Módulo 12:

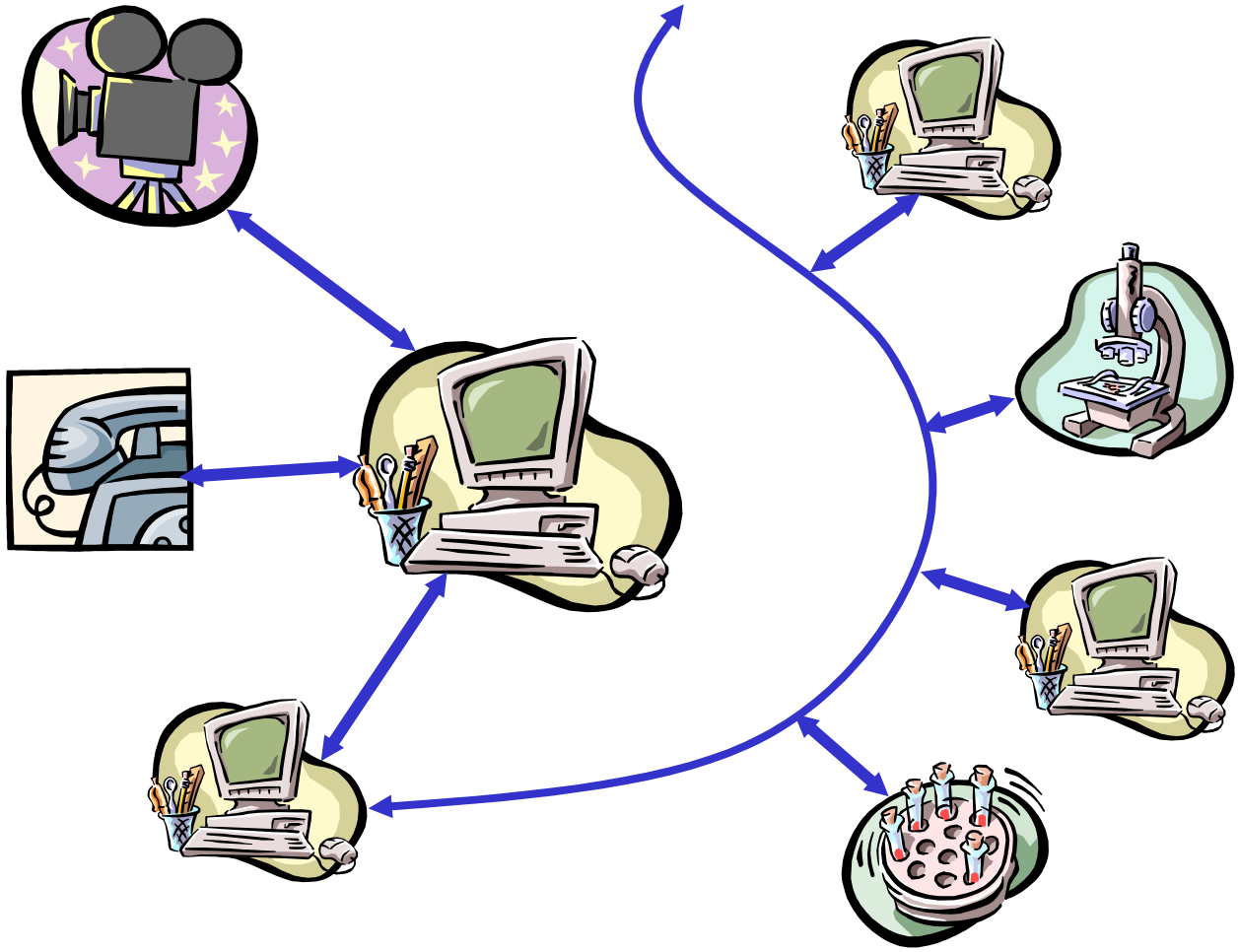
Protocolos de comunicación serial inter-chip / inter-sistema

Contenidos del módulo 12

- Comunicación asíncrona de a caracteres
- Protocolos LIN, SPI, I2C, CAN, USB, Ethernet, MIL-STD-1553B, SpaceWire.
- Control distribuido, control asignado a agentes, control asignado a mensajes.
- Las “violaciones de código” para tareas de señalización y sincronización de datos y tramas.
- Topologías Master/Slave fijas.
- Topologías multi-master.
- Caminos de comunicación MS/SM y SS.
- Mantenimiento de información temporal en sistemas distribuidos (Ethernet 1588)
- Control de errores en canales de comunicaciones

Interfases serie

- UART (RS232)
- SPI
- I2C
- LIN
- CAN
- USB
- ETHERNET

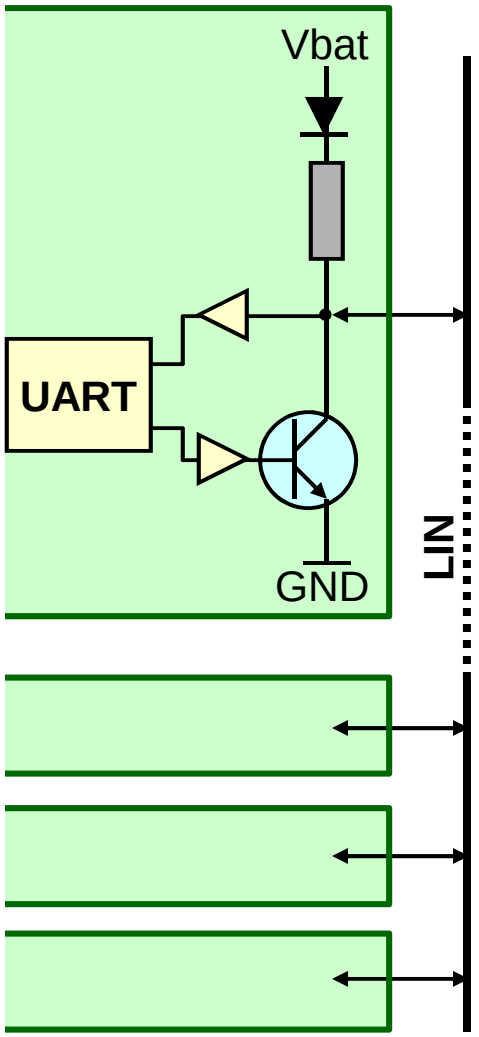


La interfase serie asincrónica (UART)

- Permite una conexión unidireccional a través de un par de cables
- La modulación es bi-nivel: ESPACIO (presente en el estado pasivo y durante parte de una transmisión) y MARCA (sólo presente durante la transmisión de un dato)
- La unidad de información (PAYLOAD) es un campo de 7 u 8 bits (el LSB se suele transmitir primero) a los que se agregan:
  - un bit de arranque (START) en MARCA
  - uno, uno y medio o dos bits de parada (STOP) en ESPACIO
  - eventuales bits adicionales (ej: PARIDAD) entre el campo de datos y STOP.
- La máxima duración continua en MARCA es 10 tiempos de bit, para un dato de 8 bits en 0x00 y paridad PAR. Una duración superior puede usarse para indicar una situación excepcional (violación de código) denominada BREAK.
- Requiere de una tasa de datos acordada. Como la sincronización se dispara en la transición de START, el máximo error tolerable de frecuencia corresponde a un corrimiento de ½ bit en 10 bits, es decir un 5%.
- La señalización física puede ser S.Ended (RS232: +/-12V), Diferencial (RS422, +/-5Vdiff)
- Agregando un estado de alta impedancia, el par de hilos puede usarse para realizar un enlace bidireccional (RS485), donde las colisiones se evitan por protocolo.

La interfase serie LIN

- **LIN** (por ***Local Interconnect Network***) ha sido desarrollada por AUDI, BMW, Daimler/Chrysler, Motorola, Volcano, Volkswagen y Volvo para aplicaciones de baja velocidad y muy bajo costo
- Sus características más destacables son:
  - un único master y múltiples esclavos, bidireccional
  - la parte lógica es realizable con una UART estándar, pues la transmision se realiza de a paquetes de 10 bits compuestos por un Start-bit, ocho bits de datos y un Stop-bit.
  - transmisión física por un único cable en que todos los agentes se conectan a través de pull-ups (1k en el Master y de 30 k en los Slaves) y salida open-collector a una tensión de 8V a 18V
  - velocidad de transmisión es de hasta 20 kbps (3 velocidades son recomendadas: 2400bps, 9600bps y 19200 bps)
  - con un método de autosincronización que no requiere relojes precisos en los esclavos
  - tiempos de latencia de transmisión acotados, y capacidad de detección de errores

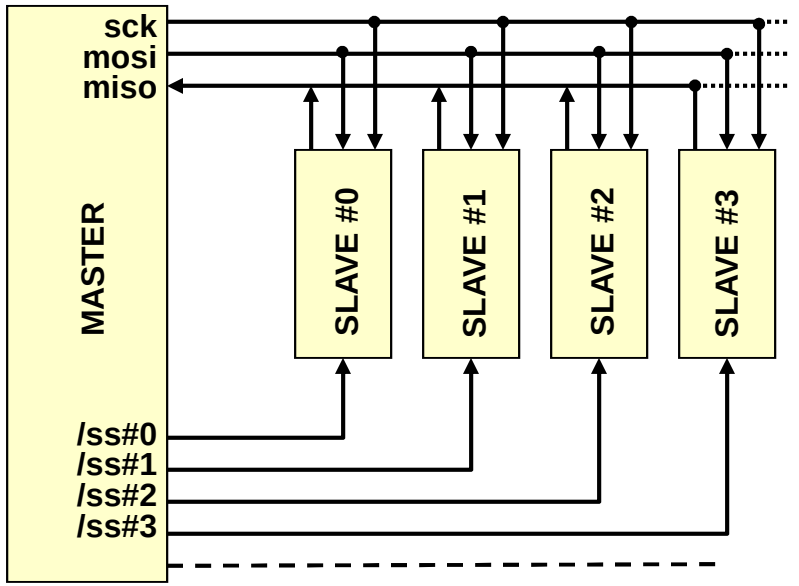


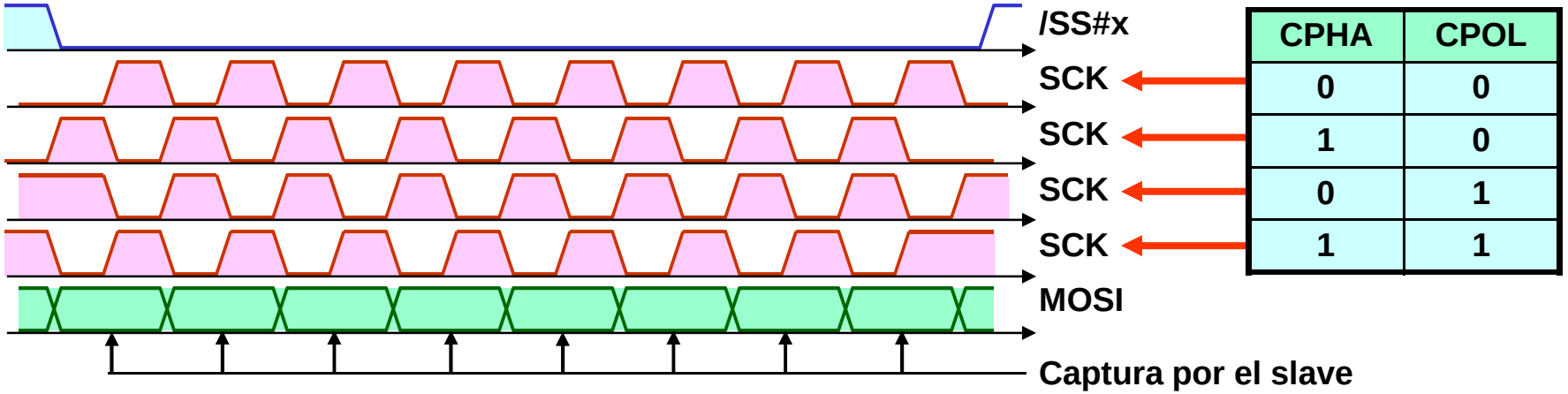
Cosas interesantes de la interfase serie LIN

- La parte lógica es realizable con una UART estándar;
  - Permite minimizar costos, es el periférico serial más habitual y difundido.
- Transmisión física bidireccional por un único cable dedicado:
  - Minimiza el costo de implementación de la red
- Todos los agentes se conectan a través de pull-ups (1k en el Master y de 30 k en los Slaves) y salida open-collector a una tensión de 8V a 18V
  - Con sólo un transceiver se adapta un uC estándar a LIN
- Un solo master y múltiples esclavos, direccionables por el mensaje del master
  - Simplicidad del software de los esclavos
- Método de autosincronización:
  - El master trasmite 0101010101111 al incio de cada frame, lo que facilita la recuperación de reloj por el esclavo sin necesidad de relojes precisos. El uso de BREAK para sincronizar tramas es una solución simple
- Tiempos de latencia acotados, y capacidad de detección de errores
  - Aptos para aplicaciones simples de control y supervisión

