Contenido

Una Introducción al Control de Flujo en la Comunicación

M. en C. Mauricio Olguín Carvajal
M. en C. Israel Rivera Zárate
Profesores del CIDETEC

Silvia Nora Chávez Morones
Fernando Mancilla Téllez
Imelda Vázquez Rojas
Alumnos de la especialidad en redes de computadoras, UNITEC Campus Sur

El siguiente trabajo muestra, desde un punto de vista informático, un panorama general del funcionamiento de los protocolos de control de flujo que operan en la capa 2 del **Modelo de Interconexión de Sistemas Abiertos** "OSI".

Comunicación

¿Te has preguntado alguna vez como sería la comunicación entre los seres humanos si no existieran reglas para establecer dicha comunicación?

Como te podrás imaginar, si no existieran reglas de comunicación (en el caso de un idioma: sintaxis, semántica, gramática, etc.), todos hablaríamos al mismo tiempo, no sabrías si la persona a la que quieres dirigir el mensaje te esta escuchando a ti o está conversando con alguna otra persona, y no podrías confirmar si el mensaje que enviaste llegó correctamente o fue escuchado por alguien indebido, etc.

Por lo tanto, para evitar esta situación contamos con nuestro sistema de comunicación, en donde existe un emisor, un medio a través del cual comunicarnos, (teléfono, papel, aire, etc.), un mensaje y un receptor.

Otra premisa que se debe considerar para lograr establecer una comunicación es el idioma utilizado. Por ejemplo, si estamos en un grupo en el que cada participante habla en diferente idioma, es necesario establecer el idioma que se utilizará y que sea entendible para todos.

Ahora bien, ¿que pasaría si este concepto lo aplicamos al ámbito informático?

En los sistemas de comunicación basados en computadoras, los datos se representan con unidades de información binaria (o bits) producidos y procesados en forma de ceros y unos.

Para que la transmisión de datos sea posible, los dispositivos de comunicación deben ser parte de un sistema de comunicación formado por 5 componentes: mensaje, emisor, receptor, medio, y protocolo.

Como te podrás dar cuenta, en el ámbito informático se adicionó el concepto protocolo que quiere decir "conjunto de reglas que gobiernan la transmisión de datos".

Como sabes, existen diferentes proveedores de equipos de cómputo, y se requiere establecer los protocolos para comunicarse. Esto se asemeja al grupo de personas que hablan diferentes idiomas por lo que es necesario establecer uno solo.

Para poder lograr este objetivo la Organización Internacional de Estandarización (ISO), creo un estándar denominado "Modelo de Interconexión de Sistemas Abiertos (OSI)", el cual permite que dos sistemas diferentes se puedan comunicar independientemente de su arquitectura.

El modelo OSI esta integrado por 7 capas, de las cuales analizaremos únicamente las 2 primeras, para comprender como se da el proceso de comunicación.

La 1er capa es la capa física, que coordina las funciones necesarias para adecuar el medio a través del cual se transmitirá la información.

La 2ª capa del modelo OSI, es la capa de enlace de datos; a través de esta capa se lograrán los siguientes objetivos:

Entrega: El sistema debe entregar los datos en el destino correcto. Es decir los datos deben ser recibidos por el dispositivo o usuario adecuado y solamente por él.

Exactitud: El sistema debe entregar los datos con exactitud. Los datos que se alteran en la transmisión son incorrectos y no se pueden utilizar.

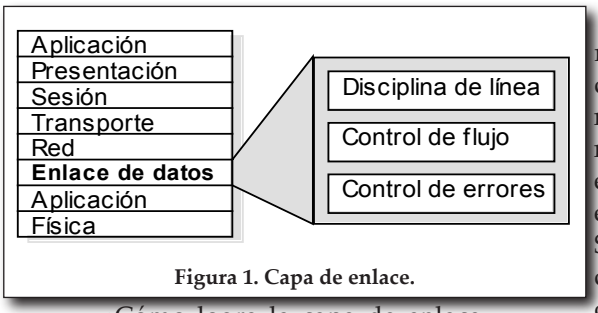

Figura 1. Capa de enlace.

¿Cómo logra la capa de enlace de datos cumplir con dichos objetivos?

La capa de enlace de datos se subdivide en las siguientes funciones, mostradas en la **Figura 1**.

Disciplina de línea: Se encarga de determinar que dispositivo puede enviar información y cuando puede hacerlo. Los métodos que utiliza para lograr esto son:

Sondeo/reconocimiento (ENQ/ACK). Este método se utiliza en enlaces dedicados, únicamente existen en el medio 2 dispositivos, uno para enviar y otro para recibir. Cualquier estación de enlace puede empezar una sesión siempre y cuando ambas tengan el mismo rango; por ejemplo, una impresora no puede empezar una comunicación con una CPU.

El dispositivo que inicia la transmisión envía una trama denominada solicitud (ENQ), preguntando si el receptor está disponible para recibir datos. El receptor debe responder con una trama de reconocimiento (ACK) si esta listo para recibir, o con una trama de reconocimiento negativo (NAK).

Si el transmisor no recibe un ACK o NAK, dentro de un límite de tiempo especificado, se asume que la trama ENQ se ha perdido en la transmisión, se desconecta y envía un reemplazo.

Un sistema que inicia la conexión realiza 3 intentos para establecer el enlace antes de abandonar. Si la respuesta a ENQ es negativa para los 3 intentos, el equipo se desconecta y reinicia el proceso en otro momento. Si la respuesta es positiva, la conexión esta lista para enviar datos. Una vez que el transmisor termina de enviar todos sus datos envía una trama de **fin de transmisión (EOT)**

Sondeo/selección (Poll/Select). Este método se utiliza en sistemas multipuntos, en donde uno de los dispositivos ha sido designado como estación primaria y los otros dispositivos son estaciones secundarias (**Figura 2**).

Siempre que un enlace multipunto este formado por un dispositivo primario y múltiples dispositivos secundarios que usan una única línea de transmisión, todos los intercambios se deben hacer a través del dispositivo primario.

Cuando el equipo primario está disponible para recibir datos, les pregunta a los secundarios si tienen algo que enviar, a esto se le denomina **sondeo,** si el primario quiere enviar datos, le indica al secundario destino que se prepare para recibir datos, a esto se le llama **selección.**

Sondeo: Para iniciar la sesión, el equipo primario envía una trama ENQ al equipo secundario preguntando si tiene datos que enviar, si se responde con una trama NAK, quiere decir que no tiene nada que enviar y se continua preguntando al siguiente dispositivo, hasta que la respuesta es positiva, en este caso el primario inicia la transmisión de datos y cuando ya no tiene algo que enviar envía una trama EOT, de fin de transmisión.

Selección: Se utiliza siempre que el primario tenga algo que enviar; el primario envía una trama SEL, incluyendo la dirección del equipo al que van dirigidos los datos, el dispositivo secundario únicamente abre la trama y lee los datos cuando reconoce su propia dirección.

La segunda disciplina de la capa de enlace de datos es:

Control de flujo: Se encarga de determinar cuantos datos se pueden enviar antes de saturar la capacidad de los dispositivos, para lo cual deben considerarse los siguientes aspectos:

1.- El flujo de datos no debe saturar al receptor; los receptores tienen una velocidad límite para procesar los datos que llegan y una cantidad de memoria limitada en donde guardar los datos que llegan. El receptor debe avisar al trasmisor que detenga el envío de datos antes de llegar a estos límites y de pedir al dispositivo transmisor que envíe menos tramas o que pare temporalmente. La información que llega debe ser procesada y comprobada antes de usarse y esta tarea es más lenta que la velocidad de transmisión para el envío de la información. Por esta razón los equipos tienen

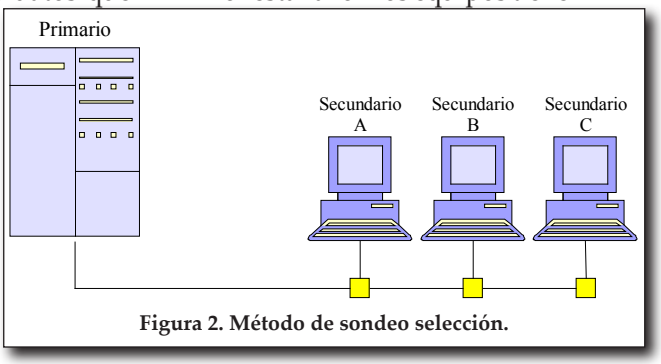

Figura 2. Método de sondeo selección.

un bloque de memoria llamado "buffer", reservado para almacenar la información que llega antes de ser procesada. Si el buffer comienza a llenarse, el receptor debe ser capaz de decirle al emisor que pare la transmisión hasta que vuelva a ser capaz de recibir

2.- Cuando una trama llega, ya sea individualmente o en grupo, notifica su recepción con un reconocimiento; si una trama llega dañada el receptor envía un mensaje de error NAK.

Se han desarrollado dos métodos para controlar el flujo de datos a través de enlaces de comunicación: Parada y espera, y ventana deslizante (**Figura 3**).

1.- Stop and Wait (parada y espera). Aquí, el emisor espera un reconocimiento después de cada trama que envía. Solamente se envía la siguiente trama cuando se ha recibido un reconocimiento. Este proceso de enviar y recibir alternativamente se repite hasta que el emisor envía una trama de fin de transmisión (EOT).

La ventaja de este método es su sencillez: cada trama es comprobada y reconocida antes de que se envíe la siguiente. La desventaja es su ineficiencia: la parada y espera es muy lenta, debido a que cada trama debe recorrer el camino hasta el receptor y un reconocimiento debe viajar del receptor al emisor antes de poder enviar la trama siguiente. Por lo tanto, cada trama está sola en la línea. Cada trama enviada y recibida usa todo el tiempo necesario para atravesar el enlace; si la distancia entre los dispositivos es larga, el tiempo que se gasta esperando un reconocimiento entre cada trama puede ser una parte importante del tiempo total de transmisión.

2.- Ventana deslizante. En este método, el emisor puede transmitir varias tramas antes de necesitar un reconocimiento.

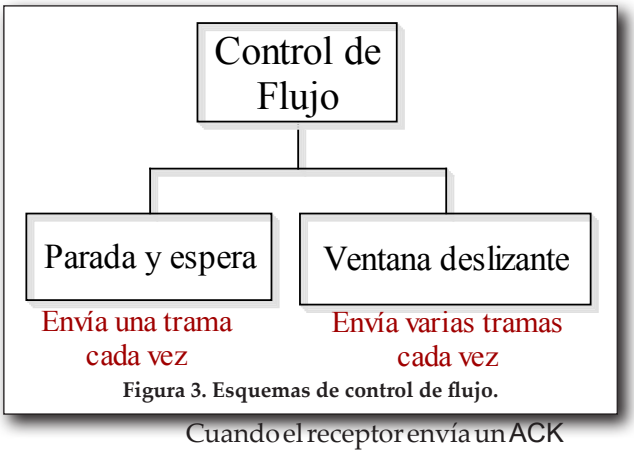

Figura 3. Esquemas de control de flujo.

Las tramas se pueden enviar una detrás de otra, lo que significa que el enlace puede transportar varias tramas de una vez y que su capacidad se puede usar de forma más eficiente. La palabra window en el término "sliding window", se refiere a un buffer extra creado tanto por el transmisor como el receptor. Las tramas se pueden retener en cualquier extremo antes de enviar un reconocimiento.

Para llevar control de la cantidad de tramas enviadas y recibidas se introduce una identificación basada en el tamaño de la ventana. La ventana deslizante usa unas cajas imaginarias en el emisor y el receptor. Esa ventana puede mantener tramas en cualquiera de los dos extremos y proporciona un límite superior en el número de tramas que se pueden transmitir antes de recibir un reconocimiento. Las tramas pueden ser reconocidas en cualquier momento sin esperar hasta que la ventana se llene y pueden ser transmitidas mientras que la ventana no está todavía llena.

Para saber qué tramas se han transmitido y cuáles se han recibido, la ventana deslizante introduce un esquema de identificación basado en el tamaño de la ventana, es decir, las tramas son numeradas como módulo-n, o sea de 0 hasta n-1.

Ejemplo: si n=8 las tramas se numeran 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3,4,5,6,7,0,1,.....

El tamaño de la ventana es n-1; en este caso es 7.

Cuando el receptor envía un ACK incluye el número de la trama que espera recibir; o sea, si recibimos un grupo de tramas que terminaron con la trama 4 envía un ACK con el número 5.

Al principio de la transmisión la ventana del emisor contiene la trama n-1. A medida que se envían las tramas, el limite izquierdo de la ventana se mueve hacia adentro, reduciendo el tamaño de la misma. Dada una ventana de tamaño w, si se han transmitido 3 tramas desde el ultimo reconocimiento , el número de tramas que quedan en la ventana es w-3. Una vez que llega un (ACK), la ventana se extiende para permitir entrar un número de tramas igual al número de tramas reconocidos por el (ACK).

Dada una ventana de tamaño 7, si se han enviado las tramas del 0 al 4 y no se ha recibido ACK la ventana sólo contiene dos tramas, 5 y 6.

Si se recibe un ACK con el número 4 quiere decir que se recibieron las *tramas* del 0 al 3 sin daño y la ventana del transmisor de expande para incluir las próximas 4 *tramas;* así, la ventana tendrá seis *tramas* (5 y 6 más 7, 0, 1 y 2). Si el ACK recibido hubiera llegado con el número 2, la ventana se hubiera expandido únicamente por dos *tramas* y tendría un total de 4 tramas (5 y 6 más 7 y 0).

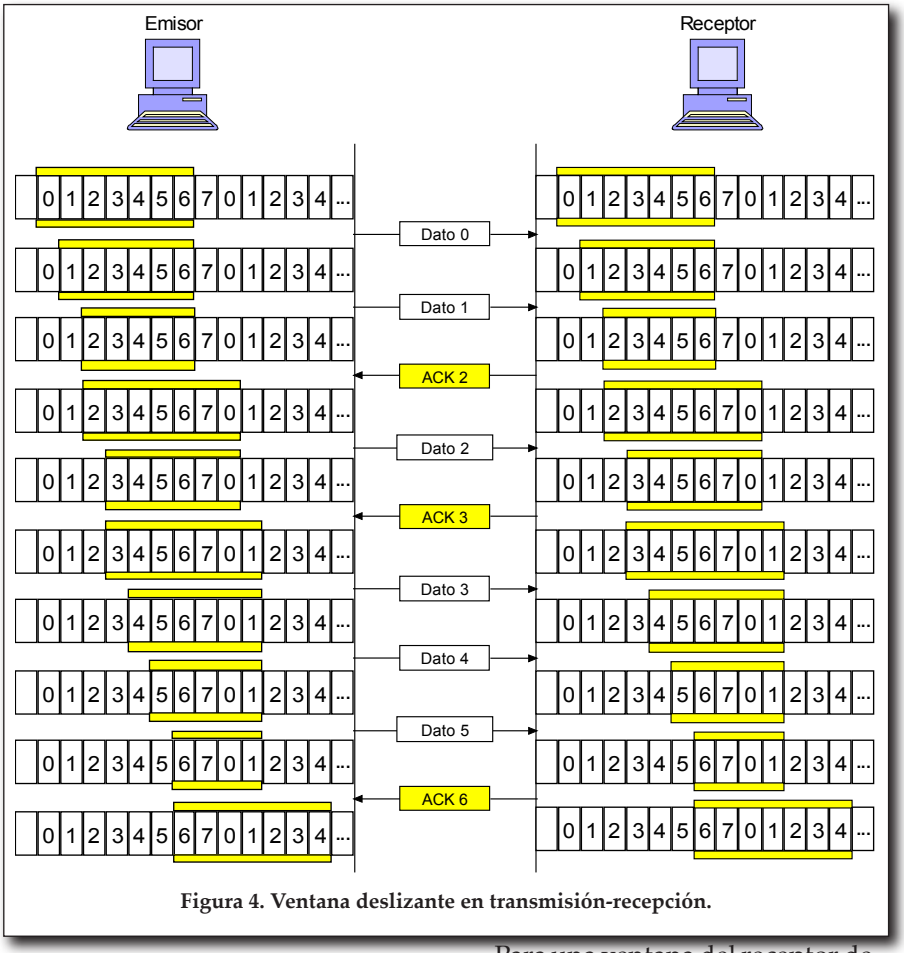

Figura 4. Ventana deslizante en transmisión-recepción.

