FTF

Tema 3: Tolerancia a fallos en el hw

1. Modos del fallo
2. Modelos de fallos y de errores
3. Técnicas de detección de errores
   - 3.1 Duplicación y Comparación
   - 3.2 Redundancia en la información
   - 3.3 Circuitos autocomprobables
   - 3.4 Detección concurrente
4. Técnicas de recuperación hacia delante
   - 4.1 Redundancia pasiva
   - 4.2 Redundancia activa
   - 4.3 Redundancia híbrida
   - 4.4 Recuperación hacia delante basada en pdr's
5. Reconfiguración en sistemas multiprocesadores

FTF

1. Modos del fallo

- El modo del fallo es su forma de manifestación
- Circuitos lógicos digitales
  - Señales a nivel fijo (pegado-a o *stuck-at*)
  - Puentes entre conexiones
  - Circuitos abiertos
- Memorias ROM
  - Direccionamiento incorrecto (fallo en el decodificador de direcciones)
  - Selección incorrecta del circuito
  - Contenido incorrecto de una celdilla (debido a un error en un transistor)
- Memorias RAM
  - Escritura múltiple debida a un acoplamiento capacitivo entre celdillas
  - Aumento del tiempo de acceso por excesiva carga capacitiva en la salida
  - Inercia en la escritura de largas series de '1' y '0' por saturación del amplificador sensor
  - Cambio transitorio del valor de una celdilla por acoplamiento de los caminos
  - Cambio del contenido de alguna celda por radiaciones α, o a ruido de recombinación
  - Ausencia de refresco por fugas en la retención de cargas entre dos refrescos sucesivos

FTF

Modos del fallo (II)

- En circuitos lógicos programables
  - Fallos a nivel fijo (‘0’ ó ‘1’)
  - Fallos de contacto o fallos en puntos de cruce
  - Fallos por puentes
- Microprocesadores
  - Señales a nivel fijo de forma transitoria
  - Retardos de propagación no especificados
  - Transformación de circuitos combinacionales en secuenciales
  - Cambios en las señales debidos a sensibilidad a patrones e instrucciones
  - Dando como errores más comunes en el sistema:
    - Acceso a una dirección de memoria no utilizada
    - Dirección de lectura inválida por acceder a un segmento inexistente
    - Código de operación inválido
    - Dirección de escritura inválida, por corresponder a un segmento de sólo lectura

FTF

2. Modelos de fallos y de errores

- Los fallos hardware son de naturaleza muy diferente:
  - Fabricación: Conexiones incorrectas, problemas con el silicio o en el empaquetado
  - Operación: Cortocircuitos, circuitos abiertos, cambio en la tensión de funcionamiento
- Aunque los fallos físicos producen las averías en los integrados, la mayoría de las veces no interesa este detalle a nivel de micro-circuito. Únicamente se valora si existe fallo o no, es decir sus consecuencias.
- Para ello se realiza una hipótesis sobre los efectos de los fallos físicos en un nivel superior:
  - Nivel de puertas y circuitos lógicos
  - Nivel de bloque funcional
  - Nivel de circuito integrado

  } A estas descripciones se les llama *modelos de fallo*
- Su objetivo es que la consideración de unos pocos fallos en el modelo de fallos sirva para describir la mayoría de los fallos físicos de interés. Los mecanismos de tolerancia a fallos únicamente considerarán los fallos del modelo.
- Un modelo de fallos representa el efecto del fallo mediante el cambio que este produce en las señales del sistema

FTF

- Modelos de fallos a nivel de puertas: De pegado-a (*Stuck-at*)
  - » Son los modelos más simples y se utilizan en el hardware tolerante a fallos
  - » Una línea o puerta tiene un fallo de pegado-a-0 cuando independientemente del valor de las entradas, se encuentra con el valor lógico 0
  - » Se pueden producir fallos múltiples o unidireccionales
  - » Otros modelos derivados son los
    - » *stuck-open* cuando un circuito combinacional se comporta como un secuencial al romperse la salida de un transistor CMOS.
    - » *stuck-on* se producen cuando el transistor conduce de forma permanente.
    - » *Por puenteado*: Modela los cortocircuitos entre líneas adyacentes
    - » *De retardos*: Modela los retrasos producidos en las puertas o caminos en el circuito.
- Modelos de fallos a nivel de función:
  - Se interesa simplemente de si los integrados o módulos funcionan correctamente
  - Se utilizan modelos que varían la función implementada en los bloques:
    - En un multiplexor: Se activa una línea incorrecta, se activan varias o ninguna
- Modelos de fallos de memoria:
  - Tipos: Una o más celdas pegadas, acoplamiento de celdas, fallos sensibles a patrones

FTF

3.- Técnicas de detección de errores

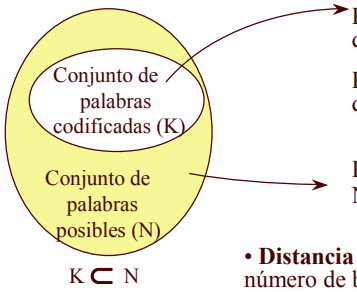
- Son de vital Importancia en los sistemas con redundancia dinámica.
- Se pueden emplear en diferentes niveles o en momentos diferentes:

  - Detección a nivel de señal:
    - Códigos detectores de errores.
    - Técnica **cara**, de **baja latencia**.
  - Detección a nivel de función:
    - Sistemas duplicados.
    - Tests de aceptación.
    - Circuitos autocomprobables.
    - Temporizadores y procesadores de guardia (Watchdogs).
    - Técnica **barata**, de **alta latencia**.
  - Detecciones periódicas:
    - Detección inicial.
      - En la puesta en marcha de un sistema.
      - Test exhaustivo, general.
    - Detección concurrente.
      - Tareas de test.
      - Test rápido, general.
    - Detección fuera de línea.
      - Mantenimiento preventivo.
      - Cuando ocurre una avería.
      - Test exhaustivo, general o particular.

FTF

3.1 – Detección a nivel de señal: Comprobadores estructurales

- El uso de códigos detectores de errores permite la verificación de la estructura interna de los datos. Estos códigos añaden redundancia en la información.



Representaciones válidas de la información (p ∈ **K**)

Representaciones no válidas de la información (p' ∈ **N–K**)

Representaciones posibles: $N = 2^n$; n: Núm. de bits de la palabra codificada

K ⊂ N

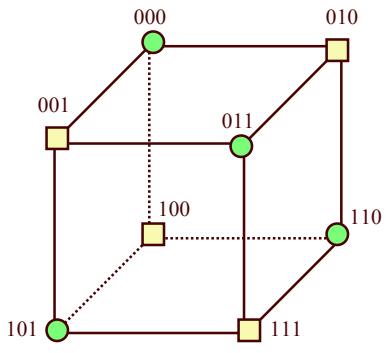
• **Distancia de Hamming**: (Entre dos palabras): Es el número de bits que difieren de una palabra a la otra.

Ejemplos: 0 0 0 0 / 0 1 0 0 → Distancia 1 　　0 0 1 0 / 0 0 0 1 → Distancia 2

FTF

- **Distancia mínima** de un código **d** (distancia del código):
  - Es la mínima distancia entre todas las posibles parejas de palabras del código.

Ejemplo: para n = 3 bits:



• Aristas del cubo: d = 1

□ → Paridad impar

○ → Paridad par

• Códigos de paridad: d = 2, y detectan errores de 1 bit. ⇒

Un código que detecte errores en "n" bits, tendrá una distancia de:

$$d \geq n+1$$

  - Un código de paridad de 3 bits se forma añadiendo 1 bit de paridad a la información original de 2 bits: N posee 8 palabras (n = 3) , y K la mitad que N (4)

Otras propiedades de los códigos

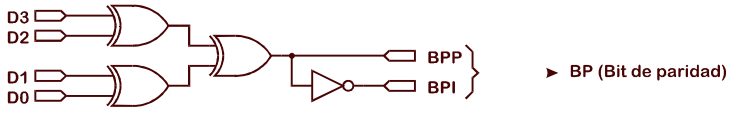
FTF

- Redundancia (R)
  - Es el número de **bits añadidos** / número de **bits de la palabra** sin codificar.
  - En un código de paridad simple, R = 1 / tamaño de la palabra
- Capacidad de detección
  - Es la média absoluta del número de errores detectados
  - En códigos de paridad, todos los errores de 1 bit y los de un número impar de bits.
- Cobertura (C)
  - Es el cociente del nº medio de errores detectados / nº total de errores de la hipótesis de diseño
  - La cobertura de detección de los errores de 1 bit (hipótesis) en códigos de paridad es del 100 % , mientras que la cobertura para errores de 2 bits es del 0 %.
  - Interesa que R sea baja y C alta.
- Códigos separables
  - Son los códigos en donde los bits de la palabra sin codificar y los del código están separados
  - Son más fáciles de implementar pero tienen menor capacidad de detección

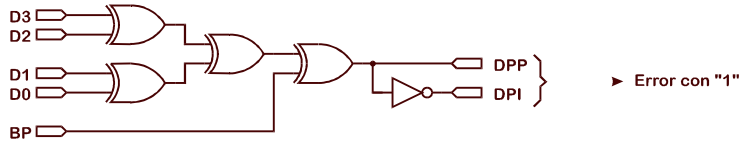
Códigos de paridad

FTF

- **Paridad bit por palabra**: Se añade 1 bit (bit de paridad) a los de datos para formar la palabra codificada. Puede ser par o impar
  - d = 2 → Detecta todos los errores de 1 bit.
  - Código separable.
  - Código muy fácil de implementar y de detectar con árboles de puertas XOR



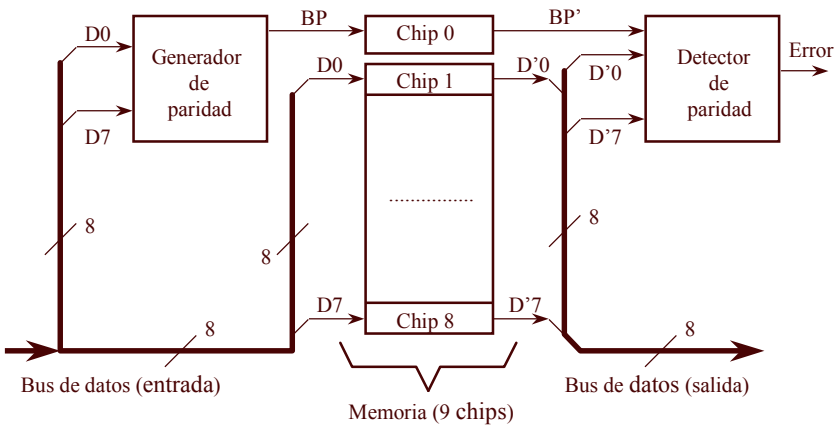
Generador del bit de paridad (4 bits)



Detector de paridad (4 bits + bit de paridad)

FTF

- Aplicación de la paridad en memorias (para 8 bits de datos):



FTF

- Comparación entre las capacidades de detección de la paridad **par** y la paridad **impar**

Tipo de error	PAR	IMPAR
1 bit	**100 %**	**100 %**
Nº impar de bits	**100 %**	**100 %**
Nº par de bits	**0 %**	**0 %**
Errores múltiples unidireccionales	**50 %**	**50 %**
Pegado-a-1	100 %	0 %
Pegado-a-0	0 %	100 %

– En los fallos de pegado-a y en los errores múltiples unidireccionales adyacentes se supone que el ancho de palabra es un número par
– Los errores múltiples unidireccionales adyacentes sólo son detectados en el caso de que la paridad de la información almacenada en el chip estropeado sea IMPAR

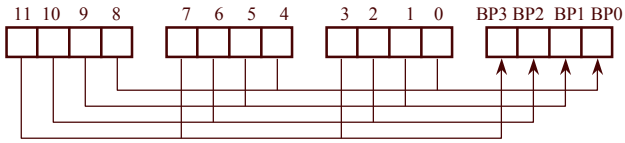
FTF

- **Paridad bit por byte:** Se divide la palabra original en dos trozos (bytes), cada uno con 1 bit de paridad diferente:

  - Capacidad de detección:
    - Errores simples y de un número impar de bits en cada byte.
    - Errores dobles si hay un error en cada byte.
    - Todos **errores de todo a "0**" (paridad impar) y **de todo a "1** " (paridad par).
    - Errores m.u.a: Difíciles de detectar, como en el caso anterior.
  - Redundancia: 2/BD.

