ELO211: Sistemas Digitales

Tomás Arredondo Vidal

1er Semestre – 2008

Este material está basado en:

- textos y material de apoyo: *Contemporary Logic Design 1st / 2nd* edition. Gaetano Borriello and Randy Katz. Prentice Hall, 1994, 2005
- material del curso ELO211 del Prof. Leopoldo Silva
- material en el sitio http://es.wikipedia.org

12-Simplificación de Maquinas Sincrónicas

Simplificación de MEFs

- Minimización de estados
  - menos estados requieren menos bits de estados
  - menos bits requieren menos ecuaciones lógicas y FFs
- Codificaciones: estados, inputs, outputs
  - codificaciones con menos bits tienen menos ecuaciones a implementar
    - pero cada una puede ser mas compleja
  - ecuaciones de estado con mas bits (e.g., uno encendido) tiene ecuaciones mas simples
    - complejidad relacionada a complejidad de diagrama de estados
  - codificación de input/output puede o no estar bajo control del diseñador

Algoritmo para minimización de estados

- Objetivo – identificar y combinar estados que tienen comportamiento equivalente
- Estados equivalentes:
  - mismo output
  - para todas las combinaciones de inputs, estados transicionan a los mismos o equivalentes estados
- Algoritmo
  1. poner todos los estados en un conjunto (P0)
  2. inicialmente dividir conjunto basado en comportamiento del output (P1)
  3. sucesivamente dividir subconjuntos resultantes basado en próximas transiciones de estados (P2, P3,...)
  4. repetir (3) hasta que no se requieran mas divisiones
     - estados que quedan en mismo conjunto son equivalentes
  - procedimiento de tiempo polinomial

Actividad

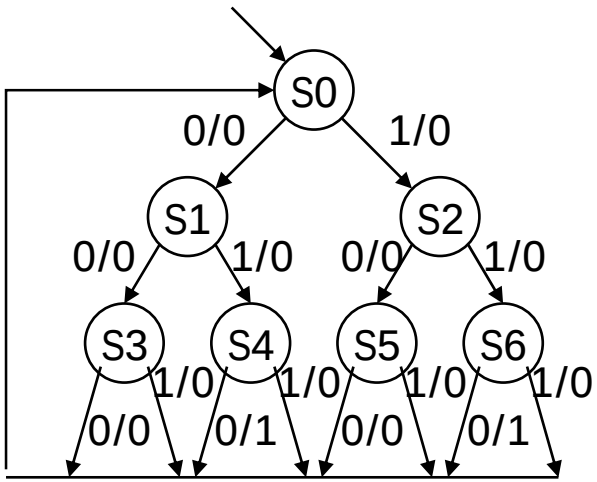
- Implementar detector de secuencia (usando Mealy) para 010 o 110 (detector no continuo, requiere secuencia completa) y simplificar usando método de particiones sucesivas

Actividad (cont)

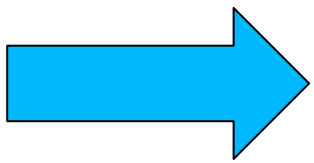
MEF Minimizada

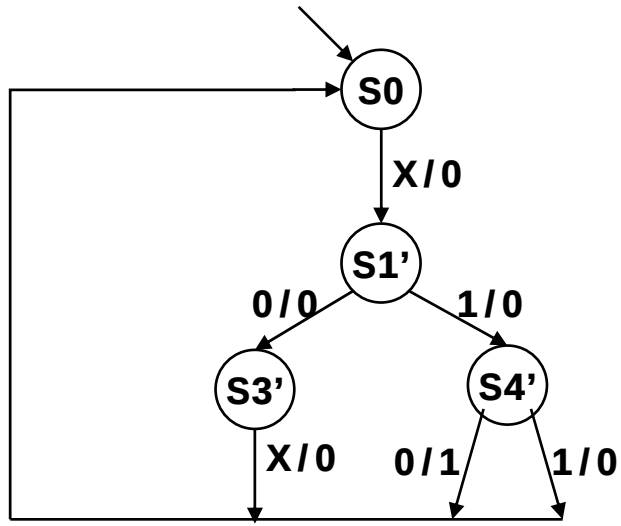
- Detector de secuencia de 010 o 110 minimizada para estados (detecta secuencia completa)

