



Tema 4: Diseño algorítmico

Tecnología y Organización de Computadores

Grado en Ingeniería Informática

Grado en Ingeniería de Computadores

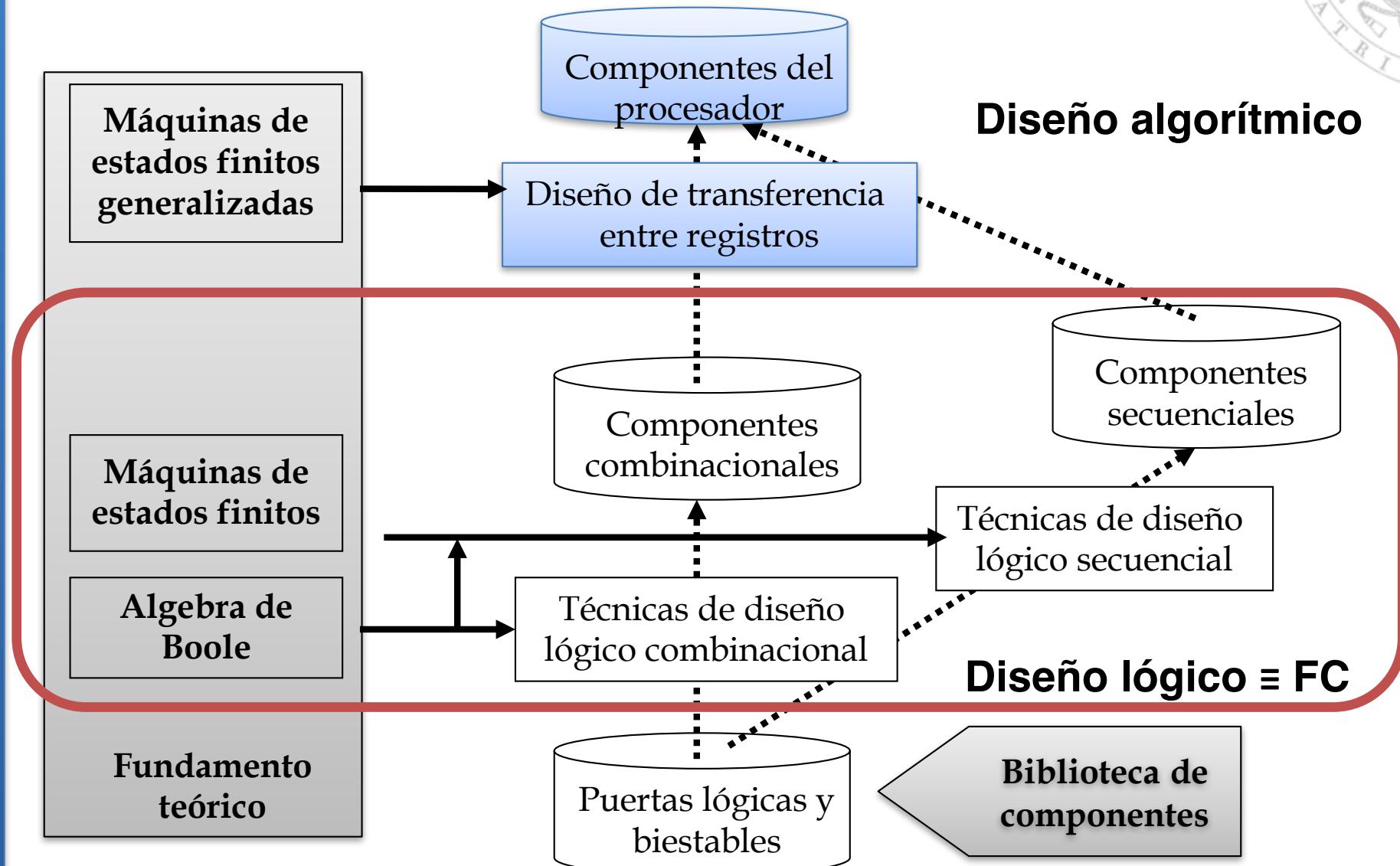
Índice



1. Introducción
2. Conocimientos previos
3. Descripción VHDL de registros
4. Diseño algorítmico
5. Optimización de ASM
6. Principios de diseño
7. Diseño RTL



Flujo de diseño





Conocimientos previos

- Especificación de sistemas secuenciales
 - FSM
 - ¿Dónde? [Fundamentos de computadores](#)
- Implementación de sistemas secuenciales
 - Optimización de estados. Biestables, Flip-flops
 - ¿Dónde? [Fundamentos de computadores](#)

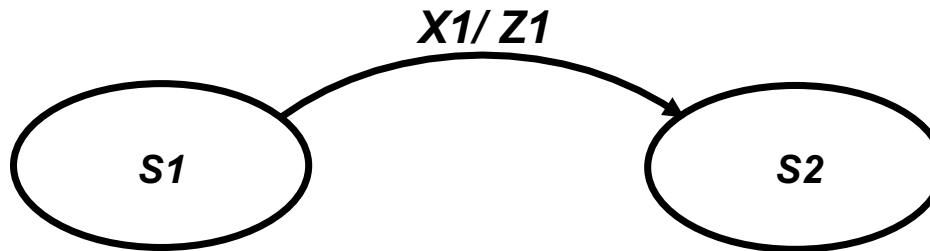


Máquinas de estados finitos (FSM)

- Mealy: Un cambio en la entrada en cualquier instante influye inmediatamente en la salida.

$$Z(t) = H(X(t), S(t))$$

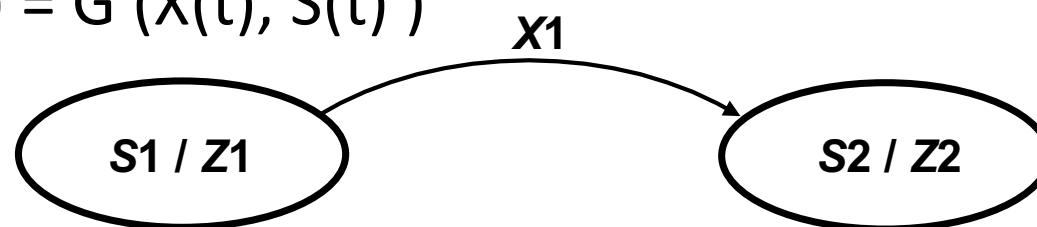
$$S(t+1) = G(X(t), S(t))$$



- Moore: Sólo el cambio del estado influye en la salida.

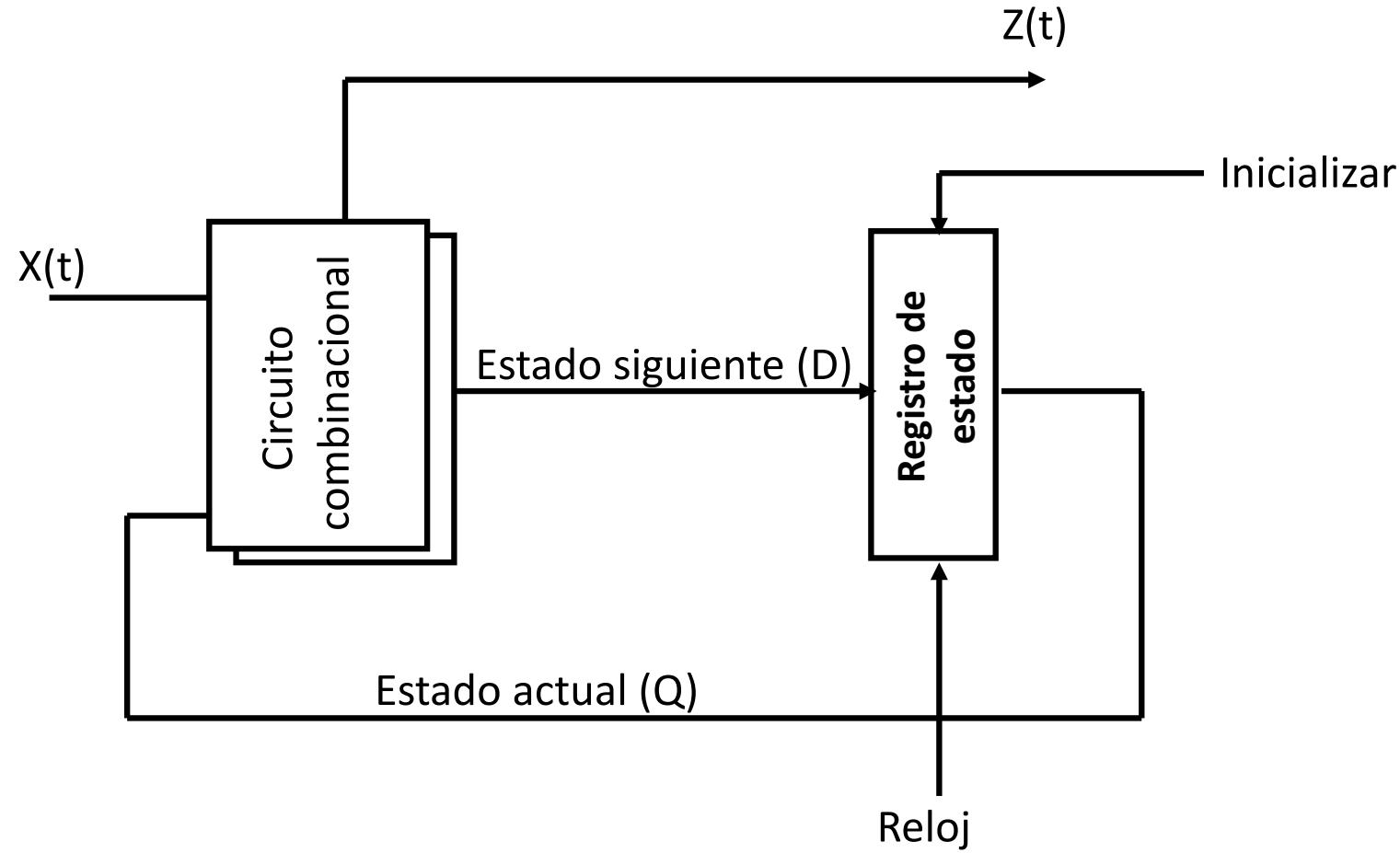
$$Z(t) = H(S(t))$$

$$S(t+1) = G(X(t), S(t))$$



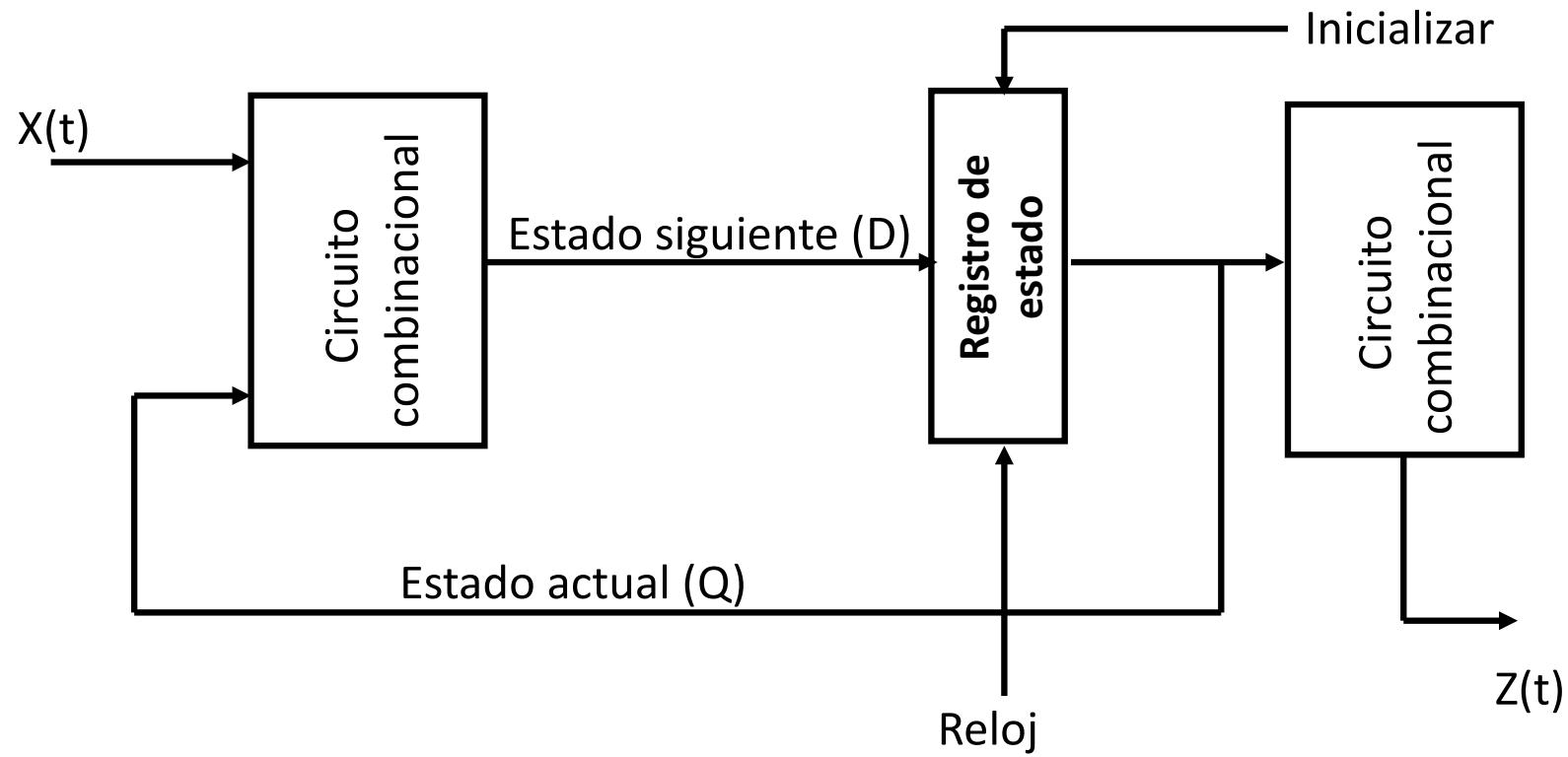


Implementación canónica. Mealy





Implementación canónica. Moore





Hipótesis funcionamiento síncrono

- Presentada en el tema 2.



Elementos de memoria

- Unos de los elementos HW más importantes de un sistema secuencial son los elementos de almacenamiento:
 - Almacenan el estado actual del sistema (máquina de control).
 - Almacenan valores intermedios (registros de datos).
- El elemento de memoria más sencillo es el biestable:
 - Se denomina biestable porque presenta dos únicos estados estables:
 - Salida 0
 - Salida 1
 - Sirve para almacenar un bit de información.



Tipos de biestables

- Según su comportamiento lógico:

- S-R
- D
- J-K
- T

S	R	Q+	D	Q+	J	K	Q+	T	Q+
0	0	Q	0	0	0	0	Q	0	Q
0	1	0	1	1	0	1	0	1	\overline{Q}
1	0	1	1	0	1	0	1	0	1
1	1	proh.			1	1	\overline{Q}		

- Según su comportamiento temporal:

- Latch
- Latch síncrono (sensible a nivel)
- Flip-flop disparado por flanco
- Flip-flop maestro-esclavo

Ecuaciones Características

R-S: $Q+ = S + \overline{R} Q$

D: $Q+ = D$

J-K: $Q+ = J \overline{Q} + \overline{K} Q$

T: $Q+ = T \overline{Q} + \overline{T} Q$

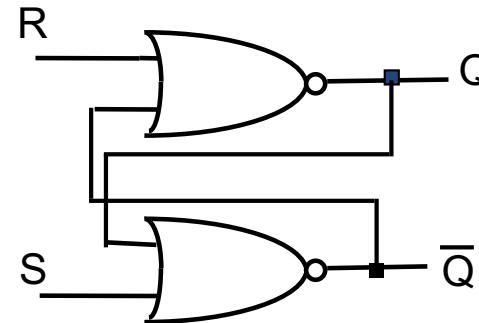
Deducidas a partir de diagramas de K
 para $Q(t+1) = Q+ = f(\text{Entradas}, Q)$



Biestable asíncrono: Latch

- La salida cambia cuando cambian las entradas.
- Ejemplo 1: S-R (con entradas activas a nivel alto).

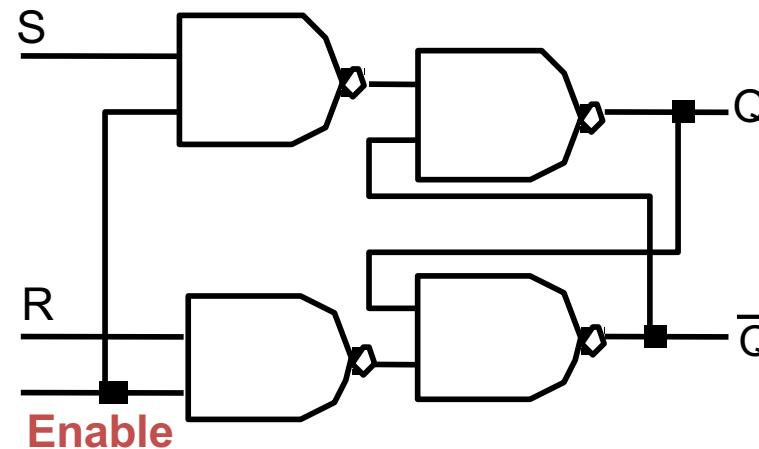
S	R	$Q(t+)$
0	0	$Q(t)$
0	1	0
1	0	1
1	1	proh.





Biestable síncrono sensible a nivel: Latch síncrono

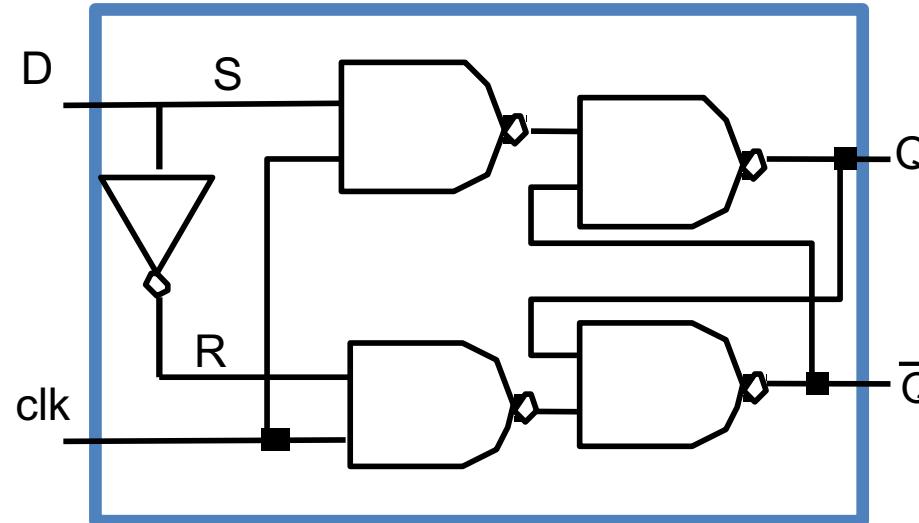
- La salida cambia cuando está activa la señal de capacitación (reloj).
- Ejemplo: S-R con entrada de capacitación.





Biestable síncrono sensible a nivel: Latch síncrono

- ¿Cómo conseguimos que sea un biestable tipo D?



clk	D	
0	0	
0	1	
1	0	
1	1	



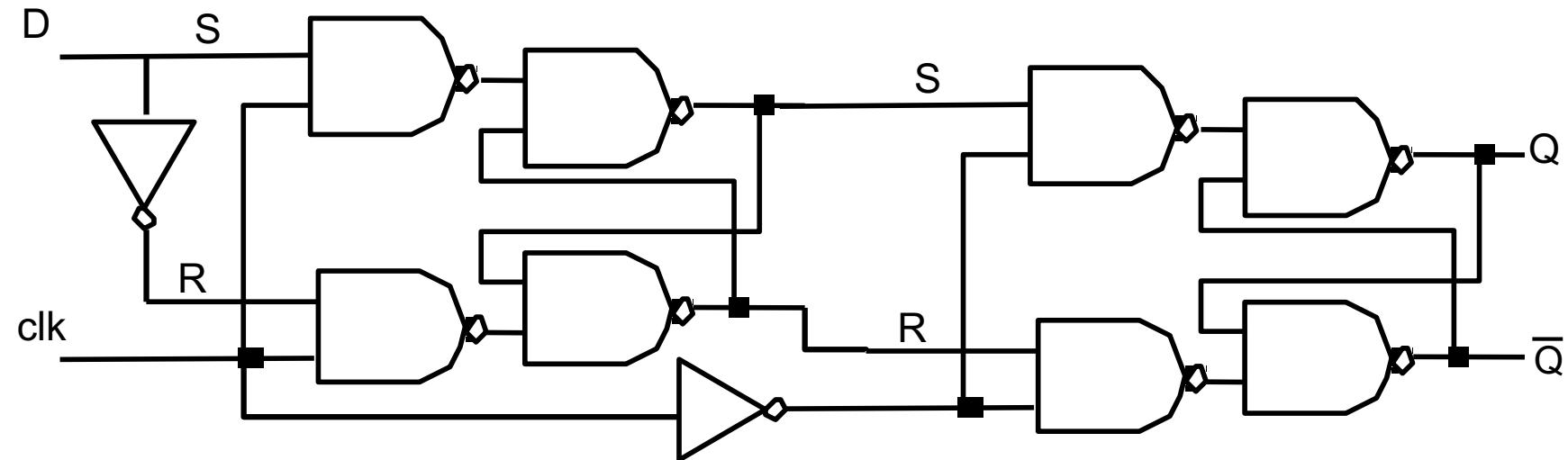
Problemas

- Si usamos latches debemos garantizar que:
 - El pulso de reloj es más corto que el retardo del latch.
 - La entradas se mantienen constantes durante el pulso de reloj.
 - Regla: **NO USAR NUNCA LATCHES**
- Usar FLIP-FLOPs: más fiables.
 - Disparados por flanco: la salida sólo varía durante la transición del reloj (que es la entrada dinámica).
 - Maestro-esclavo.



