DOCUMENTACIÓN

Escrito con la documentación y procedimientos seguidos para completar el ejercicio, así como una lista de los programas usados para realizar las diferentes partes de este.

Alonso Muñoz
Javier Funcia Tomé
Víctor

Índice

Índice de figuras

Víctor Funcia Tomé, Javier Alonso Muñoz

Objetivos y aspectos relevantes

El objetivo principal de este ejercicio es el de realizar un contador arbitrario, tanto el diseño como la implementación de este finalmente en Verilog. Comprobando su funcionamiento mediante el uso de cronogramas que se interpretaran usando GTKWave, así como usando propios mensajes de log desde verilog usando la orden $display y $monitor.

Introducción

Antes de comenzar con lo que es el diseño del componente consideramos que es importante aclarar que es lo que se va a diseñar. Como ya indicamos en los objetivos se trata de un contador arbitrario. ¿Pero, que es exactamente un contador arbitrario?

Un contador arbitrario, como su propio nombre indica, es un contador que en vez de funcionar como un contador típico que se dedica a contar ya sea descendente o ascendentemente va a seguir un patrón establecido, como podría ser 5, 8, 6, 7, 12, 6, 6, 4 y vuelta a empezar por el 5.

En nuestro caso, para conseguir este funcionamiento vamos a usar biestables JK, que nos van a permitir ir guardando el estado del contador.

En nuestro caso la secuencia a seguir será la siguiente.

3 → 7 → 6 → 6 → 15 → 14 → 7 → 10 → 12 → 14 → REPETIR

Diseño del contador

Una vez explicado que es un contador arbitrario y el funcionamiento que tenemos que conseguir vamos a comenzar con el diseño del componente.

Biestable JK

A la hora de ir almacenando los distintos valores obtenidos usaremos biestables JK, los cuales podrán almacenar 1 bit. Y debido a que necesitamos realizar un contador con número máximo de 15 (1111) necesitaremos usar 4 de estos biestables para poder realizar el contador de 4 bits.

Este biestable cuenta con 5 entradas y 2 salidas. Siendo estas las siguientes.

- Entrada J: La entrada J o SET se usa para poner a SET (1) el estado del biestable.
- Entrada K: La entrada K o RESET se usa para poner a RESET (0) el estado del biestable.
- Entrada PRESET: Le entrada PRESET sirve para poner el estado del biestable a SET (1), sin tener en cuenta ni J ni K.
- Entrada CLR o CLEAR: La entrada CLR o CLEAR es la opuesta a la PRESET, sirve para poner el estado del biestable directamente a RESET (0), sin tener en cuenta ni J ni K.
- Entrada C (Clock): La entrada C es la entrada del reloj. Sirve básicamente para hacer que el componente funcione de forma síncrona.
- Salida Q: La salida Q es en la que se ve reflejada el estado del biestable, si es 1 será 1 y si es 0 será 0.
- Salida $\bar{Q}$: La salida Q negada, como su nombre indica es básicamente el opuesto de Q.

A continuación, se puede ver tanto la representación ya del componente como si fuese un chip y como es internamente.

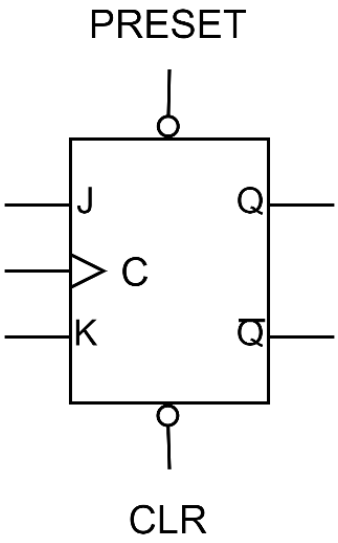

Fig. 1 Módulo biestable JK

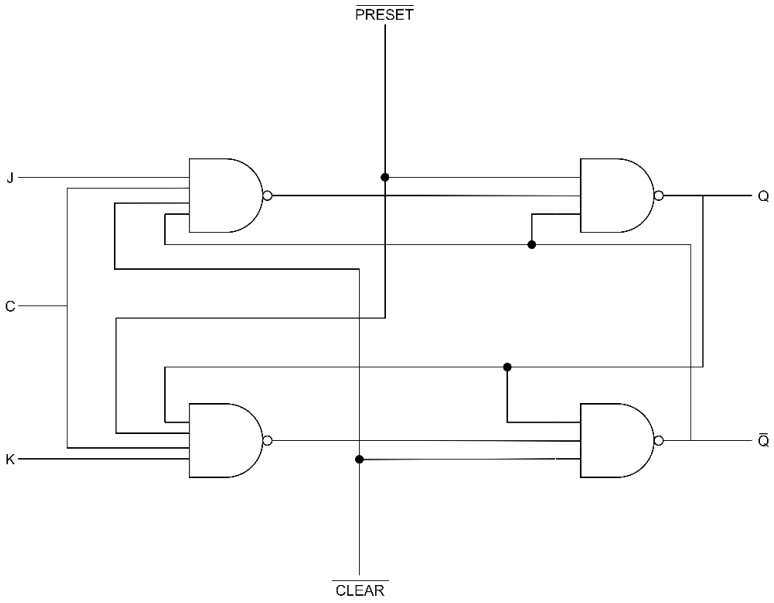

Fig. 2 Interior Módulo biestable JK

Secuencia, tablas y diagramas.

Una vez ya sabemos la secuencia que vamos a realizar, la cual ya se indicó en la introducción. Sin embargo, hay un apartado importante que no abarcamos en la introducción, y es como tratamos los números repetidos que aparecen en la secuencia.

Simplemente, lo que haremos será asignarlos a esos valores otro valor que no se encuentre en la secuencia, y posteriormente asociarlo al valor deseado cuando realicemos los mapas de Karnaugh.

3 → 7 → 6 → 6 → 15 → 14 → 7 → 10 → 12 → 14 → REPETIR

La secuencia a usar es la siguiente:

3 → 7 → 6 → 5 → 15 → 14 → 4 → 10 → 12 → 2 → REPETIR

Se ha decido cambiar los valores repetidos en la secuencia por los valores 4, 3, y 2. Básicamente lo que tendremos que conseguir ahora es que esos números los transforme al número deseado. Para lo cual vamos a realizar los mapas de Karnaugh. Pero antes veamos el diagrama de transición y las tablas de transición.

Diagrama de transición

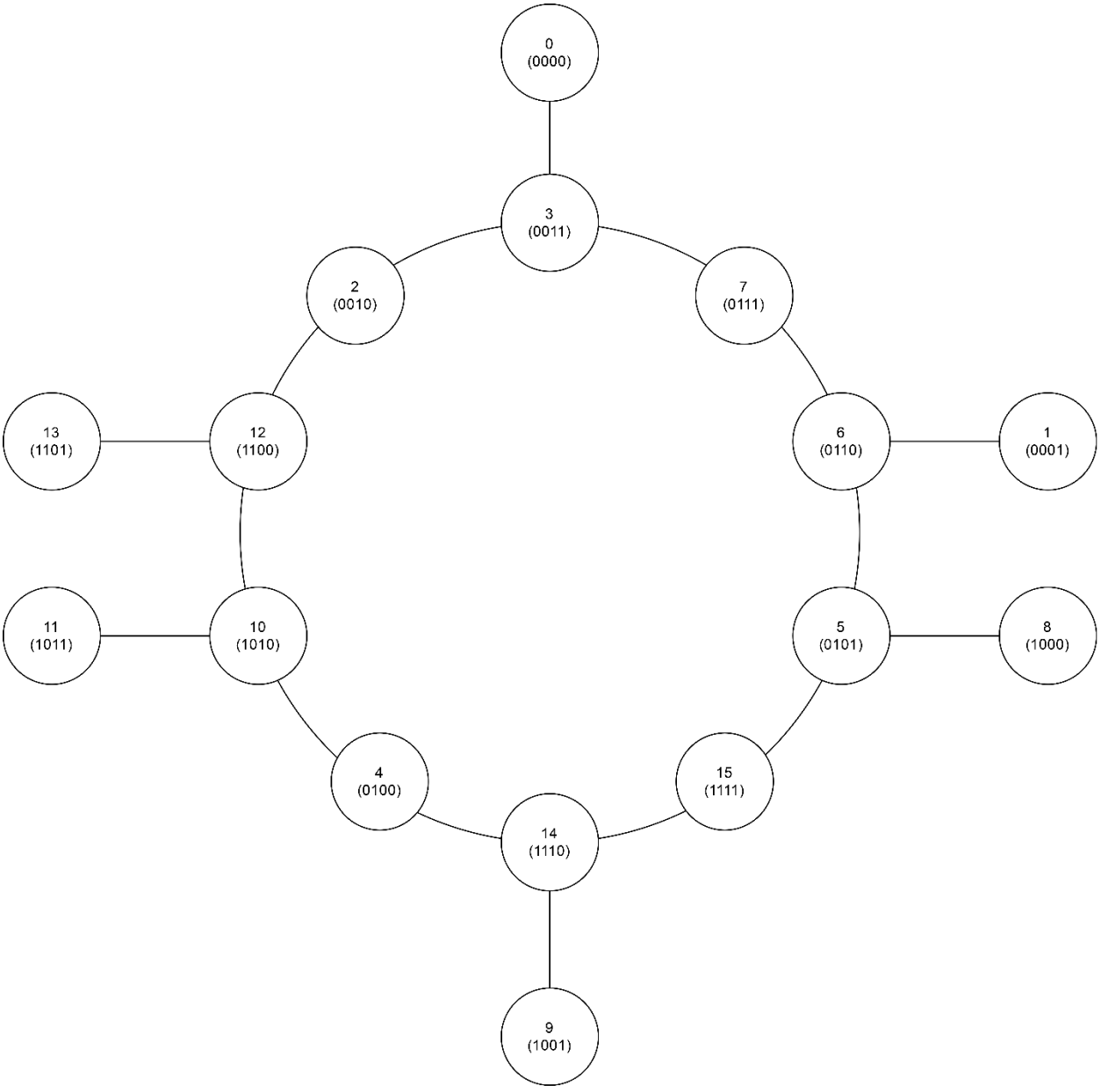

Fig. 3 Diagrama de transición

Como se puede ver en el diagrama ya se ha diseñado tomando la secuencia modificada para prevenir la repetición de números. También se puede ver como los valores que no estaban contemplados en la secuencia se han tenido que añadir, de tal forma que acaben apuntando a un valor que si está dentro de la secuencia. Así evitaremos valores indeseados y nos aseguraremos de que siempre se vaya a acabar realizando la secuencia. Haber elegido este método en vez de hacer directamente mapas de Karnaugh puede que haga de nuestro circuito uno menos optimo que requiera de una cantidad de mayor de puertas lógicas, ya que en vez de poner esos valores a X en los mapas de Karnaugh para coger la combinación más optima.

