



Control de Acceso Inteligente con RFID

Iván Cabrera Altamirano, *Tópicos Selectos en Sistemas Digitales: VHDL*

Dr. Arturo Díaz, Dr. Adriano de Luca, CINVESTAV-IPN. Departamento de Computación

Resumen— El presente documento describe el desarrollo de un sistema de control de acceso automático utilizando las credenciales inteligentes de los alumnos y personal administrativo para el Departamento de Computación del CINVESTAV para controlar el acceso a las distintas áreas del departamento; el documento presenta un análisis detallado del desarrollo de dicho sistema, incluyendo diagramas que demuestren gráficamente los algoritmos utilizados, además del desarrollo de código fuente utilizado en el mismo.

Índice de Términos—

I. INTRODUCCIÓN

EL presente documento describe el desarrollo de un sistema de control de acceso automático utilizando las credenciales inteligentes de los alumnos y personal administrativo para el departamento de computación del CINVESTAV para controlar el acceso a las distintas áreas del departamento. La justificación que se presenta para la elaboración del proyecto consiste en proveer un sistema automático de acceso a las distintas áreas del departamento eliminando las posibles fallas asociadas, por ejemplo: pérdida de llaves, uso no autorizado de áreas (salones, auditorios), entre otras. El diseño consiste de módulos que serán integrados de forma estructural para completar el proyecto

El sistema se desarrollará en un *FPGA Spartan3-E®* programado en VHDL; además, se utilizará la herramienta *ModelSim* para propósitos de simulación.

II. ANÁLISIS

A. Arquitectura del sistema: Cliente-Servidor

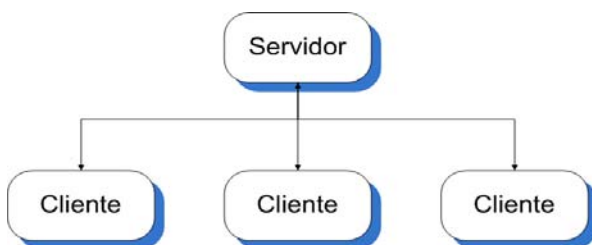


Figura 1. Sistema Cliente – Servidor

B. Componentes del sistema: Reloj de tiempo real

Por definición, tenemos que el reloj de tiempo real es el módulo encargado de llevar la hora y fecha del sistema. Éste módulo se encuentra implementado como un contador, cuenta segundos, minutos, horas, días, meses y años; cuenta con una entrada de pulsos, que debe corresponder a una frecuencia de 1Hz.

El siguiente código de implementación muestra el algoritmo para dicho módulo de reloj de tiempo real:

```

ELSIF CLK = '1' AND CLK'EVENT THEN
  Seconds <= Seconds + 1;
  IF Seconds = "111011" THEN
    Seconds <= "000000";
    Minutes <= Minutes + "000001";
    IF Minutes = "111011" THEN
      Minutes <= "000000";
      Hours <= Hours + "00001";
      IF Hours = "11000" THEN
        Hours <= "00000";
        Days <= Days + "00001";
        IF Days = "10000" THEN
          Months <= Months + "0001";
          Days <= "00001";
        ELSIF days = "11111" THEN
          IF Months = "0100" OR
             Months = "0110" OR
             Months = "1001" OR
             Months = "1011" THEN
            Months <= Months + "0001";
            Days <= "00001";
          END IF;
        ELSIF Days = "11110" THEN
          IF Months = "0010" THEN
            Months <= Months + "0001";
            Days <= "00001";
          END IF;
        ELSIF Days = "11101" THEN
          IF Months = "0010" THEN
            Months <= Months + "0001";
            Days <= "00001";
          END IF;
        END IF;
        IF Months = "1101" THEN
          Months <= "0001";
          Years <= Years + "000001";
        END IF;
      END IF;
    END IF;
  END IF;
END IF;
END IF;
  
```

Listado 1. Implementación de RTC

La estrategia consiste en realizar cuentas de pulsos de reloj e ir comparando para incrementar la variable correspondiente, es decir cuando el contador de segundos sea 60, el contador de minutos debe incrementarse. Al mismo tiempo, cuando el contador de minutos sea 60, el contador de horas debe cambiar. Esta misma estrategia se realiza con los días, meses y años.

El código mostrado fue sintetizado y simulado de forma satisfactoria obteniendo los siguientes resultados:

■ Síntesis

	Maximun Frequency	Number of Slices
Real Time Clock	179.969 MHz	88

Tabla 1. Síntesis en el módulo de reloj de tiempo real

■ Simulación

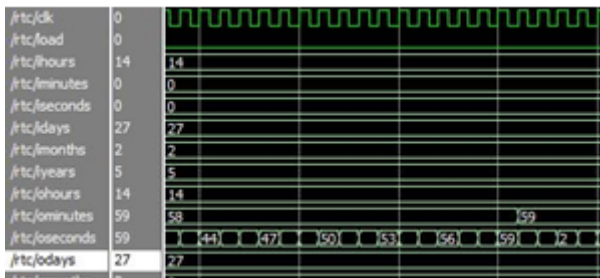


Figura 2. Simulación de Reloj de Tiempo Real

En la figura 2 se puede apreciar el funcionamiento del módulo de reloj de tiempo real, en la parte superior tenemos la entrada de ciclos de reloj, más abajo podemos apreciar cómo van cambiando las señales correspondiente a segundos y minutos respectivamente.

C. Componentes del sistema: Temporizador

Éste módulo del sistema se encarga de contar unidades de tiempo y emitir una señal cuando se ha alcanzado el tiempo especificado.

Tal módulo se implementa como un contador de pulsos, que cuando alcanza el número de ciclos correspondientes, se emite una señal para avisar que el tiempo ha transcurrido.