Biestable maestro-esclavo: FF

- Se lee la entrada en un nivel y se modifica la salida en el contrario
- Implementación física (Recordatorio)





Comparación del comportamiento temporal de los biestables

TIPO	¿Cuándo se muestran las entradas?	¿Cuándo son válidas las salidas?
Latch sin reloj	siempre	retardo de propagación desde el cambio en la entrada
Latch sensible a nivel	reloj en alta (T_{setup} y T_{hold} a cada lado del eje de bajada)	retardo de propagación desde el cambio en la entrada
Flipflop flanco de subida	transición de reloj de baja a alta (T_{setup} y T_{hold} a cada lado del eje de subida)	retardo de propagación desde flanco de subida del reloj
Flipflop flanco de bajada	transición de reloj de alta a baja (T_{setup} y T_{hold} a cada lado del eje de bajada)	retardo de propagación desde flanco de bajada del reloj
Flipflop Maestro-esclavo	transición de reloj de alta a baja (T_{setup} y T_{hold} a cada lado del eje de bajada)	retardo de propagación desde flanco de bajada del reloj



VHDL: FFD

■ FFD reset asíncrono

```
entity ff is
  port (clk : in std_logic;
        rst : in std_logic;
        d   : in std_logic;
        q   : out std_logic);
end ff;

architecture rtl of ff is
begin
  p_ff : process(clk, rst)
  begin
    if rst = '1' then
      q <= '0';
    elsif rising_edge(clk) then
      q <= d;
    end if;
  end process p_ff;
end rtl;
```

■ FFD reset síncrono

```
entity ff is
  port (clk : in std_logic;
        rst : in std_logic;
        d   : in std_logic;
        q   : out std_logic);
end ff;

architecture rtl of ff is
begin
  p_ff : process(clk)
  begin
    if rising_edge(clk) then
      if rst = '1' then
        q <= '0';
      else
        q <= d;
      end if;
    end if;
  end process p_ff;
end rtl;
```

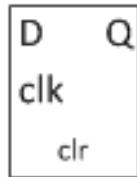


Registros

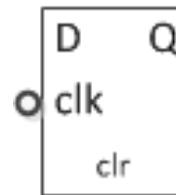
■ Tipos:

– Según temporización

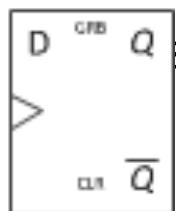
- Disparado por nivel



alto

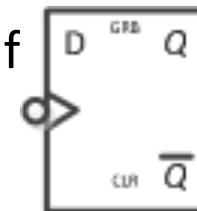


bajo



disparado por f

subida



bajada

■ Según funcionalidad

– PIPO: Carga paralelo/Salida paralela

– SIPO: Carga serie/Salida paralela

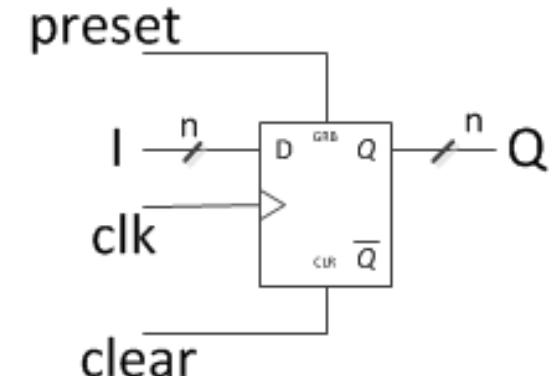
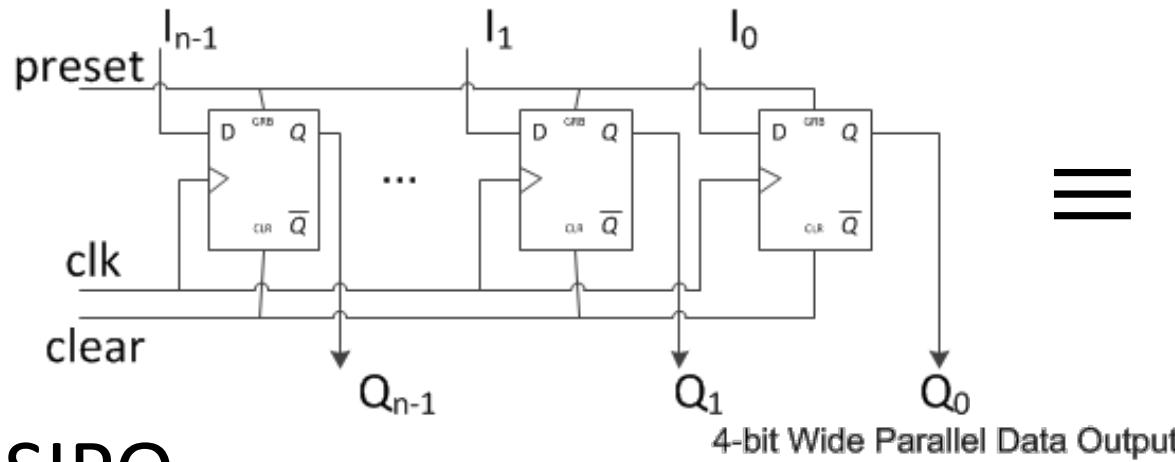
– PISO: Carga paralelo/Salida serie

– SISO: Carga serie/Salida serie

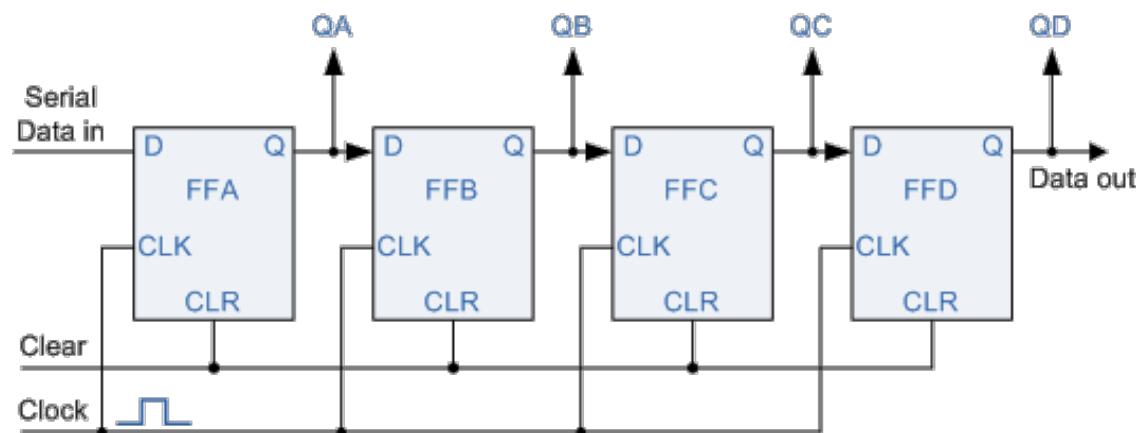


Registros. Tipos

- PIPO de n-bits



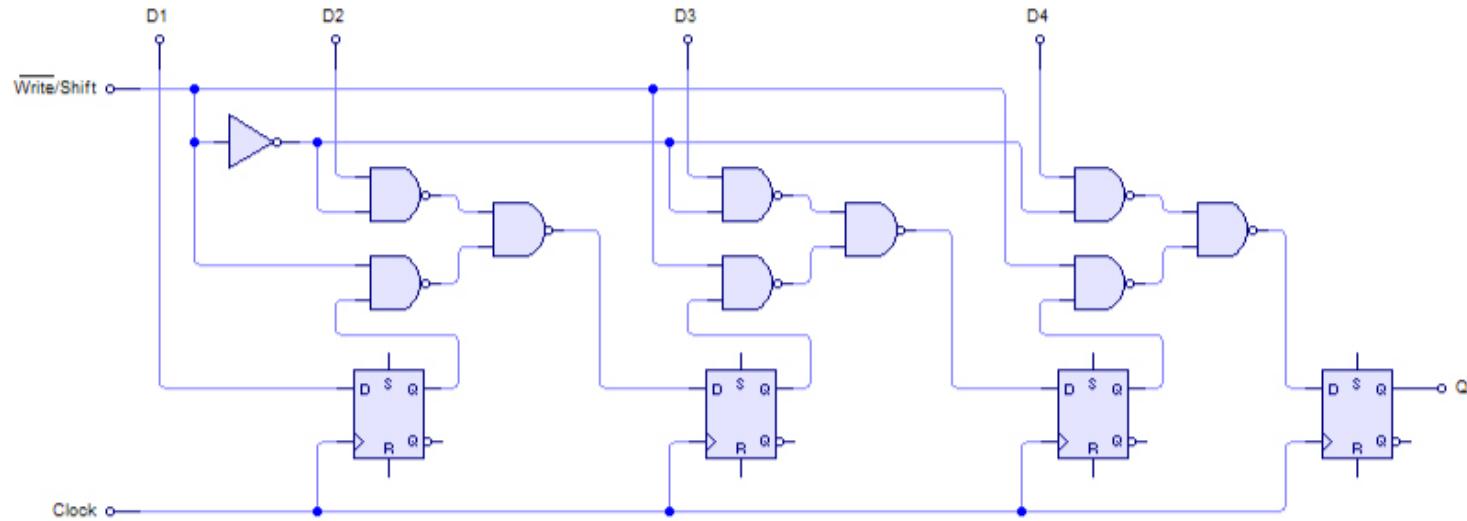
- SIPO



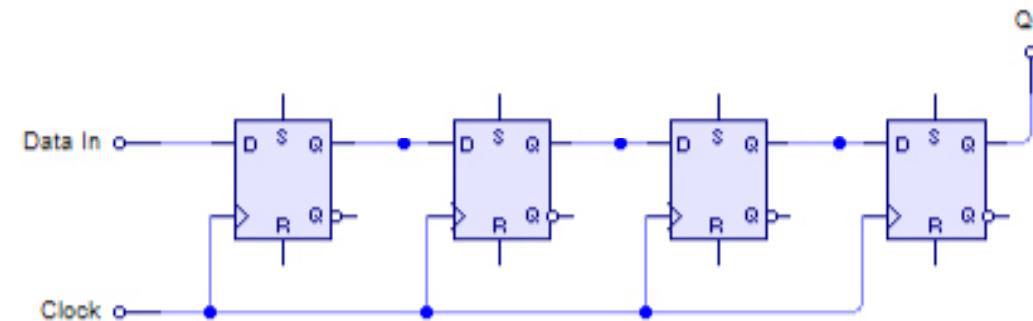


Registros. Tipos

■ PISO



■ SISO





Registros. VHDL

■ PIPO reset asíncrono

```
entity reg_pipo is
  port (clk : in std_logic;
        rst : in std_logic;
        d   : in std_logic_vector(7 downto 0);
        q   : out std_logic_vector(7 downto 0));
end reg_pipo;

architecture rtl of reg_pipo is
begin
  p_ff : process(clk, rst)
  begin
    if rst = '1' then
      q <= (others => '0');
    elsif rising_edge(clk) then
      q <= d;
    end if;
  end process p_ff;
end rtl;
```

■ PISO reset asíncrono

```
entity reg_piso is
  port (clk : in std_logic;
        rst : in std_logic;
        sh  : in std_logic;
        d   : in std_logic_vector(7 downto 0);
        q   : out std_logic);
end reg_piso;

architecture rtl of reg_piso is
  signal q_int : std_logic_vector(7 downto 0);
begin
  p_ff : process(clk, rst)
  begin
    if rst = '1' then
      q_int <= (others => '0');
    elsif rising_edge(clk) then
      if sh = '1' then
        q_int <= '0' & q_int(7 downto 1);
      else
        q_int <= d;
      end if;
    end if;
  end process p_ff;
  q <= q_int(0);
end rtl;
```



Registros. VHDL

■ SIPO reset asíncrono

```
entity reg_sipo is
  port (clk : in std_logic;
        rst : in std_logic;
        sh : in std_logic;
        d : in std_logic;
        q : out std_logic_vector(7 downto 0));
end reg_sipo;

architecture rtl of reg_sipo is
  signal q_int : std_logic_vector(7 downto 0);
begin
  p_ff : process(clk, rst)
  begin
    if rst = '1' then
      q_int <= (others => '0');
    elsif rising_edge(clk) then
      q_int <= d & q_int(7 downto 1);
    end if;
  end process p_ff;
  q <= q_int;
end rtl;
```

■ SISO reset asíncrono

```
entity reg_siso is
  port (clk : in std_logic;
        rst : in std_logic;
        sh : in std_logic;
        d : in std_logic;
        q : out std_logic);
end reg_siso;

architecture rtl of reg_siso is
  signal q_int : std_logic_vector(7 downto 0);
begin
  p_ff : process(clk, rst)
  begin
    if rst = '1' then
      q_int <= (others => '0');
    elsif rising_edge(clk) then
      q_int <= d & q_int(7 downto 1);
    end if;
  end process p_ff;
  q <= q_int(0);
end rtl;
```



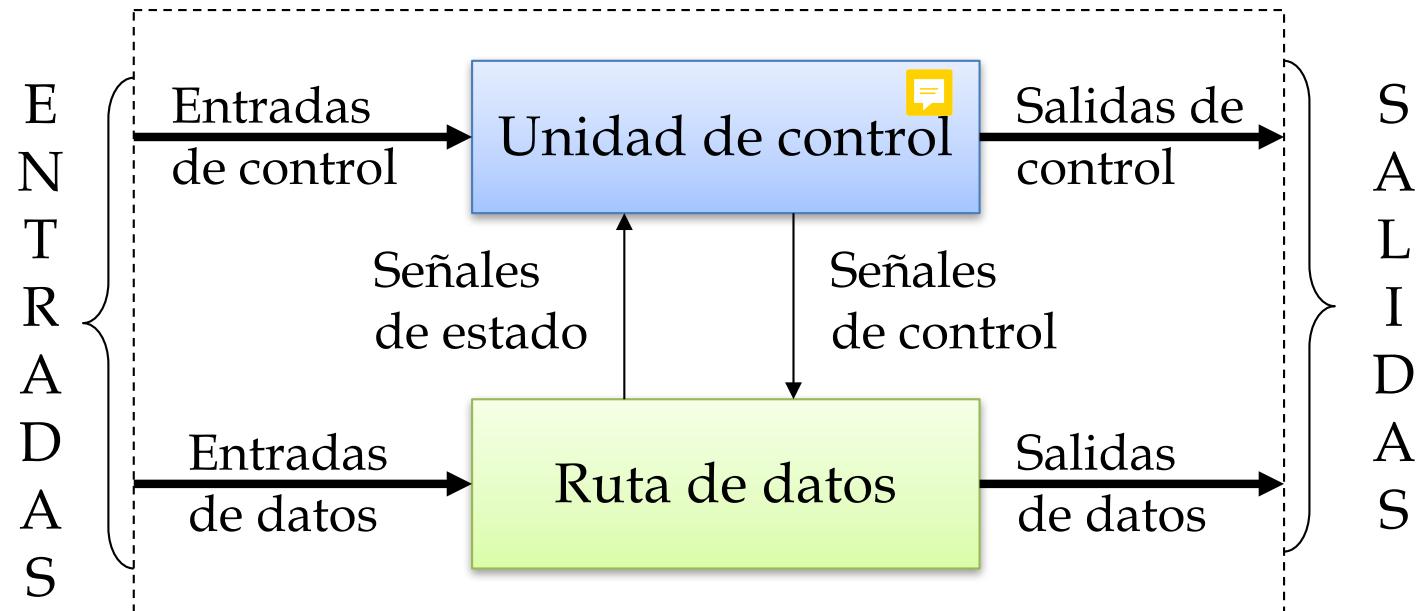
¿Qué es el diseño algorítmico?

- Modo de especificación e implementación de sistemas digitales, que permite sistematizar y automatizar en gran medida su construcción.
- Parte siempre de una especificación en la que el comportamiento del sistema se describe en forma de un algoritmo:
 - Cómo calcular la salida en función de la entrada.
- Implementación:
 - Unidad de Control y Ruta de Datos.



¿Qué son los sistemas algorítmicos?

- Sistemas secuenciales síncronos.
- Comportamiento definido IMPLÍCITAMENTE
 - No se especifica el valor de Z sino el modo de calcularlo: el algoritmo.
- Modelo:





Proceso de diseño: resumen

1. Estudio de la especificación:
 - Esquema de pasos secuenciales a seguir (algoritmo)
 - Posible HW a utilizar (módulos específicos complejos)
2. Creación de un diagrama ASM: partiendo de las conclusiones del apartado 1, crear un diagrama que cumpla las especificaciones.
3. Diseño de la unidad de control:
 - Codificar cada uno de los estados del ASM
 - Codificar el cambio de estado
 - Mediante señales internas de control
 - Mediante señales externas de control
 - Codificar las señales de control hacia la ruta de datos: cada estado tendrá asociado unos valores de TODAS las señales que controlan los módulos complejos (por ejemplo: señal load de los registros).
4. Diseño de la ruta de datos:
 - Interconectar las señales de datos externas e internas a los módulos.
 - Interconectar las señales de control (obtenidas en el apartado 3) a los módulos de la ruta de datos.



Paso 1. Estudio de la especificación

- ¿Entiendo completamente el enunciado del problema
- Esquema de pasos secuenciales a seguir (algoritmo)
 - Siempre hay varias opciones
 - ¿Cumplen las especificaciones?
 - ¿Pueden simplificarse?
- Hardware a utilizar (módulos específicos complejos)
 - ¿Qué módulos específicos complejos son necesarios?
 - ¿Existen restricciones en el número o tipo de módulos?
 - ¿Usar este hardware obliga a replantear el punto anterior?



Paso 2. Diagrama ASM

- Creación de un diagrama ASM: partiendo de las conclusiones del apartado anterior:
 - Número de estados
 - Necesito más o menos estados
 - Cuando se hacen efectivas las señales control
 - Cuándo se carga un dato en un registro
 - Cuándo deja de contar un contador
 - Cuándo es correcta una comparación
 - ...
 - Estoy realizando un diseño Moore, Mealy o una mezcla

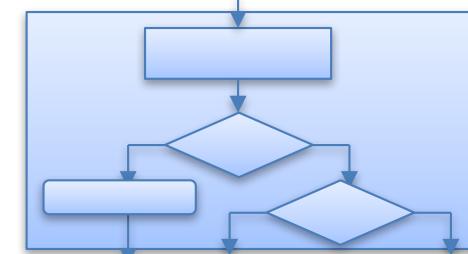
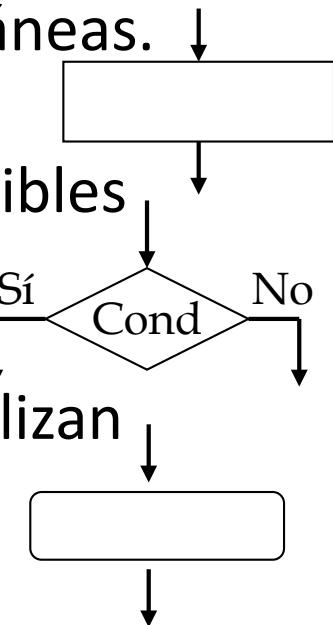


Paso 2. Diagrama ASM

- Forma gráfica de representar el algoritmo.

- Elementos:

- Caja de estado: asignaciones y operaciones simultáneas.
- Caja de decisión: bifurcación condicional con 2 posibles salidas.
- Caja de salida condicional: asignaciones que se realizan cuando se cumple una condición (Mealy).
- Bloque ASM: una caja de estado con red de cajas de decisión y de salida condicional.