Un caso que podría haber ocurrido pero que por suerte no se ha dado es el de acabar es un "bucle infinito" cambiando de un valor a otro indefinidamente. Por ejemplo, 7 (0111) → 14 (1110) → 7 (0111) → 14 (1110). Si se hubiera producido uno de esos casos hubiésemos tenido que evaluarlo posteriormente en los mapas de Karnaugh y en la tabla de transiciones, para detectar y evitar estos errores.

Tablas de transiciones

Una vez establecido la secuencia y realizado el diagrama de transición se realizarán las tablas de transiciones, en la cual veremos para los diferentes inputs en nuestros biestables las salidas deseadas para realizar correctamente el contador arbitrario.

Primero vamos a hacer una tabla simplificada en la que simplemente vamos a indicar los valores a los que tiene que "apuntar" cada valor.

0 (0000) →	3 (0011)
1 (0001) →	6 (0110)
2 (0010) →	14 (1110)
3 (0011) →	3 (0011)
4 (0100) →	7 (0111)
5 (0101) →	6 (0110)
6 (0110) →	6 (0110)
7 (0111) →	7 (0111)
8 (1000) →	5 (0101)
9 (1001) →	14 (1110)
10 (1010) →	10 (1010)
11 (1011) →	10 (1010)
12 (1100) →	12 (1100)
13 (1101) →	12 (1100)
14 (1110) →	14 (1110)
15 (1111) →	15 (1111)

Fig. 4 Transiciones de valores

En esta tabla de la izquierda se ha simplificado los valores deseados cuando tenemos un determinado input.

Los campos de color blanco quieren decir que se mantiene igual, ya que el valor deseado sigue siendo el mismo. Como por ejemplo el caso del valor 3, ya que este valor aparece en la combinación original sin repetirse.

En cambio, los de color rojo y morado quieren decir que se ha producido un cambio a la hora de obtener el valor final. En el caso de los de color rojo son valores que no están contemplados en la secuencia de valores, a los que posteriormente en el diagrama de transición se les ha asignado los valores a los cuales tienen que pasar. En el caso de los de color morado son los valores que hemos optado por poner para sustituir a los valores repetidos. Y como se puede ver en la tabla estos valores tienen que pasar a los valores indicados en la secuencia original.

Ahora vamos a realizar la tabla básica de transiciones del biestable JK para apoyarnos en ella a la hora de construir la tabla de transiciones.

Actual	Siguiente	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Fig. 5 Tabla de transiciones del biestable JK

Una vez ya tenemos la tabla de transiciones del biestable JK procederemos a realizar la tabla de transiciones siguiendo los patrones establecidos en la tabla anterior y, esta vez sí, pasando al siguiente valor de la secuencia.

	J3	K3	J2	K2	J1	K1	J0	K0
0000 → *0011*	0	X	0	X	1	X	1	X
0001 → *0110*	0	X	1	X	1	X	X	1
0010 → *0011*	0	X	0	X	X	0	1	X
0011 → *0111*	0	X	1	X	X	0	X	0
0100 → *1010*	1	X	X	1	1	X	0	X
0101 → *1111*	1	X	X	0	1	X	X	0
0110 → *0101*	0	X	X	0	X	1	1	X
0111 → *0110*	0	X	X	0	X	0	X	1
1000 → *0101*	X	1	1	X	0	X	1	X
1001 → *1110*	X	0	1	X	1	X	X	1
1010 → *1100*	X	0	1	X	X	1	0	X
1011 → *1010*	X	0	0	X	X	0	X	1
1100 → *0010*	X	1	X	1	1	X	0	X
1101 → *1100*	X	0	X	0	0	X	X	1
1110 → *0100*	X	1	X	0	X	1	0	X
1111 → *1110*	X	0	X	0	X	0	X	1

Fig. 6 Tabla de transiciones

Como se puede ver en la tabla de transiciones simplemente es una versión ampliada de la tabla de transiciones del biestable JK. Ya que para su creación simplemente se han ido tomando bit a bit los valores de izquierda a derecha e ir comparando los valores sobre la tabla del biestable JK para cada biestable, ya que como se puede ver en la tabla de transiciones contamos con 4 de ellos.

Mapas de Karnaugh

Mapas de los biestables

Una vez ya tenemos la tabla de transacciones el siguiente paso es realizar los mapas de Karnaugh para cada una de las entradas de los diferentes biestables. Que al tener 4 de ellos con 2 entradas J y K cada uno de ellos acabaremos con 8 mapas de Karnaugh.

Antes de comenzar a realizar los mismos, consideramos procedente realizar una breve explicación de estos. Básicamente un mapa de Karnaugh se usa para simplificar funciones lógicas binarias, es un diagrama que se construye en forma de tabla en cuyas filas y columnas se ponen los diferentes valores a

analizar teniendo como intersección de dicha fila y columna el valor que se obtendría al introducir el valor que forma la fila y columna.

Veamos a continuación los mapas de Karnaugh obtenidos de la tabla de transiciones.

J3

$Q_1\,Q_0$ \ $Q_3\,Q_2$	00	01	11	10
00	0	1	X	X
01	0	1	X	X
11	0	0	X	X
10	0	0	X	X

Fig. 7 Mapa Karnaugh J3

Este primer mapa será del cual obtendremos la función booleana de los bits que afectará a los datos que vaya a recibir la J del biestable 3.

Se ha tomado un conjunto de 4 bits a 1 para hacer la función booleana, tomando los valores que nos eran indiferentes como unos. De esta forma obtendremos la siguiente función booleana.

$$\overline{Q_1} \cdot Q_2$$

Una vez obtenemos la función booleana podemos ver que simplemente necesitaremos una puerta AND para realizarla. Ya que la salida del biestable 1 negada la tomaremos directamente de la salida nQ.

J2

$Q_1\,Q_0$ \ $Q_3\,Q_2$	00	01	11	10
00	0	X	X	1
01	1	X	X	1
11	1	X	X	0
10	0	X	X	1

Fig. 8 Mapa Karnaugh J2

Este segundo mapa será del cual obtendremos la función booleana de los bits que afectará a los datos que vaya a recibir la J del biestable 2.

Como se puede ver en el mapa se han tomado 3 secciones de 2x2 bits a uno para obtener la función booleana. De esta forma obtendremos la siguiente la función booleana.

$$Q_0 \cdot \overline{Q_3} + \overline{Q_1} \cdot Q_3 + \overline{Q_0} \cdot Q_3$$

Para esta función necesitaremos de 3 puertas AND y dos puertas OR. Y esta función la que asociaremos a la entrada J2.

J1

$Q_1\,Q_0$ \ $Q_3\,Q_2$	00	01	11	10
00	1	1	1	0
01	X	1	0	1
11	X	X	X	X
10	X	1	X	X

Fig. 9 Mapa Karnaugh J1

Este tercer mapa será del cual obtendremos la función booleana de los bits que afectará a los datos que vaya a recibir la J del biestable 1.

Como se puede ver en el mapa se han tomado dos secciones de 2x2 bits y una de 2x4 bits. Obteniendo la siguiente función booleana.

$$\overline{Q_3} + Q_2 \cdot \overline{Q_0} + \overline{Q_2} \cdot Q_0$$

Para esta función necesitaremos de 2 puertas AND y dos puertas OR. Y esta función es la que asociaremos a la entrada del J1.

J0

$Q_1\,Q_0$ \ $Q_3\,Q_2$	00	01	11	10
00	1	0	0	1
01	X	X	X	X
11	X	X	X	X
10	1	1	0	0

Fig. 10 Mapa Karnaugh J0

Este cuarto mapa será del cual obtendremos la función booleana de los bits que afectará a los datos que vaya a recibir la J del biestable 0.

Se ha tomado dos secciones de 2x2 obteniendo la siguiente función booleana.

$$Q_1 \cdot \overline{Q_3} + \overline{Q_2} \cdot \overline{Q_1}$$

Para esta función booleana necesitaremos de 2 puertas AND y una puerta OR.

K3

$Q_1\,Q_0$ \ $Q_3\,Q_2$	00	01	11	10
00	X	X	1	1
01	X	X	0	0
11	X	X	0	0
10	X	X	1	0

Fig. 11 Mapa Karnaugh K3

Este quinto mapa será del cual obtendremos la función booleana de los bits que afectará a los datos que vaya a recibir la K del biestable 3.

Se ha tomado una sección de 2x2 y otra de 4x1, obteniendo la siguiente función booleana.

$$\overline{Q_1} \cdot \overline{Q_0} + Q_2 \cdot \overline{Q_0}$$

Para esta función booleana necesitaremos de 2 puertas AND y una puerta OR, pero como se puede observar, el segundo AND es reutilizable, ya que es común a la que nos encontramos al realizar la función del mapa del J1.

