9.- Lenguaje de Especificación de Hardware (RTL)

Diseño Lógico 2018

Lenguaje de Esp. de Hardware

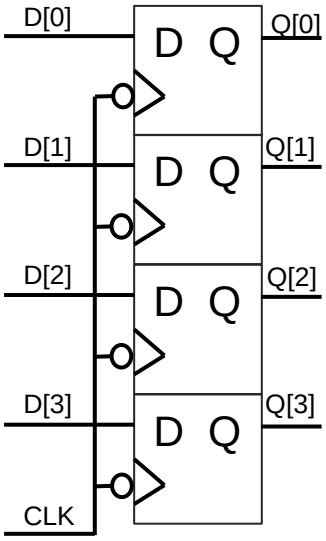
Bibliografía

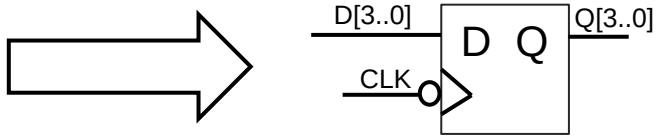
- "Digital Logic and Microprocessors" F. J. Hill y G. R. Peterson
  - Cap 9 y 10, principalmente el cap 10.

Lenguaje de Esp. de Hardware

- Antes de comenzar…….

- Registro (memoria):
  - Ej. registro de 4 bits



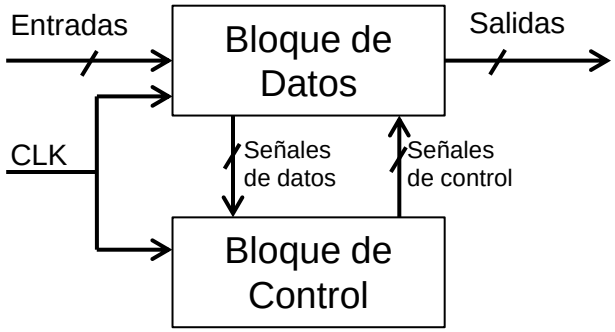


- Las señales de D y Q son independientes.
- La señal CLK es común a todos los FFs

Lenguaje de Esp. de Hardware

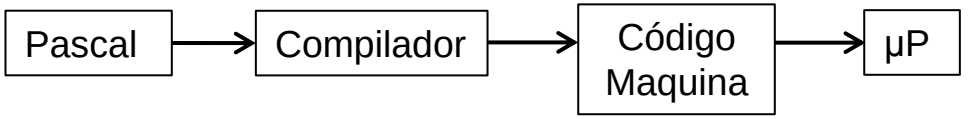
- ¿Qué pasa si los sistemas crecen?
  - Aumenta la cantidad de estados!!
  - Imaginen un diagrama de estados con 650987 estados… apenas 20 FFs!!!!!
  - Con las herramientas vistas se hace inmanejable.

- Dos temas mezclados:
  1. Necesidad de un lenguaje
  2. Separar el circuito en 2 partes
     - Bloque de Control
     - Bloque de Datos

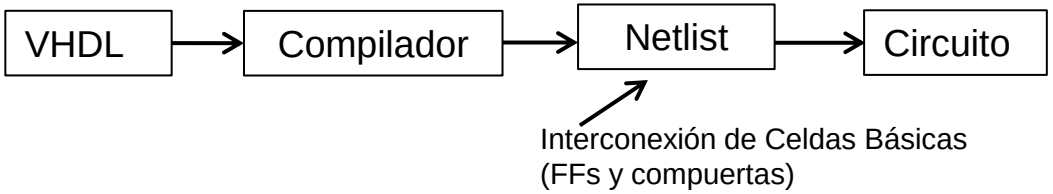


Lenguaje de Esp. de Hardware

- Lenguaje de Software
  - Pascal, Basic, C, Java, etc.



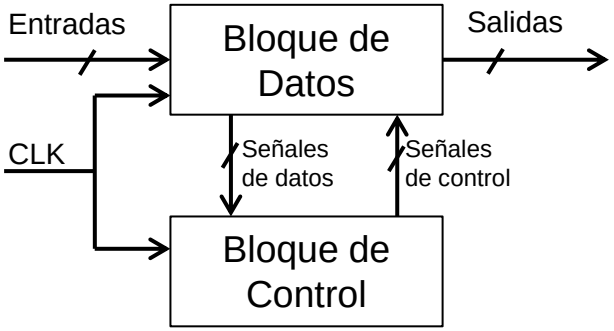
- Lenguaje de Hardware
  - Ejemplos:
    - VHDL:
      - VHSIC *(Very High Speed Integrated Circuit)*+ HDL *(Hardware Description Language)*
      - Desarrollado en los 80's para simular circuitos
      - IEEE Std 1076-1987 …. 1993….2000… 2002….2008 ….
    - Verilog:
      - Sintaxis similar al lenguaje de programación C
      - Desarrollado en 1985
  - Especifican el comportamiento de un circuito (formal)
  - Simular
  - A partir de la especificación se puede <u>sintetizar</u> un circuito en forma automática (subconjunto de VHDL o Verilog)



Lenguaje de Esp. de Hardware

RTL (Register Transfer Language)

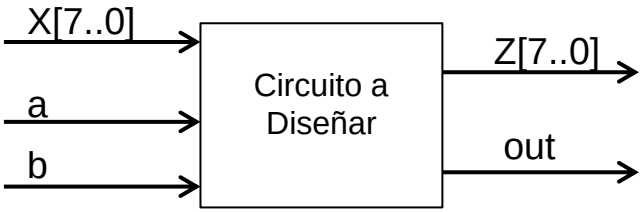
- Lenguaje simple
- Es fácil “compilar” a mano.
- Es fácil comprender los conceptos de diseño.
- Separa el circuito en Bloque de Datos y Bloque de control



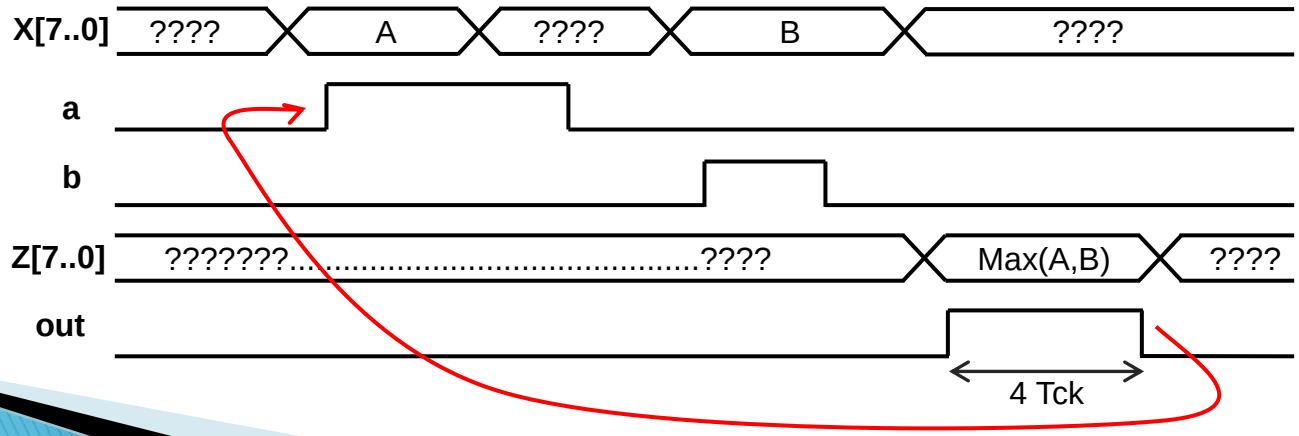
- Bloque de Datos:
  - Lógica combinatoria
  - Memoria
- Bloque de control
  - Circuito Secuencial
- Señales de Datos
  - Entradas del circuito secuencial
- Señales de Control
  - Condiciones para:
    - Lógica combinatoria
    - Transferencia de datos a Memoria

Lenguaje de Esp. de Hardware

Ejemplo: *(Cap. 10 pag. 229)*

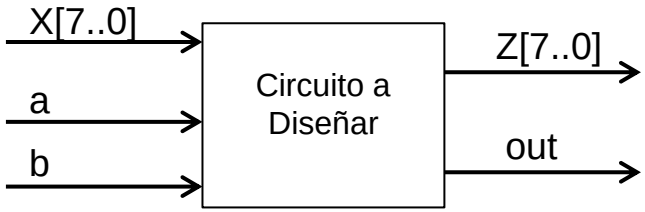


- Comparador de 2 números binarios A y B
  - Primero '**a**' pasa de 0 a 1 ➔ A[] válido en '**X[]**'
  - Tiempo después '**b**' pasa de 0 a 1 ➔ B[] válido en '**X[]**'
  - Al recibir B[], la salida '**out**' = 1 durante los 4 Tck y '**Z[]**' = Max(A[],B[]).
  - El resto del tiempo '**out**' = 0 y '**Z[]**' no está determinado.
  - Luego de los 4 Tck, se vuelve al estado inicial. Se supone que '**a**' y '**b**' ya bajaron.

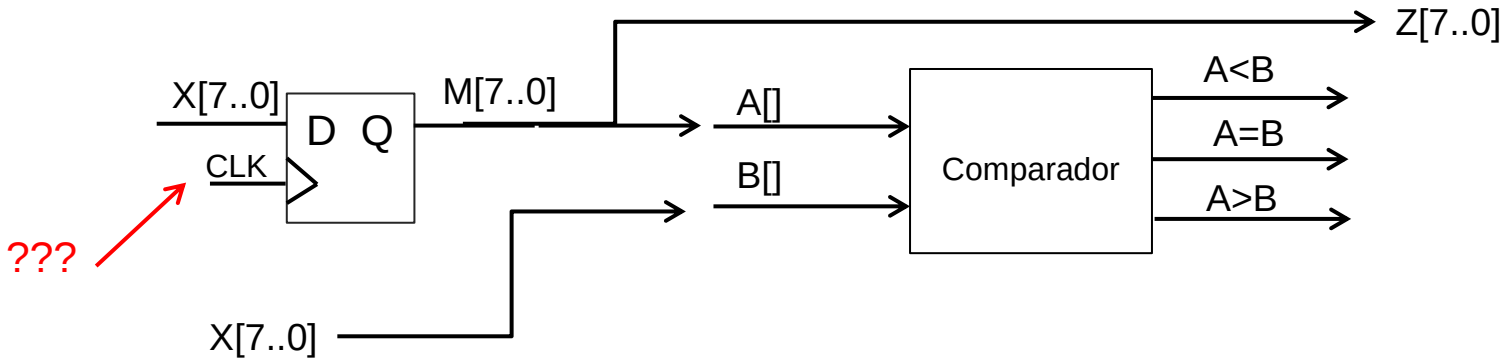


Lenguaje de Esp. de Hardware

Ejemplo: *(continuación)*



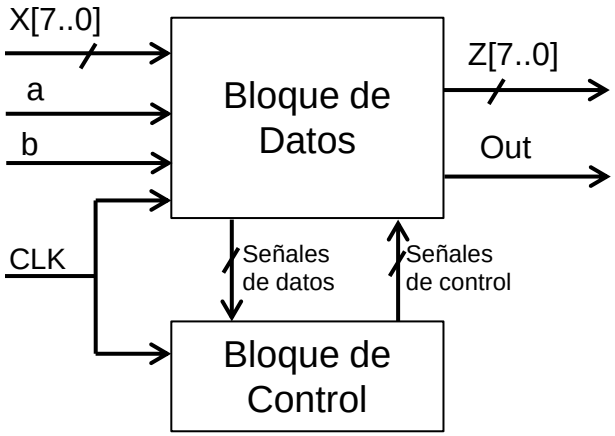
- ¿Memoria (registros)?
- ¿Qué bloques de combinatoria necesitamos?
- ¿Cómo lo hacemos?
  - a=1 ➔ guardo A en M
  - b=1 ➔ comparo M con X
  - y … Si X>M cargo X en M
  - Z = M “durante 4 Tck”



