# TOLERANCIA A FALLOS TRANSITORIOS

Intro: nosotros solamente nos centamos en fallos tansitorios en este trabajo

## Como se consigue la tolerancia

Para evitar que los fallos transitorios se conviertan en errores y provoquen circunstancias imprevistas usaremos el método de “Triple modular Redundancy”.

El método aplicado consiste en triplicar el módulo, (o su funcionalidad modificando el circuito interno, pero manteniendo los resultados esperados), y utilizar un sistema votador para detectar, y si es posible enmascarar el fallo evitando que se produzca un error.

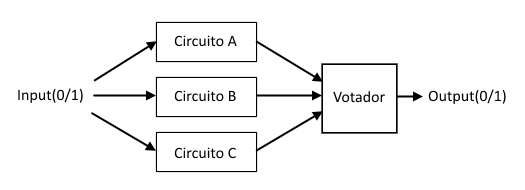


Figura 1. Sistema usando un Votador

El votador usado se conoce como “Majority Gate”. Como podemos ver en la “Figura 1”, recibe tres señales de tres módulos que aplican la misma función al mismo conjunto de valores. Las salidas de estos módulos deberían ser idénticas, salvo que se produzca un fallo en alguna puerta o biestable de los módulos.

El votador facilita la tarea de detectar las discrepancias entre los módulos debido a un fallo, y corregirlo antes de que se genere un error. Su funcionamiento se explica más detalladamente en el siguiente apartado.

## Votadores

Un votador es un sistema que recibe N entradas y tiene una salida coincidente con la mayoría de entradas.

Los votadores no están libres de fallos, ya que estos mismos pueden sufrir la radiación y provocarlos. Estos votadores están diseñados para intentar evitar un número limitado de fallos en los módulos repetidos dado por la función donde M es el número de fallos tolerados y N es el número de módulos repetidos, N debe ser un número impar.

La "Tabla de Verdad 1" muestra los valores de salida dependiendo de las entradas del Votador.

El "Mapa de Karnaught 1" es el correspondiente a la "Tabla de Verdad 1" y podemos obtener la función F(ABC) = AB + BC + AC para los valores de salida del votador.

La es el esquema del votador obtenido como resultado del diseño anterior.

//Dibujo de vtador 3 entradas de 1 bit

//Tala de verdad del votador, diagrama de karnaught e implementación final con puertas lógicas

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | C | F(ABC) |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Tabla de Verdad . Votador.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | AB |  |  |  |  |
| C |  | 00 | 01 | 11 | 10 |
|  | 0 | 0 | 0 | 1 | 0 |
|  | 1 | 0 | 1 | 1 | 1 |

Mapa de Karnaught . Votador.

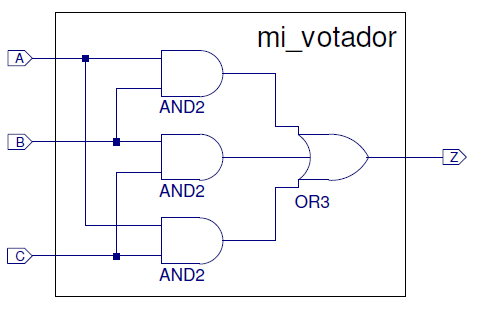
****

Figura 1. Diseño de votador con puertas lógicas.

## Configuración del entorno Xilinx

El software de Xilinx tiene por defecto una serie de opciones activadas para ayudar a que nuestro diseño ocupe la menor cantidad de silicio posible. Además de compactar y simplificar el diseño todo lo posible, también elimina partes innecesarias para el funcionamiento del mismo. Esto también incluye el hardware duplicado como en nuestro caso son los módulos triplicados.



Figura . Esquema simplificado

Si dejamos que el Xilinx simplifique nuestra lógica obtendremos algo parecido a la Figura 2, una simplificación automática para mantener la funcionalidad y disminuir retardos y el número de transistores. Esto no es lo que queremos ya que elimina toda nuestra lógica de detección y corrección de fallos transitorios.