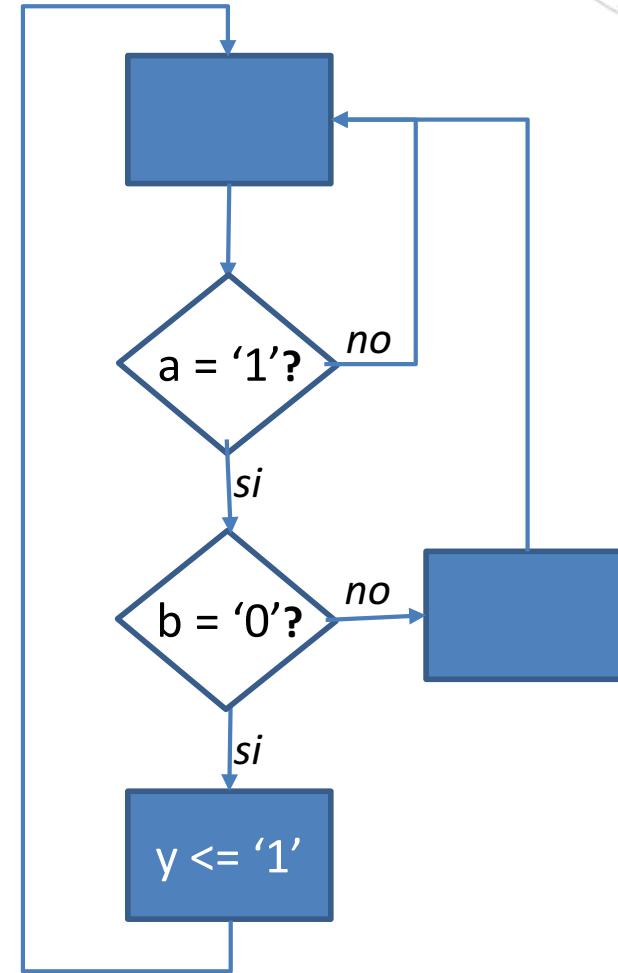
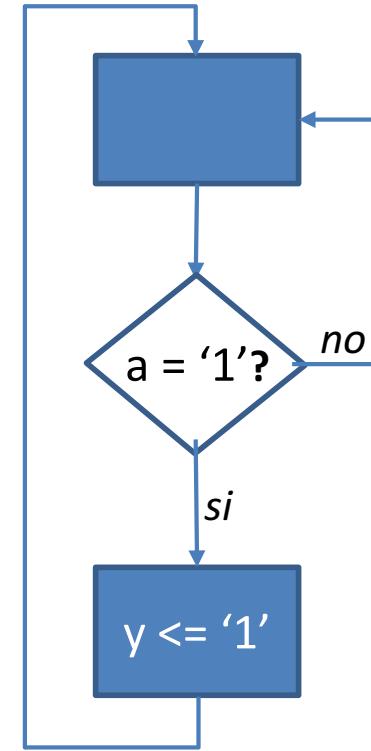
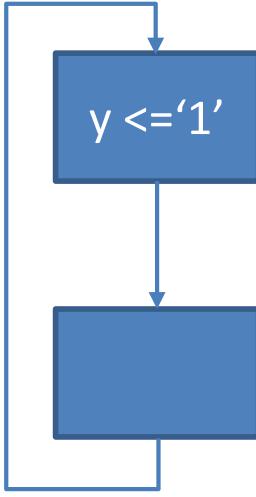


Paso 2. Diagrama ASM. Reglas

1. Cada caja de estado puede tener varias entradas pero únicamente una salida
2. Caja de decisión sólo se puede llegar hasta ella desde la caja de estado contenida en su mismo bloque ASM
3. Todas las operaciones de un bloque ASM (cajas de estado y cajas de decisión) son concurrentes
4. Un bloque ASM sólo ~~puede~~ contiene una caja de estado y cualquier número de cajas de decisión (0, 1, 2, ...)
5. El primer elemento de un bloque ASM es siempre una caja de estado. Un bloque ASM no puede contener solo cajas de decisión.
6. Siempre se entra en un bloque ASM por su caja de estado
7. Todo bloque ASM debe ser etiquetado



Paso 2. Diagrama ASM. Ejemplos





Paso 2. Diagrama ASM. Ejemplos

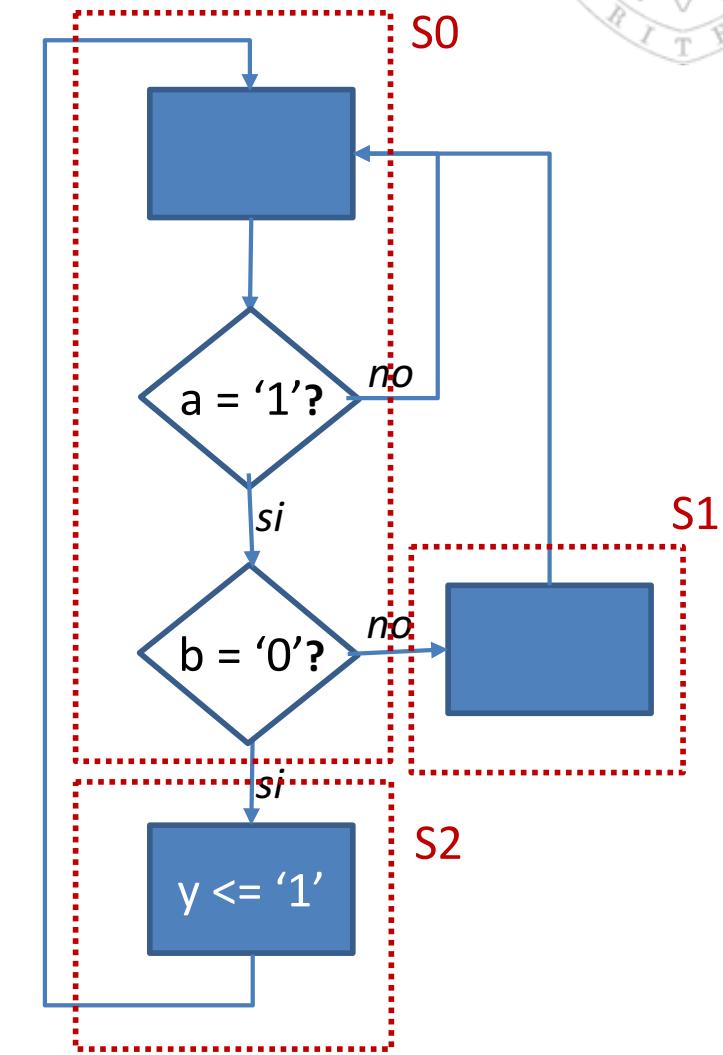
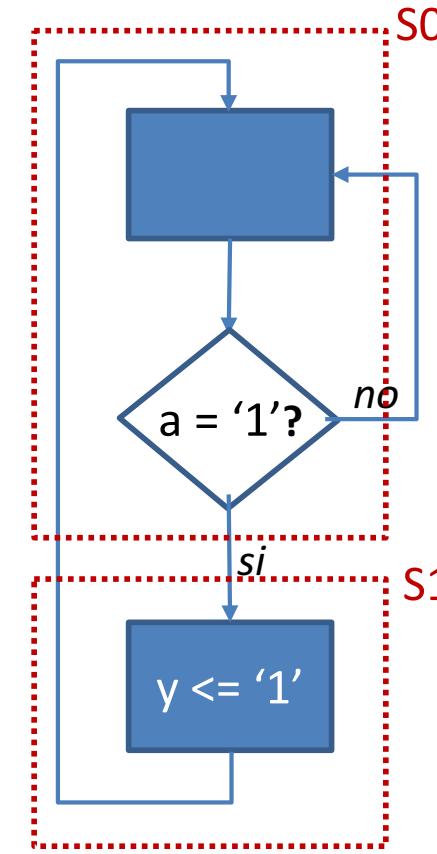
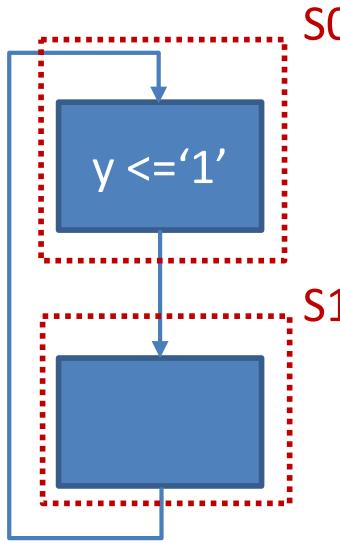




Diagrama ASM. Ejemplos de errores

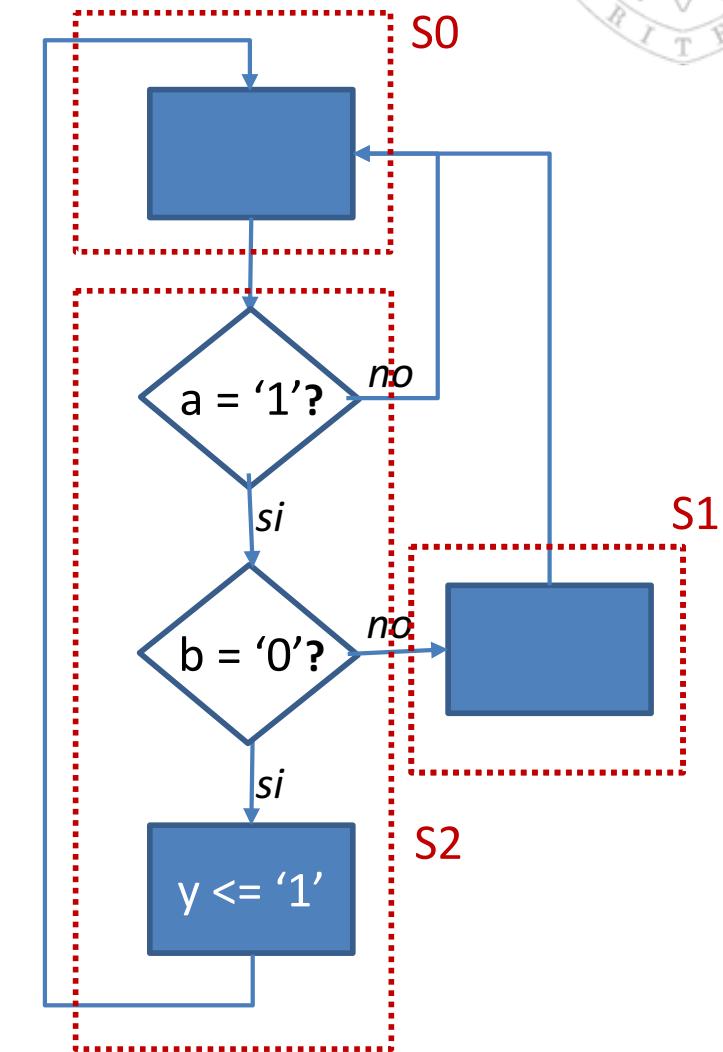
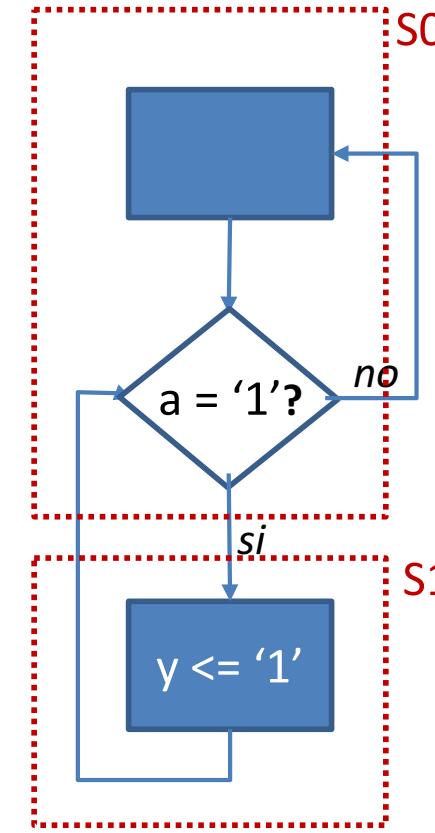
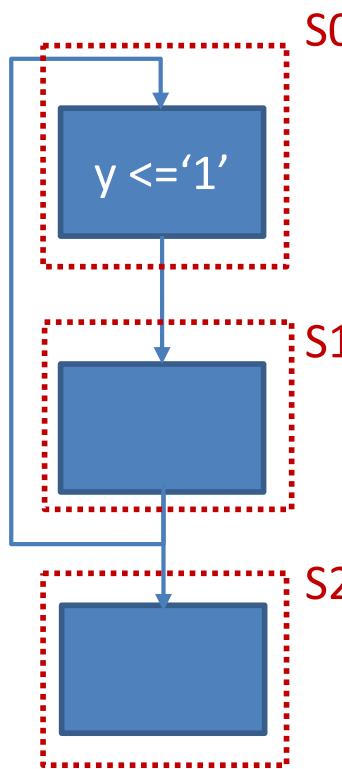




Diagrama ASM → Camino de datos

- Deducir los módulos hardware a partir de las operaciones descritas en el ASM
- Interconectar las señales de datos externas e internas a los módulos
 - Cuando un mismo módulo recibe por el mismo puerto entradas desde varios módulos → multiplexor
- Deducir las señales de control de los módulos hardware
- Interconectar las señales de control de los módulos y las señales de estado con la unidad de control
 - Señales de estado: salidas de módulos del camino de datos necesarias para que la UC tome decisiones



Diagrama ASM → Unidad control

1. La UC se implementa como una FSM
2. Cada bloque ASM se convierte en un estado de la FSM con el mismo nombre
3. Las transiciones entre bloques ASM se convierten en transiciones entre los estados equivalentes de la FSM
4. Las salidas de la UC son las señales de control y las salidas externas de control
5. Las entradas de la UC son las señales de estado y las entradas externas de control



Diagrama ASM → UC. Ejemplos

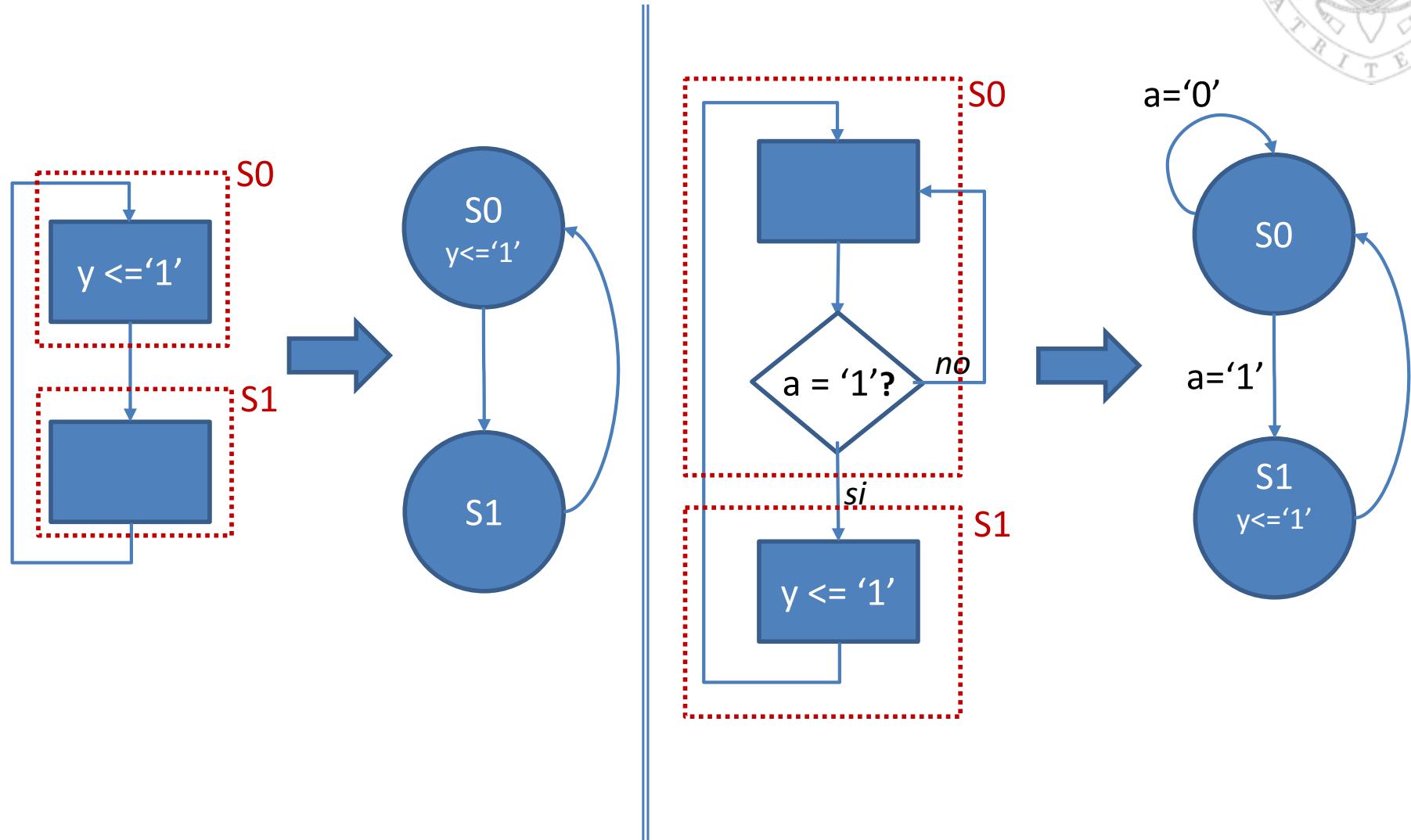
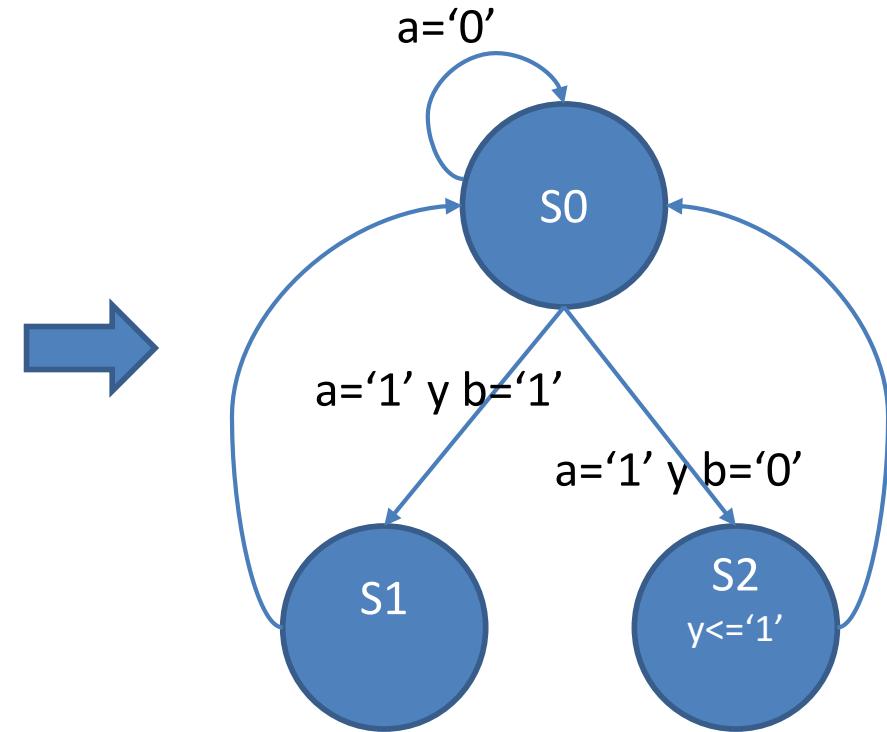
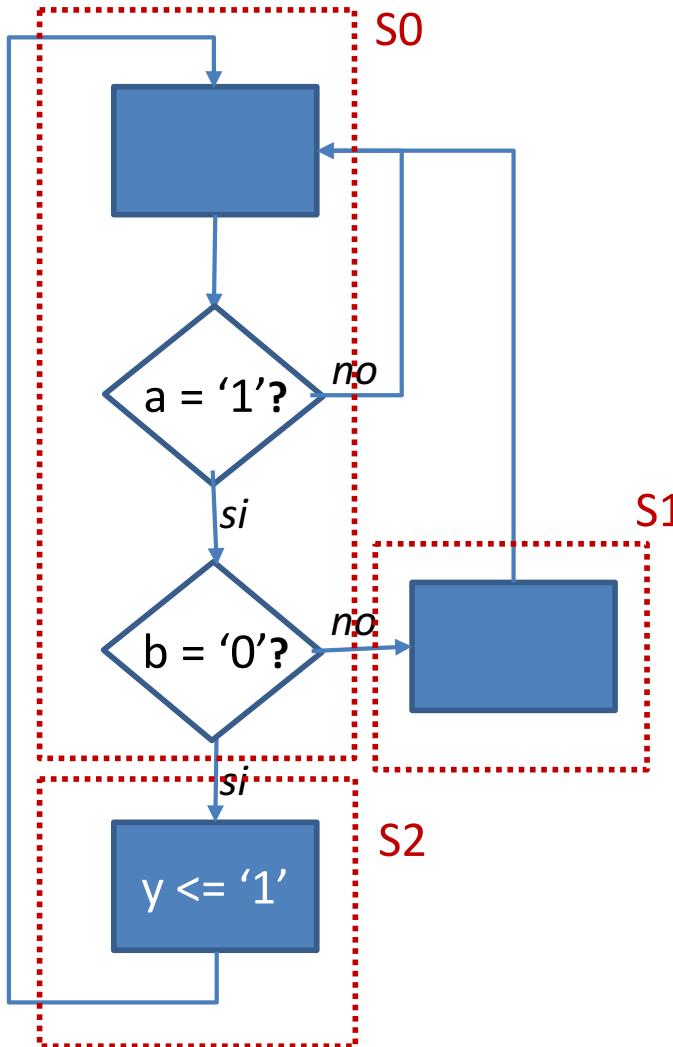




Diagrama ASM → UC. Ejemplos





Implementación unidad de control

- Registro de estado y red combinacional
 - Implementación canónica (FC-1ercuatrimestre)
- Contador + red combinacional
 - Si $S(t+1) = S(t)+1$
 - El contador actúa como registro de estado
- Registro de desplazamiento + red combinacional
 - Estado siguiente = estado actual $>>$ o $<<$
 - Registro desplazamiento realimentado o no realimentado actúa como registro de estado



Ejemplo diseño I

- Diseñar un sistema capaz de reconocer una clave
- El sistema tendrá dos modos de funcionamiento:
 - Cambiar clave: En este modo se puede introducir la nueva clave (4 bits) en un ciclo de reloj.
 - Introducir clave: en este modo se introduce la clave, un bit por cada ciclo y se compara con la clave guardada:
 - En caso de acierto se pone la señal acierto a 1 y se vuelve al estado inicial (decidir si se introduce o se cambia la clave).
 - En caso de fallo se pone la señal acierto a 0 y el sistema volvería a pedir de nuevo que se introduzca la clave (bit a bit).