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S0	S1'	S1'	0	0
0 + 1	S1'	S3'	S4'	0	0
X0	S3'	S0	S0	0	0
X1	S4'	S0	S0	1	0



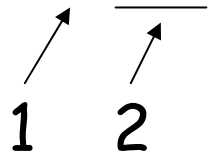
Original

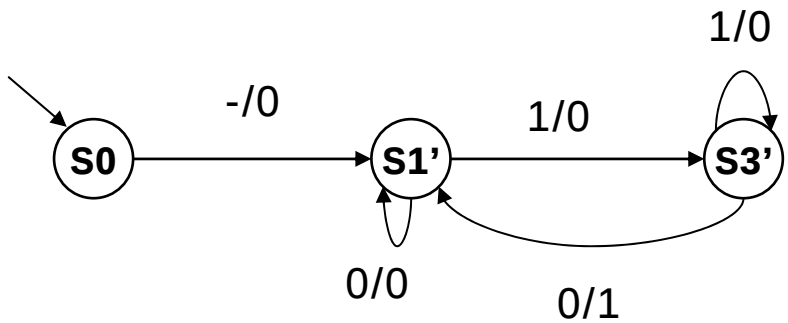




MEF Minimizada

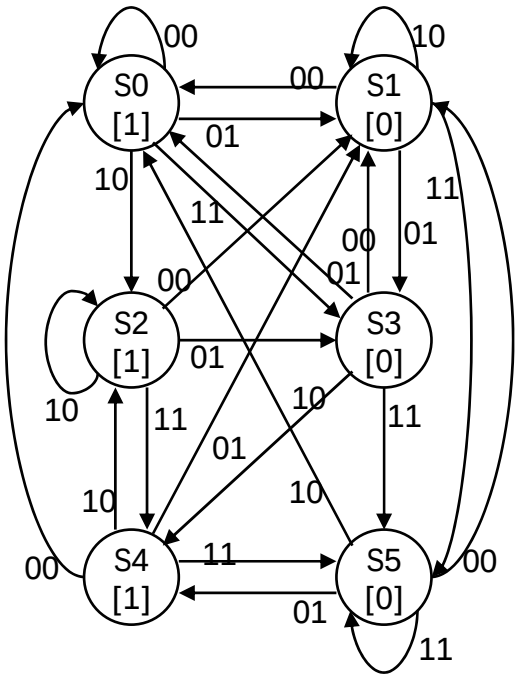
- Otra solución para el detector de secuencia de 010 o 110 (detector continuo)
- Esta versión detecta subsecuencias como secuencias validas (e.g. 01010 se detectaría como dos secuencias)





