Sistema de Comunicación Segura Usando Código Morse y Encriptación XOR*

Martin J A., Moreno Juan S. **

10 de marzo de 2025

Índice

*Rev. 1

**jmartinmo@unal.edu.co ,juamorenogo@unal.edu.co

1. Descripción general

1.1. Objetivos

A continuación se presentan los objetivos y cuales fueron cumplidos:

- Implementar un sistema de entrada de mensajes mediante pulsos cortos y largos (código Morse).✓
- Traducir el mensaje en código Morse a texto alfabético utilizando un diccionario interno.✓
- Desarrollar un generador de secuencia LFSR para la creación de una clave de encriptación.✓
- Encriptar el mensaje utilizando la clave generada y el código binario (ASCII) del mensaje.✓
- Desarrollar un proceso de recepción que permita la desencriptación del mensaje utilizando la misma clave.✓
- Visualizar el mensaje original desencriptado en una pantalla o terminal.✗

En general, todos los objetivos fueron cumplidos de manera exitosa asegurando un funcionamiento completo y visualizable.

1.2. Descripción de la solución realizada comparando con la solución propuesta

El problema a solucionar por este proyecto se basa en la necesidad de un sistema de comunicación segura que permita la transmisión de mensajes en código Morse, los cuales puedan ser encriptados y desencriptados de manera eficiente utilizando una clave generada por un generador de secuencia LFSR. La motivación detrás del problema radica en explorar métodos de comunicación digital con elementos históricos como el código Morse, combinados con técnicas modernas de encriptación para mejorar la seguridad. Las etapas pensadas son las siguientes:

- **Captura de mensajes en código Morse:** Entrada mediante pulsos cortos y largos.
- **Conversión a texto alfabético:** Uso de un diccionario interno para traducir el código Morse.
- **Generación de clave de encriptación:** Implementación de un generador LFSR (Linear Feedback Shift Register) para obtener una clave pseudoaleatoria.
- **Encriptación del mensaje:** Aplicación de la operación XOR entre la clave generada y la representación en código binario (ASCII) del mensaje.
- **Transmisión del mensaje:** Envío del mensaje encriptado a través de un canal digital.
- **Recepción y desencriptación:** Aplicación de la operación XOR con la clave original para recuperar el mensaje.
- **Visualización del mensaje:** Presentación del mensaje desencriptado en una pantalla LCD o terminal.

Por otro lado, la solución obtenida al final del proceso realizo varios cambios para proporcionar una solución mas simple pero de igual funcionalidad. Las siguientes etapas se mantuvieron:

- **Captura de mensajes en código Morse:** Entrada mediante pulsos cortos y largos.
- **Conversión a texto alfabético:** Uso de un diccionario interno para traducir el código Morse.
- **Encriptación del mensaje:** Aplicación de la operación XOR entre la clave generada y la representación en código binario (ASCII) del mensaje.
- **Recepción y desencriptación:** Aplicación de la operación XOR con la clave original para recuperar el mensaje.
- **Visualización del mensaje:** Presentación del mensaje desencriptado en una pantalla LCD o terminal.

En general, simplemente se dejo de lado la etapa de transmisión serial, debido a que, este proceso involucra variaciones del Clock, y al revisar el log de **nextpnr**, se obtuvo el siguiente resultado de estadísticas:

```
5.49. Printing statistics.

info: Device utilisation:
Info:           ICESTORM_LC:  1075/  7680    13%
Info:          ICESTORM_RAM:     0/    32     0%
Info:                 SB_IO:    29/   256    11%
Info:                 SB_GB:     8/     8   100%
Info:          ICESTORM_PLL:     0/     2     0%
Info:           SB_WARMBOOT:     0/     1     0%
```

Listing 1 – Stats del Diseño

En general, todos los parámetros están dispuestos de manera optima y con un amplio rango de operación para implementar mas funcionalidades, no obstante, el parámetro **STGB** esta al maximo. Este parámetro describe los *buffers globales del reloj*, que de manera simple, describen las señales de alta velocidad derivadas del reloj y debido a que estan al maximo, no se podria generar ninguna otra funcionalidad que dependa de un sub-reloj, por lo que, no se implementa una funcionalidad de comunicación serial debido a capacidades físicas.

En general, a excepción de la transmisión serial, la solución propuesta se logro alcanzar de manera exitosa.

