Práctica I
Contador de cinco bits

A modo de ejemplo, vamos a mostrar como se llevaría la práctica que tendría el siguiente enunciado:

Llevar a cabo el diseño de un contador de cinco bits, en el que se deberán realizar las siguientes tareas:

- Modelar en VHDL el contador de cinco bits.
- Comprobar mediante simulación el comportamiento del contador utilizando el software de XILINX.
- Utilizar el software de XILINX para sintetizar el comportamiento modelado en el primer punto (utilizar el mapeado tecnológico).
- Simular eléctricamente los bloques obtenidos en el punto anterior utilizando una familia lógica CMOS. Para ello, se debe realizar una revisión bibliográfica de la familia lógica en cuestión. Como resultado de dicha simulación se deben obtener las dimensiones óptimas de las puertas en términos de retraso de propagación (obtenido como el valor medio de los retrasos correspondientes a todas las combinaciones de entrada). Dichas dimensiones deben estar de acuerdo con las reglas de diseño de la tecnología CMOS de 0.6u.
- Obtener el layout de los bloques anteriores según las dimensiones óptimas del apartado anterior.
- Simular eléctricamente los flip-flops, utilización una configuración maestro-esclavo estándar. Para ello, se debe realizar una revisión bibliográfica de la configuración en cuestión. Como resultado de dicha simulación se deben obtener las dimensiones óptimas de las puertas en términos de retraso de propagación (obtenido como el valor medio de los retrasos correspondientes a todas las transiciones clk -> salida). Dichas dimensiones deben estar de acuerdo con las reglas de diseño de la tecnología CMOS de 0.6u.
- Obtener el layout de los flip-flops según las dimensiones óptimas del apartado anterior.
- Obtener una colocación óptima de los bloques obtenidos en el tercer punto, utilizando el algoritmo de clustering uniforme. Para ello, se debe realizar una revisión bibliográfica del algoritmo en cuestión, así como una implementación software del algoritmo en el lenguaje Tcl/Tk (basado en script, y por tanto, independiente del sistema operativo).
- Obtener un rutado óptimo utilizando el algoritmo basado en cliques. Para ello, se debe realizar una revisión bibliográfica del algoritmo en cuestión, así como una implementación software del algoritmo en el lenguaje Tcl/Tk (basado en script, y por tanto, independiente del sistema operativo).

1.1. Modelar en VHDL el contador de cinco bits

Como no se nos indica ninguna especificaicón sobre el contador que se desea implementar, vamos a permitir que el software de XILINX haga todo el trabajo estructural por nosotros. Por lo tanto, el código VHDL del contador sería el siguiente:

```
----------------------------------------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity contador5 is
   Port ( clk : in  STD_LOGIC;
          cuenta : out  STD_LOGIC_VECTOR (4 downto 0));
end contador5;

architecture Behavioral of contador5 is
        signal cuenta_int    : std_logic_vector(4 downto 0) := (others => '0');
begin

        P1: process (clk)
        begin
                if (clk = '1' and clk'event) then cuenta_int <= cuenta_int+1;
                end if;
        end process;
        cuenta <= cuenta_int;
end Behavioral;
----------------------------------------------------------------------------------------------------------------
```

De dicho código cabe destacar dos elementos:

- La utilización del operador de suma aritmética, con la que se incrementará la cuenta:
  cuenta_int = cuenta_int+1;
  Este operador puede ser utilizado ya que se ha cargado la librería aritmética con las sentencias.
  library IEEE;
  use IEEE.STD_LOGIC_ARITH.ALL;
- Para que la suma aritmética pueda ser utilizada convenientemente, todos los operandos deben estar inicializados a un valor lógico ('0' ó '1'). Por lo tanto, es necesario inicializar la señal interna cuenta_int a '0', ya que en caso contrario, estaría inicializada a 'X' (inespecificado), y el resultado de la suma seguiría siendo 'X'.
  signal cuenta_int: std_logic_vector(4 downto 0) := (others => '0');

1.2. Comprobar mediante simulación el comportamiento del contador utilizando el software de XILINX

Para poder realizar la simulación del sistema, es necesario crear un testbench, con el que llevar a cabo dicha simulación. En este testbench, sólo se debe crear la señal de reloj, pues es la única señal de entrada con la que contamos. Dicho testbench podría ser el siguiente:

```
----------------------------------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
```

```
USE ieee.numeric_std.ALL;

ENTITY contador5_tb_vhd IS
END contador5_tb_vhd;

ARCHITECTURE behavior OF contador5_tb_vhd IS

        -- Component Declaration for the Unit Under Test (UUT)
        COMPONENT contador5
        PORT(
                clk : IN std_logic;
                cuenta : OUT std_logic_vector(4 downto 0)
                );
        END COMPONENT;

        --Inputs
        SIGNAL clk :  std_logic := '0';

        --Outputs
        SIGNAL cuenta :  std_logic_vector(4 downto 0) := (others => '0');

BEGIN

        -- Instantiate the Unit Under Test (UUT)
        uut: contador5 PORT MAP(
                clk => clk,
                cuenta => cuenta
        );

        tb : PROCESS
        BEGIN
                clk <= '0';
                wait for 10 ns;
                clk <= '1';
                wait for 10 ns;
        END PROCESS;

END;
```

Utilizando dicho testbench, las formas de onda del sistema se muestran en la figura 1.1. En ella podemos apreciar que las señales de salida, cambian cuando la señal de reloj sufre una transición de subida. También podemos apreciar que cuando se alcanza el mayor número alcanzable (31), vuelve a empezar la cuenta desde 0.