Minimización mas compleja: Tabla de Implicantes

- Ejemplo: múltiples inputs



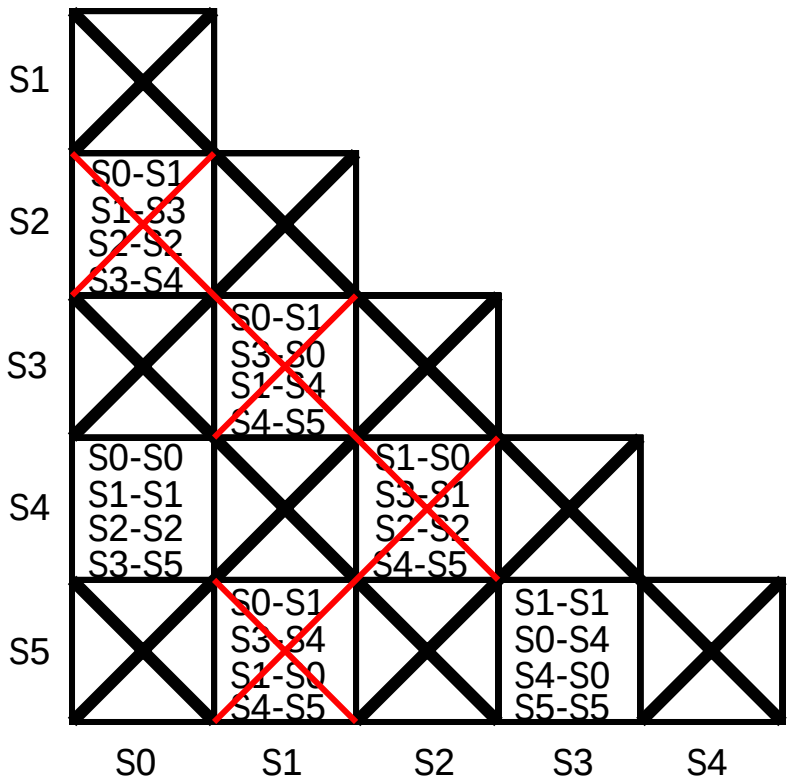
inputs

present state	next state 00	01	10	11	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S4	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

tabla de transición de estados

MEF Minimizada

- Método de tabla de implicantes para ver estados equivalentes
  - eliminar estados incompatibles basados en outputs
  - después eliminar mas celdas si es que entradas indexadas en tabla ya están eliminadas



inputs

present state	next state 00	01	10	11	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S4	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

tabla de estados original

present state	next state 00	01	10	11	output
S0'	S0'	S1	S2	S3'	1
S1	S0'	S3'	S1	S0'	0
S2	S1	S3'	S2	S0'	1
S3'	S1	S0'	S0'	S3'	0

tabla de estados minimizada
(S0==S4) (S3==S5)

Minimizando MEFs especificadas incompletamente

- Equivalencia de estados es transitiva cuando las maquina es completamente especificado (si a=b y b=c entonces a=c)
- Pero no es transitiva cuando hay don't cares presentes

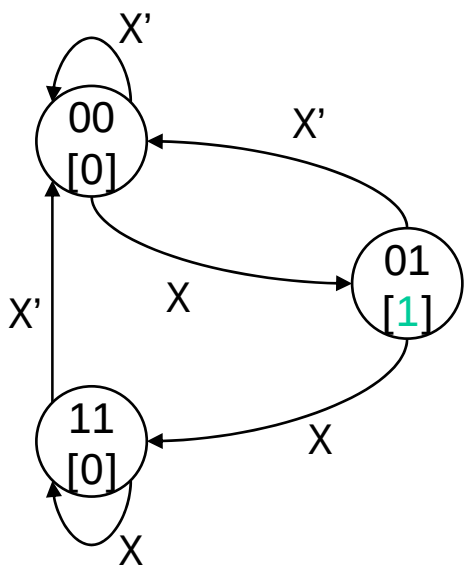
e.g.,

estado	output	
S0	- 0	S1 es compatible con S0 y S2
S1	1 -	pero S0 y S2 son incompatibles
S2	- 1	

- No hay algoritmos de tipo polinomial para determinar mejores agrupamiento de estados a conjuntos equivalentes que resultara en el mínimo numero de estados finales

Minimizando estados a veces puede no dar el mejor circuito final

- Esto ocurre cuando se complican mucho las ecuaciones lógicas para obtener un numero menor de estados (bits)
- Ejemplo: detector de cantos, el output es 1 cuando los dos últimos inputs van de 0 a 1



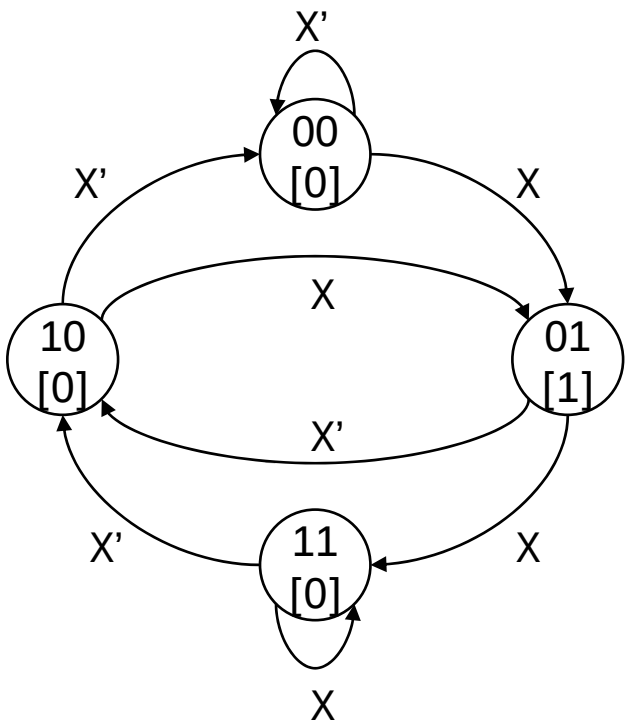
X	Q_1	Q_0	Q_1^+	Q_0^+
0	0	0	0	0
0	0	1	0	0
0	1	1	0	0
1	0	0	0	1
1	0	1	1	1
1	1	1	1	1
–	1	0	0	0

