



FACULTAD DE INGENIERIA

Universidad de Buenos Aires

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
1.1	Propósito	4
1.2	Alcance	4
2	Breve descripción del driver de display implementado en CLP	5
2.1	Interfaz de E/S del display	5
2.2	Interfaz de E/S del módulo lcd_controller	6
3	Implementación del wrapper	8
3.1	Interfaz de E/S del wrapper	8
3.2	Registros de lectura/escritura	10
4	Simulaciones	12
4.1	Testbench	12
4.2	Operación de escritura en el bus	13
4.3	Operación de lectura en el bus	14
4.4	Operación normal	15
5	Implementación en Quartus	16
5.1	Esquemático	16
5.2	Pines de entrada/salida	17
5.3	Recursos de la FPGA	19
5.4	Firmware	20

1 Introducción

1.1 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.

1.2 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.

2 Breve descripción del driver de display implementado en CLP

2.1 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:

Timing Characteristics

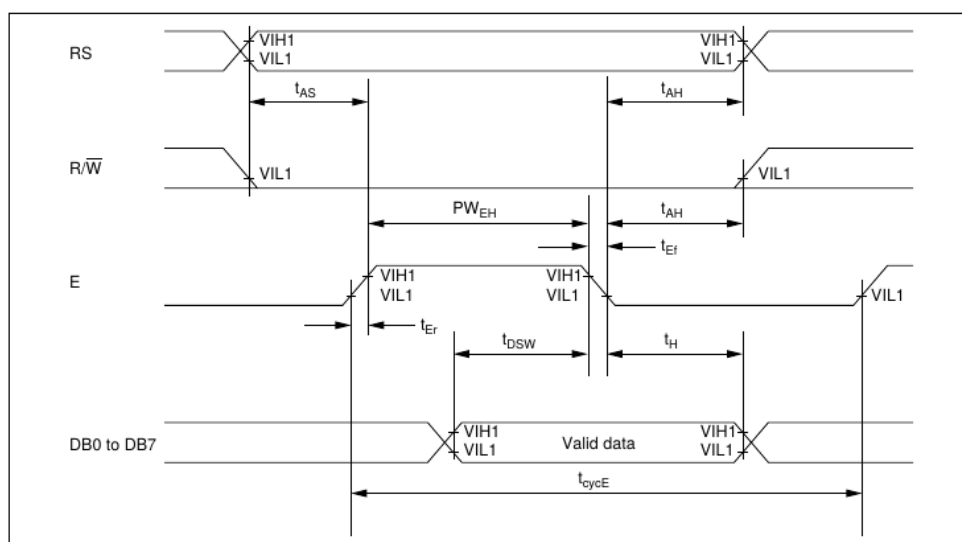
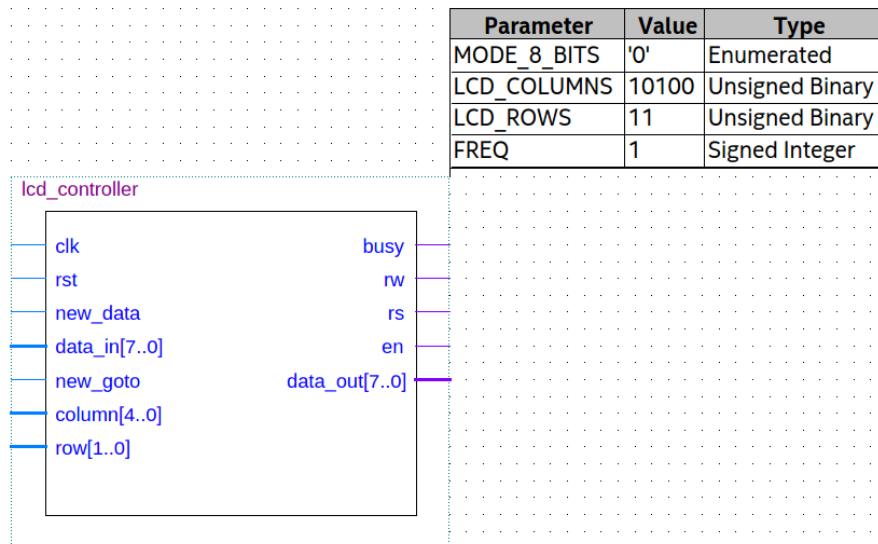


Figure 25 Write Operation

2.2 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.

*Trabajo práctico Final***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.

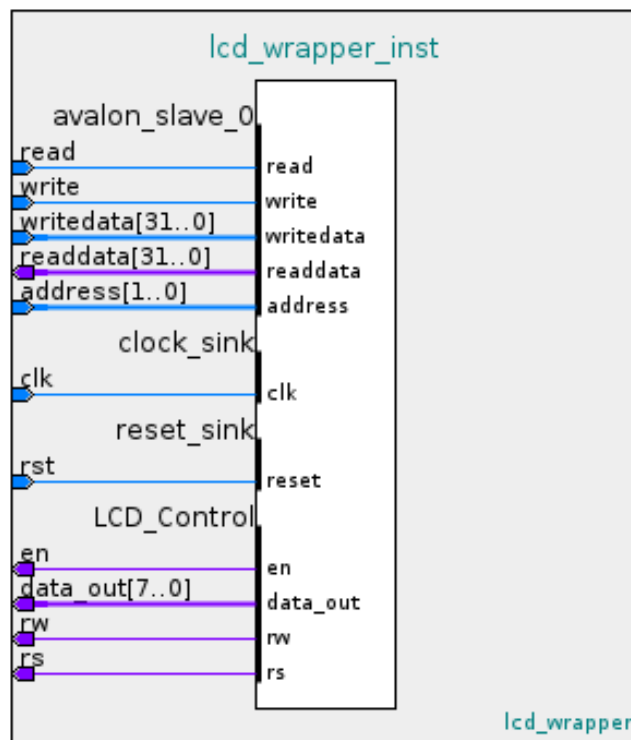
El controlador implementado en CLP realiza 3 funciones:

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

3 Implementación del wrapper

3.1 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:

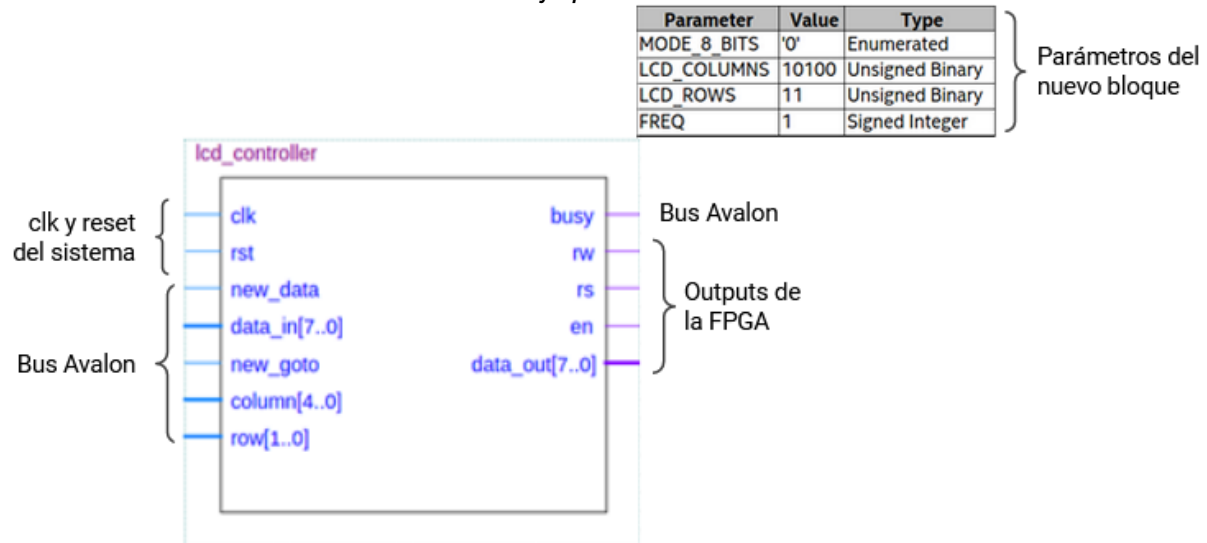
- *nw, rs, en:* señales de control al display LCD.
- *data_out:* salida paralelo al display LCD.

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:

Trabajo práctico Final



3.2 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.

2. 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 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.

3. 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

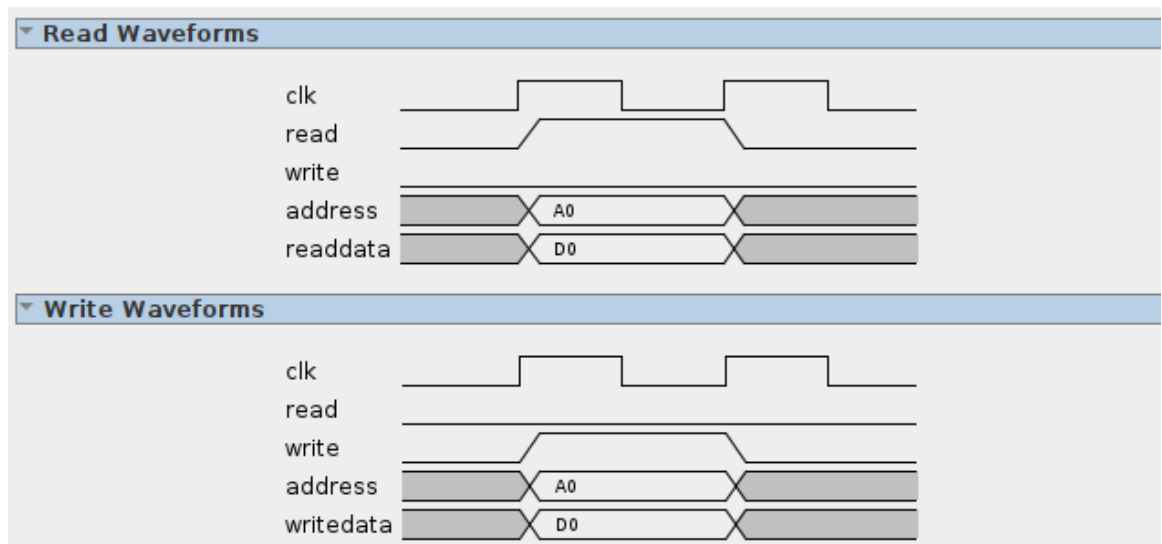
Trabajo práctico Final

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.

4 Simulaciones

4.1 Testbench

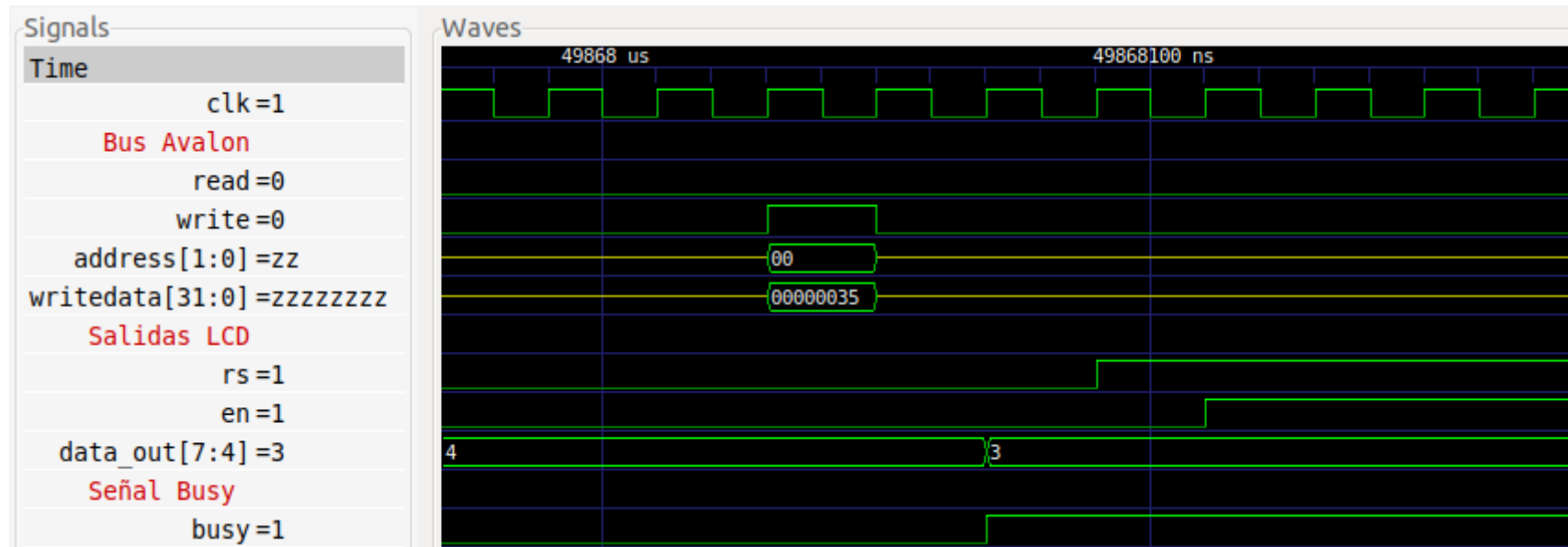
A partir de la documentación de Quartus del bus Avalon se obtuvieron las siguientes formas de ondas utilizadas para la implementación y luego para el testbench:



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.

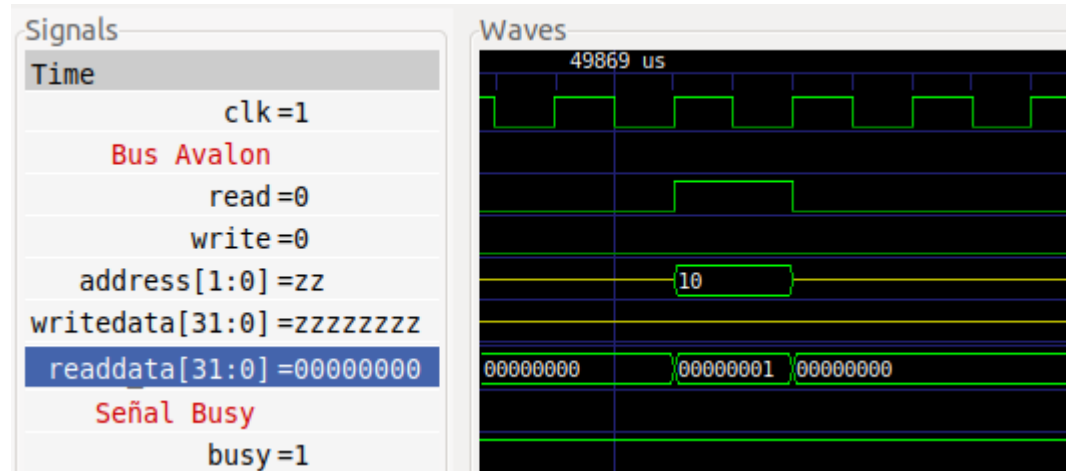
4.2 Operación de escritura en el bus



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.

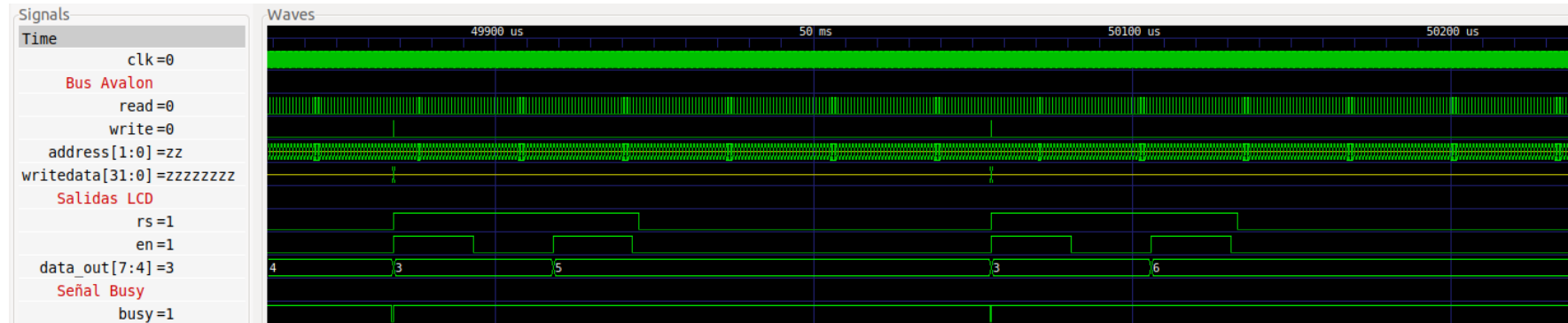
Cuando aparece el pedido de escritura de un carácter se apunta la señal **address** a 0x00. En este caso el dato a escribir es 0x35, se puede ver que luego de un clk se levanta la señal de **busy** del lcd_controller y las señales de **rs**, **en** y **data_out** comienzan a tener actividad. En particular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

4.3 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').

4.4 Operación normal



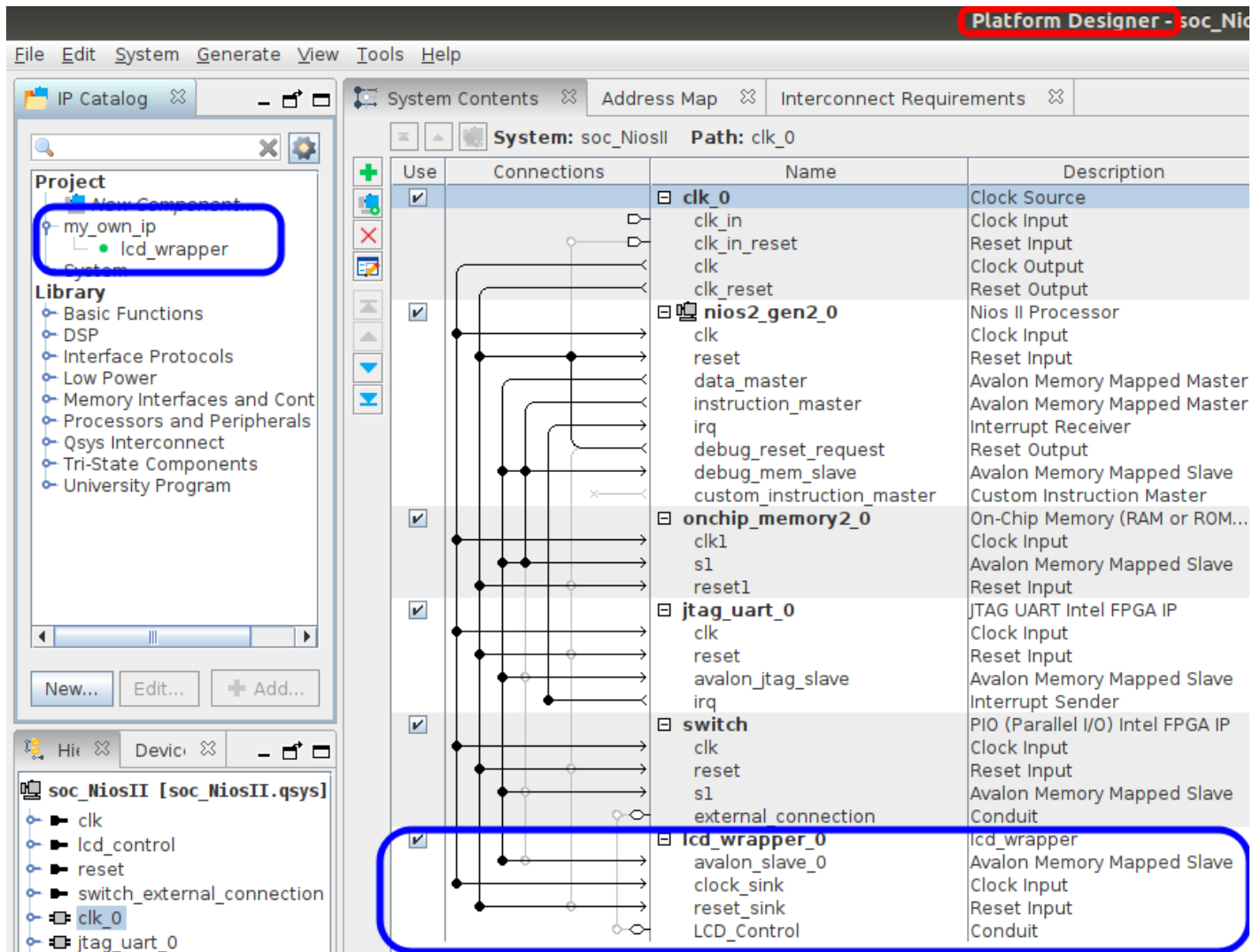
En esta simulación se muestra un zoom out del caso de la escritura del carácter 0x35 para mostrar cómo el dato se manda en dos tandas, primero 0x3 y luego 0x5. Se observa el comportamiento de todas las señales de salida al LCD, el cual es el mismo que en el trabajo de CLP.

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.

5 Implementación en Quartus

5.1 Esquemático

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



Bloque instanciados:

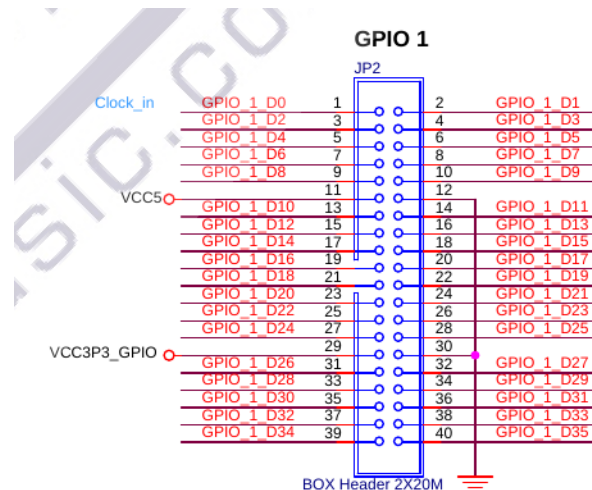
- Clock Source (clk_0): utilizando una frecuencia de 50Mhz.
- Nios II Processor (nios2_gen2_0): procesador softcore.
- On-Chip RAM Memory
- JTAG UART: utilizado en software para poder acceder al JTAG e imprimir valores por la consola.
- PIO – Parallel I/O: para poder leer los switches de la placa de desarrollo.
- lcd_wrapper: bloque diseñado para la materia.

5.2 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:

Node Name	Direction	Location
in clk_clk	Input	PIN_AF14
out lcd_control_data_out[7]	Output	PIN_AF25
out lcd_control_data_out[6]	Output	PIN_AE24
out lcd_control_data_out[5]	Output	PIN_AE23
out lcd_control_data_out[4]	Output	PIN_AD24
out lcd_control_data_out[3]	Output	PIN_AC23
out lcd_control_data_out[2]	Output	PIN_AB21
out lcd_control_data_out[1]	Output	PIN_AA21
out lcd_control_data_out[0]	Output	PIN_AB17
out lcd_control_en	Output	PIN_AH24
out lcd_control_rs	Output	PIN_AG26
out lcd_control_rw	Output	PIN_AG25
in reset_reset_n	Input	PIN_W15
in switch_exter...on_export[3]	Input	PIN_AA15
in switch_exter...on_export[2]	Input	PIN_AA14
in switch_exter...on_export[1]	Input	PIN_Y16
in switch_exter...on_export[0]	Input	PIN_AE12

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.

Trabajo práctico Final

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

5.3 Recursos de la FPGA

Fitter Summary	
 <<Filter>>	
Fitter Status	Successful - Fri May 28 19:51:14 2021
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	soc_NiosII
Top-level Entity Name	soc_NiosII
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	1,397 / 32,070 (4 %)
Total registers	2145
Total pins	17 / 457 (4 %)
Total virtual pins	0
Total block memory bits	325,504 / 4,065,280 (8 %)
Total RAM Blocks	47 / 397 (12 %)
Total DSP Blocks	3 / 87 (3 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

5.4 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:

```
#include "system.h"
#include "io.h"
```

- 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:

```
// LCD address
#define CHAR_REG      0x0
#define POSITION_REG   0x1
#define STATUS_REG    0x2
```

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:

```
IORD(LCD_WRAPPER_0_BASE, STATUS_REG)
```

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

- 1) **lcdData** para escribir un carácter:

```
void lcdData( uint8_t data )
{
    IOWR(LCD_WRAPPER_0_BASE, CHAR_REG, data);
    while ((IORD(LCD_WRAPPER_0_BASE, STATUS_REG) & BUSY) == BUSY);
}
```

- 2) **lcdGoToXY** para posicionar el cursor:

```
void lcdGoToXY( uint8_t x, uint8_t y ){
    IOWR(LCD_WRAPPER_0_BASE, POSITION_REG, x<<2 | y);
    while ((IORD(LCD_WRAPPER_0_BASE, STATUS_REG) & BUSY) == BUSY);
}
```

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:

```
void lcdSendString( const char* str )
{
    uint32_t i = 0;
    while( str[i] != 0 ) {
        lcdData( str[i] );
        i++;
    }
}
```

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:

```
lcdGoToXY( 0, 0 ); // x = 0, y = 0
lcdSendString(MSG_1_1);
lcdGoToXY( 0, 1 ); // x = 0, y = 1
lcdSendString(MSG_1_2);
lcdGoToXY( 0, 2 ); // x = 0, y = 2
lcdSendString(MSG_1_3);
lcdGoToXY( 0, 3 ); // x = 0, y = 3
lcdSendString(MSG_1_4);
```