



CCADET

CENTRO DE CIENCIAS APLICADAS Y DESARROLLO TECNOLÓGICO
UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Manual de Empleo y Servicio

Desarrollo de Bibliotecas de Comunicación en Lenguajes de Descripción de Hardware para el manejo de Interfaces mediante Sistemas de Lógica Programables en tarjetas de desarrollo de la empresa XILINX

Autores:

Alejandro Padrón Godínez

Rafael Prieto Meléndez

Alberto Herrera Becerra

Venancio Gerardo Calva Olmos

Grupo Académico de Modelado y Simulación de Procesos

Fecha:

Mayo – 2013

Tipo de proyecto: Desarrollo Tecnológico

Financiamiento (interno)

Titulo: Desarrollo de Bibliotecas de Comunicación en Lenguajes de Descripción de Hardware para el manejo de Interfaces mediante Sistemas de Lógica Programables en tarjetas de desarrollo de la empresa XILINX

Autores:

Alejandro Padrón Godínez
Rafael Prieto Meléndez
Alberto Arturo Herrera Becerra
Venancio Gerardo Calva Olmos

Grupo Académico de Modelado y Simulación de Procesos, Centro de Ciencias Aplicadas y Desarrollo Tecnológico, Universidad Nacional Autónoma de México, Circuito Exterior S/N, Ciudad Universitaria AP.70-186, C.P. 04510, México D.F. México

Resumen

En particular para la tarjeta de desarrollo Spartan 3E de la empresa Xilinx que contiene Sistemas de Lógica Programable (*FPGA* por sus siglas en ingles de: *Field Programmable Gates Arrays*), es indispensable conocer y manejar las interfaces de comunicación y de usuario. Estas interfaces pueden clasificarse como de almacenamiento, visualización, comunicación, tratamiento de datos y extracción de información. Sirven para trabajar con indicadores, pantallas de cristal líquido (*LCD, Liquid Crystal Display*), botones pulsadores, teclados, también para la adquisición de señales analógicas y digitales, además de generar señales de control para otros dispositivos. En la parte de interfaces para comunicación en serie se tienen las de buses locales y redes locales. En particular en este reporte se presentan las interfaces para el manejo de un despliegue "*display*" de 7 segmentos, el manejo de un *LCD*, para el uso del periférico "*Personal System/2*" (*PS/2*) tanto para ratón como para teclado, uso de memoria y el manejo de señales de control para un *transceiver* para la comunicación en serie universal por paquete (*USB*). Esta última interface se desarrolla principalmente para el almacenamiento masivo de datos como una aplicación a futuro. Los resultados se muestran en cada sección usando la plataforma y herramientas de la empresa *Xilinx ISE Design Suite versión 13.3*, a manera de simulaciones, imágenes o videos de la implementación de la interfaz en funcionamiento, así mismo se presentan como anexos los códigos en lenguaje de diseño de alto nivel (*VHDL* por sus siglas en ingles de: *Very High Design Language*).

Índice

Índice	3
Glosario de Términos	4
Lista de Figuras	5
Lista de Tablas	6
Introducción	7
Objetivos General	8
Objetivo Particular	8
1. Antecedentes	9
1.1 Lenguajes de Descripción de Hardware	9
1.2 Lenguajes de descripción de Hardware versus desarrollo de Software	11
1.3 Descripción de la tarjeta Spartan 3E Starter XC3S500E de Xilinx	11
2. Interfaces de comunicación y de usuario	12
2.1 Indicadores y despliegue	12
2.2 Interruptores pulsadores y teclados	24
3. Interfaz de visualización	36
3.1 <i>Display</i> de 7 segmentos y código BCD	36
3.2 Desarrollo de un reloj digital	41
4. Interfaz para el cristal liquido envío de mensajes	44
5. Interfaz PS2 y VGA	49
5.1 Interfaz “ <i>Personal System/2</i> ” (PS/2)	49
5.2 Interfaz “ <i>Video Graphics Array</i> ” (VGA)	51
5.3 Uso del Ratón mediante el puerto PS/2 y un monitor VGA	53
5.4 Uso del Teclado mediante el puerto PS/2 y el LCD	57
6. Desarrollo de un transportador “ <i>transceiver</i> ” para la interfaz USB	59
6.1 Implementación de transferencia de datos sobre USB	59
6.2 Operación del programa	59
6.3 Puntos terminales o puntos de lectura y escritura	60
6.4 Interfaz de Aplicación	60
6.5 Las opciones de configuración	60
6.6 Software Anfitrión “ <i>Host</i> ”	61
6.7 Síntesis	61
6.8 Recursos utilizados	61
6.9 Pruebas	61
6.10 Procedimiento de prueba	62
6.11 Desempeño de la implementación	62
6.12 Alcances conocidos del desarrollo	62
• Conclusiones	63
Anexo I.- Datos para las restricciones de área del BCD	64
Anexo II.- Código en VHDL para el desarrollo de un reloj digital	64
Anexo III.- Código en VHDL para el manejo de un LCD	65
Anexo IV.- Código en VHDL para el manejo del Ratón-PS/2-VGA	76
Anexo V.- Código en VHDL para el manejo del teclado	80
Anexo VI.- Código en VHDL para el <i>transceiver</i>	92
• Referencias Bibliográficas	109

Glosario de Términos

Símbolo	Significado
ASIC:	Applications-Specific Integrated Circuit, siglas en ingles.
BCD:	Binary-coded decimal, siglas en ingles.
CAD:	Computer-aided design, siglas en ingles.
DoD:	Departamento de Defensa
FIPS:	Federal Information Processing Standart, siglas en ingles.
FPGA:	Field Programmable Gates Arrays, siglas en ingles.
GSM:	Global System for Mobile communications, siglas en ingles.
IEC:	International Electrotechnical Comission, siglas en ingles.
IEEE:	Institute of Electrical and Electronic Engineers, siglas en ingles.
ISO:	International Standard Organization, siglas en ingles.
ITU:	International Telecommunication Union, siglas en ingles.
JTAG:	Joint Test Action Group, siglas en ingles.
LCD:	Liquid Crystal <i>Display</i> , siglas en ingles.
NIST:	National Institute of Standart Technology, siglas en ingles.
NSA:	National Security Agency, siglas en inglés
OSI:	Open Systems Interconnection, siglas en ingles.
PCB:	Printed Circuit Board, siglas en ingles.
PS/2:	Personal System/2, siglas en inglés, conector o puerto
SI:	Seguridad Informática
SoC:	System on Chip, siglas en ingles.
SPICE:	Simulation Program with Integrated Circuits Emphasis, siglas en ingles.
SRAM:	Static Random Access Memory, siglas en ingles.
UART:	Universal Asynchronous Receiver/Transmitter, siglas en ingles.
USB:	Universal Serial Bus, siglas en ingles.
VGA:	Video Graphics Array, siglas en ingles.
VHDL:	Very High Description Lenguage, siglas en ingles.
VHSIC:	Very High Spped Integrated Circuit, siglas en ingles.
Xilinx:	Sistema de Integración para el desarrollo de aplicaciones de FPGA

Lista de Figuras

- Figura 1. Niveles de abstracción/precisión y estilos de modelado VHDL.
- Figura 2. Desarrollo en software versus hardware: (a) niveles de abstracción y lenguajes de alto nivel y (b) esquema básico del diseño descendente con HDL.
- Figura 3. Tarjeta de desarrollo Spartan-3E fabricada por la empresa Xilinx.
- Figura 4. Led de diferentes tipos y colores.
- Figura 5. Configuración de conexiones de los led, R_{LIMIT} para I_{LED} aproximadamente 20 [mA].
- Figura 6. Posición de los 8 led en la Spartan-3E User Guide, (2008).
- Figura 7. Asignación de los pin en el archivo .UCF del área restringida.
- Figura 8. Clasificación de los segmentos.
- Figura 9. Configuración en cátodo común.
- Figura 10. Configuración en ánodo común.
- Figura 11. Encapsulado del display de 7 segmentos y punto decimal.
- Figura 12. Integrado de una matriz de led de 5x7 y sus terminales de conexión.
- Figura 13. Matriz de 5x7 con columnas de cátodos.
- Figura 14. Encendido del led columna 3 fila 5 de la matriz de 5x7.
- Figura 15. Matriz de 8x8 RGB y una aplicación directa el Bitman desarrollado por Ryota Kuwakubo.
- Figura 16. Diagrama esquemático para hacer el refresco del display de 7 segmentos.
- Figura 17. Diagrama de tiempos del refresco del display de 7 segmentos.
- Figura 18. Refresco de display mediante hardware con interfaz en serie.
- Figura 19. Esquema genérico de un display LCD, tomado de Hitachi HD44780U (LCD-II), (1998).
- Figura 20. Un LCD de 20 columnas por 4 renglones.
- Figura 21. Sistema de visualización en FPGA.
- Figura 22. Placa de desarrollo Spartan 3E: Display LCD de 16 X 2.
- Figura 23. Comandos en controlador de display alfanumérico HD44780.
- Figura 24. Codificación de datos en controlador de display alfanumérico, Spartan-3E User Guide, (2008).
- Figura 25. Placa de desarrollo Spartan 3E: Interfaz de video VGA, Spartan-3E User Guide, (2008).
- Figura 26. Control de las señales VGA para la interfaz de la tarjeta Spartan 3E (2008).
- Figura 27. Control del tiempo para la interfaz VGA, Spartan-3E User Guide, (2008).
- Figura 28. Interruptores deslizantes de la Spartan 3E.
- Figura 29. Cuatro interruptores de Botón "Push-Button" colocados alrededor del interruptor perilla.
- Figura 30. Cada interruptor de botón requiere resistencia interna a tierra en pin de entrada del FPGA.
- Figura 31. Parámetros para la conexión en el archivo .UCF de los interruptores de botón.
- Figura 32. Diagrama del interruptor de Botón y perilla con su resistencia de entrada a tierra.
- Figura 33. Eliminación de señal rebote del interruptor de botón vista en osciloscopio.
- Figura 34. Teclados en forma matricial.
- Figura 35. Operatividad del teclado matricial en un diagrama de flujo.
- Figura 36. Localización y Señales del Conector PS/2, Spartan-3E User Guide, (2008).
- Figura 37. "Scan Code" Código de exploración para el teclado con conexión PS/2, Spartan-3E User Guide, (2008).
- Figura 38. Transiciones del ratón.
- Figura 39. El ratón usa un sistema de coordenadas relativo al movimiento.
- Figura 40. Interruptor pulsante, vista posterior, uso en protoboard y dibujo frontal.
- Figura 41. Controlador Digital de Maxim.
- Figura 42. Diagrama de la arquitectura del sistema.
- Figura 43. Estructura Maestro Esclavo.
- Figura 44. Comunicación Serial Maestro Esclavo.
- Figura 45. Diagrama del "Serial Data Out" que requiere salida de tres estados.
- Figura 46. Esquema de conexión para la interface I^2 .
- Figura 47. Especificaciones para el bus de la I^2C .
- Figura 48. Diagrama de tiempos para las transferencias..
- Figura 49. Conformación de la trama en los mensajes enviados.
- Figura 50. Cronograma para un bus arbitrado.
- Figura 51. Tipo de conexiones para una comunicación serial.
- Figura 52. Circuitos Convertidores de nivel: MAX232 (TTL a RS232 y RS232 a TTL).

Figura 53. Asignación de terminales “pins” sobre los conectores tipo serie.
 Figura 54. Cronograma para la comunicación serial.
 Figura 55. Conexiones para el cable RS-232.
 Figura 56. Interface con Bus SDI-12.
 Figura 57. a) Terminales del display DC-05. b) Display de cátodo común.
 Figura 58. Diagrama del display DC05.
 Figura 59. Diagrama lógico del circuito integrado 74LS48.
 Figura 60. Diagrama de las funciones de la tabla (9), para cada segmento del display DC-05.
 Figura 61. Configuración para el BCD que será usado para el display de 7 segmentos.
 Figura 62. Simulación del encendido del display de 7 segmentos.
 Figura 63. Configuración de las terminales en el PlanAhead para las entradas (switch) y salidas (led).
 Figura 64. Prueba del código BCD en Hardware.
 Figura 65. Resumen de recursos consumidos en el FPGA de la tarjeta Spartan 3E, reporte obtenido después de la implantación BCD mediante Xilinx.
 Figura 66. Diagrama esquemático desarrollado en el paquete Xilinx Design Suite 13.3 para el reloj digital.
 Figura 67. Alambrado de los display de 7 segmentos en protoboard.
 Figura 68. Video del diseño del reloj digital operando.
 Figura 69. Recursos consumidos en el FPGA por el reloj digital, resumen obtenido después de la implantación del reloj digital en Xilinx.
 Figura 70. Configuración para lectura-escritura y reinicio, Spartan-3E User Guide, (2008).
 Figura 71. Direcciones de memoria DD RAM en Hexadecimal.
 Figura 72. Principal máquina de estados.
 Figura 73. Secuencia de Inicialización
 Figura 74. Restricciones de sincronización de la máquina de estado.
 Figura 75. Letrero implementado en el LCD de la tarjeta.
 Figura 76. Simulación para la implementación de la interfaz en el LCD.
 Figura 77. Recursos que ocupa la aplicación dentro del FPGA de la tarjeta, resumen obtenido por Xilinx.
 Figura 78. Esquemático creado para el bloque de la interfaz PS/2 en Xilinx.
 Figura 79. Diagrama de tiempo del envío y recepción de datos al dispositivo.
 Figura 80. Estructura del componente VGA.
 Figura 81. Controlador del ratón.
 Figura 82. Diagrama esquemático generado en Xilinx del bloque para la resolución del mouse.
 Figura 83. Diagrama esquemático de los controladores del ratón y para la interface PS/2 y VGA.
 Figura 84. Video del funcionamiento de las interfaces ratón-PS/2- VGA.
 Figura 85. Resumen de los recursos consumidos en el FPGA de la aplicación desarrollada en Xilinx.
 Figura 86. Video que muestra del funcionamiento del teclado PS/2 y su escritura en el LCD.
 Figura 87. Recursos consumidos por la implementación y uso de las interfaces PS/2 para el teclado y su escritura en el LCD, resumen obtenido por Xilinx.
 Figura 88. Diagrama del usb_serial con interfaz externa y sub-entidades.
 Figura 89. Ciclos de reloj para la interfaz RxFIFO.
 Figura 90. Recursos consumidos en el FPGA de la tarjeta Spartan 3E, resumen obtenido por Xilinx.

Lista de Tablas

Tabla 1. a) Configuración para encendido de los led para formar los dígitos para el display de ánodo común. b) Lógica binaria para el encendido de los led y su equivalente en hexadecimal.
 Tabla 2. Código de 3-bit para el control de colores, Spartan-3E User Guide, (2008).
 Tabla 3. Control de Tiempos Modo VGA 640x480, Spartan-3E User Guide, (2008).
 Tabla 4. Salidas del conector PS/2, Spartan-3E User Guide, (2008).
 Tabla 5. Modos de Funcionamiento.
 Tabla 6. Definiciones del bus I²C, terminología.
 Tabla 7. Protocolos estándar de comunicación en serie.
 Tabla 8. Señales del protocolo RS232.
 Tabla 9. Niveles Lógico y de Voltaje para los datos en serie.
 Tabla 10. Formato de la trama SDI-12.
 Tabla 11. Comandos y respuestas (caracteres ASCII)
 Tabla 12. Código Binario Decimal.
 Tabla 13. Funciones para cada segmento DC-05.
 Tabla 14. Tabla de verdad para el circuito integrado 74LS48.
 Tabla 15. Opciones de configuración para el dispositivo USB de transferencia.

Introducción

Desde la década de los 70s, los dispositivos de lógica estándar eran todo un avance tecnológico, y los tableros con circuitos impresos eran lo mejor de su momento. Pero algunos se preguntaron “¿qué sucede si le damos a los diseñadores la capacidad de poner diversas interconexiones en ejecución en un dispositivo más grande?” Esto permite que el diseñador integre muchos dispositivos de lógica estándar en una partición. Para ofrecer lo último en flexibilidad del diseño, Ron Cline Signetics (que fue comprado más adelante por Philips y convertido eventualmente a Xilinx), vino con la idea de dos planos de programación. Estos planos proporcionaron cualquier combinación de compuertas lógicas “AND” y “OR”, así como sustituir las compuertas AND en términos de múltiples ORs.

En este nuevo siglo, los *FPGA* son una de las tecnologías disponibles para el desarrollo de sistemas electrónicos. Representan una atractiva opción para la producción de pequeñas cantidades ya que sus costos fijos (costo del prototipo de la primera unidad) son bastante más bajos que los correspondientes en *ASIC* (*Applications-Specific Integrated Circuit*, circuitos integrados de aplicación específica). Además, en pos de reducir el tamaño y por tanto el costo unitario, una posibilidad interesante es reconfigurarlas mientras operan de modo que el mismo dispositivo pueda ejecutar diferentes funciones predefinidas.

Actualmente los sistemas electrónicos están presentes en el quehacer cotidiano, desde la electrónica de consumo a los sistemas de control industrial, pasando por las aplicaciones para automotriz, seguridad y una gama extensa en diferentes ámbitos. La tendencia de estos dispositivos con respecto a su diseño electrónico radica en la complejidad de muchos de los elementos que lo conforman, donde se exige, por ejemplo, que sean de fácil utilización y adaptables a varias aplicaciones que consuman pocos recursos y que puedan estar rápidamente a la venta. La tecnología actual de circuitos integrados permite realizar e integrar estos complejos sistemas en muy poco espacio; son los llamados sistemas embebidos (o empotrados) o *SoC* por sus siglas en inglés de “*System on Chip*”. Estos sistemas embebidos usados para resolver tareas específicas incluyen hardware y software diseñado expresamente para ello. En el caso de las *FPGA*, al ser dispositivos reconfigurables son altamente útiles para desarrollo de prototipos o pequeñas cantidades a un precio razonablemente bajo.

La evolución de la microelectrónica desde su nacimiento ha sido impresionante, en poco más de diez años se pasó del primer dispositivo integrado (*flip-flop* con unos cuantos transistores) a las primeras memorias y procesadores. A los pocos años, el diseño de circuitos integrados salió de las fábricas y se hizo accesible a los ingenieros de aplicación, dando lugar a los *ASIC* (*Application Specific Integrated Circuit*). Luego a mediados de los 80's, se puso de manifiesto la necesidad de disponer de un lenguaje estándar capaz de dar el soporte necesario al proceso completo de diseño de *chips* y sistemas electrónicos (desde la idea hasta la implementación y explotación de un desarrollo) en sus distintas etapas y niveles de abstracción y cuya complejidad, ya de por sí elevada, se iba a incrementar drásticamente hacia los años noventa, (Terés L. et al, 1998).

Estas inquietudes, canalizadas a través del *DoD* (*Department Of Defense*) de los E.U.A., llevan al nacimiento del *VHDL* (1987), que tras ser adoptado como estándar del *IEEE* (Std.-1076, 1987), se presenta como el lenguaje estándar por excelencia del diseño electrónico. Desde el primer momento se convirtió en la herramienta y el motor de base que ha facilitado e impulsado los enormes avances de las metodologías, técnicas y herramientas *CAD* de diseño electrónico de los últimos diez años. Esta evolución arrastra al propio *Verilog*, ahora también convertido en estándar del *IEEE* (Std.- 1364, 1995), que es con quien compete y lleva de la mano al *VHDL*.

En definitiva, y aunque sólo se considere al *VHDL* como herramienta de base, la implantación de este lenguaje ha supuesto tal cantidad de cambios en el diseño electrónico (nuevas metodologías de diseño de alto nivel y flujos descendentes, herramientas *CAD* de simulación/síntesis de circuitos y

sistemas, nuevos de los equipos de diseño, cambios en la organización y desarrollo de proyectos, etc.), que sin lugar a dudas, se puede hablar de un antes y un después del *VHDL*.

La idea de esta introducción es percibir que el desarrollo de este reporte está siendo implementado con las nuevas técnicas y herramientas para el diseño electrónico. Donde no ha sido nada fácil aprender un nuevo lenguaje de programación, aunque existen muchas ayudas para desarrollarlo e implementarlo que dan soporte ante cualquier obstáculo encontrado. Así los desarrollos de las interfaces aquí mostradas están en el lenguaje de programación *VHDL*. Se utiliza además la ventaja de poder realizar una simulación de los desarrollos en algunos casos antes de llevar a cabo las implantaciones (programación de *FPGA*) en hardware. En los antecedentes de este reporte, se presentarán las características genéricas de los lenguajes de descripción de hardware para su mejor comprensión. En la misma sección se realiza una comparación entre el lenguaje de descripción de Hardware contra el desarrollo en Software. No se pretende con este trabajo desaparecer o echar abajo diseños que se han realizado sino por lo contrario mostrar que esos diseños pueden ser implementados con mucho menos elementos que un diseño original. Se debe tener en mente que estas implementaciones pueden ser utilizadas en aplicaciones desarrolladas en un futuro próximo, facilitando la funcionalidad al agregar en forma modular estas interfaces.

También en la parte de los antecedentes se describe físicamente a la tarjeta Spartan 3E de Xilinx para conocer de qué elementos está conformada y cuáles otros subelementos pueden ser utilizados para el desarrollo de las interfaces. Mostrando sus características y parámetros para señales de entrada y salida, sin dejar de mencionar que se puede consultar a detalle todo esto en el manual de la tarjeta pero no la programación de las mismas. Como se mencionó en el resumen se mostrarán algunas simulaciones, imágenes y videos como resultados de las implantaciones y en los anexos estarán los códigos en *VHDL* para cada interfaz.

Objetivo General

Desarrollar y construir las interfaces de comunicación y de usuario que se pueden aprovechar con las tarjetas de desarrollo Spartan 3E de la empresa Xilinx basadas en *FPGA* para el desarrollo de sistemas digitales y aplicaciones específicas. Esto se llevará a cabo mediante las herramientas de desarrollo y simulación que se presentan en "*ISE Desing Suite vs. 13.3*", además del empleo de Lenguaje de Descripción de Hardware (*VHDL*).

Objetivos Particulares

Diseñar una interfaz para el manejo de un *display* de 7 segmentos con cátodo común empleando las salidas que tiene la tarjeta Spartan 3E.

Enviar datos al despliegue de cristal liquido de 16 caracteres por 2 líneas contenido en la tarjeta Spartan 3E.

Generar la interfaz para el uso y manejo del puerto PS/2 de la tarjeta Spartan 3E, en particular para recibir señales de un ratón y de un teclado.

Hacer una interfaz para la manipulación de datos entrada-salida en la memoria SRAM de la tarjeta Spartan 3E.

Crear una interfaz para la comunicación universal serial por paquete, usando las salidas de la tarjeta Spartan 3E.

1. Antecedentes

Conforme ha avanzado la tecnología en desarrollo y diseño de circuitos electrónicos, se ha logrado pasar del alambrado en "*protoboard*" a simulaciones de análisis y diseño asistido por computadora con herramientas tan complejas que a veces complica su entendimiento. Se buscaba acoplarse con una de estas herramientas para un manejo sencillo y poder implantar los diseños de los circuitos mediante una interfaz amigable con el usuario. Para ello era necesario iniciar con señales de entrada de acorde a la naturaleza del diseño implementado, esto a su vez para que la interpretación de los resultados, es decir las señales de salida fuera sencilla. Estos paquetes de software para diseño de circuitos eran los llamados *CAD* y algunos parecidos al *SPICE* en forma general. Una vez obtenidos los resultados requeridos se procedía a desarrollar el circuito impreso con otras herramientas diseñadas para tal tarea, un ejemplo de esto era el paquete *OrCAD*, en particular la herramienta *PCB*.

Algunos de los modelos de circuitos neuronales electrónicos diseñados por el Laboratorio de Computación Adaptable antes Ingeniería Neuronal eran implementados con este tipo de herramientas. (Padrón, 1997). Para modelos más complicados de redes se empezó a usar las herramientas de *MATLAB*, cuando las ecuaciones de los modelos se podían implementar en *SIMULINK*. La interpretación de los resultados era más fácil de realizar, cuantitativamente y cualitativamente. (Padrón et al, 2000) pp 338-349.

Luego con la ayuda de las herramientas de Xilinx para el diseño de circuitos implementados en *FPGA's*, en forma lógica o algorítmica ha permitido llevar a cabo las simulaciones e implementaciones tanto en software como en hardware de los modelos que ahora se están desarrollando en el Grupo Académico de Modelado y Simulación de Procesos. El grupo académico ha presentado un desarrollo de funciones para algunas interfaces genéricas sobre el sistema de desarrollo Spartan 3E de Xilinx, además de una interfaz para el control de periféricos entre un *FPGA* y un microcontrolador, en un reporte en los últimos años. (Prieto et al., 2011a). El uso que se dará a las interfaces desarrolladas en este reporte están enfocadas a aplicaciones de visualización (*LCD* y *display* de 7 segmentos), introducción de datos (ratón y teclado) y almacenamiento masivo de datos (memoria y *transceiver*).

Los desarrollos de interfaces asignados a cada tarjeta manufacturada por Xilinx tiene el propósito de que los usuarios puedan emplearlas de acuerdo a sus necesidades. Es decir, el tipo de interfaz que el usuario puede usar y desarrollar conforme al problema que enfrenta para resolver una tarea en particular. Programar los sistemas de desarrollo Xilinx para su implementación en los *FPGA's* en el lenguaje Verilog o VHDL de su conveniencia, hacen que este tipo de tarjetas kit tengan la versatilidad de emplear varias interfaces en cuanto estén disponibles físicamente o sea posible desarrollarlas o programarlas e integrarlas al sistema diseñado.

1.1 Lenguajes de descripción del hardware

Los lenguajes de descripción hardware (*HDL, Hardware Description Language*) surgen de la necesidad de los diseñadores de disponer de un conjunto de herramientas que permitan describir los sistemas digitales de formas diferentes, facilitando el diálogo entre los propios diseñadores, aunque también entre las herramientas de diseño asistido por computadora (*CAD, Computer Aided Design*) y entre diseñadores y herramientas. Los sistemas de desarrollo basados en lenguajes HDL permiten especificar y modelar la estructura y función de un circuito digital; incluyen herramientas para compilar, simular y sintetizar sistemas digitales. Terés L., et al, (1998). Estas herramientas de programación se utilizan para comprobar el funcionamiento del modelo antes de que sea construido. Ahora, los lenguajes *HDL* más utilizados son *VHDL* y Verilog, ya que están estandarizados por *IEEE (Institute of Electrical and Electronic Engineers)*.

VHDL proviene de *VHSIC (Very High Speed Integrated Circuit) Hardware Description Language*. *VHDL* es un lenguaje de descripción del hardware que puede utilizarse para modelar, documentar,

similar, verificar y sintetizar un sistema digital. Por tanto, abarca el ciclo completo de diseño (salvo el trazado físico o layout) desde las especificaciones iniciales hasta la construcción del prototipo hardware. Proporciona el soporte necesario para especificar su comportamiento o estructura, incluyendo jerarquías. Así mismo, es útil para metodologías de diseño ascendentes (*bottom-up*) como, sobre todo descendentes (*top-down*). La semántica y construcciones del lenguaje permiten también diseñar con facilidad bancos de prueba (*test-bench*), mediante los que se lleva a cabo la simulación de los sistemas modelados. Entre las características del VHDL de ámbito general derivado del lenguaje de alto nivel ADA (que es un lenguaje para programar sistemas en tiempo real por excelencia), dispone de tipos abstractos para definir el formato y valores de señales, variables, constantes, etc. Y proporciona amplias facilidades para la realización de algoritmos. Admite casi todos los niveles de descripción, desde el algorítmico (nivel más alto) hasta el lógico (nivel más bajo). Para ello proporciona herramientas semánticas y sintácticas que se pueden agrupar como:

- Aquellas orientadas a las descripciones del comportamiento del circuito.
- Aquellas orientadas a la descripción de las relaciones entre los distintos bloques de un circuito, es decir, su estructura.

Al realizar una descripción VHDL es importante elegir adecuadamente el nivel de función del objetivo perseguido. Por ejemplo, si se pretende sintetizar la descripción realizada, es decir, si el objetivo final es obtener un conjunto de puertas e interconexiones, no se debe emplear el nivel algorítmico, pues en general las herramientas actuales de síntesis no procesan de forma eficiente estas descripciones. Sin embargo, este nivel sí es adecuado cuando el objetivo es comprobar que un sistema complejo funciona correctamente, pues se puede describir y simular de una forma rápida y eficaz.

El lenguaje VHDL presenta tres métodos básicos para describir un circuito digital por software:

- El nivel algorítmico es el que presenta un mayor grado de abstracción (alto nivel), aquí el diseñador solo describe el comportamiento del sistema, sin preocuparse de las señales o componentes internos del mismo. También llamado nivel de comportamiento o *behavior*.
- El nivel RTL (*Register-Transfer Level*, nivel de transferencia de registros) proporciona un cierto grado de abstracción con respecto al *hardware*, pero el diseñador se ve obligado a describir las distintas señales que interactúan en un circuito y su comportamiento en función de las entradas por medio de ecuaciones lógicas y sentencias de asignación. También conocido como nivel de flujo de datos.
- El nivel lógico describe la estructura interna de un circuito basándose en unos componentes básicos definidos previamente (equivaldría a un diagrama lógico).

Toda descripción en VHDL está constituida al menos por tres tipos de elementos: bibliotecas (*libraries*), entidades (*entities*) y arquitecturas (*architectures*). El diseñador generalmente realiza la descripción de las entidades empleando elementos almacenados en las bibliotecas. En la declaración de la entidad se define el diseño como si fuera un producto encapsulado, indicando el número de pines, los puertos de entrada y salida. La entidad puede definir bien las entradas y salidas de un circuito integrado por diseñar o puede definir la interfaz de un módulo que será utilizado en un diseño más grande. Para que la descripción de un circuito sea completa se necesita, además de su declaración como entidad, una especificación de su funcionamiento, la arquitectura es la encargada de ello. Es posible realizar descripciones del funcionamiento del circuito, por lo que pueden existir varias arquitecturas para una misma entidad. (Baena, 2010).

1.2 Lenguajes de descripción de Hardware versus desarrollo de Software

Desde siempre las distintas herramientas y entornos de *CAD* han hecho uso de lenguajes y formatos para describir y manejar los distintos objetos del diseño electrónico. Algunos de estos lenguajes eran notaciones explícitas para realizar las descripciones de entrada a una cierta herramienta; mientras que otros son formatos intermedios propios de cada herramienta para agilizar sus procesos y casi siempre transparentes y desconocidos para el usuario. Sin embargo, todos ellos se limitaban al entorno de *CAD* que les era propio. Los *HDL's*, además del grado de estandarización que supusieron, adoptaron conceptos de la ingeniería del *software* para la descripción y modelado del *hardware*. Desde el punto de vista de la sintaxis, los *HDL's* son muy similares a los lenguajes de programación de alto nivel (*HLL*, *High Level Languages*) para el desarrollo de *software*. De hecho, en muchos casos, el *HDL* se deriva de un *HLL*, así el Verilog tiene muchas reminiscencias del lenguaje "C"; mientras que el *VHDL* procede del *ADA*, de quien hereda muchos de sus conceptos, propiedades y estructuras. Estas similitudes sintácticas entre ambos tipos de lenguajes, si bien pueden facilitar el aprendizaje inicial y el desarrollo de modelos de hardware a aquellos ingenieros con experiencia en programación, también encierran ciertos peligros, pues en aquellos casos en que el modelo en *HDL* se realiza para su posterior síntesis e implementación, el diseñador debe pensar y describir en términos de *hardware* aunque la sintaxis sea de *software*. Al usar estrategias de *Software* será aconsejable cuando el modelo a desarrollar sea exclusivamente para simulación, pues en ese caso se tratará de optimizar el modelo de cara a la ejecución de las distintas etapas y procesos de simulación. Los lenguajes de descripción de hardware (*HDL*) son lenguajes de alto nivel, similares a los de programación (*C*, *PASCAL*, *ADA*, etc.), con una sintaxis y semántica definidas para facilitar el modelado y descripción de circuitos electrónicos, desde las celdas de base de un *ASIC* hasta sistemas completos, pudiéndose realizar estas descripciones a distintos niveles de abstracción, precisión y estilos de modelado, tal y como se muestran en la Figura (1).

Los *HDL* nacen para modelar el comportamiento de un componente de cara a su simulación, aunque también se utilizan para describir el diseño de un circuito para su implementación a través de etapas de síntesis validadas vía la simulación, Figura (2).

Mientras en *software* se recurre a los lenguajes de alto nivel para implementar los algoritmos de forma independiente del procesador que los va a ejecutar, en el caso del hardware son los *HDL*, quienes permiten descripciones de los circuitos a alto nivel de abstracción e independientes de la implementación tecnológica final, Figura (2a).

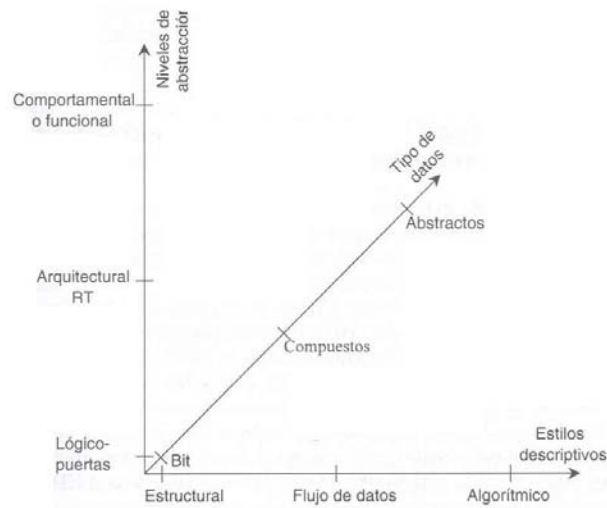


Figura 1. Niveles de abstracción/precisión y estilos de modelado VHDL.

A partir de tales descripciones, los procesos de diseño descendente “*top-down*” aplican procedimientos progresivos de síntesis, dando lugar a descripciones más detalladas de la implementación hasta alcanzar una descripción física concreta totalmente dependiente de la tecnología seleccionada. La validación de las distintas descripciones se realiza mediante los correspondientes procesos de simulación y análisis y las iteraciones de corrección resultantes, Figura (2b).

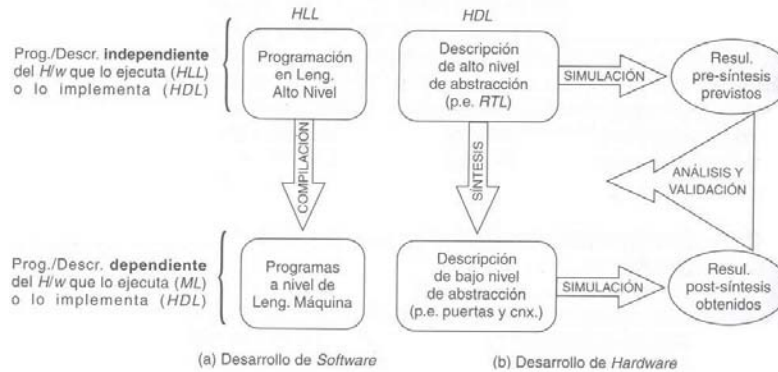


Figura 2. Desarrollo en software versus hardware: (a) niveles de abstracción y lenguajes de alto nivel y (b) esquema básico del diseño descendente con HDL.

1.3 Descripción de la tarjeta Spartan 3E Starter XC3S500E de Xilinx

La tarjeta Spartan 3E XC3S500E Starter suministra un poderoso y altísimo avance contenido dentro del desarrollo de la plataforma para diseños enfocados a la Spartan 3E FPGA de la empresa Xilinx. Este presenta un Puerto de 500K de Spartan 3E FPGA con un procesador RISC de 32 bit e interfaz DDR. La unidad “Spartan-3E starter Kit Board” proporciona las herramientas necesarias para probar el sistema. Incluye la placa de pruebas con una FPGA Spartan-3E modelo XC3S500E, fuente de alimentación, cable USB para programar el dispositivo, software para sintetizar e implementar los modelos diseñados y manuales de referencia. La tarjeta también presenta una plataforma flash Xilinx, USB y una interfaz programable paralela JTAG con numerosas opciones de configuración de FPGA a través de la tarjeta Intel Strata Flash y el ST Microelectronics Serial Flash. La tarjeta es compatible con todas las versiones de la herramienta ISE Xilinx incluyendo el WebPack gratuito. La tarjeta diseñada con fuente de poder puede ser implementada con costos no tan altos. La tarjeta Spartan 3E Starter es también compatible con el Kit de desarrollo embebido MicroBlaze y PicoBlaze de Xilinx. (Spartan-3E FPGA Starter Kit Board User Guide, 2008). Figura (3).



Figura 3. Tarjeta de desarrollo Spartan-3E fabricada por la empresa Xilinx.