Lenguaje de Esp. de Hardware

Ejemplo: *(continuación)*

- Partimos el problema en 2 bloques:



Aclaración

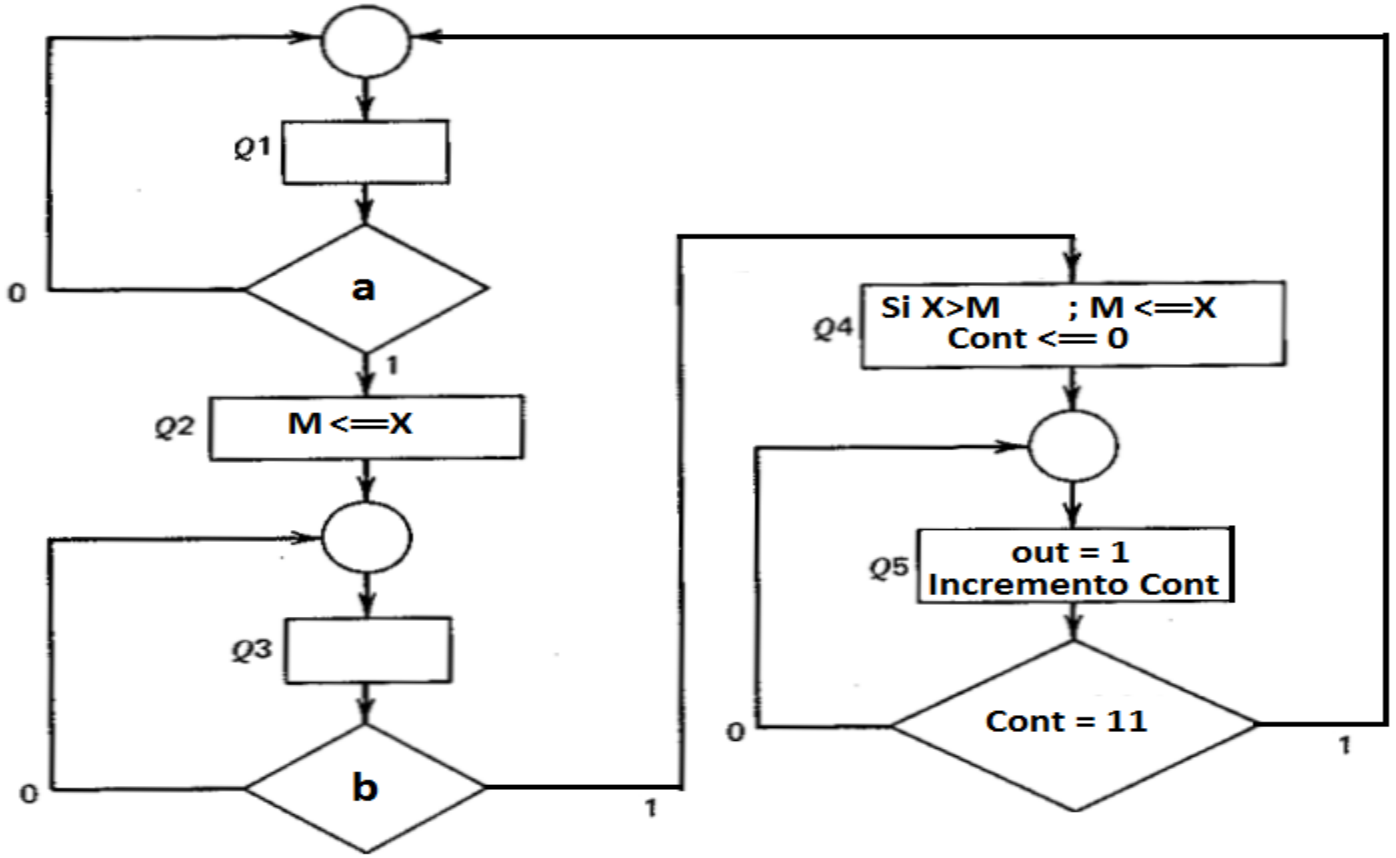
El Contador[2] se podría implementar como parte de la secuencia del Bloque de Control, pero no es lo usual y genera soluciones más confusas.

- Bloque de Datos:
  - Lógica comparador
  - M[8] para guardar el dato X[8]
  - Contador[2] para llevar la cuenta de 4 Tck
  - Fin de contador: Cont[1].Cont[0]
  - Incrementador para incrementar Contador[2]
- Señales de Datos
  - a
  - b
  - Fin Contador
  - A>B
  - ……
- Bloque de control
  - Circuito Secuencial
- Señales de Control
  - ……

Lenguaje de Esp. de Hardware

Ejemplo: *(continuación)*

Diagrama de flujo:



RTL (Register Transfer Language)

Notación RTL

- En este lenguaje mantiene una relación uno a uno con la realización del hardware.
- Esto nos va a permitir “compilarlo” a mano.

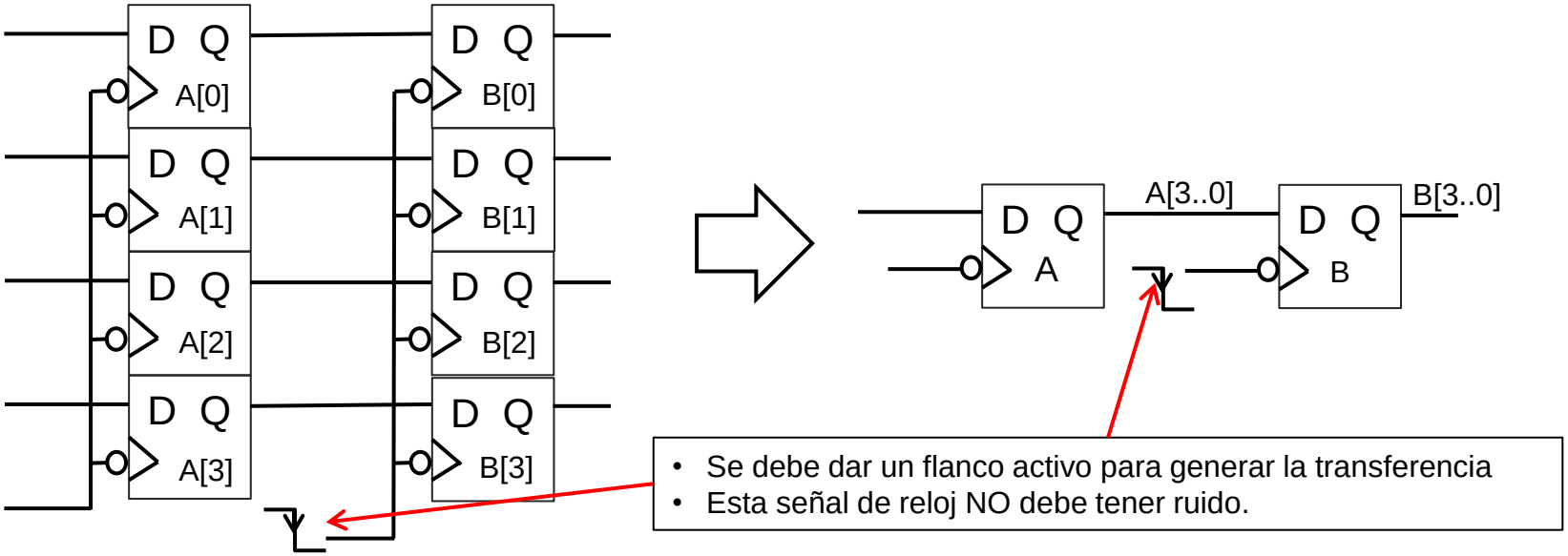
- Escalares:
  - Señales de 1 bit
  - Se representan con letra minúscula.
  - Ejemplo: a, b

- Vectoriales:
  - Señales de n bits, n > 1
  - Se representan con letra mayúscula.
  - Ejemplo para 8 bits: A[8] o A[7..0]

RTL (Register Transfer Language)

Transferencia a un registro

- B ← A ó B[] ← A[]
- Si no se especifica, la transferencia es ordenada: B[0] ← A[0], B[1] ← A[1] , etc.
- A[] no se modifica

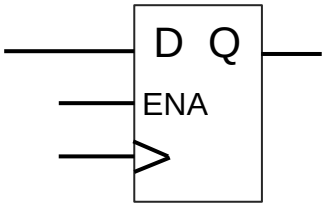


Aclaración: RTL describe que el flanco activo es el flanco de bajada. Nosotros consideraremos que puede ser cualquiera, siempre que exista coherencia en todo el circuito.

RTL (Register Transfer Language)

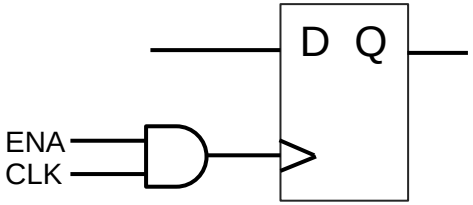
FF con Enable

- Para evitar el “ruido” en la señal de reloj vamos a utilizar FFs con Enable.

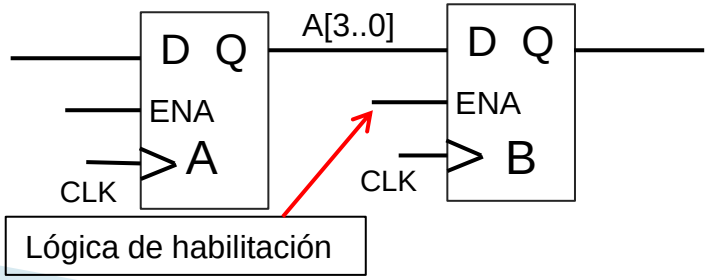


- Si ENA = 1 ➔ el FF responde a los flancos activos de reloj.
- Si ENA = 0 ➔ el FF no responde a los flancos de reloj.
- El FF es “inmune” al ruido en la señal ENA.