El siguiente es la descripción del código fuente que muestra el algoritmo implementado:

```

case ActualState is
  when IDLE =>
    if Start = '1' then
      NextCounter <= "0000000000000000";
      nextState <= COUNTING;
    end if;
  when COUNTING =>
    if unsigned(ActualCounter) = unsigned(Input) then
      Output <= '1';
      nextState <= IDLE;
    else
      NextCounter <= ActualCounter + "0000000000000001";
      Output <= '0';
    end if;
end case;

```

Listado 2. Implementación del Temporizador

El código anterior fue sintetizado y simulado satisfactoriamente obteniendo los siguientes resultados:

■ Síntesis

	Maximun Frequency	Number of Slices
Timer Compare	176.308 MHz	22

Tabla 2. Síntesis en el módulo de Temporizador

■ Simulación

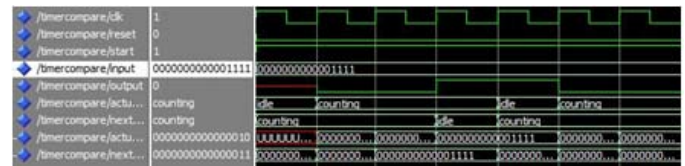


Figura 3. Simulación de Temporizador

En la figura 3 podemos ver el resultado de la simulación del temporizador, donde obtenemos un pulso cuando se ha alcanzado la cuenta de ciclos esperada.

D. Componentes del sistema: Divisor de frecuencia

Éste módulo del sistema se encarga de dividir la frecuencia principal de reloj; se implementa como un contador, donde cada determinado número de ciclos de reloj, se emite una señal y la cuenta vuelve a comenzar.

```

process (clk)
  variable contador: integer range 0 to 100000000;
  begin
    if (clk'event and clk='1') then
      if (contador <=25000000) then
        contador := contador + 1;
        clk2 <= '1';
      end if;
      if (contador >25000000) then
        if (contador <=50000000) then
          contador := contador + 1;
          clk2 <= '0';
        else
          contador :=0;
        end if;
      end if;
    end if;
  end process;

```

Listado 3. Implementación de Divisor de Frecuencia

■ Síntesis

	Maximun Frequency	Number of Slices
Frequency Divisor	96.765MHz	80

Tabla 3. Síntesis en el módulo de Divisor de frecuencia

■ Simulación

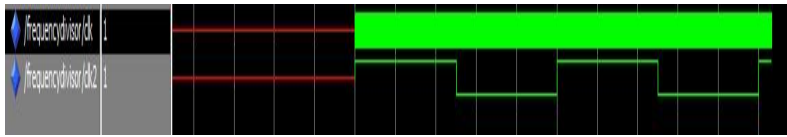


Figura 4. Simulación de Divisor de frecuencia

E. Componentes del sistema: Comunicación serial

Éste módulo del sistema se encarga de convertir un dato de 8 bits, en un dato serial a un determinado *Baudrate*; dicho módulo se implementa como una máquina de estados finitos.

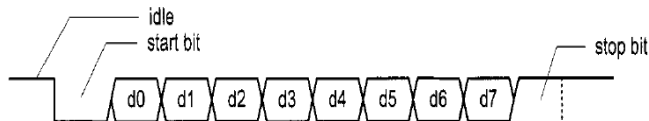


Figura 5. Comunicación Serial

En un sistema de comunicación serial asíncrona tal y como el que se especifica y diseña en esta etapa de proyecto tenemos que una palabra de n bits es convertida en pulsos seriales con las siguientes características.

Al principio tenemos un bit de inicio o *start bit*, el cual se representa como una transición de alto a bajo.

Enseguida tenemos los datos a transmitir, comenzando por el menos significativo, el número de bits a transmitir comúnmente es de 8, sin embargo es posible ver en algunos sistemas que este número varía entre 5 y 9.

Para finalizar debemos enviar un bit de parada o *stop bit*, el cual se representa como una transición de bajo a alto.

Por último, en esta implementación no se contempla utilizar el chequeo de paridad, este es un mecanismo que se utiliza para verificar si existe algún error en la transmisión de los datos.

Este mecanismo hoy en día es poco utilizado, por lo cual se omitió de esta implementación.

Otra característica no incluida en la presente implementación es el control de flujo por hardware, dado que la mayoría de los sistemas que hacen uso de un estándar de comunicación serial no hacen uso del control de flujo.

Un sistema de comunicación serial como el aquí presentado se compone de diferentes módulos, entre los que destacan:

1. Transmisor
2. Receptor

3. Buffer de almacenamiento temporal.

El módulo de transmisión se encarga de convertir un dato de n bits, en impulsos seriales, con su inicio y fin a determinado *baudrate*.

El módulo de recepción se encarga de convertir un impulso de datos seriales con inicio y fin a un dato de n bits.

El buffer de almacenamiento temporal se encarga de guardar bytes que han sido recibidos o a transmitir, esto permitirá que el sistema de comunicaciones trabaje en forma paralela a otros módulos del sistema.

a. Código del transmisor

```

CASE ActualState IS
  WHEN IDLE =>
    NextTx <= '1';
    IF TxStart = '1' THEN
      NextState <= START;
      NextSampling <= "0000";
      NextBitStream <= TxData;
    END IF;
  WHEN START =>
    NextTx <= '0';
    IF ActualSampling = 15 THEN
      NextState <= DATA;
      NextSampling <= "0000";
      NextBitCounter <= "000";
    ELSE
      NextSampling <= ActualSampling + 1;
    END IF;
  WHEN data =>
    NextTx <= ActualBitStream(0);
    IF ActualSampling = 15 THEN
      NextSampling <= "0000";
      NextBitStream <= '0' & ActualBitStream(7 DOWNTO 1);
      IF ActualBitCounter = 7 THEN
        NextState <= STOP;
      ELSE
        NextBitCounter <= ActualBitCounter + 1;
      END IF;
    ELSE
      NextSampling <= ActualSampling + 1;
    END IF;
  WHEN STOP =>
    NextTx <= '1';
    IF ActualSampling = 15 THEN
      NextState <= IDLE;
      TxComplete <= '1';
    ELSE
      NextSampling <= ActualSampling + 1;
    END IF;
END CASE;

```

Listado 4. Implementación del Transmisor

El código presentado con anterioridad implementa la transmisión serial mediante una máquina de estados finitos, con los estados de IDLE, START, DATA y STOP que corresponden a las etapas anteriormente descritas, donde en cada estado se realiza el envío de los correspondientes bits.

