



Módulo 8: Mecanismos de mitigación de fallas

Contenidos del módulo 8

- SED, SEC, SECDDED, etc.
- Redundancia triple modular
- Scrubbing

FSM robustas

La robustez de una máquina de estados está definida por la capacidad de evitar secuencias indeseadas en el caso de SEUs

El estado de una FSM está definido por el valor de los registros de estado, y la redundancia empleada depende de la forma de codificación

Se analizan los casos

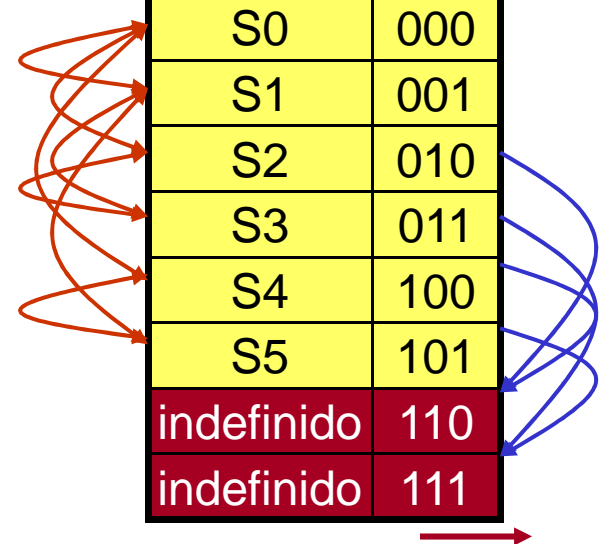
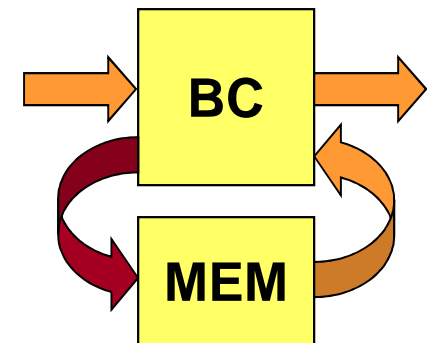
- Binario: es el caso de menor uso de registros, pero de menor capacidad de control de robustez
- One Hot: el uso de un único flipflop por estado simplifica la lógica combinatoria usada
- Hamming de distancia 2: caso binario al que se agrega la capacidad de detectar SEUs
- Hamming de distancia 3: caso binario al que se agrega la capacidad de corregir SEUs (SECDED: Single Error Correct, Double Error Detect)

***“Fault Tolerant State Machines”,
G.Burke & S.Faft, CalTech JPL
MAPLD 2004***

FSM robustas: el caso binario

- Una máquina de estados con codificación binaria es quien menos robustez ofrece, su única ventaja es que al tener menos registros, el área ocupada es menor, y por tanto menor la posibilidad de un SEU.
- La única consideración a tener en cuenta en el diseño es agregar la capacidad de detectar si a causa de un SEU la FSM cae en alguno de los estados indefinidos (*flechas azules*) y en ese caso generar la acción de recuperación que convenga:
 - Una reinicialización de la FSM
 - Un aviso de SEU al sistema
- Un SEU que produzca el cambio de un estado válido por otro estado válido es indetectable (*flechas rojas*), sólo puede salvarse usando TMR

Estado	Cod
S0	000
S1	001
S2	010
S3	011
S4	100
S5	101
indefinido	110
indefinido	111

FSM robustas: codificación Hamming de distancia 2

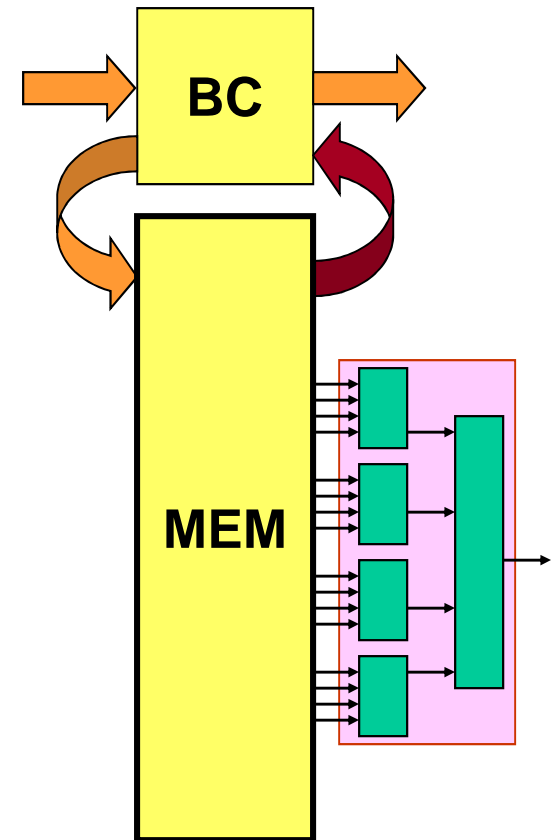
- Similar al caso binario al que se agrega la capacidad de detectar SEUs mediante el agregado de un registro de estado adicional
- Se llama SED por *Single Error Detection*
- Ese bit adicional permite que todos los códigos válidos posean entre sí al menos dos bits de diferencia (distancia de Hamming=2)
- Y de hecho este bit es un bit de paridad
- Dado que un SEU cambia sólo un bit, no puede cambiar un estado válido por otro estado válido, por lo que puede ser detectado y generar las acciones de recuperación
- Pero no permite corregir el SEU, y puede ignorar un MBU

No más
flechas
rojas!!

Estado	Cod
S0	0000
S1	0011
S2	0101
S3	0110
S4	1001
S5	1010
indefinido	0001
indefinido	0010
indefinido	0100
indefinido	0111
indefinido	1000
indefinido	1011
indefinido	1101
indefinido	1110
indefinido	1100
indefinido	1111

FSM robustas: codificación ONE-HOT

- El uso de un único flipflop por estado simplifica la lógica combinatoria usada, y por tanto el área sensible a SETs
- La posibilidad de observar si ningún estado o más de un estado están activos a la vez permite detectar el efecto de un SEU y disparar las acciones de reparación, aunque no detecta en cuál FF fue, y no corrige el efecto
- Un bloque estándar parametrizable que indique si sólo un estado (ni menos ni más) está activo es el agregado requerido para detección
- En una FPGA con tablas LUT de 4 entradas, el circuito de primer nivel vigila 4 estados por vez, y luego de requiere un árbol de concentración 4:1



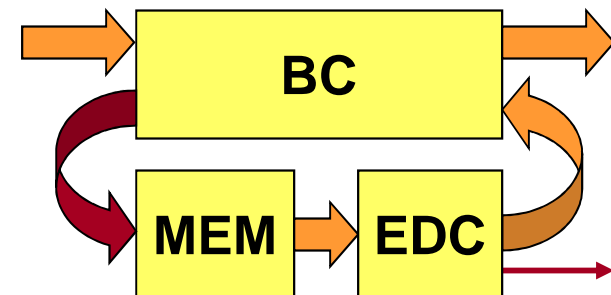
FSM robustas: uso de Hamming de distancia 3 para corregir SEU

- Los códigos de *distancia de Hamming=3* permiten corregir fallas simples (SEC)
- Son códigos lineales binarios, donde para cada entero $r \geq 2$ (bits redundantes) existe un código de longitud $n=2^r-1$ para codificar un mensaje de longitud útil de $k=2^r-r-1$ bits, que al poseer una *distancia de Hamming=3* puede corregir errores simples (pero no diferencia entre un error simple o uno doble, y en caso de un error doble realizará la corrección errónea).
 - Si $r=2$, resulta $n=3$ y $k=1$ (el majority voter!)
 - Si $r=3$, resulta $n=7$ y $k=4$. Un código de hasta 4 bits requiere el agregado de 3 bits redundantes
 - Si $r=4$ entonces $n=15$ y $k=11$, y si $r=5$, entonces $n=31$ y $k=26$.
- Con un bit extra (*códigos Hamming extendidos*) puede diferenciar entre errores de 1 o 2 bits, corregir el error de 1 bit y detectar el de 2 bits (SECDED).
- La proporción entre bits útiles y totales mejora a medida que el tamaño de la carga útil aumenta (es lógico, porque al sólo detectar un error simple en toda esa carga, al aumentar el tamaño de la carga es porque el BER disminuye)

FSM robustas: Hamming de distancia 3

- En el caso de una máquina de sólo 6 estados, su codificación binaria requiere 3 bits, el k más próximo $k=4$, y los de bits de paridad necesarios son $r=3$, lo que requeriría de un registro de 7 bits
- Sin embargo, como con $k=4$ se definen 16 combinaciones, si la cantidad de estados es 8 o menos, de las 16 combinaciones puede elegirse un subconjunto donde algún bit no cambie e ignorarlo, con lo que en vez de un registro de 7 bits alcanza uno de 6 bits
- Un error simple (SEU) en cualquiera de los 6 códigos válidos genera 6 posibles códigos inválidos que permiten corregirlo .. por ejemplo 010001 a qué estado corresponde?