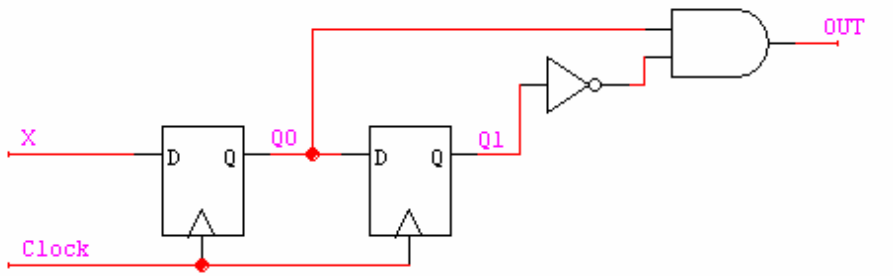
$Q_1^+ = X \ (Q_1 \text{ xor } Q_0)$

$Q_0^+ = X\, Q_1'\, Q_0'$, OUT $= Q_1'\, Q_0$

Otra implementación

- Solución "ad hoc" - no es minima pero es poco costosa y rapida





12-Simplificación de Maquinas Sincrónicas

12.1 Simplificación de Maquinas Sincrónicas

12.2 Asignación de estados

Asignación de estados

- Problema: Asignar códigos de bits para asignar a cada estado simbólico (e.g. A, B, C...)
  - numero mínimo m de FFs para codificar e estados:

    $2^{m-1} < e \leq 2^m$

  - permutaciones cuando se seleccionan k elementos de un set de n posibles:

    $(n)\ (n-1)\ (n-2)...(n-k-1) = n!/(n-k)!$

  - usando códigos de n bits de largo: se tienen 2^n códigos posibles para 1^{er} estado, 2^n-1 para 2^{ndo}, 2^n-2 para 3^{ro}, ...

  - numero de combinaciones posibles para codificar m estados usando códigos de n bits de largo ($\log n <= m <= 2^n$):

    $(2^n)\ (2^n-1)\ (2^n-2)...(2^n-m-1) = 2^n! / (2^n - m)! = ae$

  - numero enorme hasta para pequeños valores de n y m
    - intractable para maquinas de estados grandes
    - heurística es necesaria para soluciones practicas

Estrategias para asignación de estados

- Estrategias posibles
  - secuencial – simplemente enumerar estados en la forma que aparecen en la tabla de estados (puede usar codigo Gray)
  - aleatorio – elegir codigos aleatoriamente (random codes)
  - 1 activo (one-hot) – usar el mismo numero de bits de estados que el numero de estados (1 bit por estado)
  - heurísticas – reglas o algoritmos que parecen funcionar en la mayoria de los casos
  - output – usar los outputs para ayudar a codificar los estados
- No hay garantías de obtener óptimos resultados

Asignación 1-activo (one-hot)

- Simple
  - fácil de codificar
  - fácil de depurar
- Funciones lógica pequeñas
  - cada función de estado requiere solo bits de estado como inputs
- Bueno para PLDs (programmable logic devices)
  - muchos flip-flops disponibles
  - funciones simples con dependencia en pocas variables
- No practico para MEFs grandes
  - muchos estados requiere muchos FFs
  - se puede descomponer MEFs en unidades mas pequeñas que pueden usar codificación one-hot
- Muchas pequeñas variaciones a one-hot