F. Componentes del sistema: LCD

Éste módulo del sistema se encarga de inicializar y manejar una pantalla de cristal líquido (LCD) de 16x2 caracteres.

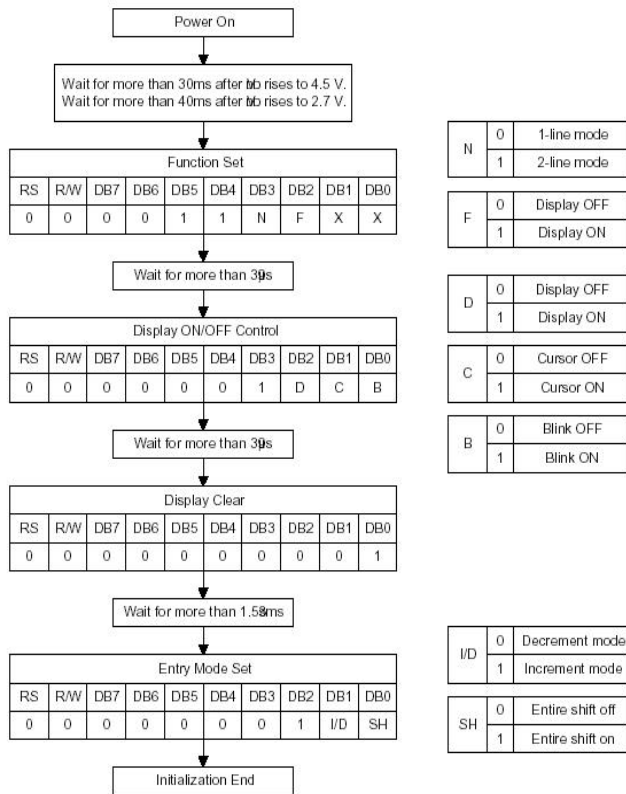


Figura 8. Rutina de inicialización de LCD.

La inicialización que es requerida para utilizar una pantalla de LCD se muestra en la figura 8.

Después de haber energizado la pantalla, es necesario esperar entre 30 y 40 ms, para mandar el primer comando, denominado *Function Set*, donde se establece el número de líneas y si el display se encontrara apagado o encendido. Después de haber enviado este comando deberemos esperar cerca de 40us.

El siguiente comando a enviar es, *Display ON/OFF Control*, donde se especifica si el display se encuentra apagado o encendido, si se muestra o no el cursor, y si este se encuentra parpadeando o no. Después de haber enviado este comando deberemos esperar cerca de 40us.

El siguiente comando a enviar es, *Display Clear*, el cual mostrara el LCD vacio. Después de haber enviado este comando deberemos esperar cerca de 1.6ms.

Por último debemos enviar el comando de *Entry Mode Set*, donde habiendo enviado este comando el LCD se encontrara listo para recibir datos y ser presentados en pantalla.

La siguiente máquina de estados muestra el funcionamiento del módulo, en base al diagrama anterior:

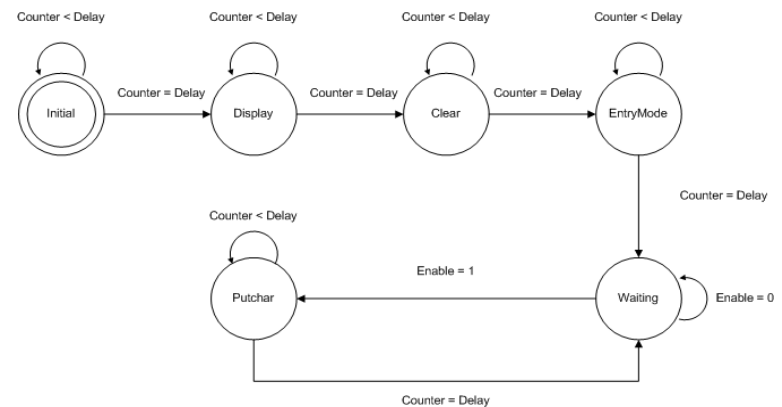


Figura 9. FSM para la operación del LCD.

El algoritmo implementado en código fuente se muestra a continuación:

```

CASE ActualState IS
WHEN INITIAL =>
  IF ActualInitialCounter < big_delay THEN
    NextInitialCounter <= ActualInitialCounter + 1;
  ELSE
    IF ActualCounter < Setup_Delay then
      LCD_ENABLE <= '1';
    ELSE
      LCD_ENABLE <= '0';
    END IF;
    lcd_data <= SET_DISPLAY;
    lcd_rs <= '0';
    lcd_rw <= '0';
    if ActualCounter = small_delay then
      NextState <= DISPLAY;
      NextCounter <= 0;
    else
      NextCounter <= ActualCounter+1;
    end if;
  end if;
WHEN DISPLAY =>
  IF ActualCounter < Setup_Delay then
    LCD_ENABLE <= '1';
  ELSE
    LCD_ENABLE <= '0';
  END IF;
  lcd_data <= DISPLAY_ON;
  lcd_rs <= '0';
  lcd_rw <= '0';
  if ActualCounter = small_delay then
    NextState <= CLEAR;
    NextCounter <= 0;
  else
    NextCounter <= ActualCounter + 1;
  end if;