1.3. Diseño Analógico

El circuito analógico se basa en la implementación general usada para las diferentes entradas y salidas de la FPGA. Los componentes generales son los siguientes:

- **LEDS**
- **Botones**
- **Display LCD 20x4**
- **Resistencias pull-down**
- **FPGA BlackIce MX**

El bloque inicial de la implementación analógica se basa en el circuito de lector morse, que se basa simplemente en 3 botones que sirven para inyectar el pulso que se convertirá según lo siguiente:

- Pulso corto ->10
- Pulso Largo ->11
- Pulso Vacio ->00

Los siguientes pulsos solo cumplen con el objetivo de eliminar el pulso inmediatamente anterior y de enviar la totalidad del resultado tal como se ve en la Figura 1:

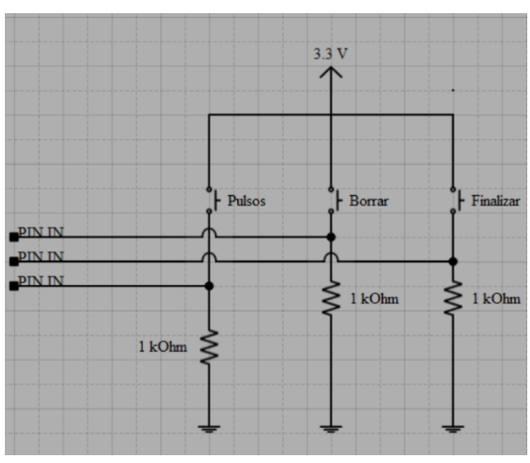

Figura 1 – Circuito analógico de entrada morse

PIN IN representa las entradas digitales en la FPGA, en este caso recibirá la entrada de 3 botones diferentes (a quienes se les aplica en el código un módulo para *debouncing*). Cada botón tiene el siguiente fin:

- **Pulsos**: Recibe los pulsos y los clasifica dependiendo de la cantidad de *ticks* (tiempo) que dure presionado. En el código se le asigna 3 posibles estados:
  - Pulso largo
  - Pulso corto
  - Vacío (Se usan 3 estados debido a que hay letras que comparten un patrón hasta cierto punto, por lo que se crea el concepto de vacío para evitar estas ambigüedades).
- **Borrar**: Este botón tiene la función de borrar el último estado del botón de pulsos que se ingresó.
- **Finalizar**: Este indica al programa que ya se terminó de escribir la palabra.

Posteriormente se genera un circuito para la muestra directa del tipo de pulso que se esta obteniendo, tal como se muestra en la Figura 2:

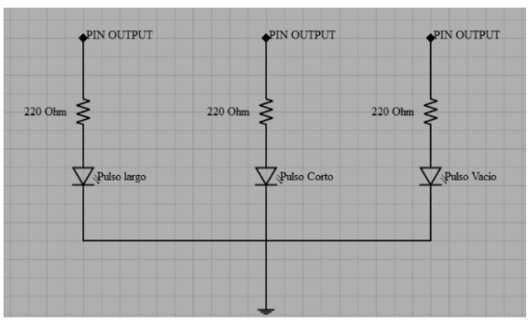

Figura 2 – Circuito analógico de LEDS

Adicionalmente se implementan 10 LED's con el objetivo de visualizar el estado del vector. Finalmente, se muestra el circuito analogico dispuesto para una LCD 20x4, tal como se muestra en la Figura 3:

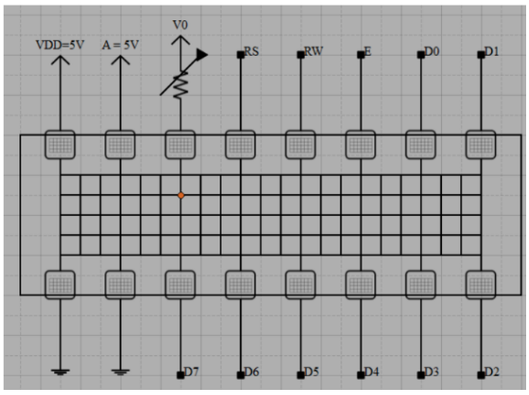

Figura 3 – Circuito analógico de LCD

Ademas se añade un pulsador que sirve como señal de control para imprimir y actualizar la información de la LCD.