Ventana Deslizante Receptora

Al inicio de una transmisión, la ventana transmisora no contiene n-1 *tramas*, sino n-1 espacios para *tramas*. A medida que llegan nuevas *tramas* el tamaño de la ventana de recepción se encoge. Por tanto, la ventana del receptor no representa el número de *tramas* recibidos sino el número que todavía se pueden recibir antes de enviar un reconocimiento (ACK).

Para una ventana de tamaño w, si se han recibido tres tramas sin devolver un reconocimiento (ACK), el número de espacios en la ventana es de w-n. En cuanto se envíe un reconocimiento, el tamaño de la ventana se expande para incluir lugares para los números de tramas iguales a los números de tramas reconocidas.

Para una ventana del receptor de tamaño 7, la ventana contiene espacios para 7 tramas, indicando que se pueden recibir antes de enviar un ACK. Con la llegada de la primera trama, la ventana del receptor se encoge, moviendo su frontera del espacio 0 al 1. La ventana se ha encogido 1, por lo que el receptor puede todavía aceptar 6 tramas antes de que tenga que enviar un ACK. Si han llegado tramas de la 0 a la 3 pero no se han reconocido, la ventana contendrá 3 espacios para tramas.

Cada vez que se envía un ACK la ventana receptora se expande para incluir espacios en función a la cantidad de tramas declarados más recientemente, menos la cantidad de tramas declaradas previamente. En una ventana con 7 tramas, si el ACK previo fue para la trama 2 y el ACK actual es para el trama 5, la ventana se expande 3 posiciones (5-2).

Otro caso, si el ACK anterior fue de la trama 3 y el actual es de la trama 1, la ventana se expande 6 lugares (1+8-3).

En el ejemplo que se utiliza, la ventana deslizante tiene siete tramas, las cuales llegan todas correctamente.

En el método de control de flujo de ventana deslizante, el tamaño de la ventana es una unidad menor que el rango del módulo, por lo que no hay ambigüedad en el reconocimiento de las tramas recibidas. Si el método nos indicara que el tamaño del marco es el mismo que el tamaño de la ventana, existiría el error siguiente.

Continuando con el ejemplo, si el marco fuese de 8 y el tamaño de la ventana también, si se envía la trama 0 y se recibe un ACK1, el emisor expande su ventana y envía las tramas 1,2.3,4.5,6,7 y 0. Si recibe de nuevo un ACK1, no está seguro de si es un duplicado del ACK anterior (duplicado por la red), o un ACK1 nuevo que confirma la recepción de las ocho tramas enviadas recientemente. Pero si el tamaño de la ventana es 7 (en lugar de 8), este escenario no puede suceder. En la **Figura 4** puede verse un esquema general de una ventana deslizante para transmisión-recepción.

Ejemplo de Ventana Deslizante

Hasta ahora nos hemos enfocado a los métodos de envió de datos, pero que pasa con los errores que se presentan durante la transmisión?

La 3a. función de la capa de enlace es el control de errores, la cual consiste en lo siguiente:

En la capa el término control de errores se refiere a los métodos de

detección de errores y retransmisión. Cada vez que un error es detectado se envía un reconocimiento negativo "NAK" y las tramas son retransmitidas. Este proceso se llama "Automatic Repeat Request" (ARQ). Se solicita un ARQ por ruido en los datos o tramas perdidos.

Conclusiones

Aunque retransmitir únicamente las tramas dañadas o las tramas perdidas puede parecer más eficiente que reenviar también tramas correctas, de hecho no es así.

Debido a la complejidad asociada a la ordenación y al almacenamiento necesario en el receptor y a la lógica extra necesaria en el emisor para seleccionar las tramas específicas para su retransmisión, el rechazo selectivo con ARQ es caro y no se usa a menudo. En otras palabras, el rechazo selectivo da mejores prestaciones, pero en la práctica se suele descartar a favor de la vuelta atrás por la sencillez de la implementación de este último.

Bibliografía

[1] http://www.monografias.com

[2] Behrouz A. *Transmisión de Datos y Redes de Comunicaciones*. 2a. Edición.

El Bus PCI Express: Generalidades

M. en C. Jesús Antonio Álvarez Cedillo
Ing. Macario García Arregui
Profesores del CIDETEC-IPN

El bus PCI (***Peripheral Component Interconnect***) ha sido el sistema estándar usado durante los últimos diez años, pero de acuerdo al tiempo de vida en el diseño de este, está comenzando a ser obsoleto. Las extensiones al modelo de PCI clásico, como son los conectores de 64 bits en velocidades de 66 a 100 MHz son muy caras, por lo que no van a la par de la tecnología que surge cada año y que cada vez es menos costosa. La 3ª generación de los dispositivos de entrada y salida llamados 3GIO[1] ("3 Generation Input Output"), ha creado una variación al bus PCI llamada PCI Express. Intel ha asegurado que sus productos contarán con el soporte para este nuevo bus, También la nueva versión de Windows, conocido como Longhorn, lo soportará sin problemas.

Historia

Desde que surgió la primera computadora personal en 1980, han existido muchos cambios en la forma como son transmitidos los datos entre una computadora y sus dispositivos periféricos, con el fin de mejorar la velocidad de transmisión y la eficiencia de los datos. El bus USB[2], el Serial ATA o la RDRAM son ejemplos del cambio desde una arquitectura en paralelo a un formato serie de alta velocidad. Los dispositivos anteriores fueron diseñados para permitir el máximo ancho de banda y ofrecer escalabilidad entre los nuevos modelos de computadoras que surgen año con año.

La compañía Intel presentó en el año de 1991, la especificación 1.0 del bus PCI. El grupo ***PCI Special Interest Group***, se encargo a partir de ese momento del desarrollo del las nuevas características del Bus, más tarde anunció la nueva revisión, la 2.0, en mayo de 1993.

En ese tiempo el bus mas popular era el Bus VESA (Local Bus (VL-bus o VLB)), esta tecnología fue creada por la asociación de estándares de Video (***Video Electronics Standards Association***), y consistía en un bus de 32 bits integrado a los dos conectores de los buses ISA clásicos, que operaba a una velocidad de 33 MHz, con un rendimiento muy significativo sobre el Bus ISA.

Curiosamente esta ultima característica fue, irónicamente, la razón principal de su hundimiento, ya que en esencia fue una extensión directa del bus externo del procesador 486, y funcionaba, por tanto, a la misma velocidad que el procesador por lo que de ahí viene su nombre de local bus. Se presentó el problema de la sobrecarga del bus: no se podían conectar demasiados dispositivos, pues podía ocurrir que las pérdidas de corriente hiciesen las señales completamente ininteligibles para los dispositivos o para el procesador. VESA aconsejaba no usar más de dos dispositivos si se trabajaba a 33 MHz, aunque se podía subir hasta tres si se añadían buffers intermedios.

Sin embargo, el problema más grave era que, al trabajar a la misma velocidad que el procesador, surgieron serios problemas, pues a mayor velocidad de los periféricos, más caros serán. En la práctica, se fabricaron muy pocos dispositivos capaces de trabajar a 40 o más MHz.

El bus PCI tiene varia ventajas sobre el bus anterior, para empezar, está aislado del bus de la CPU, pero permite a los periféricos acceder a la memoria del sistema. Además, también es capaz de actuar asíncronamente respecto del procesador, pudiendo trabajar a 25, 30 o 33 MHz. Esto significa que la velocidad del bus se mantiene constante aunque aumente la velocidad del procesador.

Además, el bus PCI permite 5 o más conectores, duplicando la oferta del bus local, y además, sin restricciones respecto a la velocidad del procesador.

Otra característica del PCI es la simplicidad de uso. El Plug and Play permite la configuración automática de los periféricos, sin que el usuario necesite asignar la IRQ, el DMA o los

puertos de entrada y salida. Además permite que varios periféricos compartan la misma interrupción, corrigiendo así los errores clásicos que tenían las microcomputadoras.

Una característica adicional es el "bus mastering", la cual permite a los dispositivos tomar control del bus y realizar transferencias entre ellos y otros dispositivos, o la memoria, sin que intervengan los procesos de la CPU, lo que reduce la latencia y la carga de trabajo del procesador.

Su introducción en los sistemas Pentium, junto con sus claros beneficios sobre sus rivales, ayudaron al PCI a ganar la guerra de los buses en 1994. Desde entonces, prácticamente todos los periféricos, desde controladores de disco duro y tarjetas de sonido hasta tarjetas de video, han sido fabricadas para este bus.

Las limitaciones del bus PCI

Con la aparición de los sistemas RAID, la Gigabit Ethernet y otros dispositivos de alta velocidad, los 133 MB/s del PCI se volvieron claramente insuficientes para manejar semejante cantidad de datos. Por eso los fabricantes de circuitos han buscado la manera de contrarrestar esta limitación.

Hasta 1997 [1], los datos de la tarjeta gráfica constituían el mayor porcentaje del tráfico del bus PCI. El Accelerated Graphics Port (AGP), presentado en el circuito 440 LX de Intel, tenía dos objetivos principalmente: aumentar el rendimiento gráfico y desplazar los datos gráficos fuera del bus PCI. Al realizarse las transferencias para los gráficos a través de otro «bus» (técnicamente, AGP no es un bus pues sólo permite un único dispositivo), el antes saturado PCI quedaba liberado para así poder atender mejor a los otros dispositivos.

Pese a todo, el AGP sólo fue un paso. El siguiente consistió en rediseñar el enlace entre el North Bridge y el South Bridge (los circuitos se dividen en estos dos componentes, manejando el primero la memoria y la parte gráfica, y el segundo los periféricos integrados en la placa madre). Los primeros circuitos, como los 440 de Intel, usaban un bus PCI para interconectar ambos elementos, con lo que el PCI no solo tenía que soportar el tráfico normal de los periféricos «en tarjetas», sino también el de los periféricos integrados en la placa madre y todo el tráfico de intercomunicación entre ambos circuitos.

Para aliviar esta situación, Intel, VIA y SiS sustituyeron el bus PCI entre ambos dispositivos por una conexión de alta velocidad. Hoy en día, gracias al bus Communications Streaming Architecture de Intel, integrado dentro del controlador de memoria de los chipsets i875/876, incluso la Gigabit Ethernet está fuera del bus PCI.

Sin embargo, aunque AGP, CSA, el AHA de Intel, el V-Link de VIA y el MuTIOL de SiS han conseguido reducir de forma bastante exitosa la carga del bus PCI, no son más que soluciones temporales.

El nuevo bus PCI Express

El bus PCI Express, antes conocido como 3GIO, está diseñado para reemplazar al PCI y cubrir las necesidades de interconexión para la próxima década. Está diseñado para soportar diversos segmentos del mercado, y como una arquitectura de E/S que unifique los equipos de escritorio, portátiles, servidores, estaciones de trabajo y dispositivos empotrados.

El objetivo principal es conseguir un menor costo que los dispositivos PCI, tanto en bajos como altos volúmenes de producción. Para ello emplea un bus serie en vez de paralelo, pues al requerir un menor número de pistas en las placas, se reducen los costes de diseño a la vez que se aumenta el rendimiento en cuanto a espacio consumido.

El PCI Express aparece ante el sistema operativo exactamente igual que el antiguo bus PCI, por lo que no habrá que hacer modificaciones sustanciales en ellos. Lo mismo se aplica a nivel de configuración y controladores de dispositivos, que serán compatibles con los actuales para PCI.

La escalabilidad en el rendimiento se consigue aumentando la frecuencia y añadiendo «rutas» al bus (habría varios buses serie funcionando en paralelo, cada uno independiente de los demás). El diseño está pensado para ofrecer una alta velocidad de transferencia en cada ruta, con baja sobrecarga y baja latencia. Permite, además, varios canales virtuales en cada ruta o enlace físico.

Finalmente, al ser una conexión punto a punto, permite que cada dispositivo tenga una conexión dedicada, evitando compartir el bus.

Otras características avanzadas son:

- Soporte de múltiples estructuras de datos.
- Capacidades avanzadas de gestión de energía.
- Conexión y desconexión «en caliente» de periféricos.

- Comprobación de errores de transmisión
- Capacidad de transferencia isócrona
- Protocolo dividido en capas y basado en envío de paquetes

Visto a alto nivel, un sistema PCI Express está formado por un sistema raíz (que estará bien en el North Bridge o en el South Bridge), uno o varios conmutadores (switches) y los dispositivos finales. La novedad aquí es el switch, el cual permite la comunicación punto a punto entre dispositivos finales, evitando enviar tráfico hasta el bridge si éste no supone problemas de coherencia en las cachés (por tratarse, por ejemplo, de transferencias a memoria).

En la parte inferior está la capa física. El enlace más simple para un sistema PCI Express consiste en dos señales diferenciales por corriente. Se incluye una señal de reloj usando el sistema 8/10b para conseguir altas velocidades de transferencia. La frecuencia inicial es de 2'5 Gb/s en cada sentido, y se espera que los avances en la tecnología del silicio permitan aumentarlo hasta los 10 Gb/s.

Una de las características más excitantes para los apasionados de la velocidad es la capacidad del PCI Express de aumentar la velocidad mediante el añadido de nuevos enlaces formando múltiples rutas paralelas. La capa física soporta anchos X1, X2, X4, X8, X12, X16 y X32. La transmisión sobre múltiples rutas es transparente al resto de las capas (ver **Figura 1**)

La capa de enlace es la encargada de garantizar la fiabilidad y la integridad de los datos para cada paquete enviado a través de un enlace PCI Express. Junto con un número de secuencia y un CRC, un protocolo

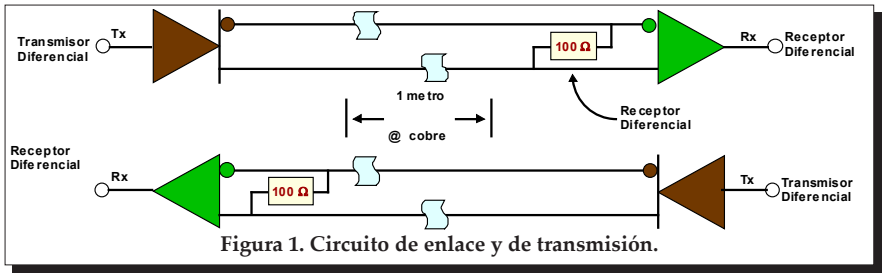

Figura 1. Circuito de enlace y de transmisión.

de control de flujo garantiza que los paquetes son transmitidos sólo cuando hay un buffer disponible para recibirlos en el otro extremo. Los paquetes corruptos se retransmiten automáticamente.

La capa de transacción crea los paquetes con las peticiones de la capa de software a la capa de enlace, implementándolas como transacciones. Cada paquete tiene un identificador único, soportando direccionamiento de 32 y 64 bits. Otros atributos extra incluyen «no-snoop», «relaxed ordering» y prioridad, y se usan para el enrutado y la calidad de servicio.

La capa de transacción se encarga de cuatro espacios de direccionamiento: memoria, I/O, configuración (estos tres ya existían en la especificación PCI) y el nuevo espacio Mensajes. Este reemplaza a ciertas señales en la especificación PCI 2.2 y elimina los «ciclos especiales» del viejo formato, lo que incluye las interrupciones, las peticiones de gestión de energía y el reinicio

Finalmente, la capa software constituye la clave para conseguir la compatibilidad software. La inicialización y el runtime no se han cambiado respecto al PCI debido a que se quiere que los sistemas operativos puedan usar PCI Express sin necesidad de modificarse. Los dispositivos son enumerados de forma que el sistema operativo pueda encontrarlos y asignarles recursos, mientras que el runtime reutiliza el modelo cargar/almacenar y memoria compartida del PCI. Sin embargo, queda por ver si realmente es necesaria la modificación, pues el «Soporte de PCI Express» es una de las características que se anuncian para Longhorn, el próximo Windows.

Las primeras implementaciones estás diseñadas para coexistir con los actuales conectores PCI. Así, un conector 1X encaja a continuación de un conector PCI, entre éste y el borde de la placa madre, de forma tal que se puede usar cualquiera de las dos tarjetas. Ver **Figura 2** en la siguiente página.

Otras novedades incluyen el separar la «caja principal» y los interfaces, y las «bahías de dispositivos», que permitirían la conexión y desconexión «en caliente» de tarjetas y otros periféricos PCI-Express.