Estado	Cod
S0	000000
S1	000111
S2	011001
S3	011110
S4	101010
S5	101101
SEU simple	36 restantes
estados indefinidos o SEU dobles	22 restantes



Códigos de Hamming: la idea atrás de cómo generar redundancia

- Numerar los bits desde 1 hasta n (Ej: n=15 si quiero un Hamming (15,11)).
- Las posiciones potencia de 2 son bits de paridad. Las demás son bits de datos
- El cálculo del bit de paridad p1 incluye los bits cuyo bit LSB=1 (1,3,5,7,...); p2 a aquellos con segundo bit de posición = 1 (2,3,6,7,10,11); p3 a los con tercer bit de posición = 1 (4 a 7,12 a 15) ; p4 a los con cuarto bit de posición = 1 (8 a 15)
 - $p1 = d1 \text{ XOR } d2 \text{ XOR } d4 \text{ XOR } d5 \text{ XOR } d7 \text{ XOR } d9 \text{ XOR } d11$
 - $p2 = d1 \text{ XOR } d3 \text{ XOR } d4 \text{ XOR } d6 \text{ XOR } d7 \text{ XOR } d10 \text{ XOR } d11$
 - $p3 = d2 \text{ XOR } d3 \text{ XOR } d4 \text{ XOR } d8 \text{ XOR } d9 \text{ XOR } d10 \text{ XOR } d11$
 - $p4 = d5 \text{ XOR } d6 \text{ XOR } d7 \text{ XOR } d8 \text{ XOR } d9 \text{ XOR } d10 \text{ XOR } d11$

posición	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
bits codificados	p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11
cobertura de cada paridad	p1	x		x		x		x		x		x		x	
	p2		x	x			x	x			x	x			x
	p4				x	x	x	x					x	x	x
	p8								x	x	x	x	x	x	x

Códigos de Hamming: cómo detectar/corregir errores

- Calcular el síndrome $s_{8421} = p_{8421}(\text{leído}) \text{ XOR } p_{8421} \text{ calculado}$. Si $s_{8421} = 0000$ no hay errores, sinó, $s_8 s_4 s_2 s_1$ identifica al bit en error
- Por ejemplo, si el síndrome es 1011, es decir hubo error de paridad en p_8 , p_2 y p_1 , el bit erróneo es el ubicado en la posición 11 (d_7), cuyo valor debe ser complementado
- Es decir: $d_7 (\text{corregido}) = d_7 (\text{leído}) \text{ XOR } (s_8 \text{ AND } \text{NOT}(s_4) \text{ AND } s_2 \text{ AND } s_1)$
- Si el error es de uno de los bits de paridad el síndrome es también correcto

posición	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
bits codificados	p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11
cobertura de cada paridad	p1	x		x		x		x		x		x		x	
	p2		x	x			x	x			x			x	x
	p4				x	x	x	x				x	x	x	x
	p8								x	x	x	x	x	x	x

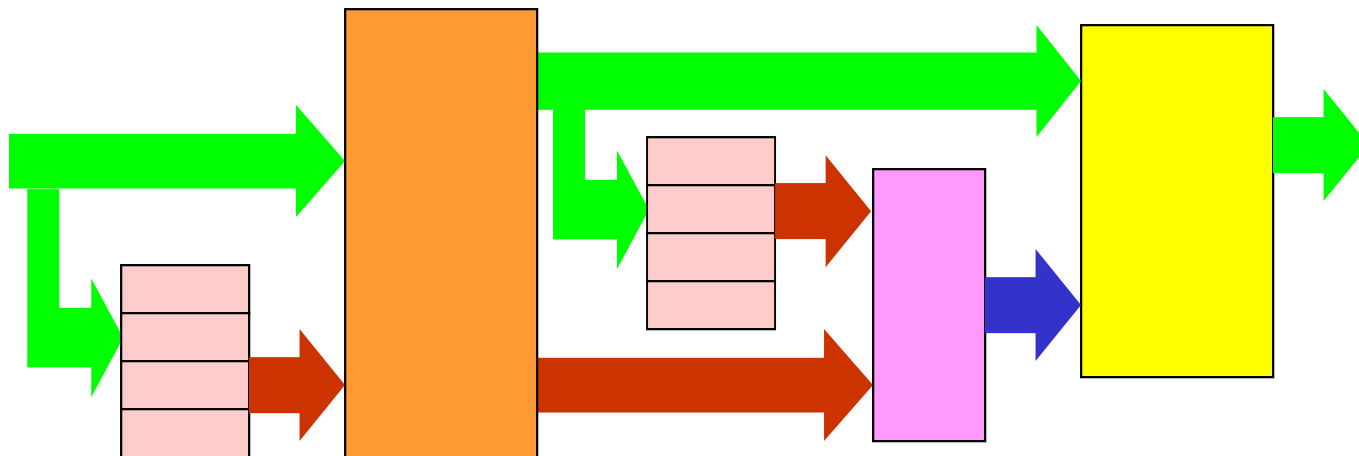
Códigos de Hamming: la idea atrás de cómo detectar/corregir errores

- Si cambian dos bits, el síndrome indicará un error incorrecto
 - Si cambian D1 y D2 resulta $S_{8421}=0110$ y se presume que cambió D3
 - Si cambian D10 y D11 resulta $S_{8421}=0001$ y se presume que cambió p1
- Al detectarse un error ($s_{8421} \neq 0$) un bit extra pp calculado como la paridad de p1..p4 más d1..d11 permitiría diferenciar errores simple (donde pp cambia) o dobles (donde pp no cambia)
- Pero calcular pp, *al depender de los valores de los pi agrega latencia*

posición		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
bits codificados		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11
cobertura de cada paridad	p1	x		x		x		x		x		x		x		x
	p2		x	x			x	x			x	x			x	x
	p4				x	x	x	x					x	x	x	x
	p8								x	x	x	x	x	x	x	x