```

```

WHEN CLEAR =>
  IF ActualCounter < Setup_Delay then
    LCD_ENABLE <= '1';
  ELSE
    LCD_ENABLE <= '0';
  END IF;
  lcd_data <= CLEAR_SCREEN_CURSOR;
  lcd_rs <= '0';
  lcd_rw <= '0';
  if ActualCounter = big_delay then
    NextState <= ENTRYMODE;
    NextCounter <= 0;
  else
    NextCounter <= ActualCounter + 1;
  end if;
WHEN ENTRYMODE =>
  IF ActualCounter < Setup_Delay then
    LCD_ENABLE <= '1';
  ELSE
    LCD_ENABLE <= '0';
  END IF;

  lcd_data <= SET_ENTRY_MODE;
  lcd_rs <= '0';
  lcd_rw <= '0';

  if ActualCounter = small_delay then
    NextState <= WAITING;
    NextCounter <= 0;
  else
    NextCounter <= ActualCounter+1;
  end if;
when WAITING =>
  if enable='1' then
    NextState <= PUTCHAR;
  end if;
when PUTCHAR =>
  IF ActualCounter < Setup_Delay then
    LCD_ENABLE <= '1';
  ELSE
    LCD_ENABLE <= '0';
  END IF;
  lcd_data <= message;
  lcd_rs <= '1';
  lcd_rw <= '0';
  if ActualCounter=big_delay then
    NextState <= WAITING;
    NextPosition <= ActualPosition+1;
    NextCounter <= 0;
  else
    NextCounter <= ActualCounter+1;
  end if;
end case;
    
```

Listado 7. Implementación del LCD

■ Síntesis

	Maximun Frequency	Number of Slices
LCD	160.059 Mhz	25

Tabla 5. Síntesis en el módulo de LCD

■ Simulación

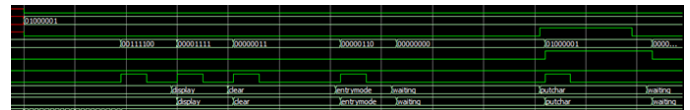


Figura 10. Simulación del módulo de LCD

G. Componentes del sistema: Cliente – Servidor

La arquitectura del sistema está basada en el esquema Cliente – Servidor, donde puede existir un solo maestro para muchos clientes.

- Servidor: Encargado de procesar las solicitudes de los clientes, validar en su base de datos y devolver resultado.
- Cliente: Encargado de realizar las lecturas de los identificadores en cada punto de acceso y de realizar la acción de apertura de puerta.

H. Componentes del sistema: Protocolo Cliente – Servidor

Para que ambos sistemas puedan comunicarse, es necesario contar con una especificación de cómo se van a comunicar en un nivel lógico, por lo tanto es necesario definir un **protocolo de comunicación**.

De tal manera que a un protocolo de petición se le da una respuesta.

Byte Inicial	Tamaño de Comando + Datos	Dirección Destino	Dirección Fuente	Comando	Datos	Verificación	Byte Final
--------------	---------------------------	-------------------	------------------	---------	-------	--------------	------------

Figura 11. Estructura del Protocolo

Nuestro protocolo tendrá los siguientes componentes:

1. Byte Inicial: Indica el inicio de una trama.
2. Tamaño del Paquete de Datos: Indica el tamaño de los datos que contiene la trama.
3. Dirección Destino: ¿A quién le envió la trama de datos?
4. Dirección Fuente: ¿Quién envía la trama de datos?
5. Datos a Enviar: ¿Qué estoy enviando? ¿y cómo lo envío? (*cmd*)
6. Cheque de Verificación: Comprueba integridad de los datos.
7. Byte Final: Indica el final de la trama.

Algunas consideraciones:

1. Byte de Inicio: 0x01
2. Direcciones (Destino & Fuente):
 - 0x00: Servidor
 - 0x01 – 0xFE: Clientes
 - 0xFF: Computadora
3. Verificación: LRC (Longitudinal Redundancy Check)
 - for (i=0; i<Tamaño; i++)
 - LRC = LRC XOR Datos(i);
1. Byte de Final: 0xFE

Comando	Descripción	Tipo	Datos Asociados	Longitud
0xC1	Existe este usuario en la base de datos?	Petición	Usuario	9 bytes
0xD1	Si existe el usuario especificado	Respuesta	Ninguno	1 byte
0xE1	No existe el usuario especificado	Respuesta	Ninguno	1 byte

Figura 12. Comandos validos para Maestro – Esclavo

■ Caso de estudio

Petición: Pregunto al Maestro (0x01) si existe el usuario 0xF1F2F3F4F5F6F7F8

Byte Inicial	Tamaño de Comando + Datos	Dirección Destino	Dirección Fuente	Comando	Datos	Verificación	Byte Final
0x01	0x09	0x01	0x02	0xC1	0xF1 0xF2 0xF3 0xF4 0xF5 0xF6 0xF7 0xF8	0xC3	0xFE

Figura 13. Estructura de una petición

Respuesta: Respondo al Cliente (0x02) que el usuario (0xF1F2F3F4F5F6F7F8) SI existe.

Byte Inicial	Tamaño de Comando + Datos	Dirección Destino	Dirección Fuente	Comando	Datos	Verificación	Byte Final
0x01	0x01	0x02	0x01	0xD1	-	0xD2	0xFE

Figura 14. Estructura de una respuesta

I. Componentes del sistema: Protocolo PC – Servidor

Definido de la misma manera que el protocolo cliente – servidor, con la diferencia de que implementa diferentes comandos.

Comando	Descripción	Tipo	Datos Asociados	Longitud
0xC2	Ajuste de RTC	Petición	Fecha, Hora	7 bytes
0xD2	RTC actualizado	Respuesta	Ninguno	1 byte
0xE2	RTC no actualizado	Respuesta	Ninguno	1 byte
0xC3	Agrega usuario	Petición	Usuario	9 bytes
0xD3	Usuario agregado	Respuesta	Ninguno	1 byte
0xE3	Usuario no agregado	Respuesta	Ninguno	1 byte
0xC4	Solicita evento	Petición	No. Evento	2 bytes
0xD4	Registro de evento	Respuesta	Evento	8 bytes
0xE4	Error al obtener evento.	Respuesta	Ninguno	1 byte

Figura 15. Comando validos para Maestro – PC.