2. Planteamiento del problema

2.1. Estructura de la solución

Inicialmente, se planteó la siguiente secuencia de módulos para la solución del problema por medio de la FPGA. En la Figura 4, se presenta el diagrama de bloques:

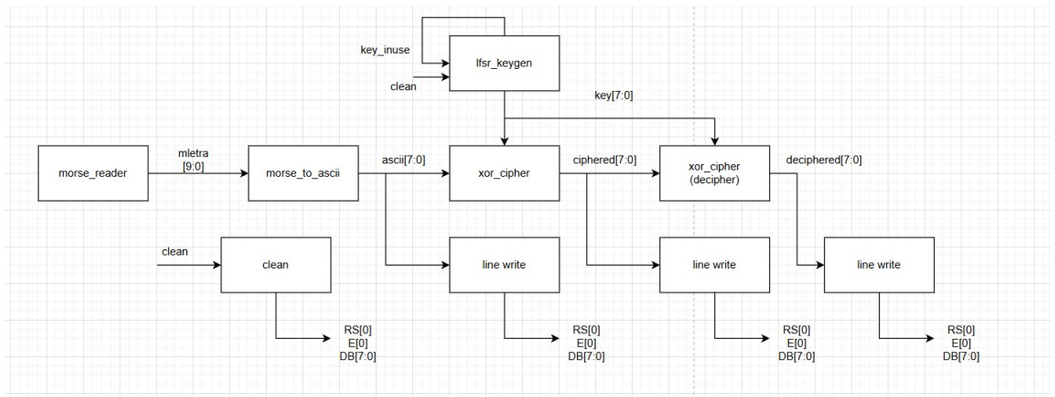

Figura 4 – Secuencia de módulos inicial.

El objetivo de esta secuencia es iniciar por medio del módulo de "morse reader", el cual se encarga de recibir los patrones de morse por medio de los botones. Acto seguido entra al módulo de "morse to ascii", en el cual se identifica el patrón de morse y se guarda en un registro la letra equivalente en ascii. Se mostraría, entonces, en pantalla la letra y a su vez se mandaría a una etapa de encriptación por medio de una llave que se va generando y variando por medio de un registro de desplazamiento lfsr. Se mostraría, luego, al palabra encriptada y luego la desencriptación.

Debido a las dificultades presenadas en el desarrollo debido a las limitaciones de la FPGA, la estructura propia del código y uso de recursos (buffers en particular), se tuvo que replantear el código para que no funcionara por medio de varias instancias de módulos sino que funcionara como un solo módulo integrado con los demás, dejando un módulo bastante extenso pero igualmente sintético y funcional. Se explica a continuación.

Se muestra la instanciación del módulo final con sus entradas y salidas:

```
1  `timescale 1ns / 1ps
2
3  module final1 (
4      input wire clk,
5      input wire button,
6      input wire del_button,
7      input wire fin_button,
8      output led_1,
9      output led_2,
10     output led_3,
11     output led_4,
12     output led_5,
13     output led_6,
14     output led_7,
15     output led_8,
16     output led_9,
17     output led_10,
18     output ledshort,
19     output lednull,
20     output ledlong,
21     input wire show,
22     output reg RS,
23     output reg RW,
24     output reg E,
25     output reg D0,
26     output reg D1,
27     output reg D2,
28     output reg D3,
29     output reg D4,
30     output reg D5,
31     output reg D6,
32     output reg D7
33 );
34 assign RW= 1'b0;
```

Listing 2 – Entradas y salidas del módulo final

- Se muestra la entrada general del clock, necesaria para todas las operaciones secuenciales, como es previsible

- Se muestran las entradas de los botones principales para el ingreso de letras en morse: el indicador de la letra, de borrar y de finalizar/enviar la letra. Se añade otro botón para mostrar en pantalla (actualizar)
- Se muestran salidas referentes a los led's, que funcionan como indicadores del guardado de los datos por cada letra según su orden de entrada y de la duración de las pulsaciones.
- Las siguientes salidas son las respectivas a las entradas de la LCD. La entrada de escritura/lectura, la entrada de habilitación, de configuración/datos, y el bus de datos.

Destaca al final de este fragmento de código que se iguala la salida de lectura/escritura a ALTO, puesto que en ningún momento se va a leer los datos de la LCD, de forma que no va a variar.

```
    reg [7:0] ascii;
    reg [7:0] ciphered;
    reg [7:0] key;
    integer counter_index_save = -1;
```