Incluso se ha tenido en cuenta a los usuarios de dispositivos portátiles, con el nuevo estándar PCMCIA denominado NEWCARD, el cual define un formato en el que dos tarjetas NEWCARD, una junto a la otra, ocupan casi el mismo espacio que una tarjeta CardBus actual. Desgraciadamente, no está diseñado para soportar gráficos, por lo que las posibilidades de actualizar el sistema gráfico de un portátil siguen siendo prácticamente inexistentes. Pese a todo, no hay que perder de vista las cosas buenas: el nuevo bus permitirá la expansión de multitud de nuevas opciones, como comunicaciones inalámbricas, capturadoras de TV de alta calidad, etc.

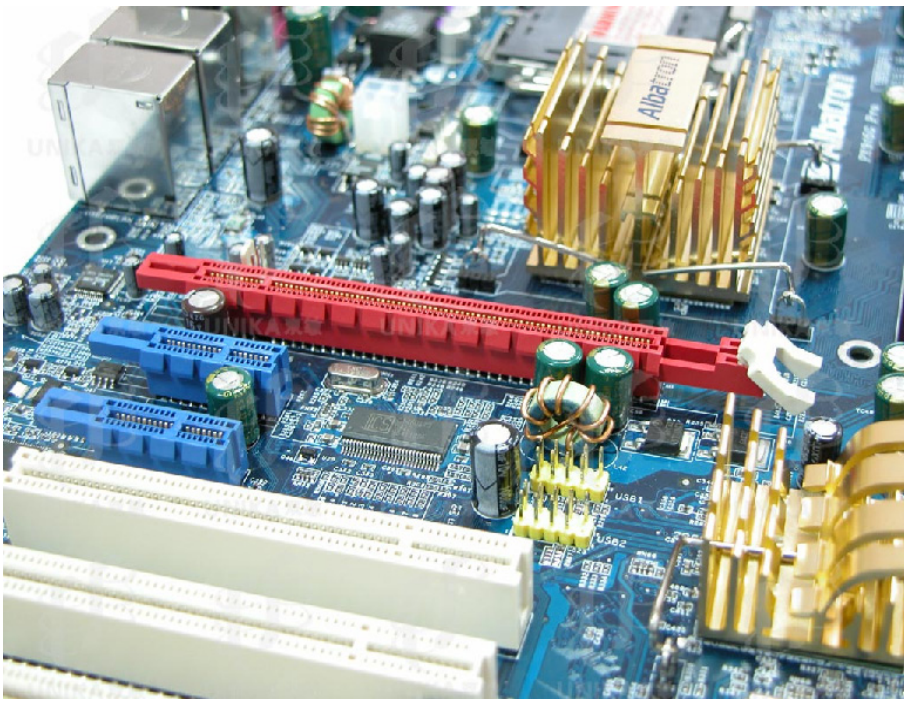

Figura 2. Coexistencia entre el Bus PCI y el PCI express. Conectores PCI, PCIex 1x y 16x.

Conclusiones

Las características antes mencionadas proporcionan una alternativa muy viable para el diseño de nuevas tarjetas en aplicaciones comunes, dada su gran velocidad. Con sus 200 MB por segundo en cada sentido para un conector X1, el PCI Express se alza como una solución económica en relación al ancho de banda por terminal, este conector X1 soporta todos los conectores (2x, 4x, 8x, 12x, 16x, and 32x) (Ver **Tabla 1**).

El chipset Grantsdale de Intel ofrecerá un enlace X16 para gráficos (lo que supone 4 Gigabytes por segundo en cada sentido), lo que duplica el ancho de banda de un AGP 8X. Esta capacidad permitirá cubrir las demandas para los próximos años.

Finalmente, la característica de ser una conexión punto a punto permitirá que cada dispositivo pueda tener una conexión dedicada, evitando compartir el bus.

Si el bus proporciona Soporte de múltiples estructuras de datos, capacidades avanzadas de gestión de energía, conexión y desconexión «en caliente», comprobación de errores de transmisión, capacidad de transferencia isócrona y protocolo dividido en capas y basado en envío de paquetes, entonces será posible realizar una amplía gama de aplicaciones electrónicas a través de este bus.

Referencias

[1] Anderson, Don; Budruk, Ravi. *PCI Express System Architecture*. Editorial Tom Shanley.

[2] Wilen, Adam; Schade, Justin P.; Thornburg, Ron . *Introduction to PCI Express: A Hardware and Software Developer's Guide*. Intel Press.

PCI-Express 1x PinOut				
Pin	Conector lado B		Conector lado A	
#	Nombre	Descripción	Nombre	Descripción
1	+12v	+12 voltaje	PRSNT#1	Detector de presencia de Hot Plug
2	+12v	+12 voltaje	+12v	+12 voltaje
3	RSVD	Reservado	+12v	+12 voltaje
4	GND	Tierra	GND	Tierra
5	SMCLK	Reloj BSMBus	JTAG2	TCK
6	SMDAT	Datos SMBus	JTAG3	TDI
7	GND	Tierra	JTAG4	TDO
8	+3.3v	+3.3 voltaje	JTAG5	TMS
9	JTAG1	+TRST#	+3.3v	+3.3 voltaje
10	3.3Vaux	+3.3 voltaje	+3.3v	+3.3 voltaje
11	WAKE#	Liga de Reactivación	PWRGD	Power Good
Llave Mecanica				
12	RSVD	Reservado	GND	Tierra
13	GND	Tierra	REFCLK+	Reloj de Referencia Par diferencial
14	HSOp(0)	Linea de Transmisión 0, Par Diferencial	REFCLK-	
15	HSOn(0)		GND	Tierra
16	GND	GTierra	HSIp(0)	Linea de Transmisión 0, Par Diferencial
17	PRSNT#2	Detector Hotplug	HSIn(0)	
18	GND	Tierra	GND	Tierra

Tabla 1. Distribución de Terminales para PCI-EXPRESS Conector X1.

Oxímetro de Pulso Basado en una Palm Parte II: Diseño del Hardware

M. en C. Israel Rivera Zárate.
M. en C. Juan Carlos Herrera Lozada
Profesores del CIDETEC-IPN.
Victor Jalil Ochoa.
Estudiante UPIICSA-IPN.

En este artículo se establece el diseño de los circuitos que constituyen el oxímetro de pulso tomando como base que, a una longitud de onda de 660 nm, la luz roja visible se absorbe más por la HbR (hemoglobina reducida o desoxigenda) que por la HbO_2 (hemoglobina oxigenada), y a una longitud de onda de 940 nm, la luz infrarroja se absorbe más por la HbO_2 que por la HbR. Estas dos luces de diferente longitud de onda (roja e infrarroja) se hacen pasar a través del árbol arterial y el porcentaje de HbO_2 y HbR son determinados por la medición de la proporción de luz roja e infrarroja transmitida hasta un foto-detector, entonces; la intensidad de la luz se reducirá logarítmicamente con la longitud de la trayectoria conforme lo establece la ley de Beer-Lambert [1].Ver **Figura 1**.

Diseño del Oxímetro

En la **Figura 2** se muestra el diagrama a bloques propuesto para el diseño del oxímetro de pulso [2].

Para llevar a cabo las pruebas ya sea en el dedo de la mano como en el lóbulo de la oreja de forma no invasiva se necesitan LED y fotodetectores miniatura. Los detectores deben ser de alta sensibilidad, ya que deben ser capaces de registrar la débil emisión que logra atravesar los tejidos.

Este problema puede solucionarse con LED de propósito especial, que han sido fabricados con un sistema interno de lentes que permiten una alta intensidad lumínica de salida; adicionalmente han sido diseñados para operarse en esquemas de pulsos de corriente donde es posible manejar una potencia promedio elevada.

Si se aplican pulsos en ambas fuentes de luz se puede emplear un único fotodetector. Dado que la frecuencia de 1 KHz es suficientemente mayor a la frecuencia del pulso arterial, se elige ésta, así como anchos de pulso de 50 ¼s. Ver **Figura 3**.

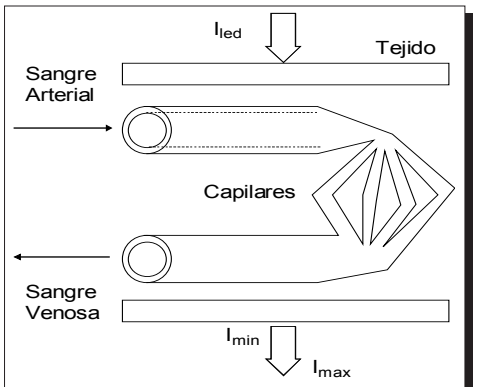

Figura 1. Modelo de la situación del punto de prueba para la medición de la SaO_2 (saturación de oxígeno arterial).

En este modo de operación se pueden obtener salidas de alta intensidad luminosa empleando corrientes de hasta 1 A debido al ciclo de trabajo reducido. La luz transmitida que se detecta es posteriormente amplificada y convertida a voltaje empleando circuitos operacionales configurados como convertidores

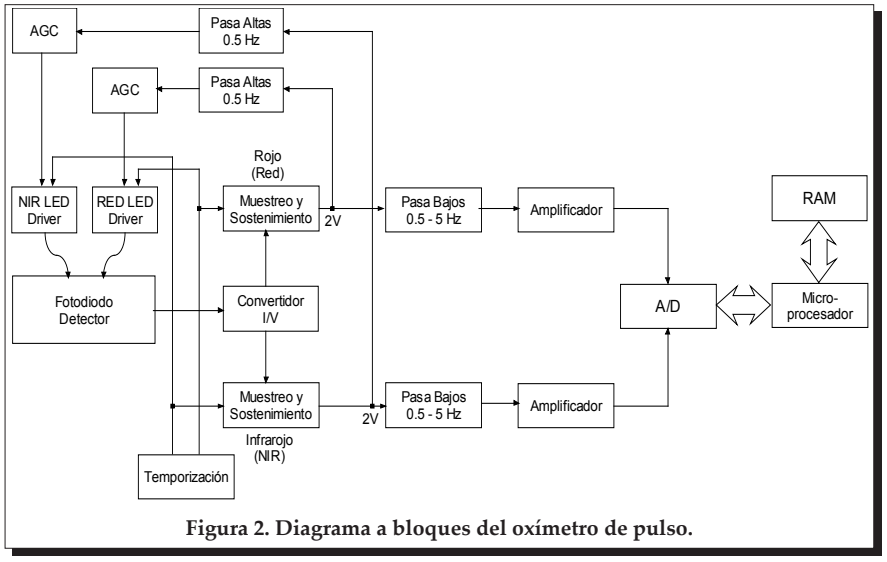

Figura 2. Diagrama a bloques del oxímetro de pulso.

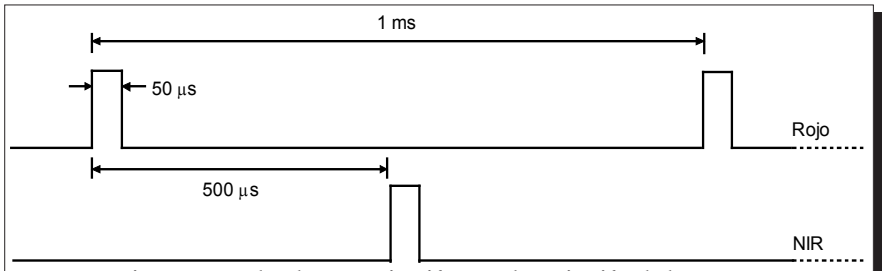

Figura 3. Señales de temporización para la excitación de los LEDs.

de corriente a voltaje. En este punto en el circuito, la señal es alimentada hacia dos secciones idénticas, correspondientes a cada una de las longitudes de onda.

Debido a que la información se manifiesta en forma de pulso, se requiere un circuito **sample and hold** (muestreo y sostenimiento) para reconstruir las formas de onda en cada una de las longitudes. Así mismo, los circuitos de temporización que controlan los circuitos de excitación de ambos LED pueden ser usados en la sección de circuitos **sample and hold**.

La salida de estos circuitos se conduce posteriormente a una sección de filtrado pasabanda diseñada para operar en las frecuencias de corte de 0.5 Hz a 5 Hz, destinados para eliminar principalmente la componente de cd así como ruido de alta frecuencia.

Las señales resultantes representan por tanto la información cardíaca síncrona en las formas de onda, la cual es posteriormente amplificada y convertida a formato digital para un posterior análisis en el microprocesador [3]. Ver **Figura 4**.

Se puede apreciar en el diagrama a bloques de la **figura 2** que la salida de cada circuito **sample and hold** es luego dirigida hacia circuitos de filtrado pasa bajos. Esta es la primera etapa del circuito de control automático de ganancia (AGC), el cual ajusta la intensidad de la luz del LED correspondiente de tal forma que el nivel de CD permanece en el mismo valor (2V) sin considerar el grosor o las características de la piel del dedo del paciente.

Existen dos razones importantes por las cuales se ha decidido el empleo del circuito AGC: primero, que la amplitud de la señal CA (la cual puede variar entre el 0.1% y el 2% del total de la señal) se encuentra dentro de un rango predefinido y esto hace que el amplificador que sigue posterior al filtro pasa banda sea fácil de diseñar. Segundo, la componente CD de la luz transmitida tanto roja como infrarroja, pueden establecerse al mismo valor en cada caso (2V).

Por lo tanto, puede emplearse una tabla almacenada en memoria y ser accesada por el microprocesador convenientemente.

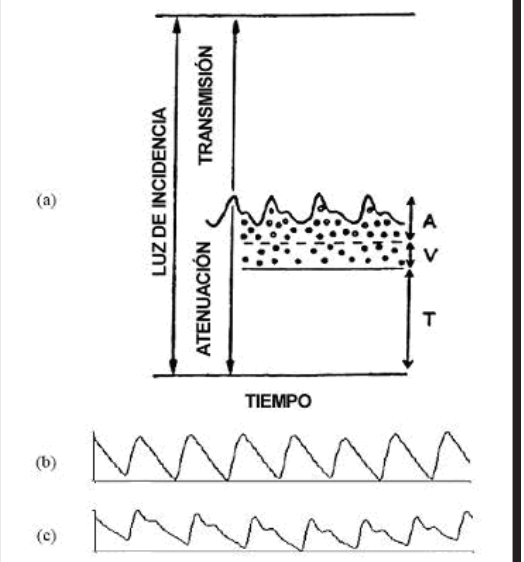

Figura 4. a) Transmisión de la luz a través de un dedo cuando la atenuación es debida a la (A) sangre arterial, (V) sangre venosa y (T) tejidos. b) y c) muestran las señales pulsatiles típicas detectadas.

Fuente de Corriente Continua para el Manejo de los LED

Se propone el circuito de la **Figura 5** para llevar acabo este proceso.

Se combina un amplificador operacional con un transistor bipolar, donde mediante la retroalimentación negativa se forza que el voltaje de salida sea igual al de entrada y que la corriente del LED sea por lo tanto Vin/R1

Si se emplean FET en vez de BJT se evita la corriente de fuga en

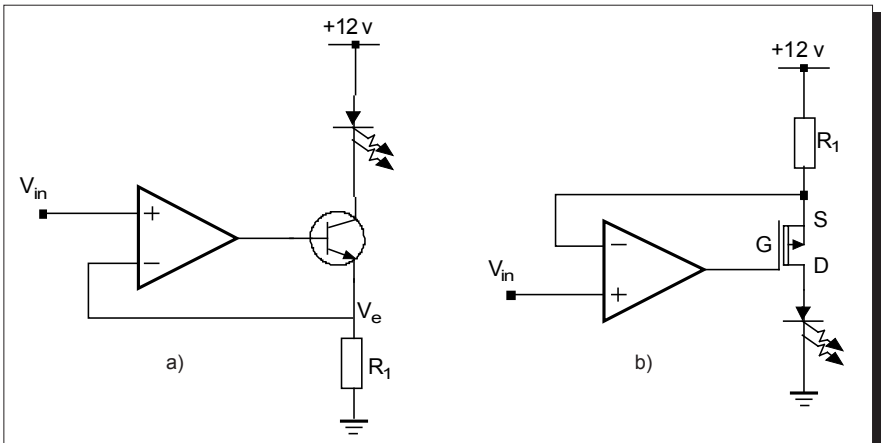

Figura 5. Dos circuitos posibles para la fuente de corriente continua de los LEDs.