K2

$Q_1\,Q_0$ \ $Q_3\,Q_2$	00	01	11	10
00	X	1	1	X
01	X	0	0	X
11	X	0	0	X
10	X	0	0	X

Fig. 12 Mapa Karnaugh K2

Este sexto mapa será del cual obtendremos la función booleana de los bits que afectará a los datos que vaya al recibir la K del biestable 2.

Se ha tomado una sección de 4x1, obteniendo la siguiente función booleana.

$$\overline{Q_1} \cdot \overline{Q_0}$$

Para esta función booleana simplemente necesitaremos dos puertas AND.

K1

$Q_3\,Q_2$ / $Q_1\,Q_0$	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	0	0	0	0
10	0	1	1	1

Fig. 13 Mapa Karnaugh K1

Este séptimo mapa será del cual obtendremos la función booleana de los bits que afectará a los datos que vaya a recibir la K del biestable 1.

Se han tomado dos secciones de 2x2 obteniendo la siguiente función booleana.

$$Q_2 \cdot \overline{Q_0} + Q_3 \cdot \overline{Q_0}$$

Para esta función booleana necesitaríamos dos puertas AND y una OR. Pudiendo reutilizar la primera puerta AND, ya que es común a funciones booleanas hechas anteriormente.

K0

$Q_3\,Q_2$ / $Q_1\,Q_0$	00	01	11	10
00	X	X	X	X
01	1	0	1	1
11	0	1	1	1
10	X	X	X	X

Fig. 14 Mapa Karnaugh K0

El ultimo mapa, será del cual obtendremos la función booleana de los bits que afectará a los datos que vaya a recibir la K del biestable 0.

Se han tomado dos secciones 2x2 y una 2x4, obteniendo la siguiente función booleana.

$$\overline{Q_2} \cdot \overline{Q_1} + Q_3 + Q_2 \cdot Q_1$$

Para esta función booleana necesitaríamos dos puertas AND y dos puertas OR. En este caso podríamos reutilizar la primera de ellas, ya que es común a funciones booleanas hechas anteriormente.

Una vez obtenidas todas las funciones booleanas gracias a los mapas de Karnaugh podremos implementar la lógica obtenida usando puertas lógicas AND y OR, y así obtener la secuencia arbitraria para la cual hemos realizados lo mapas.

Mapas de la tabla de transiciones para números repetidos

Una vez tenemos ya las ecuaciones booleanas con las cuales funcionará el contador arbitrario debemos tener en cuenta que a la hora de implementar la lógica obtenida la secuencia que conseguiremos será la siguiente.

3 → 7 → 6 → 5 → 15 → 14 → 4 → 10 → 12 → 2 → REPETIR

Obtendremos la secuencia sin números repetidos, lo cual es correcto ya que es sobre la que hemos hecho las tablas de transiciones y los mapas de Karnaugh. Entonces, ¿cómo hacemos ahora para obtener los valores deseados?

Simplemente realizaremos un módulo llamado “Cambiador”, el cual se encargará de tomar la salida del contador y transformarla al valor deseado, todo esto usando lógico booleana. Y para esto, necesitaremos hacer los respectivos mapas de Karnaugh para obtener sus funciones booleanas.

Q3

$I_1\ I_0$ \ $I_3\ I_2$	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	0	1	1
10	1	0	1	1

Fig. 15 Mapa Karnaugh Números Repetidos Q3

Este primer mapa es para la salida Q_3 del contador, se han numerado las salidas desde Q_3 hasta Q_0 indicando estas la posición del bit dentro de los 4 bits que usamos.

Por tanto, esta tabla es básicamente para el bit más a la izquierda. La lógica a seguir para realizar el mapa es simple, ya que simplemente para una entrada como la que puede ser 0000 obtendremos, en este caso, 0000, y como estamos trabajando en este caso con Q_3 el valor a introducir en la posición 00,00 del mapa será un 0. En el caso de que sea un 1 se introducirá un 1.

Y es así como acabamos con el mapa de Karnaugh para las transiciones de valores repetidos. El proceso será el mismo para los restantes, con la única diferencia de que tomaremos un Q diferente.

De este primer mapa se puede obtener la función booleana para Q_3:

$$I_3 + \overline{I_2} \cdot I_1 \cdot \overline{I_0}$$

Necesitando 1 puerta AND con 3 entradas y una puerta OR.

Q2

$I_1\ I_0$ \ $I_3\ I_2$	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	1	1	1	0

Fig. 16 Mapa Karnaugh Números Repetidos Q2

Este segundo mapa será para la salida Q_2 del contador.

El proceso que se ha seguido para obtener los diferentes valores ha sido el mismo que el realizado en el mapa anterior. Dándonos como función booleana la siguiente:

$$I_2 + \overline{I_3} \cdot I_1 \cdot \overline{I_0}$$

Necesitando 1 puerta AND con 3 entradas y una puerta OR.

Q1

$I_1\ I_0$ \ $I_3\ I_2$	00	01	11	10
00	0	1	0	0
01	0	1	0	0
11	1	1	1	1
10	1	1	1	1

Fig. 17 Mapa Karnaugh Números Repetidos Q1

Este tercer mapa será para la salida Q_1 del contador.

Del mapa podremos sacar la siguiente función booleana:

$$I_1 + I_2 \cdot \overline{I_3}$$

Necesitando 1 puerta AND y 1 puerta OR.

Q0

$I_1\ I_0$ \ $I_3\ I_2$	00	01	11	10
00	0	1	0	0
01	1	0	1	1
11	1	1	1	1
10	0	0	0	0

Fig. 18 Mapa Karnaugh Números Repetidos Q0

Y por último tendremos el mapa correspondiente a la salida Q_0 del contador.

Del mapa podremos obtener la siguiente función booleana:

$$I_1 \cdot I_0 + I_0 \cdot I_3 + I_0 \cdot \overline{I_2} + I_2 \cdot \overline{I_3} \cdot \overline{I_1} \cdot \overline{I_0}$$

La cual será la más costosa de todas requiriendo de, 3 puertas AND de 2 entradas, 1 puerta AND de 4 entradas, 3 puertas OR de dos entradas o una de 4 entradas.

Una vez ya con las funciones lógicas del contador y del cambiador podemos empezar a trabajar en el código

Diagramas de los circuitos

En este apartado explicaremos brevemente tanto el circuito del contador arbitrario como el del cambiador.

Contador Arbitrario

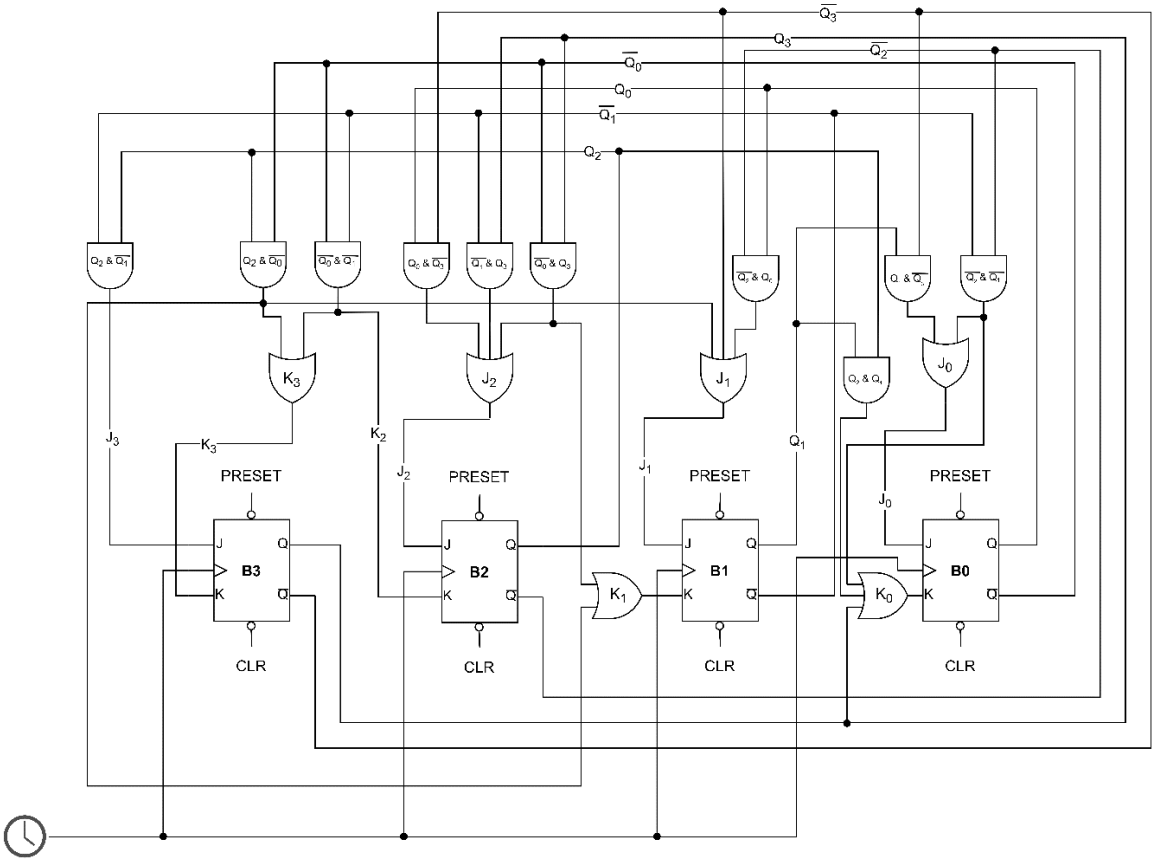

Fig. 19 Circuito Contador Arbitrario

Como se puede ver en el circuito ignoraremos las entradas PRESET y CLR del biestable, ya que no necesitaremos hacer uso de ellas.