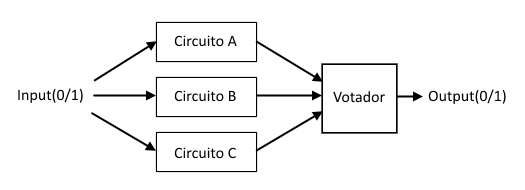


Figura . Esquema deseado

Nuestro objetivo es tener estos módulos varias veces y necesitamos que Xilinx no se tome la libertad de eliminarlos de nuestro diseño. Para ello vamos a configurar el programa para que actúe como queremos. Para obtener el diseño deseado de la Figura 3, debemos configurar el software de Xilinx para que no realice esta tarea de optimización.

### Desactivar eliminación de registros equivalentes

En este pequeño apartado voy a explicar qué opciones cambiar para que el software de Xilinx no nos cause un problema a la hora de probar nuestro diseño. El problema que tenemos, explicado anteriormente, es que modifica nuestro diseño eliminando módulos y biestables necesarios para nuestro propósito.

#### C:\TFG\TFG\Memoria\Imagenes\usar\3.3 Configuracion xilinx\DuplicateRegister_0.pngPaso 1)

Figura . Configuración, paso 1

El primer paso es asegurarnos de que tenemos seleccionado el fichero referente al módulo principal de nuestro proyecto, o el que esté seleccionado como “Top module” (Figura 4).

Aquel que nos permita ejecutar las acciones de sintetizar, implementar el diseño y generar el fichero programable en la FPGA.

#### C:\TFG\TFG\Memoria\Imagenes\usar\3.3 Configuracion xilinx\DuplicateRegister_0.png Paso 2)

Figura . Configuración, paso 2

En la sección de procesos, seleccionamos “Synthesize” (Figura 5). Esto nos permitirá ver las opciones adecuadas en el siguiente paso.

Si está seleccionado cualquier otro ítem de la lista nos mostrará un menú distinto y no podremos cambiar la configuración adecuadamente.

O simplemente no podremos seleccionar el menú de propiedades.

#### Paso 3)

Para el mostrar el menú de opciones de configuración tenemos dos alternativas.

La alternativa A (Figura 6) consiste en hacer “click izquierdo” sobre el ítem seleccionado en el paso 2. Y elegir el menú “Process Properties…”

La alternativa B (Figura 7), en la barra de menús abrimos la pestaña “Process” y elegimos el mismo menú “Process Properties…”

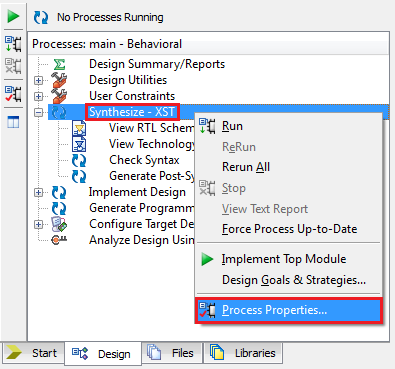
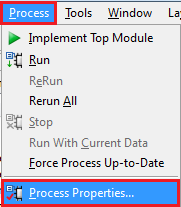


Figura . Configuración, paso 3 Alternativa A

Figura . Configuración, paso 3 Alternativa B

#### C:\TFG\TFG\Memoria\Imagenes\usar\3.3 Configuracion xilinx\DuplicateRegister_0.pngPaso 4)

Al terminar el paso 3 nos aperece una nueva ventana de configuración. Debemos seleccionar la categoría “Xilinx Specific Options”.

La propiedad que debemos cambiar se llama “Equivalent Register Removal”, por defecto está habilitada, debemos asegurarnos de deshabilitarla para que no nos desaparezcan los registros duplicados.

Una vez deshabilitada seleccionamos el botón “OK” para aplicar los cambios.

#### Resultado:

Una vez aplicados los cambios debemos sintetizar el diseño y abrir el esquema tecnológico que nos facilita el programa. ~~El esquema RTL no nos sirve para ver los cambios aplicados ya que solo muestra el diseño sin los cambios aplicados.~~ En la Figura 8 y en la Figura 9 podemos ver las diferencias entre tener habilitada la opción de optimización y deshabilitarla con un ejemplo sencillo.

La Figura 8 muestra el diseño simplificado por la herramienta de Xilinx. Podemos observar que se compone de una LUT con nuestra lógica combinacional y su salida está conectada a un único biestable (bloque rojo).