La interfase serie SPI

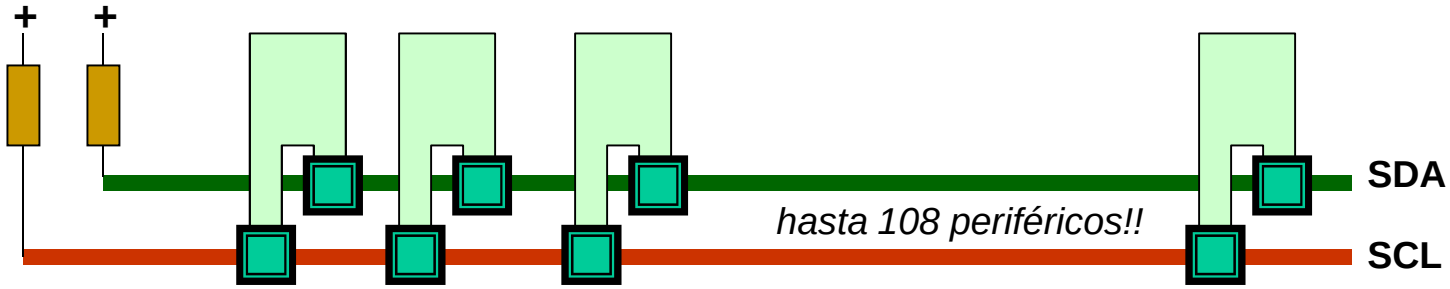
- La interfase *SPI (Serial Peripheral Interface)* provee un medio de comunicación bidireccional Full-Duplex entre un MASTER y múltiples dispositivos esclavos
- Las transferencias se realizan en paquetes de 8 bits, con una frecuencia definida por el Master a través de la señal SCK
- Mientras el Master envía datos por la línea MOSI, el esclavo "x" (seleccionado por SLAVE SELECT, o /SS#x) envía datos al Master por la línea MISO
- A favor: Hardware simple
- En contra: muchos esclavos requieren muchas /SSx





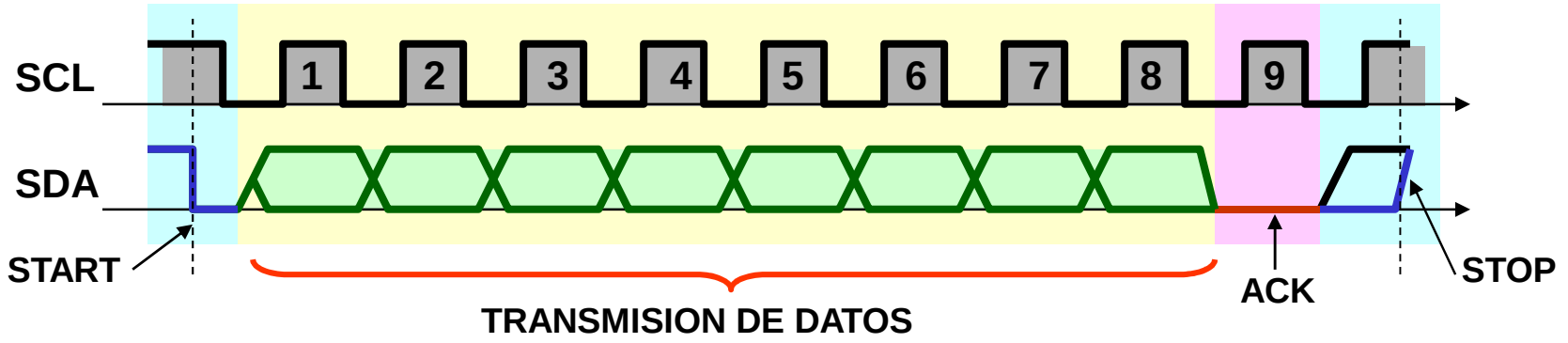
CPHA	CPOL
0	0
1	0
0	1
1	1

La interfase serie I2C (o Inter-IC)



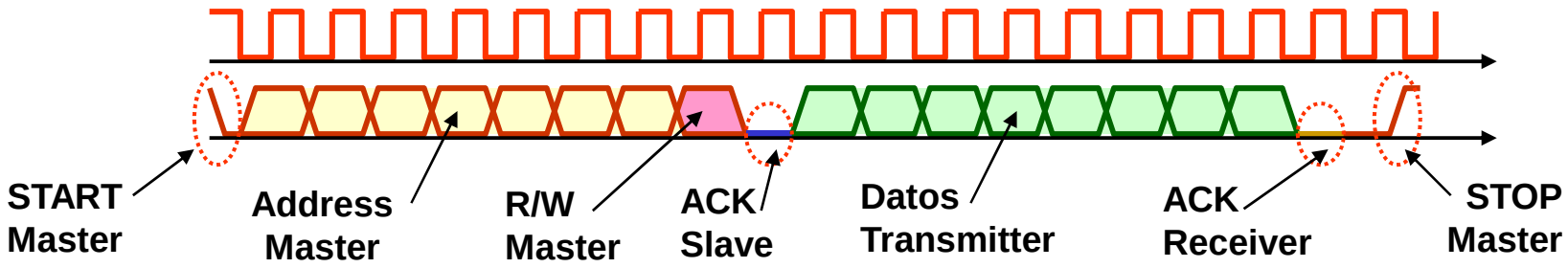
- Desarrollada por Philips/NXP, emplea sólo dos líneas bidireccionales de conexión con un pullup, comunes a todos los agentes, a las que cada uno se conecta mediante dos patas de I/O con salida de colector abierto: ***serial data (SDA)*** y ***serial clock (SCL)***.
- Cada línea tiene un estado pasivo (‘1’, cuando un pullup fuerza la tensión de la línea) y un estado activo (‘0’, cuando una salida fuerza la línea a tierra).
- La comunicación es serial, de a bytes, a velocidades que pueden ir desde 400 Kbits/s hasta 3,4 Mbits/s
- I^2C define 4 posibles tipos de agente en el bus:
  - ***Master***: quien inicia una transacción, genera las señales de reloj, y termina la transacción.
  - ***Slave***: dispositivo seleccionado por el *master*
  - ***Transmitter***: quien envía datos al bus
  - ***Receiver***: quien recibe los datos del bus
- No existe comunicación entre Slaves. Existen además conceptos de Multimaster, Arbitration y Synchronization, para aplicaciones con más de un master.

Transacción I2C en un ambiente monomaster



- Durante una transacción I^2C, los datos presentes en la línea SDA sólo pueden cambiar mientras SCL está en '0', y su valor es copiado en los flancos positivos de SCL; la violación a esta regla (cambio de SDA mientras SCL está en '1') es usada para definir dos condiciones, llamadas **START** y **STOP**, que son generadas por el *master* para señalar el inicio y fin de cada transacción, de la siguiente manera:
  - Un **START** se indica mediante una cambio de '1' a '0' en SDA mientras SCL está alto.
  - Un **STOP** se indica mediante una cambio de '0' a '1' en SDA mientras SCL está alto.
- A partir del **START**, comienza la transferencia de datos, byte a byte. Para transferir cada byte, el *transmitter* va colocando los datos bit a bit , con una velocidad definida por la señal SCL generada por el *master*; al cabo de 8 bits el *transmitter* debe liberar SDA y un noveno ciclo de SCL debe ser empleado por el *receiver* para señalar su aceptación (ACK) forzando un '0' en SDA.

El byte de control (direcciones y read/write) y los bytes sucesivos



- En cada transacción, luego de la condición START, el *master* envía un byte de control.
- Este byte se compone de 7 bits que indican la dirección del *slave* elegido, más un bit R/W que indica si los bytes siguientes irán del *master* al *slave* (transacción de escritura, R/W=0, el *master* es el *transmitter* y el *slave* es el *receiver*) o del slave al master (transacción de lectura, R/W=1, el *master* es el *receiver* y el *slave* el *transmitter*). Al recibir el byte de control el slave debe dar su ACK.
- A partir de ese momento, cada vez que el *transmitter* manda 8 bits el *receiver* debe dar su ACK y a partir de allí durante el/los bytes sucesivos hasta llegar al STOP el *trasmitter* forzará los datos en SDA y el *receiver* deberá dar los ACK
- Si bien los 7 bits de direcciones permitirían direccionar hasta 128 dispositivos distintos, las direcciones 0000xxx, 1111xxx y *11101xx* han sido reservadas para casos especiales, quedando disponibles 108 direcciones. Para definir cuál es su dirección, cada *slave* suele tener ciertos bits de esa dirección prefijados, y poseer además algunas patas que son puestas a '0' o a '1' para definir los bits restantes.

Verificar si es así

La interfase serie CAN: generalidades

- **CAN** (por ***Controller Area Network***) es un bus serial con capacidad multi-master. Las normas ISO 11898 e ISO11519-2 cubren las 2 capas inferiores del modelo ISO/OSI.
- Las versiones existentes son **CAN 1.2**, **CAN 2.0A**, **CAN 2.0B Passive y Active**
- Inicialmente desarrollado para automóviles, hoy tiene múltiples aplicaciones, mayormente en el campo industrial
- La cantidad de nodos sólo está limitada por parámetros eléctricos
- Se usa en modo *single-wire* (1 hilo + GND) y también en diferencial (2 hilos + GND)
- Posee un robusto manejo de errores y mecanismos de manejo de prioridades
- Los nodos no tienen direcciones propias por lo que los mensajes no poseen indicación de origen o destino, sólo un ID asignada por el sistema
- Tipicamente, cada tipo de evento o comando tiene un único *message ID*, y sus parámetros forman parte del campo de datos
- El *message ID* es de 11 bits en CAN 2.0A y de 11 a 29 bits en CAN 2.0B (un nodo en CAN 2.0B passive tolera IDs de 29 bits pero sólo puede usar los de 11 bits

La interfase serie CAN: el arbitraje

- Necesario para identificar una eventual situación de colisión
- Se basa en la existencia de estados dominantes (0) y recesivos (1).
- Si un nodo envía el estado dominante al bus ese será el estado del bus. Si envía un estado recesivo chequea que efectivamente ese sea el estado del bus.
- El arbitraje se realiza al transmitir el campo de arbitraje: si un nodo está enviando un bit recesivo y observa que el bus queda en estado dominante ello implica que otro nodo está enviando un mensaje de mayor prioridad por lo que se desconecta. Por ejemplo:

Nodo A	**101--------**
Nodo B	**10010001100**
Nodo C	**100101-----**
Estado del bus	**10010001100**

collision detect!

- El nodo B gana el arbitraje y es el único que completa el envío de su ID
- Esta técnica es llamada ***CSMA/CD+AMP*** (***Carrier Sense Multiple Access with Collision Detection and Arbitration on Message Priority***), y causa que cuando más de un nodo intenta transmitir, el mensaje con ID de mayor prioridad (menor valor númerico) se imponga, sin que la colisión corrompa al mensaje en curso

La interfase serie CAN: campos de un mensaje

- Inicio (***Start of frame***): contiene un START bit que sirve para sincronización y como indicador de salida del modo inactivo (IDLE)
- ***Arbitraje***: es un campo de 12 o 32 bits que además del significado del mensaje también indica su prioridad. En CAN 2.0A hay 2032 ID posibles, y en CAN 2.0B hay 536,870,912 diferentes Ids. En el caso de los ID de 11 bits se compone de:
  - Identificador (***Message ID***): 11 bits
  - ***RTR (Remote Transmission Request):*** 1 bit para pedir a un transmisor remoto que envíe datos
- ***Control***: es un campo de 6 bits formado por:
  - ***IDE (Identifier Extension):*** 1 bit que vale 0 para ID de 11 bits y 1 para ID de 29 bits
  - ***r0***: bit reservado
  - ***DLC (Data Length Code):*** 4 bits con valor 0 a 8 que indica la cantidad de bytes de datos
- ***Datos*** (0 a 8 bytes)
- ***CRC***: campo de 16 bits con 15 bits de control seguidos de uno de delimitación, recesivo
- ***ACK***: campo de 2 bits, el segundo recesivo de delimitación. Cada nodo que detecta el mensaje en el bus como correcto fuerza un 0 en el primero de estos bits
- ***EOF (End of Frame):*** 7 bits en '1'
- ***IFS (Inter Frame Space):*** sucesivos bits en '1' mientras no haya nuevo mensaje

La interfase serie CAN

- **Velocidad de transmisión**: La interfase permite enlaces a velocidades de hasta 1 Mbps en redes de hasta 40 mts de longitud. De desear mayor alcance la velocidad debe disminuirse, pudiendo llegarse a 500 mts a velocidades de 125 kbps y hasta 1 Km con velocidades de 50 kbps.
- ***Bit Stuffing***: para recuperar el reloj la interfase CAN usa el método llamado "***Bit Stuffing***", que consiste en agregar cada 5 bits sucesivos de igual valor un bit "extra" de polaridad opuesta, que es ignorado por el receptor. De este modo tanto el campo ***EOF*** con 7 bits en '1', como el ***IFS*** (un "1" permanente) constituyen "*violaciones de código*".

