

# CRC: Verificación de Redundancia Cíclica

Mario Marin  
Jose Luis Gallardo  
Daryl Hernandez

Universidad Tecnica Federico Santa Maria

*TEL211*

December 17, 2018

## 1 Introducción procesos cíclicos

- Rut
- Paridad
- Codewords

## 2 Hamming

- Distancia Hamming
- Códigos Hamming

## Definición:

Corresponde a un conjunto de **instrucciones** las cuales **resuelven un problema**, este conjunto de instrucciones se ejecutan de forma **iterativa** hasta solucionar el problema dado un **criterio de parada**.

## Criterio de parada:

Condición que al cumplirse termina el cómputo de un algoritmo.

## Pasos:

- 1 Multiplicar cada número del rut (de derecha a izquierda) con la siguiente secuencia 2,3,4,5,6,7 (la secuencia se reinicia al llegar al final)
- 2 Sumar todos los resultados de las multiplicaciones.
- 3 Divida el resultado de la suma por 11 (cálculo de módulo 11)
- 4 El resultado de la división es el número verificador.

# Código verificador rut ejemplo

**rut: 30.686.957-X**

$$7 * 2 = 14$$

$$5 * 3 = 15$$

$$9 * 4 = 36$$

$$6 * 5 = 30$$

$$8 * 6 = 48$$

$$6 * 7 = 42$$

$$0 * 2 = 0$$

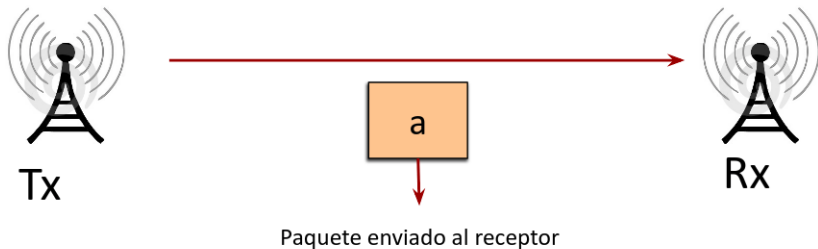
$$3 * 3 = 9$$

$$\text{Suma} = 14 + 15 + 36 + 30 + 48 + 42 + 0 + 9 = 194$$

Al dividir 194 con 11 obtenemos un resto de 7, el cual corresponde a el número verificador

Resultado:  
30.686.975-7

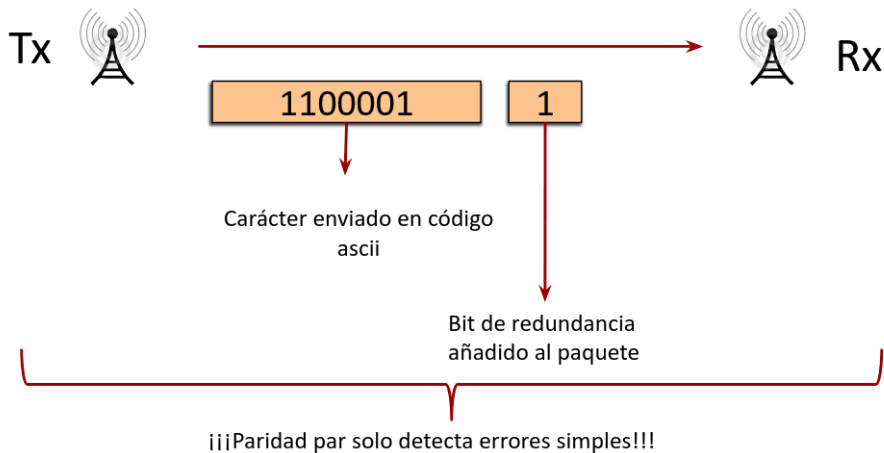
# Paridad Par



Tx: Transmisor

Rx: Receptor

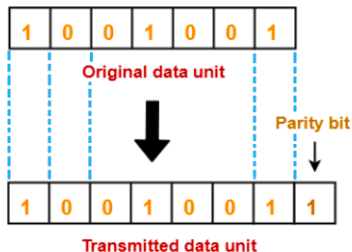
# Paridad Par



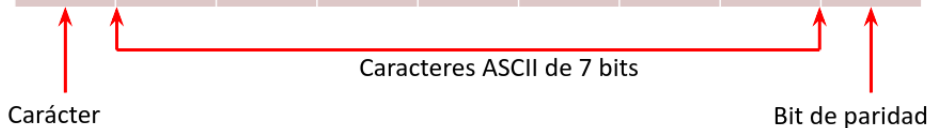


# Paridad Par

- ① Sumar todos los bits.
- ② Si la suma de bits tiene como resultado:
  - Número par: El bit de paridad es 0.
  - Número impar: El bit de paridad es 1.