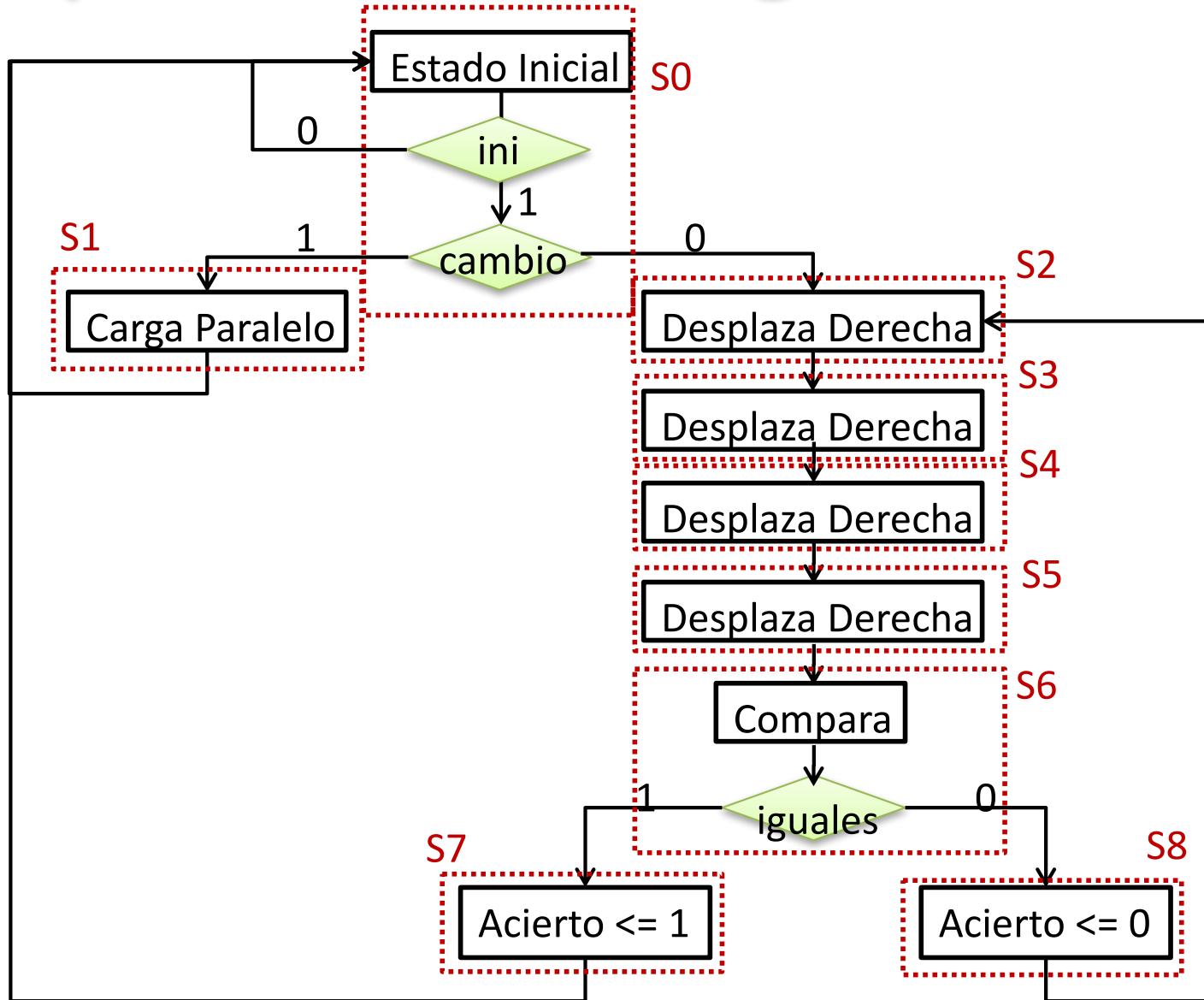


Ejemplo diseño I

- Estudio de la especificación:
 - Esquema de pasos secuenciales a seguir (algoritmo)
 - Estado inicial
 - Si cambio es 1
 - Carga paralelo de la clave
 - Vuelta al estado inicial
 - Si cambio es 0
 - Realizar 4 desplazamientos a la derecha (4 ciclos)
 - Si la clave es correcta: acierto = 1 y volver al punto 1.
 - Si la clave es incorrecta: acierto = 0 y volver al punto 3.

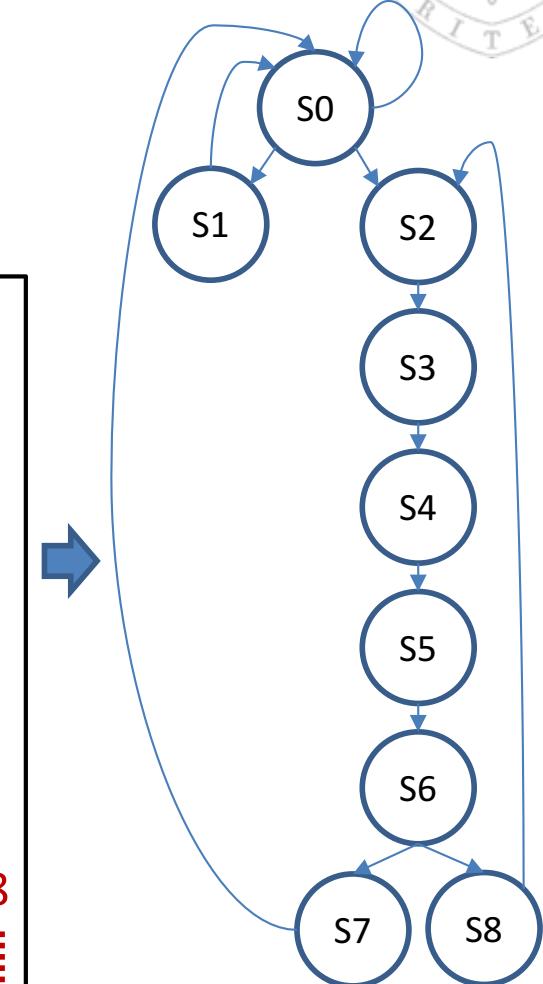
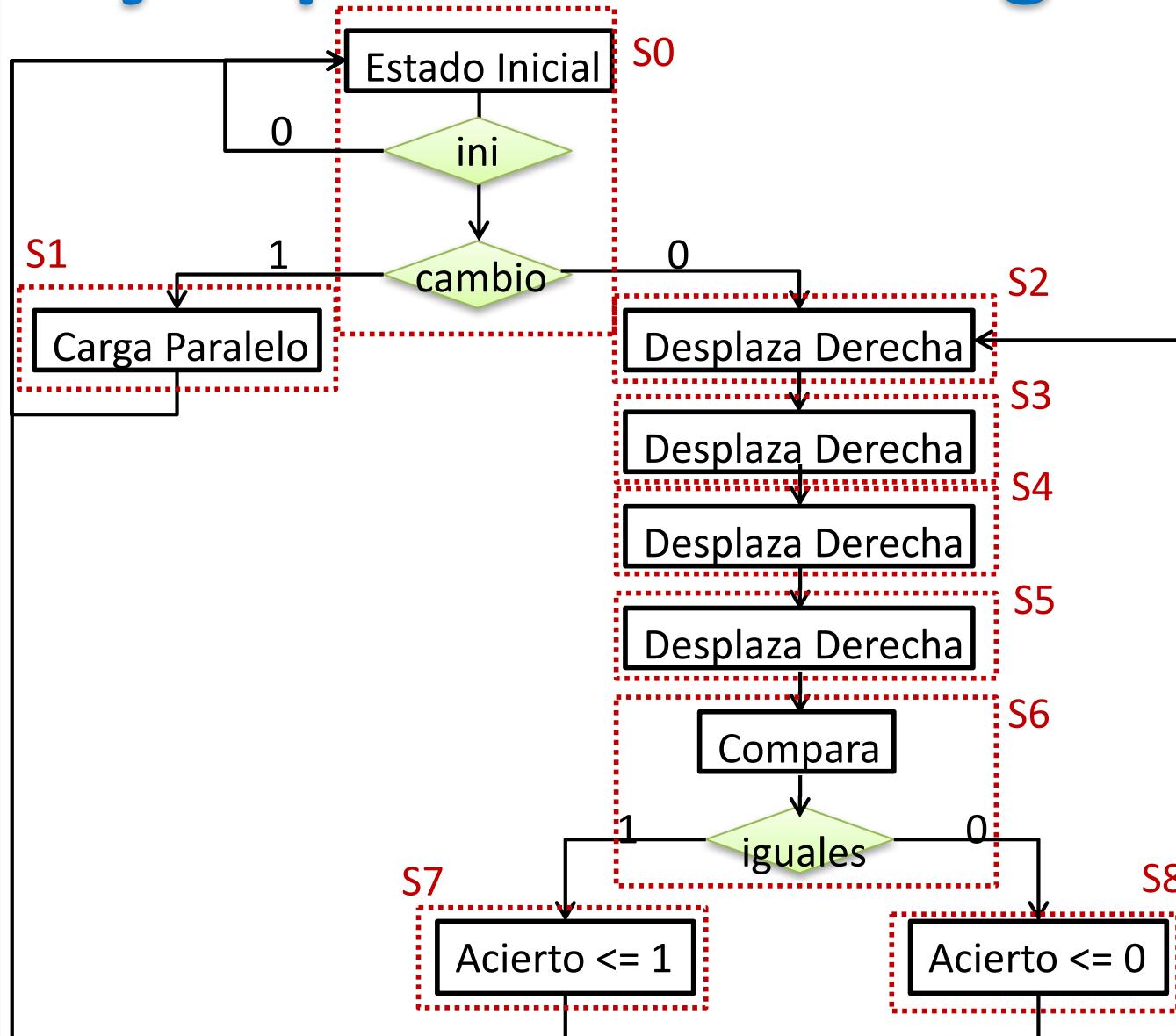


Ejemplo diseño I. Diagrama ASM





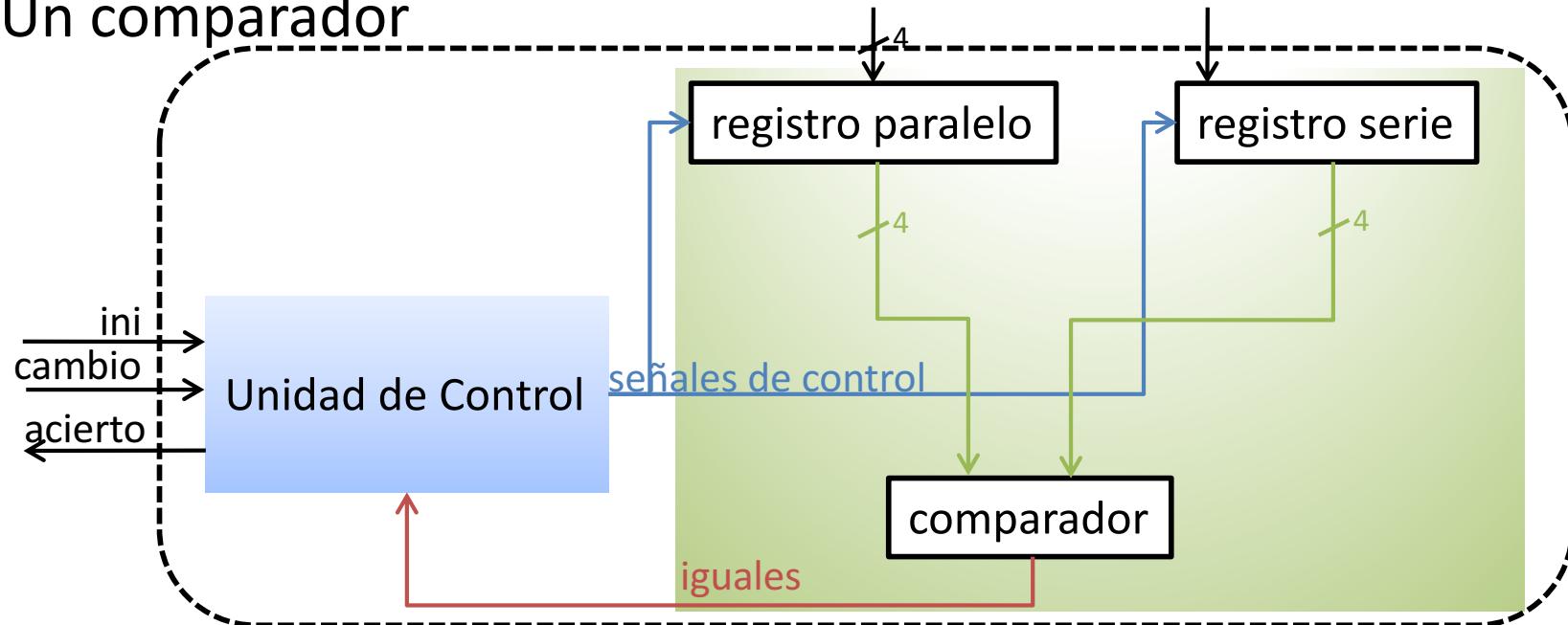
Ejemplo diseño I. Diagrama ASM





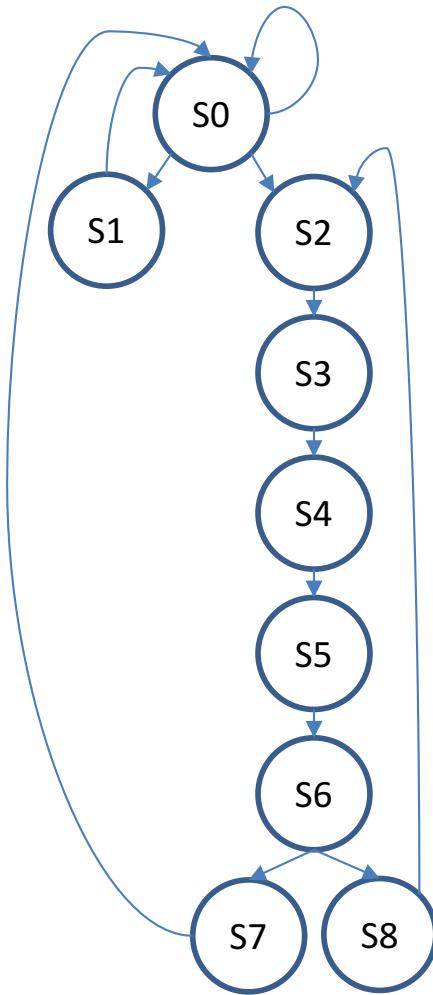
Ejemplo diseño I. Estructura

- Possible HW a utilizar (módulos específicos complejos)
 - 2 registros, uno en modo carga paralelo y otro en modo carga serie.
 - Un comparador





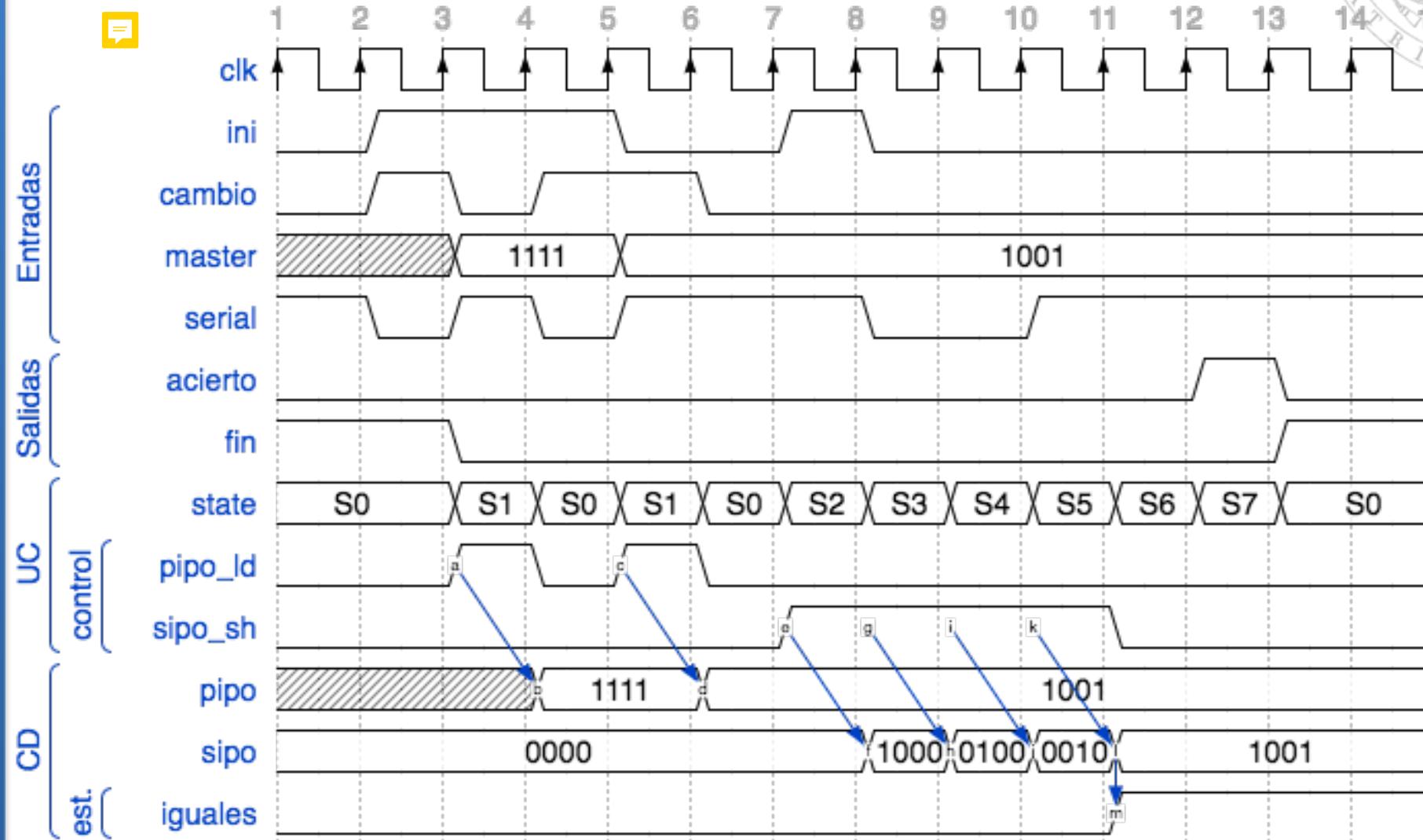
Ejemplo diseño I. Diagrama ASM



	piro_id	piso_sh	fin	acuerdo
S0	0	0	1	0
S1	1	0	0	0
S2	0	1	0	0
S3	0	1	0	0
S4	0	1	0	0
S5	0	1	0	0
S6	0	1	0	0
S7	0	0	0	1
S8	0	0	0	0



Ejemplo diseño I. Diagrama ASM

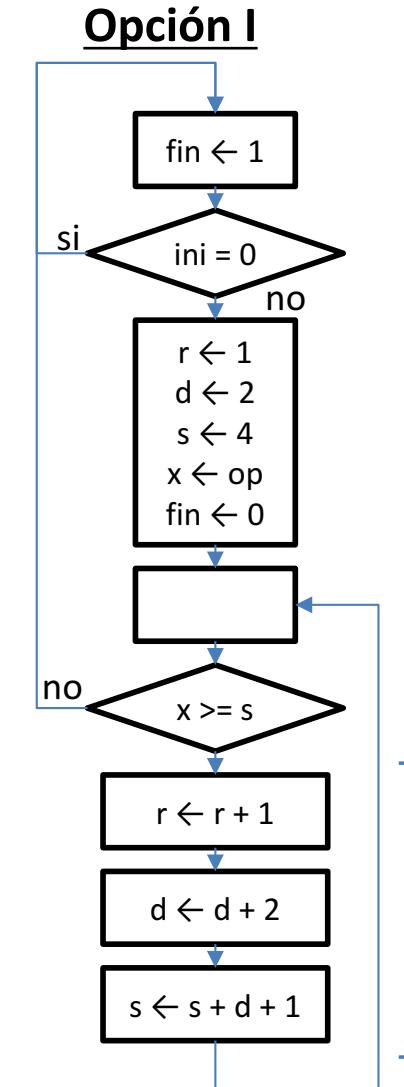
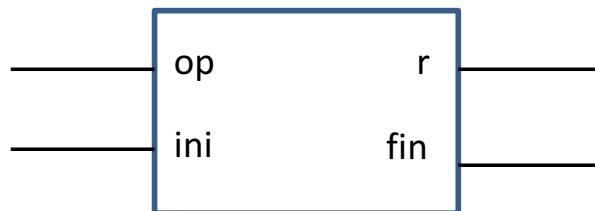




Ejemplo II. Raíz cuadrada entera

```
# In:  op
# Out: r

r := 1
d := 2
s := 4
while op >= s
    r := r + 1
    d := d + 2
    s := s + d + 1
end
```



Recursos Hw:

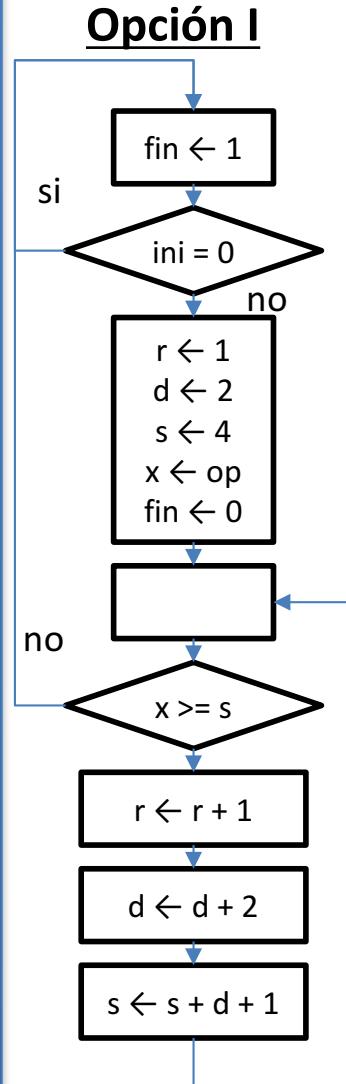
- 1 sumador
- 2 muxes

3 ciclos/iteración

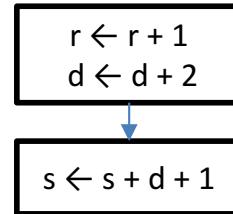
¿Puede hacerse de otra forma ?