La interfase serie USB
Universal Serial Bus

La interfase ***Universal Serial Bus*** surgió como extensión a la arquitectura de las PCs, por el esfuerzo conjunto de Compaq, DEC, IBM, Intel, Microsoft, NEC y Northern Telecom

- La rev.1.0 (Enero 1996) definió un método de comunicación a velocidades bajas (hasta 100 kbps) y medias (hasta 12 Mbps).
- La rev.1.1 (Septiembre 1998) presentó dos definiciones del *Controller Interface*: UHCI (*Universal Host Controller Interface*) y OHCI (*Open Host Controller Interface*).
- La rev.2.0 (Abril 2000) aumentó la velocidad a 480Mbps y definió un nuevo EHCI (Enhanced Host Controller Interface), *backward compatible* con la rev.1.1.

Su motivación fué facilitar:

- la integración de periféricos a bajo costo en aplicaciones de comunicaciones (telefonía y otros recursos multimedia).
- el uso y conexionado, dando propiedades de Hot Plug and Play, a dispositivos como puertas serie/paralelas, teclados, mouses, joysticks.
- la fácil expansión del número de puertas disponibles para el agregado de dispositivos periféricos con necesidad de comunicación bidireccional.

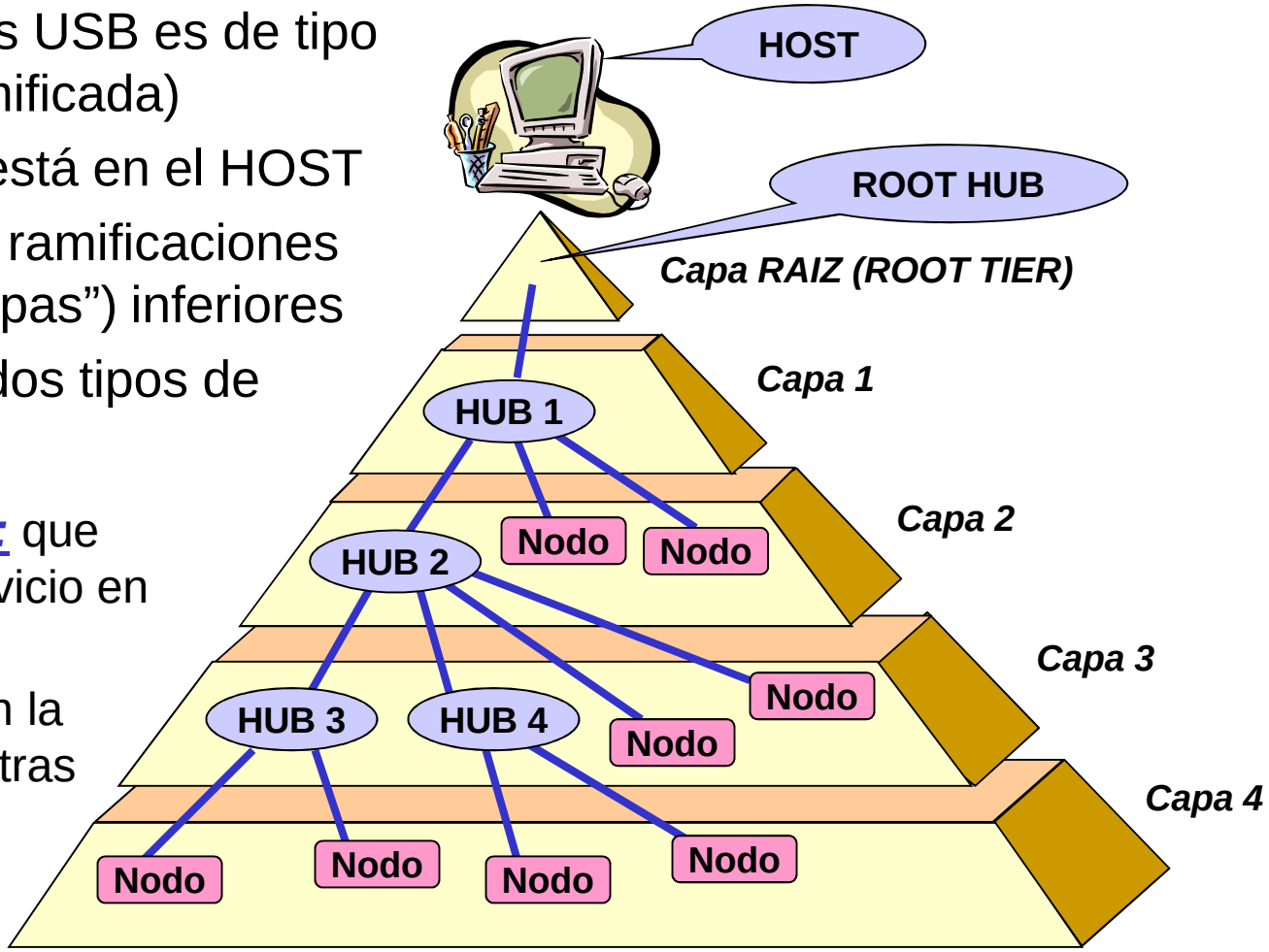
La interfase serie USB Características

Un sistema USB se describe mediante tres áreas de definición

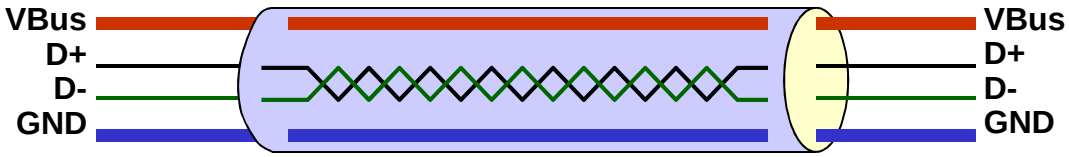
- **La interconexión USB:** es la forma en la que los dispositivos USB están conectados y cómo se comunican con el Host, lo que incluye:
  - La topología del bus (modelo de conexionado entre los dispositivos y el Host)
  - Las relaciones inter-capas del sistema
  - Los modelos de flujo de datos sobre el USB entre productores y consumidores
  - La administración del bus para permitir su uso compartido
- **El Host USB:** hay un único Host controller, cuya descripción combina hardware y software, y que a través de un Host Hub provee uno o más puntos de conexión
- **Los dispositivos USB:** se componen de:
  - Hubs: que proveen puntos adicionales de conexión
  - Funciones: que proveen servicios o agregan capacidades al sistema (mouse, teclado, joystick, printer, scanner, cámara digital, etc..)
  - La interfase permite soportar hasta 127 dispositivos físicos distintos, y cada uno de ellos con uno o más puntos finales (*endpoints*) con los que se establecen transacciones.

La interfase serie USB
La topología

- La topología del bus USB es de tipo árbol (o estrella ramificada)
- El centro (o “raiz”) está en el HOST
- Y de allí se derivan ramificaciones hacia niveles (o “capas”) inferiores
- En cada capa hay dos tipos de dispositivos:
  - ***Funciones (nodos):*** que brindan un dado servicio en esa capa
  - ***Hubs:*** que extienden la conectividad hacia otras capas



La interfase serie USB
La capa física: aspectos eléctricos



USB transfiere datos+energía con un cable de 4 hilos y por conexiones punto a punto:

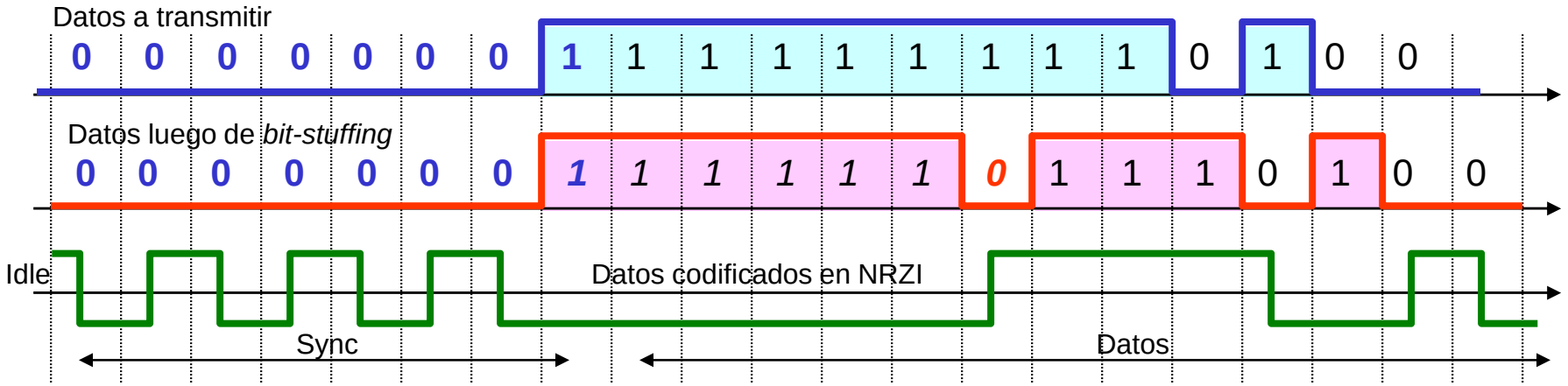
- dos hilos (D+ y D-) son usados para datos, donde:
  - la transmisión es bidireccional, de modo balanceado (o diferencial)
  - los receptores deben tener una sensibilidad de al menos 200mV y adecuada capacidad de rechazo de señales de modo comun
  - los hilos son trenzados (twisted), y su impedancia característica debe ser de 90 ohms
  - los dispositivos en sus extremos deben presentar una impedancia de terminación, que es también usada por el HUB para detectar la conexión o desconexión de los dispositivos
- otros dos hilos (VBus y GND) permiten suministrar energía a los dispositivos conectados:
  - deben ser de sección adecuada para asegurar una caída de voltaje dentro de la norma.
  - el suministro de energía es acotado y controlado por el sistema que puede desconectar (SUSPEND) o suministrar (RESUME) energía a cada dispositivo

Un cable tiene conectores distintos en cada extremo (*UpStream* y *DownStream*) lo que impide realizar conexiones tipo loop. Un dispositivo tiene al menos el conector *UpStream*.

La interfase serie USB
la capa física: formato de señales

La transmisión de datos a través del cable usa codificación NRZI con bit stuffing, donde:

- **NRZI:** método de codificación diferencial donde un '0' es representado mediante una transición (cambio de 1 a 0 o de 0 a 1) en tanto un '1' es representado mediante la ausencia de cambios (también usada en Fast Ethernet)
- **Bit stuffing:** inserción en el transmisor y remoción en el receptor de un '0' (una transición) cada seis '1' sucesivos (seis tiempos de bit sin cambios) de modo de garantizar la existencia de transiciones en la señal y la consiguiente recuperación de reloj. El uso de ***bit-stuffing*** permite transmitir los datos junto a la señal de reloj, y para ello se coloca un campo SYNC precediendo a cada paquete para que el receptor sincronice su reloj de recepción.



La interfase serie USB
Las transacciones

- El intercambio de datos en USB se realiza a través de transacciones
- Y cada transacción involucra la transmisión de hasta 3 paquetes
  - La transacción es iniciada por el Host Controller, que envía un paquete (***Token packet***) con el tipo y sentido de la transacción, la dirección (*address*) del dispositivo USB, y el componente de ese dispositivo (*endpoint number*) con quien se realizará la transacción. Todas las transacciones son entre el Host y un dispositivo, no existen transacciones entre dispositivos.
  - La fuente de datos envía un paquete de datos (***Data Packet***) o indica que no tiene nada para enviar
  - El destinatario en general responde con un ***Handshake Packet*** para indicar si la transacción fué exitosa. En ciertos casos el uso de ***NACK*** es empleado no para indicar error sino para el control de flujo
- El *canal virtual de datos* entre el Host y el endpoint de un dispositivo es llamado ***pipe***. Los *pipe* tipo "*message*" tienen una estructura definida, los de tipo "*stream*" no, y cada *pipe* tiene asociada un dado ancho de banda, tipo de servicio, y ciertas características (P.E": tamaño de buffers). Los *pipe* son creados en la configuración del dispositivo, y existe un tipo predefinido (*pipe_0*) que corresponde a la conexión con un dispositivo recién conectado a la red. Cada pipe puede ser uni o bi-direccional, y, por ejemplo, un único endpoint puede establecer dos pipes, uno HOST => endpoint y otro endpoint => HOST.

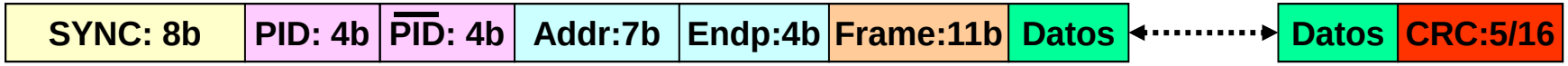
La interfase serie USB
La conexión de dispositivos

- Todos los dispositivos USB se conectan a la red en las puertas de los USB Hubs.
- Los HUBs detectan esta conexión y la informan en una palabra de estado que es consultada por el HOST, quien se entera así de que un dispositivo fue conectado o desconectado en esa puerta
- El HOST habilita entonces esa puerta y establece con ella un control pipe que emplea una dirección de default
- Mediante este pipe el HOST sabe si el dispositivo es un dispositivo funcional u otro HUB, le asigna una dirección única, y establece un nuevo pipe de control (pipe_0) empleando esta dirección.
  - Si es una función, la información del tipo de dispositivo en enviada al software del HOST para iniciar la configuración, cargar drivers de software, etc., o las que correspondan a su remoción.
  - Si es un HUB, el proceso se repite para cada una de las puertas de ese HUB que tenga conectado un dispositivo
- En este proceso se realiza la llamada BUS ENUMERATION, que es la asignación de direcciones. Como en USB los dispositivos pueden ser conectados y desconectados permanentemente, el proceso de enumeración se realiza permanentemente.