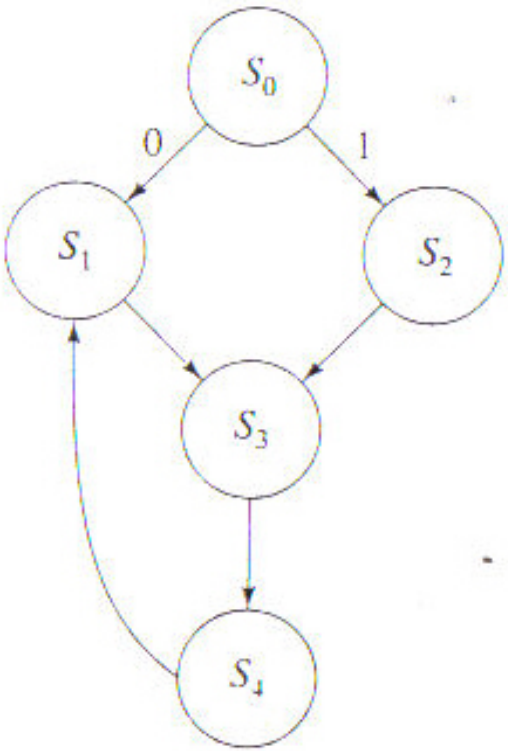
Heurísticas para asignación de estados

- Usan mapas de estados similares a mapas de Karnaugh para ver las adyacencias en asignación de estados, los cuadrados se indexan con los valores binarios de los bits de estados, el estado asignado se pone en el cuadrado
- Estos se pueden usar hasta casos con seis variables
- Una heuristica asigna estados para tener el mínimo numero de cambios de bits para todas las transiciones
- Esta heuristica probablemente no va a producir las mejores asignaciones (e.g. en un ejemplo por verse de la mínima distancia dada por el uso del codigo Gray no es tan buena como una asignación aleatoria que no minimiza el cambio de bits)

Ejemplo: Codificación usando heurística de mínimo cambio de bits

- Para la MEFs dada usando dos codificaciones distintas (random y mínimo cambio de bits entre transiciones) se obtienen los siguientes mapas de estados

FSM



random

State Name	Assignment Q_2	Q_1	Q_0
S_0	0	0	0
S_1	1	0	1
S_2	1	1	1
S_3	0	1	0
S_4	0	1	1

Assignment

Q_2 \ Q_1Q_0	00	01	11	10
0	S_0		S_4	S_3
1		S_1	S_2	

State Map

mínimo cambio

State Name	Assignment Q_2	Q_1	Q_0
S_0	0	0	0
S_1	0	0	1
S_2	0	1	0
S_3	0	1	1
S_4	1	1	1

Assignment

Q_2 \ Q_1Q_0	00	01	11	10
0	S_0	S_1	S_3	S_2
1			S_4	

State Map

Ejemplo: Codificación usando heurística de mínimo cambio de bits (cont)

- Para la MEFs dada usando dos codificaciones distintas (random y mínimo cambio de bits) se obtienen los siguientes mapas de estados

cambios: random mínimo

Transition	First Assignment Bit Changes	Second Assignment Bit Changes
S_0 to S_1	2	1
S_0 to S_2	3	1
S_1 to S_3	3	1
S_2 to S_3	2	1
S_3 to S_4	1	1
S_4 to S_1	2	2

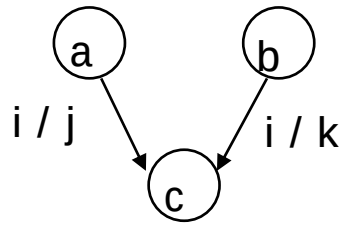
- La heurística del mínimo cambio de bits da solo 7 cambios, es simple pero no garantiza buenos resultados en la implementación del circuito.