- Sería algo aproximado (no igual) a:



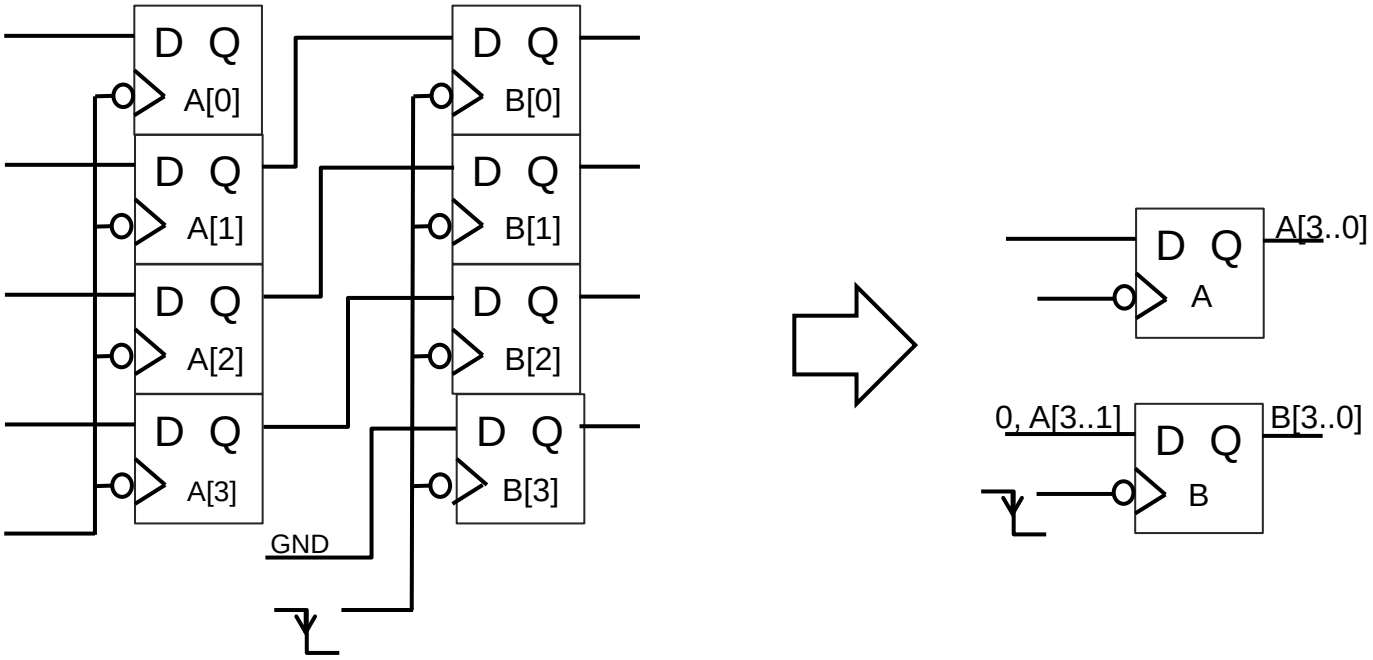
- Transferencia a un registro con FF con Enable
  - B[] ➔ A[]



RTL (Register Transfer Language)

Desplazamiento

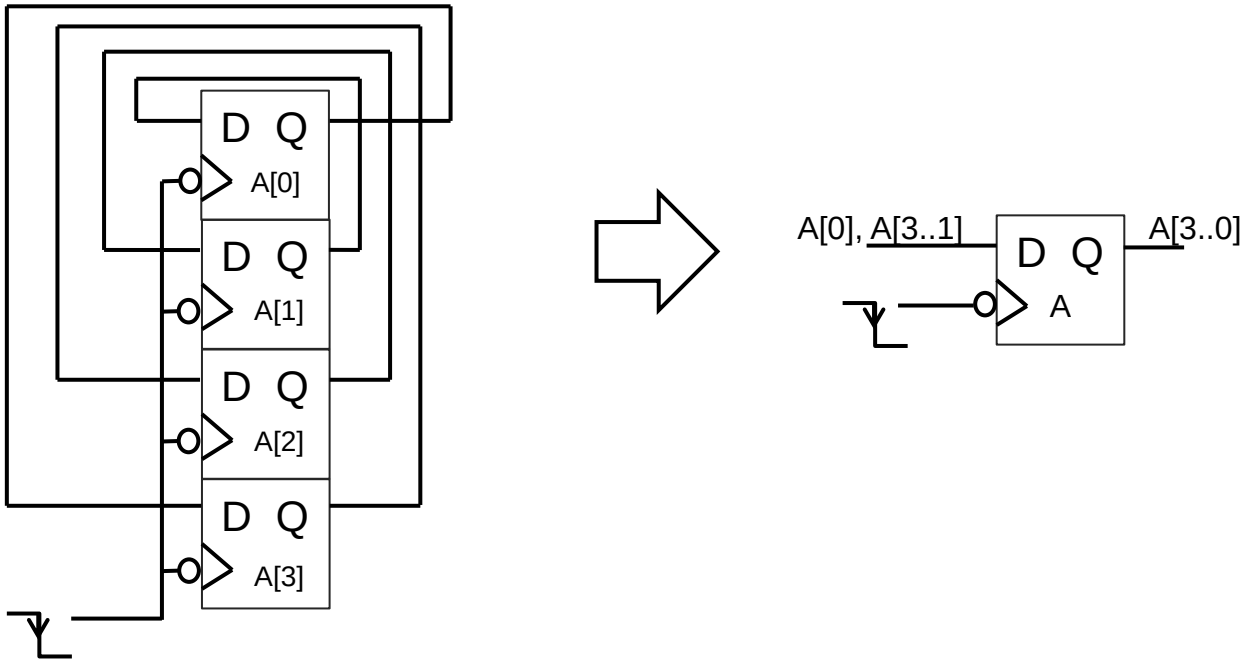
- B[] ← 0, A[3..1]



RTL (Register Transfer Language)

Shift Register

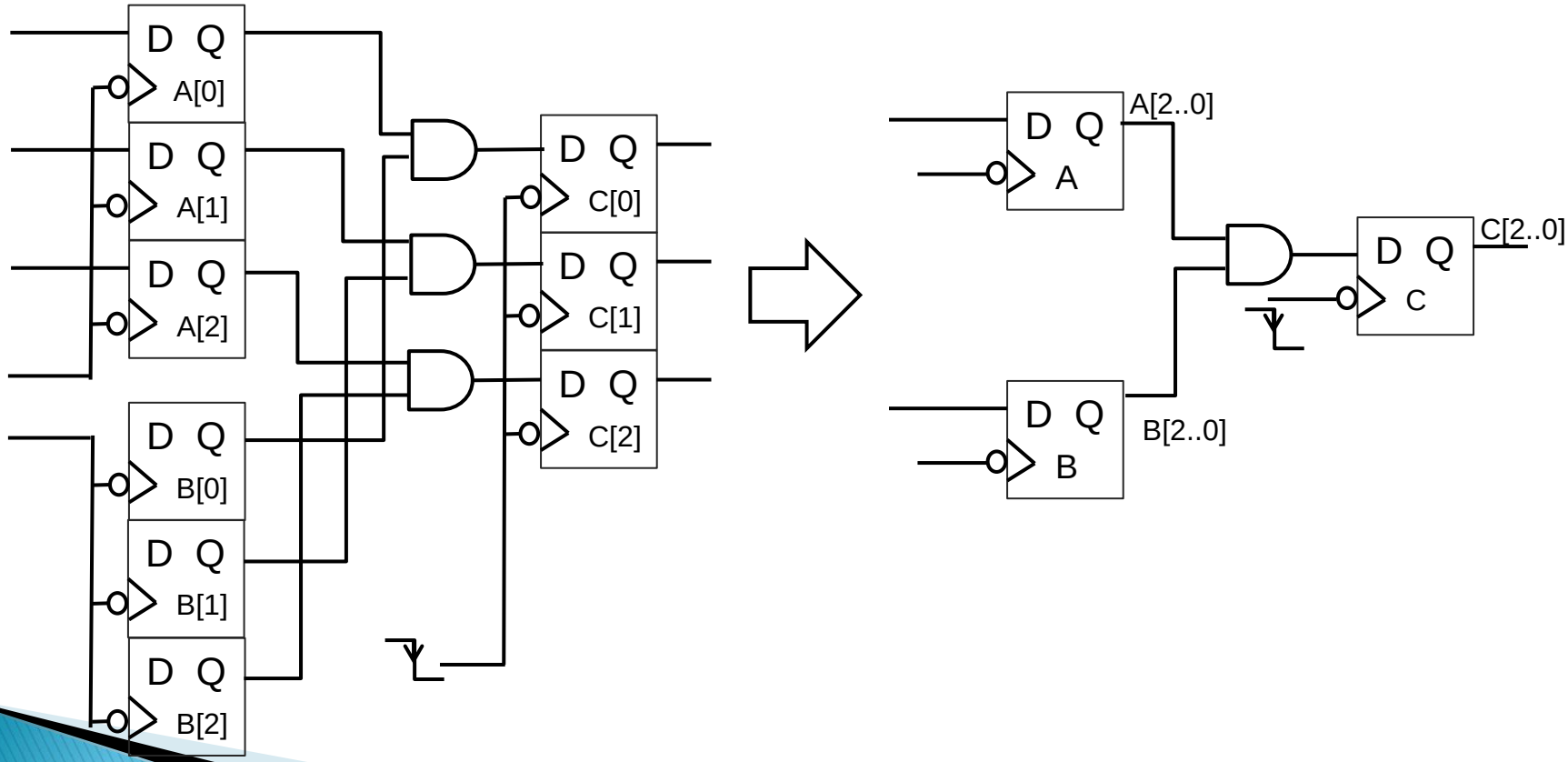
- A[] ← A[0], A[3..1]



RTL (Register Transfer Language)

Transferencia de una operación

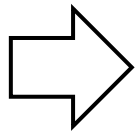
- C[] ← A[] AND B[]

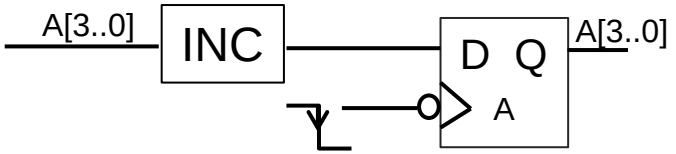


RTL (Register Transfer Language)

Contador

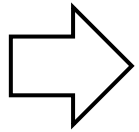
- A[] ← INC (A[])

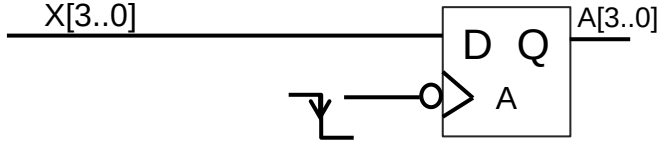




Transferencia de entrada

- A[] ← X[]
- X[] es entrada

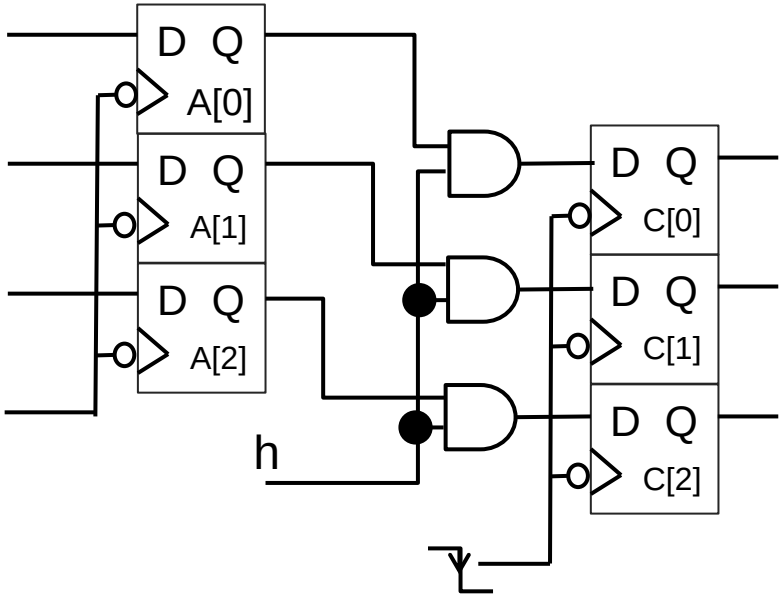


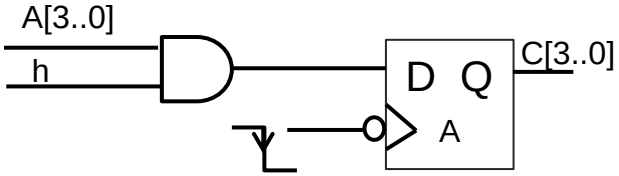


RTL (Register Transfer Language)