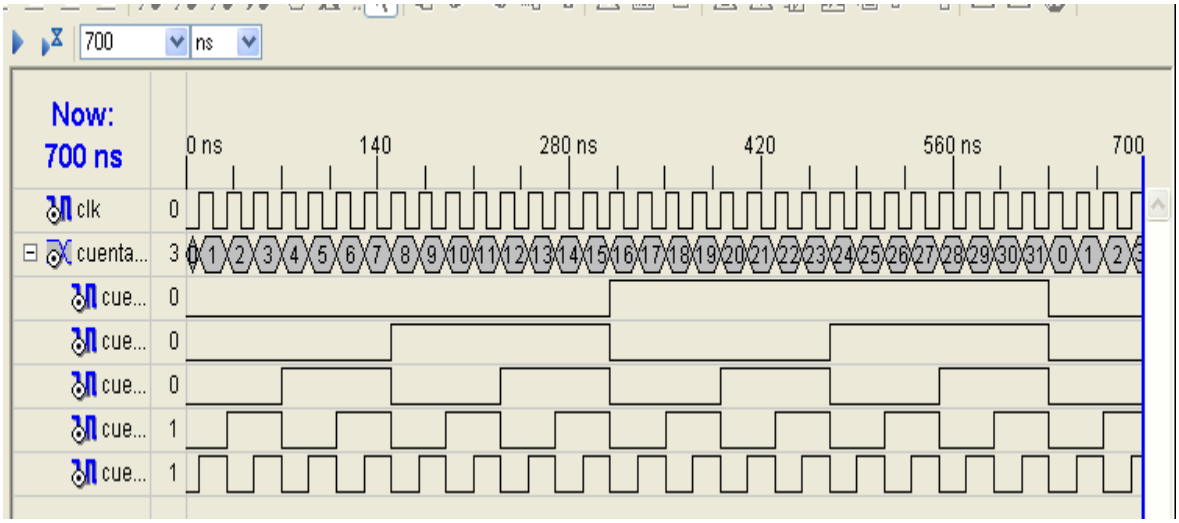


Figura 1.1.- Formas de onda de la simulación de comportamiento del contador de cinco bits.

1.3. Utilizar el software de XILINX para sintetizar el comportamiento modelado en el primer punto

A continuación, sintetizaremos el modelo VHDL del primer punto. Tras dicha síntesis, mostramos el esquemático tecnológico en la figura 1.2. En ella podemos apreciar todos los bloques necesarios para la implementación del contador.

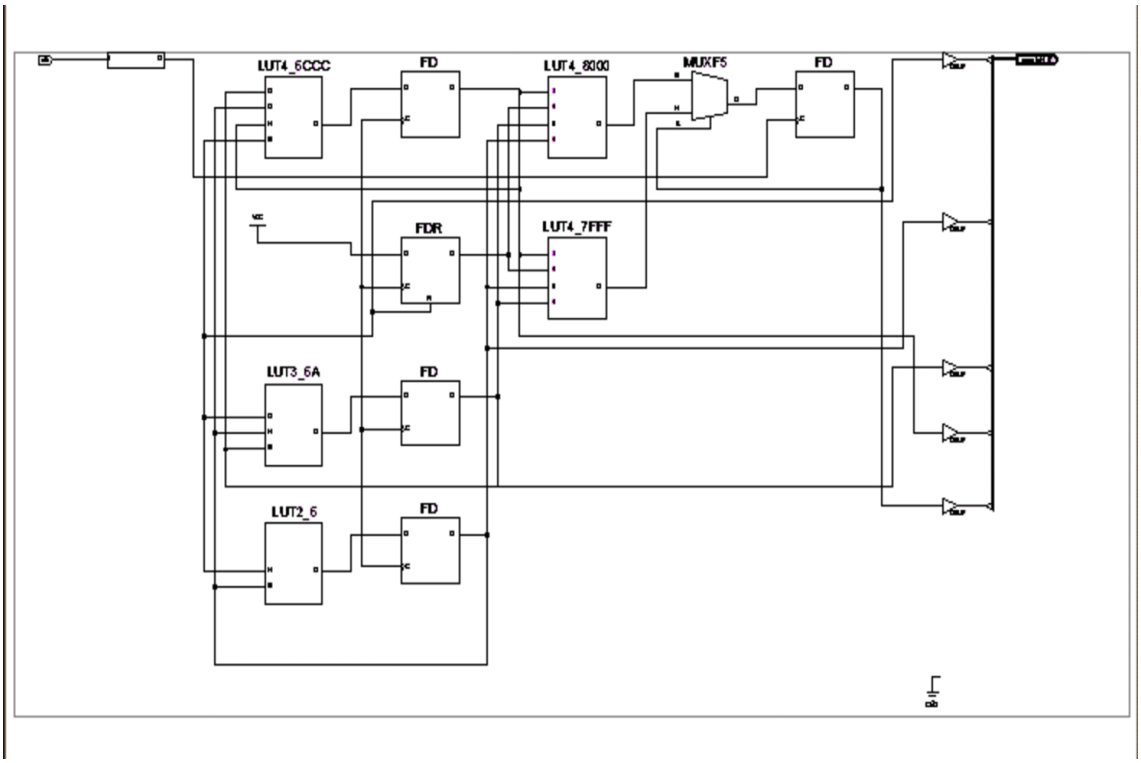


Figura 1.2.- Esquema tecnológico del modelo de contador de cinco bits mostrado en el apartado 1.1.

En dicho esquema podemos tener bloques de librería o LUTs. En el caso de las LUTs, en el mismo esquema podemos ver la función que realizan; mientras que para obtener la funcionalidad de los bloques de librería, debemos acudir a Libreries Guide (que podemos encontrar en la ayuda del Webpack de Xilinx. Para el caso del bloque FDR, mostramos en la figura 1.3 su hoja de características en la que podemos apreciar que se trata de un flip-flop tipo D con reset síncrono.

Por lo tanto, los bloques, junto con sus ecuaciones, son los siguientes:

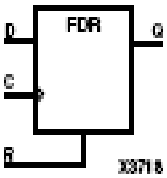
- bufgp: buffer que no tiene
- obuf: no tiene ninguna función lógica
- lut2_6: salida = $\overline{i0}\cdot i1 + i0\cdot\overline{i1}$
- lut4_8000: salida = $i0\cdot i1\cdot i2\cdot i3$
- lut4_7fff: salida = $\overline{i2}+\overline{i1}+\overline{i0}+\overline{i3}$
- lut4_6ccc: salida = $i1\cdot\overline{i2} + i0\cdot\overline{i1}\cdot i2\cdot i3 + \overline{i0}\cdot i1 + i1\cdot\overline{i3}$

FDR

D Flip-Flop with Synchronous Reset

Architectures Supported

FDR	
Spartan-II, Spartan-IIE	Primitive
Spartan-3	Primitive
Virtex, Virtex-E	Primitive
Virtex-II, Virtex-II Pro, Virtex-II Pro X	Primitive
XC9500, XC9500XV, XC9500XL	Primitive
CoolRunner XPLA3	Primitive
CoolRunner-II	Primitive



FDR is a single D-type flip-flop with data (D) and synchronous reset (R) inputs and data output (Q). The synchronous reset (R) input, when High, overrides all other inputs and resets the Q output Low on the Low-to-High clock (C) transition. The data on the D input is loaded into the flip-flop when R is Low during the Low-to-High clock transition.

The flip-flop is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.

GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2 symbol.

Inputs			Outputs
R	D	C	Q
1	X	↑	0
0	1	↑	1
0	0	↑	0

Usage

This design element is inferred in the design code; however, the element can be instantiated for cases where strict placement control, relative placement control, or initialization attributes must be applied.

