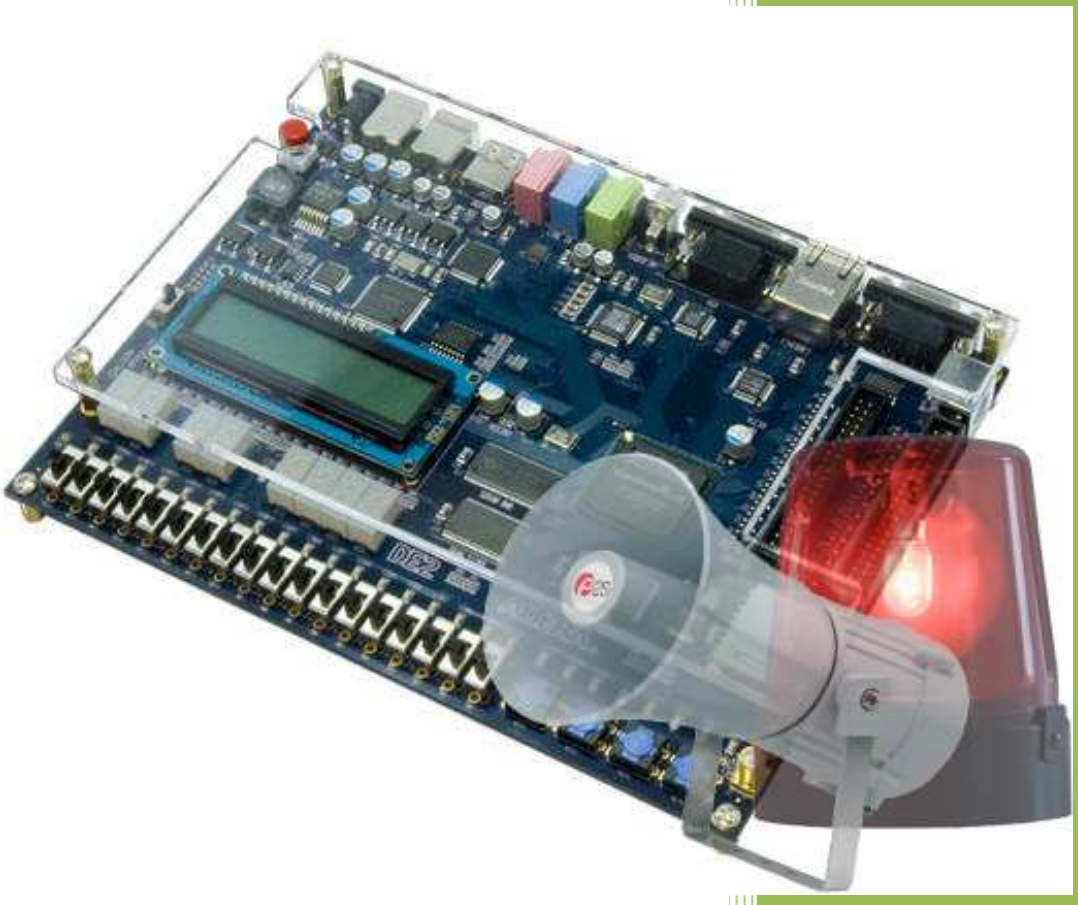
2014

PROYECTO DISEÑO Y CONSTRUCION DE ALARMA ANTI INTRUSION



Anartz Recalde, Markel Picado y Daniel Franco

UPV-EHU, Facultad de Informática, DCSD

INDICE

1. INTRODUCCION

Esta memoria trata sobre el proyecto realizado en la asignatura Diseño y Construcción de Sistemas Digitales en el curso 2013-2014.

El proyecto que documenta esta memoria trata sobre la creación de una alarma anti intrusión basando el proyecto en la construcción de un sistema digital para dicho propósito. Para la realización de este, se han utilizado diferentes herramientas y materiales como puede ser una placa DE2 de Altera, el Hyperterminal y software proporcionado por Altera para el desarrollo de circuitos en la placa como son ModelSim y Quartus.

Por lo tanto, a continuación presentaremos el proyecto realizado situándonos primeramente en el contexto del problema a resolver, objetivos que debemos alcanzar con la realización de este proyecto, explicación del proyecto, así como sus módulos internos, y por último las conclusiones generales obtenidas y las posibles mejoras.

2. ESCENARIO Y PREGUNTA MOTRIZ

Pregunta motriz:

Escenario:

La empresa Mirakonta S.L. junto con el departamento de Arquitectura y Tecnología de Computadores de la Facultad de Informática de UPV/EHU presenta la 2ª edición del concurso de diseño y construcción de sistemas digitales.

- Requisitos:
  - Estar matriculado en la asignatura DCSD
- Especificaciones funcionales mínimas del sistema a diseñar:
  - Conexión serie con PC
  - Configuración de parámetros desde PC mediante aplicación propietaria de la empresa Mirakonta
  - Activación y desactivación desde PC mediante aplicación propietaria de la empresa Mirakonta
  - Activación retardada mediante temporización programable
  - Alarma retardada mediante temporización programable
  - Desactivación mediante la introducción de una clave mediante aplicación propietaria de la empresa Mirakonta
  - Detección de sensores de presencia binarios
  - Indicación en placa y en PC del tiempo restante para activación y del tiempo para alarma
  - Activación de leds en placa para indicar la alarma y posibilidad de activar un elemento externo.

- Material disponible:
  - Herramienta de diseño Quartus
  - Herramienta de simulación ModelSim
  - Hardware de implementación: Placa DE2 de Altera
  - Instrumentación de laboratorio (laboratorio de tecnología FISS)
  - Aplicación propietaria de la empresa Mirakonta
- Premio:

  Viaje a Barcelona y visita guiada a la sala blanca del Centro Nacional de microelectrónica.

3. OBJETIVOS

Podríamos hacer una distinción entre los diferentes objetivos que se han llevado a cabo, por un lado están los **objetivos de la asignatura**y por otro lado los **objetivos del proyecto.**

Objetivos de la asignatura

a. Diseñar sistemas digitales de complejidad media y diferentes propósitos.
b. Utilizar una metodología estructurada en el diseño de sistemas digitales.
c. Conocer y analizar alternativas de construcción de sistemas digitales.
d. Identificar las fases del proceso de diseño de sistemas digitales y analizar las herramientas más utilizadas.
e. Describir un sistema digital mediante el lenguaje VHDL.
f. Editar y simular un diseño mediante herramientas CAD.
g. Construir y verificar un prototipo de un diseño utilizando un dispositivo programable (PLD).

Objetivos del proyecto

h. Conexión serie con PC.
i. Configuración de parámetros desde PC mediante aplicación propietaria de la empresa Mirakonta.
j. Activación y desactivación desde PC mediante aplicación propietaria de la empresa Mirakonta.
k. Activación retardada mediante temporización programable.
l. Alarma retardada mediante temporización programable.
m. Desactivación mediante la introducción de una clave mediante aplicación propietaria de la empresa Mirakonta.
n. Detección de sensores de presencia binarios.
o. Indicación en placa y en PC del tiempo restante para activación y del tiempo para alarma.
p. Activación de leds en placa para indicar la alarma y posibilidad de activar un elemento externo.

4. FUNCIONALIDAD DEL SISTEMA

Nada más encender el sistema las variables se inicializaran (la contraseña se pondrá a 0000, el tiempo de activación y desactivación a 30 segundos, timestamp a 0 y la versión adm3) y acto seguido pasara al estado desactivado. En este estado pueden ocurrir 2 cosas:

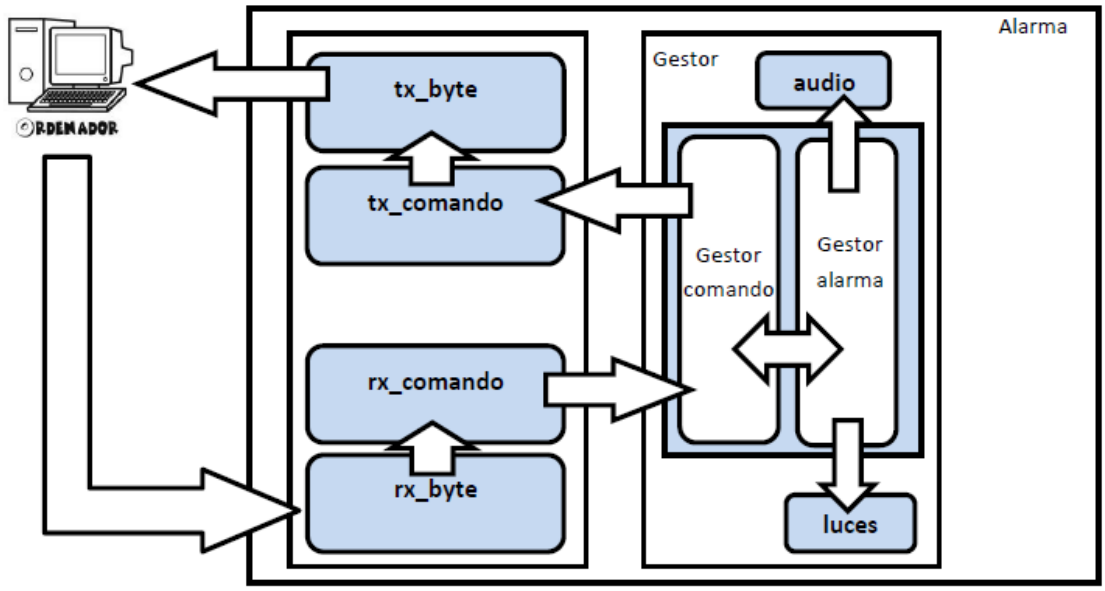
1- Se pasa a configurar el sistema con lo cual podríamos cambiar el tiempo de activación, el tiempo de desactivación y la contraseña del sistema. Para hacer dichos cambios es necesario enviar la contraseña actual una vez introducidos los parámetros. Cuando introducimos la contraseña es posible que falles, en cuyo caso se requerirá de nuevo la contraseña, o si fallas 3 veces o pasan alrededor de 12 segundos pasara a estado desactivado sin haber modificado la configuración. En caso de que la contraseña sea acertada el sistema pasara a estado desactivado pero la configuración se guardara.

2-Desde el estado desactivado también se puede pasar a estado activado. Para ello se requiere también contraseña. Como anteriormente si la contraseña se falla 3 veces o pasan alrededor de 12 segundos se volverá al estado desactivado. Si la contraseña es acertada se esperara el tiempo de activación y una vez finalice dicho tiempo pasara a estado activado. En este estado se espera una señal del sensor. Una vez reciba dicha señal la alarma esperara el tiempo de desactivación, y si este tiempo pasa la alarma pasara a estado saltar alarma en el cual el dispositivo emite luces y sonido para avisar de que tenemos un intruso. Cabe destacar que en cualquier punto del punto 2 podemos desactivar la alarma y volver al estado desactivado, para ello se requerirá también la contraseña. La diferencia respecto a la introducción de contraseña para desactivar la alarma es que si esta esta activada y se falla 3 veces se mantiene en el estado que estaba, ya que si se pasase a estado desactivado el ladro introduciendo 3 veces mal la contraseña se desactivaría

4.1. Módulos:

- rx_byte
- rx_comando
- gestor_comando
- gestor_alarma
- audio
- luces
- tx_comando
- tx_byte

4.2 Conexión entre los módulos



5. ARQUITECTURA

5.1. RX_BYTE

a . Descripción

Este módulo transmite un dato de 1 byte (8 bits, la salida byte), estos 8 bits los recibe uno por uno (por la entrada rx) y cuando tiene el byte completo en la salida lo transmite, esto nos servirá para recibir el comando con o sin parámetros (según el comando y su implementación) enviados por la aplicación Mirakonta y transmitirlos al sistema byte a byte o lo que es lo mismo carácter a carácter.

Señales de entrada:

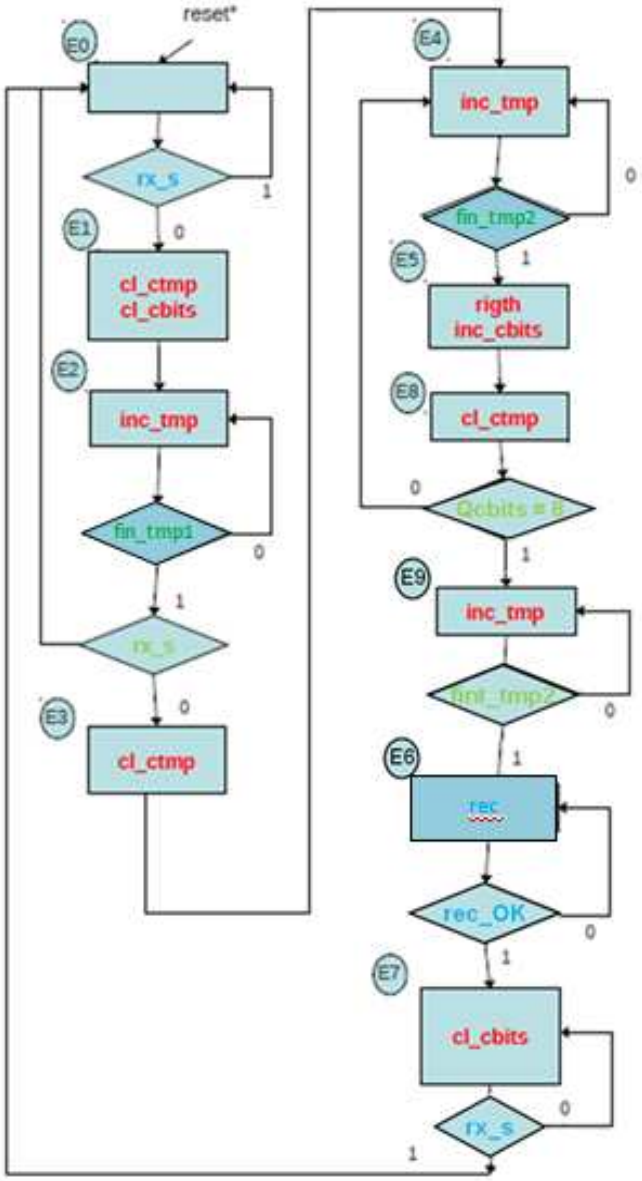
- **rx:** Bit a recibir.
- **rec_OK:**Indica que se ha recibido correctamente el byte.

Señales de salida:

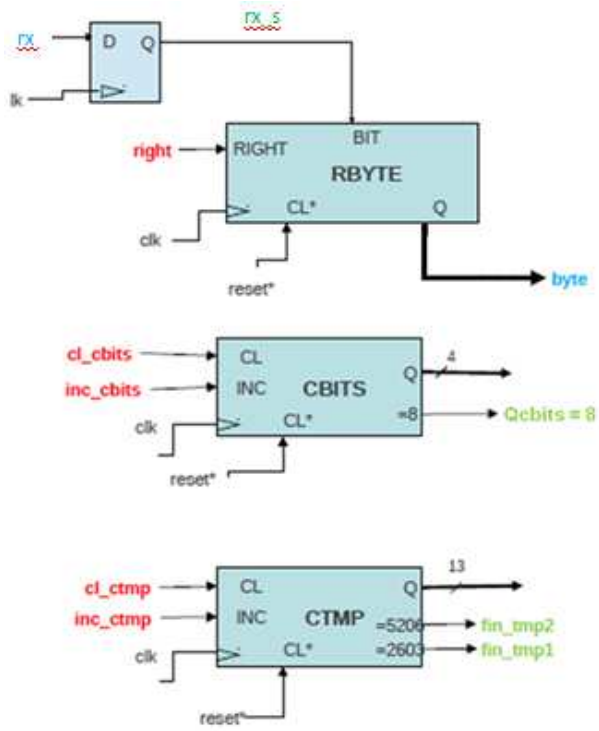
- **byte:** Dato de 8 bits para transmitir, generados por la entrada rx
- **rec:** Indica que se ha terminado la recepción del dato de 8 bits. Y da paso al siguiente modulo a para que coja el dato.

b . Unidad de Control

En el estado E0, nos quedamos esperando a que se desactive la señal rx_s y eso nos indica que a continuación viene un dato. En el estado E1 reseteamos el contador del tiempo y el del número de bits que se nos ha trasmitido, para que después en el estado E2 empecemos a contar “X” tiempo donde “X” es el tiempo especificado para validar que la señal que nos llega es para un dato y que no ha sido una caída de tensión. Una vez comprobado esto en el estado E3 volvemos a resetear el temporizador para volver a contar en el estado E4 cierto tiempo y posicionarnos en la mitad de la señal.
Desde el estado E4 hasta el estado E9 se utilizan para leer los 8 bits y una vez que tenemos estos en el estado E6 activamos la señal E6 para indicar que los hemos recibido, cuando recibimos el rec_OK que significa que ya alguien ha leído nuestro dato en este caso el modulo rx_comando pasamos al estado E7 donde reseteamos el contador de bits y nos quedamos a la espera de que la señal vuelva a el estado de inicio en este caso esperamos que se ponga a 1 ya que el 0 al final se utiliza para indicar que ha terminado de transmitir el dato de 8 bits



c . Unidad de Proceso



d . Código VHDL

```
libraryieee;
useieee.std_logic_1164.all;
useieee.numeric_std.all;

entity rx_byte IS
port
(
    reset, clk     :instd_logic;
    rx             :instd_logic;
    rec_OK         :instd_logic;
      byte         :outunsigned(7downto0);
      rec          :outstd_logic
);
end rx_byte;

architecture a of rx_byte is

-- señales internas de UP
signal rx_s       :std_logic;
signal Qcbits     :unsigned(3downto0);
signal Qrbyte     :unsigned(7downto0);
signal Qctmp      :unsigned(12downto0);

-- estados presentes y siguiente de UC
type estado is(e0,e1,e2,e3,e4,e5,e6,e7,e8,e9);
signal ep,es : estado;

-- señaless UC->UP
signal cl_cbits, cl_ctmp, INC_ctmp,right, INC_cbits :std_logic;

-- señales UP->UC
signal fintmp1, fintmp2:std_logic;
```