Operación entre vector y escalar

- A y C vectores y h escalar
- C[] ← A[] and h



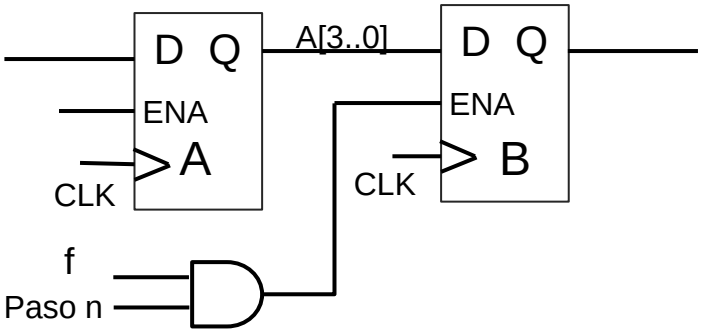


RTL (Register Transfer Language)

Transferencia condicional

- B[] * f ← A[]
- A y B vectores, f función lógica

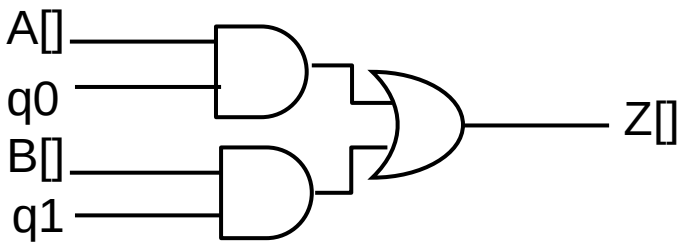
- Si f= 1 ➔ se transfiere A[] en B[]
- Si f=0 ➔ no se realiza transferencia y por tanto B mantiene su valor.



RTL (Register Transfer Language)

Conexión lógica

- NO existe transferencia, no involucra FFs.
- La conexión es lógica durante un determinado tiempo.
- Se utiliza para Salidas o señales internas.
- Por ejemplo si Z es una salida
- **En q0: Z[] = A[]**
- **En q1: Z[] = B[]**



- q1 y q0 son escalares y A, B y Z vectoriales.
- Luego a q1 y q0 le vamos a llamar Paso 1 y Paso 0.

RTL (Register Transfer Language)

Secuencia RTL

```
Module: DataMover
Input: X[2]
Output: Z[2]
Memory: A[2]; B[2]; C[2]

 1 - A ← X
 2 - C ← !A
 3 - B ← C[0],C[1]
 4 - C ← A OR B
 5 - Z = C

ENDSEQUENCE
CONTROLRESET(1)
.. .. ..
.. .. ..
END
```

TRANSFERENCIA

CONEXIÓN LÓGICA

Nombre del Módulo

Declaraciones de entradas, salidas y registros

Secuencia por paso

Fin de secuencia por pasos

Luego de un Reset va a paso 1

Secuencia para todos los pasos

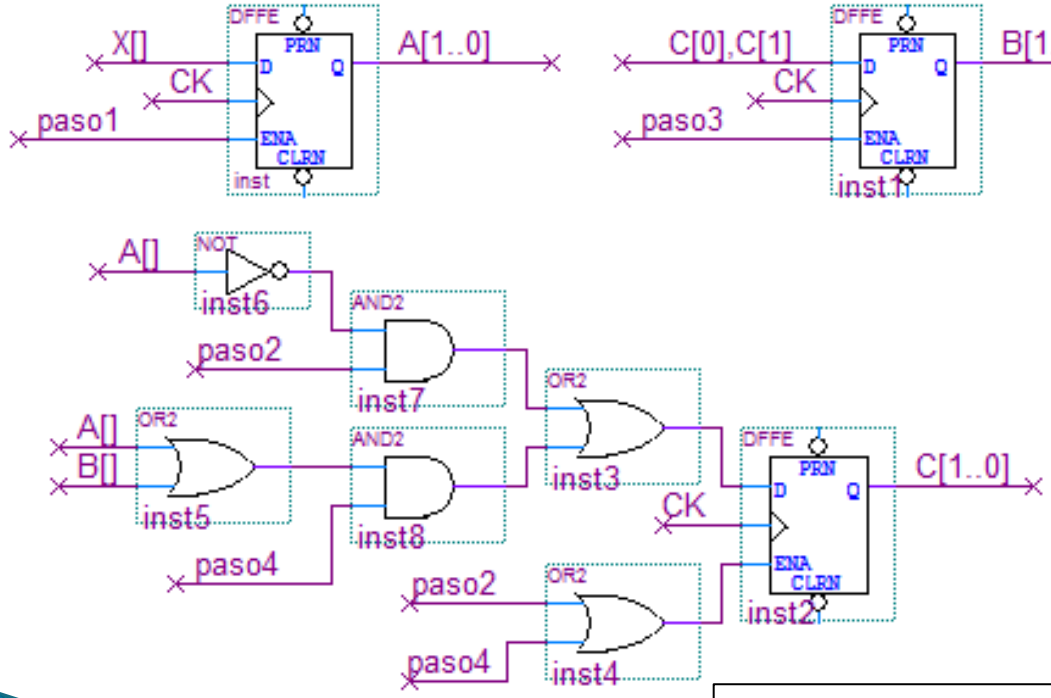
Fin de descripción

RTL (Register Transfer Language)

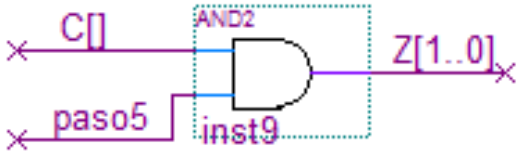
Bloque de datos

Vamos a considerar las señales ***paso_n***, n=1,2,3,4,5

- **paso_n** = 1 si estamos en el paso_n
- **paso_n** = 0 si estamos en un paso ≠ n




```
Module: DataMover
Input: X[2]
Output: Z[2]
Memory: A[2]; B[2]; C[2]

  1 - A ← X
  2 - C ← !A
  3 - B ← C[0],C[1]
  4 - C ← A OR B
  5 - Z = C

ENDSEQUENCE
CONTROLRESET(1)
END
```

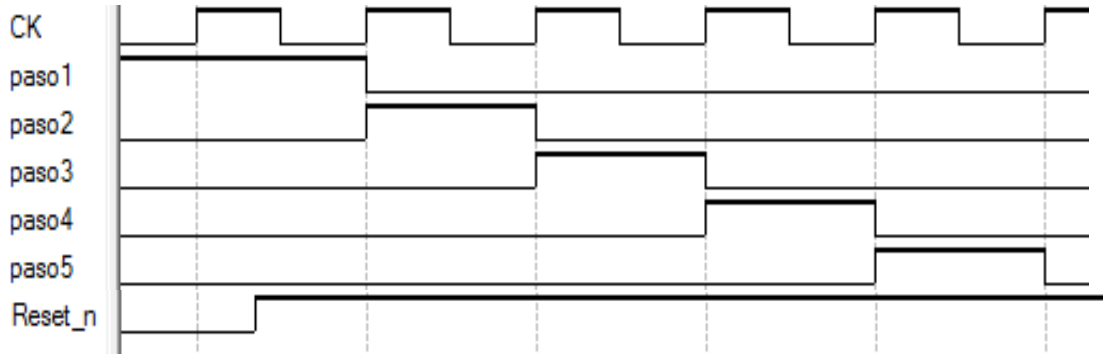


OBS: Cada memoria implica una lógica con su Registro
Cada salida implica una lógica SIN Registros

RTL (Register Transfer Language)

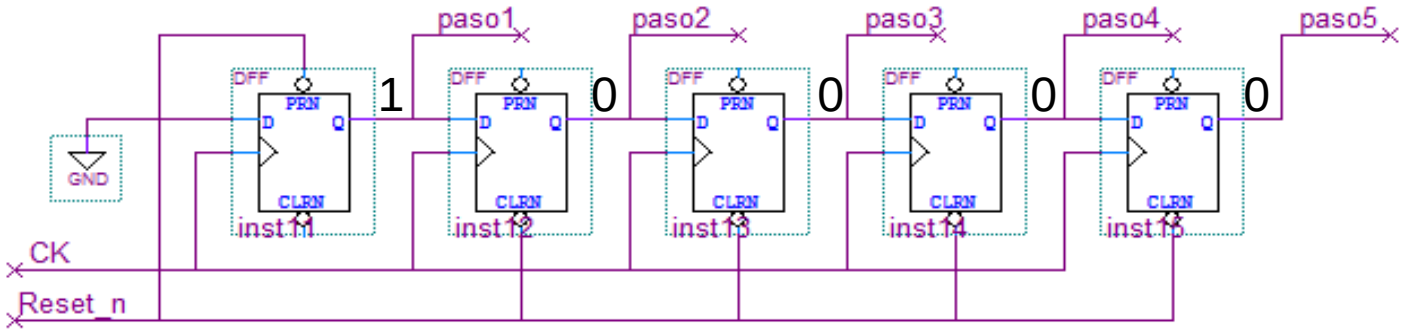
Bloque de control

De acuerdo a la descripción, los pasos son uno a continuación del otro; paso1, paso2, paso3….:



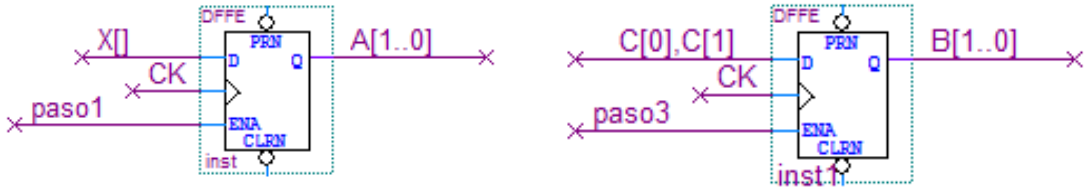

```
Module: DataMover
Input: X[2]
Output: Z[2]
Memory: A[2]; B[2]; C[2]
  1 - A ← X
  2 - C ← !A
  3 - B ← C[0],C[1]
  4 - C ← A OR B
  5 - Z = C
ENDSEQUENCE
CONTROLRESET(1)
END
```