Observando el circuito llegamos a la conclusión de que al final lo que es el módulo del contador va a contar con 11 puertas AND y 6 puertas OR, 17 puertas en total. Se ha intentado optimizar el uso de estas reusando las que eran comunes a más de una función. Este circuito, si se quisiese implementar físicamente, usando circuitos integrados existentes con sus puertas lógicas, nos costaría un buen dinero, más del que nos hubiese costado si lo hubiésemos desarrollado usando puertas tipo NAND.

Posteriormente en los anexo se incluirá una versión del contador, pero esta vez realizado con puertas NAND, comparando el precio que tendría respecto del actual.

También podemos observar cómo se ha diseñado un circuito síncrono que va a seguir un ciclo de ejecución de acuerdo con el reloj.

Cambiador

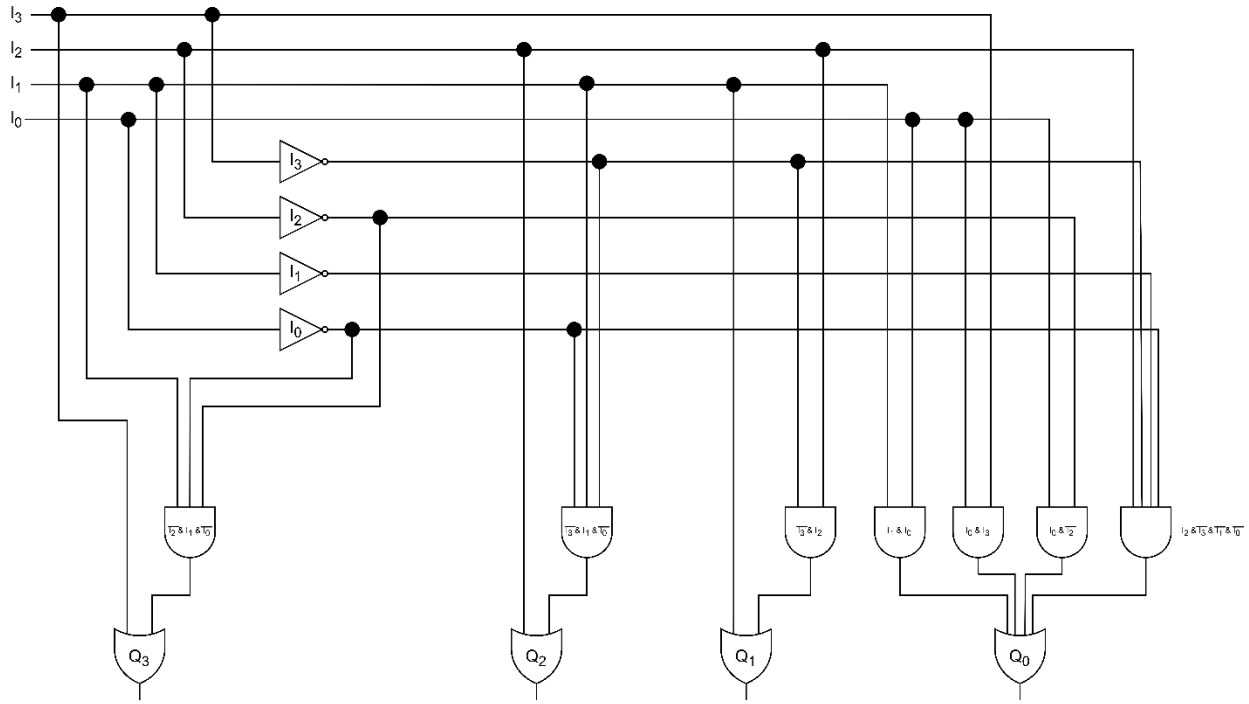

Fig. 20 Circuito Cambiador

Este circuito es el que se encargará de cambiar en el caso de que sea necesario, los bits recibidos para obtener la secuencia original del contador.

Como se puede ver usamos 4 puertas NOT para obtener los correspondientes valores negados, esto no sería necesario, ya que directamente podemos establecer 4 hilos más de entrada los cuales tengan la correspondiente salida negada que nos da el biestable. De hecho, sería una mejor decisión ya que nos ahorraríamos 4 puertas NOT. Lo hemos decidido representar usando puertas NOT para evitar liarnos con los cables. En código se ha implementado tomando las salidas negadas de los biestables.

Código desarrollado en Verilog

En esta sección vamos a poner diferentes partes del código desarrollado en Verilog que consideramos que es importante aclarar su funcionamiento.

Biestable JK

La implementación en Verilog del biestable JK ha sido la siguiente.

```
module JKDown(output reg Q, output wire nQ, input wire J, input wire K, input wire C);
    initial Q = 0;
    not(nQ, Q);

    always @(negedge C)
        case ({J, K})
            2'b10: Q = 1;
            2'b01: Q = 0;
            2'b11: Q = ~Q;
        endcase
endmodule
```

Fig. 21 Código del biestable

Este código se encargará de simular el comportamiento de un biestable JK que se actualiza en flancos de bajada del reloj. Se ha desarrollado usando la sentencia condicional *case* en la cual realizamos los cambios

pertinentes a Q dependiendo de J y K, siguiendo la tabla de transiciones que hicimos anteriormente para este biestable.

Como se puede ver a la hora de declarar el módulo se le tendrá que pasar 5 variables, las dos primeras de salida y las tres últimas de entrada.

- **Q:** Aquí se indicará a la variable en la cual se va a almacenar el valor de salida del registro Q. Esta variable almacenará el resultado de la operación que realizará el biestable cuando se le introduzcan valores a J y K y este en un flanco de bajada.
- **nQ:** Lo mismo que la variable Q, con la diferencia de que estará negada.
- **J:** Aquí se indicará el valor que va a tomar la entrada J del biestable para posteriormente realizar las operaciones pertinentes.
- **K:** El valor que va a tomar la entrada K del biestable para poder realizar las operaciones.
- **C:** Entrada del biestable la cual se va a encargar de que funcionamiento sea síncrono, siguiendo los ciclos de bajada del reloj.

La primera línea del módulo se encargará de inicializar la variable Q a 0 a la hora de crear el componente. Posteriormente inicializa el valor de nQ usando una puerta not.

Creamos un evento que se ejecutará siempre que la variable C se encuentre en un flanco de bajada. El código a ejecutar cada vez que se produzca este evento consiste básicamente en actualizar el valor de Q dependiendo los valores de J y K que se le hayan introducido al biestable.

Cambiador

```
module Cambiador(output wire [3:0] Q, input wire [3:0] I, input wire [3:0] nI);
    // Los cables necesario para realizar las operaciones definidas en los mapas de Karnaugh
    wire nI2andI1andnI0, I2andnI3, nI0andnI1andI2andnI3, nI2andI0, I3andI0andnI1, I3andI0, I1andI0;

    // Las operaciones AND obtenidas de los mapas de Karnaugh
    and(nI2andI1andnI0, nI[2], I[1], nI[0]);
    and(nI3andI1andnI0, nI[3], I[1], nI[0]);
    and(I2andnI3, I[2], nI[3]);
    and(nI0andnI1andI2andnI3, nI[0], nI[1], I[2], nI[3]);
    and(nI2andI0, nI[2], I[0]);
    and(I3andI0, I[3], I[0]);
    and(I1andI0, I[1], I[0]);

    // Las operaciones OR obtenidas de los mapas de Karnaugh
    // Como hemos decidido expresar la funcion booleana como suma de miniterminos
    // seran las ultimas operaciones a realizar.
    or(Q[3], I[3], nI2andI1andnI0);
    or(Q[2], I[2], nI3andI1andnI0);
    or(Q[1], I[1], I2andnI3);
    or(Q[0], nI0andnI1andI2andnI3, nI2andI0, I3andI0, I1andI0);
endmodule
```

Fig. 22 Código del Cambiador

El cometido de este módulo es el de cambiar los valores por los que se sustituyeron los valores repetidos por los originales, permitiéndonos obtener la secuencia inicial.

Este módulo solo recibe dos variables, una de salida y otra de entrada.

- **Q:** Es la variable donde se almacenará el nuevo número de 4 bits después de cambiarlo. Como se puede ver en la declaración tendrá un tamaño de 4 bits.
- **I:** Es la variable donde se pasará el numero de 4 bits a cambiar.
- **nI:** Es la variable donde se pasarán los valores negados de I.

Lo que son todas las puertas lógicas y operaciones a seguir se han tomado de las ecuaciones booleanas obtenidas al hacer el mapa de Karnaugh. Como se puede ver lo hemos hecho sumando los minitérminos, ya que las funciones booleanas las expresamos como suma de minitérminos. Debido a esto se puede ver como primero realizamos las operaciones AND y posteriormente finalizamos con las operaciones OR de cada bit. Siendo este el valor que almacenaremos en la variable de salida.