Heurísticas para asignación de estados (cont)

- Otra heuristica usa criterios basados en estados y inputs/outputs:
- Usar códigos adjacentes (distancia 1) para estados que comparten un próximo estado (primera prioridad)
  - agrupar 1's en mapa de próximo estado

I	Q	Q^+	O
i	a	c	j
i	b	c	k

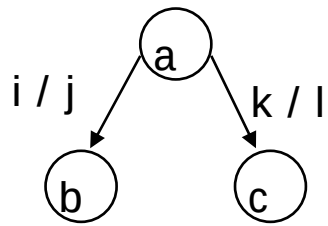
$c = i * a + i * b$



- Usar códigos adyacentes para estados que comparten estado previo (prioridad mediana)
  - agrupar 1's en mapa de próximo estado

I	Q	Q^+	O
i	a	b	j
k	a	c	l

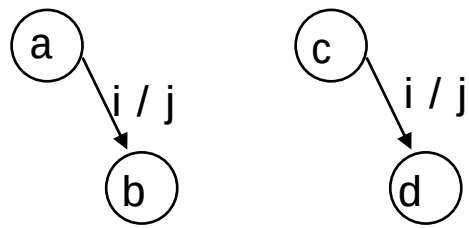
$b = i * a$
$c = k * a$



- Usar códigos adyacentes para estados que tienen un comportamiento de output igual (prioridad baja)
  - agrupar 1's en mapa de output

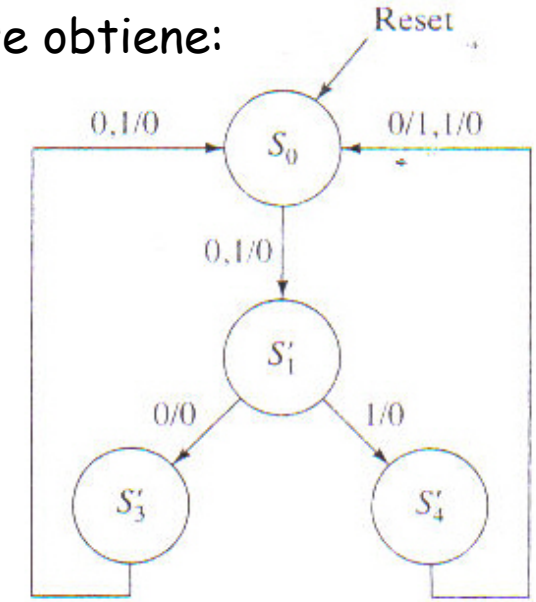
I	Q	Q^+	O
i	a	b	j
i	c	d	j

$j = i * a + i * c$
$b = i * a$
$d = i * c$



Ejemplo: Codificación usando heurística de criterios para asignar estados adyacentes

- Para la MEFs dada usando los criterios de prioridad se obtiene:
  - Prioridad alta: {S3', S4'}
  - Prioridad mediana: {S3', S4'}
  - Prioridad baja: 0/0: {S0, S1', S3'}
    1/0: {S0, S1', S3', S4'}



- Estas son dos posibles asignaciones:

Q_1 \ Q_0	0	1
0	S_0	S_3'
1	S_1'	S_4'

Q_1 \ Q_0	0	1
0	S_0	S_1'
1	S_3'	S_4'

- Ya que la MEF tiene cuatro estados podemos hacer la asignación a dos bits de estado. En general es una buena idea asignar el estado de reset al sitio 0 en el mapa de estados. Se trata de maximizar las agrupaciones de las funciones de próximo estado y output.

Ejemplo: Codificación de controlador de semáforos para intersección (carretera y camino rural) (Katz Cap.8)

- C: Detector de vehículo, TL: Temporizador (timer) largo, TS: Timer corto (short), HG: Carretera verde, rural rojo, HY: Carretera amarillo, rural rojo, FG: Carretera rojo, rural verde, FY: Carretera rojo, rural amarillo
- Outputs: ST: resetear timer, H: semáforo carretera, F: semáforo rural -> 00: Verde, 01: Amarillo, 10: Rojo

Inputs			Estado Actual (Qn)	Estado Próximo (Pn)	Outputs		
C	TL	TS			ST	H1H0	F1F0
0	–	–	HG	HG	0	00	10
–	0	–	HG	HG	0	00	10
1	1	–	HG	HY	1	00	10
–	–	0	HY	HY	0	01	10
–	–	1	HY	FG	1	01	10
1	0	–	FG	FG	0	10	00
0	–	–	FG	FY	1	10	00
–	1	–	FG	FY	1	10	00
–	–	0	FY	FY	0	10	01
–	–	1	FY	HG	1	10	01