FTF

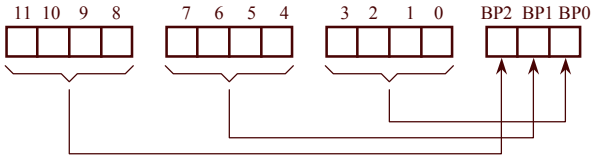
- **Paridad bit por múltiples chips**: 1 bit de paridad por cada bit de cada chip → Tantos bits de paridad (BP) como bits / chip:



  - Capacidad de detección:
    - Errores simples y de un número impar de bits en cada árbol de paridad.
    - Errores dobles si los errores afectan a árboles de paridad distintos.
    - **Todos los errores m.u.a.** (todos los errores de 1 chip entero).
    - Errores de todo a "0" o todo a "1" detectados
      - Cuando uso paridad diferente en los BPi y tengo un número par de chips
      - Cuando uso paridad impar y tengo un número impar de chips.
    - No detecta el chip averiado.
  - Redundancia: 1/CH (Siendo CH = Nº de chips)

FTF

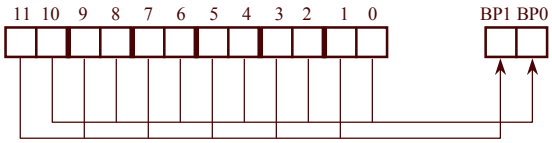
- **Paridad bit por chip**: 1 bit de paridad por cada chip → Tantos bits de paridad (BP) como chips:



  - Capacidad de detección:
    - Errores simples y de un número impar de bits en cada chip.
    - Errores dobles si hay un error en cada chip.
    - No todos los errores m.u.a. (todos los errores de 1 chip entero).
    - Todos los errores de todo a "0" o todo a "1" si se usa paridad diferente en los BPi
    - **Detecta el chip averiado**.
  - Redundancia: 1/BC (Siendo BC = Nº bits /chip)

FTF

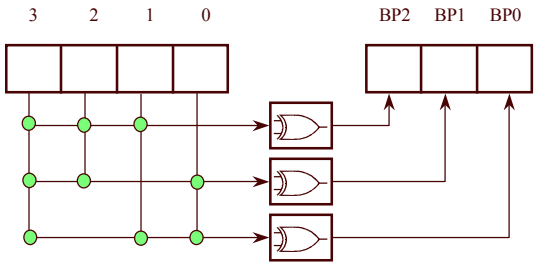
- **Paridad entrelazada**: Igual como paridad bit por múltiples chips, pero aplicada a buses: 1 bit de paridad por cada bit de cada división → Tantos bits de paridad (BP) como bits / división:



  - Capacidad de detección:
    - Errores simples y de un número impar de bits en cada árbol de paridad.
    - Errores dobles si hay un error en cada árbol de paridad.
    - **Todos los puentes entre líneas adyacentes**
    - Errores de todo a "0" o todo a "1" detectados
      - Cuando uso paridad diferente en los BPi y tengo un nº par de divisiones
      - Cuando uso paridad impar y tengo un número impar de divisiones
    - Detecta errores de los **drivers de bus**
  - Redundancia: 1/D (Siendo D = Nº de divisiones)

FTF

- **Paridad superpuesta**: Cada bit de datos aparece en más de un árbol de paridad.
  - Permiten además de detectar el error, localizar su posición
  - Por tanto pemiten también corregirlo utilizando un complementador



FTF

Códigos m-de-n (m/n)

- Existen “n” bits en la palabra de código, de los cuales “m” son “1”.
  - Capacidad de detección:
    - Tienen d = 2, luego detectan todos los errores de 1 bit.
    - **Detectan todos los errores m.u.a**, pues siempre cambian el número de “1” de la palabra original.
  - Códigos **separables**
    - Es un código i-de-2i
    - Redundancia del 100 %
    - Son de fácil implementación: codificación, decodificación y detección
  - Códigos **no separables**
    - Tienen menor redundancia
    - Al ser no separables, es difícil la implementación de codificadores y detectores.
    - Los códigos en los que n = 2m pueden ser separables.
    - Número de palabras de códigos usuales: (2/5): 10; (3/6): 20.

FTF

Ejemplos de códigos m-de-n (m/n)

Info BCD	Código 2/5
0000	00011
0001	11000
0010	10100
0011	01100
0100	10010
0101	01010
0110	01100
0111	10001
1000	01001
1001	00101

No separable

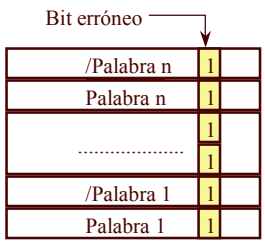
Info	Código 3/6
000	000 111
001	001 110
010	010 101
011	011 100
100	100 011
101	101 010
110	110 001
111	111 000

Separable

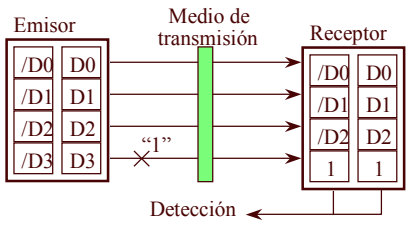
FTF

Códigos duplicados

- Construidos añadiendo a los bits de datos, una copia de éstos mismos.
  - Utilización en memorias y en transmisión de datos.
  - Ventaja: Muy sencilla codificación y detección.
  - Inconvenientes:
    - Redundancia R = 1 (Caros)
    - No detección de los errores de modo común. (afectan a las 2 copias).
    - Solución a los errores de modo común: Código duplicado complementario.



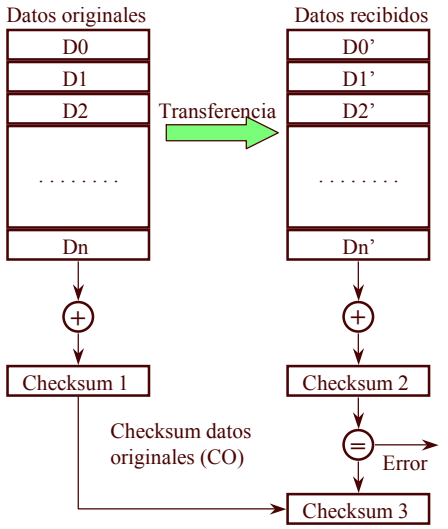
En memorias



En transmisión de datos

FTF

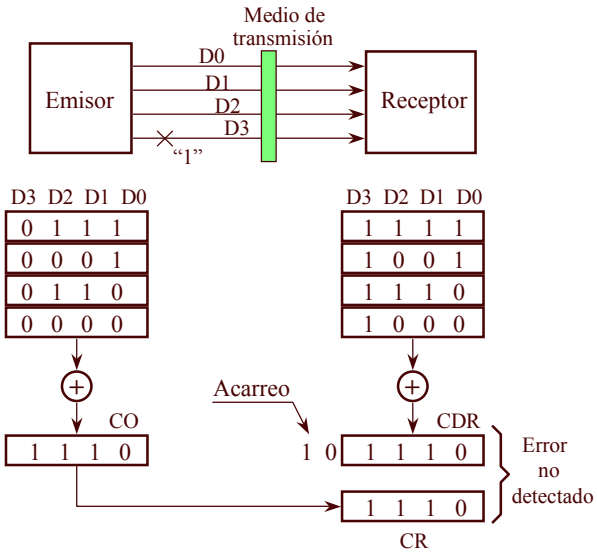
Sumas de prueba (Checksums)



- Aplicación:
  - Memorias de sólo lectura
  - Transmisión de datos
- Tipos:
  - De simple precisión
  - De doble precisión
  - De Honeywell
  - De residuo
  - De precisión extendida

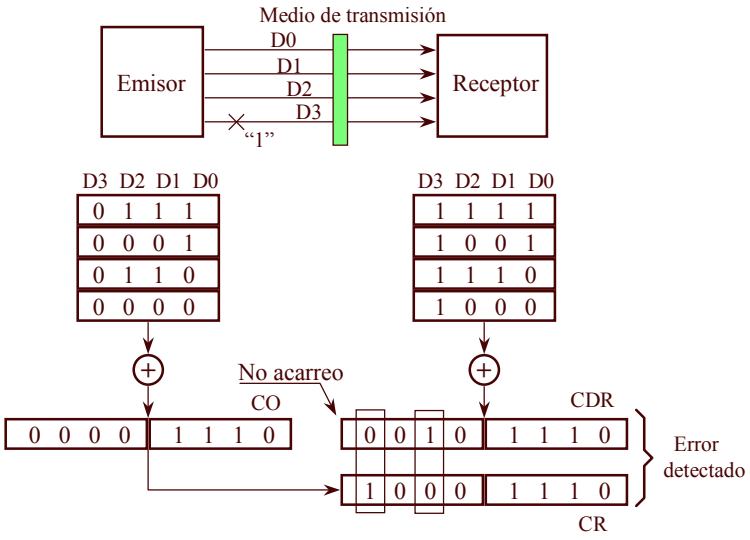
FTF

- **Checksum de simple precisión**: Suma módulo 2^n , siendo *n* el número de bits:



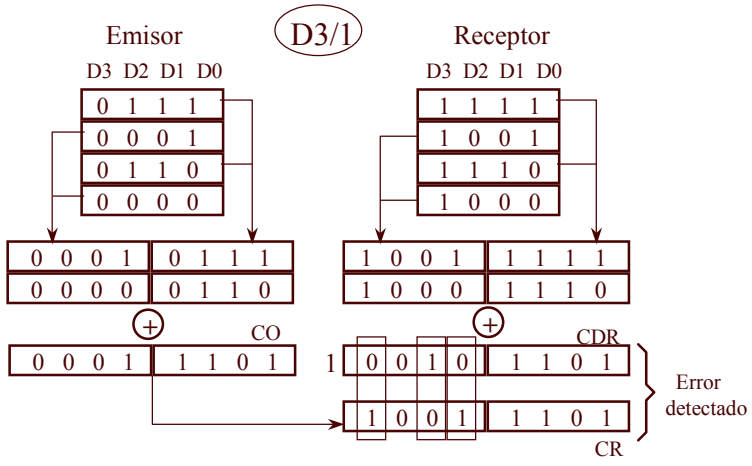
FTF

- **Checksum de doble precisión**: Suma módulo 2^{2n} , siendo *n* el número de bits. Es uno de los más utilizados:



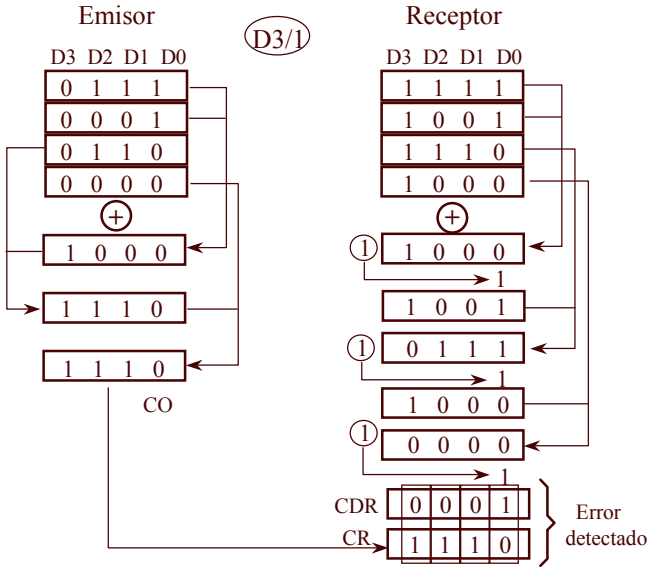
FTF

- **Checksum de Honeywell**: Modificación del de doble precisión, que intenta, para mejorar la cobertura, que un error de una columna afecte a 2 columnas en el checksum



FTF

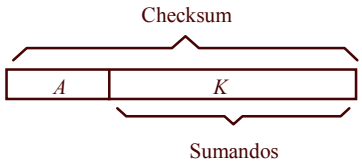
– **Checksum de residuo**:Derivado del de simple precisión, suma siempre el acarreo



FTF

- **Checksum de precisión extendida**: Es aquel que posee una cobertura en la detección de errores del 100%:
  - Para ello, no debe haber acarreo, para no perder información por él.
  - Si hay *S* sumandos, el checksum tendrá *A* bits extra añadidos a los de los sumandos, y para que no haya acarreo, deberá cumplirse que:

$$S \leq 2^A$$



S	A
1	0
2	1
3 - 4	2
5 - 8	3
9 - 64	6
65 - 128	7
129 - 256	8
257 - 1024	10

FTF

Códigos cíclicos

- Se basan en la Comprobación de la redundancia Cíclica (*Cyclid Redundancy Check*)
- Se utilizan cuando se transmite información en serie. En el caso de códigos cíclicos separables, la trama se construye concatenando después de los datos un conjunto de bits que forman la secuencia de control de trama (FCS – *Frame Control Sequence ó CRC*).
- Cada CRC se obtiene a través de su polinomio generador, cuyos coeficientes son siempre 0 ó 1 al tratarse de un CRC binario.
- No separable: Multiplica el dato por el polinomio generador (sumas en módulo 2)
- Separable:
  - *M* es la secuencia de *n* bits a transmitir (bits de datos, BD)
  - *P* es un polinomio generador de *(k+1)* bits
  - *FCS* será el resto de la división del polinomio 2^kM por P
- La información recibida estará libre de errores si es divisible por el polinomio generador del código.
- Son fáciles de implementar mediante registros de desplazamiento con carga en paralelo, realimentando la salida de datos serie.
- Detectan errores unidireccionales. Cuanto mayor sea el grado del polinomio, más bits erróneos se podrán detectar.