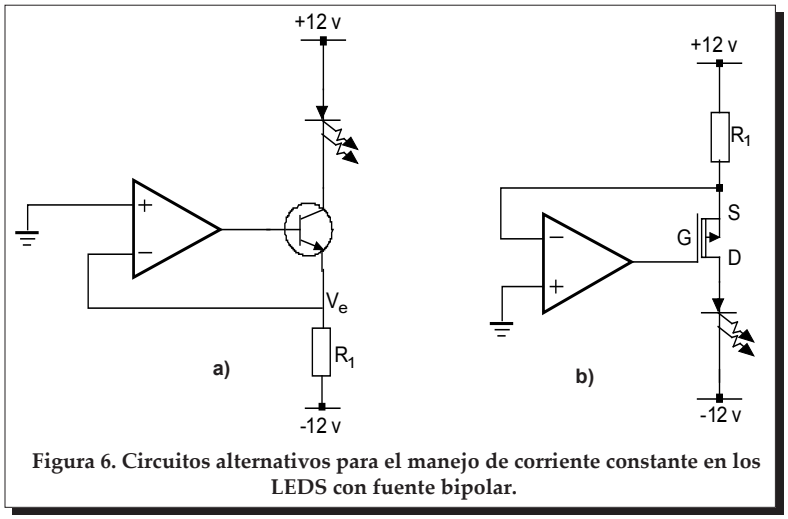

Figura 6. Circuitos alternativos para el manejo de corriente constante en los LEDS con fuente bipolar.

la base del transistor debido a que los FET no permiten la circulación de la corriente de compuerta. Cabe mencionar sin embargo que la corriente de carga queda limitada a $I_{DS\ (on)}$ del FET.

Si se emplea una fuente bipolar los circuitos de la **Figura 6** pueden simplificar significativamente el diseño donde en ambos casos la corriente del LED será 12 V / R1.

Circuito de Temporización

En esta aplicación, la precisión respecto al tiempo no es un factor importante, por lo tanto el circuito de temporización puede implementarse alrededor de un temporizador integrado 555 . Empleando la hoja de datos del integrado se determinan los siguientes valores: C = 22nF, Ra = 56KΩ y Rb = 3.3 KΩ, lo que genera un pulso de 50µs aproximadamente cada milisegundo. Ver **Figura 7**.

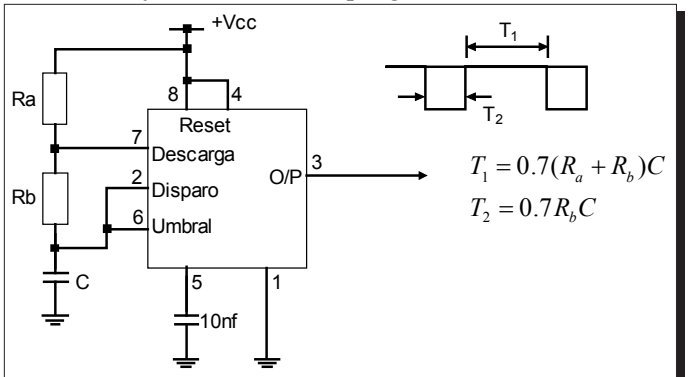

Figura 7. Circuito de generación de los pulsos del oxímetro.

Pulsos de Control

La salida de los LED se controla por la aplicación de pulsos de control mediante un MOSFET de acrecentamiento de canal n, como se muestra en la **Figura 8**. Los pulsos provienen del circuito temporizador 555 y son llevados a la compuerta del transistor. El FET necesita ser de acrecentamiento para poderse encontrar completamente encendido y apagado exclusivamente por los pulsos aplicados en su compuerta, además de soportar el máximo de corriente que ha de fluir a través de los LED.

Circuito Receptor

El elemento receptor óptico más simple es el fotodiodo. Los fotodiodos operan típicamente con polarización inversa aplicada en la unión p-n (modo fotoconductivo). Cuando la luz cae en la región de la unión, un par electrón-hueco es creado bajo la influencia del campo eléctrico creado en la unión. El hueco se mueve hacia el material tipo p y el electrón hacia el material tipo n. El resultado es que la iluminación es vista como un incremento en la corriente inversa.

Para propósitos de amplificación de la señal, la fotocorriente debe ser transformada en voltaje con una moderada impedancia de salida. Para llevar acabo esto, se propone el circuito de la **Figura 9**, el cual presenta un amplificador operacional implementado como convertidor de corriente a voltaje. Debido a la alta resistencia ocasionada por la polarización inversa del fotodiodo, el circuito operacional debe ser de tipo

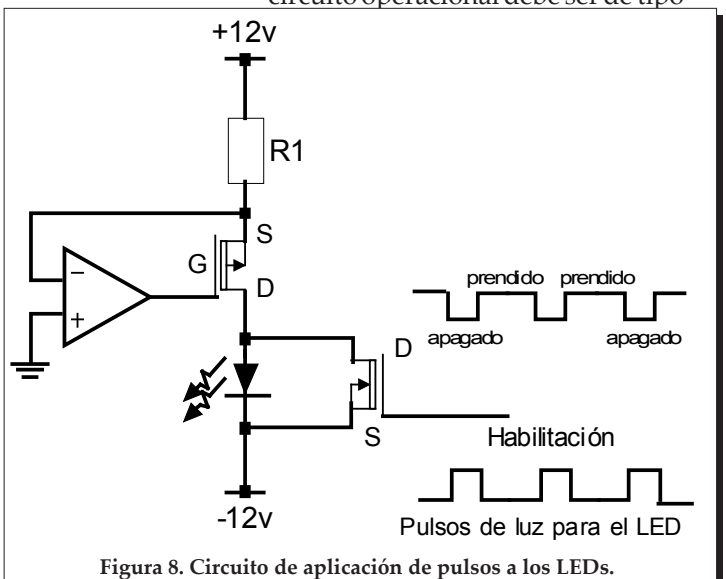

Figura 8. Circuito de aplicación de pulsos a los LEDs.

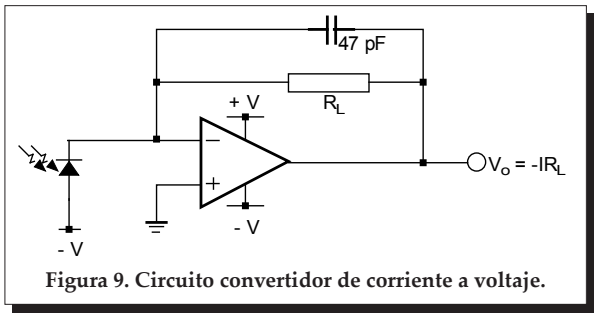

Figura 9. Circuito convertidor de corriente a voltaje.

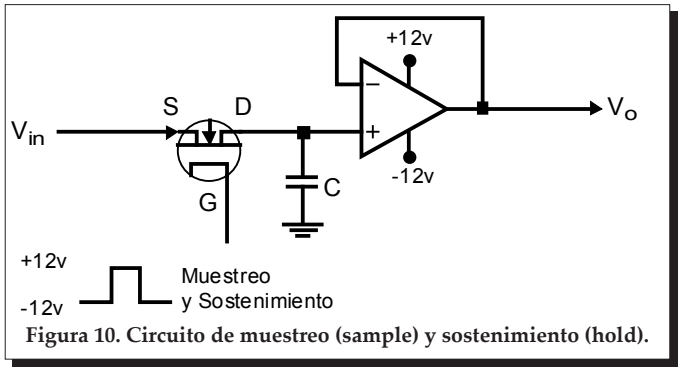

Figura 10. Circuito de muestreo (sample) y sostenimiento (hold).

valor de C debe ser lo suficientemente grande para minimizar la caída de voltaje durante el periodo de sostenimiento. Así mismo, debe considerarse que la resistencia del FET cuando está encendido (de algunas decenas de ohms típicamente) forma un filtro pasa bajos en combinación con C y por lo tanto, C debe ser suficientemente pequeño para poder seguir con suficiente precisión el paso de señales de alta velocidad.

Control Automático de Ganancia

La salida del circuito sample and hold, como se indicó en el diagrama general de la **Figura 2**, es alimentada a un filtro pasabanda que extrae la señal pulsátil previo a su posterior amplificación y análisis. La misma salida es también llevada a un filtro pasa bajo con una frecuencia de corte de 0.1 Hz, que extrae el valor deCD de la señal transmitida. Existen diferentes formas de implementar la función AGC. Una de las formas más simple es alimentar la señal de CD a un amplificador diferencial cuya otra entrada es un voltaje de referencia (un diodo zener por ejemplo). Las diferencias de estos dos voltajes se usa posteriormente para generar el voltaje V_{in} en la **Figura 5**.

Circuitos de Filtrado

La salida de los circuitos sample and hold se conduce posteriormente a una sección de filtrado pasabanda diseñado para operar en las frecuencias de corte de 0.5 Hz a 5 Hz, destinados para eliminar principalmente la componente de CD así como ruido de alta frecuencia. Las señales resultantes representan por tanto la información cardíaca síncrona, forma de onda que es posteriormente amplificada y convertida a formato digital para un posterior análisis en el microprocesador. Ver **Figura 11**.

Bibliografía

[1] Kelleher, J. F. "Pulse oximetry". En *Journal of Clinical Monitoring*. 1989, Vol. 5, pp. 37–62.

[2] Severinghaus, J. W.; Kelleher, J. F. "Recent developments in pulse oximetry". En *Anesthesiology*. 1992, Vol. 76, pp. 1018–1038.

[3] Moyle, J. T. B. *Pulse Oximetry*. 1st ed. London, U.K.: BMJ, 1994.

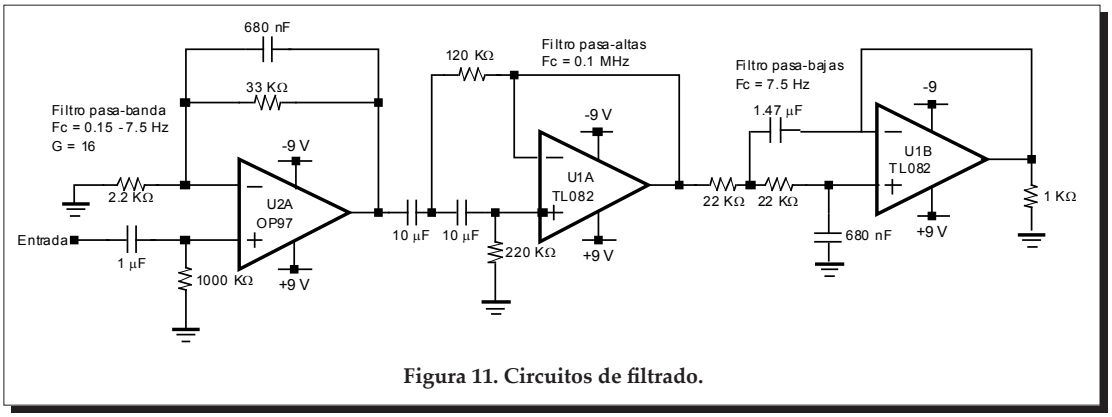

Figura 11. Circuitos de filtrado.

Introducción a la Seguridad con IP Seguro en Internet (IPSec)

M. en C. Mauricio Olguín Carbajal
M. en C. Israel Rivera Zárate
Profesores del CIDETEC-IPN
Adrián Martínez Ramírez
Erika Rocío Barrón
Luis Arturo Rivera Quimby
Fausto Israel Padilla Godínez.
Alumnos de Especialidad de Redes de Computadoras
UNITEC Campus Marina Nacional

Antes de que surgiera IPSec, las soluciones existentes en el mercado para conseguir la transmisión segura de datos en Internet por medio del protocolo IP eran implementaciones dependientes del fabricante que las promulgaba; dicho de otra manera, eran soluciones aisladas. Aunque suelen funcionar bien, tienen el inconveniente de que los protocolos de seguridad de distintos fabricantes son incompatibles entre sí o difíciles de conciliar; si una empresa y todos los agentes con los que se relaciona utilizan la misma solución tecnológica no habrá problema. Sin embargo, lo habitual es que exista cierta heterogeneidad en cuanto a los equipos de comunicaciones, sistemas operativos, medios de transporte de datos, etc…, lo que hace que el uso de una única solución de seguridad propietaria sea poco práctico.

IPSec fue publicado en el año de 1994 en el documento RFC 1636, logrando con su creación anular esa incompatibilidad, puesto que está basado en estándares siendo esto una ventaja a su favor. Su diseño es totalmente independiente del sistema operativo, de la plataforma computacional y de las tecnologías subyacentes empleadas, por lo que su interoperabilidad está asegurada (**Figura 1**). Además, es lo suficientemente abierto como para incorporar en el futuro los avances tecnológicos y criptográficos que sean necesarios. Tan es así que se incluye como parte integral en IPv6.

Introducción

Si bien es cierto en la actualidad se vive en un mundo en el que impera la necesidad de un constante intercambio de información vía Internet e Intranet, muchas veces dicha información debe ser confidencial, por lo que se debe proveer un medio que ofrezca seguridad para su transporte. Es importante señalar que cuando se menciona la palabra "seguro" no se habla únicamente a la confidencialidad de la comunicación, sino también se hace referencia a la integridad de los datos que para muchas compañías y entornos de negocios puede ser un requisito mucho más crítico que la confidencialidad.

Es por eso que surge IPSec el cuál cuenta con innumerables características así como cualidades que satisfacen las necesidades de seguridad, confidencialidad y autenticidad de la información.

IPSec proporciona servicios de seguridad a la capa IP y a todos los protocolos superiores basados en IP (TCP y UDP entre otros) y aborda

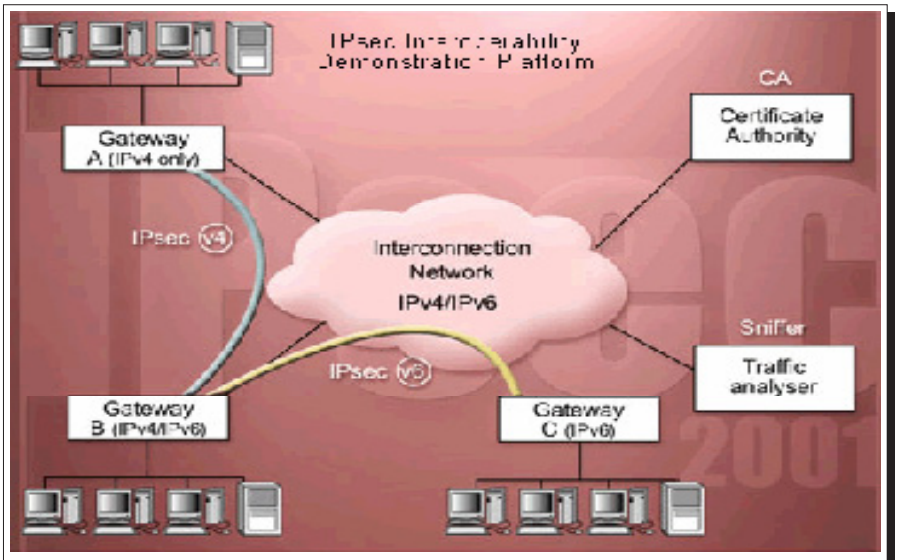

Figura 1. Plataforma IPsec.

las carencias en cuanto a seguridad del Protocolo IP. Dichas carencias son muy graves, tal como se ha constatado en los últimos años, y afectan a la infraestructura misma de las redes IP.

El Protocolo IPSec

IPSec es un estándar de gran utilidad que ofrece servicios de seguridad en redes IP de cualquier índole, el cuál está formado por un conjunto de estándares del IETF (Internet Engineering Task Force) que conjuntamente proporcionan servicios de seguridad en la capa IP de las comunicaciones entre sistemas electrónicos, y por añadidura a todos los protocolos de niveles superiores que están basados en IP (TCP, UDP, ICMP, y otros).

Hoy en día los protocolos basados en IP tienen una presencia universal en las redes telemáticas. Desde cualquier red local común hasta la propia Internet están basados en este protocolo para su funcionamiento. Uno de los problemas que enfrenta IP es la dificultad para asegurar las comunicaciones. Con "asegurar" no sólo se refiere a seguridad de acceso a sistemas (que es lo primero en que se suele pensar), sino también a establecer medios de comunicación que puedan evitar la intercepción de la información y su manipulación, así como asegurar la confidencialidad de los datos intercambiados.

Otra cuestión importante es que es capaz de proteger aplicaciones y dispositivos que, en realidad, ni siquiera conozcan la existencia de IPSec, ya que la protección se genera en las capas OSI inferiores.

Aunque muchos administradores de sistemas todavía no lo utilizan, en la actualidad IPSec está presente en la práctica en la totalidad de los equipos de comunicaciones y sistemas operativos modernos como Windows 2000/XP/2003, y otros como Solaris 10, Linux o MacOS (**Figura 2**).

