



UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

Diseño de Circuitos y Sistemas Electrónicos

Memoria de Práctica Módulo 3

Edición de Mensajes en LCD

| | |
|------------------|---|
| Título : | Edición de Mensajes en LCD |
| Fecha : | Fecha del documento |
| Versión : | Número de versión del documento |
| Autor(es) : | Eugenio Peñate Fariñas <epf.supersonic@gmail.com> |
| Grupo : | (dcse08310) |
| Resumen : | Resumen breve (entre 2 a 5 líneas) del contenido |
| Palabras Clave : | Lista de palabras clave |

Aceptada por :

Estado del Documento

Revisões y Cambios

| Versión | Cambios |
|---------|-------------------------------|
| V1.0 | <u>Creación del documento</u> |
| | |
| | |
| | |

Contenido

| | |
|--|-----------|
| DISEÑO DE CIRCUITOS Y SISTEMAS ELECTRÓNICOS..... | 1 |
| MEMORIA DE PRÁCTICA MÓDULO 3 | 1 |
| ACEPTADA POR : | 1 |
| ESTADO DEL DOCUMENTO..... | 2 |
| REVISIONES Y CAMBIOS | 2 |
| CREACIÓN DEL DOCUMENTO..... | 2 |
| INTRODUCCIÓN..... | 5 |
| OBJETIVOS..... | 6 |
| MODO DE EDICIÓN (MODO = 0) | 7 |
| MODO DE VISUALIZACIÓN (MODO=1)..... | 7 |
| ESPECIFICACIÓN DETALLADA DEL DISEÑO..... | 7 |
| LCD_CONTROL | 9 |
| CONTROL_INTERFAZ..... | 9 |
| DEBOUNCE | 10 |
| KB_TEST | 11 |
| FLUJO DE DISEÑO..... | 11 |
| DESCRIPCIÓN DE LOS BLOQUES..... | 14 |
| MÓDULO 3 | 14 |
| <i>Descripción Funcional</i> | 14 |
| <i>Señales de E/S</i> | 15 |
| <i>Símbolo</i> | 15 |
| <i>Descripción VHDL y plantillas para su utilización</i> | 16 |
| <i>Características temporales</i> | 19 |
| <i>Recursos utilizados de la FPGA</i> | 19 |
| LCD_CONTROL | 20 |
| <i>Descripción Funcional</i> | 20 |
| <i>Señales de E/S</i> | 26 |
| <i>Símbolo</i> | 26 |
| <i>Descripción VHDL y plantillas para su utilización</i> | 27 |
| <i>Test-bench y simulaciones</i> | 35 |
| <i>Recursos utilizados de la FPGA</i> | 42 |
| CONTROL_INTERFAZ..... | 42 |
| <i>Descripción Funcional</i> | 42 |
| <i>Señales de E/S</i> | 43 |
| <i>Símbolo</i> | 44 |
| <i>Descripción VHDL y plantillas para su utilización</i> | 44 |
| <i>Recursos utilizados de la FPGA</i> | 48 |
| KB_TEST | 48 |
| <i>Descripción Funcional</i> | 48 |
| <i>Señales de E/S</i> | 49 |
| <i>Símbolo</i> | 49 |
| <i>Descripción VHDL y plantillas para su utilización</i> | 49 |
| Ps2_RX | 53 |
| <i>Descripción Funcional</i> | 53 |
| <i>Señales de E/S</i> | 55 |
| <i>Símbolo</i> | 55 |

| | |
|--|-----------|
| <i>Descripción VHDL y plantillas para su utilización.....</i> | 55 |
| <i>Recursos utilizados de la FPGA</i> | 57 |
| UART | 57 |
| <i>Descripción Funcional</i> | 57 |
| <i>Señales de E/S.....</i> | 59 |
| <i>Símbolo</i> | 60 |
| <i>Descripción VHDL y plantillas para su utilización.....</i> | 61 |
| <i>Recursos utilizados de la FPGA</i> | 67 |
| RESULTADOS..... | 69 |
| PROTOTIPADO | 69 |
| CONCLUSIONES..... | 72 |
| BIBLIOGRAFÍA | 73 |
| HOJA DE CARACTERÍSTICAS DEL COMPONENTE OBTENIDO | 74 |
| NOMBRE DEL COMPONENTE | 74 |
| DESCRIPCIÓN GENERAL Y CARACTERÍSTICAS | 74 |
| <i>Ventajas</i> | 74 |
| <i>Aplicaciones</i> | 74 |
| <i>Descripción del Producto.....</i> | 74 |
| SÍMBOLO | 75 |
| DIAGRAMA DE BLOQUES..... | 75 |
| PATILLAS DE E/S..... | 75 |
| DESCRIPCIÓN VHDL Y PLANTILLA PARA SU UTILIZACIÓN..... | 76 |
| TEST-BENCH Y SIMULACIÓN..... | 76 |
| CARACTERÍSTICAS TEMPORALES (FRECUENCIA DE RELOJ, RETARDOS, ETC.) | 83 |
| RECURSOS UTILIZADOS DE LA FPGA | 83 |

Introducción.

La tercera práctica del curso consiste en realizar un diseño digital que permita editar mensajes en la pantalla LCD de la placa de evaluación Spartan3E. La introducción de texto se realizará haciendo uso de los pulsadores disponibles en la placa y a través de un teclado PS/2. El mensaje editado podrá desplazarse en pantalla (realización de scroll continuo a izquierda y derecha).

La pantalla LCD que contiene la placa de evaluación es un sencillo y económico módulo alfanumérico, el cual contiene el circuito integrado controlador que realiza la mayoría del trabajo por el diseñador. Hay que señalar que la mayoría de los módulos pequeños utilizan el controlador Hitachi HD44780. Entre los módulos más comunes se encuentran los fabricados por Sitronix, Samsung, Optrex, Epson, Hitachi, Amperex y Densitron.

Las pantallas multicarácter se suelen construir utilizando visualizadores de segmentos multiplexados cableados de forma independiente. El microprocesador que los soporta enciende de forma secuencial los caracteres deseados en cada dígito del visualizador, uno cada vez. El microprocesador es lo suficientemente rápido como para que los ojos vean el resultado como realmente se quiere obtener, con varios caracteres visualizándose al mismo tiempo. Este método de multiplexar los caracteres del visualizador se usa frecuentemente, ya que reduce de forma considerable la circuitería necesaria en comparación con los sistemas no multiplexados. Sin embargo, la multiplexación requiere que el microprocesador actualice de forma continua los datos del visualizador, y además se tiene que incrementar la circuitería cuando se añaden caracteres adicionales.

Por ejemplo, un visualizador numérico de 10 dígitos necesita aproximadamente 100 hilos y unos 20 componentes. Un visualizador alfanumérico de 10 caracteres necesita incluso más cables. El visualizador equivalente (incluidos los alfanuméricos) realizado mediante un módulo LCD necesita únicamente 10 hilos y dos componentes: el módulo LCD y un potenciómetro para el control de contraste. Utilizando un módulo LCD, cualquier diseñador puede incluir un visualizador que contenga hasta 80 caracteres con solo 10 hilos, siete de los cuales conectan el visualizador al sistema microcontrolador/microprocesador, más uno de alimentación, otro de masa y un controlador LCD para el ajuste de contraste.

Los módulos de visualización manejan de forma automática las funciones de refresco y de multiplexación. El sistema que incluye al visualizador únicamente escribe los datos que se deben visualizar junto con algunos códigos de control (como activación y desactivación de la pantalla de visualización, desplazamiento a la derecha o a la izquierda, etc.), en la memoria del módulo; a partir de este momento el circuito integrado controlador del módulo LCD hace el resto.

La mayoría de los módulos pequeños y económicos LCD disponen del circuito integrado controlador de Hitachi HD44780. Esto significa que todos ellos tienen el mismo formato estándar, con el mismo interfaz de 14 patillas y, por lo tanto, son compatibles e intercambiables unos por otros. Los tamaños más normales incluyen visualizadores con formatos de 16 x 1, 16 x 2, 20 x 2, 24 x 2 y 40 x 2. Esto significa que se puede modificar el tamaño del visualizador con únicamente insertar un módulo mayor. No se necesitan modificaciones adicionales del hardware; únicamente precisarán algún cambio los controladores de software de la aplicación concreta.

Los módulos LCD reconocen el formato ASCII estándar para las letras (mayúsculas y minúsculas) y números, además de una variedad de símbolos incluyendo ?, ¡, \$, %, y ', entre otros. En todos los modelos se soportan 192 caracteres alfanuméricos y 32 símbolos

especiales. Los módulos también permiten la utilización de ocho caracteres definidos por el usuario.

Los módulos LCD son del tipo de una matriz de puntos por cada carácter formados por cinco puntos de ancho y siete de alto (formatos de 5 x 7) o por bloques de cinco puntos de ancho por 10 de alto (formatos de 5 x 10). Estos formatos son seleccionados mediante un mandato de control.

También existe una línea de cursor, debajo de cada carácter. El formato de 5 x 10 es mucho más adecuado para algunos caracteres en minúscula como es el caso de la "g", la "y" y la "p" (es decir, las letras y símbolos con partes que descienden por debajo del nivel de escritura del resto de los caracteres). Es de destacar que la matriz de 5 x 10 encuentra los límites del visualizador independientemente de que dicho módulo tenga una o dos líneas.

Los módulos LCD soportan una serie de posibilidades de visualización que se pueden acomodar a cada tipo de aplicación. Lo que sigue es una pequeña descripción de este tipo de posibilidades:

- Activación/desactivación de la pantalla: permite que el usuario encienda o apague el visualizador a través del procesador.
- Activación/desactivación del cursor: permite seleccionar el cursor de pantalla o eliminarlo.
- Parpadeo del cursor: el usuario puede elegir entre un cursor estático o parpadeante. El carácter libre del cursor también parpadeará.
- Desplazamiento a la derecha/izquierda: desplaza los datos de la pantalla del visualizador.
- Vuelta al primer carácter de la pantalla: hace que el cursor vuelva a la primera posición de la pantalla de visualización (posición HOME) si se ha desplazado previamente.

El sistema que utiliza la pantalla LCD necesita dos controladores para poder trabajar con los módulos LCD, el de escritura de mandatos de control y el de escritura de datos. Las funciones mínimas que deben poder realizar cada uno de estos controladores son las siguientes:

Escritura de mandatos de control:

- Poner en las líneas DB0-DB7 el código de control deseado.
- Colocar la línea R/W a cero lógico.
- Colocar la línea RS a cero lógico.
- Activar la línea de habilitación.

Escritura de datos:

- Poner en las líneas DB0-DB7 el carácter deseado.
- Colocar la línea R/W a cero lógico.
- Colocar la línea RS a uno lógico.
- Activar la línea de habilitación.

Objetivos.

El diseño sintetizado deberá tener 2 modos de funcionamientos seleccionables mediante la entrada modo (mapeada al interruptor sw0 de la placa). A continuación se detallan las características de funcionamiento en cada uno de los modos.

Modo de edición (modo = 0)

Permite componer el mensaje que podrá estar formado por letras mayúsculas, minúsculas, dígitos, caracteres de puntuación y espacios en blanco. En este modo, el usuario dispondrá de los 4 pulsadores presentes en la placa, así como del interruptor de rotación.

Cada uno de los 4 pulsadores que rodean al interruptor de rotación deberá tener asociado un grupo de caracteres seleccionables para mostrar en el LCD. Cada pulsación del pulsador hará que se envíe al LCD el siguiente carácter de la lista (sin avanzar de columna). La siguiente tabla muestra el grupo de caracteres asociados a cada pulsador.

| | |
|-----------|----------------------------|
| BTN_NORTH | ABCDEFGHIJKLMNPQRSTUVWXYZ |
| BTN_SOUTH | abcdefghijklmnopqrstuvwxyz |
| BTN_EAST | 0123456789 |
| BTN_WEST | ¡"#\$%&'()*+,.-./ |

Para aceptar el carácter mostrado y pasar a la siguiente columna, se usará el interruptor de rotación. Si se gira a la derecha, se avanzará a la siguiente columna. Si se gira a la izquierda, se pasará a la columna anterior.

Como atajo, sea cual sea el carácter mostrado en la posición actual, se podrá sobrescribir con un blanco usando el propio interruptor de rotación como pulsador.

De forma adicional, se ha incluido un controlador de teclado con el que se puede introducir caracteres en el LCD. Para facilitar la escritura con el teclado se usará un modo de escritura continuo que se elegirá con el sw1=1. Si sw1=0 el cursor permanecerá en la misma columna después de escribir un carácter.

Modo de visualización (modo=1)

Presenta el mensaje editado realizando un scroll continuo en pantalla de derecha a izquierda.

En este modo el uso de cualquiera de los pulsadores usados en el modo edición deberá ser ignorado. La única acción de usuario que puede ser reconocida será la de mover sw0 de nuevo a la posición baja para terminar la visualización y volver a la edición. Cuando se entra en este modo, toda la pantalla será desplazada a izquierda y derecha continuamente. Comenzará con un desplazamiento a izquierda de 16 caracteres (con lo que desaparece de la pantalla todo el texto editado), seguido de un desplazamiento hacia la derecha de otros 16 caracteres (con lo que la pantalla vuelve al sitio original).

Especificación detallada del diseño.

El sistema principal se ha llamado “Modulo3”. Su misión principal es la de interconectar los distintos subsistemas. A continuación se muestra la interfaz de entrada salida.

Tabla 1. Entidad del Modulo3

| Puerto | Sentido | Ancho | Función | Placa |
|-----------|---------|-------|--|-------|
| clk | IN | 1 | Reloj(50MHz) | C9 |
| modo | IN | 3 | Switches de modo Sw0 – edición(0), visualización(1) Sw1 – cursor, sin avance(0), avanza(1) | |
| pb | IN | 4 | Pulsadores | |
| rot_press | IN | 1 | Pulsador del botón rotatorio | |
| rot_a | IN | 1 | Interruptor Rotatorio | |
| rot_b | IN | 1 | Interruptor Rotatorio | |
| ps2c | IN | 1 | Señal de reloj PS/2 | |
| ps2d | IN | 1 | Señal de datos PS/2 | |
| rx | IN | 1 | Recepción UART | |
| tx | OUT | 1 | Transmisión UART | |
| lcd_db | OUT | 4 | Comando/Dato al LCD | |
| lcd_e | OUT | 1 | Habilitación del LCD | |
| lcd_rs | OUT | 1 | Indica comando(0) ó dato(1) | |
| lcd_rw | OUT | 1 | Indica escritura(0), lectura(1) | |
| lcd_CE0 | OUT | 1 | Señal para usar el LCD | |
| leds | OUT | 8 | Visualización de estados | |

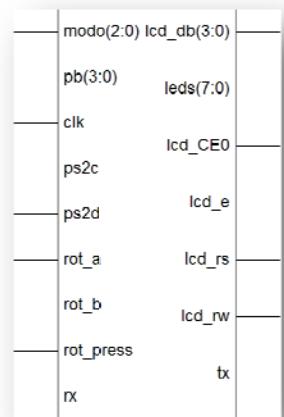


Figura 1: Modulo3

Este módulo a su vez está compuesto por tres sub-módulos: LCD_control, Control_interfaz

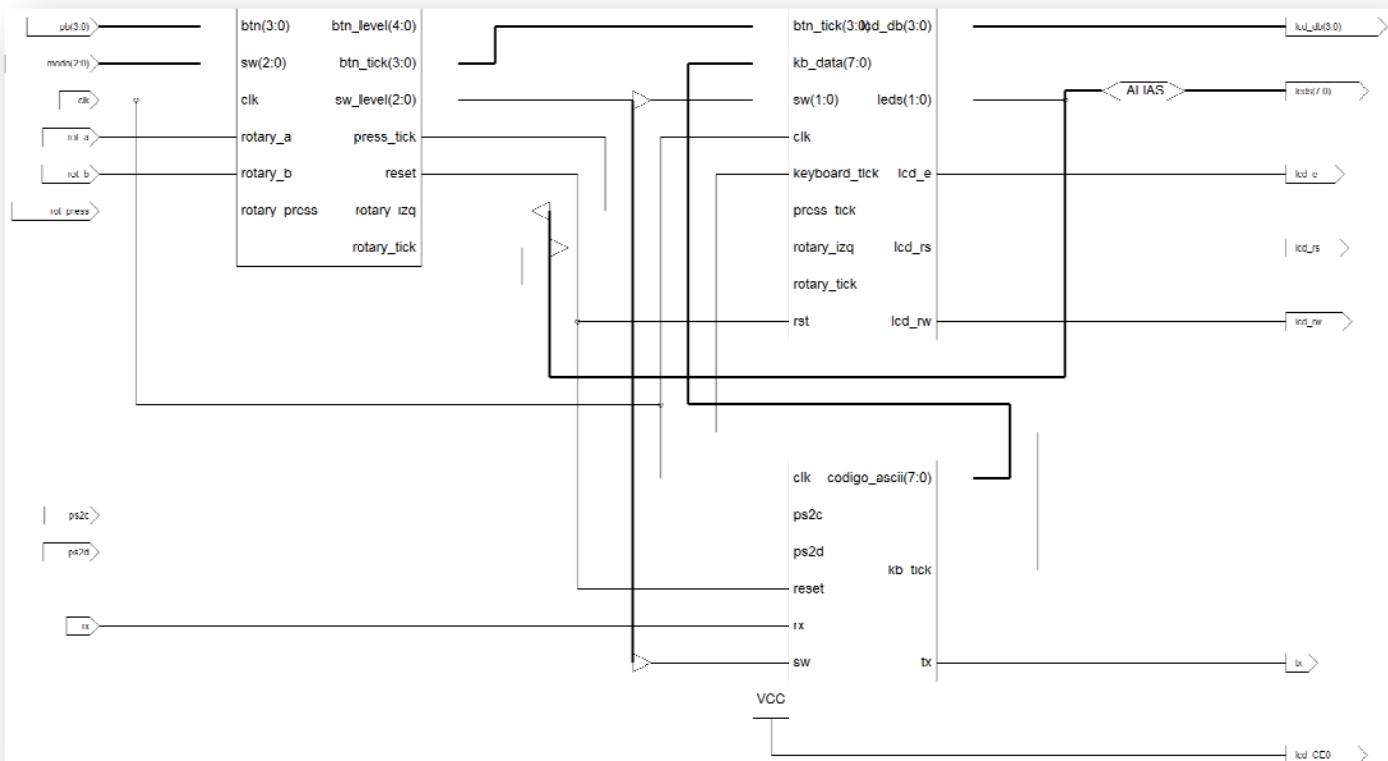


Figura 2. Diagrama de bloques del Modulo3. LCD_control, Control_interfaz y kb_test

y kb_test como se muestra en la siguiente figura. A continuación se describe las características y funcionalidad de cada módulo.

LCD_control

Este módulo contiene una máquina de estados finitos que se encarga del control del LCD. Realiza la inicialización a 4bits y la configuración inicial, al mismo tiempo que gestiona la comunicación con el LCD y controla el comportamiento del sistema. Se puede decir que éste es el núcleo del sistema. A continuación se muestra una tabla con la entidad del módulo indicando sus conexiones de entrada salida y su función.

Tabla 2.LCD_control

| Puerto | Sentido | Ancho | Función |
|---------------|---------|-------|---|
| clk | IN | 1 | Reloj(50MHz) |
| rst | IN | 1 | Reset |
| sw | IN | 2 | Switches de modo |
| btn_tick | IN | 4 | Evento en pulsadores |
| press_tick | IN | 1 | Evento en pulsador rotatorio |
| Keyboard_tick | IN | 1 | Evento en teclado |
| rotary_tick | IN | 1 | Evento en rotatorio |
| rotary_izq | IN | 1 | Indica dirección de giro 0-> giro a derechas 1-> giro a izquierda |
| kb_data | IN | 8 | ASCII en mayúsculas teclado |
| Lcd_db | OUT | 4 | Comando/Datos LCD |
| lcd_e | OUT | 1 | Habilitación del LCD |
| lcd_rs | OUT | 1 | Indica comando(0) ó dato(1) |
| lcd_rw | OUT | 1 | Indica escritura(0), lectura(1) |
| leds | OUT | 2 | Estados de la FSMD |

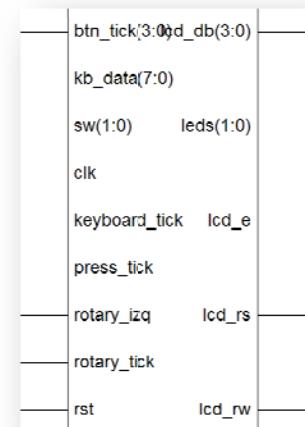


Figura 3. LCD_control

Control_interfaz

Este módulo se encarga de filtrar los rebotes mecánicos en los botones y los switches. También se encarga del realizar la lógica para utilizar el botón rotatorio y de implementar la función de reset cuando se pulsan simultáneamente los pulsadores este y oeste.

Para realizar el filtrado de los botones y switches, utiliza el componente debounce que devuelve una señal indicando el nivel del la entrada (db_level) y una señal indicando que se ha producido un evento (db_tick) cuando la entrada cambia de nivel bajo a alto. Esto es útil porque facilita la implementación del módulo LCD_control ya que se genera una señal evento de que solo está activa un ciclo de la señal del reloj. En la siguiente tabla se presenta una descripción de la entidad.

Tabla 3. Control_interfaz

| Puerto | Sentido | Ancho | Función |
|--------------|---------|-------|--|
| clk | IN | 1 | Reloj(50MHz) |
| rst | IN | 1 | Reset |
| sw | IN | 3 | Switches de modo |
| btn | IN | 4 | Botones de entrada |
| rotary_a | IN | 1 | Interruptor Rotatorio |
| rotary_b | IN | 1 | Interruptor Rotatorio |
| rotary_press | IN | 1 | Pulsador del botón rotatorio |
| reset | OUT | 1 | Reset del sistema |
| btn_level | OUT | 5 | Posición de los botones btn_level(4)->rot.press btn_level(3)->btn_north btn_level(2)->btn_east btn_level(1)->btn_south btn_level(0)->btn_west |
| btn_tick | OUT | 4 | Evento en los botones |
| sw_level | OUT | 3 | Evento en los switches |
| press_tick | OUT | 1 | Evento en el pulsador rot. |
| rotary_tick | OUT | 1 | Evento en el rotatorio |
| rotary_izq | OUT | 1 | Indica dirección de giro 0-> giro a derechas 1-> giro a izquierda |

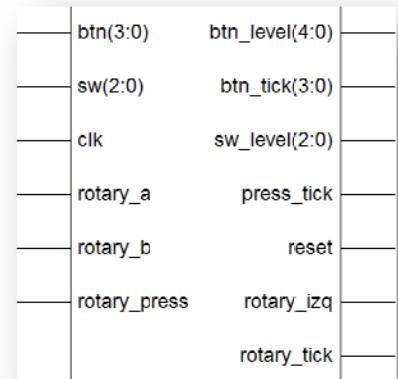


Figura 4. Control_interfaz

Debounce

Se trata de una máquina de estado finito que se encarga de filtrar los rebotes producidos por elementos mecánicos como los pulsadores y switches. Esta FSM espera a que la señal de entrada permanezca estable a nivel alto unos 40ms y genera una señal de evento en el momento en que la señal de nivel de salida cambia de nivel bajo a nivel alto. Una descripción de los pines de E/S se presenta en la siguiente tabla.

Tabla 4. Debounce

| Puerto | Sentido | Ancho | Función |
|----------|---------|-------|--|
| clk | IN | 1 | Reloj(50MHz) |
| rst | IN | 1 | Reset |
| sw | IN | 1 | Señal de entrada |
| db_level | OUT | 1 | Indica el nivel de la señal de entrada |
| Db_tick | OUT | 1 | Indica q se ha producido un evento |



Figura 5. Debounce

Kb_test

Este módulo se encarga básicamente de interconectar tres submodulos:

- Kb_code. Se encarga de la comunicación con el teclado.
- UART. Se encarga de transmitir los datos a través del puerto serie.
- Key2ascii. Se encarga de convertir el código del teclado en caracteres ASCII.

Tabla 5. Kb_test

| Puerto | Sentido | Ancho | Función |
|--------------|---------|-------|--|
| clk | IN | 1 | Reloj(50MHz) |
| reset | IN | 1 | Reset |
| sw | IN | 1 | Señal de entrada |
| ps2d | IN | 1 | Datos serie del teclado |
| ps2c | IN | 1 | Reloj puerto PS2 |
| rx | IN | 1 | Recepción puerto serie |
| tx | OUT | 1 | Transmisión puerto serie |
| kb_tick | OUT | 1 | Se produce un evento por tecla pulsada |
| Código_ascii | OUT | 8 | Código ascii de la tecla pulsada |

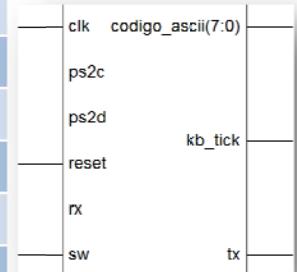


Figura 6.kb_test

Flujo de diseño.

Se ha empezado el diseño implementando los componentes de interfaz, es decir, la electrónica que controla los botones, switches, teclado y comunicación serie. Para comprobar su funcionamiento, se han utilizado los leds de la placa como interfaz de presentación. Una vez conseguido el correcto funcionamiento, se ha comenzado

En prácticamente todos los módulos se han tenido que usar circuitos secuenciales, FSM y FSMD. Siguiendo la filosofía de diseño del libro (Pong P. Chu), se han separado los elementos de memoria de los elementos combinacionales. Para mayor claridad, se usan los sufijos _next y _reg para enfatizar el siguiente valor de entrada y la salida registrada de un flipflop. También, como se ha seguido un diseño síncrono, excepto en la unidad UART que por definición es asíncrona, se ha usado el sufijo _tick para indicar que se ha producido un evento. Esta permanecerá activa durante un ciclo de la señal de reloj.

El módulo Control_Interfaz se encarga de controlar los botones, switches y botón rotatorio, y devuelve principalmente una señal de evento por cada botón, switch y rotatorio. También devuelve el estado del botón pero solamente se utiliza para presentarlo a través de los leds. El botón rotatorio, devuelve una señal evento, indicando que se ha girado y una señal _izq que si está activada indica que se girado a la izquierda, en caso contrario se ha girado a la derecha. La lógica de control ha sido extraída del manual de *Rotary Encoder Interface for Spartan-3E Starter Kit* (Ken Chapman).

El módulo kb_test se encarga de controlar el puerto serie y el teclado. El puerto serie se controla con el componente instanciado *uart_unit* que envía los datos que le entran por el puerto *w_data*, cuando la señal *wr_uart* se activa. Esta señal solamente la activa el teclado cuando recibe cada carácter. El teclado es controlado por el componente instanciado *kb_code_unit* que se compone de una unidad receptora de PS/2 (*ps2_rx_unit*), una fifo (*fifo_key_unit*) y un circuito que se encarga de almacenar en la fifo el siguiente byte después del código de ruptura que envía el teclado. Los datos provenientes de la unidad *kb_code_unit*

son pasados a través de la unidad *key2a_unit* que convierte el código del teclado en códigos ascii. Esto lo hace a través de un multiplexor que dependiendo del dato de entrada selecciona uno de los caracteres ASCII disponibles. Por simplicidad, solamente se ha rellenado el multiplexor con los caracteres en mayúsculas, el espacio, el enter y el backspace, el resto se ha rellenado con el código ascii del asterisco (*). El código de cada tecla se proporciona en el manual de la placa (Spartan-3E Starter Kit Board User Guide) y se reproduce en la siguiente imagen.