	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	P
a	1	1	0	0	0	0	1	1
b	1	1	0	0	0	1	0	1
c	1	1	0	0	0	1	1	0
z	1	1	1	1	0	1	0	1
A	1	0	0	0	0	0	1	0



# Probabilidad de error de un paquete

Suponiendo que los errores son independientes entre si.

$$P(P_{err}) = 1 - \binom{n}{0} (1 - p_b)^n$$

$P(P_{err})$ : Probabilidad error en paquete.

$n$ : Número de bits del paquete.

$P_b$ : Probabilidad de error en bit.

# Probabilidad de detección de error paridad par

**Ejemplo:** Tenemos un mensaje a transmitir de 3 bits con paridad par.

Mensaje: 011

Paridad: 0

En la siguiente tabla veremos qué ocurre si el **receptor** del mensaje recibe diferentes mensajes pero el bit de paridad correcto, permitiendo ver de forma intuitiva cuál es la probabilidad de detección del bit de paridad.

OBS: Esta marcado en verde el mensaje que se quiere enviar.

# Probabilidad de detección de error paridad par

Recordar que el mensaje original es 011

Bits	Paridad	Nº De errores	Detección de error
000	0	0	No
001	0	1	Si
010	0	1	Si
011	0	2	No
100	0	1	Si
101	0	2	No
110	0	2	No
111	0	3	Si

Observar que existe una gran cantidad de falsos negativos, lo cual baja la con fiabilidad del algoritmo.

# Códigos y Codewords

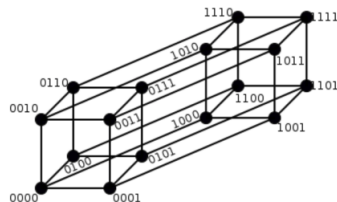
Un código corresponde a un algoritmo que genera codewords.

Un “codeword” corresponde a una secuencia de bits que cumple con algún Código.

Por ejemplo utilizando bit de paridad tenemos:

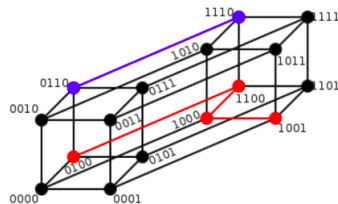
1010-0 Codeword pertenece al código.

1000-0 Codeword no pertenece al código.



# Distancia mínima de Hamming

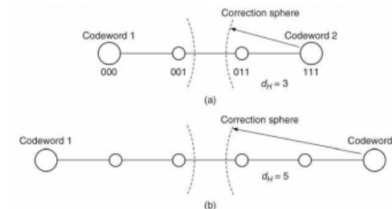
Mide la cantidad mínima de sustituciones que se requiere para pasar de un codeword a otro.



# Distancia mínima de Hamming

Si tenemos un código el cual su distancia mínima de Hamming entre todos sus codewords es “ $d$ ” entonces:

- Puede detectar “ $d - 1$ ” errores.
- Puede corregir “ $\lfloor d/2 \rfloor$ ” errores.





# Códigos Hamming

- Creados en los años 40 por Richard Hamming, quien trabajaba para Bell Labs.
- Basado en bits de paridad.
- Tiene una distancia de mínima de Hamming de 3
- Puede ser generado de manera dinámica (cíclica)



# Códigos Hamming cíclicos

Modo de operación (transmisión):

- 1 Calcular la paridad según la tabla adjunta.
- 2 Transmitir el mensaje como se indica en la tabla.

Bit position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits	p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	x		x		x		x		x		x		x		x		x		x	
	p2		x	x			x	x			x	x			x	x			x	x	
	p4				x	x	x	x					x	x	x	x					x
	p8								x	x	x	x	x	x	x						
	p16																x	x	x	x	x

# Códigos Hamming cíclicos

Modo de operación (verificación):

- 1 Calcular cada bit de paridad siguiendo la tabla.
- 2 Si el resultado son solo "0" entonces se recibió bien el mensaje, de lo contrario los bit de paridad indican cual es la posición del bit erróneo.