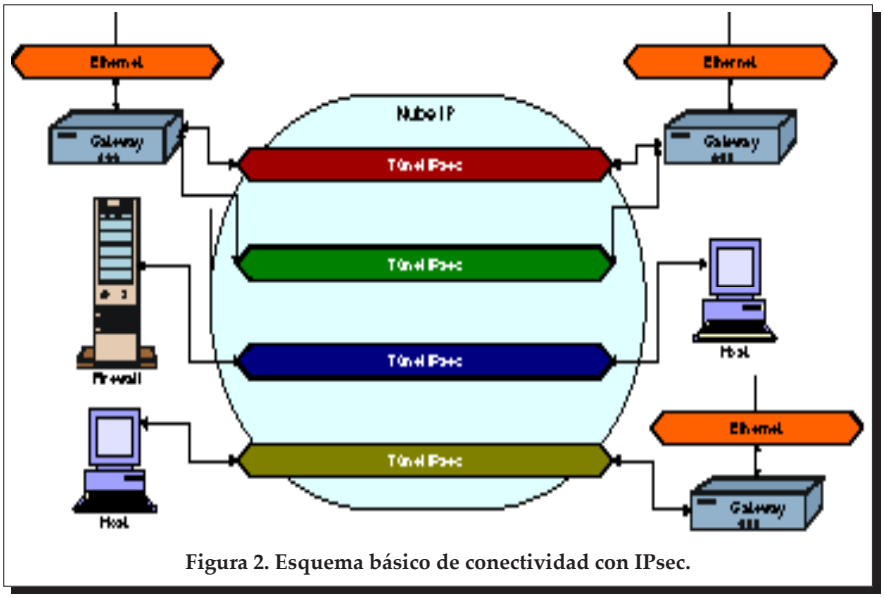

Figura 2. Esquema básico de conectividad con IPsec.

Servicios de Seguridad Ofrecidos por IPSec

IPsec tiene la capacidad de encargarse de tareas que tienen que ver con la seguridad, asumiendo la administración de todas ellas o sólo algunas en función de las necesidades que existen en cada nodo; por ejemplo:

• **Control de acceso:** Aquí se hace referencia a dos puntos importantes; autenticación y autorización. La autenticación sirve para asegurar de que los interlocutores son en verdad quienes dicen ser y que están ubicados en donde deben estar.

Por otra parte, la autorización implica que, aún cuando un interlocutor se haya autenticado éste pueda tener acceso a los recursos de red IP que solicita. Ejemplificando lo anterior, pudiera ser que un empleado cuenta con una credencial de acceso a la empresa donde labora, pero está credencial no es valida para ciertas zonas que se encuentran restringidas para dicho empleado. Lo que ofrece es algo similar a un cortafuegos con filtro de paquetes pudiéndose considerar para crearlo cuestiones como el protocolo, los puertos, las direcciones IP de origen y destino y otros parámetros técnicos de los paquetes.

• **Confidencialidad:** ofrece cifrado del tráfico de datos y ocultación del tipo de comunicación. Dicho en otras palabras, los datos son transformados carácter por carácter con la finalidad de que no puedan ser interpretados por algún intruso. Esto asegura que, aunque alguien intercepte las comunicaciones entre dos nodos, no va a ser capaz de descifrar su contenido, entendiéndose por cifrado a la transformación carácter por carácter o bit por bit.

• **Autenticación e integridad:** si bien ocultar la información es importante, la comunicación no será muy efectiva si se permite que cualquiera pueda interceptar los paquetes y modificarlos, o bien hacerlos llegar haciéndose pasar por quien no es. IPSec proporciona la capacidad de certificar que los que intervienen en la comunicación son quiénes dicen ser. También es responsable de asegurar la integridad de los datos, es decir,

que si alguien los altera durante el tránsito entre dos nodos los cambios serán detectados y no se admitirá el engaño.

• **Detección de repeticiones:** existe un tipo de ataque que permite a un usuario malintencionado capturar paquetes (aunque estén correctamente autenticados) y reenviarlos al destinatario varias veces con el objeto de que los acepte y por lo tanto el mensaje transmitido se ofusque. Para evitar este tipo de intentos, IPSec incluye un sistema de detección de paquetes repetidos. Para conseguirlo hace uso de un número de secuencia que se agrega en el encabezado de los paquetes y que va protegido por el sistema de integridad de modo que los intrusos no tienen forma de poderlo modificar.

Protocolos Básicos de IPSec

IPSec está constituido por un conjunto de estándares de criptografía que lo dotan de sus especiales características. Utiliza algoritmos de clave pública como RSA, algoritmos de resumen digital (SHA1, MD5), certificados digitales X509 y algoritmos de cifrado de clave simétrica como DES, IDEA, Blowfish o AES. Todos estos elementos forman parte de IPSec como pequeñas piezas que se pueden conectar sin interferir unas en otras. Esto hace que sea posible utilizar todo tipo de algoritmos existentes en la actualidad o en el futuro. Sin embargo, y en aras de conseguir la mayor interoperabilidad posible, la implementación mínima de IPSec debe ofrecer ciertos elementos estándar que siempre se deben soportar. En concreto, siempre estarán disponibles los algoritmos MD5 y SHA-1 para cálculo de huellas digitales (Hashes) y los algoritmos DES y triple DES para cifrado simétrico con clave privada.

El funcionamiento de IPSec está basado en la existencia de dos componentes de suma importancia:

• **El protocolo de gestión de claves llamado IKE (*Internet Key Exchange*),** que es el encargado de hacer las negociaciones de todos los parámetros necesarios de conexión y seguridad, incluyendo como su propio nombre indica, las claves utilizadas para el cifrado de datos.

• **El protocolo de seguridad**. Éste protocolo tiene como función principal proteger el tráfico de datos en IP. El estándar define dos protocolos de seguridad que pueden ser utilizados con IPSec: ***Authentication Header* (AH)** y ***Encapsulating Security Payload* (ESP)**.

A continuación se da una breve explicación de ambos. Es bueno señalar que el segundo de ellos es más completo y ofrece más funcionalidad que el primero, pero AH es una buena elección en ocasiones si no se necesita confidencialidad.

El Protocolo de Gestión de Claves

La distribución de claves de un modo seguro es fundamental para que IPSec funcione, puesto que si las claves se ven comprometidas toda la seguridad de las comunicaciones se vendría abajo y estaría en manos de cualquier intruso. Es por eso que la robustez del mecanismo de intercambio de claves es la pieza clave del sistema.

Antes de estudiar el protocolo IKE es necesario aclarar un concepto importante en IPSec: la asociación de seguridad **(SA, *Security Association*)**. Una SA es un canal de comunicación que conecta dos nodos y a través del cual se mueven en un único sentido los datos protegidos criptográficamente. Es importante resaltar el hecho de que una SA sólo gestiona datos en una dirección, es decir, de un nodo al otro con el que se comunica pero no a la inversa. Esto implica que cuando se crea una conexión IPSec protegida entre dos nodos en realidad lo que se obtienen son dos SA, una en cada sentido.

El protocolo de control IKE se encarga de intercambiar claves entre los nodos, acordar entre éstos qué algoritmos de cifrado y parámetros de control se utilizarán, y de establecer las asociaciones de seguridad (una en cada sentido). IKE no está pensado de manera específica para IPSec, sino que es un protocolo estándar de gestión de claves que se utiliza en otros ámbitos.

El proceso de negociación de la comunicación entre los dos nodos mediante IKE se lleva a cabo en dos fases. En la primera de ellas se establece un canal seguro y se autentican entre sí ambos nodos. Este canal consigue mediante un algoritmo de cifrado simétrico y un algoritmo de autenticación de mensajes. La autenticación mutua se consigue de dos formas posibles:

1. Utilización de secreto compartido. En este caso ambos nodos que se intentan comunicar deben conocer una determinada cadena de caracteres que constituye el secreto común. Mediante el uso de funciones de resumen digital (*hash*) cada nodo demuestra al otro que conoce el secreto sin tener que transmitir éste por la red. Para reforzar este mecanismo de autenticación tan débil cada par de nodos IPSec debe compartir un secreto diferente. Debido a ello, en configuraciones grandes que impliquen un número elevado de nodos, la gestión de claves con este sistema es inviable y hay que usar métodos de autenticación más robustos.

2. Utilización de certificados digitales. El uso de certificados X509v3 permite distribuir de forma segura la clave pública de cualquier nodo y solventa el problema del método anterior cuando entran en juego muchos nodos que se comunican de forma segura. Utilizando criptografía de clave pública y disponiendo de un certificado digital en cada nodo es posible comprobar la identidad de cualquiera de ellos gracias al par de claves pública / privada. La desventaja de este método es que se necesita disponer de una infraestructura de clave pública (PKI) que lo soporte, aunque no es un problema excesivamente complicado de solventar.

La segunda fase de la negociación en la comunicación es la que se encarga, tras obtener la autenticación y el canal seguro IKE, de los parámetros de seguridad específicos que se van a utilizar durante el resto de la comunicación (recuerde que IKE es un protocolo de inicio de sesión genérico que protege el establecimiento de la comunicación con el protocolo subyacente que protege, en este caso IPSec). El nodo que ha iniciado la comunicación informa al otro de todas las opciones de comunicación que tenga disponibles (algoritmos de cifrado, parámetros de éstos, etc…), con la prioridad que se haya establecido. Este último, el receptor, aceptará automáticamente la primera de las opciones ofrecidas por el emisor que coincida con las que tiene disponibles. Con esto queda establecida la sesión IPSec.

Protocolo de Seguridad AH

El protocolo de encabezado de autenticación (AH), es utilizado cuando lo único que se necesita es garantizar la autenticidad y la integridad de los paquetes IP que se intercambian en la comunicación. Es decir, asegura al nodo receptor que la información que está recibiendo procede del origen esperado y que además ésta no ha sido alterada en modo alguno durante el tránsito. Sin embargo, este protocolo no establece mecanismos para asegurar la confidencialidad de los datos, que pueden ser leídos en claro por cualquiera que los intercepte. En determinados casos puede ser una situación tolerable siendo suficiente la integridad y autenticación del origen. De este modo no se carga innecesariamente a los equipos que utilizan IPSec añadiendo un cifrado de datos si no se necesita.

Como su propio nombre deja entrever, AH basa su funcionamiento en la existencia de una cabecera de autenticación que se inserta entre la cabecera IP estándar y los datos transportados, que pueden ser TCP, UDP, etc... Su funcionamiento es, en realidad, bastante sencillo, utilizando un algoritmo de autenticación de mensajes. Lo que se hace es calcular la huella digital o hash a una combinación de una clave más el mensaje que se transmite. Esta huella digital identifica de manera única y simultánea tanto al mensaje como al emisor, ya que éste es el único, aparte del receptor, que conoce la clave utilizada. Esta clave se acuerda durante el protocolo de control IKE.

El resumen digital o extracto obtenido se incluye en la cabecera de autenticación que se transmite junto a la información. El receptor repite el cálculo en el otro extremo de la comunicación ya que tiene los elementos suficientes para hacerlo (la clave acordada durante IKE y el mensaje que se transmite en claro). Si el extracto (o huella digital) obtenido coincide con el de la cabecera de autenticación significa que ni el paquete ni el contenido han sido modificados durante el tránsito, ya que la única forma de obtener ese resultado es usando ambos elementos, y la clave sólo la conocen los dos nodos que intervienen en la comunicación.

Protocolo de Seguridad ESP

El protocolo ESP (*Encapsulating Security Payload*) ofrece la parte de la seguridad que le falta al protocolo AH: la confidencialidad. Para ello durante IKE se acuerda el modo en que se van a cifrar los datos que se transmitirán y de qué manera se incluye esta información dentro de los paquetes que se comunican. Como característica adicional ESP puede incorporar servicios de integridad y autenticación de origen, usando una técnica muy similar a la del protocolo AH.

Como es obvio, se trata de un protocolo mucho más complejo que el anterior. Encapsula los paquetes IP a transmitir utilizando una cabecera IP propia bastante compleja y una cola. Estos elementos encierran a los datos transmitidos, que se encuentran cifrados en su interior.

La confidencialidad de la información en ESP se obtiene usando un algoritmo de cifrado simétrico (que son menos costosos que los de clave pública). Lo más habitual es que se utilice un algoritmo de cifrado en bloque como DES o triple-DES, por lo que el mensaje a cifrar tiene que ser múltiplo del tamaño de dicho bloque. Este hecho obliga a veces a rellenar el mensaje para que se ajuste a esta condición, lo cual se aprovecha además para ocultar su longitud real antes del cifrado, haciendo más difícil todavía el análisis del tráfico. El proceso es similar al anterior, ambos nodos que se comunican conocen una clave que han acordado previamente. Ésta se utiliza como la clave para el algoritmo de cifrado de los datos antes de enviarlos, y también se usa para crear la cabecera de autentica-

ción. En el extremo del receptor la misma clave acordada se usa para descifrar el paquete encriptado y para comprobar la cabecera.

Es bueno recalcar que todo el sistema se vendría abajo si no existiera forma de intercambiar las claves de modo seguro. Todos los componentes de IPSec trabajan en conjunto para obtener el resultado final de alta seguridad, aunque cada uno de ellos tiene una función independiente que incluso puede ser reutilizada en otros contextos.

Modos de Funcionamiento de IPSec

Los protocolos de seguridad analizados proporcionan dos modos de funcionamiento que se pueden escoger tanto para AH como en ESP.

1. Modo transporte. Este modo de funcionamiento permite la comunicación punto a punto entre los nodos que se quieren relacionar con IPSec. Se utiliza cuando ambos extremos son capaces de utilizar directamente el protocolo IPSec.

2. Modo túnel. Este modo se utiliza cuando alguno de los dispositivos que se comunican (uno de ellos o ambos) no es el encargado de realizar las funciones de IPSec. Este es el modo de funcionamiento más habitual cuando se usan dispositivos de encaminamiento que aíslan una red privada de una pública, centralizando todo el proceso de tráfico IPSec en un único punto. De este modo, por ejemplo, los equipos internos de una red local y sus aplicaciones no tienen porqué implementar ni entender IPSec. Se comunican normalmente (sin protección alguna) con el nodo que procesa IPSec y es éste el que se encarga de realizar las funciones por todos ellos comunicándose con otro dispositivo IPSec en el otro extremo. De este modo se protegen las direcciones privadas, se centraliza la administración del protocolo de seguridad en un único punto, y se puede utilizar IPSec en sistemas que, en un principio, no estaban preparados para utilizarlo. Una de las principales aplicaciones del modo túnel es establecer de manera sencilla y barata Redes Privadas Virtuales (o VPN) a través de redes públicas. Ello permite intercomunicar entre sí, a través de Internet, redes locales o equipos aislados con las mismas garantías de seguridad que si estuviesen en una red privada, aunque usen internamente direcciones IP no válidas en la Red.

Aplicaciones de IPSec en el Día a Día

Una vez que se ha leído este documento se pudiera pensar en una multitud de aplicaciones prácticas para IPSec. Entre ellas se encuentran:

• **Control de acceso y autorización de comunicaciones.** Gracias a las capacidades de filtrado de IPSec se puede decidir exactamente cómo se realizan las comunicaciones a través de IP con cualquiera de los protocolos de alto nivel. Si además los protocolos son TCP o UDP es posible controlar qué se hace con el tráfico en función de las direcciones IP y los puertos de origen y destino. Se obtiene casi las mismas capacidades que las que ofrece básicamente un cortafuegos (salvando las distancias).

• **Conexión segura de oficinas y creación de intranets distribuidas.** Con IPSec se puede hacer que las distintas sucursales y oficinas de una empresa trabajen a través de líneas ADSL o RDSI como si estuviesen en realidad en una misma red local física y sin necesidad de líneas dedicadas punto a punto: directamente sobre Internet y con total garantía. Esto significa un elevado ahorro de costes unido a una gran comodidad.

• **Relación segura con proveedores, distribuidores, socios, y otros agentes del entorno.** Para intercambio de información comercial y técnica, emisión de datos electrónicos (EDI) y comercio electrónico entre empresas.

• **Tele trabajo y acceso de viajantes y personal desplazado.** Los trabajadores que se encuentran de viaje o trabajan desde sus casas podrán acceder a la red de la empresa con total seguridad para buscar información en ciertas bases de datos, remitir pedidos e informes, consultar su correo interno o su agenda o acceder a la Web interna departamental.