Listing 3 – Variables generales

Luego se declaran los registros respectivos para guardar temporalmente las letras en ascii, su versión cifrada, la llave de encriptación y un índice de contador para ir guardando a lo largo de una memoria bidimensional cada letra guardada y encriptada.

```
    reg [9:0] letra_final = 10'b0;
    reg PB_state, PB_down, PB_up ; //Debouncer Entry
    reg PB_state_del, PB_down_del, PB_up_del ; //Debouncer Delete
    reg PB_state_fin, PB_down_fin, PB_up_fin ; //Debouncer Send
    reg PB_state_show, PB_down_show, PB_up_show; //Debouncer Send
    wire start;
    reg [16:0] ticks_press;
    reg [9:0] letra = 10'b00000_00000; //guardado de letra morse
    reg [2:0] control1 = 3'b000;
    reg [2:0] contador = 3'b000;
    reg start_signal = 1'b0;

    Debouncer pdb (
        .clk(clk),
        .PB(button),
        .PB_state(PB_state),
        .PB_down(PB_down),
        .PB_up(PB_up)
    );
    Debouncer pdb_del (
        .clk(clk),
        .PB(del_button),
        .PB_state(PB_state_del),
        .PB_down(PB_down_del),
        .PB_up(PB_up_del)
    );
    Debouncer pdb_fin (
        .clk(clk),
        .PB(fin_button),
        .PB_state(PB_state_fin),
        .PB_down(PB_down_fin),
        .PB_up(PB_up_fin)
    );
    Debouncer pdb_show (
        .clk(clk),
        .PB(show),
        .PB_state(PB_state_show),
        .PB_down(PB_down_show),
        .PB_up(PB_up_show)
    );
    ticks_generator ticki (
        .clk(clk),
        .start(start),
        .ticks(ticks_press)
    );
```

Listing 4 – Variables e instancias para el funcionamiento de la lectura morse

Se declaran las variables respectivas para el funcionamiento del módulo integrado del "morse reader.ªquí: registros para detectar los estados del antirrebote, para la incialización y control de ticks temporales, contadores y variables de control. Se añade además un registro en el cual se va guardando la letra final y otro para la letra parcial que se va guardando. Luego de esto, las instancias a submódulos "Debouncer.vz "ticks generator.v".

```
    reg enable_pulse = 0; // Seal de control para E
    reg control=0;
    reg [7:0] DB; // Registro para almacenar los datos a enviar
    reg [3:0] state = 0;
    reg [3:0] char_index = 0;
    reg [16:0] ticks_wire;
    reg [16:0] prev_ticks = 0;
    reg [7:0] base_word [0:15];
  reg [7:0] encr_word [0:15];
  reg [7:0] decr_word [0:15];
    reg [7:0] message_1l [0:4];
    reg [7:0] message_2l [0:4];
    reg [7:0] message_3l [0:4];
    reg [7:0] to_write1l [0:14];
    reg [7:0] to_write2l [0:14];
    reg [7:0] to_write3l [0:14];
    integer i;
    reg [7:0] key = 8'b1110_01110;
    integer variable=0;

    ticks_generator_lcd d0 (
        .clk(clk),
        .start(1'b1), // Start = 0 --> No esta contando // Start = 1 --> Si esta contando
        .ticks(ticks_wire)
    );
```

Listing 5 – Variables e instancias para el funcionamiento del LCD

Se declaran, ahora las variables e instancias respectivas para el funcionamiento de la LCD. Entre estas: indicador de habilitación, una variable de control para las máquinas de estados, el registro de estado, un índice para la escritura progresiva para las letras en la LCD, registros para manejo de tiempos por ticks, y registros para las diferentes letras que se van a mostrar en la LCD. Finalmente, una instancia de ticks.

```
    initial begin

        base_word[0] = " "; base_word[1] = " "; base_word[2] = " "; base_word[3] = " ";
            base_word[4] = " "; base_word[5] = " "; base_word[6] = " "; base_word[7] = " ";
            base_word[8] = " "; base_word[9] = " ";

        encr_word[0] = " "; encr_word[1] = " "; encr_word[2] = " "; encr_word[3] = " ";
            encr_word[4] = " "; encr_word[5] = " "; encr_word[6] = " "; encr_word[7] = " ";
            encr_word[8] = " "; encr_word[9] = " ";

        decr_word[0] = " "; decr_word[1] = " "; decr_word[2] = " "; decr_word[3] = " ";
            decr_word[4] = " "; decr_word[5] = " "; decr_word[6] = " "; decr_word[7] = " ";
            decr_word[8] = " "; decr_word[9] = " ";

        message_1l[0] = "B"; message_1l[1] = "a"; message_1l[2] = "s"; message_1l[3] = "e";
            message_1l[4] = ":";

        message_2l[0] = "E"; message_2l[1] = "n"; message_2l[2] = "c";
        message_2l[3] = "r"; message_2l[4] = ":";

        message_3l[0] = "D"; message_3l[1] = "e"; message_3l[2] = "c";
        message_3l[3] = "r"; message_3l[4] = ":";
    end
```