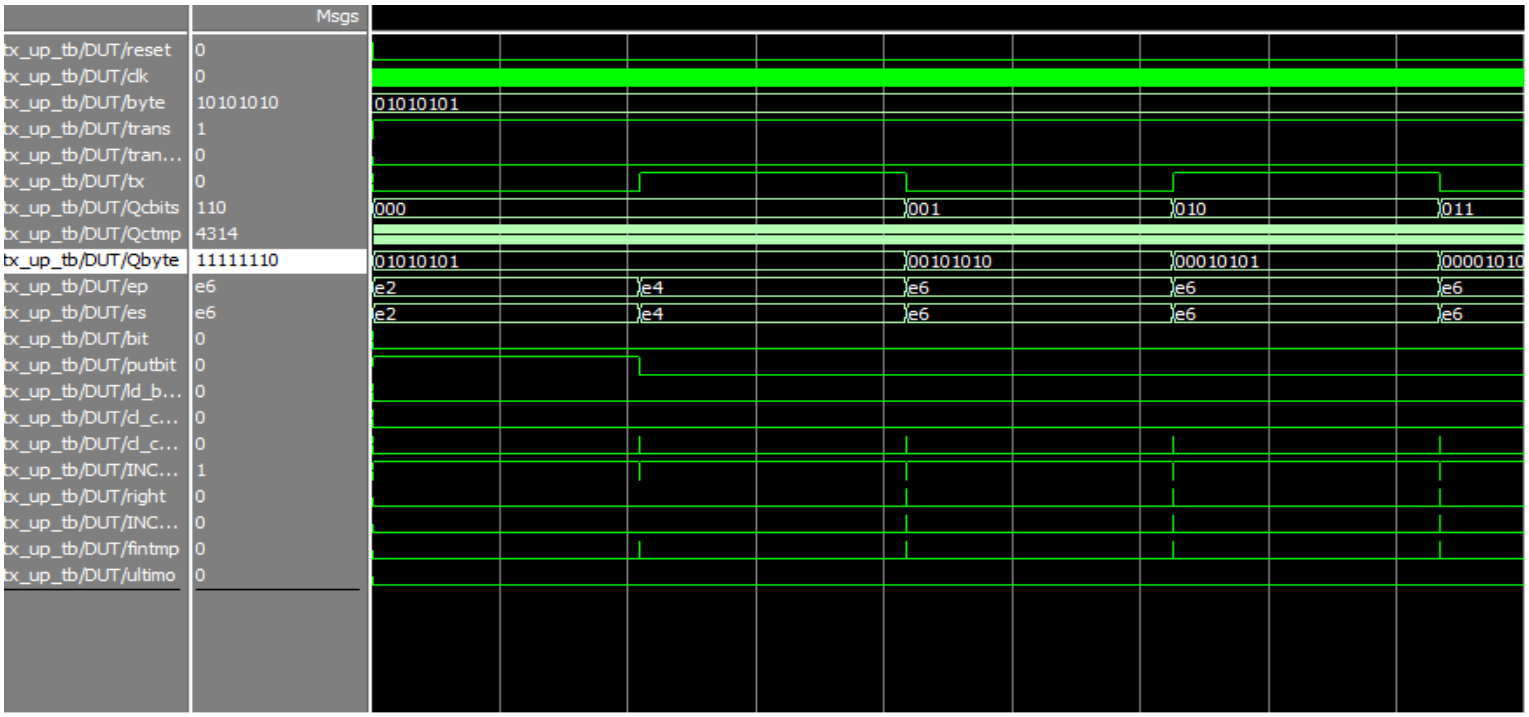
```
begin
--====================================
-- Unidad de Control
--====================================
--------------------------------------
-- Transicion de estados
process(clk)
begin
case ep is
when e0 =>
if rx_s = '1' then es <= e0;
else  es <= e1;
endif;
when e1 =>
         es <= e2;
when e2 =>
if fintmp1 = '1' and rx_s = '0' then es <= e3;
elsif fintmp1 = '0' then es <= e2;
else es <= e0;
endif;
when e3 =>
        es<=e4;
when e4 =>
if fintmp2 = '1' then es <= e5;
else  es <= e4;
endif;
when e5=>
        es<=e8;
when e8 =>
if Qcbits ="1000"then es <= e9;
else  es <= e4;
endif;
when e9 =>
if fintmp2 = '1' then es <= e6;
else  es <= e9;
endif;
when e6 =>
if rec_OK = '1' then es <= e7;
else  es <= e6;
endif;
when e7 =>
if rx_s = '0' then
es<=e7;
else
            es<=e0;
endif;

endcase;

endprocess;
--------------------------------------
-- REGISTRO de estados
process(clk, reset)
begin
if(reset='1')then
          ep <= e0;
elsif(clk'EVENTand clk='1')then
          ep <= es;
endif;
endprocess;
```

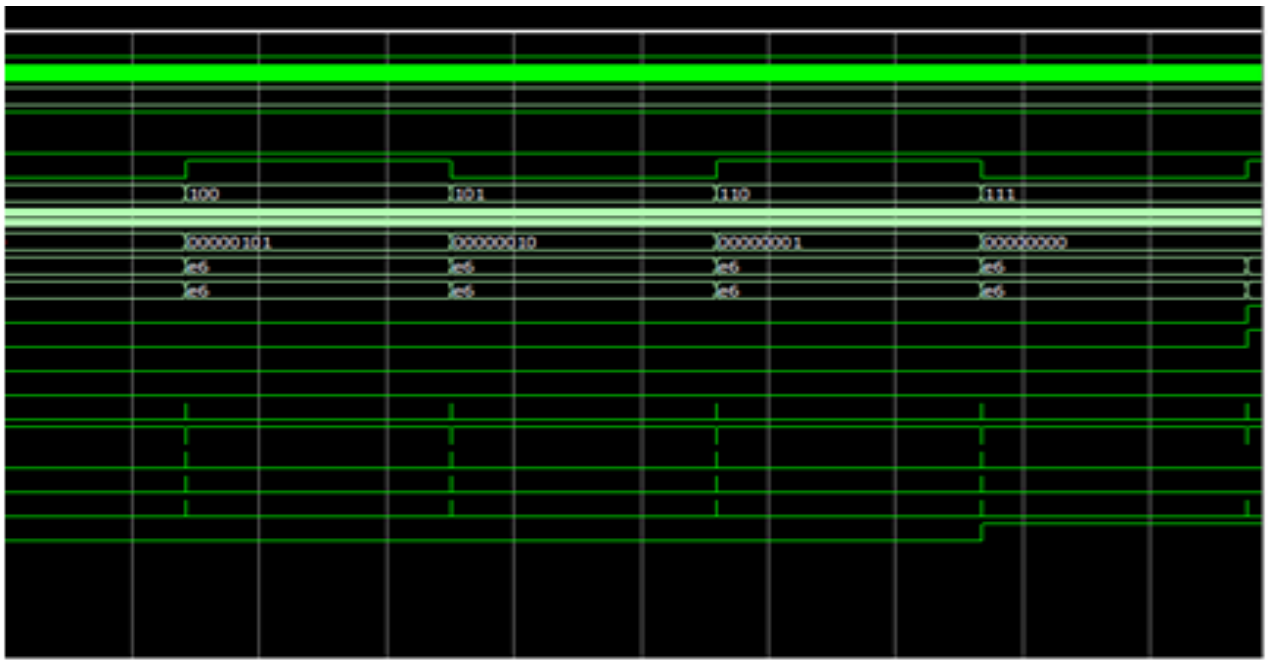
```
-------------------------------------
-- salidas de la unidad de control
cl_cbits <= '1' when(ep = e1  or ep=e7)else '0';
  cl_ctmp <= '1'  when(ep = e1 or ep=e3 or ep=e8)else '0';
  INC_ctmp <= '1'  when(ep = e2 or ep=e4 or ep=e9)else '0';
right<= '1'  when ep = e5 else '0';
  INC_cbits <= '1' when ep = e5 else '0';

--====================================
-- Unidad de Proceso
--====================================
-------------------------------
-- registro desplazamiento
process(clk, reset)
begin
if reset = '1' then
      Qrbyte <="00000000";
elsif clk'eventand clk='1' then
ifright= '1' then
        Qrbyte(6downto0)<= Qrbyte(7downto1);
        Qrbyte(7)<= rx_s;
endif;
endif;
endprocess;



-------------------------------
-- CBITS
process(clk, reset)
begin
if reset = '1' then
      Qcbits <="0000";
elsif clk'eventand clk='1' then
if cl_cbits = '1' then
          Qcbits <="0000";
elsif INC_cbits = '1' then
              Qcbits <= Qcbits+1;

endif;
endif;
endprocess;

-------------------------------
-- CTMP
process(clk, reset)
begin
if reset = '1' then
      Qctmp <="0000000000000";
elsif clk'eventand clk='1' then
if cl_ctmp = '1' then
          Qctmp <="0000000000000";
endif;
if(INC_ctmp='1' and Qctmp<"1010001011000")then
              Qctmp <= Qctmp+1;
endif;
endif;
endprocess;
-------------------------------------
-- D
process(clk)
```

```
begin
if clk'eventand clk='1' then
     rx_s <= rx;
endif;
endprocess;

 fintmp2 <= '1' when Qctmp="1010001011000"else '0';
 fintmp1 <= '1' when Qctmp="0101000101100"else '0';
 byte <= Qrbyte;
 rec <= '1' when ep = e6 else '0';
END a;
```

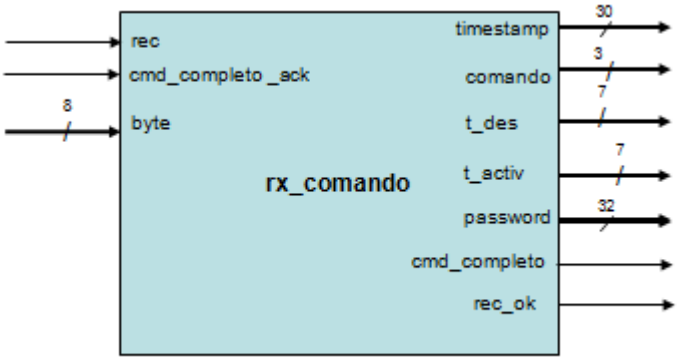
e . Simulación

Se puede observar en las siguientes imagenes de olas el comportamiento del programa tx_byte. Este programa se encarga de recibir 8 bits(1 byte) y enviar el byte entero. Junto a este informe tenemos tanto la unidad de procesamiento como la unidad de control. Con estas unidades se define el comportamiento de nuestro programa y mediante el programa model slim los hemos transformado a lenguage vhdl y con ello hemos simulado el funcionamiento. El resultado se puede observar en las siguentes imagenes(waves):





5.2. RX_COMANDO



a . Descripción

El propósito de este módulo es la construcción del comando entero enviado byte a byte por el modulo directamente conectado a este, el rx_byte. Una vez obtenido todos los bytes del comando enviado por la aplicación de Mirakonta, dependiendo del comando resultado, carga en las señales de salida los datos correspondientes.

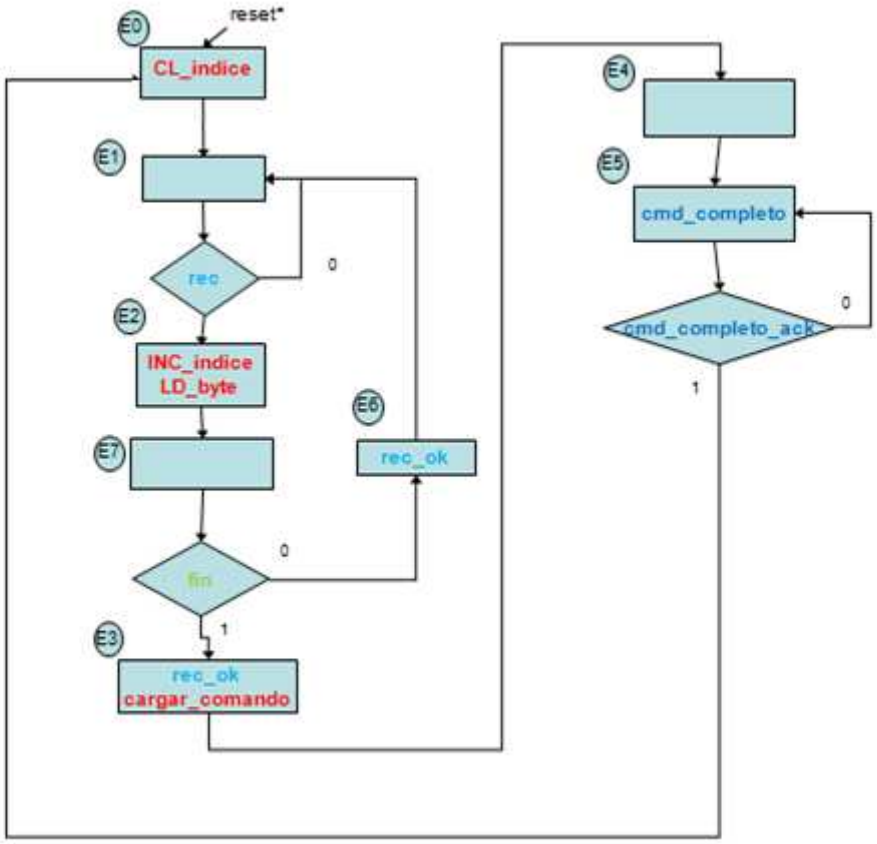
Señales de entrada:

- **rec**: Indica que ya está listo un byte.
- **cmd_completo_ack**: Indica que se ha obtenido bien el comando.
- **byte**: Byte a guardar.

Señales de salida:

- **timestamp**: 30 bits que indican el tiempo actual.
- **comando**: 3 bits que indican el comando generado.
- **t_des**: 7 bits que representan el tiempo de desactivación configurado.
- **t_activ**:7 bits que representan el tiempo de activación configurado.
- **password**: 32 bits que indican la contraseña enviada.
- **cmd_completo**:Indica que se ha generado el comando entero y está listo para su procesado.
- **rec_ok**:Indica que se ha recibido el bit correctamente.

b . Unidad de Control



Primeramente estamos en el estado E0, en el que resetearemos el índice que se utiliza para ir insertando los bytes que le llegan para construir el comando completo. De este estado pasaremos al E1 directamente.

En el estado E1 preguntaremos por la señal rec que vendrá del módulo rx_byte, la cual nos informará de que este último módulo habrá terminado de transmitir el byte. Cumplida esta condición, pasaremos al estado E2, sino seguiremos en el mismo estado E1.

En el siguiente estado, el E2, meteremos el byte proporcionado por el rx_byte en el lugar adecuado e incrementaremos el valor del índice para la siguiente inserción. De este estado se pasa directamente al estado E7 sin ningún tipo de condición.

En el estado E7 preguntaremos por la señal fin, que indica si el byte recibido es el retorno de carro o no (obsérvese que se ha insertado este último también). Cumpliéndose esta condición pasaríamos al estado E3, sino iríamos al estado E6.

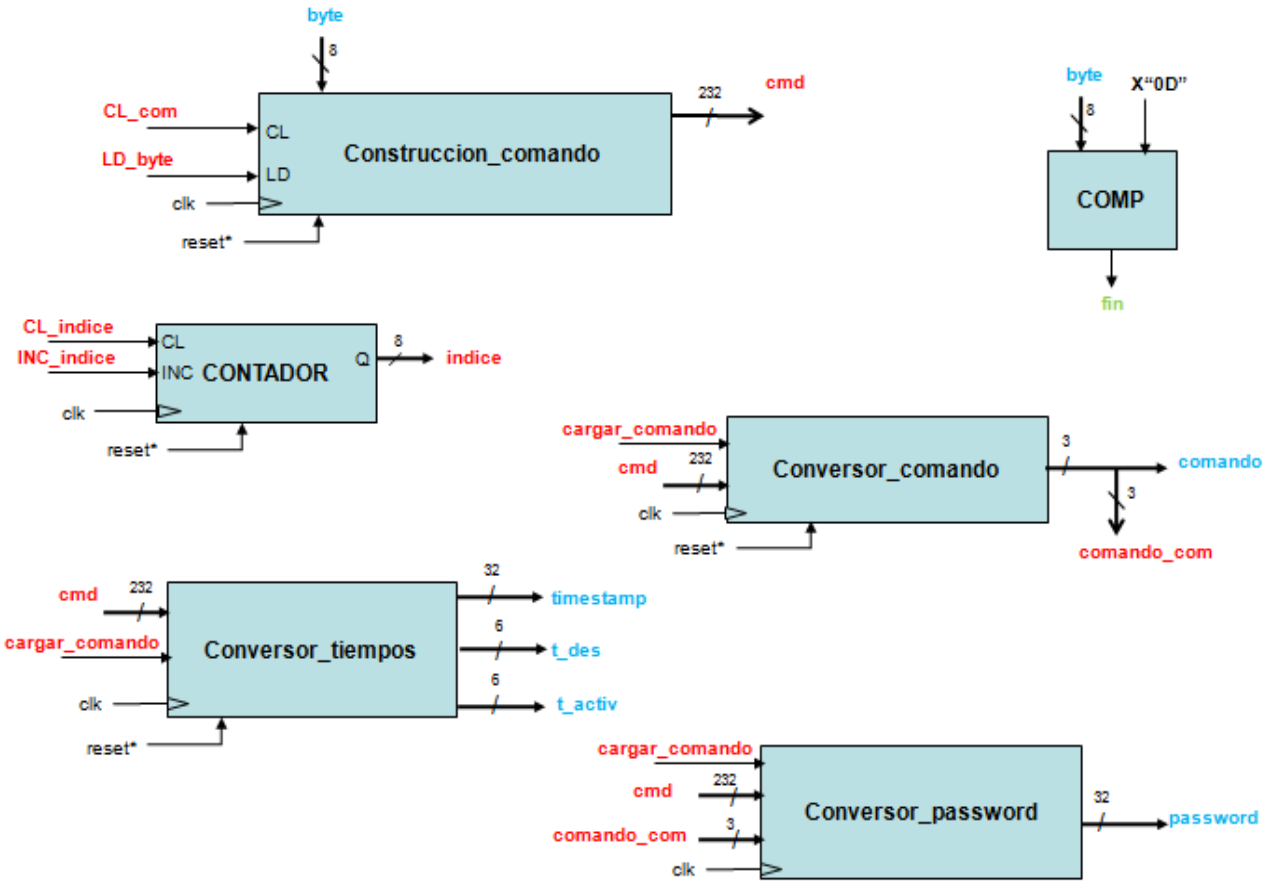
En el estado E6, se activaría la señal rec_ok para hacerle saber al módulo rx_byte que ya se ha terminado la recepción del byte enviado. Desde este estado pasamos directamente al estado E1 para esperar por un nuevo byte.