La interfase serie USB
Tipos de flujo de datos

- El flujo de datos en un pipe puede ser de 4 grandes tipos:
  - **De control**: usado para configurar un dispositivo al ser conectado, u otras funciones. Son transacciones que deben ser intrínsecamente seguras.
  - **De bloques de datos (*Bulk transactions*):** asociadas a grandes bloques de información con requerimientos de velocidad y latencia poco estrictos. Son transacciones que pueden llegar a tolerar la detección de errores (sin corregirlos, o con intentos limitados de corrección).
  - ***Interrupt transactions:*** son breves transacciones usadas para coordinar actividades que implican algún tipo de feedback
  - ***Isochronous transactions:*** para transferencia continua de cantidades importantes de información en tiempo real, es decir con requerimientos de velocidad y latencia (Por ejemplo, voz). Están asociadas a la pre-negociación de un dado ancho de banda entre el dispositivo y el HOST (*bandwidth allocation*), con el que los datos se crean, transmiten, y consumen, y deben tolerar ciertos régimen de errores.

La interfase serie USB los paquetes de datos



- **Sync Field**: 8 bits luego del estado IDLE, 7 ceros seguidos de un uno
- **Packet Identifier Field**: 4 bits que definen el tipo de paquete, seguidos por esos 4 bits negados.
  - Token: OUT (01h), IN (09h), SOF (05h), SETUP (0Dh)
  - Data: DATA0 (03h), DATA1 (0Bh)
  - Handshake: ACK (02h), NACK (0Ah), STALL (0Eh)
  - Special: PRE
- **Address Field**: 7 bits
- **Endpoint Field**: 4 bits
- **Frame Number field**: 11 bits con un número consecutivo
- **Data Field**: de 0 a 1023 bytes de datos
- **CRC Field**: 5 o 16 bits segun el tipo de paquete (definido por los PID)

Ethernet

Ethernet surgió como una interfase de conexión con las siguientes características:

- No existen maestros o esclavos o Hosts, todos los dispositivos tienen idéntica prioridad de acceso (***Multiple Access***) y un número único (***MAC number***)
- La conexión o desconexión de un dispositivo no debe afectar a los demás
- No hay mecanismos externos de control de flujo: se regula automáticamente por la detección de ocupación del canal (***Carrier sense***) y de colisiones (***Collision Detect***)
- No hay tiempos de latencia garantizados
- La ocupación del canal se hace en forma asincrónica, no usa esquemas temporizados (***slotted***), y la unidad de transmisión es un paquete (***packet***)
- Acepta la existencia de colisiones. Al detectarla un agente debe seguir una secuencia de desconexión
  - Cancelar la transmisión actual
  - Publicar la colisión (***Jamming***) y luego pasar a silencio
  - Definir el tiempo de espera para el reintento (***Binary Exponential Backoff***) en base a la historia local de colisiones previas

La interfase Ethernet

- Capa física originalmente basada en en uso de cable coaxil de 50 ohms para una topología tipo bus (10Base2, usando RG58), los problemas de reflexiones y atenuaciones llevaron que actualmente se use cable multipar trenzado (UTP Categoría 5) y topologías de tipo estrella, donde el nodo de la estrella (HUB) se encarga de distribuir la señal a los extremos.
- La señal inyectada al medio lo es a través de transformadores, por lo que se usa modulación Manchester para asegurar valor medio nulo (interfase física o PHY).
- Si bien la interfase original fue pensada para la transmisión hasta 10 Mbit por segundo, actualmente es común el empleo de redes de 100 Mbps y 1000 Mbps.
- Dada la estandarización de los encabezados el uso de este tipo de modulación permite detectar pares de cables invertidos

La interfase Ethernet 10 base-T

En el nivel primario de la capa física está un transformador de aislación, que a veces viene embebido dentro del mismo RJ-45

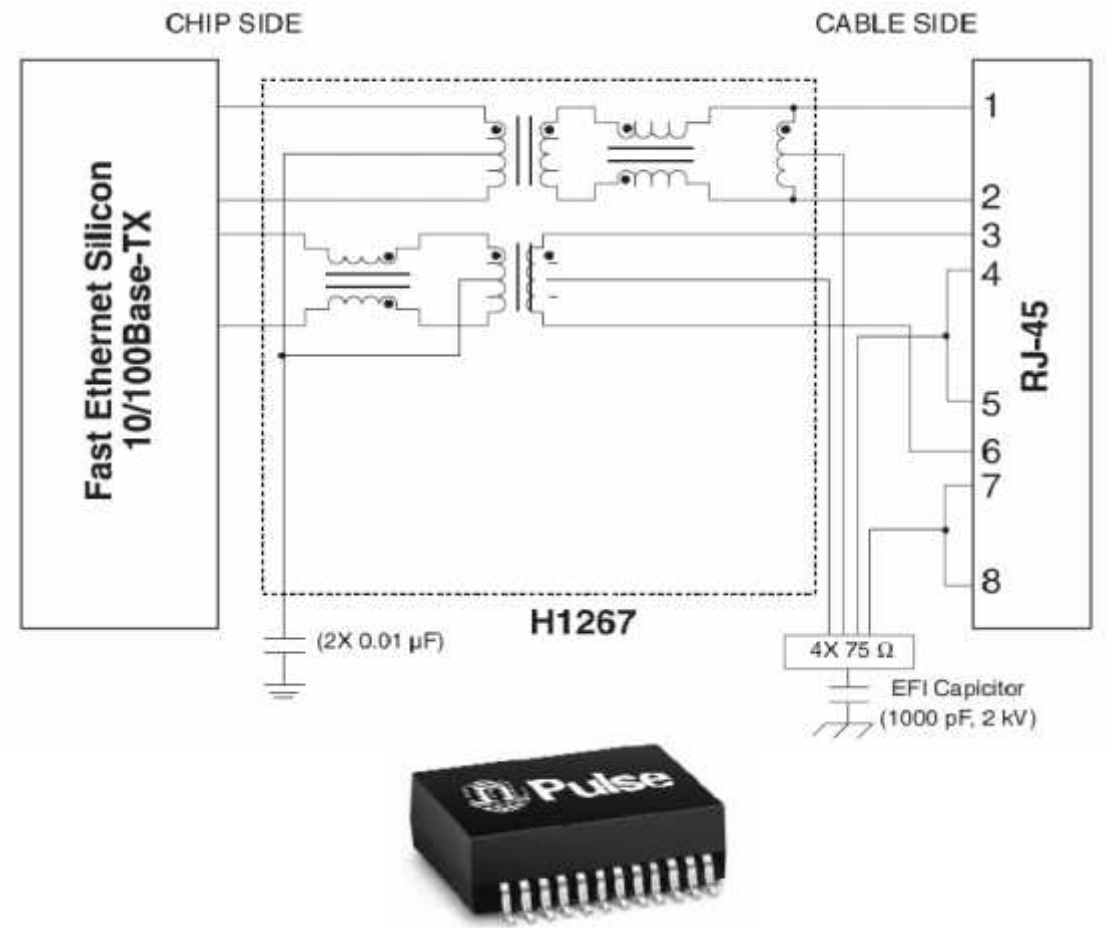
- ✓1: TX+
- ✓2: TX-

48V_out en Power Over Ethernet tipo A

- ✓3: RX+
- ✓6: RX-

48V_return en Power Over Ethernet tipo A

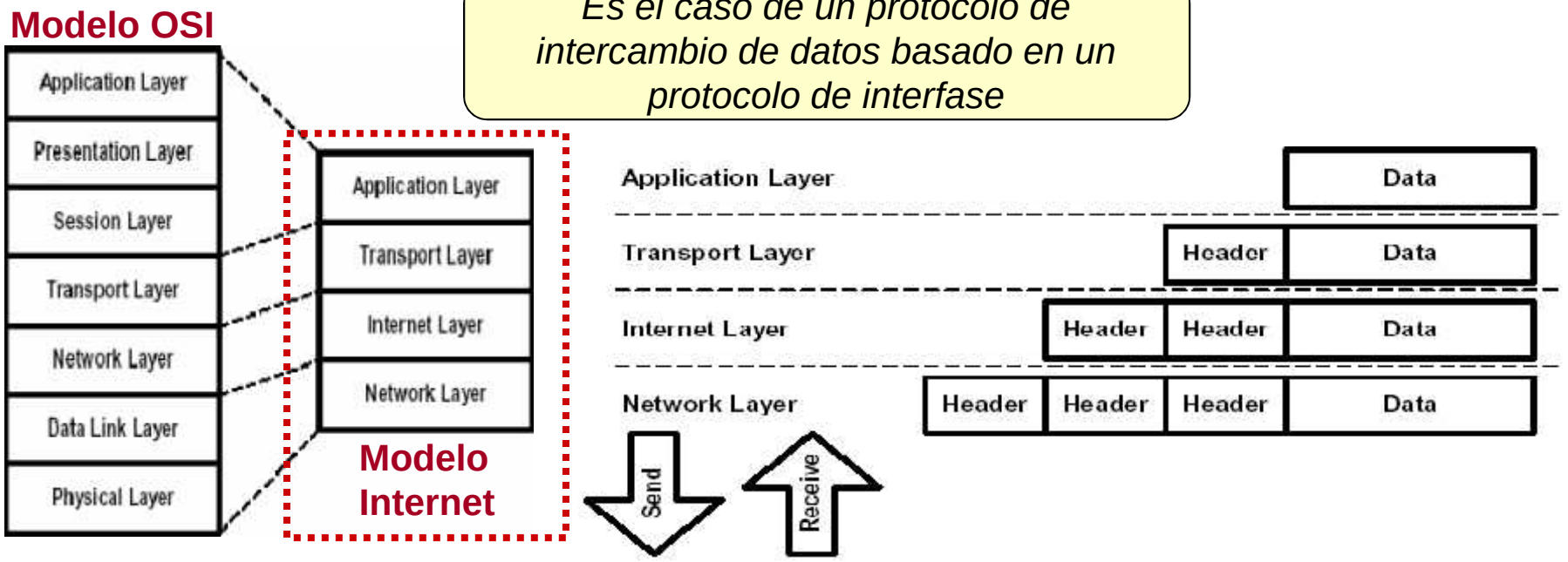
- ✓*4-5: 48V_out en Power Over Ethernet tipo B*
- ✓*7-8: 48V_return en Power Over Ethernet tipo B*



Internet: El modelo OSI y los distintos paquetes en internet

Layer Name	Function	Example
Application layer	Contains a lot of protocols defined by different applications to provide their services.	HTTP, telnet, e-mail (SMTP, POP)
Transport layer	Makes the communication between endpoints possible.	Transmission control protocol (TCP), user datagram protocol (UDP)
Internet layer	Delivery and routing of datagrams between Internet nodes.	Internet protocol (IP), Internet control message protocol (ICMP), address resolution protocol (ARP)
Network layer	Host-specific implementation of transmission of datagrams.	Ethernet (IEEE 802.3), point-to-point protocol (PPP), AX.25

Es el caso de un protocolo de intercambio de datos basado en un protocolo de interfase



Paquetes en la capa Network Layer

Un paquete de la capa de RED se forma con los siguientes campos:

- Un preámbulo de ‘1’ y ‘0’ alternados que posibilita la recuperación y sincronización del reloj del transmisor por los receptores
- Un byte para indicar el inicio de una trama llamado SOF (*Start Of Frame*)
- Dirección de destino del paquete: contiene el número MAC (MAC por Media Access Control) de 48 bits hacia el que se dirige el paquete. Este es un número único.
- Dirección de origen: número MAC de 48 bits de quien transmite el paquete
- Longitud del campo de datos o tipo del paquete.
- Datos: carga útil o Payload. Hasta un máximo de 1536 bytes en el caso de la norma IEEE 802.3, pueden agregarse bytes de relleno hasta alcanzar la longitud mínima de 64 bytes para garantizar la detección de colisiones,
- Campo de detección de errores (FCS, por *Frame Check Sequence*). La norma IEEE especifica el uso de un CRC de 32 bits con polinomio:

$$1 + X + X^2 + X^4 + X^5 + X^7 + X^8 + X^{10} + X^{11} + X^{12} + X^{16} + X^{22} + X^{23} + X^{26} + X^{32}$$

Internet Layer: IP Header

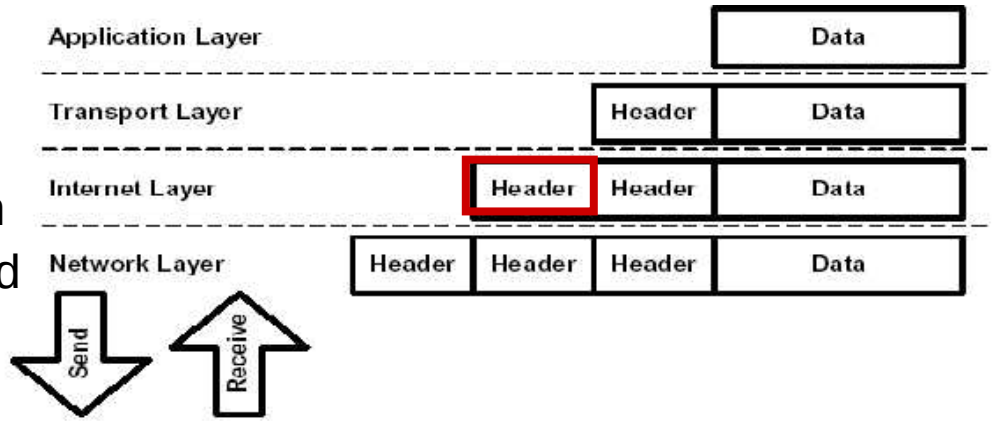
Este encabezado tiene 20 o más bytes, y se forma con los campos:

- ✓Version: versión de internet (IPv4 o IPv6)
- ✓IHL: largo del header (en palabras de 32 bits, 5 como mínimo)
- ✓Type of Service: calidad de servicio esperada
- ✓Total Length: largo del datagrama completo, con el encabezado, en bytes
- ✓Identification: un valor adicional agregado por el transmisor

Ethernet no, pero Internet sí, agrega QoS, fragmentación, tiempo de vida, etc

- ✓ Flags: modos de fragmentacion de un paquete en varios
- ✓ Fragment Offset: posicion del fragmento referida al inicio
- ✓ Time to Live: tiempo en segundos en que un paquete puede estar en la red

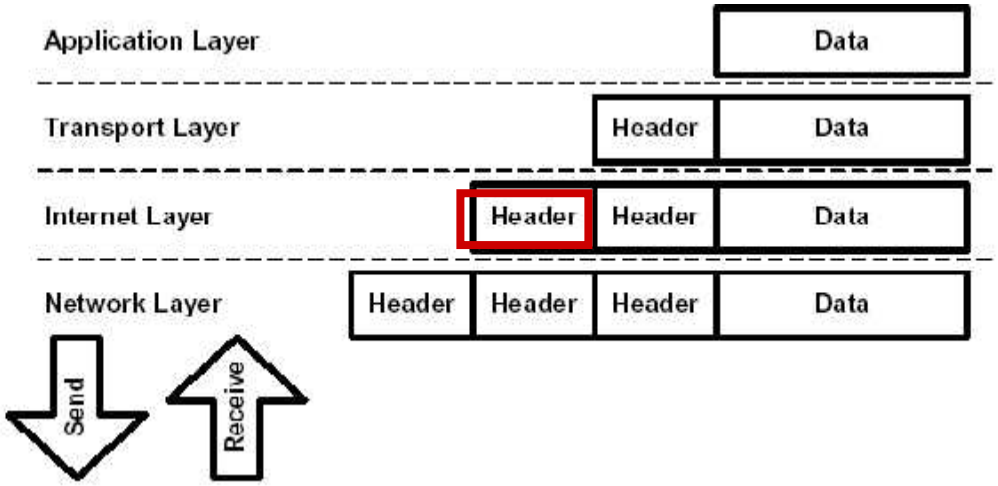
... sigue ...



Internet Layer: IP Header

- ... continuación ... :
  - Protocol: protocolo usado en la carga útil
  - Header Checksum: validación del encabezado
  - Source IP Address: 32 bits en IPv4, 48 bits en IPv6
  - Destination IP Address: 32 bits en IPv4, 48 bits en IPv6
  - Options: debe ser agregado aunque muchas veces no es usado

los 4.000 millones de posibles números de IP en IPv4 resultan pocos, por eso se agregaron 2 bytes más en IPv6

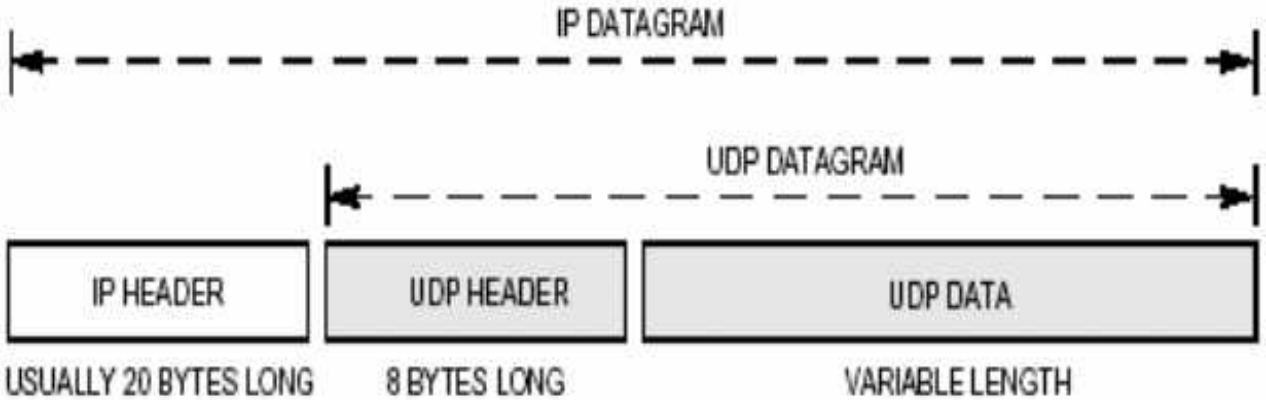


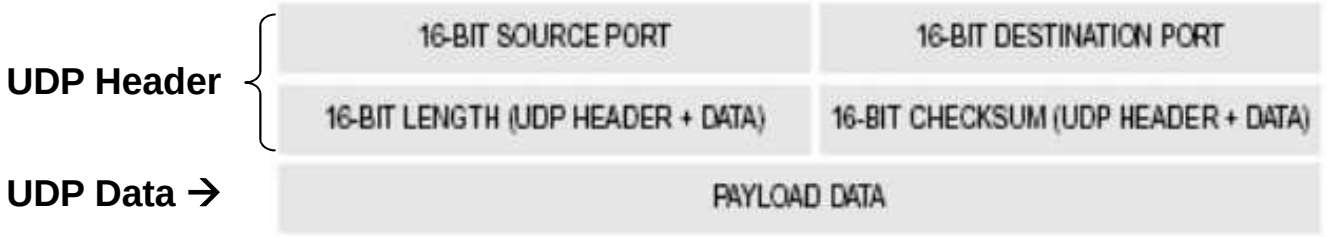
Servicios básicos: datagramas

- Un *datagrama* es un fragmento de información que es enviada con la información necesaria como para que una red pueda dirigirlo hacia el receptor.
- En general cada datagrama es ruteado en modo independiente a otros datagramas, y no provee mecanismos que aseguren control de flujo, de errores o garantía de llegada a destino.
- El llamado UDP (por *User Datagram Protocol*), es usado por servicios y protocolos de mayor nivel, como DHCP, BOOTP, DNS, que no requieren de conexiones virtuales.

Capa de transporte: UDP
User Datagram Protocol

- UDP es un protocolo básico para el envío de mensajes breves; no garantiza el correcto envío ni el orden de los paquetes y puede ser usado para envío de mensajes de broadcast. Entre los servicios que usan UDP están el Domain Name System (DNS), o Voice over IP (VoIP)

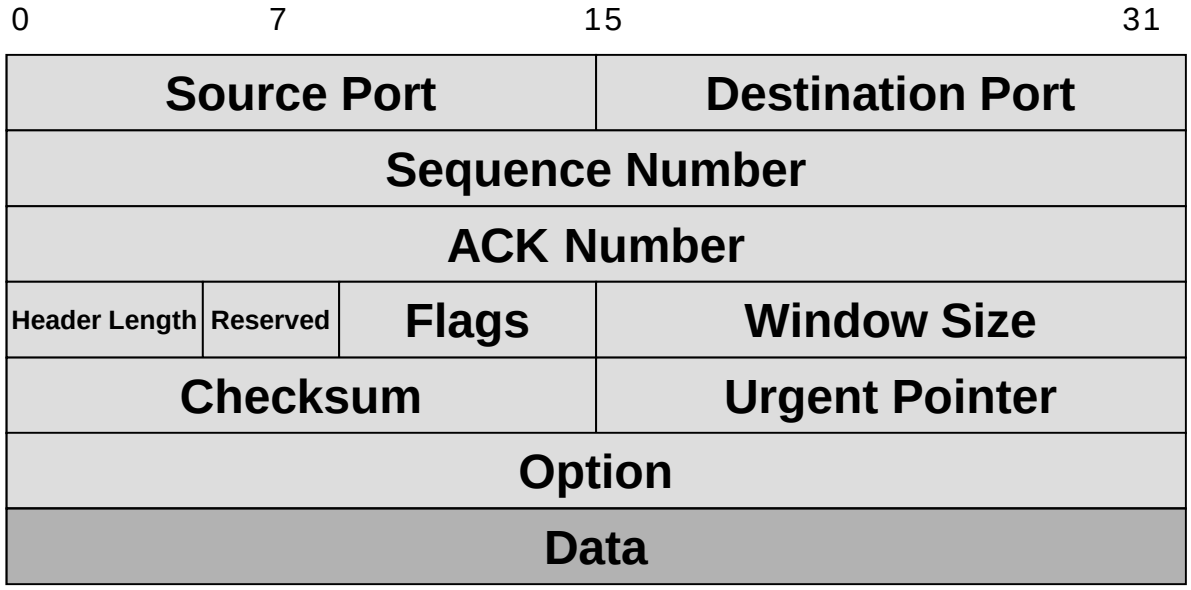




Servicios básicos: conexión virtual

- ***Conexión virtual*** o ***circuito virtual*** es un mecanismo de intercambio donde la API se encarga de las tareas de fragmentación, rearmado, numeración de los fragmentos, reintento ante errores, de modo de brindar un servicio que garantice una conexión estable aparente.
- En este caso, el llamado ***TCP*** (por ***Transmission Control Protocol***), es una capa que se establece a este fin sobre el mucho más simple protocolo IP de transporte de paquetes.
- El establecimiento de a conexión virtual no implica la reserva de recursos, como en circuitos conmutados, ni garantiza una performance temporal.

Capa de transporte: TCP
Transport Control Protocol



Capa de transporte: TCP
Transport Control Protocol

El encabezado y los datos de un paquete TCP son parte del payload de un paquete IP. El encabezado contiene

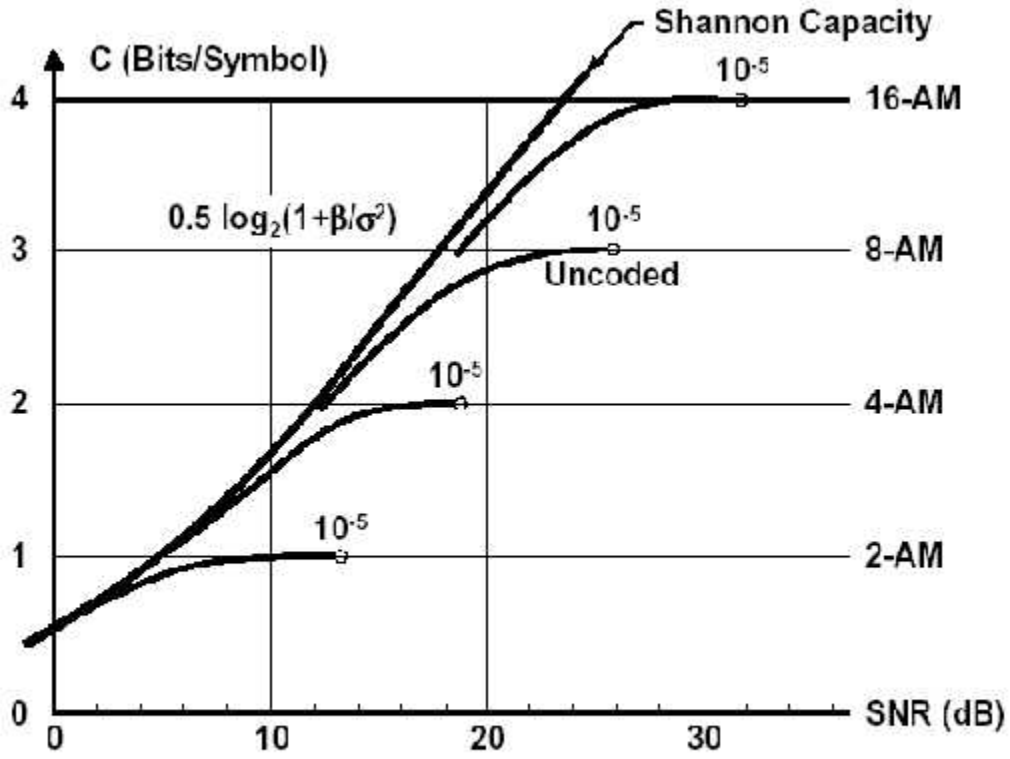
- Sequence number: numero de secuencia del primer byte de un mensaje o del primer byte de un segmento sucesivo transmitidos
- Acknowledgement number: número de secuencia del próximo byte esperado
- Data offset: tamaño del encabezado TCP en words (32 bits), de 20 a 60 bytes
- Flags: control de congestión, QoS, ACK, sincronización de números de secuencia, fin de datos, reset de la conexión.
- Window: cantidad de bytes que el receptor espera recibir
- Checksum: control de errores del encabezado Y los datos
- Urgent pointer: offset dentro de la secuencia para datos de uso especial
- Options (campo variable): opciones de manejo del campo de datos, incluyendo un registro temporal (Timestamp)

Y luego vienen los datos

Control de errores en canales de comunicación

Capacidad de un canal

- La capacidad de transmitir información por un canal está directamente relacionada con la relación entre señal y ruido en ese canal
- Más allá de la capacidad teórica (capacidad de Shannon) los métodos reales suelen generar limitaciones prácticas
- Existe enorme esfuerzo en crear nuevas técnicas, y cada día se descubren nuevos métodos



The Information Conveyed by a Real-Valued Discrete time Channel with Additive Gaussian Noise

Control de errores: detección y corrección

- A priori hay un concepto básico a aclarar: ***NO EXISTE*** un canal de comunicaciones sin errores, aunque la tasa de éstos pueda hacerse tan baja como se desee
- La solución obvia para disminuir la tasa de errores es aumentar la relación señal/ruido, pero esta solución presenta limitaciones prácticas y regulatorias
- Una alternativa es aceptar la existencia de errores y emplear métodos que sirvan para detectarlos mediante el agregado de información adicional (redundancia) cuya validez se chequea en el receptor, que eventualmente puede pedir su retransmisión (*backward error correction*)
- Otra alternativa en los casos en que pedir una retransmisión es inconveniente es agregar redundancia que permita al receptor corregir los errores que aparecen (*forward error correction*).