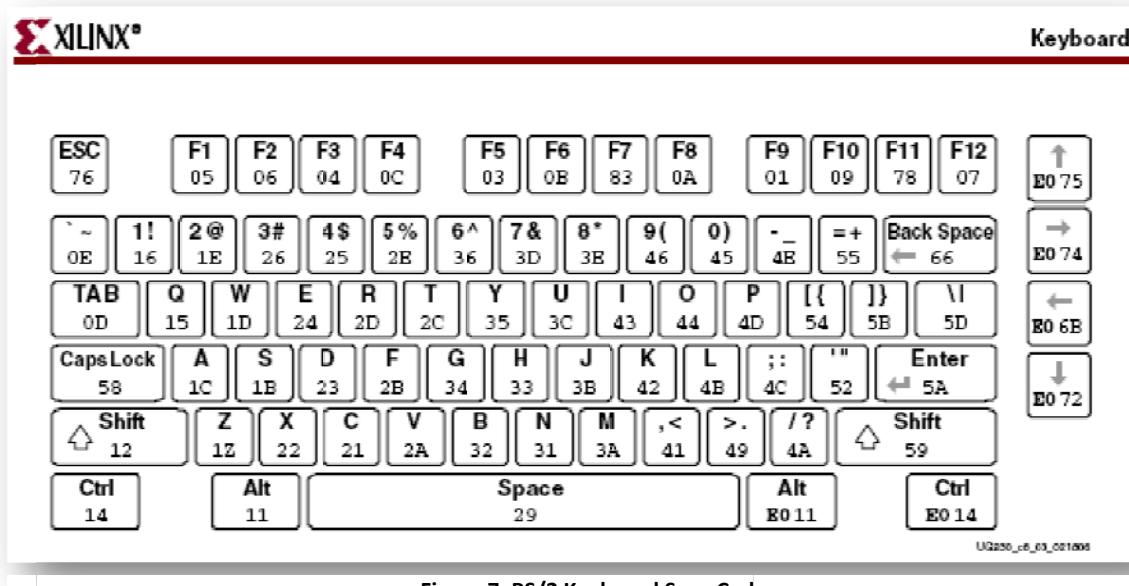


Figura 7. PS/2 Keyboard Scan Codes

El módulo LCD_control es el corazón de la práctica. Éste actúa como controlador del LCD y gestiona las operaciones que debe realizar el sistema. Se encarga de inicializar el LCD, de configurarlo, y gestionar las peticiones del usuario para presentar los datos. Está implementado con una FSMD de 23 estados sencillos. Se podría haber reducido 6 estados, a costa de complicar la lógica de salida y estado siguiente, pero por claridad en código y sencillez de la lógica, se ha decidido dejar los 23 estados. La FSMD se ayuda de un multiplexor para almacenar los mandatos de inicialización, configuración, la bienvenida inicial y para seleccionar entre mandatos predefinidos o datos de entrada. Se ha decidido usar un multiplexor en vez de una memoria principalmente por sencillez y por ser lógica combinacional no tiene que esperar un ciclo de la señal de reloj para tener disponible a su salida los datos de entrada. El multiplexor se encarga de gestionar dos señales, data_conf y rs_conf. La señal data_conf la utiliza la FSMD para enviar los datos al bus de datos del LCD y rs_conf es la señal de rs que envía la FSMD al LCD.

El camino de datos está formado por el multiplexor antes citado, sumadores, restadores, lógica para el control de los botones y registros.

Los 23 estados de la FSMD se pueden dividir en cuatro secciones:

- Inicialización. Esta sección se encarga de inicializar con el interfaz de 4 bits. Los estados se han nombrado idle, poi1..9. En idle y poi1 se esperan 15ms para dar tiempo al LCD que esté disponible. En los estados poi1 a poi9 (poi=Power On Initialization) se realiza la secuencia que indica el manual de (Spartan-3E Starter Kit Board User Guide) para usar el LCD con una interfaz de datos de 4bits.
 - Poi2. Escribe LCD_DB = 0x3 y LCD_E a nivel alto durante 12 ciclos de reloj.

- Poi3. Espera 4.1ms que son 205000 ciclos de reloj a 50MHz
- Poi4. Escribe LCD_DB = 0x3 y LCD_E a nivel alto durante 12 ciclos de reloj.
- Poi5. Espera 100 μ s que son 5000 ciclos de reloj a 50MHz
- Poi6. Escribe LCD_DB = 0x3 y LCD_E a nivel alto durante 12 ciclos de reloj.
- Poi7. Espera 40 μ s que son 2000 ciclos de reloj a 50MHz
- Poi8. Escribe LCD_DB = 0x2 y LCD_E a nivel alto durante 12 ciclos de reloj.
- Poi9. Espera 40 μ s que son 2000 ciclos de reloj a 50MHz
- Configuración del display. Después de que la inicialización se complete la interfaz de datos de cuatro bits se ha establecido. Lo siguiente es configurar el display de la siguiente forma:
 - Enviar el comando Function Set, 0x28 para configurar el display para operar en modo 4bits.
 - Enviar el comando Entry Mode Set, 0x06 para hacer que el puntero de direcciones se incremente automáticamente.
 - Enviar el comando Display On/Off, 0x0C para encender el display y deshabilitar el cursor y parpadeo.
 - Finalmente, enviar el comando Clear Display, 0x01 y esperar 1.64ms.
- Bienvenida inicial. Esta sección se encarga de presentar en pantalla una bienvenida. La secuencia que presenta en pantalla es la siguiente.



Esta sección y la anterior comparten los mismos estados: conf_hsetup, conf_hhold, conf_unus, conf_lsetup, conf_lhold, conf_40us. Estos 6 estados son necesarios para cumplir con las especificaciones de tiempo para la interfaz

de 4 bits. Con conf_hsetup se coloca en el bus de datos los 4 bits superiores y rs selecciona si es comando o dato. Con el registro c_reg se realizan las cuentas de tiempo y en este estado se permanece durante 2 ciclos de la señal de reloj para esperar 40ns. El estado siguiente es conf_hhold, que mantiene los datos del estado anterior mientras mantiene durante 12 ciclos de la señal de reloj la señal de enable lcd_e. Con conf_unus se esperan 1μs antes de enviar los 4bits menos significativos. Con conf_lsetup se colocan estos bits en el bus de datos y se esperan 2 ciclos. Con conf_lhold se activa lcd_e durante 12 ciclos y finalmente en el estado conf_40us se esperan 40μs para la mayoría de los comandos y 1.64ms en el caso de un clear.

El registro n_reg se utiliza como contador para ir seleccionando a través de un multiplexor el comando que hay que enviar en estas dos secciones. Este registro se incrementa en el estado conf_40us.

- La última sección es la del controlador del LCD. Consta de los estados mono, tx_hsetup, tx_hhold, tx_unus, tx_lsetup, tx_lhold, tx_40us.

En el estado modo se mira primeramente si está activado o no el switch(0). Si está en modo edición (sw0=0), se mira si se ha producido algún evento en el teclado, botón rotatorio o pulsadores. En el caso de que se produzca algún evento en algún pulsador se envía el registro correspondiente de cada pulsador y se incrementa. Si se produce un evento en el rotatorio, se comprueba la señal rotary_izq para ver si el giro ha sido hacia la izquierda o derecha. En caso de ser hacia la izquierda se envía X"10", y en caso contrario X"14" para mover el cursor. Por último si es el teclado el que produce el evento se envía el código ASCII que devuelve la interfaz del teclado. Cualquier otro caso como que se produzcan dos eventos al mismo tiempo no se hace nada y se vuelve al estado modo.

Para realizar la transmisión cumpliendo con las restricciones de tiempo del LCD se han creado los 6 estados tx_hsetup, tx_hhold, tx_unus, tx_lsetup, tx_lhold, tx_40us. Presentan la misma forma que los estados de conf_... Aunque estos estados podrían ser los mismos, se han separado para no complicar la lógica de estado siguiente.

Descripción de los bloques.

A continuación se describe en detalle la estructura y funcionalidad de los módulos que componen el diseño.

Módulo 3

Descripción Funcional

Este módulo se encarga principalmente de conectar los distintos subsistemas que forman el proyecto.

- LCD_control.
- Control_interfaz.
- Kb_test.

También visualiza en los 8 leds, las siguientes señales para comprobar su funcionamiento.
 leds <= state_fsmd & btn_level & rotary_izq;

Tabla 6. Conexión de los leds de la placa.

| Led | Descripción |
|--------|--|
| Led(7) | state_fsmd(1). Estado de LCD_control. |
| Led(6) | state_fsmd(0). Estado de LCD_control. |
| Led(5) | rotary_press. |
| Led(4) | btn_north. |
| Led(3) | btn_east. |
| Led(2) | btn_south. |
| Led(1) | btn_west. |
| Led(0) | rotary_izq. Se enciende cuando se gira a la izquierda. |

Señales de E/S

Las señales de entrada salida se describen en la Tabla 7. Conexiones entrada salida.

Tabla 7. Conexiones entrada salida

| Puerto | Sentido | Ancho | Función | Placa |
|-----------|---------|-------|--|-------|
| clk | IN | 1 | Reloj(50MHz) | C9 |
| modo | IN | 3 | Switches de modo Sw0 – edición(0), visualización(1) Sw1 – cursor, sin avance(0), avanza(1) | |
| pb | IN | 4 | Pulsadores | |
| rot_press | IN | 1 | Pulsador del botón rotatorio | |
| rot_a | IN | 1 | Interruptor Rotatorio | |
| rot_b | IN | 1 | Interruptor Rotatorio | |
| ps2c | IN | 1 | Señal de reloj PS/2 | |
| ps2d | IN | 1 | Señal de datos PS/2 | |
| rx | IN | 1 | Recepción UART | |
| tx | OUT | 1 | Transmisión UART | |
| lcd_db | OUT | 4 | Comando/Dato al LCD | |
| lcd_e | OUT | 1 | Habilitación del LCD | |
| lcd_rs | OUT | 1 | Indica comando(0) ó dato(1) | |
| lcd_rw | OUT | 1 | Indica escritura(0), lectura(1) | |
| lcd_CEO | OUT | 1 | Señal para usar el LCD | |
| leds | OUT | 8 | Visualización de estados | |

Símbolo

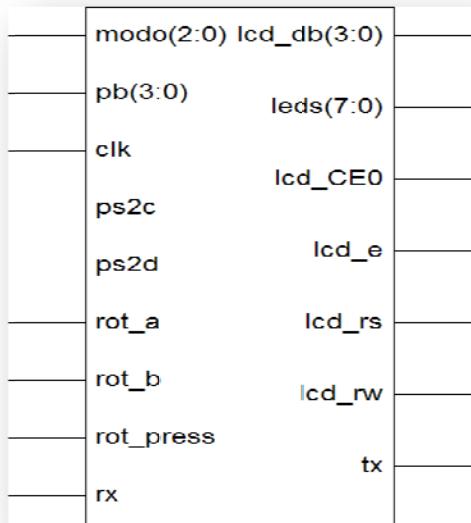


Figura 8. Símbolo del Modulo3

Descripción VHDL y plantillas para su utilización

A continuación se presenta el esquemático del circuito que se quiere describir.

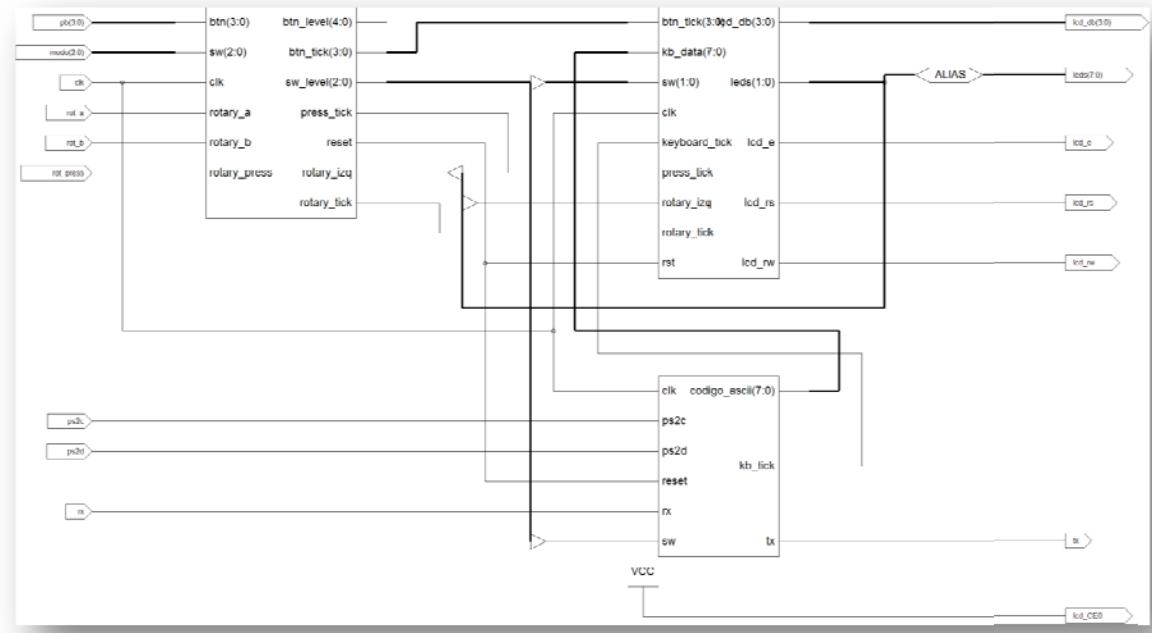


Figura 9. Esquemático del Módulo3. LCD_control, Control_interfaz y kb_test

Su descripción VHDL se presenta a continuación:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Modulo3 is
    Port ( clk : in STD_LOGIC;
           modo : in STD_LOGIC_vector(2 downto 0);
           pb : in STD_LOGIC_VECTOR (3 downto 0);
           rot_press : in STD_LOGIC;
           rot_a : in STD_LOGIC;
           rot_b : in STD_LOGIC;
           ps2c, ps2d : in std_logic;
           rx : in std_logic;
           tx : out std_logic;
           lcd_db : out STD_LOGIC_VECTOR (3 downto 0);
           lcd_e : out STD_LOGIC;
           lcd_rs : out STD_LOGIC;
           lcd_rw : out STD_LOGIC;
           lcd_ce0: out std_logic;
           leds : out STD_LOGIC_VECTOR (7 downto 0));
end Modulo3;

architecture Behavioral of Modulo3 is
    -----
    --Señales
    -----
    signal reset: std_logic;
    signal btn_tick: std_logic_vector (3 downto 0);
    signal btn_level: std_logic_vector (4 downto 0);
    signal press_tick: std_logic;
    signal rotary_izq, rotary_tick: std_logic;
    signal sw_level: std_logic_vector(2 downto 0);
    signal lcd_data: std_logic_vector (3 downto 0);
    signal ascii_code: std_logic_vector(7 downto 0);
    signal kb_tick: std_logic;
    signal state_fsm: std_logic_vector(1 downto 0);
    -----
    --Componentes
    -----
COMPONENT Control_interfaz
PORT(
    clk : IN std_logic;
    sw : IN std_logic_vector(2 downto 0);
    btn : IN std_logic_vector(3 downto 0);
    rotary_a : IN std_logic;
    rotary_b : IN std_logic;
    rotary_press : IN std_logic;
    reset : OUT std_logic;
    btn_level : OUT std_logic_vector(4 downto 0);
    btn_tick : OUT std_logic_vector(3 downto 0);
    sw_level : OUT std_logic_vector(2 downto 0);
    press_tick : OUT std_logic;
    rotary_tick : OUT std_logic;
    rotary_izq : OUT std_logic
);
END COMPONENT;

COMPONENT LCD_Control
PORT(
    clk : IN std_logic;
    rst : IN std_logic;
    sw : IN std_logic_vector(1 downto 0);
    btn_tick : IN std_logic_vector(3 downto 0);
    press_tick : IN std_logic;
    keyboard_tick : IN std_logic;
    rotary_tick : IN std_logic;
    rotary_izq : IN std_logic;
    kb_data : IN std_logic_vector(7 downto 0);
    lcd_db : OUT std_logic_vector(3 downto 0);
    lcd_e : OUT std_logic;
    lcd_rs : OUT std_logic;
    lcd_rw : OUT std_logic;

```

Edición de Mensajes en LCD

```
        leds : OUT std_logic_vector(1 downto 0)
    );
END COMPONENT;

COMPONENT kb_test
PORT(
    clk : IN std_logic;
    reset : IN std_logic;
    ps2d : IN std_logic;
    ps2c : IN std_logic;
    sw : in std_logic;
    rx : in std_logic;
    kb_tick : out STD_LOGIC;
    codigo_ascii: out std_logic_vector(7 downto 0);
    tx : OUT std_logic
);
END COMPONENT;

begin

Control_interfaz_unit: Control_interfaz PORT MAP(
    clk => clk,
    sw => modo,
    btn => pb,
    rotary_a => rot_a,
    rotary_b => rot_b,
    rotary_press => rot_press,
    reset => reset,
    btn_level => btn_level,
    btn_tick => btn_tick,
    sw_level => sw_level,
    press_tick => press_tick,
    rotary_tick => rotary_tick,
    rotary_izq => rotary_izq
);

LCD_Control_unit: LCD_Control PORT MAP(
    clk => clk,
    rst => reset,
    sw => sw_level(1 downto 0),
    btn_tick => btn_tick,
    press_tick => press_tick,
    keyboard_tick => kb_tick,
    rotary_tick => rotary_tick,
    rotary_izq => rotary_izq,
    kb_data => ascii_code,
    lcd_db => lcd_data,
    lcd_e => lcd_e,
    lcd_rs => lcd_rs,
    lcd_rw => lcd_rw,
    leds => state_fsmd
);

kb_test_unit: kb_test PORT MAP(
    clk => clk,
    reset => reset,
    ps2d => ps2d,
    ps2c => ps2c,
    rx => rx,
    sw => sw_level(2),
    kb_tick => kb_tick,
    codigo_ascii => ascii_code,
    tx => tx
);

=====
--Logica
=====
leds <= state_fsmd & btn_level & rotary_izq;
lcd_db <= lcd_data;
lcd_CE0 <='1';

end Behavioral;
```

Características temporales

Timing Summary:

Speed Grade: -4

Minimum period: 15.419ns (Maximum Frequency: 64.855MHz)

Minimum input arrival time before clock: 14.974ns

Maximum output required time after clock: 20.079ns

Maximum combinational path delay: No path found

Recursos utilizados de la FPGA

Final Results

RTL Top Level Output File Name : Modulo3.ngr

Top Level Output File Name : Modulo3

Output Format : NGC

Optimization Goal : Speed

Keep Hierarchy : NO

Design Statistics

IOs : 31

Cell Usage :

BELS : 1896

GND : 1

INV : 193

LUT1 : 37

LUT2 : 47

LUT2_D : 10

LUT2_L : 5

LUT3 : 234

LUT3_D : 14

LUT3_L : 15

LUT4 : 703

LUT4_D : 19

LUT4_L : 25

MUXCY : 268

MUXF5 : 78

MUXF6 : 16

MUXF7 : 8

VCC : 1

XORCY : 222

FlipFlops/Latches : 574

FD : 7

FDC : 291

FDCE : 256

FDE : 1

FDP : 10

FDPE : 8

FDR : 1

```
# Clock Buffers      : 1
#  BUFGP           : 1
# IO Buffers        : 30
#  IBUF            : 13
#  OBUF            : 17
```

Device utilization summary:

Selected Device : 3s500efg320-4

| | | |
|-----------------------------|------------------|-----|
| Number of Slices: | 699 out of 4656 | 15% |
| Number of Slice Flip Flops: | 574 out of 9312 | 6% |
| Number of 4 input LUTs: | 1302 out of 9312 | 13% |
| Number of IOs: | 31 | |
| Number of bonded IOBs: | 31 out of 232 | 13% |
| Number of GCLKs: | 1 out of 24 | 4% |

LCD_control

Descripción Funcional

Este modulo realiza la función de controlador del LCD. Se encarga de gestionar las peticiones del usuario y enviar los comandos oportunos al LCD. También inicializa y configura el LCD al comienzo y después de cada reset del sistema. Para realizar la configuración y la presentación inicial utiliza un contador n_reg, y un multiplexor para enviar los comandos oportunos de forma secuencial.

Cada pulsador (norte, este, sur, oeste) tiene asignado un registro de 8 bits que comienza en el código ASCII de su primer carácter. Este se incrementa con una señal de incremento y cuando llega al último carácter comienza de nuevo en el primer carácter. Por ejemplo, el botón norte recorre los caracteres del abecedario en mayúsculas, es decir de la "A" a la "Z". El código ASCII de la "A" es X"41" y el de la "Z" es X"5A". La señal de incremento para el botón norte es bttn_en y cuando se activa se incrementa el registro bttn_reg. Recordando que se usan los sufijos _reg y _next para indicar el valor registrado y de estado siguiente, el código VHDL que implementa esta función es:

```
bttn_next <= X"41" when (bttn_en='1' and bttn_reg=X"5A") else
    bttn_reg + 1 when bttn_en='1' else
    bttn_reg;
```

La máquina de estado realiza inicialmente la inicialización del LCD para una interfaz de 4 bits. Seguidamente configura el display y muestra el mensaje de bienvenida inicial. Luego pasa a un estado modo que es el que decodifica las señales que se le envía a través de los pulsadores, rotatorio y teclado y envía los comandos oportunos al LCD. A continuación se muestra el diagrama ASMD.

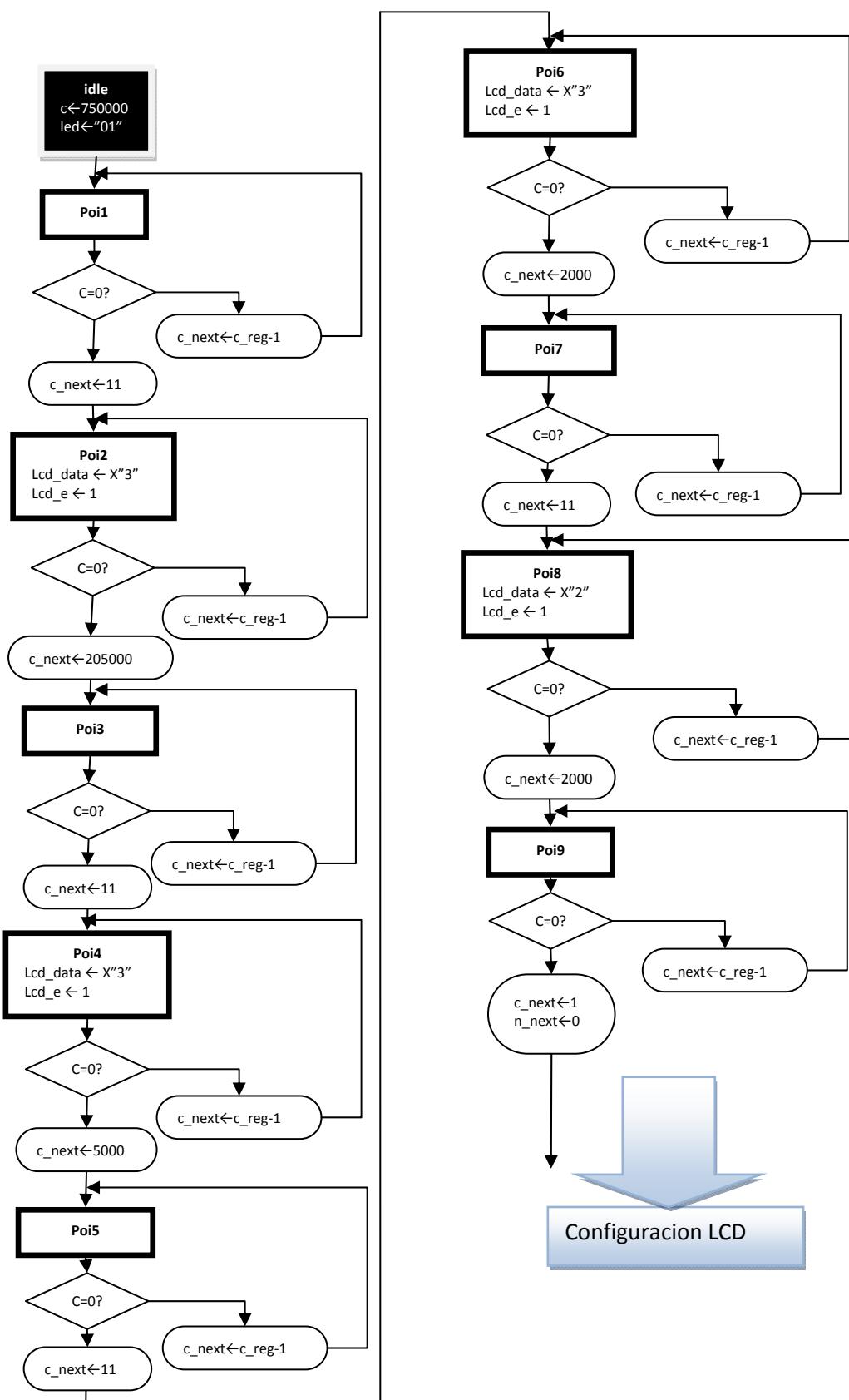


Figura 10. Diagrama ASMD de la inicialización

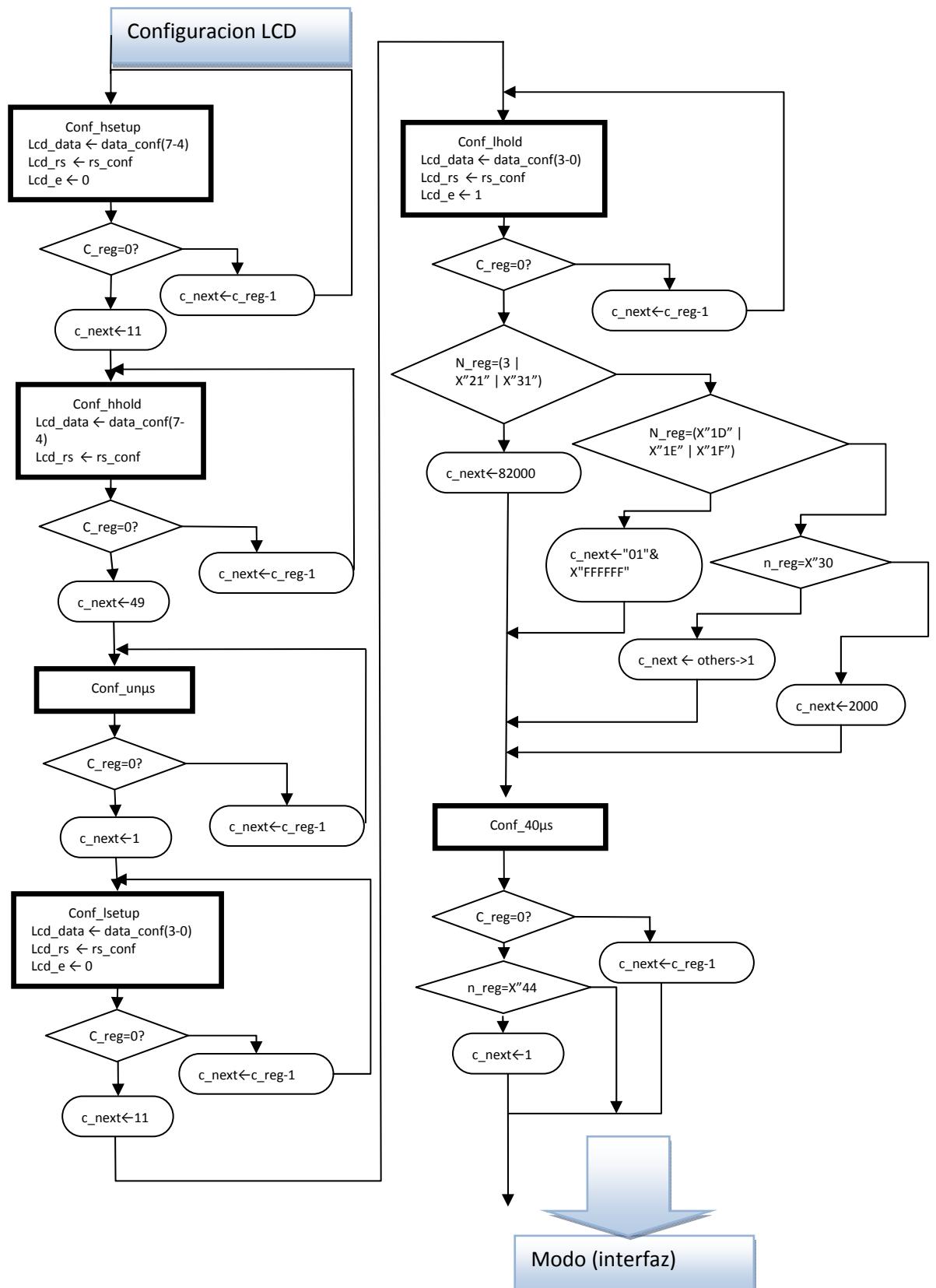


Figura 11. Diagrama ASMD de la Configuración y Bienvenida inicial

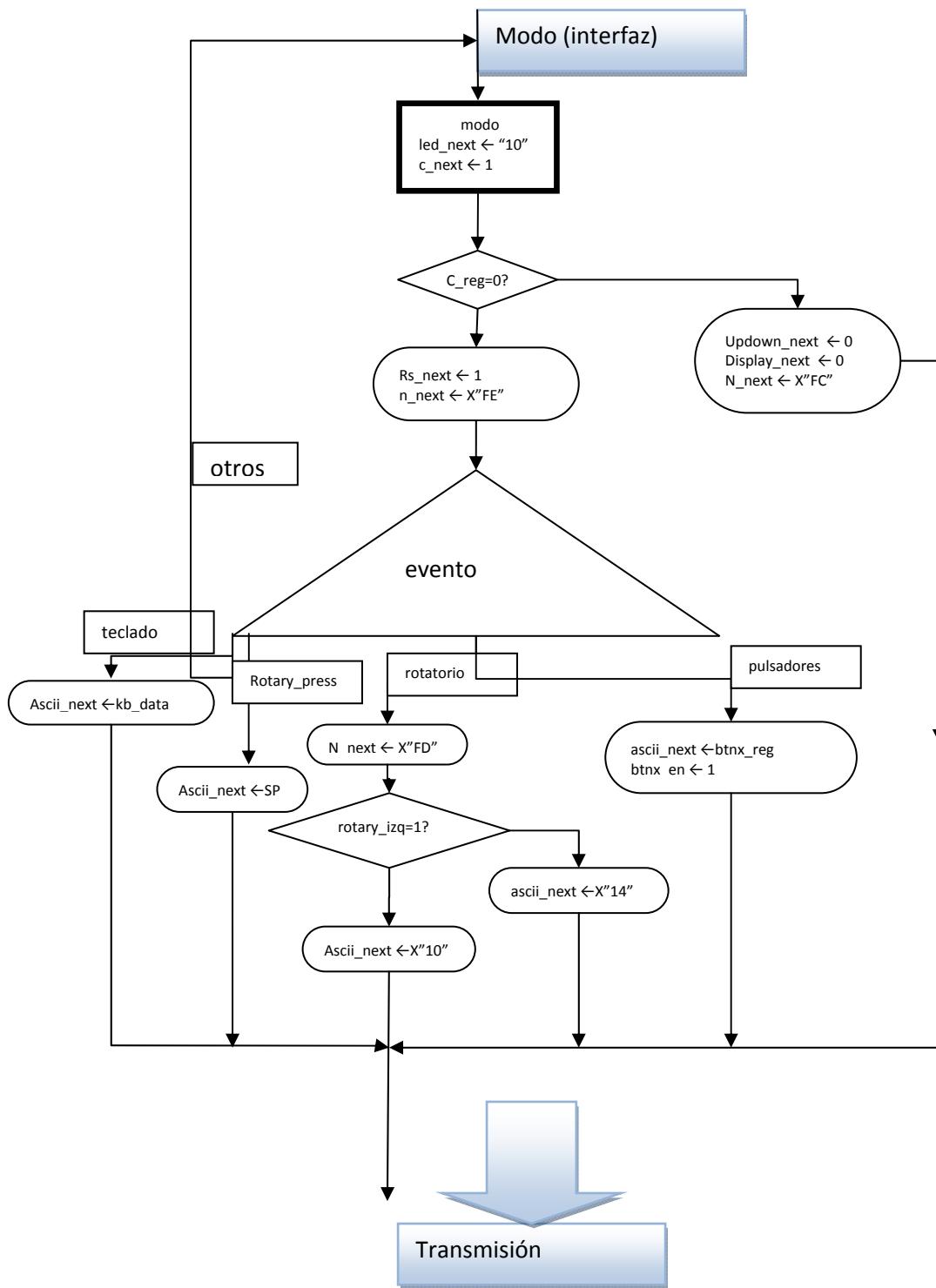


Figura 12. Diagrama ASMD del estado Modo. Decodificación de las peticiones del usuario

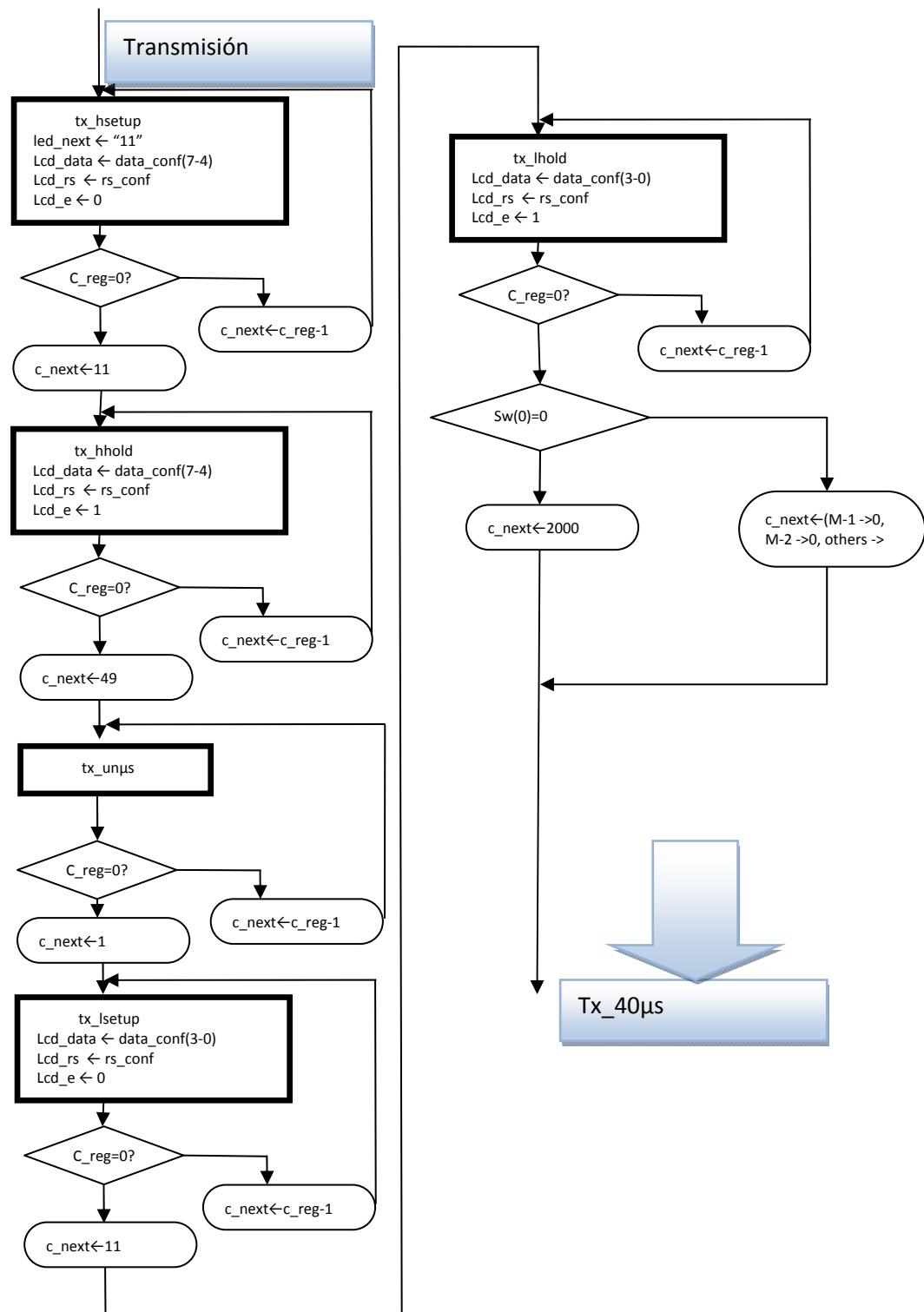


Figura 13. Diagrama ASMD de la transmisión del comando, sin el estado tx_40us.

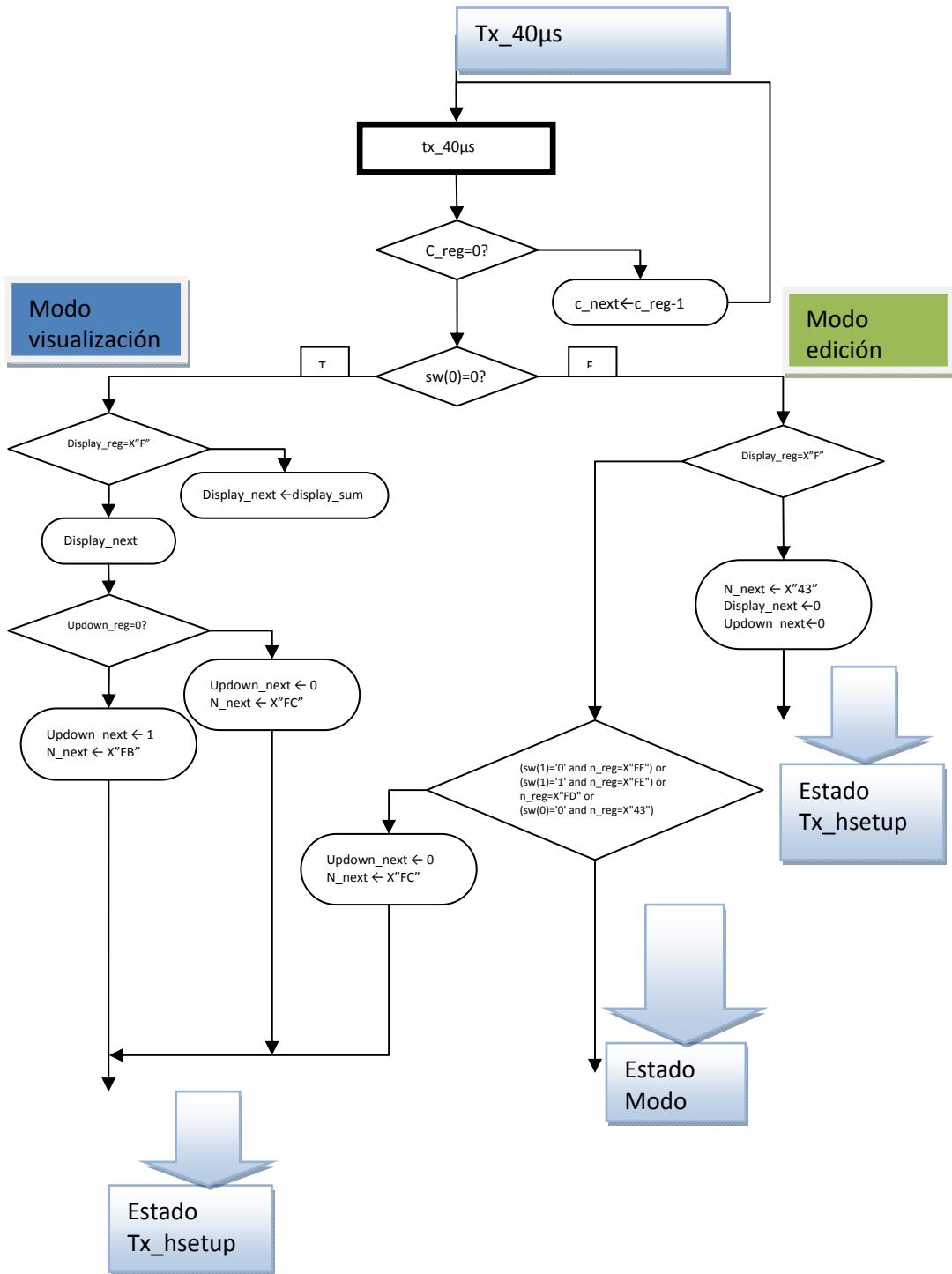


Figura 14. Diagrama ASMD del estado tx_40us

Señales de E/S

Tabla 8. LCD_control

| Puerto | Sentido | Ancho | Función |
|---------------|---------|-------|---|
| clk | IN | 1 | Reloj(50MHz) |
| rst | IN | 1 | Reset |
| sw | IN | 2 | Switches de modo |
| btn_tick | IN | 4 | Evento en pulsadores |
| press_tick | IN | 1 | Evento en pulsador rotatorio |
| Keyboard_tick | IN | 1 | Evento en teclado |
| rotary_tick | IN | 1 | Evento en rotatorio |
| rotary_izq | IN | 1 | Indica dirección de giro 0-> giro a derechas 1-> giro a izquierda |
| kb_data | IN | 8 | ASCII en mayúsculas teclado |
| Lcd_db | OUT | 4 | Comando/Datos LCD |
| lcd_e | OUT | 1 | Habilitación del LCD |
| lcd_rs | OUT | 1 | Indica comando(0) ó dato(1) |
| lcd_rw | OUT | 1 | Indica escritura(0), lectura(1) |
| leds | OUT | 2 | Estados de la FSMD |

Símbolo

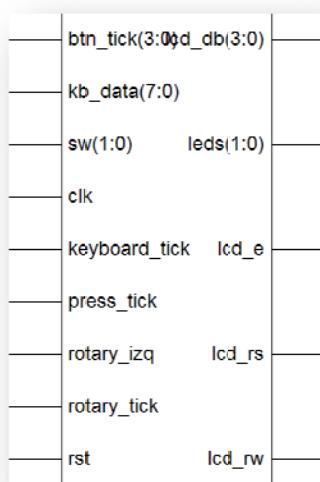


Figura 15. LCD_control

Descripción VHDL y plantillas para su utilización

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
--use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity LCD_Control is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           sw : in std_logic_vector(1 downto 0);
           btn_tick : in STD_LOGIC_VECTOR (3 downto 0);
           press_tick : in STD_LOGIC;
           keyboard_tick:in std_logic;
           rotary_tick : in STD_LOGIC;
           rotary_iqz : in STD_LOGIC;
           kb_data: in std_logic_vector(7 downto 0);
           lcd_db : out STD_LOGIC_VECTOR (3 downto 0);
           lcd_e : out STD_LOGIC;
           lcd_rs : out STD_LOGIC;
           lcd_rw : out STD_LOGIC;
           leds: out std_logic_vector(1 downto 0));
end LCD_Control;

architecture Behavioral of LCD_Control is
    constant N: natural :=8; --Número de bits para el contador de palabras de configuracion
    constant M: natural := 26; --Número de bits para el contador de tiempos
    constant SP: std_logic_vector(7 downto 0) :=X"20"; -- Espacio en blanco en ASCII
    type state_type is (idle, poi1, poi2, poi3, poi4, poi5, poi6, poi7, poi8, poi9,
                        conf_hsetup,conf_hhold, conf_unus,
                        conf_lsetup,conf_lhold,conf_40us,
                        modo,
                        tx_hsetup,tx_hhold, tx_unus,
                        tx_lsetup,tx_lhold,tx_40us);
    signal state_reg, state_next : state_type;
    signal c_reg, c_next, c_dec: unsigned(M-1 downto 0);
    signal n_reg, n_next, n_inc: unsigned(N-1 downto 0);
    signal lcd_data : std_logic_vector(3 downto 0);
    signal led_reg, led_next : unsigned(1 downto 0);
    signal data_conf: std_logic_vector(7 downto 0);
    signal rs_conf,rs_reg, rs_next: std_logic;
    signal btnn_reg, btne_reg, btns_reg, btnw_reg: unsigned(7 downto 0);
    signal btnn_next, btne_next, btns_next, btnw_next: unsigned(7 downto 0);
    signal btnn_en, btne_en, btns_en, btnw_en: std_logic;
    signal evento: std_logic_vector(6 downto 0);
    signal ascii_reg, ascii_next: std_logic_vector(7 downto 0);
    signal display_reg, display_next, display_sum: unsigned(3 downto 0); -- display_rest: unsigned(3 downto 0);
    signal updown_reg, updown_next: std_logic;
    --signal caracter : character;
begin
    ****
    --
    =====
    evento <= keyboard_tick & rotary_tick & press_tick & btn_tick;
    ****
    -- Logica secuencial FSMD
    =====
    --Estados FSM y registros
    process(clk,rst)
    begin
        if rst='1' then
            state_reg <= idle;
            c_reg <= (others => '0' );
            n_reg <= (others => '1' );
            led_reg <= (others => '0');
            btnn_reg <= X"41";
            btne_reg <= X"30";
            btns_reg <= X"61";
            btnw_reg <= X"20";
            ascii_reg <= (others => '0');
            rs_reg <='0';
            display_reg <= (others => '0');

```

```

        updown_reg <= '0';
      elsif (clk'event and clk='1') then
        state_reg <= state_next;
        c_reg <= c_next;
        n_reg <= n_next;
        led_reg <= led_next;
        bttn_reg <= bttn_next;
        btne_reg <= btne_next;
        btsn_reg <= btsn_next;
        btnw_reg <= btnw_next;
        ascii_reg <= ascii_next;
        rs_reg <= rs_next;
        display_reg <= display_next;
        updown_reg <= updown_next;
      end if;
    end process;
    =====
    -- Logica de estado siguiente para los botones
    =====
    bttn_next <= X"41" when (bttn_en='1' and bttn_reg= X"5A") else
      bttn_reg + 1 when bttn_en='1' else
      bttn_reg;
    btne_next <= X"30" when (btne_en='1' and btne_reg= X"39") else
      btne_reg + 1 when btne_en='1' else
      btne_reg;
    btsn_next <= X"61" when (btsn_en='1' and btsn_reg= X"7A") else
      btsn_reg + 1 when btsn_en='1' else
      btsn_reg;
    btnw_next <= X"20" when (btnw_en='1' and btnw_reg= X"2F") else
      btnw_reg + 1 when btnw_en='1' else
      btnw_reg;
    =====
    -- Multiplexor para la configuracion
    =====
    process(n_reg,ascii_reg,rs_reg)
    begin
      rs_conf <= '0';
      case n_reg is
        when X"00" =>          --Function set cmd 0x28
          data_conf <= X"28";
        when X"01" =>          --Entry Mode Set cmd 0x06
          data_conf <= X"06";
        when X"02" =>          --Display On/Off cmd 0x0C
          data_conf <= X"0C";
        when X"03" =>          --Clear display cmd 0x01, recordar esperar 1.64ms
          data_conf <= X"01";
      end case;
      =====
      -- Display configurado, ahora se escribe la bienvenida inicial
      =====
      when X"04" =>          --Caracter E cmd 0x45,RS='1'
        data_conf <= std_logic_vector(to_unsigned(character'pos('B'),8)); --Conversion de ASCII->std_logic
        rs_conf <= '1';
      when X"05" =>          --Caracter P cmd 0x50,RS='1'
        data_conf <= std_logic_vector(to_unsigned(character'pos('i'),8));
        rs_conf <= '1';
      when X"06" =>          --Caracter F cmd 0x46,RS='1'
        data_conf <= std_logic_vector(to_unsigned(character'pos('e'),8));
        rs_conf <= '1';
      when X"07" =>          --Caracter spc cmd 0x20,RS='1'
        data_conf <= std_logic_vector(to_unsigned(character'pos('n'),8));
        rs_conf <= '1';
      when X"08" =>          --Caracter S cmd 0x53,RS='1'
        data_conf <= std_logic_vector(to_unsigned(character'pos('v'),8));
        rs_conf <= '1';
      when X"09" =>          --Caracter u cmd 0x75,RS='1'
        data_conf <= std_logic_vector(to_unsigned(character'pos('e'),8));
        rs_conf <= '1';
      when X"0A" =>          --Caracter p cmd 0x70,RS='1'
        data_conf <= std_logic_vector(to_unsigned(character'pos('n'),8));
        rs_conf <= '1';
      when X"0B" =>          --Caracter e cmd 0x65,RS='1'
        data_conf <= std_logic_vector(to_unsigned(character'pos('i'),8));
        rs_conf <= '1';
      when X"0C" =>          --Caracter r cmd 0x72,RS='1'
        data_conf <= std_logic_vector(to_unsigned(character'pos('d'),8));
        rs_conf <= '1';
      when X"0D" =>          --Caracter r cmd 0x72,RS='1'
        data_conf <= std_logic_vector(to_unsigned(character'pos('o'),8));
        rs_conf <= '1';
    end process;
  end architecture;

```

```

when X"0E" =>                                --Salto a la siguiente linea
    data_conf <= X"CO";
    rs_conf <= '0';
when X"0F" =>                                --Caracter o cmd 0x6F,RS='1'
    data_conf <= std_logic_vector(to_unsigned(character'pos('E'),8));
    rs_conf <= '1';
when X"10" =>                                --Caracter n cmd 0x6E,RS='1'
    data_conf <= std_logic_vector(to_unsigned(character'pos('p'),8));
    rs_conf <= '1';
when X"11" =>                                --Caracter i cmd 0x69,RS='1'
    data_conf <= std_logic_vector(to_unsigned(character'pos('f'),8));
    rs_conf <= '1';
when X"12" =>                                --Caracter c cmd 0x63,RS='1'
    data_conf <= std_logic_vector(to_unsigned(character'pos(' '),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"13" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('S'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"14" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('u'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"15" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('p'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"16" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('e'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"17" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('r'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"18" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('s'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"19" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('o'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"1A" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('n'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"1B" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('i'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"1C" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('c'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"1D" =>
    data_conf <= X"08"; --apaga display
    rs_conf <= '0';
when X"1E" =>
    data_conf <= X"0C"; --enciende display
    rs_conf <= '0';
when X"1F" =>
    data_conf <= X"08";--apaga display
    rs_conf <= '0';
when X"20" =>
    data_conf <= X"0C"; --enciende display
    rs_conf <= '0';
when X"21" =>                                --Clear display cmd 0x01, recordar esperar 1.64ms
    data_conf <= X"01";
when X"22" =>                                --Caracter P cmd 0x50,RS='1'
    data_conf <= std_logic_vector(to_unsigned(character'pos('P'),8));
    rs_conf <= '1';
when X"23" =>                                --Caracter F cmd 0x46,RS='1'
    data_conf <= std_logic_vector(to_unsigned(character'pos('r'),8));
    rs_conf <= '1';
when X"24" =>                                --Caracter spc cmd 0x20,RS='1'
    data_conf <= std_logic_vector(to_unsigned(character'pos('a'),8));
    rs_conf <= '1';
when X"25" =>                                --Caracter S cmd 0x53,RS='1'
    data_conf <= std_logic_vector(to_unsigned(character'pos('c'),8));
    rs_conf <= '1';
when X"26" =>                                --Caracter u cmd 0x75,RS='1'
    data_conf <= std_logic_vector(to_unsigned(character'pos('t'),8));
    rs_conf <= '1';
when X"27" =>                                --Caracter p cmd 0x70,RS='1'
    data_conf <= std_logic_vector(to_unsigned(character'pos('i'),8));
    rs_conf <= '1';
when X"28" =>                                --Caracter e cmd 0x65,RS='1'
    data_conf <= std_logic_vector(to_unsigned(character'pos('c'),8));
    rs_conf <= '1';

```

```

when X"29" =>           --Caracter r cmd 0x72,RS='1'
    data_conf <= std_logic_vector(to_unsigned(character'pos('a'),8));
    rs_conf <= '1';
when X"2A" =>           --Caracter r cmd 0x72,RS='1'
    data_conf <= std_logic_vector(to_unsigned(character'pos(' '),8));
    rs_conf <= '1';
when X"2B" =>           --Caracter r cmd 0x72,RS='1'
    data_conf <= std_logic_vector(to_unsigned(character'pos('3'),8));
    rs_conf <= '1';
when X"2C" =>           --Salto a la siguiente linea'
    data_conf <= X"C0";
    rs_conf <= '0';
when X"2D" =>           --Caracter o cmd 0x6F,RS='1'
    data_conf <= std_logic_vector(to_unsigned(character'pos('D'),8));
    rs_conf <= '1';
when X"2E" =>           --Caracter n cmd 0x6E,RS='1'
    data_conf <= std_logic_vector(to_unsigned(character'pos('C'),8));
    rs_conf <= '1';
when X"2F" =>           --Caracter i cmd 0x69,RS='1'
    data_conf <= std_logic_vector(to_unsigned(character'pos('S'),8));
    rs_conf <= '1';
when X"30" =>           --Caracter c cmd 0x63,RS='1'
    data_conf <= std_logic_vector(to_unsigned(character'pos('E'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"31" =>           --Clear display cmd 0x01, recordar esperar 1.64ms
    data_conf <= X"01";
when X"32" =>           --Salto a la siguiente linea'
    data_conf <= X"C0";
when X"33" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('0'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"34" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('1'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"35" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('2'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"36" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('3'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"37" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('4'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"38" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('5'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"39" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('6'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"3A" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('7'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"3B" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('8'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"3C" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('9'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"3D" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('A'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"3E" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('B'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"3F" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('C'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"40" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('D'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"41" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('E'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"42" =>
    data_conf <= std_logic_vector(to_unsigned(character'pos('F'),8)); --Conversion de ASCII->std_logic
    rs_conf <= '1';
when X"43" =>           --Regreso al comienzo'
    data_conf <= X"02";
when X"44" =>           --Muestra cursor
    data_conf <= X"0E";

