

Curso de Especialización en Sistemas Embebidos

Microarquitecturas y Softcores

12va Cohorte

2021

***Trabajo Práctico Final***

Alumno:

Cristian Trinidad

Historial de cambios

|  |  |  |
| --- | --- | --- |
| **Fecha** | **Versión** | **Cambio** |
| 17/04/2021 | 1.0 | Versión Inicial |
|  |  |  |
|  |  |  |

# 

Índice

[1 Introducción 4](#_Toc73380524)

[1.1 Propósito 4](#_Toc73380525)

[1.2 Alcance 4](#_Toc73380526)

[2 Breve descripción del driver de display implementado en CLP 5](#_Toc73380527)

[2.1 Interfaz de E/S del display 5](#_Toc73380528)

[2.2 Interfaz de E/S del módulo lcd\_controller 6](#_Toc73380529)

[3 Implementación del wrapper 8](#_Toc73380530)

[3.1 Interfaz de E/S del wrapper 8](#_Toc73380531)

[3.2 Registros de lectura/escritura 10](#_Toc73380532)

[4 Simulaciones 12](#_Toc73380533)

[4.1 Testbench 12](#_Toc73380534)

[4.2 Operación de escritura en el bus 13](#_Toc73380535)

[4.3 Operación de lectura en el bus 14](#_Toc73380536)

[4.4 Operación normal 15](#_Toc73380537)

[5 Implementación en Quartus 16](#_Toc73380538)

[5.1 Esquemático 16](#_Toc73380539)

[5.2 Pines de entrada/salida 17](#_Toc73380540)

[5.3 Recursos de la FPGA 19](#_Toc73380541)

[5.4 Firmware 20](#_Toc73380542)

# Introducción

# Propósito

En este trabajo se propone tomar el driver LCD para display alfanumérico con controlador Hitachi HD44780 implementado en CLP y comunicarlos con el softcore Nios II a través del bus Avalon. Para tal fin, se va a utilizar el kit de desarrollo DE1-SoC rev C que contiene una FPGA de Altera Cyclone V modelo 5CSEMA5F31C6N.

# Alcance

Se implementará:

1. El wrapper necesario para establecer la comunicación con el controlador.
2. La generación de un IP a partir del wrapper y los archivos vhd diseñados en CLP.
3. Validación del wrapper a través de simulaciones utilizando ghdl.
4. La integración del IP con el controlador Nios II utilizando Quartus.
5. La implementación de rutinas básicas en C para el manejo del IP a través del controlador.
6. La prueba sobre display real para mostrar su funcionamiento.

## 

# Breve descripción del driver de display implementado en CLP

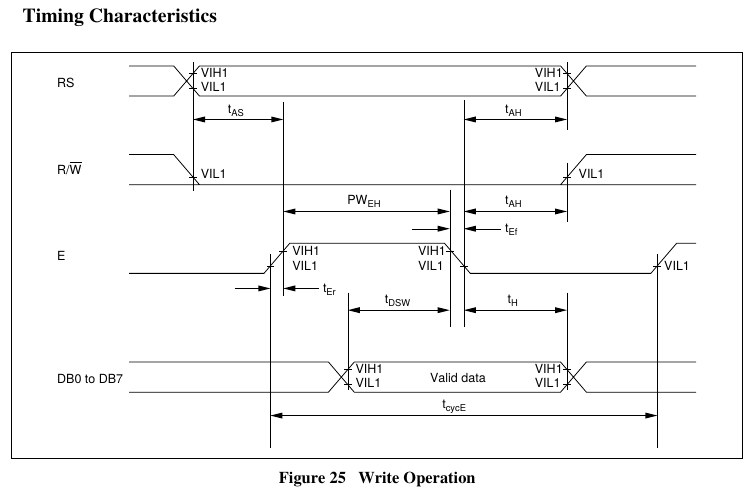
# Interfaz de E/S del display

|  |  |  |
| --- | --- | --- |
| **Pin** | **Símbolo** | **Función** |
| 1 | VSS | Power Ground |
| 2 | VDD | Power supply for logic circuit(+5V) |
| 3 | V0 | For LCD drive voltage (variable) |
| 4 | RS(C/D) | H: Display Data,  L: Display Instruction |
| 5 | R/W | H: Data Read (LCM to MPU) ;  L: Data Write (MPU to LCM) |
| 6 | EN | Enable signal.  Write mode (R/W = L) data of DB<0:7> is latched at the falling edge of E.  Read mode (R/W = H) DB<0:7> appears the reading data while E is at high level |
| 7-14 | DB0-DB7 | Data bus |
| 15 | A | Power for LED Backlight (+V) |
| 16 | K | Power for LED Backlight (Ground) |

El display cuenta con una interfaz paralela de 8-bits (DB0-7) la cual puede ser utilizada en formato 4-bits haciendo 2 escrituras o lecturas.

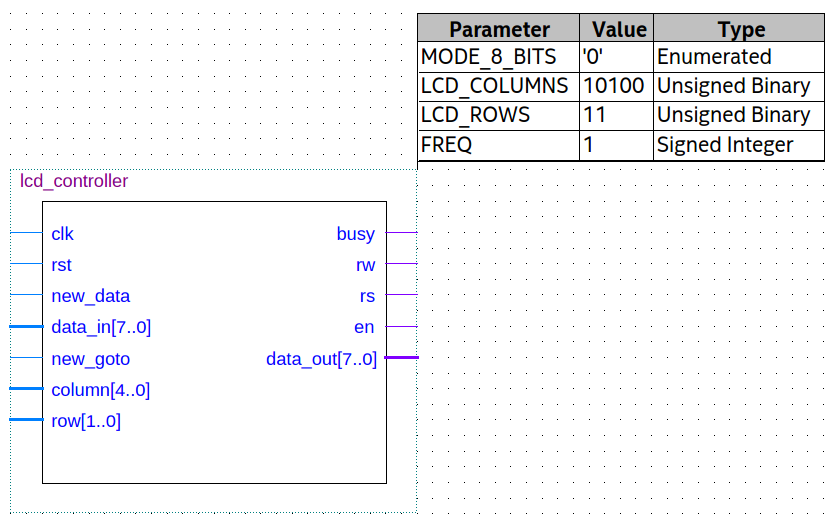
Para el manejo del display se deben comandar las señales RS, R/W, EN y DB0-7.

Ejemplo del manejo de las señales del display:



# Interfaz de E/S del módulo lcd\_controller

Para comandar las señales del LCD se implementó el siguiente controlador en CLP:



entity lcd\_controller is

generic (

MODE\_8\_BITS : std\_logic := '1'; -- 8-bits or 4-bits

LCD\_COLUMNS : std\_logic\_vector(4 downto 0) := "10100"; -- 20

LCD\_ROWS : std\_logic\_vector(1 downto 0) := "11"; -- 4

FREQ : integer := 1 -- system clock frequency in MHz

);

port (

clk : in std\_logic; --system clock

rst : in std\_logic; --reset

new\_data : in std\_logic; --new data\_in valid

data\_in : in std\_logic\_vector(7 downto 0); --data

new\_goto : in std\_logic; --new column and row valid

column : in std\_logic\_vector(4 downto 0); -- characters in a row

row : in std\_logic\_vector(1 downto 0); -- row number

busy : out std\_logic; --lcd controller busy

rw, rs, en : out std\_logic; --read/write, setup/data, and enable for lcd

data\_out : out std\_logic\_vector(7 downto 0)); --data output to LCD

end;

**Genéricos/parámetros:**

MODE\_8\_BITS: se utiliza para configurar el modo de operación en 4 u 8-bits.

LCD\_COLUMNS: número de columnas/caracteres del display a utilizar.

LCD\_ROWS: número de filas del display a utilizar.

FREQ. Frecuencia de operación del bloque, se utiliza para calcular los tiempos de espera.

**Entradas:**

*clk*: señal de reloj del bloque.

*rst*: señal de reset.

*new\_data*: nuevo carácter valido en el puerto data\_in.

*data\_in*: carácter ASCII a escribir.

*new\_goto*: nuevo pedido de posicionamiento del cursor a través de column y row.

*column*: columna donde se desea posicionar el cursor.

*row*: fila donde se desea posicionar el cursor.

**Salidas:**

*busy*: controlador ocupado.

*rw, rs, en*: señales de control al display LCD.

*data\_out*: salida paralelo al display LCD.

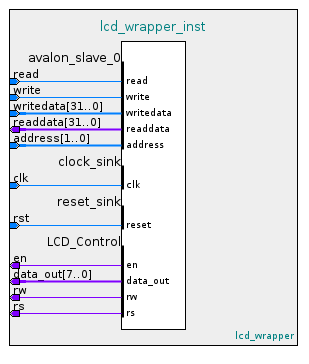
El controlador implementado en CLP realiza 3 funciones:

1. Inicialización del display.
2. Escritura de un carácter.
3. Función de posicionamiento del cursor en el display.

# Implementación del wrapper

# Interfaz de E/S del wrapper

A partir del bloque diseñado en CLP se plantea el siguiente wrapper como trabajo para MyS, se utiliza el bus Avalon compatible con el controlador Nios II:



entity lcd\_controller\_wrapper is

generic (

MODE\_8\_BITS : std\_logic := '1'; -- 8-bits or 4-bits

LCD\_COLUMNS : std\_logic\_vector(4 downto 0) := "10100"; -- 20

LCD\_ROWS : std\_logic\_vector(1 downto 0) := "11"; -- 4

FREQ : integer := 1 -- system clock frequency in MHz

);

port (

clk, rst : in std\_logic;

-- Avalon bus

read, write : in std\_logic;

address : in std\_logic\_vector(1 downto 0);

writedata : in std\_logic\_vector(31 downto 0);

readdata : out std\_logic\_vector(31 downto 0);

-- LCD outputs

rw, rs, en : out std\_logic; --read/write, setup/data, and enable for lcd

data\_out : out std\_logic\_vector(7 downto 0) --data output to LCD

);

end lcd\_controller\_wrapper;

Este wrapper instancia el módulo de CLP y toma sus mismos parámetros:

* MODE\_8\_BITS: se utiliza para configurar el modo de operación en 4 u 8-bits.
* LCD\_COLUMNS: número de columnas/caracteres del display a utilizar.
* LCD\_ROWS: número de filas del display a utilizar.
* FREQ. Frecuencia de operación del bloque, se utiliza para calcular los tiempos de espera.

Incorpora las señales necesarias para el uso de bus Avalon:

* **Entradas:**
  + *read, write*: bits que indican el pedido de una lectura o escritura desde el bus.
  + *address*: direccion offset dentro del IP para el pedido de W/R. Se utilizó un ancho de palabra de 2bits.
  + *writedata*: datos de 32 bits a escribir en el bloque wrapper cuando se recibe un pedido de escritura a través de la entrada *write*.
* **Salidas:**
  + *readdata:* datos de 32 bits a leer proporcionados por el wrapper cuando recibe un pedido de lectura a través de la entrada *read*.

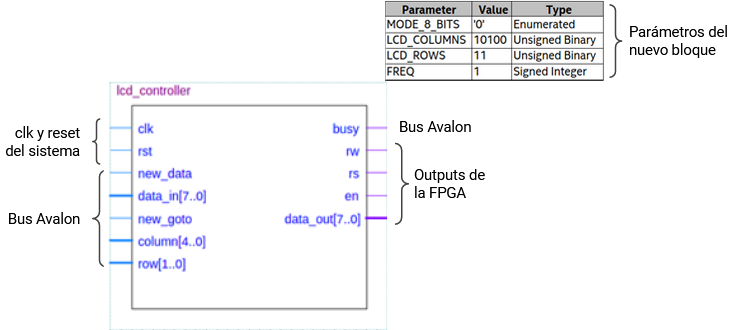
Las salidas siguientes se incluyeron en una interfaz llamada LCD\_Control y salen como I/Os del wrapper para ser conectadas en los pines de la FPGA:

* *rw, rs, en*: señales de control al display LCD.
* data\_out: salida paralelo al display LCD.

Las siguientes señales del controlador LCD de CLP son ahora manejadas en el wrapper a través de los pedidos del bus Avalon:

* **Entradas:**
  + *new\_data*: nuevo carácter valido en el puerto data\_in.
  + *data\_in*: carácter ASCII a escribir.
  + *new\_goto*: nuevo pedido de posicionamiento del cursor a través de column y row.
  + *column*: columna donde se desea posicionar el cursor.
  + *row*: fila donde se desea posicionar el cursor.
* **Salidas:**
  + *busy*: controlador ocupado.

En la siguiente figura se resume como se mapean las señales del bloque de CLP en el nuevo bloque planteado:



# Registros de lectura/escritura

Se implementaron 3 registros de 32bits para acceder al IP desde el bus Avalon:

1. CHAR: registro para escribir un carácter (write-only) – Offset 0x0

* Bits 0-7 *data\_in*: carácter a imprimir en el display
* Bits 8-31: no utilizados

Al escribir el registro CHAR con el carácter a mandar en el display, el wrapper automáticamente copia el carácter en los bits 0-7 de la señal ***writedata*** del bus Avalon a la entrada ***data\_in*** del bloque ***lcd\_controller***. Adicionalmente, el wrapper debe poner en 1 la señal ***new\_data*** para decirle al bloque ***lcd\_controller*** que debe enviar el carácter en su entrada ***data\_in*** al display.

1. POSITION: Registro para posicionar el cursor (write-only) – Offset 0x1

* Bits 0-1 *row*: fila donde se desea posicionar el cursor ()
* Bits 2-6 *column*: columna donde se desea posicionar el cursor
* Bits 7-31: no utilizados

Al escribir el registro POSITION con el los valores de *row* y *column,*  el wrapper realiza una acción similar a la anterior, copia los valores de la señal ***writedata*** del bus Avalon a las entradas ***row*** y ***column*** del bloque ***lcd\_controller***. Luego pone en 1 la señal ***new\_goto*** para decirle al bloque ***lcd\_controller*** que debe enviar el comando de posicionar el cursor con los datos en las entradas ***row*** y ***column*** al display.

1. STATUS: Registro de estado (read-only) – Offset 0x2

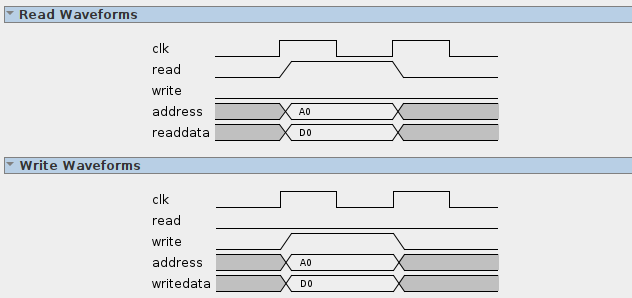
* Bit 0 *busy*: controlador ocupado.
* Bits 1-31: no utilizados

Por último, el registro STATUS es utilizado por el software para, a través del bus Avalon, consultar el estado del controlador. En este caso el estado corresponde al bit ***busy*** que indica si el controlador está ocupado enviando un carácter o posicionando el cursor. No se puede enviar otro pedido (nuevo carácter o nuevo pedido de posicionamiento del cursor) hasta que ***busy*** sea 0.

# Simulaciones

# Testbench

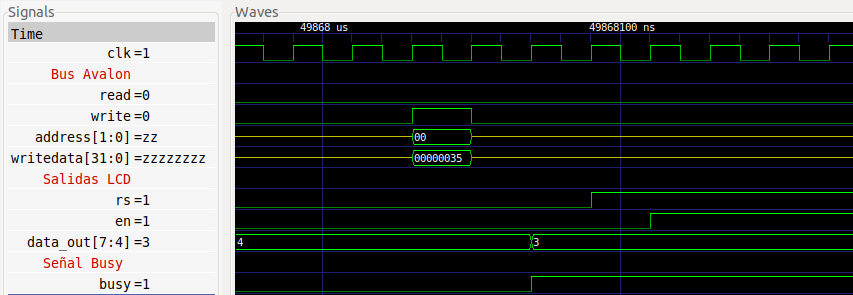
A partir de la documentación de Quartus del bus Avalon se obtuvieron las siguientes formas de ondas utilizadas para la implementación y luego para el testbench:



Se modeló el bus y se verificó que el nuevo sistema implementado responda como lo hacía el bloque implementado en CLP pero ahora manejado por el bus Avalon.

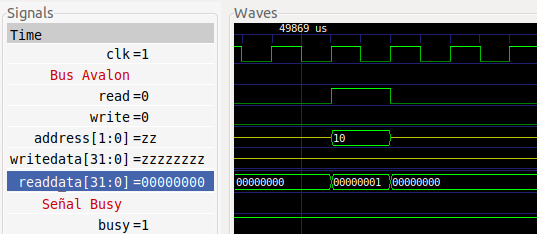
Otro detalle incluido en las simulaciones es la frecuencia de 50Mhz que ahora ingresa al módulo de CLP, ya que este fue probado con 1Mhz en la materia anterior. En el testbench modifiqué la frecuencia y verifiqué que las formas de onda aún respeten los tiempos requeridos por el display.

# Operación de escritura en el bus

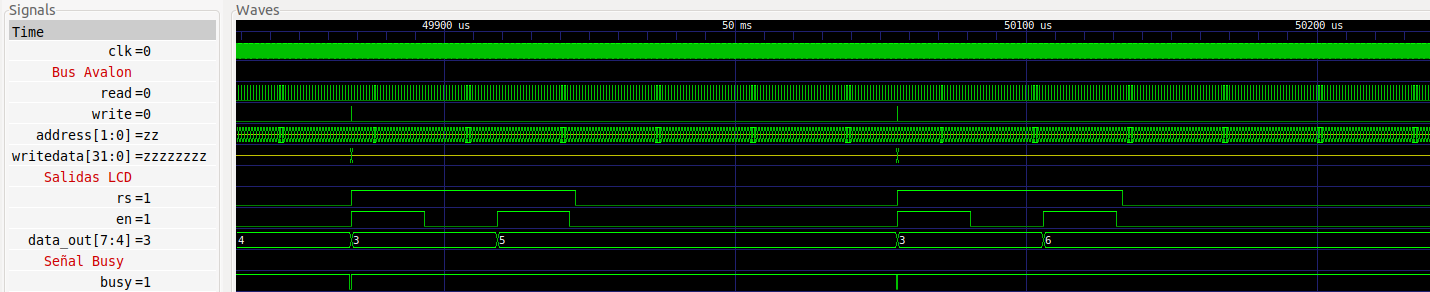
En la figura se puede observar las señales del bus Avalon (**read**, **write**, **address** y **writedata**) **address** y **writedata** están en Z antes y después del pedido de escritura o lectura.

Cuando aparece el pedido de escritura de un carácter se apunta la señal **address** a 0x00. En este caso el dato a escribir es 0x35, se puede ver que luego de un clk se levanta la señal de **busy** del lcd\_controller y las señales de **rs**, en y **data\_out** comienzan a tener actividad. En particular **data\_out** saca la parte alta del dato escrito 0x3 ya que se está trabajando en 4-bits.

# Operación de lectura en el bus

La operación de lectura es similar, pero en este caso se levanta el bit **read** del bus Avalon y el bus de direcciones apunta al registro de STATUS (0x10), en ese momento el warpper escribe en la señal **readdata** el valor del bit busy (‘1’).

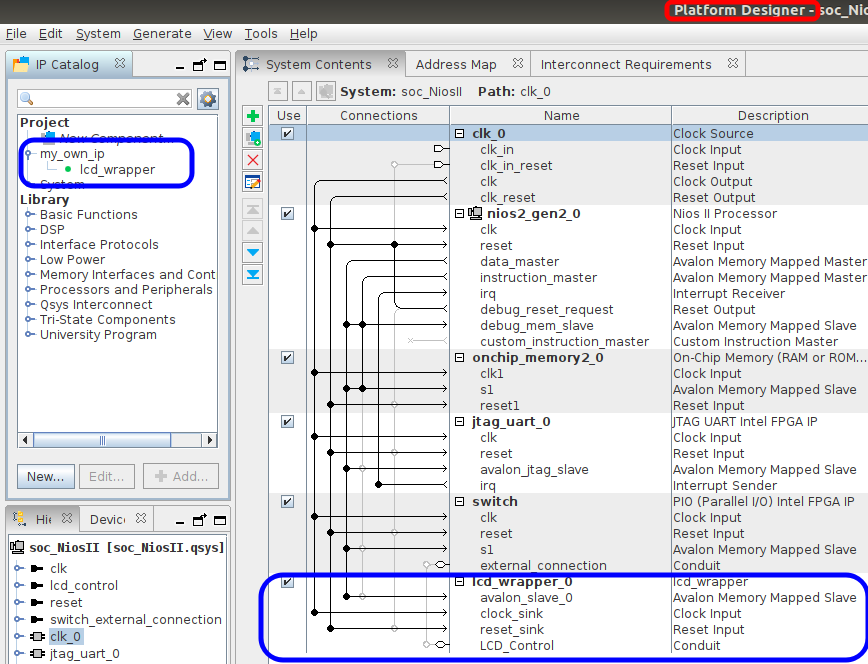
# Operación normal

En esta simulación se muestra un zoom out del caso de la escritura del carácter 0x35 para mostrar cómo el dato se manda en dos tandas, primero 0x3 y luego 0x5. Se observa el comportamiento de todas las señales de salida al LCD, el cual es el mismo que en el trabajo de CLP.

Por otro lado, se observa una gran actividad en la línea **read** y **address** del bus Avalon, esto se debe a que el testbench está preguntando todo el tiempo por el bit **busy** para saber cuándo puede enviar otro carácter. La implementación trabaja por pulleo.

# Implementación en Quartus

# Esquemático

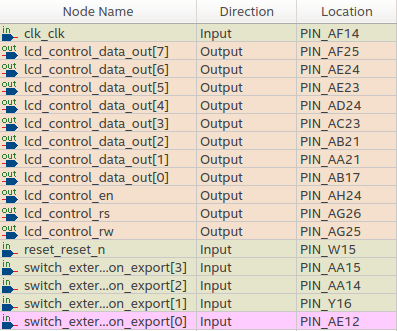
En la siguiente figura se observa el esquemático planteado en Quartus utilizando Platform Designer:

**Bloque instanciados:**

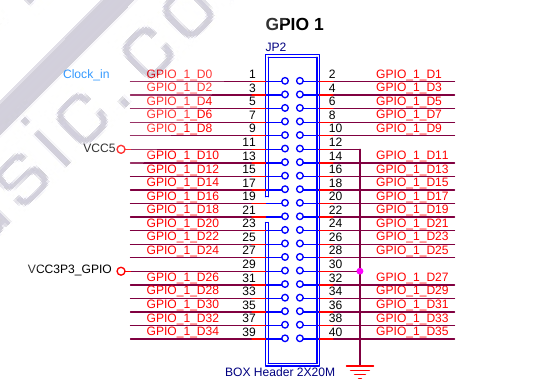
* Clock Source (clk\_0): utilizando una frecuencia de 50Mhz.
* Nios II Processor (nios2\_gen2\_0): procesador softcore.
* On-Chip RAM Memory
* JTAG UART: utilizado en software para poder acceder al JTAG e imprimir valores por la consola.
* PIO – Parallel I/O: para poder leer los switches de la placa de desarrollo.
* lcd\_wrapper: bloque diseñado para la materia.

# Pines de entrada/salida

Como se comentó, se utiliza la placa DE1-SoC rev C. La imagen de Pin Planner muestra los I/O utilizados:



La siguiente figura muestra el conector JP2 para los GPIO\_1:



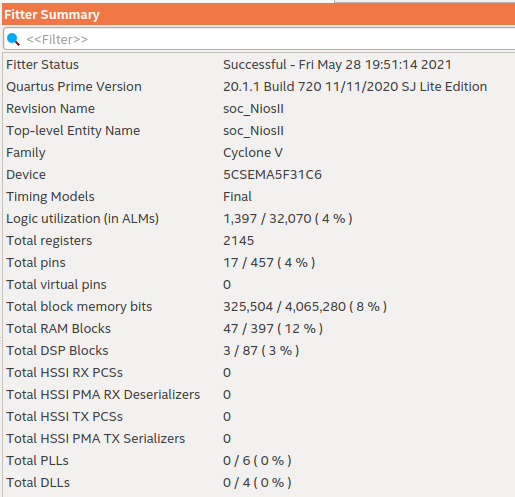
Se cuenta con un display de 20x4 líneas del tipo 2004A que está conectado para ser usado con datos de 4-bits.



Conexionado entre display y conector JP2:

|  |  |  |  |
| --- | --- | --- | --- |
| **Display** | | **Kit de desarrollo** | |
| **Pin** | **Función** | **Pin de JP2** | **Función** |
| 1 | VSS | 30 | GND |
| 2 | VDD | 29 | VCC3P3\_GPIO |
| 3 | V0 | 30 | GND |
| 4 | RS(C/D) | 13 | GPIO\_1\_D10 |
| 5 | R/W | 30 | GND |
| 6 | E | 14 | GPIO\_1\_D11 |
| 11 | DB4 | 5 | GPIO\_1\_D4 |
| 12 | DB5 | 6 | GPIO\_1\_D5 |
| 13 | DB6 | 7 | GPIO\_1\_D6 |
| 14 | DB7 | 8 | GPIO\_1\_D7 |
| 15 | A | 30 | GND |
| 16 | K | 29 | VCC3P3\_GPIO |

# Recursos de la FPGA



# Firmware

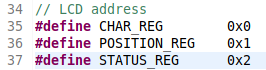
Para el firmware se utilizó la herramienta “Nios II Software build tools from Eclipse” desde Quartus.

Para la implementación se deben incluir los siguientes .h generados por la herramienta:

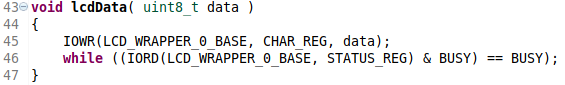


* system.h incluye las direcciones mapeadas en el mapa de memoria del controlador para los IPs colocados en Platform Designer. Para el caso del wrapper define la macro LCD\_WRAPPER\_0\_BASE.
* io.h provee las macros necesarias para leer/escribir en las IPs. Estas macros son IOWR y IORD.

Por otro lado, se definieron en el código las siguientes macros con los offsets de los 3 registros del wrapper:

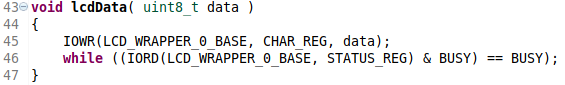


A continuación se muestra la implementación de una función de lectura al registro STATUS utilizando las macros anteriores y las provistas en system.h y io.h:

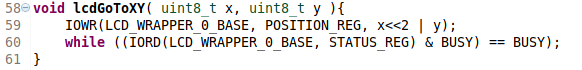


A partir de todo esto se armaron 2 funciones para acceder al wrapper:

1. **lcdData** para escribir un carácter:

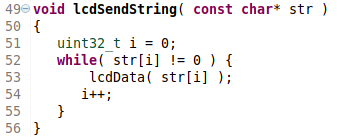


1. **lcdGoToXY** para posicionar el cursor:



Se puede observar en las funciones que luego de un pedido se tiene un while esperando que el bit **busy** sea cero nuevamente.

Luego se armó una función para enviar un string (**lcdSendString**) que básicamente llama repetidamente a la función **lcdData** para enviar los caracteres pedidos:



Por último, en el main se llaman las funciones **lcdGoToXY** y **lcdSendString** repetidas veces para escribir lo que se desee, a continuación se muestra un ejemplo:

