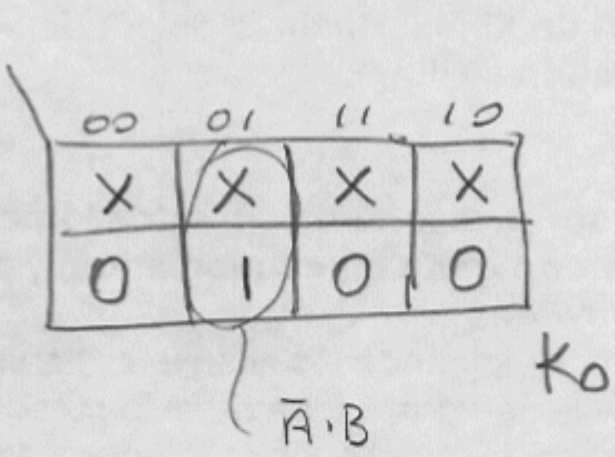
Q_0^n	E.A	E.S				Z
0	S_0	S_0	S_0	S_1	S_0	0
1	S_1	S_1	S_0	S_1	S_1	1
		00	01	10	11	
		AB				

Q_0^{n+1}				J_0 K_0			
0	0	1	0	0X	0X	1X	0X
1	0	1	1	X0	X1	X0	X0
00	01	10	11	00	01	10	11
AB				AB			

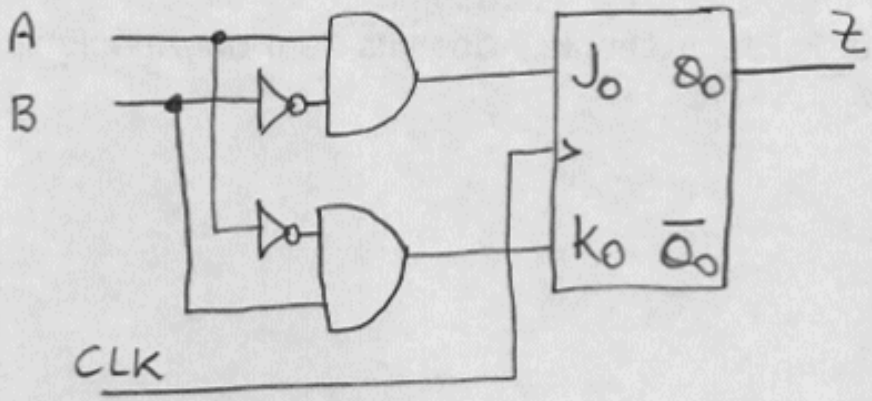
Comando de un motor con dos pulsadores A y B



$J_0 = A \cdot \bar{B}$



$K_0 = \bar{A} \cdot B$



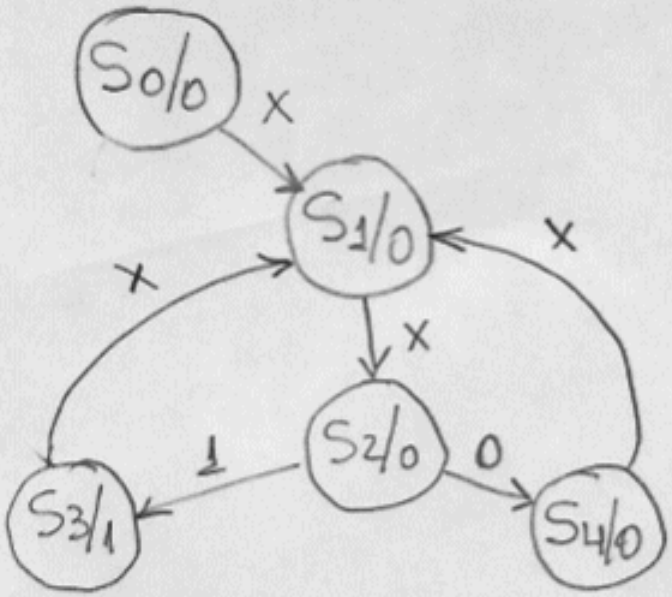
A B	J_0 K_0	Q^{n+1}
00	0 0	$\rightarrow Q^n$
01	0 1	$\rightarrow 0$
10	1 0	$\rightarrow 1$
11	0 0	$\rightarrow Q^n$

Detector de número impar en formato serie cíclico, siendo el primer bit de entrada de cada secuencia, el MSB.

Detector de números impares de 3 bits en un canal serie

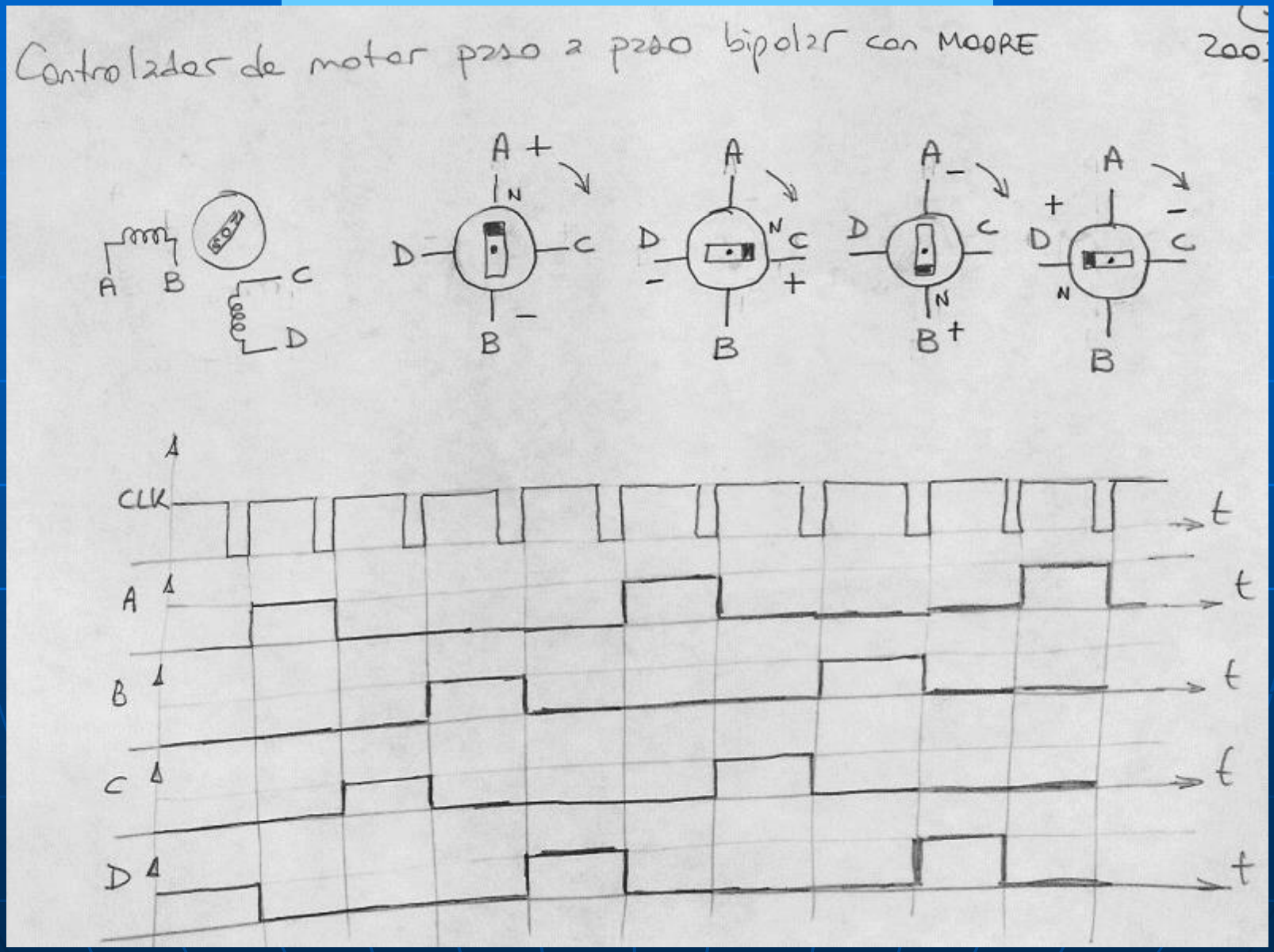
Sin solapamiento

1er bit MSB.



b2 b1 b0	
000	
001	impar ⇒ b0 = 1
010	
011	
100	
101	
110	
111	

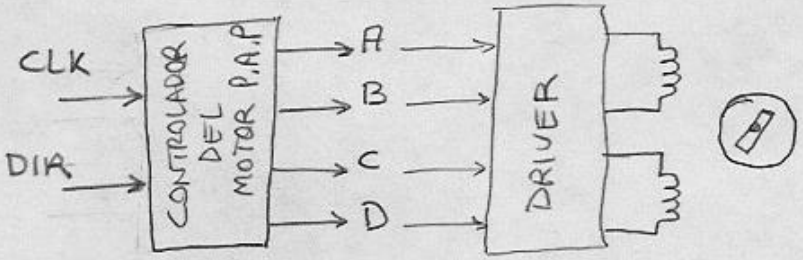
Controlador de motor paso a paso

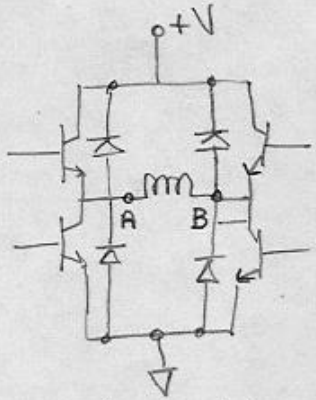


Controlador de motor paso a paso

Secuencia 2 A - C - B - D - A - C - ....