VHDL Instantiation Template

```
-- Component Declaration for FDR should be placed
-- after architecture statement but before begin keyword

component FDR
```


Figura 1.3.- Hoja de características del bloque FDR obtenida de *Libreries guide*.

- lut3_6a: salida = i0·$\overline{i1}$ + $\overline{i0}$·i1·i2 +i0·$\overline{i2}$
- muxf5: salida = i1·S + i0·$\overline{S}$
- FD: flip-flop tipo D: salida = D cuando clk sube
- FDR: flip-flop tipo D con reset síncrono: salida = '1'·$\overline{reset}$ cuando clk sube (ya que la entrada D = '1').

Por lo tanto, sólo tendremos que implementar los bloques lut*, muxf5, FD. La configuración que se va a utilizar como FDR será la misma que la de FD intercambiando las salidas complementadas y no complementadas para evitar la implementación del inversor de entrada. En el caso de los bloques combinacionales, cuando la implementación de la función complementada no aumente el número de inversores, se optará por esta implementación para evitar la necesidad del inversor de salida.

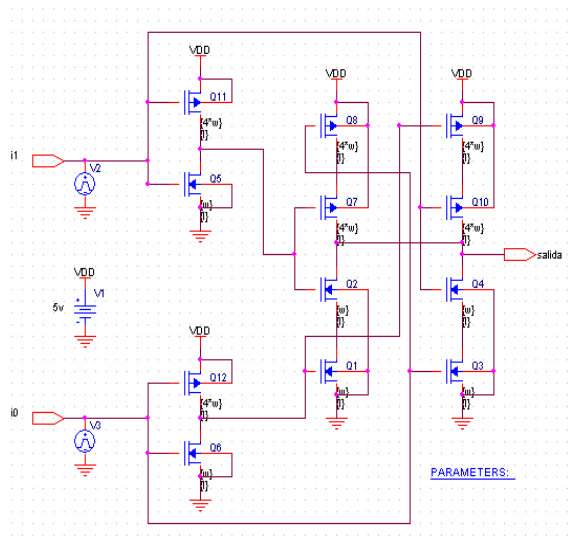
1.4. Simular eléctricamente los bloques obtenidos en el punto anterior utilizando una familia lógica CMOS

Únicamente vamos a detallar el diseño de la celda lut2_6 ya que el modo de actuación con el resto de bloques es el mismo (aunque vosotros tendréis que detallar todos los bloques). Dicha celda consiste en una operación XOR, de la cual ya tenemos la ecuación lógica (que es lo que nos hace falta para el diseño en la familia CMOS estándar). Por lo tanto, el esquema a nivel de transistores de dicha celda se muestra en la figura 1.4. Las características más dignas de consideración son las siguientes:

- Aprovechando que las conexiones series de i0·$\overline{i0}$ e i1·$\overline{i1}$ nunca se pueden dar, se ha simplificado el esquema.
- Como eran necesarios los dos inversores de entrada, se ha optado por implementar el complemento de la operación para evitar el uso del inversor de salida, y de esta forma tener una reducción en el área ocupada por el circuito.
- Para poder determinar la relación entre las anchuras de los transistores PMOS y NMOS debemos obtener los parámetros de movilidad inicial (disponible en el fichero de modelos de PSPICE, correspondiente al parámetro UO).
- Las fuentes de entrada se han elegido de tal forma que no coincida ninguna transición de las señales de entrada.
- Para llevar a cabo el análisis parámetrico, debemos indicar cuáles son los parámetros (un parámetro en PSPICE se indica con una etiqueta entre paréntesis). Por lo tanto debemos incluir un *part* denominado PARAM, en el cual se añadirán los parámetros (en este caso w y l). Como las dimensiones deben estar en el grid (con vistas a la realización del layout), debemos obtener dicho grid que coincide con el parámetro lambda del fichero tecnológico (de MICROWIND). Dicho parámetro es igual a 0.3u.

Los parámetros relativos a la simulación utilizados son los siguientes:

- Análisis transitorio hasta un tiempo igual a 100ns. Obviamente, este valor puede cambiar en función del número de entradas de cada celda, así como de los parámetros de las fuentes de entrada.
- Análisis parámetrico de un parámetro global denominado w desde 0.9um (anchura



** Dimensiones de transistores
$W_P=UO_N*W_N*UO_P = 426.9*W_N/111.5 = 4*W_N$

** Fuentes de entrada
V_V1 VDD 0 5v
V_V2 I1 0 PULSE 0 5 0 0.1n 0.1n 20n 40n
V_V3 I0 0 PULSE 0 5 10n 0.1n 0.1n 20n 40n

** Paramétros iniciales de w y l
.PARAM w=0.9u l=0.6u

** Análisis
.TRAN 0 100ns 0
.STEP LIN PARAM w 0.9u 3u 0.3u
.PROBE

Figura 1.4.- Esquema a nivel de transistores de la celda lut2_6, junto a sus principales características.

mínimo del transistor, que coincide con la anchura mínima de las capas de difusión) hasta 3um con un incremento de 0.3u (tamaño del grid, lambda). En cambio, la longitud se ha elegido la mínima de 0.6u (correspondiente a la anchura mínima de la capa de polisilicio).

- Añadir a las librerías el fichero de librería de los modelos de los transistores.

Las formas de onda de las entradas y las salidas se muestran en la figura 1.5. En ella podemos apreciar como un cambio en una señal de entrada provoca un cambio en la señal de salida.