Una vez terminada la recepción del comando completo, estaremos en el estado E3, allí activaremos la señal rec_ok para hacerle saber al rx_byte la recepción del último byte (el retorno de carro) y activaremos la señal cargar_comando para extraer del comando completo la información útil como puede ser la contraseña o los tiempos. De esta estado iremos al estado E4 sin ninguna condición.

El estado E4 será un estado de espera para asegurarse la conversión de los tiempos obtenidos del comando entero. De este estado pasaremos directamente al estado E5.

En el último estado E5, activaremos la señal cmd_completo para que el gestor de los comandos sepa que ha llegado un comando completo, y que sus datos están listos para ser recogidos. Este último estado se quedará esperando a la respuesta del gestor_comando de que ha recibido correctamente el comando desbloqueando así este módulo para ir de nuevo al primer estado E0, donde se quedará esperando para la recepción de otro comando.

c . Unidad de Proceso



d . Código VHDL

```
--  Library Clause
LIBRARYIEEE;
LIBRARYwork;


--  Use Clause
USEIEEE.STD_LOGIC_1164.ALL;
USEIEEE.std_logic_arith.all;
USEIEEE.std_logic_unsigned.all;


ENTITY rx_comando IS
PORT(
    reset, clk               :instd_logic;
rec,cmd_completo_ack      :INSTD_LOGIC;

-- Entradas y salidas del modulo
    byte                        :INSTD_LOGIC_VECTOR(7DOWNTO0);
timestamp                    :OUTnaturalrange0to999999999;
    comando                     :OUTSTD_LOGIC_VECTOR(2DOWNTO0);
    t_activ,t_des               :OUTnaturalrange0to99;
    password                    :OUTSTD_LOGIC_VECTOR(31DOWNTO0);
    cmd_completo, rec_ok        :OUTSTD_LOGIC
);
END rx_comando;


ARCHITECTURE arq_rx_comando OF rx_comando IS

--Definicion de señales internas
type estado is(e0, e1, e2, e3, e4,e5,e6,e7);
signal ep, es :estado;
signal comando_comp :STD_LOGIC_VECTOR(2DOWNTO0);
signal password_aux              :STD_LOGIC_VECTOR(31DOWNTO0);

SIGNAL cmd   :STD_LOGIC_VECTOR(231DOWNTO0);

signal trozo1           :naturalrange0to9;
signal trozo2           :naturalrange0to9;
signal trozo3           :naturalrange0to9;
signal trozo4           :naturalrange0to9;
signal trozo5           :naturalrange0to9;
signal trozo6           :naturalrange0to9;
signal trozo7           :naturalrange0to9;
signal trozo8           :naturalrange0to9;
signal trozo9           :naturalrange0to9;
signal trozo1_t_activ :naturalrange0to9;
signal trozo2_t_activ :naturalrange0to9;
signal trozo1_t_des   :naturalrange0to9;
signal trozo2_t_des   :naturalrange0to9;

-- Señales de salida de UP a UC
signal igual :std_logic;
signal indice :natural;

-- Señales de salida de UC a UP
signal cargar_comando   :std_logic;
```

```
signal LD_byte          :std_logic;
signal INC_indice       :std_logic;
signal CL_indice        :std_logic;


BEGIN

    ------------------------------------------------------------------------
    ------
--UNIDAD DE CONTROL
------------------------------------------------------------------------

------------------------------
-- Transicion de estados
------------------------------
PROCESS(clk)-- Lista de sensibilidad
BEGIN
CASE ep IS
WHEN e0 =>
                  es<= e1;
WHEN e1 =>
if rec='1' then
es<=e2;
else
                  es<=e1;
endif;
when e2 =>
                es <= e7;
WHEN e7 =>
if igual='1' then
es <= e3;
else
                    es<=e6;
endif;
WHEN e3 =>
                es <= e4;
WHEN e4 =>
                es <= e5;
WHEN e5 =>
if cmd_completo_ack='1' then
es <= e0;
else
                  es <= e5;
endif;
WHEN e6 =>
                es <= e1;
ENDCASE;

ENDPROCESS;


-----------------------
--Registro de estados
-----------------------

PROCESS(clk,reset)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                ep <= e0;
ELSE
```

```
                ep <= es;
ENDIF;
ENDIF;
ENDPROCESS;


--------------------------------------------------------------------------
---
-- SEÑALES DE CONTROL
--------------------------------------------------------------------------

CL_indice <= '1'  WHEN ep = e0 ELSE '0';
LD_byte <= '1'     WHEN ep = e2 ELSE '0';
rec_ok  <= '1'      WHEN(ep = e6 or ep = e3)ELSE '0';
INC_indice <= '1'   WHEN ep = e2    ELSE '0';
cargar_comando <='1' WHEN ep = e5   ELSE '0';
cmd_completo <= '1'     WHEN ep = e5    ELSE '0';


--------------------------------------------------------------------------
------
--UNIDAD DE PROCESO
--------------------------------------------------------------------------

----------------------
-- Comparador
----------------------

igual <= '1' when byte = X"0D"else '0';

-------------------
-- Contador
-------------------

PROCESS(clk,reset,INC_indice)
BEGIN
IF reset = '1' THEN
            indice <=0;-- Ponemos el contador a Cero
elsif clk'EVENTAND clk='1' THEN
IF CL_indice ='1' then
                indice <=0;
ELSIF inc_indice = '1' THEN
             indice <= indice +8;
ENDIF;
ENDIF;
ENDPROCESS;


-------------------------
-- Construccion_comando
-------------------------
PROCESS(clk,reset)
BEGIN
IF reset = '1'
THENcmd<=X"0000000000000000000000000000000000000000000000000000000000000000"
;
elsif clk'EVENTAND clk='1' THEN
if LD_byte ='1' then
             cmd(231- indice downto224- indice)<= byte;
endif;
ENDIF;
```

```
ENDPROCESS;



-----------------------
-- conversor_comando
-----------------------
process(clk,reset)
begin
if  clk'EVENTand clk='1' then
if reset ='1' then
         comando_comp <="000";
elsif  cargar_comando = '1' then
case cmd(231downto192)is
when X"4D4B0D0000"=>
-- MK a
           comando_comp <="000";
when X"4D4B2B560D"=>
-- MK+V a
           comando_comp <="001";
when X"4D4B2B434F"=>
-- MK+CONF:<timestamp>,<tiempo de activacion>,< tiempo de
desactivacion >,<nuevo password>
comando_comp <="010";
when X"4D4B2B5041"=>
-- MK+PASS:<password>
           comando_comp <="011";
when X"4D4B2B4143"=>
-- MK+ACT
           comando_comp <="100";
when X"4D4B2B4426"=>
-- MK+D&H a
           comando_comp <="101";
when X"4D4B2B5354"=>
-- MK+STATE  a
           comando_comp <="110";
when X"4D4B2B4445"=>
-- MK+DEACT
           comando_comp <="111";
whenothers=>
          comando_comp <="000";
endcase;
endif;
endif;
endprocess;



---------------------
-- conversor_tiempos
---------------------
PROCESS(clk,reset)
BEGIN
IF clk'EVENTAND clk='1' THEN
if  cargar_comando = '1' then

---------------------------
--  Traduccion de timestamp
---------------------------
-- timestamp esta en ---> cmd ( 167 downto 96)
case cmd(167downto160)is
```

```
when X"30"=>
                    trozo1 <=0;
when X"31"=>
                    trozo1 <=1;
when X"32"=>
                    trozo1 <=2;
when X"33"=>
                    trozo1 <=3;
when X"34"=>
                    trozo1 <=4;
when X"35"=>
                    trozo1 <=5;
when X"36"=>
                    trozo1 <=6;
when X"37"=>
                    trozo1 <=7;
when X"38"=>
                    trozo1 <=8;
when X"39"=>
                    trozo1 <=9;
whenothers=>
                    trozo1 <=0;
endcase;

case cmd(159downto152)is
when X"30"=>
                    trozo2 <=0;
when X"31"=>
                    trozo2 <=1;
when X"32"=>
                    trozo2 <=2;
when X"33"=>
                    trozo2 <=3;
when X"34"=>
                    trozo2 <=4;
when X"35"=>
                    trozo2 <=5;
when X"36"=>
                    trozo2 <=6;
when X"37"=>
                    trozo2 <=7;
when X"38"=>
                    trozo2 <=8;
when X"39"=>
                    trozo2 <=9;
whenothers=>
                    trozo2 <=0;
endcase;

case cmd(151downto144)is
when X"30"=>
                    trozo3 <=0;
when X"31"=>
                    trozo3 <=1;
when X"32"=>
                    trozo3 <=2;
when X"33"=>
                    trozo3 <=3;
when X"34"=>
                    trozo3 <=4;
when X"35"=>
```

```
                    trozo3 <=5;
when X"36"=>
                    trozo3 <=6;
when X"37"=>
                    trozo3 <=7;
when X"38"=>
                    trozo3 <=8;
when X"39"=>
                    trozo3 <=9;
whenothers=>
                    trozo3 <=0;
endcase;

case cmd(143downto136)is
when X"30"=>
                    trozo4 <=0;
when X"31"=>
                    trozo4 <=1;
when X"32"=>
                    trozo4 <=2;
when X"33"=>
                    trozo4 <=3;
when X"34"=>
                    trozo4 <=4;
when X"35"=>
                    trozo4 <=5;
when X"36"=>
                    trozo4 <=6;
when X"37"=>
                    trozo4 <=7;
when X"38"=>
                    trozo4 <=8;
when X"39"=>
                    trozo4 <=9;
whenothers=>
                    trozo4 <=0;
endcase;

case cmd(135downto128)is
when X"30"=>
                    trozo5 <=0;
when X"31"=>
                    trozo5 <=1;
when X"32"=>
                    trozo5 <=2;
when X"33"=>
                    trozo5 <=3;
when X"34"=>
                    trozo5 <=4;
when X"35"=>
                    trozo5 <=5;
when X"36"=>
                    trozo5 <=6;
when X"37"=>
                    trozo5 <=7;
when X"38"=>
                    trozo5 <=8;
when X"39"=>
                    trozo5 <=9;
whenothers=>
                    trozo5 <=0;
```

```
endcase;

case cmd(127downto120)is
when X"30"=>
                    trozo6 <=0;
when X"31"=>
                    trozo6 <=1;
when X"32"=>
                    trozo6 <=2;
when X"33"=>
                    trozo6 <=3;
when X"34"=>
                    trozo6 <=4;
when X"35"=>
                    trozo6 <=5;
when X"36"=>
                    trozo6 <=6;
when X"37"=>
                    trozo6 <=7;
when X"38"=>
                    trozo6 <=8;
when X"39"=>
                    trozo6 <=9;
whenothers=>
                    trozo6 <=0;
endcase;

case cmd(119downto112)is
when X"30"=>
                    trozo7 <=0;
when X"31"=>
                    trozo7 <=1;
when X"32"=>
                    trozo7 <=2;
when X"33"=>
                    trozo7 <=3;
when X"34"=>
                    trozo7 <=4;
when X"35"=>
                    trozo7 <=5;
when X"36"=>
                    trozo7 <=6;
when X"37"=>
                    trozo7 <=7;
when X"38"=>
                    trozo7 <=8;
when X"39"=>
                    trozo7 <=9;
whenothers=>
                    trozo7 <=0;
endcase;

case cmd(111downto104)is
when X"30"=>
                    trozo8 <=0;
when X"31"=>
                    trozo8 <=1;
when X"32"=>
                    trozo8 <=2;
when X"33"=>
                    trozo8 <=3;
```

```
when X"34"=>
                    trozo8 <=4;
when X"35"=>
                    trozo8 <=5;
when X"36"=>
                    trozo8 <=6;
when X"37"=>
                    trozo8 <=7;
when X"38"=>
                    trozo8 <=8;
when X"39"=>
                    trozo8 <=9;
whenothers=>
                    trozo8 <=0;
endcase;

case cmd(103downto96)is
when X"30"=>
                    trozo9 <=0;
when X"31"=>
                    trozo9 <=1;
when X"32"=>
                    trozo9 <=2;
when X"33"=>
                    trozo9 <=3;
when X"34"=>
                    trozo9 <=4;
when X"35"=>
                    trozo9 <=5;
when X"36"=>
                    trozo9 <=6;
when X"37"=>
                    trozo9 <=7;
when X"38"=>
                    trozo9 <=8;
when X"39"=>
                    trozo9 <=9;
whenothers=>
trozo9 <=0;
endcase;

--------------------------
--  Traduccion de t_activ
--------------------------
-- t_activ esta en ---> cmd ( 87 downto 80)
case cmd(87downto80)is
when X"30"=>
                    trozo1_t_activ <=0;
when X"31"=>
                    trozo1_t_activ <=1;
when X"32"=>
                    trozo1_t_activ <=2;
when X"33"=>
                    trozo1_t_activ <=3;
when X"34"=>
                    trozo1_t_activ <=4;
when X"35"=>
                    trozo1_t_activ <=5;
when X"36"=>
                    trozo1_t_activ <=6;
when X"37"=>
```

