

Programación en VHDL/Arquitectura

< Programación en VHDL

Sumario

Descripción de flujo de datos

Sentencias Concurrentes

WHEN ... ELSE

WITH ... SELECT ... WHEN

BLOCK

Descripción de comportamiento

PROCESS

Variables y Señales

Sentencias secuenciales

IF ... THEN ... ELSE

CASE

LOOP

NEXT Y EXIT

ASSERT

WAIT

Descripción estructural

Definición de componentes

Referencia de componentes

Como se ha dicho en el capítulo anterior, la arquitectura es lo que define cómo se comporta un circuito. En el primer capítulo también se mostraron varias arquitecturas en las que se describía un multiplexor. La primera línea de cada una era

```
ARCHITECTURE mux_comportamiento OF mux IS
  ARCHITECTURE mux_rtl OF mux IS
  ARCHITECTURE mux_structural OF mux IS
```

Los nombres de cada arquitectura son *mux_comportamiento*, *mux_rtl* y *mux_structural* respectivamente. Todas están asociadas a la entidad *mux*. El nombre de la arquitectura se usará para indicar qué arquitectura se debe usar en caso que haya varias para una misma entidad.

Después de esa línea pueden aparecer varias instrucciones para indicar la declaración de señales, componentes, funciones, etc.. Estas señales son internas, es decir, a ellas no se puede acceder desde la entidad, por los que los circuitos de nivel superior no podrían acceder a ellas. En un símil con un microprocesador, estas señales podrían ser las líneas que comunican la unidad central con la ALU, a las que no se puede acceder directamente desde el exterior del microprocesador. Obsérvese que en este caso no se indica si son entradas o salidas, puesto que al ser internas pueden ser leídas o escritas sin ningún problema. En esta parte de la arquitectura también pueden aparecer otros elementos, como pueden ser las constantes. Lo siguiente es la palabra clave **BEGIN**, que da paso a la descripción del circuito, mediante una serie de sentencias. Por lo tanto, la sintaxis de una arquitectura sería:

```
ARCHITECTURE nombre OF nombre_entidad IS
  (declaraciones)
  BEGIN
    [sentencias concurrentes]
  END [ARCHITECTURE] [nombre];
```

Un ejemplo de una arquitectura podría ser la siguiente.

```
ARCHITECTURE mux_rtl OF mux IS
  SIGNAL int1, int2, int3 : BIT;
BEGIN
  int1 <= NOT control;
  int2 <= entrada1 AND int1;
  int3 <= entrada2 AND S;
  salida <= int2 OR int3;
END mux_rtl;
```

La descripción puede ser de tres tipos:

1. Descripción de flujo de datos
2. Descripción de comportamiento
3. Descripción estructural

Descripción de flujo de datos

A la hora de plantearse crear un programa en VHDL no hay que pensar como si fuera un programa típico para ordenador. No hay que olvidar que en VHDL hay que describir un hardware, algo que no se hace en un programa para ordenador. Un circuito electrónico puede tener muchos elementos que estén ejecutando acciones a la vez, por ejemplo en un circuito puede tener una entrada que se aplique a dos puertas lógicas y de cada una obtener una salida, en este caso tendría dos caminos en los que se ejecutarían acciones (operaciones lógicas (AND,OR,NOT,IF), de unión, de intersección y de complemento) de forma paralela. Esto es lo que se llama **concurrentia**.

VHDL es un lenguaje concurrente, como consecuencia no se seguirá el orden en que están escritas las instrucciones a la hora de ejecutar el código. De hecho, si hay dos instrucciones, no tiene porqué ejecutarse una antes que otra, pueden ejecutarse a la vez.

Sentencias Concurrentes

La instrucción básica de la ejecución concurrente es la asignación entre señales a través del símbolo `<=`. Para facilitar la asignación de las señales VHDL incluye elementos de alto nivel como son instrucciones condicionales, de selección, etc, que verán a continuación.

WHEN ... ELSE

Sentencia de selección múltiple. En hardware es necesario incluir todas las opciones posibles. En este caso es obligatorio siempre acabar la expresión con un **ELSE**.

```
<señal> <asignación1> WHEN <condición1> ELSE
  <asignación2> WHEN <condición2> ELSE
  <asignación3> ...
  <asignaciónNm>;
```

Un posible ejemplo de este tipo de sentencias podría ser la siguiente.

```
$ <= "00" WHEN a = b ELSE
  "01" WHEN a > b ELSE
  "11";
```

-1 ALGEBRA DE BOOLE
 --Miniterminos y maxiterminos
 --Teorema de Morgan
 --Postulados y teoremas del álgebra de bool

-2 FUNCIONES LOGICAS Y SU MINIMIZACION
 --Función canónica y expansión a la forma canónica (conceptual)
 --Minimización y diagrama de Karnaugh
 --Representación de diagramas k y su minimización

-3 SISTEMAS DE NUMERACION Y CODIGOS
 --Sistemas binario, octal, hexadecimal (pasajes de un sistema al otro)
 --Códigos: BCD(decimal), X33, gray (reconocerlos)
 --Código Hamming (método matricial)
 --generador de código Hamming (método matricial)
 --Decodificador y corrección de código Hamming (método matricial)
 --VHDL código binario a gray

-4 ARITMETICA BINARIA
 --Complemento a 2
 --Suma y resta binaria, suma BCD (conceptos, reglas operativas, tablas de verdad y circuitos representativos)
 --Circuito sumandos 4 bits
 --Circuito sumador/restador (sin sacar el modulo del resultado)
 --Circuito sumador/restador (con modulo del resultado)
 FALTA algún problema con comparador + circuito que sea corto y facil a modo de ejemplo
 --VHDL sumador 16 bits
 --VHDL comparador 4 bits
 --VHDL atm 4 bits

-5 DECODIFICADORES/CODIFICADORES-MULTIPLEXORES/DEMUTIPLEXORES
 --MULTIPLEXOR (como generador de funciones)
 --Cuando la cantidad de combinaciones es menor a la del multiplexor
 --Cuando la cantidad de combinaciones es mayor a la del multiplexor
 --Con varios multiplexores (expansión de combinaciones - para diferentes combinaciones en las salidas)
 --Con un multiplexor pero usando variables en la entrada del multiplexor (multiplexor con función de $N+1$ variables)
 --VHDL para un mux 4:1 con salida en tercer estado (z:alta impedancia cuando esta inhibido)
 --Usando when-else (sin proces)

--Usando if-then-else-if-else (sin proces)

--VHDL ejercicio 10
 --DECODIFICADOR como generador de funciones (el decodificador tiene mas salidas que entradas)
 --Decodificador de errores para Hamming 9 bits de info con un decodificador de 3 a 8
 --VHDL decodificador 2:4 con when-else y case-when (con proces)

--VHDL decodificador BCD a siete segmentos wht-select (sin proces)

--VHDL decodificador HEX a siete segmentos case-when (con proces)

--CODIFICADORES (el codificador tiene mas entradas que salidas)
 --VHDL codificador 4:2 con prioridad usando when-else (sin proces)

-6 TECNOLOGIA
 FAUTA EJERCICIO CCON DISPLAY
 --Compuertas echas con CMOS/PMOS
 --Conceptos
 --Tiempos de transición ($t_d, t_r, t_{s, tf}$)
 --Tiempos de propagación (t_{plh}, t_{phl})
 --Margen de ruido
 --Fan-out (factor de carga en la entrada)
 --PLA, PAL (CP1D) y LUT (FPGA)

-7 FLIP FLOP/CONTADORES/MAQUINAS DE ESTADO FINITO (FSM-Finite State Machine)
 --FLIP FLOP

--SR asincrónico (tabla verdad)
 --SR sincrónico (tabla verdad + diagrama de tiempos)
 --Latch D (tabla verdad + diagrama de tiempos)
 --JK (tabla verdad + tabla de estados + diagrama de estados)
 --D (tabla verdad + tabla de estados + diagrama de estados)
 --VHDL: JK con reset asincrónico/sincrónico, activado por flanco descendente
 --VHDL: D con reset asincrónico, activado por flanco descendente

-CONTADORES
 --Contadores asincrónicos: Ascendente y descendente (diagrama de cada uno)
 --Contador sincrónico:
 ---contador Johnson (diagrama)
 ---contador en anillo (diagrama)
 ---Diagramas de Bruijn (p/ 16, 8 y 4 estados)
 --Registro de desplazamiento:
 ---Contador FF-JK octal de numeros pares e impares segun control X (0 par/1 impar)
 ---Contador FF-D de código Gray 4 bits con salida auxiliar que presente código BCD (1 digito)
 --VHDL: Contador BCD (modulo 10) de dos dígitos

--REGISTRO DE DESPLAZAMIENTO (SHIFT REGISTER)
 ---Diagramas de Bruijn (p/ 16, 8 y 4 estados)
 --Registro de desplazamiento:
 ---Entrada serie - salida paralelo
 ---Entrada paralelo - salida serie
 --VHDL: registro de desplazamiento entrada paralelo - salida serie
 --Registros de desplazamiento como contador
 --Registros de desplazamiento como contador modulo 12 con lógica de decodificación a decimal (BCD-para que sea BCD y modulo 12 al desbordar después de 9 necesita un segundo dígito)
 --Contador modulo 7 cuando m=1 y modulo 5 cuando m=0, utilizando realimentación XOR para implementar las anteriores especificaciones
 --Registro de desplazamiento como generador de secuencias
 --Generador de secuencia binaria 0-1-0-0-1-0-1-1

-CIRCUITOS SECUENCIALES
 VER SI VA ASINCRONOS-Asíncronos (se implementa con lógica combinatorial)
 VER SI VA ASINCRONOS-Elaboración del diagrama de flujo a partir del circuito o de la ecuación de excitación
 VER SI VA ASINCRONOS-Elaboración de circuito a partir del planteo o el diagrama de flujo
 -Síncronos (se implementa con lógica secuencial)
 --Detector de secuencias Mealy y Moore - conceptual
 --Máquinas de estados Mealy y Moore - conceptual
 --Detector de secuencias 101, 1011, 111 o 1101, 000 o 111 tanto para Moore como para Mealy
 --Automatas: puerta de cristal ejercicio 3 por mealy y moore
 --VHDL FSM (Finite state machine - máquina de estado finito): ejercicio 3 de automatas por Mealy y Moore usando las plantillas de VHDL
 --Contador moore (carpeta luigi)
 --Memorias y circuitos de tiempo

Siempre es obligatorio asignar algo, aunque es posible no realizar acción alguna, para ello se utiliza la palabra reservada **UNAFFECTED**. De esta forma se asignará el mismo valor que tenía la señal.

```
s1 <= d1 WHEN control = '1' ELSE UNAFFECTED;
s2 <= d2 WHEN control = '1' ELSE s2;
```

Las dos sentencias anteriores parecen iguales, pero en la segunda se produce una transacción, aspecto que en la primera no sucede.

WITH ... SELECT ... WHEN

Es similar a las sentencias CASE o SWITCH de C. La asignación se hace según el contenido de un objeto o resultado de cierta expresión.

```
WITH <señal1> SELECT
<señal2> <= <asignación1> WHEN <estado_señal1>,
<asignación2> WHEN <estado_señal2>,
...
<asignaciónN> WHEN OTHERS;
```

Un ejemplo de esta sentencia es la siguiente.

```
WITH estado SELECT
señal_foro <= "rojo",
"verde",
"amarillo" WHEN "10",
"roto"
WHEN OTHERS;
```

La cláusula WHEN OTHERS especifica todos los demás valores que no han sido contemplados. También es posible utilizar la opción que se contempló en el caso anterior (UNAFFECTED).

BLOCK

En ocasiones interesa agrupar un conjunto de sentencias en bloques. Estos bloques permiten dividir el sistema en módulos, estos módulos pueden estar compuestos de otros módulos. La estructura general es la siguiente.

```
block_id: BLOCK(expresión de guardia)
BEGIN
declaraciones
sentencias concurrentes
END BLOCK block_id;
```

El nombre del bloque es opcional (block_id), al igual que la expresión de guardia. Un ejemplo de esto podría ser el siguiente.

```
latch: BLOCK(CLK='1')
BEGIN
q GUARDED d;
END BLOCK latch;
```

A continuación se verán las estructuras más comunes de la ejecución serie y sus características.

PROCESS

Un PROCESS, como se ha dicho antes, es una sentencia concurrente en el sentido de que todos los PROCESS y todas las demás sentencias concurrentes se ejecutarán sin un orden establecido. No obstante las sentencias que hay dentro del PROCESS se ejecutan de forma secuencial.

Por lo tanto se puede decir que una estructura secuencial va en el interior de un PROCESS.

La estructura genérica de esta sentencia es:

```
PROCESS [lista de sensibilidad]
[declaración de variables]
BEGIN
[ sentencias secuenciales ]
END PROCESS;
```

La lista de sensibilidad es una serie de señales que, al cambiar de valor, hacen que se ejecute el PROCESS.

Un ejemplo sería:

```
PROCESS(señal1, señal2)
...
END PROCESS;
```

El PROCESS anterior sólo se ejecutará cuando señal1 o señal2 cambien de valor.

Variables y Señales

Hay que distinguir las señales y las variables, las señales se declaran entre ARCHITECTURE y su correspondiente BEGIN mientras que las variables se declaran entre PROCESS y su BEGIN. Dentro de un PROCESS pueden usarse ambas, pero hay una diferencia importante entre ellas: las señales sólo se actualizan al terminar el proceso en el que se usan, mientras que las variables se actualizan instantáneamente, es decir, su valor cambia en el momento de la asignación.

Un ejemplo es:

```
ENTITY ejemplo IS
PORT (c: IN std_logic;
d: OUT std_logic);
END ENTITY;

ARCHITECTURE ejemplo_arch OF ejemplo IS
BEGIN
PROCESS(c)
VARIABLE z: std_logic;
BEGIN
a=c AND b; --asignación de señales: después de ejecutarse esta linea, "a" seguirá valiendo lo mismo, sólo se actualiza al acabar el PROCESS
z:= a OR c; --asignación de variables: en el momento de ejecutarse esta linea z valdrá "a" or "c"
el valor que tenía a cuando empezó el PROCESS)
END PROCESS;
END ARCHITECTURE;
```

Descripción de comportamiento

Como la programación concurrente no siempre es la mejor forma de describir ideas, VHDL incorpora la programación serie, la cual se define en bloques indicados con la sentencia PROCESS. En un mismo diseño puede haber varios bloques de este tipo, cada uno de estos bloques corresponderá a una instrucción concurrente. Es decir, internamente la ejecución de las instrucciones de los PROCESS es serie, pero entre los bloques es concurrente.

IF ... THEN ... ELSE

Sentencias secuenciales

Permite la ejecución de un bloque de código dependiendo de una o varias condiciones.

```
IF <condición> THEN
  [sentencias 1]
ELSIF <condición2> THEN
  [sentencias 2]
ELSE [sentencias N]
END IF;
```

Un ejemplo es:

```
IF (veloj='1' AND enable='1') THEN
  salida<entrada;
ELSIF (enable='1') THEN
  salida<tmp;
ELSE
  salida<='0';
END IF;
```

CASE

Es parecido al anterior porque también ejecuta un bloque de código condicionalmente, pero en esta ocasión se evalúa una expresión en vez de una condición. Se debe recordar que se deben tener en cuenta todos los casos, es decir, incluir como última opción la sentencia WHEN OTHERS.

```
CASE <expresión> IS
  WHEN <valor1> => [sentencias1]
  WHEN <valor2> => [sentencias2]
  WHEN <rango de valores> => [sentenciasN]
  WHEN OTHERS => [sentenciasM]
END CASE;
```

Un ejemplo es:

```
CASE a IS
  WHEN 0 => b:=0;
  WHEN 1 to 50 => b:=1;
  WHEN 99 to 51 => b:=2;
  WHEN OTHERS => b:=3;
END CASE;
```

LOOP

LOOP es la forma de hacer bucles en VHDL. Sería el equivalente a un FOR o WHILE de un lenguaje convencional.

Su estructura es:

```
[etiqueta:] [WHILE <condición> | FOR <condición>] Loop
  [sentencias]
  [exit;]
  [next;]
END Loop [etiqueta];
```

Un ejemplo de bucles anidados es:

```
bucle1: LOOP
  a:=A;1;
  b:=20;
  bucle2: LOOP
    IF b < (a+b) THEN
      EXIT bucle2;
    END IF;
    b:=b+a;
    END LOOP bucle2;
    EXIT bucle1 WHEN a>10;
  END LOOP bucle1;
```

Otro ejemplo, este con FOR es:

```
bucle1: FOR a IN 1 TO 10 LOOP
  bucle2: LOOP
    IF b<(a*a) THEN
      EXIT bucle2;
    END IF;
    b:=b-a;
    END LOOP bucle2;
  END LOOP bucle1;
```

Otro más con WHILE

```
cuenta := 10;
bucle1: WHILE cuenta >= 0 LOOP
  cuenta := cuenta + 1;
  bucle2: LOOP
    IF b<(a*a) THEN
      EXIT bucle2;
    END IF;
    b := b-a;
    END LOOP bucle2;
  END LOOP bucle1;
```

NEXT Y EXIT

NEXT permite detener la ejecución actual y seguir con la siguiente.

```
[id next;]
NEXT [id bucle] [WHEN condición];
```

Como se puede suponer, la sentencia EXIT hace que se salga del bucle superior al que se ejecuta.

```
[id exit;]
EXIT [id bucle] [WHEN condición];
```

Se puede ver su uso en los ejemplos del apartado anterior.

ASSERT

Se usa para verificar una condición y, en caso de que proceda, dar un aviso.

La sintaxis es:

```

ASSERT <condición>
[REPORT <expresión>]
[SEVERITY <expresión>];

```

Este comando se estudiará en el subcapítulo de notificaciones, en la sección de bancos de prueba. Puesto que el uso de este comando se realiza únicamente en la simulación de circuitos.

WAIT

La ejecución de un bloque PROCESS se realiza de forma continuada, como si de un bucle infinito se tratara (se ejecutan todas las sentencias y se vuelven a repetir). Esto no tiene mucho sentido, puesto que continuamente se ejecutaría lo mismo una y otra vez, sería interesante poder parar la ejecución. Una forma de hacerlo es mediante las listas de sensibilidad, las cuales se han visto anteriormente, aunque existe otra forma de hacerlo mediante la sentencia WAIT, pero es algo más complejo.

```
WAIT ON lista_sensible UNTIL condición FOR timeout;
```

La lista_sensible es un conjunto de señales separadas por comas. La condición es una sentencia que activará de nuevo la ejecución. El timeout es el tiempo durante el cual la ejecución está detenida.

No es necesario utilizar las tres opciones, en caso de hacerlo la primera condición que se cumpla volverá a activar la ejecución.

```

WAIT ON pulso;
WAIT UNTIL counter = 5;
WAIT FOR 10 ns;
WAIT ON pulso, sensor UNTIL counter = 5;
WAIT ON pulso UNTIL counter = 5 FOR 10 ns;

```

Si se utiliza una lista de sensibilidad no es posible utilizar la sentencia WAIT, sin embargo si es posible utilizar varias sentencias WAIT cuando esta actúa como condición de activación. Este comando se estudiará en el subcapítulo de retrasos, en la sección de bancos de prueba.

Descripción estructural

Las dos descripciones anteriores son las más utilizadas por los diseñadores, ya que son más cercanas al pensamiento humano. Aunque existe otro tipo de descripción, que permite la realización de diseños jerárquicos, VHDL dispone de diferentes mecanismos para la descripción estructural.

Definición de componentes

En VHDL, es posible declarar componentes dentro de un diseño mediante la palabra COMPONENT. Un componente se corresponde con una entidad que ha sido declarada en otro módulo del diseño, o incluso en alguna biblioteca, la declaración de este elemento se realizará en la parte declarativa de la arquitectura del módulo que se está desarrollando. La sintaxis para declarar un componente es muy parecida a la de una entidad.

```

COMPONENT nombre [IS]
[GENERIC (lista_parametros);]
[PORT (lista_de_puertos);]
END COMPONENT nombre;

```

Si se dispone de un compilador de VHDL'93 no será necesario incluir en los diseño la parte declarativa de los componentes, es decir se pasaría a referenciarlos de forma directa. Un ejemplo de un componente podría ser el siguiente.

```

COMPONENT mux IS
  GENERIC(

```

```

    C_AMIDTH : integer;
    C_DMIDTH : integer;
    PORT (
        control : IN bit;
        entrada1 : IN bit;
        entrada2 : IN bit;
        salida : OUT bit
    );
  END COMPONENT mux;

```

Referencia de componentes

La referencia de componentes consiste en copiar en la arquitectura aquél componente que se quiera utilizar, tantas veces como sea necesario para construir el diseño. Para ello, la sintaxis que presenta la instanciación de un componente es la siguiente.

```

ref_id:
[COMPONENT] id_componente | ENTITY id_entidad [(id_arquitectura)] | CONFIGURATION id_configuración
[GENERIC MAP (parametros)]
[PORT MAP (puertos)];

```

Un ejemplo de referenciación del componente anterior sería.

```

mux_1:
COMPONENT mux
GENERIC MAP (
    C_AMIDTH => C_AMIDTH,
    C_DMIDTH => C_DMIDTH
)
PORT MAP (
    control => ctrl,
    entrada1 => e1,
    entrada2 => e2,
    salida => sal
);

```

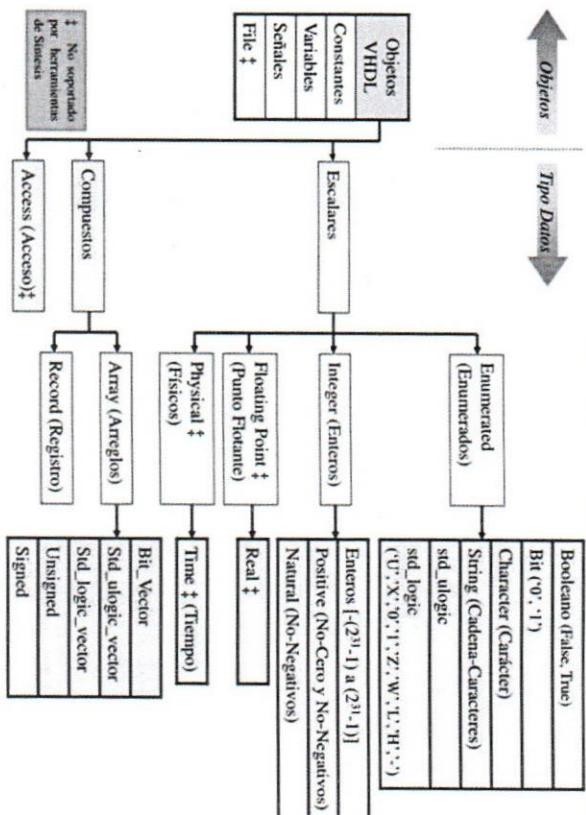
Obtenido de [«https://es.wikibooks.org/w/index.php?title=Programaci%C3%B3n_en_VHDL/Arquitectura&oldid=329005»](https://es.wikibooks.org/w/index.php?title=Programaci%C3%B3n_en_VHDL/Arquitectura&oldid=329005)

Esta página se editó por última vez el 5 may 2017 a las 18:42.