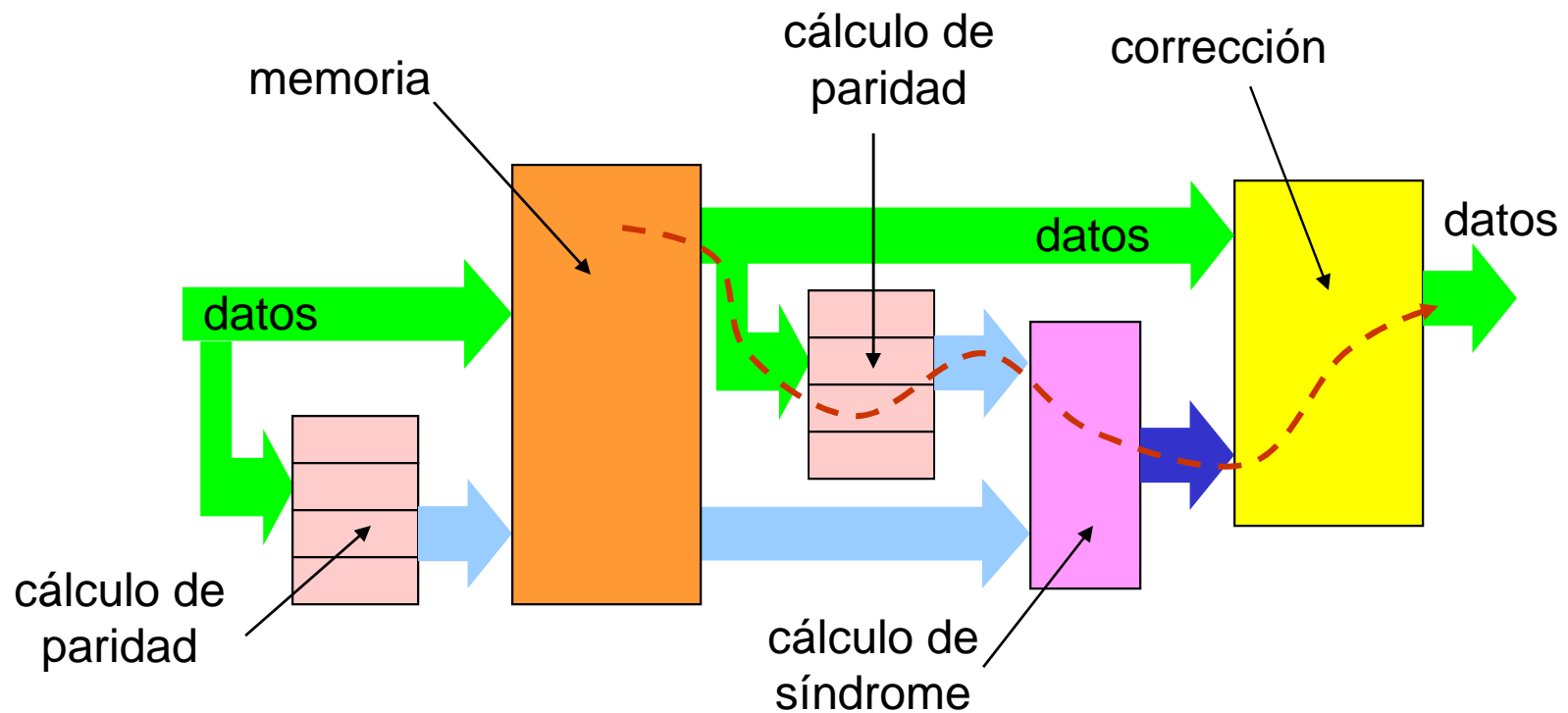
Códigos de Hamming: la idea atrás de cómo detectar/corregir errores

- Identificar la función de cada bloque y el camino crítico en tiempo durante la escritura y durante la lectura



Códigos de Hamming: la idea atrás de cómo detectar/corregir errores

- Identificar la función de cada bloque y el camino crítico en tiempo durante la escritura y durante la lectura

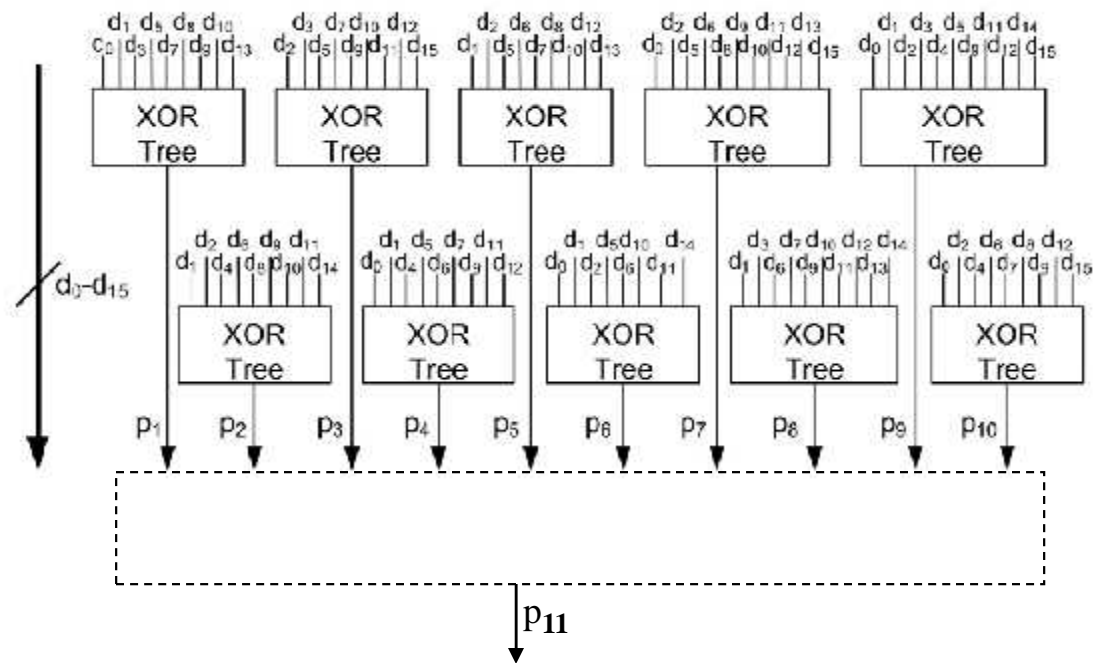


Y para más de un error? Métodos para MBU (Multiple Bit Upsets).

- Para errores simples vale considerar los códigos Hsiao, que son óptimos (ver *Hsiao70.pdf: M. Y. Hsiao, "A Class of Optimal Minimum Odd-weight-column SECDED Codes", IBM Journal of R & D Vol. 14, July 1970, pp. 395-401*)
- Para corregir errores dobles (DEC, requiere distancia de Hamming 5), los códigos requeridos son (26,16), (44,32), (78,64), (144,128). Al igual que en SEC versus SECDED, para DECTED se requiere el uso de un bit extra de paridad de los bits de paridad, cuyo mayor defecto es el agregado de latencia (ver *esscirc08.pdf*)
- Para errores triples (distancia de Hamming 7), el código Golay (24,12,8) codifica 12 bits de datos útiles mediante el agregado de otros 12 bits, con lo que el dato codificado es de 24 bits. Esta codificación provee una distancia de Hamming de 8 entre códigos, con lo que detecta TODOS los errores de hasta 7 bits, y permite corregir TODOS los errores de hasta 3 bits. Este código ha sido usado por las misiones de espacio profundo Voyager 1 & 2 al transmitir imágenes color de Júpiter y Saturno, a fines de los 70s

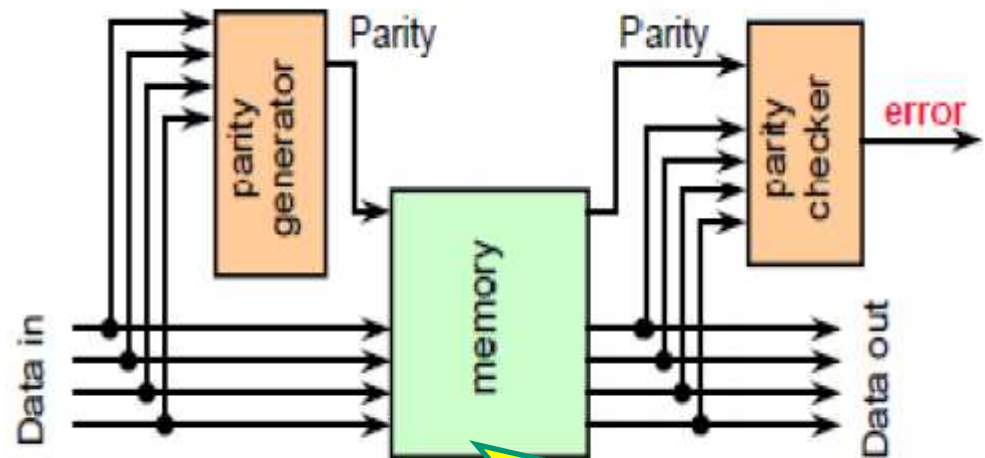
Y para más de un error? DEC vs DECTED

- Para el caso DEC, se muestran los circuitos de fanin=9 necesarios para calcular los bits de paridad
- Y se marca con líneas de puntos el árbol XOR a agregar en caso de desear Triple Error Detection, de fanin=10
- Por eso el pasaje de DEC a DECTED casi duplica la demora de los circuitos de codificación
- En la decodificación el tratamiento de p_{11} va en paralelo con la generación del síndrome y corrección