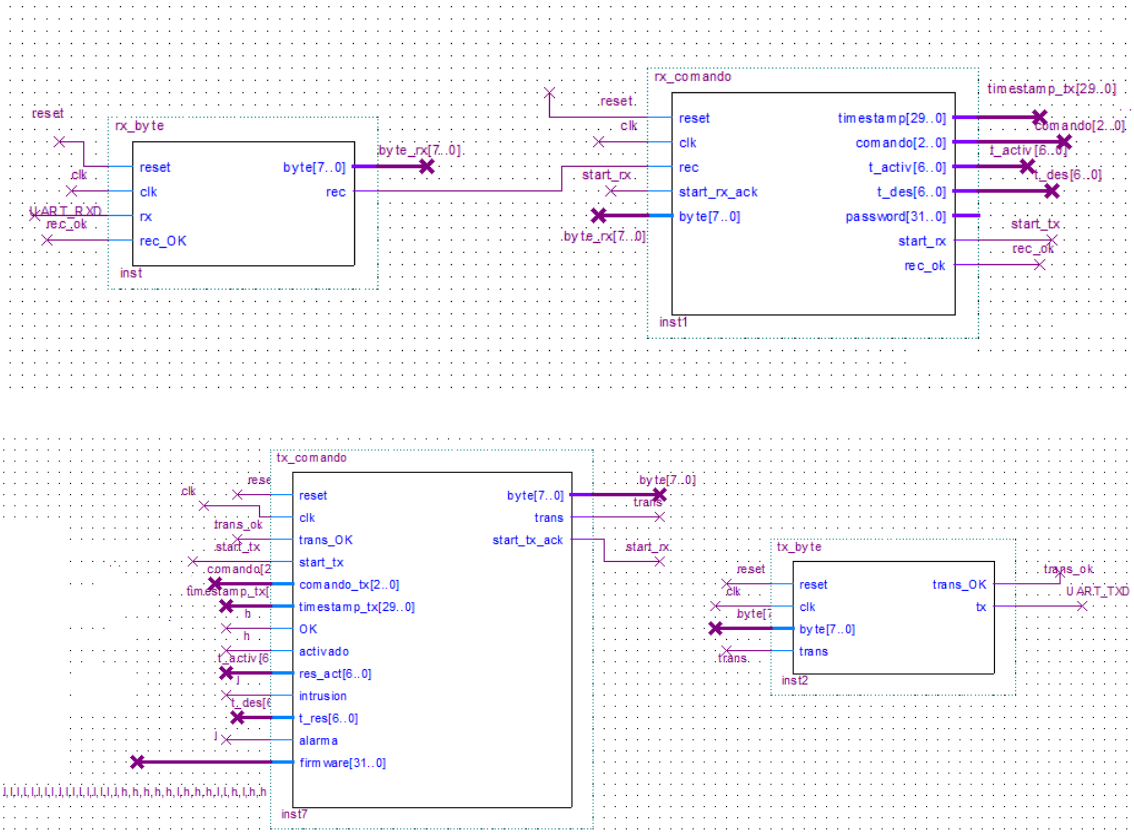
```
                    trozo1_t_activ <=7;
when X"38"=>
                    trozo1_t_activ <=8;
when X"39"=>
                    trozo1_t_activ <=9;
whenothers=>
                    trozo1_t_activ <=0;
endcase;

case cmd(79downto72)is
when X"30"=>
                    trozo2_t_activ <=0;
when X"31"=>
                    trozo2_t_activ <=1;
when X"32"=>
                    trozo2_t_activ <=2;
when X"33"=>
                    trozo2_t_activ <=3;
when X"34"=>
                    trozo2_t_activ <=4;
when X"35"=>
                    trozo2_t_activ <=5;
when X"36"=>
                    trozo2_t_activ <=6;
when X"37"=>
                    trozo2_t_activ <=7;
when X"38"=>
                    trozo2_t_activ <=8;
when X"39"=>
                    trozo2_t_activ <=9;
whenothers=>
                    trozo2_t_activ <=0;
endcase;


-------------------------
--  Traduccion de t_des
-------------------------
-- t_des esta en ---> cmd ( 63 downto 48)
case cmd(63downto56)is
when X"30"=>
                    trozo1_t_des <=0;
when X"31"=>
                    trozo1_t_des <=1;
when X"32"=>
                    trozo1_t_des <=2;
when X"33"=>
                    trozo1_t_des <=3;
when X"34"=>
                    trozo1_t_des <=4;
when X"35"=>
                    trozo1_t_des <=5;
when X"36"=>
                    trozo1_t_des <=6;
when X"37"=>
                    trozo1_t_des <=7;
when X"38"=>
                    trozo1_t_des <=8;
when X"39"=>
                    trozo1_t_des <=9;
whenothers=>
```

```
                    trozo1_t_des <=0;
endcase;

case cmd(55downto48)is
when X"30"=>
                    trozo2_t_des <=0;
when X"31"=>
                    trozo2_t_des <=1;
when X"32"=>
                    trozo2_t_des <=2;
when X"33"=>
                    trozo2_t_des <=3;
when X"34"=>
                    trozo2_t_des <=4;
when X"35"=>
                    trozo2_t_des <=5;
when X"36"=>
                    trozo2_t_des <=6;
when X"37"=>
                    trozo2_t_des <=7;
when X"38"=>
                    trozo2_t_des <=8;
when X"39"=>
                    trozo2_t_des <=9;
whenothers=>
                    trozo2_t_des <=0;
endcase;
endif;
ENDIF;

timestamp <= trozo1*100000000+ trozo2*10000000+ trozo3*1000000+
trozo4*100000+ trozo5*10000+ trozo6*1000+ trozo7*100+ trozo8*10+
trozo9;
t_des <= trozo1_t_des*10+ trozo2_t_des;
        t_activ <= trozo1_t_activ*10+ trozo2_t_activ;
ENDPROCESS;

----------------------
-- convertir_password
----------------------
PROCESS(clk,reset,cargar_comando)
BEGIN
if reset = '1' then
--password <= "00000000000000000000000000000000";
           password_aux <= X"33333333";
elsif clk'EVENTAND clk='1' THEN
if cargar_comando ='1' and comando_comp ="010"then
-- MK+CONF
                    password_aux <= cmd(39downto8);
elsif cargar_comando ='1' and comando_comp ="011"then
-- MK+PASS
                    password_aux <= cmd(167downto136);
else
                    password_aux <= password_aux;
endif;
ENDIF;
ENDPROCESS;

comando <= comando_comp;
password <= password_aux;
END arq_rx_comando;
```

e . Simulación

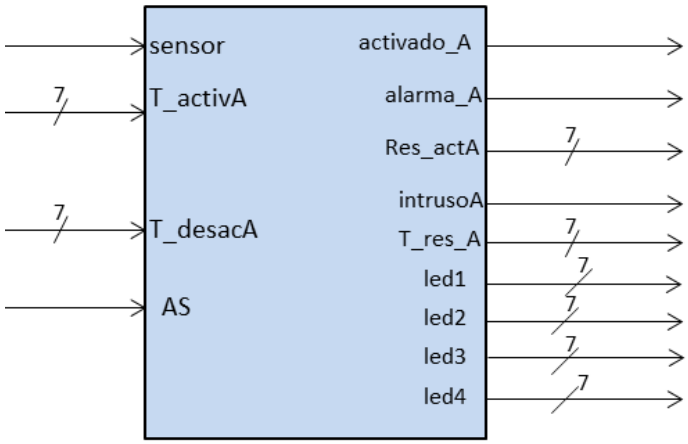
La comprobación de este módulo se basó en varias pruebas realizadas con el programa Quartus y la propia placa DE2 de Altera.

Para está realización de las pruebas, conectamos los modulos rx_byte, rx_comando, tx_comando y tx_byte. He aquí el diseño creado para testear el rx_comando en la herramienta Quartus.



A continuación, conectamos la entrada de bytes del módulo rx_byte con la entrada del puerto RS232 de la placa mediante la asignación UART_RXD y en el otro extremo conectamos la salida de cada byte del módulo tx_byte a salida del puerto RS232 mediante la asignación de UART_TXD. Realizado esto, enviamos mediante la aplicación de Mirakonta comandos y pudimos comprobar la recepción de los mismos en el SERIAL DATA de la misma aplicación.

5.3. GESTOR_ALARMA



a . Descripción

Este módulo es la parte del gestor que se encarga de modificar los estados y activar tanto las alarmas como los leds que muestran el tiempo de desactivación y activación en la placa. El gestor_alarma trabaja con la información que el gestor_comando le envía y este le devuelve al gestor_comando la información de tiempos restantes para activación y desactivación y el estado de la alarma, si esta activada o no, y si ha saltado o no.

Señales de entrada:

- **Sensor:** Indica si el sensor ha detectado algún intruso.
- **T_activA:** Se recibe el tiempo que se necesita para que se active la alarma.
- **T_desacA:** Se recibe el tiempo que se necesita para que la alarma se desactive.
- **AS:** Señal que indica que el sistema esta encendido.

Señales de salidas:

- **activado_a:** Indica si la alarma esta activada.
- **alarma_a:** Indica si la alarma a saltado o no.
- **res_ActA:** Tiempo restante para que se active la alarma una vez encendida.
- **intrusoA:** Si el sensor ha detectado intrusión una vez la alarma está en espera.
- **T_res_A:** Tiempo restante para que la alarma salte.
- **Led1:** Led que muestra el 1 digito del tiempo restante para activación de alarma.
- **Led2:** Led que muestra el 2 digito del tiempo restante para activación de alarma.
- **Led3:** Led que muestra el 1 digito del tiempo restante para que salte la alarma.
- **Led4:** Led que muestra el 2 digito del tiempo restante para que salte la alarma.

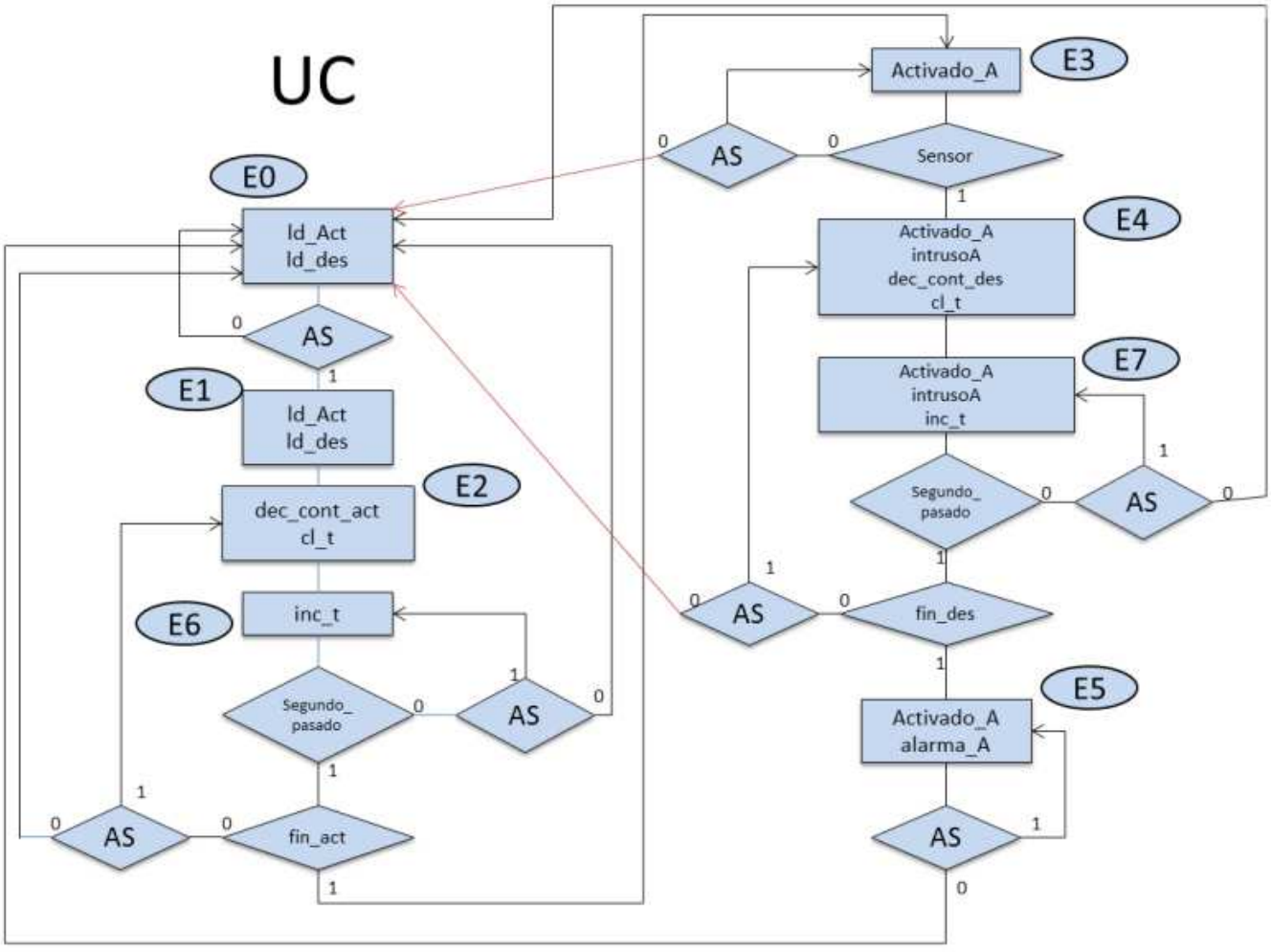
b . Unidad de Control

Al principio, nos encontramos en el estado E0. En este estado se cargan los tiempos de activación y desactivación que el gestor comando nos da. Este estado está limitado por la señal AS, el cual indica que el sistema se ha activado. Cuando esta señal se activa se pasa al estado E1.

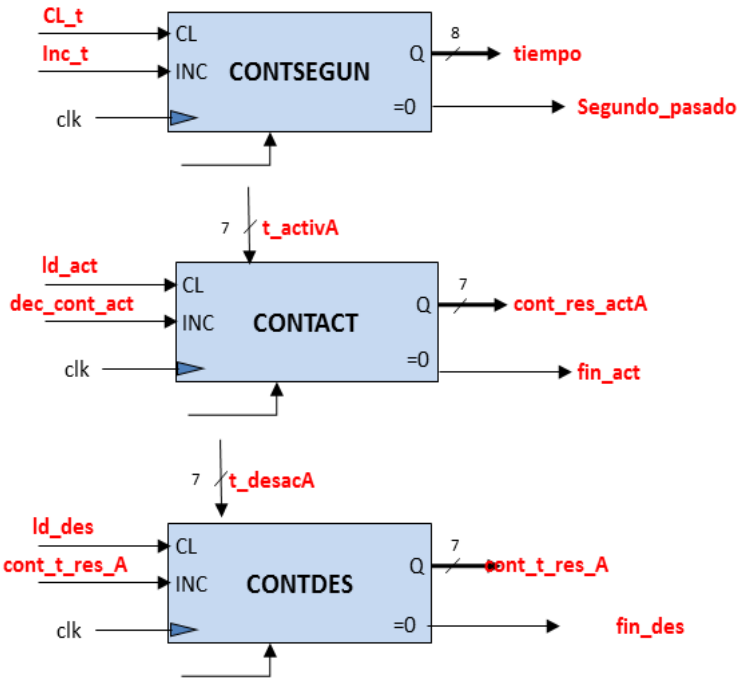
E1 es un estado de transición en el que se activan las mismas señales que en E0 para asegurarnos de que se cargan los tiempos. Desde este estado pasaremos al E2 incondicionalmente.

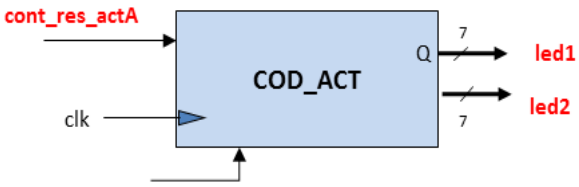
En los estados E2 y E6 se usan para controlar el tiempo. Mediante el estado E6 contamos que pase un segundo y cuando pasa se reduce el tiempo de activación. Cuando el tiempo de activación llega a 0 pasamos al estado E3, en el cual se activa la alarma.

Desde el estado E3 para pasar al estado E4 se espera a que el sensor detecte un intruso. En el estado E4 se mantiene la alarma activada y la señal de intruso, además de que comenzamos a contar el tiempo de desactivación mediante los estados E4 y E7 usando un sistema similar a los de los estados E2 y E6.Cuando el tiempo de desactivación llega a 0 se activa la señal de que la alarma a saltado y se mantiene la señal de alarma activada. Esto permanece así hasta que se recibe la señal de que AS es 0 con la cual volveríamos al estado E0. Esta señal si se recibe en cualquier lugar del algoritmo volvería al estado E0.

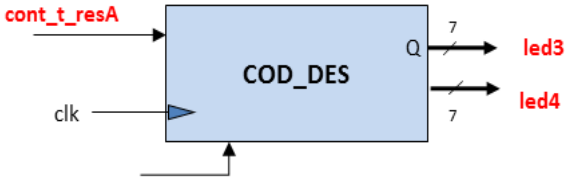


c . Unidad de Proceso





Este modulo traduce el tiempo restante para activación a un formato para que los leds lo interpreten y se muestren como números.



Este modulo traduce el tiempo restante para desactivación a un formato para que los leds lo interpreten y se muestren como números.