Ejemplo II. Raíz cuadrada entera

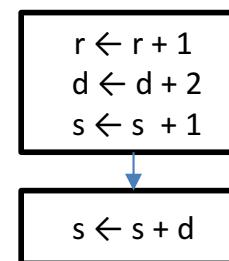


Opción II



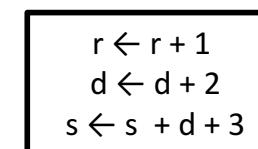
2 sumadores
2 ciclos/iteración

Opción III



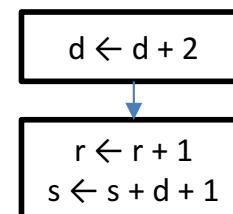
3 sumadores
2 ciclos/iteración

Opción IV



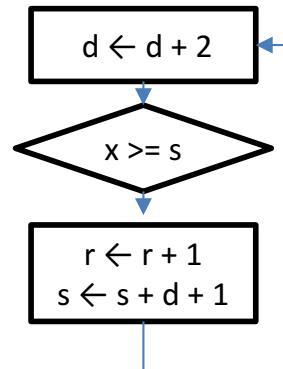
4 sumadores
1 ciclo/iteración

Opción V



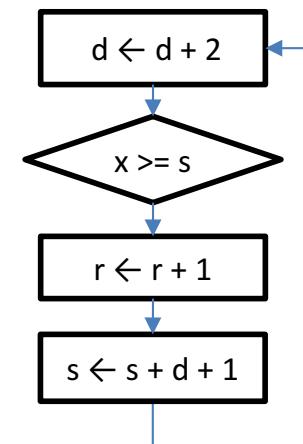
2 sumadores
2 ciclos/iteración

Opción VI



2 sumadores
1 ciclo/iteración

Opción VII

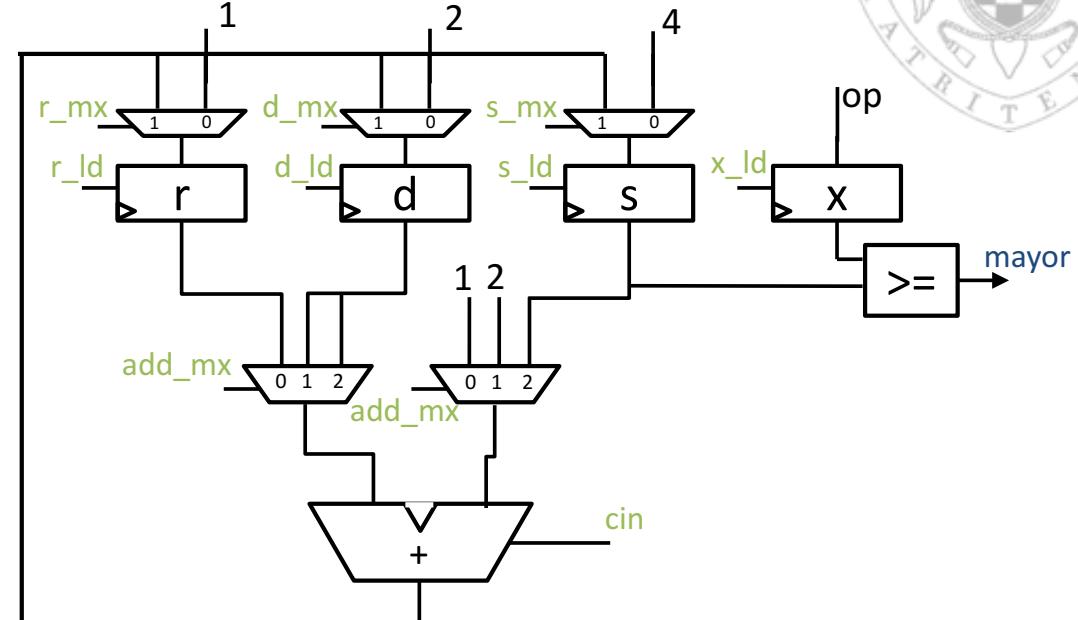
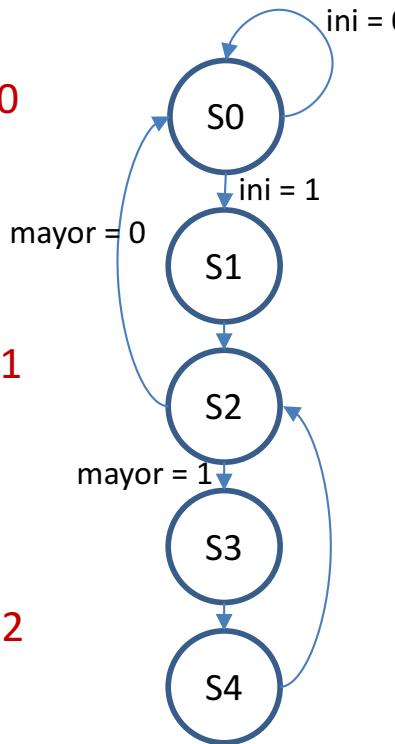
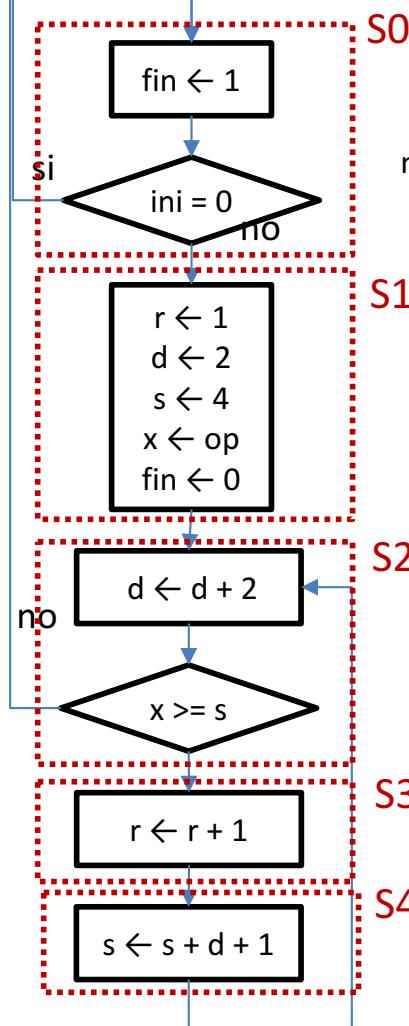


1 sumadores
2 ciclo/iteración



Ejemplo II. Raíz cuadrada entera

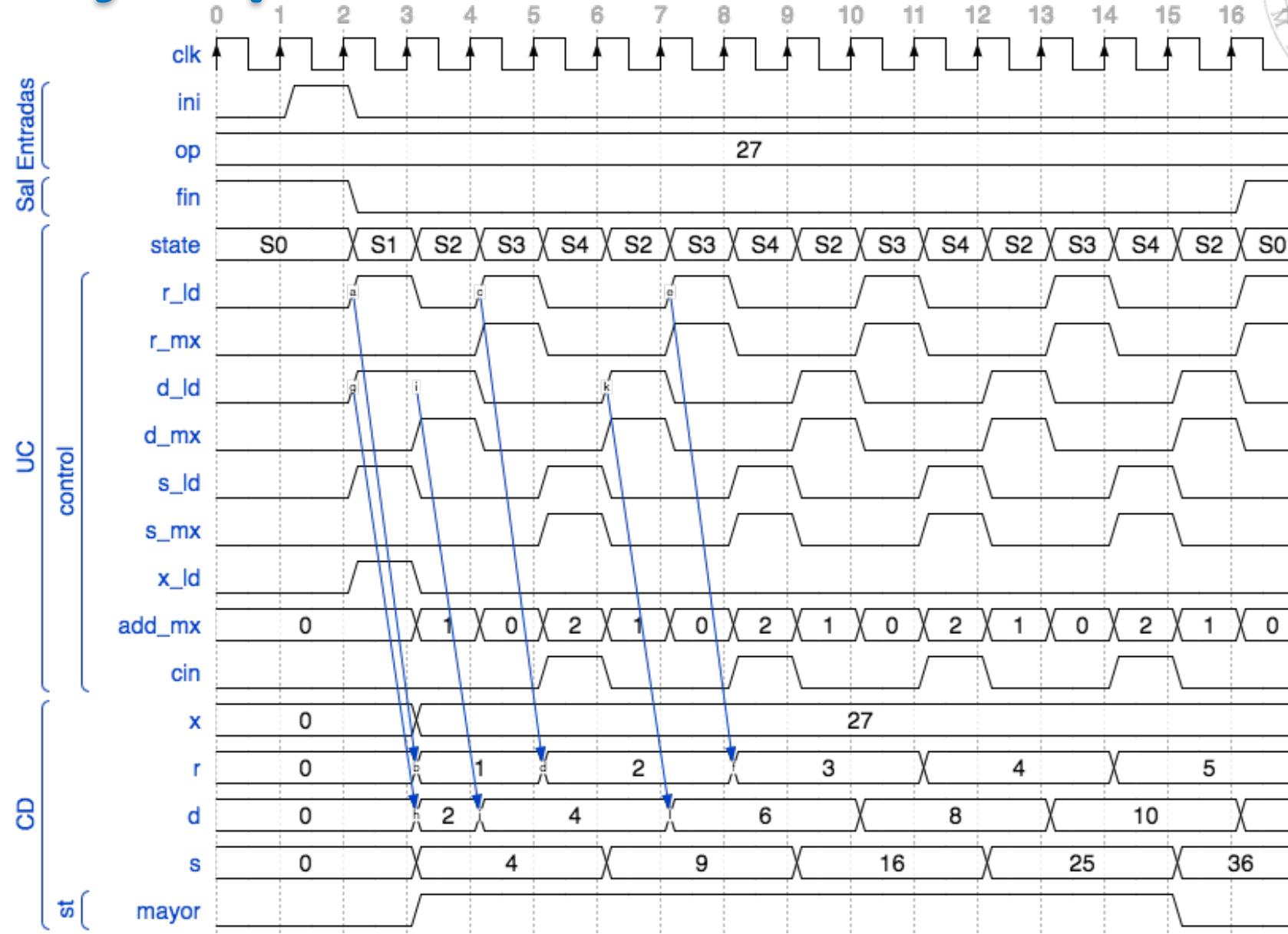
Opción VII



	r_Id	r_Mx	d_Id	d_Mx	s_Id	s_Mx	x_Id	add_Mx	cin	fin
S0	0	-(0)	0	-(0)	0	-(0)	0	-(0)	0	1
S1	1	0	1	0	1	0	1	-(0)	0	0
S2	0	-(0)	1	1	0	-(0)	0	1	0	0
S3	1	1	0	-(0)	0	-(0)	0	0	0	0
S4	0	-(0)	0	0	1	1	0	2	1	0



Ejemplo II. Raíz cuadrada entera





Optimización

- Compartición de unidades funcionales
 - Minimizar número de U.F. en la ruta de datos.
- Compartición de registros
 - Minimizar número de registros en la ruta de datos.
- Compartición de conexiones
 - Minimizar el número de conexiones en la ruta de datos.



Ejemplo I

- Vamos a presentar las técnicas usando como ejemplo el ASM de la aproximación de la raíz cuadrada de dos enteros con signo:

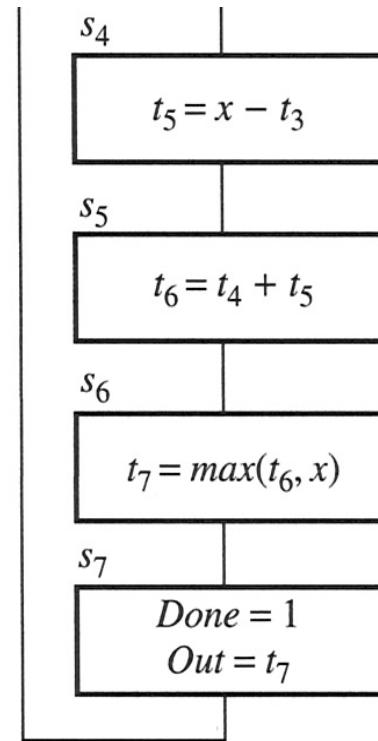
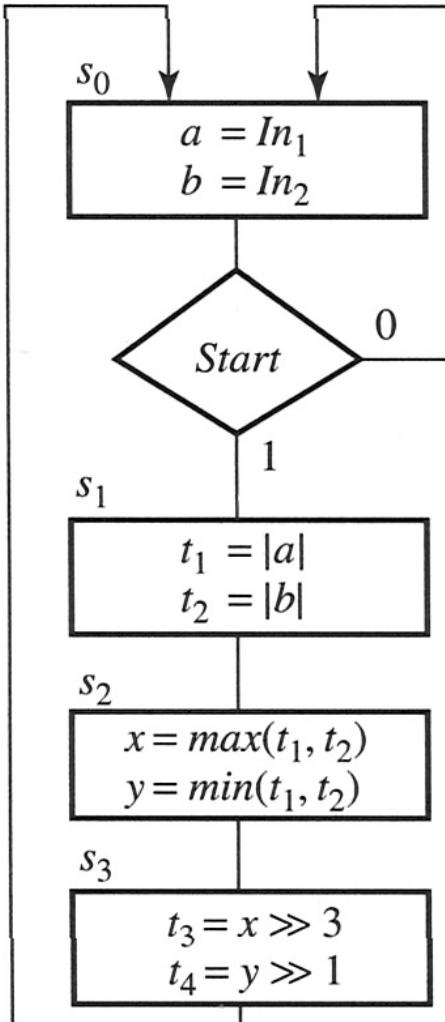
$$\sqrt{a^2 + b^2} \simeq \max((0.875 \cdot x + 0.5 \cdot y), x)$$

$$x = \max(|a|, |b|)$$

$$y = \min(|a|, |b|)$$



Ejemplo I



Recursos

11 registros
 2 abs
 1 min
 1 max
 2 desplazadores
 1 sumador
 1 restador



Compartir Registro

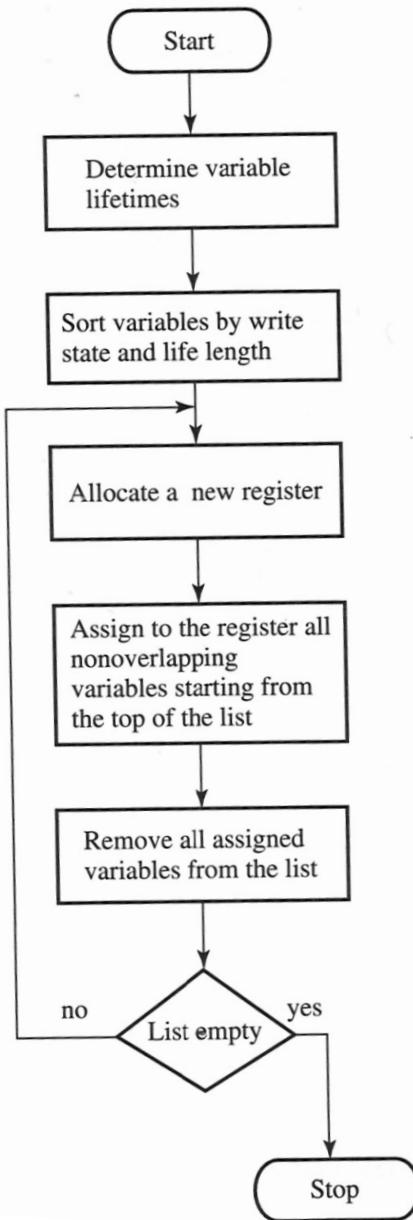
- Algoritmo
 - Determinar tiempo de vida de la variable
 - Conjunto de estados en los que la variable está viva.
 - Agrupar variables con tiempos de vida disjuntos y asignarles un mismo registros
 - Agrupación con distintos criterios: p.e. minimizar el número de registros, minimizar número de registros y mux, ...



Algoritmo del vértice izquierdo (left-edge algorithm)



Compartir Registro



	s_1	s_2	s_3	s_4	s_5	s_6	s_7
a	x						
b	x						
t_1		x					
t_2		x					
x			x	x	x	x	
y			x				
t_3				x			
t_4				x	x		
t_5					x		
t_6						x	
t_7							x
Number of live variables							
	2	2	2	3	3	2	1

$$R_1 = [a, t_1, x, t_7]$$

$$R_2 = [b, t_2, y, t_4, t_6] \rightarrow 3 \text{ registros!}$$

$$R_3 = [t_3, t_5]$$



Compartir Registro

- Dependiendo de la asignación cambia el número de MUX en la ruta de datos.
- ¿Asignación óptima?
 - Algoritmo basado en grafo de compatibilidad
- Grafo de compatibilidad
 - Vértices = variables.
 - Arista: compatibilidad entre variables
 - Aristas de incompatibilidad
 - Aristas prioritarias (opcional). Incluye etiqueta s/d

UF que usan ambos nodos

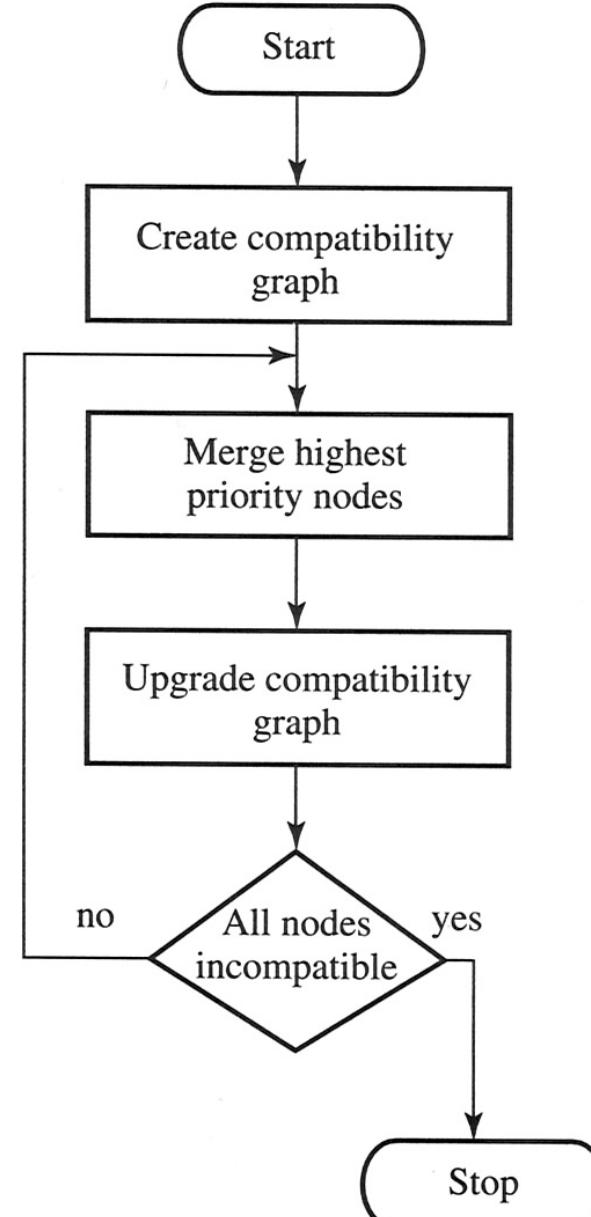
UF que generan ambos nodos





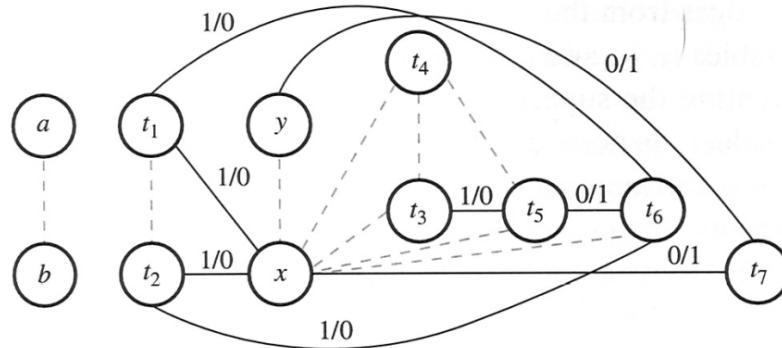
Optimización. Compartir Registro

- Partición del grafo:
agrupar vértices
conectados por aristas
prioritarias de mayor
peso y crear
supernodos.