El texto está disponible bajo la Licencia Creative Commons Atribución-CompartirIgual 3.0; pueden aplicarse términos adicionales. Véase Términos de uso para más detalles.

Lista de palabras reservadas en VHDL

abs	downto	library	postponed	subtype
access	else	linkage	procedure	then
after	elsif	literal	process	to
alias	end	loop	pure	transport
all	entity	map	range	type
and	exit	mod	record	unaffected
architecture	file	nand	register	units
array	for	new	reject	until
assert	function	next	rem	use
attribute	generate	nor	report	variable
begin	generic	not	return	when
block	group	null	rod	while
body	guarded	of	ror	
buffer	if	on	select	
bus	impure	open	severity	
case	in	or	shared	
component	inertial	others	signal	
configuration	input	out	sia	
constant	is	package	set	
disconnect	label	port	std	



A	B	S	Product	Sum
0	0	1	00	N_1
0	1	0	01	N_2
1	0	0	10	N_3
1	1	0	11	N_4

$$\overline{AB} + AB = (A+B)(\overline{A+B}) = \overline{m}(0,2) = \overline{M}(1,0)$$

función lógica multiplicativa

$$F \rightarrow F \text{ invierte} \rightarrow F \text{ en minforn}$$

AUD/OR	$\rightarrow \overline{\overline{AB}} + \overline{\overline{AB}}$	$\leftarrow (\overline{F}) \text{ en minforn}$
NAND/NND	$\rightarrow \overline{\overline{AB}} \cdot \overline{\overline{AB}}$	
OR/NAND	$\rightarrow (A+B) \cdot (A+\overline{B})$	
NOR/OR	$\rightarrow (\overline{A+B}) + (\overline{A+B})$	
AND/NOR	$\rightarrow (A+B) + (A,\overline{B})$	$\leftarrow \overline{F}$ <small>función lógica multiplicativa</small>
NAND/AUD	$\rightarrow \overline{AB} \cdot A\overline{B}$	
OR/AND	$\rightarrow (\overline{A+B}) \cdot (A+B)$	
NOR/NOR	$\rightarrow (\overline{A+B}) + (\overline{A+B})$	

Simetría en combinaciones de entradas y salidas

$$\begin{aligned} A \cdot A &= A \rightarrow [] \cdot [] = [] \\ A+A &= A \rightarrow [] + [] = [] \end{aligned}$$

HOJA FICHA

$$\overline{A+B+C} = 1 \Rightarrow A \cdot (B+C) = 0$$

función lógica distributiva

$$\boxed{A+A = A}$$

ley de idempotencia

$$\boxed{A+\overline{A} = 1}$$

ley de doble inclusión

$$\boxed{A = \overline{A}}$$

ley de dualidad

$$\begin{aligned} N_p &= 1 \quad (\text{lleva el valor 1}) \\ N_s &= 0 \quad (\text{lleva el valor 0}) \end{aligned}$$

ley de dualidad de la negación

$$\begin{array}{l|l|l} A+I=1 & A \cdot O=0 \\ A+O=A & A \cdot I=A \\ A \cdot I=0 & A \cdot A=0 \\ A \cdot A=0 & A+A=A \end{array}$$

ley complementaria

ley distributiva

$$\begin{aligned} A(B+C) &= AB + AC \\ A+(B,C) &= (A+B) \cdot (A+C) \end{aligned}$$

ley asociativa

$$\begin{aligned} A(B,C) &= (A,B) \cdot C \\ A+(B,C) &= (A+B)+C \end{aligned}$$

ley de absorción

$$\begin{aligned} A \cdot (B+\overline{B}) &= AB + A\overline{B} \\ A \cdot A\overline{B} &= A\overline{B} = A \end{aligned}$$

ley de doble inclusión

$$\begin{aligned} A \cdot (B+\overline{B}) &= A \cdot B \\ A \overline{B} + A\overline{B} &= A \end{aligned}$$

ley de absorción

TP1

Para una XOR de 1 cantidad de unidades no salda en 1 cuando lo sume de suces de las restadas sea mayor.

$$\begin{aligned} A + \overline{AB} &= A \cdot \overline{B} \\ A \overline{B} + A\overline{B} &= A \end{aligned}$$

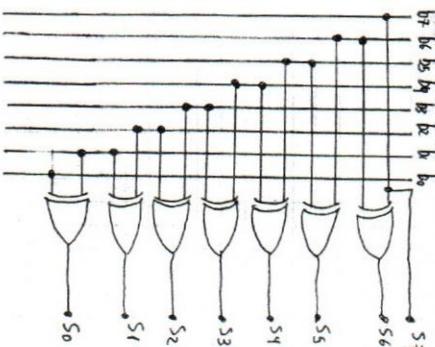
XOR

$$AB + A\overline{B} = (A+B)(A+\overline{B})$$

NOTA: es igual a cada uno o a cero. Tres situaciones:

TP3

③ Codificador de binario a Gray (implementación + VHDL)



Entrada Gray es

Port (bin : 111 std_logic_vector (7 downto 0),

gray : 001 std_logic_vector (7 downto 0)),

end gray;

Architectura código de Gray es

Begin

Process (bin)

For I in 0 to 6 loop

gray (I) <= (bin (I) XOR bin (I+1));

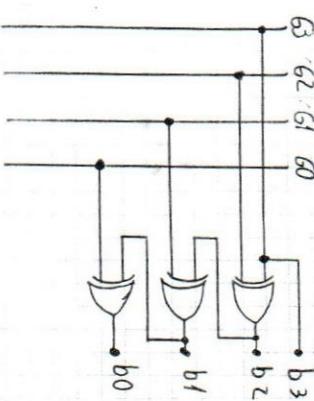
End loop;

gray (7) <= bin (7);

End process;

End design;

Codificador de Gray a binario y lista



Verificación

Descodificador x1

Entrada	Salida						
	C_4	C_2	G_1	C_3	C_6	G_3	C_5
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
3 \rightarrow 0	1	-	0	0	0	1	0
5 \rightarrow 1	0	-	0	0	1	0	0
6 \rightarrow 1	1	0	0	1	0	0	0
7 \rightarrow 1	1	1	0	0	0	0	0

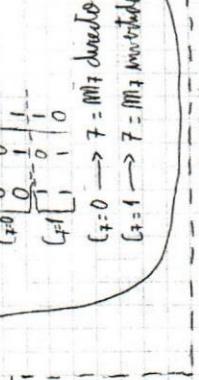
Verificación

$$\begin{array}{c} C_4 \\ C_2 \\ G_1 \\ C_3 \\ C_6 \\ G_3 \\ C_5 \end{array} \begin{array}{c} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{array} \begin{array}{c} T \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{array}$$

$$C_4 = 0 \rightarrow T = M_7 \text{ directo}$$

$$C_2 = 1 \rightarrow T = M_4 \text{ inverso}$$

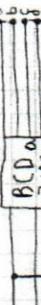
Palabra de Transmisión



$$T = M_4 \oplus M_3 \oplus M_5 \oplus K_4 \oplus K_2 \oplus K_1 \quad \text{LSB}$$

$$\text{Mejoría } K_1 \text{ y } K_2 \text{ directa}$$

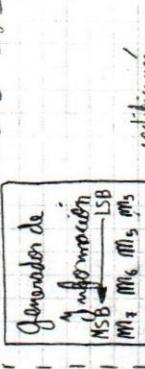
Verificación



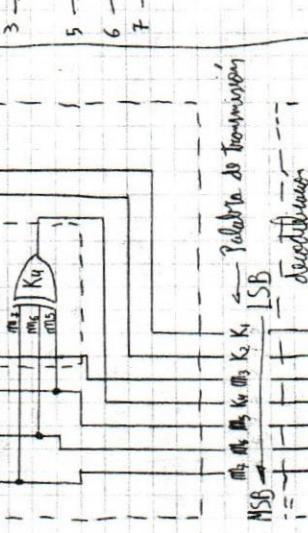
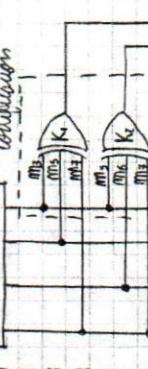
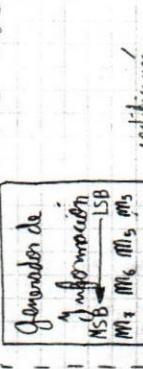
Verificación de verificación



Generador de código Hamming
de codificación



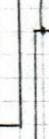
Generación de código Hamming



Verificación



Verificación



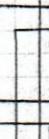
Verificación



Verificación



Verificación



Verificación



Palabra codificada (m)

$$\begin{array}{c} m_7 \quad m_6 \quad m_5 \quad m_4 \\ \hline m=0 \quad 1 \quad 0 \quad 1 \end{array} \quad \text{LSB} \quad \text{MSB=menor} \quad \text{MSB=mayor}$$

$$\begin{array}{c} m=0 \quad 1 \quad 0 \quad 1 \\ \hline \text{MSB} \quad \leftarrow \quad \text{LSB} \end{array}$$

Bit de paridad (K)

$$K > K+m+1$$

$$2^k - 1 \geq K$$

$$K-K-1 \geq m$$

$$M=4 \rightarrow K=3$$

$$T = 4+3 = 7 = T$$

Palabra de transmisión (T)

$$K \rightarrow 2^0 \rightarrow K_1, K_2, K_3, K_4, K_{16}, K_{32}, \dots$$

$$\begin{array}{c} \text{Mejoría } K_1 \text{ y } K_2 \text{ directa} \\ 0, \text{Mejoría } K_4 \text{ directa} \\ \hline T = M_7 \oplus M_6 \oplus M_5 \oplus K_4 \oplus K_2 \oplus K_1 \quad \text{LSB} \end{array}$$

$$\begin{array}{c} K_1 \quad K_2 \quad K_3 \quad K_4 \quad M_7 \quad M_6 \quad M_5 \quad M_4 \\ \hline 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \end{array}$$

$$H = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad \text{matriz de verificación}$$

$$T = M_7 \oplus M_6 \oplus M_5 \oplus K_4 \oplus M_3 \oplus M_2 \oplus K_1$$

$$K_1 = M_7 \oplus M_6 \oplus M_5 \oplus M_4$$

$$K_2 = M_6 \oplus M_5 \oplus M_4 \oplus M_3$$

$$K_3 = M_5 \oplus M_4 \oplus M_3 \oplus M_2$$

$$K_4 = M_4 \oplus M_3 \oplus M_2 \oplus M_1$$

$$T = 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1$$

Detector de errores
de código Hamming

9. Codificación

$$T = 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1$$

$$\overrightarrow{MSB} \quad \overleftarrow{LSB}$$

[H]. [I] = [E]

E: matrrix origen

Algunas
columnas
se cancelan

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} M_2 \\ M_4 \\ M_5 \\ M_7 \end{bmatrix} = \begin{bmatrix} K_1 \\ K_2 \\ K_3 \\ K_4 \end{bmatrix} \Rightarrow G_1 = K_1 \oplus M_3 \oplus M_5 \oplus M_7 \\ G_4 = K_4 \oplus M_5 \oplus M_6 \oplus M_7$$

cancelar

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

$$G_1 = 1 \oplus 0 \oplus 0 \oplus 0 = 0 \\ G_2 = 0 \oplus 1 \oplus 0 \oplus 0 = 0 \\ G_4 = 1 \oplus 0 \oplus 1 \oplus 0 = 0$$

$$\Rightarrow 0 \ 0 \ 0 \rightarrow \text{no hay error}$$

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

desaparece que M_6 se anula

$$T = M_2 \ M_3 \ M_5 \ K_4 \ M_3 \ K_2 \ K_1$$

$$T = 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ (\text{original})$$

↓

$$T = 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ (\text{despues de } b_6) \\ G_1 = 1 \oplus 1 \oplus 0 \oplus 0 = 0 \\ G_2 = 0 \oplus 1 \oplus 0 \oplus 0 = 1 \\ G_4 = 1 \oplus 0 \oplus 0 \oplus 0 = 1$$

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} M_2 \\ M_3 \\ M_5 \\ K_4 \\ M_3 \\ K_2 \\ K_1 \end{bmatrix} \Rightarrow \text{salida codificada}$$

• Modo 2 (modo de rotación)

$$\begin{bmatrix} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \leftarrow \text{polalma a codificar}$$

$$2^k - k - 1 \geq N^{pbh}$$

$$(N^{pbh} = 7; k = 4)$$

↓

$$T = 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ (\text{despues de } b_6) \\ G_1 = 1 \oplus 1 \oplus 0 \oplus 0 = 0 \\ G_2 = 0 \oplus 1 \oplus 0 \oplus 0 = 1 \\ G_4 = 1 \oplus 0 \oplus 0 \oplus 0 = 1$$

$$\begin{array}{c} b_1 \ b_2 \ b_3 \ b_4 \ b_5 \ b_6 \ b_7 \\ \hline 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{array} \leftarrow \text{salida a codificar}$$

$$K_1 = b_3 \oplus b_5 \oplus b_7 \oplus b_9 \oplus b_{11} = 0$$

$$K_2 = b_3 \oplus b_6 \oplus b_7 \oplus b_{10} \oplus b_{11} = 0$$

$$K_3 = b_5 \oplus b_6 \oplus b_7 = 0$$

$$K_4 = b_9 \oplus b_{10} \oplus b_{11} = 0$$

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} M_2 \\ M_3 \\ M_5 \\ K_4 \\ M_3 \\ K_2 \\ K_1 \end{bmatrix} \Rightarrow \text{salida codificada o a transmitir}$$

(Polalma codificada o a transmitir)

• Modo 1

$$b_1 \ b_2 \ b_3 \ b_4 \ b_5 \ b_6 \ b_7$$

$$1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \leftarrow \text{polalma a codificar}$$

$$K_1 = b_3 \oplus b_5 \oplus b_7 \oplus b_9 \oplus b_{11} = 0$$

$$K_2 = b_3 \oplus b_6 \oplus b_7 \oplus b_{10} \oplus b_{11} = 0$$

$$K_3 = b_5 \oplus b_6 \oplus b_7 = 0$$

$$K_4 = b_9 \oplus b_{10} \oplus b_{11} = 0$$

• Modo 2 (modo de rotación)

$$b_1 \ b_2 \ b_3 \ b_4 \ b_5 \ b_6 \ b_7$$

$$0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \leftarrow \text{polalma a codificar}$$

$$K_1 = b_3 \oplus b_5 \oplus b_7 \oplus b_9 \oplus b_{11} = 0$$

$$K_2 = b_3 \oplus b_6 \oplus b_7 \oplus b_{10} \oplus b_{11} = 0$$

$$K_3 = b_5 \oplus b_6 \oplus b_7 = 0$$

$$K_4 = b_9 \oplus b_{10} \oplus b_{11} = 0$$

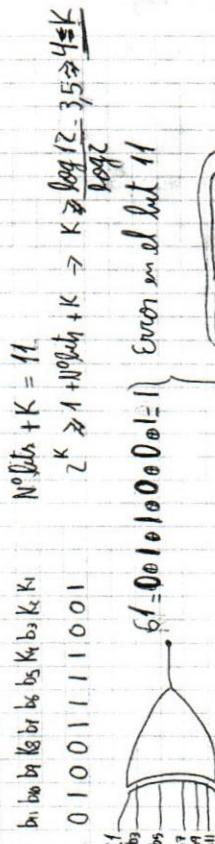
TP3

• Generador de código Hamming

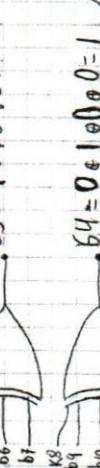
TPY

⑤ El decodificador de código Hamming

• Modo 1

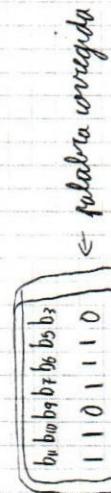


palabra corregida



• Modo 2 (método matricial)

$b_1, b_2, b_3, b_4 \mid K_1, K_2, K_3, K_4$	$N^{\text{bits}} + K = 11$
0100111001	$2^K \geq N^{\text{bits}} + K + 1 \rightarrow K \geq \frac{\log 12 - 3.5}{\log 2} \Rightarrow K=4$
64	$0 \rightarrow 0100000 = 1 = 64$
63	$0 \rightarrow 1010001 = 0 = 63$
62	$0 \rightarrow 0101000 = 1 = 62$
61	$0 \rightarrow 0010001 = 1 = 61$
60	error en el bit 11



$$\begin{array}{r}
 \begin{array}{c}
 \text{+ } 7F_6 \\
 \hline
 1000000012
 \end{array}
 & \leftarrow +7F_6 \text{ representado en C2} \\
 + & \\
 \begin{array}{c}
 1000000012 \\
 \hline
 1000000012
 \end{array}
 & \\
 \hline
 \begin{array}{c}
 \text{+ } 2z_6 \\
 \hline
 1010000010
 \end{array}
 & \leftarrow -2z_6 \rightarrow 1010000010 \\
 + & \\
 \begin{array}{c}
 1010000010 \\
 \hline
 1010000010
 \end{array}
 & \\
 \hline
 \begin{array}{c}
 \text{+ } 0z_6 \\
 \hline
 01011102
 \end{array}
 & \leftarrow 0z_6 \rightarrow 01011102 \\
 + & \\
 \begin{array}{c}
 01011102 \\
 \hline
 01011102
 \end{array}
 & \\
 \hline
 \begin{array}{c}
 \text{+ } 0_{16} \\
 \hline
 00000000
 \end{array}
 & \leftarrow 0_{16} \text{ representado en C2} \\
 + & \\
 \begin{array}{c}
 00000000 \\
 \hline
 00000000
 \end{array}
 & \\
 \hline
 \begin{array}{c}
 \text{+ } 5A_6 \\
 \hline
 11011010
 \end{array}
 & \leftarrow -5A_6 \text{ en C2}
 \end{array}$$

- ① Son siguientes números en orden binario a los expuestos en complemento a 2 bin 8 bits (1 bit más para el signo para el número).
- $+7F_6 \rightarrow 01111111$

La Bandera de Acarreo y la de Desborde en Aritmética Binaria

Operaciones aritméticas binarias, REGLAS suma y resta

Como hacer operaciones aritméticas de resto:

Considera los siguientes ejemplos:

1) Ejemplo: Complemento a DOS.

Decimal Binario
110 1101110
63 111111

a) colocar el resultado un cero más que el signo

cantidad de bits que el resultado:

binario decimal
1101110 -110
111111 -63

b) complementando dos al resultado

Comp. 1 100000
Comp. 2 1000001

c) sume el resultado más el resultado del paso b

binario decimal
111111 -63
1000001 -1

0110110 -64

d) si existe scáramento DESCARTAR y sume el resultado como

binario decimal
0110111 -65

nota este otro resultado que el resultado es positivo

binarios decimal
1010110 -10
1100001 -16

binario decimal
1110100 -17

binario decimal
0110111 -65

binario decimal
1010111 -10

binario decimal
1110100 -17

binario decimal
0110111 -65

binario decimal
1010111 -10

binario decimal
1110100 -17

binario decimal
0110111 -65

binario decimal
1010111 -10

binario decimal
1110100 -17

binario decimal
0110111 -65

binario decimal
1010111 -10

binario decimal
1110100 -17

binario decimal
0110111 -65

binario decimal
1010111 -10

binario decimal
1110100 -17

binario decimal
0110111 -65

binario decimal
1010111 -10

binario decimal
1110100 -17

binario decimal
0110111 -65

binario decimal
1010111 -10

binario decimal
1110100 -17

binario decimal
0110111 -65

binario decimal
1010111 -10

binario decimal
1110100 -17

binario decimal
0110111 -65

binario decimal
1010111 -10

binario decimal
1110100 -17

binario decimal
0110111 -65

No confundas la bandera de "acarreo" con la bandera de "desborde" en aritmética entera. Cada bandera puede ocurrir por sí misma.

A la ALU del CPU no le importa si estás llevando a cabo matemática con 0 o sin signo; la ALU siempre establece ambas banderas apropiadamente cuando efectúa cualquier operación matemática entera. La ALU no sabe nada sobre números con signo; esta solalemente efectúa las operaciones matemáticas binarias y establece las banderas apropiadamente. Depende de ti, el programador, saber qué bandera verificar después de que la operación matemática se ha efectuado.

Si tu programa trata los bits de una palabra como valores con signo de complemento a dos, debes verificar si tu aritmética activa la bandera de desborde, indicando que el resultado es incorrecto. A ti no te interesa la bandera de acarreo cuando efectúas matemática con signo de complemento a dos (esta bandera solamente es relevante para números sin signo, no con signo).

En aritmética sin signo, verifica la bandera de acarreo para detectar errores. En aritmética sin signo, la bandera de desborde no te dice nada interesante.