Otras técnicas de mitigación de SET/SEU: Information redundancy

- Es el método más empleado en memorias, donde el uso de feedback TMR resulta impráctico, y es llamado EDAC, por *error detection and correction*.
- La forma más común es el uso de códigos lineales de paridad, como Hamming (a nivel de palabra), Reed-Solomon (a nivel de bloques) y BCH codes (usados en LEON3-FT).
- Especialmente aptos para compensar MBUs en memorias



En las FPGAs, dada la posibilidad de configurar los bloques con distinto ancho de palabra (256x8, 512x4, 1Kx2, 2Kx1), la mejor manera para aprovechar EDAC es organizar la memoria de a bits, de modo que cada bit de una palabra esté físicamente lejano del otro, y así minimizar la chance de MBUs

Triple Modular Redundancy (TMR)

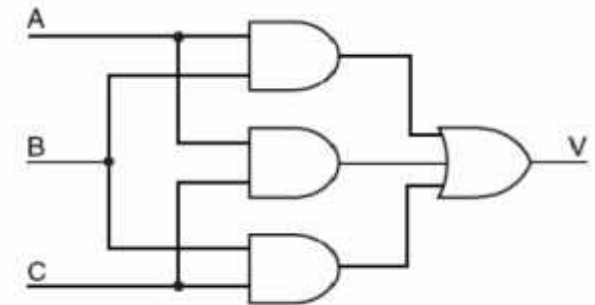
- El concepto básico de agregar módulos redundantes (cantidad impar y mayor o igual a tres, en general tres) es suponer la existencia de *Single Event Effects*, es decir **UNA** alteración del circuito por vez.
- En ese caso, si de tres señales que deberían ser idénticas UNA no lo es, la decisión del valor correcto puede tomarse en base al valor de las DOS señales restantes que son iguales entre sí (*MAXIMUM LIKELIHOOD*).
- Y, según la funcionalidad del bloque corregido (entrada, salida, registro interno) pueden realizarse o no acciones correctivas en el bloque distinto.
- La redundancia puede ser a nivel de microelectrónica (venir embebida dentro de los registros, por ejemplo) o ser creada por diseño de alto nivel (en una FPGA, por ejemplo, usando VHDL)
- Para evitar que las alteraciones se acumulen, en las FPGA basadas en SRAM conviene usar técnicas de “mantenimiento periódico” (*scrubbing*)

Triple Modular Redundancy (TMR): mecanismos de votación

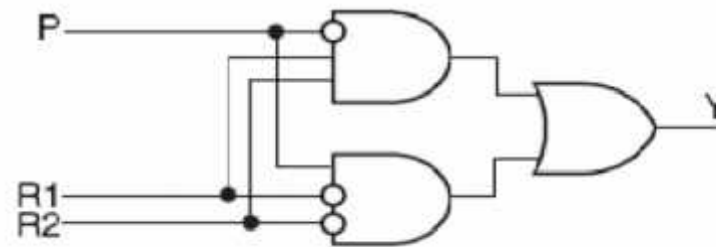
- Los dos mecanismos de votación empleados (*voters*) se llaman **MAJORITY VOTER** y **MINORITY VOTER**

- En el **MAJORITY VOTER** la salida copia el valor que esté repetido en al menos dos de las entradas

A	B	C	V
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



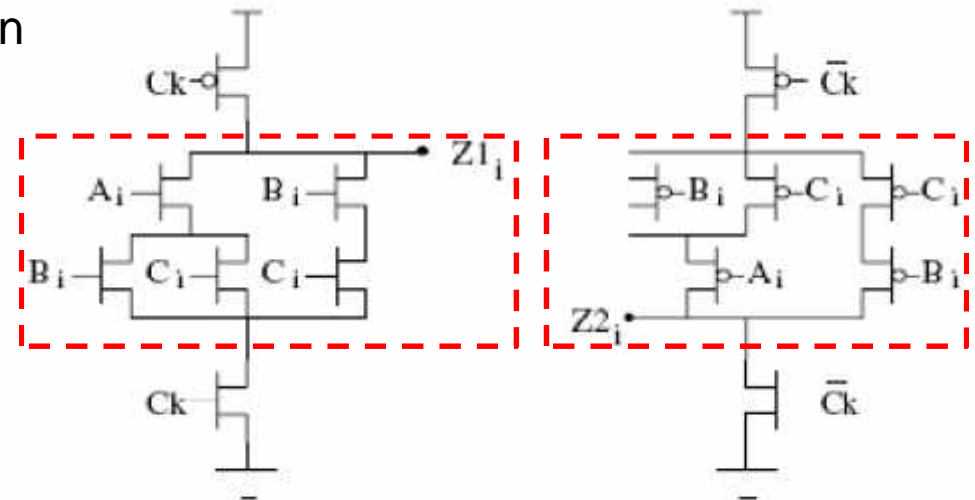
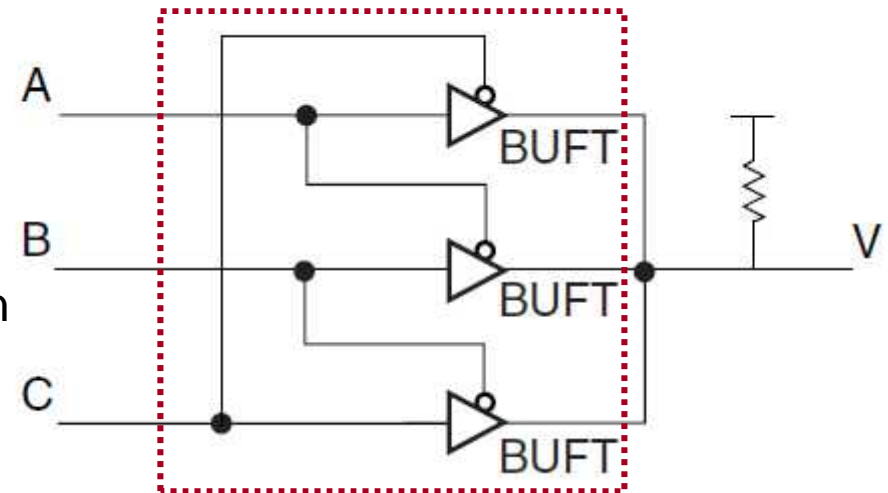
- El **MINORITY VOTER** compara una referencia P con otras dos entradas R1 y R2, y pone 1 si P está en minoría respecto al valor mayoritario del conjunto (P,R1,R2)