FTF

Códigos aritméticos

- Utilizados para comprobar circuitos que hacen operaciones aritméticas:
  - Debe preservarse el código en los resultados de estas operaciones.
  - Características principales de un código aritmético *A* para la operación aritmética *

$$A(b * c) = A(b) * A(c)$$

  - Se dice, entonces, que el código *A* es **invariante** para la operación *
  - Los códigos estudiados hasta ahora (paridad, etc.) no son invariantes ante operaciones aritméticas:
    - Para códigos separables no aritméticos puede predecirse, a partir de los BC de los operandos, los BC del resultado de la operación, pero es un problema complicado.
- Tipos principales de códigos aritméticos:
  - Códigos *AN*.
  - Códigos de residuo.
  - Códigos de residuo inverso.

FTF

- **Códigos *AN***
  - Formados multiplicando la palabra original (BD) por la constante A.
  - No separables.
  - Invariantes para la operaciones de suma y resta:

$$AN_1 + AN_2 = A(N_1 + N_2)$$
$$AN_1 - AN_2 = A(N_1 - N_2)$$

  - Para códigos binarios, A no debe ser potencia exacta de 2:
    - Sea $A = 2^a \Rightarrow$

$$N = (b_{n-1}b_{n-2}\cdots b_1b_0) = 2^{n-1}b_{n-1} + 2^{n-2}b_{n-2}+\cdots+2^1b_1 + 2^0b_0$$
$$AN = 2^aN = 2^{a+n-1}b_{n-1} + 2^{a+n-2}b_{n-2}+\cdots+2^{a+1}b_1 + 2^{a+0}b_0 + 2^{a-1}0+\cdots+2^00$$

  - Un error en cualquier bit b da lugar a una palabra que pertenece al código (es divisible siempre por A) → Este código no detecta todos errores simples.
  - Ejemplos de códigos binarios *AN*:
    - *3N*, para detectar errores de 1 bit.
    - . . . . . .
    - *15N*, utilizado por Avizienis en la ALU de su JPL--STAR Computer (1971).
    - El código *3N* es el más popular y barato, pues BC = 2 (BPC = BD + 2)

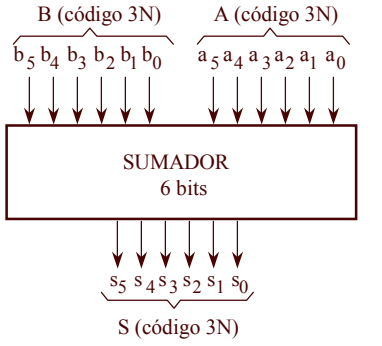
FTF

- Ejemplo: Código *3N* para 4 bits (BD):

Info	Código 3N
0 0 0 0	0 0 0 0 0 0
0 0 0 1	0 0 0 0 1 1
0 0 1 0	0 0 0 1 1 0
0 0 1 1	0 0 1 0 0 1
0 1 0 0	0 0 1 1 0 0
0 1 0 1	0 0 1 1 1 1
0 1 1 0	0 1 0 0 1 0
0 1 1 1	0 1 0 1 0 1
1 0 0 0	0 1 1 0 0 0
1 0 0 1	0 1 1 0 1 1
1 0 1 0	0 1 1 1 1 0
1 0 1 1	1 0 0 0 0 1
1 1 0 0	1 0 0 1 0 0
1 1 0 1	1 0 0 1 1 1
1 1 1 0	1 0 1 0 1 0
1 1 1 1	1 0 1 1 0 1

- d = 2 → Detecta todos errores de 1 bit
- R = 2/4 = 1/2

- Utilización del código 3N en un sumador:

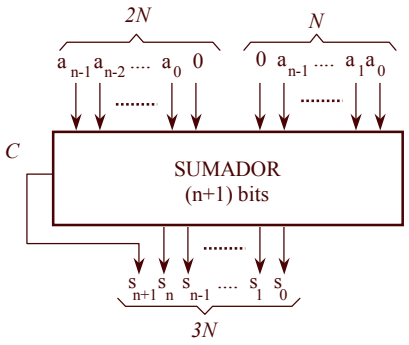


- Ejemplo de detección de 1 error en la suma:

A = 0 1 0 0 1 0 (6 en 3N)	A = 0 1 0 0 1 0 (6 en 3N)
+ B = 0 0 0 0 1 1 (1 en 3N)	+ B = 0 0 0 0 1 1 (1 en 3N)
S = 0 1 0 1 0 1 (7 en 3N)	S' = 0 1 0 1 1 1 (∉ 3N)
Suma sin fallo	Suma con s_1 p-a-1 → detección del error

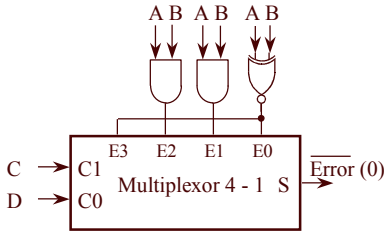
FTF

- Generador de código *3N* :



- Detector de código *3N*
  - Por ejemplo, un detector del código 3N para nº (0-5) (BPC = 4)

AB / CD	00	01	11	10	
00	1		1		E_0
01				1	E_1
11	1		1		E_3
10		1			E_2



FTF

- Códigos de residuo:
  - Son códigos separables, formados al unir los bits de datos (BD) con el residuo *r*
  - El residuo *r* de un número N es el resto que se obtiene de realizar la división entera de ese número por otro número entero *m*

$$R(N) = r = N \bmod m$$

  - Siempre se cumple que $0 \le R < m \rightarrow$ R tendrá $\log_2(m)$ bits de longitud
  - Los códigos de residuo son invariantes a las operaciones de suma:

$$R(N_1 + N_2) = (R(N_1) + R(N_2)) \bmod m$$

  - **Ejemplo**: Para N_1=14 y N_2 = 7 y siendo m=3
    - R(14) = 14 mod 3 = 2 ; R(7) = 7 mod 3 = 1

R(14+7) = R(21) = 21 mod 3 = 0

(R(14) + R(7)) mod 3 = (2 + 1) mod 3 = 0

  - La ventaja de los códigos de residuo es que son separables.
  - La redundancia es de *R* / BD

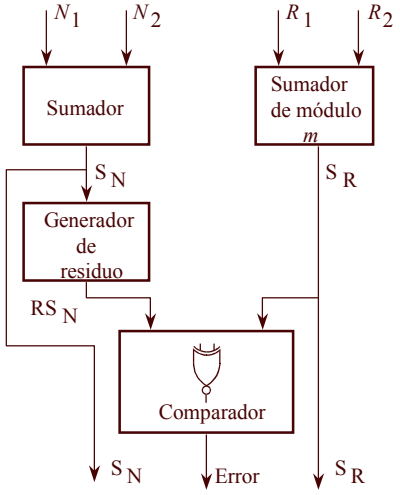
FTF

- Ejemplo: Código de residuo con *m=3* para 4 bits (BD):

Info	Código	
	Datos	Residuo
0 0 0 0	0 0 0 0	0 0
0 0 0 1	0 0 0 1	0 1
0 0 1 0	0 0 1 0	1 0
0 0 1 1	0 0 1 1	0 0
0 1 0 0	0 1 0 0	0 1
0 1 0 1	0 1 0 1	1 0
0 1 1 0	0 1 1 0	0 0
0 1 1 1	0 1 1 1	0 1
1 0 0 0	1 0 0 0	1 0
1 0 0 1	1 0 0 1	0 0
1 0 1 0	1 0 1 0	0 1
1 0 1 1	1 0 1 1	1 0
1 1 0 0	1 1 0 0	0 0
1 1 0 1	1 1 0 1	0 1
1 1 1 0	1 1 1 0	1 0
1 1 1 1	1 1 1 1	0 0

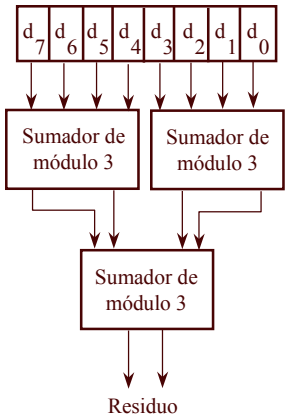
- d = 2 → Detecta todos errores de 1 bit
- R = 2/4 = 1/2

• Ejemplo de aplicación en un sumador con código de residuo separable de módulo *m*:

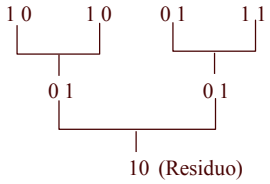


FTF

- Códigos de residuo de coste mínimo (fácil implementación):
  - Si se cumple que: $m = 2^b - 1$; con $b \geq 2$
    - entonces el número de bits a añadir será de *b*.
  - Son de coste mínimo por su **fácil implementación**, ya que la división para obtener *R* se hace a base de sumas.



Ejemplo: $N = 10100111$ (167_{10}) ; $m = 3$; $b = 2$



Se agrupan los datos en conjuntos de b bits, y se realiza la suma modulo $(2^b - 1)$ de los grupos para formar el residuo

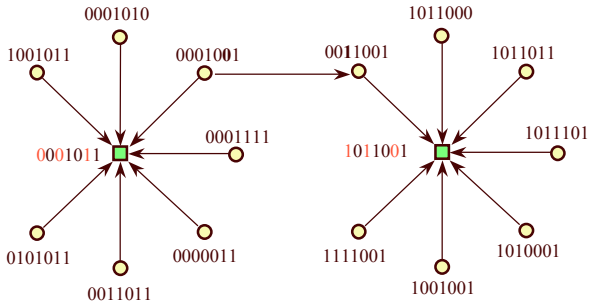
FTF

Códigos Berger

- Son códigos separables. A los bits de datos se les añade los bits de comprobación (k)
- Para hallar k se calcula el complementario del número de 1's que existen en los bits de datos
  - BPC = BD + k con $k = \log_2 (BD + 1)$
  - Ejemplo: 0111010 hay 4 1's (**100**) BPC = 0111010**011**
- Si $d = 2^k - 1$ el código se llama Código Berger de máxima longitud
  - (Ejm BD=7 y k=3)
- Ventajas:
  - Separable
  - Detecta errores múltiples direccionales

FTF

Códigos correctores de errores



- Con d = 3, hay (al menos) 2 palabras erróneas entre 2 correctas →
  - **Si falla 1 bit, se puede corregir** sustituyendo la palabra errónea por la de código más cercana, que es además la correcta (d = 1).
  - **Si fallan 2 bits**, la palabra resultante puede quedar a d = 1 de otra de código que no es la correcta, luego **no se puede corregir** (sí detectar).

FTF

- Un error de "n" bits supone:
  - Que la palabra resultante queda a $d = n$ de la palabra original perteneciente al código
  - Que para poder ser corregido, la palabra resultante deberá de quedar a $d \geq n + 1$ de cualquier otra palabra de código ⇒
- Para que un código **corrija** errores en "n" bits, se debe cumplir:

$$d \geq 2n+1$$

- Para que un código **detecte** errores en "n" bits, se debe cumplir:

$$d \geq n+1$$

- Por lo tanto, un código de distancia "d ":
  - Detectará hasta $(d - 1)$ errores.
  - Corregirá hasta $(d - 1) / 2$ errores.
- Corrección de errores ≡ Detección del bit (bits) erróneos.