En aritmética con signo, verifica la bandera de desborde para detectar errores. En aritmética con signo, la bandera de acarreo no te dice nada interesante.

Idioma

No confundas el verbo "desbordar" con la "bandera de desborde".

que algún resultado matemático no cabe en el número de bits disponibles; esta podría ser matemática entera, o matemática de punto flotante, o lo que sea. La "bandera de desborde" se activa específicamente por la ALU como se describe a continuación, y no es lo mismo que el verbo casual "desbordar".

Podríamos decir "la matemática binaria/entera desbordó el número de bits disponibles para el resultado, provocando que la bandera de acarreo se active". Nota que este uso del verbo "desbordar" "no" es lo mismo que decir "la bandera de desborde está activada". Un resultado matemático puece desbordar (el verbo) el numero de bits disponibles sin activar la bandera de desborde de la ALU.

Bandera de Acarreo

Las reglas para activar la bandera de acarreo en matemática binaria/entera son dos:

- La bandera de acarreo se activa si la adición de dos números provoca un acarreo fuera de los bits más significativos (más a la izquierda) agregados.

1111 + 0001 = 0000 (la bandera de acarreo se activa)

- La bandera de acarreo (prestado) también se activa si la resta de dos números requiere prestar hacia los bit más significativos (mas a la izquierda) restados.

0000 - 0001 = 1111 (la bandera de acarreo se activa)

De lo contrario, la bandera de acarreo se desactiva (se pone a cero).

* 0111 + 0001 = 1000 (carry flag is turned off [zero])
* 1000 - 0001 = 1011 (carry flag is turned off [zero])

En aritmética sin signo, mira la bandera de acarreo para detectar errores. En aritmética con signo, la bandera de acarreo no te dice nada interesante.

Bandera de Desborde (Overflow)

Las reglas para activar la bandera de desborde en matemática binaria entera son dos:

- Sí la suma de dos números con los bits de signo a 0 dan un número con el bit de signo activado, la bandera de "desborde"
- (overflow) se activa.

- Sí la suma de dos números con los bits de signo a 1 dan un número con el bit de signo desactivado, la bandera de "desborde" (overflow) se activa.

1000 + 1000 = 0000 (overflow flag is turned on)

* 0100 + 0001 = 0101 (overflow flag is turned off)
* 0110 + 1001 = 1111 (overflow flag is turned off)
* 1000 + 0001 = 1001 (overflow flag is turned off)
* 1100 + 1100 = 1000 (overflow flag is turned off)

Nota 1: El realizar operaciones con signo y negar el resultado si no tiene cuando se trata de valores negativos

Nota 2: Las operaciones con complemento a dos, si hay acarreo se debe sumar el acarreo (suma del complemento a dos)

Nota 3: Las operaciones aritméticas con signos negativos se realizan a través del complemento a dos

Nota que solamente necesitas ver los bits de signo (los del extremo izquierdo) de los tres números para decidir si la bandera de desborde está activada o no.

Si estás efectuando aritmética de complemento a dos (con signo), la bandera de desborde activada significa que la respuesta es incorrecta — agregaste dos números positivos y obtuviste un negativo, o agregaste dos números negativos y obtuviste un positivo.

Si estás efectuando aritmética sin signo, la bandera de desborde no significa nada y debería ignorarse.

Las reglas para el complemento a dos detectan errores al examinar el signo del resultado. Un número negativo y uno positivo agregados juntos no pueden dar un resultado erróneo, porque la suma estará entre los andenes. Ya que ambos andenos están dentro del rango permitido de números, y su suma está entre ellos, esta también debe estar entre estos. La adición de signo mezclada nunca activa la bandera de desborde.

En la aritmética con signo, mira la bandera de desborde para detectar errores. En la aritmética sin signo, la bandera de desborde no dice nada interesante.

Cómo Calcula la ALU la Bandera de Desborde

Este material es lectura opcional.

Hay varias formas automatizadas de detectar errores de desborde en la aritmética binaria de complemento a dos (para aquellos de ustedes a quienes no les gusta el método de inspección manual). Aquí están dos:

Calculando la Bandera de Desborde: Método 1

El desborde solo sucede cuando se agregan dos números con el mismo signo y se obtiene un signo diferente. Así que para detectar el desborde no nos importa ningún bit excepto los bits de signo. Ignora los demás bits.

Con dos operandos y un resultado, tenemos tres bits de Signo (cada uno siendo 0 o 1) a considerar, así que tenemos exactamente $2^3 \times 8 = 64$ posibles combinaciones de los tres bits. Solamente dos de estos 64 posibles casos son considerados desborde. A continuación están solo los bits de signo de los dos operandos adicionales y el resultado:

BITS DE SIGNO DE SUMA	num1.sign num2.sign sum.sig
-----	-----
0 0 0	0 0 0
OVER	+OVER*
0 0 1	0 0 1 (agregar dos números positivos debería dar un positivo)
0 1 0	0 1 0
0 1 1	0 1 1
1 0 0	1 0 0
1 0 1	1 0 1 (agregar dos negativos debería dar negativo)
1 1 0	1 1 0
1 1 1	1 1 1

Podemos repetir la misma tabla para la resta. Nota que restar un número positivo es lo mismo que agregar un negativo, así que las condiciones que gatillan la bandera de desborde son:

BITS DE SIGNO DE RESTA	num1.sign num2.sign sum.sig
-----	-----
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	0 1 0 (restar un negativo es lo mismo que sumar un positivo)
OVER	*OVER*
0 1 1	0 1 1 (restar un positivo es lo mismo que sumar un negativo)
1 0 0	1 0 0
1 0 1	1 0 1
1 1 0	1 1 0
1 1 1	1 1 1

Una computadora puede contener un pequeño arraylo de comprobaciones lógicas que activa la bandera de desborde, poniéndola a 1 si cualquiera de las cuatro condiciones anteriores se cumple.

Un humano solamente recordar que, cuando se efectúa matemática con signo, agregar dos números del mismo signo debe producir un resultado del mismo signo, de lo contrario se ha provocado un desborde.

Calculando la Bandera de Desborde: Método 2

Cuando se agregan dos valores binarios, considera el acarreo binario entrante del bit de mayor peso (hacia el bit de signo) y el acarreo binario que saliente de ese bit de mayor peso. (El acarreo saliente del bit de signo de mayor peso se convierte en la bandera CARRY en la ALU.)

• CI USUARIO EN EL CONGRESO DE DIAZ PUEDE ULTRIR, NO CUADRO EN UN ES ACARREO HACIA IZQUIERDA UC LA COMPUTACION CONVERGIDA, SIN LA LLEGADA

se acarrea uno hacia este y no ocurre un acarreo correspondiente. Es decir, el desborde sucede cuando hay un acarreo hacia el bit de signo pero no hay acarreo fuera del bit de signo.

La bandera OVERFLOW es el XOR del acarreo saliente al bit de signo (si lo hay) con el acarreo saliente al bit de signo (si lo hay). El desborde sucede si el acarreo entrante no es igual que el acarreo saliente.

Ejemplos (números con signo de dos bits de complemento a 2):

BITS DE SIGNO DE SUMA	num1.sign num2.sign
-----	-----
0 1	0 1
+01	+01
=====	=====
0 0	0 0

- el acarreo entrante es 1
- el acarreo saliente es 1
- 1 XOR 1 = NO HAY DESBORDE

BITS DE SIGNO DE SUMA	num1.sign num2.sign
-----	-----
1 0	1 0
+01	+01
=====	=====
1 0	1 1

- el acarreo entrante es 1
- el acarreo saliente es 0
- 1 XOR 0 = DESBORDE!

BITS DE SIGNO DE SUMA	num1.sign num2.sign
-----	-----
1 0	1 0
+10	+10
=====	=====
0 1	0 1

- el acarreo entrante es 0
- el acarreo saliente es 1
- 0 XOR 1 = DESBORDE!

BITS DE SIGNO DE SUMA	num1.sign num2.sign
-----	-----
0 1	0 1
+01	+01
=====	=====
1 1	1 1

- el acarreo entrante es 0
- el acarreo saliente es 0
- 0 XOR 0 = NO HAY DESBORDE

Nota que este método XOR solamente funciona con el acarreo "binario" entrante al "bit" de signo. Si estás trabajando con números hexadecimales, o números decimales, o números octales, también tienes acarreo, pero el acarreo no va en el "bit" de signo y no puedes aplicar XOR a ese acarreo no binario con el acarreo saliente.

Ejemplo de Adición (mostrando que el XOR no funciona para el acarreo hexadecimal):

BITS DE SIGNO DE RESTA	num1.sign num2.sign sum.sig
-----	-----
0 0 0	0 0 0
+0Ah	+0Ah
=====	=====
1 4 h	1 4 h

OVER 0 1 1 (restar un negativo es lo mismo que sumar un positivo)
OVER 0 1 0 (restar un positivo es lo mismo que sumar un negativo)
OVER 1 0 0 (restar un positivo es lo mismo que sumar un positivo)
OVER 1 0 1 (restar un positivo es lo mismo que sumar un positivo)
OVER 1 1 0 (restar un positivo es lo mismo que sumar un positivo)
OVER 1 1 1 (restar un positivo es lo mismo que sumar un positivo)

El acarreo hexadecimal de 1 que resulta de A+A no afecta el bit de signo. Si haces la operación en binario, verás que "no" hay acarreo "entrante" al bit de signo; pero hay acarreo saliente del bit de signo. Por lo tanto, el ejemplo anterior activa el desborde (OVERFLOW). (El ejemplo agrega dos números negativos y obtiene un número positivo.)

TP4

② Numeros con Negativo

- Números representar un número de 8 bits con signo

Tenemos que:

$$\begin{aligned} & \left. \begin{aligned} & \text{8 bits} \\ & \text{7 bits magnitud} \end{aligned} \right\} \text{bit negativo} (\text{MSB}) \\ + 127_{10} & \rightarrow \begin{array}{r} \text{negativo} \\ \hline \text{magnitud} \\ \text{MSB} \end{array} \end{aligned}$$

8 bits \rightarrow Número: $0_0 a 125_{10}$ (256 combinaciones)

con signo: $-128_0 a +127_{10}$ (256 combinaciones)

- El bit de signo en el número se representa siempre con el bit de mayor peso (MSB) y no con el carry out.
- Como el máximo que puedo representar en un número de 8 bits con signo es $+127$ cualquier número mayor o igual saldrá un una suma devuélva con un desborde.
- El desborde (overflow) representa en la suma un valor menor por lo tanto la bandera overflow es un indicador de un error en el resultado. La bandera overflow se la puede obtener realizando la operación XOR entre el carry en y el carry out del bit signo (MSB).

$$\begin{array}{r} 30_{16} \\ + 48_{10} \\ \hline \text{no hay overflow} \end{array} \quad \begin{array}{r} 41_{16} \\ + 65_{10} \\ \hline \text{no hay overflow} \end{array} \quad \begin{array}{r} 113_{10} \\ \leftarrow \text{max} + 127_{10} \\ \text{no hay overflow} \end{array}$$

$$C_1 = 0 \quad \begin{array}{r} \text{magnitud} \\ \hline 0'01100000 \end{array} \quad C_0 = 0 \quad \begin{array}{r} \text{magnitud} \\ \hline 0_11000010 \end{array}$$

$$\text{overflow} = C_1 \oplus C_0 = 0 \oplus 0 = 0$$

$$\begin{array}{r} 5D_{16} \\ + 22_{10} \\ \hline \text{no hay overflow} \end{array} \quad \begin{array}{r} 93_{10} \\ + 65_{10} \\ \hline \text{no hay overflow} \end{array} \quad \begin{array}{r} \text{max} + 7F_{16} \rightarrow +7F_{16} \\ \leftarrow \text{max} + 127_{10} \\ \text{no hay overflow} \end{array}$$

$$C_1 = 0 \quad \begin{array}{r} \text{magnitud} \\ \hline 0'10100010 \end{array} \quad C_0 = 0 \quad \begin{array}{r} \text{magnitud} \\ \hline 0_11111110 \end{array}$$

$$\text{overflow} = 0 \oplus 0 = 0$$

$$\begin{array}{r} 51_{16} \\ + 47_{10} \\ \hline \text{no hay overflow} \end{array} \quad \begin{array}{r} 81_{10} \\ + 47_{10} \\ \hline \text{no hay overflow} \end{array} \quad \begin{array}{r} 128_{10} \leftarrow \text{max} + 127_{10}, \\ \text{no hay overflow} \end{array}$$

$$C_1 = 1 \quad \begin{array}{r} \text{magnitud} \\ \hline 0'10101101 \end{array} \quad C_0 = 0 \quad \begin{array}{r} \text{magnitud} \\ \hline 0_11111110 \end{array}$$

$$\text{overflow} = 1 \oplus 0 = 1$$

$$\begin{array}{r} 5D_{16} \\ + 22_{10} \\ \hline \text{no hay overflow} \end{array} \quad \begin{array}{r} 93_{10} \\ + 65_{10} \\ \hline \text{no hay overflow} \end{array} \quad \begin{array}{r} \text{max} + 7F_{16} \rightarrow +80_{16} \\ \leftarrow \text{max} + 127_{10}, \\ \text{no hay overflow} \end{array}$$

$$C_1 = 1 \quad \begin{array}{r} \text{magnitud} \\ \hline 0'10101101 \end{array} \quad C_0 = 0 \quad \begin{array}{r} \text{magnitud} \\ \hline 0_11111110 \end{array}$$

$$\text{overflow} = 1 \oplus 0 = 1$$

- Si el resultado de una resta da un numero negativo no se podre determinar el valor del resultado en forma directa ya que el numero negativo esta expresaado en C2.
Para ver su valor en forma directa es necesario hacer el C2 numerante.

valores de \$B16\$	8 bits con signo (complemento)	8 bits sin signo
00000000	0	0
00000001	1	1
⋮	⋮	⋮
01111110	126	+4 F_{16}
01111111	127	- +7 9_{10}
10000000	-128	+3 A_{16}
10000001	-127	<u>+2 $1_{10} < \text{máx} -128_{10}$</u>
10000010	-126	no hay overflow
⋮	⋮	$c_1 = 1$
11111101	-3	0100111112
11111110	-2	-0011101002
11111111	-1	$G_0 = 1$ 0001010102

Resta

- La operación C2 permite realizar el trage de un numero positivo a un negativo permitiendo así convertir la operacion resta en una suma de un numero positivo mas otro en C2.
El C2 tambien permite tragar de negativos a positivos por lo que es útil también para verificar resultados que estan en C2 (negativos) obteniendo el modulo de dicho numero.
- Un numero negativo y otro positivo agregados juntos no pueden dar un resultado erroneo (overflow) ya que los dos numeros se encuentran dentro del rango representable por los bits que los representa.

• $126 - 15_{16} = 0_{10}$ → El resultado es +1010
que no hace falta aplicar el C2

• $127 - 15_{16} = 1_{10}$ → El resultado es +1010
que no hace falta aplicar el C2

• $128 - 15_{16} = 15_{16}$ → $\frac{15_{16}}{+2_{10} < \text{máx} -128_{10}}$
no hay overflow

• $129 - 15_{16} = 1_{10}$ → $\frac{1_{10}}{+0_{10} < \text{máx} -128_{10}}$
no hay overflow

• $130 - 15_{16} = 0_{10}$ → $\frac{0_{10}}{+0_{10} < \text{máx} -128_{10}}$
no hay overflow

• $126 - 15_{16} = 0_{10}$ → $\frac{0_{10}}{+0_{10} < \text{máx} -128_{10}}$
no hay overflow

TP4

(2)

$$\begin{array}{r} c_0=0 \\ -0'00000000_2 \\ +0'00000000_2 \\ \hline 0'111111_2 \end{array} \xrightarrow{\text{C2}} \begin{array}{r} +1'000000_2 \\ \hline 0'111111_2 \end{array}$$

$$c_0=0$$

$$\boxed{1'000000_2}$$

Overflow = $0 \oplus 0 = 0$
no hay overflow
El numero es - tiene
prefijo de signo
se calcula

$$\begin{array}{r} +0'111111_2 \\ +0'111111_2 \end{array} \rightarrow |+7F|_6 = |127|_0$$

$$\mu \text{ C2}$$

$$\begin{array}{r} +15_6 \\ -6A_6 \\ \hline -106_6 \end{array} \xrightarrow{\text{C2}} \begin{array}{r} +21_6 \\ -127_6 \end{array} \leftarrow \text{num } -128_0$$

no hay overflow

$$\begin{array}{r} +15_6 \\ -6A_6 \\ \hline -106_6 \end{array} \xrightarrow{\text{C2}} \begin{array}{r} +21_6 \\ -127_6 \end{array} \leftarrow \text{num } -128_0$$

no hay overflow

$$\begin{array}{r} c_1=1 \\ -00010101_2 \\ +0'11010102 \end{array} \xrightarrow{\text{C2}} \begin{array}{r} 1'11010112 \\ +1'00101102 \end{array}$$

$$\boxed{c_0=1'00000001_2}$$

$$\text{Overflow = } 1 \oplus 1 = 0 \quad (\text{C2 no funciona})$$

$$\text{no hay overflow}$$

$$\boxed{|+7F|_6 = |127|_0}$$

$$\begin{array}{r} -23_6 \\ + -5D_6 \\ \hline -93_6 \end{array} \xrightarrow{\text{C2}} \begin{array}{r} -80_6 \leftarrow \text{num } -128_0 \\ -128_6 \end{array}$$

$$\text{Overflow = } 1 \oplus 1 = 0 \quad (\text{C2 no funciona})$$

$$\text{no hay overflow}$$

$$\boxed{-80_6 = -128_0}$$

$$\begin{array}{r} -62_6 \\ + -7D_6 \\ \hline -112_6 \end{array} \xrightarrow{\text{C2}} \begin{array}{r} -210_6 \leftarrow \text{num } -128_0 \\ -128_6 \end{array}$$

$$\text{Overflow = } 1 \oplus 0 = 1 \quad (\text{C2 no funciona})$$

$$\text{no hay overflow}$$

$$\boxed{-80_6 = -128_0}$$

$$\begin{array}{r} -62_6 \\ + -7D_6 \\ \hline -112_6 \end{array} \xrightarrow{\text{C2}} \begin{array}{r} -210_6 \leftarrow \text{num } -128_0 \\ -128_6 \end{array}$$

$$\text{Overflow = } 1 \oplus 0 = 1 \quad (\text{C2 no funciona})$$

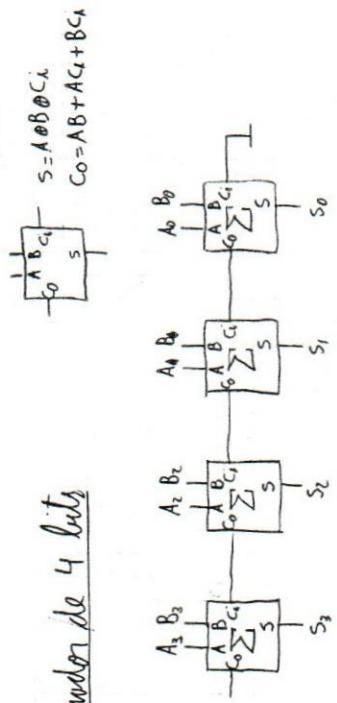
$$\text{no hay overflow}$$

$$\boxed{-80_6 = -128_0}$$

$$\begin{array}{r} 10101001_0 \\ \rightarrow (\text{error}) \end{array}$$

TP4

③ Numerador de 4 bits



Numerador 4 bits

$$S = A \oplus B \oplus C_i$$

$$C_0 = AB + AC_i + BC_i$$

$$f_1$$

AB	CD	f ₁
00	00	0
00	01	0
00	10	0
00	11	1
01	00	0
01	01	0
01	10	0
01	11	1
10	00	0
10	01	0
10	10	1
10	11	1
11	00	1
11	01	1
11	10	1
11	11	0

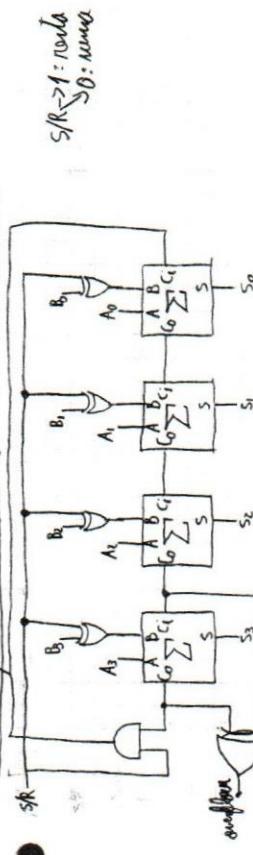
$$f_1 = A \cdot B + A \cdot C$$

$$\dots$$

S/R $\rightarrow 1$: resta

$\rightarrow 0$: suma

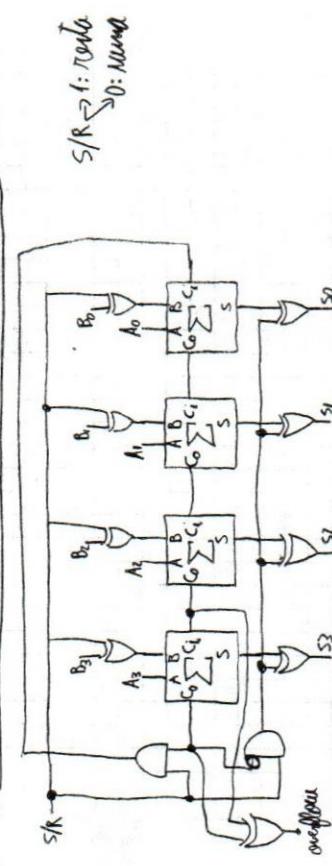
Numerador / restador 4 bits (sin modulo de la resta)



Resta $(A-B) > 0 \Rightarrow A-B = A+\bar{B}+1 - 2^4 \rightarrow$ bit de overflow si avanza?