d . Código VHDL

```
--  Library Clause
LIBRARYIEEE;
LIBRARYwork;

--  Use Clause
USEIEEE.STD_LOGIC_1164.ALL;
USEIEEE.std_logic_arith.all;
USEIEEE.std_logic_unsigned.all;


ENTITY Gestor_alarma IS
PORT(
    reset, clk     :instd_logic;
sensor,AS       :INSTD_LOGIC;-- Entradas y salidas del modulo
t_activA,t_desacA               :INSTD_LOGIC_VECTOR(6DOWNTO0);
    res_actA,t_res_A                :OUTSTD_LOGIC_VECTOR(6DOWNTO0);
    led1,led2,led3,led4          :OUTSTD_LOGIC_VECTOR(6DOWNTO0);
    activado_A,alarma_A,intrusoA                :OUTSTD_LOGIC

);
END Gestor_alarma;

ARCHITECTURE arq_Gestor_alarma OF Gestor_alarma IS


--Definicion de señales internas
type estado is(e0, e1, e2, e3, e4, e5, e6, e7);
signal ep, es                               :estado;

SIGNAL fin_act,fin_des,segundo_pasado               :STD_LOGIC;
SIGNAL ld_act,ld_des,dec_cont_act,dec_cont_des,inc_t,cl_t
:STD_LOGIC;

SIGNAL  tiempo:naturalrange0to50000000;
SIGNAL cont_res_actA,cont_t_res_A
:STD_LOGIC_VECTOR(6DOWNTO0);
SIGNAL  tiempoactiv:naturalrange0to99;
SIGNAL  decimalact:naturalrange0to99;
SIGNAL  digitoact:naturalrange0to9;
SIGNAL  tiempodes:naturalrange0to99;
SIGNAL  decimaldes:naturalrange0to99;
SIGNAL  digitodes:naturalrange0to9;


------------------------------------------------------------------------
--
--UNIDAD DE CONTROL
------------------------------------------------------------------------
--

BEGIN-- transicion de estados

PROCESS(ep,AS,fin_act,fin_des,sensor)-- Lista de sensibilidad
BEGIN
CASE ep IS
WHEN e0 =>
if AS='1' then          es <= e1;
else                                es <= e0;
endif;
```

```
WHEN e1 => es<= e2;
WHEN e2 => es<= e6;
WHEN e6 =>
if segundo_pasado='1' then

if fin_act='1' then         es <= e3;
elsif   AS='1' then es<=e2;
else es<=e0;
endif;

elsif AS='1' then es<= e6;
else es<=e0;
endif;
WHEN e3 =>
if sensor='1' then          es <= e4;
elsif AS='1' then es <= e3;
else es<=e0;
endif;
WHEN e4 => es<= e7;
WHEN e7 =>

if segundo_pasado='1' then
if fin_des='1' then es <= e5;
elsif AS='1' then es <= e4;
else es<=e0;
endif;
elsif AS='1' then es<=e7;
else es<=e0;
endif;
WHEN e5 =>
if AS='1' then          es <= e5;
else    es <= e0;
endif;

ENDCASE;

ENDPROCESS;

-------------------------------------------------------------------------
--
--Registro de estados
-----------------------

PROCESS(clk,reset)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                ep <= e0;
ELSE
                ep <= es;
ENDIF;
ENDIF;
ENDPROCESS;
-------------------------------------------------------------------------
---
-------------------------------------------------------------------------
---
-- SEÑALES DE CONTROL
-------------------------------------------------------------------------
---
```

```
ld_act          <= '1' WHEN(ep = e0 or ep = e1)
ELSE '0';
ld_des          <= '1' WHEN(ep = e0 or ep = e1)
ELSE '0';
dec_cont_act            <= '1' WHEN ep = e2
ELSE '0';
activado_A          <= '1' WHEN(ep = e3 or ep = e4 or ep = e5 or ep=
e7)
ELSE '0';
intrusoA            <= '1' WHEN(ep = e4 or ep= e7)
ELSE '0';
dec_cont_des            <= '1' WHEN ep = e4
ELSE '0';
alarma_A            <= '1' WHEN ep = e5
ELSE '0';
cl_t            <= '1' WHEN(ep=e2 or ep=e4)ELSE '0';

inc_t               <=  '1' WHEN(ep=e6 or ep=e7)ELSE '0';




-------------------------------------------------------------------------
-------
-------------------------------------------------------------------------
-------

-------------------------------------------------------------------------
------
--UNIDAD DE PROCESO
-------------------------------------------------------------------------
------


-- CONTADORES, registros, etc


PROCESS(clk,reset,cl_t,inc_t)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                tiempo <=50000000;-- Ponemos el contador a Cero
ELSIF(cl_t = '1')THEN
                tiempo <=50000000;-- cargamos contador
ELSIF(inc_t = '1')THEN
if(tiempo >0)THEN
tiempo <= tiempo -1;-- decrementamos si el valor es mayor a 0
endif;
ENDIF;
ENDIF;
ENDPROCESS;

segundo_pasado <= '1' WHEN(tiempo =0)else '0';


-------------------
-- CONTADORES
-------------------
```

```
PROCESS(clk,reset,ld_act,dec_cont_act)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                cont_res_actA <="0000000";-- Ponemos el contador a
Cero
ELSIF(ld_act = '1')THEN
                cont_res_actA <= t_activA;-- cargamos contador
ELSIF(dec_cont_act = '1')THEN
if(cont_res_actA >"0000000")THEN
                cont_res_actA <= cont_res_actA -1;-- decrementamos si
el valor es mayor a 0
endif;
ENDIF;
ENDIF;
ENDPROCESS;

    fin_act <= '1' WHEN(cont_res_actA ="0000000")else '0';

res_ActA <= cont_res_actA;

PROCESS(clk,reset,ld_act,dec_cont_act)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                cont_t_res_A <="0000000";-- Ponemos el contador a Cero
ELSIF(ld_des = '1')THEN
                cont_t_res_A <= t_desacA;-- cargamos contador
ELSIF(dec_cont_des = '1')THEN
if(cont_t_res_A >"0000000")THEN
cont_t_res_A <= cont_t_res_A -1;-- decrementamos si el valor es mayor
a 0
endif;
ENDIF;
ENDIF;
ENDPROCESS;

  fin_des <= '1' WHEN(cont_t_res_A ="0000000")else '0';
    t_res_A <= cont_t_res_A;


------------------------------------------------------------------------
-------------------------
------------------------------------------------------------------------
-------------------------
-----------------------
-- codificador para pasar la señal al led1(decimal)

PROCESS(clk,reset,cont_res_actA)

BEGIN
      tiempoactiv <= CONV_INTEGER(cont_res_ActA);
      decimalact <= tiempoactiv - tiempoactiv rem10;
case(decimalact)is
when0=> led1 <="1000000";
when10=> led1 <="1111001";
when20=> led1 <="0100100";
when30=> led1 <="0110000";
when40=> led1 <="0011001";
when50=> led1 <="0010010";
```

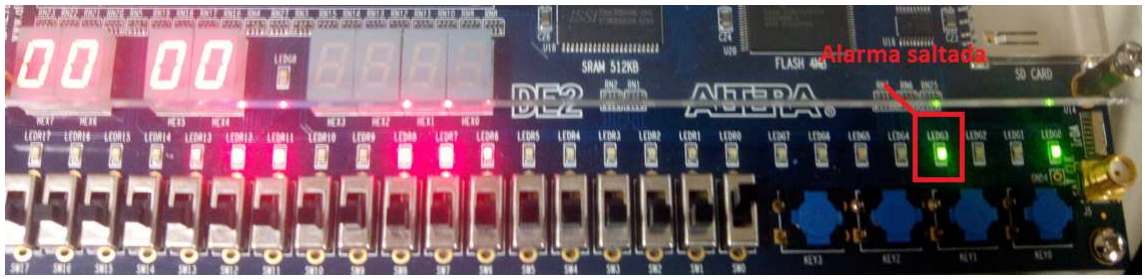
```
when60=> led1 <="0000010";
when70=> led1 <="1111000";
when80=> led1 <="0000000";
when90=> led1 <="0011000";
whenothers=> led1 <="0110110";
endcase;
-- codificador para pasar la señal al led2(decimal)
      digitoact <= tiempoactiv rem10;--rem es el restante del modulo
10
case(digitoact)is
when0=> led2 <="1000000";
when1=> led2 <="1111001";
when2=> led2 <="0100100";
when3=> led2 <="0110000";
when4=> led2 <="0011001";
when5=> led2 <="0010010";
when6=> led2 <="0000010";
when7=> led2 <="1111000";
when8=> led2 <="0000000";
when9=> led2 <="0011000";
endcase;
ENDPROCESS;

PROCESS(clk,reset,cont_t_res_A)

BEGIN
      tiempodes <= CONV_INTEGER(cont_t_res_A);
      decimaldes <= tiempodes - tiempodes rem10;
-- codificador para pasar la señal al led3(decimal)
case(decimaldes)is
when0=> led3 <="1000000";
when10=> led3 <="1111001";
when20=> led3 <="0100100";
when30=> led3 <="0110000";
when40=> led3 <="0011001";
when50=> led3 <="0010010";
when60=> led3 <="0000010";
when70=> led3 <="1111000";
when80=> led3 <="0000000";
when90=> led3 <="0011000";
whenothers=> led3 <="0110110";
endcase;
digitodes <= tiempodes rem10;--rem es el restante del modulo 10
-- codificador para pasar la señal al led4(decimal)
case(digitodes)is
when0=> led4 <="1000000";
when1=> led4 <="1111001";
when2=> led4 <="0100100";
when3=> led4 <="0110000";
when4=> led4 <="0011001";
when5=> led4 <="0010010";
when6=> led4 <="0000010";
when7=> led4 <="1111000";
when8=> led4 <="0000000";
when9=> led4 <="0011000";
endcase;
ENDPROCESS;



END arq_Gestor_alarma;
```

e . Simulación

La comprobación de este módulo se basó en varias pruebas realizadas con el programa ModelSim, Quartus y la propia placa DE2 de Altera.

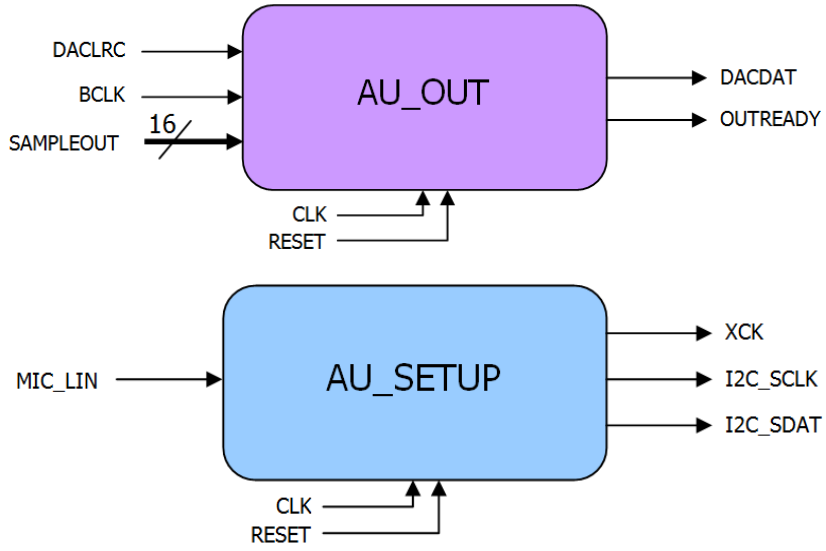
Para está realización de las pruebas para empezar mediante ModelSim creamos un wave nuevo con todas las señales y probamos que su funcionamiento parece correcto.

Después llega la parte de Quartus en la cual conectamos el modulo con diferentes interruptores de la placa. En las próximas imágenes se verán las imágenes asignadas y como en diferentes estados ciertas luces se encienden y apagan.

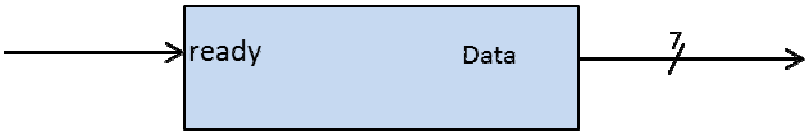


5.4. AUD_CONT, AU_SETUP Y AU_OUT

Antes de comenzar con estos tres módulos, hemos de destacar que los profesores nos han dado 2 de ellos, el au_out y au_setup, los cuales se encargan de mandar la señal al codec de la placa que genera el sonido. Estos módulos no los explicare mucho ya que nosotros no los hemos creado pero adjuntamos la documentación sobre ellos junto a este documento.



Ahora pasaremos a analizar el modulo realizado por nosotros, aud_cont.



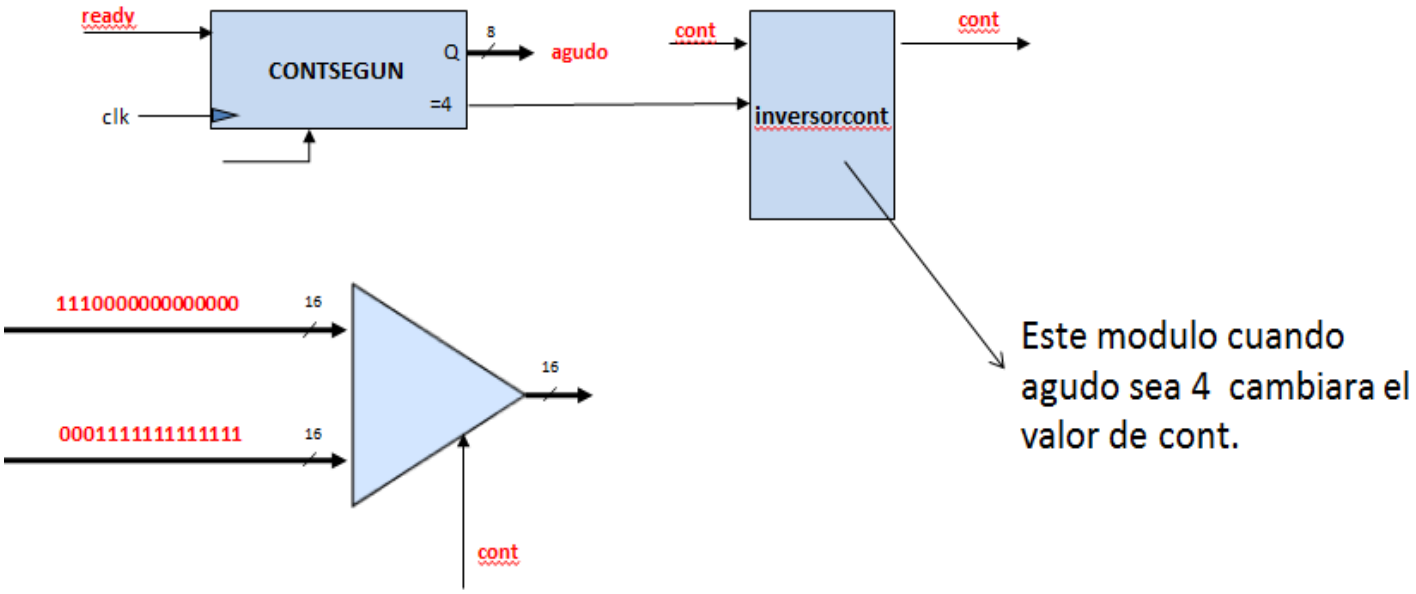
a . Descripción

Este módulo una vez recibe la señal de que la alarma ha saltado del gestor alarma (la señal ready) se encarga de generar una señal (sonido digital) que luego pasa al au_out. Esta señal es la representación de un pitido en formato analógico.

b . Unidad de Control

Este módulo es tan sencillo que carece de unidad de control.

c . Unidad de Proceso



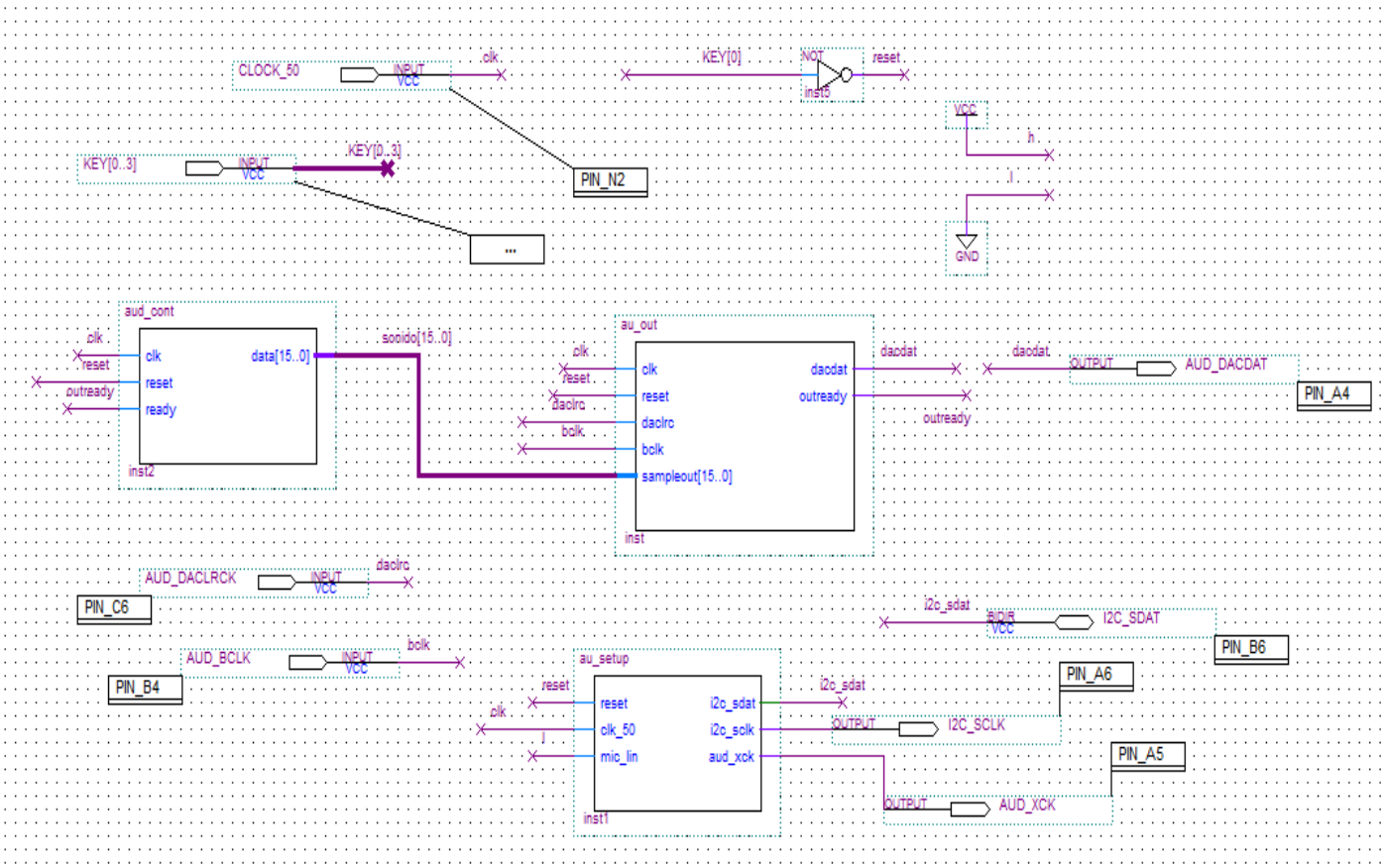
d . Código VHDL

```
LIBRARYieee;
USEieee.std_logic_1164.ALL;

ENTITY aud_cont IS
PORT(
        clk, reset      :INSTD_LOGIC;
        ready           :INSTD_LOGIC;
        alarma_activada         :INSTD_LOGIC;
        data        :OUTSTD_LOGIC_VECTOR(15DOWNTO0)
);
END aud_cont;

ARCHITECTURE a OF aud_cont IS
-- estados presente y siguiente de UC

-- registro de desplazamiento

--SIGNAL contCanal  : natural range 0 to 320;
--SIGNAL contBCLK   : natural range 0 to 65000;
SIGNAL cont         :STD_LOGIC;
SIGNAL agudo :naturalrange0to50;

BEGIN

-- contador de CANAl ------------------------
PROCESS(clk, reset)
BEGIN
IF(reset='1')THEN
            cont    <= '0';
            agudo<=0;
ELSIF(clk'EVENTAND clk='1')THEN
IF(ready='1' AND alarma_activada='1')THEN
agudo <= agudo+1;
IF(agudo=3)THEN--cambiando el valor de este numero cambiamos la
frecuencia de la señal(mas agudo mas bajo)
cont <=not cont;
                        agudo<=0;
ENDIF;
ENDIF;
ENDIF;
ENDPROCESS;

data    <="1110000000000000"WHEN cont ='1'  ELSE"0001111111111111";--
cambiando los valores cambiamos la amplitud de la señal (en
complemento A6)


END a;
```

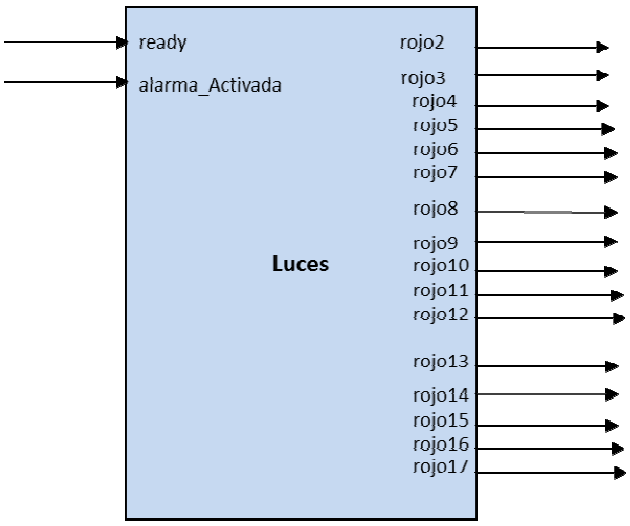
e . Simulación

La comprobación de este módulo se basó en la prueba realizada mediante Quartus. Para comprobarlo primero se probó el modulo solo mediante Quartus con la única asignación de la señal ready a un interruptor y cuando este se activaba mediante unos altavoces o cascos se comprobó que se generaba un pitido.