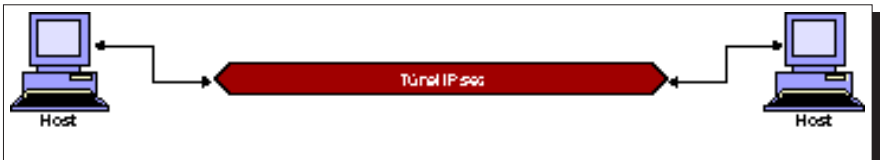

Figura 4. IPsec en modo túnel.

Conclusiones

En este artículo se ha presentado el protocolo IPSec desde el punto de vista técnico y funcional, así como algunos ejemplos de sus aplicaciones en el mundo real.

Se pudo constatar la gran importancia con la que cuenta IPSec, así como la manera en que logra conjuntar características muy importantes para poder ofrecer confidencialidad, autenticidad e integridad en una red. De igual manera se ha logrado recapitular el antes y después de la existencia de IPSEC, pudiendo dar cuenta de que dicha tecnología vino a dar una mejoría en la seguridad puesto que está basada en estándares, es decir, su diseño es independiente del sistema operativo, de la plataforma computacional y de las tecnologías subyacentes empleadas, por lo que su interoperabilidad está asegurada.

Referencias

[1] www.webopedia.com/TERM/I/IPsec.html

[2] www.netbsd.org/Documentation/network/ipsec/

Programación Básica de Interfaces de Usuario Utilizando TCL

Ing. Jesús Antonio Álvarez Cedillo.
Ing. Marlon David González Ramírez
Profesores del CIDETEC-IPN

El sistema operativo linux contó, desde su aparición en el mundo, con una herramienta de programación para la creación de interfaces de usuario llamada TCL, que es la abreviación de las palabras en inglés **"tool command language",** y formada por dos grandes secciones:

a. Una sección de órdenes. Un lenguaje de órdenes para el desarrollo de aplicaciones (**Figura 1**).

b. Una sección de lenguaje formada por un intérprete local.

Con TCL viene integrada una herramienta llamada TK, una variación de TCL que permite su uso en entornos gráficos XWindows. TCL y TK fueron diseñados y desarrollados por el profesor John Ousterhout, en la Universidad de Berkeley.

Como un lenguaje para el desarrollo de aplicaciones, su estructura es similar a otros lenguajes existentes en UNIX; además, permite la ejecución de otros programas y proporciona suficientes elementos de programación, tales como variables, ordenes para control de flujo, y la posibilidad de crear procedimientos.

Estas características permiten construir aplicaciones complejas muy fácilmente, ensamblando programas ya existentes. Una característica que distingue a TCL de otros lenguajes de órdenes es la facilidad de incorporar un intérprete común a cualquier tipo de aplicación, así es fácil estructurar una aplicación como un conjunto de órdenes primitivas que serán organizadas por medio de un guión para adecuarse mejor a las necesidades de los usuarios.

Los programas que permiten a TCL tener nuevas características de operación son llamados extensiones; la más importante es la antes mencionada TK, como herramienta de desarrollo en el entorno XWindows, que permite la creación y la manipulación de interfaces de usuario.

La técnica basada en guiones para la programación de nuevas interfaces de usuario presenta las ventajas siguientes:

a. Rapidez de desarrollo.
b. Interfaz de alto nivel con el entorno X.
c. Separación entre la interfaz y la aplicación.

El Nacimiento de TCL

El objetivo principal de Sun Microsystem cuando diseño TCL, fue

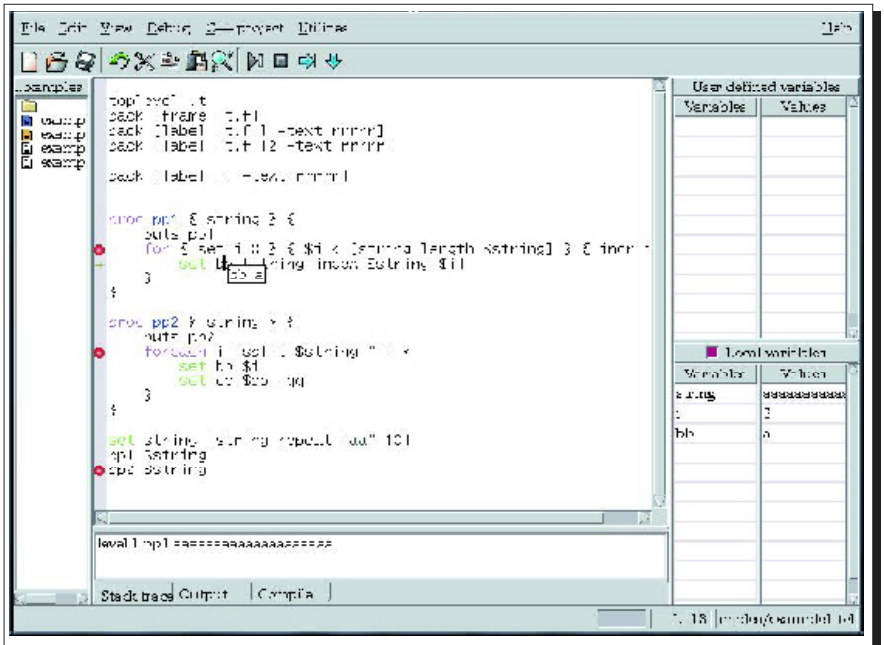

Figura 1. Sección de ordenes usando TCL STUDIO.

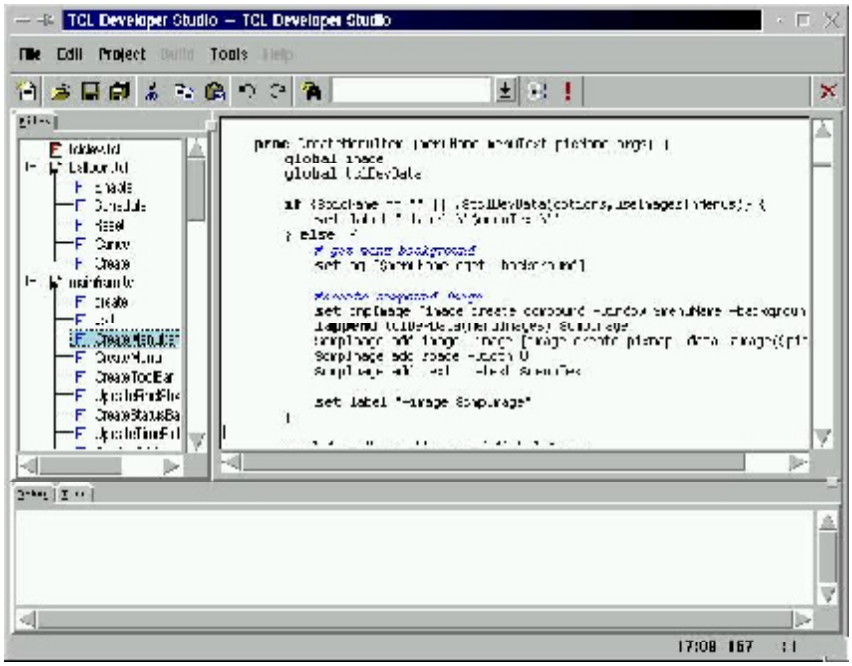

Figura 2. Ejemplo de programación en TCL STUDIO.

crear un lenguaje complementario al lenguaje de programación Java inmune a los virus informáticos y que fuera multiplataforma. Este hecho permitiría crear agentes de software que navegaran por la red Internet de una máquina a otra. En su origen, el profesor John Ousterhout y su equipo pretendían diseñar un lenguaje de muy alto nivel, con la finalidad de enlazar de forma flexible un cierto número de módulos escritos en diferentes lenguajes tales como C y Fortran, al cual le llamaron **meta programación.**

La meta programación permite al sistema que, si existe la modificación de algún módulo, este solo deberá reinterpretarse. TCL enlaza los diferentes módulos y determina si necesitan ser compilados nuevamente. En C++, la modificación de un módulo requiere regenerar las relaciones existentes entre los mismos, característica inexistente en TCL donde la modificación de los módulos no supone la re compilación del resto.

TCL, como una herramienta de órdenes, actúa de manera similar al intérprete Cshell de UNIX. El formato de las órdenes y el uso de comentarios mantienen la misma sintaxis (**Figura 2**). La sintaxis general de una orden TCLes:

orden ***argu1 argu2 argu3...***

La instrucción **orden** es el nombre de la TCL o de un procedimiento desarrollado con anterioridad. Los argumentos se toman siempre como valores de cadena; ello quiere decir que TCL es un intérprete de cadenas, en el que absolutamente todos los argumentos y órdenes se interpretan inicialmente como objetos del tipo cadena de caracteres. TCL no interpreta los argumentos, sino que los pasa directamente a la orden, quien es la encargada de interpretarlos; por ejemplo, en la siguiente orden de asignación

```
set x [expr 3*sin (35)].
```

el procedimiento set actúa como orden, x es un argumento o bien la variable asignada, y [expr 3*sin (35)] es el último argumento; los corchetes no forman parte del argumento, sino que son modificadores que indican que todo el contenido es un solo argumento.

TCL usa el carácter # para indicar que la línea que se está escribiendo corresponde a comentario del programa. Esta línea se omite durante el tratamiento del intérprete, y tiene sentido únicamente como aclaración de algún fragmento del código. Ejemplo:

```
# esto es un comentario
set x 6; # a x se le da el valor de 6
```

Variables

Una variable debe verse como un simple almacén de memoria, al que se le ha añadido una etiqueta que lo identifica en forma única. Como en muchos de los lenguajes de programación usuales, pueden utilizarse caracteres ASCII exceptuando los especiales entre los que debemos señalar los signos de exclamación, interrogación y puntuación, corchetes, llaves, signos de operadores aritméticos, el espacio en blanco, acentos, etc.

Al igual que el lenguaje C, del que TCL ha heredado muchas de sus características, en la formación de los identificadores se hace una distinción entre mayúsculas y minúsculas.

Se usa la orden **set** para la asignación de valores a una variable. Éstas pueden tener cualquier nombre y no es necesario inicializarlas antes de ser usadas. Si la variable que se va a utilizar en la asignación no existe, el intérprete TCL la crea. La sintaxis de la operación de asignación es la siguiente:

set <variable> <valor>

Por ejemplo:

```
set x 6
```

Al igual que se pueden construir variables mediante la orden set, también es posible borrar dichas variables. Para ello se utiliza la orden unset, cuyo formato es el siguiente:

```
unset <variable>
```

Por Ejemplo:

```
unset x
```

Si se busca visualizar el valor de la variable X el mensaje que mostraría seria el siguiente:

```
set X
can't read «x»: no such variable
# No se puede leer la variable, al haber sido destruida
# antes
```

La instrucción **unset** puede referirse a varias variables (separadas por espacio en blanco), en cuyo caso el resultado es la destrucción de todas las variables que se indiquen.

Si en algún momento de la ejecución del programa se desea conocer todas las variables existentes en el mismo, es posible:

```
info vars
```

El intérprete devolverá una lista con todas las variables que se encuentran vivas en ese instante.

Operaciones de Entrada y Salida

En la mayoría de los sistemas operativos, el tratamiento que se hace de los dispositivos de entrada y salida es similar al tratamiento de archivos. Es decir, el sistema operativo asocia a cada dispositivo un archivo, de manera que se consigue el tratamiento uniforme de las entradas y de las salidas independientemente de su origen o destino. Así, al dispositivo estándar de entrada, que suele coincidir con el teclado, le es asignado un archivo cuyo nombre es stdin o entrada estándar; el dispositivo estándar de salida, que suele coincidir con el monitor, tiene asignado el archivo stdout; por último, el archivo asignado al dispositivo por el que se va a dar salida a los errores se llama stderr. Las ventajas de esta forma de tratamiento son claras:

a. Permite conseguir uniformidad en las operaciones
b. Es posible redireccionar las entradas y salidas a archivos distintos al estándar, de manera que un mismo programa puede ser utilizado para tomar las entradas desde teclado, o bien de otro archivo.

La forma de indicar a TCL que se desea realizar una entrada es mediante la utilización de la sentencia gets, cuyo formato es el siguiente:

```
gets <archivo> [<línea>]
```

Donde <archivo> es el archivo del que se va tomar la entrada, y [<linea>] es un parámetro opcional en el que se depositan los valores leídos. Cuando se realiza una lectura mediante la instrucción **gets** se lee una línea completa del archivo de entrada.

Ejemplo:

Orden	Resultado
set x 25	
gets stdin x	
	999
set $x	
	999

En cuanto a las salidas, la orden TCL para realizarlas es:

```
puts <archivo> <línea>
```

Donde <archivo> es el archivo en el que deseamos mostrar la salida, y <linea> es el texto que deseamos mostrar. Cuando la salida deseamos realizarla en la pantalla o en cualquier dispositivo estándar de salida. Entonces el nombre del archivo asociado será stdout.

Por ejemplo:

```
# cálculo del área de un triángulo
puts stdout «Introduzca la longitud de la base"
Introduzca la longitud de la base
gets stdin base
4
1
puts stdout «Introduzca la longitud de la altura»
Introduzca la longitud de la altura
gets stdin altura
5
1
set area [expr $base*$altura/2]
10
puts stdout «El área del triángulo de base $base y altura $altura es $area»

El area del triángulo de base 4 y altura 5 es 10
```

En el ejemplo anterior se muestra la utilización de los agrupamientos mediante comillas dobles. En primer lugar se evalúan las expresiones que se encuentran dentro del agrupamiento, y a continuación se resuelve.

Control de Flujo

Para resolver problemas donde interviene una condicional, de manera similar a otros lenguajes de programación existe la orden if. Cuya sintaxis es la siguiente:

```
if <condición>
then :<sentencia 0 y 1>
else <sentencia Ó y 2>
```

Ante un valor verdadero de la expresión lógica, el algoritmo pasará a ejecutar una secuencia determinada de sentencias.

Por ejemplo:

```
if {$a > $maximo}
then
{
set maximo $a
}
else
{
set a $máximo
}
```

La ejecución de este ejemplo sería la siguiente:

1. En primer lugar, se evalúa la expresión $a > $máximo.

2. Si el resultado de esta expresión fuese VERDADERO (es decir, distinto de 0) entonces se ejecutaría la sentencia que asigna a la variable máximo el valor de la variable a.

3. Si por el contrario la variable tuviera un valor menor o igual a máximo, en este caso se ejecutaría la sentencia del grupo, es decir, la variable a pasaría a tomar el valor de la variable máximo.

Como se puede observar, la estructura de una selección simple permite que el grupo de sentencias que acompañan a las cláusulas then y else sean o bien una sentencia simple, o bien un conjunto de sentencias. Por tanto, se puede concluir que en ese grupo de sentencias es posible incluir otra sentencia condicional simple. Con esta estructura se obtienen las llamadas selecciones anidadas.

En TCL se permiten cláusulas de evaluación a partir de la sentencia **switch**. Incluye la posibilidad de tomar caminos según el parecido que tenga el valor de una cadena al valor esperado. De esta forma, la selección de una alternativa no viene dada por la coincidencia exacta con un valor, sino por la afinidad a dicho valor. El formato de la sentencia **switch** es el siguiente:

```
switch opción cadena
valor1   bloque1
valor2   bloque2
...
valorN   bloqueN
default bloque
```

opción es un modificador que permite determinar la forma de evaluación de los valores que debe tomar la variable. Así, existen varias posibilidades en dicha evaluación:

a) Exact, es la opción por defecto, y debe entenderse como que el valor de la cadena coincide exactamente con el valor indicado. En esta situación, la utilización de la sentencia switch coincide con las alternativas múltiples tradicionales.

b) Glob, en cuyo caso se entenderá que la comparación del valor de la cadena con el valor propuesto se realizará atendiendo a las reglas generales de comparación de cadenas existentes en UNIX

c) Regexp, en cuyo caso la coincidencia del valor de la cadena con los valores indicados se realizará atendiendo al cumplimiento por parte de la cadena de la expresión regular que se señale.

Los ciclos son considerados también en el lenguaje de TCL, y se manejan bajo la misma estructura de las órdenes de bifurcación. El primer tipo de ciclo son los ciclos controlados por contador, que suelen venir representados en los diferentes lenguajes por la estructura **for**.

En este tipo de ciclos, el control de las iteraciones del cuerpo del ciclo se realiza mediante una variable que, en cada pasada por la condición, se incrementa o se decrementa, finalizando la ejecución del ciclo cuando se alcanza un determinado valor.