A continuación se presentan algunas características de la tarjeta:

- Dispositivos Xilinx integrados:
 - o FPGA de la familia Spartan-3E modelo XC3S500E-4FG320C (con 232 pines de E/S para el usuario, 1164 CLBs o bloques lógicos configurables y 10000 celdas o módulos lógicos)
 - o Controlador CPLD CoolRunner-II CPLD (XC2C64A-5VQ44C)
 - o 4 Mbits de memoria PROM Platform Flash
 - o Incluye una Maxim DS2432 serial EEPROM con un SHA-1 integrado (firma digital)
- Reloj: oscilador de 50 [MHz]
- Otra memoria incluida:
 - o 128 [Mbits] de memoria Parallel Flash
 - o 16 [Mbits] de memoria SPI (Serial Peripheral Interface) Flash
 - o 64 [MBytes] de memoria DDR SDRAM
- Interfaces y conectores:
 - o 4 interruptores tipo pulsador
 - o 4 salidas DAC (Digital-to-Analog Converter)
 - o 2 entradas ADC (Analog-to-Digital Converter)
 - o Conexión Ethernet 10/100
 - o Puerto USB de programación
 - o 2 puertos serie RS-232
 - o Puerto PS/2 para ratón o teclado
 - o 4 interruptores tipo slide
 - o Puerto VGA para despliegue vía un conector DB15
 - o Puerto para un Reloj Auxiliar: Oscilador
- Plataforma flash Xilinx XCF04 para almacenar configuraciones de la FPGA
- 2MB ST Microelectronics Serial Flash
- Fuente de poder con tecnología lineal
- Texas Instruments TPS75003 con CI de manejo de Triple-Supply Power
- SMSC LAN83C185 Ethernet PHY
- 8 led de prueba
- Display: Pantalla LCD de 2 líneas de 16 caracteres.

2. Interfaces de Comunicación y de usuario

2.1 Indicadores y despliegue

Led.- ("Light-Emitting Diode", diodo emisor de luz en español) se refiere a un componente optoelectrónico pasivo, más concretamente un diodo que emite luz. (<http://es.wikipedia.org/wiki/Led>, 2012/12). Los led se usan como indicadores en muchos dispositivos y en iluminación. Los primeros led emitían luz roja de baja intensidad, pero los dispositivos actuales emiten luz de alto brillo en el espectro infrarrojo, visible y ultravioleta. Debido a sus altas frecuencias de operación son también útiles en tecnologías avanzadas de comunicaciones. Los led infrarrojos también se usan en unidades de control remoto de muchos productos comerciales incluyendo televisores e infinidad de aplicaciones de hogar y consumo doméstico. Los led presentan muchas ventajas sobre las fuentes de luz incandescente y fluorescente, principalmente por el bajo consumo de energía, mayor tiempo de vida, tamaño reducido, durabilidad, resistencia a las vibraciones, reducen la emisión de calor, no contienen mercurio (el cual al exponerse en el medio ambiente es altamente venenoso), en comparación con la tecnología fluorescente, no crean campos magnéticos altos como la tecnología de inducción magnética, con los cuales se crea mayor radiación residual hacia el ser humano; cuentan con mejor índice de producción cromática que otros tipos de luminarias, reducen ruidos en las líneas eléctricas, son especiales para utilizarse con sistemas fotovoltaicos (paneles solares) en comparación con cualquier otra tecnología actual; no les afecta el encendido intermitente (es decir pueden funcionar como luces estroboscópicas) y esto no reduce su vida promedio, son especiales para sistemas anti-explosión ya que cuentan con un material resistente, y en la mayoría de los colores (a excepción de los led azules), cuentan con un alto nivel de fiabilidad y duración. Figura (4).



Figura 4. Led de diferentes tipos y colores.

Los led con la potencia suficiente para la iluminación de interiores son relativamente caros y requieren una corriente eléctrica más precisa, por su sistema electrónico para funcionar con voltaje alterno, y requieren de disipadores de calor cada vez más eficientes en comparación con las bombillas fluorescentes de potencia equiparable.

Cuando un led se encuentra en polarización directa, los electrones pueden recombinarse con los huecos en el dispositivo, liberando energía en forma de fotones. Este efecto es llamado electroluminiscencia y el color de la luz (correspondiente a la energía del fotón) se determina a partir de la banda de energía del semiconductor. Por lo general, el área de un led es muy pequeña (menor a 1 mm^2), y se pueden usar componentes ópticos integrados para formar su patrón de radiación. Para conectar led de modo que iluminen de forma continua, deben estar polarizados directamente, es decir, con el polo positivo de la fuente de alimentación conectada al ánodo y el polo negativo conectado al cátodo. Además, la fuente de alimentación debe suministrarle una tensión o diferencia de potencial superior a su tensión umbral. Por otro lado, se debe garantizar que la corriente que circula por ellos no exceda los límites admisibles, lo que dañaría irreversiblemente al led. (Esto se puede hacer de manera sencilla con una resistencia R en serie con los led). Figura (5).

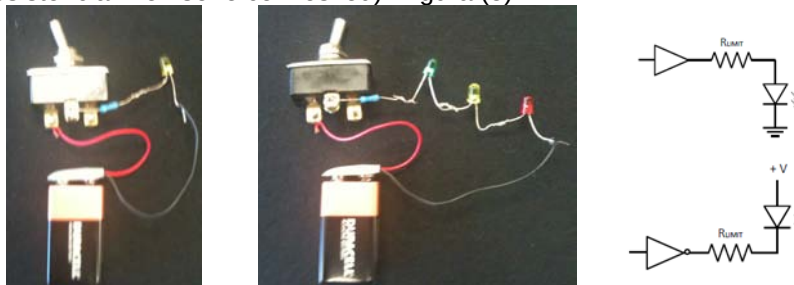


Figura 5. Configuración de conexiones de los led, R_{LIMIT} para I_{LED} aproximadamente 20 [mA].

La tarjeta kit Spartan-3E *FPGA* tiene ocho *LED* individuales de montaje superficial situados por encima de los interruptores deslizantes según lo muestra la Figura (6). Los *LED* se etiquetan desde el *LED7* hasta el *LED0*. *LED7* es el *LED* extremo izquierdo, *LED0* el *LED* de derecha. Véase Figura (6)

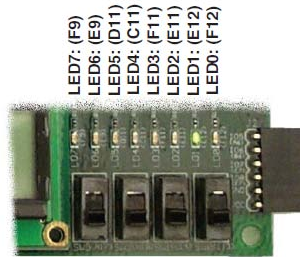


Figura 6. Posición de los 8 led en la Spartan-3E User Guide, (2008).

Cada *LED* tiene un lado conectado con la tierra y el otro lado conectado con un pin en el dispositivo de Spartan-3E vía una resistencia limitadora de 390 [ohm]. Para encender un *LED* individual, maneje la señal de control del *FPGA* asociada en alto. La Figura (7) proporciona los parámetros de UCF para los 8 led, incluyendo el pin de asignación de la entrada-salida, el estándar de la entrada-salida usado, tiempo de respuesta de la salida, y el manejo de la corriente de salida.

NET "LED<7>"	LOC = "F9"	IOSTANDARD = LVTTTL	SLEW = SLOW	DRIVE = 8 ;
NET "LED<6>"	LOC = "E9"	IOSTANDARD = LVTTTL	SLEW = SLOW	DRIVE = 8 ;
NET "LED<5>"	LOC = "D11"	IOSTANDARD = LVTTTL	SLEW = SLOW	DRIVE = 8 ;
NET "LED<4>"	LOC = "C11"	IOSTANDARD = LVTTTL	SLEW = SLOW	DRIVE = 8 ;
NET "LED<3>"	LOC = "F11"	IOSTANDARD = LVTTTL	SLEW = SLOW	DRIVE = 8 ;
NET "LED<2>"	LOC = "E11"	IOSTANDARD = LVTTTL	SLEW = SLOW	DRIVE = 8 ;
NET "LED<1>"	LOC = "E12"	IOSTANDARD = LVTTTL	SLEW = SLOW	DRIVE = 8 ;
NET "LED<0>"	LOC = "F12"	IOSTANDARD = LVTTTL	SLEW = SLOW	DRIVE = 8 ;

Figura 7. Asignación de los pin en el archivo .UCF del área restringida.

Displays de 7 segmentos.- Se inicia aprendiendo como manejar un solo dígito de 7 segmentos. Un *display* de 7 segmentos, no es ni más ni menos que un conjunto de 7 led conectados y posicionados apropiadamente. Encendiendo algunos de ellos y apagando otros podemos ir formando diferentes números. La disposición de los segmentos y la distribución de un integrado de sus terminales se muestran en la Figura (8).

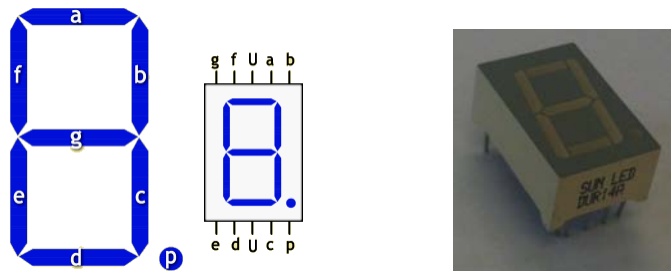


Figura 8. Clasificación de los segmentos.

En la Figura (8) cada segmento esta designado con una letra y el punto decimal se denomina "P". A la derecha está una representación del encapsulado con los pines para conectarlo a un circuito. A cada pin o pata del encapsulado se le asigna la letra correspondiente del segmento. Esto significa que, por ejemplo, con el pin "a" se puede controlar el estado del segmento "a"(encenderlo o apagarlo). Además en el encapsulado se muestran dos patillas llamadas "U", cuya función se explica a continuación. <http://tecnoface.com/tutoriales/12-practicando-con-displays-de-7-segmentos/19-conceptos-basicos-de-un-display-de-7-segmentos,2012/12>.

Entonces, se tienen 8 led colocados en forma de un dígito con punto decimal. Ahora bien, un led tiene dos extremos, *ánodo* y *cátodo*. Como en total se tienen 8 led, debería tener 16 extremos (8 ánodos y 8 cátodos), sin embargo el encapsulado solo tiene 10. Esto se hace para reducir el tamaño del encapsulado y se logra de la siguiente manera. Los 8 led se interconectan internamente de tal forma que solo se puede acceder a uno de los dos extremos de cada led. Los extremos sobrantes de cada led se conecta internamente con los demás, y este punto de unión se encuentra disponible desde el exterior del encapsulado. Debido a este artificio, se tienen dos tipos de *display* de 7 segmentos:

1) Ánodo Común: es aquel donde los ánodos de todos los led se conectan internamente al punto de unión U y los cátodos se encuentran disponibles desde afuera del integrado.

2) Cátodo Común: es aquel donde los cátodos de todos los led se conectan internamente al punto de unión U y los ánodos se encuentran disponibles desde afuera del integrado.

Estas definiciones pueden parecer confusas así que se puede observar gráficamente ambos tipos de *display* y su implementación en un circuito. Veamos el cátodo común en la Figura (9).

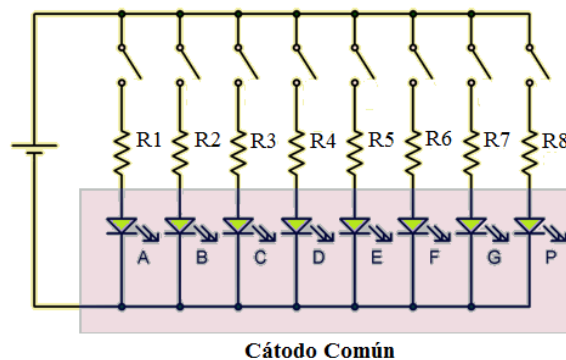


Figura 9. Configuración en cátodo común.

En el circuito anterior, el rectángulo gris representa el dígito de 7 segmentos en forma esquemática. Se observa que en el esquemático se representa cada segmento con un led, esto facilita entender el funcionamiento del circuito. En el esquema se ve claramente la conexión interna de los cátodos de todos los led, dejando disponible externamente solo el punto de su unión.

Con respecto al funcionamiento del circuito, también es muy fácil comprender lo que sucede, mientras las 8 llaves o interruptores están abiertas, no circula ninguna corriente y los 8 led están apagados. Al cerrar cualquiera de las llaves, por la misma circulará una corriente que hace encender el led correspondiente. Por ejemplo, si se cierra la 2da y 3era llave, se encenderán los segmentos "b" y "c" y en el *display* aparece el número 1.

Ahora se muestra el circuito con ánodo común en la Figura (10).

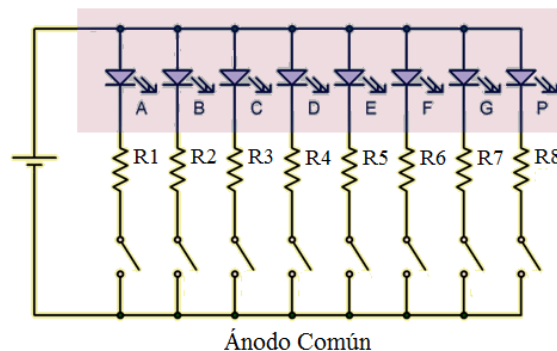


Figura 10. Configuración en ánodo común.

Se puede fácilmente notar la similitud y la diferencia con el circuito de la Figura (9). En este caso, son los ánodos los que se encuentran conectados internamente y por tal razón el punto unión ahora se conecta al terminal positivo de la batería. Nuevamente, cerrando cualquiera de las llaves, se encenderá el segmento correspondiente. Figura (11).

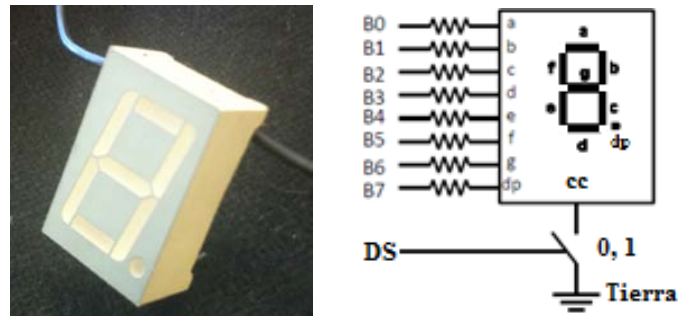


Figura 11. Encapsulado del display de 7 segmentos y punto decimal.

Los ánodos de todos los segmentos (punto de unión *U* o pata 8 del integrado, Figura 8) están conectados al terminal positivo de la fuente. En el cátodo de cada segmento se conecta la resistencia limitadora de corriente y esta a su vez es conectada al control de encendido de los led correspondientes. Al cerrar el interruptor correspondiente al segmento este se ilumina. Entonces para ir formando los números en el *display* se usarán las llaves para cerrar y dejar abiertas las que no se encienden. Esto se resume en la tabla (1a) y (1b).

Tabla 1. a) Configuración para encendido de los led para formar los dígitos para el display de ánodo común. b) Lógica binaria para el encendido de los led y su equivalente en hexadecimal.

Segmento	a	b	c	d	e	f	g
Claves	1	2	3	4	5	6	7
0	C	C	C	C	C	C	A
1	A	C	C	A	A	A	A
2	C	C	A	C	C	A	C
3	C	C	C	C	A	A	C
4	A	C	C	A	A	C	C
5	C	A	C	C	A	C	C
6	C	A	C	C	C	C	C
7	C	C	C	A	A	A	A
8	C	C	C	C	C	C	C
9	C	C	C	C	A	C	C

a) ☐ C - Cerrada ☐ A - Abierta

Número	Segmentos	Hexadecimal
0	0011 1111	0x3F
1	0000 0110	0x06
2	0101 1011	0x5B
3	0100 1111	0x4F
4	0110 0110	0x66
5	0110 1101	0x6D
6	0111 1101	0x7D
7	0000 0111	0x07
8	0111 1111	0x7F
9	0110 0111	0x67

b)

Como se puede observar, haciendo la combinación de los interruptores abiertos/cerrados, o 1s y 0s, según se indica se puede mostrar cualquier dígito en el *display* y si se requiere algunos caracteres como A, B, C, D, E, F. Posteriormente se mostrarán las características del integrado junto con el diagrama esquemático que se usó para implementar la interfaz visual con el usuario, los *display* de 7 segmentos serán controlados por la aplicación desarrollada en VHDL. También se mostrarán sus conexiones con la tarjeta Spartan 3E para encender cada *display*.

Despliegue de matrices de puntos.- Una matriz de LED consiste en un arreglo de LED que pueden ser encendidos y apagados individualmente desde un microcontrolador. Pueden pensar en ella como una pantalla de pocos pixeles en los cuales pueden presentar gráficos y textos, tanto estáticos como en movimiento. Figura (12).



Figura 12. Integrado de una matriz de led de 5x7 y sus terminales de conexión.

La Figura (13) presenta una matriz de 5x7 LED, donde las columnas corresponden a los cátodos (deben ir conectados a tierra) y las filas corresponden a los ánodos (deben ir conectados a voltaje).

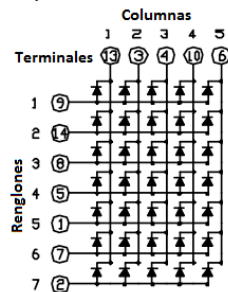


Figura 13. Matriz de 5x7 con columnas de cátodos.

Para encender un LED específico debe ubicarse la intersección entre la columna y la línea correspondiente. Por ejemplo, para encender el LED que se muestra en la figura siguiente, la columna 3 (C3) debe ser conectada a tierra (a través de una resistencia de 220 ohm) mientras que la fila 5 (R5) es conectada a voltaje 5 [V]. Ejemplo de esto se muestra en la Figura (14).

(<http://galaxi0.wordpress.com/about/salidas-y-entradas-digitales/matriz-de-leds/>, 2012/12)



Figura 14. Encendido del led columna 3 fila 5 de la matriz de 5x7.

También existen matrices de led de 8x8 que usan la combinación RGB, “red-green-blue” para mostrar casi cualquier color, éstas muestran más capacidad de manejar cada uno de los led para formar Figuras, Figura (15). Todo depende de la aplicación que se requiere llevar a cabo para interconectar estos integrados. Una aplicación usual es la de formar un cartel o letrero luminoso alfanumérico con el uso de estas matrices de led.



Figura 15. Matriz de 8x8 RGB y una aplicación directa el Bitman desarrollado por Ryota Kuwakubo.

Cuando se genere la interfaz para estas matrices se mostrarán sus características de los integrados y sus conexiones con la Spartan 3E, los led serán controlados por la aplicación desarrollada en VHDL.

Codificación y refresco mediante SW.- Para aplicaciones sencillas generalmente con *displays* de 7 segmentos. El microcontrolador realiza las funciones de codificar los datos (BCD a 7 segmentos) y del refresco periódicamente de cada dígito del *display*. En la interfaz desarrollada no se empleará el microcontrolador para esta tarea en específico, se programa directamente en VHDL el refresco. Figura (16 y 17).

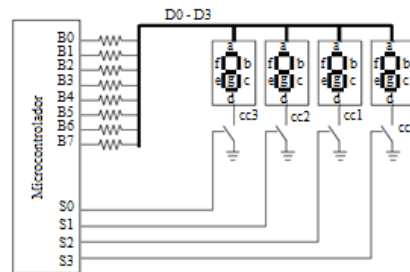


Figura 16. Diagrama esquemático para hacer el refresco del display de 7 segmentos.

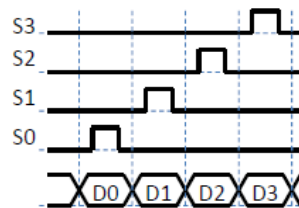


Figura 17. Diagrama de tiempos del refresco del display de 7 segmentos.

Codificación y refresco mediante HW.- Para aplicaciones sencillas con *displays* de 7 segmentos o matrices de led. Un integrado almacena en su memoria interna el contenido del *display* y refresca de forma periódica los led. El microcontrolador comunica los datos a representar. Estos sistemas suelen incorporar control de luminosidad. Figura (18).

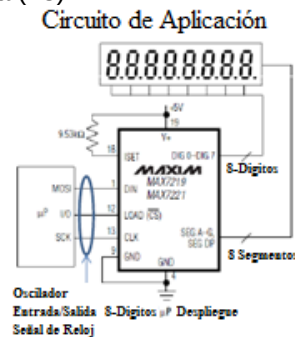


Figura 18. Refresco de display mediante hardware con interfaz en serie.

Display LCDs gráficos.- Una pantalla de cristal líquido o LCD "liquid crystal display", es una pantalla delgada y plana formada por un número de píxeles en color o monocromos colocados delante de una fuente de luz o reflectora. A menudo se utiliza en dispositivos electrónicos de pilas, ya que utiliza cantidades muy pequeñas de energía eléctrica. Incorporan interfaz serie o paralelo con el procesador, el *display* LCD y circuitos drivers, memoria y controlador. El microcontrolador envía comandos (borrar, posicionar el cursor...) y los datos a representar. Figura (19 y 20).

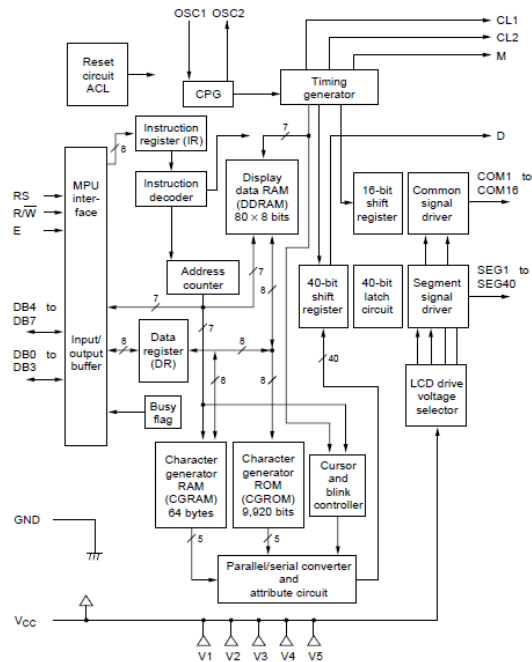


Figura 19. Esquema genérico de un display LCD, tomado de Hitachi HD44780U (LCD-II), (1998).



Figura 20. Un LCD de 20 columnas por 4 renglones.

Dentro de la tarjeta Spartan 3E está integrado un LCD cuyas características generales se muestran a continuación. Primero en la Figura (21), se muestra el esquema del sistema de visualización que usará los controles del LCD para el envío de mensajes sobre éste, una vez que se han adquirido y procesado los datos además de almacenarse en la memoria del FPGA.

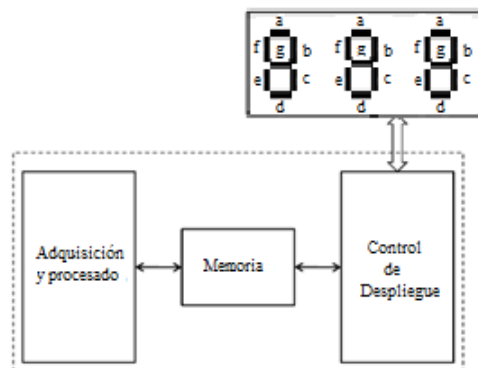


Figura 21. Sistema de visualización en FPGA.

A parte de los comandos para los controles del LCD de la Spartan 3E de 16x2 (posicionamiento del cursor, borrado, etc.) se necesita el manejador que permitirá el envío de datos. Así como la codificación de datos en el controlador sobre el display alfanumérico. Figura (22).

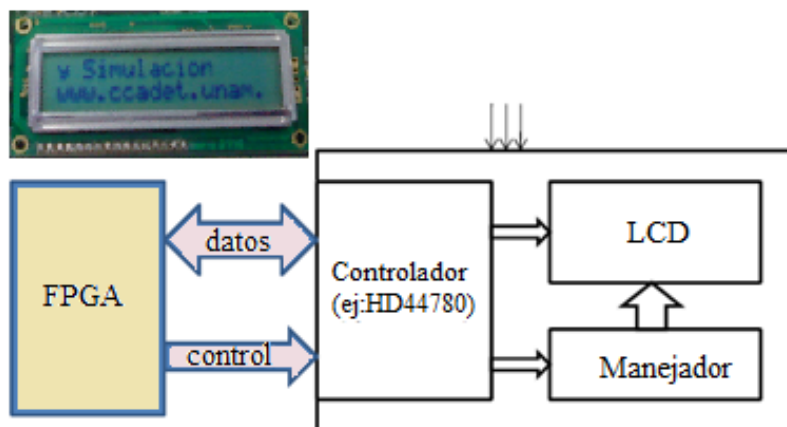


Figura 22. Placa de desarrollo Spartan 3E: Display LCD de 16 X 2.

La interfaz en paralelo que emplea el LCD tiene los datos (comandos y caracteres) sobre: D7 a D4, que es una interfaz de 4 bits. Además tiene los controles E, RS y RW. Con otras entradas en VDD, VSS y VADJ. La tarjeta tiene un *display* alfanumérico HD44780 integrados para los comandos en el controlador. Figura (23).

Instrucción	Código										Descripción	Tiempo de Ejecución (máximo cuando Fp o fosc es 270 [KHz])
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Borrar Despliegue	0	0	0	0	0	0	0	0	0	1	Borra todo el despliegue y coloca DDRAM en 0 y direcciona el contador.	
Regresa a inicio	0	0	0	0	0	0	0	0	1	—	Coloca DDRAM en 0 direcciona contador. Regresa el despliegue al inicio y desplaza a la posición original. DDRAM sin cambios.	1.52 ms
Coloca el modo	0	0	0	0	0	0	0	1	LD	S	Coloca al cursor en dirección del movimiento y desplaza el despliegue. Estas operaciones son durante la escritura y lectura.	37 µs
Despliega el control "on/off"	0	0	0	0	0	0	1	D	C	B	Coloca el despliegue (D) "on/off", Cursor (C) "on/off", y destellea el cursor en la posición del carácter (B).	37 µs
Desplaza despliegue o Cursor	0	0	0	0	0	1	S/C	R/L	—	—	Mueve el cursor y desplaza el despliegue sin cambiar el con- tenido de la DDRAM.	37 µs
Coloca la Función	0	0	0	0	1	DL	N	F	—	—	Coloca la interfase de longitud (DL), número de líneas desplegadas (N) y la fuente del carácter (F).	37 µs
Coloca la dirección CGRAM	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Direcciona CGRAM. Datos del CGRAM son enviados y recibidos despues de este comando.	37 µs
Coloca la dirección DDRAM	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Direcciona DDRAM. Datos del DDRAM son enviados y recibidos despues de este comando.	37 µs
Lee bande- ra ocupada y direcciona	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Lee la bandera ocupada (BF) indicando operación interna inicia- lizada y necesita direccionamiento del contador.	0 µs

Figura 23. Comandos del controlador de display alfanumérico HD44780.

La figura (24) muestra la codificación del controlador.

		Datos Altos del Nibble															
		DB7	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
		DB6	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1
		DB5	0	1	1	0	0	1	1	1	1	0	0	1	1	1	1
		DB4	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Datos Bajos del Nibble	xxxx0000	8g RAM															
	xxxx0001		!	1	A	Q	a	q	.	7	チ	4	ä	q			
	xxxx0010		"	2	B	R	b	r	'	イ	ツ	×	ß	θ			
	xxxx0011		#	3	C	S	c	s	」	ウ	テ	ε	ω				
	xxxx0100		\$	4	D	T	d	t	、	エ	ト	μ	Ω				
	xxxx0101		%	5	E	U	e	u	・	オ	ナ	1	ç	ü			
	xxxx0110		&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ			
	xxxx0111		'	7	G	W	g	w	ヲ	キ	ヲ	ラ	q	π			
	xxxx1000		(	8	H	X	h	x	イ	ク	ネ	リ	フ	ア			
	xxxx1001		)	9	I	Y	i	y	ウ	ケ	ル	ル	U				
	xxxx1010		*	:	J	Z	j	z	エ	コ	ハ	レ	i	チ			
	xxxx1011		+	;	K	[	k	[	オ	サ	ヒ	ロ	*	ア			
	xxxx1100		,	<	L	¥	l	¥	ハ	シ	フ	ワ	Φ	ア			
	xxxx1101		-	=	M	]	m	]	ユ	ズ	ヘ	ン	も	÷			
	xxxx1110		.	>	N	^	n	^	ヨ	セ	ホ	°	ñ				
	xxxx1111		/	?	O	_	o	_	←	ッ	ツ	マ	°	ö	■		

Figura 24. Codificación de datos en controlador de display alfanumérico, Spartan-3E User Guide, (2008).

Display de video VGA.- La Spartan 3E también incluye una interfaz de video VGA vía un conector DB15 que puede usar un monitor para enviar mensajes de mayor envergadura, así como el diseño de imágenes y video, Figura (25). Para el empleo de esta interfaz hay que hacer uso del manejador o acoplarlo para la tarea específica a desarrollar y conocer el tipo de señales que se usan para ello.

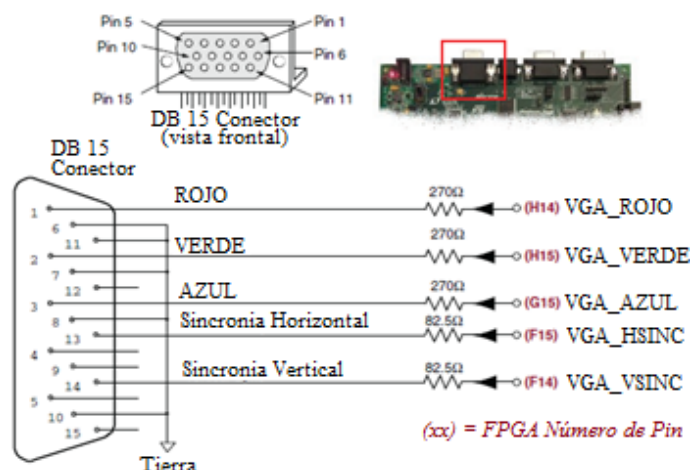


Figura 25. Placa de desarrollo Spartan 3E: Interfaz de video VGA, Spartan-3E User Guide, (2008).

Este manejador usa resistencias en serie para cada color de acuerdo al valor del bit del *VGA_RED*, *VGA_GREEN* y *VGA_BLUE*. Las resistencias en serie en combinación con la terminal de 75 [ohm] construida para el cable construido VGA, asegura que las señales de color permanecen en un intervalo VGA específico entre 0 y 70 [V]. Las señales de *VGA_HSYNC* y de *VGA_VSYNC* usan los niveles del manejador estándar de la entrada-salida *LVTTL* o *LVC MOS33*. El manejador controla las señales de *VGA_RED*, de *VGA_GREEN*, y *VGA_BLUE* arriba o abajo para generar los ocho colores mostrados en la Tabla (2).

Tabla 2. Código de 3-bit para el control de colores, *Spartan-3E User Guide*, (2008).

VGA_ROJO	VGA_VERDE	VGA_AZUL	Color resultante
0	0	0	Negro
0	0	1	Azul
0	1	0	Verde
0	1	1	Azul Cielo
1	0	0	Rojo
1	0	1	Magenta
1	1	0	Amarillo
1	1	1	Blanco

La sincronización de la señal de VGA se especifica, publica, “copyright” y vendida por la asociación de Video de los Estándares de la Electrónica (*VESA*, *Video Electronics Standard Association*). El sistema de VGA y la información siguientes de la sincronización se proporciona como ejemplo de cómo el *FPGA* puede manejar el modo monitor de VGA en 640x480.

Para una información más exacta o para la información sobre frecuencias más altas de VGA, refiérase a los documentos disponibles en la página Web de *VESA* u otras páginas Web, (<http://www.vesa.org>, 2012/12). La Figura (26) presenta la configuración para el monitor VGA y sus señales de sincronía.

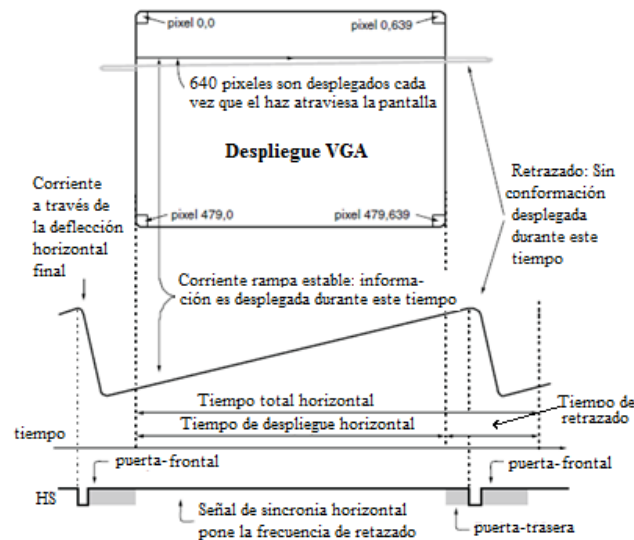


Figura 26. Control de las señales VGA para la interfaz de la tarjeta Spartan 3E (2008).

Las señales temporales de la tabla (3), son derivadas por un *display* de 640 por 480 píxeles usando un reloj de 25 MHz y uno de refresco de 60 Hz de ± 1 . La Figura (27) muestra la relación entre cada uno de los símbolos de la sincronización. La sincronización para el ancho del pulso de sincronía

(TPW) y los intervalos del frente delantero y trasero (TFP y TBP) se basan en observaciones de varios *display* VGA. Los intervalos del frente delantero y trasero son los tiempos del pulso pre y post-sincronía. La información no se puede desplegar durante esos lapsos de tiempo. Tabla 3.

Tabla 3. Control de Tiempos Modo VGA 640x480, *Spartan-3E User Guide*, (2008).

Símbolo	Parámetro	Sincronía Vertical			Sincronía Horizontal	
		Tiempo	Reloj(s)	Líneas	Tiempo	Reloj(s)
T_S	Tiempo Pulso de Sincronía	16.7 ms	416,800	521	32 μ s	800
T_{DISP}	Tiempo de Despliegue	15.36 ms	384,000	480	25.6 μ s	640
T_{PW}	Ancho de Pulso	64 μ s	1,600	2	3.84 μ s	96
T_{FP}	Puerta Frontal	320 μ s	8,000	10	640 ns	16
T_{BP}	Puerta Trasera	928 μ s	23,200	29	1.92 μ s	48

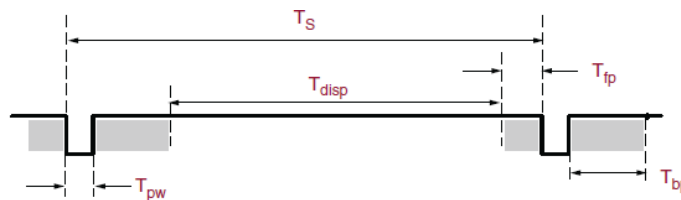


Figura 27. Control del tiempo para la interfaz VGA, *Spartan-3E User Guide*, (2008).

2.2 Interruptores, pulsadores y teclados

Interruptor (dos estados).- La tarjeta del kit Spartan®-3E tiene cuatro interruptores deslizantes, según las indicaciones de la Figura (28). Los interruptores deslizantes están situados en la esquina derecha más baja de la tarjeta y etiquetados SW3 a SW0. El interruptor SW3 es el interruptor extremo izquierdo y SW0 es el interruptor de derecha.

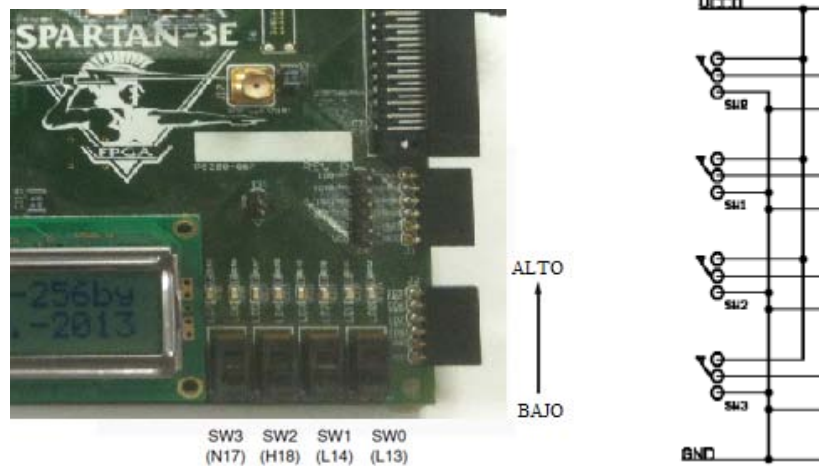
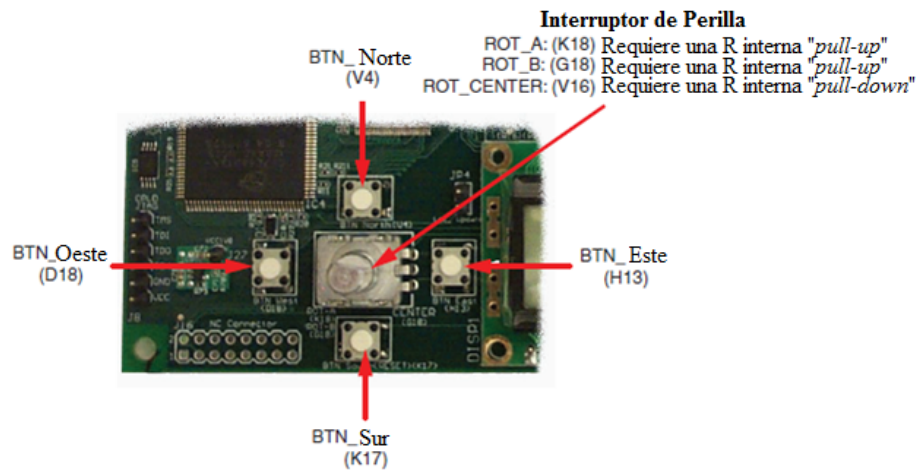


Figura 28. Interruptores deslizantes de la Spartan 3E.

Pulsador.- La tarjeta Spartan-3E tiene cuatro interruptores de botón, mostrados en la Figura (29). Los botones están situados en la esquina izquierda más baja del tablero y etiquetados *BTN_NORTH*,

BTN_EAST, BTN_SOUTH y BTN_WEST. Los “pins” o terminales del FPGA que se conectan con los interruptores de botón aparecen entre paréntesis y el UCF asociado aparece en la Figura (31).