P	R1	R2	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0

Triple Modular Redundancy (TMR): Majority voter

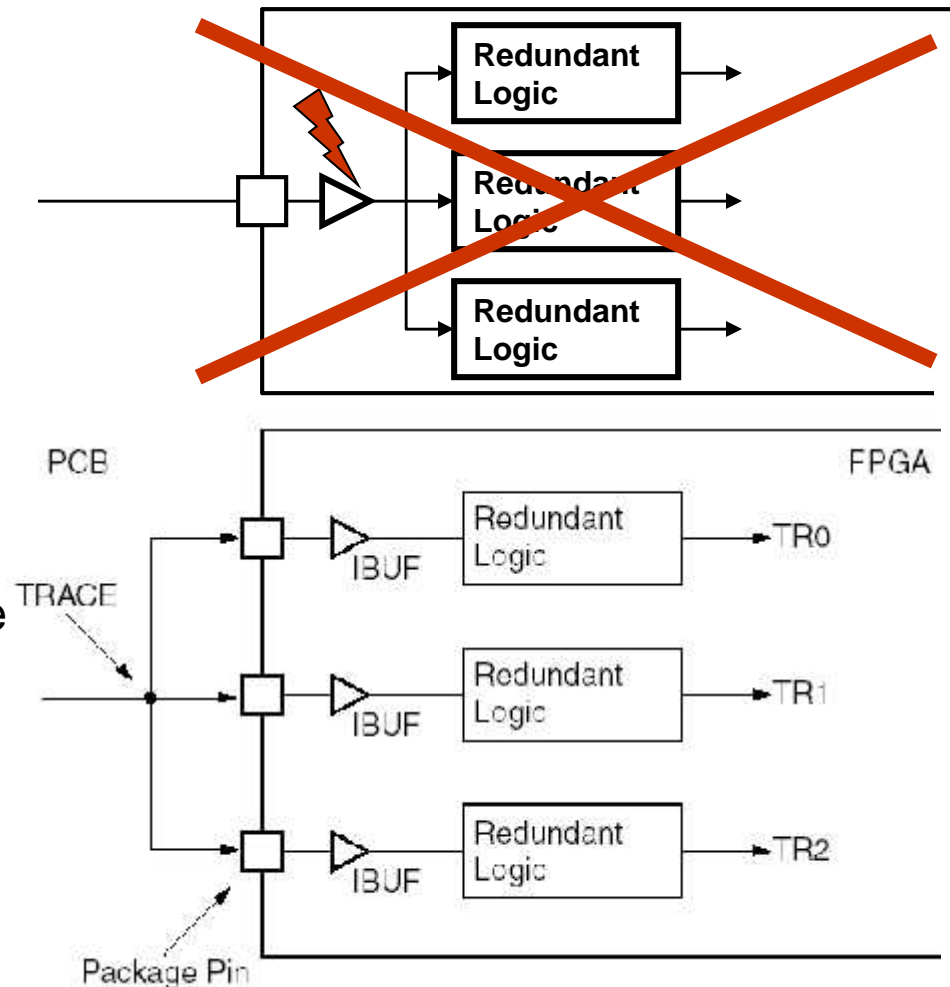
- Un **MAJORITY VOTER** puede ser implementado también mediante el uso de salidas Open Collector, y un pullup, pues el Open Collector realiza la función AND y el resistor la función OR
- En microelectrónica, la posible solución –negadora- emplea 5 transistores por fase de reloj:



NOT((B AND C)
 OR (A AND (B OR C)))

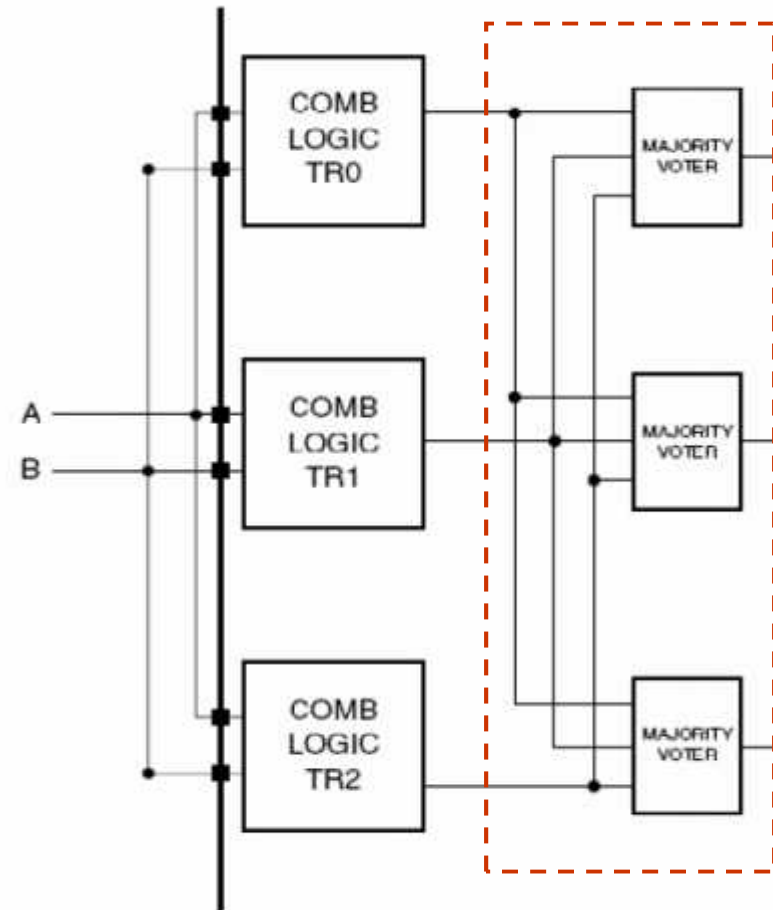
Triple Modular Redundancy: Entradas simples

- Requiere 3 patas del dispositivo por cada señal de entrada
- Es el caso de señales globales (clk, rst), donde el uso de una única pata puede comprometer a todo el componente a todo el circuito si el SEE sucede en el buffer de entrada
- En el caso de buses puede emplearse EDC, para bajar el uso de patas
- Además de mayor uso de recursos de pines, también triplica la capacidad de entrada



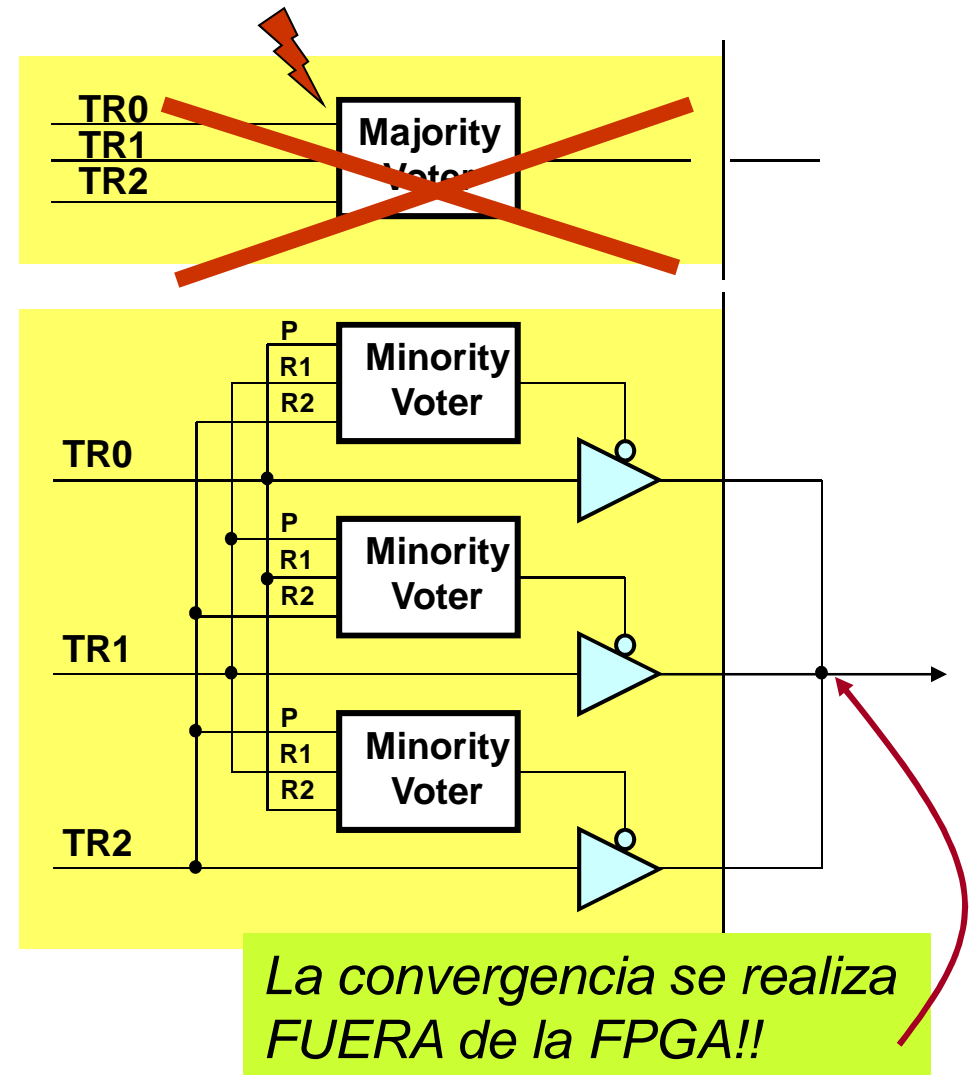
Triple Modular Redundancy: Entradas a circuitos combinatorios

- Si más de una entrada se destina a un circuito combinatorio, las tres réplicas de las señales de entrada A y B son dirigidas a tres circuitos combinatorios similares
- Y la generación de las señales replicadas de entrada se realiza mediante MAJORITY VOTERS (MJV)
- Un SEU en un bloque combinatorio o una entrada es corregido por los MJVs
- Un SEU en un MJV es corregido en las etapas siguientes



