Objetivo General.
Utilizar el FPGA para el desarrollo de circuitos secuenciales.

Objetivos específicos.

- Investigar el funcionamiento de la tarjeta de desarrollo FPGA Spartan 3E
- Utilizar las herramientas del Xilinx ISE
- Conocer y aplicar el flujo de diseño para sistemas basados en FPGA

Propuesta del problema.
Implemetar circuitos que permitan desplegar texto en la pantalla LCD del Spartan 3E.

Ejercicio 1 (Obligatorio):
Implemente una máquina de estados que permita desplegar el siguiente texto en la pantalla LCD del Spartan 3E:

Hola Mundo

Primeramente deberá escribir un código de verilog de tal manera que la herramienta de síntesis entienda que se trata de una máquina de estados. Hay varias maneras de hacer esto, un ejemplo es el siguiente:

```
`timescale 1ns / 1ps


`define STATE_RESET                 0
`define STATE_POWERON_INIT_0_A      1
`define STATE_POWERON_INIT_0_B      2
`define STATE_POWERON_INIT_1        3
`define STATE_POWERON_INIT_2_A      4
`define STATE_POWERON_INIT_3_B      5


module Module_LCD_Control
(
input wire        Clock,
input wire        Reset,
output wire       oLCD_Enabled,
output reg        oLCD_RegisterSelect,                  //0=Command, 1=Data
output wire       oLCD_StrataFlashControl,
output wire       oLCD_ReadWrite,
output reg[3:0]   oLCD_Data
);

reg rWrite_Enabled;
assign oLCD_ReadWrite = 0;               //I only Write to the LCD display, never Read from it
assign oLCD_StrataFlashControl = 1; //StrataFlash disabled. Full read/write access to LCD

reg [7:0] rCurrentState,rNextState;
reg [31:0] rTimeCount;
reg rTimeCountReset;
wire wWriteDone;

//------------------------------------------------
//Next State and delay logic
always @ ( posedge Clock )
begin
        if (Reset)
        begin
                rCurrentState = `STATE_RESET;
                rTimeCount    <= 32'b0;
```

```
        end
        else
        begin

                if (rTimeCountReset)
                        rTimeCount <= 32'b0;
                else
                        rTimeCount <= rTimeCount + 32'b1;

                rCurrentState <= rNextState;
        end
end
//------------------------------------------------
//Current state and output logic
always @ ( * )
begin

        case (rCurrentState)
        //------------------------------------------
        `STATE_RESET:
        begin
                rWrite_Enabled         = 1'b0;
                oLCD_Data              = 4'h0;
                oLCD_RegisterSelect    = 1'b0;
                rTimeCountReset        = 1'b0;

                rNextState = `STATE_POWERON_INIT_0_A;
        end
        //------------------------------------------
        /*
                Wait 15 ms or longer.
                The 15 ms interval is 750,000 clock cycles at 50 MHz.

        */
        `STATE_POWERON_INIT_0_A:
        begin
                rWrite_Enabled       = 1'b0;
                oLCD_Data            = 4'h0;
                oLCD_RegisterSelect  = 1'b0; //these are commands
                rTimeCountReset      = 1'b0;

                if (rTimeCount > 32'd750000 )
                    rNextState    = `STATE_POWERON_INIT_0_B;
                else
                    rNextState    = `STATE_POWERON_INIT_0_A;

        end
        //------------------------------------------
                `STATE_POWERON_INIT_0_B:
                begin
                rWrite_Enabled       = 1'b0;
                oLCD_Data            = 4'h0;
                oLCD_RegisterSelect  = 1'b0; //these are commands
                rTimeCountReset      = 1'b1;  //Reset the counter here

                rNextState           = STATE_POWERON_INIT_1;
        end
        //------------------------------------------
        /*
        Write SF_D<11:8> = 0x3, pulse LCD_E High for 12 clock cycles
        */
        `STATE_POWERON_INIT_1:
        begin
                rWrite_Enabled       = 1'b1;
                oLCD_Data            = 4'h3;
                oLCD_RegisterSelect  = 1'b0; //these are commands
                rTimeCountReset      = 1'b1;

                if ( wWriteDone )
                        rNextState      = `STATE_POWERON_INIT_2;
                else
                        rNextState      = `STATE_POWERON_INIT_1;

        end
        //------------------------------------------
        /*
        Wait 4.1 ms or longer, which is 205,000 clock cycles at 50 MHz.
```

```
        */
        `STATE_POWERON_INIT_2_A:
        begin
                rWrite_Enabled      = 1'b0;
                oLCD_Data           = 4'h3;
                oLCD_RegisterSelect = 1'b0; //these are commands
                 rTimeCountReset     = 1'b0;

                if (rTimeCount > 32'd205000 )
                        rNextState           = `STATE_POWERON_INIT_3;
                else
                        rNextState      = `STATE_POWERON_INIT_2;

        end
        //-----------------------------------------
        `STATE_POWERON_INIT_2_B:
        begin
                rWrite_Enabled      = 1'b0;
                oLCD_Data           = 4'h3;
                oLCD_RegisterSelect = 1'b0; //these are commands
                 rTimeCountReset     = 1'b1;

                rNextState           = `STATE_POWERON_INIT_3;

        end
        //-----------------------------------------
        default:
        begin
                rWrite_Enabled         = 1'b0;
                oLCD_Data              = 4'h0;
                oLCD_RegisterSelect    = 1'b0;
                rTimeCountReset        = 1'b0;

                rNextState = `STATE_RESET;
        end
        //-----------------------------------------
        endcase

end


endmodule
```

El código anterior no tiene todos los estados necesarios para configurar el LCD ni para desplegar el texto, sin embargo le dará una idea de una posible implentación de una máquina de estados. Aquí la lógica de próximo estado y de salida estan combinadas en el mismo bloque always.