$(A-B) < 0 \Rightarrow A-B$ dato en complemento a 2

Numerador / restador 4 bits (con modulo de la resta)



Resta $(A-B) > 0 \Rightarrow A-B = A+\bar{B}+1 - 2^4$, bit de overflow si avanza?

Resta $(A-B) < 0 \Rightarrow A-B = \overline{A+B}$

2)

TP4

• VHDL : Numerador 16 bits

Entity numerador16_in

Port (X, Y : IN signed (16 downto 1);

S : OUT signed (16 downto 1);

Cin : IN std_logic;

$Out, overflow$: OUT std_logic);

end numerador16;

Architecture sum of numerador16_in

signal numra : signed (16 downto 1);

begin

$numra <= ('0' \& X) + Y + Cin;$

$S <= numra(17 \text{ downto } 1);$

$Out <= numra(17);$

$overflow <= numra(17) \text{ XOR } X(16) \text{ XOR } Y(16) \text{ XOR } numra(16);$

end numra;

• VHDL : Comparador 4 bits

Entity comp4_m

Port (A, B : IN signed (4 downto 1);

$mult, men, Ig$: OUT std_logic);

Architecture comparador of comp4_m

begin

$mult <= '1'$ when $A > B$ else '0';

$men <= '1'$ when $A < B$ else '0';

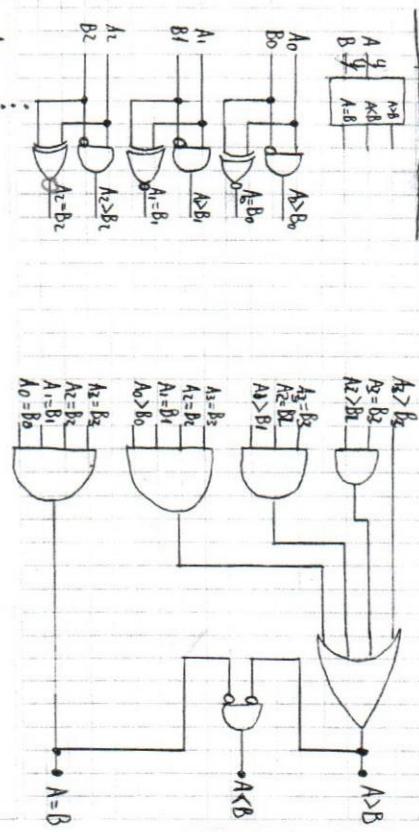
$Ig <= '1'$ when $A = B$ else '0';

end comparador

• Comparador 4 bits

A	B	$A=B$	$A>B$	$A<B$
0	0	0	0	0
0	1	0	1	0
1	0	0	0	1
1	1	0	0	0

• Comparador 4 bits



TP4

• VHDL : ALU y falt

Entity ALU

Port ($A, B : \text{IN std_logic_vector}(4 \text{ downto } 1)$;
 $S : \text{IN std_logic_vector}(3 \text{ downto } 1)$;
 $F : \text{OUT std_logic_vector}(4 \text{ downto } 1)$);

end alu;

Architecture alu of alu is

```
Begin
    process (A, B, S)
        begin
            case S is

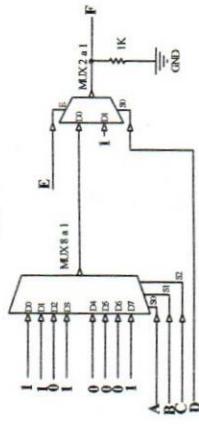
```

```
                when "000" => F <= "0000";
                when "001" => F <= B - A;
                when "010" => F <= A - B;
                when "011" => F <= A + B;
                when "100" => F <= A XOR B;
                when "101" => F <= A OR B;
                when "110" => F <= A and B;
                when "111" => F <= "1111";
            end case;
        end process;
    end alu;
```

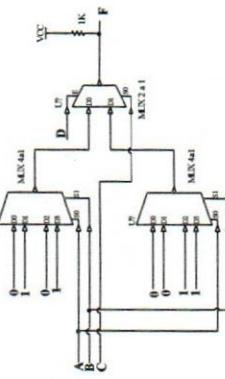
MUXES CODIFICADORES Y DECODIFICADORES

1. Determine las funciones lógicas correspondiente a los siguientes esquemas con multiplexores. Exprese la función como suma de minterminos.

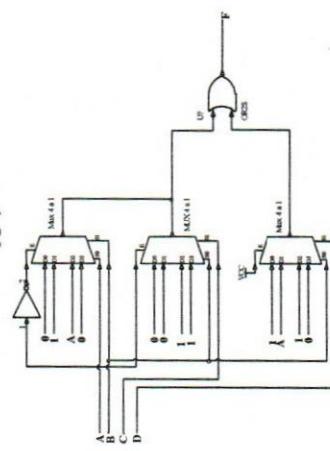
FA = ?



FB = ?



FC = ?





2. Resolver las siguientes funciones lógicas con dos multiplexores de 4:1. Junto a cada función a resolver se encuentra la referencia a la combinación de multiplexores a utilizar. Con el tipo de MUX ver la tabla de verdad para la combinación de la salida.

- $f_{a(ABC)} = \sum{0, 2, 3, 5, 7}$ [Mux1,Mux1],[Mux2,Mux2],[Mux3,Mux3]
- $f_{b(BC)} = \sum{1, 3, 4, 6, 7}$ [Mux2,Mux2],[Mux3,Mux3],[Mux1,Mux2]
- $f_{c(ABC)} = \sum{0, 2, 4, 5, 6, 7}$ [Mux3,Mux3],[Mux1,Mux2],[Mux2,Mux1]
- $f_{d(BCD)} = \sum{1, 3, 4, 7, 10, 12, 14}$ [Mux3,Mux2],[Mux1,Mux2]

Mux2

CE	A	B	out
1	x	x	1
0	0	0	ln0
0	0	1	ln1
0	1	0	ln2
0	1	1	ln3

Mux3

CE	A	B	out
1	x	x	Z
0	0	0	ln0
0	0	1	ln1
0	1	0	ln2
0	1	1	ln3

3. Implementar las siguientes funciones utilizando un multiplexor de 8 a 1.

- $f_{a(BCD)} = \prod{0, 2, 3, 5, 7, 12, 10, 14}$
- $f_{b(BCD)} = \sum{1, 3, 4, 6, 7}$
- $f_{c(BCD)} = \prod{0, 2, 4, 5, 6, 7, 10, 11, 12, 15}$
- $f_{d(BCD)} = \sum{1, 2, 3, 4, 8, 9, 10, 12, 14}$
- $f_{e(BCD)} = \prod{0, 2, 3, 5, 7, 8, 9, 12, 14, 15}$
- $f_{f(BCD)} = \sum{1, 3, 4, 6, 7}$
- $f_{g(BCD)} = \prod{0, 2, 4, 5, 6, 7, 14, 15}$
- $f_{h(BCD)} = \sum{0, 1, 4, 7, 10, 12, 14}$

4. Implementar las siguientes funciones utilizando un multiplexor de 8 a 1.

- $f_{a(BCD)} = \sum{0, 2, 3, 5, 7, 12, 10, 14}$
- $f_{b(BCD)} = \sum{1, 3, 4, 6, 7}$
- $f_{c(BCD)} = \sum{0, 2, 4, 5, 6, 7, 10, 11, 12, 15}$
- $f_{d(BCD)} = \sum{1, 2, 3, 4, 8, 9, 10, 12, 14}$
- $f_{e(BCD)} = \sum{0, 2, 3, 4, 8, 9, 10, 12, 14}$
- $f_{f(BCD)} = \sum{0, 2, 3, 5, 7, 8, 9, 12, 14, 15}$
- $f_{g(BCD)} = \prod{1, 3, 4, 6, 7}$
- $f_{h(BCD)} = \sum{0, 2, 4, 5, 6, 7, 14, 15}$

5. Diseñar un decodificador BCD a 7 segmentos utilizando multiplexores de 4 a 1 tipo CD4052. En caso de que se presente a la entrada un dato erróneo inhibir la salida utilizando la señal de selección del chip.

- Describir un multiplexor 8:1 y otro de 4:1 con control de salida para tercer estado en VHDL, utilizando "WHEN ELSE"
- <name> <= <expression> when <condition> else <expression>
- <expression>

6. Describir un sumador de 4 bits en VHDL, utilizando multiplexores.

- Adjuntar el código fuente.
- Adjuntar la imagen de la simulación.

7. Describir el decodificador de errores para un código de Hamming de 9 bits de información utilizando codificadores de 3 a 8.

- Diseñar un sistema combinacional con dos entradas A y B de 8 bits cada una, las cuales representan un número entero sin signo, y una entrada de control extra MIN/MAX. La salida Z también es de 8 bits. La salida Z=0 si A=B, Z=min(A,B) si MIN/MAX=1, Z=max(A,B) si MIN/MAX=0.
- Diseñar el sistema utilizando multiplexores.
- Describir el sistema en VHDL.
- Adjuntar la descripción en VHDL.
- Adjuntar la pantalla de simulación.

9. Implementar el decodificador de errores para un código de Hamming de 9 bits de información utilizando codificadores de 3 a 8.

- Diseñar un sistema combinacional con dos entradas A y B de 8 bits cada una, las cuales representan un número entero sin signo, y una entrada de control extra MIN/MAX. La salida Z también es de 8 bits. La salida Z=0 si A=B, Z=min(A,B) si MIN/MAX=1, Z=max(A,B) si MIN/MAX=0.

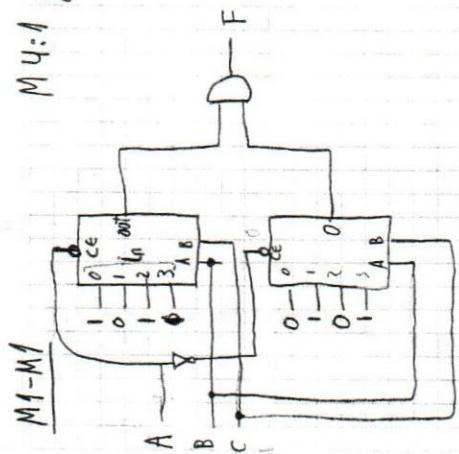
- Diseñar el sistema utilizando multiplexores.
- Describir el sistema en VHDL.
- Adjuntar la descripción en VHDL.
- Adjuntar la pantalla de simulación.

$$\textcircled{a} \quad f_d = \sum_{m_1} 1, 3, 4, 7, 10, 12, 14$$

$$M1-M1 \\ M4:1 \neq 3mn \\ M4:1 \times 2$$

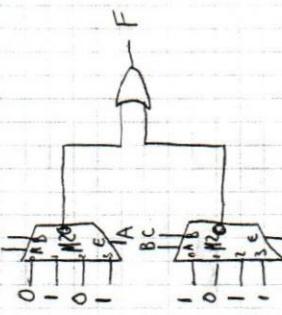
(TP5)

$$f(a) = \sum_{m_1} 0, 2, 5, 7$$



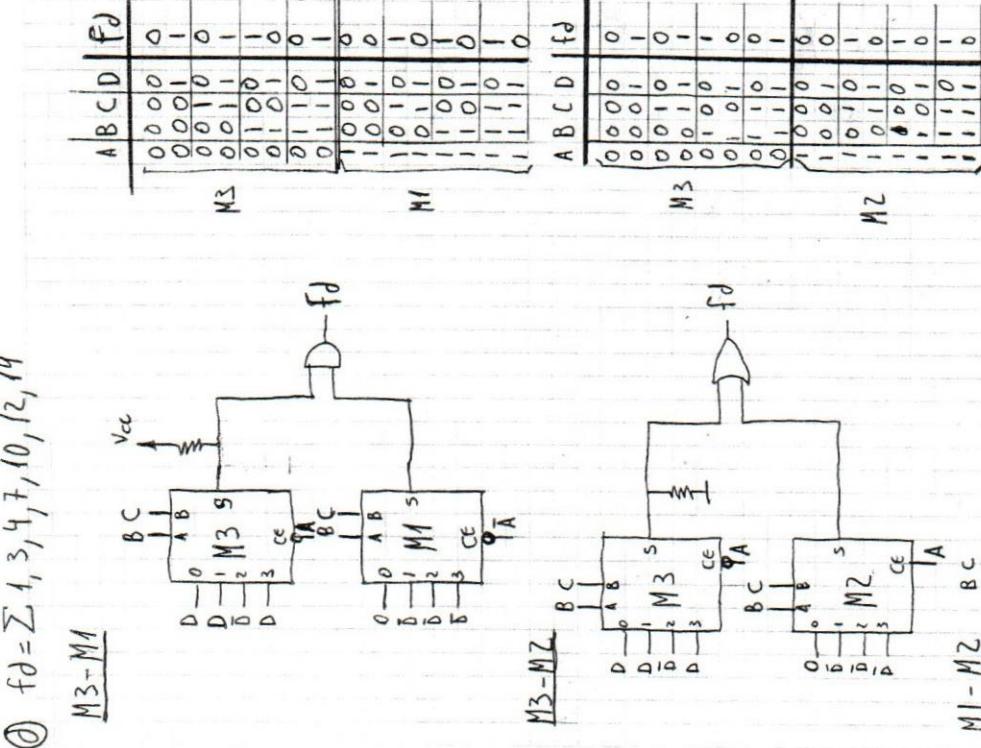
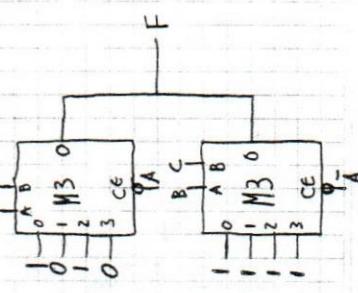
\textcircled{b}

$$M2-M2 \\ f_b = \sum_{m_1} 1, 3, 4, 6, 7$$

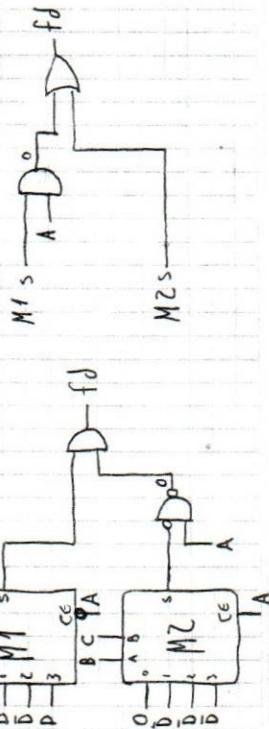


\textcircled{c}

$$M3-M3 \\ f_c = \sum_{m_1} 0, 2, 4, 5, 6, 7$$



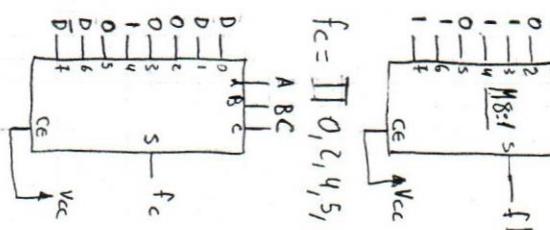
0=AS	M1	AS	0	0
1	AS	0	0	0
0	0	1	0	1
1	0	1	1	1
0	1	0	0	0
1	1	0	1	0
0	1	1	0	1
1	1	1	1	1



$$\textcircled{3} \quad \textcircled{6} \quad f_b = \sum I, 3, 4, 6, 7$$

M8:1 X1

(TP5)

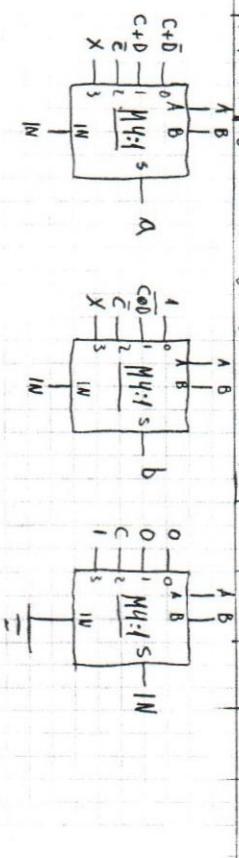


$$\textcircled{6} \quad f_c = \prod I, 0, 2, 4, 5, 6, 7, 10, 11, 12, 15$$

A	B	C	D	f _c
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

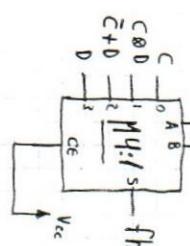
$$\textcircled{5} \quad M4:1 \quad \begin{matrix} a \\ \frac{f}{g} \\ b \\ e \\ \frac{d}{\bar{d}} \\ c \\ f \\ g \end{matrix}$$

A	B	C	D	f _h
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



$$\textcircled{4} \quad M4:1 \quad X1$$

$$f_h = \prod I, 1, 4, 7, 10, 12, 14$$



⑥ VHDL de un MUX 4:1 con control de validos para
tener estados mundoullen-else

```
entity mux41 is
  port (out : IN STD_LOGIC_VECTOR(3 DOWNTO 1);
        cont : IN STD_LOGIC_VECTOR(2 DOWNTO 1);
        in : IN STD_LOGIC;
        val : OUT STD_LOGIC);
end mux41;
architecture multiplexor of mux41 is
begin
  val <= int(0) when (c = "00" and in = '1') else
    int(1) when (c = "01" and in = '1') else
    int(2) when (c = "10" and in = '1') else
    int(3) when (c = "11" and in = '1') else
    'Z' when in = '0' else
    unaffected;
end multiplexor;
```

⑥ VHDL de un MUX 4:1 con control de validos para tener
estados mundoullen-else

```
entity mux41 is
  port (out : IN STD_LOGIC_VECTOR(4 DOWNTO 1);
        cont : IN STD_LOGIC_VECTOR(2 DOWNTO 1);
        in : IN STD_LOGIC;
        val : OUT STD_LOGIC);
end mux41;
architecture multiplexor of mux41 is
begin
  process
    begin
      if (cont = "00" and in = '1') then val <= int(0);
      elsif (cont = "01" and in = '1') then val <= int(1);
      elsif (cont = "10" and in = '1') then val <= int(2);
      elsif (cont = "11" and in = '1') then val <= int(3);
      else val <= 'Z';
      end if;
    end process;
    end multiplexor;
```

⑥ VHDL de un MUX 4:1 con control de validos para tener
estados mundoullen-else

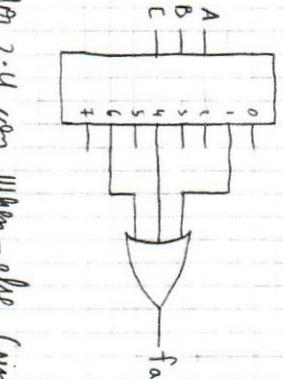
$$\textcircled{8} \quad e) f_e = \prod_{i=0}^7 \sum_{j=0}^3 f_{e_i}$$

A	B	C	D	f_e
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

(TP5)

$$\textcircled{8} \quad f_a = \prod_{i=0}^7 \sum_{j=0}^3 f_{a_i}$$

A	B	C	f_a
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0



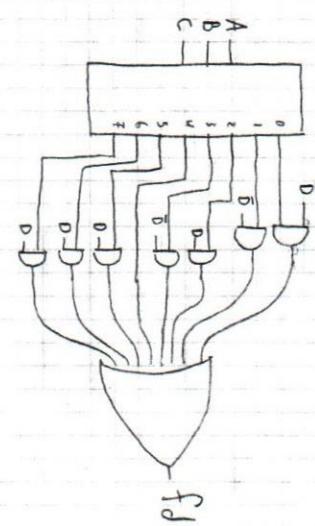
• VHDL: 3 selectwelen 2:4 von When-else (unprocess)

```
entity decs24 is
port ( out : IN STD_LOGIC_VECTOR (2 downto 1),
      enable : IN STD_LOGIC;
      sel : OUT STD_LOGIC_VECTOR (4downto 1));
end decs24;
```

begin

```
when sel <= "0001" when (out = "00" and enable = '1') else
"0010" when (out = "01" and enable = '1') else
"0100" when (out = "10" and enable = '1') else
"1000" when (out = "11" and enable = '1') else
"0000" others;
```

end decode;



$$f_d = \prod_{i=0}^7 \sum_{j=0}^3 f_{d_i}$$

A	B	C	f_d
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

S	D	f
0	0	0
0	1	0
1	0	1
1	1	0

$$\textcircled{8} \quad d) f_d = \prod_{i=0}^7 \sum_{j=0}^3 f_{d_i}$$

②

VHDL Decodifizieren

(TP5)

- VHDL Decodifizierer 2:4 von if-schleife aus - müssen
(con process)

```
Port ( out : 1W std_logic_vector (2 downto 1) ;
      enable : 1W std_logic ;
      sel : 00T std_logic_vector (4 downto 1) );
end dec24;
```

```
Architecture decodif_24_m
Begin
```

```
    process (out, enable)
    begin
```

```
        if (enable = '1') then
            case out is
                when "00" => sel <= "0001";
                when "01" => sel <= "0010";
                when "10" => sel <= "0100";
                when "11" => sel <= "1000";
            end case
```

```
        elsif (enable = '0') then
            sel <= "0000";
        end if;
    end process
end decodif;
```

- VHDL Decodifizierer BCD a 7 Segmente von 10bit - nicht (nur 8bit)

Entity dec8_m

```
Port ( out : 1W std_logic_vector (4 downto 1) ;
      sel : OUT STD_LOGIC_VECTOR (7 downto 1) );
```

end dec8;

Architecture reg_of_duo_m

```
begin
```

```
    wait (out);
```

```
    sel <= "11110" when "0000" ;
    sel <= "0110000" when "0001" ;
    sel <= "1101101" when "0010" ;
    sel <= "1110001" when "1001" ;
    sel <= "0000000" when "other" ;
```

end reg;

• VHDL Decod. HEX a Tag von case - wenn (con process)

Entity dec_m

```
Port ( out : 1W std_logic_vector (4 downto 1) ;
      sel : OUT STD_LOGIC_VECTOR (7 downto 1) );
```

end dec;