Secuencia 1 A - D - B - C - A - D - ....



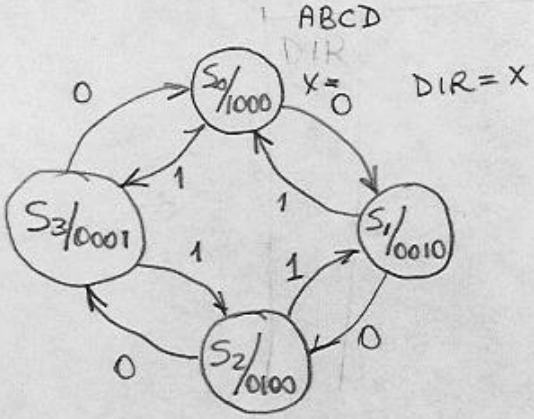


Puente H

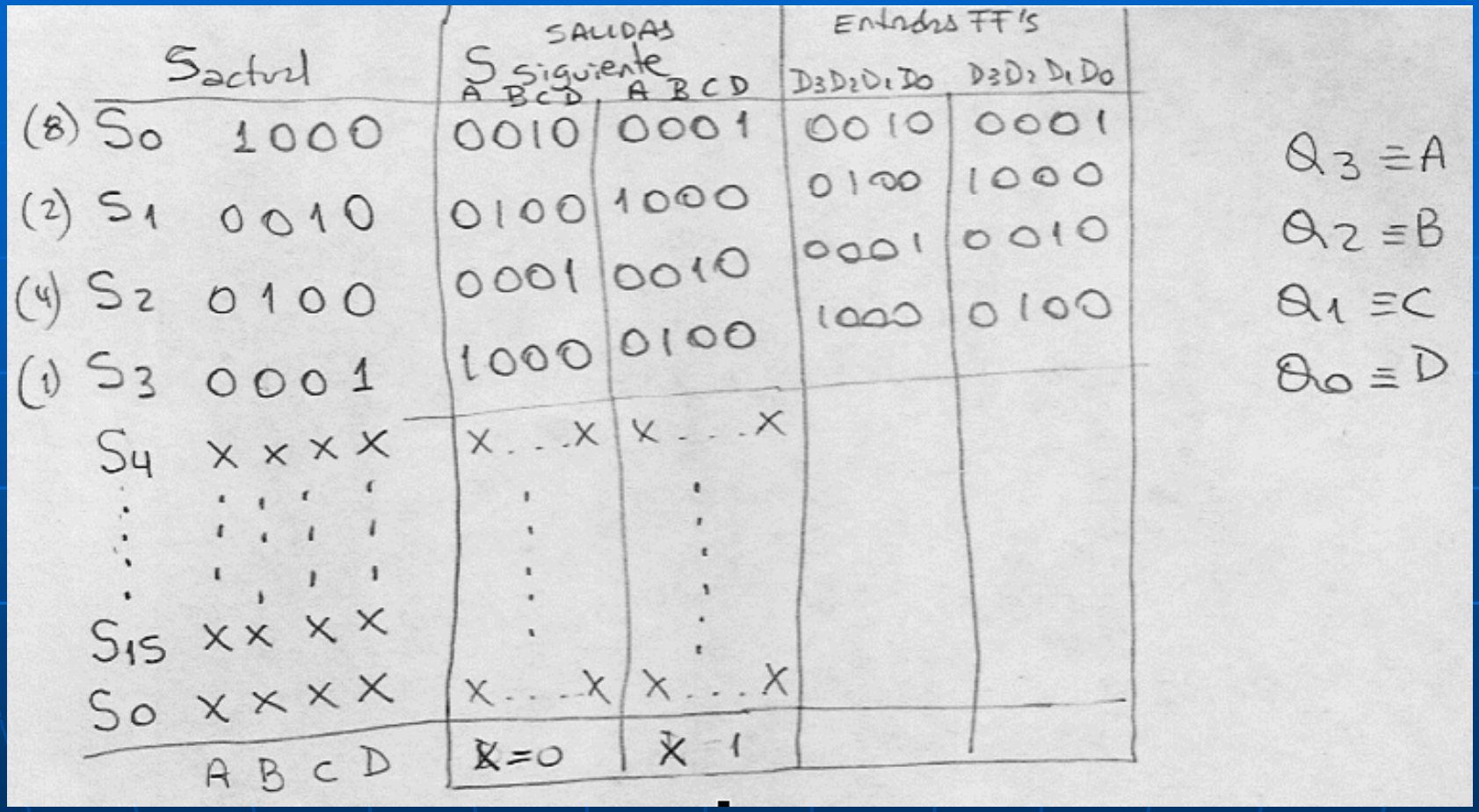
DIR = 0 → secuencia con sentido horario

DIR = 1 → " " antihorario

DIR	A	B	C	D
0	1	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	0	0	1
1	1	0	0	0
1	0	0	0	1
1	0	1	0	0
1	0	0	1	0



Controlador de motor paso a paso



	S_{actual}		Salidas $S_{siguiente}$ A B C D		Entradas FF's $D_3D_2D_1D_0$	$D_3D_2D_1D_0$
(8)	S_0	1000	0010	0001	0010	0001
(2)	S_1	0010	0100	1000	0100	1000
(4)	S_2	0100	0001	0010	0001	0010
(1)	S_3	0001	1000	0100	1000	0100
	S_4	x x x x	x...x	x...x		
	⋮	⋮ ⋮ ⋮ ⋮	⋮	⋮		
	S_{15}	x x x x				
	S_0	x x x x	x...x	x...x		
		A B C D	X=0	X=1		

$Q_3 \equiv A$

$Q_2 \equiv B$

$Q_1 \equiv C$

$Q_0 \equiv D$

Controlador de motor paso a paso

Método para lógica standard

Si se pone todas "0" para las opciones de S0 y de S4 hasta S15 tendremos:

$$D_3 = \overline{Q_3}\cdot\overline{Q_2}\cdot\overline{Q_1}\cdot Q_0\cdot\overline{X} + \overline{Q_3}\cdot\overline{Q_2}\cdot Q_1\cdot\overline{Q_0}\cdot X$$

$$D_2 = \overline{Q_3}\cdot\overline{Q_2}\cdot Q_1\cdot\overline{Q_0}\cdot\overline{X} + \overline{Q_3}\cdot\overline{Q_2}\cdot\overline{Q_1}\cdot Q_0\cdot X$$

$$D_1 = Q_3\cdot\overline{Q_2}\cdot\overline{Q_1}\cdot\overline{Q_0}\cdot\overline{X} + \overline{Q_3}\cdot Q_2\cdot\overline{Q_1}\cdot\overline{Q_0}\cdot X$$

$$D_0 = \overline{Q_3}\cdot Q_2\cdot\overline{Q_1}\cdot\overline{Q_0}\cdot\overline{X} + Q_3\cdot\overline{Q_2}\cdot\overline{Q_1}\cdot\overline{Q_0}\cdot X$$