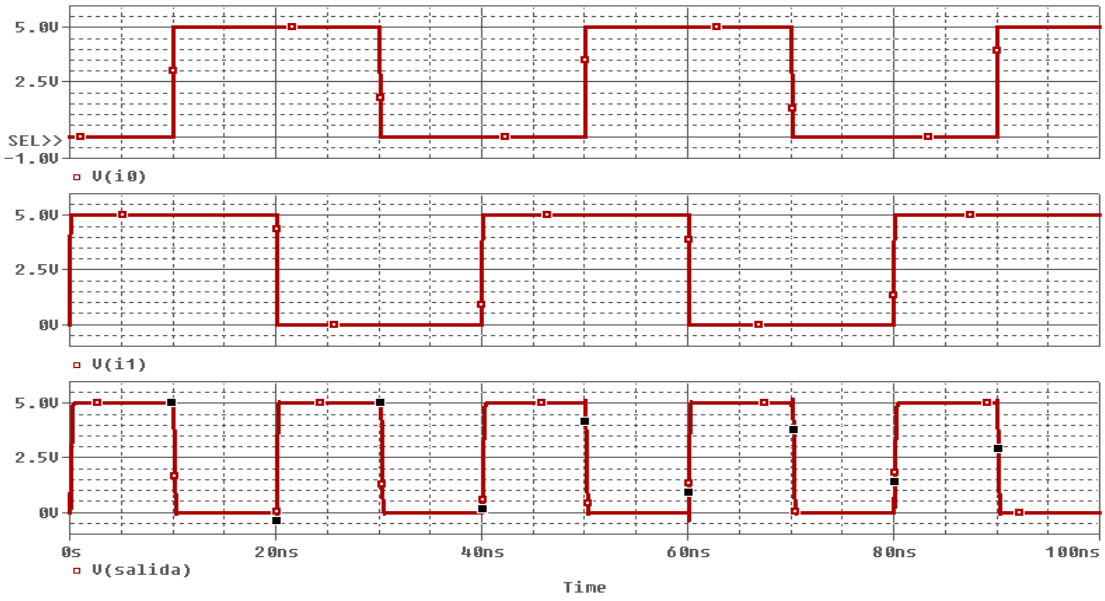


Figura 1.5.- Comportamiento de la celda lut2_6 mediante simulación eléctrica.

Una vez que hemos comprobado el comportamiento correcto, debemos obtener las dimensiones óptimas. Uno de los parámetros más utilizados para determinar las prestaciones

es el producto retraso-potencia, que consiste en multiplicar el retraso medio de propagación con el consumo medio de potencia. Para ello, y tratando de automatizar el proceso, vamos a utilizar las denominadas *Goal Function*. Definiremos dos nuevas funciones, cuyod códigos son los siguientes:

- Retraso medio de propagación

```
delay(1,2,3) = (x5+x6+x7+x8-x1-x2-x3-x4)/4
* Argumento 1 = primer dato de entrada
* Argumento 2 = segundo dato de entrada
* Argumento 3 = dato de salida
*
* Formato de un comando típico
*       traza arg|     search<dirección></comienzo/><ocurrencia:><condición>!<nº marca>
*       <dirección>:
*               forward: búsqueda hacia adelante
*               backward: búsqueda hacia atras
*       </comienzo/> : marca desde la cual empezar la búsqueda
*       <ocurrencia>: número de ocurrencia que se tomará en cuenta
*       <condición>:
*               level(<valor>): localiza el punto de la traza donde la componente y toma <valor>
*               xvalue(<valor>): localiza el punto de la traza donde la componente x toma <valor>
*       <nº marca>: valor que identifica una determinada localización
  {
       1|       search forward 1:level(50%) !1
                search forward level(50%) !2;
       2|       search forward 2:level(50%) !3
                search forward level(50%) !4;
       3|       search forward /x1/ level(50%) !5
                search forward /x2/ level(50%) !6
                search forward /x3/ level(50%) !7
                search forward /x4/ level(50%) !8;
  }
```

- Consumo medio de potencia

```
pot(1) = y1/x1
* Argumento 1 = integral de Idd*Vdd: s(Idd*Vdd)
{
       1|       search forward xvalue(100%) !1;
}
```

Obviamente, cuando se cambie de celda habrá que modificar el código relativo al retraso ya que puede cambiar, por ejemplo, el número de entradas. Por ello, se ha comentado dicho código.

Utilizando dichas funciones, se ha llevado a cabo un análisis de prestaciones (*Performance Analysis*). De dichos análisis, se han obtenido la relación de dichos parámetros en función de la anchura de los transistores mostrados en la figura 1.6, de las cuales se han obtenido los valores de la tabla 1.1.

Podemos apreciar dos comoprtamientos completamente diferenciados entre el retraso de propagación y el consumo medio de potencia. En el caso del retraso medio de propagación se observa un descenso con el aumento de la anchura del transistor, exceptuando algunos picos intermedios como el producido para una anchura de 2.1u. Si continuamos las simulaciones para unas anchuras mayores, podemos apreciar que el retraso de propagación se estabiliza, por lo que un aumento en la anchura no producirá un descenso en el retraso. Esta estabilización se alcanza para valores cercanos a las 3u.

No obstante, el consumo de potencia mantiene un aumento proporcional a la anchura (hay que indicar que la intensidad media es negativa por lo que los valores salen negativos). Por lo tanto, a medida que aumentamos la anchura de los transistores, fluirá por ellos una mayr

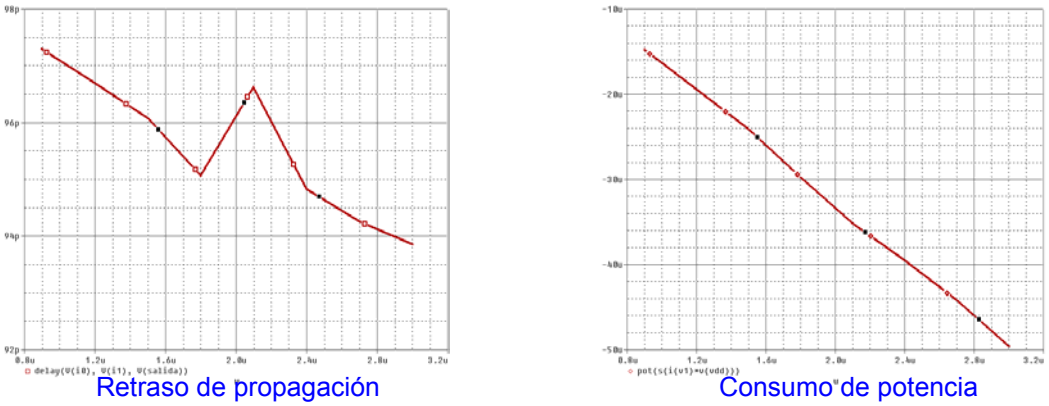


Figura 1.6.- Análisis de prestaciones.

Tabla 1.1. Resultados de simulación de lut2_6

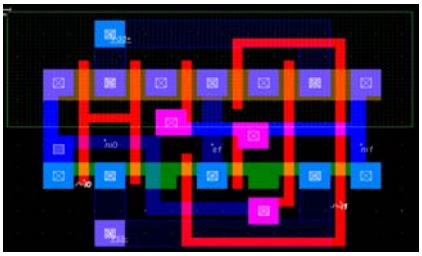
W/L	Retraso (ps)	|Potencia| (uW)	PDP (fJ)
0.9/0.6	97,30	14,71	1,43
1.2/0.6	96,69	19,40	1,88
1.5/0.6	96,07	24,08	2,31
1.8/0.6	95,08	29,73	2,83
2.1/0.6	96,61	35,19	3,40
2.4/0.6	94,83	39,53	3,75
2.7/0.6	94,26	44,16	4,16
3.0/0.6	93,85	49,63	4,66

cantidad de intensidad, y de esta forma habrá un mayor consumo de potencia. En este parámetro no encontraremos una estabilización como en el caso anterior ya que el consumo de potencia va a ser directamente proporcional a la anchura de los transistores.