Architecture reg_of_duo_m

```
begin
```

```
    process (out)
```

```
        begin
            sel <= "0000" => sel <= "01110000" ;
            sel <= "0001" => sel <= "01110000" ;
            sel <= "0100" => sel <= "11111111" ;
            sel <= "1111" => sel <= "10001111" ;
        end process;
    end reg;
```

(8)

- NHDL configurador 4:2 con precedencia cuando when - else (sin parallel)

TP5

Entradas	saldos
0,0,0,0	S1,S0,2
0,0,0,X	X,X,0
0,0,1,0	0,0,-1
0,1,X,0	-1,0,-1
0,1,X,X	1,0,-1
1,X,X,X	1,1,1

Entity cod_in

```
port ( out : IN STD_LOGIC_VECTOR (4 downto 1),
      sal : OUT STD_LOGIC_VECTOR (2 downto 1),
      z : OUT STD_LOGIC );
end cod;
```

Architecture C42 of cod_in

Begin

```
sal <= "11" when (out(3)='1' and z='1') else
```

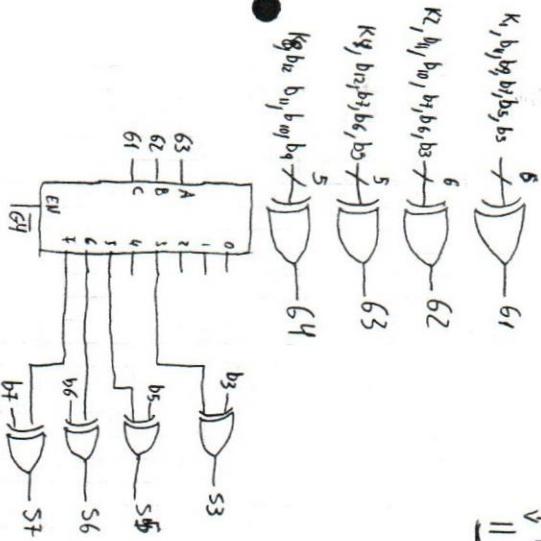
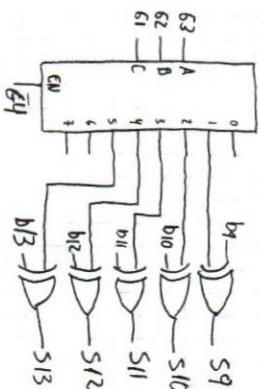
```
"10" when (out(2)='1' and z='1') else
```

```
"01" when (out(1)='1' and z='1') else
```

```
"00" when (out(0)='1' and z='1') else
```

```
"XX" other;
```

and C42;



$$\begin{array}{c} \textcircled{9} \\ \text{b}_6 \text{b}_5 \text{b}_4 \text{b}_3 \text{b}_2 \text{b}_1 \text{b}_0 \text{b}_9 \text{b}_8 \text{b}_7 \text{b}_6 \text{b}_5 \text{b}_4 \text{b}_3 \text{b}_2 \text{b}_1 \text{b}_0 \text{b}_9 \text{b}_8 \text{b}_7 \text{b}_6 \text{b}_5 \text{b}_4 \text{b}_3 \text{b}_2 \text{b}_1 \text{b}_0 \end{array}$$

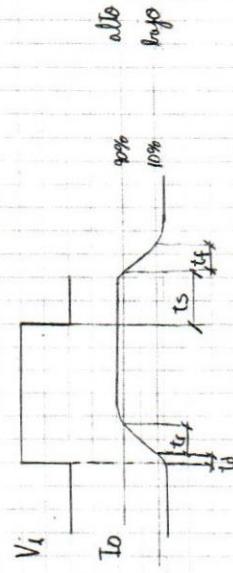
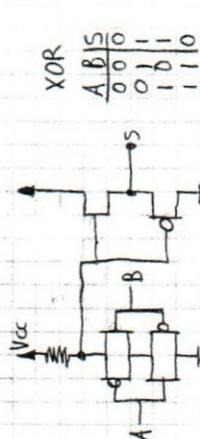
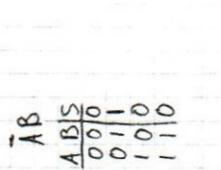
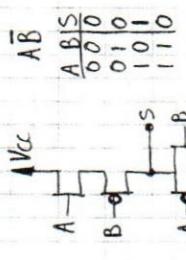
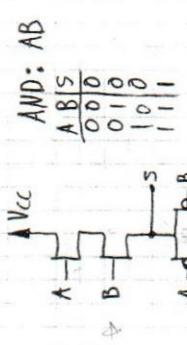
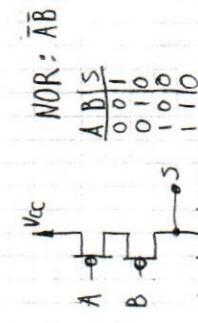
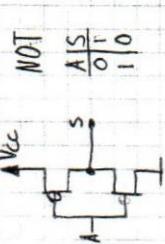
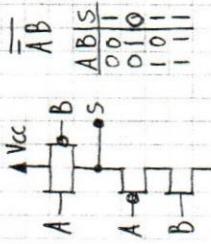
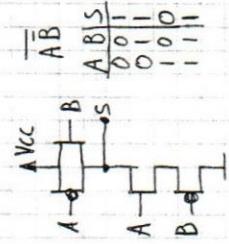
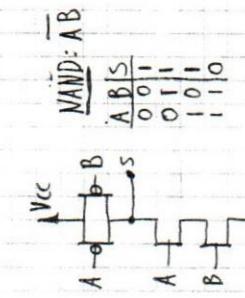
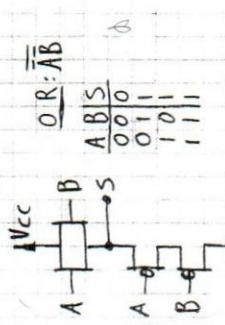
$$\begin{array}{c} \text{b}^{K-k-1} \geq b \\ \boxed{b=9; K=4} \\ \downarrow \\ 11 \geq 9 \end{array}$$

TP5

TP6

Ecuología

Compuertas con CMOS/PMOS



NOR: $\bar{A}\bar{B}$

A	B	S	$\bar{A}\bar{B}$
0	0	0	1
0	1	0	0
1	0	0	0
1	1	1	0

AND: AB

A	B	S	AB
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

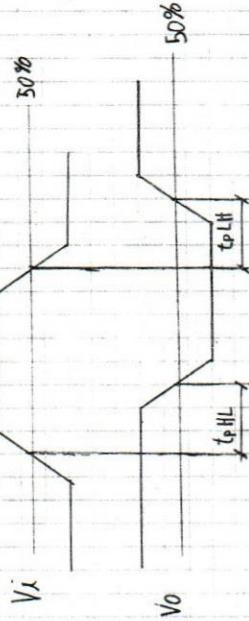
t_d (Tiempo de retardo): Es el tiempo necesario para cargar los capacitores de la unión para que el dominio vale de la puerta de salida a la otra.

t_r (Tiempo rise - Tiempo de subida): Es el tiempo en el que la corriente de salida se eleva del 10% al 90%.

t_s (Time storage - Tiempo de almacenamiento): El periodo de la carga de salida de la base, mientras no saturado no mas carga su circuito terrestro.

t_f (Time fall - Tiempo de caída): Es el tiempo en que la corriente de salida se del 90% al 10%.

Ecuación de propagación:



$$t_p = \frac{t_{PHL} + t_{PLH}}{2}$$

(Tiempo de retardo en la propagación)

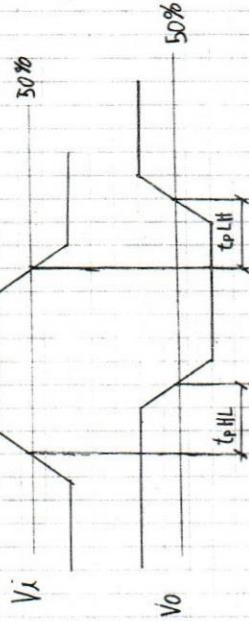
Tiempo de Transistor



t_d (Tiempo de retardo): Es el tiempo necesario para cargar los capacitores de la unión para que el dominio vale de la puerta de salida a la otra.

t_r (Tiempo rise - Tiempo de subida): Es el tiempo en el que la corriente de salida se eleva del 10% al 90%.

t_s (Time storage - Tiempo de almacenamiento): El periodo de la carga de salida de la base, mientras no saturado no mas carga su circuito terrestro.



$$t_p = \frac{t_{PHL} + t_{PLH}}{2}$$

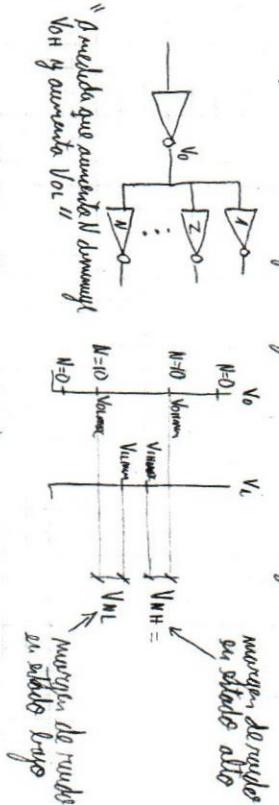
(Tiempo de retardo en la propagación)

(TP6)

Estructura

Margen de ruido

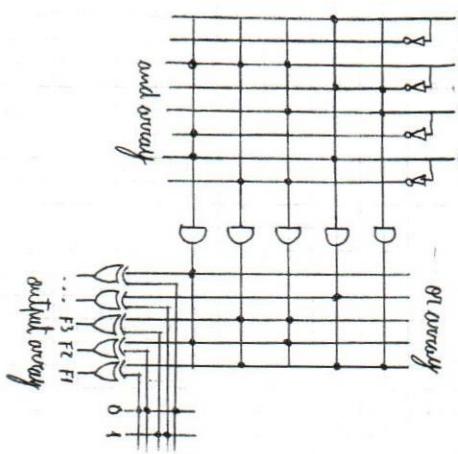
En la tolerancia en los niveles de entrada, que al superarla el fabricante no garantiza los niveles logicos



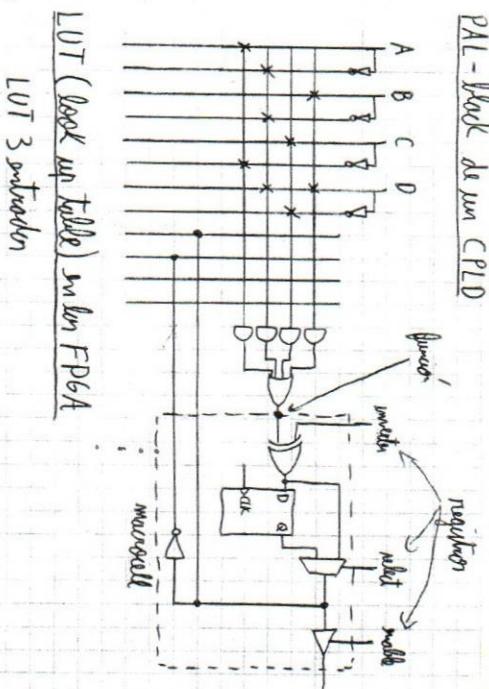
Fan-out (factor de carga de entrada)

Número de cargas máximas que representa una entrada calculable sin que el nivel de corriente de entrada

PLA (Programmable logic array)

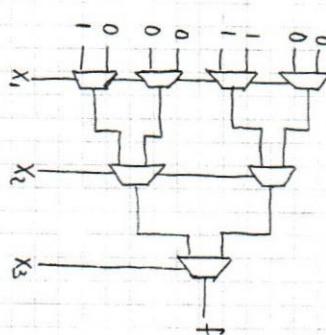


PAL - Block de un CPLD



LUT (Look up table) en un FPGA

x_3	x_2	x_1	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0



TP7) Flip Flop

Setzen

• 5R Auswertung

• Tabelle der Werte

t	t+1	S	R	Q(t)	Q(t+1)
0	0	0	0	0	0
0	1	0	1	0	1
1	0	1	0	1	0
1	1	1	1	1	1

$$Q = \bar{Q} = 1$$

• SR Auswertung

t	t+1	S	R	Q(t)	Q(t+1)
0	0	0	0	0	0
0	1	0	1	0	1
1	0	1	0	1	0
1	1	1	1	1	1

- Melden CK=1, Q nimmt den Wert von S an
- S oder R wenn kein Flanco aufgeprägt an CK, Q rettet den Wert von R

• D Auswertung

t	t+1	D	Q(t)	Q(t+1)
0	0	0	0	0
1	0	1	1	1
0	1	0	1	0

- Melden CK=1, Q nimmt den Wert von D an
- Wenn kein Flanco aufgeprägt an CK, Q rettet den Wert von D

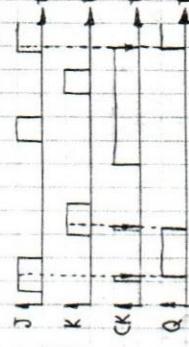
Flip-Flop

JK kontrahieren auswerten

• Tabelle der Werte

t	t+1	J	K	Q(t)	Q(t+1)
0	0	0	0	0	0
0	1	0	1	0	1
1	0	1	0	1	0
1	1	1	1	1	1

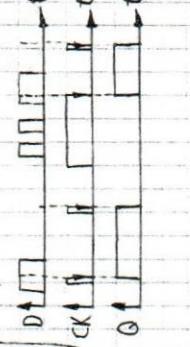
- hole wieder bei einem Flanco de los estados en CK, Q toma el que sea en J & K



• Tabelle der Werte

t	t+1	J	K	Q(t)	Q(t+1)
0	0	0	0	0	0
0	1	0	1	0	1
1	0	1	0	1	0
1	1	1	1	1	1

- hole wieder bei einem Flanco de los estados en CK, Q toma el que sea en J & K



• Tabelle der Werte

t	t+1	D	Q(t)	Q(t+1)
0	0	0	0	0
1	0	1	1	1
0	1	0	1	0

- Melden CK=1, Q nimmt den Wert von D an
- Wenn kein Flanco aufgeprägt an CK, Q rettet den Wert von D



(TP7) FF JK flop

VHDL: FF JK flop reset asynchronous, outputs for flops don't work

Entity FFJK is

```
Port( J,K,R,Clock:IN std_logic;
      Q : OUT std_logic );
```

end FFJK;

Architecture FF of FFJK is

begin

process (CK,R)

begin

if (—) then

Q <= '0';

elsif (CK'event and CK='0' and R='1'

if (J='0' and K='1') then

Q <= '0';

elsif (J='1' and K='0') then

Q <= '1';

elsif (J='1' and K='1') then

Q <= (not Q);

end if;

end process;

end FF;

VHDL: This flop D can reset asynchron., set. on flops don't work

Entity FFD is

```
Port( D,CK,R:IN std_logic;
      Q,Qn:OUT std_logic );
```

end FFD;

Architecture F F of FFD is

begin

process (CK,R)

begin

if (CK'event and CK='0' and R='0') then

Q <= D;

else Q <= '0';

end if;

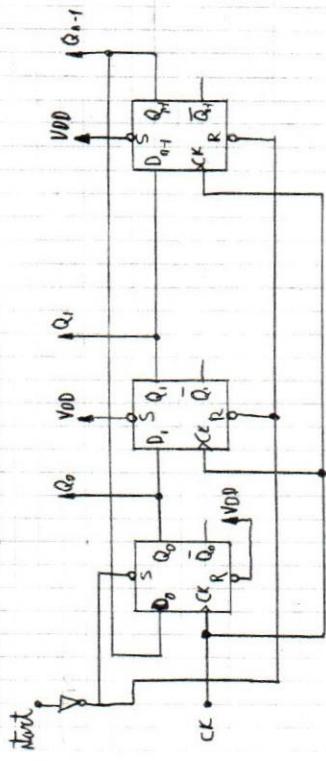
Qn <= not Q;

end process;

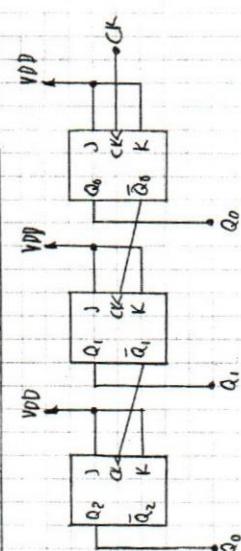
end FFD;

Contador

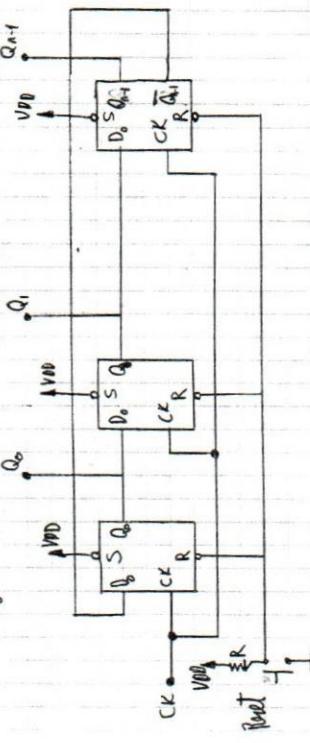
Contador Binario (univalue) N bits



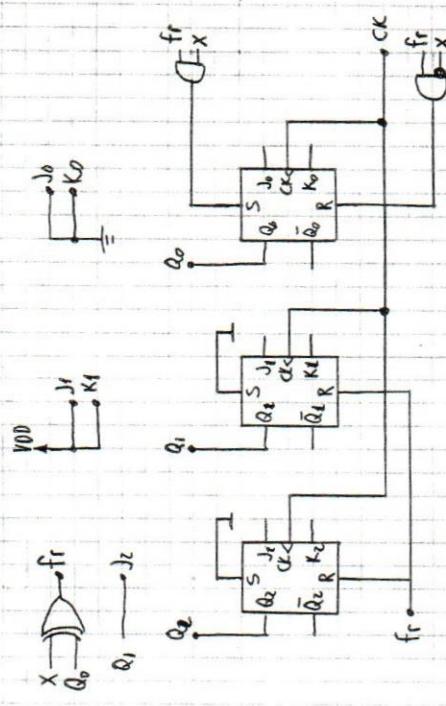
Contador Minimo Dependente



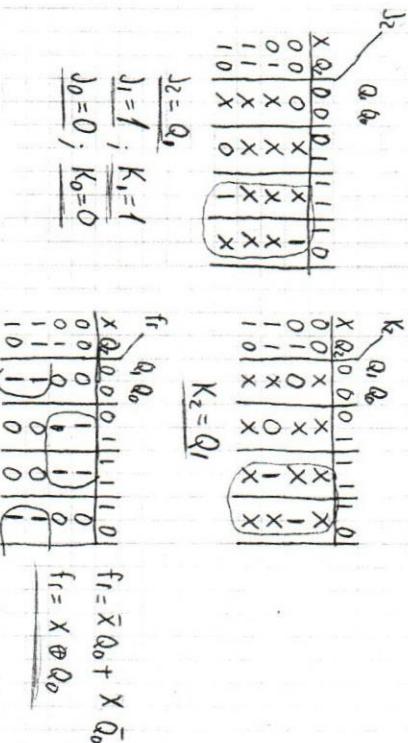
Contador Johnson (minicircular) N bits



(TP7)



$$\begin{array}{l} \text{X=0} \\ \text{X=1} \end{array} \Rightarrow \begin{array}{l} T_E = 1 \\ T_E = 0 \end{array}$$



	$t+1$	Q_4	Q_3	Q_2	Q_1	J_2	K_2	J_1	K_1	J_0	K_0	f_T	Reset
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	1	0
2	0	0	0	1	0	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0	0	0	0
4	0	0	1	0	1	0	0	0	0	0	0	0	0
5	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	1	0	0	1	0	0	0	0	0	0	0	0
7	0	1	0	1	0	0	0	0	0	0	0	0	0
8	0	1	0	1	1	0	0	0	0	0	0	0	0
9	0	1	1	0	0	0	0	0	0	0	0	0	0
10	0	1	1	0	0	0	0	0	0	0	0	0	0
11	0	1	1	0	1	0	0	0	0	0	0	0	0
12	0	1	1	1	0	0	0	0	0	0	0	0	0
13	0	1	1	1	1	0	0	0	0	0	0	0	0
14	1	0	0	0	0	0	0	0	0	0	0	0	0
15	1	0	0	0	1	0	0	0	0	0	0	0	0
16	1	0	0	1	0	0	0	0	0	0	0	0	0
17	1	0	1	0	0	0	0	0	0	0	0	0	0
18	1	0	1	0	1	0	0	0	0	0	0	0	0
19	1	0	1	1	0	0	0	0	0	0	0	0	0
20	1	1	0	0	0	0	0	0	0	0	0	0	0
21	1	1	0	0	1	0	0	0	0	0	0	0	0
22	1	1	0	1	0	0	0	0	0	0	0	0	0
23	1	1	1	0	0	0	0	0	0	0	0	0	0
24	1	1	1	0	1	0	0	0	0	0	0	0	0

TP7

Contador FF-JK de 4 bits en modo secuencia, regresa a cero al llegar a 15.