También comprobamos el modulo juntándolo con el módulo de gestor_alarma para ver que cuando el sensor detecta un intruso y pasa el tiempo de desactivación la alarma suena.



5.5. LUCES



a . Descripción

Este módulo se encarga de que cuando el gestor_alarma pasa a estado alarma saltada haga unas luces que iluminan la placa y alertan de que el intruso ha sido detectado y no se ha desactivado la alarma.

Señales de Entrada:

- **ready:** Indica si el sensor ha detectado algún intruso.
- **alarma_activada:** Indica si el sensor ha detectado algún intruso.

Señales de Salida:

- **Rojo[2-17]:** Señales para indicar si los leds rojos de la placa del 2 al 17 tienen que encenderse.

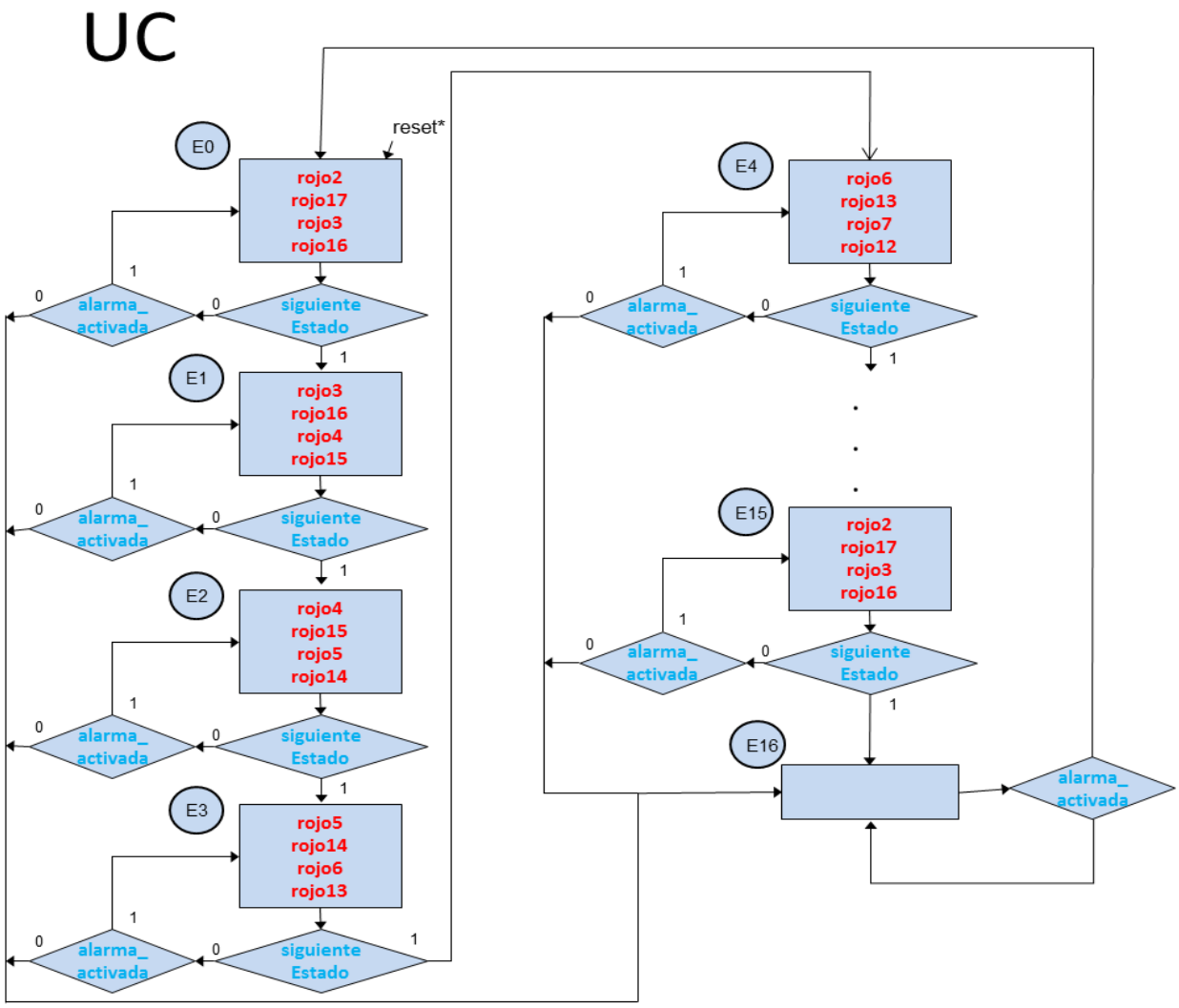
b . Unidad de Control

Este algoritmo se basa en una serie de estados mediante los cuales se activan ciertas luces. Cada estado representa una serie de luces. Lo que el algoritmo nos señala es que cuando le llega la señal de cambiar estado, mientras la alarma siga activada, se pasara al siguiente estado, en el cual se encenderán unas luces diferentes. El tiempo para cambiar al estado siguiente viene dado por la unidad de proceso mediante un contador.

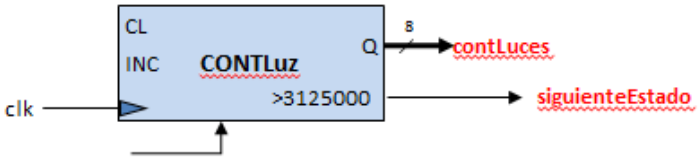
Cuando la alarma se desactiva en cualquier momento pasamos al estado E16 en el cual no se activa la señal alguna, digamos que este estado es cuando la alarma no ha saltado. Y cuando la alarma se activa pasara al estado E0. Desde cualquier estado si se desactiva la alarma pasaremos al estado E16 directamente.

Ya que el algoritmo se repite del estado 0 al 15 pero cambian las señales que se mandan hemos decidido poner las señales que se encienden a continuación:

E0: rojo2, rojo17, rojo3, rojo16
E1: rojo3, rojo16, rojo4,rojo15
E2: rojo4,rojo15,rojo5,rojo14
E3: rojo5,rojo14,rojo6,rojo13
E4: rojo6,rojo13,rojo7,rojo12
E5: rojo7,rojo12,rojo8,rojo11
E6: rojo8,rojo11,rojo9,rojo10
E7: rojo9, rojo10
E8: rojo9,rojo10
E9: rojo8,rojo11,rojo9,rojo10
E10: rojo7,rojo12,rojo8,rojo11
E11: rojo6,rojo13,rojo7,rojo12
E12: rojo5,rojo14,rojo6,rojo13
E13: rojo4,rojo15,rojo5,rojo14
E14: rojo3, rojo16, rojo4,rojo15
E15: rojo2, rojo17, rojo3, rojo16



c . Unidad de Proceso



d . Código VHDL

```
--  Library Clause
LIBRARYIEEE;
LIBRARYwork;


--  Use Clause
USEIEEE.STD_LOGIC_1164.ALL;
USEIEEE.std_logic_arith.all;
USEIEEE.std_logic_unsigned.all;


ENTITY luces IS
PORT(
        clk, reset      :INSTD_LOGIC;
        ready           :INSTD_LOGIC;
        alarma_activada         :INSTD_LOGIC;
        rojo2       :OUTSTD_LOGIC;
        rojo3       :OUTSTD_LOGIC;
        rojo4       :OUTSTD_LOGIC;
        rojo5       :OUTSTD_LOGIC;
        rojo6       :OUTSTD_LOGIC;
        rojo7       :OUTSTD_LOGIC;
        rojo8       :OUTSTD_LOGIC;
        rojo9       :OUTSTD_LOGIC;
        rojo10      :OUTSTD_LOGIC;
        rojo11      :OUTSTD_LOGIC;
        rojo12      :OUTSTD_LOGIC;
        rojo13      :OUTSTD_LOGIC;
        rojo14      :OUTSTD_LOGIC;
        rojo15      :OUTSTD_LOGIC;
        rojo16      :OUTSTD_LOGIC;
        rojo17      :OUTSTD_LOGIC

);
END luces;

ARCHITECTURE arq_luces OF luces IS


TYPE estado is(e0, e1, e2, e3, e4, e5,
e6,e7,e8,e9,e10,e11,e12,e13,e14,e15,e16);
SIGNAL ep, es   : estado;
SIGNAL cont         :STD_LOGIC;
SIGNAL contLUCES        :naturalrange0to50000000;
SIGNAL agudo :naturalrange0to50;
SIGNAl siguienteEstado:STD_LOGIC;
SIGNAL clcont :STD_LOGIC;


-------------------------------------------------------------------------
--
--UNIDAD DE CONTROL
-------------------------------------------------------------------------
--

BEGIN-- transicionn de estados

PROCESS(ep,clk,siguienteEstado)-- Lista de sensibilidad
BEGIN
```

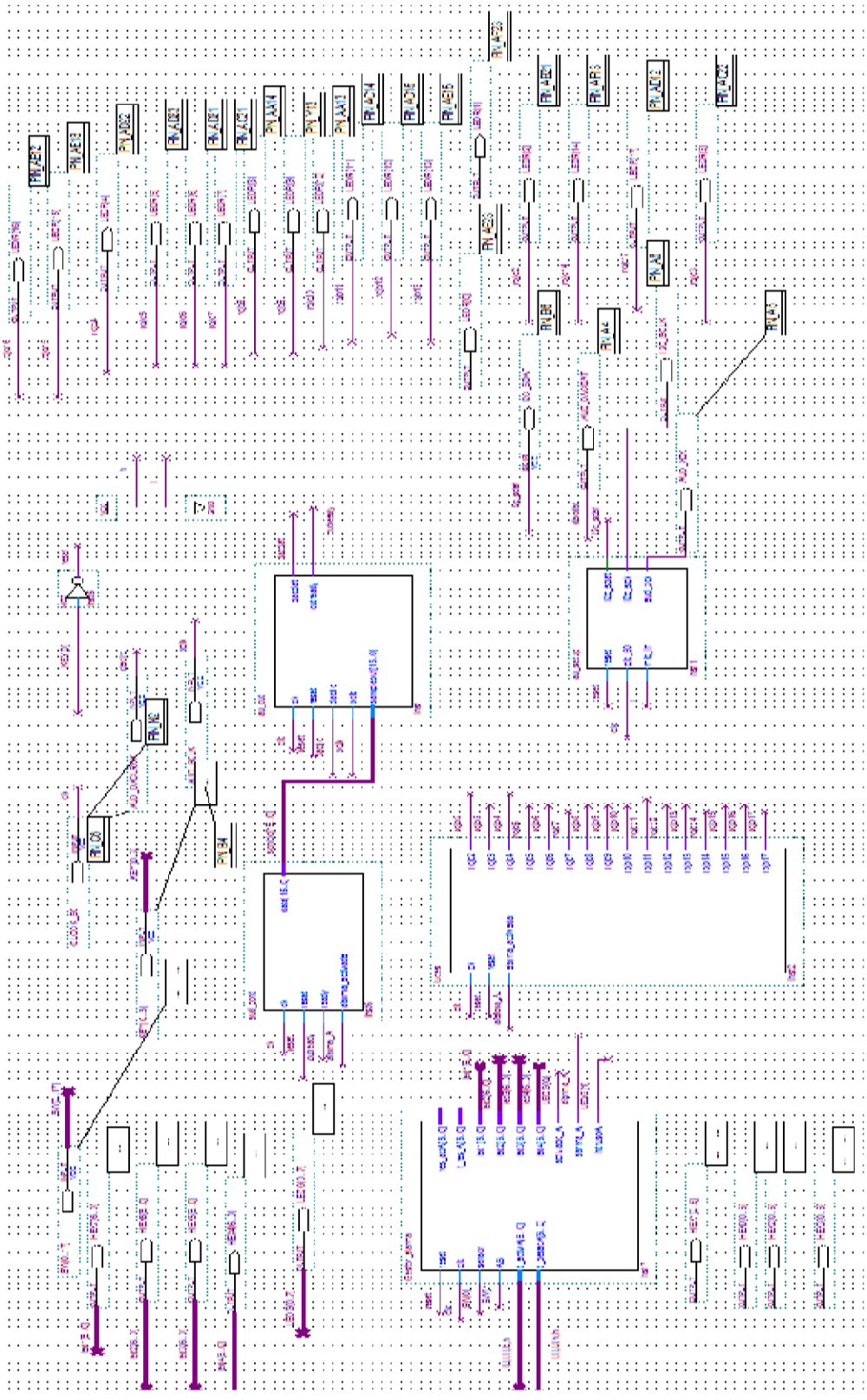
```
CASE ep IS
WHEN e0 =>
IF siguienteEstado='1' THEN es <= e1;
ELSIF alarma_activada='0' THEN es<=e16;
ELSE es<= e0;
ENDIF;
WHEN e1 =>
IF siguienteEstado='1' THEN es <= e2;
ELSIF alarma_activada='0' THEN es<=e16;
ELSE es<= e1;
ENDIF;
WHEN e2 =>
IF siguienteEstado='1' THEN es <= e3;
ELSIF alarma_activada='0' THEN es<=e16;
ELSE es<= e2;
ENDIF;
WHEN e3 =>
IF siguienteEstado='1' THEN es <= e4;
ELSIF alarma_activada='0' THEN es<=e16;
ELSE es<= e3;
ENDIF;
WHEN e4 =>
IF siguienteEstado='1' THEN es <= e5;
ELSIF alarma_activada='0' THEN es<=e16;
ELSE es<= e4;
ENDIF;
WHEN e5 =>
IF siguienteEstado='1' THEN es <= e6;
ELSIF alarma_activada='0' THEN es<=e16;
ELSE es<= e5;
ENDIF;
WHEN e6 =>
IF siguienteEstado='1' THEN es <= e7;
ELSIF alarma_activada='0' THEN es<=e16;
ELSE es<= e6;
ENDIF;
WHEN e7 =>
IF siguienteEstado='1' THEN es <= e9;
ELSIF alarma_activada='0' THEN es<=e16;
ELSE es<= e7;
ENDIF;
WHEN e8 =>
IF siguienteEstado='1' THEN es <= e9;
ELSIF alarma_activada='0' THEN es<=e16;
ELSE es<= e8;
ENDIF;
WHEN e9 =>
IF siguienteEstado='1' THEN es <= e10;
ELSIF alarma_activada='0' THEN es<=e16;
ELSE es<= e9;
ENDIF;
WHEN e10 =>
IF siguienteEstado='1' THEN es <= e11;
ELSIF alarma_activada='0' THEN es<=e16;
ELSE es<= e10;
ENDIF;
WHEN e11 =>
IF siguienteEstado='1' THEN es <= e12;
ELSIF alarma_activada='0' THEN es<=e16;
ELSE es<= e11;
ENDIF;
```

```
WHEN e12 =>
IF siguienteEstado='1' THEN es <= e13;
ELSIF alarma_activada='0' THEN es<=e16;
ELSE es<= e12;
ENDIF;
WHEN e13 =>
IF siguienteEstado='1' THEN es <= e14;
ELSIF alarma_activada='0' THEN es<=e16;
ELSE es<= e13;
ENDIF;
WHEN e14 =>
IF siguienteEstado='1' THEN es <= e15;
ELSIF alarma_activada='0' THEN es<=e16;
ELSE es<= e14;
ENDIF;
WHEN e15 =>
IF siguienteEstado='1' THEN es <= e0;
ELSIF alarma_activada='0' THEN es<=e16;
ELSE es<= e15;
ENDIF;
WHEN e16 =>
IF alarma_activada='1' THEN es <= e0;
ELSE es<= e16;
ENDIF;

ENDCASE;

ENDPROCESS;

------------------------------------------------------------------------
--
--Registro de estados
-----------------------

PROCESS(clk,reset)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                ep <= e0;
ELSE
                ep <= es;
ENDIF;
ENDIF;
ENDPROCESS;
------------------------------------------------------------------------
---


-- CONTADORES, registros, etc


PROCESS(clk, reset,siguienteEstado)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                contLUCES <=0;

ELSIF(contLUCES>3125000)THEN
              contLUCES<=0;
ELSE contLUCES<=contLUCES+1;
ENDIF;
```