Contador Arbitrario

```
module ContadorArbitrario(output wire [3:0] SecuenciaSalida, input wire C);
    wire [3:0] nQ; // La variable donde almacenar la salida negada de los estados de los biestable.
    wire [3:0] Q; // La variable donde se almacenará la salida de los biestables.

    // Hilos necesarios para realizar las operaciones definidas al realizar los mapas de Karnaugh
    wire nQ1andQ2, Q0andnQ3, nQ1andQ3, nQ0andQ3;
    wire Q2andnQ0, nQ2andQ0, Q1andnQ3, nQ2andnQ1;
    wire nQ1andnQ0, Q3andnQ0, Q2andQ1;
    wire J2, J1, J0, K3, K1, K0;

    // Las puertas AND definidas en los mapas para el correcto funcionamiento del contador
    and(nQ1andQ2, nQ[1], Q[2]);
    and(Q0andnQ3, nQ[3], Q[0]);
    and(nQ1andQ3, nQ[1], Q[3]);
    and(nQ0andQ3, nQ[0], Q[3]);

    and(Q2andnQ0, Q[2], nQ[0]);
    and(nQ2andQ0, nQ[2], Q[0]);
    and(Q1andnQ3, Q[1], nQ[3]);
    and(nQ2andnQ1, nQ[2], nQ[1]);

    and(nQ1andnQ0, nQ[1], nQ[0]);
    and(Q3andnQ0, Q[3], nQ[0]);
    and(Q2andQ1, Q[2], Q[1]);

    // Las puertas OR que al definirse las funciones como suma de miniterminos seran las ultimas ejecutar.
    or(J2, Q0andnQ3, nQ1andQ3, nQ0andQ3);
    or(J1, nQ[3], Q2andnQ0, nQ2andQ0);
    or(J0, Q1andnQ3, nQ2andnQ1);
    or(K3, nQ1andnQ0, Q2andnQ0);

    or(K1, Q2andnQ0, Q3andnQ0);
    or(K0, nQ2andnQ1, Q[3], Q2andQ1);

    // Los 4 biestables necesarios para realizar el contador de 4 bits.
    JKDown JK0(Q[0], nQ[0], J0, K0, C);
    JKDown JK1(Q[1], nQ[1], J1, K1, C);
    JKDown JK2(Q[2], nQ[2], J2, nQ1andnQ0, C);
    JKDown JK3(Q[3], nQ[3], nQ1andQ2, K3, C);

    // El Cambiador del cual ya tomaremos la salida del contador
    Cambiador c(SecuenciaSalida, Q, nQ);
endmodule
```

Fig. 23 Código del contador arbitrario

Y por último el contador arbitrario usando los biestables JK y el cambiador mencionado previamente.

Resumidamente de lo que se encarga el circuito es de realizar la cuenta arbitraria sin números repetidos de la cual establecimos sus funciones booleanas al hacer los mapas de Karnaugh, para posteriormente pasar la salida al cambiador, obteniendo el valor correspondiente en cada caso. De tal forma que al final como valor de salida del contador iremos obteniendo los diferentes valores de salida de la cuenta con números repetidos.

Este módulo recibe dos variables, una de salida y otra de entrada.

- **SecuenciaSalida:** Es la variable donde se almacenará el números de 4 bits resultante de contador tratado por el cambiador, para obtener los valores originales.
- **C:** Es la variable que recibirá el ciclo de reloj actual, para hacer de nuestro contador un contador síncrono y así poder pasar este valor a los biestables.

Igual que en el caso del cambiador, se han realizado las funciones booleanas como sumas de minitérminos, de tal forma que las últimas operaciones que realizamos son las de la puerta OR.

Después de realizar todas las operaciones con puertas lógicas pertinentes, le pasamos a cada biestable los valores que este espera para poder conseguir que nuestro contador funcione correctamente.

Y, por último, pasamos esta salida obtenida de los biestable al cambiador, para obtener el valor correspondiente, el cual almacenaremos directamente en la salida, para poder tratarlo posteriormente.

Hemos puesto el cambiador como parte del contador ya que consideramos que forma parte de este, ya que es necesario para obtener la secuencia original.

Pruebas realizadas

A continuación, se indicarán las pruebas que hemos realizado sobre el contador y sus diferentes componentes para comprobar su correcto funcionamiento.

Pruebas para números fuera de la secuencia

Para comprobar que se llega correctamente a la secuencia desde los valores externos se ha creado un archivo de Verilog donde se va comprobando que, empezando desde uno de estos valores se llega correctamente a la secuencia.

```
$monitor($time, " | | %b(%d) ---> %b (%d)", aux, aux, Q, Q);
$dumpfile("contadorArbitrarioNumerosFueraSecuencia.dmp");
$dumpvars(2, contador);
C = 0;

$display("Empezamos en 0: ");

// Haremos que los estados de los biestable sea 0, para que empiece a contar desde el numero 0
// Se ha usado el operador <= en vez del = para que se hagan todas las asignaciones a la vez
// y evitar valores no deseados
contador.JK3.Q <= 0;
contador.JK2.Q <= 0;
contador.JK1.Q <= 0;
contador.JK0.Q <= 0;
#24

// Haremos que la secuencia empiece en el 1. Asignando el estado correspondiente a los
// diferentes biestables.
$display("Empezamos en 1: ");

contador.JK3.Q <= 0;
contador.JK2.Q <= 0;
contador.JK1.Q <= 0;
contador.JK0.Q <= 1;
#24

// Haremos que la secuancia empiece en 8
$display("Empezamos en 8: ");

contador.JK3.Q <= 1;
contador.JK2.Q <= 0;
contador.JK1.Q <= 0;
contador.JK0.Q <= 0;
#24

// Haremos que la secuancia empiece en 9
$display("Empezamos en 9: ");

contador.JK3.Q <= 1;
contador.JK2.Q <= 0;
contador.JK1.Q <= 0;
contador.JK0.Q <= 1;
#24

// Haremos que la secuancia empiece en 11
$display("Empezamos en 11: ");

contador.JK3.Q <= 1;
contador.JK2.Q <= 0;
contador.JK1.Q <= 1;
contador.JK0.Q <= 1;
#24

// Haremos que la secuancia empiece en 13
$display("Empezamos en 13: ");

contador.JK3.Q <= 1;
contador.JK2.Q <= 1;
contador.JK1.Q <= 0;
contador.JK0.Q <= 1;

#24
$dumpoff;
$finish;
```

Fig. 24 Código para probar números fuera de la secuencia

Como se puede ver en la parte de código dentro del begin lo que hacemos será almacenar las variables en un archivo para abrir con un visor que nos permita analizar cronogramas.

Luego, también tendremos un monitor en el cual iremos sacando por pantalla una parte de la secuencia del contador.

Y por último vamos asignando los valores deseados a los biestables para forzarles a empezar la secuencia por los valores que vamos a comprobar.

Obviamente falta la parte del código donde se declaran las variables así como se va actualizando la variable contador. No se ha introducido en la captura ya que nos ocuparía una hoja entera.

Una vez lo ejecutamos obtendremos la siguiente salida por pantalla.

```
Empezamos en 0:
                0 | | 0000( 0) ---> 0000 ( 0)
                2 | | 0011( 3) ---> 0011 ( 3)
                4 | | 0111( 7) ---> 0111 ( 7)
                6 | | 0110( 6) ---> 0110 ( 6)
```

Fig. 25 Valores para la entrada 0

Se puede ver que del 0 pasa al 3 como se indica en el diagrama de transiciones. Para luego continuar con la secuencia.

```
Empezamos en 1:
               24 | | 0001( 1) ---> 0001 ( 1)
               26 | | 0110( 6) ---> 0110 ( 6)
               28 | | 0101( 5) ---> 0110 ( 6)
               30 | | 1111(15) ---> 1111 (15)
```

Fig. 26 Valores para la entrada 1

Se puede ver que del 1 pasa al 6 como se indica en el diagrama de transiciones. Para luego continuar con la secuencia.

```
Empezamos en 8:
               48 | | 1000( 8) ---> 1000 ( 8)
               50 | | 0101( 5) ---> 0110 ( 6)
               52 | | 1111(15) ---> 1111 (15)
               54 | | 1110(14) ---> 1110 (14)
```

Fig. 27 Valores para la entrada 8

Se puede ver que del 8 pasa al 5 como se indica en el diagrama de transiciones. Ademas, se puede observar como el cambiador esta funcionado ya que se esta pasando del 5 al 6.

```
Empezamos en 9:
               72 | | 1001( 9) ---> 1001 ( 9)
               74 | | 1110(14) ---> 1110 (14)
               76 | | 0100( 4) ---> 0111 ( 7)
               78 | | 1010(10) ---> 1010 (10)
```

Fig. 28 Valores para la entrada 9

Se puede ver que del 9 pasa al 14 como se indica en el diagrama de transiciones. Para luego continuar con la secuencia.

```
Empezamos en 11:
               96 | | 1011(11) ---> 1011 (11)
               98 | | 1010(10) ---> 1010 (10)
              100 | | 1100(12) ---> 1100 (12)
              102 | | 0010( 2) ---> 1110 (14)
```

Fig. 29 Valores para la entrada 11

Se puede ver que del 11 pasa al 10 como se indica en el diagrama de transiciones. Para luego continuar con la secuencia.

```
Empezamos en 13:
              120 | | 1101(13) ---> 1101 (13)
              122 | | 1100(12) ---> 1100 (12)
              124 | | 0010( 2) ---> 1110 (14)
              126 | | 0011( 3) ---> 0011 ( 3)
              128 | | 0111( 7) ---> 0111 ( 7)
```

Fig. 30 Valores para la entrada 13

Se puede ver que del 13 pasa al 12 como se indica en el diagrama de transiciones. Para luego continuar con la secuencia.

Si tomamos la salida completa de alguno de estos casos podremos observar lo siguiente:

```
Empezamos en 0:
                0 | | 0000( 0) ---> 0000 ( 0)
                2 | | 0011( 3) ---> 0011 ( 3)
                4 | | 0111( 7) ---> 0111 ( 7)
                6 | | 0110( 6) ---> 0110 ( 6)
                8 | | 0101( 5) ---> 0110 ( 6)
               10 | | 1111(15) ---> 1111 (15)
               12 | | 1110(14) ---> 1110 (14)
               14 | | 0100( 4) ---> 0111 ( 7)
               16 | | 1010(10) ---> 1010 (10)
               18 | | 1100(12) ---> 1100 (12)
               20 | | 0010( 2) ---> 1110 (14)
               22 | | 0011( 3) ---> 0011 ( 3)
```

Fig. 31 Secuencia Completa

Como se puede ver en la imagen, la secuancia a seguir comienza en el tiempo 2 para posteriormente acabar en el 22, que es a partir del cual la secuancia se empieza a repetir.