Contador

TP 7

⑦ Contador codifico gray de 4 bits usando FF-D. Obtener una salida cuando que presente el código BCD ($CJ=0x0$)

Tabla de estados FF-D

Q_3	Q_2	Q_1	Q_0	D
0	0	0	0	0
0	0	0	1	-
0	0	1	0	-

Tabla del trabajo del contador

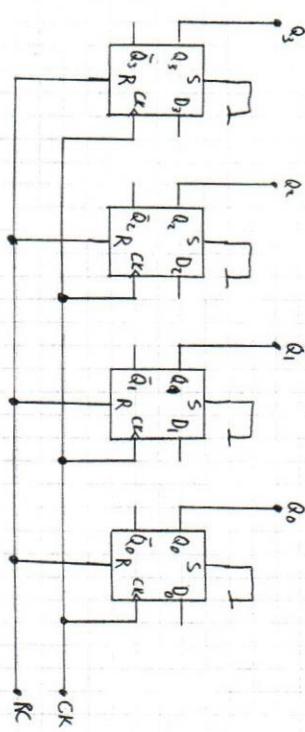
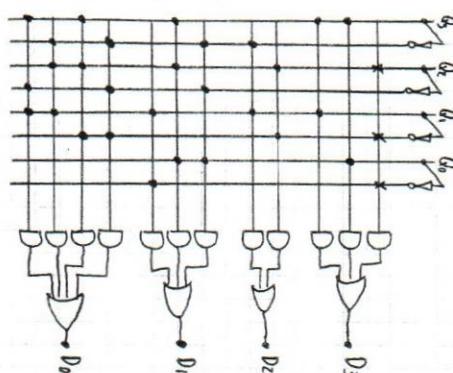
t	Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	D	$FF-D$	BCD
	Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	D	D_3	S_3
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	1	1	0
2	0	0	1	0	0	0	1	0	0	0	1
3	0	0	1	1	0	0	1	1	0	0	2
4	0	1	0	0	0	0	0	0	0	0	3
5	0	1	0	1	0	0	0	1	1	1	4
6	0	1	1	0	0	0	1	0	0	0	5
7	0	1	1	1	0	0	1	1	1	1	6
8	1	0	0	0	0	0	0	0	0	0	7
9	1	0	0	1	0	0	0	1	1	1	8
10	1	0	1	0	0	0	0	0	0	0	9
11	1	0	1	1	0	0	0	1	1	1	10
12	1	1	0	0	0	0	0	0	0	0	11
13	1	1	0	1	0	0	0	1	1	1	12
14	1	1	1	0	0	0	0	0	0	0	13
15	1	1	1	1	0	0	0	1	1	1	14

$$D_3 = Q_2 \bar{Q}_3 + Q_3 Q_0 + Q_3 Q_1 \\ D_2 = Q_2 \bar{Q}_1 + \bar{Q}_3 Q_1 \\ D_1 = \bar{Q}_3 \bar{Q}_2 \bar{Q}_1 + Q_2 Q_3 Q_1 + \bar{Q}_3 Q_2 Q_1 + Q_3 \bar{Q}_2 Q_1$$

$$D_0 = \bar{Q}_3 \bar{Q}_2 \bar{Q}_1 + Q_2 Q_3 Q_1 + \bar{Q}_3 Q_2 Q_1 + Q_3 \bar{Q}_2 Q_1$$

Contadores

Implementación



Contadores

(TP7)

VHDL : Contador BCD (modulo 10) de don Shagun

Entity BCD is
Port (CK, Clear, E: IN std_logic;
BCD0, BCD1: OUT std_logic_vector(3 downto 0));
end BCD;

Architecture contador of BCD is
Begin

Process (CK)

Begin
if (CK'event and (CK='1') then

BCD1 = "0000";
BCD2 <= "0000";

else (E = '1') then

if (BCD0 = "1001") then

BCD0 <= "0000";

if (BCD1 = "1001") then

BCD1 <= "0000";

else BCD1 <= BCD1 + 1;

end if;

else BCD0 <= BCD0 + 1;

end if;

and if;

and process;

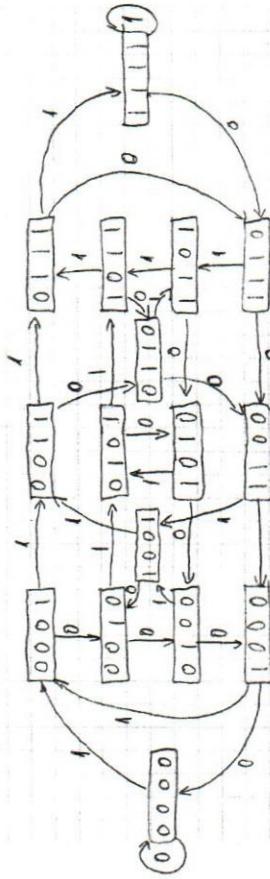
and contador;

(TP7)

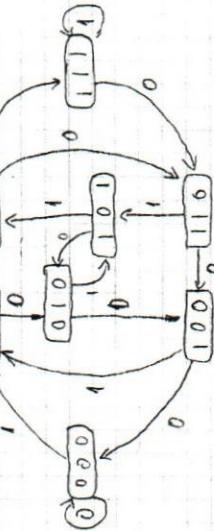
Mult register

Diagramme de Bruygn

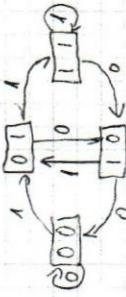
16 bits (entrée 16 nouveau), 4 entrée (y flut)



8 bits (entrée 8 nouveau), 3 entrée (3 bits)



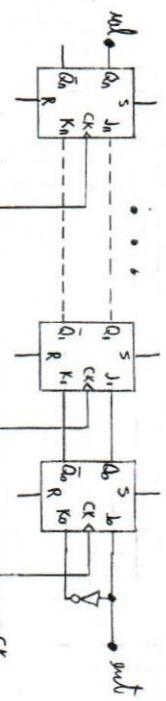
4 bits (entrée 4 nouveau), 2 entrée (2 bits)



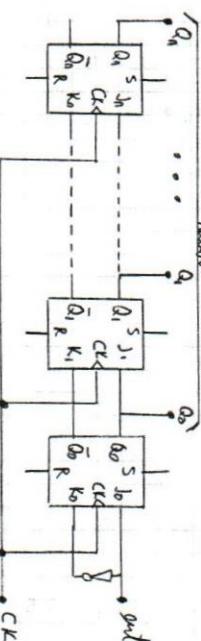
(TP7)

Shift register

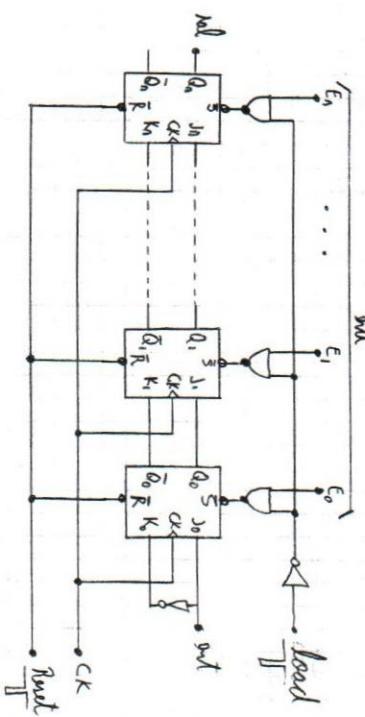
Entrada serie - valido nro



Entrada serie - valido nro



Entrada paralelo - valido nro



1. Reset
2. Load
3. CK

VHDL: Shift register entrada paralelo - valido nro

```

entity Shift4b is
    port( R : IN std_logic;        -- 4 dígitos 1;
          CK,L,WE : IN std_logic;  -- 4 dígitos 1;
          Q : OUT std_logic_vector(4 downto 1));
end;

```

and shift4b;
architecture shift of shift4b is
begin
process(CK,WE and CK='1')
begin
 if (L='1') then
 Q <= R;
 else
 Q(4) <= Q(4);
 Q(3) <= Q(3);
 Q(2) = Q(4);
 Q(1) = Q(4);
 Q(0) <= WE;
 end if;
end process;
end;

and present;

and shift;

TP7

Shift register

- ⑩ Contador modulo 12 con registro de desarrollo y logica de reajuste. Desarrollar la logica de desarrollo memoria una validada dentro.

Diagrama 16 gatillos

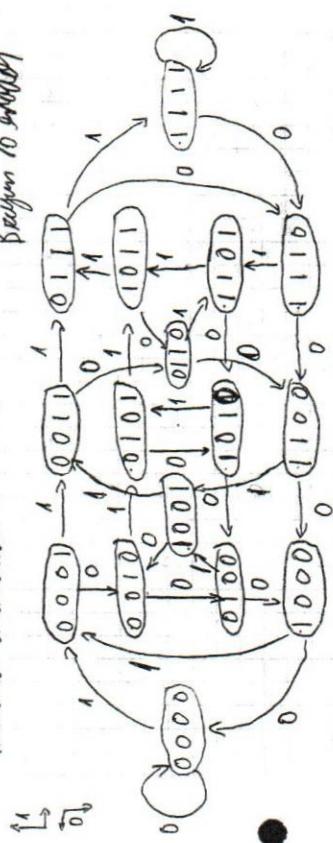


Diagrama de estados

$$\begin{array}{c} 0000 \rightarrow 0001 \rightarrow 0011 \rightarrow 0111 \rightarrow 1110 \rightarrow 1101 \\ \swarrow \quad \downarrow \quad \uparrow \quad \downarrow \quad \uparrow \\ 1000 \leftrightarrow 0110 \leftrightarrow 0101 \leftrightarrow 1011 \leftrightarrow 1010 \end{array}$$

Tabla de estados

Q ₃	Q ₂	Q ₁	Q ₀	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

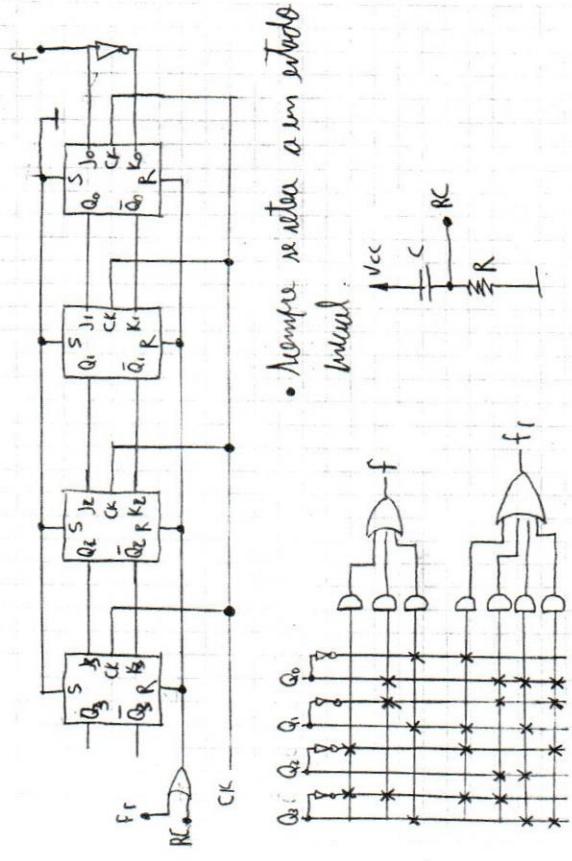
f: logica de restablecimiento
f_r: logica de reset (con pin valido restablecido)

Q ₃	Q ₂	Q ₁	Q ₀	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

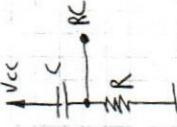
$$f = \bar{Q}_3 \bar{Q}_2 Q_1 \bar{Q}_0 + \bar{Q}_3 Q_2 \bar{Q}_1 Q_0 + Q_3 Q_2 Q_1 \bar{Q}_0 + Q_3 \bar{Q}_2 \bar{Q}_1 Q_0$$

$$f_r = \bar{Q}_3 \bar{Q}_2 Q_1 + \bar{Q}_3 Q_2 \bar{Q}_1 Q_0 + \bar{Q}_3 Q_2 Q_1 \bar{Q}_0 + Q_3 \bar{Q}_2 Q_1 Q_0$$

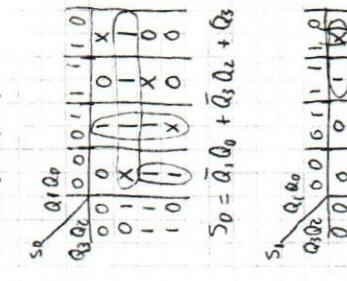
Diagrama de tiempos



• Necesita de tener a un ritmo de trabajo.



s: trabajo en desarrollo (señal válida en q)



$$f_r = \bar{Q}_3 \bar{Q}_2 Q_1 + \bar{Q}_3 Q_2 \bar{Q}_1 Q_0 + \bar{Q}_3 Q_2 Q_1 \bar{Q}_0 + Q_3 \bar{Q}_2 Q_1 Q_0$$

$$S_1 = \bar{Q}_3 \bar{Q}_2 Q_1 + \bar{Q}_3 Q_2 \bar{Q}_1 Q_0 + Q_3 \bar{Q}_2 Q_1 \bar{Q}_0$$

TP7

Multiregistro

(10)

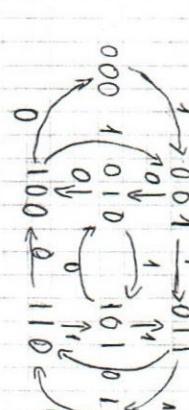
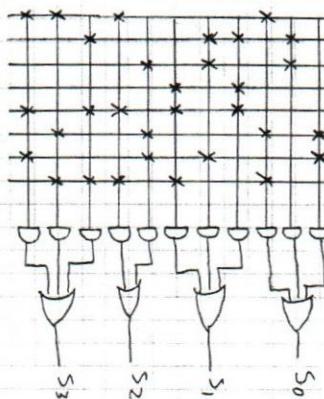
S_2	$Q_1 Q_0$	$Q_3 Q_2$	$Q_5 Q_4$	$Q_7 Q_6$
0 0 0	0 0 0	0 0 0	0 0 0	1 1 1
0 1 0	X	0 0 0	0 X 0	1 1 0
1 0 0	0	X	0 0 0	1 1 1
1 0 0	0	0	X	0 0 0

$$S_2 = Q_2 \bar{Q}_1 Q_0 + Q_3 \bar{Q}_2 \bar{Q}_1$$

Implementación - Registros decimales

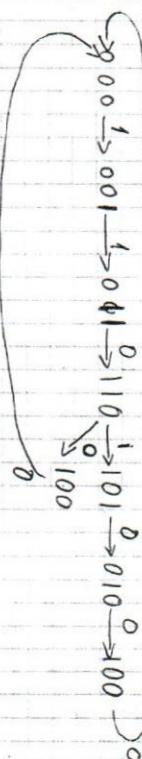
S_2	$Q_1 Q_0$	$Q_3 Q_2$	$Q_5 Q_4$	$Q_7 Q_6$
0 0 0	0 0 0	0 0 0	0 0 0	1 1 1
0 1 0	X	0 0 0	0 X 0	1 1 0
1 0 0	0	X	0 0 0	1 1 1
1 0 0	0	0	X	0 0 0

$$S_3 = \bar{Q}_3 Q_1 \bar{Q}_0 + Q_3 \bar{Q}_2 \bar{Q}_1 \bar{Q}_0 + Q_3 Q_2 Q_0$$



(14) Requerimientos de implementación de tres etapas para generar una secuencia de longitud 7, cuando el control $m=1$ y otra de longitud 5 cuando $m=0$. Utilizar un generador de registros de desplazamiento utilizando implementación con XOR para implementar los anteriores multiregistros.

Diagrama de estados



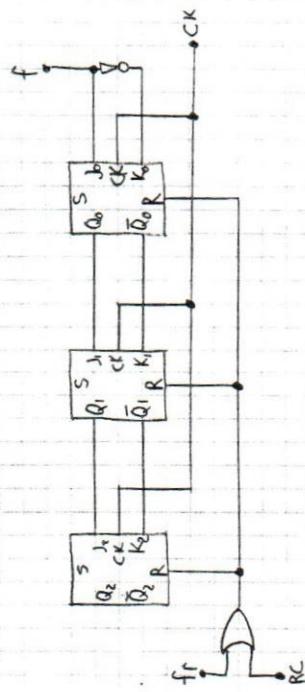
	Q_1	Q_0	f
0	0	0	
0	0	0	
0	0	0	
0	0	0	
0	0	0	
0	0	0	
0	0	0	

Tabla de trabajo

f: función rellamamiento
f_c: función reset

Múltiplos registradores

(14) Implementación



TP7

Múltiplos registradores

(14) Generar la secuencia binaria 0-1-0-0-1-0-1-1 usando registros de desplazamiento

Diagrama de Brayn-Sintesis

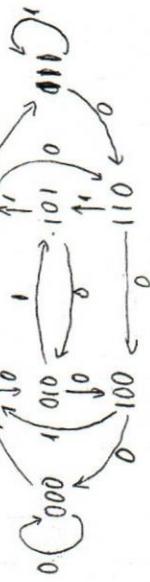


Diagrama de estados

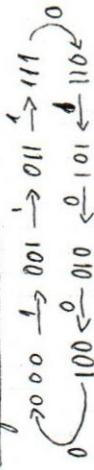


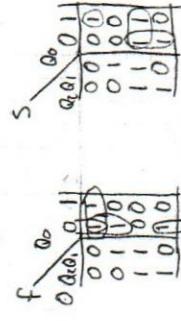
Tabla de estados

S ₂	S ₁	S ₀	f	S
0	0	0	1	0
0	0	1	-	0
0	1	-	0	0
1	-	0	0	0
1	-	1	0	0
-	0	0	0	0
-	0	1	0	0
-	1	0	0	0
-	1	1	0	0

• No hace falta función next (f_t) (usar solo los estados)

f: logica de redondeo hacia cero.

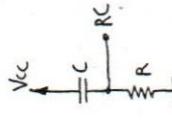
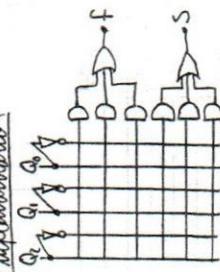
s: logica de la salida deseada



$$f = \bar{Q}_1 \bar{Q}_1 + \bar{Q}_2 \bar{Q}_0 + \bar{Q}_1 \bar{Q}_0$$

$$S = \bar{Q}_1 \bar{Q}_1 + Q_2 \bar{Q}_0 + Q_1 Q_1$$

Implementación



Múltiplos registradores

(14) Generar la secuencia binaria 0-1-0-0-1-0-1-1 usando registros de desplazamiento

Diagrama de Brayn-Sintesis

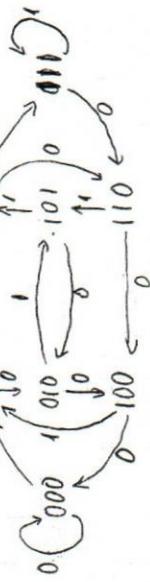


Diagrama de estados

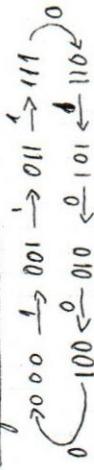


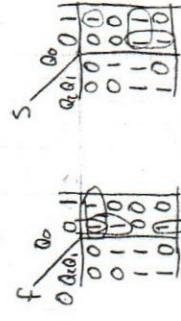
Tabla de estados

S ₂	S ₁	S ₀	f	S
0	0	0	1	0
0	0	1	-	0
0	1	-	0	0
1	-	0	0	0
1	-	1	0	0
-	0	0	0	0
-	0	1	0	0
-	1	0	0	0
-	1	1	0	0

• No hace falta función next (f_t) (usar solo los estados)

f: logica de redondeo hacia cero.