- Notas:** 1. Todas las entradas de los interruptores de botón requieren una resistencia a tierra.
2. El interruptor BTN_SUR es usado como reset del FPGA en algunas aplicaciones.

Figura 29. Cuatro interruptores de Botón “Push-Button” colocados alrededor del interruptor perilla.

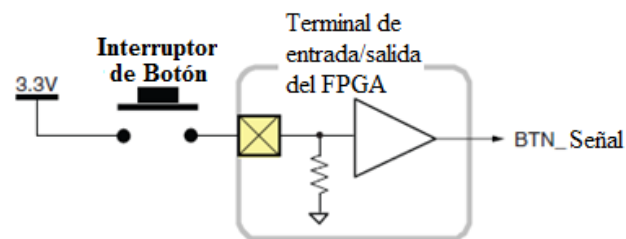


Figura 30. Cada interruptor de botón requiere resistencia interna a tierra en pin de entrada del FPGA.

```
NET "BTN_EAST" LOC = "H13" | IOSTANDARD = LVTTTL | PULLDOWN ;
NET "BTN_NORTH" LOC = "V4" | IOSTANDARD = LVTTTL | PULLDOWN ;
NET "BTN_SOUTH" LOC = "K17" | IOSTANDARD = LVTTTL | PULLDOWN ;
NET "BTN_WEST" LOC = "D18" | IOSTANDARD = LVTTTL | PULLDOWN ;
```

Figura 31. Parámetros para la conexión en el archivo .UCF de los interruptores de botón.

En la parte central de los interruptores de botón se encuentra un interruptor de perilla y su diagrama esquemático se muestra en la Figura (32).

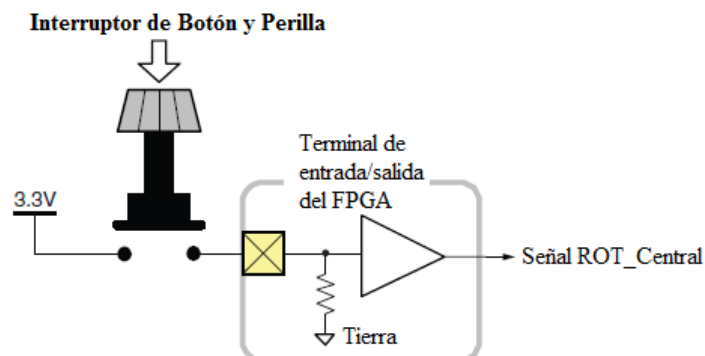


Figura 32. Diagrama del interruptor de Botón y perilla con su resistencia de entrada a tierra.

Eliminación de rebotes.- la eliminación de los rebotes se pueden realizar de dos forma por software o por hardware, esto se muestra en la Figura (33).

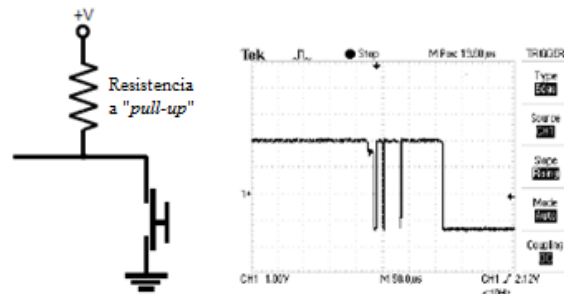


Figura 33. Eliminación de señal rebote del interruptor de botón vista en osciloscopio.

Teclados matriciales.- para comunicación con el usuario suele conectarse un teclado para la introducción de datos que espera el sistema desarrollado para procesar la información solicitada. Este teclado puede ser en forma matricial como se muestra en la Figura (34), donde también se presenta un esquema de interconexión de la matriz.

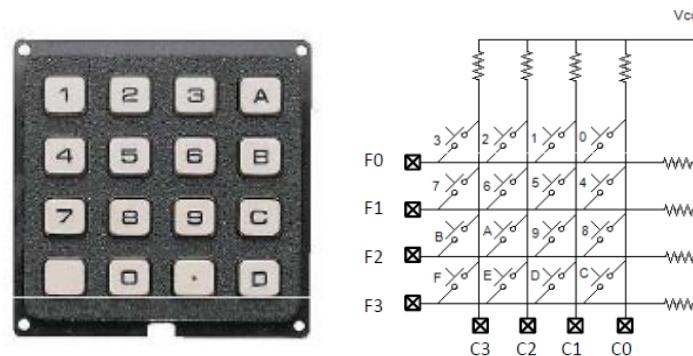


Figura 34. Teclados en forma matricial.

A continuación se presenta un diagrama de flujo para la operación de un teclado en forma matricial, Figura (35).

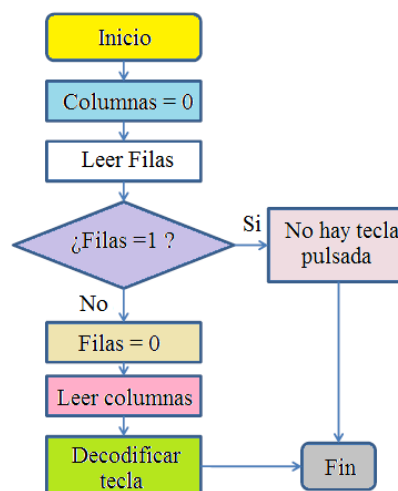


Figura 35. Operatividad del teclado matricial en un diagrama de flujo.

Teclados y ratones: Interfaz serie PS/2.- La tarjeta de desarrollo Spartan 3E incluye un puerto PS/2 con un conector estándar de 6-pin de *mini-DIN*, etiquetado con J14 en la tarjeta. La Figura (36) muestra el conector PS/2 y la Tabla (4) muestra las señales en el conector. Solamente las terminales 1 y 5 se fijan al conector del FPGA.

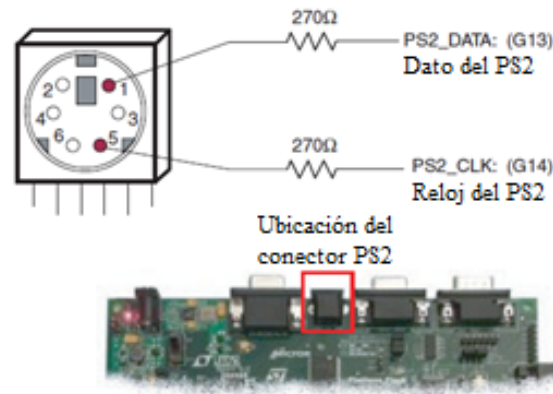


Figura 36. Localización y Señales del Conector PS/2, Spartan-3E User Guide, (2008).

Tabla 4. Salidas del conector PS/2, Spartan-3E User Guide, (2008).

PS/2 DIN Pin	Señal	FPGA Pin
1	Datos (PS2_Data)	G13
2	Reservado	G13
3	Tierra	Tierra
4	+5V	—
5	Reloj (PS2_CLK)	G14
6	Reservado	G13

El ratón y el teclado de la PC utilizan el bus serial de dos hilos PS/2 para comunicarse con un dispositivo anfitrión “host”, el Spartan-3E FPGA en este caso. El bus PS/2 incluye el reloj y los datos, por otro lado el ratón y el teclado manejan el bus con sincronizaciones idénticas de la señal y ambos utilizan 11-bit de palabras que incluyen un bit de inicio, uno de parada y un bit de paridad impar. Sin embargo, los paquetes de los datos se organizan diferentes para un ratón que para un teclado. Además, la interfaz del teclado permite transferencias de datos bidireccionales así que el dispositivo del anfitrión puede encender LED en cada estado en el teclado.

Teclado.- El teclado utiliza un manejador “open-collector” de modo que el teclado o el host anfitrión puedan controlar el “bus” de dos hilos. Si el host nunca envía datos al teclado, entonces el host puede utilizar las terminales como simples entradas. El PS/2-style de teclado emplea códigos de exploración “scan-code” para comunicar los datos proporcionados al presionar alguna tecla del teclado, muchos de los teclados hoy funcionan al estilo PS/2. Cada tecla tiene un simple código, este único código se envía siempre que se presione la tecla correspondiente. Los códigos de la exploración para la mayoría de las teclas aparecen en la Figura (37). Si se presiona la tecla y se deja oprimida, el teclado en varias ocasiones envía el código de la exploración a cada 100 [ms]. Cuando se lanza una tecla, el teclado envía un código de la tecla para arriba “F0”, seguido por el código de la exploración de la tecla lanzada. El teclado envía el mismo código de la exploración, cuando una tecla tiene oprimida la tecla de cambio y no-cambio “shift and non-shift” de minúsculas a mayúsculas dado que los caracteres no cambian de lugar, de igual forma cuando se active la tecla de mayúsculas. El host anfitrión determina cual carácter fue provisto. Algunas teclas, llamadas las llaves extendidas, envían un “E0” delante del código de la exploración y además, puede ser que envíen más de un código de la exploración. Cuando se lanza una

llave extendida, un código de la tecla para arriba de “E0 F0” es enviado, seguido por el código de la exploración.

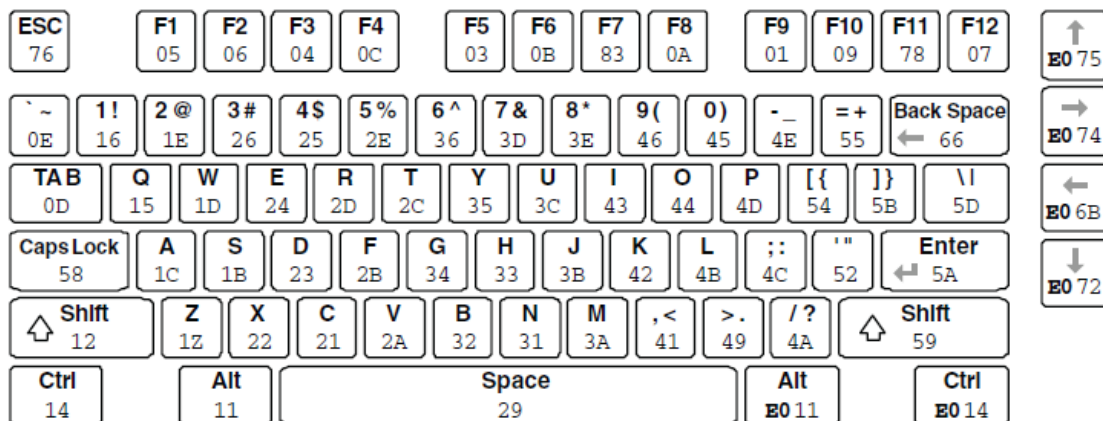


Figura 37. “Scan Code” Código de exploración para el teclado con conexión PS/2, *Spartan-3E User Guide*, (2008).

Ratón.- Un ratón genera una señal de reloj y de datos cuando este se mueve; estas señales permanecen en alto, indicando el estado del *IDLE*. Cada vez que se mueve el ratón, el ratón envía tres palabras de 11-bit hacia el host anfitrión. Cada palabra de 11-bit contiene un bit “0” de inicio, seguido por 8-bits de datos (*LSB* primero), seguido por un bit de paridad impar, y terminando con un bit “1” de parada. Cada transmisión de datos contiene 33-bits en total, donde los bits 0, 11, y 22 son de inicio “0”, y los bits 10, 21, y 32 son de parada “1”. Los tres campos de 8-bits de datos contienen datos del movimiento según las indicaciones de la Figura (38). Los datos son válidos en el flanco que cae del reloj, y el período de reloj es 20 a 30 [KHz].

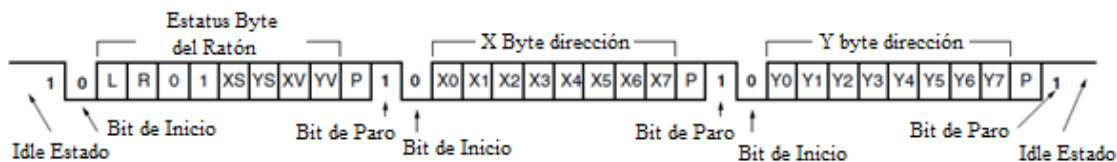


Figura 38. Transiciones del ratón.

Un ratón de *PS/2-style* emplea un sistema coordinado relativo (véase Figura 39), en donde el movimiento del ratón a la derecha genera un valor positivo en el campo de X, y el movimiento a la izquierda genera un valor negativo. Asimismo, el movimiento del ratón para arriba genera un valor positivo en el campo de Y, y el movimiento hacia abajo representa un valor negativo. Los bits XS y YS en el cuadrante de estado definen la muestra de cada valor, donde un ‘1’ indica un valor negativo.

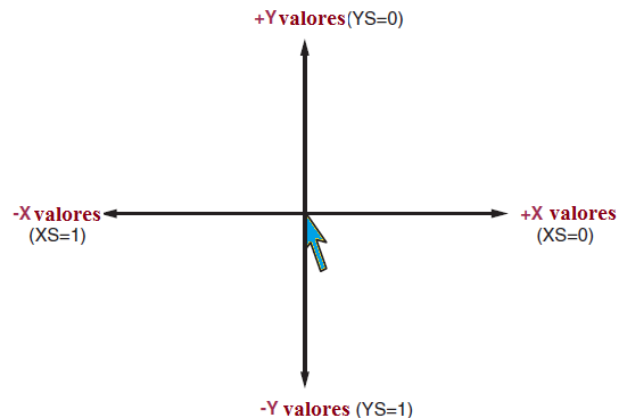


Figura 39. El ratón usa un sistema de coordenadas relativo al movimiento.

La magnitud de los valores de X y de Y representa el índice del movimiento del ratón. Cuanto más grande es el valor, más rápidamente el ratón se está moviendo. Los bits XV y YV en el cuadrante de estado indican cuando los valores de X o de Y exceden su valor máximo, una condición del desbordamiento. Un '1' indica cuando ocurre un desbordamiento. Si el ratón se mueve continuamente, los 33-bits de transmisiones se repiten a cada 50 [ms]. Los campos L y R en el cuadrante de estado indican los botones presionados izquierda y/o derecha del ratón. Un '1' indica que se está presionando el botón de ratón asociado.

Teclados, pulsadores y barras capacitivos.- Existen varias técnicas para hacer funcionar un teclado, los empleados hoy en día son a base de botones o barras que al presionarse envían señales de pulso. (Veáse Figura 40).



Figura 40. Interruptor pulsante, vista posterior, uso en protoboard y dibujo frontal.

Necesitan funciones de temporización + conversión A/D. (véase Figura 41)

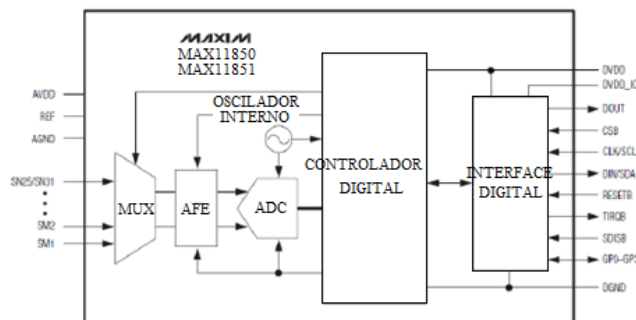


Figura 41. Controlador Digital de Maxim.

Comunicación serie

Arquitectura del sistema.- (Figura 42)

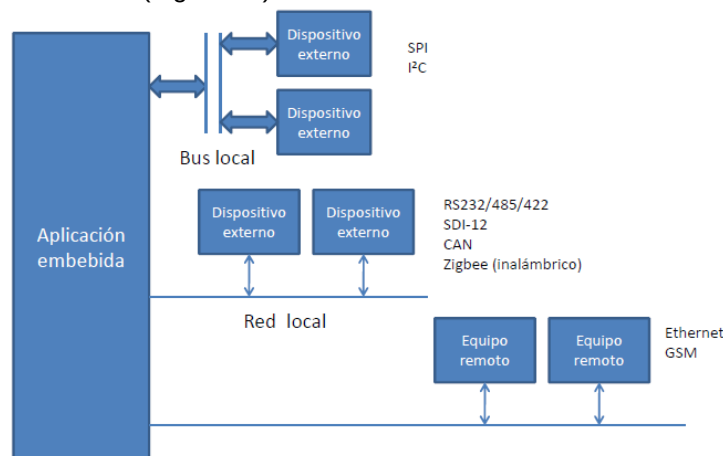


Figura 42. Diagrama de la arquitectura del sistema.

Buses locales

Interfaces de comunicación serie síncronas.- (véase Figura 43)



Figura 43. Estructura Maestro Esclavo.

- Estándar de comunicación SPI (Serial Peripheral Interface)
- Estándar I²C (Inter-Integrated Circuits)

Interfaz SPI: Características.-

Propuesto por Motorola

Señales:

- SCK (Serial Clock)
 - SDI (Serial Data In)
 - SDO (Serial Data Out)
 - SS (Slave Select)
- (Véase Figura 44).

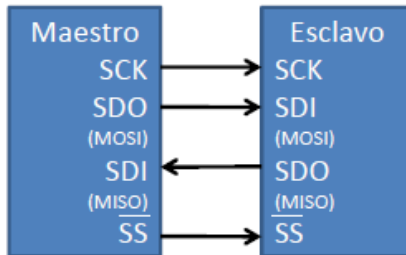


Figura 44. Comunicación Serial Maestro Esclavo.

- ✓ Full duplex (*microwire* es una variante *half_duplex*).
- ✓ HW y protocolo muy sencillo.

Interfaz SPI: Conexiones.- (Figura 45).

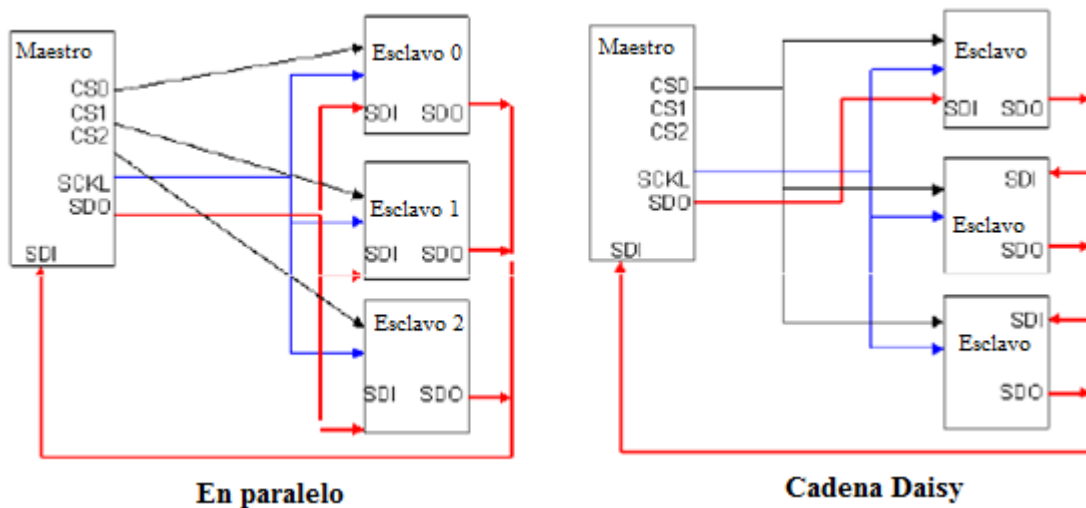


Figura 45. Diagrama del "Serial Data Out" que requiere salida de tres estados.

Interfaz SPI: Modos de funcionamiento.- (Tabla 5)

Dependen de:

- Polaridad de la señal de reloj
- Estado inactivo a nivel bajo
- Estado inactivo a nivel alto
- Fase del reloj
- Bit en SDO antes del primer flanco de reloj
- Bit en SDO tras el primer flanco de reloj

Tabla 5. Modos de funcionamiento.

Modo	Polaridad	Fase
0	BAJO	BIT
1	BAJO	IDLE
2	ALTO	BIT
3	ALTO	IDLE

Interfaz I²C: Características Básicas.-

Desarrollado por Philips Semiconductor,
Version 1.0 1992 Standard (100-Kbps), Fast (400-Kbps)
Version 2.0 1998 High Speed (3.4-Mbps)

Orientado a comunicación de distintos integrados en una placa de circuito impreso

Características (Tabla 6)

- ✓ Serie Síncrono 2 hilos (SDA, SCL)
- ✓ Maestro-Esclavo Dispositivos identificados por una dirección (7 bits o 10bits)
- Bidireccional float high, driven low

Tabla 6. Definiciones del bus I²C, terminología.

Término	Descripción
Transmisor	Dispositivo que envía datos al "bus"
Receptor	Dispositivo que recibe datos desde el "bus"
Maestro	Dispositivo que inicializa una transferencia, genera señales de reloj y termina una transmisión
Esclavo	Dispositivo direccionado por el Maestro
Multi-Maestro	Más que un Maestro puede controlar el "bus" al mismo tiempo sin dañar el mensaje
Arbitro	Proceso para asegurar que, si más de un Maestro intenta controlar el "bus", solo permite que uno lo haga y lograr que el mensaje no se dañe
Sincronización	Proceso para sincronizar las señales de reloj de dos o más dispositivos.

Esquema de Conexión.- (Figura 46)

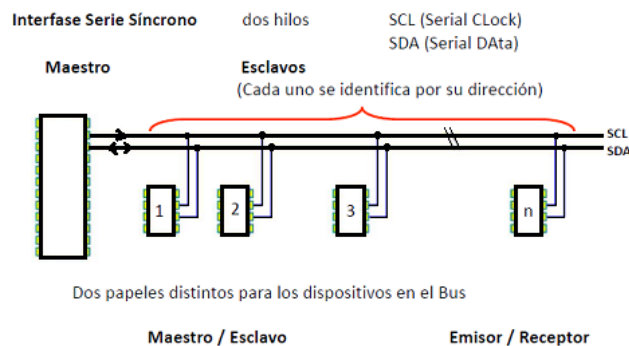


Figura 46. Esquema de conexión para la interfaz I².

Conexión a nivel físico.- (Figura 47)

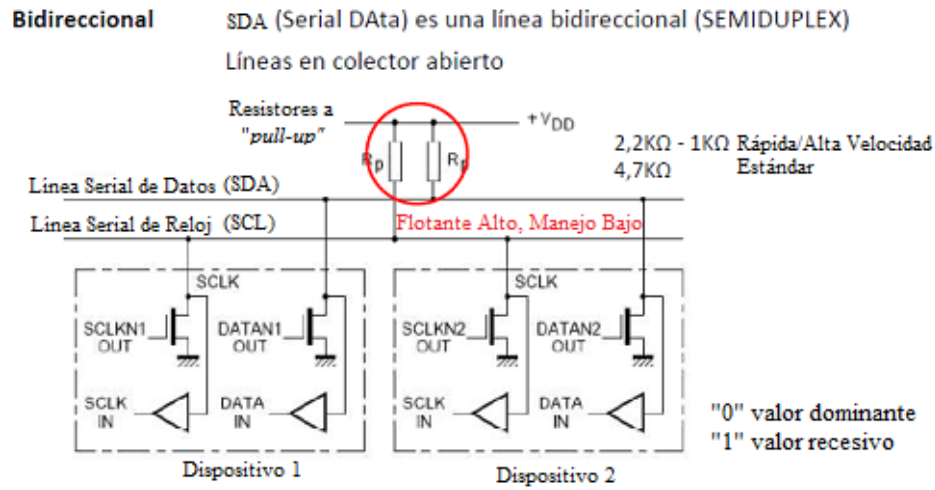


Figura 47. Especificaciones para el bus de la I²C.

Transferencias – Inicio / Fin (Maestro).- (Figura 48)

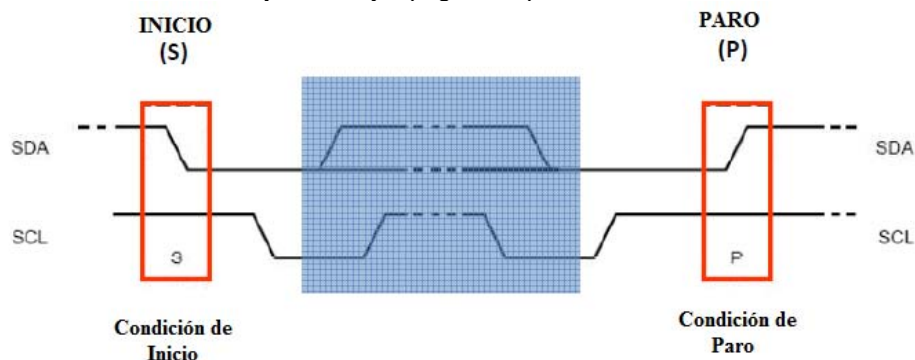


Figura 48. Diagrama de tiempos para las transferencias.

Transferencias – Reconocimiento "Acknowledge".-

Formato de los mensajes (direcciones de 7 bits).- (Figura 49)

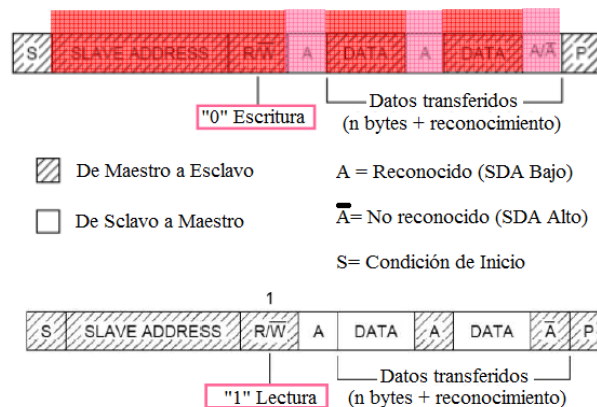


Figura 49. Conformación de la trama en los mensajes enviados.

Bus Arbitrado.- (Figura 50)

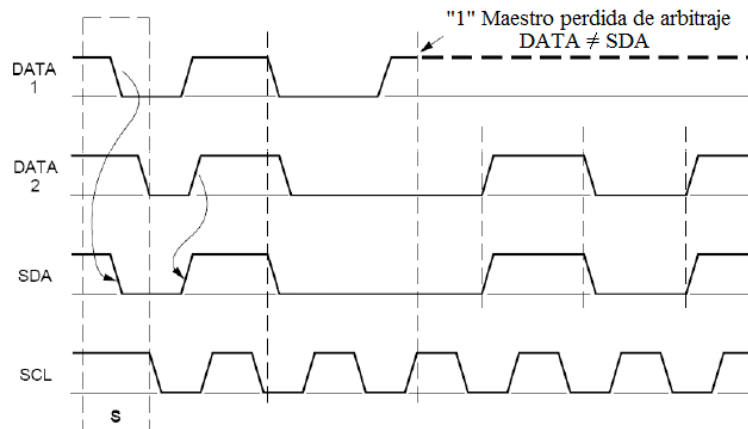


Figura 50. Cronograma para un bus arbitrado.

Redes locales.- (Tabla 7)

Tabla 7. Protocolos estándar de comunicación en serie.

Interfase	Formato	Num. Dispositivos	Separación max.	Velocidad de Tx (bits/seg)
RS-232	Serie Asinc.	2	15 – 30 m	20k
IrDA	Serie Asinc.	2	2 m	115k
SPI	Serie Sinc.	8	3 m	2.1M
I2C	Serie Sinc.	40	5.5 m	400k
USB	Serie Asinc.	127	5 m	12M

Origen del Standard RS232.-

Diseñado para comunicar un sistema *DTE* con un sistema *DCE*.

DTE (Data Terminal Equipment) usualmente, un Ordenador.

DCE (Data Communications Equipment) usualmente, un Modem.

Empleando dos líneas de datos, *TX* (Transmisión) y *RX* (Recepción), (Figura 51)

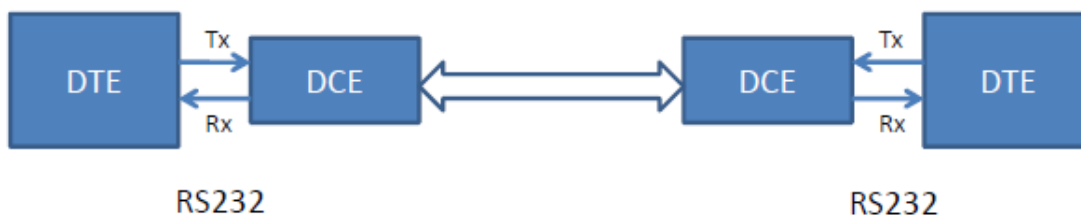


Figura 51. Tipo de conexiones para una comunicación serial.

Norma RS-232: Niveles lógicos.- (Figura 52)

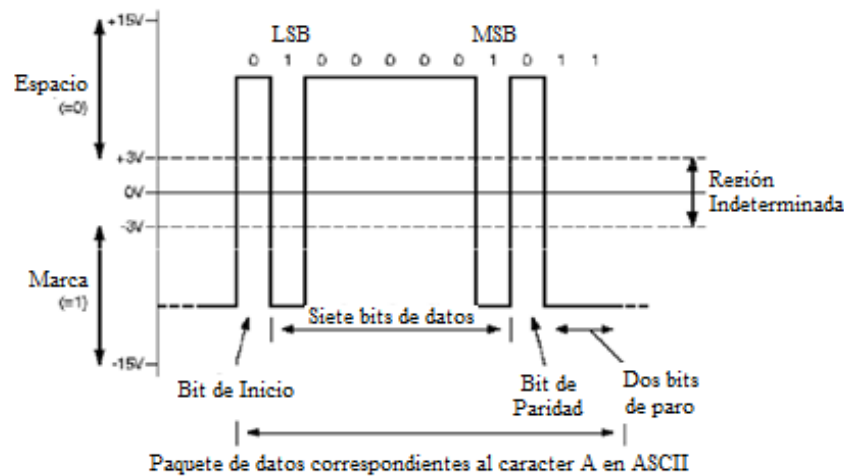


Figura 52. Circuitos Convertidores de nivel: MAX232 (TTL a RS232 y RS232 a TTL).

Norma RS-232: Conectores.- (Figura 53)

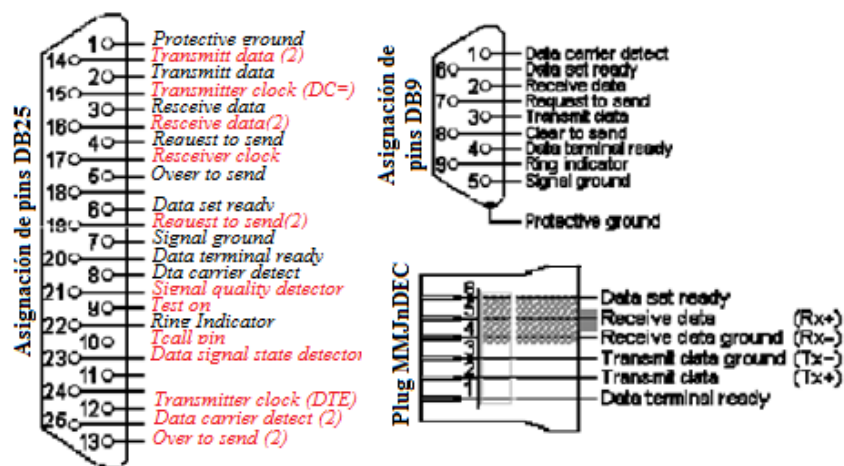


Figura 53. Asignación de terminales "pins" sobre los conectores tipo serie, tomado del "RS-232 Manual, 2001".

Norma RS-232: Formato de los mensajes.- (Figura 54)

Al no existir reloj → Necesidad de sincronizar RX y TX

- Velocidad de transmisión fija predeterminada
- Bits de comienzo y final

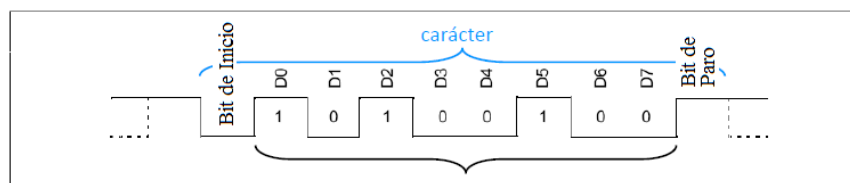


Figura 54. Cronograma para la comunicación serial.

Se configura en una comunicación asíncrona:

- Velocidad de transmisión
- Datos de 9 bits → Utilización de un bit de paridad
- Número de bits de parada: 1, 1.5, 2

Señales del Protocolo RS232.- (Tabla 8)

Tabla 8. Señales del protocolo RS232.

Nombre	Dirección DTE ↔ DCE	Función	Comentarios
TD	⇒	<i>Transmitted Data</i>	Par de Datos
RD	⇐	<i>Received Data</i>	
RTS	⇒	<i>Request to Send</i>	Par de Handshake
CTS	⇐	<i>Clear to Send</i>	
DTR	⇒	<i>Data Terminal Ready</i>	Par de Handshake
DSR	⇐	<i>Data Set Ready</i>	
DCD	⇐	<i>Data Carrier Detect</i>	Habilitan DTE
RI	⇐	<i>Ring Indicator</i>	

Nota: La columna de la Función se escribe en ingles por las siglas de la primera columna.

Norma RS-232: Conexión “null-modem”.- (Figura 55)

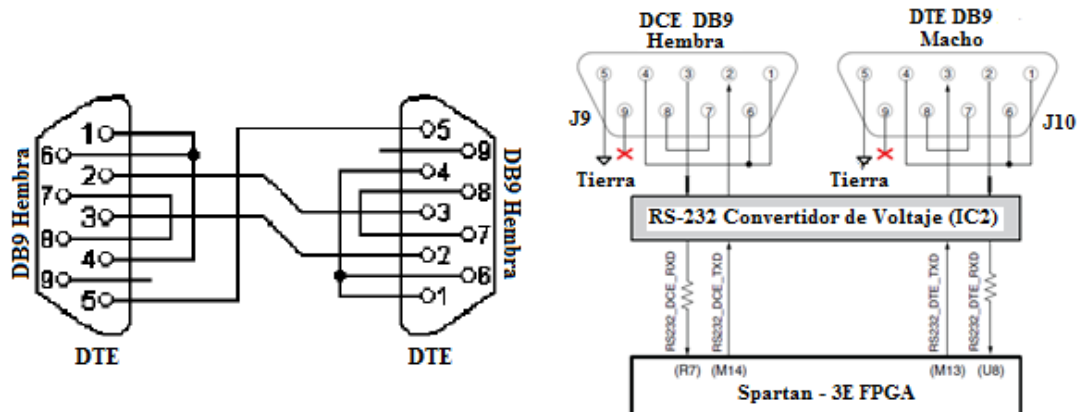


Figura 55. Conexiones para el cable RS-232.

SDI-12: serial data interface at 1200 baud.-

SDI-12 se utiliza en aplicaciones con los siguientes requisitos:

- Sistemas alimentados por baterías con mínimos consumos de corriente
- Existe un único equipo de lectura y varios sensores conectados en un solo cable.
- “Smart-sensors”: sensores basados en microprocesador que realizan algoritmos complejos de calibración u otro tipo de procesamiento. (www.sdi-12.org,2012/12), (Veáse Figura 56 y Tabla 9)

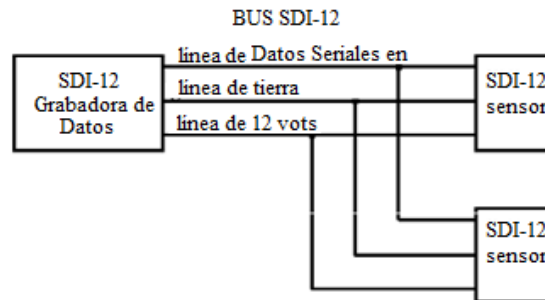


Figura 56. Interface con Bus SDI-12.

Tabla 9. Niveles Lógico y de Voltaje para los datos en serie.

Condición	Estado Binario	Rango de Voltaje
marcado	1	de -0.5 a 1.0 [V]
espaciado	0	de 3.5 a 5.5 [V]
transición	indefinido	de 1.0 a 3.5 [V]

La razón de transferencia para SDI-12 es de 1200 [baud], la tabla (9) muestra el formato de la trama para SDI-12. Tabla (10 y 11)

Tabla 10. Formato de la trama SDI-12.

1 bit de inicio
7 bits de datos, el bit menos significativo transmitido primero
1 bit de paridad, paridad
1 bit de paro

Tabla 11. Comandos y respuestas (caracteres ASCII)

Nombre	Comando	Respuesta
Cortado	Continuo espaciado por al menos 12 [ms]	Ninguna
Reconocimiento activado	a!	a<CR><LF>
Identificación enviada	a!	allccccccmmmmmmvvvxxx...xx<CR><LF>

3. Interfaz para Visualización

3.1 Display de 7 Segmentos y código BCD

Como se mencionó en el primer objetivo particular se desarrollara la implementación de una interfaz visual con el usuario partiendo de un *display* de 7 segmentos con cátodo común. En particular se usa el *display* DC-05 cuyas características se presentan a continuación, Figura (57a y 57b).

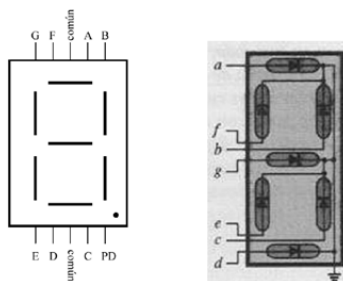


Figura 57. a) Terminales del display DC-05. b) Display de cátodo común.

Cada segmento (A, B, C, D, E, F, G y el punto decimal PD) son diodos LED que al recibir UNO en su ánodo, y CERO en su cátodo, debe encender. Si el display es DC-05, se trata de un display de cátodo común. Todos los cátodos de los segmentos están interconectados a las terminales indicadas como 'común' y estas dos terminales en el dispositivo deben conectarse a tierra. Los segmentos (A, B, C, D, E, F, G y DP) reciben UNOS para encender. El diagrama del integrado y sus dimensiones del display DC05 se muestran en la Figura (58).