Por lo tanto, la principal característica de los ciclos controlados por contador es que se conoce el número de iteraciones que se van a realizar del ciclo,

En TCL, la sentencia **for** es similar a la sentencia correspondiente del lenguaje C, con las mismas particularidades. La estructura sintáctica es la siguiente.

```
for <inicial> <test> <final> <cuerpo>
```

Donde <inicial> corresponde a la sentencia o grupo de sentencias de TCL y que sirve como inicialización de nuestra variable contador. Esta sentencia se ejecuta únicamente la primera vez que se ejecuta el cuerpo del ciclo, ignorándose en el resto de iteraciones.

<test> corresponde a la condición de salida del ciclo, y se ejecutará siempre como primera instrucción del cuerpo del ciclo, excepto en la primera iteración en que se ejecuta tras la inicialización. El grupo de sentencias < fina1 > se ejecuta tras el cuerpo del ciclo en cada iteración, y se utiliza para la modificación del valor del contador

Ejemplo: Calcular el factorial de un número n, leído previamente del teclado:

```
#Lectura del número a leer
puts stdout “Introduzca un número entero mayor que cero”
gets stdin numero
#Estudio del factorial
  if $numero<0
  then {puts stdout “No existe el factorial de $numero”}
     else { set factorial 1;
for { set i 1} {$i<$numero} { incr i }
  {set factorial [expr $factorial*$i]}
puts stdout “ $numero != $factorial”}
```

La entrada con centinela es empleada, cuando no se conoce el número de ejecuciones del ciclo, sino que éste va a estar ejecutándose hasta que desde la entrada de datos

se produzca un evento que determine la finalización del ciclo. Esta variable es el centinela. Dependiendo de la forma en que se evalúe el centinela, se puede hablar de dos tipos de ciclos:

a) De evaluación previa, en los que la condición de finalización del ciclo se evalúa al inicio del ciclo, y el cuerpo se repite mientras dicha condición sea verdad. La condición es evaluada incluso antes de entrar por primera vez en el cuerpo del ciclo por medio de la estructura **while**.

b) Evaluación posterior, en los que la condición se evalúa al final de la ejecución del cuerpo del ciclo. En la primera iteración, la condición no es evaluada, por lo que este tipo de ciclos asegura que el cuerpo del ciclo se ejecuta siempre al menos una vez, sentencia **repeat until**

Por ejemplo, deseamos calcular el promedio de las calificaciones de los alumnos de una clase y se tiene un archivo de entrada en el cual existe una marca de fin de archivo que indica cuándo han terminado los datos. Se establece que esta marca será la introducción por teclado del valor 1.

```
gets stdin nota
set media 0
set numAlumnos 1
while {$alumno <> Óý1}{
     incr numAlumnos 1;
   set media {expr media + nota};
     gets stdin nota }
set media {expr media/ 1{ decr numAlumnos
1 }
puts stdout “La nota media es $media”
```

El último tipo de ciclos son los controlados por variable. La evaluación del final del flujo del ciclo se hace mediante una variable que va tomando valores en un rango determinado, indicados en la propia sentencia. Básicamente, este tipo de ciclos son idénticos a los ciclos controlados por contador, solo que ahora no es necesario decrementar o incrementar el valor de la variable al final de la ejecución del ciclo.

El nuevo valor de la variable que hace las veces de contador se extrae secuencialmente de entre los valores de una lista y en este tipo de ciclos se conoce con exactitud el número de iteraciones que se van a realizar del cuerpo del ciclo. La estructura que va a tener este tipo de ciclo es la siguiente:

foreach <variable> <lista> <cuerpo>

<variable> tomará en cada iteración el valor que se encuentre en la lista de valores que se señalen a continuación.

Ejemplo: Se calculará la suma de los diez primeros números primos.

```
set suma 0
foreach primo { 1 2 3 5 7 11 13 17 19 23}{
  set suma {expr suma + primo } }
```

Un procedimiento se define en TCL por medio de la orden **proc**. Su sintaxis es:

proc <nombre> <parámetros> <cuerpo>

Ejemplo:

```
proc factorial { n }
{
set acum 1;
for {set i 1} {$i<=$n} { set acum [expr
$i*$acum]}
return $acum
}
```

Estructuras de Datos Complejas

Para TCL las cadenas de caracteres son el único tipo de dato. Esto es, los objetos son todos una secuencia de caracteres o signos; debido a ello existen múltiples órdenes que permiten el tratamiento de las cadenas de caracteres de una forma mucho más extensa que en C o Pascal.

La sintaxis general para la orden **string** es:

string <operación> <variable> <otros argumentos>

El primer argumento determina qué hacer; el segundo sobre qué cadena se va a realizar la acción, y el resto, si los hay, depende de la operación a realizar. En el caso de que la orden implique la utilización de un índice de valores se comienza a contar desde 0, al igual que ocurre en C. El término **end** se refiere al último carácter de la cadena.

Funciones Básicas

a) **string compare** cadena1 cadena2: Compara cadenas. Devuelve 0 si son iguales, 1 si cadena1 es menor que cadena2 y 1 en otro caso.

b) **string first** cadena1 cadena2: Devuelve el índice en cadena2 en que comienza la primera ocurrencia de cadena 1 o 1 en otro caso.

c) **string index** cadena índice: Devuelve el carácter número índice de cadena.

d) **string length** cadena: Devuelve la longitud de cadena.

e) **string match** modelo cadena: Devuelve 1 si en cadena se encuentra modelo, en caso contrario, devuelve 0.

g) **string range** cadena i j: Devuelve la subcadena de cadena que comienza en i y acaba en el j.

h) **string tolower** cadena: Convierte cadena a mayúsculas.

i) **string toupper** cadena: Convierte cadena a minúsculas.

j) **string trim** cadena caracteres: Elimina M comienzo y M final de cadena los caracteres indicados por caracteres. Por defecto caracteres son los espacios en blanco.

k. **String trimleft** cadena caracteres: Igual que en el caso anterior, pero solo elimina caracteres del comienzo de cadena.

l. **String trimright** cadena caracteres: Como en el caso anterior, pero solo elimina caracteres del final de cadena.

Ejemplo:

```
set x “me llamo Antonio”
set y “, Alvarez”
string length $x
string length $y
```

Una lista es una estructura de datos en la que sus elementos son todos del mismo tipo. La principal característica de las listas, y que las diferencia de otras estructuras de datos es que se desconoce a priori el número de elementos que las componen. Es decir, las operaciones elementales que tendrán estas estructuras de datos serán las operaciones de inserción y borrado. Las listas se implementan en TCL como si se tratara de cadenas.

Debido al tratamiento especial que se hace de las listas, éstas se deben usar únicamente si van a contener pocos elementos, o bien para la construcción de órdenes que se evaluarán posteriormente. Para el manejo de mayores volúmenes de datos las matrices son más adecuadas y eficientes.

Órdenes relacionadas con las listas:

a) **list** argu1 argu2: Crea una lista con todos los argumentos.

b) **index** lista i: Devuelve el elemento enésimo de la lista.

c) **lenght** lista:Devuelve el número de elementos de una lista.

d) **range** lista i j :Devuelve el conjunto formado por los elementos de¡ i al j en la lista.

e) **append** lista arg1 arg2: Añade elementos a lista.

f) **insert** lista i ar1 ar2: Inserta en lista a partir de la posición i los argumentos, generando una lista nueva.

g) **replace** lista i j ar1 ar2: Reemplaza los elementos entre el i y el j de una lista por los argumentos. Genera una nueva lista.

h) **search** modo lista valor: Devuelve el índice del elemento de lista que coincide con valor. El modo puede ser –exact -glob ó regexp. Devuelve 1 si no se encuentra.

i) **sort** indicador lista: Ordena elementos de una lista según el indicador, que puede ser: ascii, integer, real, increasing, decreasing ó command

j) **concat** lista1 lista2 lista3: Junta varias listas en una sola.

El tratamiento que se hace de las matrices en TCL difiere bastante al de otros lenguajes de programación, especialmente en el aspecto de que no se hace mención explícita al tamaño del arreglo en el momento de su declaración, cosa que sí es necesario hacer en otros lenguajes como C o Pascal.

Una matriz es una variable con un índice, el cual es tratado como una cadena de caracteres, por lo que se puede pensar en las matrices como aplicaciones de una cadena (el índice) en otra cadena (el valor del elemento de la matriz). A pesar de ello, internamente se implementa como una tabla hash, estableciéndose una correspondencia entre cada cadena utilizada como índice y cualquier valor ordinal, por lo que el tiempo de acceso a cada elemento es prácticamente el mismo. El orden de los índices viene determinado por el orden de las cadenas de caracteres.

Para señalar el índice del arreglo se delimita con paréntesis. Los elementos de una matriz se definen con **set** y se obtienen por medio de la sustitución $. Por tanto, para que la matriz esté totalmente definida habrá que definir todos y cada uno de los elementos de la misma.

Ejemplo:

```
set matriz (0) Manolo set matriz (2) Pepe
```

La orden **array** devuelve información sobre las variables de matrices y esta formada por las siguientes funciones:

a) **array exist** matriz: Devuelve 1 si matriz es una variable de tipo array

b) **array get** matriz: Devuelve una lista que alterna entre el índice y los valores de la matriz.

c) **array names** matriz modelo: Devuelve los valores del índice de matriz ó solo los coincidentes con el patrón modelo.

d) **Matriz** 1ista Inicializa: Una matriz a partir de una lista que debe tener valores que alternan entre el índice y el valor

e) **array size** matriz: Devuelve el tamaño de matriz

Ejemplo:

```
set media 0
for {set i 0} {$i<100} {incr i} {
   gets stdin nombre ($i);
   gets sidin nota ($i);
   sel media [expr $media+$nota ($i)]
}
set media [expr $media/100.0]
for {set i 0} {$i<100} {incr i}
   if {$nota ($i) <media}
   then {puts stdout "$nombre ($i) ha aprobado"
}
```

Conclusión

La principal aportación de este trabajo es mostrar la capacidad de TCL como lenguaje de programación y sus diversas extensiones a través de un estudio mas detallado de los eventos en cada una de las modalidades soportadas, correspondientes al diseño de una interfaz de usuario. Como puede observarse, este tipo de herramienta es una opción económica, confiable y de trabajo multiplataforma que permite la integración de capacidades inteligentes a la interfaz y la integración de los sistemas en un solo entorno de programación.

Las principales ventajas detectadas fueron las siguientes:

a. Sencillez de programación.

b. Rapidez en el desarrollo de las aplicaciones (Tecnología RAD).

c. Gran velocidad comparado con otros lenguajes interpretados.

d. Facilidad de modificación de las aplicaciones.

e. Multiplataforma.

f. Gran número de extensiones gratuitas.

g. Posibilidad de incorporar nuevos comandos en lenguaje C/C++.

De la misma manera se presentaron los siguientes inconvenientes:

a. Excesivamente lento comparado con los lenguajes compilados.

b. Necesidad del intérprete para ejecutar una aplicación.

c. Difícil de depurar, debido a que, a diferencia de un compilador, el intérprete sólo «traduce» el código que se ejecuta; pudiendo quedar partes del código sin depurar porque el intérprete nunca las haya ejecutado.

Finalmente, el conocer a fondo una extensión gráfica como el TK, permitirá tener como limite en la creación de una interfaz tan solo la imaginación.

Bibliografía

[1] Ousterhout, John. *An Introduction to TCL and TK*. Addison-Wesley Publishing, 1993

[2] Harrison, M.; Mclennan, M. *Effective Tcl/Tk Programming*. Addison-Wesley, 1998.

[3] Página de SUN script: www.sunscript.com

Inversores de Giro para Motores a Pasos en Dispositivos de Lógica Programable

Juan Carlos Herrera Lozada,
Juan Carlos González Robles,
Agustín Cruz Contreras
Profesores del CIDETEC-IPN

El control de motores a pasos (unipolares o bipolares) tiene una gran aplicación práctica. Básicamente se busca invertir el sentido del giro en cualquier momento sin caer en condiciones críticas.

Es importante reconocer las diferentes alternativas de implementación; por lo general, se resuelve utilizando algún microcontrolador u otro driver monolítico con funcionamiento específico.

En este trabajo se consideran dispositivos de lógica programable (PLDs), siendo alternativas de bajo costo (como en el caso de un GAL), con magnífico desempeño y velocidad, reconfigurables y adaptables a aplicaciones simples o complejas (utilizando FPGA o CPLD).

Teoría de Motores a Pasos

Un motor a pasos (*stepper motor*) tiene la propiedad de moverse de un paso a otro, por cada pulso de reloj que se le aplique. Así, puede realizar 15 pasos en un mismo sentido si se le aplican 15 pulsos de reloj. Dependiendo de las características del motor, es posible tener pasos muy pequeños (por ejemplo de 1.8°, por lo que después de 200 pulsos completará una vuelta a razón de 1.8° x 200 = 360°) o pasos muy grandes (por ejemplo de 90°, completando una vuelta con 4 pulsos a razón de 90° x 4 = 360°).

Un motor de este tipo es muy socorrido en diseños que requieren posicionar con exactitud el rotor. Lo anterior resulta difícil en los motores de CD, debido a que estos giran libremente al aplicar un voltaje y si se desea detener el rotor es necesario retirar el voltaje de alimentación, lo cual no garantiza que el rotor se detendrá en una posición predefinida. De cualquier modo, existen técnicas que controlan la duración de pulsos de reloj que pueden hacer que un motor a CD se posicione de manera exacta, como sucede con los servo motores. En cuanto a su funcionamiento y configuración, los motores a pasos se clasifican en tres tipos: unipolares, bipolares y multifase. Nos enfocamos sólo al motor unipolar, cuyo control es el más simple.

Un motor a pasos consta de dos partes principales: el rotor y el estator (Figura 1). El rotor es la parte central del motor, conformada por un imán permanente que gira debido a que el estator tiene bobinas que cuando se excitan adecuadamente generan un campo electromagnético que produce el movimiento del imán en alguna dirección. Lo anterior indica que, para que el motor dé un paso, basta excitar la o las bobinas correspondientes.

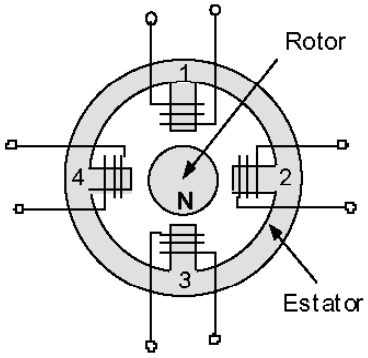

Figura 1. Motor a Pasos de 4 Fases.

El motor unipolar que se utilizó en este trabajo[1] es de 4 fases, o bien, 2 bobinas individuales donde cada una de ellas está separada por un tap o conexión central (punto común); de ahí que se consideren 4 bobinas en vez de dos (**Figura 2**). Cada bobina tiene asociados dos cables para la alimentación tanto positiva como negativa; sin embargo, el tap central es común a todas, por lo que físicamente sólo vemos 5 cables: el común

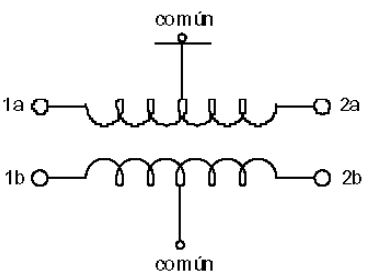

Figura 2. Distribución de las Bobinas de un Motor Unipolar de 4 Fases.

y los cables individuales para cerrar la alimentación de las 4 bobinas.

Es importante notar que si se decide conectar el común a GND, se excitará las bobinas con Vm (voltaje nominal del motor). Si por el contrario, se conecta el común a Vm, se necesitará excitarlas con GND. Todo se resume a polarizar correctamente cada bobina. Consideremos el caso en que se conecta el común a GND y las bobinas se excitan con Vm. De acuerdo a las características eléctricas del motor (12 V, 150 mA, 75 Ω en cada bobina), con el cable negro como **común**, el cable blanco y el verde conformando **1a** y **2a** respectivamente, así como el cable rojo y el cable café como **1b** y **2b**, es posible sugerir la **Figura 3**.

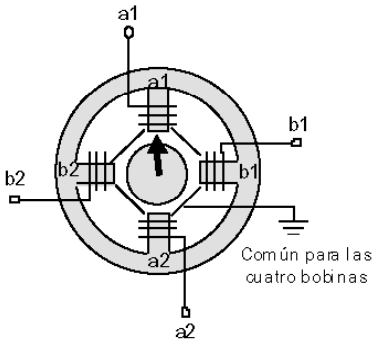

Figura 3. Motor a Pasos de 4 Fases, con Conexiones Propuestas.