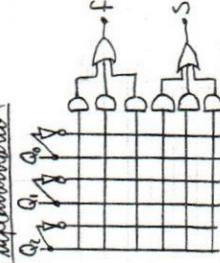
s: logica de la salida deseada



$$f = \bar{Q}_1 \bar{Q}_1 + \bar{Q}_2 \bar{Q}_0 + \bar{Q}_1 \bar{Q}_0$$

$$S = \bar{Q}_1 \bar{Q}_1 + Q_2 \bar{Q}_0 + Q_1 Q_1$$

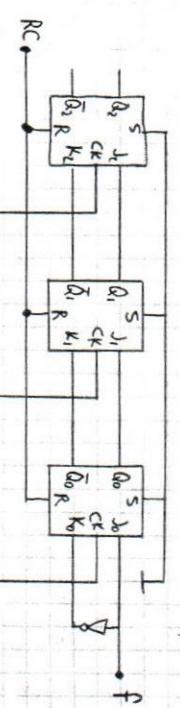
Implementación



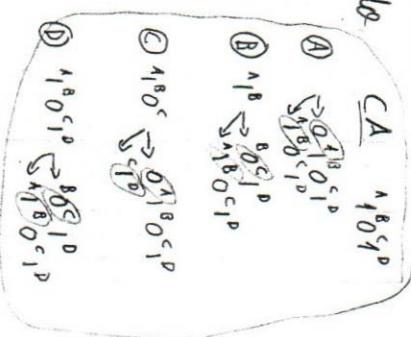
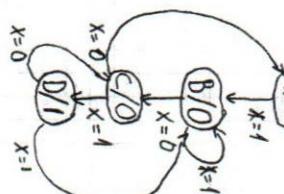
TP7

TP 7

Conjunto Reservado
Numeros - Moore



- El contador de reserva 101 nos muestra devolviendo una señal a uno CI: 00(A)
- La respuesta de estado



- Tabla de estados

X	t	t+1
0	A	A
0	B	C
0	C	A
0	D	B
1	A	B
1	B	D
1	C	C
1	D	B

← diferencia
con
Meady

- Tabla de estados FF-JK
- | A | J | K | Q ₀ |
|---|---|---|----------------|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

$$J = \bar{Q}_0 \quad K = Q_0 + \bar{X} \bar{Q}_0$$

- Tabla de estados
- | t | t+1 |
|---|---|
| X | Q ₀ Q ₁ Q ₂ Q ₃ |
| 0 | 0 0 0 0 |
| 0 | 0 0 0 1 |
| 0 | 0 0 1 0 |
| 0 | 0 0 1 1 |
| 0 | 0 1 0 0 |
| 0 | 0 1 0 1 |
| 0 | 0 1 1 0 |
| 0 | 0 1 1 1 |
| 1 | 1 0 0 0 |
| 1 | 1 0 0 1 |
| 1 | 1 0 1 0 |
| 1 | 1 0 1 1 |
| 1 | 1 1 0 0 |
| 1 | 1 1 0 1 |
| 1 | 1 1 1 0 |
| 1 | 1 1 1 1 |

$$\begin{array}{l} J = Q_0 \\ K = Q_0 + \bar{X} \bar{Q}_0 \end{array}$$

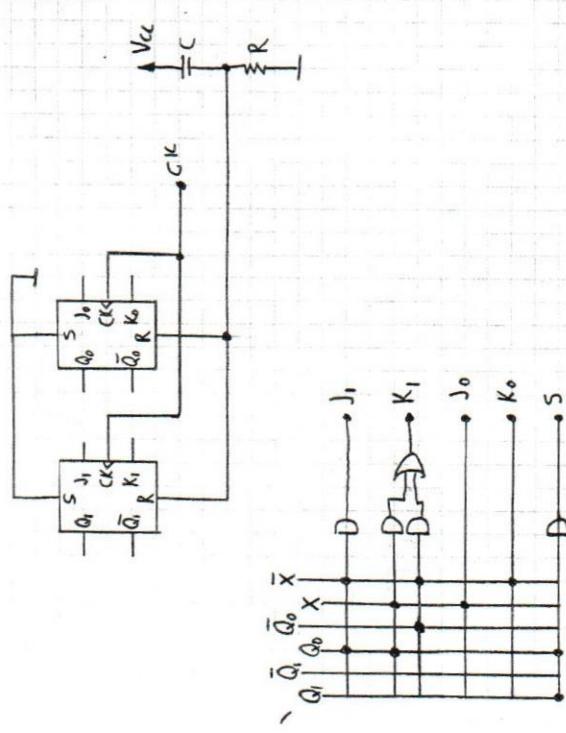
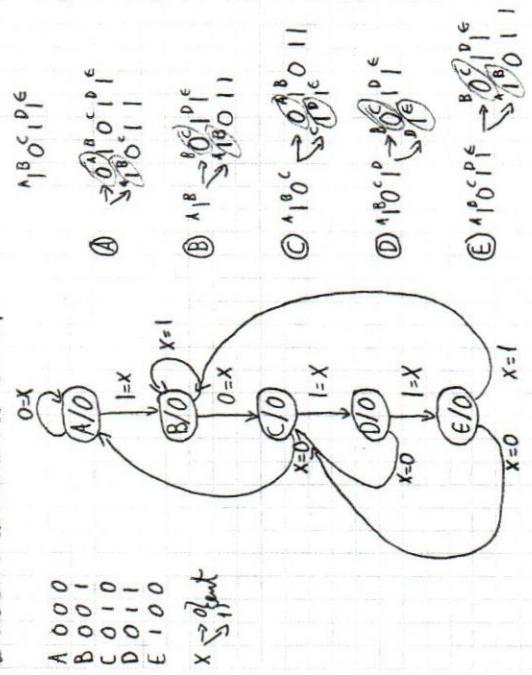
$$J_0 = X \quad ; \quad K_0 = \bar{X} \quad ; \quad S = Q_1 Q_0$$

- N° implementación

(TP7)

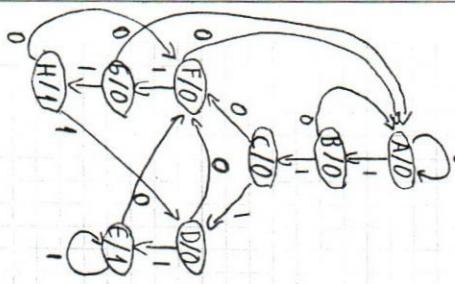
Cuadros Resueltos
Diseño - Moore

Detector de memoria 1011



↓ Estotor de numero 1111 0 11011

$$\left. \begin{array}{c} 1^B 1^C 0^F \\ 1^B 1^C 1^G \\ 1^B 1^C 1^H \end{array} \right\} \quad \left. \begin{array}{c} 1^B 1^C \\ 1^B 1^C \\ 1^B 1^C \end{array} \right\} \quad \left. \begin{array}{c} 1^B 1^C \\ 1^B 1^C \\ 1^B 1^C \end{array} \right\}$$



$$\textcircled{A} \xrightarrow{0^B 1^C} \textcircled{D} \textcircled{B} \textcircled{C}$$

$$\textcircled{B} \xrightarrow{1^B} \textcircled{D} \textcircled{A} \textcircled{C}$$

$$\textcircled{C} \xrightarrow{1^B 1^C} \textcircled{D} \textcircled{F} \textcircled{G}$$

$$\textcircled{D} \xrightarrow{1^B 1^C} \textcircled{E} \textcircled{F} \textcircled{G}$$

$$\textcircled{E} \xrightarrow{1^B 1^C 0^F} \textcircled{F} \textcircled{G}$$

$$\textcircled{F} \xrightarrow{1^B 1^C 1^G} \textcircled{G}$$

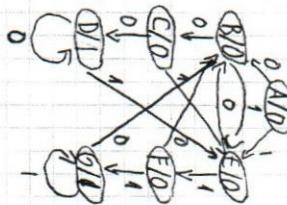
$$\textcircled{G} \xrightarrow{1^B 1^C 1^H} \textcircled{F} \textcircled{E}$$

$$\textcircled{H} \xrightarrow{1^B 1^C 0^F 1^G} \textcircled{F} \textcircled{E}$$

(TP7)

↓ Estotor de numero 1110 000

$$0^B 0^C 0^D \\ 1^E 1^F 1^G$$



$$\textcircled{A} \xrightarrow{0^B 0^C 0^D} \textcircled{E} \textcircled{F} \textcircled{G}$$

$$\textcircled{B} \xrightarrow{1^B} \textcircled{E} \textcircled{F} \textcircled{G}$$

$$\textcircled{C} \xrightarrow{1^B 0^C} \textcircled{E} \textcircled{F}$$

$$\textcircled{D} \xrightarrow{1^B 0^C 0^D} \textcircled{E} \textcircled{F}$$

$$\textcircled{E} \xrightarrow{1^E} \textcircled{B} \textcircled{C}$$

$$\textcircled{F} \xrightarrow{1^E 1^F} \textcircled{B} \textcircled{C}$$

$$\textcircled{G} \xrightarrow{1^E 1^F 1^G} \textcircled{B} \textcircled{C}$$

Unión numero - Moore
Número - Moore

(TP7)

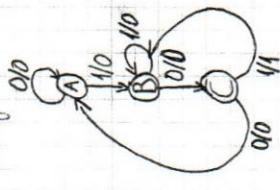
Contador numerico

• Z alfa de excitacion

t	Q ₁ Q ₀	t+1	Q ₁ Q ₀	K ₁	J ₀ K ₀	S
0	0 0	0 0	0 0	0 X	0 X	0
0	0 1	1 1	1 1	1 X	X -	0
0	1 1	0 0	X -	X -	X -	0
0	0 0	0 1	0 X	1 X	1 X	0
0	0 1	0 1	0 X	0 X	0 X	0
1	1 1	1 0	X -	X -	X -	0
1	1 0	0 1	0 X	0 X	0 X	-

Dector de numero 101 por Mealy obtenido una
salida a 1. CI: 00 (A)

• Diagrama de estados



- A $\xrightarrow{0} 01$
- B $\xrightarrow{1} 01$
- C $\xrightarrow{1} 10$
- D $\xrightarrow{1} 10$

• Z alfa de estados

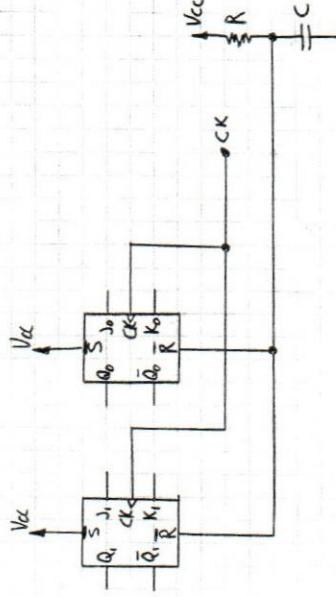
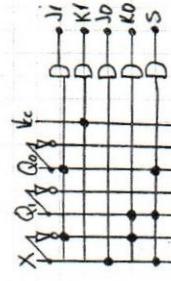
X	t	t+1	S	Q ₁	Q ₀
0	A	A	0	0	0
0	B	C	0	0	-
0	C	A	0	-	0
0	B	B	0	-	0
1	A	B	0	-	-
1	B	C	0	-	-
1	C	A	0	-	-
1	B	B	0	-	-

• Z alfa de estados FF-JK

t	Q ₁	Q ₀	J	K
0	0	0	0	X
0	0	-	0	X
0	-	0	0	X
1	-	0	0	X

t	Q ₁ Q ₀	t+1	Q ₁ Q ₀	K ₁	J ₀ K ₀	S
0	0 0	0 0	0 1	1 X	X -	0
0	0 1	1 1	1 1	1 X	X -	0
0	1 1	0 0	X -	X -	X -	0
0	0 0	0 1	0 X	1 X	1 X	0
0	0 1	0 1	0 X	0 X	0 X	-
1	1 1	1 0	X -	X -	X -	0
1	1 0	0 1	0 X	0 X	0 X	-

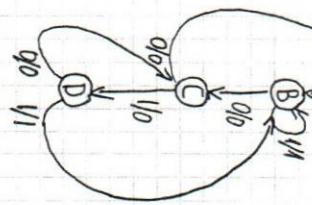
t	Q ₁ Q ₀	t+1	Q ₁ Q ₀	K ₁	J ₀ K ₀	S
0	0 0	0 0	0 1	1 X	X -	0
0	0 1	1 1	1 1	1 X	X -	0
0	1 1	0 0	X -	X -	X -	0
0	0 0	0 1	0 X	1 X	1 X	0
0	0 1	0 1	0 X	0 X	0 X	-
1	1 1	1 0	X -	X -	X -	0
1	1 0	0 1	0 X	0 X	0 X	-



(TP7)

Comutación secuencial
Numeros - Mealy

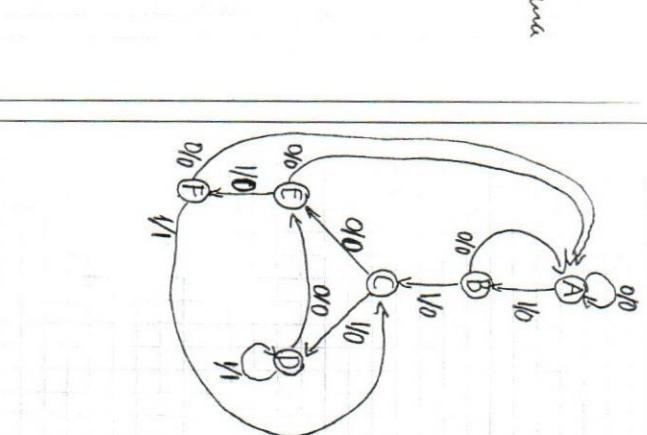
D) Estotor de memoria 1011 por Mealy ademas una
salida a 1. CI: 00 (A)



- A $0^B 1^C 0^D 1^E$
- B $1^B 0^C 0^D 1^E$
- C $1^B 1^C 0^D 1^E$
- D $0^B 0^C 1^D 1^E$

$1^B 0^C 0^D 1^E$

A 00
B 01
C 10
D 11



- A $0^B 1^C 0^D 1^E 1^F$
- B $1^B 0^C 0^D 1^E 1^F$
- C $1^B 1^C 0^D 1^E 1^F$
- D $0^B 0^C 1^D 1^E 1^F$
- E $1^B 1^C 0^D 1^E 1^F$
- F $1^B 1^C 1^D 0^E 1^F$

$1^B 1^C 0^D 1^E 1^F$

D) Estotor de memoria 1111 o 11011 por Mealy

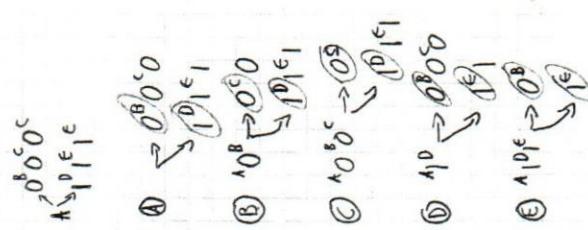
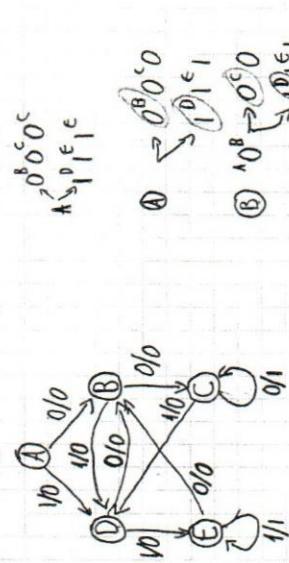
$1^B 1^C 1^D 1^E 1^F$



TP 7

Autómato Mealy

Estado de secuencia 000 o 111 por Mealy



Guia Auxiliar Problemas de autómatas.

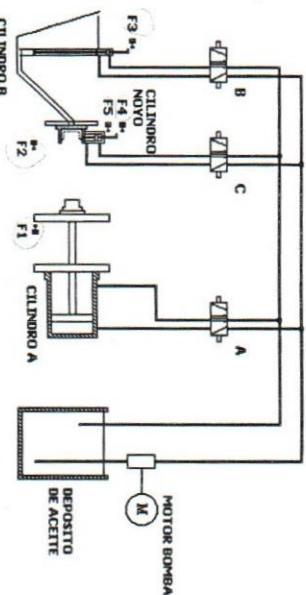
- 1- Una puerta se abre al activar un pulsador P. La apertura se produce hasta que alcanza el topo de apertura detectado por el sensor A. A partir de ese momento se produce el cierre de la puerta, hasta alcanzar el topo de cierre detectado por el sensor C, y en ese momento se produce la parada.
- 2- Diseñar el circuito correspondiente. Describir en VHDL. La puerta se controla por dos salidas, S1 Y S0 cuando S1=1 la puerta se abre, cuando S0=1 la puerta se cierra. Diseñar el automata, dibujar el circuito y describir en VHDL.
- 3- Un limpiaparabrisas tiene pulsadores de marcha y parada- M Y P. Si se pulsa M se activa, si se pulsa P se para cuando se detecta el sensor de reposo R en uno. Esto hace que el sistema pare siempre en el mismo lugar. Diseñar el autómata de Mealy que resuelva el problema, dibujar el circuito y describir en VHDL.
- 4- Hay que controlar las puertas de cristal de un edificio. Disponemos de un detector de personas P, de un detector de puerta abierta A y de un detector de puerta cerrada C. Si viene una persona se abre la puerta. Si la puerta empieza a abrirse, completa su ciclo entero aunque desaparece la persona. Si sigue apareciendo gente la puerta permanece abierta continuamente. Si cuando estaba cerrándose la puerta aparece una persona, la puerta vuelve a abrirse, completando un nuevo ciclo. Diseñar el autómata, dibujar el circuito y describir en VHDL.

Estado final
Punto de salida

Ejercicio del seminario de la guia 7 este es el
ejercicio 125 de la carpeta de 1º año

Juguetón 2

Ejercicio 1
Prensa de inyección de plásticos.



El ciclo que debe realizar la máquina, al encender el motor de la bomba con el interruptor 10, es el siguiente:

Cerrar el molde mediante el avance del cilindro A.

Dosificar el plástico fundido a inyectar, mediante la subida del cilindro B.

Inyectar el plástico mediante la bajada del B.

Realizar una pausa para permitir que el aire salga del molde.

Introducir los hoyos, para configurar la pieza, mediante la bajada de C.

Realizar una pausa para permitir la solidificación.

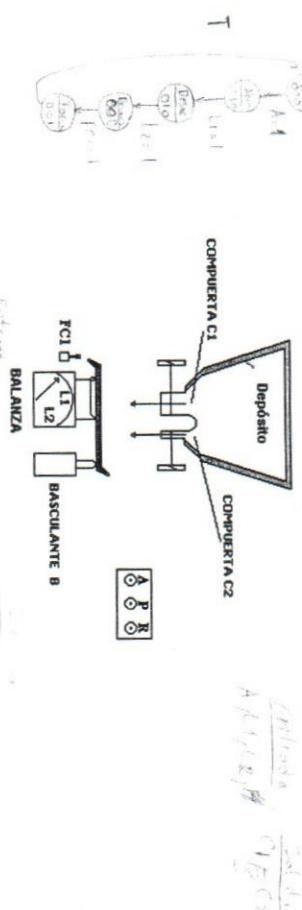
Abrir el molde.

Realizar una pausa para permitir la solidificación.

Abrir el noyo.

Abrir el molde, extraer la pieza, y recomenzar el ciclo.

Ejercicio 2
Pesado preciso de sustancias.



Al pulsar el botón de arranque A se abren las dos compuertas C1 y C2.

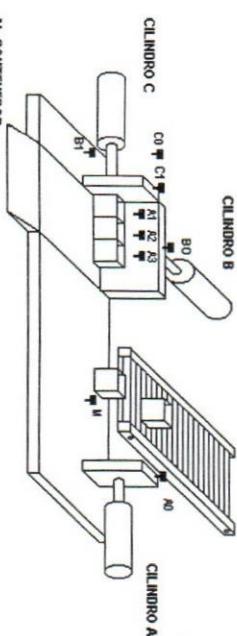
1



Cuando la aguja de la balanza llegue a L1 se desactiva C1. Al llegar a L2 se desactiva C2 (Compuerta de atmósfera). Despues del pesaje se vacía la balanza por medio del basculante B. Al volver a la posición de reposo, no debe afectar el paso de la aguja por L1.

Ejercicio 3

Controlar una apiladora. Las piezas a apilar llegan desde una cinta transportadora y son detectadas por un sensor M. Para la primera pieza detectada, el vástago del cilindro A avanza hasta el fin de carrera A1 y luego retrocede hasta A0. Para la segunda y tercera pieza, el movimiento de A es similar al anterior, sólo que avanza hasta los fines de carrera A2 y A3, respectivamente.



AL CONTINUAR

Una vez apiladas las tres piezas, y después del retroceso de A, el vástago del cilindro C retrocede hasta C0. En este momento avanza el vástago del cilindro B hasta B1 y luego retrocede a B0.

A continuación se regresa a la posición inicial avanzando C hasta C1 y terminando el ciclo. A partir de este momento se podrá iniciar un nuevo ciclo con la llegada de nuevas piezas.

Los avances de A sólo se harán cuando esté activado el sensor M y el fin de carrera A0.

Ejercicio 4

Diseñar un automata de Moore que implemente un contador modulo 4 reversible según la linea de control E0; si vale 1 es ascendente, si vale 0 es descendente. Cuando se produzca el paso de un modo a otro se empezará por el primer estado de la nueva secuencia.

Ejercicio 5

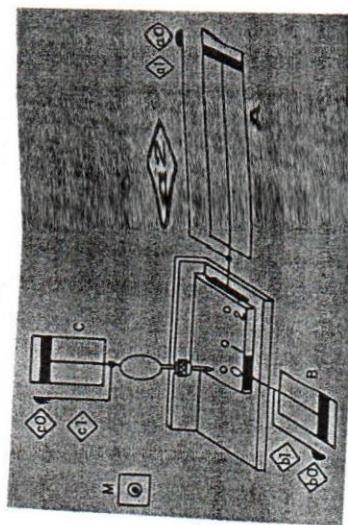
Un secador de pelo, S, en un vestuario tiene un pulsador P, y un detector de persona bajo el secador, D. Cuando se pulsa P (P=1) y hay alguien debajo (D=1), el secador se activa (S=0), y lo seguirá haciendo mientras haya alguien debajo, aunque se suelte el pulsador. El secador se para cuando no hay nadie debajo, controlar la actividad del secador.

2



Ejercicio 6

Se necesita realizar tres perforaciones en una pieza de metal, las distancias entre las perforaciones se encuentra predefinida. Se cuenta con una entrada M de círculo, el operario **cuenta** coloca la pieza y presiona el botón, en ese momento se acciona el cilindro A que **leva** la pieza hasta accionar el final de carrera A0, luego se cierra el cilindro B, y se realiza la perforación, luego se libera B y se desplaza A hasta al1, se repite el ciclo hasta **repetir** las tres perforaciones.



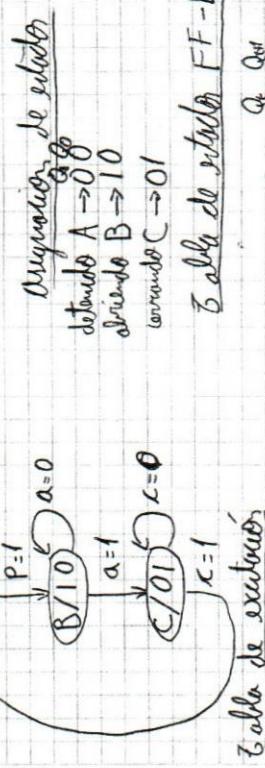
Funcionamiento de automatas

① Una puerta se abre al activar un pulsador P. La puerta se cierra al producirse hasta que alcanza el tope de apertura detectado por el sensor A. Al retirar de ese momento se produce el cierre de la puerta hasta alcanzar el tope de cierre de la puerta C y en ese momento se produce la parada.

Diseñar el circuito correspondiente. De acuerdo VHDL. Se pone el control de los dos valvulas, S1 y S0 cuando S1=1 la puerta se abre, cuando S0=1 la puerta se cierra. Diseñar el automata, dibujar el circuito y escribir en VHDL.