FTF

Código **SEC** (Single Error Correcting) de Hamming

- Código de paridad, separable, que corrige errores de 1 bit (d=3).
- IDEA: Los bits de información se dividen en grupos y a cada uno se le asigna un bit de paridad (**paridad superpuesta**). Los grupos se forman de manera que cada bit de datos (d_0,d_1, ..., d_k) aparece en más de un árbol de paridad (p_0,p_1, ... , p_c).
  - Se puede por tanto localizar el error
  - Se puede corregir mediante la complementación del bit erroneo

- Los "c" bits del vector de síndrome deben proporcionar combinaciones únicas para cada uno de los bits de datos o de código que pueden fallar ⇒

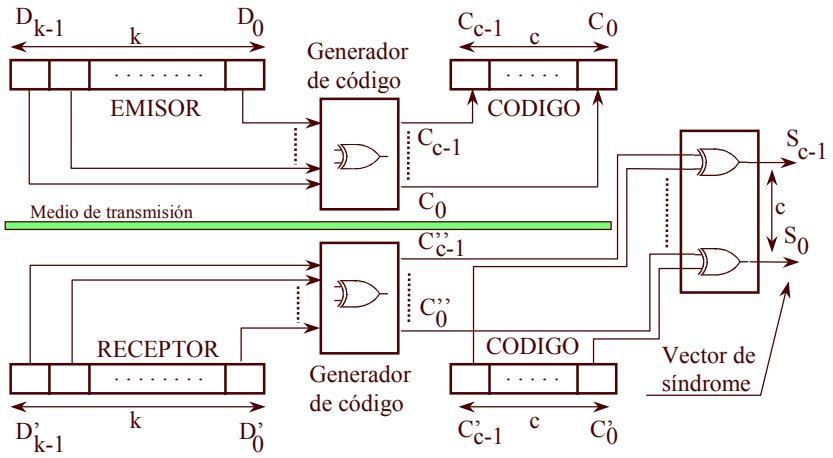
$$2^c \geq k + c + 1$$

Ejemplos de códigos SEC

k	c	Código SEC	Redundancia (%)
4	3	(7,4)	75%
8	4	(12,8)	50%
16	5	(21,16)	31,25%
32	6	(38,32)	18,75%
64	7	(71,64)	10,94%

FTF

Sistema con código SEC de Hamming



FTF

- Si se tienen 4 bits de información (d3,d2,d1 y d0), se pueden particionan en grupos a los cuales se le añade otro bit de paridad:

  (d3,d1, d0, c0)
  (d3, d2, d0, c1)
  (d3, d2, d1, c2)

Bit Erróneo	Bits de código afectados
d0	c0, c1
d1	c0, c2
d2	c1, c2
d3	c0, c1, c2
c0	c0
c1	c1
c2	c2

- Las combinaciones de los bits de código en los que influye el bit erróneo son únicas, por lo tanto éste siempre se podrá localizar
- Proceso de **codificación**
  1. Se codifican los datos para obtener los bits de código (bits de paridad) utilizando un generador de código
- Proceso de **comprobación**
  1. Se vuelve a codificar la información
  2. Se construye un vector de síndrome S que es el resultado de realizar la or-exclusiva de los bits de código originales con los que se acaban de generar.
  3. Este vector puede indicar directamente el bit erróneo si se ordenan los bits de forma apropiada. Si el síndrome es todo 0's indica que no ha habido error.

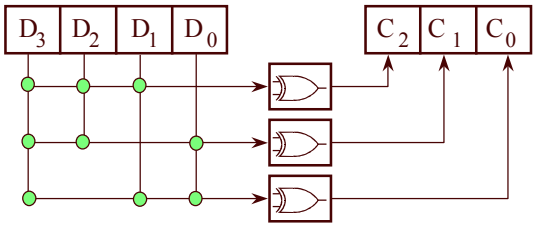
FTF

El orden de los bits de código y datos será:

7	6	5	4	3	2	1
D_3	D_2	D_1	C_2	D_0	C_1	C_0

Palabra codificada

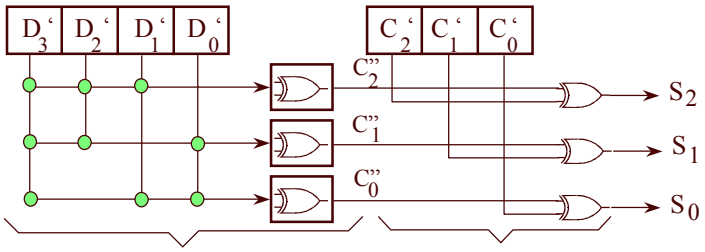
– Generador de código SEC (7,4):



$D_i \rightarrow$ Datos originales
$C_j \rightarrow$ Código original

FTF

– Generador del vector de síndrome:



Generador de código

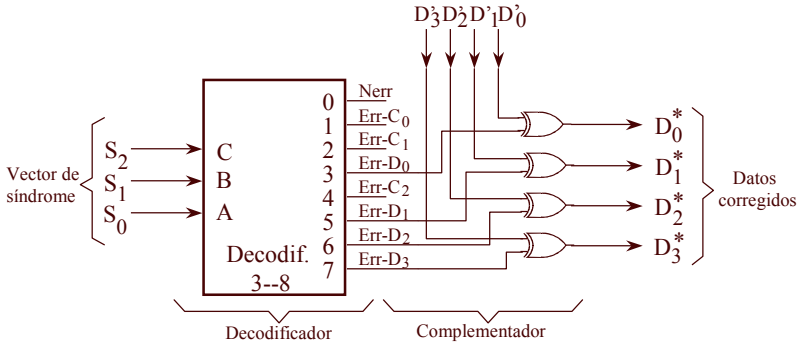
Comparador

$D_i' \rightarrow$ Datos recibidos
$C_j' \rightarrow$ Código recibido
$C_j'' \rightarrow$ Código generado
$S \rightarrow$ Vector de síndrome

S_2	S_1	S_0	Bit erróneo
0	0	0	No error
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

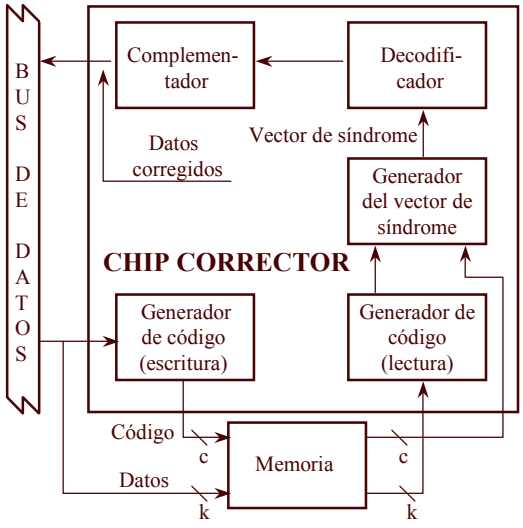
FTF

- Corrector de errores



FTF

- Circuitos integrados comerciales correctores de errores:



Ejemplos de chips correctores de errores comerciales (16 bits):

- Intel 8206
- Motorola MC68540
- AMD AM2960
- AMD AMZ8160
- National Sem. DP8400
- Fujitsu MB1412A

FTF

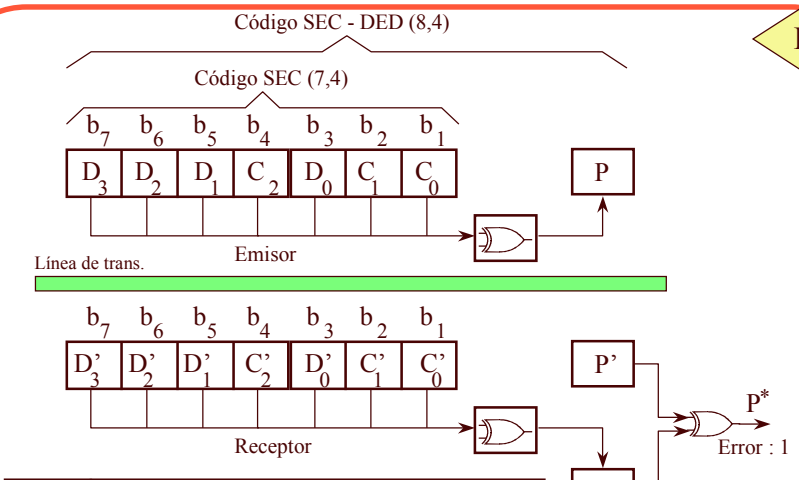
- El código **SEC - DED** (Single Error Correcting - Double Error Detecting) **de Hamming:**
  - Evita los problemas del SEC cuando intenta corregir errores de 2 bits, provocando más errores de los iniciales:

Para el código (7,4):

$$ArbolPar_2 = d_3 \oplus d_2 \oplus d_1 \oplus c_2$$
$$ArbolPar_1 = d_3 \oplus d_2 \oplus d_0 \oplus c_1$$
$$ArbolPar_0 = d_3 \oplus d_1 \oplus d_0 \oplus c_0$$

  - Si falla c0 y c1 → {S2 S1 S0} = {0 1 1} → ¡ Se corrige d0!
  - Si falla c0 y d0 → {S2 S1 S0} = {0 1 0} → ¡ Se corrige c1!
  - Si falla c0 y c2 → {S2 S1 S0} = {1 0 1} → ¡ Se corrige d1!
  - Para evitar estos problemas para errores de dos bits, el código SEC-DED detecta los errores de dos bits, impidiendo la corrección de un bit erróneo.
  - El código SEC-DED se forma a partir del SEC añadiendo 1 bit de paridad.

FTF

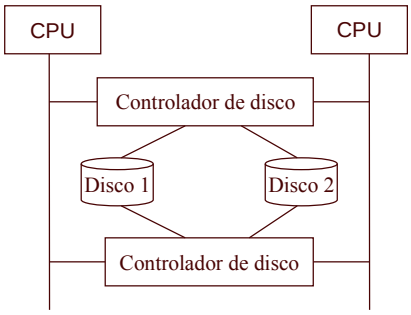


Síndrom	P*	Situación
000	0	No hay error (ni de 1 ni de 2 bits)
No 000	1	Error de 1 bit → Habilitación del complementador
No 000	0	Error de 2 bits → Deshabilitación complementador
000	1	?? Posible situación de triple error o fallo b. paridad

FTF

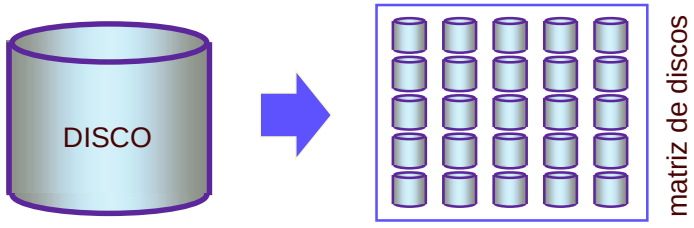
Memoria estable

- Es una de las formas de redundancia en la información más utilizadas para conseguir un almacenamiento fiable de los datos.
- Disk Shadowing
  - La información se duplica en **2 discos** (discos espejo -*disk mirroring-).*
  - Evita puntos únicos de fallos: discos y controladores replicados
  - El driver de acceso a disco se encarga del acceso a los dos dispositivos.



FTF

- RAID - *Redundant Arrays of Inexpensive Disks*
  - Diversos estudios sobre las operaciones de E/S pusieron de manifiesto que el 80% de las mismas se localizaban sobre un área que cubría únicamente un 20% del espacio disponible. A pesar de la existencia de múltiples dispositivos, si este 20% recaía sobre uno sólo, el rendimiento bajaba
  - Debido a la imposibilidad de localizar esta área, se procedió a repartir la información sobre varios dispositivos, en una técnica llamada "*striping*". El usuario observaba un único dispositivo lógico formado por varios discos.
  - **Ventajas**: Aumentan las prestaciones al distribuirse las operaciones entre los discos (un acceso servido por varios discos en paralelo y varios accesos servidos concurrentemente), son más baratos y si junto muchos puedo obtener un dispositivo lógico de gran capacidad.
  - **Problemas**: Se precisan controladores especiales, el MTBF del conjunto es elevado ya que una avería de un disco produce una avería total ⇒ poca fiabilidad

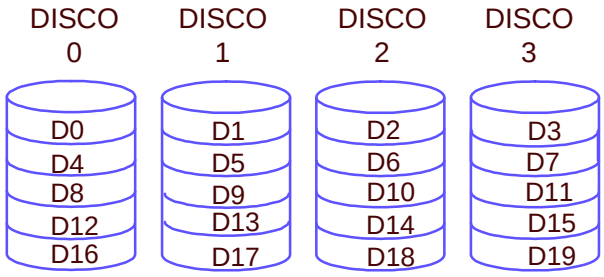


FTF

- En función de cómo se distribuyen los datos y de cómo se implementa la redundancia se definen los diferentes niveles RAID.
  - Sobre este esquema, ¿cómo aumentar la fiabilidad?
    - Hay que añadir algún disco redundante para que el fallo de un disco no implique una avería general: **RAID**
  - Hay distintos tipos de RAIDS
    - Los discos espejo son el tipo más elemental de RAID, aunque tienen un 100% de sobrecarga
    - Normalmente se dispone de un número *n* de discos más *1* que efectúa misiones de *check disk*
    - RAID nivel 0, 1, 2, 3, 4, 5 y 6
      - Diferentes estrategias de redundancia y reparto de la información
    - Mecanismos de redundancia
      - Replicación de datos *(mirroring)*
      - Códigos de paridad
        - Disco de paridad
        - Paridad distribuida
      - Códigos tolerantes a fallos múltiples

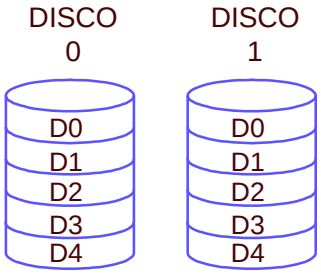
FTF

- RAID nivel 0
  - Los datos se dividen en bloques que se reparten por los discos (*data striping*). Si el sistema es multiusuario conviene que las peticiones quepan en un bloque para poder atender a varios usuarios a la vez.
  - No existe redundancia.
  - Si falla un disco falla todo el dispositivo.
  - Es el más rápido y eficiente.