Ejemplo: Codificación de controlador de semáforos para intersección (cont)

- Tabla de posibles codificaciones m=4, n=2
- ae = $2^n!$ / $(2^n - m)!$ = $2^2!$ / $(2^2 - 4)!$ = 4! / (0)! = 24 combinaciones

HG	HY	FG	FY	HG	HY	FG	FY
00	01	10	11	10	00	01	11
00	01	11	10	10	00	11	01
00	10	01	11	10	01	00	11
00	10	11	01	10	01	11	00
00	11	01	10	10	11	00	01
00	11	10	01	10	11	01	00
01	00	10	11	11	00	01	10
01	00	11	10	11	00	10	01
01	10	00	11	11	01	00	10
01	10	11	00	11	01	10	00
01	11	00	10	11	10	00	01
01	11	10	00	11	10	01	00

Ejemplo: Codificación de controlador de semáforos para intersección (cont)

- Usando la codificación secuencial (Gray code), HG=00, HY=01, FG=11, FY=10 se generan 7 ecuaciones. Estas ecuaciones son para P1, P0, ST, H1, H0, F1, F0.
  - Estas tienen un fan-in de cinco (tres inputs mas dos bits de estados), son relativamente fácil de implementar en un PLD o FPGA. Estas se pueden optimizar usando lógica multinivel (ver Katz Cap 8)
- Usando una codificación aleatoria, se podría generar: HG=00, HY=10, FG=01, FY=11. Para este caso especifico y sin minimizar las ecuaciones son similares para el caso anterior. Los principales cambios es en algunos literales en las ecuaciones.
  - Si se minimiza (ver Katz Cap 8) las diferencias son mayores y se puede realizar esta lógica con solo dos niveles y con un fan in máximo de 3 inputs. Compuertas mas pequeñas son mas rápidas y también reducen el cableado en el circuito (es útil contar literales para comparar complejidad del circuito).

Ejemplo: Codificación de controlador de semáforos para intersección (cont)

- Usando una codificación 1 activo, se podría usar: HG=0001, HY=0010, FG=0100, FY=1000. El estado se almacena en 4 FFs y se puede usar otro FF para sincronizar el output.
- Presenta muchas oportunidades para simplificar. En los mapas de Karnaugh todos los estados con mas de un bit tendrían un don't care.

Inputs			Estado Actual	Estado Próximo	Outputs		
C	TL	TS	Q3Q2Q1Q0	P3P2P1P0	ST	H1H0	F1F0
0	–	–	0001	0001	0	00	10
–	0	–	0001	0001	0	00	10
1	1	–	0001	0010	1	00	10
–	–	0	0010	0010	0	01	10
–	–	1	0010	0100	1	01	10
1	0	–	0100	0100	0	10	00
0	–	–	0100	1000	1	10	00
–	1	–	0100	1000	1	10	00
–	–	0	1000	1000	0	10	01
–	–	1	1000	0001	1	10	01

Ejemplo: Codificación de controlador de semáforos para intersección (cont)

- Usando la codificación basada en 1 activo del ejemplo anterior se tendrían los siguientes estados (4 FFs):
  - HG = 0001
  - HY = 0010
  - FG = 0100
  - HY = 1000
- Y las siguientes ecuaciones:
  - $P3 = (C' \cdot Q2) + (TL \cdot Q2) + (TS' \cdot Q3)$
  - $P2 = (TS \cdot Q1) + (C \cdot TL' \cdot Q2)$
  - $P1 = (C \cdot TL \cdot Q0) + (TS' \cdot Q1)$
  - $P0 = (C' \cdot Q0) + (TL' \cdot Q0) + (TS \cdot Q3)$
  - $ST = (C \cdot TL \cdot Q0) + (TS \cdot Q1) + (C' \cdot Q2) + (TL \cdot Q2) + (TS \cdot Q3)$
  - $H1 = Q3 + Q2$
  - $H0 = Q1$
  - $F1 = Q1 + Q0$
  - $F0 = Q3$
- Las expresiones son fáciles de implementar en PAL y FPGA para un bajo numero de estados