```

```

when X"45" =>                      --Cursor
    data_conf <= X"10";
--*****
when X"FB" =>                      --Display left
    data_conf <= X"1C";
when X"FC" =>                      --Display derecha
    data_conf <= X"18";
when X"FD" =>                      --Cursor display
    data_conf <= ascii_reg;
when X"FE" =>                      --Datos de entrada
    data_conf <= ascii_reg;
    rs_conf <= rs_reg;
when X"FF" =>                      --Cursor una posicion a la izq
    data_conf <= X"10";
when others =>
    data_conf <= X"00";
end case;
end process;
--*****
--logica de estado siguiente & salidas
=====
lcd_db <= lcd_data;
leds <= std_logic_vector(led_reg);
c_dec <= c_reg - 1;
n_inc <= n_reg + 1;
display_sum <= display_reg + 1;
display_rest <= display_reg - 1;
process(state_reg,c_reg,led_reg,c_dec,n_reg,n_inc,data_conf,rs_conf,sw,display_reg,display_sum,--display_rest,
        updown_reg,evento,ascii_reg,rs_reg,btnn_reg,btne_reg,btns_reg,btnw_reg,kb_data,rotary_izq)
begin
    state_next <= state_reg;
    c_next <= c_reg;
    n_next <= n_reg;
    lcd_data <= (others => '0');
    lcd_e <= '0';
    lcd_rs <= '0';
    lcd_rw <= '0';
    btnn_en <= '0';
    btne_en <= '0';
    btns_en <= '0';
    btnw_en <= '0';
    led_next <= led_reg;
    ascii_next <= ascii_reg;
    rs_next <= rs_reg;
    display_next <= display_reg;
    updown_next <= updown_reg;
    case state_reg is
        when idle =>
            c_next <= to_unsigned(750000,M);--750000
            state_next <= poi1;
            led_next <= "01";
--Power ON
--=====
        when poi1 =>
            if (c_reg=0) then
                if (c_reg=0) then
                    state_next <= poi2;
                    c_next <= to_unsigned(11,M);
                else
                    c_next <= c_dec;
                end if;
            when poi2 =>
                lcd_data <= X"3";
                lcd_e <= '1';
                if c_reg=0 then
                    state_next <= poi3;
                    c_next <= to_unsigned(205000,M);--205000
                else
                    c_next <= c_dec;
                end if;
            when poi3 =>
                if (c_reg=0) then
                    state_next <= poi4;
                    c_next <= to_unsigned(11,M);
                else
                    c_next <= c_dec;
                end if;
            when poi4 =>
                lcd_data <= X"3";

```

```

lcd_e <= '1';
if c_reg=0 then
    state_next <= poi5;
    c_next <= to_unsigned(5000,M)--5000
else
    c_next <= c_dec;
end if;
when poi5 =>
    if (c_reg=0) then
        state_next <= poi6;
        c_next <= to_unsigned(11,M);
    else
        c_next <= c_dec;
    end if;
when poi6 =>
    lcd_data <= X"3";
    lcd_e <= '1';
    if c_reg=0 then
        state_next <= poi7;
        c_next <= to_unsigned(2000,M)--2000
    else
        c_next <= c_dec;
    end if;
when poi7 =>
    if (c_reg=0) then
        state_next <= poi8;
        c_next <= to_unsigned(11,M);
    else
        c_next <= c_dec;
    end if;
when poi8 =>
    lcd_data <= X"2";
    lcd_e <= '1';
    if c_reg=0 then
        state_next <= poi9;
        c_next <= to_unsigned(2000,M)--2000
    else
        c_next <= c_dec;
    end if;
when poi9 =>
    if (c_reg=0) then
        state_next <= conf_hsetup;
        c_next <= to_unsigned(1,M);
        n_next <= (others=>'0');
    else
        c_next <= c_dec;
    end if;
=====
--Configuracion LCD
=====
when conf_hsetup =>
    lcd_data <= data_conf(7 downto 4);
    lcd_rs <= rs_conf;
    lcd_e <= '0';
    if c_reg=0 then
        state_next <= conf_hhold;
        c_next <= to_unsigned(11,M);
    else
        c_next <= c_dec;
    end if;
when conf_hhold =>
    lcd_data <= data_conf(7 downto 4);
    lcd_rs <= rs_conf;
    lcd_e <= '1';
    if c_reg=0 then
        state_next <= conf_unus;
        c_next <= to_unsigned(49,M)--1us
    else
        c_next <= c_dec;
    end if;
when conf_unus =>
    if c_reg=0 then
        state_next <= conf_lsetup;
        c_next <= to_unsigned(1,M);
    else
        c_next <= c_dec;
    end if;
when conf_lsetup =>
    lcd_data <= data_conf(3 downto 0);
    lcd_rs <= rs_conf;

```

```

lcd_e <= '0';
if c_reg=0 then
    state_next <= conf_lhold;
    c_next <= to_unsigned(11,M);
else
    c_next <= c_dec;
end if;
when conf_lhold =>
    lcd_data <= data_conf(3 downto 0);
    lcd_rs <= rs_conf;
    lcd_e <= '1';
    if c_reg=0 then
        state_next <= conf_40us;
        if ((n_reg=3) or (n_reg=X"21") or (n_reg=X"31")) then      --Espera 1.64ms cuando
            c_next <= to_unsigned(82000,M); --1.64ms
        elsif ((n_reg = X"1D") or (n_reg =X"1E") or (n_reg =X"1F") or (n_reg =X"20")) then
            c_next <= "01"& X"FFFFFF";
        elsif n_reg=X"30" then
            c_next <= (others => '1');
        else
            c_next <= to_unsigned(2000,M); --40us
        end if;
    else
        --En cualq otro caso espera 40us
        c_next <= to_unsigned(2000,M); --40us
    end if;
end if;
when conf_40us =>
    if (c_reg=0) then
        if (n_reg < X"44" ) then          -- Cuenta hasta que se inicialice el LCD
            state_next <= conf_hsetup;
            n_next <= n_inc;
        else
            state_next <= modo;
        end if;
        c_next <= to_unsigned(1,M);
    else
        c_next <= c_dec;
    end if;
=====
--Interfaz     evento <= keyboard_tick & rotary_tick & press_tick & btn_tick;
=====
when modo =>
    led_next <= "10";
    c_next <= to_unsigned(1,M);
    state_next <= tx_hsetup;
    if sw(0)=0' then      -- modo edición
        rs_next <= '1';
        n_next <= X"FE";
    case evento is
        when "1000000" => --Teclado
            ascii_next <= kb_data;
        when "0100000" => --Botón rotatorio
            n_next <= X"FD";
            if rotary_izq='1' then   -- mueve el cursor a la izq
                ascii_next <= X"10";
            else
                ascii_next <= X"14";
            end if;
        when "0010000" => --Botón rotatorio press
            ascii_next <= SP;
        when "0001000" => --Botón norte
            ascii_next <= std_logic_vector(btnn_reg);
            btnn_en <= '1';
        when "0000100" => --Botón este
            ascii_next <= std_logic_vector(btne_reg);
            btne_en <= '1';
        when "0000010" => --Botón sur
            ascii_next <= std_logic_vector(btns_reg);
            btns_en <= '1';
        when "0000001" => --Botón oeste
            ascii_next <= std_logic_vector(btnw_reg);
            btnw_en <= '1';
        when others =>
            state_next <= modo;
    end case;
else
    state_next <= modo;           -- modo visualización

```

```

        updown_next <='0';
        display_next <= X"0";
        n_next<= X"FC";
    end if;

=====
--Transmision
=====
when tx_setup =>
    led_next <= "11";
    lcd_data <= data_conf(7 downto 4);
    lcd_rs <= rs_conf;
    lcd_e <= '0';
    if c_reg=0 then
        state_next <= tx_hhold;
        c_next <= to_unsigned(11,M);
    else
        c_next <= c_dec;
    end if;
when tx_hhold =>
    lcd_data <= data_conf(7 downto 4);
    lcd_rs <= rs_conf;
    lcd_e <= '1';
    if c_reg=0 then
        state_next <= tx_unus;
        c_next <= to_unsigned(49,M); --1us
    else
        c_next <= c_dec;
    end if;
when tx_unus =>
    if c_reg=0 then
        state_next <= tx_lsetup;
        c_next <= to_unsigned(1,M);
    else
        c_next <= c_dec;
    end if;
when tx_lsetup =>
    lcd_data <= data_conf(3 downto 0);
    lcd_rs <= rs_conf;
    lcd_e <= '0';
    if c_reg=0 then
        state_next <= tx_lhold;
        c_next <= to_unsigned(11,M);
    else
        c_next <= c_dec;
    end if;
when tx_lhold =>
    lcd_data <= data_conf(3 downto 0);
    lcd_rs <= rs_conf;
    lcd_e <= '1';
    if c_reg=0 then
        state_next <= tx_40us;
        if sw(0)='0' then
            c_next <= to_unsigned(2000,M); --40us
        else
            c_next <= (M-1=>'0', M-2=>'0', others =>'1');
        end if;
    else
        c_next <= c_dec;
    end if;
when tx_40us =>
    if (c_reg=0) then
        if sw(0)='1' then
            state_next <= tx_hsetup;
            if display_reg = X"F" then
                display_next <= X"0";
                if updown_reg='0' then --cuenta ascendente pasa a
descendente
                    updown_next<='1';
                    n_next <= X"FB";
                else
                    updown_next<='0';
                    n_next <= X"FC";
                end if;
            else
                display_next <= display_sum;
            end if;
        end if;
        if updown_reg='0' then --cuenta ascendente
--
```

```

-- if display_reg = X"F" then --ha llegado al final
-- display_next<= X"F";
-- updown_next <= '1';
-- n_next <= X"FB";
else
  display_next<= display_sum;
end if;

-- cuenta descendente
else
  if display_reg = 0 then --ha llegado al comienzo
    display_next<= X"0";
    updown_next <= '0';
    n_next <= X"FC";
  else
    display_next<= display_rest;
  end if;
end if;

-- modo edición
if display_reg /= 0 then
  n_next <= X"43";
  display_next <= X"0";
  updown_next <= '0';
  state_next <= tx_hsetup;
elsif ((sw(1)='0' and n_reg=X"FF")or (sw(1)='1' and n_reg=X"FE") or
      n_reg=X"FD" or (sw(0)='0' and n_reg=X"43")) then
  state_next <= modo;
else
  n_next <= X"FF";
  c_next <= to_unsigned(1,M);
  state_next <= tx_hsetup;
end if;
end if;
else
  c_next <= c_dec;
end if;
end case;
end process;
end Behavioral;

```

Las plantillas para poder intanciar éste componente se presentan a continuación.

```

COMPONENT LCD_Control
PORT(
  clk : IN std_logic;
  rst : IN std_logic;
  sw : IN std_logic_vector(1 downto 0);
  btn_tick : IN std_logic_vector(3 downto 0);
  press_tick : IN std_logic;
  keyboard_tick : IN std_logic;
  rotary_tick : IN std_logic;
  rotary_iqz : IN std_logic;
  kb_data : IN std_logic_vector(7 downto 0);
  lcd_db : OUT std_logic_vector(3 downto 0);
  lcd_e : OUT std_logic;
  lcd_rs : OUT std_logic;
  lcd_rw : OUT std_logic;
  leds : OUT std_logic_vector(1 downto 0)
);
END COMPONENT;

```

```

Inst_LCD_Control: LCD_Control PORT MAP(
  clk => ,
  rst => ,
  sw => ,
  btn_tick => ,
  press_tick => ,
  keyboard_tick => ,
  rotary_tick => ,
  rotary_iqz => ,
  kb_data => ,
  lcd_db => ,
  lcd_e => ,
  lcd_rs => ,
  lcd_rw => ,
  leds =>
);

```

Test-bench y simulaciones

A continuación se presenta el vector de test realizado a este módulo.

Se comprueba que el reset funciona y se comprobar que va al estado idle. Cuando se deja de aplicar el reset, la máquina empieza el proceso de inicialización con el estado poi1.

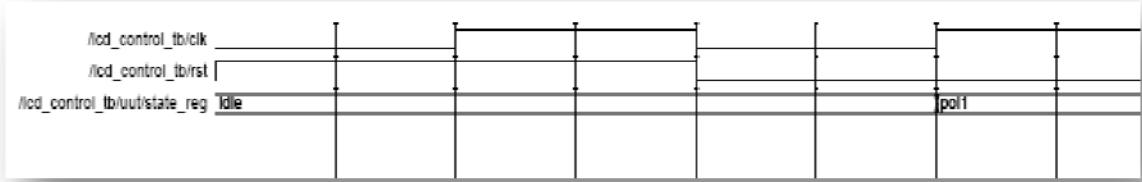


Figura 16. Reset, idle y poi1

Espera de 15ms en poi1.

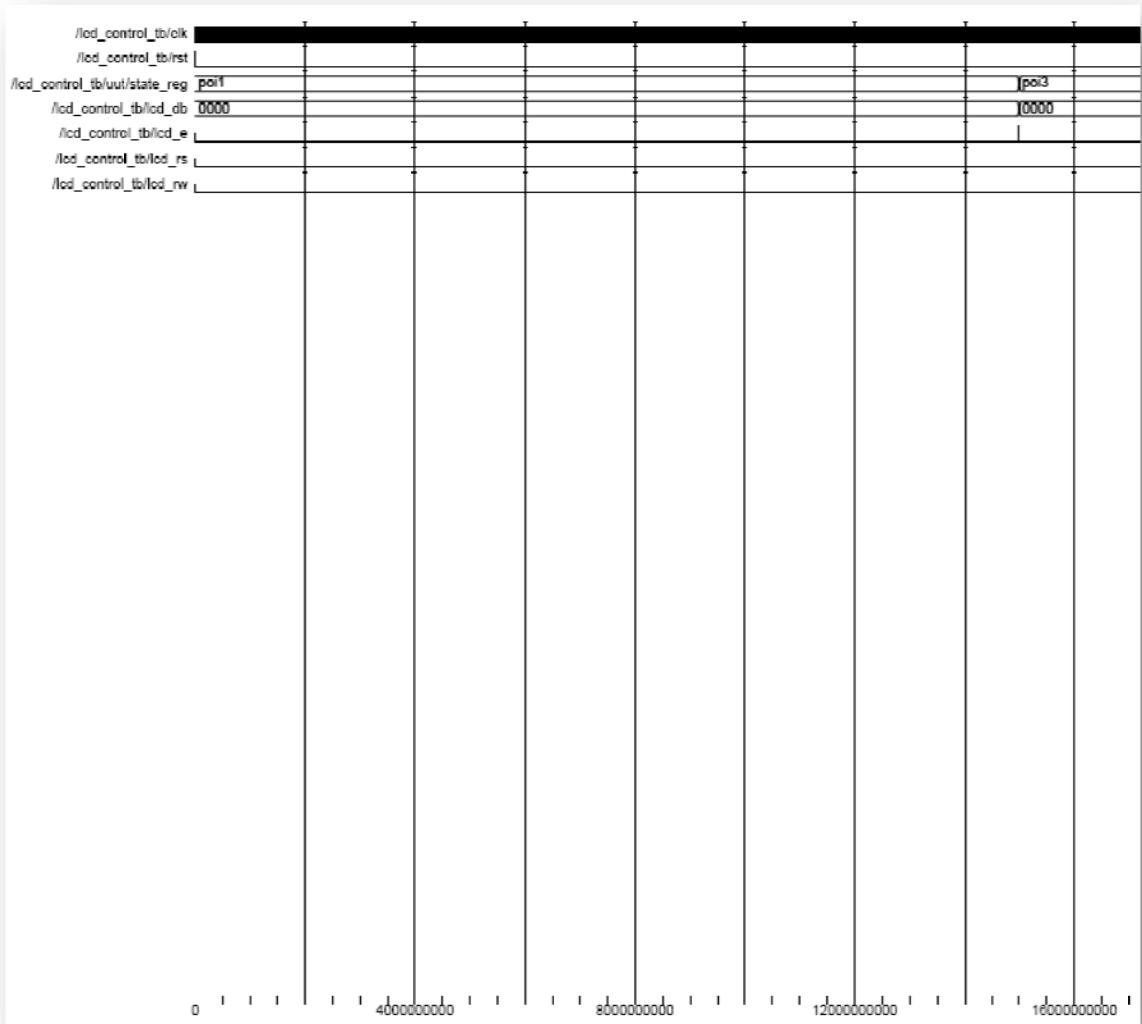


Figura 17. Espera 15ms en poi1

En poi2 se envía en LCD_DB= X"3" y lcd_e=1 durante 12 ciclos de la señal de reloj.

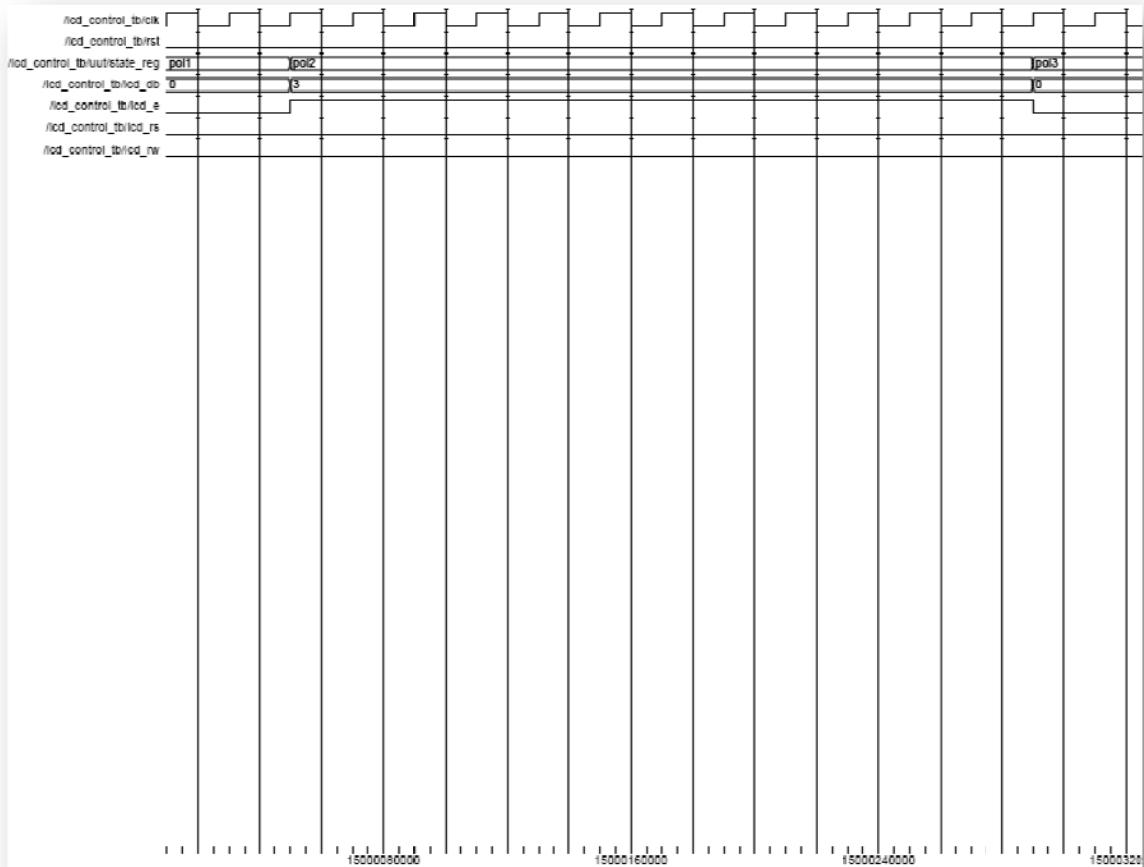


Figura 18. Envío de la primera trama de inicialización. $Lcd_db=X''3''$

Al igual que en poi4 poi6 y poi8 que se envía $Lcd_db=X''2''$.

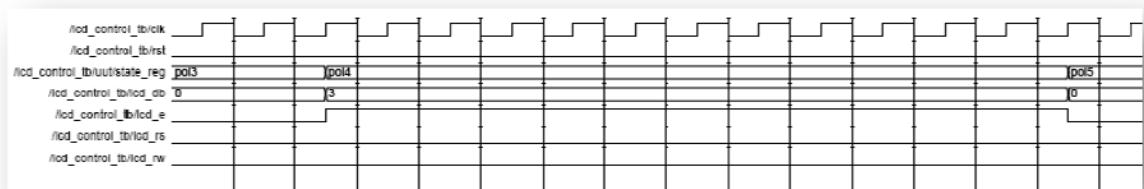


Figura 19. Envío en poi4 $Lcd_db=X''3''$

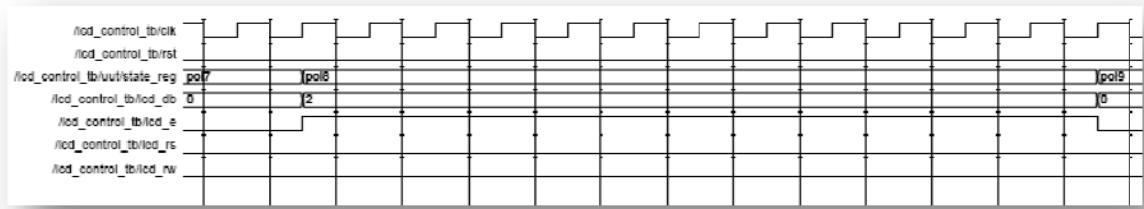


Figura 20. Envío en poi8 $Lcd_db=X''2''$

Se cumplen los tiempos en configuración. 2 ciclos de reloj en conf_hsetup y 12 en conf_hhold.

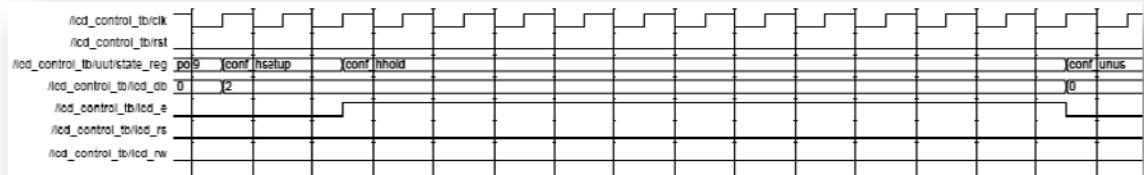


Figura 21. Envío de los 4 bits más significativos y tiempos durante el envío de configuración. 2 ciclos en conf_hsetup y 12 en conf_hhold

En conf_unus se esperan 50 ciclos de reloj

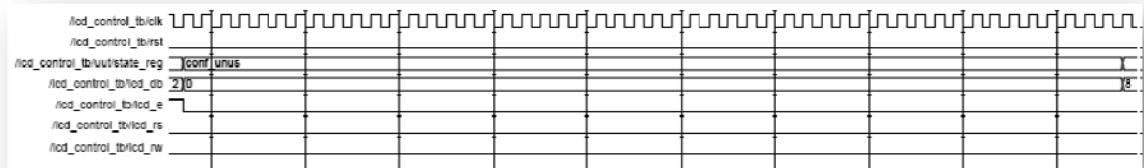


Figura 22. 1us en conf_unus

El envío de los 4 bits menos significativos

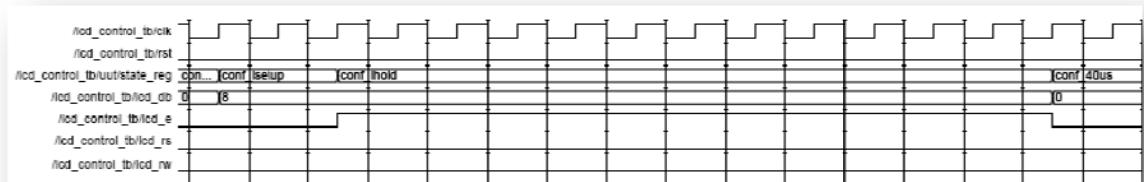


Figura 23. Envío de los 4 bits menos significativos.

Espera del estado conf_40us

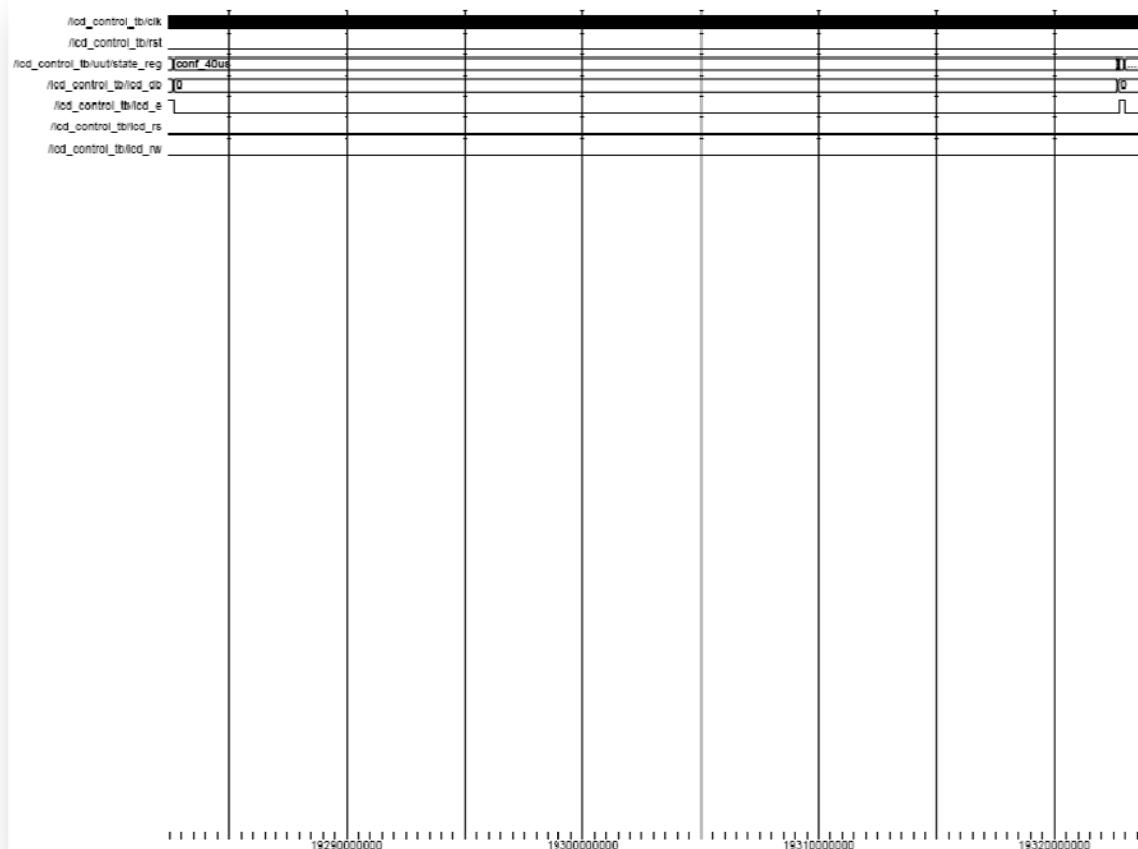


Figura 24. Estado conf_40us

Configuración y Bienvenida. Para realizar esta simulación se ha cambiado el tiempo de espera del parpadeo de la imagen. Se ha usado el tiempo de un clear 1.64ms para separar cada.