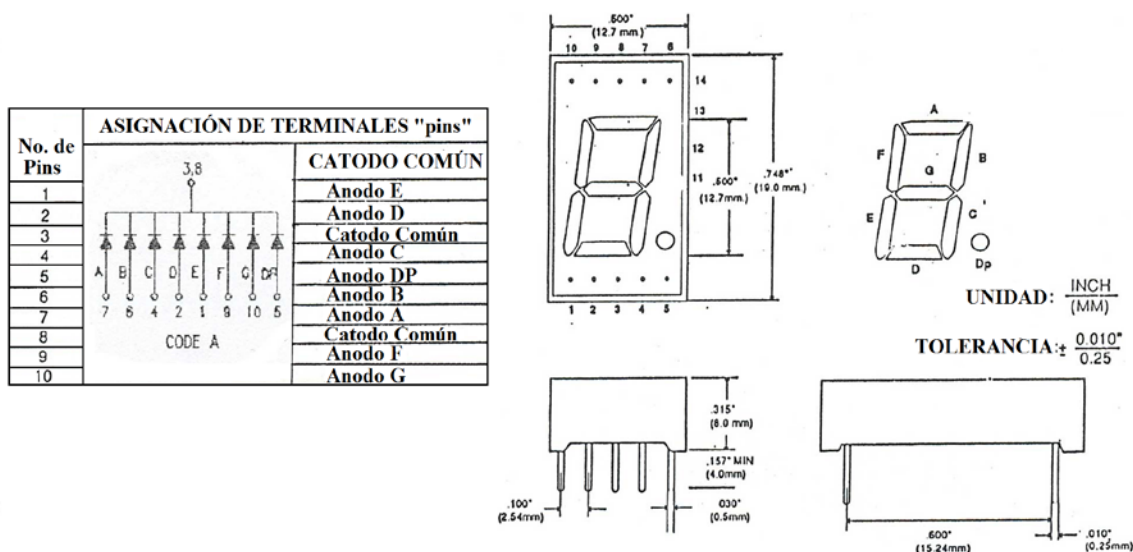


Figura 58. Diagrama del display DC05.

Para poder encender varios led y formar un dígito en el display es necesario generar un código, conocido como BCD (Binary Code Decimal), este código se presenta la Tabla (12).

Tabla 12. Código Binario Decimal.

BCD	w	x	Y	z	A	B	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

Luego usando la Tabla (8) y mapas de Karnaugh se obtuvieron las siguientes funciones que describen al circuito digital a implementar, Tabla (13).

Tabla 13. Funciones para cada segmento DC-05.

$f_a = xz + y + w + x'z'$
$f_b = w + x' + y'z' + yz$
$f_c = w + x + y' + z$
$f_d = w + yz' + x'z' + x'y' + xy'z$
$f_e = yz' + x'z' +$
$f_f = y'z' + xz' + w + xy'$
$f_g = w + xy' + yz' + x'y$

El código binario decimal puede tomarse directamente de un circuito integrado *DM74LS48*, que es un código *BCD* a un *display* de 7 segmentos decodificador construido por "National Semiconductor". Para el cual se muestra su tabla de verdad en la Tabla (14) y su diagrama lógico en la Figura (59).

Tabla 14. Tabla de verdad para el circuito integrado 74LS48.

Designación Numérica - Resultados del Despliegue															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Tabla de Verdad

Función o Decimal	Entadas						Salidas							
	LT	RBI	A ₃	A ₂	A ₁	A ₀	BI/RBO	a	b	c	d	e	f	g
0 (Note 1)	H	H	L	L	L	L	H	H	H	H	H	H	H	L
1 (Note 1)	H	X	L	L	L	H	H	L	H	H	L	L	L	L
2	H	X	L	L	H	L	H	H	H	L	H	H	L	H
3	H	X	L	L	H	H	H	H	H	H	H	L	L	H
4	H	X	L	H	L	L	H	L	H	H	L	L	H	H
5	H	X	L	H	L	H	H	H	L	H	H	L	H	H
6	H	X	L	H	H	L	H	L	L	H	H	H	H	H
7	H	X	L	H	H	H	H	H	H	H	L	L	L	L
8	H	X	H	L	L	L	H	H	H	H	H	H	H	H
9	H	X	H	L	L	H	H	H	H	H	L	L	H	H
10	H	X	H	L	H	L	H	L	L	L	H	H	L	H
11	H	X	H	L	H	H	H	L	L	H	H	L	L	H
12	H	X	H	H	L	L	H	L	H	L	L	L	H	H
13	H	X	H	H	L	H	H	H	L	L	H	L	H	H
14	H	X	H	H	H	L	H	L	L	L	H	H	H	H
15	H	X	H	H	H	H	H	L	L	L	L	L	L	L
BI (Note 2)	X	X	X	X	X	X	L	L	L	L	L	L	L	L
RBI (Note 3)	H	L	L	L	L	L	L	L	L	L	L	L	L	L
LT (Note 4)	L	X	X	X	X	X	H	H	H	H	H	H	H	H

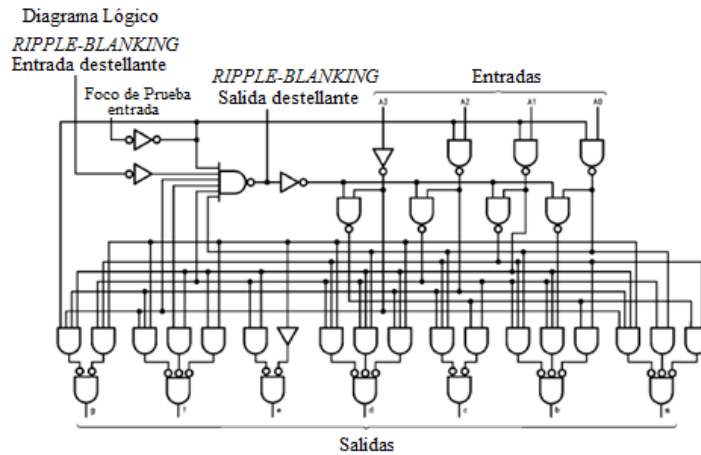
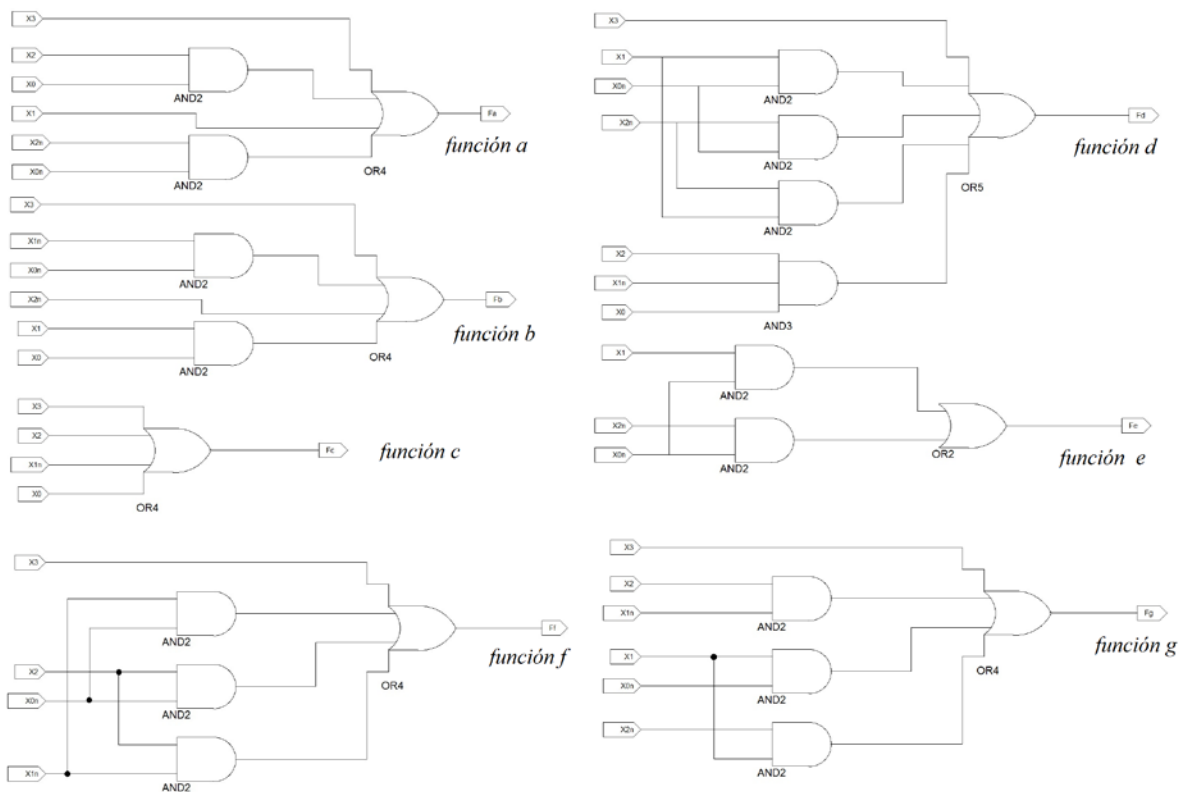


Figura 59. Diagrama lógico del circuito integrado 74LS48.

Cada una de las funciones de la Tabla (11) se implementa por separado en el sistema de desarrollo (*software*) de la empresa “Xilinx Design Suite versión 13.3” y se muestran en los diagramas esquemáticos de la Figura (60), que son parecidas a las del integrado de la Figura (59). Cabe señalar que esta implementación en la plataforma de desarrollo Xilinx se lleva a cabo en forma esquemática, en cual puede ser sustituido con su semejante código en VHDL.



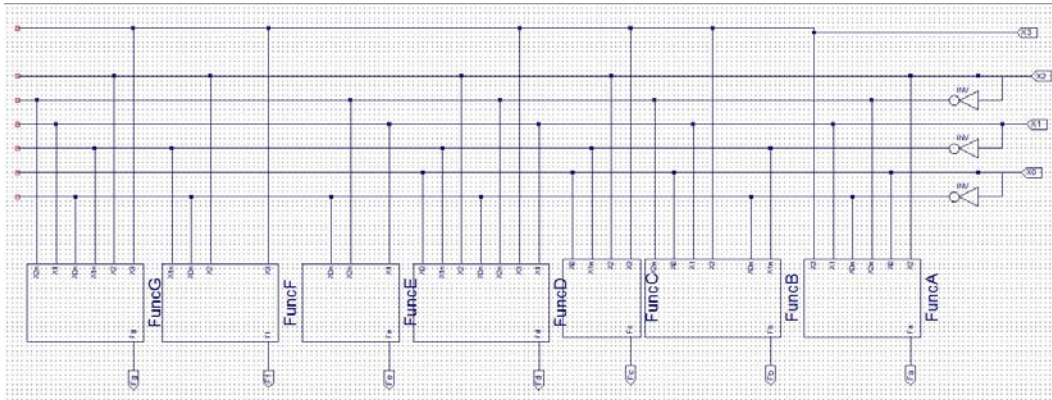


Figura 61. Configuración para el BCD que será usado para el display de 7 segmentos.

Para la prueba de este desarrollo se pudo emplear dos procedimientos, uno directamente desde el sistema de desarrollo en la parte de simulación, colocando a las entradas 4 señales en el tiempo con periodos de 2, 4, 8, 16 μs respectivamente. Con esto se muestra en el tiempo un contador del 0 al 9 que son los dígitos que se desean encender el *display* de 7 segmentos, las salidas serán las terminales f_a hasta f_g , la simulación de esta primera implementación se observa en la Figura (62).

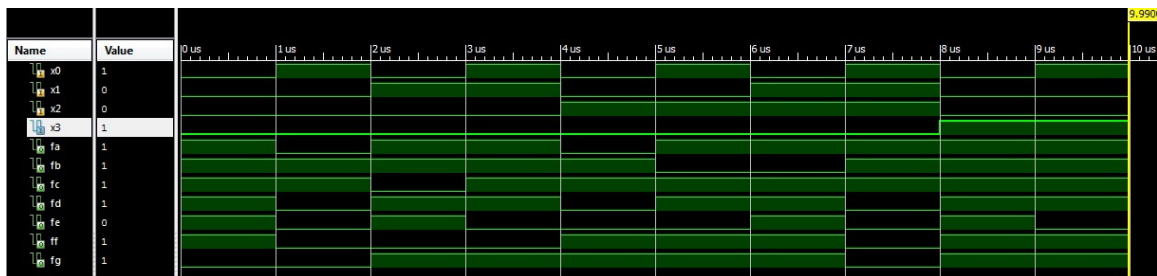


Figura 62. Simulación del encendido del display de 7 segmentos.

Analizando la simulación se observa que en el periodo entre 0 y 1 μs , la entrada para X_3 , X_2 , X_1 y X_0 es el número binario "0000". En este intervalo se encienden las salidas f_a , f_b , f_c , f_d , f_e , f_f y f_g permanece apagada, lo cual en el *display* de 7 segmentos representa el número "0". Esto se puede seguir haciendo en los siguientes intervalos de tiempo hasta llegar al binario "1001" que representa el número "9", donde todas las salidas están encendidas excepto f_e .

El segundo procedimiento de prueba es en la parte de implementación bajar el programa al *FPGA* xc3s500e con el *IMPAC* y usando la configuración de área en el archivo *.UCF* del *PlanAhead*, las terminales de los 4 interruptores deslizantes y la de los 8 led de la tarjeta Spartan 3E, la Figura (63) muestra dos ventanas donde se configuran las terminales. Estas definiciones se mostraron en la Figura (6) y (7) de la sección 2.1 de este trabajo, las cuales también se pueden consultar directamente en el manual de la tarjeta.

Name	Dir	Neg Diff Pair	Site	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type
Fa Output			F12	0	LVC MOS25	2.5		12 SLOW		
Fb Output			E12	0	LVC MOS25	2.5		12 SLOW		
Fc Output			E11	0	LVC MOS25	2.5		12 SLOW		
Fd Output			F11	0	LVC MOS25	2.5		12 SLOW		
Fe Output			C11	0	LVC MOS25	2.5		12 SLOW		
Ff Output			D11	0	LVC MOS25	2.5		12 SLOW		
Fg Output			E9	0	LVC MOS25	2.5		12 SLOW		
X0 Input			L13	1	LVC MOS25	2.5		12 SLOW		
X1 Input			L14	1	LVC MOS25	2.5		12 SLOW		
X2 Input			H18	1	LVC MOS25	2.5		12 SLOW		
X3 Input			N17	1	LVC MOS25	2.5		12 SLOW		

Figura 63. Configuración de las terminales en el PlanAhead para las entradas (switch) y salidas (led).

Cuando se baja al hardware el programa del *BCD* al *display* de 7 segmentos en este caso a los led de la tarjeta Spartan 3E, se puede probar si por ejemplo se activa en alto el *SW1* que representa al número “0010” en binario y “2” en decimal se encenderán los led *LD0*, *LD1*, *LD3*, *LD4* y *LD6*, correspondientes a los pins en el archivo de restricción de área .UCF: *F12*, *E12*, *F11*, *C11*, y *E9* de la tarjeta, esto se muestra en la Figura (64).

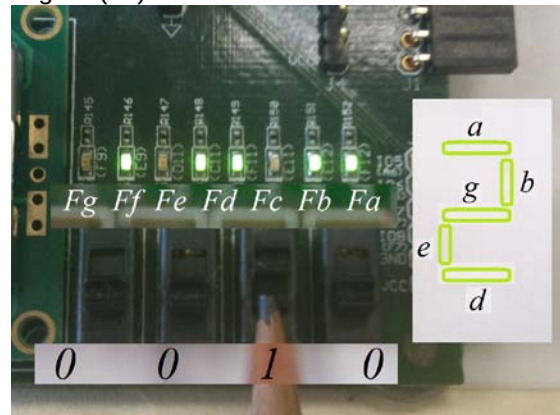


Figura 64. Prueba del código BCD en Hardware.

El resumen de recursos consumidos en el *FPGA* se muestra en la Figura (65). Datos para las restricciones de área del *BCD* en el Anexo I al final de este reporte.

bcd2seg7 Project Status (08/31/2011 - 14:53:00)				
Project File:	bcd2seg7.xise	Parser Errors:	No Errors	
Module Name:	bcd2seg7	Implementation State:	Programming File Generated	
Target Device:	xc3s500e-4fg320	• Errors:	No Errors	
Product Version:	ISE 13.3	• Warnings:	No Warnings	
Design Goal:	Balanced	• Routing Results:	All Signals Completely Routed	
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:		
Environment:	System Settings	• Final Timing Score:	0 (Timing Report)	
Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Number of 4 input LUTs	7	9,312	1%	
Number of occupied Slices	4	4,656	1%	
Number of Slices containing only related logic	4	4	100%	
Number of Slices containing unrelated logic	0	4	0%	
Total Number of 4 input LUTs	7	9,312	1%	
Number of bonded IOBs	11	232	4%	
Average Fanout of Non-Clock Nets	3.09			

Figura 65. Resumen de recursos consumidos en el *FPGA* de la tarjeta Spartan 3E, reporte obtenido después de la implantación BCD mediante Xilinx.

3.2 Desarrollo de un Reloj Digital

Para poder ver en operación la interfaz visual se considera ahora desarrollar la aplicación del reloj digital en 4 dispositivos *DC05* (*display* de 7 segmentos) para formar un *display* de 4 dígitos y mostrar las horas-minutos-segundos del reloj.

Objetivo.- Realizar un reloj digital que cuente las 24 horas de un día, los dígitos se mostrarán en 4 *displays* de 7 segmentos.

Planteamiento del Problema

Se pide hacer el algoritmo en el lenguaje *VHDL* para implementar un reloj digital con una cuenta de 0 a 23:59, el periodo en el cambio de cuenta es de un segundo, el dígito correspondiente al valor del contador se visualizará en un *display* de 7 segmentos. El modelo tendrá dos señales para inicializar la cuenta y ponerla a cero (*reset*) y la otra dejará el último valor del contador y lo mantendrá hasta el cambio de esta (*habilitación*).

Manejo del algoritmo en VHDL para implementar un reloj de 0 a 23:59

Se consideró la frecuencia del reloj = 50 [MHz], que equivale a un segundo, por lo que se implementó un contador con valor 50000000 para proporcionar la señal de reloj de un segundo. Se estableció un proceso que lee la señal de reloj de un segundo e inicia la cuenta de 0 a 9 para el contador de segundos, cada vez que reinicia su cuenta se establece una señal de reloj para generar un cambio de cuenta de minutos, que a su vez en cada reinicio de cuenta de minutos se establece un cambio de cuenta en horas hasta las 23:59. Se estableció con la estructura “*with select*” la conversión de binario a decimal para mostrarlo en el *display* de 7 segmentos. Se estableció un multiplexor para poder visualizar los dígitos correspondientes a segundos, minutos, horas, debido a que los *displays* están en paralelo.

Programa que define al contador

Para el programa contador se tienen 3 entradas, una de reloj, otra para el reinicio y otra para el habilitador o cambio entre horas y minutos-segundos. Se tienen 6 salidas de 4-bit que son para cada uno de los dígitos del reloj, es decir una para cada elemento visualizado, esto es una para las unidades de segundo, otra decenas de segundo, unidades de minuto, decenas de minuto, unidades de hora y una última para las decenas de hora. Visto de esta forma se tendrán 6 contadores condicionados a reinicio y a cambio de valor de acuerdo a la cuenta del reloj, para entenderlo mejor el primer contador se vuelve a 0 cuando llega a 9, el segundo se vuelve a 0 cuando llega a 5 (para los segundos), el tercer contador se vuelve a 0 cuando llega a 9, el cuarto se vuelve a 0 cuando llega a 5 (para los minutos), el quinto contador se vuelve a 0 cuando llega a 3 y el sexto se vuelve a cero cuando llega a 2 (para las horas). También se generaron dentro de la arquitectura 7 señales de reloj, 5 de las cuales son de reloj, otra para el habilitador y la última para un contador auxiliar de 4-bit.

Programa que define al Reloj Digital

A continuación se describirá el generador de pulsos de reloj con periodo de un segundo, y otro con un periodo de 6 milisegundos para el cambio de los cátodos del *display*. Para el programa del reloj se tienen una entrada de reloj con periodo de 20 [ns] y dos salidas una de periodo de 1 [s] y la otra que indica la salida del valor del contador en su respectivo *display*. En la parte de la arquitectura se generaron dos constantes enteras una de 50000000 y otra de 25000, además de una señal que produce los cambios en los cátodos de los *display* para colocar cada dígito en cada uno de éstos.

Programa que define al Multiplexor

En el caso del multiplexor empleado para el desarrollo del reloj digital se usaron 8 entradas y 2 salidas. 6 de estas entradas son de 4-bit provenientes del contador para cada dígito del reloj, hay una entrada de control de 2-bit que proviene del reloj y una última entrada que es la del habilitador para el cambio horas, minutos-segundos. Las dos salidas son de 4-bit, una de ellas directamente a los cátodos del *DC05* y la otra al decodificador *BCD* de 7 segmentos.

Programa que define al BCD de 7 Segmentos

Para el *BCD* de 7 segmentos se tienen una entrada y una salida, la entrada proveniente del multiplexor es de 4-bit y la salida de 7-bit van alambradas a los 7 segmentos en paralelo de los *DC05*.

El listado del programa en VHDL para cada uno de los módulos y las especificaciones de las terminales usadas de la tarjeta Spartan 3E (archivo.UCF) del reloj digital se puede consultar en el Anexo II de este reporte.

Creados los programas en VHDL de los módulos se vuelve hacer uso de las herramientas de Xilinx para crear los símbolos esquemáticos y unir entradas con salidas de los bloques generados. La Figura (66) muestra el esquemático que se armó con cada módulo desarrollado para el reloj digital.

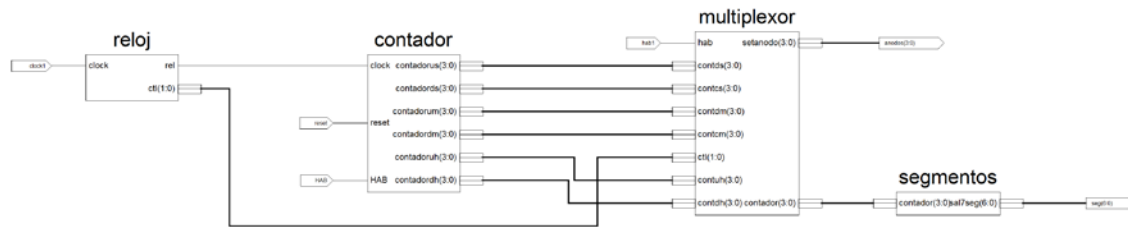


Figura 66. Diagrama esquemático desarrollado en el paquete Xilinx Design Suite 13.3 para el reloj digital.

Luego como la tarjeta de desarrollo Spartan 3E no tiene físicamente los *display* de 7 segmentos, se alambraron en un “protoboard” para realizar las pruebas y obtener final mente los resultados, Figura (67).

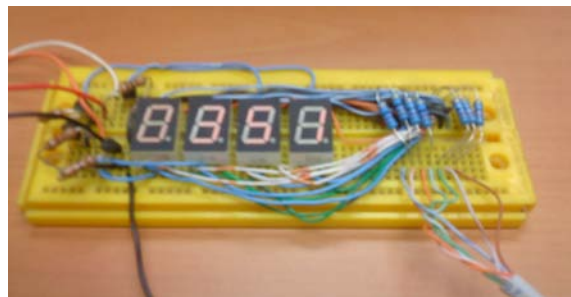


Figura 67. Alambrado de los display de 7 segmentos en protoboard.

Ahora se mostrara en un video el funcionamiento del reloj digital con cuenta de 0 a 59 segundos, de 0 a 59 minutos y de 0 a 23 horas, Figura (68), *Nota: hacer doble click sobre la imagen.*

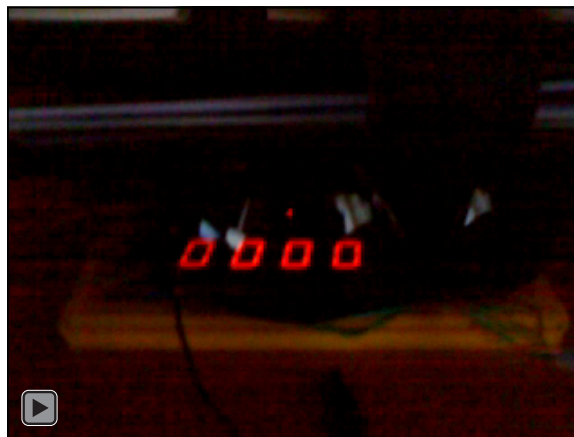


Figura 68. Video del diseño del reloj digital operando.

El ejemplo es bastante claro en cuanto a la descripción de un reloj implementado en la tarjeta de programación Spartan 3E.

La Figura (69) muestra los recursos consumidos dentro del *FPGA* el desarrollo del reloj digital y en el Anexo II de este reporte se muestra la implementación del programa en lenguaje *VHDL*.

watcho Project Status (05/21/2013 - 13:39:47)			
Project File:	RELOJ.xise	Parser Errors:	No Errors
Module Name:	watcho	Implementation State:	Synthesized
Target Device:	xc3s500e-sfg320	• Errors:	No Errors
Product Version:	ISE 13.3	• Warnings:	14 Warnings (0 new)
Design Goal:	Balanced	• Routing Results:	
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	
Environment:	System Settings	• Final Timing Score:	

Device Utilization Summary (estimated values)				[-]
Logic Utilization	Used	Available	Utilization	
Number of Slices	67	4656	1%	
Number of Slice Flip Flops	98	9312	1%	
Number of 4 input LUTs	120	9312	1%	
Number of bonded IOBs	15	232	6%	
Number of GCLKs	2	24	8%	

Figura 69. Recursos consumidos en el FPGA por el reloj digital, resumen obtenido después de la implantación del reloj digital en Xilinx.

4. Interfaz para el cristal liquido envío de mensajes

El LCD (la pantalla de cristal liquido) es un dispositivo de gran ayuda en el campo de la electrónica, es una pantalla que se forma de un número de pixeles monocromáticos (el más utilizado es de resolución 2 x 16); la interfaz entre la tarjeta SPARTAN 3E y la LCD se da por 4 pines de datos < SF_D> y 4 pines de control <lcd_rs, lcd_rw, lcd_e, sf_ce0>, Figura (70).

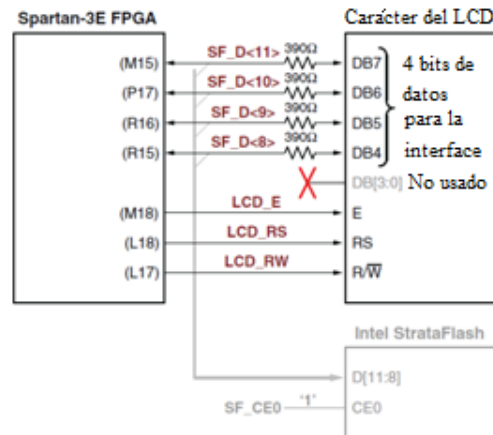


Figura 70. Configuración para lectura-escritura y reinicio, Spartan-3E User Guide, (2008).

Una descripción detallada de la operación de un LCD es lejana más allá del alcance de este documento; sin embargo, es con todo de mérito entender la operación general de un dispositivo LCD. Esencialmente, qué lo hace un LCD negro y blanco tradicional es accionar el interruptor para qué se llamen los pixeles en el código de exploración y permitir o bloquear la luz. Estos pixeles se consideran ser los puntos que componen la muestra en la pantalla; por supuesto, en este caso particular los pixeles están claramente visibles y son cuadrados perfectos. En todo caso, para generar algo que tenga la forma de la letra "I", el LCD accionaría el interruptor de los pixeles en una línea vertical recta.

Nuevamente hay dos formas fundamentales para poder visualizar el funcionamiento del LCD que consisten en:

1. La puesta en práctica del hardware para controlar la pantalla del *LCD*.
2. Una simulación del hardware para verificar funcionalidad.

Recordando cómo se hizo anteriormente es más fácil primero programar la tarjeta y verificar el diseño del hardware en la simulación. Desafortunadamente, esto es cierto solo en proyectos muy pequeños, pero los proyectos llegan a ser tan grandes que se hace más problemático encontrar la solución sin las fuentes requeridas. En varias ocasiones la programación de la tarjeta a manera de ensayo y error puede ocasionar solamente pérdida de tiempo.

Inicialización, Configuración y Escritura del LCD

Hay tres pasos principales al usar la pantalla, el primero es la inicialización de la interfaz misma, el segundo es que se establecen los comandos para las opciones de pantalla y el tercero que son la escritura de los datos del carácter, como se mostró en la sección (2.1) pág. 18 de este reporte.

Inicialización

La secuencia de inicialización primero establece cual función de 4-bits *FPGA* se desea usar, estos 9 puntos aparecen en el manual de la tarjeta Spartan 3E (pg. 53).

Configuración

El Segundo paso involucra la configuración actual a la memoria del *LCD*. Para configurar el *LCD* hay cuatro comandos que se refieren a: "*Function Set, Entry Mode, Display On/Off y Clear Display*". Los parámetros específicos para cada comando están en el manual de la tarjeta Spartan 3E (págs. 53-54).

Escritura

El tercer y último paso involucra al proceso actual de escritura de datos a la *DD-RAM*, esto se hace de acuerdo con:

1. Especifica la dirección de inicio con el comando *Set DD-RAM Address*, ver Figura (71) para las direcciones de memoria.
2. Despliega un carácter con el comando *Write Data*, Figura (24) de este reporte de la sección 2.1.

Direccionamiento de los Caracteres en el Despliegue																Direcciones no desplegadas	
1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10 ... 27
2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50 ... 67
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17 ... 40

Figura 71. Direcciones de memoria DD RAM en Hexadecimal.

Objetivo

Hacer uso del *LCD* de 16 caracteres por 2 líneas incluido en la tarjeta Spartan 3E Starter kit para desplegar el letrero "*ALEJANDRO PADRON GODINEZ UNAMCCAD*", y conocer más acerca del diseño lógico digital en el proceso.

Proceso

1. Implementar en hardware el control del *LCD*.
2. Verificar en hardware y en software (Simulación).
3. Programar la Spartan 3E Starter Kit.

Implementación

Este proyecto requiere 3 máquinas de estado. Uno para la alimentación sobre secuencia de inicialización, una para transmitir los comandos y los datos al *LCD* y por último una para comenzar la alimentación de la secuencia de inicialización, entonces se configura y se escribe en el *LCD*. Observe que uno tendrá que cerciorarse de que las máquinas de estado estén sincronizadas correctamente, es decir, que la máquina principal de estado sigue estando en el estado de "*INIT*" hasta que la máquina de estado de inicialización está en el estado "*DONE*", y que los estados subsecuentes no cambien de comando hasta que los datos sean transferidos completamente por la máquina de estado de transmisión.

La Figura (72) muestra la principal máquina de estados que controla la secuencia de inicialización y la máquina de estado de transmisión. Note que cada uno de los comandos implícitamente requiere la máquina de estado de transmisión.

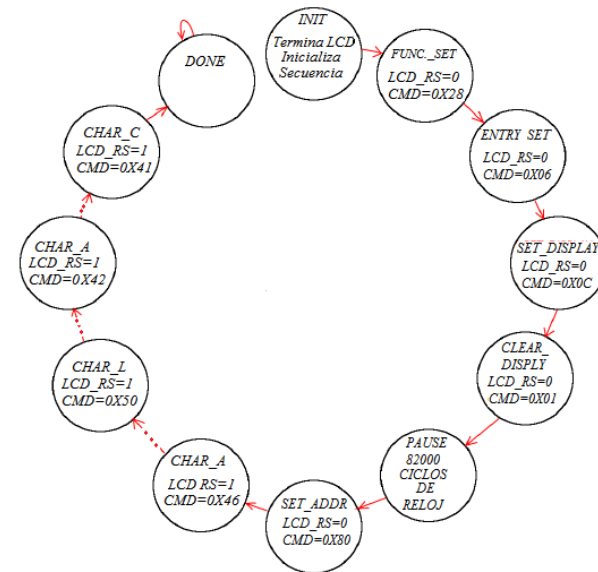


Figura 72. Principal máquina de estados.

La Figura (73) es la máquina de estado de la secuencia de inicialización. Se activa solamente cuando la máquina principal de estado afirma el "INIT". Asegúrese que haya una manera de notificar a la máquina principal de estado que la inicialización se termino y que la interfaz de 4-bits se ha establecido.

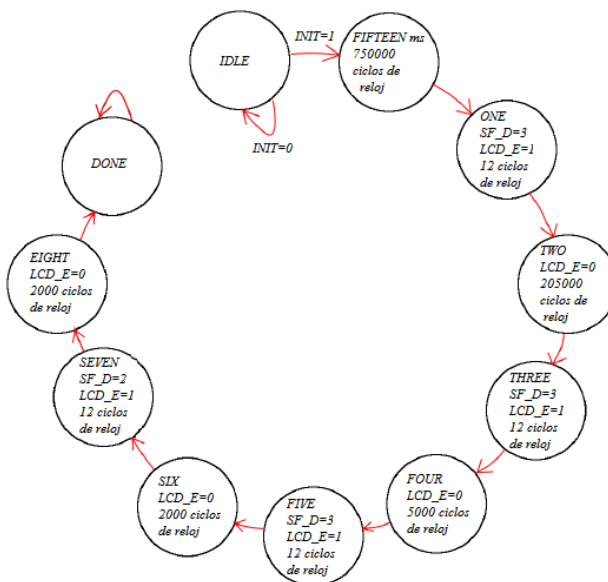


Figura 73. Secuencia de Inicialización

Esta es la última máquina de estados y potencialmente la más compleja. La principal máquina de estado fijará los valores de LCD_RS y los valores de los "nibbles" altos y bajos de un comando al LCD. Esta máquina de estado verificará simplemente los tiempos estrictos de la sincronización dados por un

comando del *LCD*, si no sería totalmente innecesario. Una vez más un comando será transferido solamente una vez que el *tx_init* sea confirmado por la máquina principal de estado. Figura (74).

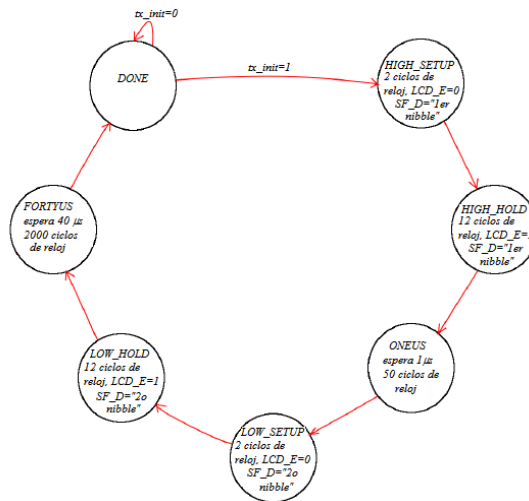


Figura 74. Restricciones de sincronización de la máquina de estado.

Programa que define la aplicación del LCD

El programa tiene 2 entradas de un bit, una para el reloj y otra para el reinicio, tiene 1 salida *SF_D* de 4-bits para el código los nibbles del carácter, otras 4 salidas de un bit correspondientes a los comandos del *LCD*, estas son *LCD_E*, *LCD_RS*, *LCD_RW* y *SF_CEO*. También se usa otra salida pero ahora de 8-bits para encender los LED de la tarjeta como prueba de que se envían los caracteres a la pantalla de cristal liquido. En la parte de arquitectura hay que declarar todas las señales de inicialización, configuración y escritura del *LCD*, además de declarar como “*type*” los caracteres que se van a imprimir sobre el *LCD*. Dos de estos comandos importantes *type* son los de posicionamiento del cursor en la primera línea (*set_addr*) y al inicio de la segunda línea (*set_addr_s1*) y obviamente la terminación que se denota por *DONE*. En este programa se encuentran las tres máquinas de estado descritas en la implementación y mostradas en las Figuras 72, 73 y 74.

El resultado de la implementación de la interfaz que hace uso del *LCD* de la tarjeta Spartan 3E se muestra en la Figura (75 y 76), en la primera de ellas se muestra el objetivo que se deseaba obtener.



Figura 75. Letrero implementado en el LCD de la tarjeta.

La Figura (76) muestra una simulación de una puesta en práctica del funcionamiento del controlador de hardware para el *LCD*. Esta simulación demuestra la forma en que las máquinas de estado trabajan juntas. Mientras que la secuencia de inicialización acaba, los comandos de los estados de la máquina principal de estado comienzan.