En dicha figura se interpreta que para *a1=1, b1=0, a2=0* y *b2=0*, el rotor apuntará hacia la bobina **a1**, debido a que es la única que está excitada. Si se desea que el rotor aparezca en la posición marcada por la **Figura 4**, es necesario aplicar *a1=0, b1=0, a2=1* y *b2=0*.

La secuencia debe respetar obligatoriamente un orden; se sobreentiende que el rotor no puede pasar instantáneamente de la posición en la **figura 3** a la posición marcada por la **figura 4** sin haber recorrido las posiciones previas, no importando el sentido del giro. Opcionalmente,

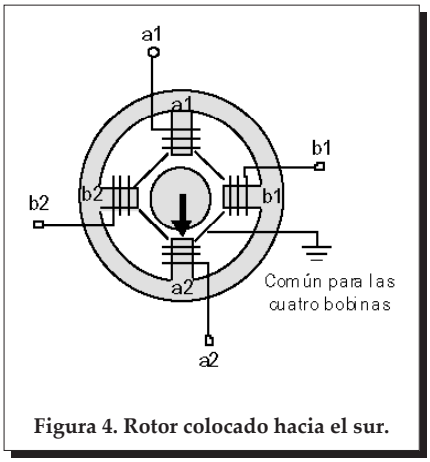

Figura 4. Rotor colocado hacia el sur.

el rotor puede adquirir una posición intermedia, si se excitan dos bobinas coincidentes (cercanas) al mismo tiempo (**Figura 5**). Lo anterior implica un mayor torque, al igual que una mayor demanda de corriente. El estado aplicado es *a1=0, b1=1, a2=1* y *b2=0*.

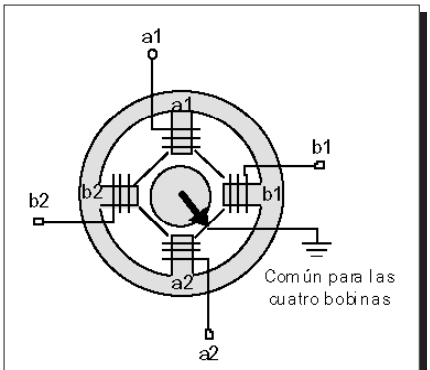

Figura 5. Excitación Simultánea de Dos Bobinas.

Se tienen tres posible secuencias a seguir: *Paso Completo Wave Drive, Paso Completo Normal* y *Medio Paso*. Para la primera secuencia de movimiento se plantea la siguiente tabla, excitando de manera individual cada bobina.

Tabla 1. Wave Drive.

Estado	a1	b1	a2	b2
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0

Diseño del Inversor de Giro en un GAL

Un código aproximado en ABEL HDL, dirigido a un GAL16V8, para resolver la secuencia de *medio paso* se lista a continuación. En la máquina de estados se ha incluido un estado inicial para obligar el inicio del conteo (E0 = 0, 0, 0, 0), por lo que los estados reales de la secuencia se desplazan una posición y después del estado final E8 no se regresa a E0, sino a E1. La variable dir permite cambiar el giro del motor. La variable stop tiene lógica negativa y permite detener el motor manteniendo el estado presente de la máquina de estados, lo cual es recomendable para garantizar una secuencia de restablecimiento lógica. El código se sintetizó con ayuda del software de desarrollo *isp Lever* de *Lattice Semiconductors*.

```
MODULE motor_pasos
«Secuencia de Medio Paso
reloj,dir,stop pin 1,2,3;
«salidas registradas
a1, b1, a2, b2 pin 14,15,16,17 istype 'reg, dc';

«declaración de set
sreg=[a1,b1,a2,b2];
E0=[0,0,0,0];
E1=[1,0,0,0];
E2=[1,1,0,0];
E3=[0,1,0,0];
E4=[0,1,1,0];
E5=[0,0,1,0];
E6=[0,0,1,1];
E7=[0,0,0,1];
E8=[1,0,0,1];

Equations
sreg.clk=reloj;
state_diagram sreg

state E0:
IF dir THEN E1 ELSE E8;

state E1:
IF !stop THEN E1
ELSE IF dir THEN E2 ELSE E0;

state E2:
IF !stop THEN E2
ELSE IF dir THEN E3 ELSE E1;
```

```
state E3:
IF !stop THEN E3
ELSE IF dir THEN E4 ELSE E2;

state E4:
IF !stop THEN E4
ELSE IF dir THEN E5 ELSE E3;

state E5:
IF !stop THEN E5
ELSE IF dir THEN E6 ELSE E4;

state E6:
IF !stop THEN E6
ELSE IF dir THEN E7 ELSE E5;

state E7:
IF !stop THEN E7
ELSE IF dir THEN E8 ELSE E6;

state E8:
IF !stop THEN E8
ELSE IF dir THEN E1 ELSE E7;

END
```

El motor necesita un voltaje de 12 Volts y una corriente de 150 mA por bobina. El GAL no entrega estos valores nominales, por lo que es necesario agregar una etapa de potencia a su salida, para amplificar la corriente. Así mismo, se debe considerar un voltaje de 5 Volts para el GAL y uno de 12 Volts para el motor. La etapa de potencia (driver) se puede implementar con un integrado monolítico ULN2003 que entrega hasta 500mA y presenta la característica de ser un driver con trabajo invertido, es decir, complementará los datos entrantes. En realidad no existe problema alguno, porque sólo basta con conectar el común a los 12 Volts nominales y conservar la misma tabla de secuencias establecida, ya que un «1» al entrar al driver se convertirá en un «0».

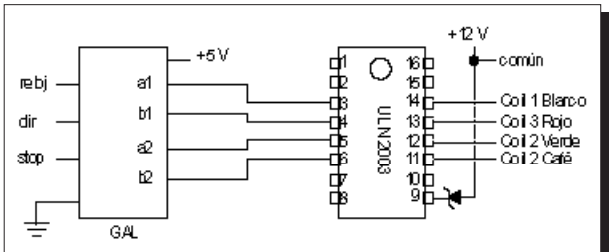

Figura 6. Circuito Controlador en un GAL16V8.

El circuito que se construyó se muestra en la **Figura 6**. El diodo zener es de 12 Volts.

Circuito Controlador en un GAL16V8

Las **Figuras 7 y 8**, muestran la implementación física. Los leds se utilizaron para verificar el cambio en las salidas. Para el voltaje nominal del motor (12 Volts) se utilizó un regulador 7812 y para la alimentación del GAL (5 Volts) se utilizó el voltaje de la tarjeta de desarrollo del FPGA, la cual se comentará posteriormente.

Diseño del Inversor de Giro en un FPGA

La arquitectura FPGA tiene diferencias con la propia de los CPLD; sin embargo, el diseño y la implementación son similares. Recordando que las herramientas de síntesis lógicas estándar soportan descripciones en ABEL, VHDL y Verilog, se realizó una descripción adicional del mismo controlador en VHDL. La intención es proporcionar una idea simple de cómo es posible definir una macro reutilizable para controlar varios motores a la vez, gracias al gran número de terminales y recursos lógicos dentro de un dispositivo de este tipo.

El código que resuelve el control a medio paso, se muestra a continuación. La descripción se realizó en VHDL.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

—Secuencia de Medio Paso
entity motor_pasos is
port (
```

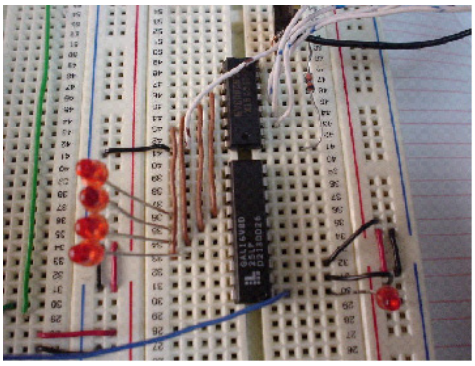

Figura 7. Controlador en un GAL16V8.

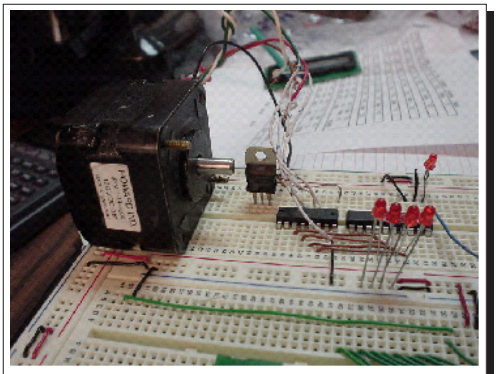

Figura 8. Circuito Controlador con Motor, vista frontal.

```
RELOJ, RESET, STOP, DIR: in STD_LOGIC;
DATO_MOTOR: out STD_LOGIC_VECTOR(3 downto 0)
);
end motor_pasos;

architecture motor_arch of motor_pasos is
type state_type is (INICIA, CERO, UNO, DOS, TRES, CUATRO, CINCO, SEIS, SIETE);
signal estado, estado_siguiente: state_type;
begin
arranque_motor:process (RELOJ, RESET)
begin
if RESET='0' then
estado <= INICIA;
elsif RELOJ='1' and RELOJ'event then
estado <= estado_siguiente;
end if;
end process arranque_motor;

estados_motor:process (estado, DIR, STOP)
begin
case estado IS
when INICIA => if STOP='0' then
estado_siguiente <= INICIA;
elsif DIR='1' then
estado_siguiente <= CERO;
else estado_siguiente <= SIETE;
end if;
```

```
when CERO => if STOP='0' then
estado_siguiente <= CERO;
elsif DIR='1' then
estado_siguiente <= UNO;
else estado_siguiente <= SIETE;
end if;

when UNO => if STOP='0' then
estado_siguiente <= UNO;
elsif DIR='1' then
estado_siguiente <= DOS;
else estado_siguiente <= CERO;
end if;

when DOS => if STOP='0' then
estado_siguiente <= DOS;
elsif DIR='1' then
estado_siguiente <= TRES;
else estado_siguiente <= UNO;
end if;

when TRES => if STOP='0' then
estado_siguiente <= TRES;
elsif DIR='1' then
estado_siguiente <= CUATRO;
else estado_siguiente <= DOS;
end if;

when CUATRO => if STOP='0' then
estado_siguiente <= CUATRO;
elsif DIR='1' then
estado_siguiente <= CINCO;
else estado_siguiente <= TRES;
end if;

when CINCO => if STOP='0' then
estado_siguiente <= CINCO;
elsif DIR='1' then
estado_siguiente <= SEIS;
else estado_siguiente <= CUATRO;
end if;

when SEIS => if STOP='0' then
estado_siguiente <= SEIS;
elsif DIR='1' then
estado_siguiente <= SIETE;
else estado_siguiente <= CINCO;
end if;

when SIETE => if STOP='0' then
estado_siguiente <= SIETE;
elsif DIR='1' then
estado_siguiente <= CERO;
else estado_siguiente <= SEIS;
end if;

END CASE;
end process estados_motor;

salida:process(estado)
begin
case estado IS
when INICIA => DATO_MOTOR <=
«0000»;
when CERO   => DATO_MOTOR <=
«1000»;
```

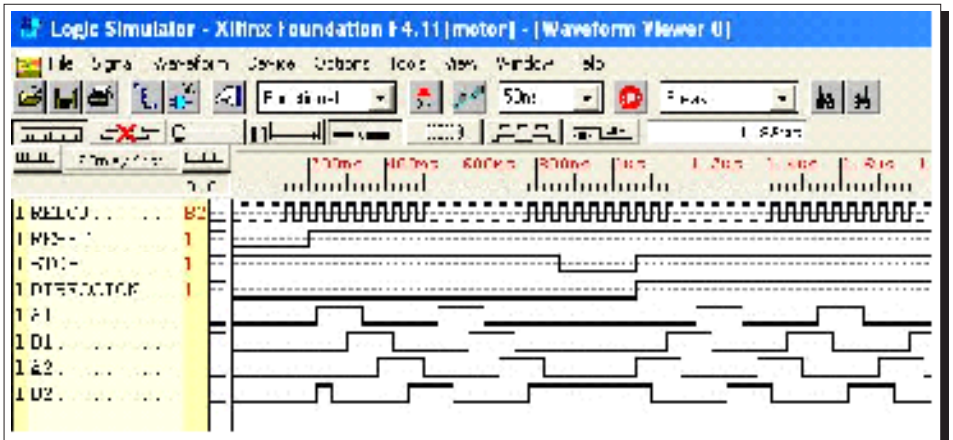

Figura 9. Resultados de Simulación Para el Controlador, en VHDL.

```
when UNO    => DATO_MOTOR <=
«1100»;
when DOS    => DATO_MOTOR <=
«0100»;
when TRES   => DATO_MOTOR <=
«0110»;
when CUATRO => DATO_MOTOR <=
«0010»;
when CINCO  => DATO_MOTOR <=
«0011»;
when SEIS   => DATO_MOTOR <= «0001»;
when SIETE  => DATO_MOTOR <=
«1001»;
when others => NULL;
END CASE;
end process salida;

end motor_arch;
```

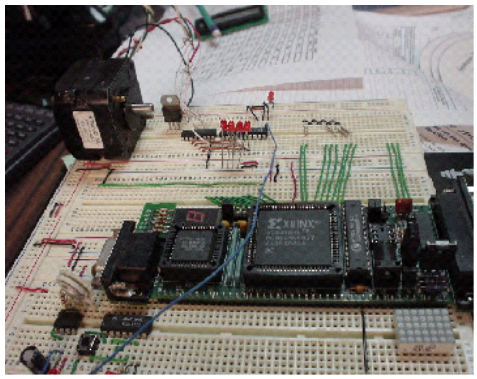

Figura 11. Controlador en FPGA.

La implementación del controlador se realizó sobre un FPGA XC4010XL de Xilinx (tarjeta XESS) sintetizando el código con el ambiente de desarrollo *Foundation* (**Figuras 9** y **10**), obteniendo resultados satisfactorios.

La frecuencia de trabajo es de hasta 6MHz, aunque para elementos de respuesta mecánica son suficientes 10 KHz.

De manera similar al circuito del GAL, fue necesario el dispositivo ULN2003 para compensar la demanda de corriente en las bobinas del motor. La **Figura 11**, muestra la tarjeta de desarrollo utilizada para validar los resultados.

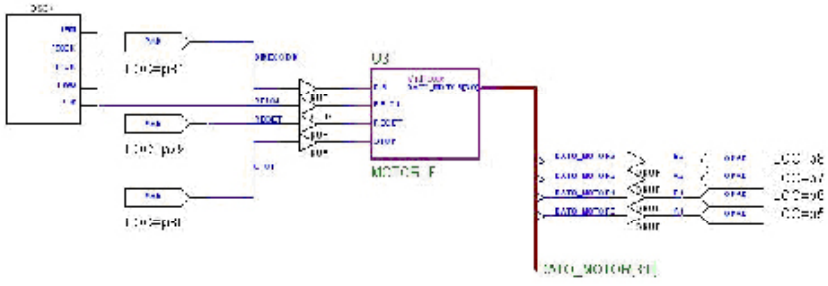

Figura 10. Diagrama Esquemático del Diseño en Foundation.

Conclusiones

El inversor de giro implementado en el GAL16V8 resultó muy económico en comparación a los drivers monolíticos existentes. La lógica descrita es muy simple, por lo que el diseñador puede optar por el HDL de su preferencia sin cambios drásticos en el código. Es posible realizar un controlador para dos motores en un GAL22V10, manteniendo la misma filosofía de diseño.

En código se hace extensivo para dispositivos de arquitectura avanzada, como FPGA y CPLD. Tratando los controladores como macros dentro de un esquemático se facilita la implementación de sistemas completos para controlar varios motores de forma paralela. Por tratarse de estándares, los códigos en ABEL y VHDL pueden sintetizarse sobre dispositivos de otros fabricantes (Actel, Altera, Lattice), vía software de propietario.

Los trabajos planteados a futuro implican una comparación detallada con la alternativa de microcontroladores. La iniciativa presentada por los dispositivos de lógica programable por sí misma deriva en una mayor versatilidad, aunado al diseño contemporáneo de la electrónica digital para actualizar los programas de estudio de la Ingeniería Electrónica y áreas afines.

Bibliografía

[1] Sebastian Michael J. *Application - Specific Integrated Circuits*. Addison Wesley, 2000.

[2] *Synopsis, FPGA Express with Verilog HDL and VHDL, Reference Manual*. Xilinx in line, 2002.

[3] Walerly, John. *Digital Design*. Prentice Hall, 2002.