a. Implementación del protocolo Cliente – Servidor – PC

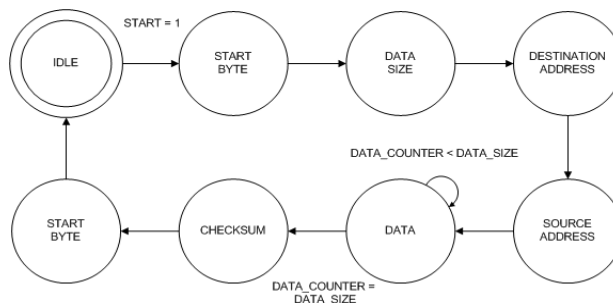


Figura 16. Maquina de Estados Finitos para la implementación del protocolo de comunicaciones.

```

case ActualState is
when IDLE =>
    NextSalida <= "00000000";
    if start = '1' then
        nextState <= START_BYTE;
    end if;
when START_BYTE =>
    NextSalida <= START_BYTE_CONS;
    nextState <= DATA_SIZE;
when DATA_SIZE =>
    NextSalida <= OUTPUT_BUFFER(0);
    nextState <= DESTINATION_ADDRESS;
when DESTINATION_ADDRESS =>
    NextSalida <= OUTPUT_BUFFER(1);
    nextState <= SOURCE_ADDRESS;
when SOURCE_ADDRESS =>
    NextSalida <= OUTPUT_BUFFER(2);
    nextState <= COMMAND;
when COMMAND =>
    NextSalida <= OUTPUT_BUFFER(3);
    nextState <= DATA_STATE;
when DATA_STATE =>
    IF ActualByteCounter = conv_integer(OUTPUT_BUFFER(0)) THEN
        NextState <= CHECKSUM;
        NextByteCounter <= 0;
    ELSE
        NextSalida <= OUTPUT_BUFFER((4+ActualByteCounter));
        NextByteCounter <= ActualByteCounter + 1;
    END IF;
when CHECKSUM =>
    NextSalida <= CHECKSUM_VF;
    nextState <= STOP_BYTE;
when STOP_BYTE =>
    NextSalida <= STOP_BYTE_CONS;
    nextState <= IDLE;
end case;
    
```

Listado 8. Implementación del Protocolo de Comunicación

■ Síntesis

	Maximun Frequency	Number of Slices
Protocolo	161.147 Mhz	179

Tabla 6. Síntesis en el módulo de Protocolo

■ Simulación

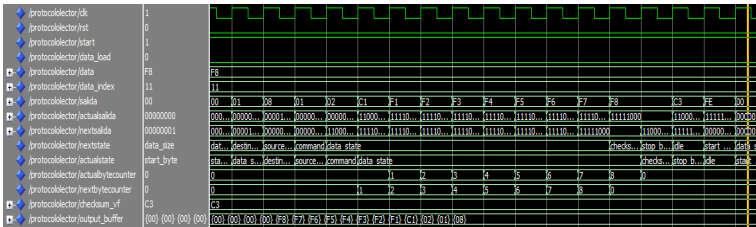


Figura 17. Simulación del modulo Protocolo

J. Componentes del sistema: Lector de tarjetas RFID

a. Investigación



- Texas Instruments S6400.
- Tecnología de Alta Frecuencia HF (13.56 Mhz).
- ISO15693 / ISO14443.
- Interface: RS485 o Wiegand.
- 12V @ 300 mA.
- Protocolo de Petición – Respuesta.

• Protocolo:

- Petición / Respuesta.
- Lector: Direcccionado / No Direcccionado
- Tarjeta: Direcccionado / No Direcccionado
- Comandos:
 - Inventory
 - Read One Block
 - Write One Block
 - Block One Block
 - Read Multiple Blocks
 - Write AFI (Application Family Identifier).
 - Lock AFI
 - Write DSFID (Data Storage Format Identifier).
 - Lock DSFID
 - Read System Information
 - Read Lock Block Status

b. Request to reader

SOH	Length	Device Type	Command Flags	Command	Request Data	Checksum
0x01	Longitud Incluye SOH	0x10 Access Control	Request Flags	Reader / ISO15693 Commands	Data	LRC

Figura 18. Estructura del Protocolo de Comunicación

Bit	Parameter
0	0 = 1/256, 1 = ¼
1	0 = Slow Read, 1 = Fast Read
2	0 = 100%, 1 = 10% TX Modulation
3	0 = FSK, 1 = ASK Tag Modulation
4	0 = Non-Addressed, 1 = Addressed
5	0 = Reader Not Addressed, 1 = Reader Addressed
6	RFU
7	Always 1 for Request

Figura 19. Opciones para el campo Request Flags.

Command	Description
0xE0	Reader Version
0xE5	Reader Reset
0xE7	Reader Setup
0xEB	Reader Information
0xED	Set Reader Mode
0xEE	Activate LED
0xEF	Activate Audio

Figura 20. Comandos disponibles en el Lector S6400.

Command	Description
0x01	Inventory
0x20	Read Single Block
0x21	Write Single Block
0x22	Lock Block
0x23	Read Multiple Blocks
0x27	Write AFI
0x28	Lock AFI
0x29	Write DSFID
0x2A	Lock DSFID
0x2B	Read Tag Information
0x2C	Read Lock Block Status

Figura 21. Comandos disponibles para identificadores ISO15693.

c. Response from Reader

SOH	Length	Device Type	Command Flags	Command	Response Data	Checksum
0x01	Longitud Incluye SOH	0x10 Access Control	Response Flags	Reader / ISO15693 Commands	Data	LRC

Figura 22. Estructura de la respuesta desde el Lector S4100.

Bits	Parameter
0 – 1	Error Flags
2 – 7	Reserved

Figura 23. Campo *Response Flags*.

Bits	Parameter
00	No Error
01	Tag Error
10	Reader Error
11	Reserved

Figura 24. Análisis del Campo *Error Flags*.

Comandos:

Lector: Direcccionado

Identificador: Direcccionado

- Inicializar Lector:
- Inventory:
 - Request
 - Response
- Read One Block:
 - Request
 - Response

Tanto el lector como el identificador se utilizaran en modo direcccionado, es decir le hablaremos específicamente a un lector o a un identificador, esto es dado que esta tecnología permite direcccionar más de un lector / identificador, es necesario a referirnos a uno en específico.