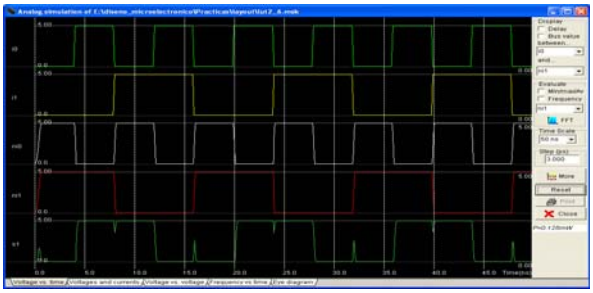
Si consideramos el parámetro PDP, debemos multiplicar las columnas correspondientes de la tabla 1.1. Este parámetro me indicará cuál de los parámetros anteriores tiene una mayor influencia. En este caso, el consumo de potencia tiene mayor influencia por lo que elegiremos las dimensiones mínimas, es decir, w=0.9um y l=0.6um.

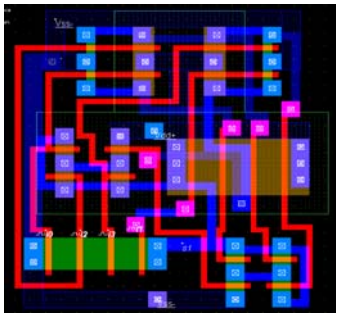
1.5. Obtener el layout de los bloques anteriores

Utilizando las dimensiones anteriores, se han realizado los layouts de los bloques combinacionales en una tecnología CMOS estándar de 0.6 µm. Dichos layouts se muestran en la figura 1.7. Adjunto a los layouts podemos observar la simulación de dichos layouts, comprobando que tienen el comportamiento esperado.

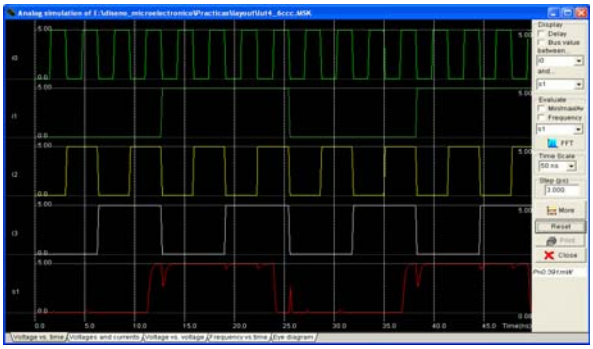


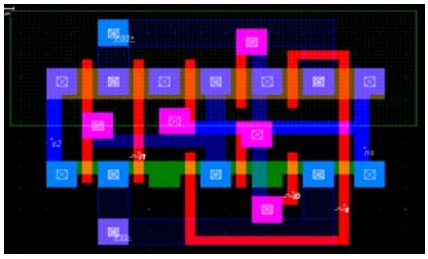
LUT2_6



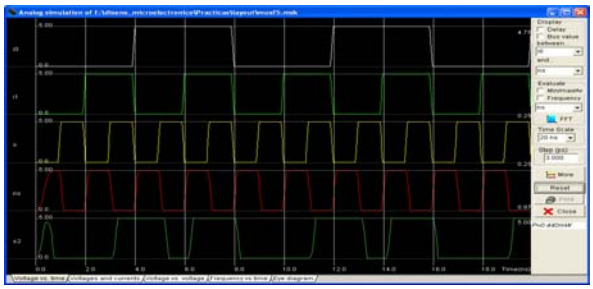


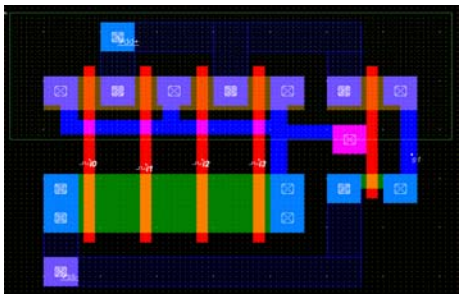
LUT4_6CCC



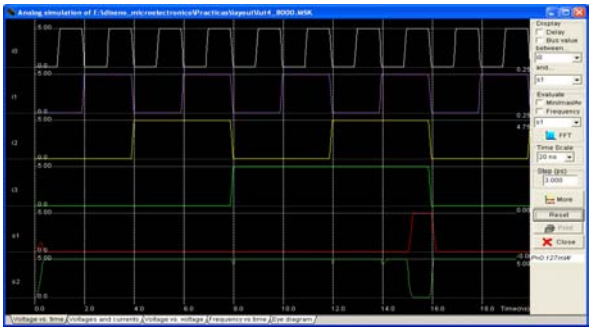


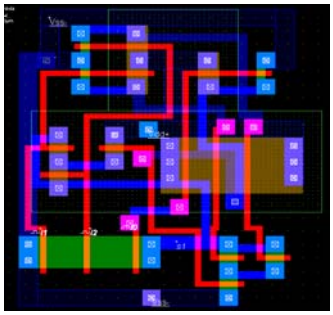
MUXF5





LUT4_8000





LUT3_6A

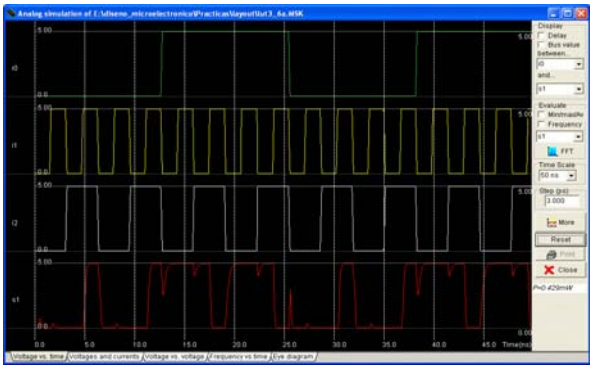


Figura 1.7.- Layouts de los bloques combinacionales.

1.6. Simular eléctricamente los flip-flops, utilización una configuración maestro-esclavo estándar

En el caso de los flip-flops seguiremos los mismos pasos que con los elementos combinacionales. Por lo tanto, en la figura 1.8 mostramos el esquema del flip-flop tipo D según una configuración maestro-esclavo convencional, así como su comportamiento.