1FF por cada paso



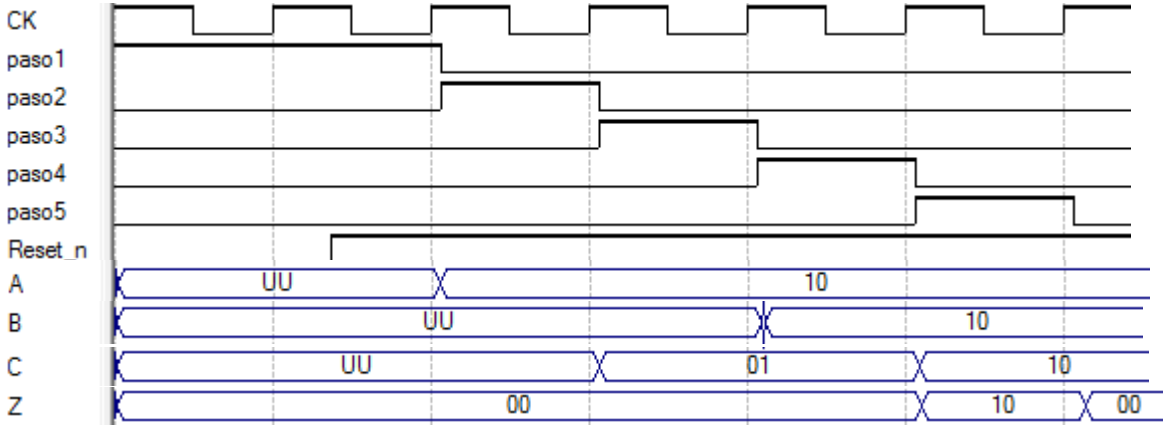
RTL (Register Transfer Language)

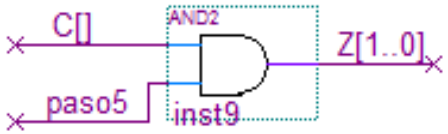
¿En que instante se hacen las transferencias?




```
Module: DataMover
Input: X[2]
Output: Z[2]
Memory: A[2]; B[2]; C[2]
  1 - A ← X
  2 - C ← !A
  3 - B ← C[0],C[1]
  4 - C ← A OR B
  5 - Z = C
ENDSEQUENCE
CONTROLRESET(1)
END
```

Supongamos X=10 e inicialmente: A=B=C=UU (valor desconocido)





IMPORTANTE:

- Las transferencias siempre se hacen al final del paso.
- Las salidas se “conectan” durante todo el paso y vale “0” en los pasos que no se usan.

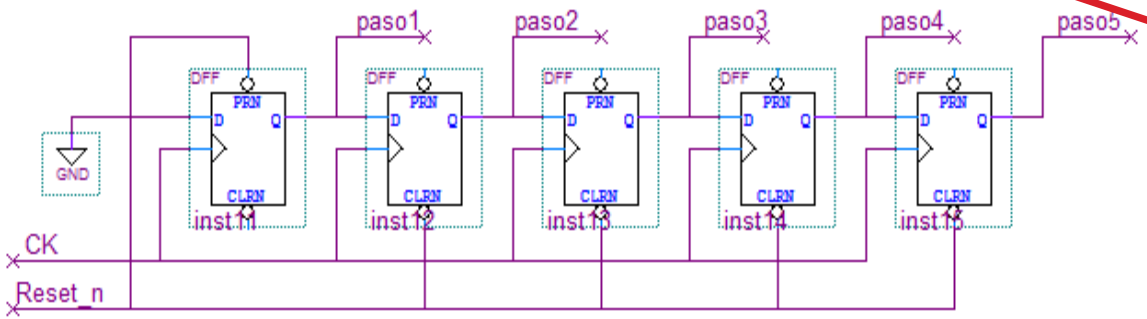
RTL (Register Transfer Language)

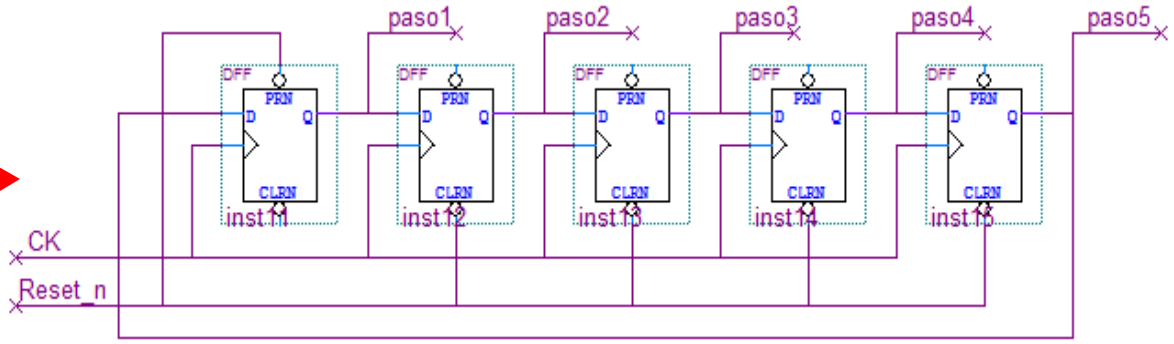
Bifurcación incondicional

Notación: ➔ (N° paso)

Si en el ejemplo del DataMover queremos que luego del paso 5 ir al paso 1 hay que cambiar…..

```
Module: DataMover
Input: X[2]
Output: Z[2]
Memory: A[2]; B[2]; C[2]
  1 - A ← X
  2 - C ← !A
  3 - B ← C[0],C[1]
  4 - C ← A OR B
  5 - Z = C
    ➔ (1)
ENDSEQUENCE
CONTROLRESET(1)
END
```





RTL (Register Transfer Language)

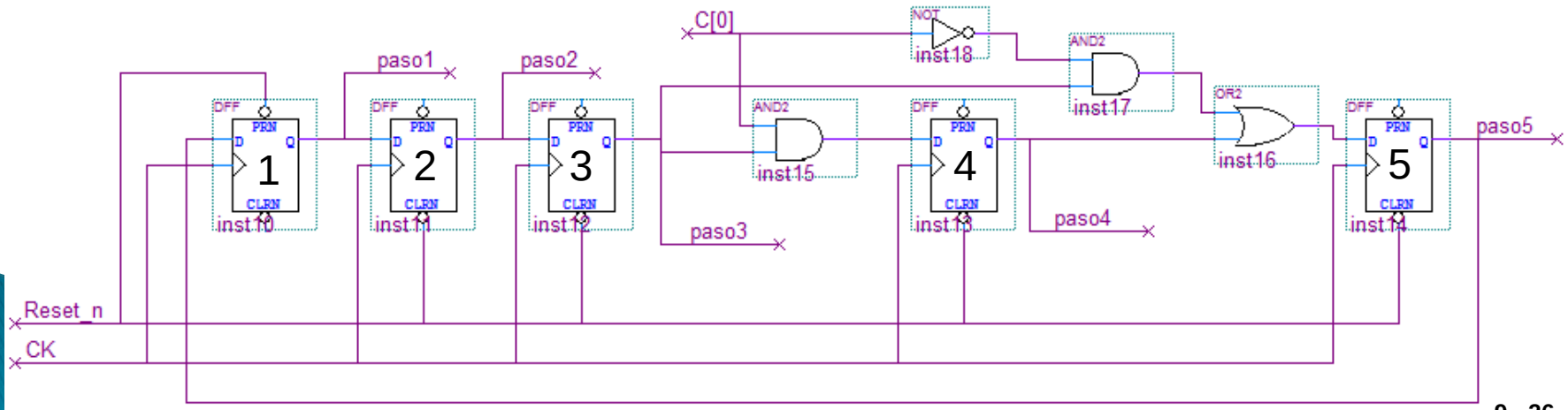
Bifurcación condicional

Notación: ➔ $(f_1; f_2; \ldots. f_n)/ (S_1, S_2, \ldots S_n)$

fi: funciones combinatorias disjuntas (solo una por vez vale 1)
ni: n° de paso

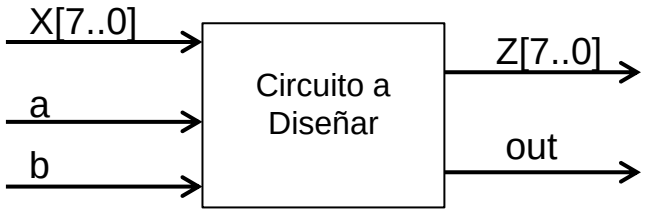
Si fi = 1 ➔ el próximo paso debe ser Si
Si fi = 0 para todo i ➔ se ejecuta el próximo paso.
Para evitar errores, es mejor especificar todo

```
Module: DataMover
Input: X[2]
Output: Z[2]
Memory: A[2]; B[2]; C[2]
  1 - A ← X
  2 - C ← !A
  3 - B ← C[0],C[1]
      ➔(C[0],/C[0])/(4,5)
  4 - C ← A OR B
  5 - Z = C
      ➔ (1)
ENDSEQUENCE
CONTROLRESET(1)
END
```

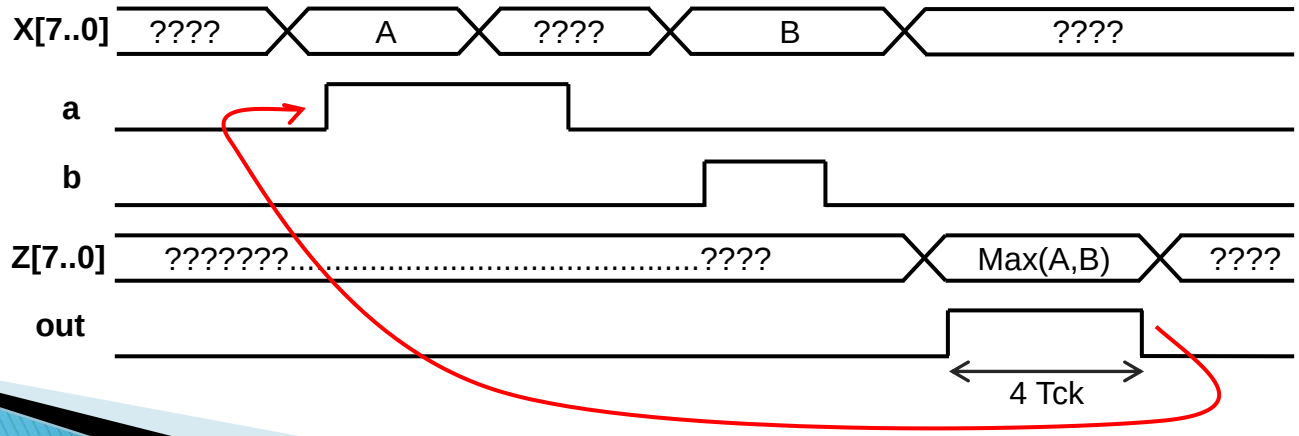


Diseño RTL

Comparador: (VOLVEMOS AL EJEMPLO)