Existe la opción de utilizar el modo no direcccionado, sin embargo al momento de existir más de un lector / identificador tendríamos una colisión con lo cual obtendríamos datos corruptos.

En la siguiente sección se hace el análisis de peticiones y respuestas para diferentes comandos.

Inventory Request:

01 11 00 10 A2 01 31 30 31 30 32 36 39 00 00 9E 61

SOH	Length	Device Type	Command Flags	Command	Request Data	Checksum
0x01	0x11 0x00	0x10	0xA2	0x01	0x31 0x30 0x31 0x30 0x32 0x36 0x39 0x00 0x00	0x9E 0x61
SOH	17 Bytes	Access Control	0xA2 = 1010 0010 Reader Addressed Fast Read	Inventory	Reader Address: 1010269 Read All AFI Cards	LRC

Figura 25. Análisis de *Inventory Request*.

Inventory Response:

01 12 00 10 00 01 00 01 63 36 BA 06 00 00 07 E0 0D F2

SOH	Length	Device Type	Command Flags	Command	Response Data	Checksum
0x01	0x12 0x00	0x10	0x00	0x01	0x01 0x63 0x36 0xBA 0x06 0x00 0x00 0x07 0xE0	0x0D 0xF2
SOH	18 Bytes	Access Control	No Error	Inventory	One Identifier Found ID: E007000006BA3663	LRC

Figura 26. Análisis de *Inventory Response*.

Read Block Request:

01 19 00 10 B2 20 31 30 31 30 32 36 39 00 63 36 BA 06 00
00 07 E0 00 A9 56

SOH	Length	Device Type	Command Flags	Command	Request Data	Checksum
0x01	0x19 0x00	0x10	0xB2	0x20	0x31 0x30 0x31 0x30 0x32 0x36 0x39 0x00 0x00	0xA9 0x56
SOH	25 Bytes	Access Control	0xB2: 1011 0010 Reader Addressed ID Addressed Fast Read	Read One Block	Reader Address: 1010269 ID: E007000006BA3663 Block: 0	LRC

Figura 27. Análisis de Read Block Request.

Read Block Response:

01 0E 00 10 00 20 AA 0A 00 00 00 00 9F 60

SOH	Length	Device Type	Command Flags	Command	Response Data	Checksum
0x01	0x0E 0x00	0x10	0x00	0x20	0x0A 0x00 0x00 0x00 0x00	0x9F 0x60
SOH	14 Bytes	Access Control	No Error	Read One Block	Data: 0000000A Block: 0	LRC

Figura 28. Análisis de Read Block Response.

Initialize Reader Request:

01 12 00 10 A0 ED 31 30 31 30 32 36 39 00 0E 00 7D 82

SOH	Length	Device Type	Command Flags	Command	Request Data	Checksum
0x01	0x12 0x00	0x10	0xA0	0xED	0x31 0x31 0x30 0x30 0x32 0x36 0x39 0x00 0x0E 0x00	0x7D 0x82
SOH	18 Bytes	Access Control	0xA0: 1010 0000 Reader Addressed	Set Reader Mode	Reader: 1010269 Baudrate: 9600 Led / Audio OFF	LRC

Figura 29. Análisis de Initialize Reader Request.

Setup Reader Request:

01 10 00 10 A0 E7 31 30 31 30 32 36 39 00 7B 84

SOH	Length	Device Type	Command Flags	Command	Request Data	Checksum
0x01	0x10 0x00	0x10	0xA0	0xE7	0x31 0x31 0x30 0x30 0x32 0x36 0x39 0x00	0x7B 0x84
SOH	16 Bytes	Access Control	0xA0: 1010 0000 Reader Addressed	Reader Setup	Reader: 1010269	LRC

Figura 30. Análisis de Setup Reader Request.

Setup Reader Response:

01 0D 00 10 00 E7 00 2C 00 30 00 E7 18

SOH	Length	Device Type	Command Flags	Command	Request Data	Checksum
0x01	0x0D 0x00	0x10	0x00	0xE7	00 2C 00 30 00	0xE7 0x18
SOH	16 Bytes	Access Control	No Error	Reader Setup	Wiegand Mode Off ISO15693 Audio/Led ON Baudrate: 9600 AFI: Access Control Only Default Encryption Key	LRC

Figura 31. Análisis de Setup Reader Response.

■ Implementación

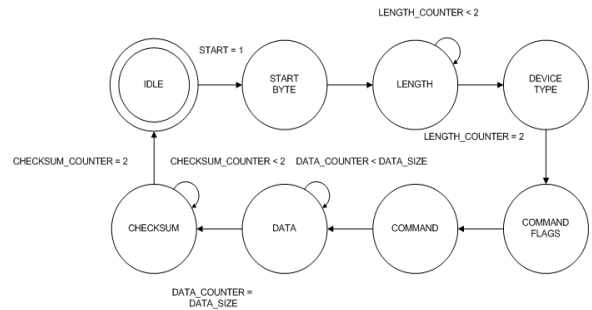


Figura 32. Maquina de Estados Finitos que implementa el protocolo RFID.

El algoritmo implementado en código fuente es el siguiente:

```

case ActualState is
when IDLE_STATE =>
    NextSalida <= "00000000";
    if start = '1' then
        nextState <= START_BYTE_STATE;
    end if;
when START_BYTE_STATE =>
    NextSalida <= START_BYTE_CONS;
    nextState <= LENGTH_STATE;
when LENGTH_STATE =>
    IF ActualByteCounter = 2 THEN
        NextState <= DEVICE_TYPE_STATE;
        NextByteCounter <= 0;
    ELSE
        NextSalida <= OUTPUT_BUFFER(ActualByteCounter);
        NextByteCounter <= ActualByteCounter + 1;
    END IF;
when DEVICE_TYPE_STATE =>
    NextSalida <= OUTPUT_BUFFER(2);
    nextState <= COMMAND_FLAGS_STATE;
when COMMAND_FLAGS_STATE =>
    NextSalida <= OUTPUT_BUFFER(3);
    nextState <= COMMAND_STATE;
when COMMAND_STATE =>
    NextSalida <= OUTPUT_BUFFER(4);
    nextState <= DATA_STATE;
when DATA_STATE =>
    IF ActualByteCounter = conv_integer(OUTPUT_BUFFER(0)) - 8 THEN
        NextState <= CHECKSUM_STATE;
        NextByteCounter <= 0;
    ELSE
        NextSalida <= OUTPUT_BUFFER((5+ActualByteCounter));
        NextByteCounter <= ActualByteCounter + 1;
    END IF;
when CHECKSUM_STATE =>
    IF ActualByteCounter = 2 THEN
        NextState <= IDLE_STATE;
        NextByteCounter <= 0;
    ELSE
        NextSalida <= CHECKSUM_BUFFER(ActualByteCounter);
        NextByteCounter <= ActualByteCounter + 1;
    END IF;
end case;
    
```

Listado 9. Implementación del Protocolo RFID.