Listing 6 – Bloque de inicialización

Se da un bloque de inicialización en el cual se colocan los valores iniciales de los registros de escritura para la LCD, de los cuales algunos cambiarán y otros no, pero que son necesarios para ejecutar las siguientes partes del código.

```
    always @(posedge clk) begin

        if(PB_state & ~PB_down) begin // BOTON ABIERTO = 1
            start_signal <= 1'b0;
```

```
        end else if(~(PB_state & ~PB_down)) begin // BOTON CERRADO = 0
            start_signal <= 1'b1;
        end

        if(PB_down_del) begin
            letra <= {2'b00, letra[9:2]};

        end else if (PB_state_del & ~PB_down_del) begin
            if (ticks_press > 17'b0 && ticks_press < 2000) begin // PULSO CORTO 10
                ledshort <=1'b1;
                ledlong <= 1'b0;
                lednull <= 1'b0;
                if(PB_down) begin
                    letra <= {letra[7:0], 2'b10 }; // Operacion de concatenacion y desplazamiento
                end
            end else if(ticks_press >= 4000 && ticks_press <= 7000 ) begin // PULSO LARGO 11
                ledshort <=1'b1;
                ledlong <= 1'b1;
                lednull <= 1'b0;
                if(PB_down) begin
                    letra <= {letra[7:0], 2'b11 }; // Operacion de concatenacion y desplazamiento
                end
            end else if (ticks_press > 7000) begin // PULSO VACIO 00
                ledshort<= 1'b1;
                ledlong <= 1'b1;
                lednull <= 1'b1;
                if(PB_down) begin
                    letra <= {letra[7:0], 2'b00 }; // Operacion de concatenacion y desplazamiento
                end
            end else if (ticks_press == 0) begin
                ledshort<= 1'b0;
                ledlong <= 1'b0;
                lednull <= 1'b0;
            end
        end

        if(~PB_down_fin) begin // BOTON ABIERTO = 1 // BOTON FINAL
            $display("abierto");
        end else if(PB_down_fin) begin // BOTON CERRADO = 0 --> Envio de datos al final
                    case (letra)
                    10'b00_00_00_00_00: ascii <= ""; // ESPACIO: sin entrada (00 00 00 00 00)
                    10'b10_11_00_00_00: ascii <= "A"; // A: .- (10 11 00 00 00)
                    10'b11_10_10_10_00: ascii <= "B"; // B: -... (11 10 10 10 00)
                    10'b11_10_11_10_00: ascii <= "C"; // C: -.-. (11 10 11 10 00)
                    10'b11_10_10_00_00: ascii <= "D"; // D: -.. (11 10 10 00 00)
                    10'b10_00_00_00_00: ascii <= "E"; // E: . (10 00 00 00 00)
                    10'b10_10_11_10_00: ascii <= "F"; // F: ..-. (10 10 11 10 00)
                    10'b11_11_10_00_00: ascii <= "G"; // G: --. (11 11 10 00 00)
                    10'b10_10_10_10_00: ascii <= "H"; // H: .... (10 10 10 10 00)
                    10'b10_10_00_00_00: ascii <= "I"; // I: .. (10 10 00 00 00)
                    10'b10_11_11_11_00: ascii <= "J"; // J: .--- (10 11 11 11 00)
                    10'b11_10_11_00_00: ascii <= "K"; // K: -.- (11 10 11 00 00)
                    10'b10_11_10_10_00: ascii <= "L"; // L: .-.. (10 11 10 10 00)
                    10'b11_11_00_00_00: ascii <= "M"; // M: -- (11 11 00 00 00)
                    10'b11_10_00_00_00: ascii <= "N"; // N: -. (11 10 00 00 00)
                    10'b11_11_11_00_00: ascii <= "O"; // O: --- (11 11 11 00 00)
                    10'b10_11_11_10_00: ascii <= "P"; // P: .--. (10 11 11 10 00)
                    10'b11_11_10_11_00: ascii <= "Q"; // Q: --.- (11 11 10 11 00)
                    10'b10_11_10_00_00: ascii <= "R"; // R: .-. (10 11 10 00 00)
                    10'b10_10_10_00_00: ascii <= "S"; // S: ... (10 10 10 00 00)
                    10'b11_00_00_00_00: ascii <= "T"; // T: - (11 00 00 00 00)
                    10'b10_10_11_00_00: ascii <= "U"; // U: ..- (10 10 11 00 00)
                    10'b10_10_10_11_00: ascii <= "V"; // V: ...- (10 10 10 11 00)
                    10'b10_11_11_00_00: ascii <= "W"; // W: .-- (10 11 11 00 00)
                    10'b11_10_10_00_00: ascii <= "X"; // X: -..- (11 10 10 00 00)
                    10'b11_10_11_11_00: ascii <= "Y"; // Y: -.-- (11 10 11 11 00)
                    10'b11_11_10_10_00: ascii <= "Z"; // Z: --.. (11 11 10 10 00)
                    10'b11_11_11_11_11: ascii <= "0"; // 0: -----
                    10'b10_11_11_11_11: ascii <= "1"; // 1: .----
                    10'b10_10_11_11_11: ascii <= "2"; // 2: ..---
                    10'b10_10_10_11_11: ascii <= "3"; // 3: ...--
                    10'b10_10_10_10_11: ascii <= "4"; // 4: ....-
                    10'b10_10_10_10_10: ascii <= "5"; // 5: .....
                    10'b11_10_10_10_10: ascii <= "6"; // 6: -....
                    10'b11_11_10_10_10: ascii <= "7"; // 7: --...
                    10'b11_11_11_10_10: ascii <= "8"; // 8: ---..
```

```
                10'b11_11_11_11_10: ascii <= "9"; // 9: ----.
            endcase

            base_word[counter_index_save-1] <= ascii;
            ciphered <= key ^ letra;
            encr_word[counter_index_save-1] <= ciphered;

            counter_index_save <= counter_index_save + 1;

            if(counter_index_save==20) begin
                counter_index_save<=0;
            end
            letra <= 0;

            if(variable == 0)begin
                base_word[0]=" ";
                variable<=1;
                counter_index_save <= 0;
            end
        end
    end
```