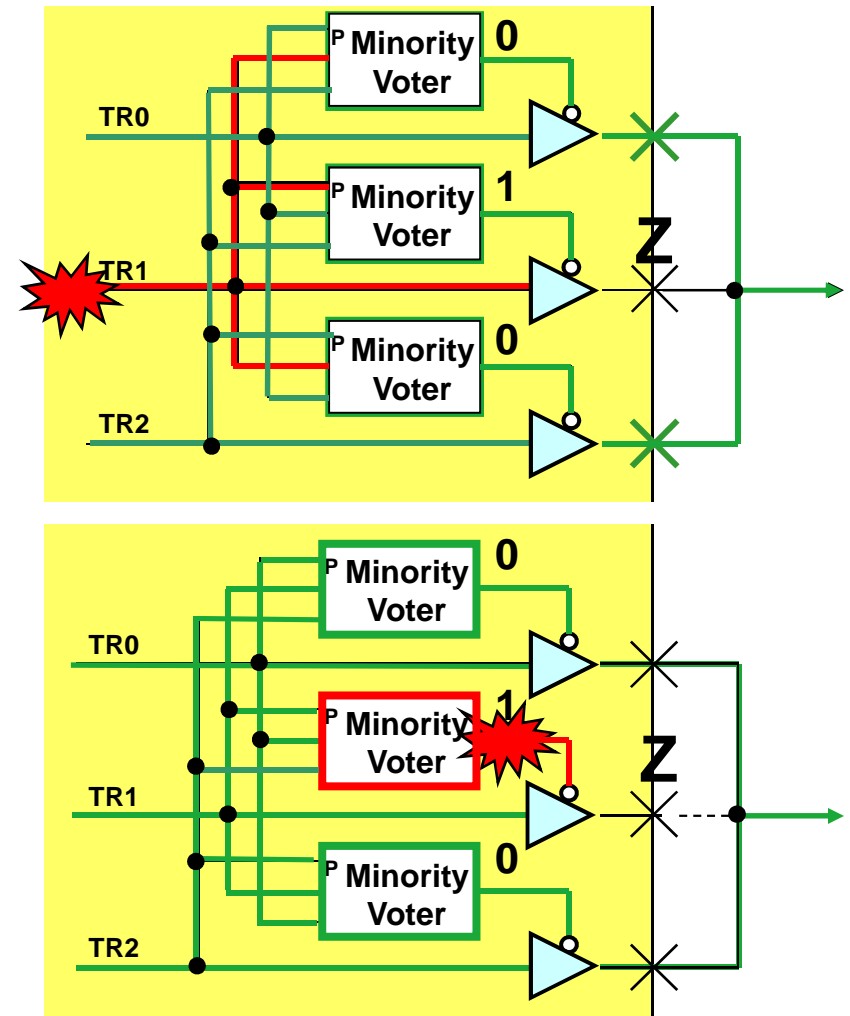
Triple Modular Redundancy: Salidas triplicadas

- Usar un MAJORITY VOTER para generar una salida es susceptible a que un SEU en el Voter produzca una salida incorrecta
- Las salidas pueden ser triplicadas, requiriendo tres patas por cada señal de salida
- Cada MINORITY VOTER detecta si su circuito P está en minoría con el valor mayoritario, y en ese caso pone esa salida en Tri-State
- Los votadores son triplicados



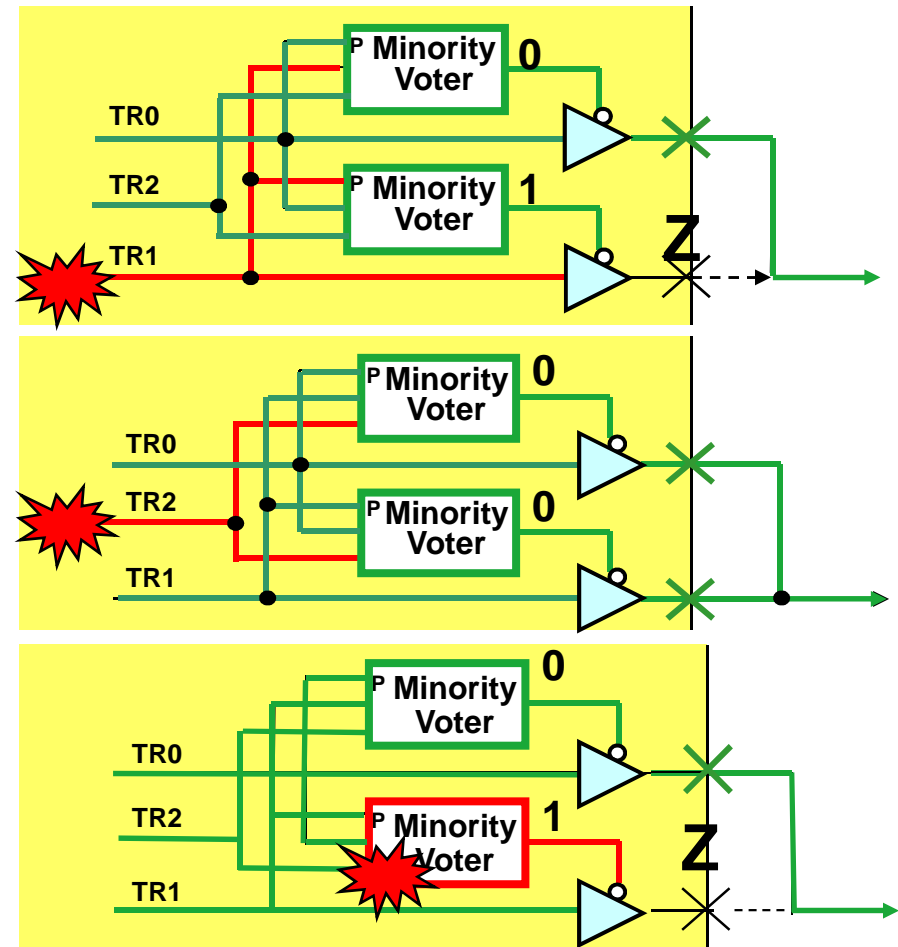
Triple Modular Redundancy: Salidas triplicadas

- La ocurrencia de un SEU puede darse en los circuitos de entrada, en los votadores, o en los buffers de I/O
- Si ocurre en los circuitos de entrada (*datapath*) el MINORITY VOTER bloquea ese camino poniendo al buffer en tri-state
- Si ocurre en un MINORITY VOTER éste pone en tri-state su propio circuito, pero los demás circuitos mantienen la salida
- Se nota que siempre al menos dos salidas están activas a la vez, lo que es redundante



Triple Modular Redundancy: Salidas duplicadas (JPL)

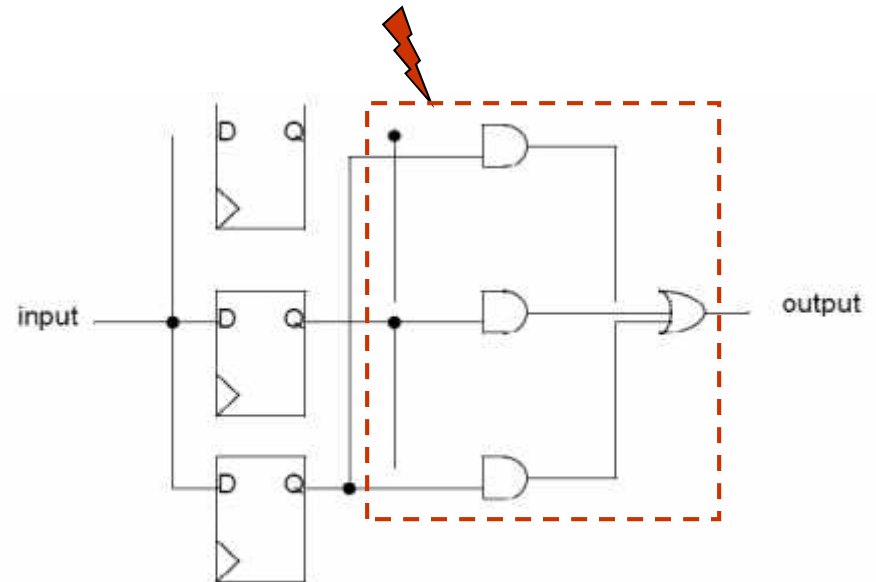
- El mismo comportamiento puede lograrse usando solo dos salidas y dos MINORITY VOTERS (MNVs)
- Si la falla se da en una línea que tiene un MV (TR0 o TR1) su salida es puesta en Tri-state
- Si se da en la línea que no tiene salida ni MV asignado (TR2) no pasa nada
- Si se da en un MV lo peor que puede pasar es que su salida vaya a tri-state, pero igual el valor es sostenido por la otra salida
- Este esquema disminuye el uso de patas y de silicio,



by Gary Swift at JPL

Triple Modular Redundancy: FlipFlops

- La redundancia a nivel de cada fliflop puede lograrse mediante tres flipflops y un MAJORITY VOTER
- El efecto de un SEU es compensado por el VOTER
- El efecto de un SET en el VOTER importa si afecta a la lógica en la ventana de oportunidad, y según cuál sea la duración del SET