■ Síntesis

	Maximun Frequency	Number of Slices
Protocolo RFID	135.820 Mhz	406

Tabla 7. Síntesis en el módulo de Protocolo RFID

M. UART: Descomposición a nivel de entidades

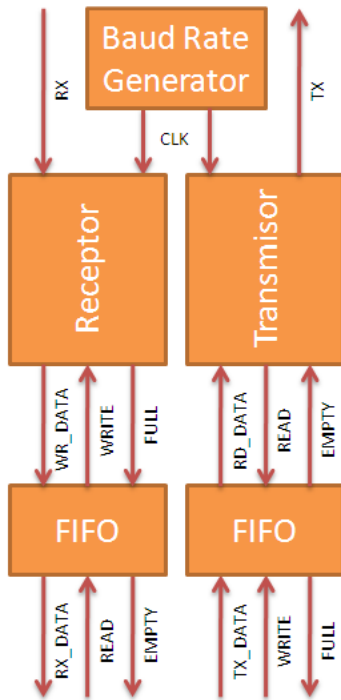


Figura 38. Arquitectura de la UART, a nivel entidades.

a. FIFO

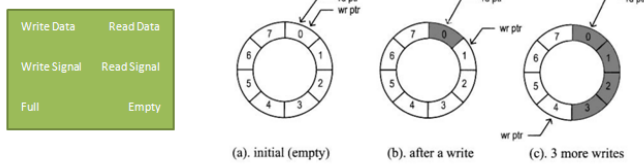


Figura 39. Estructura del buffer FIFO.

N. Controlador esclavo: Descomposición modular

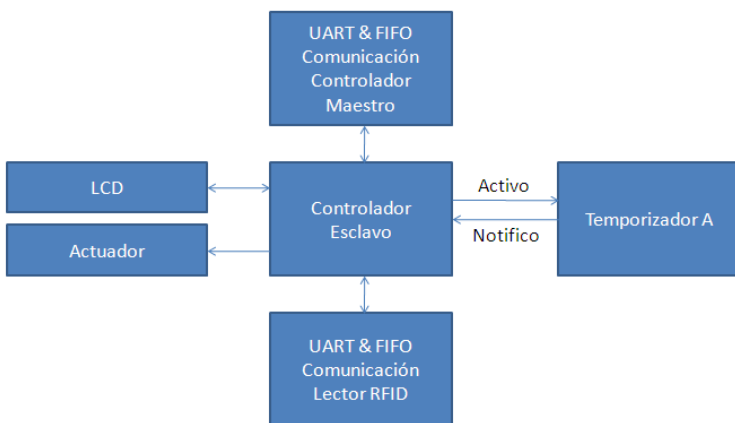


Figura 40. Arquitectura del controlador esclavo.

a. Funcionamiento del módulo de controlador esclavo visto desde el lector de tarjetas RFID:

1. Manda una petición de lectura al modulo de RFID.
2. Espera 100 ms (activando el temporizador). Y lee la respuesta del modulo de RFID (que está en el FIFO).
3. Veo los datos leídos y si corresponden a un identificador mando leer el bloque 0 al modulo de RFID.
4. Espero 100 ms (activando el temporizador). Y leo la respuesta del modulo de RFID (que está en el FIFO).
5. Con los datos obtenidos en 3 y 5, guardo datos en memoria y espero que el controlador maestro me pregunte.

Su representación en máquina de estados finita es la siguiente:

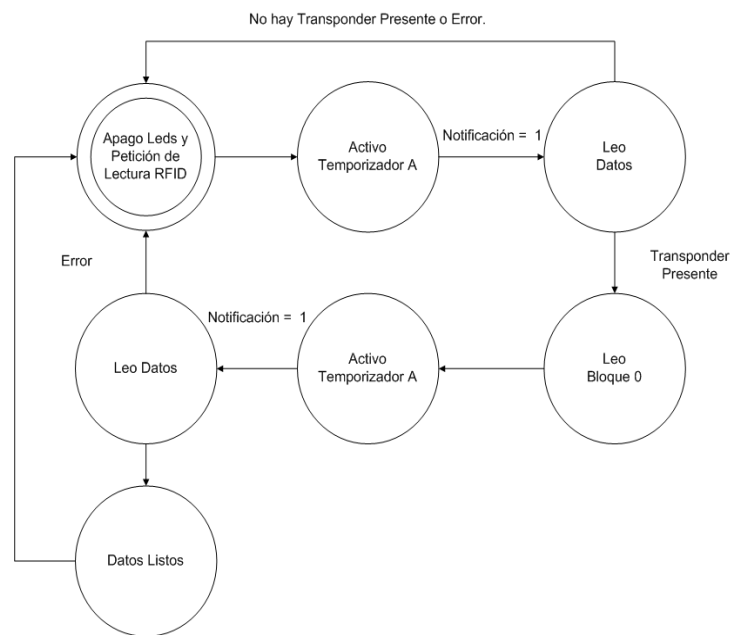


Figura 41. Maquina de Estados Finitos para el Controlador Esclavo, lado Lector de RFID.

b. Funcionamiento del módulo visto desde el lado maestro:

1. Reviso si hay peticiones del Maestro.
2. Si hay peticiones, las leo, respondo y activo temporizador.
3. Cuando el temporizador haya terminado leo la respuesta del maestro y hago lo que me indique. (Prender LED o no – Actuador –).
4. Vuelvo a revisar.