Es muy importante que la herramienta de síntesis entienda que se está descirbiendo una máquina de estados y no otro circuito. Si usted copia el codigo anterior y lo sintetiza verá que el reporte de la herramienta de síntesis despliega lo siquiente:

```
Found finite state machine <FSM_0> for signal <rCurrentState>.
    -----------------------------------------------------------------------
    | States             | 3                                              |
    | Transitions        | 4                                              |
    | Inputs             | 3                                              |
    | Outputs            | 3                                              |
    | Clock              | Clock                     (rising_edge)        |
    | Reset              | Reset                     (positive)           |
    | Reset type         | synchronous                                    |
    | Reset State        | 00000000                                       |
    | Encoding           | automatic                                      |
    | Implementation     | LUT                                            |
    -----------------------------------------------------------------------
...

   Analyzing FSM <FSM_0> for best encoding.
Optimizing FSM <rCurrentState/FSM> on signal <rCurrentState[1:2]> with gray encoding.
----------------------
 State    | Encoding
```

```
----------------------
 00000000 | 00
 00000001 | 01
 00000010 | 11
 00000011 | unreached
 00000100 | unreached
----------------------
```

Implemente una máquina de estados que imprima el texto “Hola Mundo” en la pantalla LCD. Para esto deberá observar la documentación de la tarjerta de desarrollo del spartan 3E, la cual se encuentra disponible en le sitio web del curso.

Es obligatorio que el reporte de la herramienta de síntesis muestre que su circuito es una maquina de estados como en el ejemplo anterior.

Ejercicio 2 (Obligatorio):

Configurar la LCD y desplegar texto son cosas por naturaleza secuenciales. Dado que todas las operaciones de la MiniALU del experimento-2 duran 2 ciclos de reloj, es posible utilizarla para desplegar el texto.

Implemente una lógica en la mini ALU que configure el LCD, esto lo podrá hacer mediante código en la ROM o usando alguna otra forma que crea conveniente.

Agregue las siguientes instrucciones a la MiniALU para escribir un carácter a la vez, primero los 4 bits altos y luego los 4 bits más bajos.

La instrucción puede tener el siguiente formato:

```
LCD  <Destino Nulo> , < Fuente1 >, <Fuente2 Nulo>
```

Esta instrucción debe mandar al LCD solo los 4 bits más significativos de Fuente1 e ignorar el resto de la palabra. Dado que la LCD acepta 4 bits a la vez, deberá implementar una instrucción de corrimiento a la izquierda a la cual llamará SHL.

```
SHL <Destino > , < Fuente1 >, <Fuente2>
```

La Fuente 1 deberá ser el registro que desea correr a la izquierda y la fuente 2 deberá ser el número de bits que desea correr.

Un ejemplo de pseudo-código puede ser el siguiente:

```
//Inicialize el LCD ...


//Escriba el Nibble mas significativo
```

```
`STO `R1, `H_   //Cargue la letra H
`LCD,  4'b0, `R1, 8'b0
`NOP     //Implemente un delay de alguna manera
//Escriba el Nibble menos significativo
`SHL, `R1,`R1, 8'd4
`LCD,  4'b0, `R1, 8'b0
`NOP     //Implemente un delay de alguna manera
//Carge la siguiente letra
```

Ejercicio 3 (Obligatorio):

Escribir los caracteres del ejercicio anterior hubiera sido más sencillo si la miniALU soportara llamados a sub-rutinas.

Modifique la miniALU de manera que soporte subrutinas. Para esto puede implementar una instrucción CALL y una instrucción RET con el siguiente formato:

Operación (4 bits)	Destino (8 bits)	Fuente 1 (8 bits)	Fuente 2 (8 bits)
CALL	Valor inmmediato que contiene la dirección en la ROM donde se encuentra el código a ejecutar	Nulo	Nulo
RET	Nulo	Nulo	Nulo

Note que la instrucción CALL es similar a la operación JMP con la diferencia de que deberá guardar la dirección de retorno en un Flip-Flop.

La instrucción RET tambien es similar a la operación JMP con la diferencia de que siempre deberá saltar a la direción de retorno previamente salvada por CALL.

Dado que la Mini-ALU no cuenta con una pila, puede usar cualquiera de los registros R1,...,R7 para pasar los parametros a la función y devolver el valor de retorno.

Observe el siguiente ejemplo en pseudo-código que implementa un sub-rutina que suma R1 + R2 y devuelve el valor en R3

```
`define MY_FUNCTION_ADD 16'hCAFE

//Guarde el primer argumento de entrada a la funcion
`STO, `R1, 16'hACED
//Guarde el segundo  argumento de entrada a la funcion
`STO, `R2, 16'hDEAD

//llamar la subrutina
`CALL, ` MY_FUNCTION_ADD, 16'b0
```

```
…..
//-------------------------------------------------------
//La funcion que suma dos valores
// MY_FUNCTION_ADD
`ADD, `R3, R1, R2
//salir de la subrutina
`RET, 24'b0
//-------------------------------------------------------
```

Finalmente, implemente una subrutina para imprimir un caracter en la pantalla LCD.