Bit position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits	p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	X		X		X		X		X		X		X		X		X		X	
	p2		X	X			X	X			X	X			X	X			X	X	
	p4				X	X	X	X					X	X	X	X					X
	p8								X	X	X	X	X	X	X						
	p16																X	X	X	X	X

# Probabilidad de error de un paquete

Suponiendo que los errores son independientes entre sí y con corrección de 1 error.

$$P(P_{err}) = 1 - \left( \binom{n}{0} (1 - p_b)^n + \binom{n}{1} (1 - p_b)^{n-1} p_b \right)$$

$P(P_{err})$ : Probabilidad error en paquete.

$n$ : Número de bits del paquete.

$P_b$ : Probabilidad de error en bit.

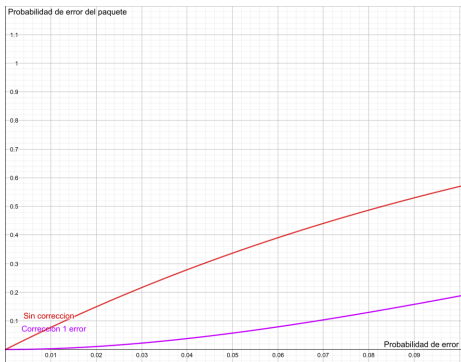


Figure: Paquete de 8 bits

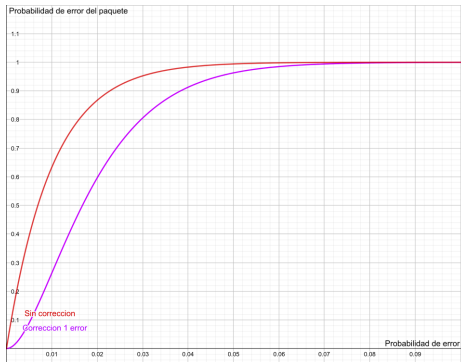


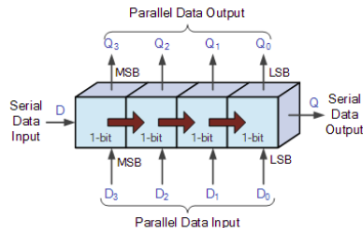
Figure: Paquete de 100 bits

# Códigos Hamming

Parity bits	Total bits	Data bits	Name	Rate
2	3	1	Hamming(3,1) (Triple repetition code)	$1/3 \approx 0.333$
3	7	4	Hamming(7,4)	$4/7 \approx 0.571$
4	15	11	Hamming(15,11)	$11/15 \approx 0.733$
5	31	26	Hamming(31,26)	$26/31 \approx 0.839$
6	63	57	Hamming(63,57)	$57/63 \approx 0.905$
7	127	120	Hamming(127,120)	$120/127 \approx 0.945$
8	255	247	Hamming(255,247)	$247/255 \approx 0.969$
...				
$m$	$n = 2^m - 1$	$k = 2^m - m - 1$	Hamming( $2^m - 1, 2^m - m - 1$ )	$(2^m - m - 1)/(2^m - 1)$

# ¿Qué?

- **CRC:** Cyclic Redundancy Check
- Verifican **integridad** de la información
- **No** es un método de cifrado
- Es realizado mediante **operaciones binarias** (XOR) y la salida siempre tiene el mismo tamaño
- Altamente utilizado en redes de computadores



# Un poco de historia ...

- Creado en 1961 por William Wesley Peterson
- Detectar errores en redes de comunicaciones
- Se llaman así porque el valor del check es redundante (expande el mensaje sin añadir información)
- “Si llega un bit erróneo, hay una alta probabilidad de que los adyacentes estén erróneos”.





- Nacimiento: 22 de Abril 1924 - Muskegon, Michigan, USA
- Ph.D., University of Michigan, 1954
- Áreas de interés:
  - Codigos de detección de errores
  - Teoría de detección de señales
  - Lenguajes de programación
  - Programación de sistemas
  - Conectividad y Redes de PC
- Premios y medallas:
  - Claude Shannon Award (1981)
  - IEEE Centennial Medal (1984)
  - Japan Prize (1999)



# ¿Cómo?

- Verifica integridad de la información: añade un código al final del bloque de datos enviado.
- Emisor envía la información y calcula el CRC. Receptor recibe la información y calcula (por su cuenta) el CRC, verificando que ambos códigos coincidan.
- ¿Coinciden? Existe una alta probabilidad de que el mensaje sea correcto (siempre existe la posibilidad de falsos positivos).
- ¿No coinciden? Corrupto.