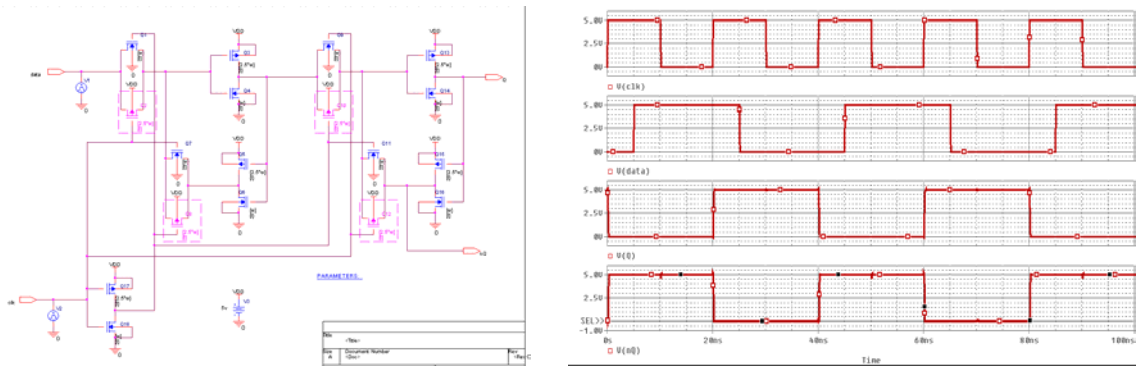


Figura 1.8.- Esquema a nivel de transistores del flip-flop tipo D, junto a su simulación con PSPICE.

Mediante estas simulaciones, y utilizando las goal functions definidas en el caso de los elementos combinacionales, hemos obtenido los valores del retraso de propagación para ambas salidas, Q y nQ, y el consumo medio de potencia. Estas medidas se muestran en la figura 1.9 y en la tabla 1.2.

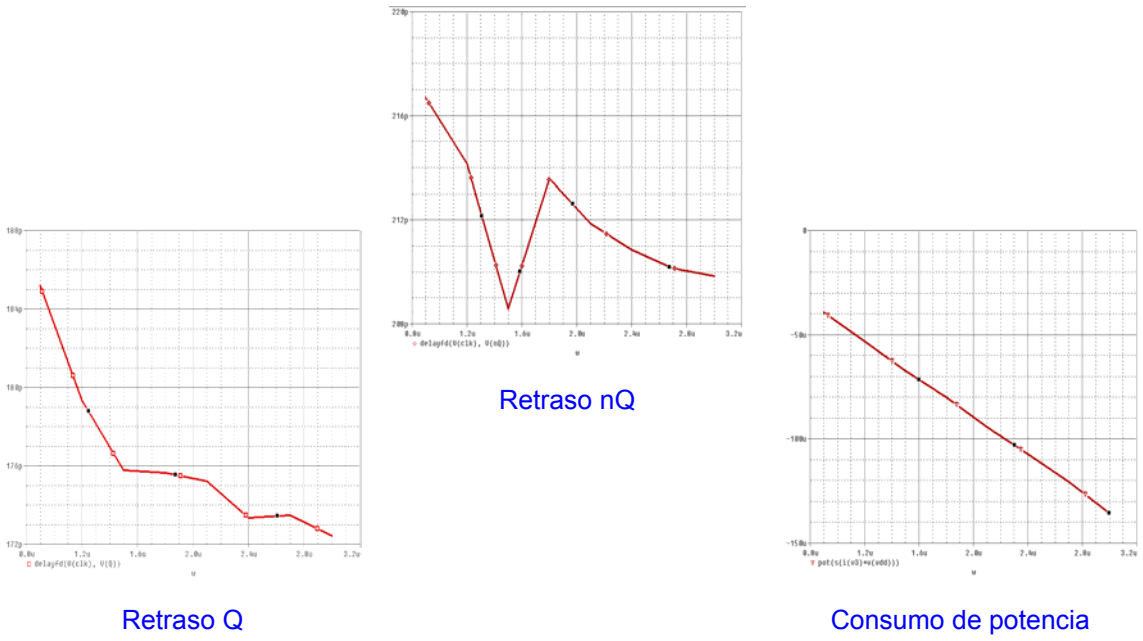


Figura 1.9.- Medidas tomadas de las simulaciones eléctricas.

Por lo tanto, obtenemos que las dimensiones óptimas son las mismas que la de los bloques combinacionales.

Tabla 1.2. Resultados de simulación de FD

W/L	Retraso_q (ps)	Retraso_nq (ps)	|Potencia| (uW)	PDP_q (fJ)	PDP_nq (fJ)
0.9/0.6	185,17	216,69	39,41	7,30	8,54
1.2/0.6	179,36	214,14	53,09	9,52	11,37
1.5/0.6	175,77	208,59	67,22	11,82	14,02
1.8/0.6	175,64	213,59	80,06	14,06	17,10
2.1/0.6	175,23	211,85	94,39	16,54	20,00
2.4/0.6	173,34	210,85	107,14	18,57	22,59
2.7/0.6	173,47	210,13	120,60	20,92	25,34
3.0/0.6	172,44	209,84	135,45	23,36	28,42

1.7. Obtener el layout de los flip-flops según las dimensiones óptimas del apartado anterior

Teniendo en cuenta las dimensiones obtenidas en el apartado anterior, se ha realizado el layout del flip-flop, mostrado en la figura 1.10, junto a su simulación de comportamiento.

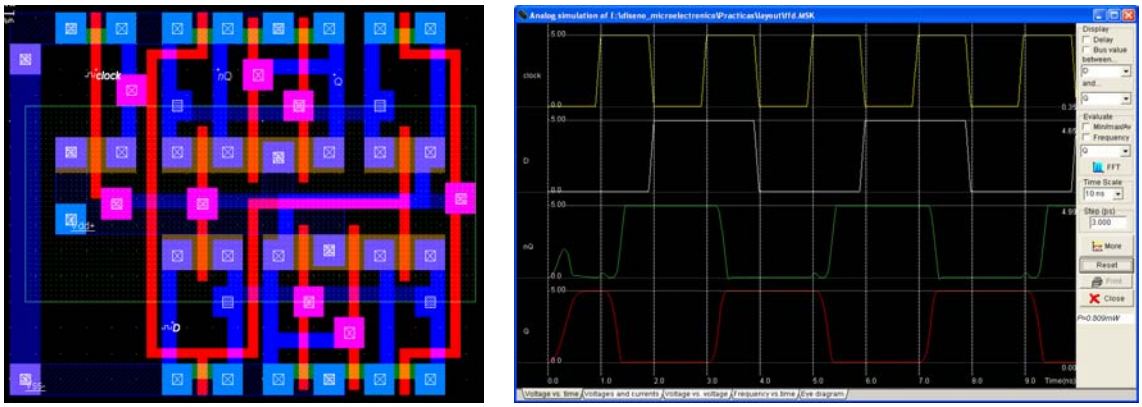


Figura 1.10.- Layout del flip-flop tipo D, junto con su simulación.