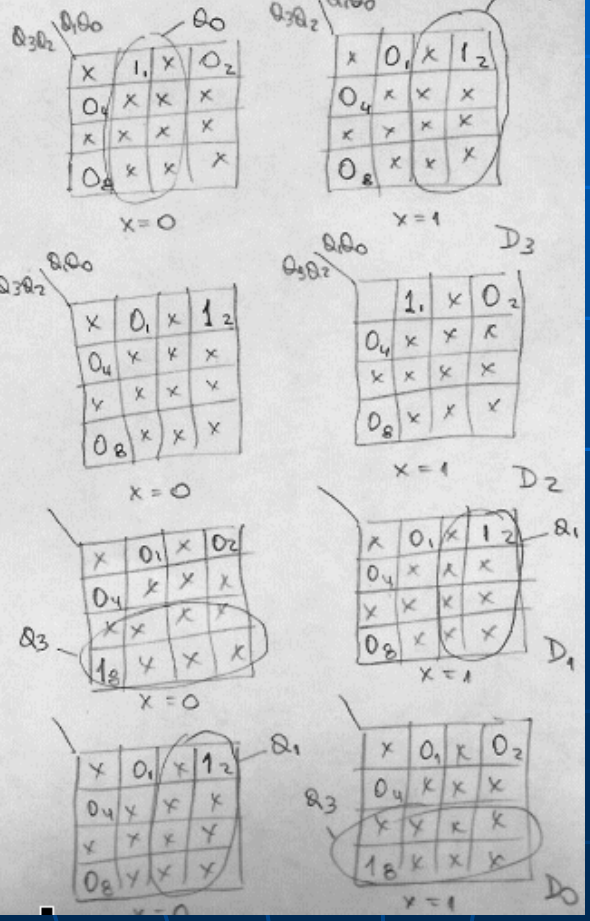
$A = Q_3$
$B = Q_2$
$C = Q_1$
$D = Q_0$

$$D_3 = Q_0\cdot\overline{X} + Q_1\cdot X$$

$$D_2 = Q_1\cdot\overline{X} + Q_0\cdot X$$

$$D_1 = Q_3\cdot\overline{X} + Q_1\cdot X$$

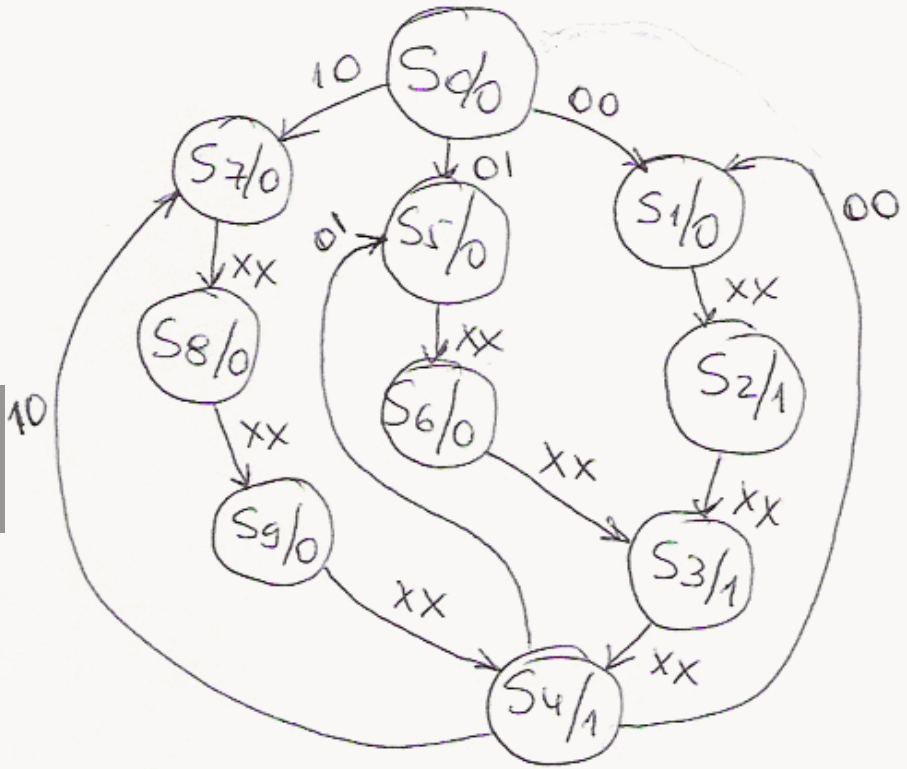
$$D_0 = Q_1\cdot\overline{X} + Q_3\cdot X$$



Sintetizar un generador de señales tal que genere la siguiente secuencia en forma cíclica:

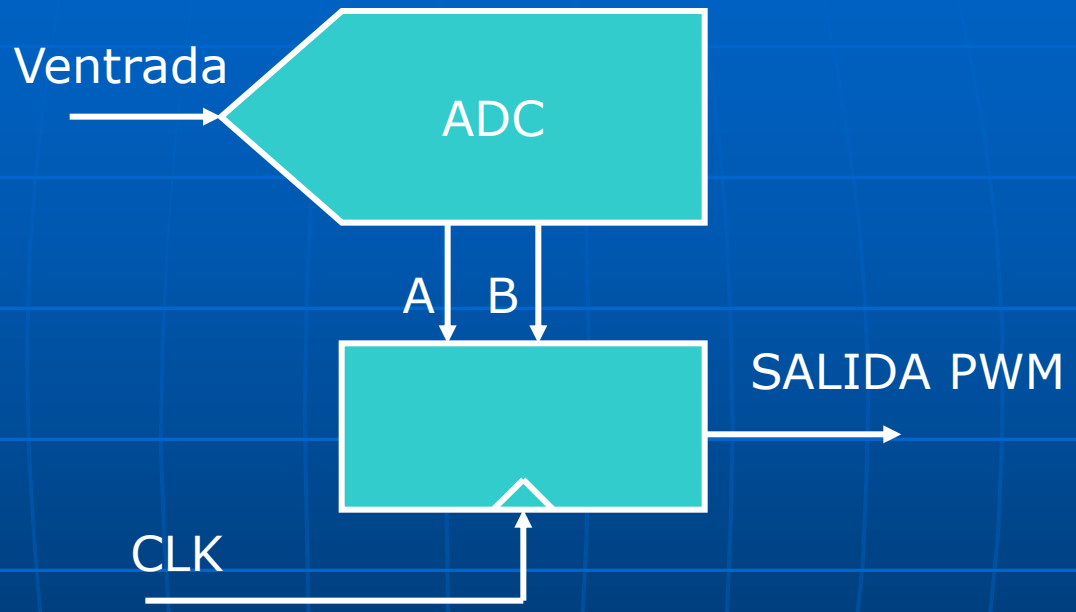
AB	secuencia
00	0 1 1 1
01	0 0 1 1
10	0 0 0 1
11	- - - -

Qué hacemos con esta combinación??



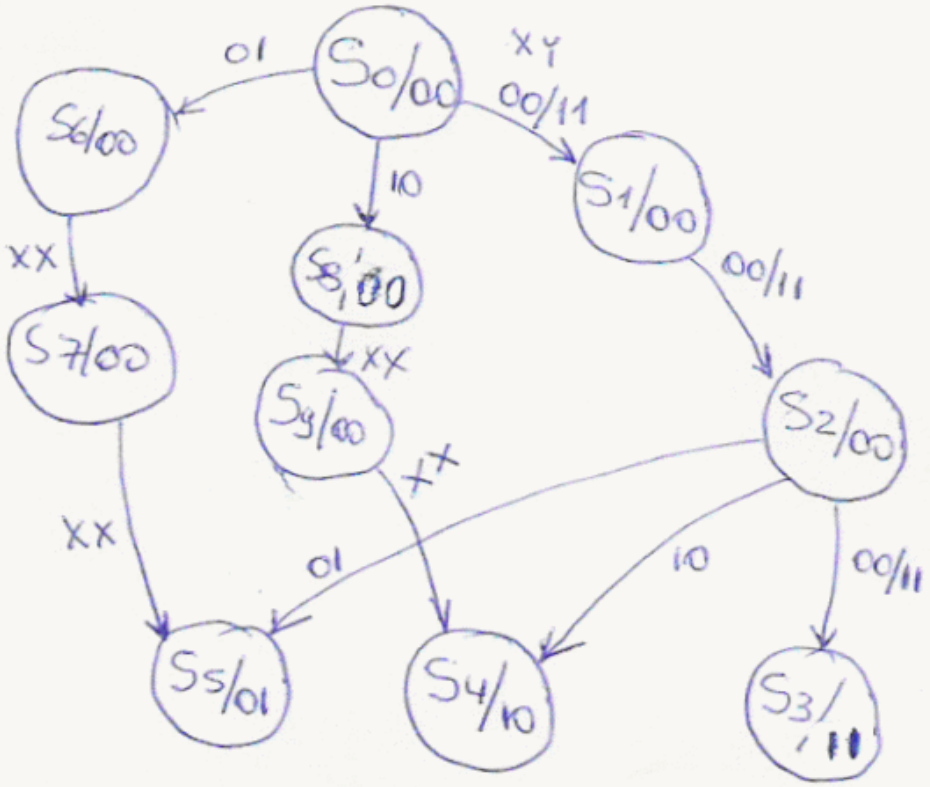
Las entradas se evalúan en el primer ciclo de reloj de la secuencia.