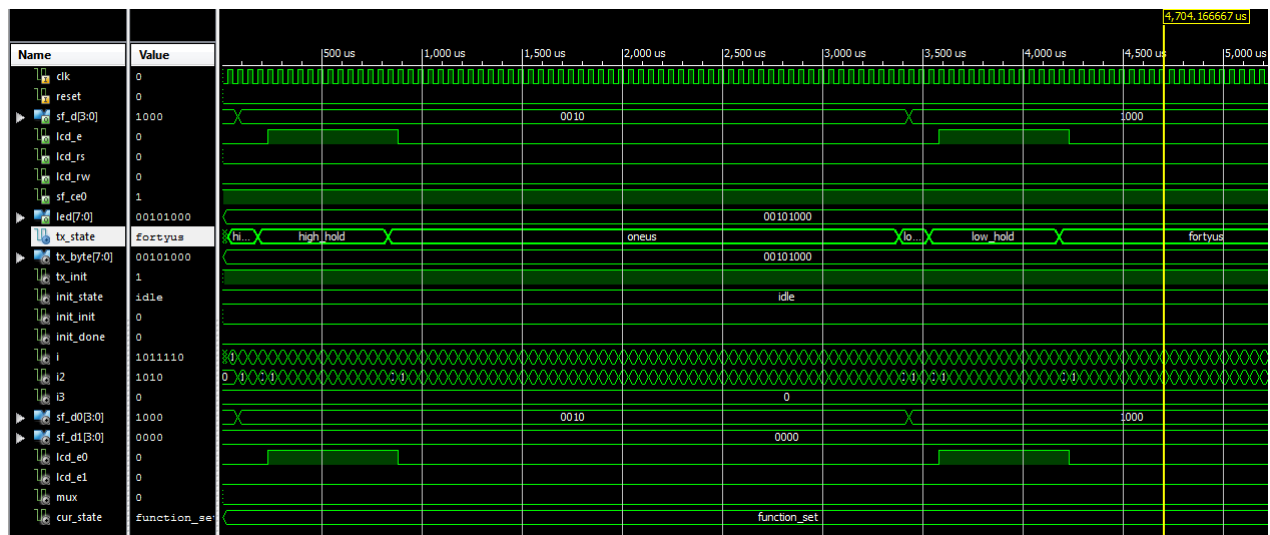


Figura 76. Simulación para la implementación de la interfaz en el LCD.

El resumen de los recursos consumidos en el *FPGA* se muestra en la Figura (77) para el envío de mensajes en el *LCD*, en esta muestra que sólo se ocupa un 3% de la capacidad total.

El listado del programa en *VHDL* para la interfaz del *LCD* y las restricciones de área de las terminales usadas en la tarjeta *Spartan 3E* (archivo.*UCF*) se puede consultar en el Anexo III de este reporte.

lcd Project Status (01/11/2012 - 20:24:16)				
Project File:	tutorial_lcd.xise	Parser Errors:	No Errors	
Module Name:	lcd	Implementation State:	Programming File Generated	
Target Device:	xc3s500e-4fg320	Errors:	No Errors	
Product Version:	ISE 13.3	Warnings:	5 Warnings (0 new)	
Design Goal:	Balanced	Routing Results:	All Signals Completely Routed	
Design Strategy:	Xilinx Default (unlocked)	Timing Constraints:	All Constraints Met	
Environment:	System Settings	Final Timing Score:	0 (Timing Report)	
Device Utilization Summary				[+]
Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	116	9,312	1%	
Number of 4 input LUTs	185	9,312	1%	
Number of occupied Slices	143	4,656	3%	
Number of Slices containing only related logic	143	143	100%	
Number of Slices containing unrelated logic	0	143	0%	
Total Number of 4 input LUTs	231	9,312	2%	
Number used as logic	185			
Number used as a route-thru	46			
Number of bonded IOBs	18	232	7%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	3.32			

Figura 77. Recursos que ocupa la aplicación dentro del *FPGA* de la tarjeta, resumen obtenido por *Xilinx*.

5. Interfaz PS/2 y VGA

5.1 Interfaz “Personal System/2” (PS/2)

El módulo de interfaz *PS/2* pone un interfaz en ejecución bidireccional genérico *PS/2*. Puede ser utilizado con cualquier dispositivo compatible *PS/2* y ofrece a sus clientes una manera conveniente de intercambiar datos. El interfaz transparente envuelve el octeto que se enviará en una trama por el *PS/2*,

genera el bit de paridad, y envía la trama un bit a la vez al dispositivo. Similarmente, al recibir datos del dispositivo *PS/2*, el interfaz recibe la trama, comprueba para saber si hay paridad, extrae los datos útiles, y los transmite al cliente. Si ocurre un error al enviar o recibiendo un octeto, el cliente es informado fijando la línea de salida de error arriba. De esta manera el cliente puede volver a enviar los datos o publicar un comando de volver a enviar al dispositivo.

La interfaz física *PS/2* usa cuatro líneas. En el conector *PS/2* los 6-pins están asignados como sigue:

1. Data
2. No Implementada
3. Tierra
4. V_{CC} (+5V)
5. Reloj
6. No Implementada

La línea del reloj lleva al reloj del dispositivo con una gama de frecuencia de 10 a 16.7 [KHz] (30 a 50 [μ s]). Cuando la línea esta a marcha lenta se pone en alta impedancia. El reloj se genera solamente cuando el dispositivo es el que envía o recibe los datos. Los datos y las líneas del reloj están ambos a colector abierto colocando arriba a los resistores a V_{CC} . Una interfaz "opencollector" (a colector abierto) tiene dos estados posibles: ("Z") impedancia baja ("0") o alta.

Cuando un dispositivo desea enviar un octeto, coloca la señal del reloj abajo y el anfitrión (es decir, la interfaz *PS/2*) reconoce que el dispositivo está enviando datos. Cuando el anfitrión desea enviar datos, hace una petición de enviar. Esto se hace llevando a cabo la señal por lo menos 100 [μ s] abajo del reloj, entonces la señal de datos se cae.

Después, se lanza la señal del reloj (colocada en alta impedancia). El dispositivo comienza a generar impulsos de reloj. Al recibir datos, los bits se leen en la señal de datos (*PS/2_data*) en el flanco que cae del reloj (*PS/2_clk*). Al enviar datos, el dispositivo lee los bits de la señal de datos en el flanco de levantamiento del reloj. Una trama de un octeto abarca un valor de 11 bits para enviar. De la trama se envía un bit a la vez el bit menos significativo, o bit de inicio, y se recibe de la misma manera.

- | | |
|---------|--|
| • STOP | bit de parada, siempre '1' |
| • PAR | bit de paridad, paridad non; el número de bits de '1' en los datos de bit juntos con la paridad non. |
| • D0-7 | bits de datos |
| • START | bit de inicio, siempre '0' |

Al recibir, el registro de la trama se cambia de posición hacia la derecha, poniendo el primer bit en la señal de datos en la posición más significativa. De esta manera el primer bit enviado alcanzará al bit menos significativo de la trama una vez que se hayan recibido todos los bits.

Al enviar los datos, el bit menos significativo de la trama se pone en la señal de datos y la trama se cambia de posición o se desplaza a la derecha. Al lanzar la señal del reloj durante la petición de enviar, el dispositivo lee la línea de datos e interpreta los datos como el primer bit de la trama. Consecuentemente, al enviar, solamente diez cambios de la trama serán realizadas. Para el bit de paridad, hay una tabla de las operaciones de búsqueda que se define en una *ROM* 256x1. Mientras que la interfaz es quien envía o recibe los datos, la señal de salida ocupada pasa a alto. Cuando el interfaz está en estado ocioso, la señal de salida es baja. Después de enviar todos los bits de la trama, el dispositivo debe reconocer los datos enviados. El anfitrión lanza la señal de datos (la señal del reloj se activa) después de que se envíe el bit pasado. El dispositivo trae la trama de datos y la trama cuando el reloj pasa por el flanco de bajada, en orden, para reconocer los datos. Si el dispositivo no reconoce los datos, entonces se fija la salida errada.

Para generar es necesario el retraso, aquí se usan los contadores. Por ejemplo, para generar el retraso de 100 [μs], un contador de 14 bits que cuenta hasta 10.000 se utiliza. Porque el interfaz se diseñó para funcionar en 100 [MHz], por lo tanto $10,000 \times 10 \text{ [ns]} = 100 \text{ [μs]}$. Figura (78)

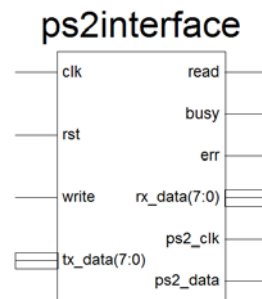


Figura 78. Esquemático creado para el bloque de la interfaz PS/2 en Xilinx.

Definición de Puertos para la interfaz PS/2 Modular: Figura (79)

PS/2_clk:	pin entrada/salida, línea de reloj de la interface PS/2
PS/2_data:	pin entrada/salida, línea de reloj de la interface PS/2
Clk:	pin de entrada, señal de reloj del sistema
Rst:	pin de entrada, señal de reinicio del sistema
tx_data:	pin de entrada, 8 bits, los datos del cliente para ser enviados al dispositivo
write:	pin de entrada, del cliente, deben estar activos para un período de reloj cuando entonces el cliente desea enviar los datos al dispositivo y los datos que se enviarán son válidos en tx_data.
rx_data:	pin de salida, 8 bits, datos recibidos por el cliente desde el dispositivo.
Read:	pin de salida, al cliente para un período de reloj en que los nuevos datos están disponibles.
Busy:	pin de salida, cliente activo mientras envía o recibe datos.
Err:	pin de salida, cliente activo para un periodo de reloj cuando un error ha ocurrido durante el envío o recepción.

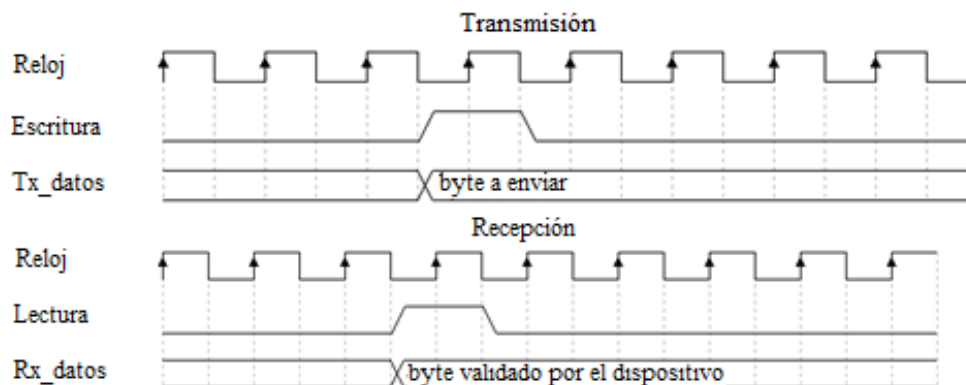


Figura 79. Diagrama de tiempo del envío y recepción de datos al dispositivo.

La interfaz PS/2 recibe octetos del dispositivo PS/2 (ratón o teclado) y los envía más lejos al regulador del ratón o del teclado. Los datos se envían en el puerto de salida del Rx_data. La señal leída está activa para un período de reloj en que un nuevo octeto está disponible en Rx_data. Cuando la entrada del error pasa a alto para un reloj (que indica un error mientras que el interfaz PS/2 recibió un

octeto), el regulador incorpora el estado del reajuste. La interfaz *PS/2* también consigue octetos del regulador del ratón o del teclado y los envía más lejos al dispositivo. Los datos se reciben en el puerto de salida del *Tx_data*. La señal de salida de escribir debe estar activada para un período de reloj en que el *Tx_data* contiene el comando o los datos para ser enviado al ratón o al teclado.

5.2 Interfaz “Video Graphics Array” (VGA)

El componente de la referencia de VGA genera las señales para exhibir una imagen en un monitor estándar de VGA. Puede manejar las resoluciones 640x480 y 800x600 y es compatible con las exhibiciones del *CRT* y del *LCD*. Las señales y la sincronización de VGA descritas aquí refieren al haz electrónico que explora la pantalla del monitor del *CRT* del VGA. La imagen real se compone de una secuencia tres de las líneas horizontales que se restauran constantemente.

El proceso de refresco de la pantalla comienza en la esquina y los pixeles izquierdos superiores un pixel a la vez de izquierda a derecha. En el final de la primera fila, los incrementos de la fila y la dirección de columna se reajusta a la primera columna. Una vez que se haya pintado la pantalla entera, el proceso de la refresco comienza otra vez. La señal video rediseña la pantalla 60 veces por segundo. Este restaura la tarifa permite el movimiento de la imagen y reduce el parpadeo de la pantalla. El restaure de las tarifas más arriba que 60 [Hz] se utilizan en monitores de PC.

Descripción Funcional

El componente de VGA se estructura en tres submódulos: el *vga_controller_640_60* maneja la resolución 640x480, el *vga_controller_800_60* maneja la resolución 800x600, y el *vga_selector* selecciona las salidas correctas para cada resolución. Los dos reguladores trabajan semejantemente. Las señales de pulso de la sincronía vertical y horizontal (*CONTRA* y los *HS*) indican el principio de un nuevo bastidor o fila.

La señal de la sincronía vertical indica al monitor comenzar a exhibir una nueva trama, y el monitor empieza en la esquina izquierda superior con el pixel (0,0). La señal de sincronía horizontal dice al monitor comenzar a exhibir un de giro nuevo desde el borde izquierdo de la pantalla. Mientras que el haz está volviendo a la columna izquierda para comenzar otra exploración horizontal, los datos del pixel no se exhiben y las señales del *RGB* son todas fijas en cero (el color negro). La Figura (80) muestra la estructura.

Definición de Puertos

Reset:	señal global de reinicio
clk_25MHz:	entrada, 25 [MHz] señal de reloj generada por un <i>DCM</i>
clk_40MHz:	entrada, 40 [MHz] señal de reloj generada por un <i>DCM</i>
resolution:	entrada, resolution_switcher selecciona activa la resolución
hs:	salida, al monitor, selecciona la señal de sincronía horizontal
vs:	salida, al monitor, selecciona la señal de sincronía vertical
blank:	salida, al cliente, selecciona una señal en blanco
hcount:	salida, 11 bits, al cliente, selecciona un contador de pixel horizontal
vcout:	salida, 11 bits, al cliente, selecciona un contador de pixel vertical

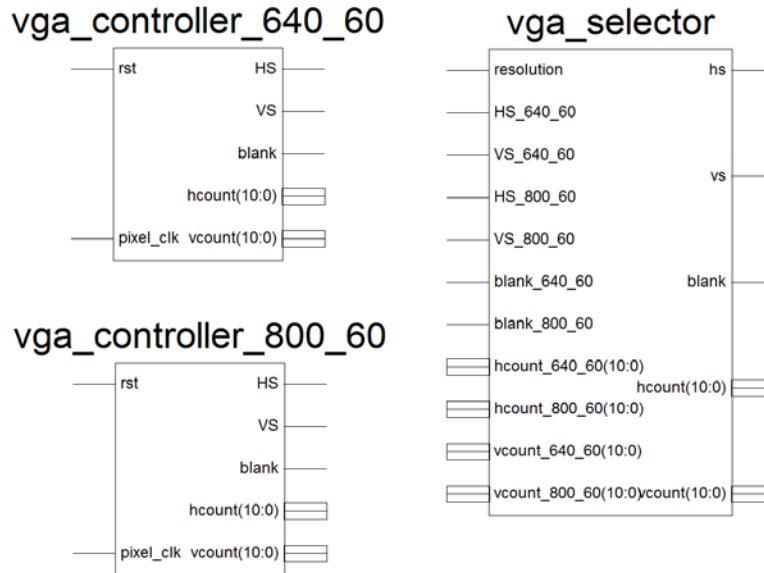


Figura 80. Estructura del componente VGA.

El controlador VGA 640x480

El `vga_controller_640_60` utiliza un reloj 25[MHz] para generar los pulsos de la sincronía del vídeo para el modo de 640x480 a 60[Hz]. También proporciona los contadores horizontales y verticales para el pixel actualmente mostrado y una señal en blanco que sea activa cuando el pixel no está dentro de la pantalla visible (las salidas del color se reajustan a 0).

La señal en blanco se retrasa un período de reloj del pixel (40 [ns]) de cuando el pixel sale de la pantalla visible para explicar el retraso del pixel entubado. Este retraso sucede porque los datos del color no llegan a los pins del monitor al mismo tiempo que los contadores que indican el pixel actualmente mostrado (la lectura de memoria se retrasa, se retrasa la sincronización). Las sincronizaciones del ancho del pulso para los intervalos de frente y retraso se basan en observaciones de las exhibiciones de VGA. Los intervalos del pórtico son los tiempos pre-sincronía y del pulso post-sincronía, durante los cuales la información no puede ser exhibida.

EL Controlador VGA 800x600

El `vga_controller_800_60` utiliza un reloj 25 [MHz] para generar los pulsos de la sincronía del vídeo para el modo de 800x600 60 [Hz]. La funcionalidad es similar al componente `vga_controller_640_60`.

El Selector VGA

El `vga_selector` contiene la lógica que selecciona entre la salida de los reguladores de VGA dependiendo de la resolución usada. Dependiendo de la resolución de la señal de entrada, las señales de salida de los reguladores de VGA se seleccionan y se envían a otros componentes de la lógica en el diseño que necesiten estas salidas, o al monitor. Si el pin de la resolución se coloca en alto, se selecciona la resolución 800x600 y si se coloca abajo, se selecciona la resolución 640x480.

5.3 Uso del Ratón mediante el puerto PS/2 en un monitor VGA

El componente de referencia del ratón es un regulador del ratón con interfaz *PS/2*. Hay muchas aplicaciones para este componente, pero el más común es un ratón en un monitor *VGA*. La mayor parte de las inicializaciones y los acercamientos descritos aquí implican este uso común.

El componente de referencia del ratón se comunica a través de la interfaz *PS/2* y se inicializa en el ciclo inicial o el reinicio. Una vez que se permita el modo del movimiento en sentido vertical (si el ratón tiene una rueda), la resolución y la tasa de muestreo se fija, y se permite la divulgación de los datos (espera para que los paquetes de los datos lleguen del ratón). Entonces los paquetes de datos del movimiento se decodifican para generar las señales de salida que representan desplazamientos de *X* y de *Y*, estados del botón, y la rotación de la rueda del ratón (*Z*, cuando sea disponible).

Descripción Funcional

Según las indicaciones de la Figura (64), el componente de referencia del ratón se compone de tres módulos principales: el módulo de interfaz *PS/2*, que es responsable de la comunicación entre el ratón y el controlador; el controlador del ratón, que recibe los paquetes de datos y las salidas de la posición del ratón y los estados del botón; además la información de la resolución del ratón, que proporciona los límites activos de la pantalla en las fronteras de *X* y de *Y*. El tercer componente recibe la información de la resolución y calcula los valores máximos en las fronteras de *X* y de *Y* lo que fija la posición inicial del ratón.

El Controlador del Ratón

Cuando se tiene el estado del reinicio, el controlador reinicia el ratón y comienza el procedimiento de inicialización. Cuando se lleva a cabo el “*BAT*” (prueba de aseguramiento básico por sus siglas en ingles) y envía el resultado de la prueba al *host* anfitrión, seguido por la identificación del ratón, *00h*. Entra después en modo de flujo de datos, pero con la divulgación de los datos inhabilitada. Esto significa que no enviará los paquetes de datos del movimiento cuando se mueve el ratón o el estado de los cambios del botón. Tampoco tiene el modo de movimiento en sentido vertical habilitado.

El controlador primero reinicia el ratón, espera el resultado del “*BAT*” y la identificación del ratón, y después comienza la habilitación del procedimiento en modo del movimiento en sentido vertical. El procedimiento consiste en fijar sucesivamente la tasa de muestreo a 200, y luego a 100, y de finalmente a 80. Después de que se haga esto, el controlador solicita la identificación del ratón enviando el octeto de *F2h* al ratón (comando de petición de la identificación). Si la identificación recibida es *00h* entonces el ratón no tiene rueda, o si la identificación es *03h*, así el ratón ha incorporado el modo rueda e incluirá la información del movimiento de *Z* en sus paquetes de datos del movimiento. Observe que el reporte de los datos todavía está inhabilitado en este punto. Después, la resolución se fija a ocho counts/mm y la razón del muestreo se fija a 40 muestras por segundo. (Veáse Figura 81).

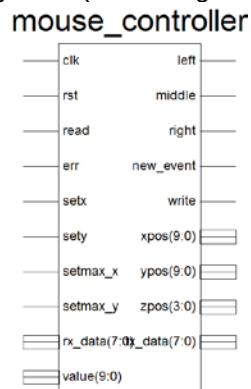


Figura 81. Controlador del ratón.

El último comando enviado al ratón es el comando para habilitar el reporte (*F4h*). Después de esto, el ratón envía los paquetes de datos del movimiento cada vez que se mueve el ratón, los estados del botón cambian o se mueve la rueda del ratón (cuando está disponible). El controlador ahora incorpora otro estado, esperando octetos del ratón. El ratón envía el octeto del reconocimiento (*FAh*) después de que cada octeto que recibe del host anfitrión.

Cuando la rueda del ratón esta deshabilitada o el ratón no tiene una rueda, los paquetes de datos del movimiento consisten de tres octetos enviados consecutivamente. Cuando se permite el modo del movimiento en sentido vertical, envía cuatro octetos.

El primer octeto recibido contiene los estados del botón en los menos tres bits significativos, si uno de estos bits es alto, entonces el botón de ratón correspondiente se presiona. El bit de la muestra de X y de Y es el bit con más valor significativo del movimiento de X y de Y. Se representan los movimientos de X y de Y usando codificación de nueve-bits de complemento a dos. Así, el intervalo del movimiento es -256 a 255. El movimiento delta de la última posición del ratón se envía a cualquier eje. El ratón no pierde de vista la posición absoluta del ratón. El controlador utiliza los movimientos del delta recibidos para no perder de vista la posición absoluta. El ratón utiliza la esquina bajo-izquierda para el origen de las fronteras, significa que el ratón reporta un desplazamiento positivo en X cuando se está movido hacia la derecha y negativo (el bit de muestra de X es el "1") cuando está movido a la izquierda. Cuando el ratón se mueve hacia arriba, la dislocación de Y es positiva, y cuando se está moviendo hacia abajo es negativo. El monitor utiliza la esquina izquierda para el origen de las fronteras. El controlador niega el desplazamiento de Y antes de agregarla a la posición anterior de Y para divulgar la posición concerniente al origen de las hachas del monitor. La posición actual del ratón puede ser fijada en cualquier momento poniendo la posición horizontal o vertical deseada (esto es el origen de las fronteras del monitor) respecto al valor de entrada. Entonces se fija X en alto como la señal de entrada (se fija Y para la posición vertical) para un período de reloj.

De igual forma, los límites superiores del ratón pueden ser fijados poniendo el valor de esos límites en la entrada del valor y poniendo el *setmax_x* o el *setmax_y* para un período de reloj. Se hace esto cuando la resolución cambia al tamaño donde el cursor del ratón (su punto caliente) está en medio de la pantalla y también evita que el cursor salga del área visible.

Módulo Informador para la Resolución del Ratón

La *resolution_mouse_informer* implementa la lógica en ejecución que fija la posición del ratón cuando se enciende el *FPGA* o cuando la resolución cambia. También fija los límites del ratón según la actual resolución. El ratón se centra para la resolución actual-seleccionada y los límites se fijan por consiguiente. Figura (82).

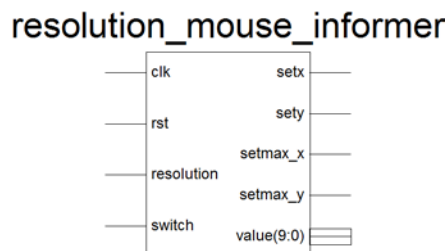


Figura 82. Diagrama esquemático generado en Xilinx del bloque para la resolución del mouse.

De esta manera, el cursor del ratón aparecerá en el centro de la pantalla en la posición de inicio, y cuando cambia la resolución no saldrá de la pantalla. La posición y los límites, se fijan poniendo la coordenada del punto central en el valor de la salida y activando la señal determinada correspondiente

(se fija-x para la posición horizontal, se fija-y para la posición vertical, el setmax_x para el valor máximo horizontal, y setmax_y para el valor máximo vertical).

Programa que define las interfaces Ratón – PS/2 - VGA

El programa implementado en *VHDL* tiene las siguientes características en cuanto a sus entradas y salidas se refiere:

11 - Entradas

clk: entrada de 1 bit;
reset: entrada de 1 bit;
pixel_clk: entrada de 1 bit;
xpos: entrada de 10 bits;
ypos: entrada de 10 bits;
hcount: entrada de 11 bits;
vcount: entrada de 11 bits;
blank: entrada de 11 bits;
red_in: entrada de 4 bits;
green_in: entrada de 4 bits;
blue_in: entrada de 4 bits.

3 - Salidas

red_out: salida de 4 bits;
green_out: salida de 4 bits;
blue_out: salida de 4 bits;

Para el caso específico de los colores de salida se usan 3 dip switch de la tarjeta.

Para el caso de los botones del ratón se usaron 3 led de la tarjeta.

Para las dos señales que maneja el programa se enlistan las siguientes:

Señal *xdiff*: de 4 bits para la diferencia entre la posición actual y la referencia del ratón.

Señal *ydifff*: de 4 bits para la diferencia entre la posición actual y la referencia del ratón.

La implementación de las aplicaciones de las secciones 5.1, 5.2 y 5.3 se pueden observar conjuntamente en el diagrama esquemático de la Figura (83), que corresponden al manejo del puerto *PS/2*, al manejo del monitor *VGA* y al manejo o control del ratón. En el caso de la resolución para el ratón se toma únicamente la de 640 X 480 y para el *VGA* la resolución de 640 X 480, la resolución de 600 X 800 se deja anclada a tierra. Para el manejo de los relojes se empleo un divisor de frecuencia ya que el reloj de la tarjeta es de 50 [MHz] y la implementación usa 25 y 50 [MHz].

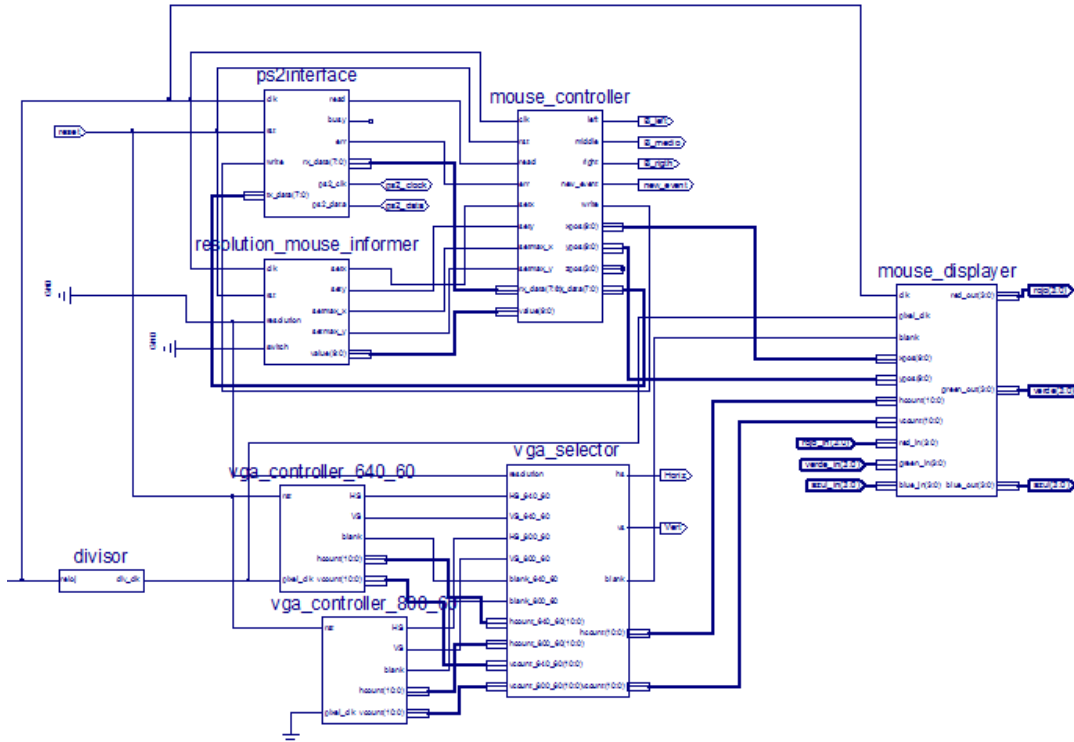


Figura 83. Diagrama esquemático de los controladores del ratón y para la interface PS/2 y VGA.

En el video de la Figura (84) podemos ver el funcionamiento de las interfaces en la tarjeta Spartan 3E. Nota hacer doble click sobre la imagen.



Figura 84. Video del funcionamiento de las interfaces ratón-PS/2- VGA.

Para el caso del consumo de recursos en el *FPGA* estos se muestran en la Figura (85) y el código de la implementación en *VHDL* se muestra en el Anexo IV al final del reporte.

raton Project Status				
Project File:	Raton.xise	Parser Errors:	No Errors	
Module Name:	raton	Implementation State:	Programming File Generated	
Target Device:	xc3s500e-4fg320	• Errors:	No Errors	
Product Version:	ISE 13.3	• Warnings:	38 Warnings (0 new)	
Design Goal:	Balanced	• Routing Results:	All Signals Completely Routed	
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	All Constraints Met	
Environment:	System Settings	• Final Timing Score:	0 (Timing Report)	
Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	279	9,312	2%	
Number of 4 input LUTs	534	9,312	5%	
Number of occupied Slices	353	4,656	7%	
Number of Slices containing only related logic	353	353	100%	
Number of Slices containing unrelated logic	0	353	0%	
Total Number of 4 input LUTs	582	9,312	6%	
Number used as logic	534			
Number used as a route-thru	48			
Number of bonded IOBs	34	232	14%	
Number of BUFGLUXs	2	24	8%	
Average Fanout of Non-Clock Nets	3.58			

Figura 85. Resumen de los recursos consumidos en el FPGA de la aplicación desarrollada en Xilinx.

5.4 Uso del Teclado mediante el Puerto PS/2 en un LCD

Para el desarrollo de esta aplicación se hace uso de dos interfaces de la tarjeta Spartan 3E que son: el puerto PS2 y el manejo de datos sobre el *display* de cristal líquido. Esta aplicación se desarrollo en Xilinx versión 13.3 y se emplearon dos drivers para poder escribir mensajes en el LCD desde el teclado, los cuales se pueden mostrar a continuación.

Manejador PS/2.- En la sección 5.1 se mostró que el puerto PS/2 contiene 6 pines: 1 pin de dato, 1 pin de reloj, el pin de Vcc y GND y dos pines restantes sin utilizar. El pin del dato es el más importante ya que permite la comunicación serial entre el teclado y la tarjeta SPARTAN 3E. Este dato tiene recepción como transmisión teniendo el siguiente tratamiento de señal: 1 bit de start, 8 bits de dato, 1 bit de stop, *odd parity*. El puerto de salida *scancode* dado en el driver del teclado permite que enviara el código binario de todos los caracteres del teclado para poder luego mostrarlos en la LCD. Para ver con detalle el puerto PS/2 hay que ir a la Figura (36) y a la Tabla (4) de este reporte, o simplemente revisar el manual de la tarjeta de desarrollo. Para ver el código binario que usa el driver del teclado hay que ir a la Figura (37) de este reporte.

El LCD es un dispositivo de gran ayuda en el campo de la electrónica, es una pantalla de cristal líquida que se forma de un número de pixeles monocromos (el más utilizado es de resolución 2 x 16); la interfaz entre la tarjeta SPARTAN 3E y la LCD se da por 4 pines de datos < SF_D > correspondientes a (M15, P17, R16 Y R15) y 4 pines de control < lcd_rs, lcd_rw, lcd_e, sf_ce0 > que son los (L18, L17, y M18) respectivamente, estos están representados en la Figura (57) de la sección 4 de este reporte.

Programa que define el uso del Teclado mediante el puerto PS/2 y el LCD

Entradas

ps2_data: entrada de 1 bit que es dato del teclado;
ps2_clk: entrada de 1 bit que es dato del teclado;
reset: entrada de 1 bit para reiniciar el programa;
clk: entrada de un bit que es el reloj general del programa;

Salidas

sf_d: salida de 4 bits; --lcd pines de datos
lcd_e, lcd_rs, lcd_rw, sf_ce0: salida de 1 bit; --lcd pines de control

Señales

Señal código: de 8 bits;--nos ayuda a identificar las teclas del teclado
señal1 de 8 bits para cada carácter del lcd en total 32;
señal2 de 8 bits para cada carácter del lcd en total 32;
Señal temp_1, temp_2, temp_3: de 2 bits para control "00";

Señal a: de 1 bit;
 Señal busy, puls: de 1 bit;
 Señal b: de 4 bits para el manejo de tiempos;
 Señal d: de 4 bits para el manejo de tiempos;
 Señal t: de 1 bit:=0';
 Señal reseteo: std_logic:=0';
 Señal segundos: integer:=0;
 Señal x1,x2,x3,x4,x5: bit;
 Señal i: integer range 0 to 50000000:=0;
 Máquina de 64 estados;
 Señal edo_presente, edo_futuro: estados:= d0;

Esta implementación se basó en sólo la programación para el manejo del PS/2 y del manejo del cristal líquido de la tarjeta Spartan 3E y se muestra su funcionamiento en la Figura (86), *Nota: hacer doble click en la Figura*. De igual forma el código de la implementación se muestra en el Anexo V al final del reporte.



Figura 86. Video que muestra del funcionamiento del teclado PS/2 y su escritura en el LCD.

El consumo de recursos sobre el FPGA de la aplicación se muestra en la Figura (87).

interps2_lcd Project Status (12/09/2011 - 14:17:36)				
Project File:	Interps2_lcd.xise	Parser Errors:	No Errors	
Module Name:	interps2_lcd	Implementation State:	Programming File Generated	
Target Device:	xc3e500e-4fg320	Errors:	No Errors	
Product Version:	ISE 13.3	Warnings:	136 Warnings (10 new)	
Design Goal:	Balanced	Routing Results:	All Signals Completely Routed	
Design Strategy:	Xilinx Default (unlocked)	Timing Constraints:	All Constraints Met	
Environment:	System Settings	Final Timing Score:	0 (Timing Report)	
Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Total Number Slice Registers	414	9,312	4%	
Number used as Flip Flops	157			
Number used as Latches	257			
Number of 4 input LUTs	513	9,312	5%	
Number of occupied Slices	455	4,656	9%	
Number of Slices containing only related logic	455	455	100%	
Number of Slices containing unrelated logic	0	455	0%	
Total Number of 4 input LUTs	573	9,312	6%	
Number used as logic	513			
Number used as a route-thru	60			
Number of bonded IOBs	12	232	5%	

Figura 87. Recursos consumidos por la implementación y uso de las interfaces PS/2 para el teclado y su escritura en el LCD, resumen obtenido por Xilinx.

6. Desarrollo de un transportador “transceiver” para la interfaz USB

En esta sección se plantea un sistema donde se cubran algunos métodos computacionales, algoritmos aritméticos y técnicas de mejora del diseño necesario para obtener implementaciones eficientes y de alto rendimiento de un protocolo de comunicación “Universal Serie Bus” en plataformas de hardware reconfigurable, conocido también como “transceiver”. Esto se llevará a cabo mediante

programación estandarizada por la IEEE (ANSI/IEEE 1076-1993) como es el lenguaje de programación VHDL, que es usado para describir circuitos digitales.

6.1 Implementación de transferencia de datos sobre USB

Esta implementación en lenguaje “VHDL” se realiza en combinación con un circuito integrado de transferencia *USB* compatible con la *UTMI* (*Transceiver Macrocell Interface*: por sus siglas en inglés). Este programa trabaja como un dispositivo *USB* que transfiere una cadena de bytes en ambas direcciones sobre el bus de comunicación.

Una de las características de la implementación es una aplicación de interfaz en términos del tiempo de la transferencia del byte “*byte-at-time*”, **primero en entrar – primero en salir** (*FIFO*: por sus siglas en inglés). Las complicaciones alrededor del protocolo de comunicación *USB* están contenidas dentro de la implementación, como son: paquetes de bits, puntos finales o colas de datos y descriptores. El desarrollo de la implementación soporta los estándares *USB 1.1* y *USB 2.0*; con “*full-speed*” de 12 [Mbits/s] o de “*high-speed*” con unos 480 [Mbits/s].

También el desarrollo es compatible con la clase de dispositivos estándar de comunicación *CDC-ACM* (*Communication Device Class – Association for Computing Machinery*), lo cual implica que no son necesarios manejadores especiales en la computadora.

Entre otras características están la de reemplazar la comunicación *RS-232* con la *PC*, el dispositivo aparece como una *TTY* o como un puerto *COM* virtual, no es necesario modificar el software de la *PC*. Es de rápido intercambio de datos con la *PC*. La capacidad de alta velocidad es soportada por el programa e intercambia 30 [Mbyte/s] entre el *FPGA* y la *PC*. Hay que tomar en cuenta que el *USB* transfiere tasas que dependen de las características de la tarjeta madre de la *PC* y de la carga del *CPU*. Software personalizado será necesario en la *PC*, los manejadores de los puertos *COM* virtuales no están desarrollados para altas tasas de transferencias de datos.

6.2 Operación del programa

El dispositivo se enciende después de un pulso de “reset” (inicio), entonces el programa prepara al bus *USB*. Si la capacidad para alta velocidad es soportada, el programa intenta un modo de alta velocidad con el “host”. De otra manera o en caso de que la negociación de alta velocidad falla, el programa opera en modo de “full-speed”. Esto responde a los estándares requeridos por el dispositivo para que el host obtenga una dirección y publique el descriptor del dispositivo.

El descriptor del dispositivo indica compatibilidad con *CDC-ACM*. El *ID* del producto, la *ID* del vendedor y la versión del producto pueden ser configuradas en un tiempo compatible. Adicionalmente, cadenas de texto opcional pueden especificarse por el nombre del fabricante, el nombre del producto y el número de serie del producto. Figura (88).