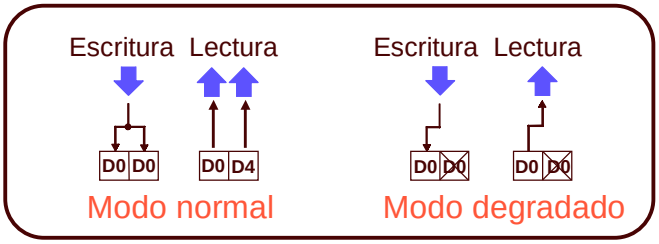


FTF

- RAID nivel 1



- Son discos espejo
- Si falla un disco no se interrumpe el servicio ni se pierden los datos
- Coste elevado

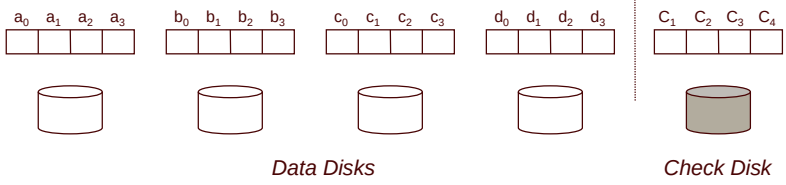


FTF

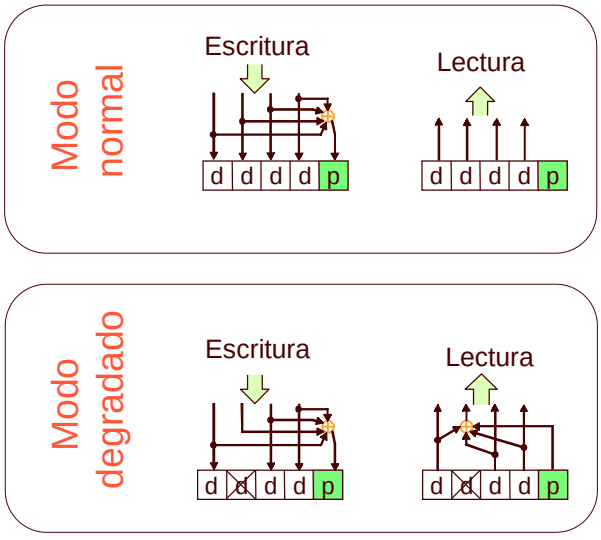
- RAID nivel 0+1, 1+0, 10

FTF

- **RAID 3**
  - Los datos se distribuyen en trozos pequeños entre los discos.
  - Trabajan todos a la vez de forma sincronizada (Sólo una petición en cada momento).
  - Existe redundancia en un disco adicional que guarda la paridad.
  - El error en un disco se detecta mediante los códigos ECC del sector y el dato se puede reconstruir mediante la paridad del disco adicional.
- **RAID 4**
  - Se utilizan bloques grandes de datos para que cada petición quepa en uno
  - Se pueden realizar lecturas simultáneas si no hay error (no se lee paridad)
- **RAID 5 (Array de paridad rotatorio)**
  - Se utilizan bloques grandes de datos para que cada petición quepa en uno sólo
  - Se pueden realizar escrituras simultáneas (sólo se usan dos discos)

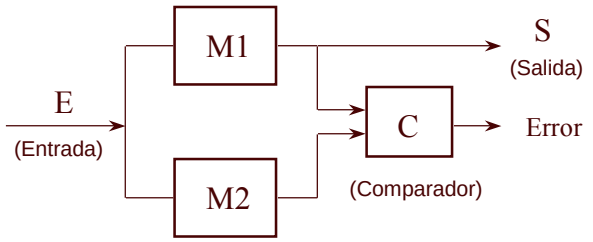


FTF



FTF

3.2 - Duplicación y comparación



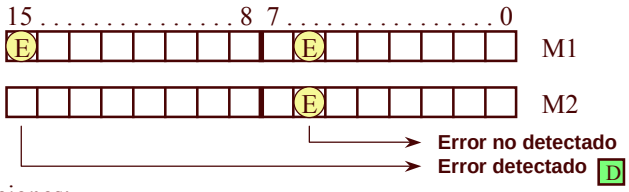
M1, M2: Módulos con igual función

Problemas:

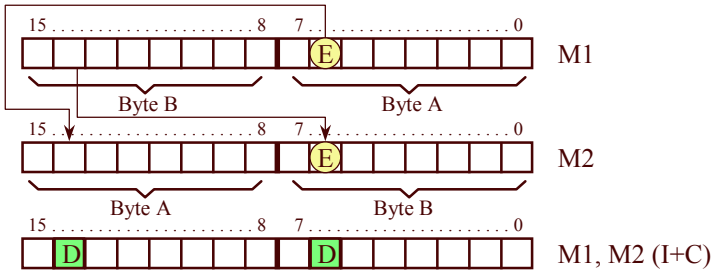
- Fallos de modo común: En memorias RAM y otros circuitos VLSI.
- Fallos de diseño

FTF

- Fallos de modo común en memorias:

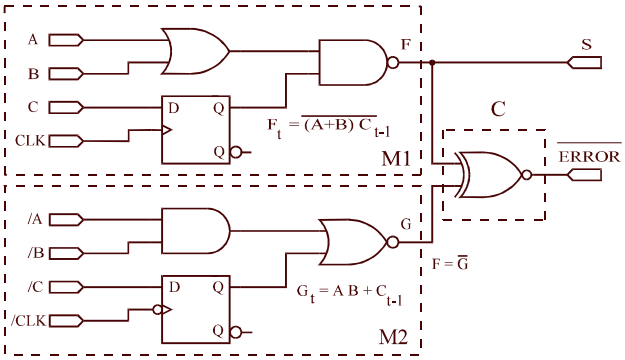


- Soluciones:
  - Integrados diferentes o posiciones no contíguas
  - Intercambiar y comparar (I+C):



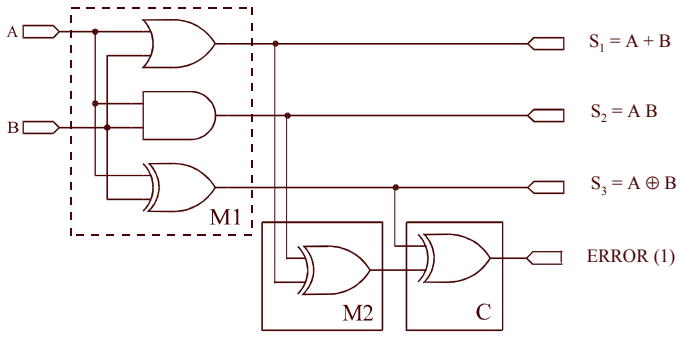
FTF

- Fallos de modo común en otros chips VLSI:
- Soluciones:
  - Técnica de diversificación funcional mediante el uso de lógica complementaria en cada módulo



FTF

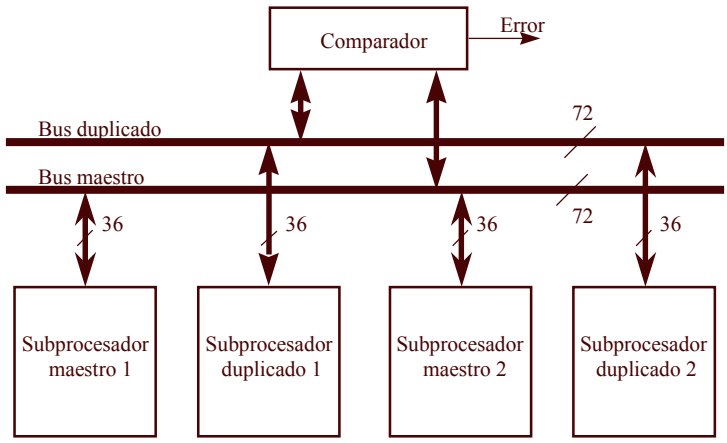
- Duplicación sin el doble de componentes (ALU AM 2901):



$$M_2 = S_1 \oplus S_2 = (A+B) \oplus AB = S_3 = A \oplus B$$

FTF

- Duplicación a nivel de bus: UNIVAC 1100/60:



FTF

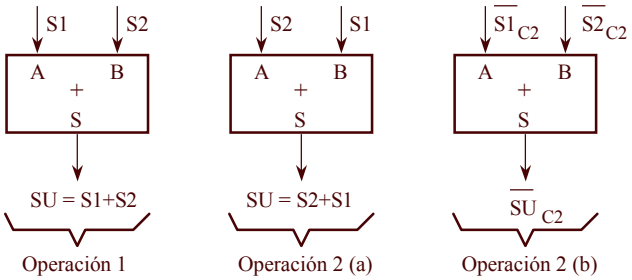
Análisis del coste y de las prestaciones en los sistemas duplicados

- Coste > 2*coste de sistema simple + coste comparador
- Pérdida de prestaciones por:
  - En sistemas VLSI
    - Pérdida de tiempo en la sincronización de los módulos.
    - Pérdida de tiempo por el retardo del comparador.
  - En memorias, pérdida de tiempo en la realización de dos copias y en su comprobación posterior.
  - En memorias con la técnica de intercambiar y comparar, mayor pérdida de tiempo.
  - En procesadores duplicados, disminución de prestaciones por aumento de tráfico en el bus: Aumento del ancho de banda de bus consumido.

FTF

Mejoras en el coste: Redundancia temporal

- Consiste en la utilización de un solo módulo que realiza 2 veces la misma operación, comparando los resultados obtenidos.
  - Ventajas:
    - Menor coste por menor redundancia.
  - Inconvenientes:
    - Tiempo de respuesta mayor del doble.
    - No detección de los errores permanentes.
      - Solucion:



FTF

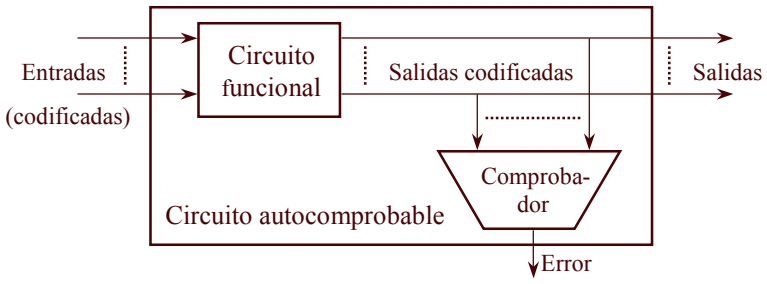
Mejoras en las prestaciones en procesadores duplicados

- Duplicación a nivel de tarea:
  - División entre tareas normales (no duplicadas) y de test (duplicadas)
  - División entre tareas normales (no duplicadas) y peligrosas (duplicadas)
- Utilización de buses especiales para intercambio de información entre procesadores:
  - Ejemplo: Dynabus de TANDEM
  - Problema: Elevado coste, sólo para aplicaciones especiales.

FTF

3.3 - Circuitos autocomprobables (self-checking)

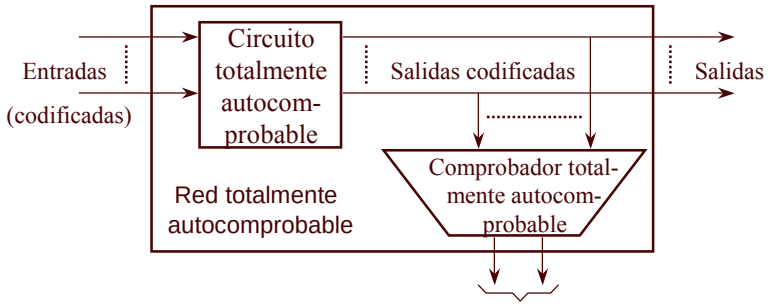
- Circuitos con la capacidad interna de autocomprobarse.