IMPLEMENTAR UN GENERADOR PWM CON UN CONVERSOR Y MÁQUINA DE ESTADOS MOORE



El bloque de síntesis es el del caso anterior donde las entradas de datos A y B provienen ahora del conversor analógico-digital.
Se debe sincronizar la actividad del ADC para que éste cambie los datos A y B cada 4 ciclos de reloj de la máquina de estados.

Sintetizar un circuito comparador de magnitud de dos números binarios sin signo de 3 bits transmitidos en serie (se manda bit más significativo primero)



SALIDAS

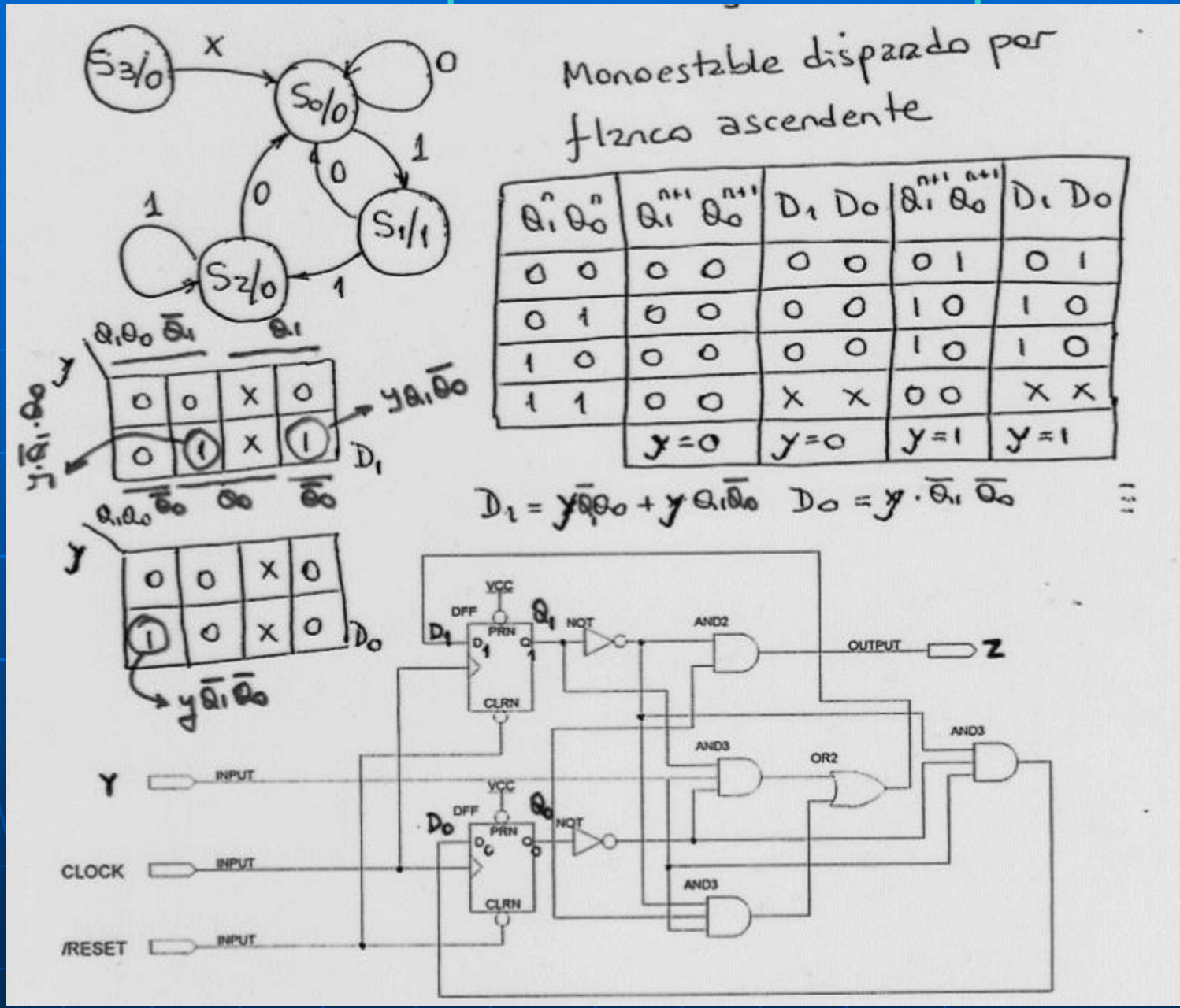
Z_1 Z_0	CONDICI
0 0	en proces
0 1	X < Y
1 0	X > Y
1 1	X = Y

Se usan 4 FF's

Síntesis por Moore en ambiente esquemático

Circuito monoestable disparado por flanco ascendente

Método para lógica standard

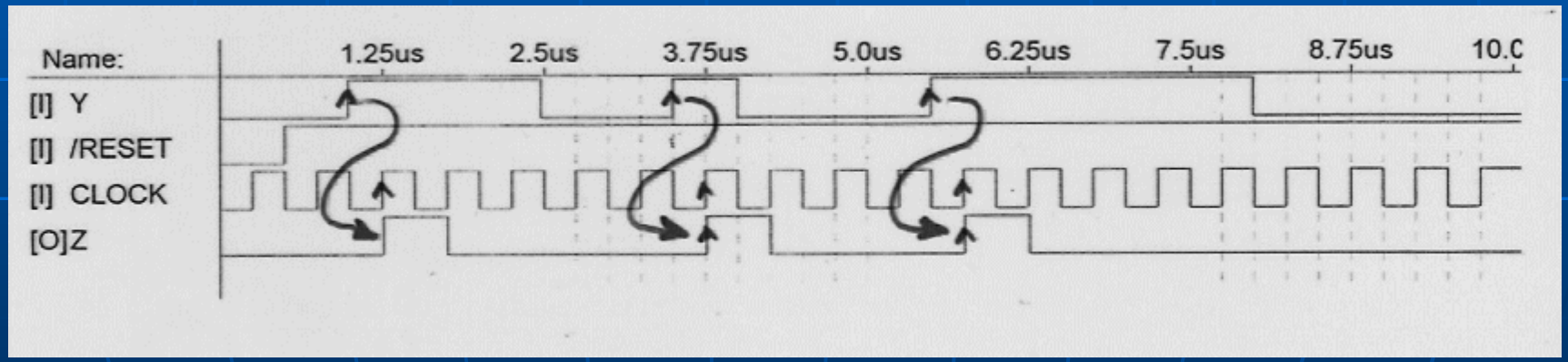


$Q_1^n Q_0^n$	$Q_1^{n+1} Q_0^{n+1}$	D_1 D_0	$Q_1^{n+1} Q_0^{n+1}$	D_1 D_0
0 0	0 0	0 0	0 1	0 1
0 1	0 0	0 0	1 0	1 0
1 0	0 0	0 0	1 0	1 0
1 1	0 0	x x	0 0	x x
	y=0	y=0	y=1	y=1