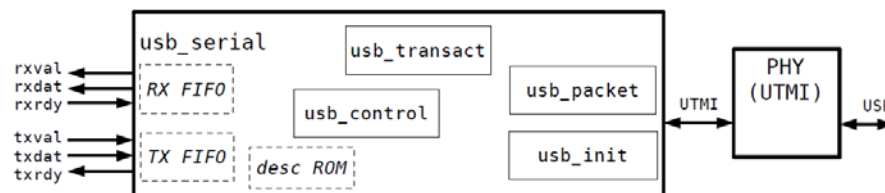


Figura 88. Diagrama del *usb_serial* con interfaz externa y sub-entidades.

6.3 Puntos terminales o puntos de lectura y escritura

Dos terminales (endpoints) son usados para transferir datos, uno para enviar y otro para recibir. Una terminal extra es implementada y requerida por la especificación de la *CDC-ACM* pero el tránsito sobre esa terminal es ignorado. Los datos recibidos se toman en una *FIFO* hasta que son aceptadas por la aplicación. De igual forma la aplicación coloca los datos en la transmisión *FIFO* hasta que puedan ser enviados sobre el bus *USB*. La aplicación interactúa con estos *FIFO* de un byte a la vez. El programa requiere una señal de reloj de 60 [MHz], sincronizada con las señales de las interfaces *UTMI*. Esta señal de reloj es normalmente generada por el *transceiver UTMI*.

Nota: Se requiere un circuito integrado conocido como *transceiver* USB 2.0 entre el programa y las líneas físicas del *USB*. Este *transceiver* debe ser compatible con el estándar (*USB 2.0 Transceiver Macrocell Interface*). Debe usarse una interfaz de 8-bit de datos a 60 [MHz].

6.4 Interfaz de Aplicación

La aplicación recibe datos leídos desde el *Rx FIFO*, cuando la aplicación esta lista para leer datos, la señal *RxRDY* se confirma. En respuesta a *RxRDY*, si la *FIFO* no está vacía, el programa confirma la señal *RxVAL* y coloca un byte de dato sobre *RxDAT*. Siempre que ambas señales *RxRDY* y *RxVAL* están en alto sobre un flanco de subida del reloj, el programa asume que el byte ha sido consumado y se mueve sobre el siguiente byte. Si la aplicación no desea leer más datos, debería confirmar a la señal *RxRDY* antes del siguiente flanco de subida del reloj.

La aplicación envía datos colocándolos en la *Tx FIFO*, mientras que la *FIFO* no está llena el programa confirma *TxRDY*. Cuando la aplicación esta lista para enviar los datos, se confirma *TxVAL* y coloca un byte de dato sobre *TxDAT*. Siempre que ambas señales *TxRDY* y *TxVAL* están en alto sobre un flanco de subida del reloj el programa acepta un byte de *TxDAT* y lo coloca en el *FIFO*. La aplicación debe también proporcionar el siguiente byte de dato sobre *TxDAT* o des- confirmar *TxVAL* antes del siguiente flanco de subida del reloj. Figura (89).

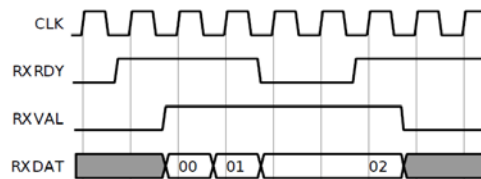


Figura 89. Ciclos de reloj para la interfaz *RxFIFO*.

6.5 Las opciones de configuración

Las opciones de configuración se muestran en la Tabla (15).

Tabla 15. Opciones de configuración para el dispositivo *USB* de transferencia.

Dato Genérico	Función que describe	Tipo de dato
<i>VENDORID</i>	Vendedor ID reportada en el descriptor del dispositivo	16 - bit
<i>PRODUCTID</i>	Producto ID reportada en el descriptor del dispositivo	16 - bit
<i>VERSIONID</i>	Producto versión reportada en el descriptor del dispositivo	16 - bit
<i>VENDORSTR</i>	Nombre Opcional del Fabricante	String
<i>PRODUCTSTR</i>	Descripción del producto opcional	String
<i>SERIALSTR</i>	Cadena del número de serie opcional	String
<i>HSSUPPORT</i>	Incluye soporte para el modo de alta velocidad	Booleano
<i>SELFPOWERED</i>	Posible bit de autopotencia y palabra de estatus del descriptor	Booleano
<i>RxBUFSIZE_BITS</i>	Tamaño del <i>FIFO</i> recepción	7-12
<i>TxBUFSIZE_BITS</i>	Tamaño del <i>FIFO</i> transmisión	7-12

6.6 Software Anfitrión “Host”

La transferencia de datos se hace usando dos terminales, los datos enviados por el host al dispositivo a través de la terminal “*endpoint1_out*” actuales se reciben en el *FIFO*. Los datos requeridos en la *FIFO* de transmisión son enviados al host a través de la terminal “*endpoint1_in*”.

El dispositivo es accesible a partir de que el software se ejecuta sobre el host. Esto comúnmente requiere interacción con el manejador del *USB* en el sistema operativo, es decir, depende del sistema operativo empleado.

Bajo el sistema operativo Windows, el dispositivo trabaja con el manejador *usbser.sys* (proporcionado con windows) y aparecerá como un puerto *COMn* extra. Luego un manejador *.INF*

personalizado puede ser necesario para decirle al Sistema Operativo Windows que ese manejador será usado. Para esto se debe observar el archivo `fpgaser.inf` para ver cómo puede realizarse. Note que el `productID` y el `vendorID` en la documentación/`usb/gadget_serial.txt` en el kernel de la trayectoria de Linux. En la versión de Windows 200 de `usbser.sys` esta notoriamente quebrado y no debería usarse. Windows XP pareciera hacerlo mejor. El control de flujo no ha sido probado y probablemente no funcione correctamente.

6.7 Síntesis

El código VHDL desarrollado puede ser sintetizado en *ISE Project* de la suite de Xilinx versión 13.3 para tarjetas spartan-3E y su familia de dispositivos *FPGA*. Nota: este desarrollo puede ser portable a otras plataformas de *FPGA*'s. Salvamos el problema que interfería con el bloque de las *RAM* ya que el *FIFO* receptor, el *FIFO* transmisor y descriptor *ROM* están codificados en *VHDL* de tal manera que la herramienta de Xilinx para sintetizar automáticamente infiere sobre los bloques primitivos de *RAM*. Para la implementación es necesario editar el archivo de restricción de área *.UCF* para la tarjeta Spartan-6, que contiene los *FPGA*'s de la familia *XC3S500e-4fg320* y además configurar el *transceiver* contenido dentro de la tarjeta.

6.8 Recursos utilizados

Indicación de los requisitos de recurso (Figura 90), basada en la síntesis de `usb_serial` como el diseño del nivel superior para un Xilinx *XC3S500e-4fg320*. (Xilinx *Webpack* 13.3).

usbtest Project Status			
Project File:	usb_trans.xise	Parser Errors:	No Errors
Module Name:	usbtest	Implementation State:	Synthesized
Target Device:	xc3s500e-4fg320	•Errors:	No Errors
Product Version:	ISE 13.3	•Warnings:	31 Warnings (0 new)
Design Goal:	Balanced	•Routing Results:	
Design Strategy:	Xilinx Default (unlocked)	•Timing Constraints:	
Environment:	System Settings	•Final Timing Score:	

Device Utilization Summary (estimated values)				[-]
Logic Utilization	Used	Available	Utilization	
Number of Slices	687	4656		14%
Number of Slice Flip Flops	393	9312		4%
Number of 4 input LUTs	1300	9312		13%
Number of bonded IOBs	31	232		13%
Number of BRAMs	4	20		20%
Number of GCLKs	1	24		4%

Figura 90. Recursos consumidos en el *FPGA* de la tarjeta Spartan 3E, resumen obtenido por Xilinx.

6.9 Pruebas

Se realizan pruebas para el programa desarrollado en *VHDL* y su implementación sobre la tarjeta Spartan-3E de Xilinx para probar la transferencia de datos *USB*. Para esta prueba se debe implementar una máquina de estados que debe responder ya sea a comandos físicos de la tarjeta o comandos recibidos desde un host. Se puede usar un minicom (sobre Linux) o una hiper-terminal (sobre windows) para conectar el dispositivo y hacerlo funcionar. Existe un *script Python* conocido como *testdev.py* que prueba varios escenarios de control de flujo de datos. También existe un programa desarrollado en lenguaje C, `perftest.c` que emplea la librería `libsub` para probar el ancho de banda de la transferencia de datos [2].

6.10 Procedimiento de prueba

Con el dispositivo Spartan-3E conectado sobre un host que tiene un sistema operativo Linux, se procede a realizar los siguientes pasos:

- 1.- Se debe construir el archivo `usbtest.bit` necesario para la implementación sobre la tarjeta, mediante la corrida de la implementación en Xilinx.
- 2.- Bajar el archivo *.BIT* antes creado al modulo *FPGA* de la tarjeta.

3.- Ejecutar “*lsusb*” para checar que el dispositivo es reconocido. Observar también los mensajes del kernel para checar que *CDC-ACM* soporta, lo cual debería aparecer como un nuevo dispositivo *ttyACM0*.

4.- Use el *minicom* para conectar al */dev/ttyACM0*. Escribir “*HOLA*” y presionar “enter”. El dispositivo debería responder “*ALOH*”.

5.- Ejecutar *Phyton testdev.py /dev/ttyACM0* para empezar la prueba exhaustiva, anotar el tiempo que tarda la máquina en que se probará. Esta prueba debería tardar cerca de 20 minutos dependiendo la potencialidad de la máquina usada.

6. Ejecutar *perftest -d fb9a:fb9a* (como administrador o root) para empezar la prueba del desempeño.

6.11 Desempeño de la implementación

El desempeño de la implementación se muestra en la Tabla (16).

Tabla 16. Prueba realizada en Ubuntu 11-04 con libusb-1.0, usando requisitos asíncronos.

Velocidad del Bus	Ancho de Banda medido de entrada en [byte/s]	Ancho de Banda medido de salida en [byte/s]
Full-Speed	1088118	1088073
High-Speed	43727704	32635212

6.12 Alcances conocidos del desarrollo

- La característica de *TEST_MODE* no es implementada (las cuales son obligatorias para dispositivos de alta velocidad). Esto no debería afectar la operación normal de los dispositivos, pero el dispositivo se podría conformar con las pruebas de falla del funcionamiento del USB.
- El control establecido estrictamente no verifica los requerimientos del host. Como un resultado pueden aparecer requisitos inválidos del host mediante envío de errores. Esto no debería suceder con un controlador del host correcto pero probablemente podría causar fallas en las pruebas del USB.
- En esta configuración del descriptor, el programa negaría soportar comandos del estilo *AT*. Lo cual no es cierto, pero es necesario traer el reconocedor del dispositivo mediante el manejador de Linux *CDC-ACM*. En realidad, el programa ignora los requisitos obligatorios *CDC* de *SET _ENCAPSULATED_COMMAND* y *GET _ENCAPSULATED _RESPONSE*.
- En ciertas situaciones, el dispositivo puede enviar más bytes que los esperados por el host. Esto esta abanderado como un error “babble” emitido por el host. Puede ocurrir por ejemplo cuando el host envía un *IN request*, el cual es un múltiplo del tamaño del paquete máximo. Para evitar esto, siempre envía un *IN request* con el tamaño de transferencia colocado como un múltiplo del paquete máximo. Note que esto no es un “bug” en el programa del dispositivo, sino que es una inconveniencia en la pila del software del USB.
- Dependiendo de la plataforma *FPGA* y de la tarjeta empleada el encendido de transmisión del bus podría no responder a los request iniciales del host después de un “*plugging in*”, es decir, introducir el dispositivo. Por ejemplo, un *FPGA Xilinx* necesita comúnmente más de 100 [ms] para inicializarse después de encenderse. Debido a este retardo de tiempo el host habría intentado comunicarse con el dispositivo. La mayor parte de los sistemas operativos tendrían éxito en el reintento del procedimiento de inicialización.
- La interfaz de aplicación sobre el programa debe estar sincronizada a los 60 [MHz] *UTMI* del reloj. Esto es un inconveniente cuando el programa esta embebido en grandes sistemas con su propio reloj de muestreo. Para sobre llevar este problema debe crearse una separación del dominio de reloj dentro del programa para desacoplar el reloj de la aplicación del reloj *UTMI*.
- Hay varias fallas en la versión de Windows 2000 del *usbser.sys*. Esto se conoce como el congelamiento cuando se recibe un inesperado paquete de tamaño cero. Esto sucede si el dispositivo envía un múltiplo exacto del tamaño máximo de los paquetes mientras el host intenta leer exactamente la misma cantidad de datos.

Conclusiones

Las interfaces desarrolladas proporcionarán soporte a diseños e implementaciones futuros del Grupo Académico de Modelado y Simulación de Procesos, para desarrollar con eficiencia sus actividades y proyectos. En este trabajo se desarrollaron las interfaces mostrando su funcionamiento, en algunas de ellas en videos demostrativos, además de mostrar sus diagramas esquemáticos de la construcción y se presentaron en los anexos los códigos de los programas para cada aplicación desarrollada, en lenguaje *VHDL* que soporta la plataforma de Xilinx en la tarjeta de desarrollo Spartan 3E Starter kit. Cabe señalar que estas interfaces pueden ser migradas o utilizadas en otros sistemas de desarrollo de hardware reconfigurable, en si en otras tarjetas de desarrollo, con sus debidos ajustes en las restricciones de área de los archivos *.UCF* y algunas conexiones.

Resumiendo en este trabajo se desarrollaron interfaces para aplicaciones visuales como: despliegue de 7 segmentos (en particular se diseño un reloj digital), envío de mensajes al *display* de cristal liquido de la tarjeta Spartan 3E, el despliegue de un cursor de ratón en un monitor VGA, control de movimientos del ratón y manejo de colores dentro del monitor VGA. También se mostraron interfaces para interruptores, pulsadores y teclados como: Ratón (botones del ratón, izquierdo-medio-derecho) y teclado mediante el puerto *PS/2*. Se desarrolló una interface de comunicación como el diseño del *transceiver* para la comunicación *USB*.

Este trabajo complementa varios diseños del grupo desarrollados en esta plataforma, aunque en realidad todavía se tiene trabajo a futuro. Las perspectivas del mismo es desarrollar otro tipo de interfaces para que puedan dar, como ya se mencionó, soporte a otras aplicaciones específicas. Un ejemplo de esto es el envío, almacenaje y manipulación de datos en memorias *RAM* de manera masiva.

En particular se plantea trabajar en el área de seguridad de la información en hardware, el diseño y desarrollo de algoritmos criptográficos, además de implementación en hardware de firmas o huellas digitales. Estos proyectos además están dirigidos al diseño y desarrollo de sistemas embebidos autónomos.

El desarrollo de este sistema de integración (software y hardware reconfigurable) para la transferencia de datos *USB* en un *FPGA* junto con el *transceiver* puede ser utilizado para cualquier aplicación que use este protocolo de comunicación. Este desarrollo tiene sus limitaciones como se explicó en los alcances conocidos del dispositivo *USB* implementado. Siendo los más importantes las tasas de transferencia de datos y la velocidad de respuesta de los *FPGA*, porque la implementación puede ser conectada y probada en cualquier equipo con puerto *USB* que reconozca al dispositivo con los manejadores apropiados. El interés de este desarrollo para nuestro grupo académico de modelado y simulación es la transferencia de información cifrada mediante un algoritmo de criptográfico por bloques convertido a flujo con modos de operación antes presentados en otros trabajos (Padrón et al., 2011) y (Prieto et al., 2011). El código en *VHDL* de la implementación se muestra en el Anexo VI al final del reporte.

El empleo de estas bibliotecas se puede extender para otras familias de tarjetas de integración, a parte de la empresa Xilinx, siempre y cuando éstas tengan integradas las interfaces y conectores correspondientes. Aunque esto no es una limitante, ya que como lo hicimos en este trabajo en la sección (6.3), se pueden implementar o desarrollar algunas otras interfaces para una aplicación en particular. Para el mismo desarrollo de estas nuevas interfaces la ventaja es que ya no se tiene que iniciar de cero si no se pueden retomar las implementadas en este trabajo.

Anexo I.- Datos para las restricciones de área del BCD

```
Archivo .UCF
# PlanAhead Generated physical constraints
NET "X0" LOC = L13;
NET "X1" LOC = L14;
NET "X2" LOC = H18;
NET "X3" LOC = N17;
NET "Fa" LOC = F12;
NET "Fb" LOC = E12;
NET "Fc" LOC = E11;
NET "Fd" LOC = F11;
NET "Fe" LOC = C11;
NET "Ff" LOC = D11;
NET "Fg" LOC = E9;
```

Anexo II.- Código en VHDL para el desarrollo de un reloj digital

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;
entity reloj is
    Port ( clock : in std_logic;
          rel  : out std_logic;
          ctl  : out std_logic_vector(1 downto 0));
end reloj;
architecture Behavioral of reloj is
    constant value : integer := 50E6;
    constant value1: integer := 25E3;
    signal rel1 : std_logic;
    begin
        process (clock)
            variable clk: integer range 0 to value;
            variable clk1: integer range 0 to value1;
            begin
                if (clock'event and clock = '1') then
                    if (clk = value) then
                        clk := 0;
                        rel <= '1';
                    else
                        clk := clk+1;
                        rel <= '0';
                    end if;
                end if;
                if (clock'event and clock = '1') then
                    if (clk1 = value1) then
                        rel1 <= '1';
                        clk1 := 0;
                    else
                        clk1 := clk1+1;
                        rel1 <= '0';
                    end if;
                end if;
            end process;
        end process;
```

```

process (rel1)
variable contador : std_logic_vector(1 downto 0);
begin
    if (rel1'event and rel1 = '1') then
        if(contador = "11") then
            contador := "00";
        else
            contador := contador + 1;
        end if;
    end if;
end if;

ctl <= contador;
end process;
end Behavioral;

```

Archivo .UCF

```

#PACE: Start of Constraints generated by PACE
#PACE: Start of PACE I/O Pin Assignments
NET "anodos<0>" LOC = "D7" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6 ;
NET "anodos<1>" LOC = "C7" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6;
NET "anodos<2>" LOC = "F8" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6;
NET "anodos<3>" LOC = "E8" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6;
NET "clock1" LOC = "C9" ;
NET "HAB" LOC = "L13" | IOSTANDARD = LVTTTL | PULLUP;
NET "hab1" LOC = "L14" | IOSTANDARD = LVTTTL | PULLUP;
NET "reset" LOC = "K17" | PULLDOWN ;
NET "seg<0>" LOC = "B4" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6 ;
NET "seg<1>" LOC = "A4" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6 ;
NET "seg<2>" LOC = "D5" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6 ;
NET "seg<3>" LOC = "C5" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6 ;
NET "seg<4>" LOC = "A6" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6 ;
NET "seg<5>" LOC = "B6" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6 ;
NET "seg<6>" LOC = "E7" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6 ;
NET "HAB" CLOCK_DEDICATED_ROUTE = FALSE;

```

Anexo III.- Código en VHDL para el manejo de un LCD

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.teclao_pcg.all;
use work.lcdpkg.all;
-- Asignacion de Pines : clk(C9), reset(L13), ps2_clk(G14), ps2_data(G13), - lcd_rs(L18), lcd_rw(L17), lcd_e(M18),
sf_ce0(D16),
-- sf_d<0>(R15),sf_d<1>(R16), sf_d<2>(P17), sf_d<3>(M15)
entity interps2_lcd is
--entity ucontrol is
-- Declaracion de Puertos IN/OUT
Port (ps2_data,ps2_clk
: in STD_LOGIC; --DATOS TECLADO -----
reset,clk
: in std_logic; --reset general lcd y clk general
sf_d
: out std_logic_vector(3 downto 0); --lcd pines de datos
lcd_e, lcd_rs, lcd_rw, sf_ce0 : out std_logic); --lcd pines de control
end interps2_lcd;
--
end entity ucontrol;
architecture Behavioral of interps2_lcd is
--architecture Behavioral of ucontrol is
signal codigo: STD_LOGIC_VECTOR (7 downto 0); --NOS AYUDA A IDENTIFICAR LAS LETRAS DEL
TECLADO
signal w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11,w12,w13,w14,w15,w16,w17,w18,w19,
w20,w21,w22,w23,w24,w25,w26,w27,w28,w29,w30,w31,w32: std_logic_vector(7 downto 0);

```

```

signal
    s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15,s16,s17,s18,s19,s20,s21,s22,
    s23,s24,s25,s26,s27,s28,s29,s30,s31,s32: std_logic_vector(7 downto 0);
signal temp_1, temp_2, temp_3: std_logic_vector (1 downto 0) := "00";
signal a: std_logic;
signal busy,puls: std_logic;
signal b: std_logic_vector(3 downto 0);
signal d: std_logic_vector(3 downto 0);
signal t: std_logic:=0';
signal reseteo: std_logic:=0';
signal segundos: integer:=0;
signal x1,x2,x3,x4,x5: bit;
signal i: integer range 0 to 50000000:=0;
type estados is (d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12,d13,d14,d15,d16,d17,d18,d19,d20,
d21,d22,d23,d24,d25,d26,d27,d28,d29,d30,d31,d32,d33,d34,d35,d36,d37,d38,
d39,d40,d41,d42,d43,d44,d45,d46,d47,d48,d49,d50,d51,d52,d53,d54,d55,d56, d57,d58,d59,d60,d61,d62,d63);
signal edo_presente, edo_futuro:estados:= d0;
begin -- Empiezan los port map de cada driver
    -- Driver del teclado
u1:teclado port map (
    reseteo => reseteo,
    clk => clk,
    rst => reset,
    ps2_clk => ps2_clk,
    ps2_data => ps2_data,
    scancode => codigo,
    parity => OPEN,
    busy => busy,
    rdy => puls,
    error => OPEN);
    -- Driver de la LCD
u2: lcd_opciones port map ( clk =>clk, reset => reset,
w1 => w1, -- primera posicion primera linea
w2 => w2, w3 => w3, w4 => w4, w5 => w5, w6 => w6, w7 => w7, w8 => w8, w9 => w9, w10=> w10, w11=> w11,
w12=> w12, w13=> w13, w14=> w14, w15=> w15, w16=> w16, w17=> w17, w18=> w18, w19=> w19, w20=> w20,
w21=> w21, w22=> w22, w23=> 23,
w24=> w24, w25=> w25, w26=> w26, w27=> w27, w28=> w28, w29=> w29, w30=> w30, w31=> w31, w32=> w32,
    -- ultima posicion segunda linea

sf_d => sf_d,
num => "0000",
lcd_e => lcd_e,
lcd_rs => lcd_rs,
lcd_rw => lcd_rw,
sf_ce0 => sf_ce0);

    -- escribimos en la LCD
w1 <=s1; -- w2 <=s2; -- w3 <=s3; -- w4 <=s4; -- w5 <=s5; -- w6 <=s6; -- w7 <=s7; -- w8 <=s8; -- w9 <=s9; --
w10<=s10;-- w11<=s11;--
w12<=s12; -- w13<=s13;-- w14<=s14;-- w15<=s15;-- w16<=s16;-- w17<=s17;-- w18<=s18;-- w19<=s19;-- w20<=s20;--
- w21<=s21;--
w22<=s22; -- w23<=s23;-- w24<=s24;-- w25<=s25;-- w26<=s26;-- w27<=s27;-- w28<=s28;-- w29<=s29;-- w30<=s30;--
- w31<=s31;--
w32<=s32;--

    -- Diseño MEF que controla el teclado y LCD - Proceso 1 Combinacional
proceso1: proceso(edo_presente,codigo) -- lista sensitiva de estado_presente y scancode de teclado
variable key_press: std_logic_vector (7 downto 0);

begin
case codigo is
when x"1C" =>key_press:="01000001";--a 00011100

```

```

when x"32" =>key_press:="01000010";--b 00110010
when x"21" =>key_press:="01000011";--c 00100001
when x"23" =>key_press:="01000100";--d 00100011
when x"24" =>key_press:="01000101";--e 00100100
when x"2B" =>key_press:="01000110";--f 00101011
when x"34" =>key_press:="01000111";--g 00110100
when x"33" =>key_press:="01001000";--h 00110011
when x"43" =>key_press:="01001001";--i 01000011
when x"3B" =>key_press:="01001010";--j 00111011
when x"42" =>key_press:="01001011";--k 01000010
when x"4B" =>key_press:="01001100";--l 01001011
when x"3A" =>key_press:="01001101";--m 00111010
when x"31" =>key_press:="01001110";--n 00110001
when x"44" =>key_press:="01001111";--o 01000100
when x"4D" =>key_press:="01010000";--p 01001101
when x"15" =>key_press:="01010001";--q 00010101
when x"2D" =>key_press:="01010010";--r 00101101
when x"1B" =>key_press:="01010011";--s 00011011
when x"2C" =>key_press:="01010100";--t 00101100
when x"3C" =>key_press:="01010101";--u 00111100
when x"2A" =>key_press:="01010110";--v 00101010
when x"1D" =>key_press:="01010111";--w 00011101
when x"22" =>key_press:="01011000";--x 00100010
when x"35" =>key_press:="01011001";--y 00110101
when x"1A" =>key_press:="01011010";--z 00011010
when x"29" =>key_press:="10100000";--espacio 00101001
-----
when x"16" =>key_press:="00110001";--1 00010110
when x"1E" =>key_press:="00110010";--2 00011110
when x"26" =>key_press:="00110011";--3 00100110
when x"25" =>key_press:="00110100";--4 00100101
when x"2E" =>key_press:="00110101";--5 00101110
when x"36" =>key_press:="00110110";--6 00110110
when x"3D" =>key_press:="00110111";--7 00111101
when x"3E" =>key_press:="00111000";--8 00111110
when x"46" =>key_press:="00111001";--9 01000110
when x"45" =>key_press:="00110000";--0 01000101
when x"66" =>key_press:="10100000";--retroceder 01100110
-----
when x"69" =>key_press:="00110001";--1 00010110
when x"72" =>key_press:="00110010";--2 00011110
when x"7A" =>key_press:="00110011";--3 00100110
when x"6B" =>key_press:="00110100";--4 00100101
when x"73" =>key_press:="00110101";--5 00101110
when x"74" =>key_press:="00110110";--6 00110110
when x"6C" =>key_press:="00110111";--7 00111101
when x"75" =>key_press:="00111000";--8 00111110
when x"7D" =>key_press:="00111001";--9 01000110
when x"70" =>key_press:="00110000";--0 01000101
-----
when others=> key_press:="00000000";
end case;

```

case edo_presente is when d0=>

```

if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codigo=x"34"
)or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codigo=x"31"
)or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(codigo=x"3
C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(codigo=x"
16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(codigo=x
"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(codigo=x
"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))

```

```

        and (puls='1')) then s1<=key_press;
            edo_futuro<=d1;
        else edo_futuro<=d0;
        end if;

when d1=>reseteo<='1';
    edo_futuro<=d2;

when d2=> reseteo<='0';

if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codigo=x"34"
)or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codigo=x"31"
)or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(codigo=x"3
C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(codigo=x"
16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(codigo=x
"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(codigo=x
"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))
    and (puls='1')) then s2<=key_press;
        edo_futuro<=d3;
    else edo_futuro<=d2;
    end if;
when d3=> reseteo<='1';
    edo_futuro<=d4;

when d4=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod
igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(co
digo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))
        and (puls='1')) then
            s3<=key_press;
            edo_futuro<=d5;
        else edo_futuro<=d4;
        end if;

when d5=> reseteo<='1';
    edo_futuro<=d6;

when d6=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod
igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(co
digo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))
        and (puls='1')) then
            s4<=key_press; edo_futuro<=d7;
        else edo_futuro<=d6;
        end if;

when d7=> reseteo<='1';
    edo_futuro<=d8;

when d8=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod

```



```

igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(cod
igo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))

        and (puls='1')) then
            s5<=key_press; edo_futuro<=d9;
        else edo_futuro<=d8;
    end if;

when d9=> reseteo<='1';
    edo_futuro<=d10;

when d10=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod
igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(co
digo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))

        and (puls='1')) then
            s6<=key_press; edo_futuro<=d11;
        else edo_futuro<=d10;
    end if;

when d11=> reseteo<='1';
    edo_futuro<=d12;

when d12=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod
igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(co
digo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))

        and (puls='1')) then
            s7<=key_press; edo_futuro<=d13;
        else edo_futuro<=d12;
    end if;

when d13=> reseteo<='1';
    edo_futuro<=d14;

when d14=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod
igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(co
digo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))

        and (puls='1')) then
            s8<=key_press; edo_futuro<=d15;
        else edo_futuro<=d14;
    end if;

when d15=> reseteo<='1';
    edo_futuro<=d16;

```



```

        s16<=key_press; edo_futuro<=d31;
    else edo_futuro<=d30;
    end if;

when d31=> reseteo<='1';
    edo_futuro<=d32;

when d32=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod
igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(co
digo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))

        and (puls='1')) then
            s17<=key_press; edo_futuro<=d33;
        else edo_futuro<=d32;
        end if;

when d33=> reseteo<='1';
    edo_futuro<=d34;

when d34=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod
igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(co
digo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))

        and (puls='1')) then
            s18<=key_press; edo_futuro<=d35;
        else edo_futuro<=d34;
        end if;

when d35=> reseteo<='1';
    edo_futuro<=d36;

when d36=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod
igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(co
digo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))
        and (puls='1')) then
            s19<=key_press; edo_futuro<=d37;
        else edo_futuro<=d36;
        end if;

when d37=> reseteo<='1';
    edo_futuro<=d38;

when d38=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod

```

```

igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(cod
digo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))
    and (puls=1')) then
        s20<=key_press; edo_futuro<=d39;
    else edo_futuro<=d38;
    end if;

when d39=> reseteo<='1'; edo_futuro<=d40;

when d40=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod
igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(co
digo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))
        and (puls=1')) then
            s21<=key_press; edo_futuro<=d41;
        else edo_futuro<=d40;
        end if;

when d41=> reseteo<='1';
    edo_futuro<=d42;

when d42=> reseteo<='0';

    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod
igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(co
digo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))
        and (puls=1')) then
            s22<=key_press; edo_futuro<=d43;
        else edo_futuro<=d42;
        end if;

when d43=> reseteo<='1';
    edo_futuro<=d44;

when d44=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod
igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(co
digo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))
        and (puls=1')) then
            s23<=key_press; edo_futuro<=d45;
        else edo_futuro<=d44;
        end if;

when d45=> reseteo<='1'; edo_futuro<=d46;

when d46=> reseteo<='0';

```

```

        if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod
igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(co
digo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))

            and (puls='1')) then
                s24<=key_press; edo_futuro<=d47;
            else edo_futuro<=d46;
            end if;

when d47=> reseteo<='1';
    edo_futuro<=d48;

when d48=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codigo=x"34"
)or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codigo=x"31"
)or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(codigo=x"3
C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(codigo=x"
16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(codigo=x
"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(codigo=x
"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))
        and (puls='1')) then
            s25<=key_press; edo_futuro<=d49;
        else edo_futuro<=d48;
        end if;

when d49=> reseteo<='1';
    edo_futuro<=d50;

when d50=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod
igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(co
digo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))
        and (puls='1')) then
            s26<=key_press; edo_futuro<=d51;
        else edo_futuro<=d50;
        end if;

when d51=> reseteo<='1';
    edo_futuro<=d52;

when d52=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod
igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(co
digo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))
        and (puls='1')) then
            s27<=key_press; edo_futuro<=d53;
        else edo_futuro<=d52;
        end if;

```

```

when d53=> reseteo<='1';
    edo_futuro<=d54;

when d54=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod
igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(co
digo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))
    and (puls='1')) then
        s28<=key_press; edo_futuro<=d55;
    else edo_futuro<=d54;
    end if;

when d55=> reseteo<='1';
    edo_futuro<=d56;

when d56=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod
igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(co
digo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))
    and (puls='1')) then
        s29<=key_press; edo_futuro<=d57;
    else edo_futuro<=d56;
    end if;

when d57=> reseteo<='1';
    edo_futuro<=d58;

when d58=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod
igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(co
digo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))
    and (puls='1')) then
        s30<=key_press; edo_futuro<=d59;
    else edo_futuro<=d58;
    end if;

when d59=> reseteo<='1';
    edo_futuro<=d60;

when d60=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codigo=x"34"
)or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codigo=x"31"
)or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(codigo=x"3
C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(codigo=x"
16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(codigo=x
"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(codigo=x
"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))
    and (puls='1')) then

```

```

        s31<=key_press; edo_futuro<=d61;
    else edo_futuro<=d60;
    end if;

when d61=> reseteo<='1';
    edo_futuro<=d62;

when d62=> reseteo<='0';
    if(((codigo=x"1C")or(codigo=x"32")or(codigo=x"21")or(codigo=x"23")or(codigo=x"24")or(codigo=x"2B")or(codi
go=x"34")or(codigo=x"33")or(codigo=x"43")or(codigo=x"3B")or(codigo=x"42")or(codigo=x"4B")or(codigo=x"3A")or(codi
go=x"31")or(codigo=x"44")or(codigo=x"4D")or(codigo=x"15")or(codigo=x"2D")or(codigo=x"1B")or(codigo=x"2C")or(cod
igo=x"3C")or(codigo=x"2A")or(codigo=x"1D")or(codigo=x"22")or(codigo=x"35")or(codigo=x"1A")or(codigo=x"29")or(co
odigo=x"16")or(codigo=x"1E")or(codigo=x"26")or(codigo=x"25")or(codigo=x"2E")or(codigo=x"36")or(codigo=x"3D")or(c
odigo=x"3E")or(codigo=x"46")or(codigo=x"45")or(codigo=x"66")or(codigo=x"69")or(codigo=x"72")or(codigo=x"7A")or(c
odigo=x"6B")or(codigo=x"73")or(codigo=x"74")or(codigo=x"6C")or(codigo=x"75")or(codigo=x"7D")or(codigo=x"70"))
    and (puls='1')) then
        s32<=key_press; edo_futuro<=d63;
    else edo_futuro<=d62;
    end if;

when d63=> reseteo<='1';
    edo_futuro<=d0;

end case;
end process proceso1;
-- Proceso 2 Secuencial
proceso2: process(clk,reset) -- Lista sensitiva por reloj y reset de la memoria de la MEF
begin
    if reset ='1' then
        edo_presente <= d0;
        elsif(clk'event and clk='1')
        then edo_presente<=edo_futuro;
        else null;
    end if;
end process proceso2;
end Behavioral;

```

Archivo .UCF

```

# PlanAhead Generated physical constraints
NET "clk" LOC = C9;
NET "LCD_E" LOC = M18;
NET "LCD_RS" LOC = L18;
NET "LCD_RW" LOC = L17;
NET "ps2_clk" LOC = G14;
NET "ps2_data" LOC = G13;
NET "reset" LOC = K17 | PULLDOWN;
NET "SF_CE0" LOC = D16;
NET "SF_D[0]" LOC = R15;
NET "SF_D[1]" LOC = R16;
NET "SF_D[2]" LOC = P17;
NET "SF_D[3]" LOC = M15;

```

Anexo IV.- Código en VHDL para el manejo del Ratón-PS/2-VGA

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- simulation library
library UNISIM;

```



```

use UNISIM.VComponents.all;
-- the mouse_displayer entity declaration
-- read above for behavioral description and port definitions.
entity mouse_displayer is
port (
    clk      : in std_logic;
    pixel_clk: in std_logic;
    xpos     : in std_logic_vector(9 downto 0);
    ypos     : in std_logic_vector(9 downto 0);
    hcount   : in std_logic_vector(10 downto 0);
    vcount   : in std_logic_vector(10 downto 0);
    blank    : in std_logic;
    red_in   : in std_logic_vector(3 downto 0);
    green_in : in std_logic_vector(3 downto 0);
    blue_in  : in std_logic_vector(3 downto 0);
    red_out  : out std_logic_vector(3 downto 0);
    green_out: out std_logic_vector(3 downto 0);
    blue_out : out std_logic_vector(3 downto 0)
);
-- force synthesizer to extract distributed ram for the
-- displayrom signal, and not a block ram, because the block ram
-- is entirely used to store the image.
attribute rom_extract : string;
attribute rom_extract of mouse_displayer: entity is "yes";
attribute rom_style : string;
attribute rom_style of mouse_displayer: entity is "distributed";
end mouse_displayer;
architecture Behavioral of mouse_displayer is
-----
-- CONSTANTS
-----
type displayrom is array(0 to 255) of std_logic_vector(1 downto 0);
-- the memory that holds the cursor.
-- 00 - black
-- 01 - white
-- 1x - transparent
constant mouserom: displayrom := (
"00","00","11","11","11","11","11","11","11","11","11","11","11","11","11",
"00","01","00","11","11","11","11","11","11","11","11","11","11","11","11",
"00","01","01","00","11","11","11","11","11","11","11","11","11","11","11",
"00","01","01","01","00","11","11","11","11","11","11","11","11","11","11",
"00","01","01","01","01","00","11","11","11","11","11","11","11","11","11",
"00","01","01","01","01","01","00","11","11","11","11","11","11","11","11",
"00","01","01","01","01","01","01","00","11","11","11","11","11","11","11",
"00","01","01","01","01","01","01","01","00","11","11","11","11","11","11",
"00","01","01","01","01","01","00","00","00","11","11","11","11","11","11",
"00","01","01","01","01","01","00","11","11","11","11","11","11","11","11",
"00","00","11","11","00","01","01","00","11","11","11","11","11","11","11",
"00","11","11","11","00","01","01","00","11","11","11","11","11","11","11",
"11","11","11","11","11","00","01","00","11","11","11","11","11","11","11",
"11","11","11","11","11","11","00","00","11","11","11","11","11","11","11"
);
-- width and height of cursor.
constant OFFSET: std_logic_vector(4 downto 0) := "10000"; -- 16
-----
-- SIGNALS
-----
--signal rompos: std_logic_vector(7 downto 0) := (others => '0');
-- pixel from the display memory, representing currently displayed

```