- Un circuito es **totalmente autocomprobable** para un conjunto de fallos F, si para cualquier fallo ∈ F se cumple que para cualquier palabra codificada de la entrada, no produce nunca una palabra codificada incorrecta en la salida (seguro), y además existe al menos una palabra codificada en la entrada que produce una palabra no codificada en la salida (autotesteable)

FTF

- **Redes totalmente autocomprobables**:

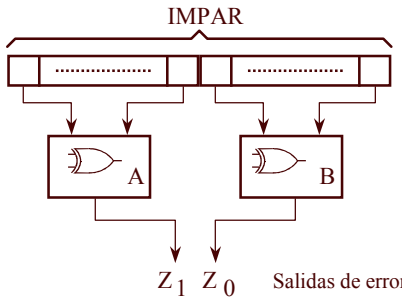


- Un **comprobador** es **totalmente autocomprobable** si es un circuito totalmente autocomprobable y además cualquier palabra codificada en la entrada produce siempre una palabra codificada en la salida y viceversa (de código disjunto).
  - Salida de error detecta errores de circuito y de comprobador
  - Salida de error debe estar codificada (al menos 2 bits):
    - Código duplicado : (00, 11) → No detecta errores de todo a "0" y todo a "1"
    - Código 1 de 2: (01, 10) → Detecta errores de todo a "0" y todo a "1"

FTF

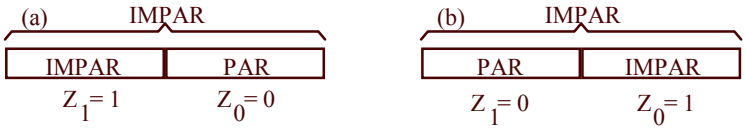
- Comprobadores totalmente autocomprobables para códigos de paridad:

a) Para paridad de entrada impar:



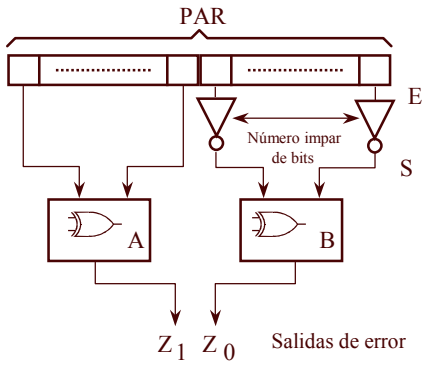
• A, B: Arboles de paridad (Detectores de paridad impar).

– Casos posibles:



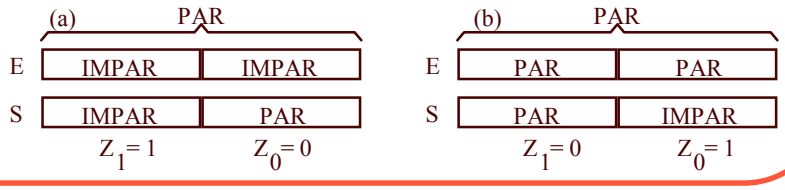
FTF

b) Paridad de entrada par:



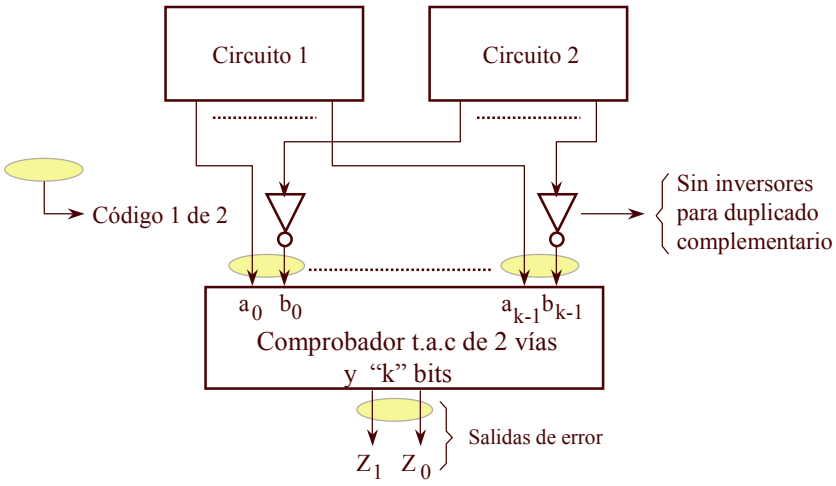
• A, B: Arboles de paridad (Detectores de paridad impar).

– Casos posibles:



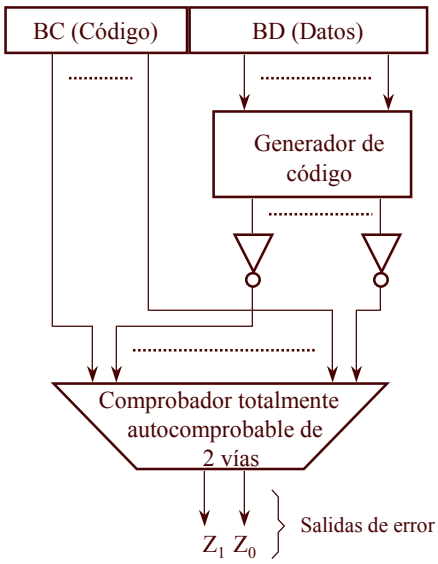
FTF

- Comprobadores totalmente autocomprobables (t.a.c) de dos vías (two rail totally self-checking checkers):
  - Aplicación a códigos duplicados (o duplicados complementarios):



FTF

- Comprobadores t.a.c. para códigos separables:



FTF

3.4- Detección concurrente de errores

- En los sistema con microprocesadores, la mayoría de los fallos son transitorios, por lo que se necesita establecer una metodología para detectarlos.

Método	*Función a observar*
Flujo de programa no válido	Secuencia impropia de instrucciones
Dirección de la instrucción inválida	Busqueda de una instrucción en un segmento de datos
Memoria no inicializada	Acceso a memoria a una dirección existente pero con información inválida
Dirección de lectura inválida	Dirección de lectura para datos en el segmento de código o no existente
Código de operación inválido	Instrucción ilegal
Dirección de escritura inválida	Intento de escritura en memoria de sólo lectura
Memoria no existente	Acceso a una posición sin memoria

FTF

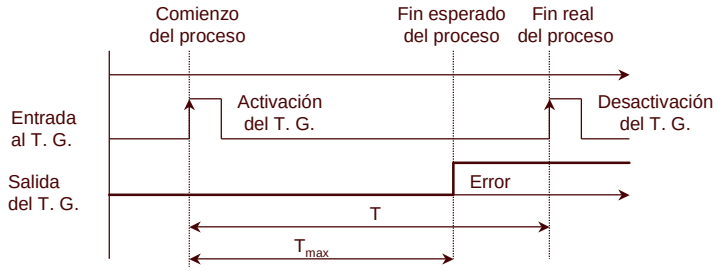
Detección de errores por el propio microprocesador

- Los P de 8 bits no se diseñaron para detectar fallos, sin embargo, los posteriores ya incluyen técnicas de detección de fallos
  - Lectura de instrucción ilegal
  - Instrucciones no implementadas
  - Acceso a memoria no permitido
  - Operación aritmética incorrecta
- Para evitar errores se establecen niveles de privilegio
  - En el 68000 existe un modo usuario y otro supervisor
  - En el iapx 286 hay 4 niveles de privilegio
    - Núcleo del S.O.
    - S.O. compartido
    - Código y datos compartidos
    - Código y datos privados
- Los de 32 bits se diseñan con características para ser comprobados (Testability design)
  - Se autodetecta el componente con error (486)
  - Instrucciones para el manejo y ejecución de excepciones (Intel 80960)
  - Terminales de E/S para el manejo de errores (Acceso erróneo, averia en la inicialización)
  - Monitorización de las transferencias del bus por medio de temporizadores.

FTF

Detección de errores por control de flujo del programa

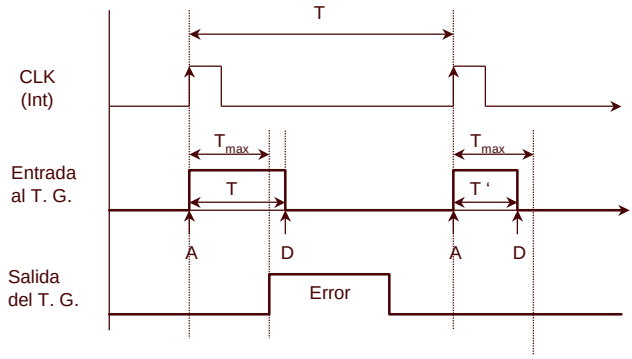
- Estudio de la secuencia de instrucciones
- Es la mejor forma para la detección de errores, aunque la más difícil
- **Temporizadores de guardia**
  - Se utilizan para:
    - Control del tiempo de ejecución del programa (Control de flujo)



T_{max} = Tiempo máximo de ejecución del proceso
T = Tiempo de duración real del proceso

FTF

- Comprobación general del sistema

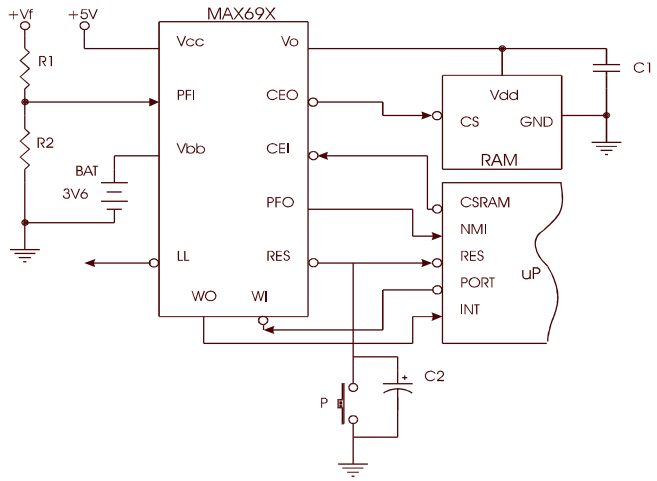


A = Activación
D = Desactivación

T' = Tiempo de disparo del Temporizador de guardia
T_{max} = Tiempo estimado de ejecución del programa de autotest

En cada interrupción se ejecutan programas de autotest

FTF

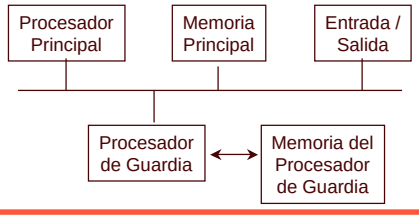


Comprobador general: Detector de fallo de red y watchdog

FTF

Procesadores de Guardia

- Problema: Los T. G. no controlan los errores que no afectan al tiempo
- Un procesador de guardia es un coprocesador que se utiliza para la detección concurrente de errores a nivel de sistema. Realiza un control exhaustivo del flujo del programa mediante la monitorización del bus del sistema.
- Método: Se compara el comportamiento observado con el correcto que se guarda en la memoria local del P.G (fase de inicialización y de comprobación).
- Ventajas frente a la duplicación:
  - Su utilización supone cambios mínimos en el diséño
  - Al ser sistemas independientes, no les afectan los errores en modo común
  - El P.G. Tiene una complejidad menor que el procesador duplicado
  - Al ser una detección concurrente, no se disminuyen las prestaciones
  - Permite detectar errores de programación y de diseño



FTF

- Mecanismo:
  - El programa se divide en nodos y se forma un grafo que representa el flujo de control del programa. Cada nodo tiene asignada una firma o etiqueta.
  - El P.G. debe ser informado de las firmas y de la relación entre ellas. Esta información se le transfiere en la inicialización y da lugar al programa del P.G.
  - Durante la ejecución el P.G. va calculando las firmas (o acepta las que le transmite el procesador principal) en cada nodo, y las compara con las almacenadas (o con las extraídas del programa del procesador).
- Existen diferentes configuraciones en función de cómo se calcula la firma:
  - ***Asignada***: Es de tipo aleatorio
  - ***Derivada u obtenida***: Se calcula a partir de la información del nodo

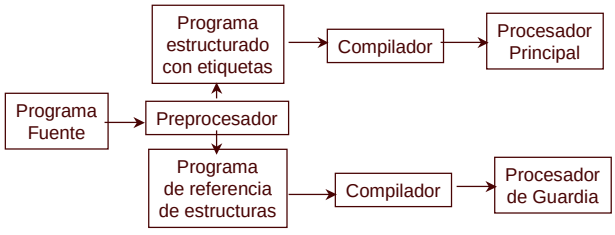
FTF

Firma asignada:

Se realiza una comprobación de la integridad de las estructuras de control del programa (*Structural Integrity Checking - SIC*). El P.G. realiza un seguimiento de las etiquetas que le envía el procesador principal para comprobar su correcta secuencia.