Síntesis por Moore en ambiente esquemático

Circuito monoestable disparado por flanco ascendente

Resultados de la simulación

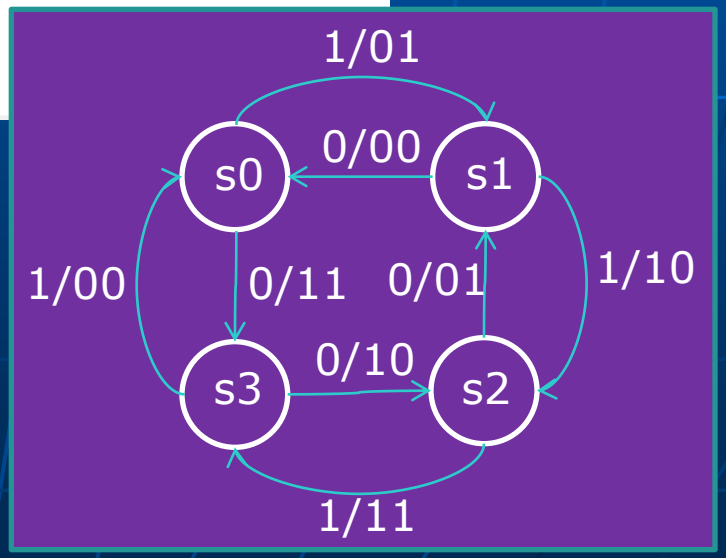


Síntesis por Mealy en VHDL

Síntesis de contador up-down con máquina de Mealy. Descripción en VHDL

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity mealy_counter_2bits is

    port
    (
        clk      : in  std_logic;
        dir      : in  std_logic;
        reset    : in  std_logic;
        data_out : out std_logic_vector(1 downto 0)
    );

end entity;

architecture rtl of mealy_counter_2bits is

    type state_type is (s0, s1, s2, s3);

    signal state : state_type;

begin
    process (clk, reset, dir)
    begin
        if reset = '1' then
            state <= s0;
        elsif (rising_edge(clk)) then
            case state is
                when s0=>
                    if dir = '1' then
                        state <= s1;
                    else
                        state <= s3;
                    end if;
                when s1=>
                    if dir = '1' then
                        state <= s2;
                    else
                        state <= s0;
                    end if;
                when s2=>
                    if dir = '1' then
                        state <= s3;
                    else
                        state <= s1;
                    end if;
                when s3=>
                    if dir = '1' then
                        state <= s0;
                    else
                        state <= s2;
                    end if;
            end case;
        end if;
    end process;
```

```vhdl
    process (state, dir)
    begin
        case state is
            when s0=>
                if dir = '1' then
                    data_out <= "01";
                else
                    data_out <= "11";
                end if;
            when s1=>
                if dir = '1' then
                    data_out <= "10";
                else
                    data_out <= "00";
                end if;
            when s2=>
                if dir = '1' then
                    data_out <= "11";
                else
                    data_out <= "01";
                end if;
            when s3=>
                if dir = '1' then
                    data_out <= "00";
                else
                    data_out <= "10";
                end if;
        end case;
    end process;
end rtl;
```



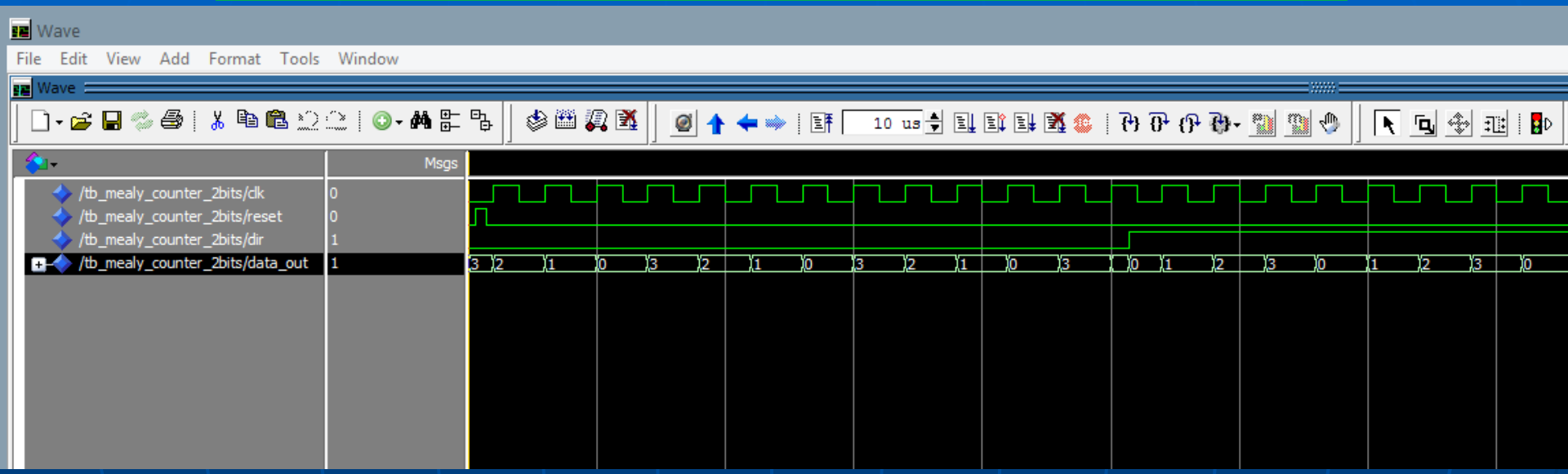
Síntesis por Mealy en VHDL

TEST BENCH DESCRIPTO EN VHDL del contador sintetizado con Mealy

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity tb_mealy_counter_2bits is
7  end tb_mealy_counter_2bits;
8
9  architecture test of tb_mealy_counter_2bits is
10
11    component mealy_counter_2bits
12       Port (   clk      : in std_logic;
13                dir      : in std_logic;
14                reset    : in std_logic;
15                data_out : out std_logic_vector(1 downto 0)
16           );
17    end component;
18
19 signal clk           : std_logic;
20 signal reset         : std_logic;
21 signal dir           : std_logic;
22 signal data_out      : std_logic_vector(1 downto 0);
23
24 begin
25
26  uut: mealy_counter_2bits port map (  clk => clk,
27                                       dir => dir,
28                                       reset => reset,
29                                       data_out => data_out
30                                    );
31
32    gen_reloj : process  -- Reloj de 200 ns de periodo y 50% de ciclo de trabajo
33       begin
34          clk <= '0';
35          wait for 100 ns;
36          clk <= '1';
37          wait for 100 ns;
38    end process gen_reloj;
39
40    estimulos : process
41
42    begin
43             dir   <= '0';
44             reset <= '0';
45             wait for 30 ns;
46             reset <= '1';
47             wait for 40 ns;
48             reset  <= '0';
49             wait for 2.5 us;
50             dir <= '1';
51             wait for 2.5 us;
52
53    end process estimulos;
54
55 end test;
```

Síntesis por Mealy en VHDL

Simulación del contador implementado con máquina de Mealy

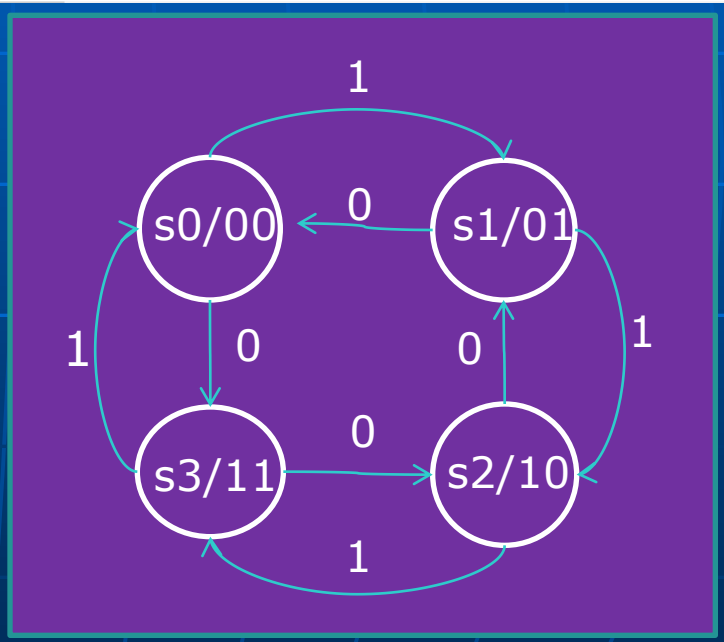


Síntesis por Moore en VHDL

Síntesis de contador up-down con máquina de Moore. Descripción en VHDL

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity moore_counter_2bits is

    port
    (
       clk      : in  std_logic;
       dir      : in  std_logic;
       reset    : in  std_logic;
       data_out : out std_logic_vector(1 downto 0)
    );

end entity;

architecture rtl of moore_counter_2bits is

    type state_type is (s0, s1, s2, s3);

    signal state : state_type;

begin
    process (clk, reset, dir)
    begin
       if reset = '1' then
          state <= s0;
       elsif (rising_edge(clk)) then
          case state is
             when s0=>
                if dir = '1' then
                   state <= s1;
                else
                   state <= s3;
                end if;
             when s1=>
                if dir = '1' then
                   state <= s2;
                else
                   state <= s0;
                end if;
             when s2=>
                if dir = '1' then
                   state <= s3;
                else
                   state <= s1;
                end if;
             when s3=>
                if dir = '1' then
                   state <= s0;
                else
                   state <= s2;
                end if;
          end case;
       end if;
    end process;
```