1.8. Obtener una colocación óptima de los bloques obtenidos en el tercer punto, utilizando el algoritmo de clustering uniforme

El algoritmo de clustering consiste en ir colocando los diferentes bloques según sus conexiones, es decir, el primer bloque colocado es aquel que tenga un mayor número de conexiones. Para ello, mostramos cada uno de los bloques con sus respectivas conexiones:

```
LUT4_6CCC-->cuenta_int(0), cuenta_int(3), cuenta_int(1), cuenta_int(2); result(3)
LUT3_6A    -->cuenta_int(2), cuenta_int(1), cuenta_int(0); result(2)
LUT2_6     -->cuenta_int(1), cuenta_int(0); result(1)
```

```
LUT4_8000 -->cuenta_int(1), cuenta_int(2), cuenta_int(0), cuenta_int(3); N13
LUT4_7FFF -->cuenta_int(2), cuenta_int(1), cuenta_int(0), cuenta_int(3); N12
MUXF5     -->N13, N12; cuenta_int(4); result(4)
FD1       -->clk, result(3); cuenta_int(3)
FDR2      -->clk, cuenta_int(0); cuenta_int(0)
FD3       -->clk, result(2); cuenta_int(2)
FD4       -->clk, result(1); cuenta_int(1)
FD5       -->clk, result(4); cuenta_int(4)
```

Como los bloques LUT4_8000 y LUT4_7FFF son los mismos pero las salidas estarían antes y después del inversor de salida, únicamente vamos a utilizar uno de ellos. Por lo tanto, el orden de colocación será el siguiente: LUT4_6CCC, LUT4_8000 (LUT4_7FFF), LUT3_6A, MUXF5, LUT2_6, FD1, FDR2, FD3, FD4, FD5. Luego una posible colocación sería la mostrada en la figura 1.11.

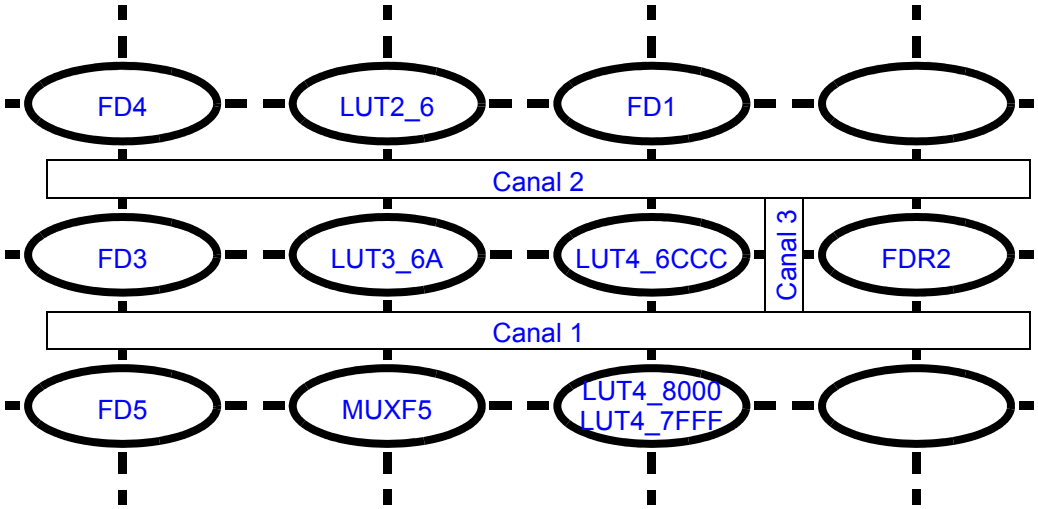


Figura 1.11.- Colocación utilizando el algoritmo de clustering uniforme

1.9. Obtener un rutado óptimo utilizando el algoritmo basado en cliques

Para ilustrar este algoritmo de rutado, vamos a centrarnos en el canal 2. En primer lugar, hay que obtener las redes que tenemos en cada lado del canal; dichas redes se muestran en la figura 1.12(a). De estas redes obtendremos el grafo de compatibilidad, como el complemento de los grafos de restricciones verticales y horizontales, mostrado en la figura 1.12(b). Dicho grafo nos muestra las diferentes redes que pueden compartir una pista ya que no se solapan ni horizontalmente ni verticalmente. Dichas redes están agrupadas en cliques, las cuales están unidas todas entre sí. Los cliques de dicho canal se muestran en la figura 1.12(c). La colocación de las pistas empezarán por el límite superior, así que calcularemos el peso de cada clique como la diferencia entre los pines del límite superior y del límite inferior de las redes existentes en cada clique; luego si dicho valor es muy positivo querrá decir que existe más pines en el límite superior que en el inferior, por lo que para minimizar el rutado, su pista debería estar creca del límite superior. Clasificando los cliques por sus pesos, empezamos a colocarlos en las pistas desde el límite superior, obteniendo el rutado mostrado en la figura 1.12(a).

En la figura 1.13, mostramos un posible layout del contador completo, junto a la simulación de su comportamiento.