Proceso:

1. Se reconocen las estructuras de control de flujo del lenguaje de alto nivel (secuencia / selección / repetición / llamada a procedimiento)
2. Se etiquetan las estructuras con una firma aleatoria dando lugar al programa de referencia de estructuras.
3. El P.G. Comprueba las etiquetas que recibe



FTF

Programa Fuente

```
readln(maxbucle);
FOR bucle := 1 TO maxbucle DO
  BEGIN
    cuenta := 1;
    suma:= 0;
    readln(n);
    REPEAT
      read(x);
      suma := suma+x;
      cuenta := cuenta+1
    UNTIL cuenta > n;
    media := suma/n
  END
END
```

P. PROCESADOR PRINCIPAL

```
enviar_firma(62);
readln(maxbucle);
enviar_firma(350);
BEGIN
  FOR bucle := 1 TO maxbucle DO
  BEGÍN
     enviar_firma(-50 )
     cuenta := 1;
     suma := 0;
     enviar_firma(341);
     readln(n);
     enviar_firma(430);


     REPEAT;
       enviar_firma(-80);
       read(x);
       suma := suma+x;
       cuenta:=cuenta+1;
       enviar_firma(26)

     UNTIL cuenta > n;
     enviar_firma(-79);
     media:=suma/n;
     enviar_firma(59)
  END

  enviar_firma(-51)
END
```

P. PROCESADOR DE GUARDIA

```
esperar_firma;
IF firma<>62 THEN error(62);
esperar_firma;
IF firma<>350 THEN error(350);
BEGIN

  esperar_firma;
  WHILE firma = -50 DO

    esperar_firma;
    IF firma<>341 THEN error(341);
    esperar_firma;
    IF firma<>430 THEN error(430);

    esperar_firma;
    REPEAT
      IF firma<>-80 THEN error(-80);


       esperar_firma;
       IF firma<>26 THEN error(26);
       esperar_firma ;
    UNTIL firma<>-80;
    IF firma<>-79 THEN error(-79);
    esperar_firma;
    IF firma<>59 THEN error(59);
    esperar_firma;
  END
  IF firma<>-51 THEN error(-51);
END
```


FTF

Firma derivada

Es un proceso más complejo, ya que el P.G tienen que calcular la firma.

- Las firmas se calculan y se almacenan en la memoria del P.G.
- Mientras se ejecuta el programa principal, se calculan las firmas y se compara

Insertada:

El programa principal se modifica para incluir instrucciones que indican al P.G. donde comienza el nodo, donde termina y el valor de la firma.

La firma se calcula realizando la XOR del nodo (aunque a veces se incluye en el procesador principal la firma de un camino completo)

No Insertada:

Las firmas no se insertan en el programa principal sino que se almacenan en la memoria del P.G., formando parte de algún campo en las instrucciones.

El Procesador principal y el de guardia ejecutan de forma sincronizada programas con idénticos diagramas de flujo. A cada nodo del P. Principal le corresponde una única instrucción en el P.G.

Detección de errores hw con programas de comprobación

FTF

- Consiste en ejecutar un programa de test al circuito: sencillo y barato
- Según el momento de realización de los tests
  - Comprobación en la inicialización del sistema (Rutina de RST)
  - Comprobación durante el funcionamiento del sistema (watch-dog)
  - Comprobación cuando se detecta el error (programas de test)
  - Comprobación de mantenimiento
- Se aplican al procesador, a la memoria o a la E/S
  - Procesador: Se activa un temporizador de guardia y se realizan las siguientes fases
    - Instrucciones sencillas
    - Comprobación de buses y acceso a memoria
    - Comprobación de todo el sistema
  - Memoria ROM: Mediante Checksums o firmas
  - Memoria RAM: Son las memorias más sensibles a los fallos
    - Programas de prueba de los fabricantes:
      - Muchos mecanismos de fallo / Tiempos de ejecución largos
    - Programas de prueba concurrentes
      - Menor número de mecanismos de fallos / Tiempo de ejecución cortos
      - Son destructivos
  - Periféricos: Son muy diversos
    - Para periféricos duales (E-S, A/D-D/A) se forma un bucle
    - Temporizador de guardia que se desactiva con salidas al leer un dato por la entrada

4 – Técnicas de recuperación hacia delante

FTF

- La redundancia se basa en la replicación de módulos para lograr la detección del error e incluso la reconfiguración, incrementando la fiabilidad
- Tipos de sistemas hardware redundantes:
  - Las **técnicas pasivas** enmascaran los errores para evitar la producción de la avería. Su objetivo es conseguir la tolerancia a fallos sin necesidad de acciones posteriores por parte del operador o del sistema (Muy rápida).
  - Las **técnicas activas**, también llamadas redundancia dinámica detectan la presencia de los errores y realizan alguna acción para eliminar el hardware averiado del sistema mediante la reconfiguración del mismo.
    - Utilizan un proceso de detección, localización y recuperación del error
  - Las **técnicas híbridas** son una mezcla de las técnicas pasivas y las activas
    - Los errores se enmascaran para evitar la generación de errores en los resultados.
    - También se utilizan técnicas de detección, localización y recuperación del error para incrementar la tolerancia a fallos y reemplazar los componentes averiados por repuestos.
    - Se utilizan en computaciones críticas de tiempo real, pero son muy caros.
- Los sistemas se diferencian principalmente en el tiempo de latencia en la recuperación de errores y en las coberturas

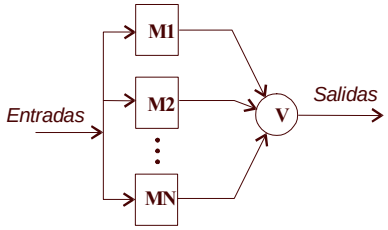
FTF

4.1. - Redundancia pasiva

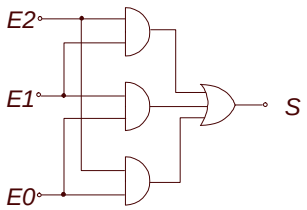
- Utiliza mecanismos de votación para enmascarar la ocurrencia de los errores.
- Se basan en la existencia de N módulos (procesadores, memorias) que realizan la misma función más un votador que realiza el voto por mayoría.
- La salida es correcta mientras la mayoría de los módulos tengan un funcionamiento correcto. Para permitir *F* fallos, necesitaremos *N* módulos, con $N = 2F + 1$
- Los sistemas redundantes con N módulos son los más extendidos cuando se requiere una alta fiabilidad. El sistema más utilizado es el TMR.
- **Problemas**:
  - La fiabilidad del sistema depende del votador. Es un punto único de fallo.
  - Se puede votar al fallo.
  - Elección de un votador hardware o software (barato y flexible, pero lento).
  - Los resultados correctos pueden no coincidir completamente. Se puede utilizar la técnica de selección del valor medio o despreciar los bits menos significativos.
  - Para la votación se precisa de cierta sincronización que introduce retardos.

FTF

- Diseño del votador.



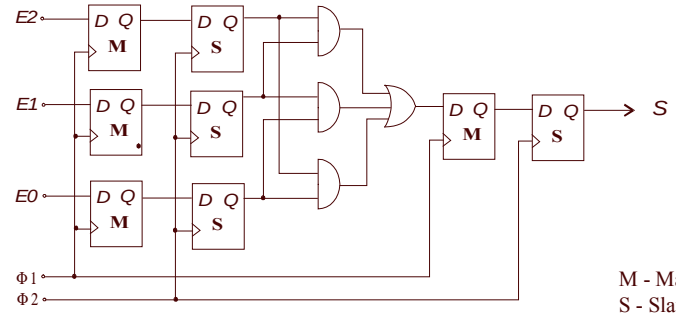
E2	E1	E0	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

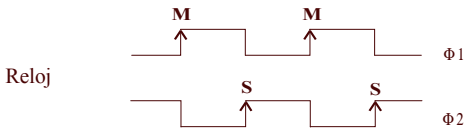


$S = E2\,E1 + E2\,E0 + E1\,E0$

FTF

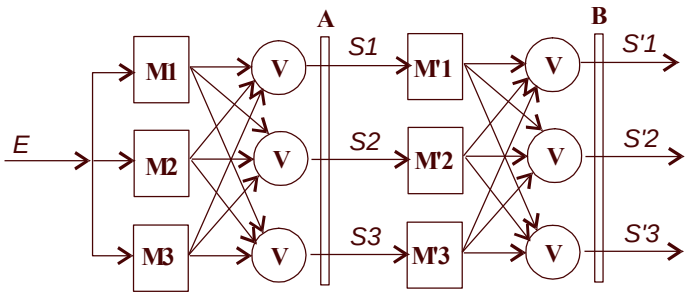
- Votador sincronizado





FTF

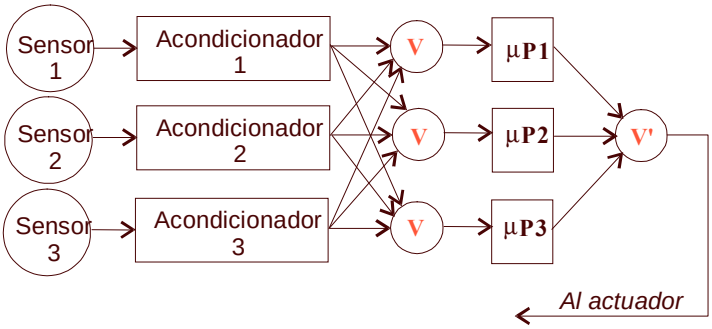
- Sistema TMR con votadores triplicados para 2 etapas



- El sistema tolera el fallo de un módulo o un votador por etapa
- La ventaja es la contención de fallos.

FTF

- La votación puede ocurrir en cualquier nivel

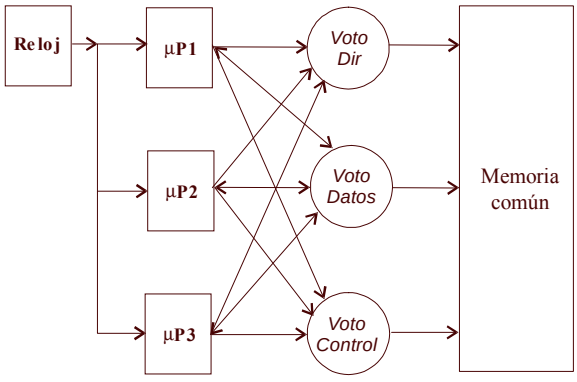


V : Votador sensores (Entradas)
V' : Votador actuador (Salida)

Estructura para sistemas de control

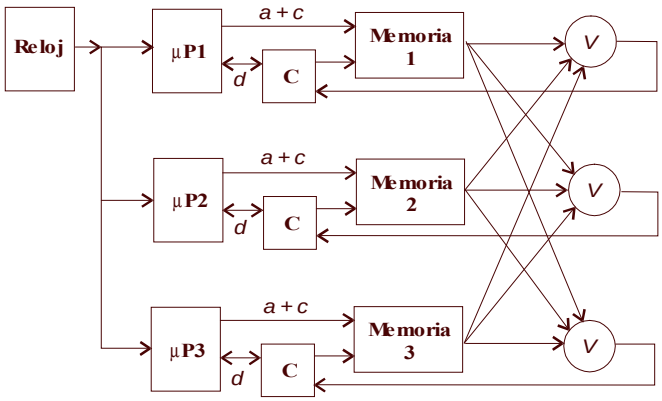
FTF

- Sistemas NMR en microprocesadores



Sistemas con memoria común

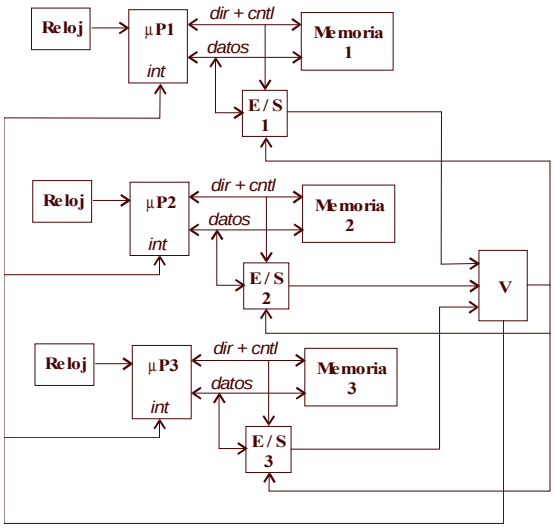
FTF



a + c : dirección + control
d : datos
C: Conmutador

Sistema con memoria no común

FTF



Sistema con memoria no común con voto por E/S

FTF

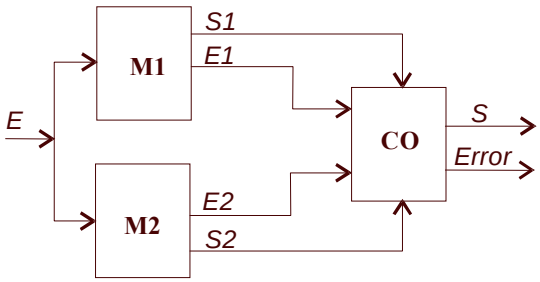
4.2. - Redundancia activa

- Se utilizan técnicas de detección, localización, aislamiento y recuperación de errores.
- No se emplean técnicas para prevenir la activación del error, por tanto la redundancia activa se aplica en sistemas donde se permite la existencia de resultados erróneos temporales, con la condición de que el sistema vuelva a recuperar eventualmente su estado correcto (satélites).
- En la reconfiguración, el sistema puede:
  - Activar un repuesto
  - Funcionar de forma degradada