Si nos fijamos en los valores vemos como la secuancia establecida se esta cumpliendo correctamente, tanto el de la secuancia con numeros repetidos como la que no los tiene.

3 → 7 → 6 → 6 → 15 → 14 → 7 → 10 → 12 → 14 → REPETIR

3 → 7 → 6 → 5 → 15 → 14 → 4 → 10 → 12 → 2 → REPETIR

Podemos ver como en los valors a la izquierda de la flecha se corresponden con los valores de la secuencia con valores no repetidos. Secuancia que al pasar por el cambiador obtendriamos los valores de la derecha de la flecha, correspondiendose con la secuencia original con numeros repetidos.

Simplemente con este archivo de prueba hemos comprobado lo siguiente:

1. Como los valores fuera de la secuencia no nos dan ningun tipo de error, ya que se contemplaron a la hora de hacer las tablas de transiciones.
2. Que la secuencia para la que hemos diseñado el contador es la esperada, ya que nos saca la secuencia de valores sin repetir con la que hemos diseñado los mapas de Karnaugh.
3. Que el cambiador funciona correctamente, ya que como se puede observar la secuancia una vez pasada por el cambiador nos dara la secuancia original con los valores repetidos.

El cronograma lo comentaremos mas adelante en la sección de cronogramas.

Cronogramas

A continuación, veremos el cronograma y se hará una breve explicación de lo que está ocurriendo en nuestro circuito.

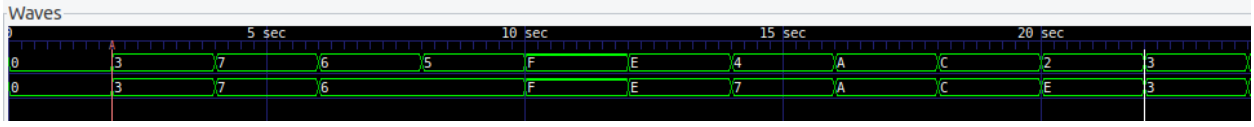

Fig. 32 Cronograma secuencia original y sin números repetidos

En esta primera imagen del cronograma podemos observar como la secuencia, tanto la que no tiene números repetidos como la original una vez tratada por el cambiador es la correcta, la que se encuentra en los dos cursores.

Veamos que está ocurriendo en cada salida individual.

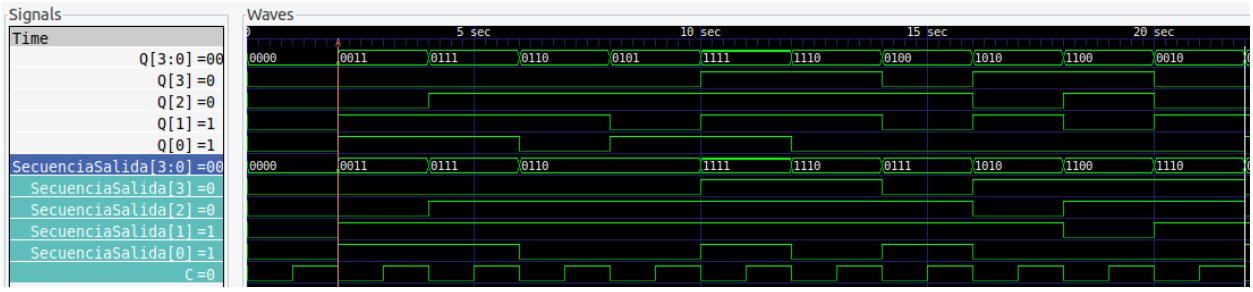

Fig. 33 Cronograma con las salidas bit a bit

Lo primero que se puede ver a simplemente vista del cronograma es que, efectivamente, los valores solo se actualizan cuando el contador (C) se encuentra en un flanco de bajada.

Ahora, analicemos el cómo de cada cambio en la secuencia aplicando las funciones booleanas obtenidas para cada bit.

- **De 0 (0000) a 3 (0011)**
  - Entrada J del biestable 3: Al aplicarse la función $\overline{Q_1} \cdot Q_2$ correspondiente a J3 obtendremos de salida un 0, ya que las correspondientes entradas serían 1 * 0, dando como resultado de la función un 0.
  - Entrada J del biestable 2: Al aplicarse la función $Q_0 \cdot \overline{Q_3} + \overline{Q_1} \cdot Q_3 + \overline{Q_0} \cdot Q_3$ correspondiente a J2 obtendremos la salida 0, ya que las correspondientes entradas serían 0 * 1 + 1 * 0 + 1 * 0 dando 0 + 0 + 0, lo cual como resultado daría 0.
  - Entrada J del biestable 1: Al aplicarse la función $\overline{Q_3} + Q_2 \cdot \overline{Q_0} + \overline{Q_2} \cdot Q_0$ correspondiente a J1 obtendremos de salida 1, ya que las correspondientes entradas serían 1 + 0 * 1 + 1 * 0 dando 1 + 0 + 0, lo cual como resultado daría 1.
  - Entrada J del biestable 0: Al aplicarse la función $Q_1 \cdot \overline{Q_3} + \overline{Q_2} \cdot \overline{Q_1}$ correspondiente a J0 obtendremos de salida 1, ya que las correspondientes entradas serían 0 * 1 + 1 * 1 dando 0 + 1, lo cual como resultado daría 1.

  Con lo cual, podemos ver que efectivamente se va a realizar un SET de los valores del biestable 0 y 1. Haciendo que pasen a almacenar el valor 1. Dando como salida 0011. No nos hace falta saber los valores de K de los biestables ya que una vez J valga 1 nos es indiferente el valor de K, ya que va a realizar el SET.

- **De 3 (0011) a 7 (0111)**
  - Entrada J del biestable 3: Al aplicar la función booleana del J3 obtendríamos de salida el valor 0, ya que 0 * 0 nos daría 0.
  - Entrada J del biestable 2: Al aplicar la función booleana del J2 obtendríamos de salida el valor 1, ya que 1 * 1 + 0 * 0 + 0 * 0 nos daría 1 + 0 + 0, dando como resultado 1.
  - Entrada K del biestable 1: En este caso vamos a ver directamente el valor de entrada de K1, ya que respecto al estado anterior este biestable debería mantener el estado 1. Para lo cual siguiendo la tabla de transiciones del biestable JK vemos que el valor K tiene que valer 0 para que se mantenga el 1. Y aplicando la función booleana $Q_2 \cdot \overline{Q_0} + Q_3 \cdot \overline{Q_0}$ nos saldría 0 * 0 + 0 * 0 lo cual daría como resultado 0, haciendo que, efectivamente, se mantenga el estado a 1 del biestable 1.

- Entrada K del biestable 0. En este caso se tendrá que cumplir lo mismo que en el caso anterior, que el valor de K0 valga 0 para mantener el estado a 1. Y aplicando la función $\overline{Q_2} \cdot \overline{Q_1} + Q_3 + Q_2 \cdot Q_1$ vemos como 1 * 0 + 0 + 0 * 1 nos daría 0, el resultado esperado.

Podemos ver como en este caso se han recibido las entradas necesarias tanto en J y K para mantener a 0 el primer bit, hacer un SET del segundo y mantener el estado a 1 de los dos últimos.

- **De 7 (0111) a 6 (0110)**
  - Entrada J del biestable 3: Como vemos seguimos necesitando que la entrada que vaya a recibir sea 0 para mantener su estado a 0. Y la función sería: 0 * 1 dando efectivamente 0.
  - Entrada K del biestable 2: Necesitaremos que el valor que recibe sea 0 para poder mantener el estado a 1. Y aplicando la función $\overline{Q_1} \cdot \overline{Q_0}$ saldría 0 * 0 que daría como resultado 0, manteniendo el estado a 1
  - Entrada K del biestable 1: Veremos que aplicando la función booleana saldrá 0 para poder mantener el estado a 1. Siendo esta 1 * 0 + 0 * 0 dando 0 como resultado. Manteniendo el estado a 1.
  - Entrada K del biestable 0: En este caso, para hacer RESET del estado y ponerlo a 0 necesitaremos que la entrada que recibe K0 sea 1. Y aplicando la función 0 * 0 + 0 + 1 * 1 saldría 0 + 0 + 1 dando como resultado 1, lo cual haría un RESET, poniendo el estado a 0.

A partir de este estado no nos queda por comprobar ninguna posibilidad más dentro de las que nos permiten los biestables, ya que hemos abarcado tanto el SET, como RESET y mantener el estado.

- **De 6 (0110) a 5 (0101)**
  - Entrada J del biestable 3: Tendrá que tomar el valor de 0 para mantener el valor. Y aplicando la función saldría 0 * 1 dando como resultado 0.
  - Entrada K del biestable 2. Tendrá que tomar el valor de 0 para mantener el estado a SET. 0 * 1 = 0.
  - Entrada K del biestable 1. Tendrá que tomar el valor de 1 para hacer un RESET y cambiar el estado a 0. 1 * 1 + 0 * 1 = 1.
  - Entrada J del biestable 0. Tendrá que tomar el valor de 1 para hacer un SET y cambiar el estado a 1. 1 * 1 + 0 * 0 = 1.

En este caso se puede observar como el valor en Q[3:0] del cronograma vale 0101, mientras que en SecuanciaSalida[3:0] vale 0110. Esto es debido a que el cambiador está aplicando correctamente las funciones booleanas para obtener la secuencia de salida deseada. Siendo estas funciones las siguientes.

- Entrada $I3$ primer bit (más a la izquierda): Aplicando la función booleana para este bit $I_3 + \overline{I_2} \cdot I_1 \cdot \overline{I_0}$ obtendremos 0 + 0 * 0 * 0 obteniendo 0 de resultado.
- Entrada $I2$ segundo bit: Aplicando la función booleana para este bit $I_2 + \overline{I_3} \cdot I_1 \cdot \overline{I_0}$ obtendremos 1 + 1 * 0 * 0 obteniendo 1 de resultado.
- Entrada $I1$ tercer bit: Aplicando la función booleana para este bit $I_1 + I_2 \cdot \overline{I_3}$ obtendremos 0 + 1 * 1 obteniendo 1 de resultado.