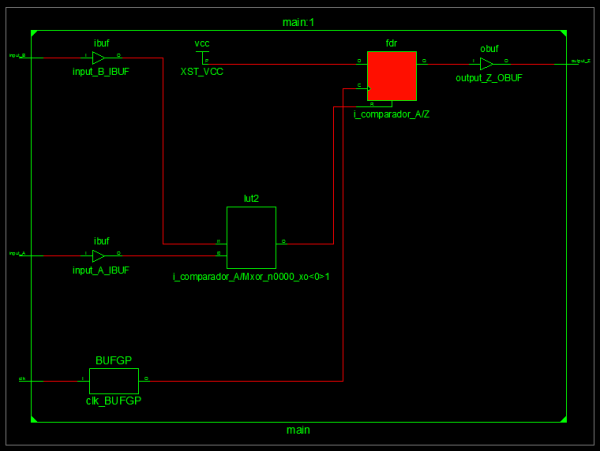
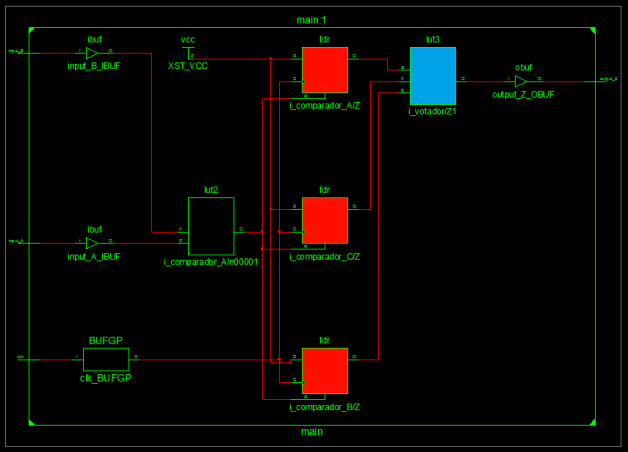
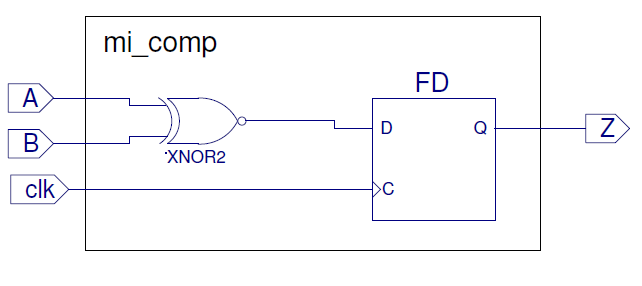
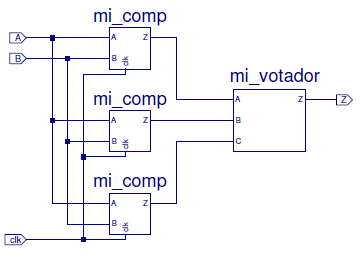
Sin embargo en la Figura 9 vemos la misma LUT con la misma lógica, conectada a los tres biestables (bloques rojos) que queríamos en un principio, y observamos que en el diseño anterior se había eliminado nuestro votador (bloque azul), esto eliminaba todo el propósito de nuestro trabajo.

Figura . Esquema tecnológico simplificado

Figura . Esquema tecnológico triplicado con votador

TAREA





Implementas en vhdl un modulo sencillo con un bistble a la salida

modulo principal contiene 3 copias del modulo anterior conectados al votador

configuracion XILINX, busca como no eliminar elementos repetidos, poreque este diseño eliminaria los modulos repetidos y entonces tambien eliminaria elvotador

TAREA

## injeccion fallos

cada modulo sencillo tiene 3 lineas invertir salida, salida a 0, salida a 1

### MODO POST-INJECION

esto se hace mediante logica combinaional a la salida del biestable, para no introducir otro biestable que cambiaria la temporrizacion en etapas del procesador

### MODO PRE-INJECCION

en paralelo a la generacion de la entrada al biestable tambien funciona la lgica de injecciond efallo,

quizas el modo anterior introduce un retardo para las sieuigente etapa

este quizas iintroduce menos