Ejemplo: Codificación de controlador de semáforos para intersección (cont)

- Usando la codificación basada en outputs se reutilizan los outputs como bits de estado (5 FFs para ST, H1, H0, F1, F0)
  - Porque usar bits para los próximos estados cuando se pueden usar los bits de outputs previos? (e.g. El estado HG puede ocurrir con solo dos patrones de outputs previos: 00010 y 11001)
- El estado actual ahora se representa con los "outputs anteriores" y los outputs actuales con los inputs determinan el proximo estado (como siempre)

Inputs			Estado Actual	Outputs		
C	TL	TS	ST H1 H0 F1 F0	ST	H1H0	F1F0
0	–	–	HG: 00010 + 11001	0	00	10
–	0	–	HG: 00010 + 11001	0	00	10
1	1	–	HG: 00010 + 11001	1	00	10
–	–	0	HY: 10010 + 00110	0	01	10
–	–	1	HY: 10010 + 00110	1	01	10
1	0	–	FG: 10110 + 01000	0	10	00
0	–	–	FG: 10110 + 01000	1	10	00
–	1	–	FG: 10110 + 01000	1	10	00
–	–	0	FY: 11000 + 01001	0	10	01
–	–	1	FY: 11000 + 01001	1	10	01

HG = ST' H1' H0' F1 F0' + ST H1 H0' F1' F0
HY = ST H1' H0' F1 F0' + ST' H1' H0 F1 F0'
FG = ST H1' H0 F1 F0' + ST' H1 H0' F1' F0'
HY = ST H1 H0' F1' F0' + ST' H1 H0' F1' F0

→ Sabemos que estamos en el estado HG si es que los outputs previos son 00010 o 11001. No necesitamos los bits de estados y solo hay que implementar 5 funciones (una para cada output) en vez de 7 (2 para bits de estado)

Ejemplo: Codificación de controlador de semáforos para intersección (cont)

- Usando la codificación basada en outputs del ejemplo anterior se tendrían los siguientes estados (con fan in de 5):
  - HG = ST' H1' H0' F1 F0' + ST H1 H0' F1' F0
  - HY = ST H1' H0' F1 F0' + ST' H1' H0 F1 F0'
  - FG = ST H1' H0 F1 F0' + ST' H1 H0' F1' F0'
  - HY = ST H1 H0' F1' F0' + ST' H1 H0' F1' F0
- Y las siguientes ecuaciones (similares a 1 activo):
  - ST = (C ·TL · HG) + (TS · HY) + (C' · FG) + (TL · FG) + (TS · FY)
  - H1 = FG + FY
  - H0 = HY
  - F1 = HG + HY
  - F0 = FY
- Todavía usamos 5 FFs como en el ejemplo 1 activo (con FF de output sincronizado), pero las ecuaciones para HG, HY, FG y FY suman un poco de complejidad (requiere mas lógica).
- Puede no ser el mejor método para la maquina de estados completa pero se podrían usar algunos de los outputs para eliminar estados

Resumen: Métodos para asignar estados

- Para codificaciones eficientes usar cerca del mínimo numero de bits de estados
  - el mejor de 10 aleatorio es adecuado (en general tan bien como heurística)
  - métodos heurísticas no están cerca de ser óptimos pero funcionan porque tratan de maximizar las agrupaciones de funciones de próximo estado y output (ver Katz 8.2)
  - usado para diseño de chips
- Codificación 1-hot
  - fácil para maquinas de estados pequeñas
  - genera ecuaciones pequeñas con una complejidad fácil de estimar
  - común en FPGAs y otras lógicas programables
- Codificación basada en output
  - ad hoc – sin herramientas
  - método común usado por diseñadores
  - produce circuitos pequeños para la mayoría de MEFs

Resumen de simplificación de lógica secuencial

- Minimización de estados
  - mas fácil en maquinas completamente especificadas
  - en general es computacionalmente muy difícil (con don't cares)
- Asignación de estados
  - muchas heurísticas
  - codificación de outputs es atractivo (especialmente para implementaciones PAL o FPGA)