Listing 7 – Estructura .ªlways"para recepción de letras, guardado y encriptación

Esta sección de código mezcla los siguientes módulos individuales: "morse reader", "morse to asciiz "xor cipher", a parte de guardar las diferentres letras ingresadas. Los primeros condicionales corresponden al control de las entradas temporizadas para morse, y cuando se pulsa el botón de envío se ejecuta una secuencia de pasos según la cual se traduce la palabra de morse a ASCII, se guarda en un lugar específico de ciertos registros para la LCD, se controla para los índices siguientes y se cifra a la vez esta letra y se guarda la encriptación para que se muestre en pantalla. Esta tarea se ejecuta siempre, de forma que siempre se está pendiente de las acciones de los botones y siempre puede guardarse una nueva letra dada la combinación de los botones en particular.

```
    always @(posedge clk) begin

        // Construir los mensajes extendidos
        for (i = 0; i < 10; i = i + 1) begin
            to_write1l[i] <= message_1l[i];
            to_write2l[i] <= message_2l[i];
            to_write3l[i] <= message_3l[i];
        end
        for (i = 0; i < 20; i = i + 1) begin
            to_write1l[i+5] <= base_word[i];
            to_write2l[i+5] <= encr_word[i];
            to_write3l[i+5] <= decr_word[i];
        end

        if (ticks_wire != prev_ticks) begin
            case (state)
                0: begin // configuracin a modo 8 bits, 2 lneas
                //led_9 <= 1'b0;
                    if(control==0) begin
                        enable_pulse <= 0;
                        control <= 1;
                        RS <= 0;
                        DB <= 8'b00111100;
                    end else begin
                        enable_pulse <= 1;
                        control <= 0;
                        state <= state + 1;
                    end
                end
                1: begin // clear Display
                    if(control==0) begin
                        enable_pulse <= 0;
                        control <= 1;
                        RS <= 0;
                        DB <= 8'b00000001;
                    end else begin
                        enable_pulse <= 1;
                        control <= 0;
                        state <= state + 1;
                    end
```

```
                end
                2: begin // return Home
                    if(control==0) begin
                        enable_pulse <= 0;
                        control <= 1;
                        RS <= 0;
                        DB <= 8'b00000010;
                    end else begin
                        enable_pulse <= 1;
                        control <= 0;
                        state <= state + 1;
                    end
                end
                3: begin // display ON, sin cursor
                    if(control==0) begin
                        enable_pulse <= 0;
                        control <= 1;
                        RS <= 0;
                        DB <= 8'b00001100;
                    end else begin
                        enable_pulse <= 1;
                        control <= 0;
                        state  <= state + 1;
                    end
                end

                4: begin // direccin de la primera lnea
                    if(control==0) begin
                        enable_pulse <= 0;
                        control <= 1;
                        RS <= 0;
                        DB <= 8'h80; // Direccin 0x00
                    end else begin
                        enable_pulse <= 1;
                        control <= 0;
                        state <= state + 1;
                    end
                end
                5: begin // escribir "Base:+WORD"
                    if(control==0) begin
                        enable_pulse <= 0;
                        control <= 1;
                        RS <= 1;
                        DB <= to_write1l[char_index];
                        char_index <= char_index + 1;
                    end else begin
                        enable_pulse <= 1;
                        control <= 0;
                        if (char_index == 15) begin
                            char_index <= 0;
                            state <= state + 1;
                        end
                    end
                end
                6: begin // direccin de la segunda lnea
                    if(control==0) begin
                        enable_pulse <= 0;
                        control <= 1;
                        RS <= 0;
                        DB <= 8'hC0; // Direccin 0x14
                    end else begin
                        enable_pulse <= 1;
                        control <= 0;
                        state <= state + 2;
                    end
                end
                7: begin // escribir "Encr:++"
                    if(control==0) begin
                        enable_pulse <= 0;
                        control <= 1;
                        RS <= 1;
                        DB <= to_write2l[char_index];
                        char_index <= char_index + 1;
                    end else begin
                        enable_pulse <= 1;
                        control <= 0;
```

```
                    if (char_index == 15) begin
                        char_index <= 0;
                        state <= 8;
                    end
                end
            end
            8: begin // direccin de la tercera lnea
                if(control==0) begin
                    enable_pulse <= 0;
                    control <= 1;
                    RS <= 0;
                    DB <= 8'h94; // Direccin 0x40
                end else begin
                    enable_pulse <= 1;
                    control <= 0;
                    state <= state + 1;
                end
            end
            9: begin // escribir "Decr:"
            //led_9 <= 1'b1;
                if(control==0) begin
                    enable_pulse <= 0;
                    control <= 1;
                    RS <= 1;
                    DB <= to_write2l[char_index];
                    char_index <= char_index + 1;
                end else begin
                    enable_pulse <= 1;
                    control <= 0;
                    if (char_index == 15) begin
                        char_index <= 0;
                        state <= 10;
                    end
                end
            end
            10: begin //IDLE
                if (~(PB_state_show & ~PB_down_show)) begin
                    state <= 0;
                    //led_10<=1'b0;
                end else begin
                    $display("awa");
                    //led_10<= 1'b1;
                end
            end
        endcase
        prev_ticks <= ticks_wire;
    end
end

// **Segundo always** para manejar el Enable 'E' de manera separada
always @(posedge clk) begin
    E <= enable_pulse;
end

// Asignar los bits de DB a las salidas D0-D7
always @(*) begin
    D0 = DB[0];
    D1 = DB[1];
    D2 = DB[2];
    D3 = DB[3];
    D4 = DB[4];
    D5 = DB[5];
    D6 = DB[6];
    D7 = DB[7];
end
```