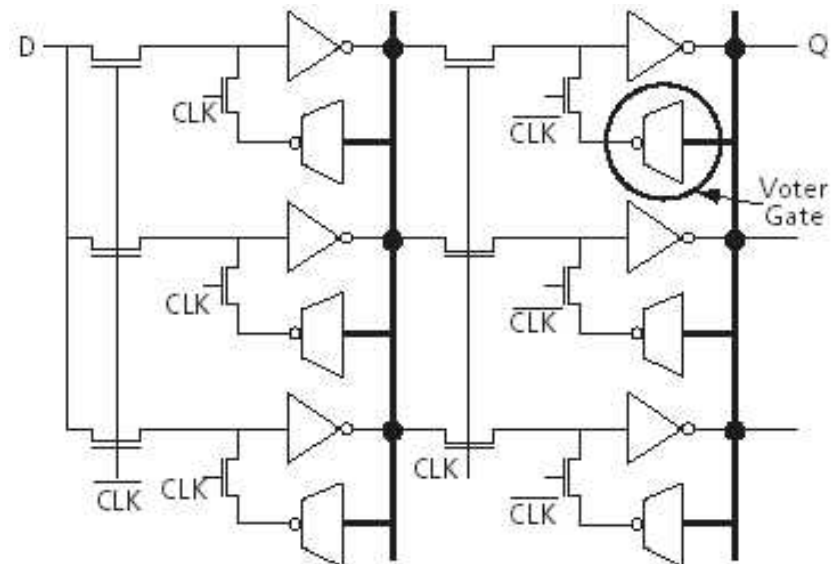
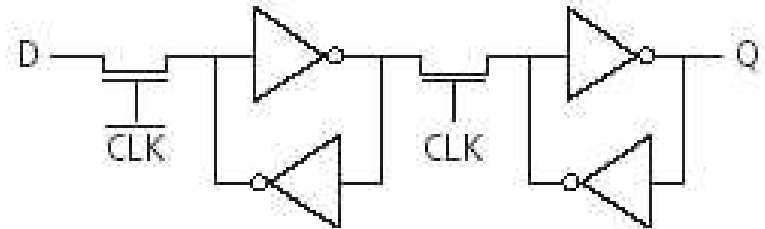


Redundancia embebida en los flipflops de Actel

Es un cambio del esquema Master-Slave Latch bifase de las FPGAs RTSX-A

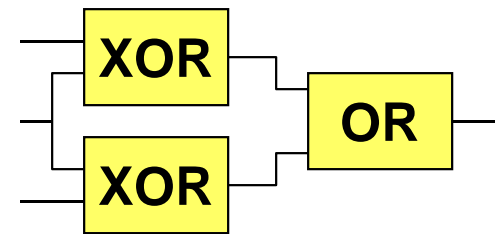
En las celdas “R” de las RTSX-S se triplican los recursos empleados

- El inversor de realimentación es reemplazado por un circuito de votación
- La distribución espacial de las compuertas busca evitar el MBU
- No requiere ser considerado en el VHDL, porque es nativo



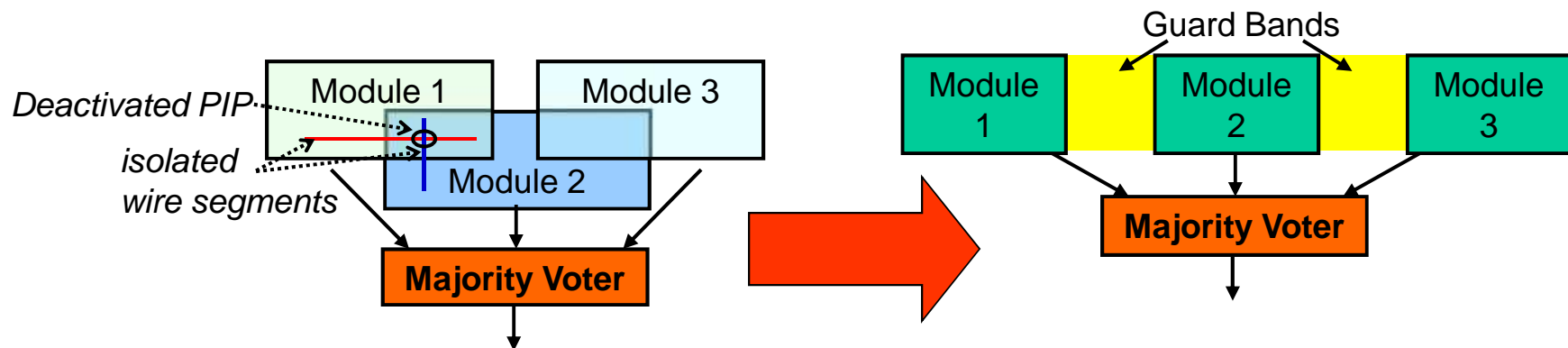
Triple Modular Redundancy: Voting Fault

- En un MAJORITY VOTER las tres entradas deberían ser iguales (todas '1' o todas '0')
- Lo mismo en un MINORITY VOTER
- Si se genera un circuito extra que detecte cuando una entrada es distinta a las otras dos, se dispone de un indicador de falla
- Este circuito puede permitir detectar la ocurrencia de SEUs
- Debe ser temporizado para ignorar los SETs y hazards por skew
- Y eventualmente disparar un proceso de SCRUBBING
- O hacer actuar un WatchDog



Implementación con HDL de Modular Redundancy (TMR) en FPGAs

- La distribución de macroceldas en la FPGA define un nivel de entrecruzamiento de circuitos lógicos elevados
- La posibilidad que una falla de SEU influya sobre un punto de conexionado que afecte varios módulos es elevada
- Y por tanto un falla simple puede tener el efecto de un MBU
- Y destruir las ventajas del TMR
- Ésto obliga a que en el diseño se deba considerar el place&route
- Y aislar las posibles fallas con regiones de guarda



Revisar

***“Functional Triple Modular Redundancy (FTMR). VHDL Design Methodology for Redundancy in Combinatorial and Sequential Logic”.
Design and Assessment Report
by Sandi Habinc, www.gaisler.com
ESA contract, No. 15102/01/NL/FM(SC) CCN-3.***

Define un package y componentes que describen en VHDL los elementos necesarios para una implementación TMR