```

-- pixel of the cursor, if the cursor is being display at this point
signal mousepixel: std_logic_vector(1 downto 0) := (others => '0');
-- when high, enables displaying of the cursor, and reading the
-- cursor memory.
signal enable_mouse_display: std_logic := '0';
-- difference in range 0-15 between the vga counters and mouse position
signal xdiff: std_logic_vector(3 downto 0) := (others => '0');
signal ydiff: std_logic_vector(3 downto 0) := (others => '0');
begin
  -- compute xdiff
  x_diff: process(clk)
    variable temp_diff: std_logic_vector(10 downto 0) := (others => '0');
  begin
    if(rising_edge(clk)) then
      temp_diff := hcount - ('0' & xpos);
      xdiff <= temp_diff(3 downto 0);
    end if;
  end process x_diff;

  -- compute ydiff
  y_diff: process(clk)
    variable temp_diff: std_logic_vector(10 downto 0) := (others => '0');
  begin
    if(rising_edge(clk)) then
      temp_diff := vcount - ('0' & ypos);
      ydiff <= temp_diff(3 downto 0);
    end if;
  end process y_diff;
  -- rompos <= (ydiff & xdiff) when rising_edge(clk);
  -- read pixel from memory at address obtained by concatenation of
  -- ydiff and xdiff
  mousepixel <= mouserom(conv_integer(ydiff & xdiff))
    when rising_edge(pixel_clk);
  -- set enable_mouse_display high if vga counters inside cursor block
  enable_mouse: process(pixel_clk)
  begin
    if(rising_edge(pixel_clk)) then
      if(hcount >= xpos and hcount < (xpos + OFFSET) and
        vcount >= ypos and vcount < (ypos + OFFSET))
      then
        enable_mouse_display <= '1';
      else
        enable_mouse_display <= '0';
      end if;
    end if;
  end process enable_mouse;
  -- if cursor display is enabled, then, according to pixel
  -- value, set the output color channels.
  process(pixel_clk)
  begin
    if(rising_edge(pixel_clk)) then
      -- if in visible screen
      if(blank = '0') then
        -- in display is enabled
        if(enable_mouse_display = '1') then
          -- white pixel of cursor
          if(mousepixel = "01") then
            red_out <= (others => '1');
            green_out <= (others => '1');
            blue_out <= (others => '1');
          -- black pixel of cursor

```

```

        elsif(mousepixel = "00") then
            red_out <= (others => '0');
            green_out <= (others => '0');
            blue_out <= (others => '0');
            -- transparent pixel of cursor
            -- let input pass to output
        else
            red_out <= red_in;
            green_out <= green_in;
            blue_out <= blue_in;
        end if;
        -- cursor display is not enabled
        -- let input pass to output.
    else
        red_out <= red_in;
        green_out <= green_in;
        blue_out <= blue_in;
    end if;
    -- not in visible screen, black outputs.
else
    red_out <= (others => '0');
    green_out <= (others => '0');
    blue_out <= (others => '0');
end if;
end if;
end process;
end Behavioral;

```

Archivo .UCF

```

# PlanAhead Generated physical constraints
NET "B_left" LOC = F9;
NET "B_medio" LOC = E9;
NET "B_rigth" LOC = D11;
NET "Horiz" LOC = F15;
NET "Vert" LOC = F14;
NET "azul[0]" LOC = G15;
NET "azul_in[0]" LOC = L13;
NET "clock" LOC = C9;
NET "new_event" LOC = F12;
NET "ps2_clock" LOC = G14;
NET "ps2_data" LOC = G13;
NET "reset" LOC = K17;
NET "rojo[0]" LOC = H14;
NET "rojo_in[0]" LOC = H18;
NET "verde[0]" LOC = H15;
NET "verde_in[0]" LOC = L14;
# PlanAhead Generated IO constraints
NET "azul_in[3]" PULLDOWN;
NET "azul_in[1]" PULLDOWN;
NET "azul_in[2]" PULLDOWN;
NET "reset" PULLDOWN;
NET "rojo_in[3]" PULLDOWN;
NET "rojo_in[1]" PULLDOWN;
NET "rojo_in[2]" PULLDOWN;
NET "verde_in[3]" PULLDOWN;
NET "verde_in[1]" PULLDOWN;
NET "verde_in[2]" PULLDOWN;

```

Anexo V.- Código en VHDL para el manejo del teclado y LCD

```

library          IEEE;
use              IEEE.STD_LOGIC_1164.ALL;
use              IEEE.STD_LOGIC_ARITH.ALL;
use              IEEE.STD_LOGIC_UNSIGNED.ALL;
use              work.teclao_pcg.all;
use work.lcdpkg.all;

-- Asignacion de Pines : clk(C9), reset(L13), ps2_clk(G14), ps2_data(G13), - lcd_rs(L18), lcd_rw(L17), lcd_e(M18),
sf_ce0(D16),
-- sf_d<0>(R15),sf_d<1>(R16), sf_d<2>(P17), sf_d<3>(M15)
entity interps2_lcd is                                --entity ucontrol is
-- Declaracion de Puertos IN/OUT
Port (ps2_data,ps2_clk                                : in STD_LOGIC; --DATOS TECLADO -----
      reset,clk                                         : in std_logic;    --reset general lcd y clk general
      sf_d                                              : out std_logic_vector(3 downto 0); --lcd pines de datos
      lcd_e, lcd_rs, lcd_rw, sf_ce0 : out std_logic); --lcd pines de control

end interps2_lcd;
end entity ucontrol;

architecture Behavioral of interps2_lcd is
signal codigo: STD_LOGIC_VECTOR (7 downto 0);
DEL TECLADO
signal w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11,w12,w13,w14,w15,w16,w17,w18,w19,w20,w21,w22,w23,w24,w25,
w26,w27,w28,w29,w30,w31,w32: std_logic_vector(7 downto 0);
signal s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15,s16,s17,s18,s19,s20,s21,s22, s23,s24,s25,s26,s27,s28,
s29,s30,s31,s32: std_logic_vector(7 downto 0);
signal temp_1, temp_2, temp_3: std_logic_vector (1 downto 0) := "00";
signal a: std_logic;
signal busy,puls: std_logic;
signal b: std_logic_vector(3 downto 0);
signal d: std_logic_vector(3 downto 0);
signal t: std_logic:=0;
signal reseteo: std_logic:=0;
signal segundos: integer:=0;
signal x1,x2,x3,x4,x5: bit;
signal i: integer range 0 to 50000000:=0;
type estados is (d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12,d13,d14,d15,d16,d17,d18,d19,d20, d21,d22,d23,d24,
d25,d26,d27,d28,d29,d30,d31,d32,d33,d34,d35,d36,d37,d38,d39,d40,d41,d42,d43,d44,d45,d46,d47,d48,d49,d50,
d51, d52, d53,d54, d55,d56, d57,d58,d59,d60,d61,d62,d63);
signal edo_presente, edo_futuro:estados:= d0;
begin -- Empiezan los port map de cada driver
-- Driver del teclado
u1:teclado port map (
    reseteo => reseteo,
    clk => clk,
    rst => reset,
    ps2_clk => ps2_clk,
    ps2_data => ps2_data,
    scancode => codigo,
    parity => OPEN,
    busy => busy,
    rdy => puls,
    error => OPEN);
-- Proceso 2 Secuencial
proceso2: process(clk,reset) -- Lista sensitiva por reloj y reset de la memoria de la MEF
begin
    if reset ='1' then

```

```

        edo_presente <= d0;
        elsif(clk'event and clk='1')
        then edo_presente<=edo_futuro;
        else null;
    end if;
end process proceso2;
end Behavioral;

```

Para el teclado:

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity teclado is
generic(
    FREQ : natural := 100_000 -- frequency of the main clock (KHz)
);
port(
    clk          : in      std_logic;          -- main clock
    rst          : in      std_logic;          -- asynchronous reset
    ps2_clk      : in      std_logic;          -- clock from keyboard
    reseteo      : in      std_logic;
    ps2_data     : in      std_logic;          -- data from keyboard
    scancode     : out std_logic_vector(7 downto 0); -- key scancode
    parity       : out std_logic;              -- parity bit for scancode
    busy         : out std_logic;              -- busy receiving scancode
    rdy          : out std_logic;              -- scancode ready pulse
    error        : out std_logic              -- error receiving scancode
);
end entity teclado;
architecture arch of teclado is
    constant YES      : std_logic := '1';
    constant NO       : std_logic := '0';
    constant PS2_FREQ : natural := 10; -- keyboard clock frequency (KHz)
    constant TIMEOUT  : natural := FREQ / PS2_FREQ; -- ps2_clk quiet timeout
    constant KEY_RELEASE : std_logic_vector(7 downto 0) := "11110000"; -- scancode sent when key is released
    signal timer_x, timer_r : natural range 0 to TIMEOUT; -- counts time since last PS/2 clock edge
    signal bitcnt_x, bitcnt_r : natural range 0 to 11; -- counts number of received scancode bits
    signal ps2_clk_x, ps2_clk_r : std_logic_vector(5 downto 1); -- PS/2 clock synchronization / edge detect shift register
    signal ps2_clk_fall_edge : std_logic; -- pulses on falling edge of PS/2 clock
    signal ps2_clk_rise_edge : std_logic; -- pulses on rising edge of PS/2 clock
    signal ps2_clk_edge : std_logic; -- pulses on either edge of PS/2 clock
    signal ps2_clk_quiet : std_logic; -- pulses when no edges on PS/2 clock for TIMEOUT
    signal sc_x, sc_r : std_logic_vector(9 downto 0); -- scancode shift register
    signal keyrel_x, keyrel_r : std_logic; -- this flag is set when the key release scancode is received
    signal scancode_rdy : std_logic; -- indicates when any scancode has been received
    signal rdy_x, rdy_r : std_logic; -- this flag is set when scancode for the pressed key is ready
    signal error_x, error_r : std_logic; -- this flag is set when an error occurs
begin

```

```

-- shift the level on the PS/2 clock into a shift register
ps2_clk_x <= ps2_clk_r(4 downto 1) & ps2_clk;

-- look at the PS/2 clock levels stored in the shift register and find rising or falling edges
ps2_clk_fall_edge <= YES when ps2_clk_r(5 downto 2) = "1100" else NO;
ps2_clk_rise_edge <= YES when ps2_clk_r(5 downto 2) = "0011" else NO;
ps2_clk_edge <= ps2_clk_fall_edge or ps2_clk_rise_edge;

-- shift the keyboard scancode into the shift register on the falling edge of the PS/2 clock
sc_x <= ps2_data & sc_r(9 downto 1) when ps2_clk_fall_edge = YES else sc_r;

-- clear the timer right after a PS/2 clock edge and then keep incrementing it until the next edge
timer_x <= 0 when ps2_clk_edge = YES else timer_r + 1;

-- indicate when the PS/2 clock has stopped pulsing and is at a high level.
ps2_clk_quiet <= YES when timer_r = TIMEOUT and ps2_clk_r(2) = '1' else NO;

-- increment the bit counter on each falling edge of the PS/2 clock.

-- reset the bit counter if the PS/2 clock stops pulsing or if there was an error receiving the scancode.

-- otherwise, keep the bit counter unchanged.
bitcnt_x <= bitcnt_r + 1 when ps2_clk_fall_edge = YES else
0 when ps2_clk_quiet = YES or error_r = YES else
bitcnt_r;

-- a scancode has been received if the bit counter is 11 and the PS/2 clock has stopped pulsing
scancode_rdy <= YES when bitcnt_r = 11 and ps2_clk_quiet = YES else NO;

-- look for the scancode sent when the key is released
keyrel_x <= YES when sc_r(scancode'range) = KEY_RELEASE and scancode_rdy = YES else
NO when rdy_r = YES or error_r = YES else
keyrel_r;

-- the scancode for the pressed key arrives after receiving the key-release scancode
rdy_x <= YES when keyrel_r = YES and scancode_rdy = YES else NO;

-- indicate an error if the clock is low for too long or if it stops
pulsing in the middle of a scancode
error_x <= YES when (timer_r = TIMEOUT and ps2_clk_r(2) = '0') or
(ps2_clk_quiet = YES and bitcnt_r/=11 and bitcnt_r/=0) else
error_r;
scancode <= sc_r(scancode'range);
output scancode
parity <= sc_r(scancode'high+1);
parity bit for the scancode
busy <= YES when bitcnt_r/=0 else NO;
busy signal when receiving a scancode
rdy <= rdy_r;

-- output scancode ready flag
error <= error_r;
-- output error flag
-- update the various registers
process(rst, clk, reseteo)
begin
    if rst = YES or reseteo = '1' then
        ps2_clk_r <= (others => '1');
    start by assuming PS/2 clock has been high for a while

```

```

        sc_r <= (others => '0');
-- clear scancode register
        keyrel_r <= NO;
        -- key-release scancode has not been received yet
        rdy_r <= NO;
        -- no scancodes received yet
        timer_r <= 0;
        -- clear PS/2 clock pulse timer
        bitcnt_r <= 0;
        -- clear scancode bit counter
        error_r <= NO;
        -- clear any errors
    elsif rising_edge(clk) then
        ps2_clk_r <= ps2_clk_x;
        sc_r <= sc_x;
        keyrel_r <= keyrel_x;
        rdy_r <= rdy_x;
        timer_r <= timer_x;
        bitcnt_r <= bitcnt_x;
        error_r <= error_x;
    end if;
end process;

end architecture arch;

```

Para el *Display* de Cristal Líquido

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity lcd_opciones is
port(clk, reset : in std_logic;
w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11,w12,w13,w14,w15,w16,w17,w18,w19,w20,w21,w22,w23,w24,
w25,w26,w27,w28,w29,w30,w31,w32: in std_logic_vector(7 downto 0);
SF_D : out std_logic_vector(3 downto 0);
num : in std_logic_vector(3 downto 0);
LCD_E, LCD_RS, LCD_RW, SF_CE0 : out std_logic);
end lcd_opciones;

architecture behavior of lcd_opciones is
type tx_sequence is (high_setup, high_hold, oneus, low_setup, low_hold, fortyus, done);
signal tx_state : tx_sequence := done;
signal tx_byte : std_logic_vector(7 downto 0);
signal tx_init : std_logic := '0';
type init_sequence is (idle, fifteenms, one, two, three, four, five, six, seven, eight, done);
signal init_state : init_sequence := idle;
signal init_init, init_done : std_logic := '0';
signal i : integer range 0 to 750000 := 0;
signal i2 : integer range 0 to 2000 := 0;
signal i3 : integer range 0 to 82000 := 0;
signal SF_D0, SF_D1 : std_logic_vector(3 downto 0);
signal LCD_E0, LCD_E1 : std_logic;
signal mux : std_logic;

type display_state is (init, function_set, entry_set, set_display, clr_display, pause, set_addr, set_addr_sl,
char_1,char_2,char_3,char_4,char_5,char_6,char_7,char_8,char_9,char_10,char_11,char_12,char_13,char_14,
char_15, char_16,char_17,char_18, char_19, char_20, char_21, char_22, char_23,char_24, char_25,char_26,
char_27, char_28,char_29, char_30, char_31,char_32, done);
signal cur_state : display_state := init;

```

```

begin
SF_CEO <= '1'; --disable intel strataflash
LCD_RW <= '0'; --write only
--The following "with" statements simplify the process of adding and removing states.
--when to transmit a command/data and when not to
with cur_state select
tx_init <= '0' when init | pause | done,
'1' when others;
--control the bus
with cur_state select
mux <= '1' when init,
'0' when others;
--control the initialization sequence
with cur_state select
init_init <= '1' when init,
'0' when others;
--register select
with cur_state select
LCD_RS <= '0' when function_set|entry_set|set_display|clr_display|set_addr|set_addr_sl,
'1' when others;
--what byte to transmit to lcd
--refer to datasheet for an explanation of these values
with cur_state select
tx_byte <= "00101000" when function_set,
"00000110" when entry_set,
"00001100" when set_display, --0X0C
"00000001" when clr_display, --0X01
"10000000" when set_addr, --0X80 First Line
"11000000" when set_addr_sl, --0XC0 Second line
w1 when char_1, w2 when char_2, w3 when char_3, w4 when char_4, w5 when char_5, w6 when char_6,
w7 when char_7, w8 when char_8, w9 when char_9, w10 when char_10, w11 when char_11, w12 when char_12,
w13 when char_13, w14 when char_14, w15 when char_15, w16 when char_16, w17 when char_17, w18 when char_18,
w19 when char_19, w20 when char_20, w21 when char_21, w22 when char_22, w23 when char_23, w24 when char_24,
w25 when char_25, w26 when char_26, w27 when char_27, w28 when char_28, w29 when char_29, w30 when char_30,
w31 when char_31, w32 when char_32,

"00000000" when others;
--main state machine
display: process(clk, reset)
begin
if(reset='1') then
cur_state <= function_set;
elsif(clk='1' and clk'event) then
case cur_state is
--refer to initialize state machine below
when init =>
if(init_done = '1') then
cur_state <= function_set;
else
cur_state <= init;
end if;
--every other state but pause uses the transmit state machine
when function_set =>
if(i2 = 2000) then
cur_state <= entry_set;
else
cur_state <= function_set;
end if;
when entry_set =>
if(i2 = 2000) then
cur_state <= set_display;

```



```

else
cur_state <= entry_set;
end if;
when set_display =>
if(i2 = 2000) then
cur_state <= clr_display;
else
cur_state <= set_display;
end if;
when clr_display =>
i3 <= 0;
if(i2 = 2000) then
cur_state <= pause;
else
cur_state <= clr_display;
end if;
when pause =>
if(i3 = 82000) then
cur_state <= set_addr;
i3 <= 0;
else
cur_state <= pause;
i3 <= i3 + 1;
end if;
when set_addr =>
if(i2 = 2000) then
cur_state <= char_1;
else
cur_state <= set_addr;
end if;

```

--- AQUÍ VAN LOS CARACTERES

```

when char_1 =>
if(i2 = 2000) then
cur_state <= char_2;
else
cur_state <= char_1;
end if;
when char_2 =>
if(i2 = 2000) then
cur_state <= char_3;
else
cur_state <= char_2;
end if;
when char_3 =>
if(i2 = 2000) then
cur_state <= char_4;
else
cur_state <= char_3;
end if;
when char_4 =>
if(i2 = 2000) then
cur_state <= char_5;
else
cur_state <= char_4;
end if;
when char_5 =>
if(i2 = 2000) then
cur_state <= char_6;
else
cur_state <= char_5;

```

```

end if;
when char_6 =>
if(i2 = 2000) then
cur_state <= char_7;
else
cur_state <= char_6;
end if;
when char_7 =>
if(i2 = 2000) then
cur_state <= char_8;
else
cur_state <= char_7;
end if;
when char_8 =>
if(i2 = 2000) then
cur_state <= char_9;
else
cur_state <= char_8;
end if;
when char_9 =>
if(i2 = 2000) then
cur_state <= char_10;
else
cur_state <= char_9;
end if;
when char_10 =>
if(i2 = 2000) then
cur_state <= char_11;
else
cur_state <= char_10;
end if;
when char_11 =>
if(i2 = 2000) then
cur_state <= char_12;
else
cur_state <= char_11;
end if;
when char_12 =>
if(i2 = 2000) then
cur_state <= char_13;
else
cur_state <= char_12;
end if;
when char_13 =>
if(i2 = 2000) then
cur_state <= char_14;
else
cur_state <= char_13;
end if;
when char_14 =>
if(i2 = 2000) then
cur_state <= char_15;
else
cur_state <= char_14;
end if;
when char_15 =>
if(i2 = 2000) then
cur_state <= char_16;
else
cur_state <= char_15;
end if;

```

```

when char_16 =>
if(i2 = 2000) then
cur_state <= set_addr_sl;
else
cur_state <= char_16;
end if;
--Segunda linea
when set_addr_sl =>
if(i2 = 2000) then
cur_state <= char_17;
else
cur_state <= set_addr_sl;
end if;
when char_17 =>
if(i2 = 2000) then
cur_state <= char_18;
else
cur_state <= char_17;
end if;
when char_18 =>
if(i2 = 2000) then
cur_state <= char_19;
else
cur_state <= char_18;
end if;
when char_19 =>
if(i2 = 2000) then
cur_state <= char_20;
else
cur_state <= char_19;
end if;
when char_20 =>
if(i2 = 2000) then
cur_state <= char_21;
else
cur_state <= char_20;
end if;
when char_21 =>
if(i2 = 2000) then
cur_state <= char_22;
else
cur_state <= char_21;
end if;
when char_22 =>
if(i2 = 2000) then
cur_state <= char_23;
else
cur_state <= char_22;
end if;
when char_23 =>
if(i2 = 2000) then
cur_state <= char_24;
else
cur_state <= char_23;
end if;
when char_24 =>
if(i2 = 2000) then
cur_state <= char_25;
else
cur_state <= char_24;
end if;

```

```

when char_25 =>
if(i2 = 2000) then
cur_state <= char_26;
else
cur_state <= char_25;
end if;
when char_26 =>
if(i2 = 2000) then
cur_state <= char_27;
else
cur_state <= char_26;
end if;
when char_27 =>
if(i2 = 2000) then
cur_state <= char_28;
else
cur_state <= char_27;
end if;
when char_28 =>
if(i2 = 2000) then
cur_state <= char_29;
else
cur_state <= char_28;
end if;
when char_29 =>
if(i2 = 2000) then
cur_state <= char_30;
else
cur_state <= char_29;
end if;
when char_30 =>
if(i2 = 2000) then
cur_state <= char_31;
else
cur_state <= char_30;
end if;
when char_31 =>
if(i2 = 2000) then
cur_state <= char_32;
else
cur_state <= char_31;
end if;
when char_32 =>
if(i2 = 2000) then
cur_state <= set_addr;
else
cur_state <= char_32;
end if;
-----
when others =>
cur_state <= done;
end case;
end if;
end process display;
-----Termina Maq. ESTADOS

with mux select
SF_D <= SF_D0 when '0', --transmit
SF_D1 when others; --initialize
with mux select
LCD_E <= LCD_E0 when '0', --transmit
LCD_E1 when others; --initialize

```

```

--specified by datasheet
transmit : process(clk, reset, tx_init)
begin
if(reset='1') then
tx_state <= done;
elsif(clk='1' and clk'event) then
case tx_state is
when high_setup => --40ns
LCD_E0 <= '0';
SF_D0 <= tx_byte(7 downto 4);
if(i2 = 2) then
tx_state <= high_hold;
i2 <= 0;
else
tx_state <= high_setup;
i2 <= i2 + 1;
end if;
when high_hold => --230ns
LCD_E0 <= '1';
SF_D0 <= tx_byte(7 downto 4);
if(i2 = 12) then
tx_state <= oneus;
i2 <= 0;
else
tx_state <= high_hold;
i2 <= i2 + 1;
end if;
when oneus =>
LCD_E0 <= '0';
if(i2 = 50) then
tx_state <= low_setup;
i2 <= 0;
else
tx_state <= oneus;
i2 <= i2 + 1;
end if;
when low_setup =>
LCD_E0 <= '0';
SF_D0 <= tx_byte(3 downto 0);
if(i2 = 2) then
tx_state <= low_hold;
i2 <= 0;
else
tx_state <= low_setup;
i2 <= i2 + 1;
end if;
when low_hold =>
LCD_E0 <= '1';
SF_D0 <= tx_byte(3 downto 0);
if(i2 = 12) then
tx_state <= fortyus;
i2 <= 0;
else
tx_state <= low_hold;
i2 <= i2 + 1;
end if;
when fortyus =>
LCD_E0 <= '0';
if(i2 = 2000) then
tx_state <= done;
i2 <= 0;

```

```

else
tx_state <= fortyus;
i2 <= i2 + 1;
end if;
when done =>
LCD_E0 <= '0';
if(tx_init = '1') then
tx_state <= high_setup;
i2 <= 0;
else
tx_state <= done;
i2 <= 0;
end if;
end case;
end if;
end process transmit;
-----
--specified by datasheet
power_on_initialize: process(clk, reset, init_init) --power on initialization sequence
begin
if(reset='1') then
init_state <= idle;
init_done <= '0';
elsif(clk='1' and clk'event) then
case init_state is
when idle =>
init_done <= '0';
if(init_init = '1') then
init_state <= fifteenms;
i <= 0;
else
init_state <= idle;
i <= i + 1;
end if;
when fifteenms =>
init_done <= '0';
if(i = 750000) then
init_state <= one;
i <= 0;
else
init_state <= fifteenms;
i <= i + 1;
end if;
when one =>
SF_D1 <= "0011";
LCD_E1 <= '1';
init_done <= '0';
if(i = 11) then
init_state<=two;
i <= 0;
else
init_state<=one;
i <= i + 1;
end if;
when two =>
LCD_E1 <= '0';
init_done <= '0';
if(i = 205000) then
init_state<=three;
i <= 0;
else

```

```

init_state<=two;
i <= i + 1;
end if;
when three =>
SF_D1 <= "0011";
LCD_E1 <= '1';
init_done <= '0';
if(i = 11) then
init_state<=four;
i <= 0;
else
init_state<=three;
i <= i + 1;
end if;
when four =>
LCD_E1 <= '0';
init_done <= '0';
if(i = 5000) then
init_state<=five;
i <= 0;
else
init_state<=four;
i <= i + 1;
end if;
when five =>
SF_D1 <= "0011";
LCD_E1 <= '1';
init_done <= '0';
if(i = 11) then
init_state<=six;
i <= 0;
else
init_state<=five;
i <= i + 1;
end if;
when six =>
LCD_E1 <= '0';
init_done <= '0';
if(i = 2000) then
init_state<=seven;
i <= 0;
else
init_state<=six;
i <= i + 1;
end if;
when seven =>
SF_D1 <= "0010";
LCD_E1 <= '1';
init_done <= '0';
if(i = 11) then
init_state<=eight;
i <= 0;
else
init_state<=seven;
i <= i + 1;
end if;
when eight =>
LCD_E1 <= '0';
init_done <= '0';
if(i = 2000) then
init_state<=done;

```

```

i <= 0;
else
init_state<=eight;
i <= i + 1;
end if;
when done =>
init_state <= done;
init_done <= '1';
end case;
end if;
end process power_on_initialize;
end behavior;

```

Anexo VI.- Código en VHDL para el *transceiver*

```

library ieee;
use ieee.std_logic_1164.all, ieee.numeric_std.all;
use work.usb_pkg.all;
entity usb_serial is
  generic (
    -- Vendor ID to report in device descriptor.
    VENDORID : std_logic_vector(15 downto 0);
    -- Product ID to report in device descriptor.
    PRODUCTID : std_logic_vector(15 downto 0);
    -- Product version to report in device descriptor.
    VERSIONBCD : std_logic_vector(15 downto 0);
    -- Optional description of manufacturer (max 126 characters).
    VENDORSTR : string := "";
    -- Optional description of product (max 126 characters).
    PRODUCTSTR : string := "";
    -- Optional product serial number (max 126 characters).
    SERIALSTR : string := "";
    -- Support high speed mode.
    HSSUPPORT : boolean := false;
    -- Set to true if the device never draws power from the USB bus.
    SELFPOWERED : boolean := false;

    -- Size of receive buffer as 2-logarithm of the number of bytes.
    -- Must be at least 10 (1024 bytes) for high speed support.
    RXBUFSIZE_BITS: integer range 7 to 12 := 11;
    -- Size of transmit buffer as 2-logarithm of the number of bytes.
    TXBUFSIZE_BITS: integer range 7 to 12 := 10 );
  port (
    -- 60 MHz UTMI clock.
    CLK : in std_logic;
    -- Synchronous reset; clear buffers and re-attach to the bus.
    RESET : in std_logic;
    -- High for one clock when a reset signal is detected on the USB bus.
    -- Note: do NOT wire this signal to RESET externally.
    USBRST : out std_logic;
    -- High when the device is operating (or suspended) in high speed mode.
    HIGHSPEED : out std_logic;
    -- High while the device is suspended.
    -- Note: This signal is not synchronized to CLK.
    -- It may be used to asynchronously drive the UTMI SuspendM pin.
    SUSPEND : out std_logic;
    -- High when the device is in the Configured state.
    ONLINE : out std_logic;

```



```

-- High if a received byte is available on RXDAT.
RXVAL :    out std_logic;
-- Received data byte, valid if RXVAL is high.
RXDAT :    out std_logic_vector(7 downto 0);
-- High if the application is ready to receive the next byte.
RXRDY :    in  std_logic;
-- Number of bytes currently available in receive buffer.
RXLEN :    out std_logic_vector((RXBUFSIZE_BITS-1) downto 0);
-- High if the application has data to send.
TXVAL :    in  std_logic;
-- Data byte to send, must be valid if TXVAL is high.
TXDAT :    in  std_logic_vector(7 downto 0);
-- High if the entity is ready to accept the next byte.
TXRDY :    out std_logic;
-- Number of free byte positions currently available in transmit buffer.
TXROOM :   out std_logic_vector((TXBUFSIZE_BITS-1) downto 0);
-- Temporarily suppress transmissions at the outgoing endpoint.
-- This gives the application an opportunity to fill the transmit
-- buffer in order to blast data efficiently in big chunks.
TXCORK :   in  std_logic;
PHY_DATAIN : in std_logic_vector(7 downto 0);
  PHY_DATAOUT : out std_logic_vector(7 downto 0);
  PHY_TXVALID : out std_logic;
  PHY_TXREADY : in  std_logic;
  PHY_RXACTIVE : in  std_logic;
  PHY_RXVALID : in  std_logic;
  PHY_RXERROR : in  std_logic;
  PHY_LINESTATE : in std_logic_vector(1 downto 0);
  PHY_OPMODE :   out std_logic_vector(1 downto 0);
PHY_XCVRSELECT: out std_logic;
PHY_TERMSELECT: out std_logic;
  PHY_RESET :   out std_logic );
end entity usb_serial;

architecture usb_serial_arch of usb_serial is
  -- Byte array type
  type t_byte_array is array(natural range <>) of std_logic_vector(7 downto 0);
  -- Conditional expression.
  function choose_int(z: boolean; a, b: integer)
    return integer is
  begin
    if z then return a; else return b; end if;
  end function;
  -- Conditional expression.
  function choose_byte(z: boolean; a, b: std_logic_vector)
    return std_logic_vector is
  begin
    if z then return a; else return b; end if;
  end function;
  -- Create USB string descriptor from VHDL string.
  function strdesc_from_string(s: in string)
    return t_byte_array is
    variable y: t_byte_array(0 to 2 + 2*s'length - 1);
  begin
    assert s'length <= 126;
    -- bLength = descriptor length in bytes
    y(0) := std_logic_vector(to_unsigned(2 + 2 * s'length, 8));
    -- bDescriptorType = string descriptor
    y(1) := X"03";
    -- String contents in UTF16-LE.
    if s'length > 0 then

```

```

    for i in s'range loop
        y(2*(i-s'low)+2) := std_logic_vector(
            to_unsigned(character'pos(s(i)), 8));
        y(2*(i-s'low)+3) := (others => '0');
    end loop;
end if;
return y;
end function;
-- Maximum packet size according to protocol.
constant MAX_FSPACKET_SIZE: integer := 64;
constant MAX_HSPACKET_SIZE: integer := 512;
-- Width required for a pointer that can cover either RX or TX buffer.
constant BUFPTR_SIZE : integer := choose_int(RXBUFSIZE_BITS > TXBUFSIZE_BITS, RXBUFSIZE_BITS,
TXBUFSIZE_BITS);
-- Data endpoint number.
constant data_endpt : std_logic_vector(3 downto 0) := "0001";
constant notify_endpt : std_logic_vector(3 downto 0) := "0010";
-- Descriptor ROM
-- addr 0 .. 17 : device descriptor
-- addr 20 .. 29 : device qualifier
-- addr 32 .. 98 : full speed configuration descriptor
-- addr 112 .. 178 : high speed configuration descriptor
-- addr 179 : other_speed_configuration hack
-- addr 192 .. 195 : string descriptor 0 = supported languages
-- addr 196 .. : 3 string descriptors: vendor, product, serial
constant DESC_DEV_ADDR : integer := 0;
constant DESC_DEV_LEN : integer := 18;
constant DESC_QUAL_ADDR : integer := 20;
constant DESC_QUAL_LEN : integer := 10;
constant DESC_FSCFG_ADDR : integer := 32;
constant DESC_FSCFG_LEN : integer := 67;
constant DESC_HSCFG_ADDR : integer := 112;
constant DESC_HSCFG_LEN : integer := 67;
constant DESC_OTHERSPEED_ADDR : integer := 179;
constant DESC_STRLANG : integer := 192;
constant DESC_STRVENDOR : integer := 196;
constant DESC_STRPRODUCT : integer := DESC_STRVENDOR + 2 + 2*VENDORSTR'length;
constant DESC_STRSERIAL : integer := DESC_STRPRODUCT + 2 + 2*PRODUCTSTR'length;
constant DESC_END : integer := DESC_STRSERIAL + 2 + 2*SERIALSTR'length;

constant descrom_dev: t_byte_array(0 to 19) := (
    -- 18 bytes device descriptor
    X"12", -- bLength = 18 bytes
    X"01", -- bDescriptorType = device descriptor
    choose_byte(HSSUPPORT, X"00", X"10"), -- bcdUSB = 1.10 or 2.00
    choose_byte(HSSUPPORT, X"02", X"01"),
    X"02", -- bDeviceClass = Communication Device Class
    X"00", -- bDeviceSubClass = none
    X"00", -- bDeviceProtocol = none
    X"40", -- bMaxPacketSize0 = 64 bytes
    VENDORID(7 downto 0), -- idVendor
    VENDORID(15 downto 8),
    PRODUCTID(7 downto 0), -- idProduct
    PRODUCTID(15 downto 8),
    VERSIONBCD(7 downto 0), -- bcdDevice
    VERSIONBCD(15 downto 8),
    choose_byte(VENDORSTR'length > 0, X"01", X"00"), -- iManufacturer
    choose_byte(PRODUCTSTR'length > 0, X"02", X"00"), -- iProduct
    choose_byte(SERIALSTR'length > 0, X"03", X"00"), -- iSerialNumber
    X"01", -- bNumConfigurations = 1
    -- 2 bytes padding

```

```

X"00", X"00" );
constant descrom_qual: t_byte_array(0 to 11) := (
-- 10 bytes device qualifier
X"0a",          -- bLength = 10 bytes
X"06",          -- bDescriptorType = device qualifier
X"00", X"02",   -- bcdUSB = 2.0
X"02",          -- bDeviceClass = Communication Device Class
X"00",          -- bDeviceSubClass = none
X"00",          -- bDeviceProtocol = none
X"40",          -- bMaxPacketSize0 = 64 bytes
X"01",          -- bNumConfigurations = 1
X"00",          -- bReserved
-- 2 bytes padding
X"00", X"00" );
constant descrom_fscfg: t_byte_array(0 to 79) := (
-- 67 bytes full-speed configuration descriptor
-- 9 bytes configuration header
X"09",          -- bLength = 9 bytes
X"02",          -- bDescriptorType = configuration descriptor
X"43", X"00",   -- wTotalLength = 67 bytes
X"02",          -- bNumInterfaces = 2
X"01",          -- bConfigurationValue = 1
X"00",          -- iConfiguration = none
choose_byte(SELFPOWERED, X"c0", X"80"), -- bmAttributes
X"fa",          -- bMaxPower = 500 mA
-- 9 bytes interface descriptor (communication control class)
X"09",          -- bLength = 9 bytes
X"04",          -- bDescriptorType = interface descriptor
X"00",          -- bInterfaceNumber = 0
X"00",          -- bAlternateSetting = 0
X"01",          -- bNumEndpoints = 1
X"02",          -- bInterfaceClass = Communication Interface
X"02",          -- bInterfaceSubClass = Abstract Control Model
X"01",          -- bInterfaceProtocol = V.25ter (required for Linux CDC-ACM driver)
X"00",          -- iInterface = none
-- 5 bytes functional descriptor (header)
X"05",          -- bLength = 5 bytes
X"24",          -- bDescriptorType = CS_INTERFACE
X"00",          -- bDescriptorSubtype = header
X"10", X"01",   -- bcdCDC = 1.10
-- 4 bytes functional descriptor (abstract control management)
X"04",          -- bLength = 4 bytes
X"24",          -- bDescriptorType = CS_INTERFACE
X"02",          -- bDescriptorSubtype = Abstract Control Mgmt
X"00",          -- bmCapabilities = none
-- 5 bytes functional descriptor (union)
X"05",          -- bLength = 5 bytes
X"24",          -- bDescriptorType = CS_INTERFACE
X"06",          -- bDescriptorSubtype = union
X"00",          -- bMasterInterface = 0
X"01",          -- bSlaveInterface0 = 1
-- 5 bytes functional descriptor (call management)
X"05",          -- bLength = 5 bytes
X"24",          -- bDescriptorType = CS_INTERFACE
X"01",          -- bDescriptorSubType = Call Management
X"00",          -- bmCapabilities = no call mgmnt
X"01",          -- bDataInterface = 1
-- 7 bytes endpoint descriptor (notify IN)
X"07",          -- bLength = 7 bytes
X"05",          -- bDescriptorType = endpoint descriptor
X"82",          -- bEndpointAddress = IN 2

```