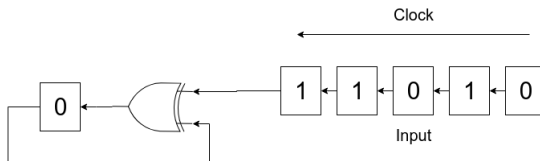
# ¿Cómo?

- Son simples de implementar y, debido a su estructura y diseño, no generan retardos y/o necesidad de procesamiento adicional
- Implementación: Shift Feedback Register (SFR) en conjunto con un “Generator Polynomial” (depende del largo que se quiera implementar)
- En caso de información correcta: el chequeo (en receptor) genera puros 0's binarios (lo cual representa que no hay errores)

# Ejemplo, bit de paridad par

Polinomio generador:  $x+1$

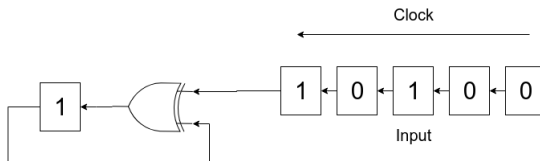
Mensaje: 11010



# Ejemplo, bit de paridad par

Polinomio generador:  $x+1$

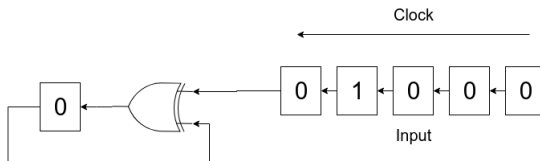
Mensaje: 11010



# Ejemplo, bit de paridad par

Polinomio generador:  $x+1$

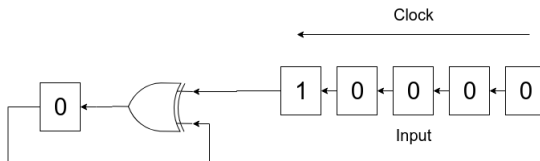
Mensaje: 11010



# Ejemplo, bit de paridad par

Polinomio generador:  $x+1$

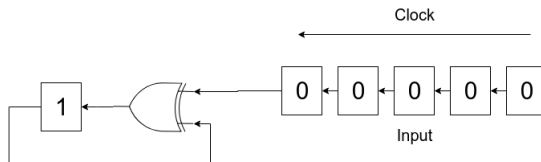
Mensaje: 11010



# Ejemplo, bit de paridad par

Polinomio generador:  $x+1$

Mensaje: 11010



El resultado final se encuentra en el registro al terminar de alimentar el mensaje al sistema

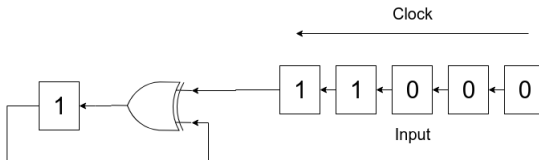
Mensaje a enviar: 11010+1



# Ejemplo, bit de paridad impar

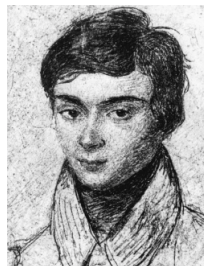
Polinomio generador:  $x+1$

Mensaje: 11010



# Évariste Galois

- Matemático Francés nació el 25 de octubre 1811.
- Murió a los 20 años de edad.
- Sus contribuciones a la matemática incluyen:
  - Fundador de la teoría de grupos.
  - Inventor de los campos de Galois.
  - Fracciones continuas.
  - Análisis de integrales Abelianas.