El costo asociado al uso de Triple Module Redundancy

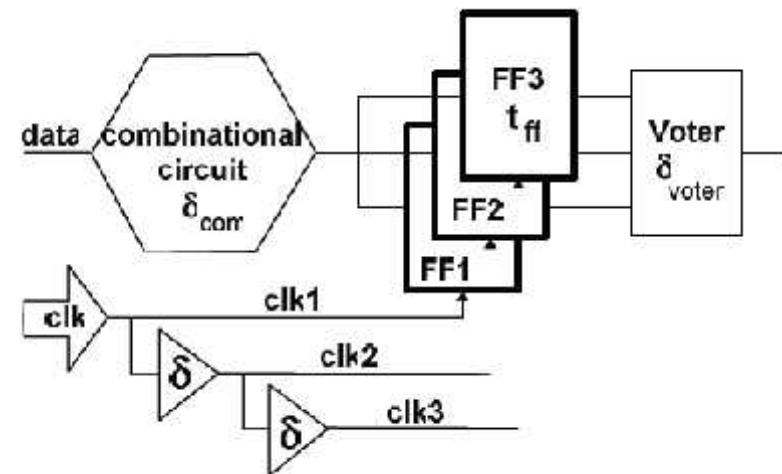
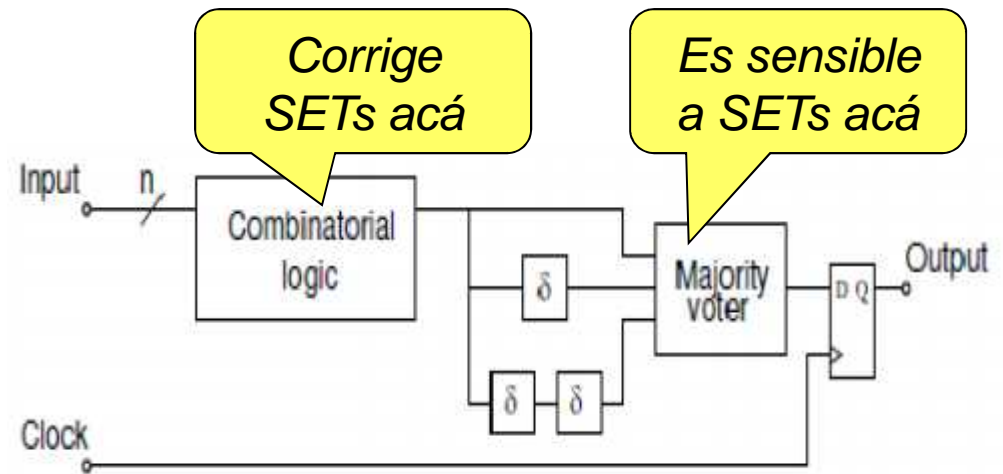
- Uso de muchos más recursos lógicos
 - La cantidad de lógica usada crece más de 3 veces
 - Es necesario tener en cuenta el proceso de fitting, para aumentar la resistencia a MBUs reales (o inducidos por SEU en áreas comunes de cableado)
- Consume más del triple de potencia
- La velocidad de operación del sistema baja
 - Los caminos de realimentación de Feedback TMR lo hacen inherentemente lento
 - La necesidad de aislar bloques mediante floorplanning y por el mayor uso de recursos afecta también la velocidad
- Requiere un conocimiento detallado de la arquitectura de la FPGA que a veces no es público

Variaciones en el uso de TMR: uso selectivo

- Identificar regiones críticas que pueden hacer que el efecto del SEU se mantenga, o las lógicas basadas en estados
- Aplicar TMR sólo en estas secciones
 - Es un método NO SISTEMÁTICO de diseño, sino que depende de la aplicación
- Las publicaciones sobre Selective TMR sugieren:
 - Mitigación de errores cercana al 90% respecto al uso de TMR exhaustivo
 - Mejor aprovechamiento del dispositivo
 - Menos consumo de potencia
- Requiere herramientas que analicen la gravedad de los efectos de un SEU y evitar el uso selectivo arbitrario

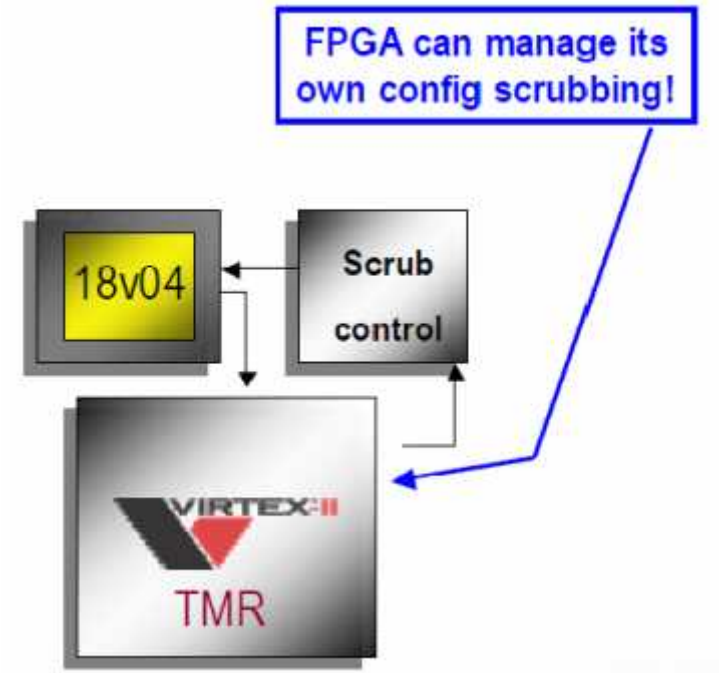
Otras técnicas de mitigación de SET/SEU: Time redundancy

- El empleo de retardos en caminos múltiples pueden eliminar aquellos SET cuya duración sea menor al tiempo de retardo
- Es una técnica que al ser usada en los relojes de varios flipflops permite también corregir las consecuencias de SETs en las señales de reloj
- El costo es la pérdida de velocidad del sistema por los retardos combinatorios y el uso de varias redes de distribución de reloj



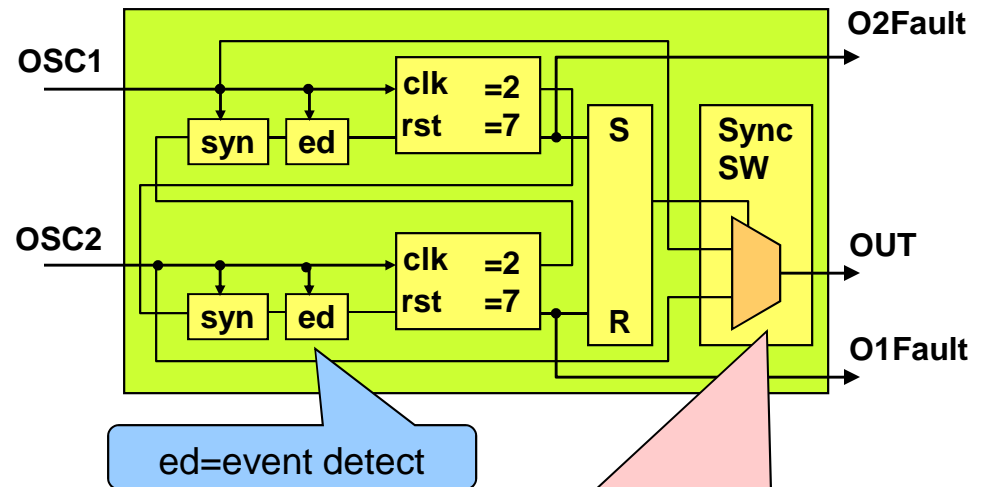
Scrubbing: mitigación de los SEE por reconfiguración del componente

- One advantage of reprogrammable FPGAs (especially SRAM-based) is to allow the **configuration memory to be refreshed**.
- Normally some external hardware is required to realize this technique; also the device usually has to be reset during reconfiguration, losing all data in the on-chip user memory.
- Some devices support **readback and partial reconfiguration**. This allows to check the configuration memory for errors, and correct selected sections of the configuration RAM **without resetting the device**, while keeping all the current data in the on-chip memory, minimising the effect of upsets.
- These approaches **require a trusted data source** (e.g. Radhard PROMs) for the backup bitstream.



Redundancia en el reloj

- En un sistema que recibe una señal de reloj externa la falla de ese reloj es una falla catastrófica
- El uso de dos generadores de reloj en vez de uno puede mejorar la disponibilidad del sistema
- Y ser informada al sistema de modo que puedan tomarse las acciones correctivas



A diferencia de un conmutador de relojes no coherentes estándar, que asegura que no se violen los tiempos T_{ON} ni T_{OFF} de cada reloj, en este caso debe considerarse que un reloj fallido equivale a T_{ON} o T_{OFF} infinito, según en que estado haya quedado la salida de ese reloj