```

X"03",          -- bmAttributes = interrupt data
X"08", X"00",   -- wMaxPacketSize = 8 bytes
X"ff",          -- bInterval = 255 frames
-- 9 bytes interface descriptor (data class)
X"09",          -- bLength = 9 bytes
X"04",          -- bDescriptorType = interface descriptor
X"01",          -- bInterfaceNumber = 1
X"00",          -- bAlternateSetting = 0
X"02",          -- bNumEndpoints = 2
X"0a",          -- bInterfaceClass = Data Interface
X"00",          -- bInterfaceSubClass = none
X"00",          -- bInterfaceProtocol = none
X"00",          -- iInterface = none
-- 7 bytes endpoint descriptor (data IN)
X"07",          -- bLength = 7 bytes
X"05",          -- bDescriptorType = endpoint descriptor
X"81",          -- bEndpointAddress = IN 1
X"02",          -- bmAttributes = bulk data
X"40", X"00",   -- wMaxPacketSize = 64 bytes
X"00",          -- bInterval
-- 7 bytes endpoint descriptor (data OUT)
X"07",          -- bLength = 7 bytes
X"05",          -- bDescriptorType = endpoint descriptor
X"01",          -- bEndpointAddress = OUT 1
X"02",          -- bmAttributes = bulk data
X"40", X"00",   -- wMaxPacketSize = 64 bytes
X"00",          -- bInterval
-- 13 bytes padding
X"00", X"00", X"00", X"00", X"00", X"00", X"00", X"00",
X"00", X"00", X"00", X"00", X"00" );

constant descrom_hscfg: t_byte_array(0 to 79) := (
-- 67 bytes high-speed configuration descriptor
-- 9 bytes configuration header
X"09",          -- bLength = 9 bytes
X"02",          -- bDescriptorType = configuration descriptor
X"43", X"00",   -- wTotalLength = 67 bytes
X"02",          -- bNumInterfaces = 2
X"01",          -- bConfigurationValue = 1
X"00",          -- iConfiguration = none
choose_byte(SELFPOWERED, X"c0", X"80" ), -- bmAttributes = self-powered
X"fa",          -- bMaxPower = 500 mA
-- 9 bytes interface descriptor (communication control class)
X"09",          -- bLength = 9 bytes
X"04",          -- bDescriptorType = interface descriptor
X"00",          -- bInterfaceNumber = 0
X"00",          -- bAlternateSetting = 0
X"01",          -- bNumEndpoints = 1
X"02",          -- bInterfaceClass = Communication Interface
X"02",          -- bInterfaceSubClass = Abstract Control Model
X"01",          -- bInterfaceProtocol = V.25ter (required for Linux CDC-ACM driver)
X"00",          -- iInterface = none
-- 5 bytes functional descriptor (header)
X"05",          -- bLength = 5 bytes
X"24",          -- bDescriptorType = CS_INTERFACE
X"00",          -- bDescriptorSubtype = header
X"10", X"01",   -- bcdCDC = 1.10
-- 4 bytes functional descriptor (abstract control management)
X"04",          -- bLength = 4 bytes
X"24",          -- bDescriptorType = CS_INTERFACE
X"02",          -- bDescriptorSubtype = Abstract Control Mgmt

```

```

X"00",          -- bmCapabilities = none
-- 5 bytes functional descriptor (union)
X"05",          -- bLength = 5 bytes
X"24",          -- bDescriptorType = CS_INTERFACE
X"06",          -- bDescriptorSubtype = union
X"00",          -- bMasterInterface = 0
X"01",          -- bSlaveInterface0 = 1
-- 5 bytes functional descriptor (call management)
X"05",          -- bLength = 5 bytes
X"24",          -- bDescriptorType = CS_INTERFACE
X"01",          -- bDescriptorSubType = Call Management
X"00",          -- bmCapabilities = no call mgmnt
X"01",          -- bDataInterface = 1
-- 7 bytes endpoint descriptor (notify IN)
X"07",          -- bLength = 7 bytes
X"05",          -- bDescriptorType = endpoint descriptor
X"82",          -- bEndpointAddress = IN 2
X"03",          -- bmAttributes = interrupt data
X"08", X"00",   -- wMaxPacketSize = 8 bytes
X"0f",          -- bInterval = 2**14 frames
-- 9 bytes interface descriptor (data class)
X"09",          -- bLength = 9 bytes
X"04",          -- bDescriptorType = interface descriptor
X"01",          -- bInterfaceNumber = 1
X"00",          -- bAlternateSetting = 0
X"02",          -- bNumEndpoints = 2
X"0a",          -- bInterfaceClass = Data Interface
X"00",          -- bInterfaceSubClass = none
X"00",          -- bInterfaceProtocol = none
X"00",          -- iInterface = none
-- 7 bytes endpoint descriptor (data IN)
X"07",          -- bLength = 7 bytes
X"05",          -- bDescriptorType = endpoint descriptor
X"81",          -- bEndpointAddress = IN 1
X"02",          -- bmAttributes = bulk data
X"00", X"02",   -- wMaxPacketSize = 512 bytes
X"00",          -- bInterval
-- 7 bytes endpoint descriptor (data OUT)
X"07",          -- bLength = 7 bytes
X"05",          -- bDescriptorType = endpoint descriptor
X"01",          -- bEndpointAddress = OUT 1
X"02",          -- bmAttributes = bulk data
X"00", X"02",   -- wMaxPacketSize = 512 bytes
X"00",          -- bInterval = never NAK
-- other_speed_configuration hack
X"07",
-- 12 bytes padding
X"00", X"00", X"00", X"00", X"00", X"00", X"00", X"00", X"00", X"00", X"00",
X"00", X"00", X"00", X"00");
constant descrom_str0: t_byte_array(0 to 3) := (
-- string descriptor 0 (supported languages)
X"04",          -- bLength = 4
X"03",          -- bDescriptorType = string descriptor
X"09", X"04"); -- wLangId[0] = 0x0409 = English U.S.
-- Concatenate all descriptors.
constant descrom_pre: t_byte_array(0 to DESC_END-1) :=
descrom_dev & descrom_qual &
descrom_fscfg & descrom_hscfg &
descrom_str0 &
strdesc_from_string(VENDORSTR) &
strdesc_from_string(PRODUCTSTR) &

```

```

    strdesc_from_string(SERIALSTR) ;
-- Truncate descriptor data to keep only the necessary pieces;
-- either just the full-speed stuff, or full-speed plus high-speed,
-- or full-speed plus high-speed plus string descriptors.
constant descrom_have_strings : boolean :=
    (VENDORSTR'length > 0 or PRODUCTSTR'length > 0 or
     SERIALSTR'length > 0);
constant descrom_len : integer :=
    choose_int(HSSUPPORT or descrom_have_strings,
               choose_int(descrom_have_strings, DESC_END, 192), 112);

constant descrom: t_byte_array(0 to descrom_len-1) :=
    descrom_pre(0 to descrom_len-1);
signal descrom_start: unsigned(9 downto 0);
signal descrom_raddr: unsigned(9 downto 0);
signal descrom_rdat: std_logic_vector(7 downto 0);
-- RX buffer
signal rxbuf: t_byte_array(0 to (2**RXBUFSIZE_BITS-1));
signal rxbuf_rdat: std_logic_vector(7 downto 0);
-- TX buffer
signal txbuf: t_byte_array(0 to (2**TXBUFSIZE_BITS-1));
signal txbuf_rdat: std_logic_vector(7 downto 0);
-- Interface to usb_init
signal usbi_usbrst : std_logic;
signal usbi_highspeed : std_logic;
signal usbi_suspend : std_logic;
-- Interface to usb_packet
signal usbp_chirpk : std_logic;
signal usbp_rxact : std_logic;
signal usbp_rxdy : std_logic;
signal usbp_rxfn : std_logic;
signal usbp_rxdat : std_logic_vector(7 downto 0);
signal usbp_txact : std_logic;
signal usbp_txrdy : std_logic;
signal usbp_txdat : std_logic_vector(7 downto 0);
-- Interface to usb_transact
signal usbt_in : std_logic;
signal usbt_out : std_logic;
signal usbt_setup : std_logic;
signal usbt_ping : std_logic;
signal usbt_fin : std_logic;
signal usbt_endpt : std_logic_vector(3 downto 0);
signal usbt_nak : std_logic;
signal usbt_stall : std_logic;
signal usbt_nyet : std_logic;
signal usbt_send : std_logic;
signal usbt_isync : std_logic;
signal usbt_osync : std_logic;
signal usbt_rxdy : std_logic;
signal usbt_rxdat : std_logic_vector(7 downto 0);
signal usbt_txrdy : std_logic;
signal usbt_txdat : std_logic_vector(7 downto 0);
-- Interface to usb_control
signal usbc_addr : std_logic_vector(6 downto 0);
signal usbc_confd : std_logic;
signal usbc_clr_in : std_logic_vector(1 to 2);
signal usbc_clr_out : std_logic_vector(1 to 2);
signal usbc_sethlt_in : std_logic_vector(1 to 2);
signal usbc_sethlt_out : std_logic_vector(1 to 2);
signal usbc_dscbusy : std_logic;
signal usbc_dscrd : std_logic;

```

```

signal usbc_dsctyp : std_logic_vector(2 downto 0);
signal usbc_dscinx : std_logic_vector(7 downto 0);
signal usbc_dscoff : std_logic_vector(7 downto 0);
signal usbc_dsclen : std_logic_vector(7 downto 0);
signal usbc_selfpowered : std_logic;
signal usbc_in : std_logic;
signal usbc_out : std_logic;
signal usbc_setup : std_logic;
signal usbc_ping : std_logic;
signal usbc_nak : std_logic;
signal usbc_stall : std_logic;
signal usbc_nyet : std_logic;
signal usbc_send : std_logic;
signal usbc_isync : std_logic;
signal usbc_txdat : std_logic_vector(7 downto 0);
-- State machine
type t_state is (
    ST_IDLE, ST_STALL, ST_NAK,
    ST_INSTART, ST_INSEND, ST_INDONE, ST_OUTRECV, ST_OUTNAK );
signal s_state : t_state := ST_IDLE;
-- Endpoint administration (PHY side).
signal s_rxbuf_head : unsigned(RXBUFSIZE_BITS-1 downto 0) := to_unsigned(0, RXBUFSIZE_BITS);
signal s_rxbuf_tail : unsigned(RXBUFSIZE_BITS-1 downto 0);
signal s_txbuf_head : unsigned(TXBUFSIZE_BITS-1 downto 0);
signal s_txbuf_tail : unsigned(TXBUFSIZE_BITS-1 downto 0) := to_unsigned(0, TXBUFSIZE_BITS);
signal s_txbuf_stop : unsigned(TXBUFSIZE_BITS-1 downto 0);
signal s_bufptr : unsigned(BUFPTR_SIZE-1 downto 0) := to_unsigned(0, BUFPTR_SIZE);
signal s_txprev_full : std_logic := '0'; -- Last transmitted packet was full-size
signal s_txprev_acked : std_logic := '1'; -- Last transmitted packet was ack-ed.
signal s_isync : std_logic := '0';
signal s_osync : std_logic := '0';
signal s_halt_in : std_logic_vector(1 to 2) := "00";
signal s_halt_out : std_logic_vector(1 to 2) := "00";
signal s_nyet : std_logic := '0';
-- Buffer state (application side).
signal q_rxbuf_head : unsigned(RXBUFSIZE_BITS-1 downto 0);
signal q_rxbuf_tail : unsigned(RXBUFSIZE_BITS-1 downto 0) := to_unsigned(0, RXBUFSIZE_BITS);
signal q_txbuf_head : unsigned(TXBUFSIZE_BITS-1 downto 0) := to_unsigned(0, TXBUFSIZE_BITS);
signal q_txbuf_tail : unsigned(TXBUFSIZE_BITS-1 downto 0);
-- Control signals (PHY side).
signal s_reset : std_logic;
signal s_txcork : std_logic;
-- Status signals (application side).
signal q_usbrst : std_logic;
signal q_online : std_logic;
signal q_highspeed : std_logic;
-- Receive buffer logic (application side).
signal q_rxval : std_logic := '0';
signal q_rxbuf_read : std_logic;
signal q_txbuf_rdy : std_logic;
begin
-- Check buffer size.
assert ((not HSSUPPORT) or (RXBUFSIZE_BITS >= 10))
    report "High-speed device needs at least 1024 bytes RX buffer";
-- Bus reset logic
usb_init_inst : usb_init
    generic map (
        HSSUPPORT => HSSUPPORT )
    port map (
        CLK => CLK,
        RESET => s_reset,

```

```

I_USBRST      => usbi_usbrst,
I_HIGHSEED    => usbi_highspeed,
I_SUSPEND     => usbi_suspend,
P_CHIRPK      => usbp_chirpk,
PHY_RESET     => PHY_RESET,
PHY_LINESTATE => PHY_LINESTATE,
PHY_OPMODE    => PHY_OPMODE,
PHY_XCVRSELECT => PHY_XCVRSELECT,
PHY_TERMSELECT => PHY_TERMSELECT );
-- Packet level logic
usb_packet_inst : usb_packet
port map (
    CLK          => CLK,
    RESET        => usbi_usbrst,
    P_CHIRPK      => usbp_chirpk,
    P_RXACT       => usbp_rxact,
    P_RXRDY       => usbp_rxrdy,
    P_RXFIN       => usbp_rxfin,
    P_RXDAT       => usbp_rxdat,
    P_TXACT       => usbp_txact,
    P_TXRDY       => usbp_txrdy,
    P_TXDAT       => usbp_txdat,
    PHY_DATAIN    => PHY_DATAIN,
    PHY_DATAOUT   => PHY_DATAOUT,
    PHY_TXVALID   => PHY_TXVALID,
    PHY_TXREADY   => PHY_TXREADY,
    PHY_RXACTIVE  => PHY_RXACTIVE,
    PHY_RXVALID   => PHY_RXVALID,
    PHY_RXERROR   => PHY_RXERROR );
-- Transaction level logic
usb_transact_inst : usb_transact
generic map (
    HSSUPPORT     => HSSUPPORT )
port map (
    CLK          => CLK,
    RESET        => usbi_usbrst,
    T_IN         => usbt_in,
    T_OUT        => usbt_out,
    T_SETUP      => usbt_setup,
    T_PING       => usbt_ping,
    T_FIN        => usbt_fin,
    T_ADDR       => usbc_addr,
    T_ENDPT      => usbt_endpt,
    T_NAK        => usbt_nak,
    T_STALL      => usbt_stall,
    T_NYET       => usbt_nyet,
    T_SEND       => usbt_send,
    T_ISYNC      => usbt_isync,
    T_OSYNC      => usbt_osync,
    T_RXRDY      => usbt_rxrdy,
    T_RXDAT      => usbt_rxdat,
    T_TXRDY      => usbt_txrdy,
    T_TXDAT      => usbt_txdat,
    I_HIGHSEED   => usbi_highspeed,
    P_RXACT       => usbp_rxact,
    P_RXRDY       => usbp_rxrdy,
    P_RXFIN       => usbp_rxfin,
    P_RXDAT       => usbp_rxdat,
    P_TXACT       => usbp_txact,
    P_TXRDY       => usbp_txrdy,
    P_TXDAT       => usbp_txdat );

```



```

-- Default control endpoint
usb_control_inst : usb_control
generic map (
    NENDPT      => 2 )
port map (
    CLK          => CLK,
    RESET        => usbi_usbrst,
    C_ADDR       => usbc_addr,
    C_CONFD      => usbc_confd,
    C_CLRIN      => usbc_clr_in,
    C_CLROUT     => usbc_clr_out,
    C_HLTIN      => s_halt_in,
    C_HLTOUT     => s_halt_out,
    C_SHLTIN     => usbc_sethlt_in,
    C_SHLTOUT    => usbc_sethlt_out,
    C_DSCBUSY    => usbc_dsdbusy,
    C_DSCRD      => usbc_dscrd,
    C_DSCTYP     => usbc_dsctyp,
    C_DSCINX     => usbc_dscinx,
    C_DSCOFF     => usbc_dscoff,
    C_DSCLLEN    => usbc_dsclen,
    C_SELFPOWERED => usbc_selfpowered,
    T_IN         => usbc_in,
    T_OUT        => usbc_out,
    T_SETUP      => usbc_setup,
    T_PING       => usbc_ping,
    T_FIN        => usbt_fin,
    T_NAK        => usbc_nak,
    T_STALL      => usbc_stall,
    T_NYET       => usbc_nyet,
    T_SEND       => usbc_send,
    T_ISYNC      => usbc_isync,
    T_OSYNC      => usbt_osync,
    T_RXRDY      => usbt_rxdy,
    T_RXDAT      => usbt_rxdat,
    T_TXRDY      => usbt_txrdy,
    T_TXDAT      => usbc_txdat );

-- Assign usb_serial output signals.
USBRST    <= q_usbrst;
HIGHSPD    <= q_highspeed;
SUSPEND    <= usbi_suspend;
ONLINE     <= q_online;
RXVAL      <= q_rxval;
RXDAT      <= rxbuf_rdat;
RXLEN      <= std_logic_vector(q_rxbuf_head - q_rxbuf_tail);
TXRDY      <= q_txbuf_rdy;
TXROOM     <= std_logic_vector(q_txbuf_tail - q_txbuf_head - 1);

-- Assign usb_control input signals
usbc_in    <= usbt_in  when (usbt_endpt = "0000") else '0';
usbc_out    <= usbt_out when (usbt_endpt = "0000") else '0';
usbc_setup  <= usbt_setup when (usbt_endpt = "0000") else '0';
usbc_ping   <= usbt_ping  when (usbt_endpt = "0000") else '0';
usbc_selfpowered <= '1' when SELFPOWERED else '0';

-- Assign usb_transact input lines
usbt_nak    <= usbc_nak when (usbt_endpt = "0000") else
    '1'      when (s_state = ST_NAK) else
    '0';
usbt_stall  <= usbc_stall when (usbt_endpt = "0000") else
    '1'      when (s_state = ST_STALL) else

```

```

        '0';
usb_t_nyet <= usbc_nyet when (usb_t_endpt = "0000") else
    s_nyet;
usb_t_send <= usbc_send when (usb_t_endpt = "0000") else
    '1' when (s_state = ST_INSEND) else
    '0';
usb_t_isync <= usbc_isync when (usb_t_endpt = "0000") else
    s_isync;
usb_t_txdat <= usbc_txdat when (usb_t_endpt = "0000" and usbc_dsdbusy = '0') else
    descrom_rdat when (usb_t_endpt = "0000") else
    txbuf_rdat;
-- Buffer logic.
q_rxbuf_read <= (RXRDY or (not q_rxval)) when (q_rxbuf_tail /= q_rxbuf_head) else '0';
q_txbuf_rdy <= '1' when (q_txbuf_head + 1 /= q_txbuf_tail) else '0';
-- Connection between PHY-side and application-side signals.
-- This could be a good place to insert clock domain crossing.
q_rxbuf_head <= s_rxbuf_head;
s_rxbuf_tail <= q_rxbuf_tail;
s_txbuf_head <= q_txbuf_head;
q_txbuf_tail <= s_txbuf_tail;
s_txcork <= TXCORK;
s_reset <= RESET;
q_online <= usbc_confd;
q_usbrst <= usbi_usbrst;
q_highspeed <= usbi_highspeed;

-- Lookup address/length of the selected descriptor (combinatorial).
process (usbc_dsctyp, usbc_dscinx, usbi_highspeed)
    variable s: unsigned(9 downto 0);
    variable n: unsigned(7 downto 0);
begin
    s := to_unsigned(0, s'length);
    n := to_unsigned(0, n'length);
    case usbc_dsctyp is
        when "001" => -- device descriptor
            s := to_unsigned(DESC_DEV_ADDR, s'length);
            n := to_unsigned(DESC_DEV_LEN, n'length);
        when "010" => -- configuration descriptor
            if usbc_dscinx = X"00" then
                if HSSUPPORT and (usbi_highspeed = '1') then
                    s := to_unsigned(DESC_HSCFG_ADDR, s'length);
                    n := to_unsigned(DESC_HSCFG_LEN, n'length);
                else
                    s := to_unsigned(DESC_FSCFG_ADDR, s'length);
                    n := to_unsigned(DESC_FSCFG_LEN, n'length);
                end if;
            end if;
        when "011" => -- string descriptor
            if descrom_have_strings then
                case usbc_dscinx is
                    when X"00" => -- supported languages
                        s := to_unsigned(DESC_STRLANG, s'length);
                        n := to_unsigned(4, n'length);
                    when X"01" => -- vendor name
                        s := to_unsigned(DESC_STRVENDOR, s'length);
                        n := to_unsigned(2 + 2*VENDORSTR'length, n'length);
                    when X"02" => -- product name
                        s := to_unsigned(DESC_STRPRODUCT, s'length);
                        n := to_unsigned(2 + 2*PRODUCTSTR'length, n'length);
                    when X"03" => -- serial number
                        s := to_unsigned(DESC_STRSERIAL, s'length);

```

```

        n := to_unsigned(2 + 2*SERIALSTR'length, n'length);
    when others =>
        -- unsupported descriptor index
    end case;
end if;
when "110" => -- device qualifier
    if HSSUPPORT then
        s := to_unsigned(DESC_QUAL_ADDR, s'length);
        n := to_unsigned(DESC_QUAL_LEN, n'length);
    end if;
when "111" => -- other speed configuration
    if HSSUPPORT and (usbc_dscinx = X"00") then
        if usbi_highspeed = '1' then
            s := to_unsigned(DESC_FSCFG_ADDR, s'length);
            n := to_unsigned(DESC_FSCFG_LEN, n'length);
        else
            s := to_unsigned(DESC_HSCFG_ADDR, s'length);
            n := to_unsigned(DESC_HSCFG_LEN, n'length);
        end if;
    end if;
when others =>
    -- unsupported descriptor type
end case;
descrom_start <= s;
usbc_dsclen <= std_logic_vector(n);
end process;

-- Main application-side synchronous process.
process is
begin
    wait until rising_edge(CLK);

    if RESET = '1' then

        -- Reset this entity.
        q_rxbuf_tail <= to_unsigned(0, RXBUFSIZE_BITS);
        q_txbuf_head <= to_unsigned(0, TXBUFSIZE_BITS);
        q_rxval <= '0';

    else

        -- Read data from the RX buffer.
        if q_rxbuf_read = '1' then
            -- The RAM buffer reads a byte in this cycle.
            q_rxbuf_tail <= q_rxbuf_tail + 1;
            q_rxval <= '1';
        elsif RXRDY = '1' then
            -- Byte consumed by application; no new data yet.
            q_rxval <= '0';
        end if;

        -- Write data to the TX buffer.
        if (TXVAL = '1') and (q_txbuf_rdy = '1') then
            -- The RAM buffer writes a byte in this cycle.
            q_txbuf_head <= q_txbuf_head + 1;
        end if;
    end if;
end process;

-- Main PHY-side synchronous process.
process is
    variable v_max_txsize : unsigned(TXBUFSIZE_BITS-1 downto 0);
    variable v_rxbuf_len_lim : unsigned(RXBUFSIZE_BITS-1 downto 0);

```

```

variable v_rxbuf_tmp_head : unsigned(RXBUFSIZE_BITS-1 downto 0);
variable v_rxbuf_pktroom : std_logic;
begin
wait until rising_edge(CLK);
-- Determine the maximum packet size we can transmit.
if HSSUPPORT and usbi_highspeed = '1' then
    v_max_txsize := to_unsigned(MAX_HSPACKET_SIZE, TXBUFSIZE_BITS);
else
    v_max_txsize := to_unsigned(MAX_FSPACKET_SIZE, TXBUFSIZE_BITS);
end if;

-- Determine if there is room for another packet in the RX buffer.
-- We need room for the largest possible incoming packet, plus
-- two CRC bytes.
if HSSUPPORT then
    v_rxbuf_len_lim := to_unsigned(2**RXBUFSIZE_BITS - MAX_HSPACKET_SIZE - 2, RXBUFSIZE_BITS);
else
    v_rxbuf_len_lim := to_unsigned(2**RXBUFSIZE_BITS - MAX_FSPACKET_SIZE - 2, RXBUFSIZE_BITS);
end if;
if HSSUPPORT and s_state = ST_OUTRECV then
    -- Currently receiving a packet; compare against the temporary
    -- tail pointer to decide NYET vs ACK.
    v_rxbuf_tmp_head := resize(s_bufptr, RXBUFSIZE_BITS);
else
    -- Not receiving a packet (or NYET not supported);
    -- compare against the tail pointer to decide NAK vs ACK.
    v_rxbuf_tmp_head := s_rxbuf_head;
end if;
if v_rxbuf_tmp_head - s_rxbuf_tail < v_rxbuf_len_lim then
    v_rxbuf_pktroom := '1';
else
    v_rxbuf_pktroom := '0';
end if;
-- State machine
if s_reset = '1' then
    -- Reset this entity.
    s_state      <= ST_IDLE;
    s_rxbuf_head <= to_unsigned(0, RXBUFSIZE_BITS);
    s_txbuf_tail <= to_unsigned(0, TXBUFSIZE_BITS);
    s_bufptr     <= to_unsigned(0, BUFPTR_SIZE);
    s_txprev_full <= '0';
    s_txprev_acked <= '1';
    s_isync      <= '0';
    s_osync      <= '0';
    s_halt_in    <= "00";
    s_halt_out   <= "00";

elsif usbi_usbrst = '1' then

    -- Reset protocol state.
    s_state      <= ST_IDLE;
    s_txprev_full <= '0';
    s_txprev_acked <= '1';
    s_isync      <= '0';
    s_osync      <= '0';
    s_halt_in    <= "00";
    s_halt_out   <= "00";

else

    case s_state is

```

```

when ST_IDLE =>
  -- Idle; wait for a transaction
  s_nyet <= '0';
  if (usbt_endpt = data_endpt) and (usbt_in = '1') then
    -- Start of IN transaction
    if s_halt_in(1) = '1' then
      -- Endpoint halted
      s_state <= ST_STALL;
    elsif (s_txbuf_tail /= s_txbuf_head and s_txcork = '0') or s_txprev_full = '1' then
      -- Prepare to send data
      s_bufptr <= resize(s_txbuf_tail, s_bufptr'length);
      s_state <= ST_INSTART;
    else
      -- We have no data to send
      s_state <= ST_NAK;
    end if;
  elsif (usbt_endpt = data_endpt) and (usbt_out = '1') then
    -- Start of OUT transaction
    if s_halt_out(1) = '1' then
      -- Endpoint halted
      s_state <= ST_STALL;
    elsif v_rxbuf_pktroom = '1' then
      -- Prepare to receive data
      s_bufptr <= resize(s_rxbuf_head, s_bufptr'length);
      s_state <= ST_OUTRECV;
    else
      -- We have no room to store a new packet
      s_state <= ST_OUTNAK;
    end if;
  elsif HSSUPPORT and (usbt_endpt = data_endpt) and (usbt_ping = '1') then
    -- Start of PING transaction
    if v_rxbuf_pktroom = '1' then
      -- There is room in the RX buffer for another packet; do nothing (ACK).
      s_state <= ST_IDLE;
    else
      -- There is no room in the RX buffer; respond with NAK.
      s_state <= ST_NAK;
    end if;
  elsif (usbt_endpt = notify_endpt) and (usbt_in = '1') then
    -- The notify endpoint simply NAK's all IN transactions
    if s_halt_in(2) = '1' then
      -- Endpoint halted
      s_state <= ST_STALL;
    else
      s_state <= ST_NAK;
    end if;
  end if;

  -- Reset sync bits when the control endpoint tells us.
  s_isync <= s_isync and (not usbc_clr_in(1));
  s_osync <= s_osync and (not usbc_clr_out(1));

  -- Set/reset halt bits when the control endpoint tells us.
  s_halt_in(1) <= (s_halt_in(1) or usbc_sethlt_in(1)) and (not usbc_clr_in(1));
  s_halt_in(2) <= (s_halt_in(2) or usbc_sethlt_in(2)) and (not usbc_clr_in(2));
  s_halt_out(1) <= (s_halt_out(1) or usbc_sethlt_out(1)) and (not usbc_clr_out(1));

when ST_STALL =>
  -- Wait for end of transaction
  if (usbt_in = '0') and (usbt_out = '0') and (usbt_ping = '0') then

```

```

    s_state <= ST_IDLE;
end if;

when ST_NAK =>
    -- Wait for end of transaction
    if (usbt_in = '0') and (usbt_out = '0') and (usbt_ping = '0') then
        s_state <= ST_IDLE;
    end if;

when ST_INSTART =>
    -- Prepare to send data; read first byte from memory.
    if usbt_in = '0' then
        -- Transaction canceled.
        s_state <= ST_IDLE;
    elsif (s_txbuf_tail = s_txbuf_head) or
        (s_txprev_acked = '0' and resize(s_bufptr, TXBUFSIZE_BITS) = s_txbuf_stop) or
        (s_txprev_acked = '1' and s_txcork = '1') then
        -- The TX buffer is empty, or a previous empty packet
        -- is unacknowledged, or the TX buffer is corked;
        -- must send an empty packet.
        s_state <= ST_INDONE;
    else
        -- Send a non-empty packet.
        if s_txprev_acked = '1' then
            -- Set up a size limit for this packet.
            s_txbuf_stop <= s_txbuf_tail + v_max_txsize;
        end if;
        s_bufptr <= s_bufptr + 1;
        s_state <= ST_INSEND;
    end if;

when ST_INSEND =>
    -- Sending data
    if usbt_in = '0' then
        -- Transaction canceled.
        s_state <= ST_IDLE;
    elsif usbt_txrdy = '1' then
        -- Need to provide the next data byte;
        -- stop when we reach the end of the TX buffer;
        -- stop when we reach the packet size limit.
        if (resize(s_bufptr, TXBUFSIZE_BITS) = s_txbuf_head) or
            (resize(s_bufptr, TXBUFSIZE_BITS) = s_txbuf_stop) then
            -- No more bytes
            s_state <= ST_INDONE;
        else
            s_bufptr <= s_bufptr + 1;
        end if;
    end if;

when ST_INDONE =>
    -- Done sending packet; wait for ACK.
    if usbt_in = '0' then
        -- No acknowledgement
        s_txprev_acked <= '0';
        -- Set limit for next packet to the same point
        s_txbuf_stop <= resize(s_bufptr, TXBUFSIZE_BITS);
        -- Done
        s_state <= ST_IDLE;
    elsif usbt_fin = '1' then
        -- Got acknowledgement
        s_txprev_acked <= '1';
    end if;

```

```

        -- Update buffer tail
        s_txbuf_tail <= resize(s_bufptr, TXBUFSIZE_BITS);
        -- Flip sync bit
        s_isync <= not s_isync;
        -- Remember if this was a full-sized packet.
        if s_txbuf_tail + v_max_txsize = resize(s_bufptr, TXBUFSIZE_BITS) then
            s_txprev_full <= '1';
        else
            s_txprev_full <= '0';
        end if;
        -- Done
        s_state <= ST_IDLE;
    end if;

when ST_OUTRECV =>
    -- Receiving data
    if usbt_out = '0' then
        -- Transaction ended.
        -- If the transaction was succesful, usbt_fin has been
        -- asserted in the previous cycle and has triggered
        -- an update of s_rxbuf_head.
        s_state <= ST_IDLE;
    elsif (usbt_fin = '1') and (usbt_osync = s_osync) then
        -- Good packet received; discard CRC bytes
        s_rxbuf_head <= resize(s_bufptr, RXBUFSIZE_BITS) - 2;
        s_osync <= not s_osync;
    elsif usbt_rxdy = '1' then
        -- Got data byte
        s_bufptr <= s_bufptr + 1;
        if HSSUPPORT then
            -- Set NYET if there is no room to receive
            -- another packet after this one.
            s_nyet <= not v_rxbuf_pktroom;
        end if;
    end if;

when ST_OUTNAK =>
    -- Receiving data while we don't have room to store it
    if usbt_out = '0' then
        -- End of transaction
        s_state <= ST_IDLE;
    elsif (usbt_rxdy = '1') and (usbt_osync = s_osync) then
        -- This is a new (non-duplicate) packet, but we can
        -- not store it; so respond with NAK.
        s_state <= ST_NAK;
    end if;

end case;
end if;
end process;

-- It is always a fight to get the synthesizer to infer block RAM.
-- The problem is we need dual port RAM with read-enable signals.
-- The recommended coding style, with registered read addresses,
-- does not work in this case.
-- The code below generates three RAM blocks on the Xilinx Spartan-3,
-- but it is doubtful whether it will work on other FPGA families.
-- Write to RX buffer.
process (CLK) is
begin
    if rising_edge(CLK) then
        if s_state = ST_OUTRECV and usbt_rxdy = '1' then
            rxbuf(to_integer(resize(s_bufptr, RXBUFSIZE_BITS))) <= usbt_rxd;

```

```

        end if;
    end if;
end process;
-- Read from RX buffer.
process (CLK) is
begin
    if rising_edge(CLK) then
        if q_rxbuf_read = '1' then
            rxbuf_rdat <= rxbuf(to_integer(q_rxbuf_tail));
        end if;
    end if;
end process;
-- Write to TX buffer.
process (CLK) is
begin
    if rising_edge(CLK) then
        if TXVAL = '1' then
            txbuf(to_integer(q_txbuf_head)) <= TXDAT;
        end if;
    end if;
end process;
-- Read from TX buffer.
process (CLK) is
begin
    if rising_edge(CLK) then
        if (usbt_txrdy = '1') or (s_state = ST_INSTART) then
            txbuf_rdat <= txbuf(to_integer(resize(s_bufptr, TXBUFSIZE_BITS)));
        end if;
    end if;
end process;
-- Read from descriptor memory.
process (CLK) is
begin
    if rising_edge(CLK) then
        if usbc_dscrd = '1' then
            if HSSUPPORT and unsigned(usbc_dscoff) = 1 and usbc_dsctyp = "111" then
                -- Disguise the configuration descriptor as an
                -- other_speed_configuration descriptor.
                descrom_raddr <= to_unsigned(DESC_OTHERSPEED_ADDR, descrom_raddr'length);
            else
                descrom_raddr <= descrom_start + resize(unsigned(usbc_dscoff), descrom_raddr'length);
            end if;
        end if;
    end if;
end process;
assert to_integer(descrom_raddr) < descrom'length;
descrom_rdat <= descrom(to_integer(descrom_raddr));

end architecture usb_serial_arch;

```


Referencias Bibliográficas

- Baena A., (2010). “*DISEÑO SOBRE FPGA DE UNA UNIDAD ARITMÉTICA DECIMAL*”. Tesis de Titulación en Ingeniería Técnica Industrial en Electrónica Industrial. Universitat Rovira i Virgili, Cataluña España.
- Hitachi Ltd. (1998). Semiconductor & Integrated Circuits HD44780U (LCD-II), “Dot Matrix Liquid Crystal Display Controller/Driver”.
- <http://es.wikipedia.org/wiki/Led>. Nombre de la Página: “LED”. Fecha de consulta: Diciembre-2012.
- <http://galaxi0.wordpress.com/about/salidas-y-entradas-digitales/matriz-de-leds/>. Nombre de la Página: “Matriz de Leds”. Fecha de consulta: Diciembre-2012
- www.sdi-12.org. Nombre de la Página: “Interfaces”. Fecha de consulta: Diciembre-2012
- <http://tecnoface.com/tutoriales/12-practicando-con-displays-de-7-segmentos/19-conceptos-basicos-de-un-display-de-7-segmentos>. Nombre de la Página: “Display de 7 segmentos”. Fecha de consulta: Diciembre-2012
- <http://www.vesa.org/>. Nombre de la Página: “VGA”, fecha de consulta: Diciembre-2012
- <http://www.libusb.org/>. Nombre de la Página: “USB”, fecha de consulta: Diciembre-2012
- Padrón A., (1997). *DISEÑO DE CIRCUITOS ELECTRÓNICOS PARA GENERAR FUNCIONES DE ACTIVACIÓN EMPLEADAS EN REDES NEURONALES ARTIFICIALES*. Tesis de Maestría en Ingeniería Eléctrica. División de Estudios de Posgrado de la Facultad de Ingeniería - UNAM.
- Padrón A., Pérez J. L., Prieto R., Herrera A., (2000). *DYNAMICAL BEHAVIOR OF AN ELECTRONIC NEURON OF COMMUTATION*. Lecture Notes in Artificial Intelligence 1793 (Subseries of Lecture Notes in Computer Science) / MICAI 2000 Advances in Artificial Intelligence / Cairo O, Sucar L.E, y Cantu A.J. / Berlin; Heidelberg; New York; Barcelona; Hong Kong; London; Milan; Paris; Singapore; Tokyo / Ed. SPRINGER / 2000 / ISBN 3-540-67354-7 / 2000 / pp 338-349.
- Padrón A., Velázquez J., Prieto R., Herrera A., Calva G., (2011). *FPGA IMPLEMENTATION OF CIPHER BLOCK AES WITH OPERATION MODE CFB*, “Second International Congress Instrumentation and Applied Sciences”, Puebla-México.
- Prieto R. Padrón A., Herrera A., Calva G., (2011a). Desarrollo de Funciones para el Manejo de Periféricos por Medio de un FPGA, Clave de control: II-INME-2013-336.
- Prieto R. Padrón A., Herrera A., Calva G., (2011b). *GENERIC PLATAFORM FOR IMPLEMENTATION OF STREAM CIPHERS IN FPGA’S*, “Second International Congress Instrumentation and Applied Sciences”, Puebla-México.
- Spartan-3E FPGA Starter Kit Board User Guide, (2008). UG230 (v1.1) June 20.
- Terés L., Torroja Y., Olcoz S., Villar E. (1998). VHDL Lenguaje estándar de diseño electrónico, Ed. McGrawHill, España.