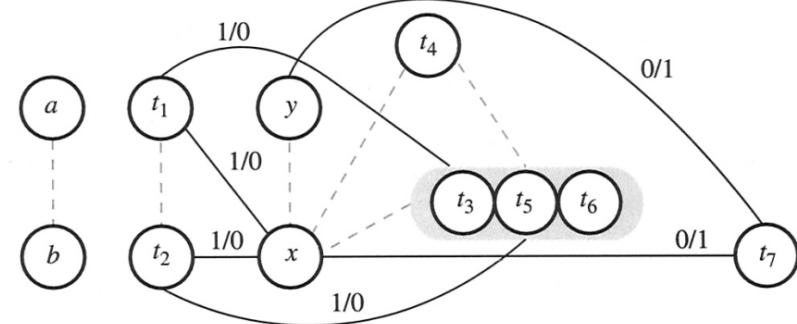
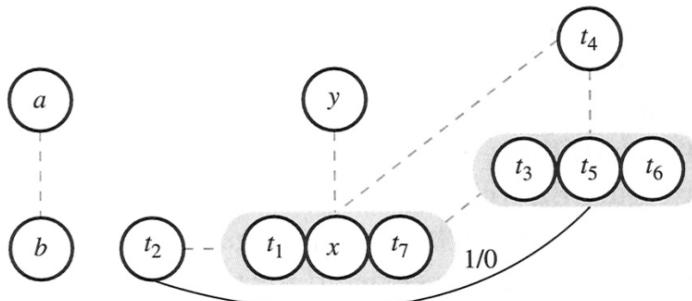
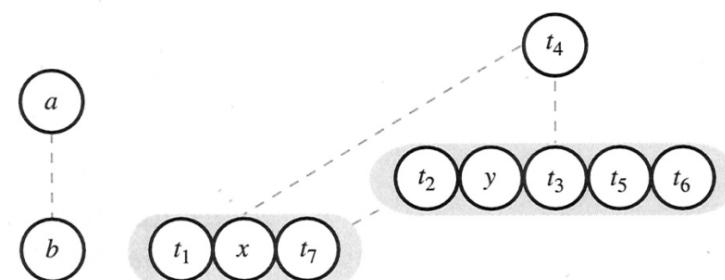




Optimización. Compartir Registro



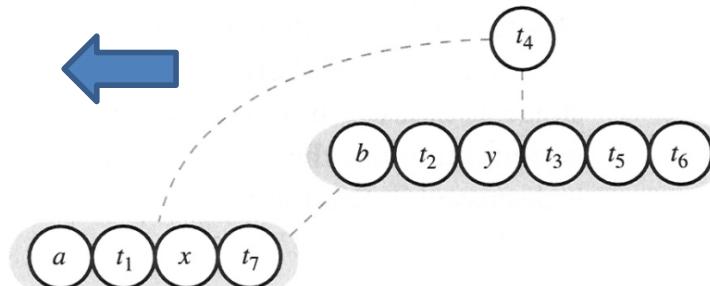
(a) Initial compatibility graph

(b) Compatibility graph after merging t_3 , t_5 , and t_6 (c) Compatibility graph after merging t_1 , x , and t_7 (d) Compatibility graph after merging t_2 and y

$$R_1 = [t_4]$$

$$R_2 = [b, t_2, y, t_3, t_5, t_6]$$

$$R_3 = [a, t_1, x, t_7]$$



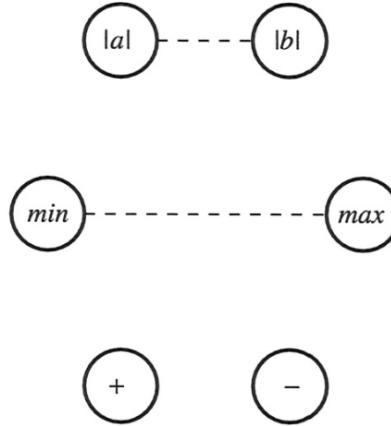


Optimización. Compartir UF

- Agrupar UF sencillas en UF más complejas: UF multifunción.
- ¿Cuándo? UF multifunción y el coste de conexión es menor que el coste de las UF sencillas.
- ¿Cómo? Algoritmo de particionamiento grafo de compatibilidad

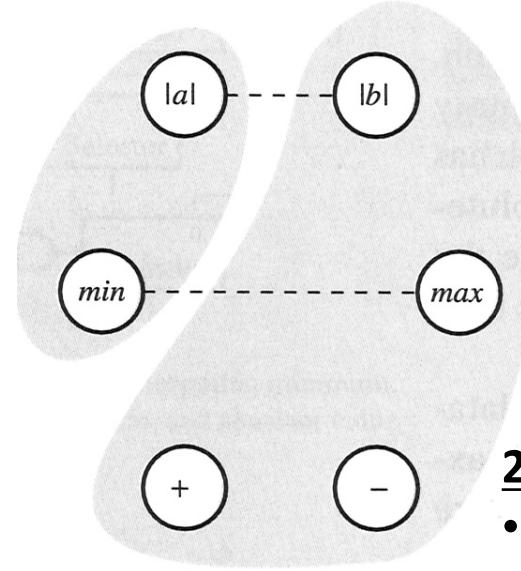


Optimización. Compartir UF



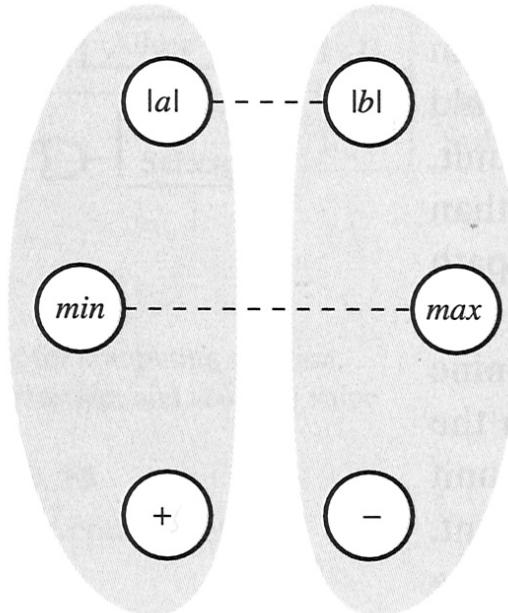
2 UF:

- 2 ABS
- min
- max
- -
- +



2 UF:

- ABS, min
- ABS, max, -, +

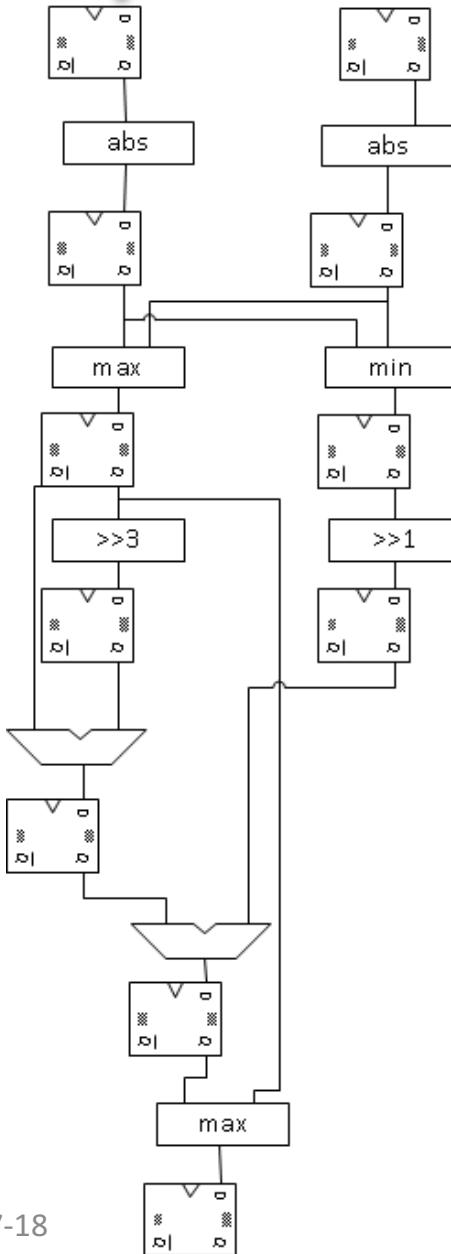


2 UF:

- ABS, min, +
- ABS, max, -



Ejemplo I. Compartir registro & UF

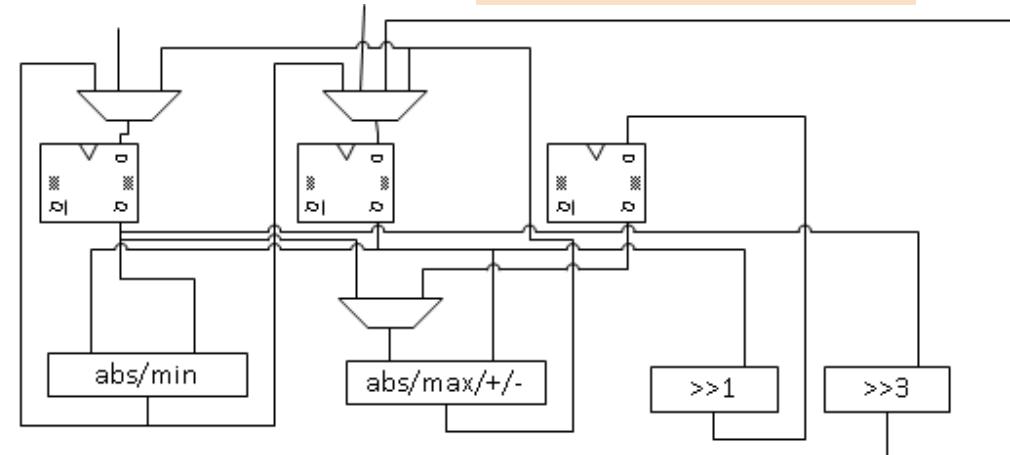


Original

- 11 registros
- 5 INV
- 6 +
- 4 MUX
- 2 desplazadores

Optimizada

- 3 registros
- 2 AND
- 1 INV
- 1 XOR
- 2 +
- 2 MUX
- 2 desplazadores



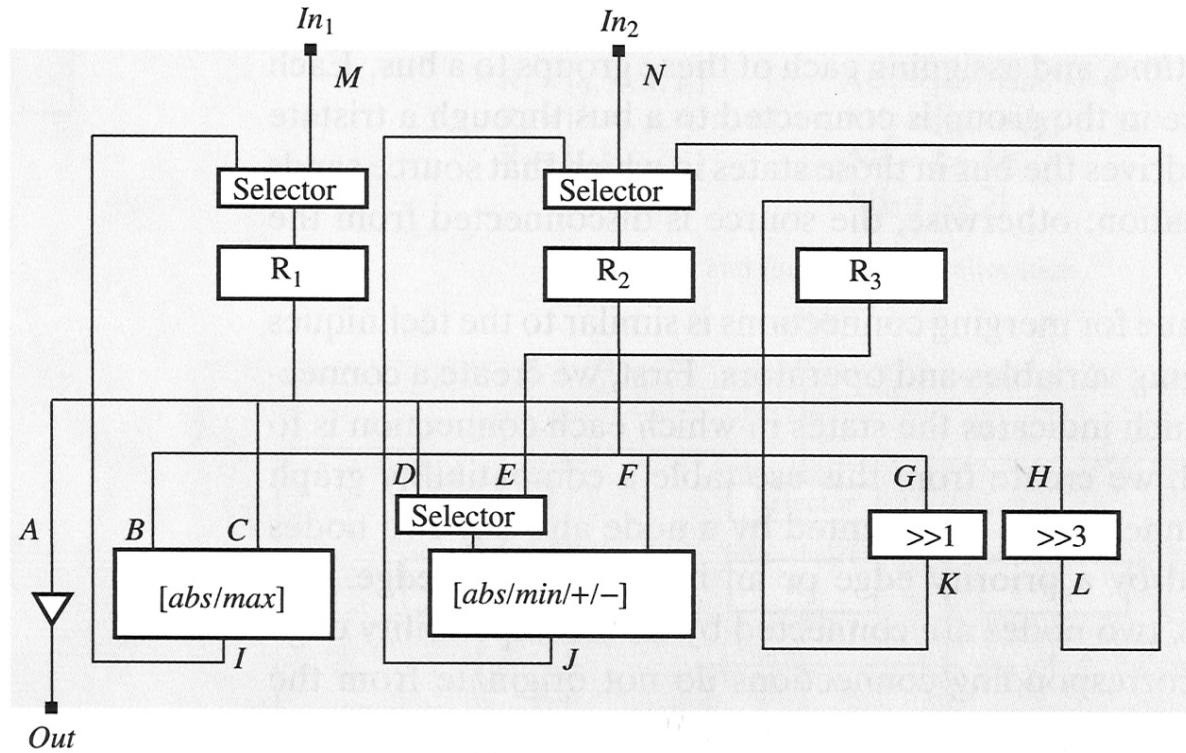


Optimización. Compartir conexión

- Basado en grafo de compatibilidad:
 - Tabla de uso de conexiones.
 - Grafo de compatibilidad:
 - Vértices son conexiones.
 - Arista:
 - Incompatibles: no tienen la misma fuente pero son usadas en el mismo ciclo.
 - Prioritarias: tienen una misma fuente o un mismo destino.



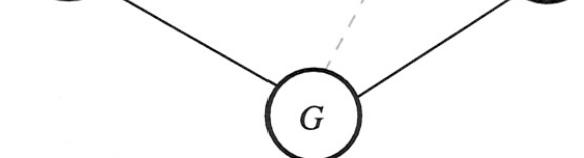
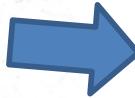
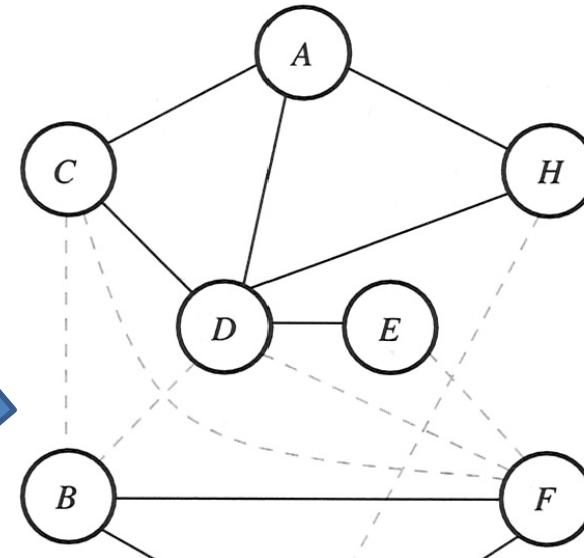
Optimización. Compartir conexión



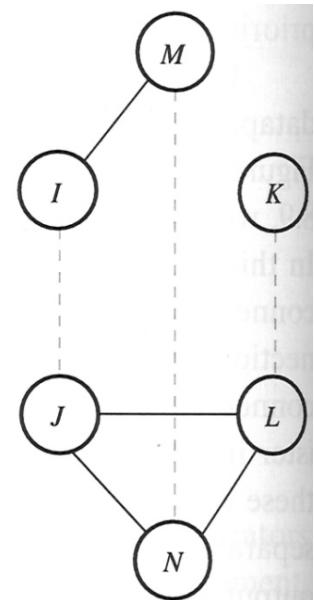


Optimización. Compartir conexión

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7
A								x
B		x				x		
C	x	x				x		
D		x		x				
E				x	x			
F	x	x		x	x			
G			x					
H			x					
I	x	x				x		
J	x	x		x	x			
K			x					
L			x					
M	x							
N	x							

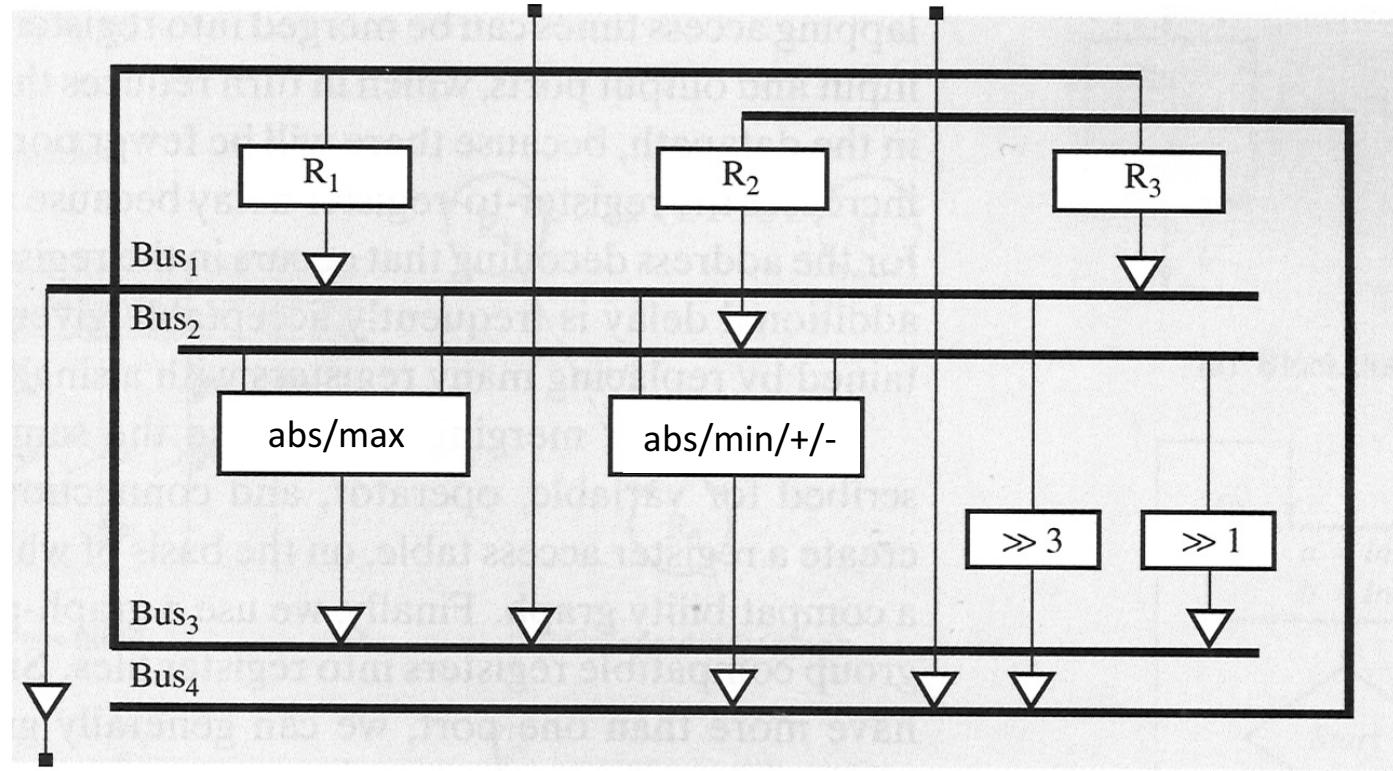


$\text{Bus1} = \{A, C, D, E, H\}$
 $\text{Bus2} = \{B, F, G\}$
 $\text{Bus3} = \{I, K, M\}$
 $\text{Bus4} = \{J, L, N\}$