Por Moore
Estado / S1 S0

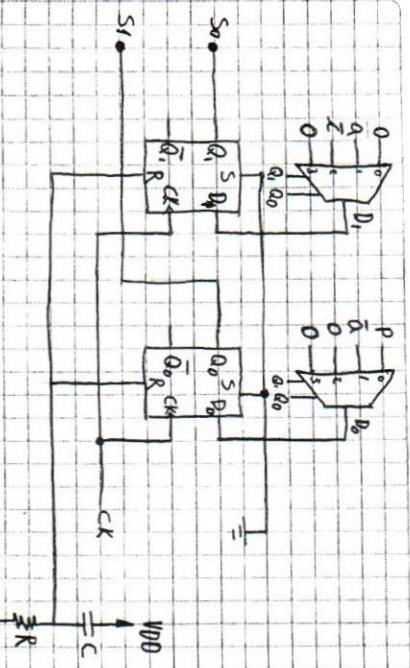
P = pulsador abierto
a = tope apertura
c = tope cierre



Q ₀	Q ₁	D
0	0	0
0	1	-
1	0	-
1	1	0

Q ₀	Q ₁	D
0	0	0
0	1	-
1	0	-
1	1	0

- Implementar con FF-D se hace mas facil con quel numero variable son C/FF, y la logica de realmente es mas rapido con MUX



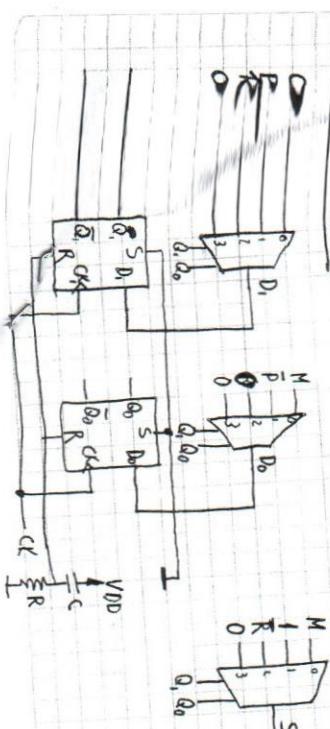
②

M = marcha } pulsador
P = parada } pulsador
R: reloj \rightarrow reloj

Ingresos de estado (Mealy)

Ingresos de estados

Estado
detenido 0 0
funciona 0 1
reloj 1 0
Cola de estados FF-D



M P R	Q1	Q2	D1	S
0 X X	0 0	0 0	0 0	0
1 X X	0 0	0 0	0 1	1
X 0 X	0 1	0 1	0 1	0
X 1 X	0 1	1 0	1 0	1
X X 0	1 0	0 0	0 0	0
X X 1	1 0	0 0	0 0	0

Q1	Q2	D
0	0	0
0	1	1
1	0	0
1	1	1

Funcionamiento de automata

Guia auxiliar de autores

④ Entradas

E_1	E_2		A	B
0	0	X	0	0
0	1	woto	0	-
1	0	X	-	0
1	1	Largo	-	-

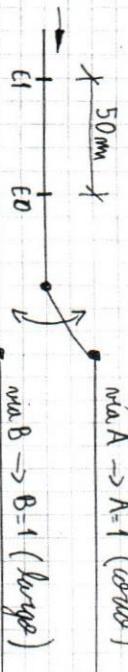


Diagrama de estado (Mealy)

$E_1 E_2 / A \quad B$

Diagrama de estado

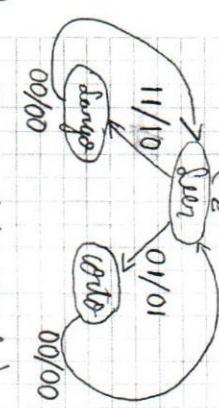


Tabla de estados (Mealy)

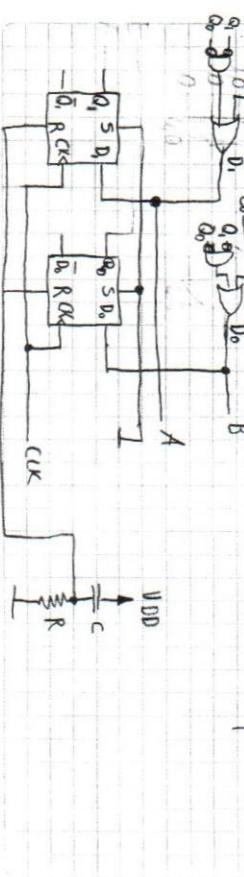
Tabla de estados FF-D

E_1	E_2	t^n	D	$D_i = A$	$D_i = B$
0	0	00	00	0	0
0	1	00	01	0	1
1	0	00	01	1	0
1	1	10	10	1	0
0	0	01	00	0	0
0	1	01	00	0	0
0	0	00	00	0	0

```

*****HEALY STATE MACHINE*****
architecture behavioral of nombre_de_la_entidad is
begin
    type state_type is (st1 <name_state>, st2 <name_state>, ...);
    declare
        los nombres de los estados, st1_estado1...
        signal state, next_state : state_type;
        signal <output>_i : type;
        declaran las salidas con el mismo nombre que la declarada en la
        pero agregandole _1, salida_1
    begin
        SYNC PROC: process (<clock>)
            que define la parte secuencial y su lista de sensibilidad
            begin
                if (<clock>'event and <clock> = '1') then
                    reset son variables declaradas con el mismo nombre en la entidad
                    cumple la condicion reset se fuerza al estado presente (state) a un
                    state <= st1_name.state;
                    inicial y la salida a un valor inicial
                    <output> <= '0';
                    cumple reset la maquina de estado avanza a un siguiente estado
                    actualiza la salida fisica (salida declarada en la entidad)
                    state <= next.state;
                    <output> <= <output>_i;
                    end if;
                else
                    <output> <= '0';
                    end if;
                end process;
            end if;
            else
                cuando reset la maquina de estado avanza a un siguiente estado
                actualiza la salida fisica (salida declarada en la entidad)
                state <= next.state;
                <output> <= <output>_i;
                end if;
            end process;
        end process;
    end process;
    que define la SALIDA de acuerdo al valor de ESTADO/ENTRADA y por eso
    lista de sensibilidad depende tanto de el estado de la maquina como de
    entradas al mismo tiempo. Situacion que diferencia a Mealy de Moore
    begin
        if (state = st3_name) and <input> = '1' then
            condicion de estado/entrada
            <output>_i <= '1';
        else
            <output>_i <= '0';
        end if;
    end process;
    que define la logica de la TRANSICION
    begin
        next_state <= state;
        case (state) is
            when st1_name =>
                if <input>_1 = '1' then
                    next_state <= st12_name;
                end if;
            when st2_name =>
                if <input>_2 = '1' then
                    next_state <= st3_name;
                end if;
            when st3_name =>
                next_state <= st1_name;
            when others =>
                next_state <= st1_name;
        end case;
    end process;

```



```

***** Automatas guia 1 - Ejercicio 1 (Por Moore) *****
***** STATE MACHINE *****
end behavioral;

***** MOORE STATE MACHINE *****
architecture behavioral of nombre_de_la_entidad is
type state_type is (st1-<name_state>, st2-<name_state>, ...);
signal state, next_state : state_type;
signal <output>i : std_logic;

SYNc_PROC: process (<clock>)
begin
  if (<clock>'event and <clock> = '1') then
    if (reset = '1') then
      state <= st1-<name_state>;
      <outputs> <= "0";
    else
      state <= next_state;
      <outputs> <= <output>i;
    end if;
  end if;
end process;

OUTPUT_DECODE: process (state)
begin
  -- que su
  -- que la maquina, situacion que
  -- difiere a Moore de Mealy. Ver que la salida se actualiza solo de
  -- acuerdo al valor del estado.
  -- <output>i <= '1';
  -- <output>i <= '0';
end process;

NEXT_STATE_DECODE: process (state, <input1>, <input2>, ...)

begin
  next_state <= state;
  case (state) is
    when st1-<name> =>
      if <input_1> = '1' then
        next_state <= st2-<name>;
      end if;
    when st2-<name> =>
      if <input_2> = '1' then
        next_state <= st3-<name>;
      end if;
    when st3-<name> =>
      next_state <= st1-<name>;
      when others =>
        next_state <= st1-<name>;
  end case;
end process;
end behavioral;

```

entity ejercicioautomata is
 port(P, A, C, reset, clk: in std_logic;
 S_out : out std_logic_vector(1 downto 0));
end ejercicioautomata;

architecture Behavioral of ejercicioautomata is
 -- This is a sample state-machine using enumerated types,
 -- This will allow the synthesis tool to select the appropriate
 -- encoding style and will make the code more readable.

--Insert the following in the architecture before the begin keyword
 -- Use descriptive names for the states, like st1_reset, st2_search
 type state_type is (st1_detenido, st2_abriendo, st3_cerrando);
 signal state, next_state : state_type;
 -- Declare internal signals for all outputs of the state-machine
 signal S1 : std_logic_vector(1 downto 0); -- example output signal
 -- other outputs

--Insert the following in the architecture after the begin keyword
 SYNC_PROC: process (clk)
begin
 if (clk'event and clk = '1') then
 if (reset = '1') then
 state <= st1_detenido;
 S <= "00";
 else
 state <= next_state;
 S <= S_i;
 end if;
 end if;
end process;

--MOORE State-Machine - Outputs based on state only
 OUTPUT_DECODE: process (state)
begin
 -- insert statements to decode internal output signals
 -- below is simple example
 if state = st1_detenido then
 S1 <= "00";
 elsif state = st2_abriendo then
 S1 <= "10";
 else
 S1 <= "01";
 end if;
end process;

NEXT_STATE_DECODE: process (state, P, A, C)
begin
 -- declare default state for next_state to avoid latches
 next_state <= state; --default is to stay in current state
 -- insert statements to decode next_state
 -- below is a simple example
 case (state) is

```

when st1_detenido =>
  **** Autonoma 1 - Ejercicio 2 **** (Por Mealy)
  ****
  if P = '1' then
    next_state <= st2_abriendo;
  elseif P = '0' then
    next_state <= st1_detenido;
  end if;

when st2_abriendo =>
  if A = '1' then
    next_state <= st3_cerrando;
  elseif A = '0' then
    next_state <= st2_abriendo;
  end if;

when st3_cerrando =>
  if C = '1' then
    next_state <= st1_detenido;
  elseif C = '0' then
    next_state <= st3_cerrando;
  end if;

when others =>
  next_state <= st1_detenido;
end case;
end process;

entity limpiaparabrisas is
  port(M, P, R, CLK, reset: in std_logic;
       S: out std_logic);
end limpiaparabrisas;

architecture behavioral of limpiaparabrisas is
begin
  SYNC_PROC: process (<clock>)
    begin
      if (CLK'event and CLK = '1') then
        if (reset = '1') then
          state <= st1_detenido;
          S <= '0';
        else
          state <= next_state;
          S <= S_i;
        end if;
      end if;
    end process;

OUTPUT_DECODE: process (state, M, P, R)
begin
  if (state = st1_detenido and M = '1') then
    S_i <= '1';
  elseif (state = st1_detenido and M = '0') then
    S_i <= '0';
  elseif (state = st2_funciona and P = '1') then
    S_i <= '1';
  elseif (state = st2_funciona and P = '0') then
    S_i <= '1';
  elseif (state = st3_reposo and R = '1') then
    S_i <= '0';
  elseif (state = st3_reposo and R = '0') then
    S_i <= '1';
  end if;
end process;

NEXT_STATE_DECODE: process (state, M, P, R)
begin
  next_state <= state;
  case (state) is
    when st1_detenido =>
      if M = '1' then
        next_state <= st2_funciona;
      elseif M = '0' then
        next_state <= st1_detenido;
    end if;

    when st2_funciona =>
      if P = '1' then
        next_state <= st3_reposo;
      elseif P = '0' then
        next_state <= st2_funciona;
    end if;
  end case;
end process;

```

```

***** Automatas guia 1 - Ejercicio 3 (Por Mealy) *****
***** Automatas guia 1 - Ejercicio 3 (Por Mealy) *****

library IEEE;
use librerias_a_usar.ALL;
***

entity puertacristalmealy is
port(P, A, C, reset, CLK: in STD_LOGIC;
      S: out STD_LOGIC_VECTOR(1 downto 0));
end puertacristalmealy;

architecture mealy of puertacristalmealy is
type state_type is (st1_cerrado, st2_abriendo, st3_cerrando);
signal state, next_state : state_type;
signal Si : STD_LOGIC_VECTOR(1 downto 0);

begin
puertacristalmealy:
process(CLK)
begin
if (CLK'event and CLK = '1') then
  if (reset = '1') then
    state <= st1_cerrado;
    S <= "00";
  else
    state <= next_state;
    S <= Si;
  end if;
end if;
end process;

SYNC_PROC: process(CLK)
begin
if (CLK'event and CLK = '1') then
  if (reset = '1') then
    state <= st1_cerrado;
    S <= "00";
  else
    state <= next_state;
    S <= Si;
  end if;
end if;
end process;

OUTPUT_DECODE: process(state, P, A, C)
begin
if (state = st1_cerrado and P = '1') then
  Si <= "01";
else
  Si <= "00";
end if;
if (state = st2_abriendo and P = '0') then
  Si <= "10";
else
  Si <= "01";
end if;
if (state = st3_cerrando and P = '1') then
  Si <= "01";
else
  Si <= "00";
end if;
if (state = st3_cerrando and P = '0' and C = '1') then
  Si <= "10";
else
  Si <= "10";
end if;
end process;

NEXT_STATE_DECODE: process(state, P, A, C)
begin
next_state <= state;
case (state) is
when st1_cerrado =>
  if P = '1' then
    next_state <= st2_abriendo;
  elsif P = '0' then
    next_state <= st1_cerrado;
  end if;
when st2_abriendo =>
  if A = '1' then
    next_state <= st3_cerrando;
  elsif A = '0' then
    next_state <= st2_abriendo;
  end if;
end case;
end process;

```

```

when st3_cerrando =>
    when others =>
        next_state <= st2_abriendo;
    end case;
end process;

entity pueracristalmoore is
    port(P, A, C, reset, CLK: in std_logic;
         S: out std_logic_vector(1 downto 0));
end pueracristalmoore;

architecture moore of pueracristalmoore is
type state_type is (st1_detenido, st2_apertura, st3_cierre);
signal state, next_state : state_type;
signal S_i : std_logic_vector(1 downto 0);

begin
    process(CLK)
    begin
        if (CLK='0' and CLK = '1') then
            if (reset = '1') then
                state <= st1_detenido;
            else
                state <= next_state;
                S_i <= S_i;
            end if;
        end process;
    end process;

    process(state)
    begin
        if state = st1_detenido then
            S_i <= "00";
        elsif state = st2_apertura then
            S_i <= "01";
        elsif state = st3_cierre then
            S_i <= "10";
        end if;
    end process;
end architecture;

process(state)
begin
    if state = st1_detenido then
        next_state <= st2_apertura;
    elsif state = st2_apertura then
        next_state <= st1_detenido;
    end if;
    when st2_apertura =>
        if A = '1' then
            next_state <= st3_cierre;
        else
            next_state <= st2_apertura;
        end if;
    when st3_cierre =>
        if A = '1' then
            next_state <= st1_detenido;
        else
            next_state <= st2_apertura;
        end if;
    end case;
end behavioral;

```

```

***** Automatas guia 1 - Ejercicio 4 (Por Mealy)*****
library IEEE;
use librerias_a_usar.ALL;
...

entity trenaes is
    port(E: in STD_logic; --vector (1 downto 0);
         clock, reset: in STD_logic;
         S: out STD_logic_vector (1 downto 0));
end trenaes;

architecture mealy of trenaes is

type state_type is (st1_leer, st2_largo, st3_corto);

signal state, next_state : state_type;
signal S1 : STD_logic_vector (1 downto 0);

begin

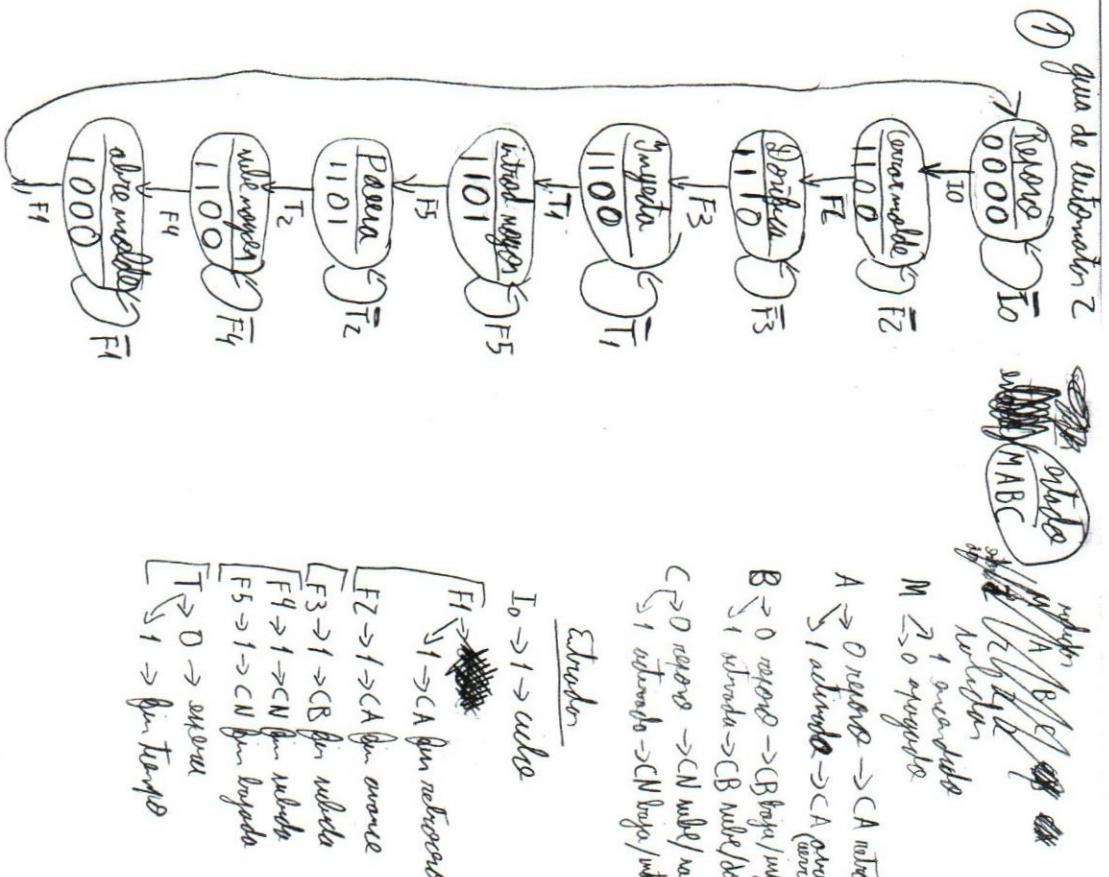
SYNC_PROC: process (clock)
begin
    if (clock'event and clock = '1') then
        if (reset = '1') then
            state <= st1_leer;
            S <= "00";
        else
            state <= next_state;
            S <= S1;
        end if;
    end if;
end process;

OUTPUT_DECODE: process (state, E)
begin
    if (state = st1_leer and E = "00") then
        S1 <= "00";
    elsif (state = st1_leer and E = "01") then
        S1 <= "01";
    elsif (state = st1_leer and E = "11") then
        S1 <= "10";
    elsif (state = st2_largo and E = "00") then
        S1 <= "00";
    elsif (state = st3_corto and E = "00") then
        S1 <= "00";
    end if;
end process;

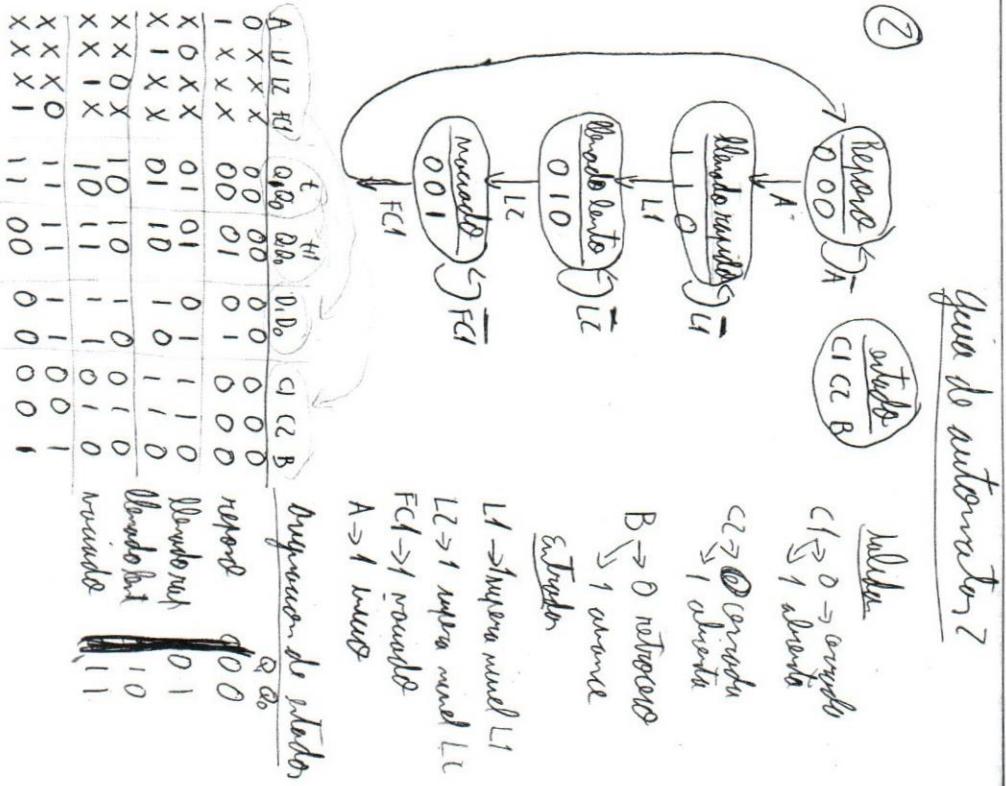
NEXT_STATE_DECODE: process (state, E)
begin
    next_state <= state; -
    case (state) is
        when st1_leer =>
            if E = "11" then
                next_state <= st2_largo;
            elsif E = "00" then
                next_state <= st1_leer;
            end if;
        when st2_largo =>
            if E = "00" then
                next_state <= st1_leer;
            end if;
        when st3_corto =>

```

Guia de Automotor 2



Guia de automotor 2



JK

Tabelle verstaet				
S	R	J	K	Q _{t+1}
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
1	0	0	0	1
1	0	0	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Tabelle verstaet		
Q _t	D _{t+1}	D
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Tabelle verstaet	
Q _t	Q _{t+1}
0	0
0	1
1	0
1	1

gute Automaten 2