Listing 8 – Estructura .ªlways"para escritura permanente en la LCD

Se muestra un bloque .ªlways"que consiste de una actualización constante de los datos a mostrar en pantalla y la respectiva máquina de estados para el manejo de la LCD; luego, dos bloques .ªlways.ªuxiliares para asignar los registros a las entradas de la LCD. La máquina de estados consiste en una serie de pasos secuenciales que configuran en modo 8 bits, 2 líneas en modo 8 bits, 2 líneas, que limpian el display, pone el cursor al inicio (Home), prende el display y quita el cursor de visualización, se pone en cada línea respectiva y se escribe aquello que se quiere escribir. Luego, por medio de un estado idle que tiene una entrada de control por medio

de un botón "show", permite reciclar y actualizar los datos en la pantalla.

```
    assign start = start_signal;

    //Asignaciones leds funcionales
    always @(posedge clk) begin
        if(button) begin
            led_v3 <= 1'b1;
        end else if (del_button) begin
            led_v3 <= 1'b1;
        end else if (fin_button) begin
            led_v3 <= 1'b1;
        end else begin
            led_v3 <= 1'b0;
        end
    end

assign led_1 = letra[0]; //LED PRENDIDO = BOTON ABIERTO
assign led_2 = letra[1];
assign led_3 = letra[2];
assign led_4 = letra[3];
assign led_5 = letra[4];
assign led_6 = letra[5];
assign led_7 = letra[6];
assign led_8 = letra[7];
assign led_9 = letra[8];
assign led_10 = letra[9];


endmodule
```

Listing 9 – Finalización del código

Finalmente, solo se termina por hacer conexiones por asignación de salidas con ciertos registros manejados a lo largo del código.