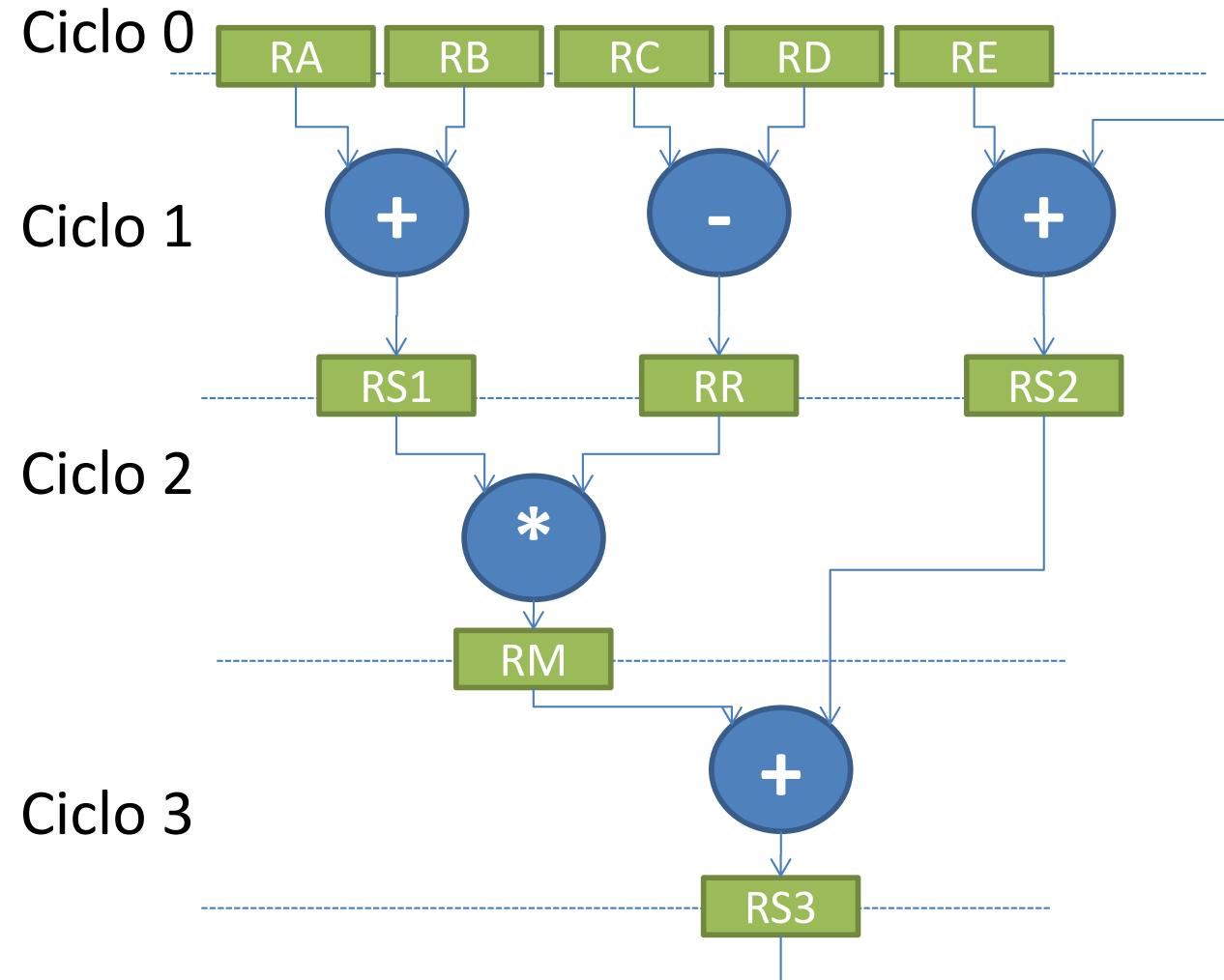


Ejemplo I. Resultado final





Ejemplo II





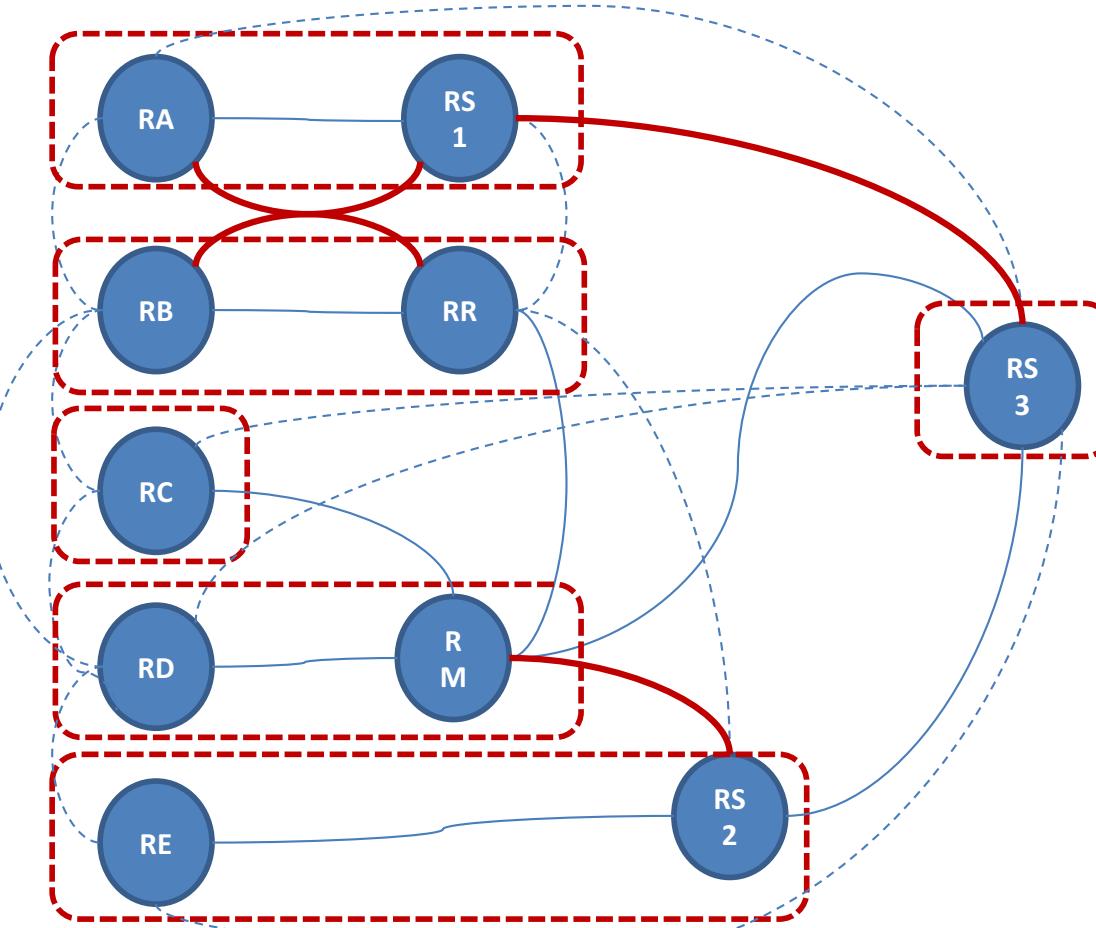
Ejemplo II. Compartir registros

- No son necesarios 11 registros

	S0	S1	S2
RA	X		
RB	X		
RC	X		
RD	X		
RE	X		
RF	X		
RS1		X	
RR		X	
RS2		X	X
RM			X
RS3	X		



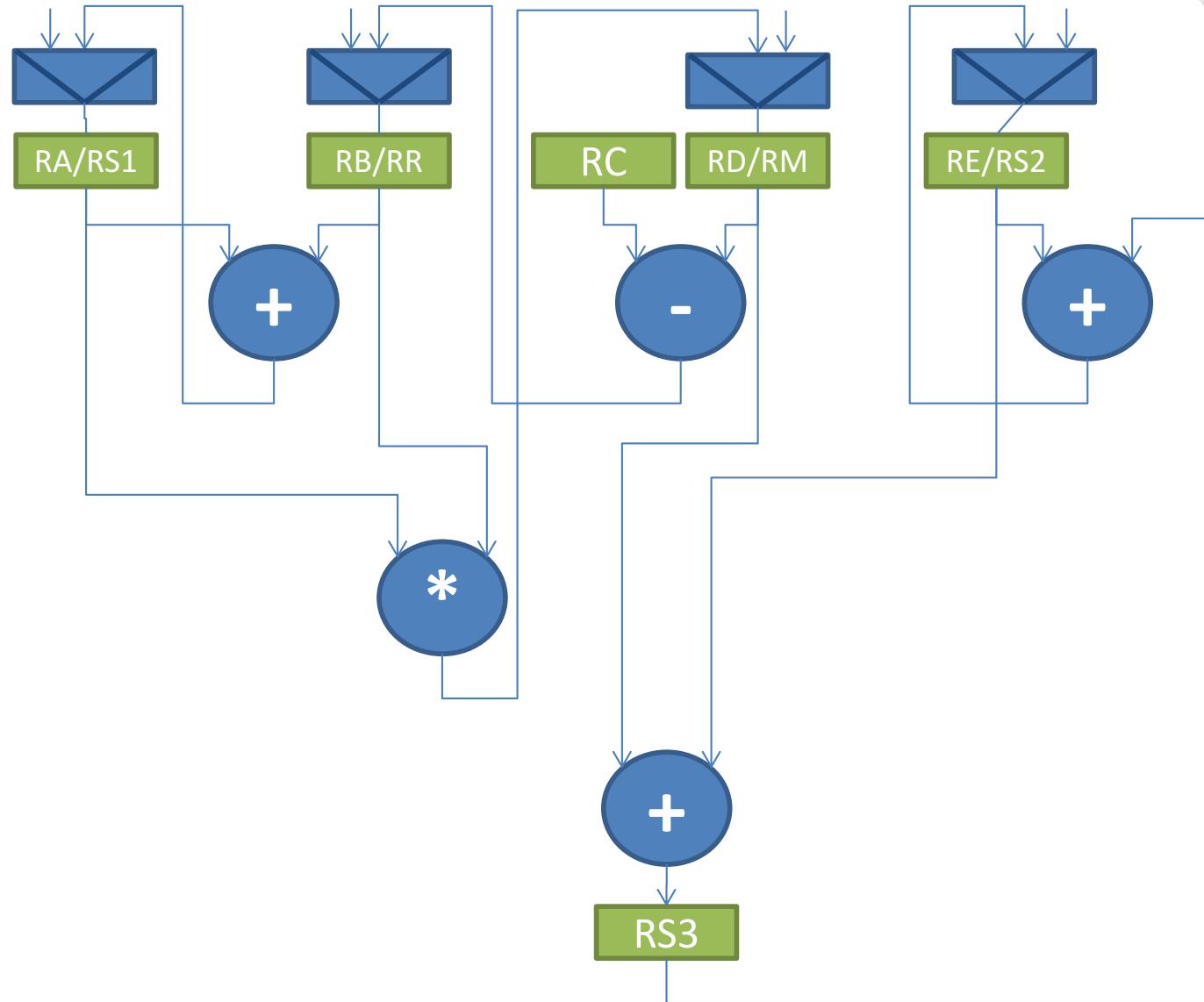
Ejemplo II. Compartir registro

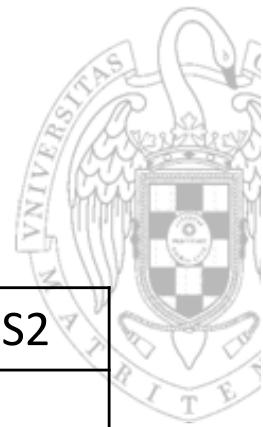


Nota: Para simplificar el esquema no están dibujadas todas las aristas



Ejemplo II. Resultado compartir reg

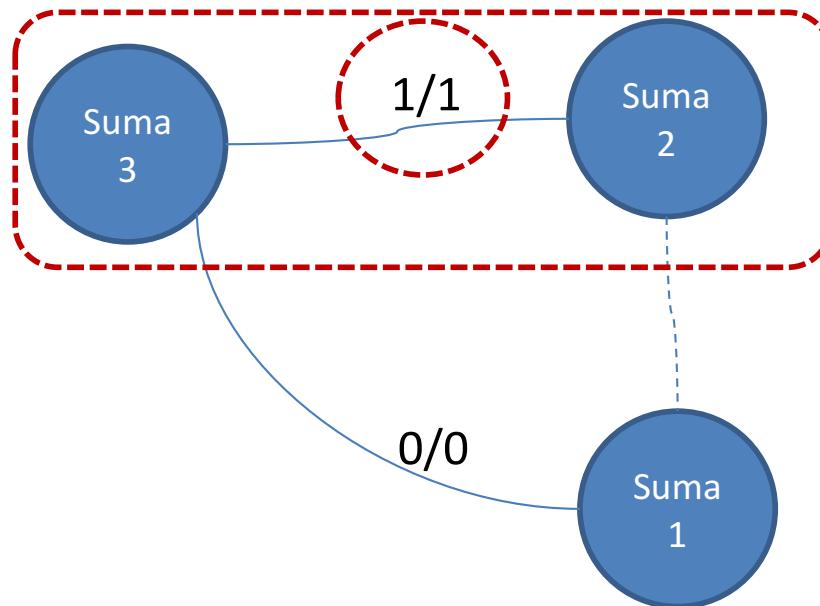




Ejemplo II. Compartir UF

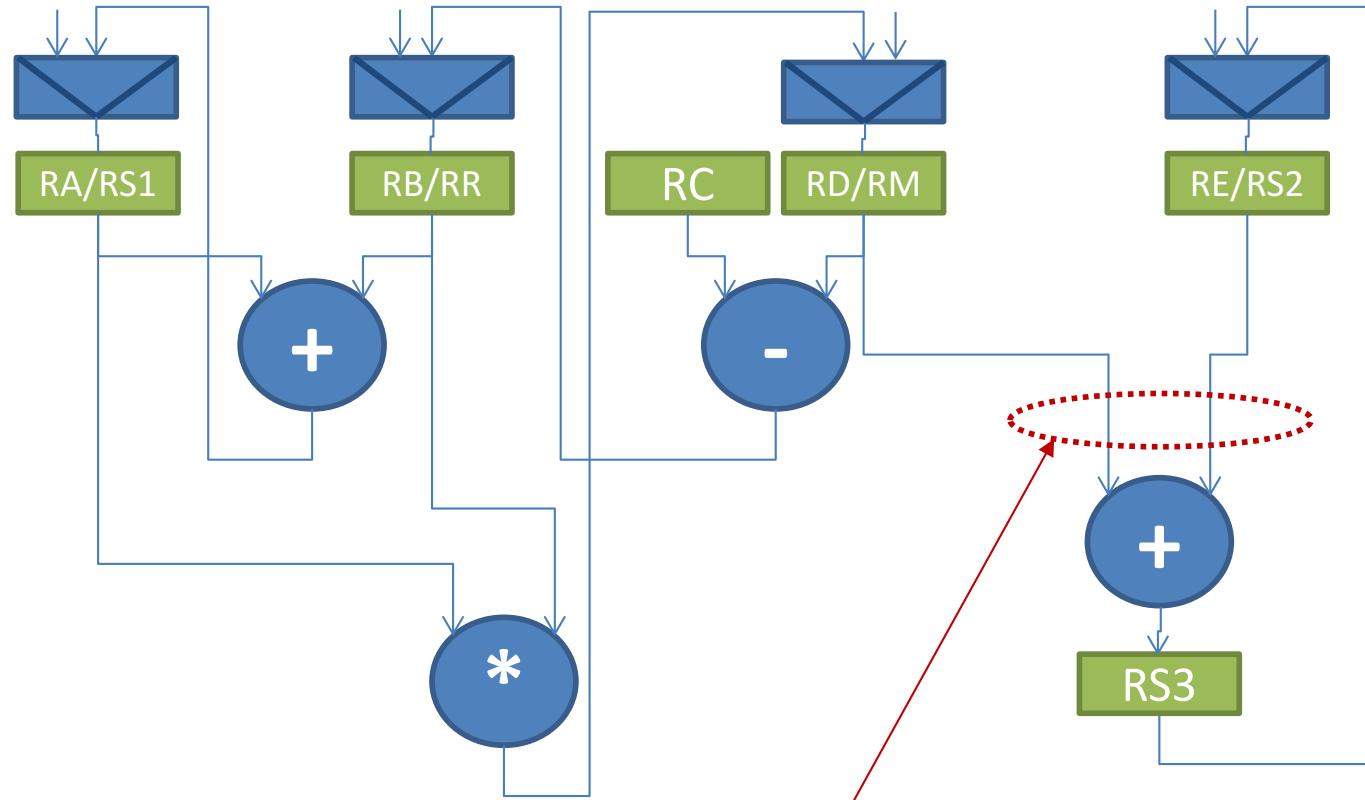
- No hacen falta 3 sumadores
- ¿Compartir Suma 3 con Suma1 o Suma 3 con Suma 2?

	S0	S1	S2
Suma1	X		
Suma2	X		
Suma3			X
Resta1	X		
Mult1		X	





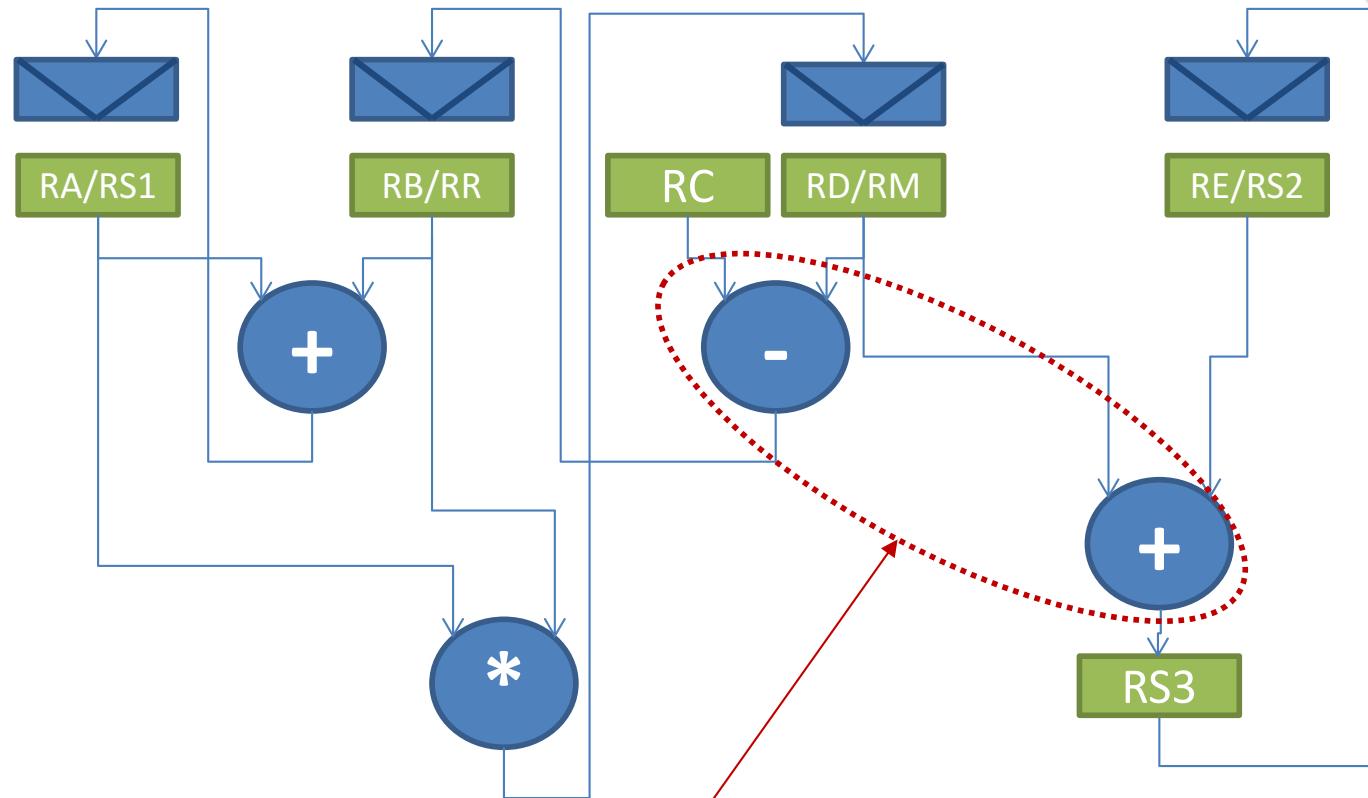
Ejemplo II. Resultado



Para este ejemplo no se necesitan mux a la entrada de la UF compartida, pero no siempre es posible



Ejemplo II. Resultado



Podría ser interesante la transformación de un sumador, en un sumador restador.
Sin embargo en este ejemplo no mejoraría nada porque son necesarios al menos tres
sumadores a la vez



Up-down y Bottom-up

- ¿Cómo nos acercamos al problema?
 - Diseñamos primero los componentes más sencillos y los unimos para crear componentes más complicados (bottom-up)
 - Diseñamos el comportamiento del circuito a alto nivel y cada vez vamos acercándonos más al diseño hardware (up-down)
 - Diseño algorítmico.



Divide y vencerás

- En teoría de la programación el término divide y vencerás hace referencia a uno de los paradigmas de programación más importante.
 - Éste implica la resolución recursiva de un problema dividiéndolo en dos o más sub-problemas de igual tipo o similar. El proceso continúa hasta que éstos llegan a ser lo suficientemente sencillos como para que se resuelvan directamente.
- Podemos aplicar la misma metodología en hardware
 - Simplificando el problema original al dividirlo.
 - Es mucho más fácil y eficiente solucionar problemas pequeños.



Iterativo

- Se trata de resolver un problema mediante aproximaciones sucesivas a la solución, empezando desde una estimación inicial
 - Deseamos obtener un circuito con un determinado tiempo de ciclo.
 - Deseamos obtener un circuito con una determinado consumo de potencia.
- Partiremos de una solución inicial obtenida según la sección anterior (diseño algorítmico) e iremos modificando los componentes y/o unidad de control para conseguir los objetivos de rendimiento.