- Se opera con polinomios, pertenecientes a una variable real  $x$ , con coeficientes pertenecientes a un cuerpo finito  $GF(2)$
- $GF(2)$  = Galois Field of 2 elements. Es el cuerpo finito más pequeño. Se compone de 0, 1. Posee los operadores suma (bitwise XOR) y multiplicación (bitwise AND)
- (\*) Bit string = bit array = machine words
- Se quiere obtener un divisor polinomial tal que garantice buenas propiedades para detectar errores efectivamente y minimizando la probabilidad de colisión de bits.
- Los coeficientes de estos están dados por cada dígito del bit string, siguiendo una estructura de anillo (matemático)

Se sigue el siguiente procedimiento (genérico para todos los CRC):

- 1 Escoger el tamaño de checksum (ej:  $C$  bits)
- 2 Añadir una cantidad de  $C$ -bits 0 al final del mensaje
- 3 Este nuevo mensaje es el dividendo
- 4 “Polynomial generator” corresponde a los primeros  $n$ -bits de este nuevo mensaje, es el divisor del mensaje. Tiene largo “ $C+1$ ”.
- 5 El checksum es el resto de la división. Tiene largo “ $C$ ”.

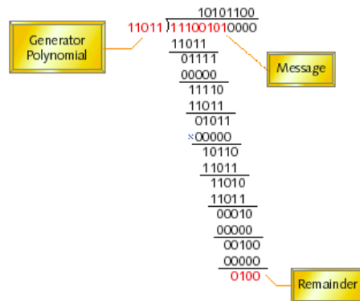


- ¿Cómo obtener este divisor polinomial, llamado “polynomial generator”?
- Si bien uno mismo podría calcular y/o obtener uno, existen valores ya obtenidos producto del trabajo de académicos e investigadores afines al área de análisis aritmético polinomial (álgebra lineal, combinatoria y teoría de conjuntos) tales que producen los mejores resultados.
- El análisis del origen de esto escapa al curso (DMAT).

	Checksum Width	Generator Polynomial
CRC-CCITT	16 bits	100010000000100001
CRC-16	16 bits	110000000000000101
CRC-32	32 bits	100000100110000010001110110110111

- **¿Por qué usar el resto y no el resultado de la división?**
- Porque el resto cambia inmediatamente si algún bit del mensaje difiere. Sumas, productos y cuocientes no comparten esta característica.

Ejemplo: obtener el checksum de 4 bits del mensaje 11100101, considerando como polinomio generador 11011.

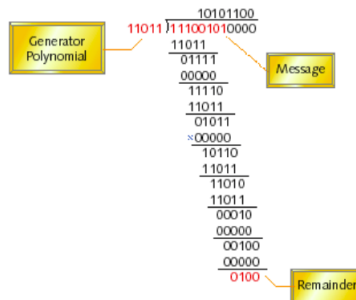


# Matemáticas de CRC

Ejemplo: obtener el checksum de 4 bits del mensaje 11100101, considerando como polinomio generador 11011.

Respuesta:

- Checksum:  $C = 0100$
- Paquete:  $M + C =$   
**111001010100**



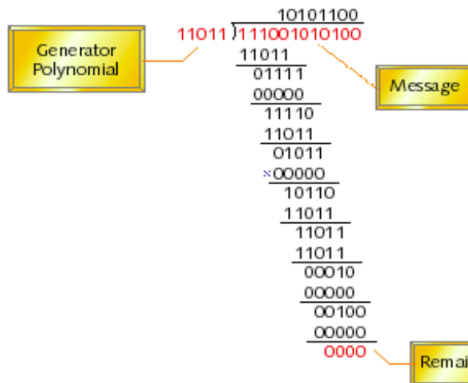


# Matemáticas de CRC

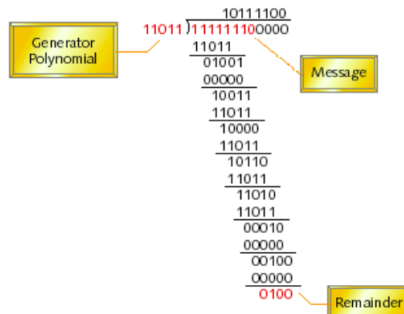
Ejemplo: Verificación de checksum del mensaje 111001010100, considerando como polinomio generador 11011.

Respuesta:

- Checksum:  $C = 0000$
- Como era de esperarse, el resultado es 0, pues el checksum añadido corresponde al resto de la división.



Ejemplo: obtener el checksum de 4 bits del mensaje 11111110, considerando como polinomio generador 11011.

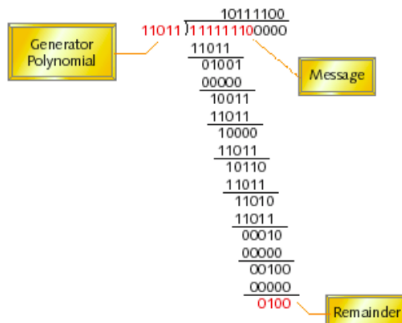


# Matemáticas de CRC

Ejemplo: obtener el checksum de 4 bits del mensaje 11111110, considerando como polinomio generador 11011.