- Entrada I0 ultimo bit: Aplicando la función booleana para este bit $I_1 \cdot I_0 + I_0 \cdot I_3 + I_0 \cdot \overline{I_2} + I_2 \cdot \overline{I_3} \cdot \overline{I_1} \cdot \overline{I_0}$ obtendremos 0 * 1 + 1 * 0 + 1 * 0 + 1 * 1 * 1 * 0 dando 0 + 0 + 0 + 0 teniendo como resultado 0.

Y así obtendremos de salida 0110 para una entrada 0101. Pasando así del 6 al 5.

- **De 5 (0101) a 15 (1111)**
  - Entrada J del biestable 3: Tendrá que tomar el valor 1 para hacer el SET. Y aplicando la función $\overline{Q_1} \cdot Q_2$ con los valores 1 * 1 saldría 1 como resultado.
  - Entrada K del biestable 2: Tendrá que tomar el valor 0 para mantener el estado a 1. Y aplicando la función $\overline{Q_1} \cdot \overline{Q_0}$ nos quedaría 1 * 0 dando 0 como resultado.
  - Entrada J del biestable 1: Tendrá que tomar el valor 1 para hacer el SET. Y aplicando la función $\overline{Q_3} + Q_2 \cdot \overline{Q_0} + \overline{Q_2} \cdot Q_0$ con los valores 1 + 1 * 0 + 0 * 1 saldría 1 como resultado.
  - Entrada K del biestable 0: Tendrá que tomar le valor 0 para mantener el estado a 1. Y aplicando la función $\overline{Q_2} \cdot \overline{Q_1} + Q_3 + Q_2 \cdot Q_1$ nos quedaría 0 * 1 + 0 + 1 * 0 dando 0 como resultado.
- **De 15 (1111) a 14 (1110)**
  - Entrada K del biestable 3: Tendrá que tomar el valor 0 para mantener el estado a 1. Por lo tanto, la función $\overline{Q_1} \cdot \overline{Q_0} + Q_2 \cdot \overline{Q_0}$ para los valores 0 * 0 + 1 * 0 dará 0 como resultado.
  - Entrada K del biestable 2: Tendrá que tomar el valor 0 para mantener le estado a 1. Por lo tanto, la función $\overline{Q_1} \cdot \overline{Q_0}$ para los valores 0 * 1 dará 0 como resultado.
  - Entrada K del biestable 1: Tendrá que tomar el valor 0 para mantener el estado a 1. Por lo tanto, la función $Q_2 \cdot \overline{Q_0} + Q_3 \cdot \overline{Q_0}$ para los valores 1 * 0 + 1 * 0 dará 0 como resultado.
  - Entrada K del biestable 0: Tendrá que tomar el valor 1 para mantener hacer un RESET y establecer el estado a 0. Por lo tanto, la función $\overline{Q_2} \cdot \overline{Q_1} + Q_3 + Q_2 \cdot Q_1$ para los valores 0 * 0 + 1 + 1 * 1 dará 1 como resultado.
- **De 14 (1110) a 4 (0100)**
  - Entrada K del biestable 3: Tendrá que tomar el valor 1 para hacer un RESET y cambiar el estado a 0. Por lo tanto, la función $\overline{Q_1} \cdot \overline{Q_0} + Q_2 \cdot \overline{Q_0}$ para los valores 0 * 1 + 1 * 1 dará q 1 como resultado.
  - Entrada K del biestable 2: Tendrá que tomar el valor 0 para mantener el estado a 1. Por lo tanto, la función $\overline{Q_1} \cdot \overline{Q_0}$ para los valores 0 * 1 dará 0 como resultado.
  - Entrada K del biestable 1: Tendrá que tomar le valor 1 para hacer un RESET y cambiar el estado a 0. Por lo tanto, la función $Q_2 \cdot \overline{Q_0} + Q_3 \cdot \overline{Q_0}$ para los valores 1 * 1 + 1 * 1 dará 1 como resultado.
  - Entrada J del biestable 0: Tendrá que tomar el valor 0 para mantener el estado a 0. Por lo tanto, la función $Q_1 \cdot \overline{Q_3} + \overline{Q_2} \cdot \overline{Q_1}$ para los valores 1 * 0 + 0 * 0 dará 0 como resultado.

En este caso, al igual que en el caso del 5 (0101) el cambiador va a realizar un cambio del número de 4 bits 0100 a 0111 siguiendo los siguientes pasos.

- Entrada I3 primer bit (más a la izquierda): Aplicando la función $I_3 + \overline{I_2} \cdot I_1 \cdot \overline{I_0}$ para los valores 0 + 0 * 0 * 1 obtendríamos 0 como resultado.
- Entrada I2 segundo bit: Aplicando la función $I_2 + \overline{I_3} \cdot I_1 \cdot \overline{I_0}$ para los valores 1 + 1 * 0 * 1 obtendríamos 1 como resultado.

- Entrada $I1$ tercer bit: Aplicando la función $I_1 + I_2 \cdot \overline{I_3}$ para los valores 0 + 1 * 1 obtendríamos 1 como resultado.
- Entrada $I0$ ultimo bit: Aplicando la función $I_1 \cdot I_0 + I_0 \cdot I_3 + I_0 \cdot \overline{I_2} + I_2 \cdot \overline{I_3} \cdot \overline{I_1} \cdot \overline{I_0}$ para los valores 0 * 0 + 0 * 0 + 0 * 0 + 1 * 1 * 1 * 1 obtendremos 1 como resultado.

Y una vez agrupamos estos valores en el orden correspondientes obtendremos 0111 (7), el valor de la secuencia original.

- **De 4 (0100) a 1010 (10):**
  - Entrada J del biestable 3: Tendrá que tomar el valor 1 para hacer un SET y establecer el estado del biestable a 1. Por lo tanto, la función $\overline{Q_1} \cdot Q_2$ para los valores 1 * 1 dará 1 como resultado.
  - Entrada K del biestable 2: Tendrá que tomar el valor 1 para hacer un RESET y establecer el estado del biestable a 0. Por lo tanto, la función $\overline{Q_1} \cdot \overline{Q_0}$ para los valores 1 * 1 dará 1 como resultado.
  - Entrada J del biestable 1: Tendrá que tomar el valor 1 para hacer un SET y establecer el estado del biestable a 1. Por lo tanto, la función $\overline{Q_3} + Q_2 \cdot \overline{Q_0} + \overline{Q_2} \cdot Q_0$ para los valores 1 + 1 * 1 + 0 * 0 dará 1 como resultado.
  - Entrada J del biestable 0: tendrá que tomar el valor 0 para mantener el estado a 0. Por lo tanto, la función $Q_1 \cdot \overline{Q_3} + \overline{Q_2} \cdot \overline{Q_1}$ para los valores 0 * 1 + 0 * 1 dará 0 como resultado.
- **De 10 (1010) a 12 (1100)**
  - Entrada K del biestable 3. Tendrá que tomar el valor 0 para mantener el estado a 1. Por lo tanto, la función $\overline{Q_1} \cdot \overline{Q_0} + Q_2 \cdot \overline{Q_0}$ para los valores 0 * 1 + 0 * 1 dará 0 como resultado.
  - Entrada J del biestable 2. Tendrá que tomar el valor 1 para hacer un SET y cambiar el estado a 1. Por lo tanto, la función $Q_0 \cdot \overline{Q_3} + \overline{Q_1} \cdot Q_3 + \overline{Q_0} \cdot Q_3$ para los valores 0 * 0 + 0 * 1 + 1 * 1 dará 1 como resultado.
  - Entrada K del biestable 1. Tendrá que tomar el valor 1 para hacer un RESET y cambiar el estado a 0. Por lo tanto, la función $\overline{Q_2} \cdot \overline{Q_1} + Q_3 + Q_2 \cdot Q_1$ para los valores 1 * 0 + 1 + 0 * 1 dará 1 como resultado.
  - Entrada J del biestable 0. Tendrá que tomar el valor 0 para mantener el estado del biestable a 0. Por lo tanto, la función $Q_1 \cdot \overline{Q_3} + \overline{Q_2} \cdot \overline{Q_1}$ para los valores 1 * 0 + 1 * 0 dará 0 como resultado.
- **De 12 (1100) a 2 (0010)**
  - Entrada K del biestable 3. Tendrá que tomar el valor 1 para hacer un RESET y cambiar el estado a 0. Por lo tanto, la función $\overline{Q_1} \cdot \overline{Q_0} + Q_2 \cdot \overline{Q_0}$ para los valores 1 * 1 + 1 * 1 dará 1 como resultado.
  - Entrada K del biestable 2. Tendrá que tomar el valor 1 para hacer un RESET y cambiar el estado a 0. Por lo tanto, la función $\overline{Q_1} \cdot \overline{Q_0}$ para los valores 1 * 1 dará 1 como resultado.
  - Entrada J del biestable 1. Tendrá que tomar el valor 1 para hacer un SET y cambiar el estado a 1. Por lo tanto, la función $\overline{Q_3} + Q_2 \cdot \overline{Q_0} + \overline{Q_2} \cdot Q_0$ para los valores 0 + 1 * 1 + 0 * 0 dará 1 como resultado.
  - Entrada J del biestable 0. Tendrá que tomar el valor 0 para mantener el estado a 0. Por lo tanto, la función $Q_1 \cdot \overline{Q_3} + \overline{Q_2} \cdot \overline{Q_1}$ para los valores 0 * 0 + 0 * 1 dará 0 como resultado.

En este caso, igual que en los anteriores el cambiador va a realizar un cambio del número de 4 bits 0010 a 1110 siguiendo los siguientes pasos.