2.2. Tamaño de Solución

Se genera un log que obtiene la siguiente información:

```
    5.49. Printing statistics.

=== final1 ===

   Number of wires:             927
   Number of wire bits:        2140
   Number of public wires:      927
   Number of public wire bits: 2140
   Number of ports:              29
   Number of port bits:          29
   Number of memories:            0
   Number of memory bits:         0
   Number of processes:           0
   Number of cells:            1466
     $print                       2
     $scopeinfo                   6
     SB_CARRY                   241
     SB_DFF                     177
     SB_DFFE                    196
     SB_DFFESR                   79
     SB_DFFESS                   37
     SB_DFFSR                    81
     SB_LUT4                    647
```

Listing 10 – Bloque de inicialización

Para calcular el número aproximado de transistores equivalentes en el diseño, analizamos los principales componentes utilizados en la FPGA:

- **SB_LUT4 (Look-Up Tables de 4 entradas):** Las LUT4 son esenciales en las FPGA para implementar funciones lógicas. Cada una se construye con una memoria SRAM de 16 bits y circuitería de selección. Una LUT4 utiliza aproximadamente 16 transistores.

$$647 \times 16 = 10,352 \text{ transistores}$$

- **Flip-Flops (SB_DFF, SB_DFFE, SB_DFFESR, SB_DFFESS, SB_DFFSR):** Los flip-flops son registros de almacenamiento utilizados en circuitos secuenciales. En este diseño se emplean diferentes tipos de flip-flops, y cada uno usa en promedio 20 transistores.

$$570 \times 20 = 11,400 \text{ transistores}$$

- **SB_CARRY (Cadenas de propagación de acarreo):** Los elementos de acarreo se utilizan en operaciones aritméticas y generalmente requieren 4 transistores cada uno.

$$241 \times 4 = 964 \text{ transistores}$$

- **Estimación total de transistores:** Sumando los transistores de todos los elementos principales:

$$10,352 + 11,400 + 964 = 22,716 \text{ transistores}$$

Este cálculo proporciona una aproximación razonable de la cantidad de transistores utilizados en el diseño de la FPGA.

3. Conclusiones

- La FPGA usada fue Open Source, lo que facilito su proceso de desarrollo al poder usar herramientas directas desde la consola, también limito las aplicaciones y a su vez, tiene alta dependencia al kernel UNIX en sistemas operativos Linux, lo que añade la dificultad del manejo de todo el desarrollo al tener en cuenta las problemáticas con este sistema.
- Se intento generar soluciones de manipulación de la LCD sin usar maquinas de estado, no obstante, rápidamente se volvió obvio la facilidad que el uso de maquinas de estado en la configuración del LCD.
- Cualquier implementación de electrónica digital, implica el uso de miles o hasta millones de transistores.
- La conversión de código Morse a texto alfabético fue implementada exitosamente mediante un diccionario interno, facilitando la interpretación de los mensajes por parte del sistema.
- La encriptación del mensaje utilizando la clave fija. Por otro lado, el proceso de desencriptación, aunque aún no fue implementado, sigue el mismo principio de la encriptación y se espera que su implementación no represente mayores desafíos técnicos.
- La visualización del mensaje en un display LCD ha sido uno de los aspectos más retadores. A la final, se opto por hacer una implementación directa usando comunicacion paralela.
- La modularización del código en Verilog ha facilitado el desarrollo y la depuración del sistema, permitiendo la implementación de cada funcionalidad de manera independiente antes de su integración final.
- Aunque se han superado varios retos técnicos, aún existen aspectos por resolver, especialmente en la generación aleatoria de la clave LFSR, la transmisión mediante UART y la integración completa del display LCD.
- En términos generales, el proyecto ha demostrado la viabilidad de combinar técnicas históricas de comunicación, como el código Morse, con enfoques modernos de encriptación digital, proporcionando un esquema funcional para la transmisión segura de mensajes.

4. Trabajos Futuros

Se propone estudiar, comprender y realizar soluciones futuras enfocadas en la comunicación serial, tales como protocolos UART e I2C, debido a que, por limitaciones técnicas de la FPGA, no se pudo generar ningún tipo de acercamiento a esto.

Referencias

- Gajski, D. D. (1997). *Principles of Digital Design*. Prentice Hall.
- Wakerly, J. (2003). *Diseño Digital, Ed. 3 – Principios y Prácticas*.
- Ciletti, M. D. (2003). *Advanced Digital Design with the Verilog HDL*. Prentice Hall India.
- Harris, D., Harris, S. (2004). *Digital Design and Computer Architecture*. Morgan Kaufmann.
- Zeidman, B. (2002). *Designing with FPGAs and CPLDs*. Elsevier.
- Ashenden, P. (2008). *Digital Design: An Embedded Systems Approach Using Verilog*. Morgan Kaufmann.
- Grout, I. (2008). *Digital Systems Design with FPG*. Newnes.