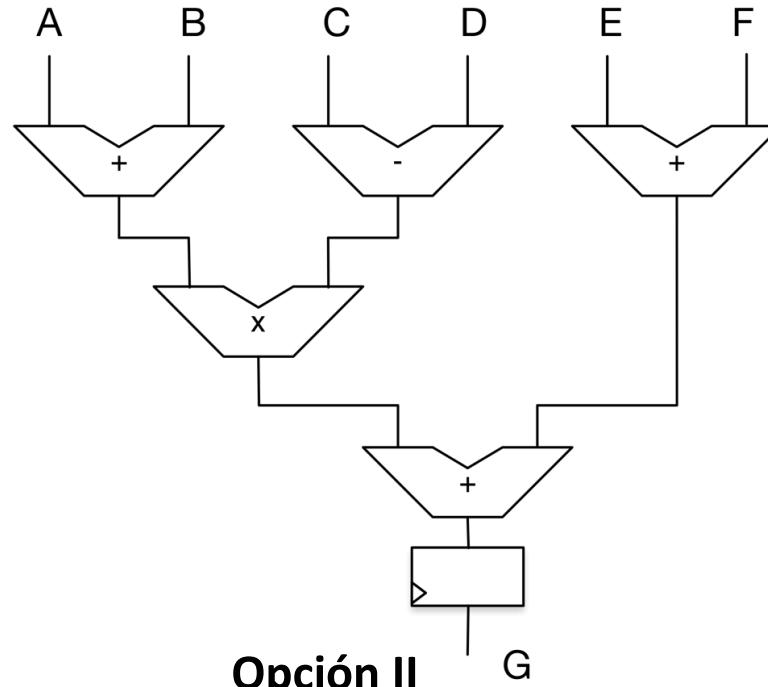


Diseño RTL

- El diseño RTL es la solución natural al problema de implementar un sistema algorítmico.
- Se describe el circuito final como un conjunto de registros y unidades funcionales. Se indica en cada ciclo qué registros y que unidades funcionales se están utilizando.
 - Podríamos decir que así funciona la CPU
- En las siguientes ejemplos vamos a crear código VHDL a partir de un diseño RTL.



Diseño RTL. Ejemplo (I)



Opción II

```
architecture rtl of dsp is
begin
  p_reg : process(clk, rst)
  begin
    if rst = '1' then
      g <= (others => '0');
    elsif rising_edge(clk) then
      g <= ((a+b) * (c-d)) + e + f;
    end if;
  end process p_reg;
end rtl;
```

Opción I

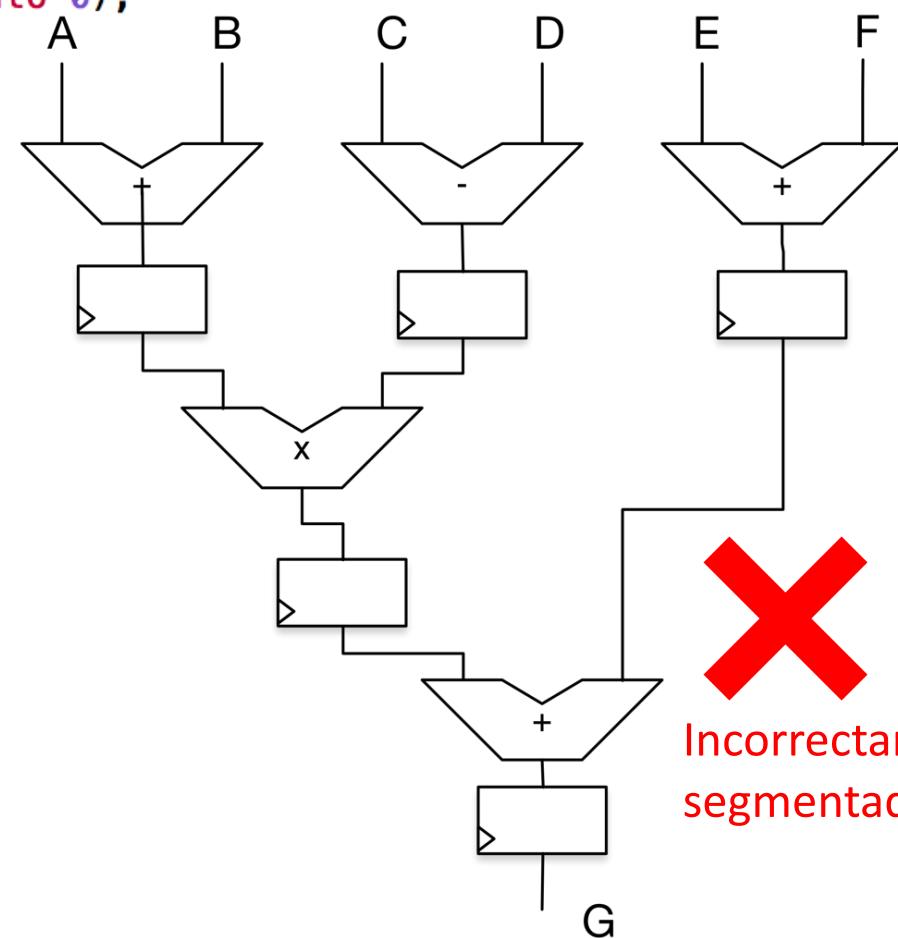
```
architecture rtl of dsp is
signal s1, r, s2, m : unsigned(7 downto 0);
begin
  s1 <= a + b;
  r <= c - d;
  s2 <= e + f;
  m <= s1 * r;
  p_reg : process(clk, rst)
  begin
    if rst = '1' then
      g <= (others => '0');
    elsif rising_edge(clk) then
      g <= m + s2;
    end if;
  end process p_reg;
end rtl;
```



Diseño RTL. Ejemplo (II)

- ¿Qué obtenemos usando el siguiente código?

```
architecture rtl of dsp is
  signal s1, r, s2, m : unsigned(7 downto 0);
begin
  p_reg : process(clk, rst)
  begin
    if rst = '1' then
      s1 <= (others => '0');
      r  <= (others => '0');
      s2 <= (others => '0');
      m  <= (others => '0');
      g  <= (others => '0');
    elsif rising_edge(clk) then
      s1 <= a + b;
      r  <= c - d;
      s2 <= e + f;
      m  <= s1*r;
      g  <= m + s2;
    end if;
  end process p_reg;
end rtl;
```



Incorrectamente
segmentado

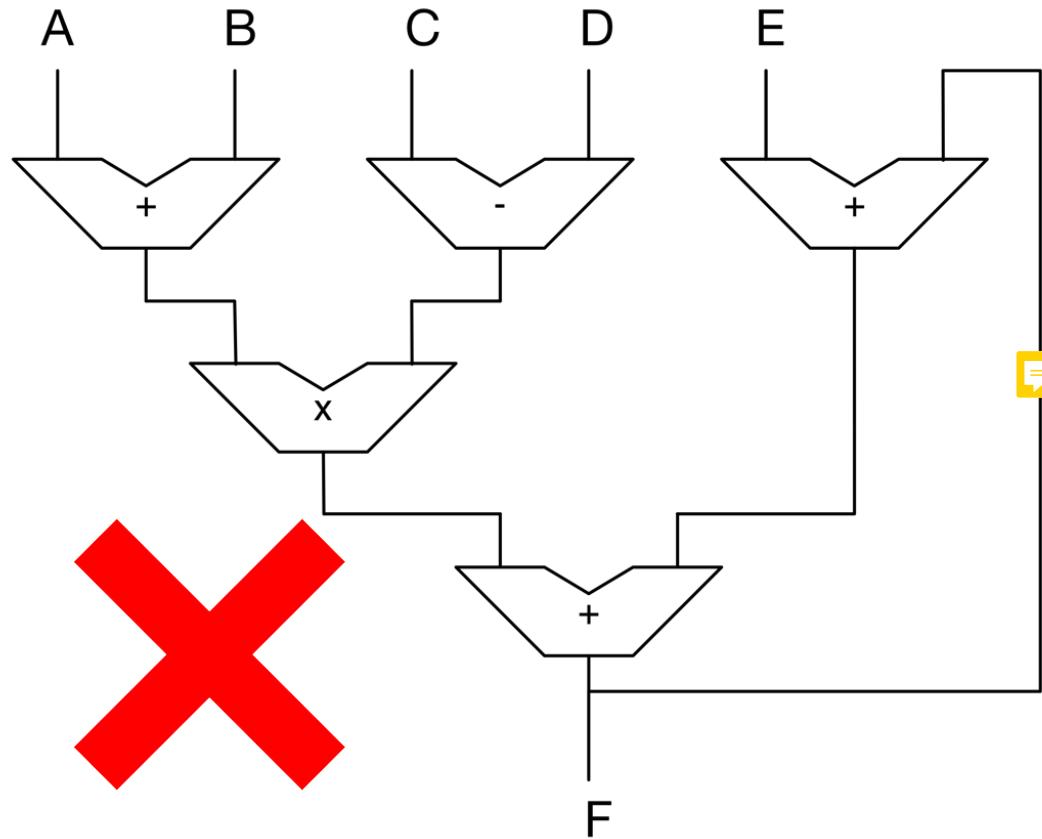


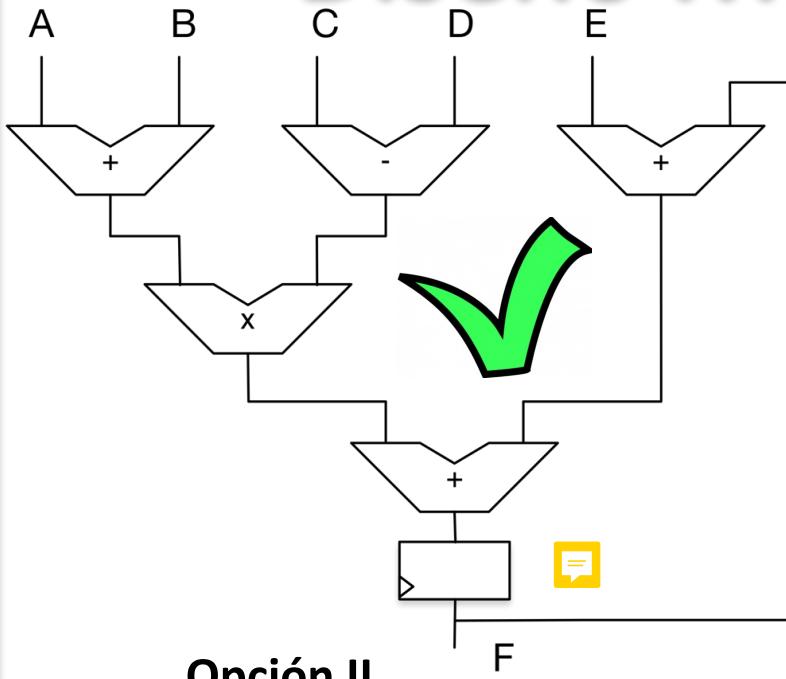
Diseño RTL. Ejemplo (II)

- ¿Cómo segmentar correctamente?

```
architecture rtl of dsp is
    signal s1, r, s2, m, s2_d1 : unsigned(7 downto 0);
begin
    p_reg : process(clk, rst)
    begin
        if rst = '1' then
            s1    <= (others => '0');
            r     <= (others => '0');
            s2    <= (others => '0');
            s2_d1 <= (others => '0');
            m     <= (others => '0');
            g     <= (others => '0');
        elsif rising_edge(clk) then
            s1    <= a + b;
            r     <= c - d;
            s2    <= e + f;
            s2_d1 <= s2;
            m     <= s1*r;
            g     <= m + s2_d1;
        end if;
    end process p_reg;
end rtl;
```

Diseño RTL. Ejemplo (III)



Opción II

```

architecture rtl of dsp is
  signal s3 : unsigned(7 downto 0);
begin
  p_reg : process(clk, rst)
  begin
    if rst = '1' then
      s3 <= (others => '0');
    elsif rising_edge(clk) then
      s3 <= ((a+b) * (c-d)) + e + s3;
    end if;
  end process p_reg;
  f <= s3;
end rtl;
  
```

Opción I

```

architecture rtl of dsp is
  signal s1, r, s2, m, s3 : unsigned(7 downto 0);
begin
  s1 <= a + b;
  r <= c - d;
  s2 <= e + s3;
  m <= s1*r;
  p_reg : process(clk, rst)
  begin
    if rst = '1' then
      s3 <= (others => '0');
    elsif rising_edge(clk) then
      s3 <= m + s2;
    end if;
  end process p_reg;
  f <= s3;
end rtl;
  
```





Diseño RTL. Ejemplo (IV)

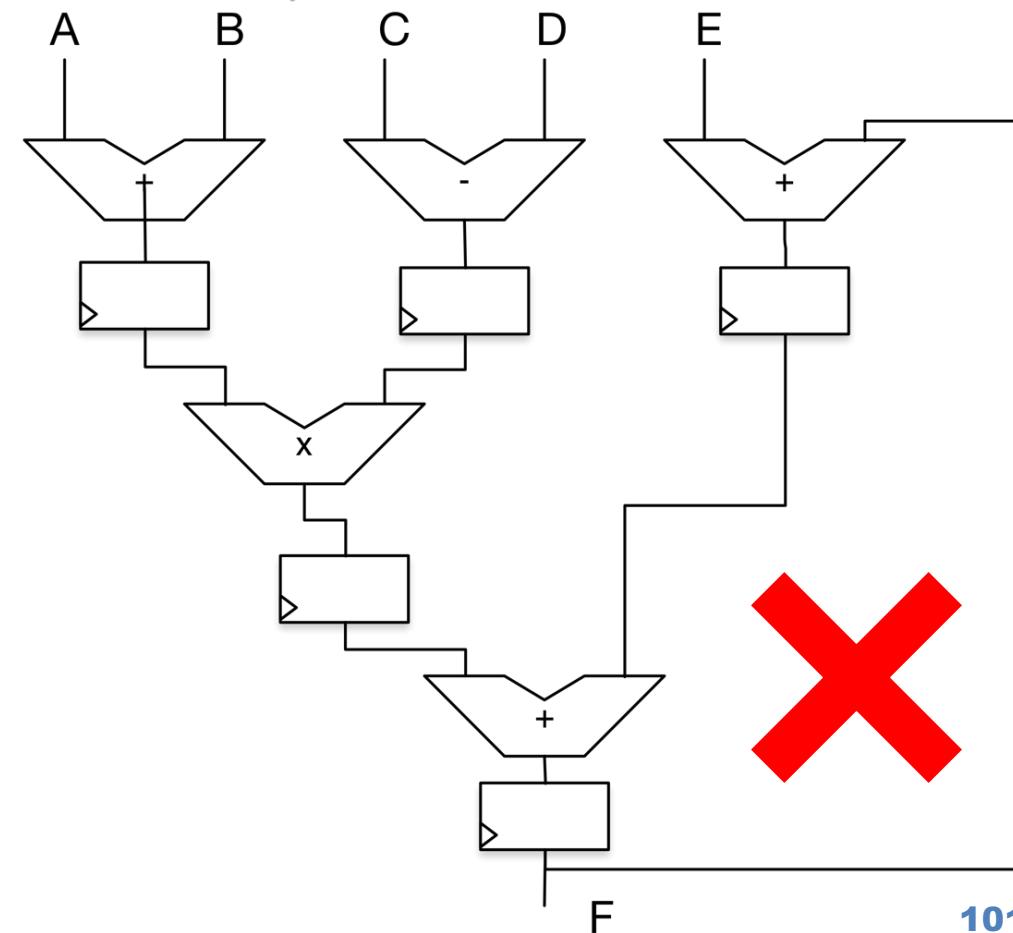
- ¿Qué obtenemos usando el siguiente código?

```

architecture rtl of dsp is
  signal s1, r, s2, m, s3 : unsigned(7 downto 0);
begin
  p_reg : process(clk, rst)
  begin
    if rst = '1' then
      s1 <= (others => '0');
      r <= (others => '0');
      s2 <= (others => '0');
      m <= (others => '0');
      s3 <= (others => '0');

    elsif rising_edge(clk) then
      s1 <= a + b;
      r <= c - d;
      s2 <= e + s3;
      m <= s1*r;
      s3 <= m + s2;
    end if;
  end process p_reg;
  f <= s3;
end rtl;

```



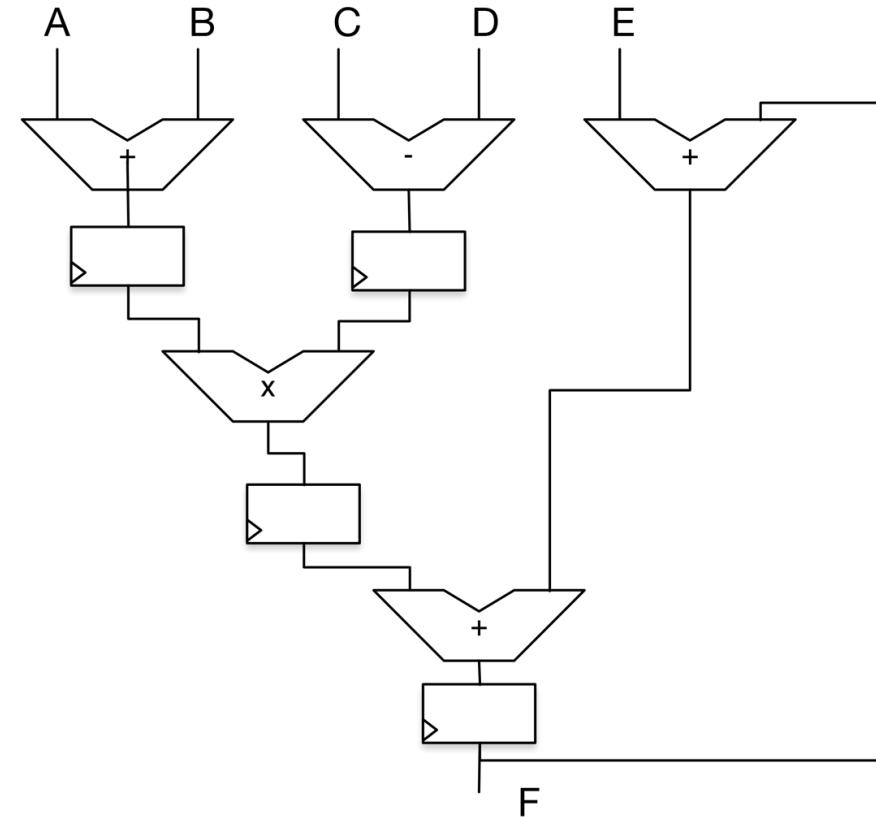


Diseño RTL. Ejemplo (V)

- ¿Cómo arreglarlo?

```

architecture rtl of dsp is
  signal s1, r, s2, m, s3 : unsigned(7 downto 0);
begin
  s2 <= e + s3;
  p_reg : process(clk, rst)
  begin
    if rst = '1' then
      s1 <= (others => '0');
      r  <= (others => '0');
      m  <= (others => '0');
      s3 <= (others => '0');
    elsif rising_edge(clk) then
      s1 <= a + b;
      r  <= c - d;
      m  <= s1*r;
      s3 <= m + s2;
    end if;
  end process p_reg;
  f <= s3;
end rtl;
  
```

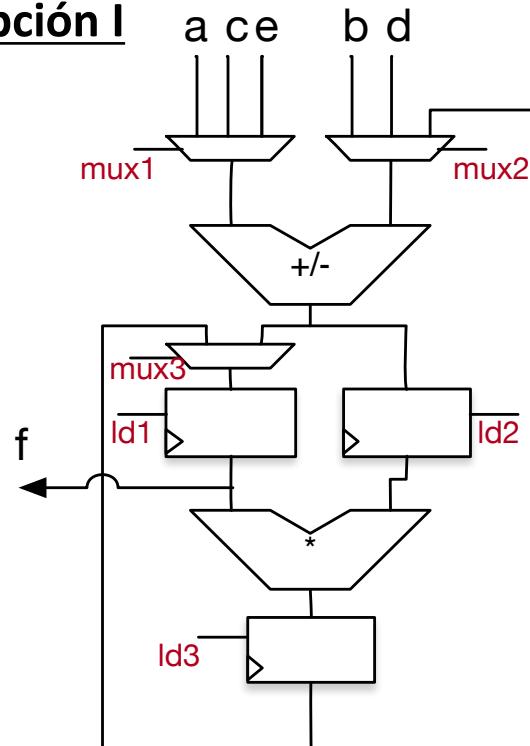




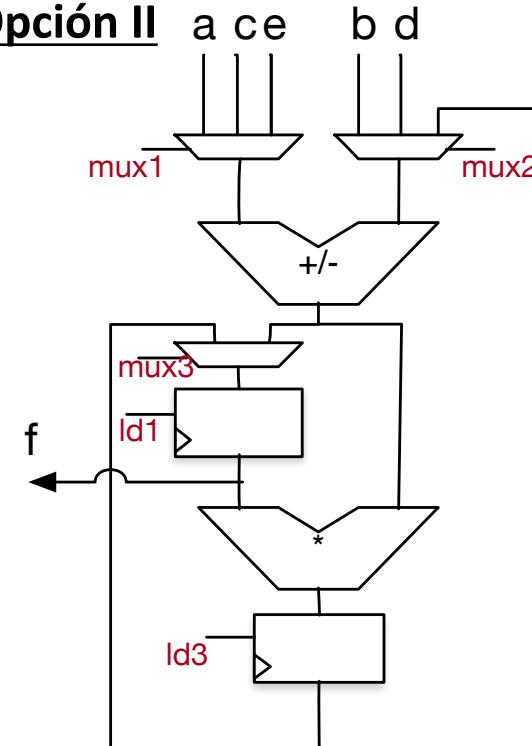
Diseño RTL

- La solución anterior es rápida pero no reutiliza hardware
- ASM más lento pero reutiliza hardware

Opción I



Opción II



Camino
crítico
más lento