- Entrada $I3$ primer bit (más a la izquierda): Aplicando la función $I_3 + \overline{I_2} \cdot I_1 \cdot \overline{I_0}$ para los valores 0 + 1 * 1 * 1 dará 1 como resultado.
- Entrada $I2$ segundo bit: Aplicando la función $I_2 + \overline{I_3} \cdot I_1 \cdot \overline{I_0}$ para los valores 0 + 1 * 1 * 1 dará 1 como resultado.
- Entrada $I1$ tercer bit: Aplicando la función $I_1 + I_2 \cdot \overline{I_3}$ para los valores 1 + 0 * 1 dará 1 como resultado.
- Entrada $I0$ ultimo bit: aplicando la función $I_1 \cdot I_0 + I_0 \cdot I_3 + I_0 \cdot \overline{I_2} + I_2 \cdot \overline{I_3} \cdot \overline{I_1} \cdot \overline{I_0}$ para los valores 1 * 0 + 0 * 0 + 0 * 1 + 0 * 1 * 0 * 1 dará 0 como resultado.

Y una vez agrupamos estos valores en el orden correspondiente obtendremos el numero de 4 bits 1110 (14)

- **De 2 (0010) a 3 (0011)**
  - Entrada J del biestable 3. Tendrá que tomar el valor 0 para mantener el estado a 0. Por lo tanto, la función $\overline{Q_1} \cdot Q_2$ para los valores 0 * 0 dará 0 como resultado.
  - Entrada J del biestable 2. Tendrá que tomar el valor 0 para mantener el estado a 0. Por lo tanto, la función $Q_0 \cdot \overline{Q_3} + \overline{Q_1} \cdot Q_3 + \overline{Q_0} \cdot Q_3$ para los valores 0 * 1 + 0 * 0 + 1 * 0 dará 0 como resultado.
  - Entrada K del biestable 1. Tendrá que tomar el valor 0 para mantener el estado a 1. Por lo tanto, la función $\overline{Q_2} \cdot \overline{Q_1} + Q_3 + Q_2 \cdot Q_1$ para los valores 1 * 0 + 0 + 0 * 1 dará 0 como resultado.
  - Entrada J del biestable 0. Tendrá que tomar el valor 1 para hacer un SET y cambiar el estado del biestable a 1. Por lo tanto, la función $Q_1 \cdot \overline{Q_3} + \overline{Q_2} \cdot \overline{Q_1}$ para los valores 1 * 1 + 1 * 0 dará 1 como resultado.

Y a partir de este valor de 4 bits se repetiría toda la secuencia explicada previamente.

Anexo 1. Circuito usando puertas NAND

Se ha desarrollado el mismo circuito usando exclusivamente puertas NAND, de esta forma se simplificará el montaje del circuito y se reducirán los costes, ya que los circuitos integrados que tienen exclusivamente puertas NAND tienen un coste menor que los que tienen puertas AND o OR. Por eso hemos hecho este circuito que se asemejaría mas a uno real, para el cual hemos usado 5 circuitos integrados CD4011BE con 4 puertas NAND de 2 entradas, un circuito SN74LS20N con 2 puertas NAND de 4 entradas y dos circuitos CD4023BMT con 3 puertas NAND de 3 entradas. Ya que contamos con 19 puertas NAND de dos entradas, dos puertas NAND de 4 entradas y 5 puertas NAND de 3 entradas.

Lo cual todo nos saldría por 0,319 euros cada CD4023BMT, 1,0915 euros cada SN74LS20N y 0,634 euros cada CD4011BE. Costándonos un total de 5,218 euros.

Con este presupuesto podríamos desarrollar tanto el contador como el cambiador usando puertas NAND.

Ahora veamos como implementaríamos los circuitos usando puertas AND y OR.

Necesitaríamos 4 circuitos integrados SN74LS08N (1,16 euros) con 4 puertas AND de 2 entradas cada una, un circuito integrado SN74LS11N (0,855 euros) con 3 puertas AND de 3 entradas cada una, un circuito integrado CD4082BE (0,52 euros) con 2 puertas AND de 4 entradas cada una, 2 circuitos integrados CD4071BE (0,634 euros) con 4 puertas OR con 2 entradas cada una, un circuito CD4075BM96 (0,442 euros) con 3 puertas OR con 3 entradas cada una y un circuito integrado CD4072BE (0,615 euros) con 2 puertas OR con 4 entradas cada una. Ya que contamos con 15 puertas AND de 2 entradas, 2 puertas AND de 3 entradas, 1 puerta AND de 4 entradas, 6 puertas OR de 2 entradas, 3 puertas OR de 3 entradas y 1 puerta OR de 4 entradas.

Lo cual nos saldría por $4 * 1{,}16 + 1 * 0{,}855 + 1 * 0{,}52 + 2 * 0{,}634 + 1 * 0{,}442 + 1 * 0{,}615$ quedándonos un total de 8,34 euros o lo que es lo mismo, nos saldría 3,13 euros mas caro que el circuito desarrollado con puertas NAND. Además que usando puertas NAND requerimos de menos circuitos integrados que usando puertas AND y OR.

Los circuitos integrados elegidos se han elegido a modo de estimación, para poder hacernos una idea de cual seria el precio total por el que nos saldría implementar el circuito. Ya que como se puede ver en los datasheets de estos componentes que se encuentran en la carpeta *datasheets* muchos de los elegidos tienen las patillas para insertar y otros tienen patillas para soldar. A la hora de realizar una implementación real se elegirían los chips correspondiente a nuestra necesidad. EN nuestro caso como solo queríamos hacer una estimación del precio no hemos tenido en cuenta este factor, ya que no vamos a implementarlo físicamente.

Versión en Verilog de los circuitos usando NAND

```
module ContadorArbitrarioNAND(output wire [3:0] SecuenciaSalida, input wire C);
    wire [3:0] nQ;
    wire [3:0] Q;
    wire J3, J2, J1, J0, K3, K2, K1, K0;
    wire Q2nandnQ1, Q0nandnQ3, nQ1nandQ3, nQ0nandQ3, Q2nandnQ0, nQ2nandQ0, Q1nandnQ3, nQ2nandnQ1, nQ0nandnQ1, Q2nandQ1;

    // J3
    nand(Q2nandnQ1, Q[2], nQ[1]);
    nand(J3, Q2nandnQ1);

    // J2
    nand(Q0nandnQ3, Q[0], nQ[3]);
    nand(nQ1nandQ3, nQ[1], Q[3]);
    nand(nQ0nandQ3, nQ[0], Q[3]);
    nand(J2, Q0nandnQ3, nQ1nandQ3, nQ0nandQ3);

    // J1
    nand(Q2nandnQ0, Q[2], nQ[0]);
    nand(nQ2nandQ0, nQ[2], Q[0]);
    nand(J1, Q2nandnQ0, Q[3], nQ2nandQ0);

    // J0
    nand(Q1nandnQ3, Q[1], nQ[3]);
    nand(nQ2nandnQ1, nQ[2], nQ[1]);
    nand(J0, Q1nandnQ3, nQ2nandnQ1);

    // K3
    nand(nQ0nandnQ1, nQ[0], nQ[1]);
    nand(K3, Q2nandnQ0, nQ0nandnQ1);

    // K2
    nand(K2, nQ0nandnQ1);

    // K1
    nand(K1, Q2nandnQ0, nQ0nandQ3);

    // K0
    nand(Q2nandQ1, Q[2], Q[1]);
    nand(K0, nQ[3], Q2nandQ1, nQ2nandnQ1);

    JKDown JK0(Q[0], nQ[0], J0, K0, C);
    JKDown JK1(Q[1], nQ[1], J1, K1, C);
    JKDown JK2(Q[2], nQ[2], J2, K2, C);
    JKDown JK3(Q[3], nQ[3], J3, K3, C);

    Cambiadornand c(SecuenciaSalida, Q, nQ);
endmodule
```

Fig. 34 Código Contador Puertas NAND

La funcionalidad no será distinta al código realizado usando puertas AND y OR, simplemente cambiará la implementación física.

```
module Cambiadornand(output wire [3:0] O, input wire [3:0] I, input wire [3:0] nI);
    // Los cables necesario para realizar las operaciones definidas en los mapas de Karnaugh
    wire nI2andI1andnI0, I2andnI3, nI0andnI1andI2andnI3, nI2andI0, I3andI0andnI1, I3andI0, I1andI0;

    // Las operaciones NAND obtenidas de los mapas de Karnaugh y la simplificacion desde miniterminos

    //O3
    nand(nI2andI1andnI0, nI[2], I[1], nI[0]);
    nand(O[3], nI[3], nI2andI1andnI0);

    //O2
    nand(nI3andI1andnI0, nI[3], I[1], nI[0]);
    nand(O[2], nI[2], nI3andI1andnI0);

    //O1
    nand(I2andnI3, I[2], nI[3]);
    nand(O[1], nI[1], I2andnI3);

    //O0
    nand(nI0andnI1andI2andnI3, nI[0], nI[1], I[2], nI[3]);
    nand(nI2andI0, nI[2], I[0]);
    nand(I3andI0, I[3], I[0]);
    nand(I1andI0, I[1], I[0]);
    nand(O[0], nI0andnI1andI2andnI3, nI2andI0, I3andI0, I1andI0);
endmodule
```

Fig. 35 Código Cambiador Puertas NAND

Aquí lo mismo, la funcionalidad no variara respecto a su homologo usando puertas AND y OR. Simplemente variara la implementación física.

Bibliografía

- Aplicación WEB usada para realizar los diagramas: draw.io
- Página web usada para la búsqueda de los chips: rs-online
- Página web de la que se han tomado los datasheets: Alldatasheet
- Página web usada de guía y para resolver algunas dudas a la hora de realizar el proyecto: avellano