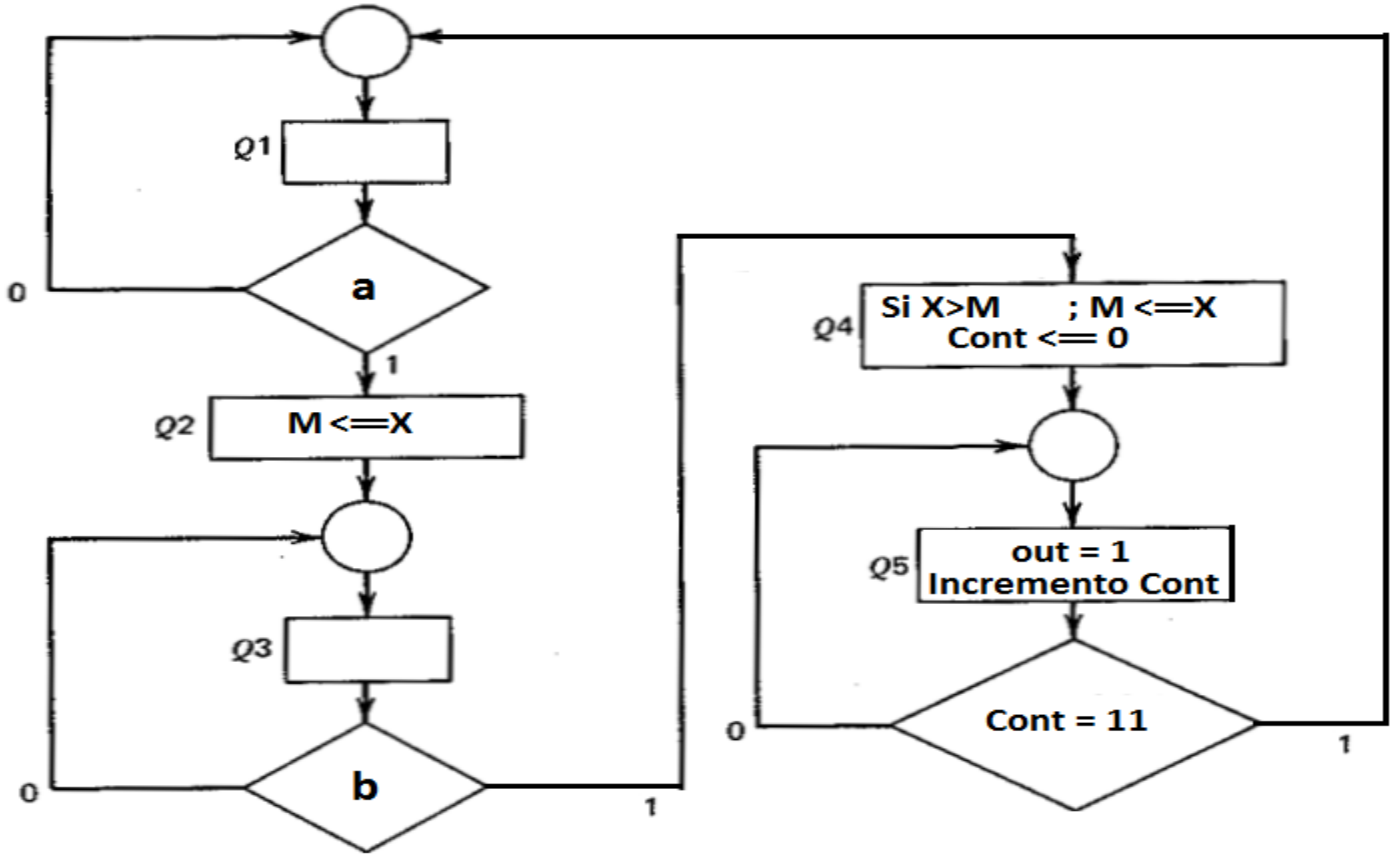


- Comparador de 2 números binarios A y B
  - Primero '**a**' pasa de 0 a 1 ➔ A[] válido en '**X[]**'
  - Tiempo después '**b**' pasa de 0 a 1 ➔ B[] válido en '**X[]**'
  - Al recibir B[], la salida '**out**' = 1 durante los 4 Tck y '**Z[]**' = Max(A[],B[]).
  - El resto del tiempo '**out**' = 0 y '**Z[]**' no está determinado.
  - Luego de los 4 Tck, se vuelve al estado inicial. Se supone que '**a**' y '**b**' ya bajaron.



Diseño RTL

Comparador: *(continuación)*



Diseño RTL

Secuencia RTL

```
Module: Comparador
Input: X[8], a, b
Output: Z[8], out
Memory: M[8]; Cont[2]

 1 - ➔(a, /a)/ (2, 1)
 2 - M[] ← X[]
 3 - ➔(b, /b)/ (4, 3)
 4 - M[] * Mayor(X[],M[]) ← X[]
     Cont[] ← 0
 5 - out = 1
     Z[]=M[]
     Cont[] ← INC( Cont[])
     ➔(Cont[0].Cont[1], /(Cont[0].Cont[1]))/ (1, 5)
ENDSEQUENCE
CONTROLRESET(1)
END
```

Bloque combinatorio
Mayor = 1 si X[]>M[]

Diseño RTL

Secuencia RTL (versión mejorada)

```
Module: Comparador
Input: X[8], a, b
Output: Z[8], out
Memory: M[8]; Cont[2]

  1 - M[] ← X[]          ;Cuando a=1, cargo en M[]el X[] correcto.
      →(a, /a)/ (2, 1)
  2 - M[] * (Mayor(X[],M[]) and b) ← X[]
      →(b, /b)/ (3, 2)
      Cont[] ← 0
  3 - out = 1
      Cont[] ← INC( Cont[])
      →(fin, /fin)/ (1, 3)
ENDSEQUENCE
CONTROLRESET(1)
Z[]=M[]       ; Conectado en todos los pasos, aunque solo me importa en el
                paso 3. Circuito más sencillo.
fin = Cont[0].Cont[1]  ; defino una señal
END
```

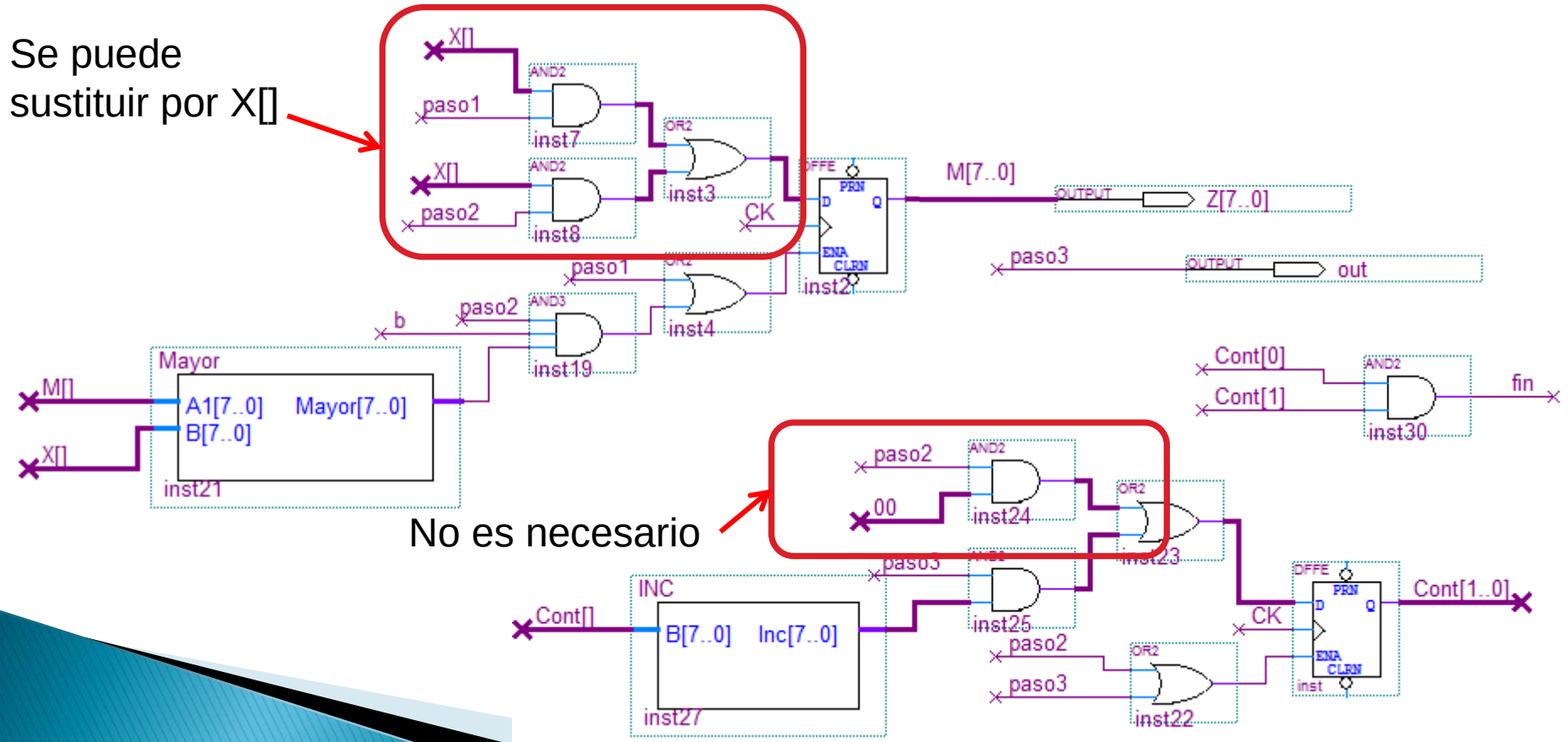
Dada la secuencia: → obtenemos el circuito “único”.
→ podemos simular

Diseño RTL

Bloque de datos (2 memorias, 2 salidas y ……)

- DEBE haber correspondencia “exacta”: secuencia ←→ circuito.
- Solo se pueden simplificar algunas lógicas.

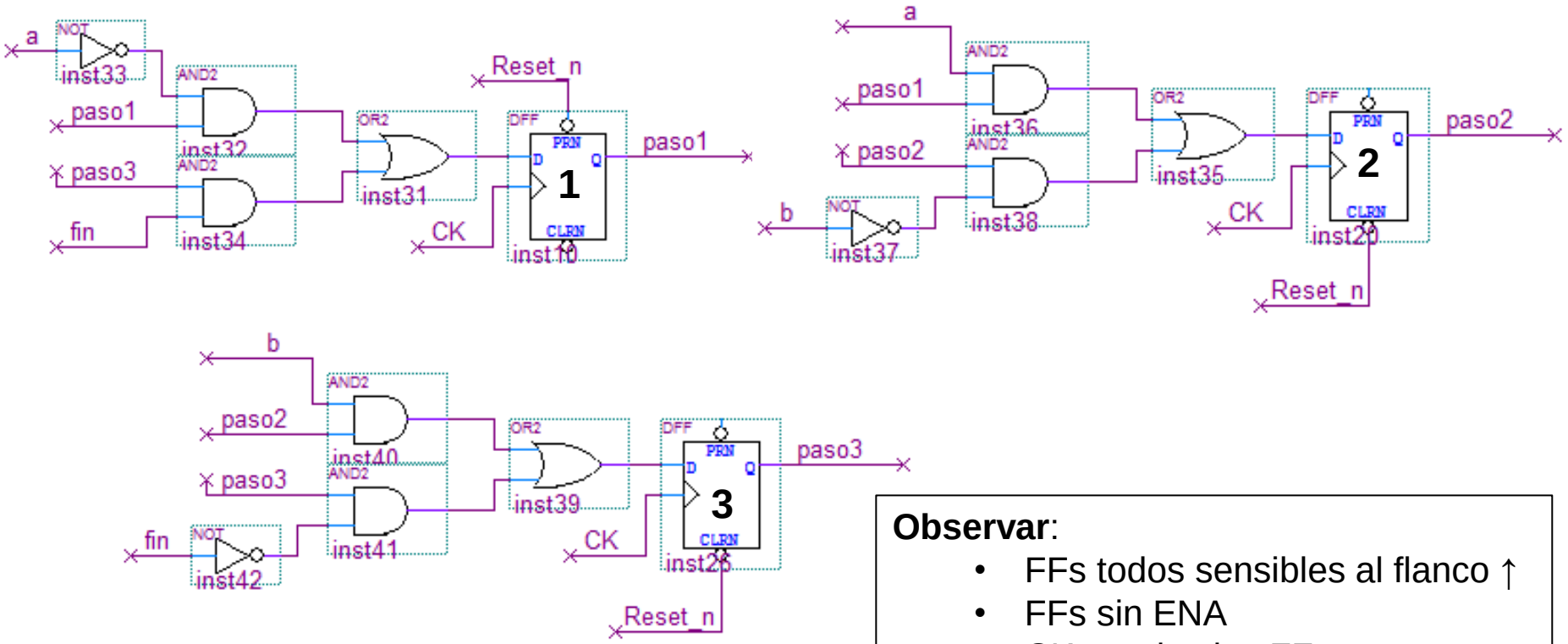
Se puede sustituir por X[]