FTF

- Sistema duplicado con repuesto (dúplex)

Repuesto:
•On line
•Off line

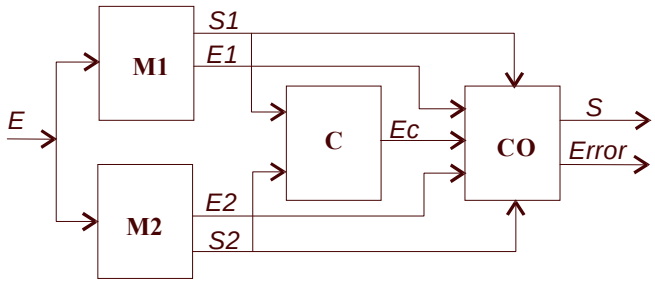


E = Entradas
S = Salidas
E1, E2, Error = Salidas de error

M1, M2 = Módulos con la misma función
CO = Conmutador

FTF

- Sistema duplicado con reconfiguración (dual)



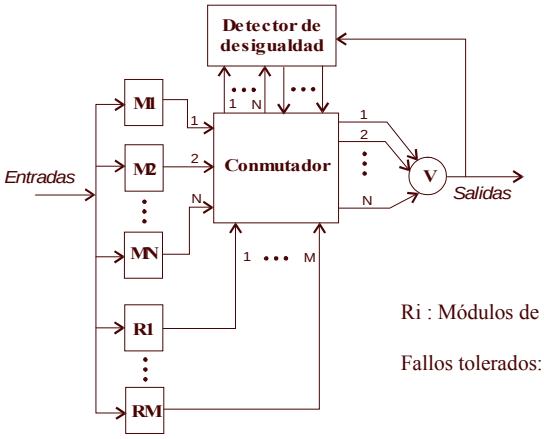
E = Entradas
S = Salidas
E1, E2, Ec = Salidas de error

M1, M2 = Módulos con la misma función
C = Comparador
CO = Conmutador

FTF

4.3. - Redundancia híbrida

- Sistemas NMR con repuesto

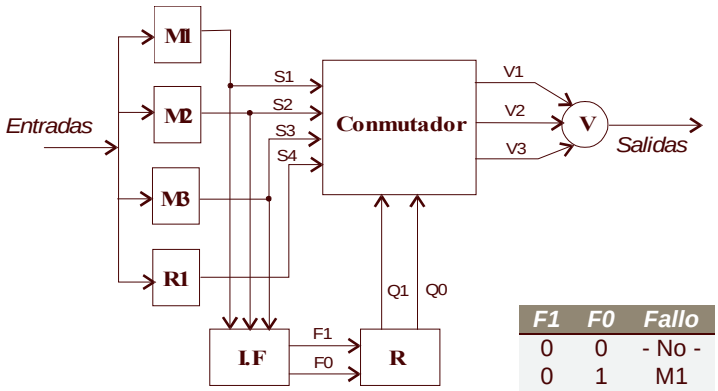


Ri : Módulos de reserva

Fallos tolerados: *M+(N-1)DIV2*

FTF

Ejemplo para sistema TMR con un repuesto

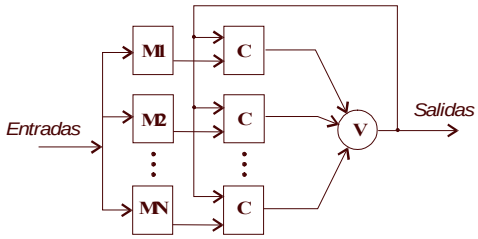


F1	F0	Fallo
0	0	- No -
0	1	M1
1	0	M2
1	1	M3

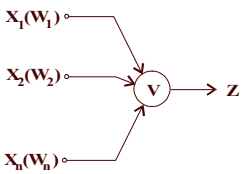
I.F. : Identificador de fallo
R: Registro
V: Votador

FTF

- Sistemas con voto adaptativo (*adaptative voting - Self-purging)*



C: Conmutador
V: Votador (Puerta umbral binaria, o puerta de autopurga)



Puerta umbral binaria:

$$\sum_{i=1}^{i=n} x_i w_i \geq T \Rightarrow Z = 1$$

$$\sum_{i=1}^{i=n} x_i w_i < T \Rightarrow Z = 0$$

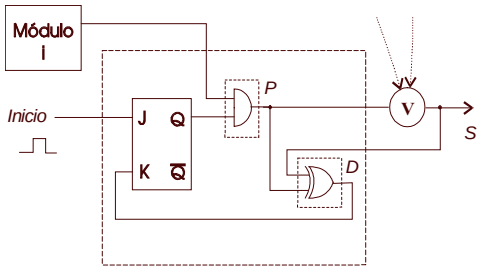
X_i: Entrada i
W_i : Peso de la entrada i
T: Umbral

FTF

Ejemplo de puerta umbral digital:

- Si T=2, i=3, w_i=1, el sistema es un votador mayoritario
- Suma Ponderada: ≥ 2 ; Z = 1
  < 2 ; Z = 0
- Fallos tolerados: *(N-2)*

X_3	X_2	X_1	S. P.	Z
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	2	1
1	0	0	1	0
1	0	1	2	1
1	1	0	2	1
1	1	1	3	1

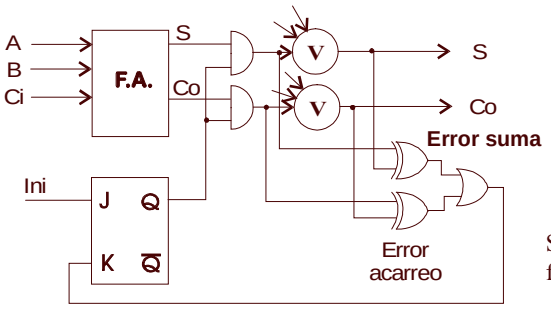


Conmutador

P = Puerta de autopurga
D = Detector del fallo

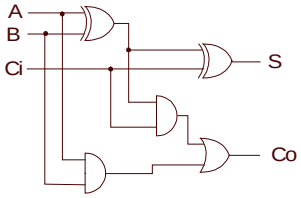
FTF

Sistema TMR en un sumador



A, B : Sumandos
S : Suma
Ci : Acarreo de entrada
Co : Acarreo de salida

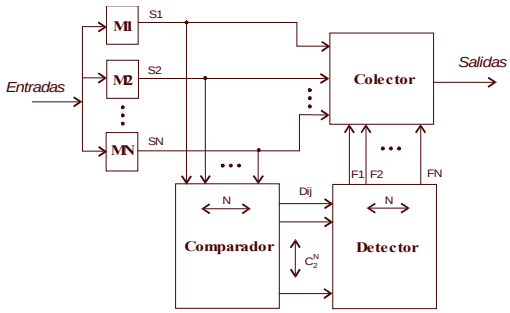
Sistema de desconexión ante el fallo de cualquier S ó Co



F.A.: Sumador completo

FTF

- Redundancia por autocomprobación (Shift-out Redundancy)

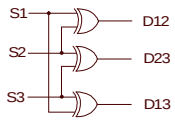


Comparador: Compara las salidas de los módulos 2 a 2.
Salida: 0 Igual salida
1 Diferente salida
Detector: Detecta el módulo con fallo si su salida es minoritaria con respecto a los demás
Salida: 0 Módulo con salida mayoritaria
1 Módulo con salida minoritaria
Colector: Determina la salida en función de las salidas de los módulos y del detector

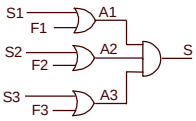
FTF

Sistema con 3 módulos

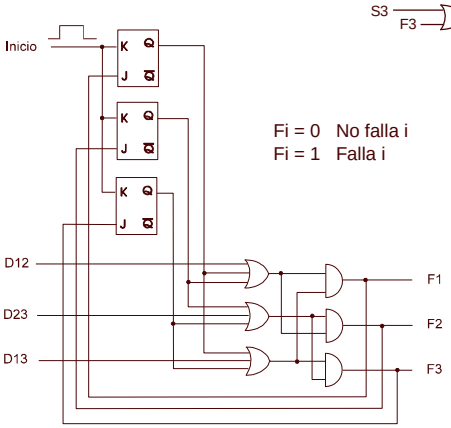
Comparador



Dij: 0 i y j en acuerdo
1 i y j en desacuerdo



Colector

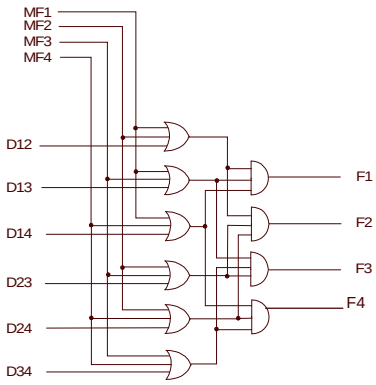


Fi = 0 No falla i
Fi = 1 Falla i

Detector

FTF

Detector para 4 módulos

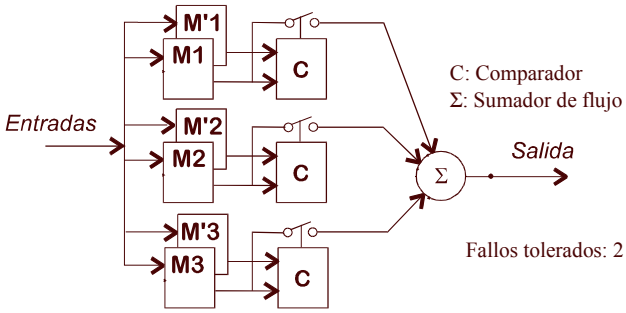


Dij: Entradas del Comparador
Fi: Salidas
MFi: Memoria de Fi

Se toleran 2 fallos
Se detecta el 3er fallo
N módulos: Se toleran N-2 fallos

FTF

- Arquitectura Triple - Duplex



C: Comparador
Σ: Sumador de flujo

Fallos tolerados: 2

• Esta arquitectura puede extenderse a N módulos:
Fallos tolerados con *N* módulos: *N-1*

FTF

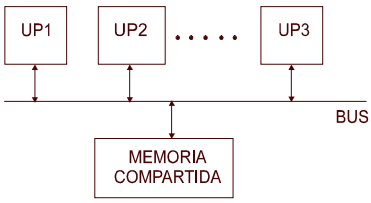
4.4 Recuperación hacia delante basada en pdr's

- Este método es apropiado en sistemas que sufren fallos transitorios. En ellos se almacenan puntos de recuperación (pdr's) o *checkpoints*.
- Se tiene un sistema dúplex que establece puntos de recuperación (pdr) y los compara para detectar errores. Para averiguar el módulo averiado se utiliza una técnica llamada RFCS (Roll Forward Checkpointing Scheme)
  - Se activa un repuesto que (en concurrencia con los otros dos módulos) repite la operación realizada
  - Cuando se detecta el módulo de proceso averiado, su estado se hace consistente con el bueno
- La secuencia de eventos que tiene lugar es la siguiente:

  (t1) Se ejecuta una copia del proceso en el repuesto con el pdr previo a la detección del error
  (t2) Se comprueba el pdr del repuesto con el pdr de las tareas A y B establecido en t1
  (t3) El módulo averiado continua con el pdr del bueno
  También se compara el pdr del módulo bueno en t2 con el del repuesto en t3 para comprobar que el módulo bueno no ha fallado en el intervalo de tiempo que va de t1 a t2.
- Existen otras técnicas optimistas y pesimistas basadas en la anterior cuando se emplean módulos autocomprobantes.

FTF

5. Reconfiguración en multiprocesadores



- Estrategias
  - Prestaciones constantes: Módulos de reserva
  - Recursos constantes: Degradación
- Problemas:
  - Diagnosis de fallos: rápido y fiable para evitar la contaminación
  - Interconexión: rápida y TaF
  - Recuperación de tareas: Mecanismos de comunicación y sincronización

FTF

- El componente averiado debe ser desconectado del sistema y éste se debe reconfigurar.
- Aproximaciones:
  - Redes de interconexión: En la reconfiguración se busca obtener la misma topología
    - Un hipercubo de grado N se reconfigura en otro de grado M, con M < N.
    - Problemas: La degradación es considerable llegando a ser hasta del 100 %
  - Rutado inteligente: Los mensajes utilizan caminos alternativos y se utiliza la degradación funcional.
    - Apropiado para escenarios donde las aplicaciones no precisan de una topología concreta para ejecutarse.
  - Utilización de módulos de proceso y de enlaces de repuesto: Cuando ocurren fallos se restaura la configuración original
- Tipos de redes de interconexión
  - Buses, crossbars, multiestado, hipercubo, de Bruijn, mesh y árboles