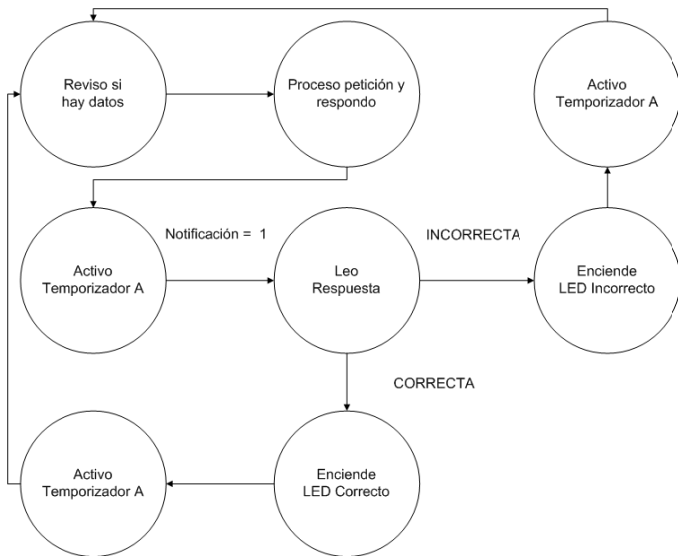


Figura 42. Maquina de estados finitos para el controlador esclavo, lado Comunicación con el Maestro.

O. Controlador Esclavo: Descomposición a nivel entidades

El sistema a su más alto nivel tal y como se muestra en la figura 45 tiene una entrada de reloj, una UART para comunicarse con la PC y otra para comunicarse con el lector de RFID.

Este sistema esta compuesto por 2 entidades: MAESTRO y ESCLAVO. Las cuales se conectan internamente mediante sus puertos de Recepción y Transmisión.

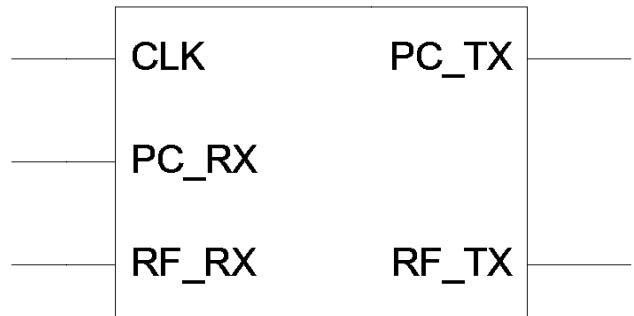


Figura 45. Sistema Completo

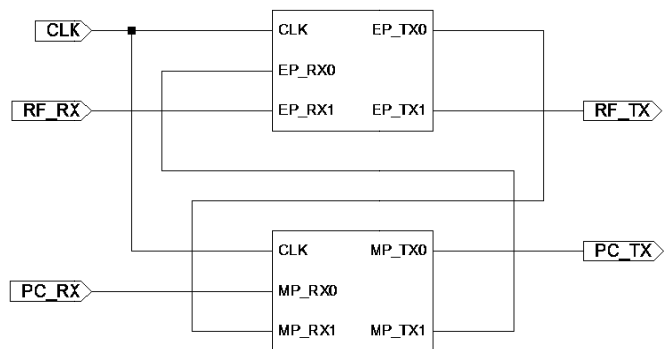


Figura 46. Maestro y Esclavo interconectados internamente.

Figura 43. Arquitectura del controlador esclavo, mostrando las entidades que lo conforman.

La estructura completa del sistema, es tal y como se muestra a continuación:

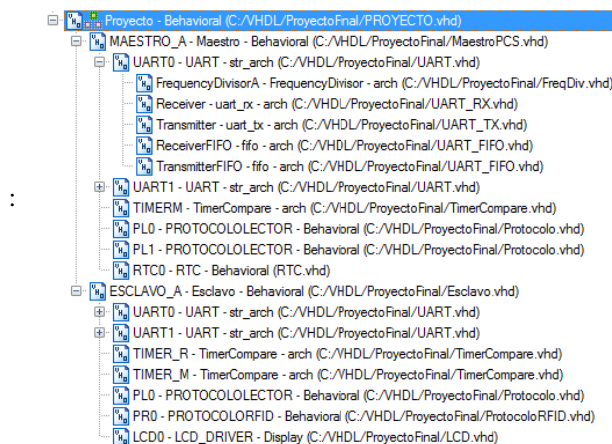


Figura 44. Estructura General del Sistema.

III. CONCLUSIONES

El uso de VHDL para el diseño de circuitos control lógicos es muy recomendable, ya que facilita mucho el diseño debido a su forma de programación, que puede ser un mezcla de estilos, donde podemos modelar diseños muy complejos a partir de diseños muy simples.

Una de las desventajas de este lenguaje, es que aun no existen bibliotecas estandarizas con suficientes componentes, por ejemplo de comunicación por lo que es necesario crear esos componentes desde cero, lo cual complica el diseño e incrementa de forma notable el tiempo de desarrollo.

En lo referente al proyecto, se tomo una estrategia de diseñar módulos y después integrarlos para crear el sistema completo, si bien esta estrategia es adecuada, nuestra limitada

experiencia diseñando en este lenguaje no nos permitió considerar algunos detalles, que en primeras instancias no significaron inconvenientes, sin embargo al momento de llegar a una integración total, estos detalles se convirtieron en significativos. Lo cual indudablemente impacto en el resultado final del proyecto.

Aun así considero esta experiencia como bastante enriquecedora, ya que tuve la oportunidad de adquirir mas conocimientos, y de llegar a un nivel de profundidad mayor en relación a proyectos anteriores.

IV. REFERENCIAS

Roth, Charles – Digital Systems Design using VHDL.

Chu, Pong – FPGA Prototyping by VHDL Examples.

Texas Instruments – HF Training S6400 Host Protocol.