Diseño RTL

Bloque de control

- 1 FF por paso: son 3 FF

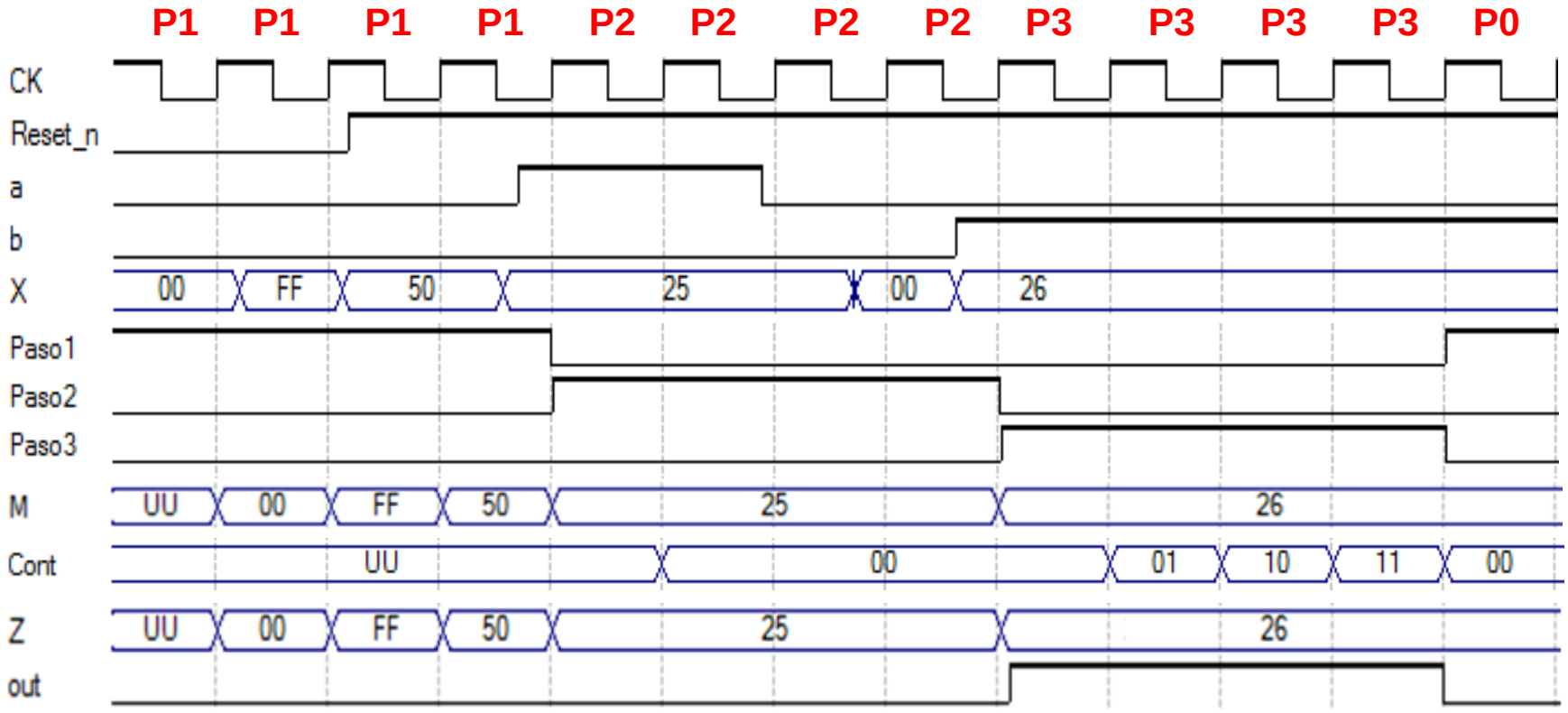


Observar:

- FFs todos sensibles al flanco ↑
- FFs sin ENA
- CK a todos los FFs
- ControlReset(1) : Reset_n

Diseño RTL

Simulacion



- Los pasos cambian un tiempo tp luego del flanco de reloj.
- Las memorias (M y Cont) se cargan al final del paso, luego del flanco de reloj

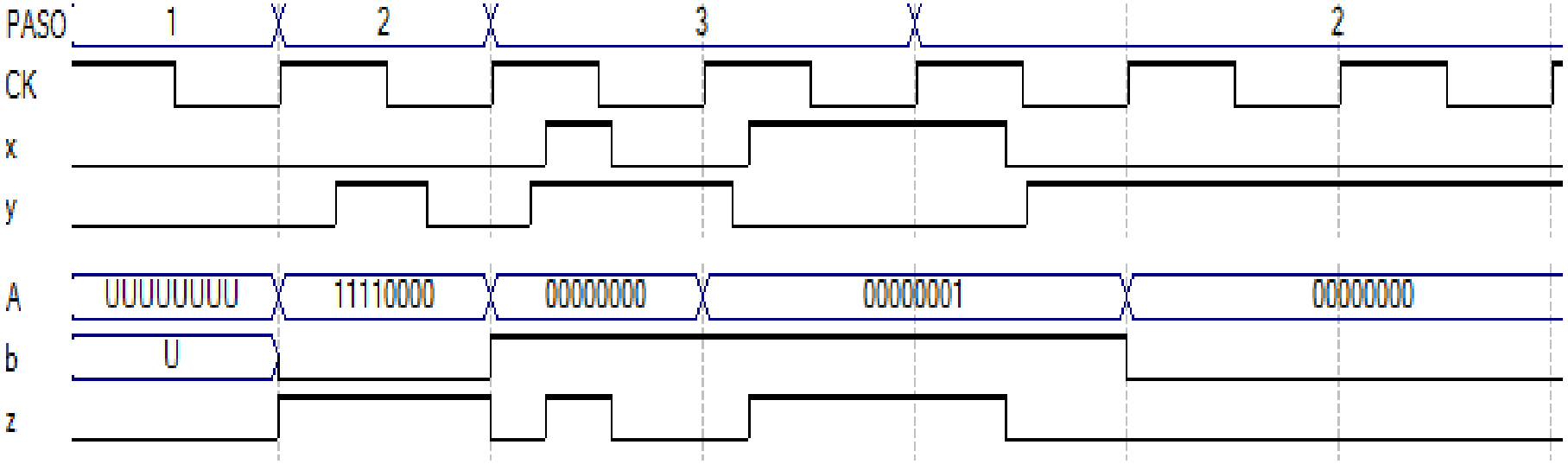
Ejemplo RTL

Secuencia RTL

```
Module:
Ejemplo
Input: x , y
Output: z
Memory: A[8], b

 1 – A[] ← 1111 0000
     b ← 0

 2 – A[] ← 0
     b ← A[7]
     z = x OR A[6]
     ➔(y, /y)/ (2, 3)

 3 – A[0] * y ← b . /A[0]
     z = b . x
     ➔(y, /y)/ (3, 2)

ENDSEQUENCE
CONTROLRESET(1)
END
```

Ejemplo RTL

Simulación

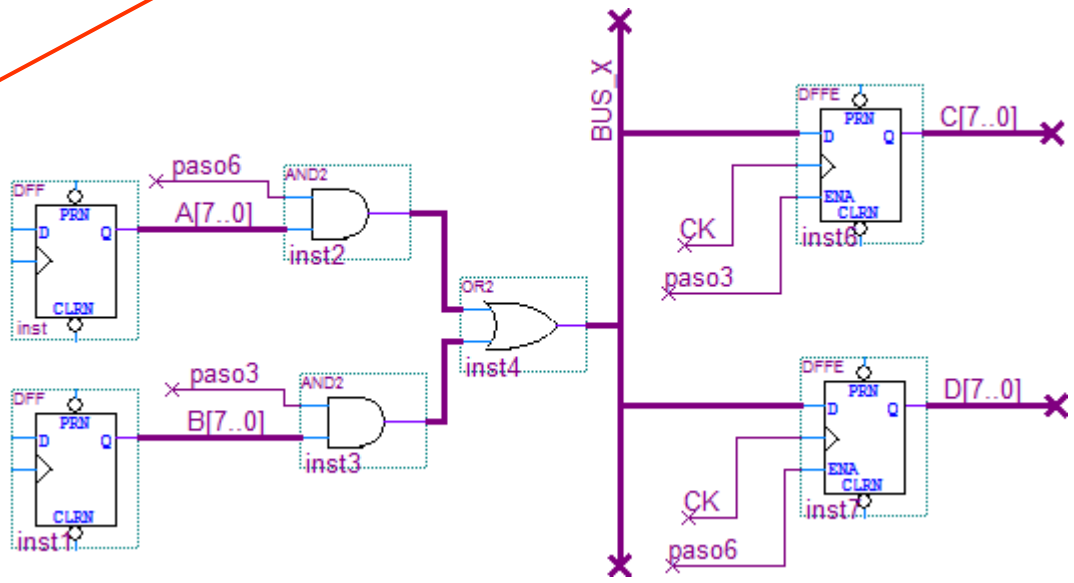


RTL - BUSES

Sintaxis:

```
Module: Buses
Input: ……
Output: ……
Memory: A[8]; B[8], C[8], D[8] …..
BUSES: BUS_X[8]

 1 –
 2 –
 3 – BUS_X[] = B[]
     C[] ← BUS_X[]
 4 –
 5 –
 6 – BUS_X[] = A[]
     D[] ← BUS_X[]
```

Siempre se declaran de a 2



RTL – A tener en cuenta

Transferencia	Conexión lógica
B ← F	Z = G
AL FINAL DEL PASO	DURANTE EL PASO
Un cambio en F en la mitad del paso no afecta a B	Un cambio en G en la mitad del paso se ve reflejado en Z.
El Valor de F solo importa en el flanco de reloj, al final del paso	El valor de G importa durante TODO el paso.
B siempre cambia sincronizado con el reloj	Z puede cambiar en cualquier instante
Circuito con FFs (registro)	Circuito es lógica pura, **SIN FFs**