```vhdl

    process (state)
    begin
       case state is
          when s0=>
                data_out <= "00";
          when s1=>
                data_out <= "01";
          when s2=>
                data_out <= "10";
          when s3=>
                data_out <= "11";
       end case;
    end process;
end rtl;

```



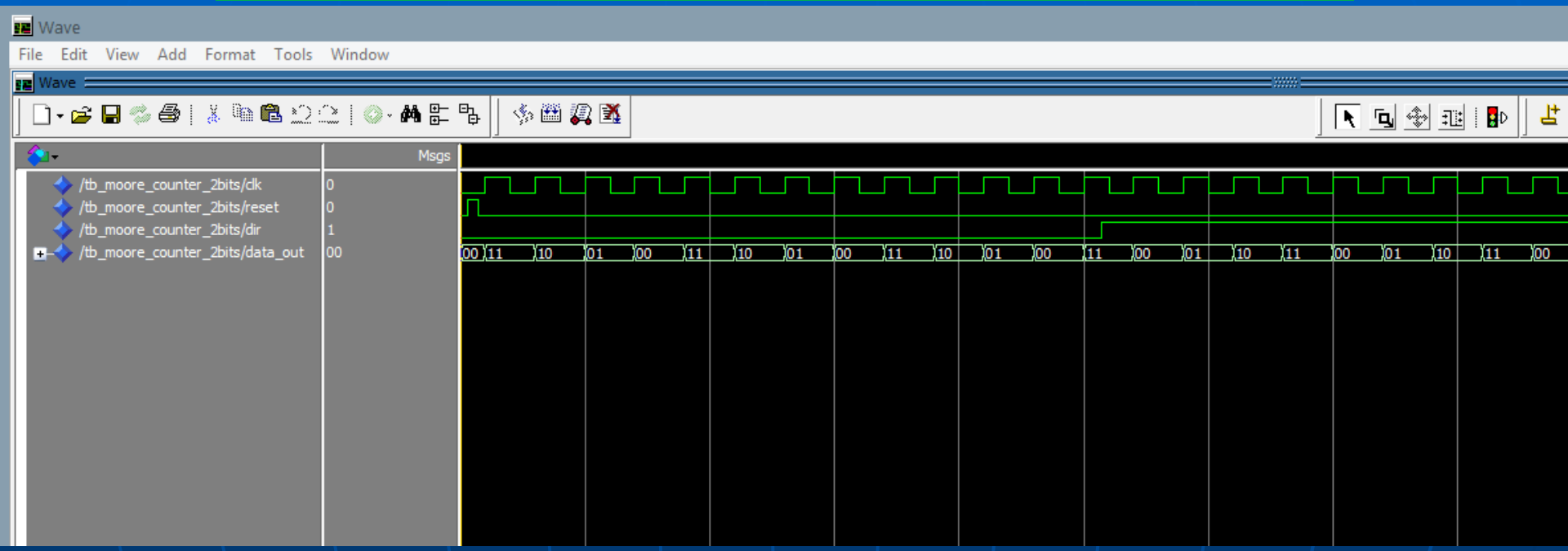
Síntesis por Moore en VHDL

TEST BENCH DESCRIPTO EN VHDL del contador sintetizado con Moore

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity tb_moore_counter_2bits is
7  end tb_moore_counter_2bits;
8
9  architecture test of tb_moore_counter_2bits is
10
11    component moore_counter_2bits
12       Port (   clk      : in std_logic;
13                dir      : in std_logic;
14                reset    : in std_logic;
15                data_out : out std_logic_vector(1 downto 0)
16           );
17    end component;
18
19 signal clk           : std_logic;
20 signal reset         : std_logic;
21 signal dir           : std_logic;
22 signal data_out      : std_logic_vector(1 downto 0);
23
24 begin
25
26  uut: moore_counter_2bits port map (  clk => clk,
27                                       dir => dir,
28                                       reset => reset,
29                                       data_out => data_out
30                                    );
31
32    gen_reloj : process  -- Reloj de 200 ns de periodo y 50% de ciclo de trabajo
33       begin
34          clk <= '0';
35          wait for 100 ns;
36          clk <= '1';
37          wait for 100 ns;
38    end process gen_reloj;
39
40    estimulos : process
41
42    begin
43             dir   <= '0';
44             reset <= '0';
45             wait for 30 ns;
46             reset <= '1';
47             wait for 40 ns;
48             reset  <= '0';
49             wait for 2.5 us;
50             dir <= '1';
51             wait for 2.5 us;
52
53  end process estimulos;
54
55 end test;
```

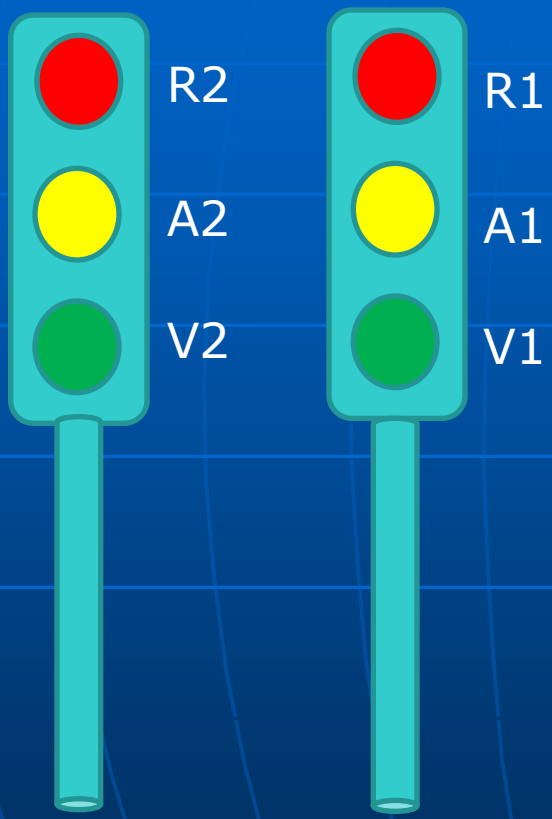
Síntesis por Moore en VHDL

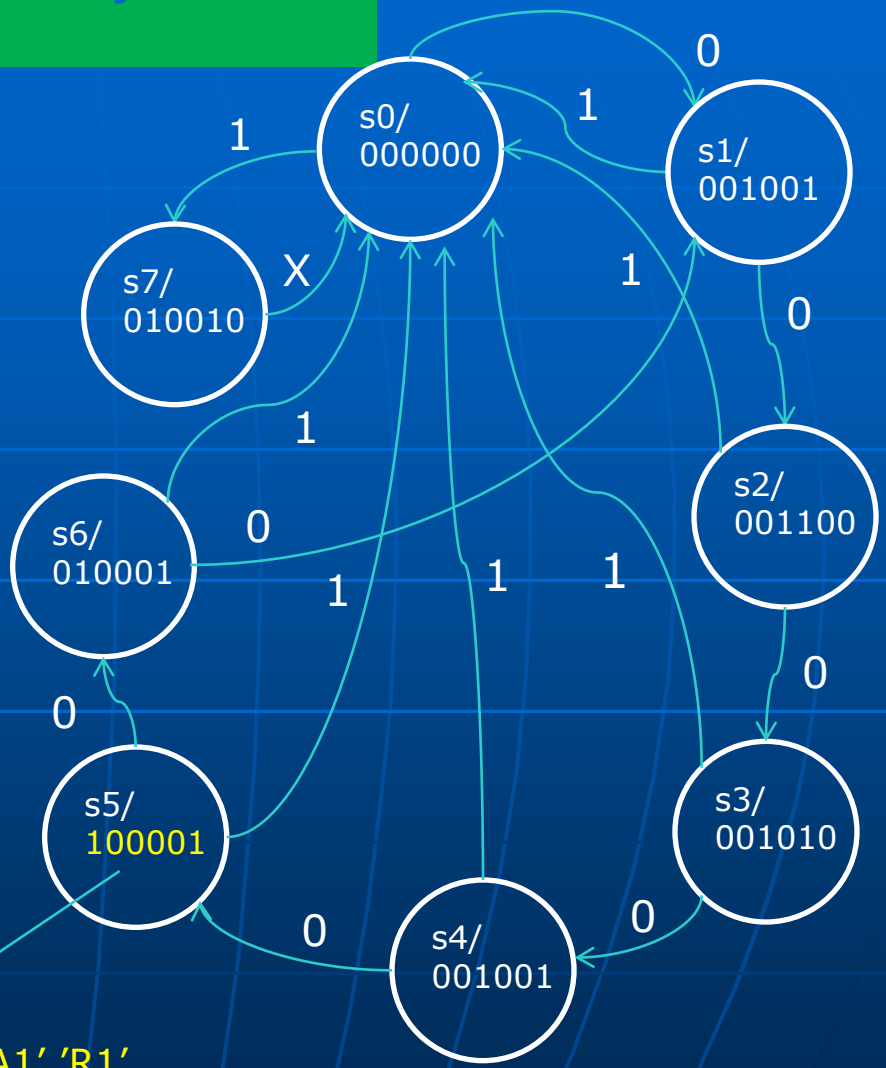
Simulación del contador implementado con máquina de Moore