Tipos de errores

- Es diferente al caso de los errores inducidos por efectos extraordinarios (como la radiación) que son catalogados en SET, SEU, SEL.
- En un canal de comunicación la señal sufre la influencia combinada del ruido térmico, señales externas indeseadas (ruido electromagnético, jamming) combinado al efecto de atenuación (fading), distorsión por suma de múltiples caminos (multipath).
- El comportamiento de estas señales indeseadas puede ser:
  - de modelado estadístico estable (caso de señales ergódicas, como el ruido térmico)
  - de modelado variable pero de cambio dinámico lento (caso multipath y fading) que permite el diseño de sistemas adaptativos (AGC para el fading)
  - de ocurrencia impulsiva (caso de bursts de ruido electromagnético)
  - interferencias de energía concentrada (jamming)
- Cada uno de estos casos motiva distintas soluciones

Tipos de errores

- Previo a definir una estrategia de detección y corrección de errores debe definirse y dar valores al modelo del tipo de errores a detectar y/o corregir:
  - *Aleatorios*: es un error no sistemático, definido por su probabilidad de ocurrencia, usando una unidad de medida llamada BER (bit error rate) como un porcentaje de la cantidad de bits que es probable que resulten alterados en un mensaje, dividido por la longitud total del mensaje. En un canal con ruido blanco de distribución gaussiana, el BER está directamente relacionado a la relación de potencia señal a ruido (SNR).
  - *Tipo burst*: un error burst corresponde a una secuencia contigua de L símbolos tal que el primer y último bit de la secuencia están errados y no existe dentro de la secuencia subsecuencias de M o más símbolos correctos (el primer y último bit de la secuencia están separados “al menos” M tiempos de bit). El largo L, la probabilidad de ocurrencia, y M definen la estrategia de corrección de errores a seguir.
- Una vez definido el tipo de error que se desea combatir, también es necesario definir qué BER es aceptable y en qué condición

Cuándo sirve la detección de errores?

- ✓ La detección de errores y retransmisión de datos conviene en enlaces punto a punto de bajo tiempo de tránsito, asincrónicos, que pueden tolerar los retardos variables causados por la retransmisión del bloque errado, y con bloques de dimensión acotada
- ✓ Cuando el tiempo de tránsito es elevado, entre que se envía un mensaje hasta que llega el pedido de retransmisión pueden haber sido transmitidos muchos datos, por lo que la pérdida de eficiencia es enorme. Algunos algoritmos de numeración de paquetes y de buffers deslizantes (sliding window) pueden atenuar parcialmente este efecto, retransmisitendo sólo los bloques dañados.
- ✓ En casos donde el uso de FEC es impracticable por complejidad, costo, o ancho de banda, la detetcción de errores puede ser usada para estrategias alternativas (repetir un cuadro de imagen dañado, por ejemplo, o “reinventarlo” interpolando entre el previo y el posterior)

Cuándo sirve la corrección de errores?

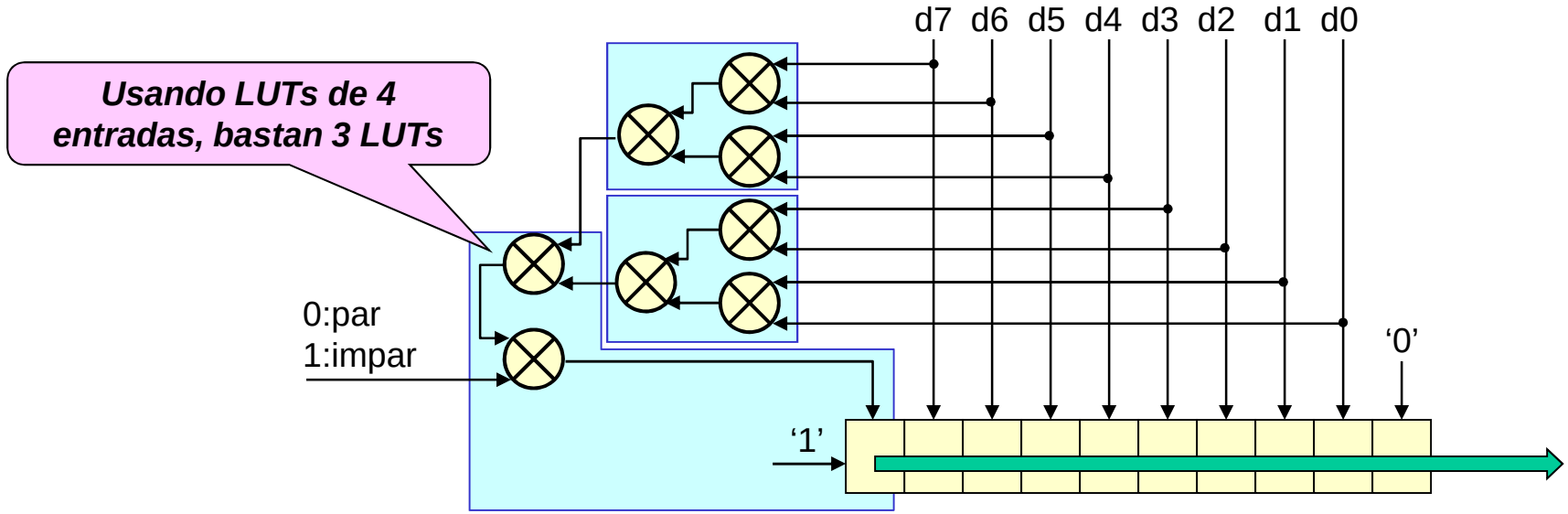
- ✓ Cuando la sincronía de recepción es importante (por ejemplo, en video) no es aceptable volver atrás y recomenzar, por la pérdida de continuidad de la imagen.
- ✓ En el caso de información de tipo broadcast, con muchos receptores, la retransmisión individual tampoco es aceptable.
- ✓ Cuando no existe la posibilidad de pedir retransmisión (información en un CD, datos que provienen de una sonda en el espacio profundo).
- ✓ Cuando el tiempo de tránsito es tan elevado, que los algoritmos de numeración de paquetes y de buffers deslizantes resultan imprácticos.
- ✓ En estos casos es de utilidad agregar un caudal conocido de información extra que permita que el receptor “repare” aquellos datos recibidos con error. Esta redundancia agregada es lógicamente mayor cuanto mayor se desea que sea la capacidad de autocorrección.

Métodos de detección

- ✓Si el ruido del canal genera errores aislados, la tasa de error es baja y los mensajes cortos, es razonable emplear métodos sensibles a la detección de errores simples, tal como la paridad (distancia de Hamming=2). Es el caso de la comunicación serie asincrónica, donde los paquetes son de sólo 8 bits (1 carácter).

- ✓SI los mensajes son largos o el ruido es tipo “burst” crece la posibilidad de que el mensaje incluya varios errores (simples o múltiples), y se emplean métodos como el CRC (Cyclic Redundancy Check). Es el caso de los archivos magnéticos (Hard Disk), o métodos de comunicaciones (CAN, PCI-Express, Ethernet, entre otros) donde los paquetes pueden llegar a ser de varios miles de bits.

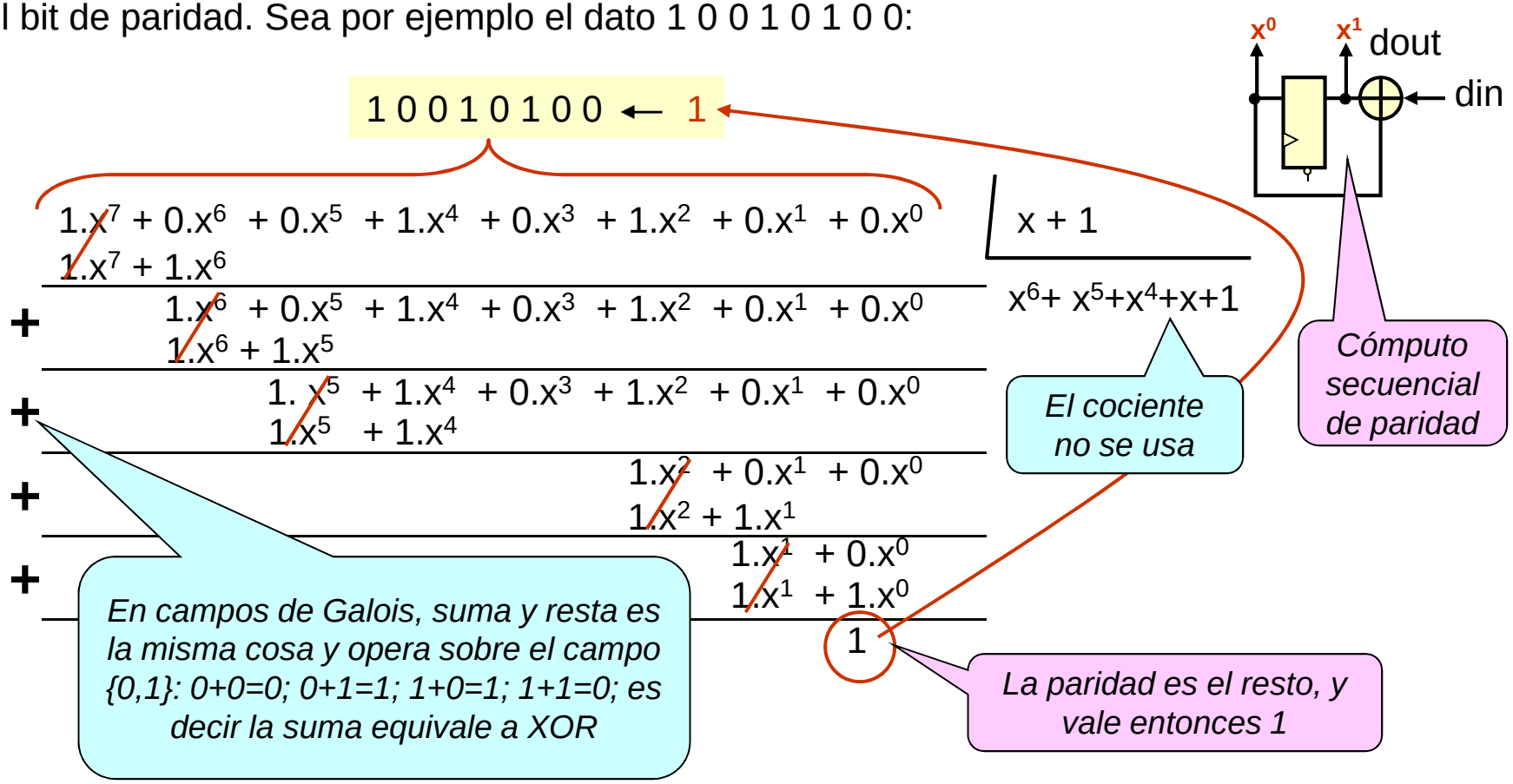
Control de errores en enlaces serie: Paridad



- El método de paridad agrega al mensaje un bit tal que la cantidad total de '1' en el mensaje sea par (paridad PAR) o impar (paridad IMPAR).
- Este método detecta la existencia de errores en TODOS los mensajes donde la cantidad de errores sea impar (1,3,5,7,9) pero NINGUN error doble, cuádruple, sextuple u octuple.
- En condiciones de buena SNR y para comunicaciones asincrónicas (con secuencias de 11 bits: Start, Stop, Paridad, y 8 bits de payload) la probabilidad de ocurrencia de más de un error es muy baja, por lo que en ese caso la paridad es un método conveniente

La paridad vista como una división de polinomios

- La paridad puede ser vista como la evaluación de un ***cociente de polinomios***, donde el dato es el ***dividendo***, el ***divisor*** vale (**x+1**), y donde el ***resto*** de la división define el valor del bit de paridad. Sea por ejemplo el dato 1 0 0 1 0 1 0 0:



La división de polinomios.. y si agrego la paridad al dato?