RTL – A tener en cuenta

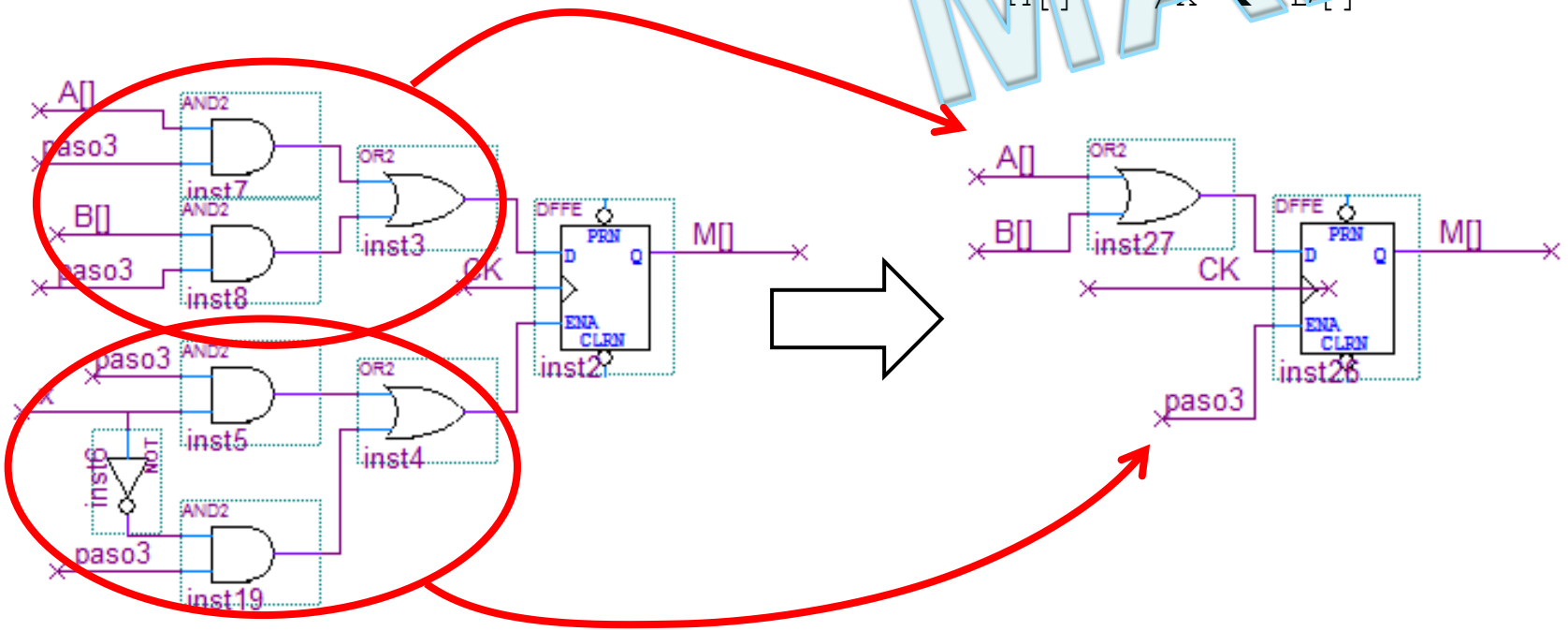
¿Como escribo la secuencia? (supongo estoy en paso 3)

Si x=1 entonces M[] ← A[]
sino M[] ← B[]

Solución 1:

```
3 – M[] * x ← A[]
    M[] * /x ← B[]
```

MAL



RTL – A tener en cuenta

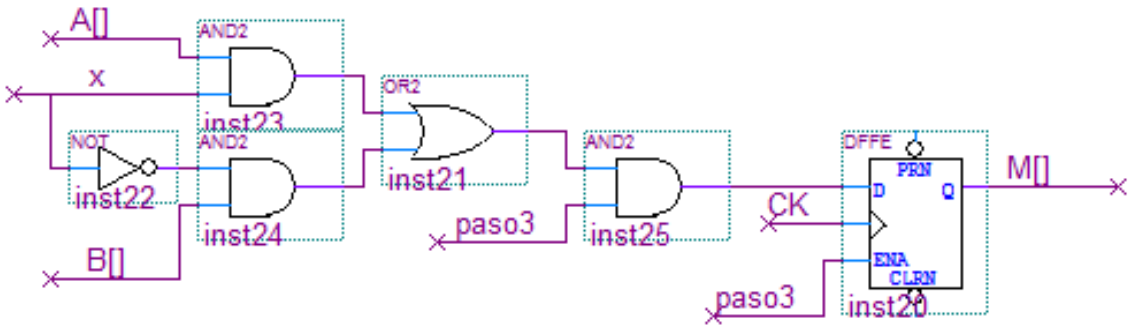
¿Como escribo la secuencia? (supongo estoy en paso 3)

Si x=1 entonces M[] ← A[]
sino M[] ← B[]

⇨

Solución 2:

```
3 – M[] ← x.A[] OR /x.B[]
```

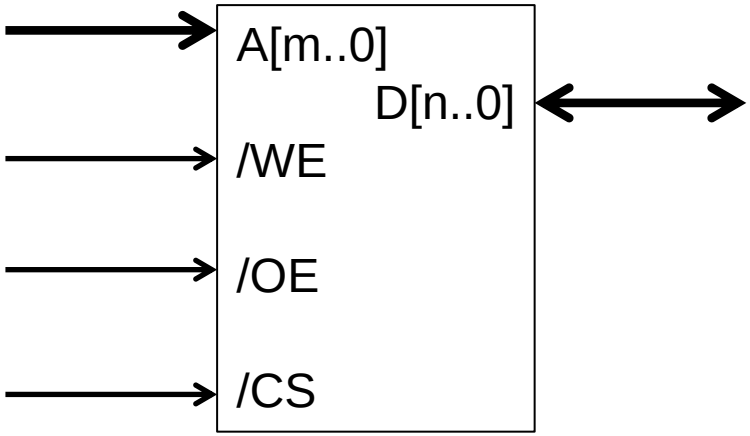


RAM – Memoria de Lectura Escritura

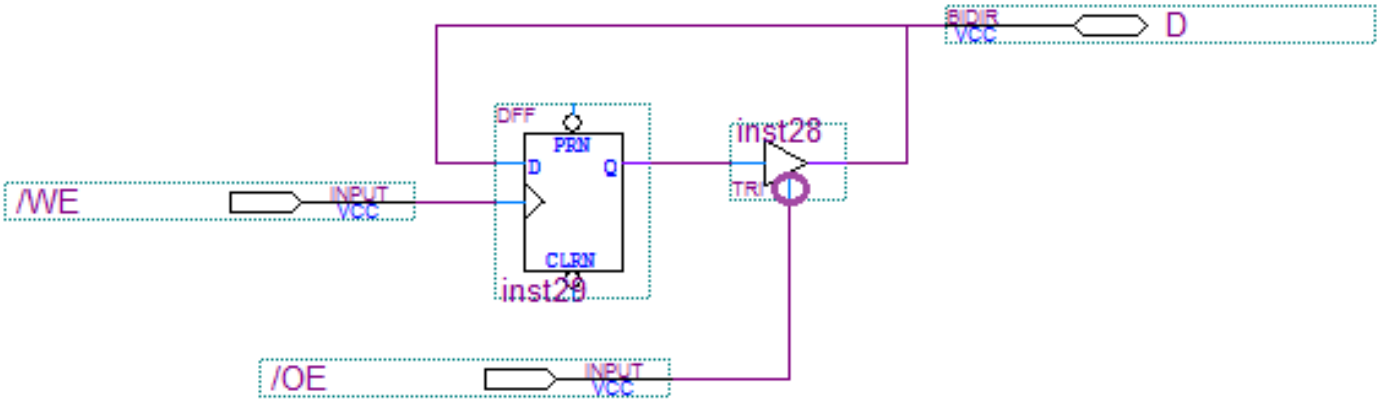
A[]: Bus de direcciones (m+1 bits)
D[]: Bus de datos (n+1 bits)

/WE: Write Enable
/OE: Output Enable
/CS: Chip Select

} En general, activas por nivel bajo

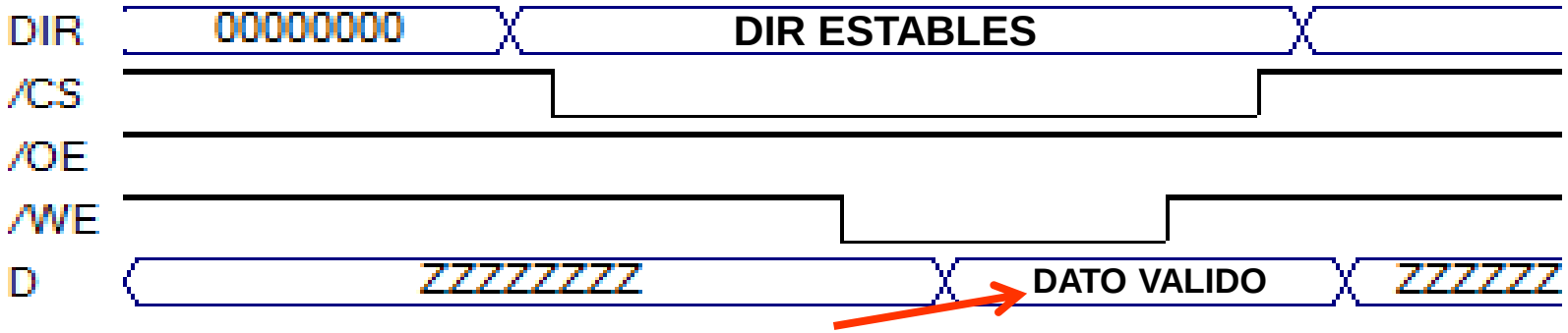


/CS: si está inactivo pone los datos en 3er estado.
/OE: si está inactivo pone los datos en 3er estado



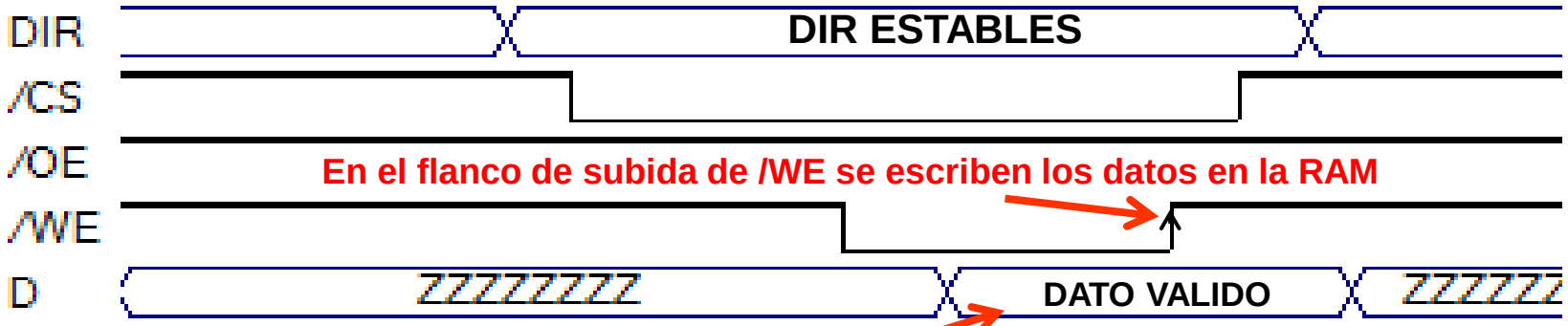
RAM – Memoria de Lectura Escritura

Ciclo de lectura



Datos salientes. La memoria pone los datos en el bus de dato.

Ciclo de escritura



Datos entrantes. Un dispositivo externo pone los datos en el bus de datos