Respuesta:

- Checksum:  $C = 0100$
- Paquete:  $M + C =$   
**111111100100**
- Podemos ver que con respecto al ejemplo anterior existe una colisión de códigos. (mismo CRC para dos palabras distintas)



- El largo del checksum  $C$  juega un rol importante en la detección de errores. La capacidad de detección de errores depende del largo de este.
- Largo  $C$  implica  $2^C$  posibles valores de checksum, de los cuales sólo uno es el válido. Existe la posibilidad de que dos o más mensajes tengan el mismo checksum, lo que implica que el receptor reciba un mensaje errado (creyendo que es correcto)

- Existe la posibilidad de que dos o más mensajes, de un cierto largo  $M$ , tengan el mismo valor de checksum (largo  $C$ ) significando que el receptor reciba uno de ellos como correcto (aún cuando está errado). Si uno de ellos es recepcionado, entonces su probabilidad de error queda dada por:

$$Error = 1/2^C$$

$$Correcto = 1 - Error$$

- Robustez en la detección depende netamente del largo del checksum.

# Matemáticas de CRC: Probabilidades

n-bits CRC	Detección de errores (correcto)
8	99,6094 %
16	99,9985
32	99,9999

Capaz de detectar mensajes con error en:

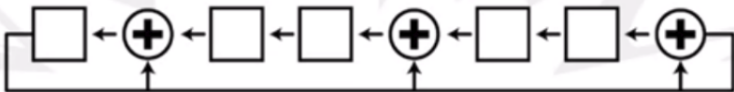
- 1 bit (bit de paridad)
- 2 bits (sin importar orden)
- n-bits, siendo n impar
- ráfaga de m-bits, siendo m el tamaño del checksum

# **Demostración con Mathematica!**



# Ejemplo #1

- Se quiere enviar la cadena: **1010011**
- Se escoge el largo que tendrá el procesado de nuestra cadena. Se escogió como largo **5**. Por ende, se deben añadir cinco 0's al final de la cadena resultando: **101001100000**

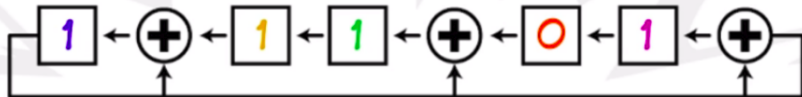


## Ejemplo #2

- **101001100000**
- Para empezar, se rellena el registro con sólo 0's. Luego, se realizan las operaciones XOR para cada bit. Una vez realizado, se realiza la operación SHIFT (desplazando los bits un espacio hacia adelante). En particular, al final de la trama entrará el primer bit de la cadena (1010011) pero luego de realizar XOR con el bit obtenido a la salida.
- Se realizan los XOR correspondientes y entra el segundo bit de la cadena, repitiendo lo anterior hasta terminar con la cadena completa (llámese completa a la que incluye los 0's al final)

# FALTA DIAPO 47

# Resultado Ejemplo



a transmitir: **101001111101**

Cada

- $n$ -bit CRC =  $n$ -bit CRC check value
- Dependen del tipo de CRC que se quiera implementar
- Para combinación  $n$ -bit, existe una función polinomial que la implementa (depende de lo que se quiera hacer o transmitir)
- La función polinomial representa el tamaño del residuo en la división





- Capa de Enlace: Protocolo Ethernet
- Bit de paridad: 1-bit CRC
- Ethernet: Implementa CRC-32 al final de datagrama
- Procesadores: Estándar CRC-32 en el instruction set SSE4.2 de la arquitectura x86-64.
- Discos Duros: Chequeo de sectores dañados/errados

<https://crccalc.com/>.

# Gracias por su atención



# References

-  [Reinaldo Vallejos](#)  
Anexo 1 – Códigos (libro probabilidades)
-  [Grote Hahn, Walter](#)  
ELO-346 – Clases: "Codificación de Canal"
-  [Jorge Flores – Yanara Velasquez – Camilo Vera](#)  
TEL-211 - Presentación: "Detección de Errores"
-  [Michael Barr](#)  
CRC Series Part #2, Mathematics and Theory ([Link](#))