- Si al dato evaluado **1 0 0 1 0 1 0 0** le agrego la paridad calculada (**1**) me queda el nuevo dato **1 0 0 1 0 1 0 0 1**. Si este valor es usado como el dividendo, y el divisor sigue siendo (**x+1**), el resto de la división (**1** o **0**) define ahora si hay o no error :

1 0 0 1 0 1 0 0 1 — *La paridad se agrega luego del dato*

$1.x^8 + 0.x^7 + 0.x^6 + 1.x^5 + 0.x^4 + 1.x^3 + 0.x^2 + 0.x^1 + 1.x^0$ | $x + 1$

$1.x^8 + 1.x^7$ | $x^7 + x^6 + x^5 + x^2 + x^1 + 1$

$+ \quad 1.x^7 + 0.x^6 + 1.x^5 + 0.x^4 + 1.x^3 + 0.x^2 + 0.x^1 + 1.x^0$

$1.x^7 + 1.x^6$

$+ \quad 1.x^6 + 1.x^5 + 0.x^4 + 1.x^3 + 0.x^2 + 0.x^1 + 1.x^0$

$1.x^6 + 1.x^5$

$+ \quad 1.x^3 + 0.x^2 + 0.x^1 + 1.x^0$

$1.x^3 + 1.x^2$

$+ \quad 1.x^2 + 0.x^1 + 1.x^0$

$1.x^2 + 1.x^1$

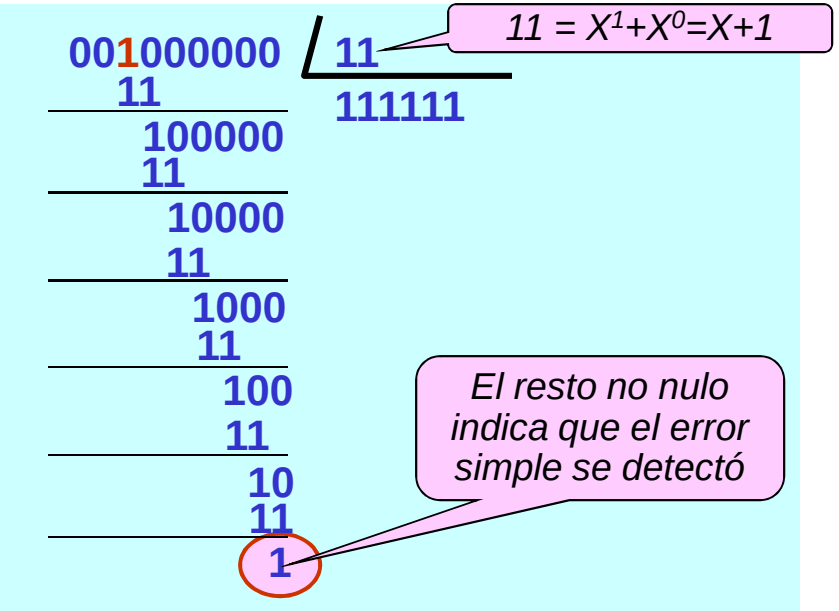
$+ \quad 1.x^1 + 1.x^0$

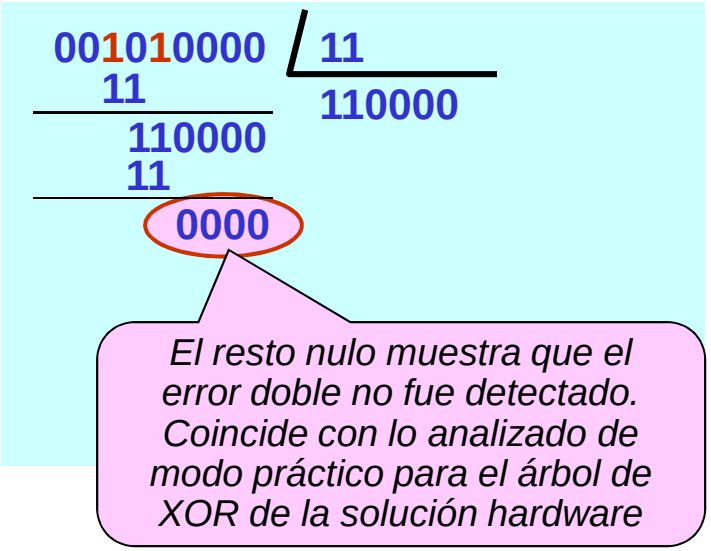
$1.x^1 + 1.x^0$

0 — *El resto me indica la existencia o no de errores*

La división de polinomios..cómo interpreto los errores?

- Si el polinomio trasmitido es **100101001** y se recibe **101101001** (un error en el tercer bit) puede interpretarse como que se sumó **001000000** al polinomio transmitido. Como vale que el cociente y resto de una suma de polinomios es igual a la suma del cociente y resto de los polinomios individuales, y como el polinomio original tiene resto cero, no se detectará el error si el polinomio de error al ser divido por (**x+1**) también tiene resto nulo.
- Por ejemplo, en el caso del error simple **001000000** y del error doble **001010000** :





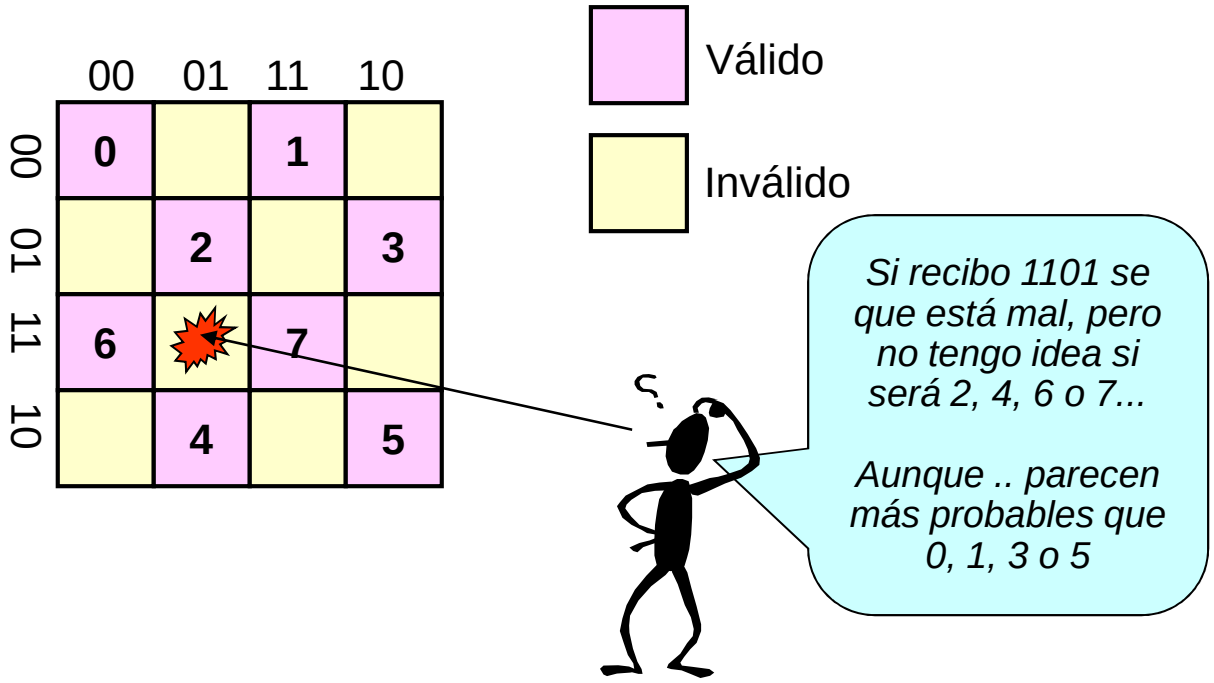
Criterios de agregado de redundancia

- Al agregar redundancia a los datos la cantidad de bits aumenta
- Si el dato original tiene N bits, tiene 2^N posibles valores o combinaciones. Con agregar un bit de redundancia la cantidad de valores posibles se duplica, y así sigue duplicándose por cada bit extra
- Si se agregan M bits redundantes, de los 2^{N+M} posibles valores sólo 2^N corresponden a valores válidos y el resto a valores erróneos
- Cuando se recibe un valor errado (de ***N+M*** bits), la idea de "máxima probabilidad de ocurrencia" (*maximum likelihood*) se usa para elegir cuál el valor correcto más probable
- Para ello se usa un concepto de distancia, en general asociado a cuántos bits cambian entre ***dato válido+redundancia*** y el ***dato recibido*** (*distancia de Hamming*).
- En los casos en que sólo puede detectarse que el código es erróneo, pero no cuál es el código correcto más probable (por haber varios igualmente probables), se habla de detección de errores
- En los casos en que puede encontrarse cuál es el código correcto más probable, se habla de corrección de errores

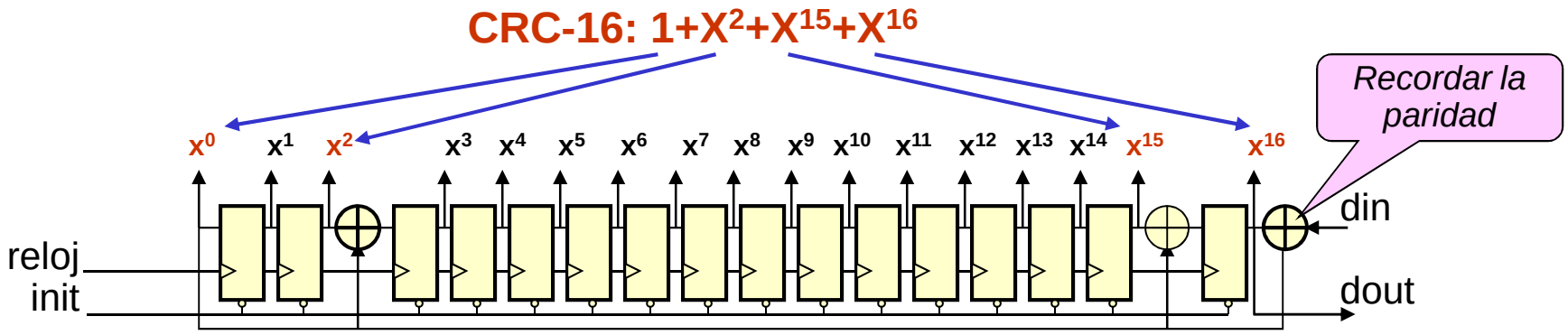
La distancia en el caso de la paridad

- Al agregar un bit de paridad a los datos la cantidad de combinaciones se duplica
- Si se usan mapas de Karnaugh (caracterizados por la propiedad de adyacencia, donde sólo cambia un bit (distancia de Hamming=1) al moverse un cuadro en sentido horizontal o vertical) se puede ver porqué puede detectar pero no puede corregir errores de un bit.