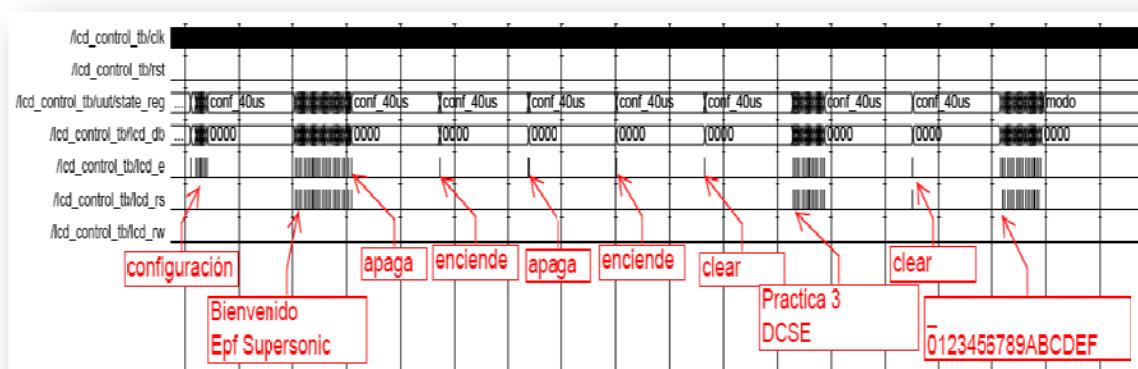


Figura 25. Visión global de la configuración y presentación inicial. Se han reducido los tiempos de espera para poder realizar la simulación.

Se presiona el botón norte.

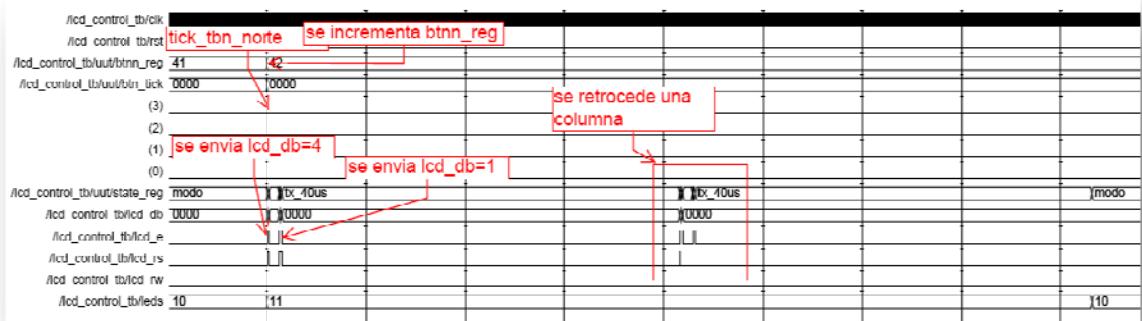


Figura 26. Se produce un evento en el botón norte, se envía el registro bttn_reg y se incrementa. Por último, si sw1=0 se envía el comando de retroceder una columna para que el cursor aparezca como si no hubiese avanzado.



Figura 27. Envío del registro bttn_reg.

Para realizar esta simulación se han reducido los tiempos en la bienvenida a 1.64ms y se ha usado el siguiente test bench

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;

ENTITY LCD_control_tb IS
END LCD_control_tb;

ARCHITECTURE behavior OF LCD_control_tb IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT LCD_Control
PORT(
    clk : IN std_logic;
    rst : IN std_logic;
    sw : IN std_logic_vector(1 downto 0);
    btn_tick : IN std_logic_vector(3 downto 0);
    press_tick : IN std_logic;
    keyboard_tick : IN std_logic;
    rotary_tick : IN std_logic;
    rotary_iqz : IN std_logic;
    kb_data : IN std_logic_vector(7 downto 0);
    lcd_db : OUT std_logic_vector(3 downto 0);
    lcd_e : OUT std_logic;
    lcd_rs : OUT std_logic;
    lcd_rw : OUT std_logic;
    leds : OUT std_logic_vector(1 downto 0)
);
END COMPONENT;

```

```

--Inputs
signal clk : std_logic := '0';
signal rst : std_logic := '0';
signal sw : std_logic_vector(1 downto 0) := (others => '0');
signal btn_tick : std_logic_vector(3 downto 0) := (others => '0');
signal press_tick : std_logic := '0';
signal keyboard_tick : std_logic := '0';
signal rotary_tick : std_logic := '0';
signal rotary_izq : std_logic := '0';
signal kb_data : std_logic_vector(7 downto 0) := (others => '0');

--Outputs
signal lcd_db : std_logic_vector(3 downto 0);
signal lcd_e : std_logic;
signal lcd_rs : std_logic;
signal lcd_rw : std_logic;
signal leds : std_logic_vector(1 downto 0);

-- Clock period definitions
constant clk_period : time := 20ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: LCD_Control PORT MAP (
        clk => clk,
        rst => rst,
        sw => sw,
        btn_tick => btn_tick,
        press_tick => press_tick,
        keyboard_tick => keyboard_tick,
        rotary_tick => rotary_tick,
        rotary_izq => rotary_izq,
        kb_data => kb_data,
        lcd_db => lcd_db,
        lcd_e => lcd_e,
        lcd_rs => lcd_rs,
        lcd_rw => lcd_rw,
        leds => leds
    );

    -- Clock process definitions
    clk_process :process
    begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
    end process;

    --rst <= '1', '0' after clk_period;
    -- Stimulus process
    stim_proc: process
    begin

        wait for 40ms;

        wait until falling_edge(clk);
        btn_tick <= "1000";      -- botón norte
        wait until falling_edge(clk);
        btn_tick <= "0000";

        wait;
    end process;

```

Recursos utilizados de la FPGA

Synthesizing Unit <LCD_Control>.

Related source file is "D:/Designs/DCSE/LCD_Spartan3E/LCD_Control.vhd".

Found finite state machine <FSM_0> for signal <state_reg>.

| | | |
|----------------|-------------------|--|
| States | 23 | |
| Transitions | 60 | |
| Inputs | 16 | |
| Outputs | 25 | |
| Clock | clk (rising_edge) | |
| Reset | rst (positive) | |
| Reset type | asynchronous | |
| Reset State | idle | |
| Power Up State | idle | |
| Encoding | automatic | |
| Implementation | LUT | |

Found 8-bit register for signal <ascii_reg>.

Found 8-bit up counter for signal <btne_reg>.

Found 8-bit up counter for signal <btnn_reg>.

Found 8-bit up counter for signal <btns_reg>.

Found 8-bit up counter for signal <btnw_reg>.

Found 26-bit subtractor for signal <c_dec>.

Found 26-bit register for signal <c_reg>.

Found 4-bit register for signal <display_reg>.

Found 4-bit adder for signal <display_reg\$addsub0000> created at line 360.

Found 2-bit register for signal <led_reg>.

Found 8-bit register for signal <n_reg>.

Found 8-bit adder for signal <n_reg\$addsub0000> created at line 359.

Found 1-bit register for signal <rs_reg>.

Found 8-bit comparator less for signal <state_reg\$cmp_lt0000> created at line 522.

Found 1-bit register for signal <updown_reg>.

Summary:

inferred 1 Finite State Machine(s).

inferred 4 Counter(s).

inferred 50 D-type flip-flop(s).

inferred 3 Adder/Subtractor(s).

inferred 1 Comparator(s).

Unit <LCD_Control> synthesized.

Control_interfaz

Descripción Funcional

Su función básica es generar una señal de evento por cada pulsador, indicar el nivel de cada switch, proporcionar información del sentido de giro del rotatorio y los eventos, y filtrar los glitches que se producen de forma mecánica.

Para filtrar e indicar eventos en los pulsadores y switches utiliza el módulo debounce que se encarga de esta función. De esta forma se instancian tantos componentes como botones y switches haya. Cada componente proporciona información acerca de su estado y si se ha

producido un evento, es decir, si ha cambiado del estado 0 al 1. En el caso de los pulsadores se utiliza las señales eventos (utilizan el sufijo _tick) y los switches utilizan el valor filtrado que proporciona el componente.

Para el caso del botón rotatorio se ha utilizado un filtro que proporciona información acerca del sentido del giro, y si se ha girado. El sentido del giro lo proporciona con una señal que si está a nivel alto indica que se ha girado a la izquierda.

Señales de E/S

Tabla 9. Control_interfaz

| Puerto | Sentido | Ancho | Función |
|---------------------|---------|-------|--|
| clk | IN | 1 | Reloj(50MHz) |
| rst | IN | 1 | Reset |
| sw | IN | 3 | Switches de modo |
| btn | IN | 4 | Botones de entrada |
| rotary_a | IN | 1 | Interruptor Rotatorio |
| rotary_b | IN | 1 | Interruptor Rotatorio |
| rotary_press | IN | 1 | Pulsador del botón rotatorio |
| reset | OUT | 1 | Reset del sistema |
| btn_level | OUT | 5 | Posición de los botones btn_level(4)->rot.press btn_level(3)->btn_north btn_level(2)->btn_east btn_level(1)->btn_south btn_level(0)->btn_west |
| btn_tick | OUT | 4 | Evento en los botones |
| sw_level | OUT | 3 | Evento en los switches |
| press_tick | OUT | 1 | Evento en el pulsador rot. |
| rotary_tick | OUT | 1 | Evento en el rotatorio |
| rotary_izq | OUT | 1 | Indica dirección de giro 0-> giro a derechas 1-> giro a izquierda |

Símbolo

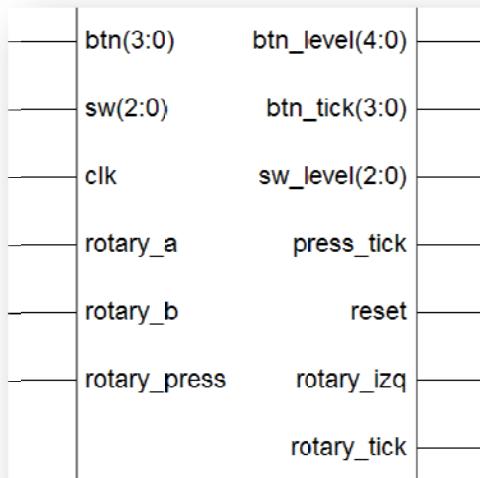


Figura 28. Símbolo control_interfaz

Descripción VHDL y plantillas para su utilización

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Control_interfaz is
  Port ( clk : in STD_LOGIC;
         sw : in STD_LOGIC_vector(2 downto 0);
         btn : in STD_LOGIC_VECTOR (3 downto 0);
         rotary_a : in STD_LOGIC;
         rotary_b : in STD_LOGIC;
         rotary_press : in STD_LOGIC;
         reset : out STD_LOGIC;
         btn_level : out STD_LOGIC_VECTOR (4 downto 0);
         btn_tick : out STD_LOGIC_VECTOR (3 downto 0);
         sw_level : out STD_LOGIC_vector(2 downto 0);
         press_tick : out STD_LOGIC;
         rotary_tick : out STD_LOGIC;
         rotary_izq : out STD_LOGIC);
end Control_interfaz;

architecture Behavioral of Control_interfaz is
  -----
  --Señales
  -----
  signal btn_press, btn_n, btn_e, btn_s, btn_w: std_logic;
  signal btn_n_tick, btn_e_tick, btn_s_tick, btn_w_tick: std_logic;
  signal sw0,sw1,sw2: std_logic;
  signal rst: std_logic;

  -- Signals used to interface to rotary encoder
  --
  signal rotary_a_in : std_logic;
  signal rotary_b_in : std_logic;
  signal rotary_in : std_logic_vector(1 downto 0);
  signal rotary_q1 : std_logic;
  signal rotary_q2 : std_logic;
  signal delay_rotary_q1 : std_logic;
  signal rotary_event : std_logic;
  signal rotary_left : std_logic;

```

```

=====
--Componentes
=====
COMPONENT debounce
PORT(
    clk : IN std_logic;
    reset : IN std_logic;
    sw : IN std_logic;
    db_level : OUT std_logic;
    db_tick : OUT std_logic
);
END COMPONENT;
begin
=====
--Reset a cero
=====
    rst <= '0';
=====
--Componentes Instanciados para botones y switches
=====
    btn_n_unit: debounce PORT MAP(
        clk => clk,
        reset => rst,
        sw => btn(3),
        db_level => btn_n,
        db_tick => btn_n_tick
    );
    btn_e_unit: debounce PORT MAP(
        clk => clk,
        reset => rst,
        sw => btn(2),
        db_level => btn_e,
        db_tick => btn_e_tick
    );
    btn_s_unit: debounce PORT MAP(
        clk => clk,
        reset => rst,
        sw => btn(1),
        db_level => btn_s,
        db_tick => btn_s_tick
    );
    btn_w_unit: debounce PORT MAP(
        clk => clk,
        reset => rst,
        sw => btn(0),
        db_level => btn_w,
        db_tick => btn_w_tick
    );
    sw0_unit: debounce PORT MAP(
        clk => clk,
        reset => rst,
        sw => sw(0),
        db_level => sw0,
        db_tick => open
    );
    sw1_unit: debounce PORT MAP(
        clk => clk,
        reset => rst,
        sw => sw(1),
        db_level => sw1,
        db_tick => open
    );
    sw2_unit: debounce PORT MAP(
        clk => clk,
        reset => rst,
        sw => sw(2),
        db_level => sw2,
        db_tick => open
    );
    btn_press_unit: debounce PORT MAP(
        clk => clk,
        reset => rst,
        sw => rotary_press,
        db_level => btn_press,
        db_tick => press_tick
    );
=====
--Logica de salida para botones y switches
=====

```

Edición de Mensajes en LCD

```
btn_level <= btn_press & btn_n & btn_e & btn_s & btn_w;
btn_tick <= btn_n_tick & btn_e_tick & btn_s_tick & btn_w_tick;
sw_level <= sw2 & sw1 & sw0;
reset <= btn_e and btn_w;

=====
--Logica para el boton rotatorio
=====
rotary_filter: process(clk)
begin
    if clk'event and clk='1' then
        --Sincronizacion de las entradas
        rotary_a_in <= rotary_a;
        rotary_b_in <= rotary_b;

        rotary_in <= rotary_b_in & rotary_a_in;

        case rotary_in is
            when "00" => rotary_q1 <= '0';
                           rotary_q2 <= rotary_q2;
            when "01" => rotary_q1 <= rotary_q1;
                           rotary_q2 <= '0';
            when "10" => rotary_q1 <= rotary_q1;
                           rotary_q2 <= '1';
            when "11" => rotary_q1 <= '1';
                           rotary_q2 <= rotary_q2;
            when others => rotary_q1 <= rotary_q1;
                           rotary_q2 <= rotary_q2;
        end case;

        end if;
    end process rotary_filter;
    --
    -- Flanco de subida en rotary_q1 indica que ha habido una rotacion
    -- y el estado de rotary_q2 indica la direccion. 1 para izq y 0 para derecha
    --
    direccion: process(clk)
    begin
        if clk'event and clk='1' then
            delay_rotary_q1 <= rotary_q1;
            if rotary_q1='1' and delay_rotary_q1='0' then
                rotary_event <= '1';
                rotary_left <= rotary_q2;
            else
                rotary_event <= '0';
                rotary_left <= rotary_left;
            end if;

            end if;
        end process direccion;
    --
    rotary_tick <= rotary_event;
    rotary_izq <= rotary_left;
end Behavioral;
```

Para su instanciación se usará:

```
COMPONENT Control_interfaz
PORT(
    clk : IN std_logic;
    sw : IN std_logic_vector(2 downto 0);
    btn : IN std_logic_vector(3 downto 0);
    rotary_a : IN std_logic;
    rotary_b : IN std_logic;
    rotary_press : IN std_logic;
    reset : OUT std_logic;
    btn_level : OUT std_logic_vector(4 downto 0);
    btn_tick : OUT std_logic_vector(3 downto 0);
    sw_level : OUT std_logic_vector(2 downto 0);
    press_tick : OUT std_logic;
```

```
    rotary_tick : OUT std_logic;
    rotary_izq : OUT std_logic
);
END COMPONENT;
Inst_Control_interfaz: Control_interfaz PORT MAP(
    clk => ,
    sw => ,
    btn => ,
    rotary_a => ,
    rotary_b => ,
    rotary_press => ,
    reset => ,
    btn_level => ,
    btn_tick => ,
```

```
sw_level=>,
press_tick=>,
rotary_tick=>,
```

```
); rotary_izq=>
```

El módulo debounce

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
--use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;
--- Uncomment the following library declaration if instantiating
--- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity debounce is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           sw : in STD_LOGIC;
           db_level : out STD_LOGIC;
           db_tick : out STD_LOGIC);
end debounce;

architecture imp_fsmd_arch of debounce is
    constant N: integer :=21; --filtro de 2^N*20ns=40ms
    type state_type is (zero, wait0, one, wait1);
    signal state_reg, state_next : state_type;
    signal q_reg, q_next : unsigned (N-1 downto 0);
    --signal q_load, q_dec, q_zero :std_logic;
begin
    --FSMD estados y registros de datos
    process(clk, reset)
    begin
        if reset='1' then
            state_reg <= zero;
            q_reg <= (others=>'0');
        elsif (clk'event and clk='1') then
            state_reg <= state_next;
            q_reg <= q_next;
        end if;
    end process;

    --FSMD camino de datos y logica de estado siguiente
    process(state_reg, sw, q_reg, q_next)
    begin
        q_next <=q_reg;
        db_tick <='0';
        state_next <= state_reg;
        case state_reg is
            when zero =>
                db_level <='0';
                if (sw='1') then
                    state_next <= wait1;
                    q_next <= (others=>'1');
                end if;
            when wait1 =>
                db_level <='0';
                if (sw='1') then
                    q_next <= q_reg - 1;
                    if (q_next=0) then
                        db_tick <='1';
                        state_next <= one;
                    end if;
                else --sw=0
                    state_next <= zero;
                end if;
            when one =>
                db_level <='1';
                if (sw='0') then
                    q_next <= (others=>'1');
                    state_next <= wait0;
                end if;
            when wait0 =>
                db_level <='1';
                if (sw='0') then
                    q_next <= q_reg -1;
                    if (q_next=0) then
                        state_next <= zero;
                    end if;
                end if;
        end case;
    end process;

```

```
        end if;
      else --sw='1'
        state_next <= one;
      end if;
    end case;
  end process;
end imp_fsmd_arch;
```

Y la plantilla para su instanciaión

```
COMPONENT debounce
PORT(
  clk : IN std_logic;
  reset : IN std_logic;
  sw : IN std_logic;
  db_level : OUT std_logic;
  db_tick : OUT std_logic
);
END COMPONENT;
```

```
Inst_debounce: debounce PORT MAP(
  clk => ,
  reset => ,
  sw => ,
  db_level => ,
  db_tick =>
);
```

Recursos utilizados de la FPGA

Synthesizing Unit <debounce>.

Related source file is "D:/Designs/DCSE/LCD_Spartan3E/debounce.vhd".

Found finite state machine <FSM_1> for signal <state_reg>.

| | | |
|----------------|-------------------|--|
| States | 4 | |
| Transitions | 10 | |
| Inputs | 2 | |
| Outputs | 5 | |
| Clock | clk (rising_edge) | |
| Reset | reset (positive) | |
| Reset type | asynchronous | |
| Reset State | zero | |
| Power Up State | zero | |
| Encoding | automatic | |
| Implementation | LUT | |

Found 21-bit subtractor for signal <q_next\$share0000> created at line 63.

Found 21-bit register for signal <q_reg>.

Summary:

inferred 1 Finite State Machine(s).

inferred 21 D-type flip-flop(s).

inferred 1 Adder/Subtractor(s).

Unit <debounce> synthesized.

Kb_test

Descripción Funcional

Este módulo es opcional, y se han seguido el libro de (Pong P. Chu) para su diseño. Éste módulo simplemente se encarga de interconectar el teclado, la unidad UART y el módulo de conversión a ASCII, key2ascii. La función de la unidad UART es la de enviar todos los caracteres recibidos del teclado a través del puerto serie. Los elementos instanciados son los siguientes:

- kb_code_unit
- uart_unit
- key2a_unit

kb_code_unit contiene un módulo receptor de ps2, un módulo que implementa una fifo para almacenar los datos recibidos en ps2 y una electrónica que le dice a la fifo que guarde

sólo el primer byte que viene detrás del código de ruptura del teclado F0. Esto se realiza con una pequeña FSM de dos estados.

Uart_unit consiste en un módulo receptor serie asíncrono, una fifo para almacenar los datos recibidos, un transmisor serie asíncrono, una fifo para almacenar los datos a enviar, y un contador que genera pulsos a una velocidad 16 veces mayor que la velocidad de comunicación.

Por último, key2a_unit, convierte los scan code del teclado en caracteres ASCII. De momento solamente están implementados las letras en mayúsculas. Esto se realiza a través de un multiplexor que con el código scan code, selecciona la entrada que contiene el código ASCII de la tecla pulsada.

Señales de E/S

Tabla 10. Kb_test

| Puerto | Sentido | Ancho | Función |
|--------------|---------|-------|--|
| clk | IN | 1 | Reloj(50MHz) |
| reset | IN | 1 | Reset |
| sw | IN | 1 | Señal de entrada |
| ps2d | IN | 1 | Datos serie del teclado |
| ps2c | IN | 1 | Reloj puerto PS2 |
| rx | IN | 1 | Recepción puerto serie |
| tx | OUT | 1 | Transmisión puerto serie |
| kb_tick | OUT | 1 | Se produce un evento por tecla pulsada |
| Código_ascii | OUT | 8 | Código ASCII de la tecla pulsada |

Símbolo

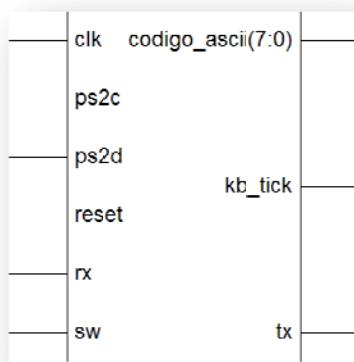


Figura 29. Kb_test

Descripción VHDL y plantillas para su utilización

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
--use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity kb_test is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           ps2d : in STD_LOGIC;
           ps2c : in STD_LOGIC;
           sw : in std_logic;
           rx : in std_logic;
           kb_tick : out STD_LOGIC;
           codigo_ascii: out std_logic_vector(7 downto 0);
           tx : out STD_LOGIC);
end kb_test;

architecture Behavioral of kb_test is
    -- signal scan_data, w_data: std_logic_vector(7 downto 0);
    signal kb_not_empty, kb_buf_empty: std_logic;
    signal key_code, ascii_code, uart_data: std_logic_vector(7 downto 0);
    signal rx_empty, rx_not_empty: std_logic;
    signal wr_uart: std_logic;

begin
    kb_code_unit: entity work.kb_code(Behavioral)
        port map(
            clk => clk,
            reset => reset,
            ps2d => ps2d,
            ps2c => ps2c,
            rd_key_code => kb_not_empty,
            key_code => key_code,
            kb_buf_empty => kb_buf_empty);

    uart_unit: entity work uart(Behavioral)
        port map(
            clk => clk,
            reset => reset,
            rd_uart => '1',
            wr_uart => wr_uart,
            rx => rx,
            w_data => ascii_code,
            tx_full => open,
            rx_empty => rx_empty,
            r_data => uart_data,
            tx => tx);

    key2a_unit: entity work.key2ascii(Behavioral)
        port map(
            key_code => key_code,
            ascii_code => ascii_code);

    rx_not_empty <= not rx_empty;
    kb_not_empty <= not kb_buf_empty;
    wr_uart <= kb_not_empty when sw='0' else rx_not_empty;
    codigo_ascii <= ascii_code when sw='0' else uart_data;
    kb_tick <= kb_not_empty when sw='0' else rx_not_empty;
end Behavioral;
```

Para su instanciación se utiliza:

```
COMPONENT kb_test
PORT(
    clk : IN std_logic;
    reset : IN std_logic;
    ps2d : IN std_logic;
    ps2c : IN std_logic;
    sw : IN std_logic;
    rx : IN std_logic;
    kb_tick : OUT std_logic;
    codigo_ascii : OUT std_logic_vector(7 downto 0);
    tx : OUT std_logic
);
END COMPONENT;
```

```
Inst_kb_test: kb_test PORT MAP(
    clk => ,
    reset => ,
    ps2d => ,
    ps2c => ,
    sw => ,
    rx => ,
    kb_tick => ,
    codigo_ascii => ,
    tx =>
);
```

El módulo kb_code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity kb_code is
    generic(W_SIZE: integer :=2);           --2^W_SIZE words in Fifo
    Port ( clk : in STD_LOGIC;
            reset : in STD_LOGIC;
            ps2d : in STD_LOGIC;
            ps2c : in STD_LOGIC;
            rd_key_code : in STD_LOGIC;
            key_code : out STD_LOGIC_VECTOR (7 downto 0);
            kb_buf_empty : out STD_LOGIC);
end kb_code;

architecture Behavioral of kb_code is
    constant BRK: std_logic_vector (7 downto 0) := "11110000";
    type statetype is (wait_brk, get_code);
    signal state_reg, state_next: statetype;
    signal scan_out: std_logic_vector(7 downto 0); --, w_data
    signal scan_done_tick, got_code_tick: std_logic;
begin
    begin
        =====
        --Instanciacion
        =====

        ps2_rx_unit: entity work.ps2_rx(Behavioral)
            port map(
                clk => clk,
                reset => reset,
                ps2d => ps2d,
                ps2c => ps2c,
                rx_en => '1',
                rx_done_tick => scan_done_tick,
                dout => scan_out);
        fifo_key_unit: entity work.fifo(Behavioral)
            port map(
                clk => clk,
                reset => reset,
                rd => rd_key_code,
                wr => got_code_tick,
                w_data => scan_out,
                empty => kb_buf_empty,
                full => open,
                r_data => key_code);

        =====
        --Instanciacion
        =====
        process(clk, reset)
        begin
            if reset = '1' then
                state_reg <= wait_brk;
            elsif (clk'event and clk='1') then
```

```

        state_reg <= state_next;
    end if;
end process;

process(state_reg, scan_done_tick, scan_out)
begin
got_code_tick <= '0';
state_next <= state_reg;
case state_reg is
    when wait_brk => --espera por el break code 0xF0
        if scan_done_tick='1' and scan_out =BRK then
            state_next <=get_code;
        end if;
    when get_code =>-- Guarda el siguiente codigo
        if scan_done_tick='1' then
            got_code_tick <= '1';
            state_next <= wait_brk;
        end if;
end case;
end process;
end Behavioral;
```

La plantilla para su utilización

```

COMPONENT kb_code
PORT(
    clk : IN std_logic;
    reset : IN std_logic;
    ps2d : IN std_logic;
    ps2c : IN std_logic;
    rd_key_code : IN std_logic;
    key_code : OUT std_logic_vector(7 downto 0);
    kb_buf_empty : OUT std_logic
);
END COMPONENT;
```

```

Inst_kb_code: kb_code PORT MAP(
    clk => ,
    reset => ,
    ps2d => ,
    ps2c => ,
    rd_key_code => ,
    key_code => ,
    kb_buf_empty =>
);
```

El módulo key2ascii

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
--use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity key2ascii is
    Port ( key_code : in STD_LOGIC_VECTOR (7 downto 0);
           ascii_code : out STD_LOGIC_VECTOR (7 downto 0));
end key2ascii;

architecture Behavioral of key2ascii is

begin
    with key_code select
        ascii_code <=
            std_logic_vector(to_unsigned(character'pos('0'),8)) when X"45", -- 0
            std_logic_vector(to_unsigned(character'pos('1'),8)) when X"16", -- 1
            std_logic_vector(to_unsigned(character'pos('2'),8)) when X"1E", -- 2
            std_logic_vector(to_unsigned(character'pos('3'),8)) when X"26", -- 3
            std_logic_vector(to_unsigned(character'pos('4'),8)) when X"25", -- 4
            std_logic_vector(to_unsigned(character'pos('5'),8)) when X"2E", -- 5
            std_logic_vector(to_unsigned(character'pos('6'),8)) when X"36", -- 6
            std_logic_vector(to_unsigned(character'pos('7'),8)) when X"3D", -- 7
            std_logic_vector(to_unsigned(character'pos('8'),8)) when X"3E", -- 8
            std_logic_vector(to_unsigned(character'pos('9'),8)) when X"46", -- 9

            std_logic_vector(to_unsigned(character'pos('A'),8)) when X"1C", -- A
            std_logic_vector(to_unsigned(character'pos('B'),8)) when X"32", -- B
            std_logic_vector(to_unsigned(character'pos('C'),8)) when X"21", -- C
            std_logic_vector(to_unsigned(character'pos('D'),8)) when X"23", -- D
            std_logic_vector(to_unsigned(character'pos('E'),8)) when X"24", -- E
            std_logic_vector(to_unsigned(character'pos('F'),8)) when X"2B", -- F
            std_logic_vector(to_unsigned(character'pos('G'),8)) when X"34", -- G
            std_logic_vector(to_unsigned(character'pos('H'),8)) when X"33", -- H
```

```

    std_logic_vector(to_unsigned(character'pos('I'),8)) when X"43", -- I
    std_logic_vector(to_unsigned(character'pos('J'),8)) when X"3B", -- J
    std_logic_vector(to_unsigned(character'pos('K'),8)) when X"42", -- K
    std_logic_vector(to_unsigned(character'pos('L'),8)) when X"4B", -- L
    std_logic_vector(to_unsigned(character'pos('M'),8)) when X"3A", -- M
    std_logic_vector(to_unsigned(character'pos('N'),8)) when X"31", -- N
    std_logic_vector(to_unsigned(character'pos('O'),8)) when X"44", -- O
    std_logic_vector(to_unsigned(character'pos('P'),8)) when X"4D", -- P
    std_logic_vector(to_unsigned(character'pos('Q'),8)) when X"15", -- Q
    std_logic_vector(to_unsigned(character'pos('R'),8)) when X"2D", -- R
    std_logic_vector(to_unsigned(character'pos('S'),8)) when X"1B", -- S
    std_logic_vector(to_unsigned(character'pos('T'),8)) when X"2C", -- T
    std_logic_vector(to_unsigned(character'pos('U'),8)) when X"3C", -- U
    std_logic_vector(to_unsigned(character'pos('V'),8)) when X"2A", -- V
    std_logic_vector(to_unsigned(character'pos('W'),8)) when X"1D", -- W
    std_logic_vector(to_unsigned(character'pos('X'),8)) when X"22", -- X
    std_logic_vector(to_unsigned(character'pos('Y'),8)) when X"35", -- Y
    std_logic_vector(to_unsigned(character'pos('Z'),8)) when X"1A", -- Z

    "00100000" when X"29", -- SPACIO
    "00001101" when X"5A", -- ENTER
    "00001000" when X"66", -- BACKSPACE
    std_logic_vector(to_unsigned(character'pos('*'),8)) when others; --
  end Behavioral;

```

La plantilla para su utilización.

```

COMPONENT key2ascii
PORT(
  key_code : IN std_logic_vector(7 downto 0);
  ascii_code : OUT std_logic_vector(7 downto 0)
);
END COMPONENT;

```

```

Inst_key2ascii: key2ascii PORT MAP(
  key_code => ,
  ascii_code =>
);

```

Ps2_rx

Descripción Funcional

Su función es recibir los datos provenientes por el puerto ps2. Para ello utiliza una máquina de estados finitos con 3 estados. Para más información acerca del funcionamiento del puerto ps2 se puede consultar el manual de la placa, (Spartan-3E Starter Kit Board User Guide)

El estado idle se encarga de esperar a se habilite la recepción a través de la señal rx_en y que se produzca un flanco de bajada en la señal de datos del puerto ps2. Si estas dos condiciones se cumplen, se inicializa el registro n con el valor 9 (1start+8datos+1paridad) y se añade el bit de start al registro de desplazamiento b y se pasa al estado dps.

En dps cuando se produce un flanco de bajada de la señal del reloj que envía el teclado se captura el dato que hay en ese momento en la señal de datos y se comprueba si n ha llegado a cero (se ha capturado todos los datos). Si es así, se pasa al siguiente estado load, si no, se decrementa en una unidad el registro n y se vuelve a dps.

En el estado load, se activa la señal rx_done_tick, indicando que se ha recibido un dato y salta al comienzo, idle. El diagrama ASMD se presenta en Figura 30. Diagrama ASMD de la unidad ps2_rx.

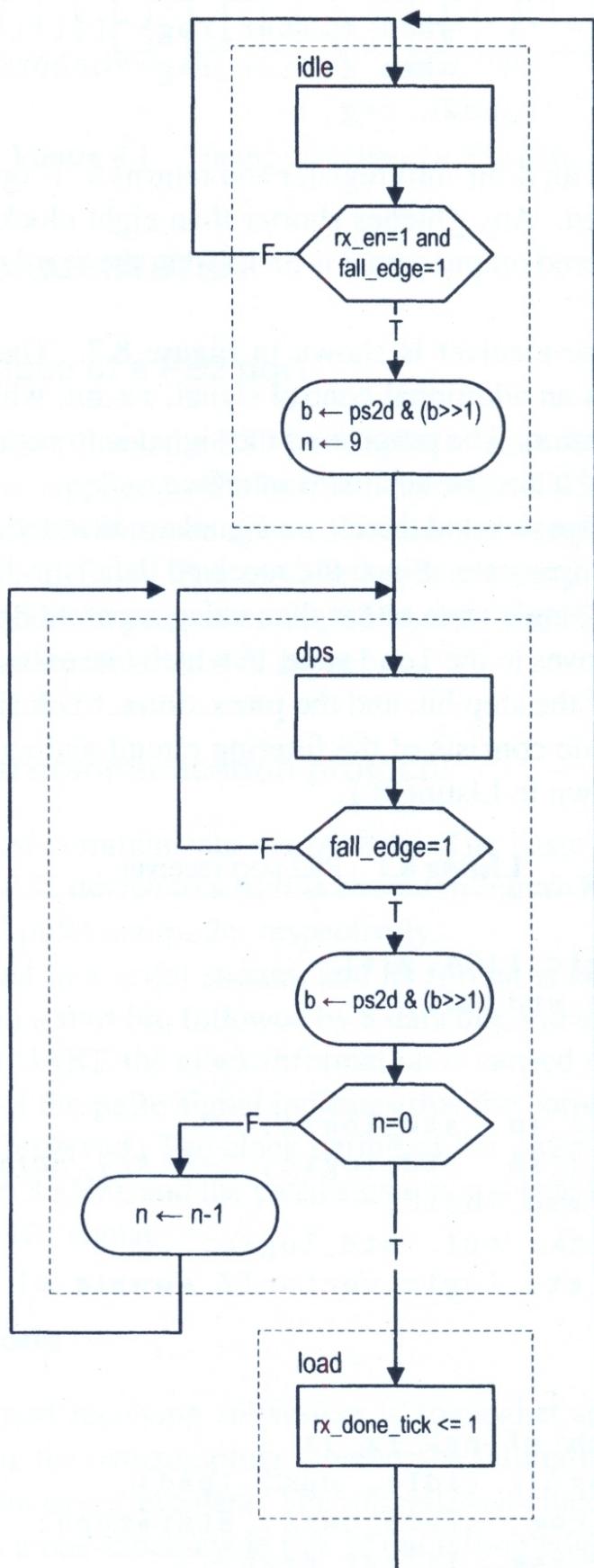


Figura 30. Diagrama ASMD de la unidad `ps2_rx`

Señales de E/S

Tabla 11. Ps2_rx

| Puerto | Sentido | Ancho | Función |
|--------------|---------|-------|----------------------------------|
| clk | IN | 1 | Reloj(50MHz) |
| reset | IN | 1 | Reset |
| ps2d | IN | 1 | Datos serie del teclado |
| ps2c | IN | 1 | Reloj puerto PS2 |
| rx_en | IN | 1 | Habilitación de recepción |
| dout | OUT | 8 | Datos recibidos |
| rx_done_tick | OUT | 1 | Se produce un evento por recibir |

Símbolo

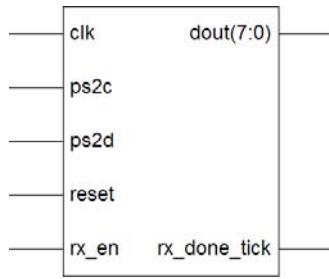


Figura 31. Símbolo ps2_rx

Descripción VHDL y plantillas para su utilización

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
--use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;
--- Uncomment the following library declaration if instantiating
--- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ps2_rx is
  Port ( clk : in STD_LOGIC;
         reset : in STD_LOGIC;
         ps2d : in STD_LOGIC;
         ps2c : in STD_LOGIC;
         rx_en : in STD_LOGIC;
         rx_done_tick : out STD_LOGIC;
         dout : out STD_LOGIC_VECTOR (7 downto 0));
end ps2_rx;

architecture Behavioral of ps2_rx is
  type statetype is (idle, dps, load);
  signal state_reg, state_next: statetype;
  signal filter_reg, filter_next: std_logic_vector(7 downto 0);
  signal f_ps2c_reg, f_ps2c_next: std_logic;
  signal b_reg, b_next: std_logic_vector(8 downto 0);
  signal n_reg, n_next: unsigned(3 downto 0);
  signal fall_edge: std_logic;
  -- signal paridad: std_logic;
begin
  =====
  --Filtro y flanco de bajada para ps2c

```

```

=====
process(clk, reset)
begin
    if reset='1' then
        filter_reg <= (others=>'0');
        f_ps2c_reg <= '0';
    elsif (clk'event and clk='1') then
        filter_reg <= filter_next;
        f_ps2c_reg <= f_ps2c_next;
    end if;
end process;

filter_next <= ps2c & filter_reg(7 downto 1);
f_ps2c_next <= '1' when filter_reg = X"FF" else
                                '0' when filter_reg = X"00" else
                                f_ps2c_reg;
fall_edge <= f_ps2c_reg and (not f_ps2c_next);
=====

-- fsm para extraer los 8bits de datos
=====

-- registros
process(clk,reset)
begin
    if reset='1' then
        state_reg <= idle;
        n_reg <= (others => '0');
        b_reg <= (others => '0');
    elsif (clk'event and clk='1') then
        state_reg <= state_next;
        n_reg <= n_next;
        b_reg <= b_next;
    end if;
end process;
--lógica de estado siguiente
process(state_reg, n_reg, b_reg, fall_edge, rx_en, ps2d)
begin
    rx_done_tick <='0';
    state_next <= state_reg;
    n_next <= n_reg;
    b_next <= b_reg;
    case state_reg is
        when idle =>
            if fall_edge='1' and rx_en='1' then
                -- estoy en el bit de start
                b_next <= ps2d & b_reg(10 downto 1);
                n_next <= "1001"; -- 9 = 10-1
                state_next <= dps;
            end if;
        when dps =>      --8 bits datos + 1 paridad + 1 stop
            if fall_edge='1' then
                if n_reg = 0 then -- Bit de stop entonces no guardo en registro
                    state_next <= load;
                else
                    -- En otro caso guardo,
                    b_next <= ps2d & b_reg(8 downto 1);
                    n_next <= n_reg - 1;
                end if;
            end if;
        when load =>
            -- 1 ultimo ciclo para completar el último desplazamiento
            state_next <= idle;
            rx_done_tick <= '1';
    end case;
end process;
--Salida
dout <= b_reg(7 downto 0); --bits de datos
-- paridad <= b_reg(8);
end Behavioral;

```

Y su plantilla para poder instanciarlo

```

COMPONENT ps2_rx
PORT(
    clk : IN std_logic;
    reset : IN std_logic;
    ps2d : IN std_logic;
    ps2c : IN std_logic;
    rx_en : IN std_logic;
    rx_done_tick : OUT std_logic;
    dout : OUT std_logic_vector(7 downto 0)
);
END COMPONENT;

Inst_ps2_rx: ps2_rx PORT MAP(
    clk => ,
    reset => ,
    ps2d => ,
    ps2c => ,
    rx_en => ,
    rx_done_tick => ,
    dout =>
);

```

Recursos utilizados de la FPGA

Synthesizing Unit <ps2_rx>.

Related source file is "D:/Designs/DCSE/LCD_Spartan3E/ps2_rx.vhd".

Found finite state machine <FSM_2> for signal <state_reg>.

| | | |
|----------------|-------------------|--|
| States | 3 | |
| Transitions | 6 | |
| Inputs | 2 | |
| Outputs | 4 | |
| Clock | clk (rising_edge) | |
| Reset | reset (positive) | |
| Reset type | asynchronous | |
| Reset State | idle | |
| Power Up State | idle | |
| Encoding | automatic | |
| Implementation | LUT | |

Found 9-bit register for signal <b_reg>.

Found 1-bit register for signal <f_ps2c_reg>.

Found 8-bit register for signal <filter_reg>.

Found 4-bit register for signal <n_reg>.

Found 4-bit subtractor for signal <n_reg\$addsub0000> created at line 107.

Summary:

inferred 1 Finite State Machine(s).

inferred 22 D-type flip-flop(s).

inferred 1 Adder/Subtractor(s).

Unit <ps2_rx> synthesized.

UART

Descripción Funcional

Universal asynchronous receiver and transmitter(UART) es un circuito que envía datos paralelos a través de una línea serie. La unidad incluye el transmisor y receptor. El transmisor es esencialmente un registro de desplazamiento especial que carga los datos en paralelo y los desplaza a una velocidad determinada. La línea serie está siempre a 1 cuando no hay actividad. La transmisión comienza con el bit de start que pone a 0 la línea, seguido de los bits de datos, un bit de paridad opcional y termina con el bit de stop que es un 1. El número de bits de datos pueden ser 6, 7, o 8 y el bit de stop puede ser 1, 1.5 o 2.

En una comunicación asíncrona no hay información del reloj, así que el receptor solo puede recibir bajo unos parámetros predeterminados. Se puede usar el esquema de sobre

muestreo para estimar el punto medio de los bits transmitidos y entonces sensarlo en ese punto.

Lo más común es usar una velocidad de muestreo 16 veces mayor a la velocidad de transmisión. Asumiendo que la comunicación utiliza N bits de datos y M bits de stop, la técnica de sobre muestreo sería como sigue.

1. Se espera a que la señal recibida se vuelva a 0, el comienzo del bit de start y entonces comienza un contador a avanzar.
2. Cuando llegue a 7, nos encontraremos en la mitad del bit de stop y reiniciaremos la cuenta.
3. Cuando el contador llegue a 15, estaremos en la mitad del primer bit de datos. Se recoge el valor y se desplaza dentro de un registro de desplazamiento y se reinicia la cuenta.
4. Se repite el paso 3 N-1 veces para recibir los bits restantes.
5. Si se está usando bit de paridad, se repite el punto 3 una vez más.
6. Se repite el punto 3 M veces para obtener el bit de stop.

El diagrama ASMD se presenta en la figura Figura 32. Diagrama ASMD de la unidad uart_rx.

El estado idle se encarga de esperar a que se produzca un cero en la señal de datos. Esto indica que se ha recibido el bit de start y que comienza la comunicación. Entonces se inicia s con 0 y se pasa el estado start.

En star se espera a recibir la señal s_tick (que su frecuencia es 16 veces mayor que la velocidad de transmisión) y se comprueba si s=7. Si es se reinicia la cuenta en s y se inicia el contador de datos n a cero y se pasa al estado data. Si no es 7 se incrementa s y se vuelve al estado start.

En data se comprueba si s ha llegado a 15 en cada impulso de s_tick y si no ha llegado se incrementa s y se vuelve a data. Si ha llegado se captura el dato en el registro de desplazamiento b y se reinicia la cuenta de s. Seguidamente se comprueba si se han capturado todos los datos (n=D_BIT-1) y si así es, se pasa al estado stop, si no se incrementa n y se vuelve a data.

En el estado stop se espera a que finalice el bit de stop.

La estructura del transmisor es muy parecida a la del receptor. En este caso se utiliza igualmente la señal s_tick, pero la divide en 16 con un contador para tener un reloj para poder generar los datos a la velocidad correcta.

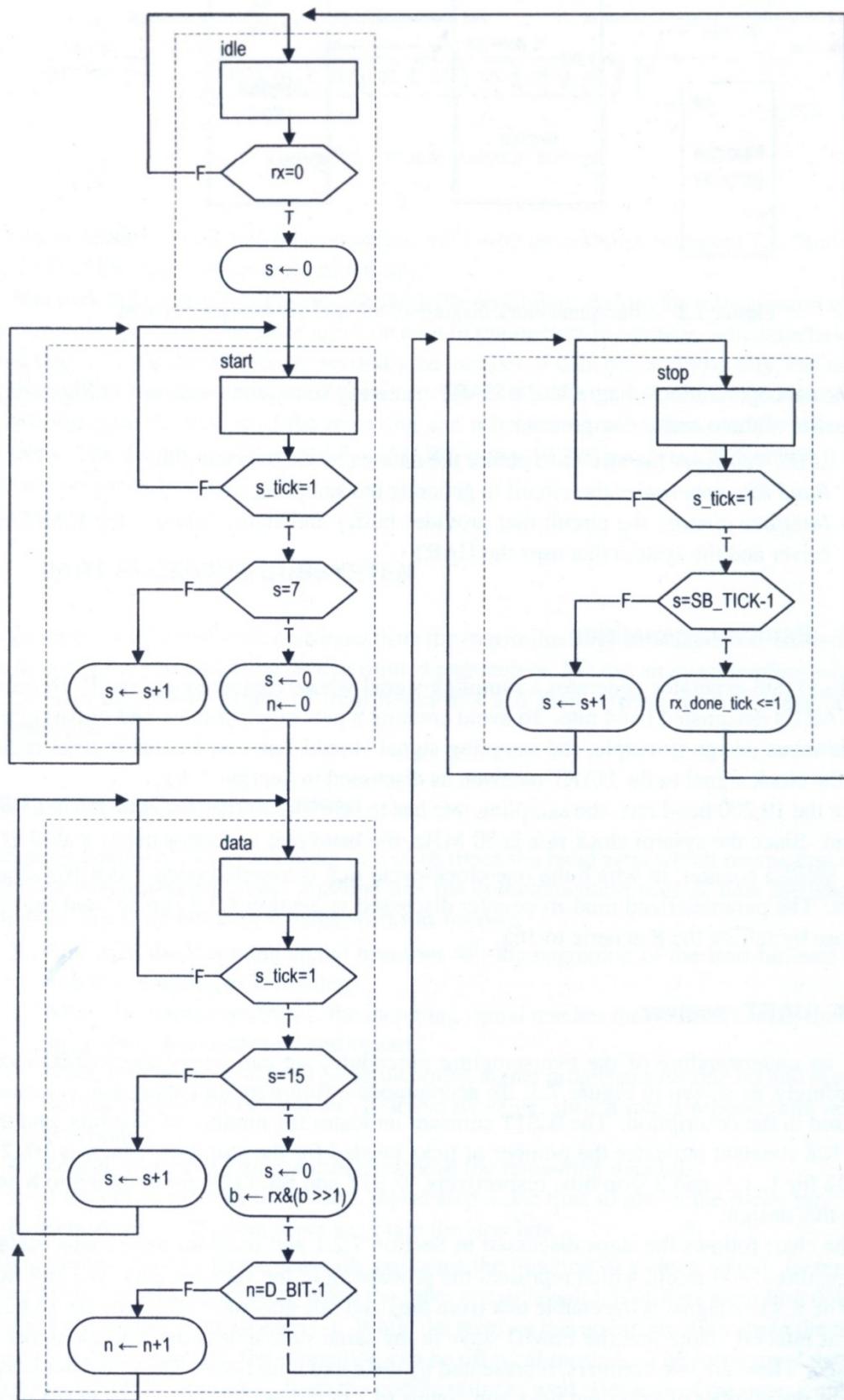


Figura 32. Diagrama ASMD de la unidad uart_rx

Señales de E/S

Símbolo

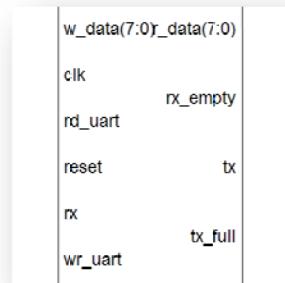


Figura 33. UART

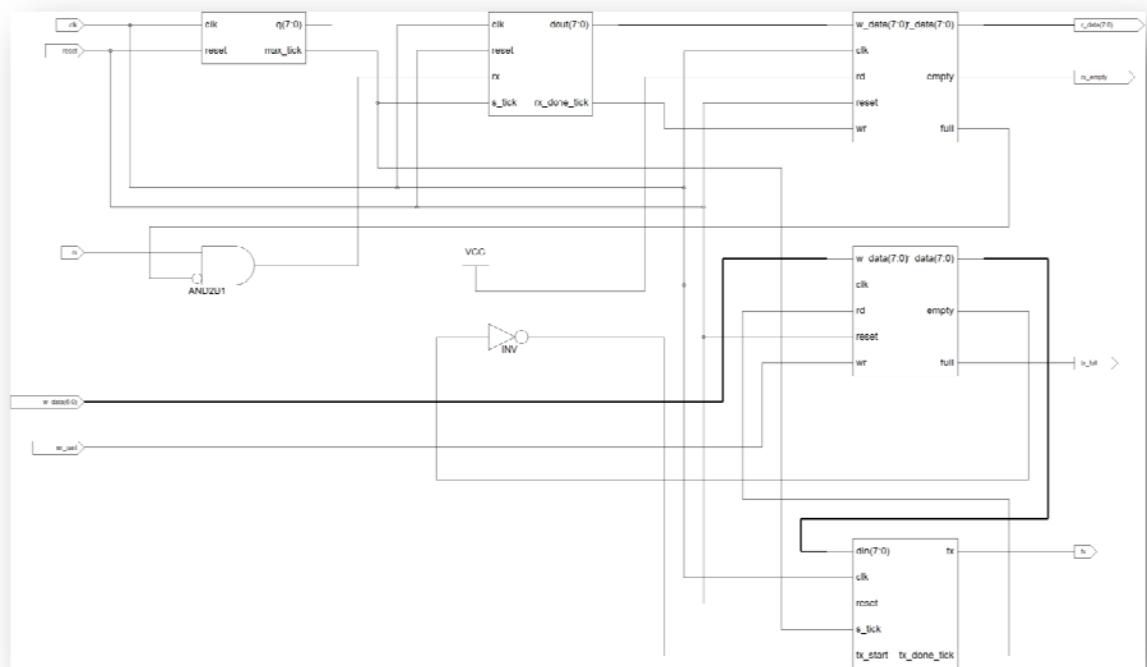


Figura 34. Componentes de la unidad uart

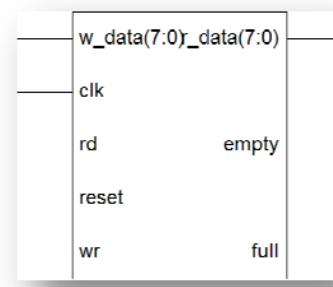
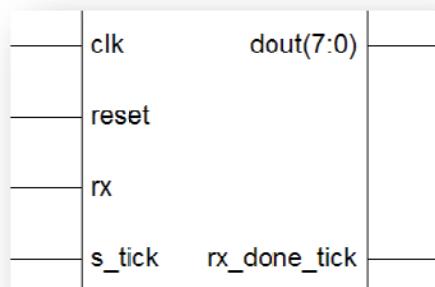


Figura 35. Fifo

Figura 36.Uart_rx

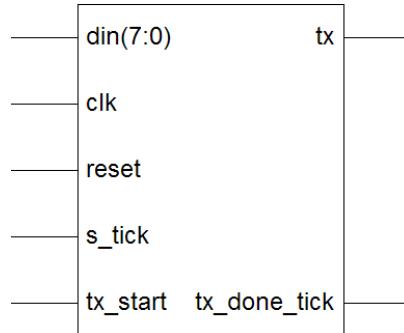


Figura 37. Unidad uart_tx

Descripción VHDL y plantillas para su utilización

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
--use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity uart is
    generic(
        --Valores por defecto
        --19200 baud, 8 bit datos, 1 bit stop, 2^2 fifo
        DBIT: integer :=8;           --nº bits datos
        SB_TICK: integer :=16;      --ticks for stop bits, 16/24/32 para 1/1.5/2 bits de stop
        DVSR: integer :=163;        --baud rate divisor
        DVSRR_BIT: integer :=8;     --bits de DVSR
        FIFO_W: integer :=2;        --bits de dir en fifo, las palabras en FIFO=2^FIFO_W
    );
    Port ( clk : in STD_LOGIC;
            reset : in STD_LOGIC;
            rd_uart, wr_uart : in STD_LOGIC;
            rx : in STD_LOGIC;
            w_data : in STD_LOGIC_VECTOR (7 downto 0);
            tx_full : out STD_LOGIC;
            rx_empty : out STD_LOGIC;
            r_data : out STD_LOGIC_VECTOR (7 downto 0);
            tx : out STD_LOGIC);
end uart;

architecture Behavioral of uart is
    signal tick: std_logic;
    signal rx_done_tick: std_logic;
    signal tx_fifo_out: std_logic_vector(7 downto 0);
    signal rx_data_out: std_logic_vector(7 downto 0);
    signal tx_empty, tx_fifo_not_empty: std_logic;
    signal tx_done_tick: std_logic;
    signal rx_full, rx_fifo_not_full: std_logic;
    signal recibe: std_logic;
begin
    baud_gen_unit: entity work.mod_m_counter(Behavioral)
        generic map(M=>DVSR, N=>DVSRR_BIT)
        port map(clk => clk, reset => reset, q => open, max_tick=>tick);
    uart_rx_unit: entity work uart_rx(Behavioral)
        generic map(DBIT=>DBIT, SB_TICK=>SB_TICK)
        port map(
            clk => clk,
            reset => reset,
            rx => recibe,
            s_tick => tick,
            rx_done_tick => rx_done_tick,
            dout => rx_data_out
        )

```

```

        );
fifo_rx_unit: entity work fifo(Behavioral)
    generic map(B=>DBIT, W=>FIFO_W)
    port map(
        clk => clk,
        reset => reset,
        rd => rd_uart,
        wr => rx_done_tick,
        w_data => rx_data_out,
        empty => rx_empty,
        full => rx_full,
        r_data => r_data
    );
rx_fifo_not_full <= not rx_full;
recibe <= rx and rx_fifo_not_full;

fifo_tx_unit: entity work fifo(Behavioral)
    generic map(B=>DBIT, W=>FIFO_W)
    port map(
        clk => clk,
        reset => reset,
        rd => tx_done_tick,
        wr => wr_uart,
        w_data => w_data,
        empty => tx_empty,
        full => tx_full,
        r_data => tx_fifo_out
    );
uart_tx_unit: entity work uart_tx(Behavioral)
    generic map(DBIT=>DBIT, SB_TICK=>SB_TICK)
    port map(
        clk => clk,
        reset => reset,
        tx_start => tx_fifo_not_empty,
        s_tick => tick,
        din => tx_fifo_out,
        tx_done_tick => tx_done_tick,
        tx => tx
    );
tx_fifo_not_empty <= not tx_empty;
end Behavioral;

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
--use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;
--- Uncomment the following library declaration if instantiating
--- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mod_m_counter is
    generic(
        N: integer := 4;                      --numero de bits
        M: integer := 10                         -- mod M
    );
    Port ( clk : in STD_LOGIC;
            reset : in STD_LOGIC;
            max_tick : out STD_LOGIC;
            q : out STD_LOGIC_VECTOR (N-1 downto 0));
end mod_m_counter;

architecture Behavioral of mod_m_counter is
    signal r_reg, r_next: unsigned(N-1 downto 0);
begin
    --registro
    process(clk, reset)
    begin
        if reset='1' then
            r_reg <= (others => '0');
        elsif (clk'event and clk='1') then
            r_reg <= r_next;
        end if;
    end process;

    -- logica de estado siguiente
    r_next <= (others => '0') when r_reg = (M-1) else
        r_reg + 1;

```

```
-- logica de salida
q <= std_logic_vector(r_reg);
max_tick <= '1' when r_reg=(M-1) else '0';
end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
--use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;
--- Uncomment the following library declaration if instantiating
--- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity uart_rx is
    generic(
        DBIT : integer :=8; --nº de bits de datos
        SB_TICK : integer :=16 --nº de ticks para el bit de stop (16->1, 24->1.5, 32->2bits)
    );
    Port ( clk : in STD_LOGIC;
            reset : in STD_LOGIC;
            rx : in STD_LOGIC;
            s_tick : in STD_LOGIC;
            rx_done_tick : out STD_LOGIC;
            dout : out STD_LOGIC_VECTOR (7 downto 0));
end uart_rx;

architecture Behavioral of uart_rx is
    type state_type is (idle, start, data, stop);
    signal state_reg, state_next : state_type;
    signal s_reg, s_next: unsigned (3 downto 0);
    signal n_reg, n_next: unsigned (2 downto 0);
    signal b_reg, b_next: std_logic_vector (7 downto 0);
begin
    begin
        --estados FSMD & registro de datos
        process (clk, reset)
        begin
            if reset='1' then
                state_reg <= idle;
                s_reg <= (others =>'0');
                n_reg <= (others =>'0');
                b_reg <= (others =>'0');
            elsif (clk'event and clk='1') then
                state_reg <= state_next;
                s_reg <= s_next;
                n_reg <= n_next;
                b_reg <= b_next;
            end if;
        end process;
        --logica de estado siguiente & camino de datos y unidades funcionales
        process(state_reg, s_reg, n_reg, b_reg, s_tick, rx)
        begin
            state_next <= state_reg;
            s_next <= s_reg;
            n_next <= n_reg;
            b_next <= b_reg;
            rx_done_tick <= '0';
            case state_reg is
                when idle =>
                    if rx='0' then
                        state_next <= start;
                        s_next <= (others =>'0');
                    end if;
                when start =>
                    if s_tick = '1' then
                        if s_reg=7 then
                            s_next <= (others =>'0');
                            n_next <= (others =>'0');
                            state_next <= data;
                        else
                            s_next <= s_reg + 1;
                        end if;
                    end if;
                when data =>
                    if s_tick = '1' then
                        if s_reg=15 then
                            b_next <= rx & b_reg(7 downto 1);
                        end if;
                    end if;
            end case;
        end process;
    end;
end;
```

```

        s_next <= (others =>'0');
        if n_reg=(DBIT-1) then
            state_next <= stop;
        else
            n_next <= n_reg + 1;
        end if;
    else
        s_next <= s_reg + 1;
    end if;
end if;
when stop =>
    if (s_tick = '1') then
        if s_reg = (SB_TICK-1) then
            state_next <= idle;
            rx_done_tick <= '1';
        else
            s_next <= s_reg + 1;
        end if;
    end if;
end case;
end process;
dout <= b_reg;
end Behavioral;

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
--use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity uart_tx is
    generic(
        DBIT : integer :=8;      --nº de bits de datos
        SB_TICK : integer := 16 --nº de ticks para el bit de stop
    );
    Port ( clk : in STD_LOGIC;
            reset : in STD_LOGIC;
            tx_start : in STD_LOGIC;
            s_tick : in STD_LOGIC;
            din : in STD_LOGIC_VECTOR (7 downto 0);
            tx_done_tick : out STD_LOGIC;
            tx : out STD_LOGIC);
end uart_tx;

architecture Behavioral of uart_tx is
    type state_type is (idle, start, data, stop);
    signal state_reg, state_next: state_type;
    signal s_reg, s_next: unsigned(3 downto 0);
    signal n_reg, n_next: unsigned(2 downto 0);
    signal b_reg, b_next: std_logic_vector(7 downto 0);
    signal tx_reg, tx_next: std_logic;
begin
    --estados FSMD & registros de datos
    process(clk,reset)
    begin
        if reset='1' then
            state_reg <= idle;
            s_reg <= (others => '0');
            n_reg <= (others => '0');
            b_reg <= (others => '0');
            tx_reg <= '1';
        elsif (clk'event and clk='1') then
            state_reg <= state_next;
            s_reg <= s_next;
            n_reg <= n_next;
            b_reg <= b_next;
            tx_reg <= tx_next;
        end if;
    end process;
    -- logica de estado siguiente & camino de datos
    process(state_reg, s_reg, n_reg, b_reg, s_tick, tx_reg, tx_start, din)
    begin
        state_next <= state_reg;
        s_next <= s_reg;

```

```

n_next <= n_reg;
b_next <= b_reg;
tx_next <= tx_reg;
tx_done_tick <= '0';
case state_reg is
    when idle =>
        tx_next <= '1';
        if tx_start='1' then
            state_next <= start;
            s_next <= (others => '0');
            b_next <= din;
        end if;
    when start =>
        tx_next <= '0';
        if (s_tick = '1') then
            if s_reg=15 then
                state_next <= data;
                s_next <= (others => '0');
                n_next <= (others => '0');
            else
                s_next <= s_reg + 1;
            end if;
        end if;
    when data =>
        tx_next <= b_reg(0);
        if (s_tick = '1') then
            if s_reg=15 then
                s_next <= (others => '0');
                b_next <= '0' & b_reg(7 downto 1);
                if n_reg=(DBIT-1) then
                    state_next <= stop;
                else
                    n_next <= n_reg + 1;
                end if;
            else
                s_next <= s_reg + 1;
            end if;
        end if;
    when stop =>
        tx_next <= '1';
        if (s_tick = '1') then
            if s_reg=(SB_TICK-1) then
                state_next <= idle;
                tx_done_tick <= '1';
            else
                s_next <= s_reg + 1;
            end if;
        end if;
    end case;
end process;

tx <= tx_reg;

```

end Behavioral;

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
--use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity fifo is
    generic( B: natural :=8; --numero de bits
              W: natural :=4 --numero de bits de direccion
            );
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           rd : in STD_LOGIC;
           wr : in STD_LOGIC;
           w_data : in STD_LOGIC_VECTOR (B-1 downto 0);
           empty : out STD_LOGIC;
           full : out STD_LOGIC;
           r_data : out STD_LOGIC_VECTOR (B-1 downto 0));

```

```

end fifo;

architecture Behavioral of fifo is
    type reg_file_type is array (2**W-1 downto 0) of std_logic_vector(B-1 downto 0);
    signal array_reg : reg_file_type;
    signal w_ptr_reg, w_ptr_next, w_ptr_succ : std_logic_vector (W-1 downto 0);
    signal r_ptr_reg, r_ptr_next, r_ptr_succ : std_logic_vector (W-1 downto 0);
    signal full_reg, empty_reg, full_next, empty_next : std_logic;
    signal wr_op: std_logic_vector (1 downto 0);
    signal wr_en : std_logic;

begin
    =====
    -- Banco de registro
    =====
    process(clk,reset)
    begin
        if (reset = '1') then
            array_reg <= ( others=>(others=>'0'));
        elsif (clk'event and clk='1') then
            if (wr_en='1') then
                array_reg(to_integer(unsigned(w_ptr_reg))) <= w_data;
            end if;
        end if;
    end process;
    --lectura
    r_data <= array_reg(to_integer(unsigned(r_ptr_reg)));
    --Write enable solo cuando fifo no esta llena
    wr_en <= wr and (not full_reg);

    =====
    -- Logica de control de la fifo
    =====

    process(clk,reset)
    begin
        if (reset = '1') then
            w_ptr_reg <= (others =>'0');
            r_ptr_reg <= (others =>'0');
            full_reg <= '0';
            empty_reg <= '0';
        elsif (clk'event and clk ='1') then
            w_ptr_reg <= w_ptr_next;
            r_ptr_reg <= r_ptr_next;
            full_reg <= full_next;
            empty_reg <= empty_next;
        end if;
    end process;

    --Valor siguiente de los punteros
    w_ptr_succ <= std_logic_vector(unsigned(w_ptr_reg)+1);
    r_ptr_succ <= std_logic_vector(unsigned(r_ptr_reg)+1);

    --logica de estado siguiente para los punteros de lectura y escritura
    wr_op <= wr & rd;
    process(w_ptr_reg,w_ptr_succ,r_ptr_reg,r_ptr_succ,wr_op,empty_reg,full_reg)
    begin
        w_ptr_next <= w_ptr_reg;
        r_ptr_next <= r_ptr_reg;
        full_next <= full_reg;
        empty_next <= empty_reg;
        case wr_op is
            when "00" => -- no op
            when "01" => -- read
                if (empty_reg /= '1') then --Si no esta vacio
                    r_ptr_next <= r_ptr_succ;
                    full_next <= '0';
                    if (r_ptr_succ = w_ptr_reg) then
                        empty_next <= '1';
                    end if;
                end if;
            when "10" => -- write
                if(full_reg /= '1') then --Si no esta lleno
                    w_ptr_next <= w_ptr_succ;
                    empty_next <= '0';
                    if (w_ptr_succ = r_ptr_reg) then
                        full_next <= '1';
                    end if;
                end if;
            when others => --write/read
        end case;
    end process;

```

```

        w_ptr_next <= w_ptr_succ;
        r_ptr_next <= r_ptr_succ;
    end case;
end process;

--salidas
full <= full_reg;
empty <= empty_reg;

end Behavioral;

```

Recursos utilizados de la FPGA

Synthesizing Unit <mod_m_counter>.

Related source file is "D:/Designs/DCSE/LCD_Spartan3E/mod_m_counter.vhd".

Found 8-bit adder for signal <r_next\$addsub0000> created at line 55.

Found 8-bit register for signal <r_reg>.

Summary:

inferred 8 D-type flip-flop(s).

inferred 1 Adder/Subtractor(s).

Unit <mod_m_counter> synthesized.

Synthesizing Unit <uart_rx>.

Related source file is "D:/Designs/DCSE/LCD_Spartan3E/uart_rx.vhd".

Found finite state machine <FSM_3> for signal <state_reg>.

| | | |
|----------------|-------------------|--|
| States | 4 | |
| Transitions | 12 | |
| Inputs | 5 | |
| Outputs | 5 | |
| Clock | clk (rising_edge) | |
| Reset | reset (positive) | |
| Reset type | asynchronous | |
| Reset State | idle | |
| Power Up State | idle | |
| Encoding | automatic | |
| Implementation | LUT | |

Found 8-bit register for signal <b_reg>.

Found 3-bit register for signal <n_reg>.

Found 3-bit adder for signal <n_reg\$addsub0000> created at line 98.

Found 4-bit register for signal <s_reg>.

Found 4-bit adder for signal <s_reg\$share0000> created at line 74.

Summary:

inferred 1 Finite State Machine(s).

inferred 15 D-type flip-flop(s).

inferred 2 Adder/Subtractor(s).

Unit <uart_rx> synthesized.

Synthesizing Unit <fifo_2>.

Related source file is "D:/Designs/DCSE/LCD_Spartan3E/fifo.vhd".

Found 8-bit 4-to-1 multiplexer for signal <r_data>.

Found 32-bit register for signal <array_reg>.

Found 1-bit register for signal <empty_reg>. Found 2-bit comparator equal for signal <empty_reg\$cmp_eq0000> created at line 111. Found 1-bit 4-to-1 multiplexer for signal <empty_reg\$mux0000> created at line 105. Found 1-bit register for signal <full_reg>. Found 2-bit comparator equal for signal <full_reg\$cmp_eq0000> created at line 119. Found 1-bit 4-to-1 multiplexer for signal <full_reg\$mux0000> created at line 105. Found 2-bit register for signal <r_ptr_reg>. Found 2-bit 4-to-1 multiplexer for signal <r_ptr_reg\$mux0000> created at line 105. Found 2-bit adder for signal <r_ptr_succ\$add0000> created at line 95. Found 2-bit register for signal <w_ptr_reg>. Found 2-bit 4-to-1 multiplexer for signal <w_ptr_reg\$mux0000> created at line 105. Found 2-bit adder for signal <w_ptr_succ\$add0000> created at line 94.

Summary:

- inferred 38 D-type flip-flop(s).
- inferred 2 Adder/Subtractor(s).
- inferred 2 Comparator(s).
- inferred 14 Multiplexer(s).

Unit <fifo_2> synthesized.

Synthesizing Unit <uart_tx>.

Related source file is "D:/Designs/DCSE/LCD_Spartan3E/uart_tx.vhd".

Found finite state machine <FSM_4> for signal <state_reg>.

| | | |
|----------------|-------------------|--|
| States | 4 | |
| Transitions | 12 | |
| Inputs | 4 | |
| Outputs | 4 | |
| Clock | clk (rising_edge) | |
| Reset | reset (positive) | |
| Reset type | asynchronous | |
| Reset State | idle | |
| Power Up State | idle | |
| Encoding | automatic | |
| Implementation | LUT | |

Found 8-bit register for signal <b_reg>. Found 3-bit register for signal <n_reg>. Found 3-bit adder for signal <n_reg\$addsub0000> created at line 107. Found 4-bit adder for signal <s_next\$add0000> created at line 95. Found 4-bit register for signal <s_reg>. Found 1-bit register for signal <tx_reg>.

Summary:

- inferred 1 Finite State Machine(s).
- inferred 16 D-type flip-flop(s).
- inferred 2 Adder/Subtractor(s).

Unit <uart_tx> synthesized.

Resultados.

Se resumen los resultados obtenidos en cuanto a recursos consumidos de la FPGA, ruta crítica y potencia consumida

Se resumen los resultados obtenidos en cuanto a recursos consumidos de la FPGA:

Selected Device : 3s500efg320-4

| | | | | |
|-----------------------------|------|--------|------|-----|
| Number of Slices: | 699 | out of | 4656 | 15% |
| Number of Slice Flip Flops: | 574 | out of | 9312 | 6% |
| Number of 4 input LUTs: | 1302 | out of | 9312 | 13% |
| Number of IOs: | 31 | | | |
| Number of bonded IOBs: | 31 | out of | 232 | 13% |
| Number of GCLKs: | 1 | out of | 24 | 4% |

A continuación se observa los resultados obtenidos de la herramienta en cuanto a potencia consumida. Se puede observar que se estima un consumo de potencia total de 110mW.

| Power summary | I(mA) | P(mW) |
|-----------------------------------|-------|-------|
| Total estimated power consumption | 110 | |
| --- | | |
| Total Vccint 1.20V | 34 | 41 |
| Total Vccaux 2.50V | 18 | 45 |
| Total Vcco33 3.30V | 8 | 25 |
| --- | | |
| Clocks | 3 | 4 |
| Inputs | 0 | 0 |
| Logic | 1 | 1 |
| Outputs | | |
| Vcco33 | 6 | 18 |
| Signals | 4 | 4 |
| --- | | |
| Quiescent Vccint 1.20V | 26 | 31 |
| Quiescent Vccaux 2.50V | 18 | 45 |
| Quiescent Vcco33 3.30V | 2 | 7 |

Prototipado.

Las siguientes imágenes muestran el resultado de la práctica. Primeramente aparece el mensaje Bienvenido Epf Supersonic, parpadeando, seguidamente aparece el mensaje Practica 3 DCSE y finalmente en lo exigido por la práctica, la segunda fila rellenada con los número en hexadecimal desde 0 a F.



Figura 38.Bienvenida inicial 1

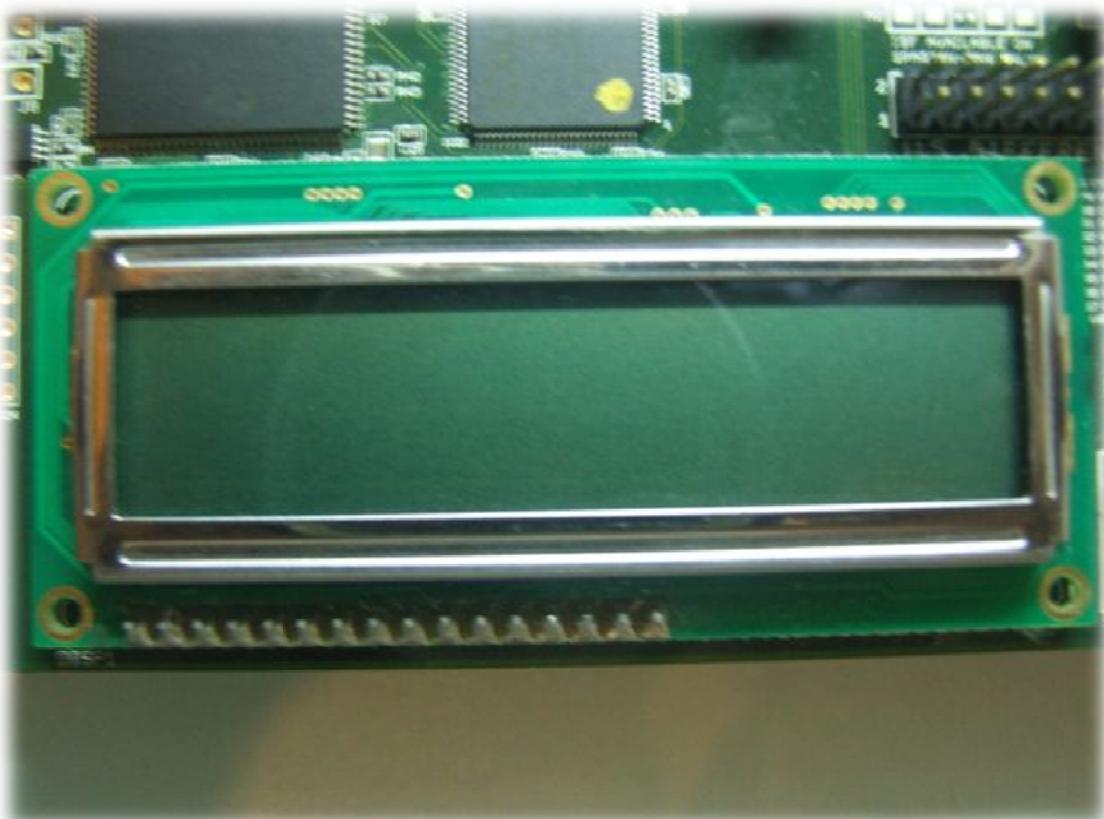


Figura 39.Bienvenida inicial 2



Figura 40. Bienvenida inicial 3



Figura 41. Bienvenida inicial 4

Conclusiones.

Cuando se necesita utilizar un visualizador que pueda manejar números, letras e incluso símbolos se puede utilizar un sencillo y económico módulo alfanumérico LCD, el cual contiene un circuito integrado controlador que realiza la mayoría del trabajo por el diseñador.

Los módulos LCD soportan una serie de posibilidades de visualización que se pueden acomodar a cada tipo de aplicación. Lo que sigue es una pequeña descripción de este tipo de posibilidades:

- Activación/desactivación de la pantalla: permite que el usuario encienda o apague el visualizador a través del procesador.
- Activación/desactivación del cursor: permite seleccionar el cursor de pantalla o eliminarlo.
- Parpadeo del cursor: el usuario puede elegir entre un cursor estático o parpadeante. El carácter libre del cursor también parpadeará.
- Desplazamiento a la derecha/izquierda: desplaza los datos de la pantalla del visualizador.
- Vuelta al primer carácter de la pantalla: hace que el cursor vuelva a la primera posición de la pantalla de visualización (posición HOME) si se ha desplazado previamente.

Hay una variedad de formas de realizar la interface de un módulo LCD con un procesador, microprocesador o circuito dedicado (ASIC, FPGA). Los módulos LCD tienen dos modos de interface de hardware: uno de 4bits y otro de 8 bits. En el modo de 4 bits, cada byte de datos se transfiere al módulo mediante dos operaciones de escritura. El modo de 4 bits utiliza únicamente los cuatro bits superiores de las líneas de datos (DB7-DB4). En el modo de 8 bits, los bytes de datos se transfieren con una única operación de escritura que ahorra tiempo utilizando las 8 líneas de datos (DB7-DB0). La única ventaja que se obtiene del modo de interface de 4bits es el ahorro de líneas del bus de datos. el modo de 8 bits es ligeramente más sencillo de llevar a cabo, por ello es preferible utilizar el modo de 8bits, a menos que como es nuestro caso, se tenga un número limitado de patillas de entrada salida.

Para poder controlar un módulo LCD, se puede usar desde un micro a una sencilla máquina de estados como la que se ha planteado en este diseño. Para describir el circuito se prefiere separar los elementos de memoria de los elementos combinacionales. Así se simplifica el código, y se consiguen mejores resultados.

Referencias

- [1] FPGA Prototyping by VHDL Examples – Pong P. Chu – Wiley Interscience
- [2] Spartan-3E Starter Kit Board User Guide
- [3] Elektor Nº125 – octubre 1990
- [4] Rotary Encoder Interface for Spartan-3E Starter Kit – Ken Chapman- Xilinx Ltd
- [5] HD44780U (LCD-II) Dot Matrix Liquid Crystal Display Controller/Driver. Hitachi Ltd.

Bibliografía

Ken Chapman. *Rotary Encoder Interface for Spartan-3E Starter Kit*. Xilinx Ltd.

Pong P. Chu. *FPGA Prototyping by VHDL Examples*. Wiley Inter Science.

Proyectos con Displays LCD. (Octubre 1990). *Elektor Nº125* , 14-21.

Spartan-3E Starter Kit Board User Guide.

Hoja de características del componente obtenido

Nombre del componente

Modulo 3

Descripción general y características

Ventajas

Interfaz cómoda

Utilización de un teclado PS/2 para escribir

Modo visualización y modo edición

Dos modos de escritura: con avance de línea y sin avance de línea

Visualización de los datos del teclado a través de un puerto serie asíncrono

Aplicaciones

Presentación de mensajes

Carteles Luminosos

Descripción del Producto

Este circuito utiliza una FSMD para controlar un módulo alfanumérico LCD. Realiza el proceso de inicialización, configuración y bienvenida inicial. Permite dos modos de funcionamiento:

Modo visualización

La pantalla está continuamente desplazándose a derecha e izquierda.

Modo edición

Permite al usuario modificar el contenido de la pantalla a través de 5 pulsadores, un botón rotatorio y un teclado. Con el switch1 se cambia el modo de escritura, con avance o sin avance del cursor con cada carácter recibido. El modo sin avance se usa para ingresar caracteres a través de los pulsadores, actuando de forma similar a como lo hace un teléfono móvil. El modo con avance se introduce para escribir con un teclado PS/2. Esto es útil porque con el teclado se obtiene directamente el carácter deseado. A continuación se muestra una tabla indicando los caracteres que tiene asignado cada pulsador.

Tabla 12. Asignación de caracteres a los pulsadores.

| | |
|-----------|----------------------------|
| BTN_NORTH | ABCDEFGHIJKLMNPQRSTUVWXYZ |
| BTN_SOUTH | abcdefghijklmnopqrstuvwxyz |
| BTN_EAST | 0123456789 |
| BTN_WEST | !#\$%&'()*+,-./ |

Símbolo

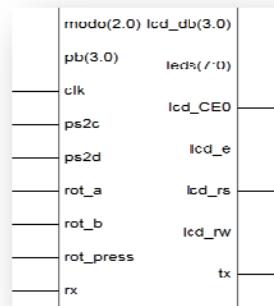


Figura 42. Módulo3

Diagrama de bloques

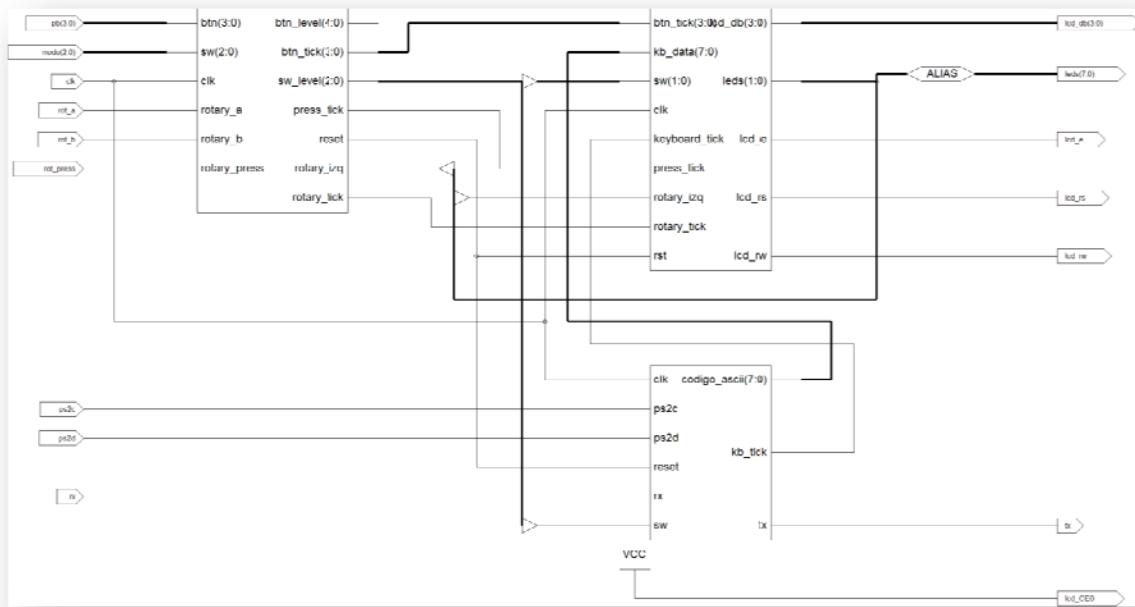


Figura 43. Diagrama de bloques del Módulo3

Patillas de E/S

Tabla 13. Entradas Salida del Modulo3

| Puerto | Sentido | Ancho | Función | Placa |
|-----------|---------|-------|--|-------|
| clk | IN | 1 | Reloj(50MHz) | C9 |
| modo | IN | 3 | Switches de modo Sw0 – edición(0), visualización(1) Sw1 – cursor, sin avance(0), avanza(1) | |
| pb | IN | 4 | Pulsadores | |
| rot_press | IN | 1 | Pulsador del botón rotatorio | |
| rot_a | IN | 1 | Interruptor Rotatorio | |

| | | | | |
|----------------|-----|---|---------------------------------|--|
| rot_b | IN | 1 | Interruptor Rotatorio | |
| ps2c | IN | 1 | Señal de reloj PS/2 | |
| ps2d | IN | 1 | Señal de datos PS/2 | |
| rx | IN | 1 | Recepción UART | |
| tx | OUT | 1 | Transmisión UART | |
| lcd_db | OUT | 4 | Comando/Dato al LCD | |
| lcd_e | OUT | 1 | Habilitación del LCD | |
| lcd_rs | OUT | 1 | Indica comando(0) ó dato(1) | |
| lcd_rw | OUT | 1 | Indica escritura(0), lectura(1) | |
| lcd_CE0 | OUT | 1 | Señal para usar el LCD | |
| leds | OUT | 8 | Visualización de estados | |

Descripción VHDL y plantilla para su utilización

```
COMPONENT Modulo3
PORT(
    clk : IN std_logic;
    modo : IN std_logic_vector(2 downto 0);
    pb : IN std_logic_vector(3 downto 0);
    rot_press : IN std_logic;
    rot_a : IN std_logic;
    rot_b : IN std_logic;
    ps2c : IN std_logic;
    ps2d : IN std_logic;
    rx : IN std_logic;
    tx : OUT std_logic;
    lcd_db : OUT std_logic_vector(3 downto 0);
    lcd_e : OUT std_logic;
    lcd_rs : OUT std_logic;
    lcd_rw : OUT std_logic;
    lcd_CE0 : OUT std_logic;
    leds : OUT std_logic_vector(7 downto 0)
);
END COMPONENT;
```

```
Inst_Modulo3: Modulo3 PORT MAP(
    clk => ,
    modo => ,
    pb => ,
    rot_press => ,
    rot_a => ,
    rot_b => ,
    ps2c => ,
    ps2d => ,
    rx => ,
    tx => ,
    lcd_db => ,
    lcd_e => ,
    lcd_rs => ,
    lcd_rw => ,
    lcd_CE0 => ,
    leds =>
);
```

Test-bench y simulación

A continuación se presenta el vector de test realizado a este módulo.

Se comprueba que el reset funciona y se comprobar que va al estado idle. Cuando se deja de aplicar el reset, la máquina empieza el proceso de inicialización con el estado poi1.

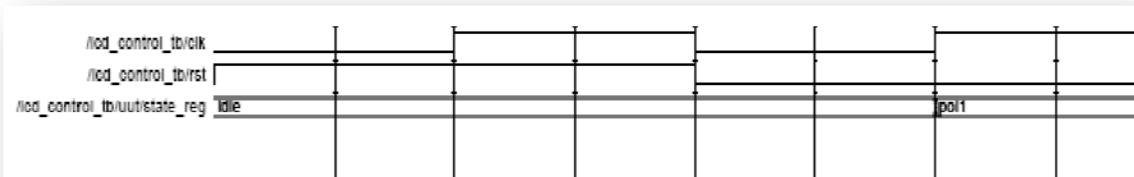


Figura 44. Reset, idle y poi1

Espera de 15ms en poi1.

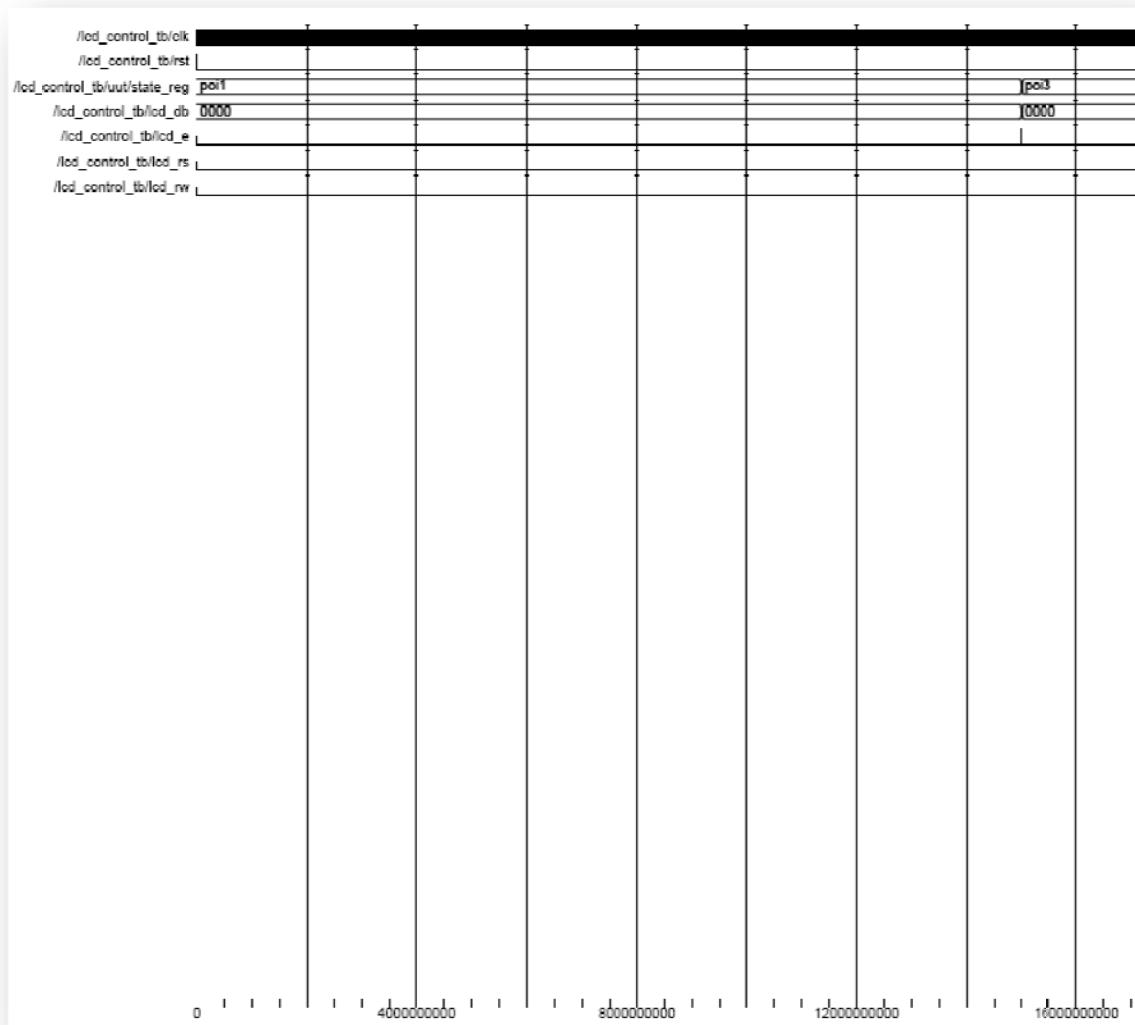


Figura 45. Espera 15ms en poi1

En poi2 se envía en LCD_DB= X"3" y lcd_e=1 durante 12 ciclos de la señal de reloj.

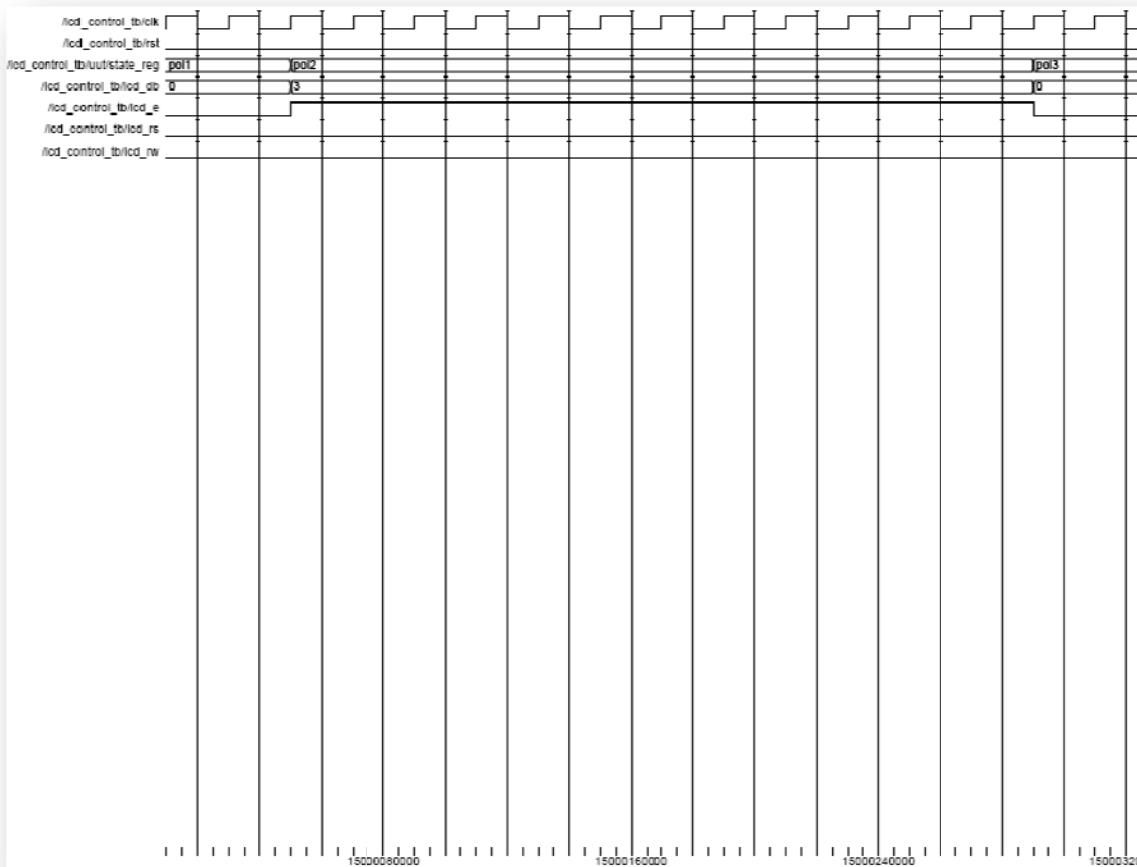


Figura 46. Envío de la primera trama de inicialización. `Lcd_db=X"3"`

Al igual que en poi4 poi6 y poi8 que se envía lcd db=X"2".

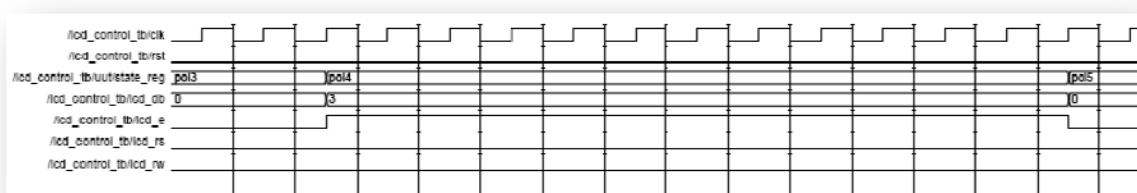


Figura 47. Envío en poi4 lcd_db=X"3"

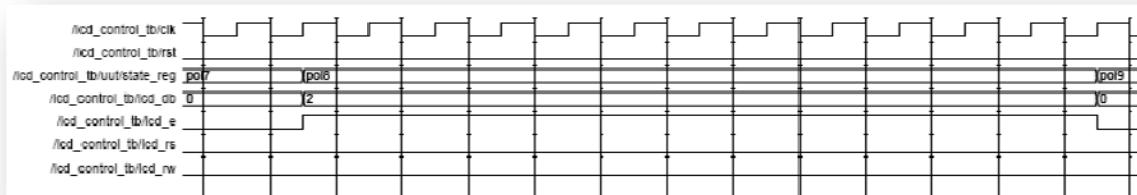


Figura 48. Envío en poi8 lcd_db=X"2"

Se cumplen los tiempos en configuración. 2 ciclos de reloj en conf_hsetup y 12 en conf_hhold.

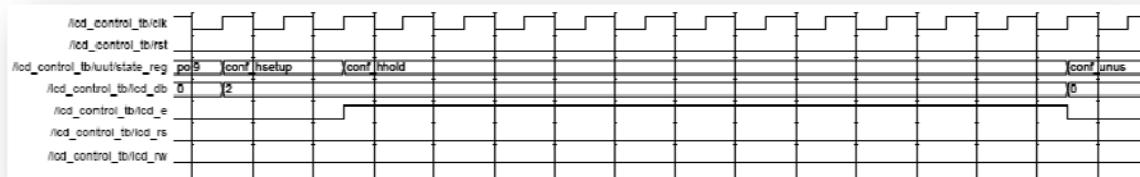


Figura 49. Envío de los 4 bits más significativos y tiempos durante el envío de configuración. 2ciclos en conf_hsetup y 12 en conf_hhold

En conf_unus se esperan 50 ciclos de reloj

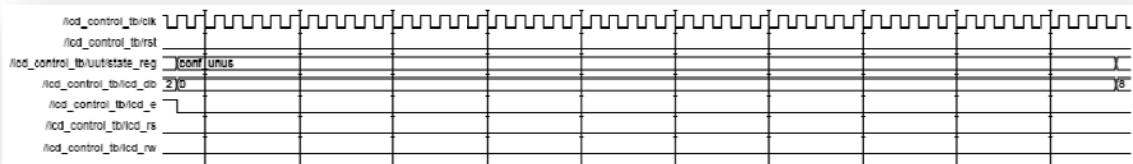


Figura 50. 1us en conf_unus

El envío de los 4 bits menos significativos

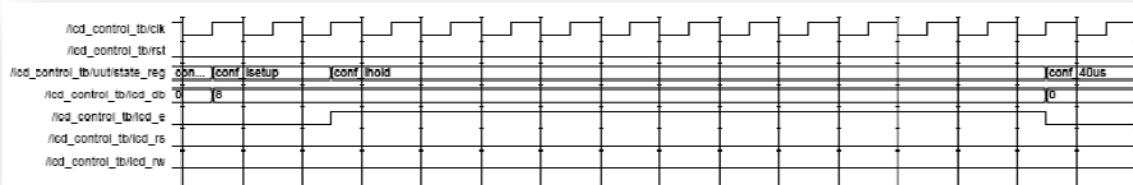


Figura 51. Envío de los 4 bits menos significativos.

Espera del estado conf_40us

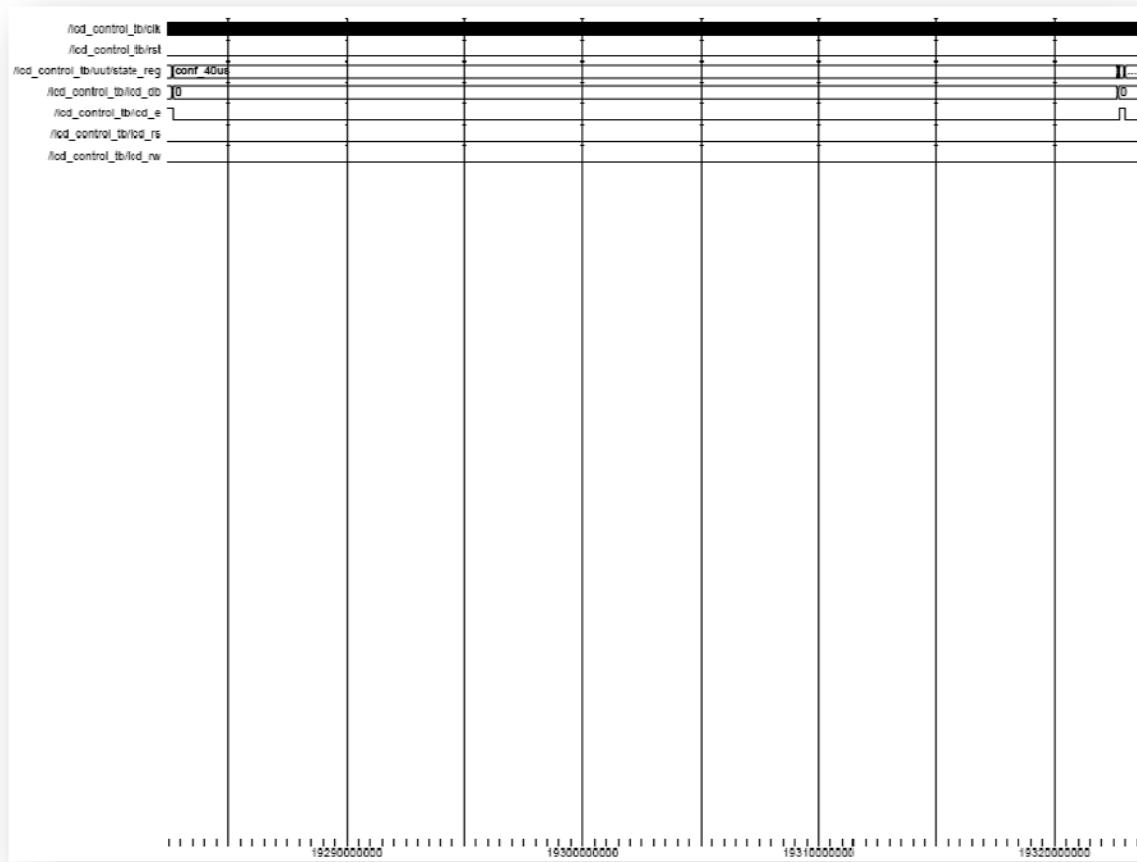


Figura 52. Estado conf_40us

Configuración y Bienvenida. Para realizar esta simulación se ha cambiado el tiempo de espera del parpadeo de la imagen. Se ha usado el tiempo de un clear 1.64ms para separar cada.

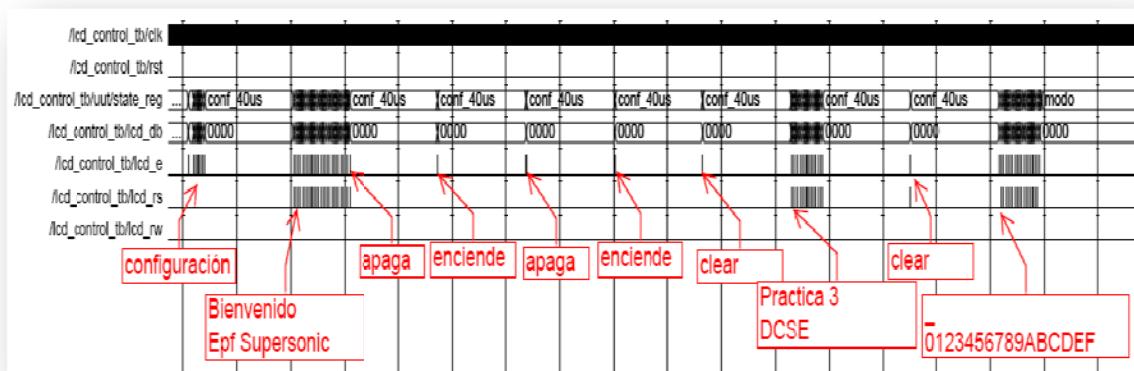


Figura 53. Visión global de la configuración y presentación inicial. Se han reducido los tiempos de espera para poder realizar la simulación.

Se presiona el botón norte.

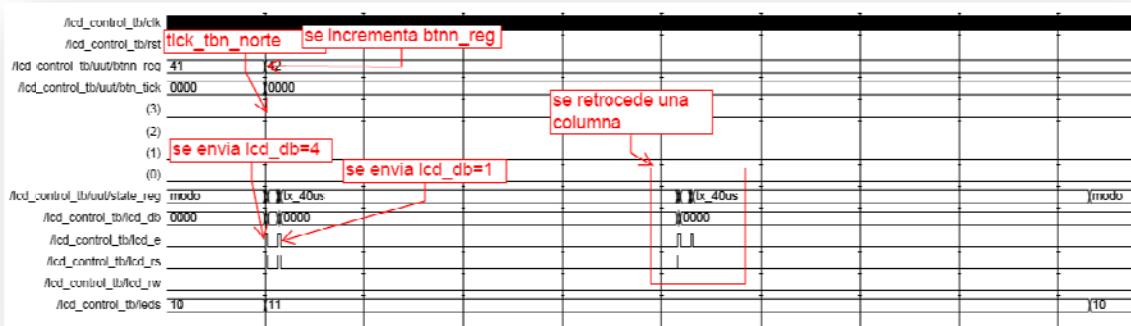


Figura 54. Se produce un evento en el botón norte, se envía el registro bttn_reg y se incrementa. Por último, si sw1=0 se envía el comando de retroceder una columna para que el cursor aparezca como si no hubiese avanzado.

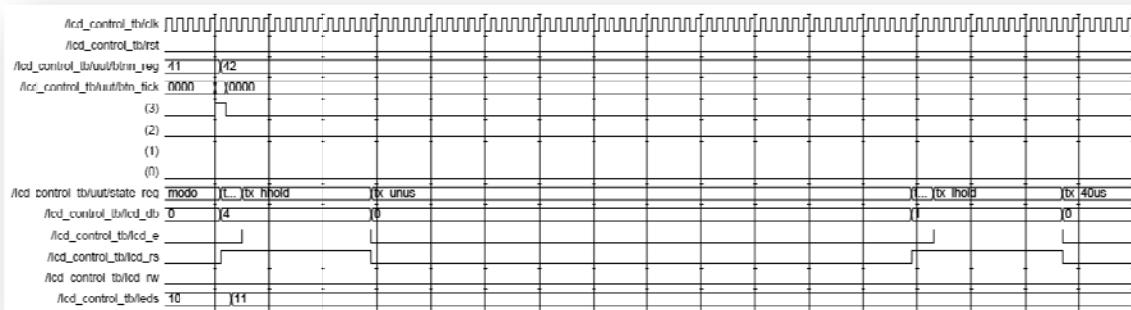


Figura 55. Envío del registro bttn_reg.

Para realizar esta simulación se han reducido los tiempos en la bienvenida a 1.64ms y se ha usado el siguiente test bench

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY LCD_control_tb IS
END LCD_control_tb;

ARCHITECTURE behavior OF LCD_control_tb IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT LCD_Control
PORT(
    clk : IN std_logic;
    rst : IN std_logic;
    sw : IN std_logic_vector(1 downto 0);
    btn_tick : IN std_logic_vector(3 downto 0);
    press_tick : IN std_logic;
    keyboard_tick : IN std_logic;
    rotary_tick : IN std_logic;
    rotary_lzq : IN std_logic;
    kb_data : IN std_logic_vector(7 downto 0);
    lcd_db : OUT std_logic_vector(3 downto 0);
    lcd_e : OUT std_logic;
    lcd_rs : OUT std_logic;
    lcd_rw : OUT std_logic;
    leds : OUT std_logic_vector(1 downto 0)
);

```

```
END COMPONENT;

--Inputs
signal clk : std_logic := '0';
signal rst : std_logic := '0';
signal sw : std_logic_vector(1 downto 0) := (others => '0');
signal btn_tick : std_logic_vector(3 downto 0) := (others => '0');
signal press_tick : std_logic := '0';
signal keyboard_tick : std_logic := '0';
signal rotary_tick : std_logic := '0';
signal rotary_izq : std_logic := '0';
signal kb_data : std_logic_vector(7 downto 0) := (others => '0');

--Outputs
signal lcd_db : std_logic_vector(3 downto 0);
signal lcd_e : std_logic;
signal lcd_rs : std_logic;
signal lcd_rw : std_logic;
signal leds : std_logic_vector(1 downto 0);

-- Clock period definitions
constant clk_period : time := 20ns;

BEGIN

  -- Instantiate the Unit Under Test (UUT)
  uut: LCD_Control PORT MAP (
    clk => clk,
    rst => rst,
    sw => sw,
    btn_tick => btn_tick,
    press_tick => press_tick,
    keyboard_tick => keyboard_tick,
    rotary_tick => rotary_tick,
    rotary_izq => rotary_izq,
    kb_data => kb_data,
    lcd_db => lcd_db,
    lcd_e => lcd_e,
    lcd_rs => lcd_rs,
    lcd_rw => lcd_rw,
    leds => leds
  );

  -- Clock process definitions
  clk_process :process
  begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
  end process;

  rst <= '1', '0' after clk_period;
  -- Stimulus process
  stim_proc: process
  begin

    wait for 40ms;

    wait until falling_edge(clk);
    btn_tick <= "1000";      -- boton norte
    wait until falling_edge(clk);
    btn_tick <= "0000";

    wait;
  end process;
```

Características temporales (frecuencia de reloj, retardos, etc.)

Todos los tiempos están calculados a partir de una señal de Reloj de 50MHz.
El retardo máximo es de 15.419ns (Frecuencia máxima: 64.855MHz)

Recursos utilizados de la FPGA

Design Statistics

IOs : 31

Cell Usage :

| | |
|---------------------|--------|
| # BELS | : 1896 |
| # GND | : 1 |
| # INV | : 193 |
| # LUT1 | : 37 |
| # LUT2 | : 47 |
| # LUT2_D | : 10 |
| # LUT2_L | : 5 |
| # LUT3 | : 234 |
| # LUT3_D | : 14 |
| # LUT3_L | : 15 |
| # LUT4 | : 703 |
| # LUT4_D | : 19 |
| # LUT4_L | : 25 |
| # MUXCY | : 268 |
| # MUXF5 | : 78 |
| # MUXF6 | : 16 |
| # MUXF7 | : 8 |
| # VCC | : 1 |
| # XORCY | : 222 |
| # FlipFlops/Latches | : 574 |
| # FD | : 7 |
| # FDC | : 291 |
| # FDCE | : 256 |
| # FDE | : 1 |
| # FDP | : 10 |
| # FDPE | : 8 |
| # FDR | : 1 |
| # Clock Buffers | : 1 |
| # BUFGP | : 1 |
| # IO Buffers | : 30 |
| # IBUF | : 13 |
| # OBUF | : 17 |

Device utilization summary:

Selected Device : 3s500efg320-4

Number of Slices: 699 out of 4656 15%

Number of Slice Flip Flops: 574 out of 9312 6%
Number of 4 input LUTs: 1302 out of 9312 13%
Number of IOs: 31
Number of bonded IOBs: 31 out of 232 13%
Number of GCLKs: 1 out of 24 4%