Síntesis por Moore en VHDL

Síntesis de un control de dos semáforos implementado con máquina de estados Moore y ciclo de reloj variable. Descripción en VHDL





Síntesis por Moore en VHDL

```
1  --*************************************************************************************************
2  -- Descripcion en VHDL de un controlador de semaforos
3  -- En este proyecto se describe un controlador de dos semaforos "1" y "2"con modos
4  -- diurno y nocturno, controlado por la entrada "modo".
5  -- Las salidas de las 3 lamparas por semaforo se presentan en "salida" donde:
6  -- salida(5) = luz verde semaforo 2, V2.
7  -- salida(4) = luz amarilla semaforo 2, A2.
8  -- salida(3) =  luz roja semaforo 2, R2.
9  -- salida(2) = luz verde semaforo 1, V1.
10 -- salida(1) = luz amarilla semaforo 1, A1.
11 -- salida(0) =  luz roja semaforo 1, R1.
12 -- La secuencia de encendido de las lmaparas se hace de la siguiente manera:
13 -- Modo diurno (modo = '0'): R2-R1 -> R2-V1 -> R2-A1 -> R2-R1 -> V2-R1 -> A2-R1 -> REPITE
14 -- Modo nocturno (modo = '1'): apagago-apagado -> A2-A1 -> REPITE.
15 -- Para hacer el sistema mas realistico se definen ciclos de reloj variable segun en que
16 -- estado se encuentre:
17 -- R2-R1 dura  3000 ms.          0BB8 EN HEXADECIMAL.
18 -- R2-V1 dura 20000 ms.          4E20 "       "     .
19 -- R2-A1 y A2-R1 duran 2000 ms.  07D0 "       "     .
20 -- V2-R1 dura 40000 ms.          9C40 "       "     .
21 -- apagado-apagado dura 1000 ms. 03E8 EN HEXADECIMAL.
22 -- A2-A1 dura 1000 ms.           03E8 EN HEXADECIMAL.
23 -- Para ello se usa un reloj base de 50 MHz que se divide hasta 1KHz (1 ms de periodo).
24 -- Ese reloj entra a un contador preseteable con un dato que se modifica (count_max) cada vez
25 -- que se pasa a un nuevo estado.
26 -- Resultado: Funciona OK.
27 -- Archivo: semaforo.vhd
28 -- Sergio Noriega ISLD 2016
29 -- *************************************************************************************************
30
31 library ieee;
32 use ieee.std_logic_1164.all;
33 use ieee.std_logic_arith.all;
34 use ieee.std_logic_unsigned.all;
35
36 entity semaforo is
37
38    Port ( clock    : in std_logic;    --ENTRADA DE 50 MHZ
39           modo     : in std_logic;    --Entrada para ajuste diurno-nocturno
40           reset    : in std_logic;    --Reset
41           salida   : out std_logic_vector (5 downto 0)
42         );
43
44 end semaforo;
45
46 architecture Comportamiento of semaforo is
47
48    signal clock_1ms, clock_div, temporal, clock_prog : std_logic;
49    signal count_max                    : std_logic_vector (15 downto 0); --Hasta 65 segundos
50    signal counter                      : integer range 0 to 24999 := 0;
51    signal counter2                     : integer range 0 to 60000 := 0;
52
53    type state_type is (s0, s1, s2, s3, s4, s5, s6, s7);
54    signal estado : state_type;
55
```

Síntesis por Moore en VHDL

```
56  begin
57
58   divisor_de_frec: process (reset, clock) -- Genera un reloj de 1ms de periodo
59                       begin
60                        if reset = '1' then temporal <= '0';
61                                            counter <= 0;
62                         elsif rising_edge(clock) then
63                          if (counter = 24999) then
64                             temporal <= NOT(temporal);
65                             counter <= 0;
66                          else
67                             counter <= counter + 1;
68                          end if;
69                          end if;
70                    end process;
71
72   clock_1ms <= temporal; --Onda cuadrada de 1KHz y 50% de duty cycle.
73
74  clock_var :   process (clock_1ms, reset)  --Genera el reloj de periodo variable
75                 begin
76                    if rising_edge (clock_1ms) then
77                     if counter2 < count_max  then
78                     clock_prog <= '0';
79                    counter2 <= counter2 + 1;
80                    else counter2 <= 0; clock_prog <= '1';
81                    end if;
82                    end if;
83                end process;
84
85         clock_div <= clock_prog;
86
87  gen_salidas:  process (clock_div, reset, modo)
88                 begin
89                    if reset = '1' then estado <= s0;
90                    elsif (clock_div'event and clock_div = '1') then
91                       case estado is
92                          when s0 => if modo = '0' then estado <= s1;
93                                     else estado <= s7; end if;
94                          when s1 => if modo = '0' then estado <= s2;
95                                     else estado <= s0; end if;
96                          when s2 => if modo = '0' then estado <= s3;
97                                     else estado <= s0; end if;
98                          when s3 => if modo = '0' then estado <= s4;
99                                     else estado <= s0; end if;
100                         when s4 => if modo = '0' then estado <= s5;
101                                    else estado <= s0; end if;
102                         when s5 => if modo = '0' then estado <= s6;
103                                    else estado <= s0; end if;
104                         when s6 => if modo = '0' then estado <= s1;
105                                    else estado <= s0; end if;
106                         when s7 => if modo = '0' then estado <= s0;
107                                    else estado <= s0; end if;
108                      end case;
109                   end if;
110                end process;
```

Este proceso genera un reloj de 1KHz y 50% de ciclo de trabajo.

Este proceso genera un reloj de período variable en base al reloj anterior. En ‘0’ estará ‘count_max’ x 1ms y en ‘1’ 1ms.

Este proceso junto al siguiente forman la máquina de estado Moore que responde a la entrada ‘modo’.

Síntesis por Moore en VHDL

```
111
112         process (estado)
113            begin
114               case estado is
115                  when s0 => salida <= "000000"; --Todas las luces apagadas
116                             count_max <= x"03E8"; -- 1 segundo
117                  when s1 => salida <= "001001"; --Prendidas R2 y R1
118                             count_max <= x"0BB8"; -- 3 segundos
119                  when s2 => salida <= "001100"; --Prendidas R2 y V1
120                             count_max <= x"4E20"; --  20 segundos
121                  when s3 => salida <= "001010"; --Prendidas R2 y A1
122                             count_max <= x"07D0";  -- 2 segundos
123                  when s4 => salida <= "001001"; --Prendidas R2 y R1
124                             count_max <= x"0BB8";  -- 3 segundos
125                  when s5 => salida <= "100001"; --Prendidas V2 y R1
126                             count_max <= x"9C40";  -- 40 segundos
127                  when s6 => salida <= "010001"; --Prendidas A2 y R1
128                             count_max <= x"07D0";  -- 2 segundos
129                  when s7 => salida <= "010010"; --Prendidas A2 y A1
130                             count_max <= x"03E8";  -- 1 segundo
131               end case;
132         end process;
133
134   end Comportamiento;
135
136
```

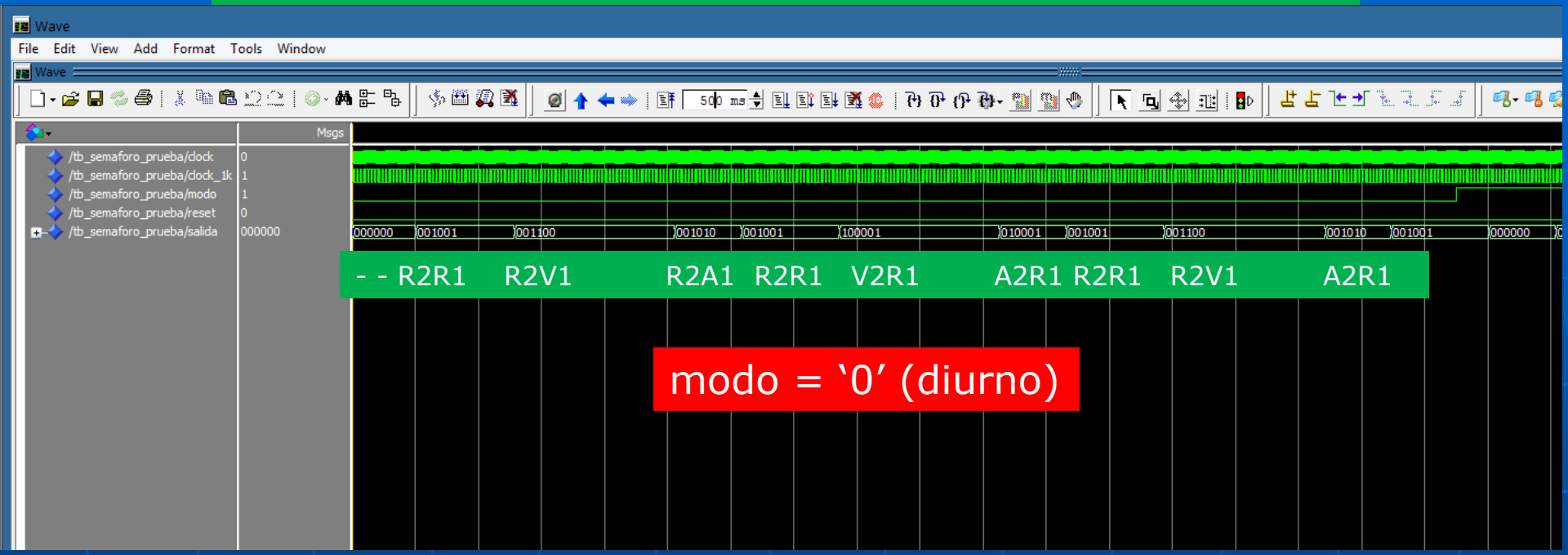
En este segundo proceso de la máquina de estados, se definen los valores de 'salida', y los valores de precarga (count_max) del proceso 'clock_var' que sirve para generar el reloj de período variable de dicha máquina de estados.
Ejemplo: Para el estado V2R1 el valor en hexa es '9C40' ó 40.000, es decir, el período de 'clock_div' será 1 ms x 40.000 = 40 segundos. Ese es el tiempo en que estarán las lámparas V2 y R1 encendidas.

Síntesis por Moore en VHDL

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity tb_semaforo is
end tb_semaforo;

architecture test of tb_semaforo is

   component semaforo
   Port ( clock    : in std_logic;
          modo     : in std_logic;
          reset    : in std_logic;
          salida   : out std_logic_vector(5 downto 0)
        );
   end component;

   signal clock    : std_logic;
   signal modo     : std_logic;
   signal reset    : std_logic;
   signal salida   : std_logic_vector (5 downto 0);

begin

  uut: semaforo port map      (     clock => clock,
                                    modo => modo,
                                    reset => reset,
                                    salida => salida
                              );

   gen_reloj : process -- Reloj de 20 ns de periodo y 50% de ciclo de trabajo
      begin
         clock <= '0';
         wait for 10 ns;
         clock <= '1';
         wait for 10 ns;
   end process gen_reloj;

   estimulos : process
   begin
      reset <= '0';
      modo <= '0';
      wait for 300ns;
      reset <= '1';
      wait for 500ns;
      reset <= '0';
      wait for 100000 ns;

  end process estimulos;
end test;
```

Síntesis por Moore en VHDL

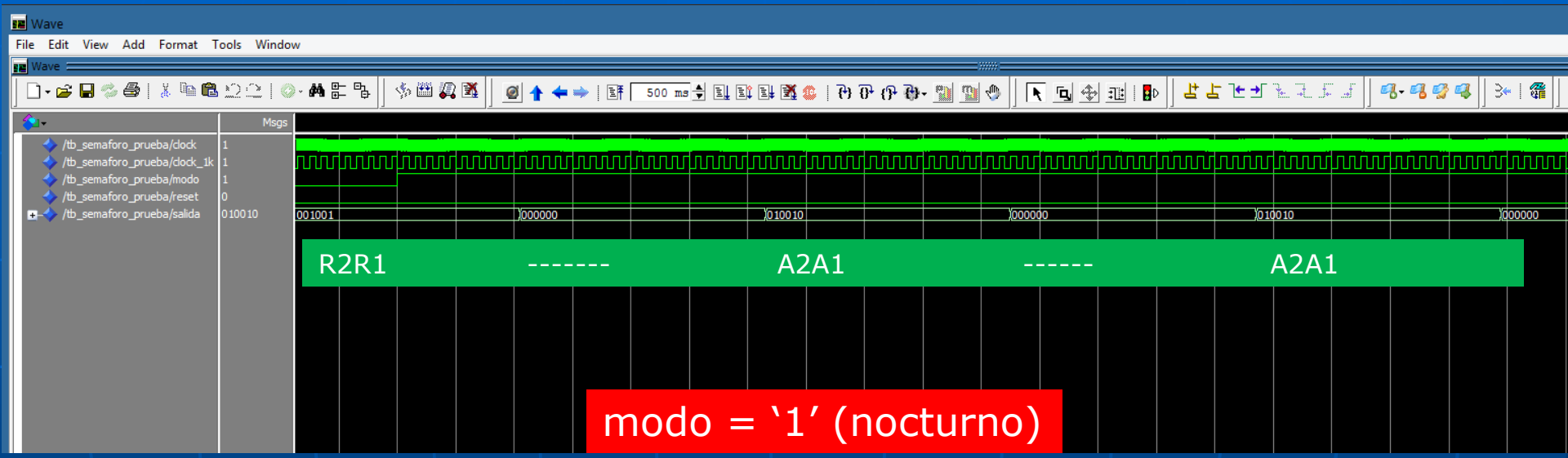
Simulación del semáforo implementado con máquina de Moore



Esta simulación se hizo en otro proyecto donde solo se agregó la salida de reloj de 1KHz para test y modificando los valores de ‘count_max’ a fin de disminuir los tiempos de simulación

Síntesis por Moore en VHDL

Simulación del semáforo implementado con máquina de Moore



Análisis y Síntesis

Método de síntesis por HDL (Lenguaje de Descripción de Hardware)

Empresas tales como Altera, Xilinx, Micro Semi, Lattice, QuickLogic, etc., brindan una serie de dispositivos lógicos programables por hardware y sus correspondientes herramientas para diseño de pequeños volúmenes de producción, siendo realizados directamente por el usuario.

Otras proveen soluciones para grandes volúmenes de fabricación, denominadas ASICs (Application Specific Integrated Circuit), tales como Texas Instruments, Microchip, IBM, LSI Logic, etc.

Para cualquiera de estas opciones, existen soluciones que permiten diseñar y testear los diseños empleando técnicas de descripción textual de sistemas y componentes, empleando lenguajes conocidos como de «descripción de hardware», siendo los mas relevantes VHDL y Verilog.

Con el avance de las técnicas de fabricación ya se encuentran incluso disponibles plataformas con soluciones que combinan las necesidades de integrar tanto lo analógico como lo digital.

Análisis y Síntesis

Bibliografía:

Apuntes de teoría:

- "Análisis y Síntesis". S. Noriega.

Libros:

- "Sistemas Digitales". R. Tocci, N. Widmer, G. Moss. Ed. Prentice Hall.
- "Diseño Digital". M. Morris Mano. Ed. Prentice Hall. 3ra edición.
- "Diseño de Sistemas Digitales". John Vyemura. Ed. Thomson.
- "Diseño Lógico". Antonio Ruiz, Alberto Espinosa. Ed. McGraw-Hill.
- "Digital Design:Principles & Practices". John Wakerly. Ed. Prentice Hall.
- "Diseño Digital". Alan Marcovitz. Ed. McGraw-Hill.
- "Electrónica Digital". James Bignell, R. Donovan. Ed. CECSA.
- "Técnicas Digitales con Circuitos Integrados". M. Ginzburg.
- "Fundamentos de Diseño Lógico y Computadoras". M. Mano, C. Kime. Ed. Prentice Hall.
- "Teoría de conmutación y Diseño lógico". F. Hill, G. Peterson. Ed. Limusa