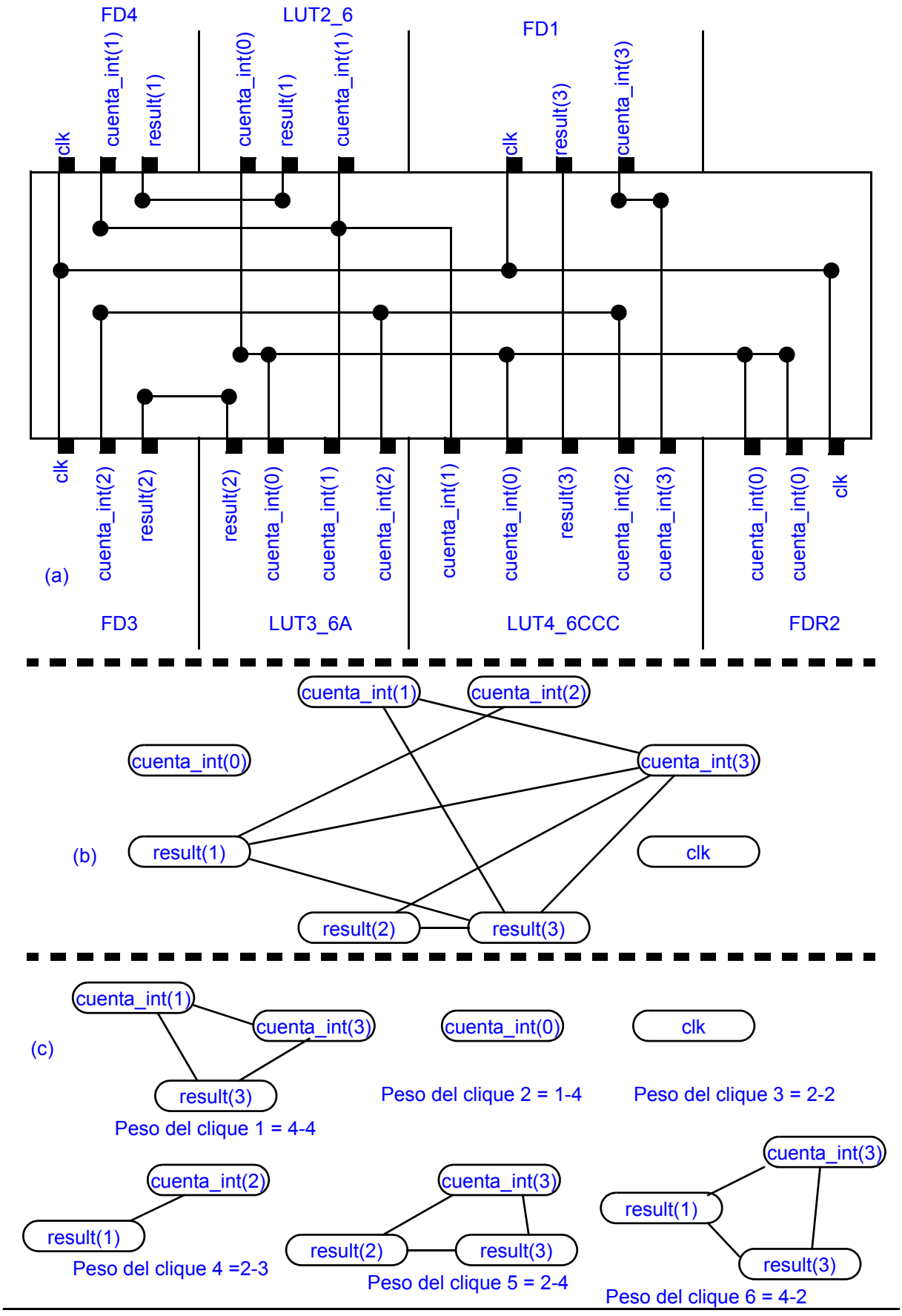


Figura 1.12.- (a) Canal de rutado; (b) grafo de compatibilidad; y (c) cliques de dicho grafo.

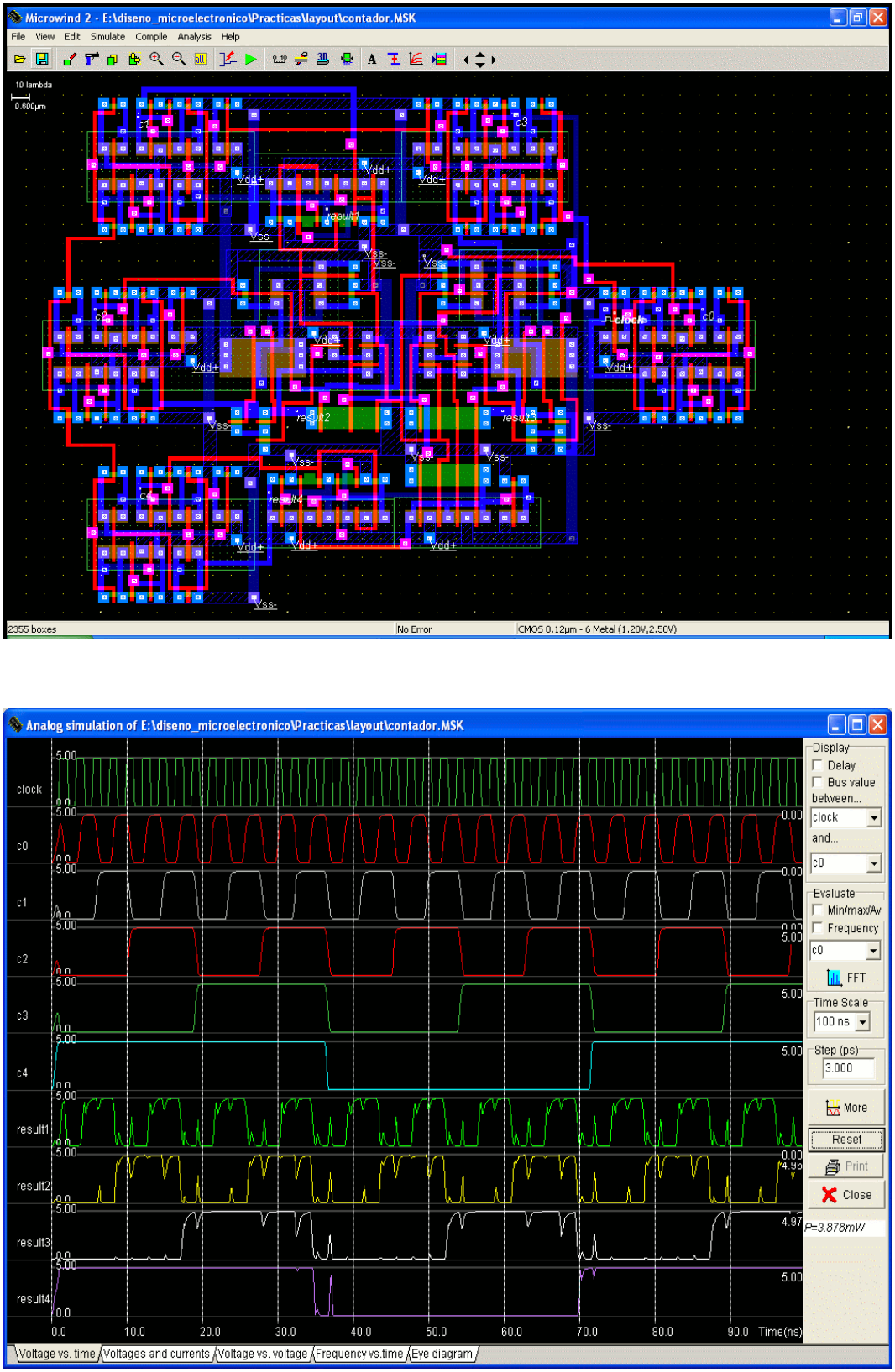


Figura 1.13.- Layout del contador completo, junto a su simulación.