Dato	Paridad
000 = 0	0
001 = 1	1
010 = 2	1
011 = 3	0
100 = 4	1
101 = 5	0
110 = 6	0
111 = 7	1



Control de errores en enlaces serie: Generador/tester de CRC

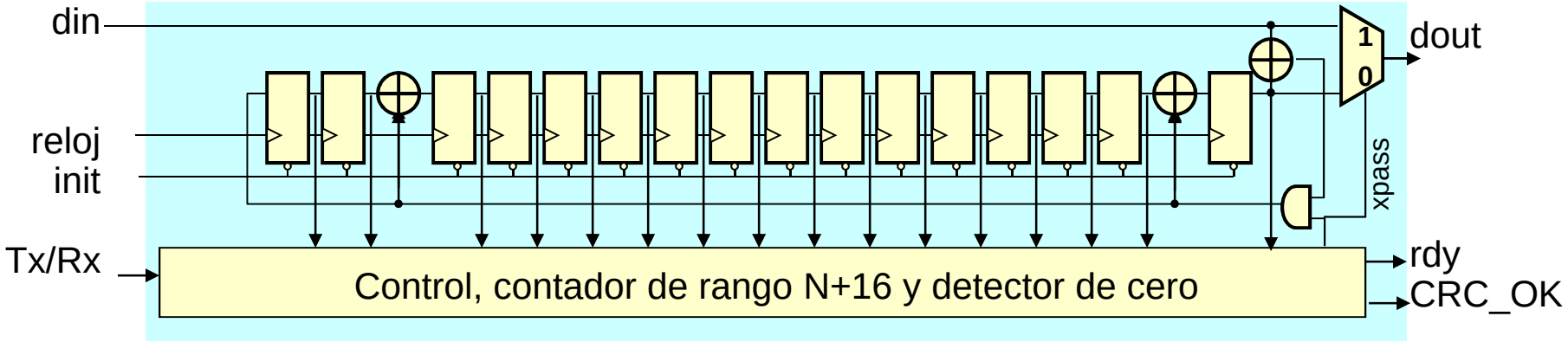


- El CRC (*Cyclic Redundancy Checking*) emplea LFSRs (*Linear Feedback Shift Registers*), pero agregando como variable adicional un canal de datos que ingresan en forma serial. De este modo, el proceso de generación de CRC es como sigue:
  - El registro se inicializa con todos sus bits en un valor predefinido (usualmente '1..1')
  - Se ingresan los datos en forma serial
  - Al finalizar este ingreso, el código resultante en el registro es el CRC
- De igual modo, el proceso de verificación de CRC por el receptor es el siguiente
  - El registro se inicializa con todos sus bits en el mismo valor predefinido ('1..1')
  - Se ingresan los datos en forma serial, y luego de ellos el CRC
  - Al finalizar este ingreso, el código resultante (el resto) debe ser todos ceros

Fuente: 74F401 en www.fairchildsemi.com

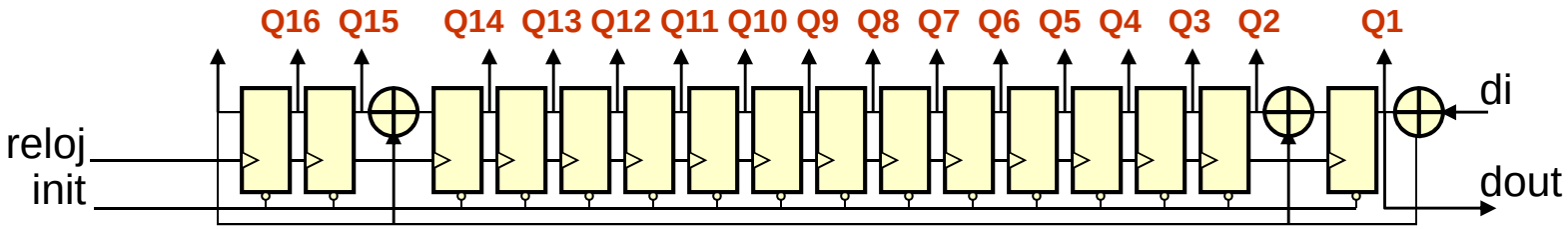
Generador/tester de CRC

- Además del CRC-16 existen otras versiones de CRC muy usadas:
  - CRC-12: $1 + X + X^2 + X^3 + X^{11} + X^{12}$
  - CRC-CCITT: $1 + X^5 + X^{12} + X^{16}$
  - CRC-32: $1 + X + X^2 + X^4 + X^5 + X^7 + X^8 + X^{10} + X^{11} + X^{12} + X^{16} + X^{22} + X^{23} + X^{26} + X^{32}$
- donde CRC-12 es usada con datos de 6 bits, CRC-16 y CRC-CCITT para caracteres de 8 bits, y CRC-32 en canales de comunicación sincrónicos (por ejemplo la norma IEEE-802 especifica el uso de CRC-32).



- Un generador/testeador de ***CRC-16*** para transmisión y recepcion de mensajes de N bits podría tener una arquitectura como la que se indica en el gráfico

Cómputo paralelo de CRC-16



- Dados los 16 registros R[16..1] usados en un CRC-16 y su valor inicial Q16..Q1, y conocidos 8 datos sucesivos de entrada Di (D1..D8), puede evaluarse qué sucede luego de 8 ciclos de reloj

	R16	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1
T=0	Q16	Q15	Q14	Q13	Q12	Q11	Q10	Q9	Q8	Q7	Q6	Q5	Q4	Q3	Q2	Q1
T=1	Q1 D1	Q16	Q15 Q1 D1	Q14	Q13	Q12	Q11	Q10	Q9	Q8	Q7	Q6	Q5	Q4	Q3	Q2 Q1 D1
T=2	Q2 Q1 D1 D2	Q1 Di	Q16 Q2 Q1 D1 D2	Q15 Q1 Di	Q14	Q13	Q12	Q11	Q10	Q9	Q8	Q7	Q6	Q5	Q4	Q3 Q2 Q1 D1 D2

Perez, A. 1983. "Byte-wise CRC Calculations." IEEE Micro. June. 40-50

Cómputo paralelo de CRC-16

- Siguiendo el análisis hasta T=8, y llamando ***Xi*** a ***Di XOR Qi***, se puede llegar a:

	R16	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1
T=8	X8 X7 X6 X5 X4 X3 X2 X1	X7 X6 X5 X4 X3 X2 X1	X8 X7	X7 X6	X6 X5	X5 X4	X4 X3	X3 X2	Q16 X2 X1	Q15 X1	Q14	Q13	Q12	Q11	Q10	Q9 X8 X7 X6 X5 X4 X3 X2 X1

- Viendo que un elemento lógico tiene 4 entradas, y que cada **Xi** requiere dos variables, pueden definirse términos **Xij = (Xi xor Xj)**, solucionables con un único LE, y llamando **Xf = (X87 xor X65 xor X43 xor X21)**, queda:

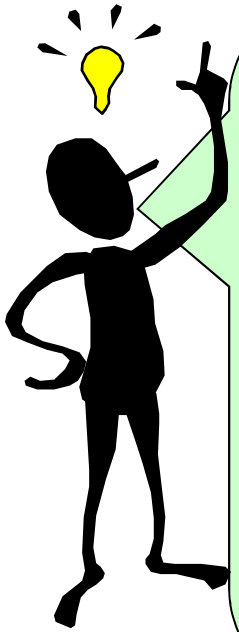
	R16	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1
T=8	Xf	X76 X54 X32 X1	X87	X76	X65	X54	X43	X32	Q16 X21	Q15 X1	Q14	Q13	Q12	Q11	Q10	Q9 Xf

Perez, A. 1983. "Byte-wise CRC Calculations." IEEE Micro. June. 40-50

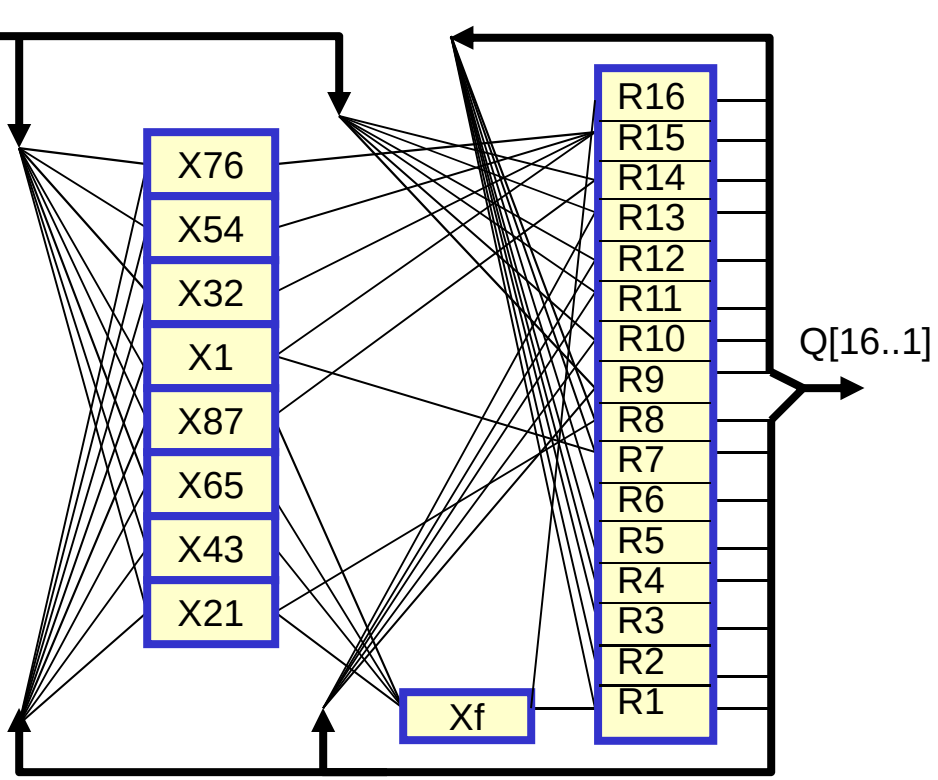
Cómputo paralelo de CRC-16

- En base a las expresiones resultantes puede evaluarse la red de conexiones:

	R16	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1
T8	Xf	X76,X54 X32,X1	X87	X76	X65	X54	X43	X32	Q16 X21	Q15 X1	Q14	Q13	Q12	Q11	Q10	Q9 Xf



- ***Sólo necesito los 16 registros de CRC más otros 9 operando en modo combinatorio para calcular un nuevo byte de CRC en un único ciclo de reloj (modo NORMAL)***
- ***El peor caso para la velocidad es el cálculo de R1, que requiere atravesar dos LE combinatorios***
- ***Conviene poner X87, X65, X43, X21, Xf y R1 en un clique***
- ***Usando "register packing" los recursos usados son menores (Xf es generado por R16, X1 por R7,..), puede bajar a 17 LEs, y ser tan eficiente como la solución serie***



El cómputo paralelo de CRC-16 en software

Si se analiza el resultado del cálculo de CRC de a byte:

	R16	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1
T8	Xf	X76,X54 X32,X1	X87	X76	X65	X54	X43	X32	Q16 X21	Q15 X1	Q14	Q13	Q12	Q11	Q10	Q9 Xf
T1	Xf	X76,X54 X32,X1	X87	X76	X65	X54	X43	X32								
T2									X21	X1	0	0	0	0	0	Xf

Surge que es muy fácil calcular el CRC-16 por software en base al uso de dos tablas de 256 bytes, usando al dato de entrada de 8 bits como puntero:

```
unsigned int crc_16 (unsigned int crc_actual, unsigned char dato) {
  unsigned int crc_h,crc_l,crcLB, crcHB;
  crc_LB = crc_actual & 0x00FF; crc_HB = crc_actual >> 8;
// parte alta del nuevo CRC depende de dato y la parte baja del CRC_actual
  crc_h = ((tab_crc_h[dato]) ^ (tab_crc_h[crc_LB])) << 8;
// la parte baja depende de dato y de las partes alta y baja del CRC previo
  crc_l = crc_HB ^ tab_crc_l[dato] ^ (tab_crc_l[crc_LB];
  return (crc_h | crc_low);}
```

donde las tablas usadas son del tipo

```
unsigned char tab_crc_h [256] = {..,..,..}; // los datos se calculan con T1
unsigned char tab_crc_l [256] = {..,..,..}; // los datos se calculan con T2
```

El CRC visto como una división de polinomios. Propiedades

- La generación de CRC se basa en el mismo principio que la generación de paridad, sólo que usando un polinomio de cálculo de mayor orden, en vez de (x+1).
- Como en la paridad, el generador de CRC divide la secuencia de datos por el divisor (realizado con registros y XOR) y agrega el resto luego de los datos.
- Y cuando el receptor divide la secuencia de datos+resto por el mismo polinomio, el resto debe dar cero. Surge que la habilidad de detección de error del CRC elegido depende de cuáles polinomios de error son divisibles sin generar resto por el de CRC.
- El CRC se adapta a la detección de errores en secuencias de mayor longitud, donde existe la posibilidad de más de un error, y también la ocurrencia de bursts.
- En este caso, las virtudes del método se basan en estimar qué porcentaje de errores simples detecta, qué porcentaje de dobles, de triples, de cuádruples, etc...
  - Para el caso de errores aislados, CRC detecta TODOS los errores de un bit.
  - Para el caso de bursts, un CRC de ***n***-bits es capaz de detectar un burst simple de hasta ***n*** bits y una fracción de bursts de mayor longitud, así como todos los errores tipo burst que afectan un número impar de bits

Todo ésto forma parte de la teoría de álgebra lineal y campos de Galois

Corrección de errores: corrección por bloques o secuencial

- Por bloques (Por ejemplo: Hamming, Reed Solomon):
  - En este caso, antes de transmitir un bloque de datos útiles se calculan los datos a agregar, y recién entonces se transmiten. Del lado receptor, se recibe el bloque aumentado, y en base a ellos se recuperan corregidos los datos útiles. Un caso usual es tomar 238 bytes de datos, agregar 16 bytes de redundancia, y un byte de sincronismo de bloque (un 7,14% de información extra). Este agregado permite detectar hasta 16 bytes errados y corregir hasta 8 bytes errados dentro del bloque.
- Secuencial (Por ejemplo: convolucional + Viterbi):
  - En este caso se generan bits de salida extra a medida que los datos van saliendo, a través de un generador de respuesta finita al impulso, con una relación (bits útiles/bits transmitidos) que usualmente puede ser desde ½ (un bit extra por cada bit útil) hasta 9/10 (un bit extra por cada 9 bits útiles).
  - A la llegada se recuperan los datos en base a evaluar cuál es la secuencia más probable, usando la idea de un árbol de trayectorias posibles (Trellis).

Corrección de errores tipo burst interleaving de datos de bloques

- Además de agregar información extra en cada paquete, para compensar los errores burst pueden emplearse técnicas que tratan de hacerlos más “random”, distribuyéndolos en varios paquetes, como el interleaving: