

Curso de Especialización en Sistemas Embebidos

Circuitos Lógicos Programables

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](#_Toc69582065)

[1.1 Propósito 4](#_Toc69582066)

[1.2 Alcance 4](#_Toc69582067)

[2 Breve descripción del display 5](#_Toc69582068)

[2.1 Interfaz de E/S 5](#_Toc69582069)

[2.2 Modos de comunicación 5](#_Toc69582070)

[2.3 Rutina de inicialización 7](#_Toc69582071)

[2.4 LCD Timing 8](#_Toc69582072)

[3 Implementación 10](#_Toc69582073)

[3.1 Introducción 10](#_Toc69582074)

[3.2 Interfaz de E/S del módulo lcd\_controller 10](#_Toc69582075)

[3.3 Constantes para instrucciones 11](#_Toc69582076)

[3.4 Constantes para temporización 12](#_Toc69582077)

[3.5 Arquitectura del módulo lcd\_controller 13](#_Toc69582078)

[3.6 Máquina de estados principal 13](#_Toc69582079)

[3.7 Interfaz de E/S del módulo lcd\_write 15](#_Toc69582080)

[3.8 Máquina de estado del módulo lcd\_write 17](#_Toc69582081)

[4 Simulaciones 18](#_Toc69582082)

[4.1 Salida de reset e inicialización (modo 4-bits) 18](#_Toc69582083)

[4.2 Espera de 40ms (medición) 19](#_Toc69582084)

[4.3 Espera de 5ms después de primer instrucción SET (data out = 0x30) 19](#_Toc69582085)

[4.4 Escritura de un carácter en modo 4-bits 20](#_Toc69582086)

[4.5 Escritura de una instrucción/función en modo 4-bits 21](#_Toc69582087)

[4.6 Escritura de un carácter en modo 8-bits 22](#_Toc69582088)

[4.7 Escritura de una instrucción/función en modo 8-bits 23](#_Toc69582089)

[5 Implementación en Quartus 24](#_Toc69582090)

[5.1 Esquemático 24](#_Toc69582091)

[5.2 Pines de entrada/salida 25](#_Toc69582092)

[5.3 Warnings 27](#_Toc69582093)

[5.4 Recursos de la FPGA 29](#_Toc69582094)

[5.5 Simulaciones en FPGA 33](#_Toc69582095)

[5.5.1 Salida de reset, inicialización (modo 4-bits) y espera de 40ms 33](#_Toc69582096)

[5.5.2 Zoom secuencia de inicialización 34](#_Toc69582097)

[5.5.3 Escritura de un carácter en modo 4-bits 35](#_Toc69582098)

[5.5.4 Escritura de una instrucción/función en modo 4-bits 36](#_Toc69582099)

# Introducción

# Propósito

En este trabajo se propone implementar un driver para un display alfanumérico con controlador Hitachi HD44780. 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. Las rutinas de inicialización del display para 4 y 8-bits. Se utilizaran macros que pueden ser modificadas para cambiar las opciones de inicialización.
2. Posibilidad de operar el display con interface de 4-bits o 8-bits a través de un parámetro al bloque principal.
3. Escritura de un carácter.
4. Función de posicionamiento del cursor en el display.
5. Se puede configurar el tamaño del display a utilizar por medio de parámetros en el bloque principal.

Este controlador fue implementado en software usando C en el ESP32, ya que es parte de mi trabajo final, por la tanto lo voy a utilizar como guía para implementar en hardware (VHDL) rutinas que realicen la misma funcionalidad.

La inicialización del display ocurre automáticamente después del reset del driver. Luego de esto, el bloque que lo instancia puede escribir caracteres y/o posicionar el cursor donde se desee escribir.

El driver cuenta con una salida de busy para indicar que se está inicializando o procesando algún pedido.

## 

# Breve descripción del display

# Interfaz de E/S

|  |  |  |
| --- | --- | --- |
| **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.

# Modos de comunicación

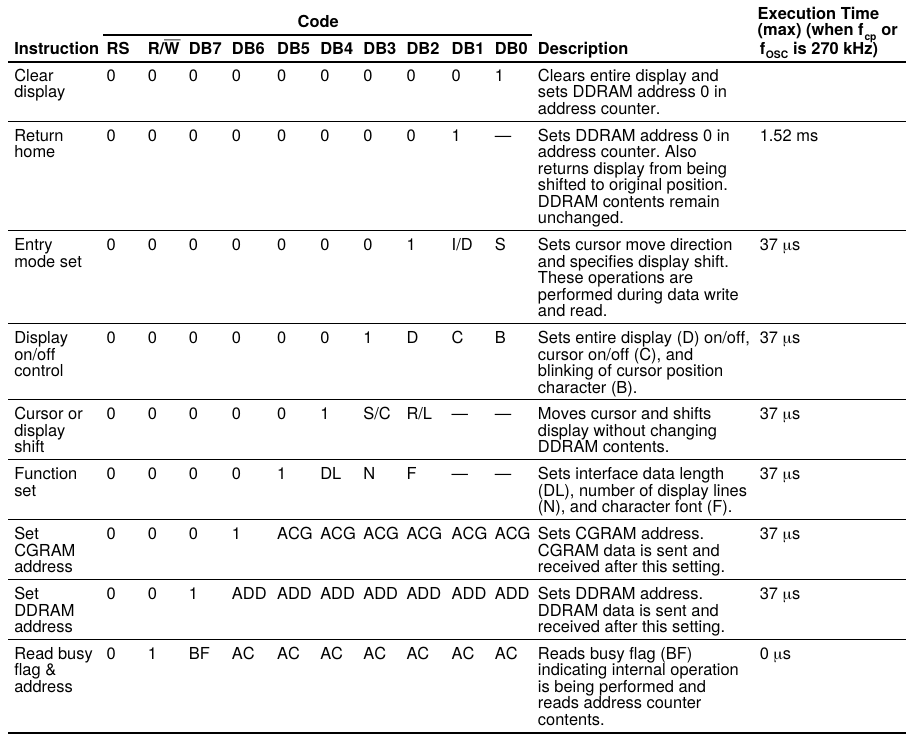
Para utilizar el display se cuenta con 2 tipos de operaciones, las cuales se diferencian por el bit de salida RS:

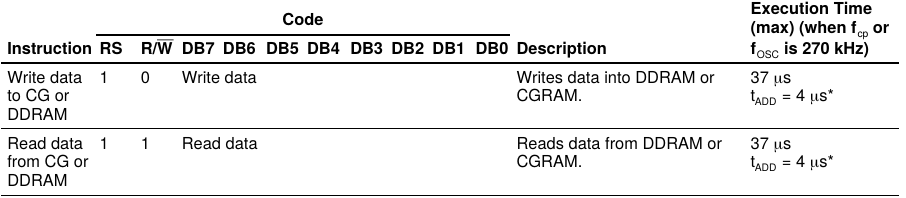
1. Escribir/Leer una Instrucción (Display Instruction)
2. Escribir/Leer un carácter (Display Data)

En trabajo solo se realizan rutinas de escrituras, por lo cual la señal R/W quedara siempre en bajo.

**Escribir/Leer una Instrucción:**

Se utiliza para inicializar el display, setear el modo de operación 4/8-Bits, activar/desactivar el blinking, posicionar el cursor en el display, etc. En estas operaciones el bit RS se debe poner a 0. En la siguiente tabla se muestran las instrucciones disponibles:





En este trabajo se van a realizar las siguientes operaciones para la inicialización del display:

* Función Set: LCD\_FUNCTIONSET
* Display ON/OFF control: LCD\_DISPLAYCONTROL
* Clear display: LCD\_CLEARDISPLAY
* Entry mode set: LCD\_ENTRYMODESET

Y luego se implementa una función adicional para posicionar el cursor en el display:

* Set DDRAM address: LCD\_SETDDRAMADDR

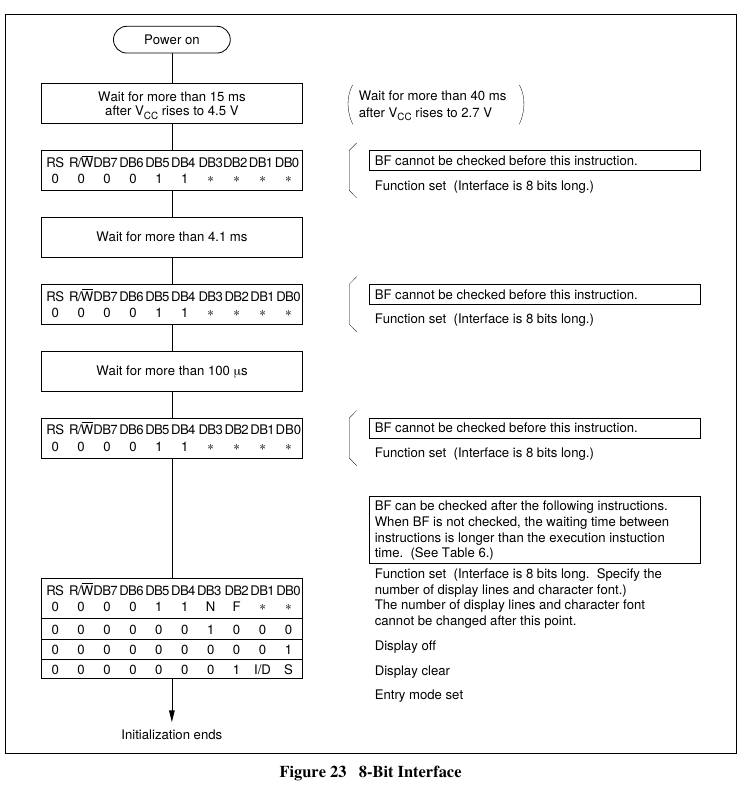
Las instrucciones, parámetros y tiempos asociados a estos se agregaron como constantes en el archivo lcd\_controller.vhd. Notar que las instrucciones pueden tomar diferentes tiempos en ser ejecutados.

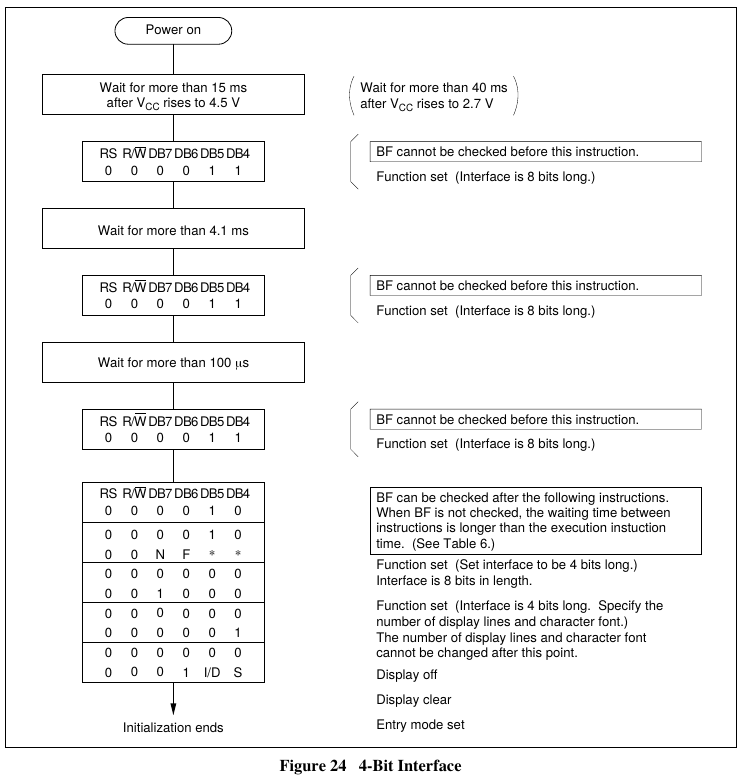
**Escribir/Leer un carácter.**

Se utiliza para escribir los caracteres en formato ASCII propiamente dicho. En esta operación el bit RS se debe poner a 1.

# Rutina de inicialización

Se implementan las siguientes secuencias de inicialización para 8 y 4-bits:

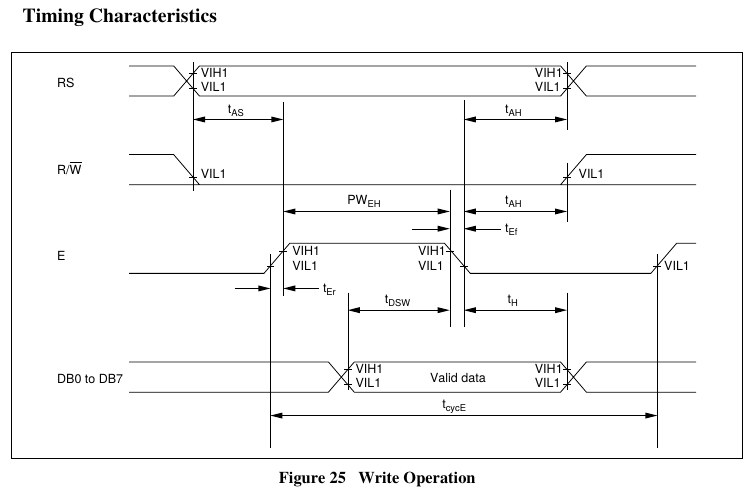




Los pasos de inicialización son similares salvo por que en el modo de 4-bits se manda la función SET 4 veces y en el de 8-bits 3 veces en los primeros pasos de la inicialización.

# LCD Timing

Se debieron tener en cuenta los tiempos más importantes para lograr el trabajo. Los tiempos se agregaron como constantes en los archivos lcd\_controller.vhd y lcd\_write.vhd.



# Implementación

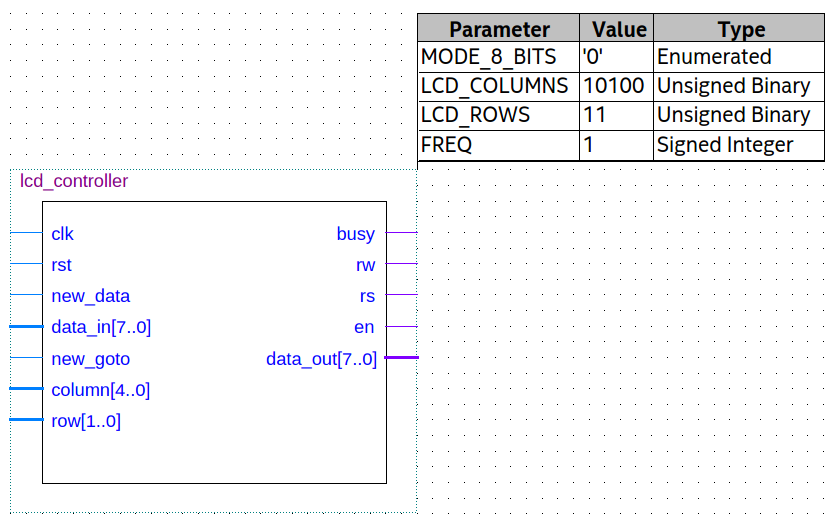
# Introducción

El sistema está compuesto por 2 módulos: lcd\_controller y lcd\_write.

El módulo lcd\_controller es el módulo top-level el cual contiene una máquina de estados que ordenadamente inicializa el display y luego queda lista para aceptar instrucciones o caracteres.

Este módulo principal instancia al módulo lcd\_write que se encarga de resolver la interfaz física con el display, básicamente el manejo de las señales RS, EN y DB0-7.

# Interfaz de E/S del módulo lcd\_controller



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.

# Constantes para instrucciones

Instrucciones disponibles en el display:

-- LCD commands

constant LCD\_CLEARDISPLAY : std\_logic\_vector(7 downto 0) := "00000001";

constant LCD\_RETURNHOME : std\_logic\_vector(7 downto 0) := "00000010";

constant LCD\_ENTRYMODESET : std\_logic\_vector(7 downto 0) := "00000100";

constant LCD\_DISPLAYCONTROL : std\_logic\_vector(7 downto 0) := "00001000";

constant LCD\_CURSORSHIFT : std\_logic\_vector(7 downto 0) := "00010000";

constant LCD\_FUNCTIONSET : std\_logic\_vector(7 downto 0) := "00100000";

constant LCD\_SETCGRAMADDR : std\_logic\_vector(7 downto 0) := "01000000";

constant LCD\_SETDDRAMADDR : std\_logic\_vector(7 downto 0) := "10000000";

Como se comentó se implementan 5 instrucciones:

- Function Set: LCD\_FUNCTIONSET

- Display ON/OFF control: LCD\_DISPLAYCONTROL

- Clear display: LCD\_CLEARDISPLAY

- Entry mode set: LCD\_ENTRYMODESET

- Set DDRAM address: LCD\_SETDDRAMADDR

De estos, solo los 3 primeros aceptan parámetros/flags predefinidos los cuales se muestran a continuación:

-- flags for function set

constant LCD\_8BITMODE : std\_logic\_vector(7 downto 0) := "00010000";

constant LCD\_4BITMODE : std\_logic\_vector(7 downto 0) := "00000000";

constant LCD\_2LINE : std\_logic\_vector(7 downto 0) := "00001000";

constant LCD\_1LINE : std\_logic\_vector(7 downto 0) := "00000000";

constant LCD\_5x10DOTS : std\_logic\_vector(7 downto 0) := "00000100";

constant LCD\_5x8DOTS : std\_logic\_vector(7 downto 0) := "00000000";

-- flags for display entry mode

constant LCD\_ENTRYRIGHT : std\_logic\_vector(7 downto 0) := "00000000";

constant LCD\_ENTRYLEFT : std\_logic\_vector(7 downto 0) := "00000010";

constant LCD\_ENTRYSHIFT : std\_logic\_vector(7 downto 0) := "00000001";

constant LCD\_ENTRYNOSHIFT : std\_logic\_vector(7 downto 0) := "00000000";

-- flags for display on/off control

constant LCD\_DISPLAYON : std\_logic\_vector(7 downto 0) := "00000100";

constant LCD\_DISPLAYOFF : std\_logic\_vector(7 downto 0) := "00000000";

constant LCD\_CURSORON : std\_logic\_vector(7 downto 0) := "00000010";

constant LCD\_CURSOROFF : std\_logic\_vector(7 downto 0) := "00000000";

constant LCD\_BLINKON : std\_logic\_vector(7 downto 0) := "00000001";

constant LCD\_BLINKOFF : std\_logic\_vector(7 downto 0) := "00000000";

# Constantes para temporización

Se definieron las siguientes constantes para ser usadas en el contador de tiempos para satisfacer el timing requerido por el display:

-- LCD delay Times

constant LCD\_POWER\_UP\_WAIT\_US : integer := 40000 \* FREQ; -- Wait for more than 40 ms after VCC rises to 2.7 V

constant LCD\_STARTUP\_WAIT\_1\_US : integer := 5000 \* FREQ; -- Wait for more than 4.1 ms

constant LCD\_STARTUP\_WAIT\_2\_US : integer := 150 \* FREQ; -- Wait for more than 100 μs

constant LCD\_LOW\_WAIT\_US : integer := 25 \* FREQ; -- 25 us

constant LCD\_HIGH\_WAIT\_US : integer := 100 \* FREQ; -- 100 us

constant LCD\_CMD\_WAIT\_US : integer := 110 \* FREQ; -- Wait time for every command 45 us

constant LCD\_CLR\_DISP\_WAIT\_US : integer := 3000 \* FREQ; -- Clear Display 3 ms

constant LCD\_RET\_HOME\_WAIT\_US : integer := 2000 \* FREQ; -- Return Home 1.52 ms

# Arquitectura del módulo lcd\_controller

El módulo implementado tiene básicamente 2 grandes partes:

* Una máquina de estados para controlar la inicialización del display y el posterior envió de caracteres o pedidos de posicionamiento del cursor.
* Instancia al módulo lcd\_write. Este módulo se utiliza para manejar las señales que se envían al display (rw, rs, en y data\_out) de forma tal de hacerlas independientes de la máquina de estados previamente descripta.

# Máquina de estados principal

En la figura de la próxima página se muestra la máquina de estados utilizada en el bloque lcd\_controller.

**Estados:**

type state\_t is (POWER\_UP, INIT\_1, INIT\_2, INIT\_3, FUNCTIONSET\_4\_BITS, FUNCTIONSET, DISPLAYCONTROL, LCD\_CLEAR, ENTRYMODESET, WAITING, READY);

La máquina de estados se va recorriendo en el orden mostrado a continuación:

POWER\_UP

INIT\_1

INIT\_2

INIT\_3

FUNCTIONSET\_4\_BITS

FUNCTIONSET

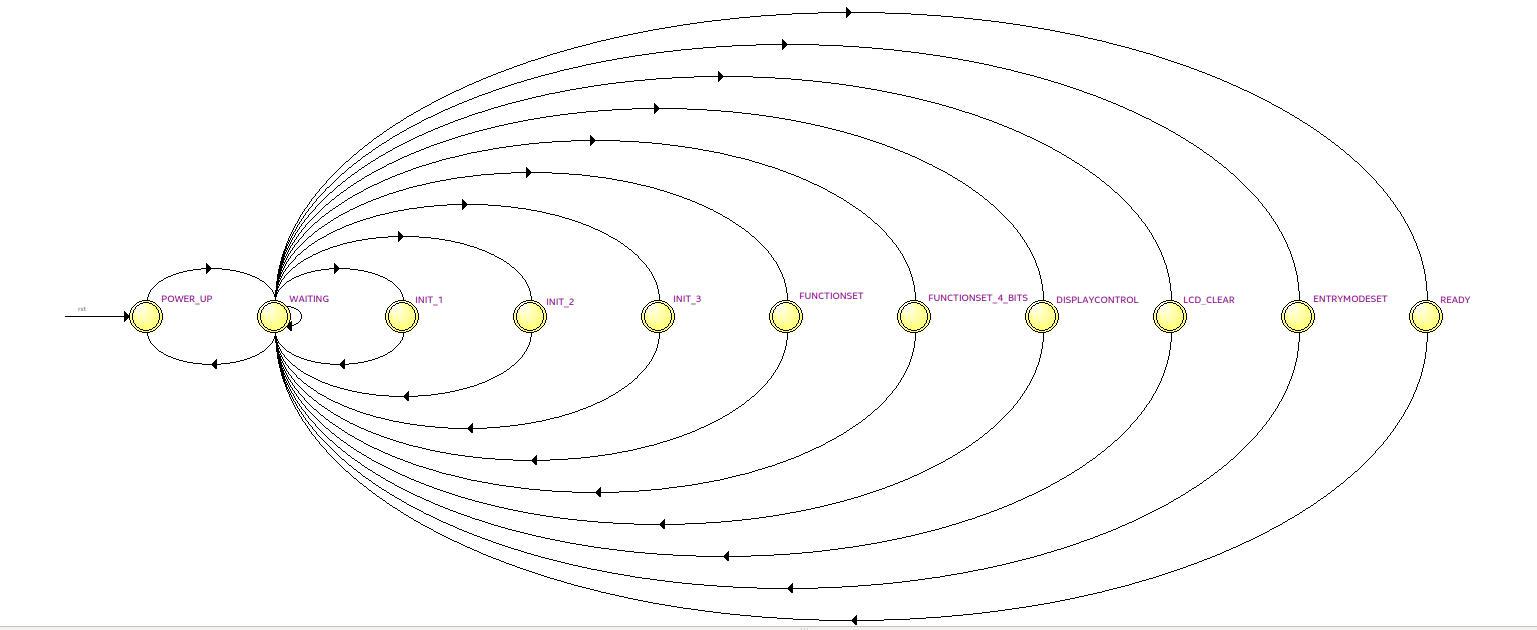
DISPLAYCONTROL

LCD\_CLEAR

ENTRYMODESET

READY

Siempre pasando por el estado WAITING donde se tiene el contador para esperar los tiempos de inicialización entre instrucciones/envió de caracteres.



*Máquina de estados principal*

**Descripción de estados:**

WAITING: estado utilizado para realizar todas la esperas de tiempos necesarios.

*Estados del proceso de inicialización:*

* POWER\_UP: espera de 40 ms (LCD\_POWER\_UP\_WAIT\_US) especificada en el datasheet del controlador para esperar a VCC. Luego ir a WAITING y luego a INIT\_1.
* INIT\_1: primera instrucción SET y espera de al menos 4.1 ms (LCD\_STARTUP\_WAIT\_1\_US, se esperan 5ms). Luego ir a WAITING y luego a INIT\_2.
* INIT\_2: segunda instrucción SET y espera de al menos 100us (LCD\_STARTUP\_WAIT\_2\_US). Luego ir a WAITING y luego a INIT\_3.
* INIT\_3: tercera instrucción SET y espera de un a normal (LCD\_CMD\_WAIT\_US). Ir a WAITING y luego a FUNCTIONSET\_4\_BITS si el controlador está configurado en 4bits o ir a FUNCTIONSET si está en 8-bits.
* FUNCTIONSET\_4\_BITS: cuarta instrucción SET para modo de 4-bits y espera de una instrucción normal. Ir a WAITING y luego a FUNCTIONSET.
* FUNCTIONSET: en esta función SET se configura entre otras cosas el modo 8-bits o 4-bits (LCD\_8BITMODE o LCD\_4BITMODE). Ir a WAITING y luego a DISPLAYCONTROL.
* DISPLAYCONTROL: en esta función se configura que el display va a estar encendido (LCD\_DISPLAYON) y también el cursor visible (LCD\_CURSORON). Ir a WAITING y luego a LCD\_CLEAR.
* LCD\_CLEAR: se manda la instrucción CLEAR LCD como lo especifica la secuencia de inicialización y se espera 3ms (LCD\_CLR\_DISP\_WAIT\_US), luego se va al estado ENTRYMODESET.
* ENTRYMODESET: último estado de inicialización, se utiliza para setear el modo de entrada: desde la izquierda (LCD\_ENTRYLEFT).

*Estado en funcionamiento normal:*

* READY: este estado se está esperando los bits *new\_data* o *new\_goto* para mandar un nuevo carácter (*data\_in*) o posicionar el cursor en el valor especificado por *column* y *row*. Para que los pedidos se hagan efectivos se debe esperar el tiempo estándar de instrucción (LCD\_CMD\_WAIT\_US) que está especificado en 110us, para tal fin se salta al estado WAITING y luego se retorna a READY.

# Interfaz de E/S del módulo lcd\_write

entity lcd\_write is

generic (

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

);

port (

clk : in std\_logic; --system clock

rst : in std\_logic; --reset

cmd : in std\_logic; --cmd = 1, data = 0

mode\_8\_bits : in std\_logic; -- 8-bits or 4-bits

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

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

busy : out std\_logic; --block busy

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

data\_out : out std\_logic\_vector(7 downto 0)

);

end;

**Genéricos/parámetros:**

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.

*cmd*: indica si se desea mandar una función o un carácter.

*mode\_8\_bits*: indica el ancho de bus, ya que dependiendo de este se deben generar una transacción o dos transacciones.

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

*data\_in*: dato a enviar, puede ser una función o un carácter ASCII a escribir.

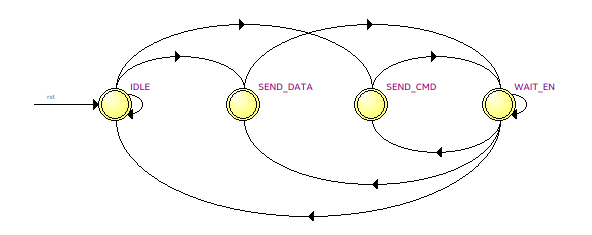
**Salidas:**

*busy*: bloque ocupado

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

*data\_out*: salida paralelo al display LCD

# Máquina de estado del módulo lcd\_write



Estados:

IDLE: en este estado el bloque se encuentra esperando un nuevo pedido de envió al display por parte del módulo lcd\_controller por medio de la señal *new\_data*. En función de la entrada *cmd* decide si ir a SEND\_DATA o SEND\_CMD.

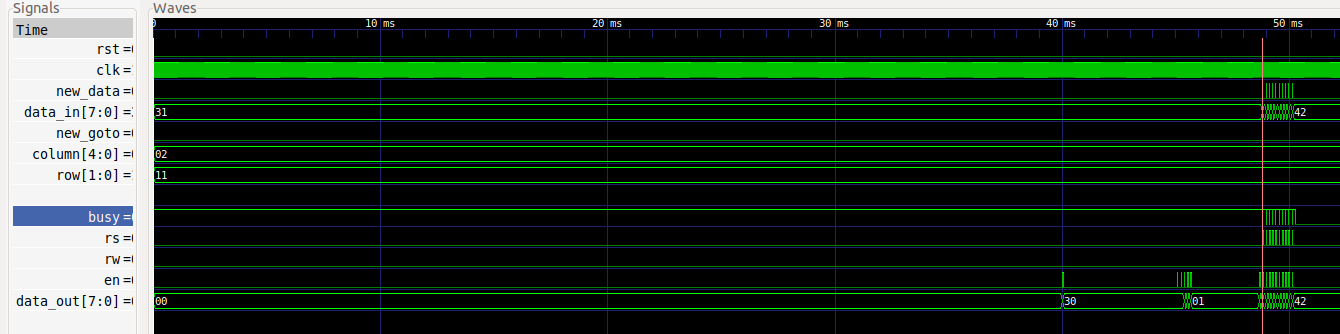
SEND\_DATA: envía caracteres al LCD a través del correcto manejo de las señales RW, RS, EN y data\_out.

SEND\_CMD: envía las instrucciones solicitadas al LCD a través del correcto manejo de las señales RW, RS, EN y data\_out.

WAIT\_EN: se utiliza para contar el tiempo necesario para la señal EN.

# Simulaciones

# Salida de reset e inicialización (modo 4-bits)

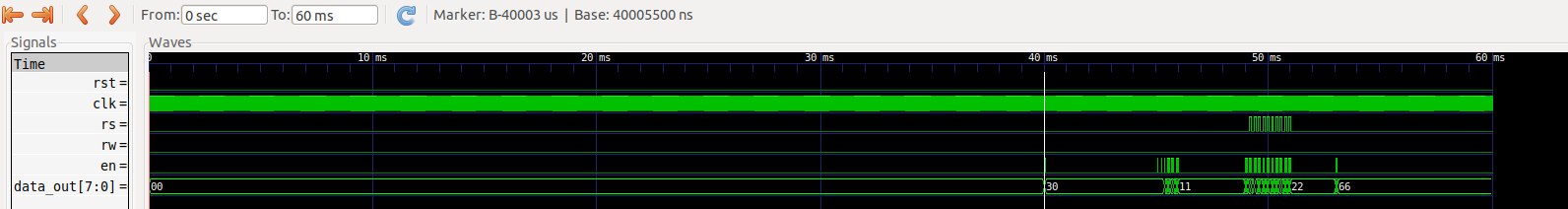


Hasta donde se ve el cursor se envían todas las instrucciones necesarias para inicializar el display. Se observa un periodo de tiempo extenso al principio (40ms), luego hay actividad en las líneas, envió del primer instrucción SET, una espera grande nuevamente (5ms), una a seguidilla de instrucciones, luego otra espera grande (instrucción CLEAR, espera > 1.5 ms).

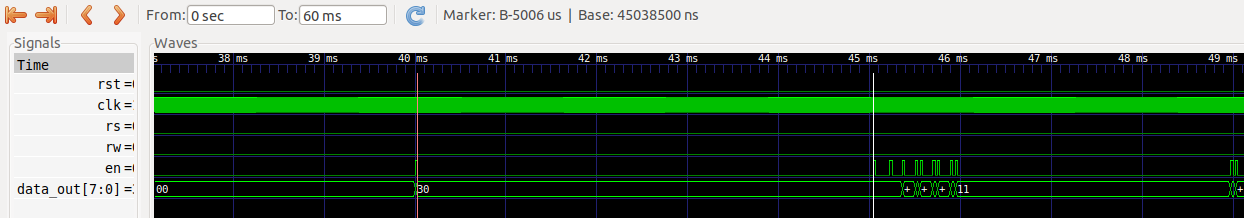
En la inicialización la señal RS permanece en bajo ya que es la condición para decirle al display que estos son instrucciones.

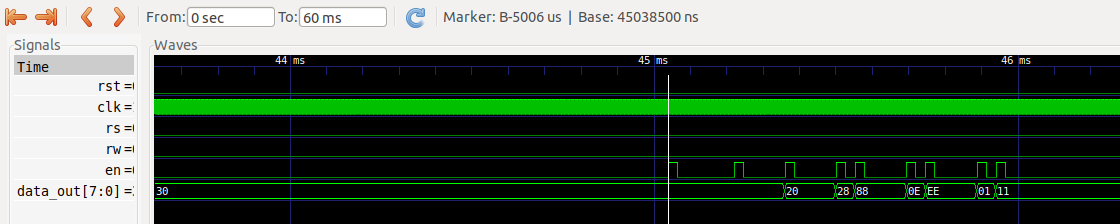
Luego del marcador rojo, se empiezan mandar caracteres e instrucciones por parte del banco de pruebas.

# Espera de 40ms (medición)



# Espera de 5ms después de primer instrucción SET (data out = 0x30)





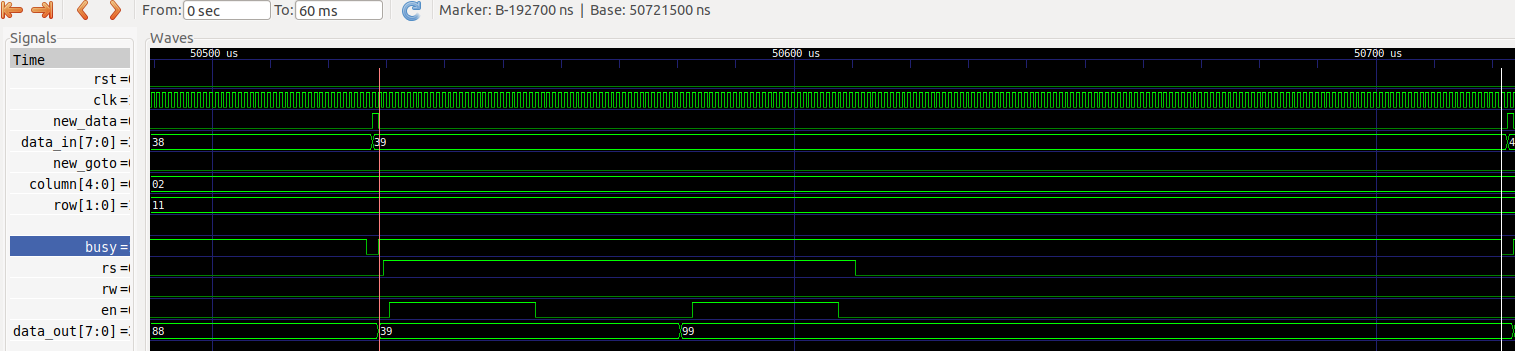
Aquí se muestran las 2 instrucciones SET, después de la primera mostrada anteriormente, para sumar las 3 necesarias. Las instrucciones pueden ser diferenciadas por la señal EN que se pone en 1. Luego del tercer SET, se manda un cuarto SET sin parámetros (0x20) el cual le indica al display que va a funcionar en modo 4-bits. Y partir de aquí, todas las transacciones, que deben ser de 8-bits, se mandan de a 4-bits, por lo tanto la señal EN debe ser levantada 2 veces por paquete, primero se envía la parte alta y luego la baja.

# Escritura de un carácter en modo 4-bits

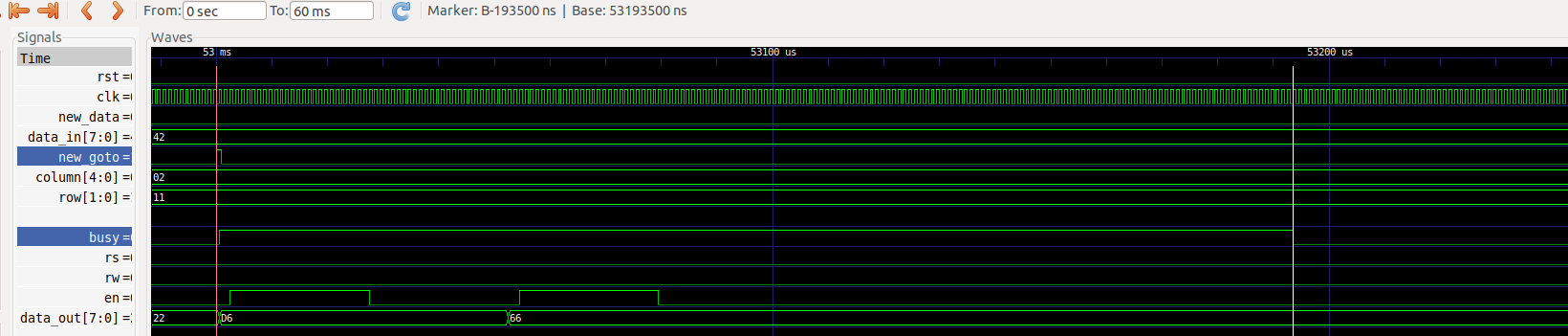
En la simulación se quiere escribir el carácter 0x39 (*data\_in*), luego de setear la señal *new\_data*, la señal *busy* y RS se levantan. El paquete se manda en dos tandas en la parte alta de *data\_out* (4-bits mode), primero el 0x3 y luego el 0x9 (la parte baja de *data\_out* se debe ignorar). Para indicar dato valido en *data\_out* la señal EN se levanta en ambos casos por un tiempo de LCD\_EN\_PULSE\_WAIT\_US.

La señal RS se baja un tiempo después del segundo EN, y la *busy* se baja después de un tiempo LCD\_CMD\_WAIT\_US, necesario para que el display puede aceptar otra instrucción/carácter.

Un detalle es que las diferentes señales RS, EN y data\_out se levantan en diferentes tiempos para cumplir con la temporización del display.

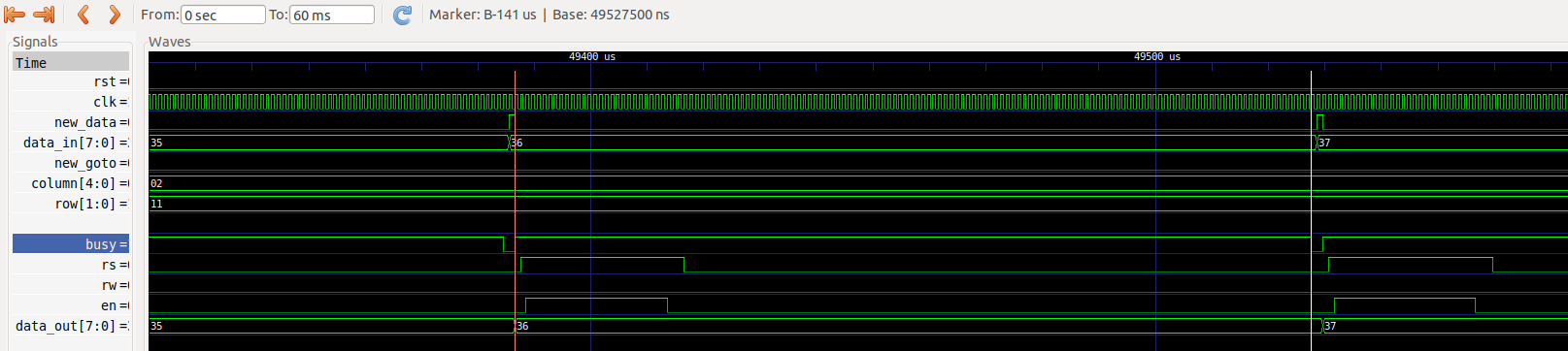


# Escritura de una instrucción/función en modo 4-bits



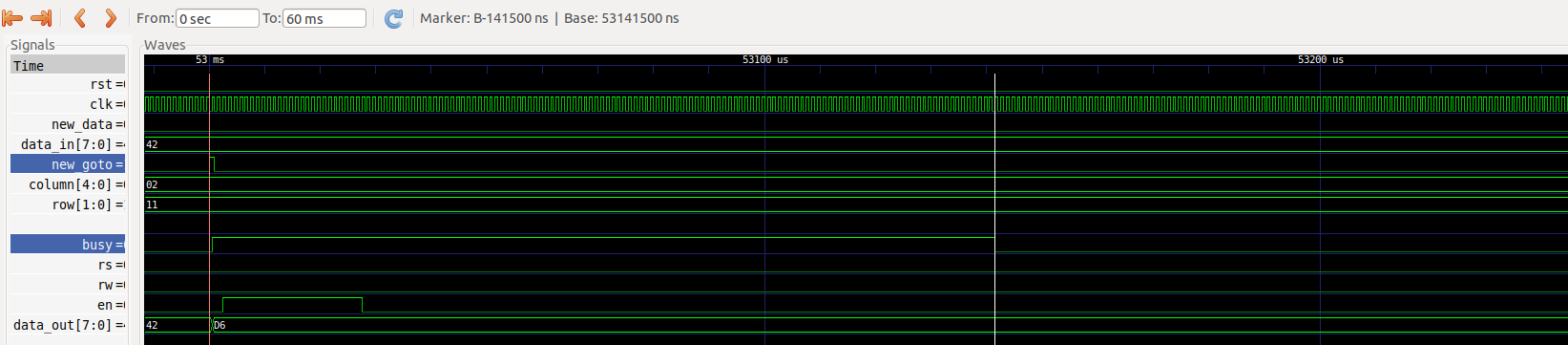
Es idéntico al anterior solo que RS permanece en cero. La diferencia aquí es que la instrucción que se envía es la de posicionar el cursor (LCD\_SETDDRAMADDR). Esta instrucción no es parte de la inicialización sino que se implementó para poder posicionar el cursor donde se desee. Para utilizar esta instrucción se levanta la señal *new\_goto* en vez de *new\_data*, y las señales *column* y *row* son tomadas en cuenta para calcular el dato de salida que resulta ser 0xD6.

# Escritura de un carácter en modo 8-bits



Este caso es similar al caso de 4-bits solo que el paquete (*data\_in* = 0x36) se envía en una sola transacción, EN se levanta una sola vez y se usa el ancho completo de *data\_out*. Nuevamente se respetan los tiempos pedidos por el display.

# Escritura de una instrucción/función en modo 8-bits

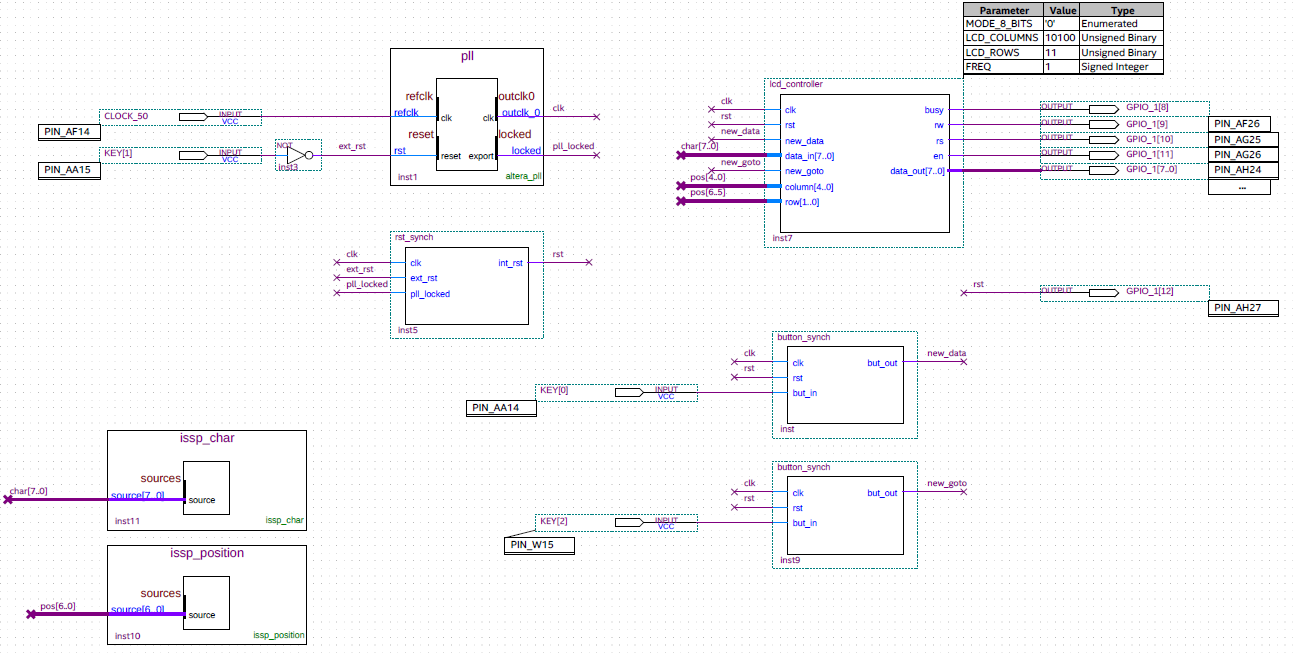


Este caso se escribe la posición nuevamente (LCD\_SETDDRAMADDR) como en el caso de 4-bits.

# Implementación en Quartus

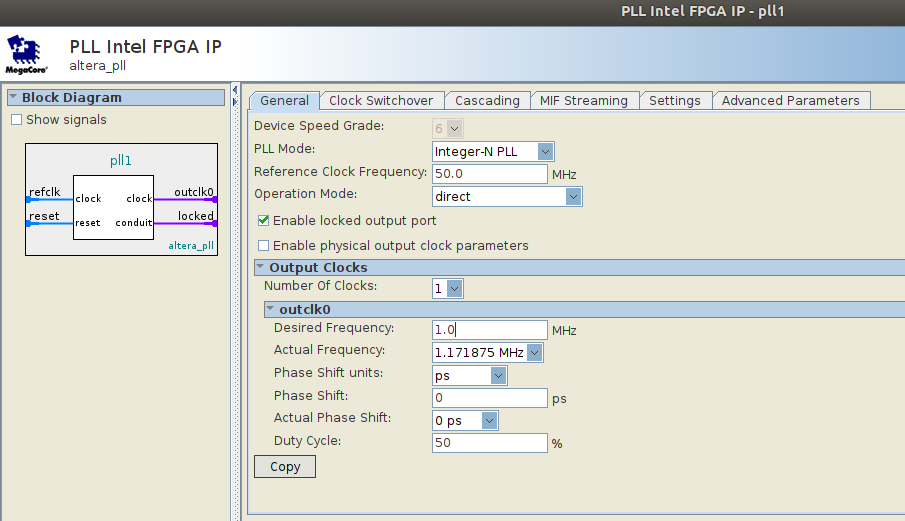
# Esquemático

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



**Bloque instanciados:**

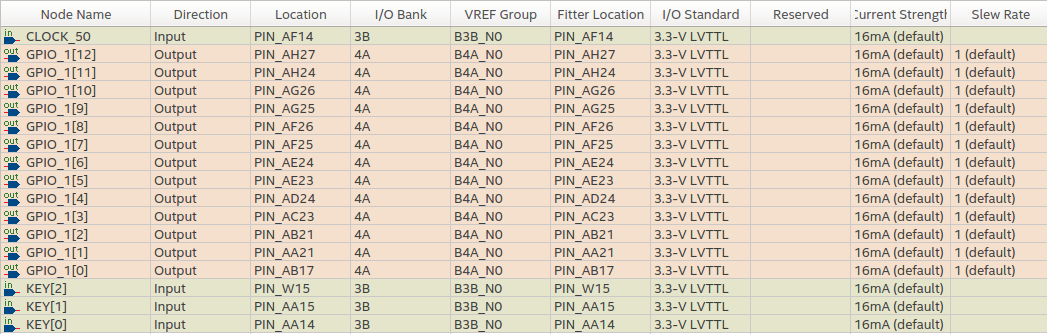
* pll: PLL con entrada de reset y salida de locked obtenido del IP Catalog de Quartus. La entrada de clk a la FPGA es de 50Mhz y se selecciona una salida del PLL de 1Mhz, que resulto en 1.171875 Mhz.



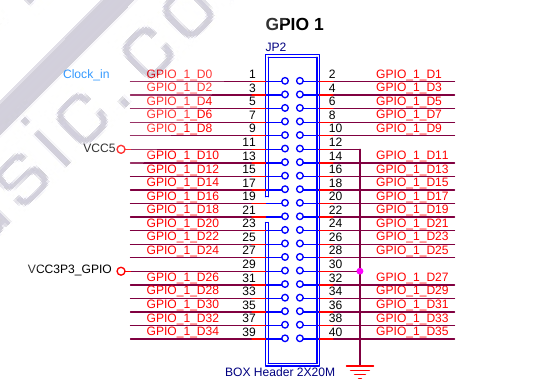
* rst\_synch: sincronizador de reset, bloque creado por mí para sincronizar la señal de reset externa (KEY[1]). Incluye la espera de la señal de PLL lockeado.
* button\_synch: sincronizadores para teclas KEY[0] y KEY[2] usadas para señales *new\_data* y *new\_goto* respectivamente.
* issp\_char y issp\_position: In-System Source & Probe IP obtenido del IP Catalog para simular las entradas paralelo de data\_in[7..0], column[4..0] y row[1..0].

# 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.

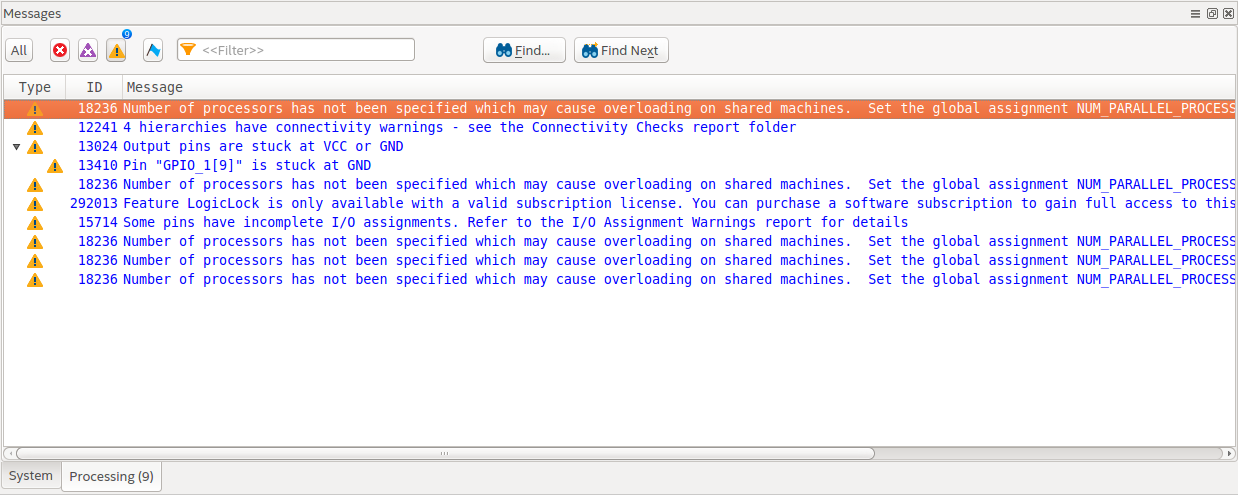


Conexionado entre display y conector JP2:

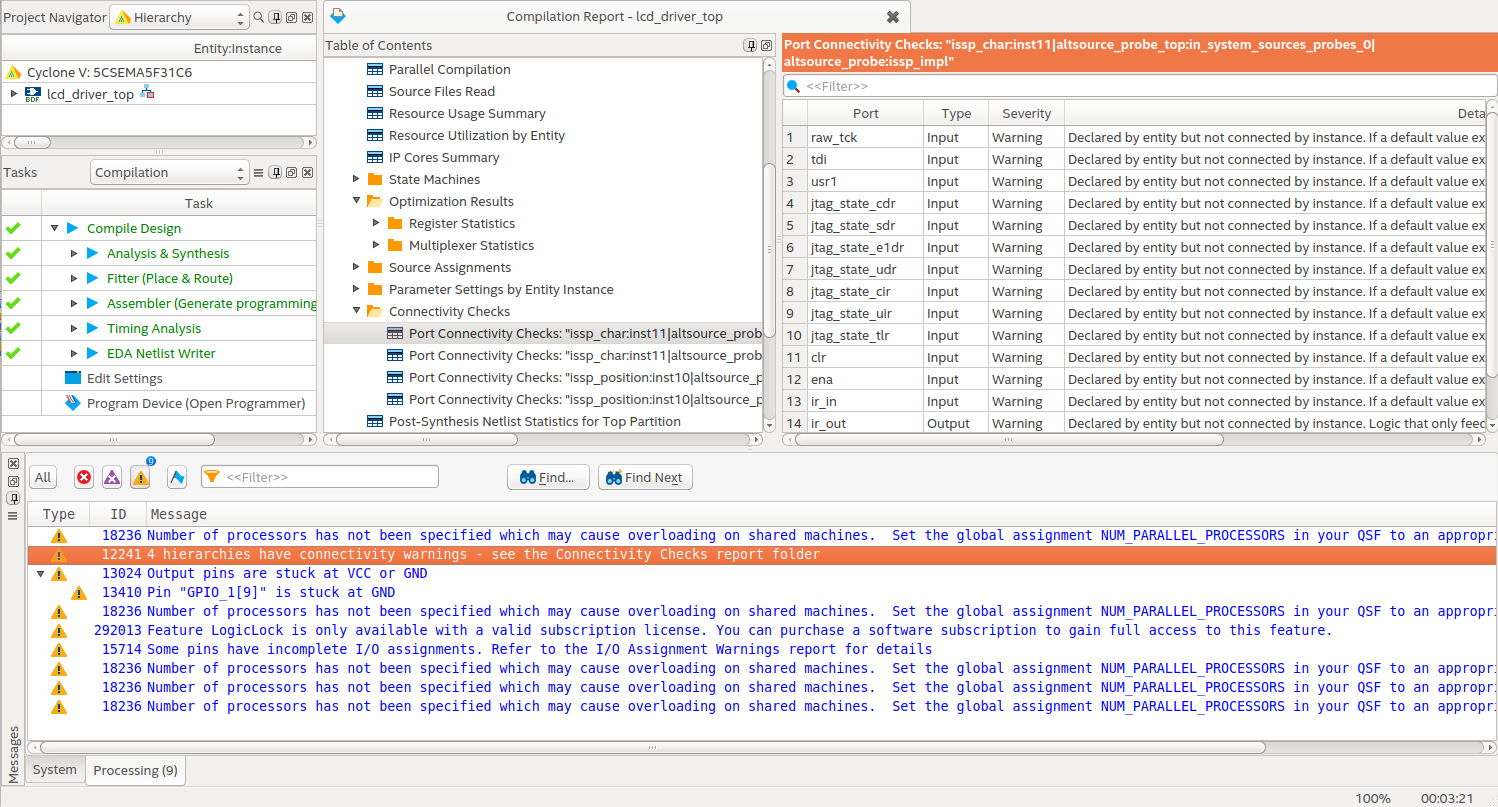
|  |  |  |  |
| --- | --- | --- | --- |
| **Display** | | **Kit de desarrollo** | |
| **Pin** | **Función** | **Pin de JP2** | **Función** |
| 1 | VSS | 30 | GND |
| 2 | VDD | 29 | VCC3P3\_GPIO |
| 3 | V0 | 30 | GND |
| 4 | RS(C/D) | 13 | GPIO\_1\_D10 |
| 5 | R/W | 30 | GND |
| 6 | E | 14 | GPIO\_1\_D11 |
| 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 |

# Warnings

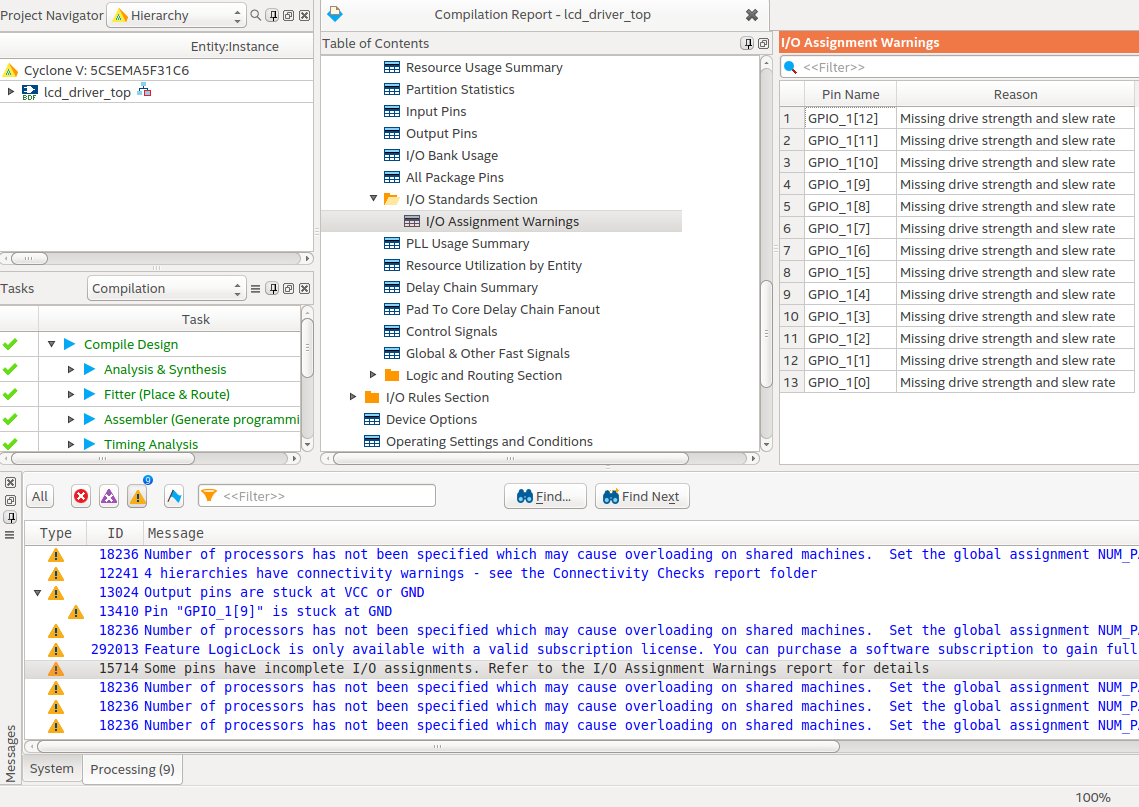
Se limpiaron todos los warnings que aparecían, solo quedaron los siguientes:



De los cuales, sacando los 18236 referidos al uso de los procesadores de la pc, el 12241 se debe a los bloques ISSP auxiliares para generar las señales de simulación, no los consideré críticos:

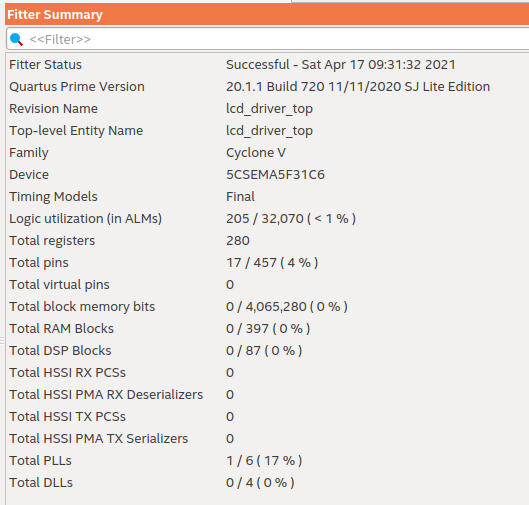


El 15714 se debe a la no asignación de drive strength y slew rate, tampoco creo que son críticos pero este trabajo, pero están analizados al menos:



Y el 130410 que indica que el pin GPIO\_1[9] esta GND es debido a que este pin es el R/W que no lo estoy utilizando ya que solo se implementaron lecturas.

# Recursos de la FPGA

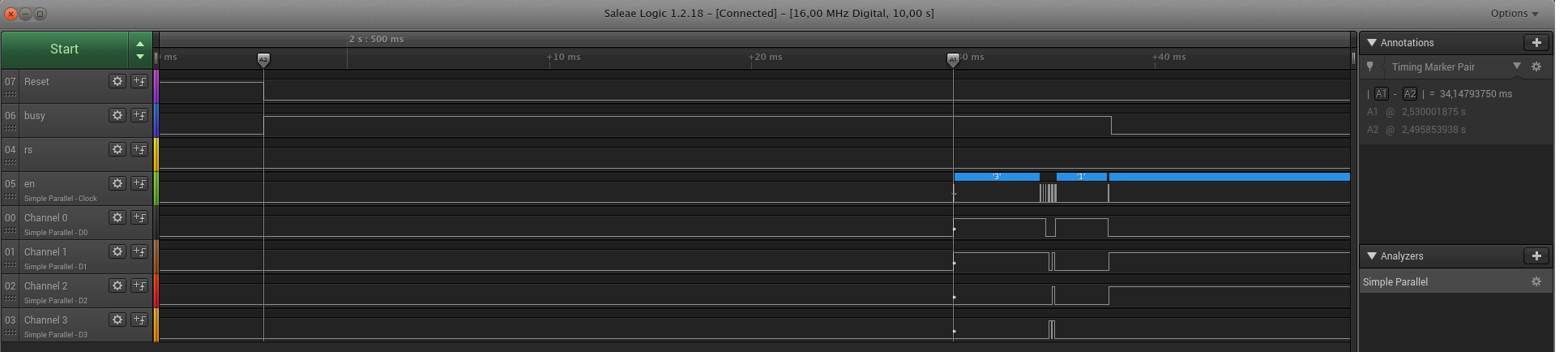


Reporte detallado:

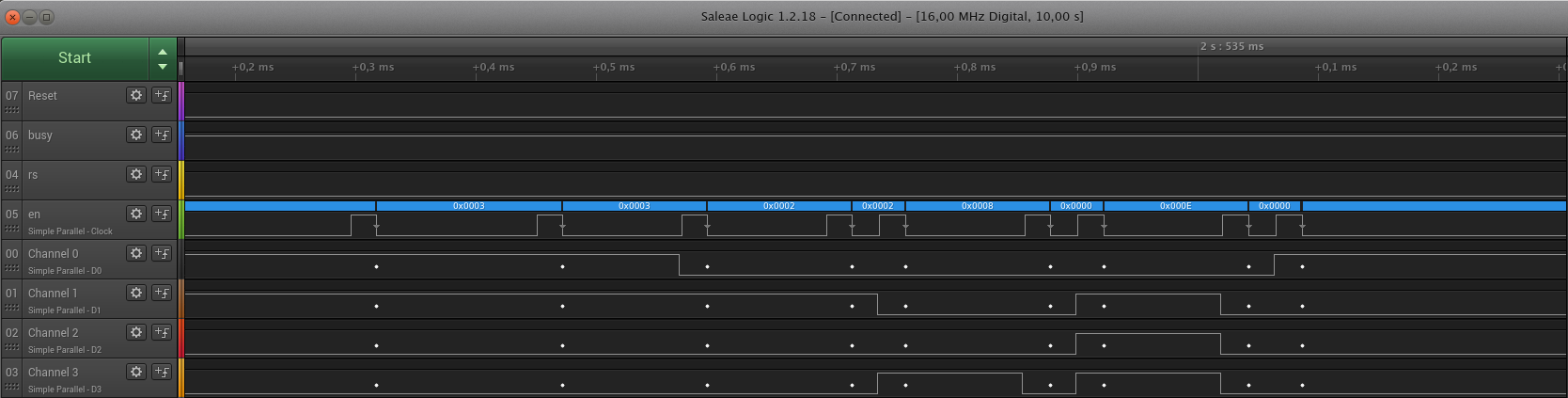
|  |  |  |
| --- | --- | --- |
| **Resource** | **Usage** | **%** |
| Logic utilization (ALMs needed / total ALMs on device) | 205 / 32,070 | < 1 % |
| ALMs needed [=A-B+C] | 205 |  |
| [A] ALMs used in final placement [=a+b+c+d] | 232 / 32,070 | < 1 % |
| [a] ALMs used for LUT logic and registers | 94 |  |
| [b] ALMs used for LUT logic | 106 |  |
| [c] ALMs used for registers | 32 |  |
| [d] ALMs used for memory (up to half of total ALMs) | 0 |  |
| [B] Estimate of ALMs recoverable by dense packing | 27 / 32,070 | < 1 % |
| [C] Estimate of ALMs unavailable [=a+b+c+d] | 0 / 32,070 | 0 % |
| [a] Due to location constrained logic | 0 |  |
| [b] Due to LAB-wide signal conflicts | 0 |  |
| [c] Due to LAB input limits | 0 |  |
| [d] Due to virtual I/Os | 0 |  |
|  |  |  |
| Difficulty packing design | Low |  |
|  |  |  |
| Total LABs: partially or completely used | 34 / 3,207 | 1 % |
| -- Logic LABs | 34 |  |
| -- Memory LABs (up to half of total LABs) | 0 |  |
|  |  |  |
| Combinational ALUT usage for logic | 339 |  |
| -- 7 input functions | 9 |  |
| -- 6 input functions | 68 |  |
| -- 5 input functions | 77 |  |
| -- 4 input functions | 26 |  |
| -- <=3 input functions | 159 |  |
| Combinational ALUT usage for route-throughs | 26 |  |
|  |  |  |
| Dedicated logic registers | 280 |  |
| -- By type: |  |  |
| -- Primary logic registers | 251 / 64,140 | < 1 % |
| -- Secondary logic registers | 29 / 64,140 | < 1 % |
| -- By function: |  |  |
| -- Design implementation registers | 251 |  |
| -- Routing optimization registers | 29 |  |
|  |  |  |
| Virtual pins | 0 |  |
| I/O pins | 17 / 457 | 4 % |
| -- Clock pins | 3 / 8 | 38 % |
| -- Dedicated input pins | 3 / 21 | 14 % |
|  |  |  |
| Hard processor system peripheral utilization |  |  |
| -- Boot from FPGA | 0 / 1 ( 0 % ) |  |
| -- Clock resets | 0 / 1 ( 0 % ) |  |
| -- Cross trigger | 0 / 1 ( 0 % ) |  |
| -- S2F AXI | 0 / 1 ( 0 % ) |  |
| -- F2S AXI | 0 / 1 ( 0 % ) |  |
| -- AXI Lightweight | 0 / 1 ( 0 % ) |  |
| -- SDRAM | 0 / 1 ( 0 % ) |  |
| -- Interrupts | 0 / 1 ( 0 % ) |  |
| -- JTAG | 0 / 1 ( 0 % ) |  |
| -- Loan I/O | 0 / 1 ( 0 % ) |  |
| -- MPU event standby | 0 / 1 ( 0 % ) |  |
| -- MPU general purpose | 0 / 1 ( 0 % ) |  |
| -- STM event | 0 / 1 ( 0 % ) |  |
| -- TPIU trace | 0 / 1 ( 0 % ) |  |
| -- DMA | 0 / 1 ( 0 % ) |  |
| -- CAN | 0 / 2 ( 0 % ) |  |
| -- EMAC | 0 / 2 ( 0 % ) |  |
| -- I2C | 0 / 4 ( 0 % ) |  |
| -- NAND Flash | 0 / 1 ( 0 % ) |  |
| -- QSPI | 0 / 1 ( 0 % ) |  |
| -- SDMMC | 0 / 1 ( 0 % ) |  |
| -- SPI Master | 0 / 2 ( 0 % ) |  |
| -- SPI Slave | 0 / 2 ( 0 % ) |  |
| -- UART | 0 / 2 ( 0 % ) |  |
| -- USB | 0 / 2 ( 0 % ) |  |
|  |  |  |
| M10K blocks | 0 / 397 | 0 % |
| Total MLAB memory bits | 0 |  |
| Total block memory bits | 0 / 4,065,280 | 0 % |
| Total block memory implementation bits | 0 / 4,065,280 | 0 % |
|  |  |  |
| Total DSP Blocks | 0 / 87 | 0 % |
|  |  |  |
| Fractional PLLs | 1 / 6 | 17 % |
| Global signals | 1 |  |
| -- Global clocks | 1 / 16 | 6 % |
| -- Quadrant clocks | 0 / 66 | 0 % |
| -- Horizontal periphery clocks | 0 / 18 | 0 % |
| SERDES Transmitters | 0 / 100 | 0 % |
| SERDES Receivers | 0 / 100 | 0 % |
| JTAGs | 1 / 1 | 100 % |
| ASMI blocks | 0 / 1 | 0 % |
| CRC blocks | 0 / 1 | 0 % |
| Remote update blocks | 0 / 1 | 0 % |
| Oscillator blocks | 0 / 1 | 0 % |
| Impedance control blocks | 0 / 4 | 0 % |
| Hard Memory Controllers | 0 / 2 | 0 % |
| Average interconnect usage (total/H/V) | 0.2% / 0.2% / 0.2% |  |
| Peak interconnect usage (total/H/V) | 6.2% / 6.1% / 6.4% |  |
| Maximum fan-out | 192 |  |
| Highest non-global fan-out | 148 |  |
| Total fan-out | 2366 |  |
| Average fan-out | 3.41 |  |

# Simulaciones en FPGA

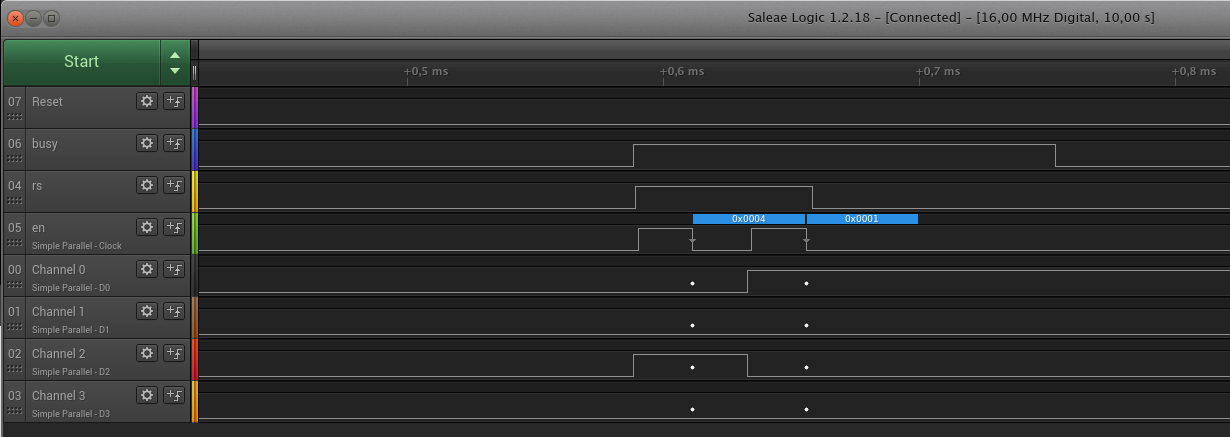
# Salida de reset, inicialización (modo 4-bits) y espera de 40ms



# Zoom secuencia de inicialización



# Escritura de un carácter en modo 4-bits



# Escritura de una instrucción/función en modo 4-bits