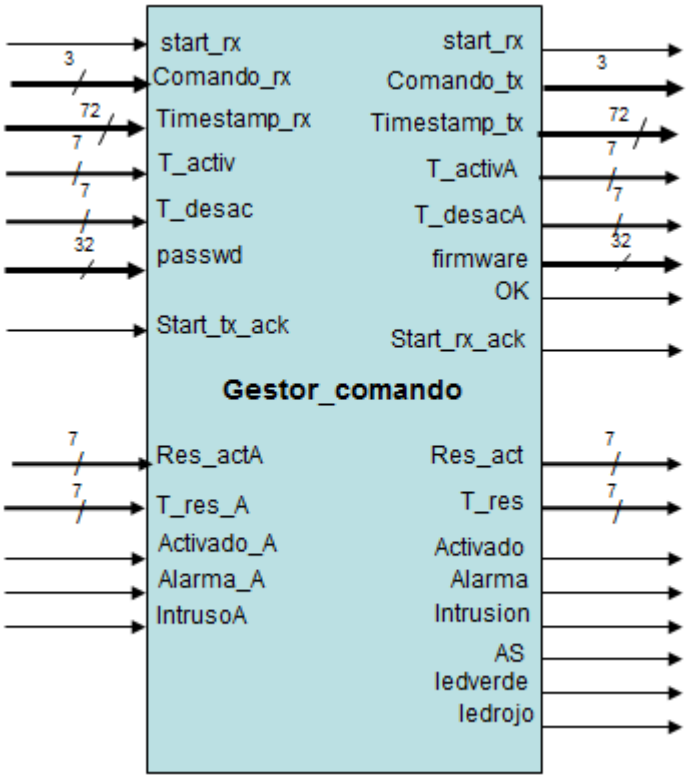
```
ENDIF;
ENDPROCESS;
    siguienteEstado <= '1' WHEN(contLUCES>3125000)ELSE '0';

-----------------------
-- SALIDAS DEL CIRCUITO
-----------------------
    rojo2       <= '1' WHEN(ep=e0 OR ep=e15)ELSE '0';
rojo17      <= '1' WHEN(ep=e0 OR ep=e15)ELSE '0';
    rojo3       <= '1' WHEN(ep=e0 OR ep=e1 OR ep=e15 OR ep=e14)ELSE
'0';
    rojo16      <= '1' WHEN(ep=e0 OR ep=e1 OR ep=e15 OR ep=e14)ELSE
'0';
    rojo4       <= '1' WHEN(ep=e1 OR ep=e2 OR ep=e14  OR ep=e13)ELSE
'0';
    rojo15      <= '1' WHEN(ep=e1 OR ep=e2 OR ep=e14  OR ep=e13)ELSE
'0';
    rojo5       <= '1' WHEN(ep=e2 OR ep=e3 OR ep=e13  OR ep=e12)ELSE
'0';
    rojo14      <= '1' WHEN(ep=e2 OR ep=e3 OR ep=e13  OR ep=e12)ELSE
'0';
    rojo6       <= '1' WHEN(ep=e3 OR ep=e4 OR ep=e12 OR ep=e11)ELSE
'0';
    rojo13      <= '1' WHEN(ep=e3 OR ep=e4 OR ep=e12 OR ep=e11)ELSE
'0';
    rojo7       <= '1' WHEN(ep=e4 OR ep=e5 OR ep=e11 OR ep=e10)ELSE
'0';
    rojo12      <= '1' WHEN(ep=e4 OR ep=e5 OR ep=e11 OR ep=e10)ELSE
'0';
    rojo8       <= '1' WHEN(ep=e5 OR ep=e6 OR ep=e10 OR ep=e9)ELSE
'0';
    rojo11      <= '1' WHEN(ep=e5 OR ep=e6 OR ep=e10 OR ep=e9)ELSE
'0';
    rojo9       <= '1' WHEN(ep=e6 OR ep=e7 OR ep=e9 OR ep=e8)ELSE '0';
    rojo10      <= '1' WHEN(ep=e6 OR ep=e7 OR ep=e9 OR ep=e8)ELSE '0';


END arq_luces;
```

e . Simulación

La simulación de este módulo se hizo directamente en la placa por lo que no ha requerido mucha comprobación. Le hemos juntado con el módulo de gestor_alarma y los módulos que componen el sonido que ya teníamos juntados y los probamos



5.6. GESTOR COMANDO



a . Descripción

Este modulo está pensado para gestionar el comando que recibimos por rx_comando, este modulo según el comando que se le pasa hará diferentes acciones, como puede ser, activar la alarma, desactivar la alarma, cambiar la configuración de la alarma, validar la contraseña… y informara al modulo tx_comando de el comando que se ha recibido y si todo ha ido bien o mal, para que este envié a la aplicación Mirakonta una respuesta.

Señales de entrada:

- **cmd_completo:** Indica al modulo que empiece a gestionar el comando.
- **Comando_rx:** El comando que recibe.
- **Timestamp_rx:** el tiempo que recibe.
- **T_activ:** Tiempo para activar la alarma
- **T_desac:** Tiempo para desactivar la alarma.
- **Passwd:** Contraseña que recibe.
- **Start_tx_ack:** Le indica que el modulo tx a terminado de gestionar la petición.
- **Activado_A:** Si la alarma esta activada.
- **Alarma_A:** Si ha saltado la alarma.
- **Res_actA:** Tiempo restante para la activación de la alarma.
- **intrusoA:** Si hay intrusos o no.
- **T_res_A:** Tiempo restante para que salte la alarma.

Señales de salida:

- **Start_tx:** envía una petición al modulo tx_comando para que empiece con la gestión de envió.
- **Comando_tx:** el comando que debe de enviar el modulo tx.
- **Timestamp_tx:** el tiempo que debe de enviar el modulo tx.
- **T_activA:** El tiempo que debe de utilizar el gestor de alarma para la activación de esta.
- **T_desacA:** El tiempo que debe de utilizar el gestor de alarma para la desactivación de esta.
- **Activado:** Estado de la alarma (1 activada 0 desactivada).
- **OK:** toma el valor 1 si el comando se a gestionado correctamente y 0 si ha ocurrido algún error.
- **Res_act:** Tiempo restante para la activación de la alarma.
- **Intrusion:** 1 si ha habido alguna intrusión y 0 si no.
- **T_res:** Tiempo restante para que salte la alarma.
- **Alarma:** Si la alarma a saltado o no.
- **cmd_completo_ack:** Se utiliza para indicar al modulo rx_comando que ya hemos terminado con la gestión de su petición.
- **Ledverde:** Para encender el led verde.
- **Ledrojo:** Para encender el led rojo.
- **AS:** Para decirle al modulo de gestión de alarma que active el sistema de alarma.
- **Firmware:** Versión.

b . Unidad de Control.

Para empezar en el estado E0, se ponen todos los valores por defecto de la alarma, en el estado E1 esperamos a que nos de el rx_comando la señal de que ya tiene el comando completo y que podemos empezar y en el estado E2 cargamos el comando en nuestro registro y reseteamos los contadores del número de comandos state que hemos recibido y el del número de intentos de meter la contraseña.

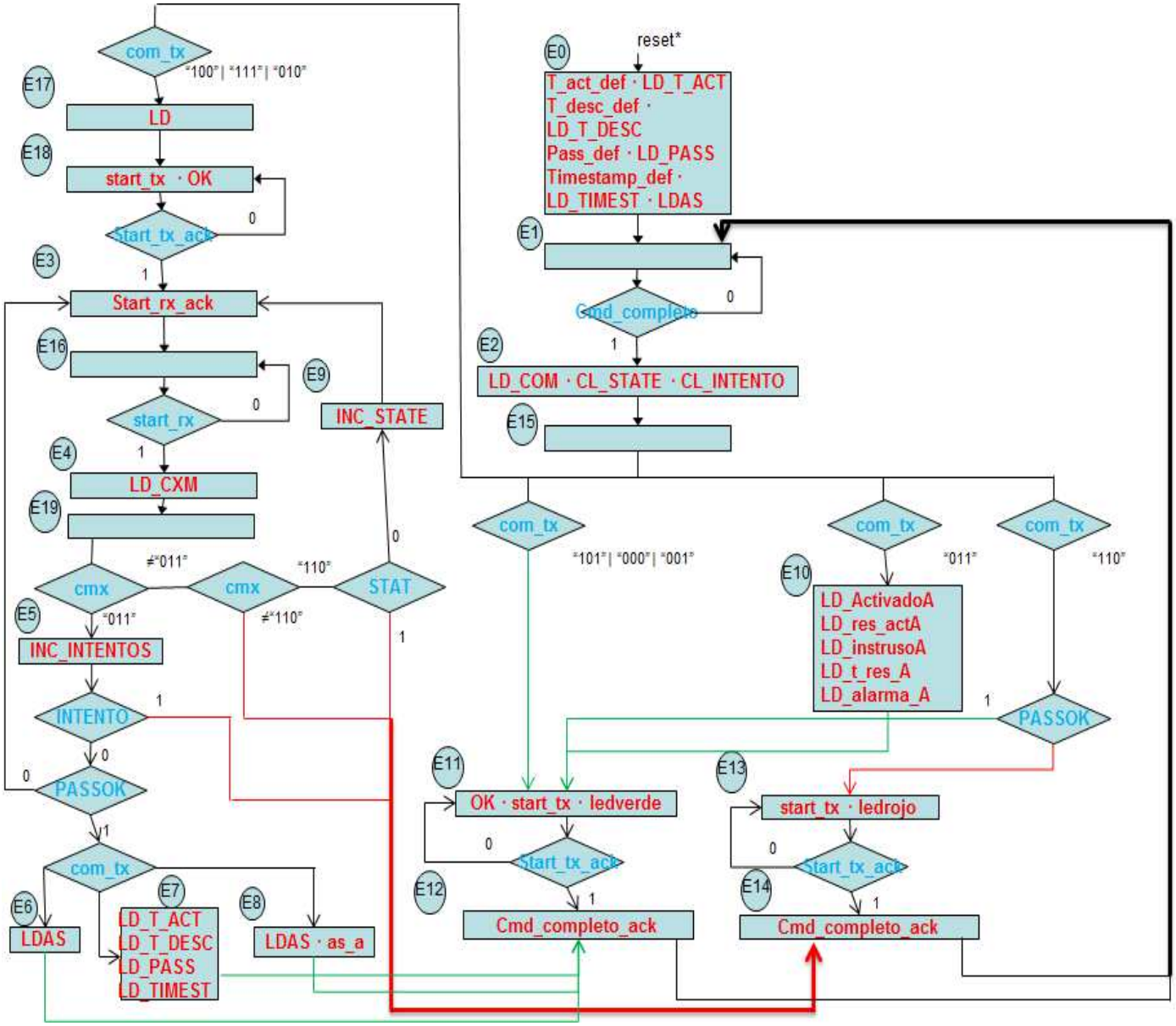
Posteriormente en el estado E15 dependiendo el comando que hayamos recibido vamos por un camino u otro, pasare a explicar dos de estos caminos ya que los otros dos se ven claramente lo que hacen.

En el caso de que el comando que hemos recibido sea 100 o 111 o 010, pasamos al estado E17 donde cargaremos los datos de configuración en unos registros auxiliares por si posteriormente se ha introducido bien la contraseña y el comando recibido es el de configuración pasemos a guardarlos en la placa. Para esto desde el estado E18 hasta el estado E5 lo que se hace es lo siguiente, se espera a que el tx_comando procese la respuesta a la aplicación mirakonta, posteriormente esperamos a recibir un comando y lo cargamos en un registro auxiliar, si el comando que se recibe no es ni el de la contraseña(011) o el de state(110) directamente vamos al estado E14 y volvemos al inicio, en caso contrario, contaremos el número de veces que llega el comando state demanera que nos sirve como temporizador si llegan 3 states y nadie ha metido la pass correcta volveremos al inicio, lo mismo pasara en caso de que metamos la contraseña mal 3 veces.
En el caso de que introduzcamos la contraseña de manera correcta miramos cual fue el comando que nos llegó al principio, si el de configuración, el de activación o el de desactivación y según eso hacemos una cosa u otra.

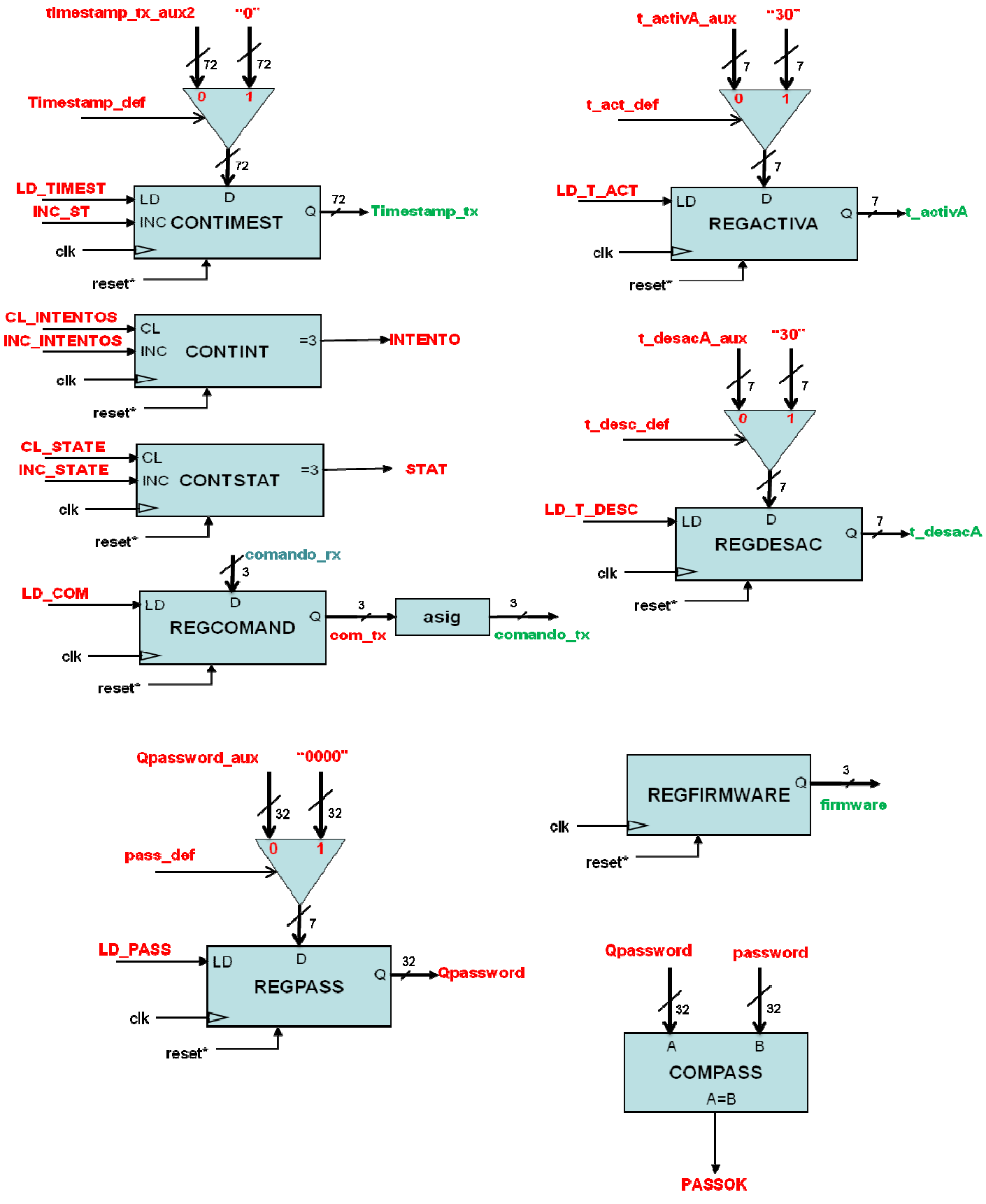
Si fue el comando de configuración iremos al estado E7 y guardaremos la nueva configuración después vamos al estado E12 y volvemos a empezar
En el caso de que sea el de desactivar iremos al estado E6 y continuamos como en el estado anterior
Y por último el de activación que iría al estado E8 y seguiría como el resto al E12 y luego al inicio E1.

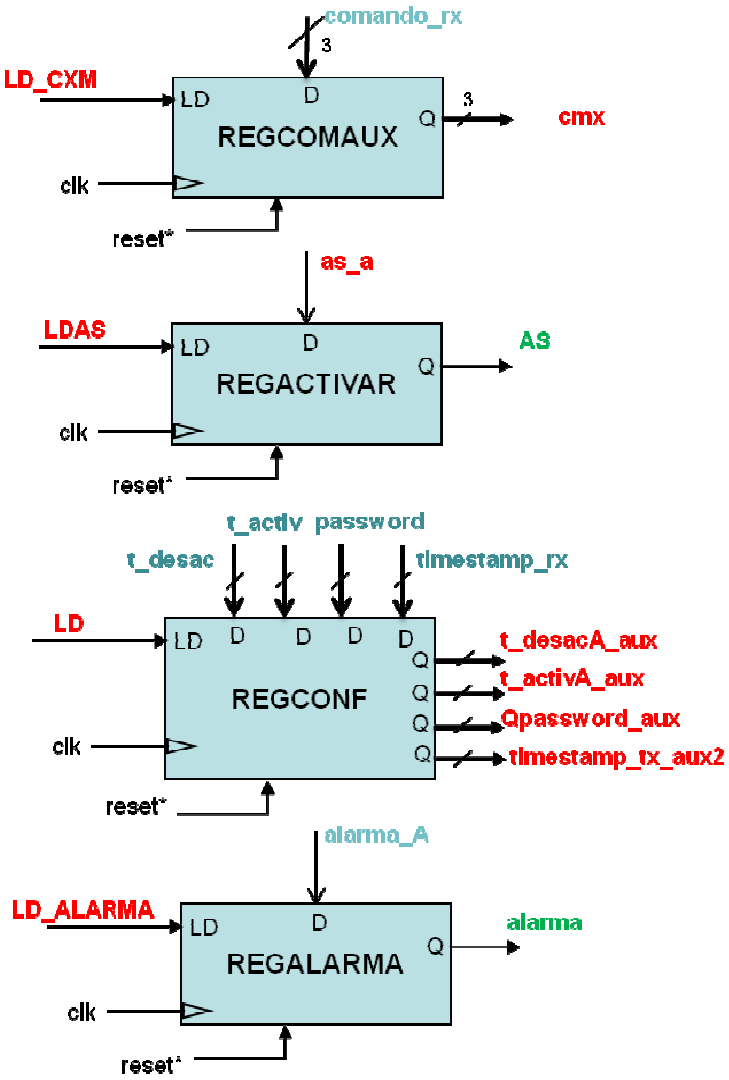
Si el comando que nos llega en el estado E15 es el de state, iremos al estado E10 donde cargaremos el estado actual del sistema después directamente iremos al estado E11 donde le diremos al módulo de tx_comando que transmita la respuesta y esperaremos a que nos responda con el start_tx_ack para ir al estado E12 y activar la señal de cmd_completo y para indicar al rx_comando que ya hemos procesado su petición y volvemos al estado E1 a la espera de más comandos

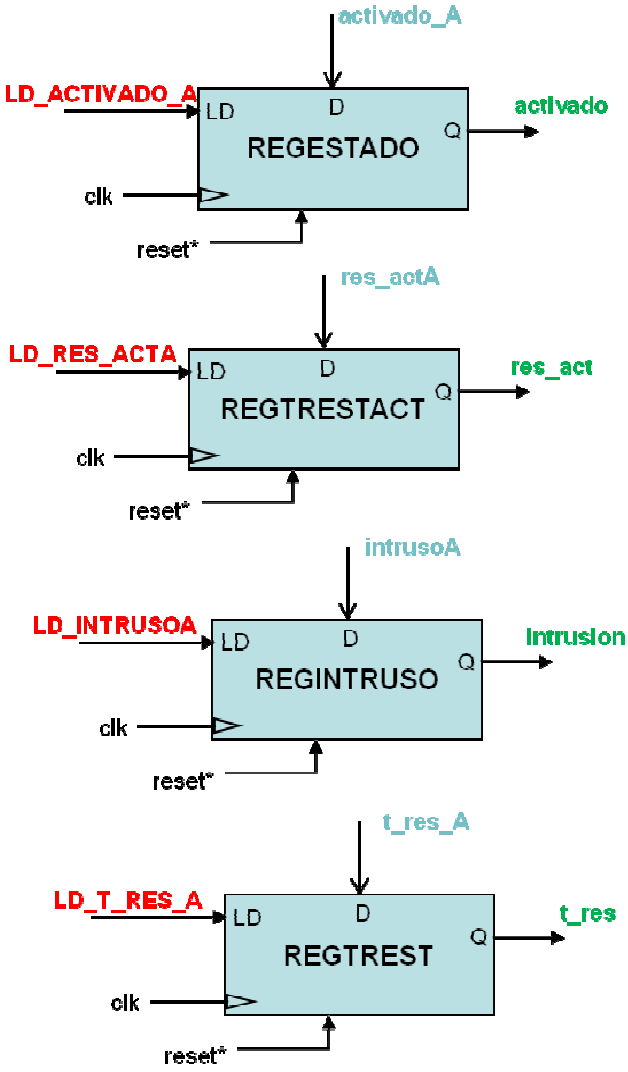
Para el resto de comandos como son el de ver la versión de la placa, el de la contraseña… se ve bastante claro en la imagen cual sería la manera de procesarlos


com_tx
"100"| "111"| "010"
E17
LD
E18
start_tx · OK
0
Start_tx_ack
E3
1
Start_rx_ack
E16
E9
0
start_rx
INC_STATE
1
E4
LD_CXM
E19
≠"011"
"110"
cmx
cmx
STAT
0
E5
"011"
≠"110"
1
INC_INTENTOS
INTENTO
1
0
0
PASSOK
1
com_tx
E6
E7
E8
LDAS
LD_T_ACT
LD_T_DESC
LD_PASS
LD_TIMEST
LDAS · as_a
reset*
E0
T_act_def · LD_T_ACT
T_desc_def ·
LD_T_DESC
Pass_def · LD_PASS
Timestamp_def ·
LD_TIMEST · LDAS
E1
0
Cmd_completo
1
E2
LD_COM · CL_STATE · CL_INTENTO
E15
com_tx
"101"| "000"| "001"
com_tx
"011"
com_tx
"110"
E10
LD_ActivadoA
LD_res_actA
LD_instrusoA
LD_t_res_A
LD_alarma_A
1
PASSOK
E11
E13
OK · start_tx · ledverde
start_tx · ledrojo
0
Start_tx_ack
0
Start_tx_ack
E12
1
E14
1
Cmd_completo_ack
Cmd_completo_ack

c . Unidad de Proceso.




comando_rx
3
LD_CXM
LD
D
REGCOMAUX
Q
3
cmx
clk
reset*
as_a
LDAS
LD
D
REGACTIVAR
Q
AS
clk
reset*
t_activ
password
t_desac
timestamp_rx
LD
LD
D
D
D
D
REGCONF
Q
Q
Q
Q
t_desacA_aux
t_activA_aux
Qpassword_aux
timestamp_tx_aux2
clk
reset*
alarma_A
LD_ALARMA
LD
D
REGALARMA
Q
alarma
clk
reset*


activado_A
LD_ACTIVADO_A
LD
D
REGESTADO
Q
activado
clk
reset*
res_actA
LD_RES_ACTA
LD
D
REGTRESTACT
Q
res_act
clk
reset*
intrusoA
LD_INTRUSOA
LD
D
REGINTRUSO
Q
intrusion
clk
reset*
t_res_A
LD_T_RES_A
LD
D
REGTREST
Q
t_res
clk
reset*

d. Código VHDL

```
--  Library Clause
LIBRARYIEEE;
LIBRARYwork;


--  Use Clause
USEIEEE.STD_LOGIC_1164.ALL;
USEIEEE.std_logic_arith.all;
USEIEEE.std_logic_unsigned.all;


ENTITY GESTOR_COMANDO IS
PORT(

    reset,clk       :INSTD_LOGIC;
--modulo RX
--------ENTRADAS
    cmd_completo    :INSTD_LOGIC;
comando_rx     :INSTD_LOGIC_VECTOR(2DOWNTO0);
    timestamp_rx    :INNATURALrange0to999999999;
    t_activ         :INNATURALrange0to99;
    t_desac         :INNATURALrange0to99;
    password        :INSTD_LOGIC_VECTOR(31DOWNTO0);
--------SALIDAS
    cmd_completo_ack    :OUTSTD_LOGIC;

--modulo TX
--------ENTRADAS
    start_tx_ack    :INSTD_LOGIC;
--------SALIDAS
    start_tx        :OUTSTD_LOGIC;
    comando_tx      :OUTSTD_LOGIC_VECTOR(2DOWNTO0);
    timestamp_tx    :OUTNATURALRANGE0TO999999999;
    activado        :OUTSTD_LOGIC;
    OK              :OUTSTD_LOGIC;
    res_act         :OUTNATURALrange0to99;
    t_res           :OUTNATURALrange0to99;
    intrusion       :OUTSTD_LOGIC;
    alarma          :OUTSTD_LOGIC;
    firmware        :OUTSTD_LOGIC_VECTOR(31DOWNTO0);

--modulo GestorAlarma
--------ENTRADAS
    activado_A      :INSTD_LOGIC;
alarma_A       :INSTD_LOGIC;
    res_actA        :INNATURALrange0to99;
    intrusoA        :INSTD_LOGIC;
    t_res_A         :INNATURALrange0to99;
--------SALIDAS
    t_activA        :OUTNATURALrange0to99;
    t_desacA        :OUTNATURALrange0to99;
    AS              :OUTSTD_LOGIC;

----------------------
    ledVerde        :OUTSTD_LOGIC;
    ledRojo         :OUTSTD_LOGIC
);
END GESTOR_COMANDO;
```

```
ARCHITECTURE a OF GESTOR_COMANDO IS

type estado is(e0, e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11, e12,
e13, e14,e15,e16,e17,e18,e19);
signal ep, es                              :estado;
--Definicion de señales internas
SIGNAL estado_sig       :naturalrange0to6;
--CONTADOR Timestamp
SIGNAL timestamp_def,LD_TIMEST,INC_ST              :STD_LOGIC;
--CONTADOR intentos
SIGNAL INTENTO,CL_INTENTOS,INC_INTENTOS            :STD_LOGIC;
--CONTADOR state
SIGNAL STAT,CL_STATE,INC_STATE              :STD_LOGIC;
--Registro Tiempo de Activacion
SIGNAL t_act_def,LD_T_ACT                          :STD_LOGIC;
--Registro Tiempo de Desactivacion
SIGNAL t_desc_def,LD_T_DESC                        :STD_LOGIC;
--Registro Password y comparadores Password
SIGNAL pass_def,LD_PASS                    :STD_LOGIC;
signal PASSOK :STD_LOGIC;
SIGNAL Qpassword,Qpassword_aux
:STD_LOGIC_VECTOR(31DOWNTO0);
--Registro Comando,ComandoAuxiliar(cxm),Activado,Tiempo restante de
activacion, intruso, tiempo restante para señal de alarma, señal de
alarma
SIGNAL
LD_COM,LD_CXM,LDAS,as_a,LD_ACTIVADO_A,LD_RES_ACTA,LD_INTRUSOA,LD_T_RES
_A,LD_ALARMA,LD        :STD_LOGIC;
SIGNAL com_tx,cmx
:STD_LOGIC_VECTOR(2DOWNTO0);
SIGNAL timestamp_tx_aux,timestamp_tx_aux2
:NATURALrange0to999999999;
SIGNAL state                               :NATURALrange0to9;
SIGNAL intentos                            :NATURALrange0to9;
SIGNAL  tiempo:naturalrange0to50000000;
SIGNAL t_activA_aux                        :NATURALrange0to99;
SIGNAL t_desacA_aux                        :NATURALrange0to99;
--------------------------------------------------------------------------
--
--UNIDAD DE CONTROL
--------------------------------------------------------------------------
--

BEGIN-- transicion de estados

PROCESS(ep, cmd_completo,PASSOK,start_tx_ack,com_tx,clk)-- Lista de
sensibilidad
BEGIN
CASE ep IS
WHEN e0 =>
                  es <= e1;
WHEN e1 =>
if(cmd_completo='1')then
es <= e2;
else
                   es <= e1;
endif;
when e2 =>
                 es <= e15;
when e3 =>
               es <= e16;
```

```
when e4 =>
                es <= e19;
WHEN e5 =>
if(INTENTO='0')then
if(PASSOK='1')then
if(com_tx="111")then
                            es <= e6;
elsif(com_tx="010")then
es <= e7;
elsif com_tx ="100"then
                            es <= e8;
endif;
else
                        es <= e3;
endif;
else
                    es <= e14;
endif;
WHEN e6 =>
                es <= e12;
WHEN e7 =>
                es <= e12;
WHEN e8 =>
                 es <= e12;
WHEN e9 =>
                  es <= e3;
WHEN e10 =>
                  es <= e11;
WHEN e11 =>
if(start_tx_ack='1')then
es <= e12;
else
                        es <= e11;
endif;
WHEN e12 =>
                   es <= e1;
WHEN e13 =>
if(start_tx_ack='1')then
es <= e14;
else
                        es <= e13;
endif;
WHEN e14 =>
                 es <= e1;
WHEN e15 =>
if(com_tx="000")then
es <= e11;
elsif(com_tx="101")then
es <= e11;
elsif(com_tx="001")then
                    es <= e11;
elsif(com_tx="011")then
if(PASSOK='1')then
es <= e11;
else
                        es <= e13;
endif;
elsif(com_tx="110")then
es <= e10;
elsif(com_tx="010")then
                    es <= e17;
```

```
elsif(com_tx="100")then
                    es <= e17;
elsif com_tx ="111"then
                    es <= e17;
endif;
WHEN e16 =>
if(cmd_completo='1')then
es <= e4;
else
                    es <= e16;
endif;
WHEN e17 =>
                es <= e18;
WHEN e18 =>
if(start_tx_ack='1')then
es <= e3;
else
                        es <= e18;
endif;
WHEN e19 =>
if(cmx="011")then
                    es <= e5;
elsif(cmx="110")then
if(STAT='1')then
es <= e14;
else
                      es <= e9;
endif;
else
                    es <= e14;
endif;
ENDCASE;

ENDPROCESS;

-------------------------------------------------------------------------
--
--Registro de estados
-----------------------
process(clk, reset)
begin
if(reset='1')then
          ep <= e0;
elsif(clk'EVENTand clk='1')then
          ep <= es;
endif;
endprocess;
-------------------------------------------------------------------------
---
-------------------------------------------------------------------------
---
-- Salidas de la unidad de control
-------------------------------------------------------------------------
---

t_act_def <= '1' when(ep=e0)else '0';
INC_STATE <= '1' when(ep=e9)else '0';
INC_INTENTOS <= '1' when(ep=e5)else '0';
CL_STATE <= '1' when(ep=e2)else '0';
CL_INTENTOS <= '1' when(ep=e2)else '0';
LD_T_ACT <= '1' when(ep=e0 or ep=e7)else '0';
```

```
t_desc_def <= '1' when(ep=e0)else '0';
LD_T_DESC <= '1' when(ep=e0 or ep=e7)else '0';
pass_def <= '1' when(ep=e0)else '0';
LD_PASS <= '1' when(ep=e0 or ep=e7)else '0';
timestamp_def <= '1' when(ep=e0)else '0';
LD_TIMEST <= '1' when(ep=e0 or ep=e7)else '0';
LD_COM <= '1' when(ep=e2)else '0';
OK  <= '1' when(ep=e11 or ep=e18)else '0';
ledVerde <= '1' when(ep=e11)else '0';
ledRojo <= '1' when( ep=e13 )else '0';
start_tx <= '1' when(ep=e11 or ep=e13 or ep=e18)else '0';
cmd_completo_ack <= '1' when(ep=e3 or ep=e12 or ep=e14)else '0';
as_a <= '1' when(ep=e8)else '0';
LDAS <= '1' when(ep=e0 or ep=e8 or ep=e6)else '0';
LD_ACTIVADO_A <= '1' when(ep=e10)else '0';
LD_RES_ACTA     <= '1' when(ep=e10)else '0';
LD_INTRUSOA   <= '1' when(ep=e10)else '0';
LD_T_RES_A    <= '1' when(ep=e10)else '0';
LD_ALARMA     <= '1' when(ep=e10)else '0';
LD_CXM    <= '1' when(ep=e4)else '0';
LD   <= '1' when(ep=e17)else '0';

-------------------------------------------------------------------------
-------
-------------------------------------------------------------------------
-------

-------------------------------------------------------------------------
------
--UNIDAD DE PROCESO
-------------------------------------------------------------------------
------


-- CONTADORES, registros, etc

-- CONTADORES, registros, etc


PROCESS(clk,reset)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                tiempo <=50000000;-- Ponemos el contador a Cero
ENDIF;
if(tiempo >0)THEN
                tiempo <= tiempo -1;-- decrementamos si el valor es
mayor a 0
else
                tiempo <=50000000;
endif;
ENDIF;
ENDPROCESS;
    INC_ST <= '1' when tiempo =1else '0';




-------------------
```

```
-- CONTADOR TIEMPO + MULTIPLEXOR
-------------------

PROCESS(clk,reset,INC_ST)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                timestamp_tx_aux <=0;-- Ponemos el contador a Cero
ELSIF(LD_TIMEST = '1')THEN
if(timestamp_def='1')then-- Ponemos el valor por defecto
timestamp_tx_aux <=0;
else
                  timestamp_tx_aux <= timestamp_tx_aux2;
endif;
ELSIF(INC_ST = '1')THEN
                timestamp_tx_aux <= timestamp_tx_aux +1;-- Contamos
ENDIF;
ENDIF;
ENDPROCESS;
    timestamp_tx <= timestamp_tx_aux;
-------------------
-- CONTADOR INTENTOS
-------------------

PROCESS(clk,reset)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                intentos <=1;-- Ponemos el contador a Cero
ELSIF(CL_INTENTOS = '1')THEN
                intentos <=1;
ELSIF(INC_INTENTOS = '1')THEN
intentos <= intentos +1;-- Contamos
ENDIF;
ENDIF;
ENDPROCESS;

    INTENTO <= '1' when(intentos >2)else '0';
-------------------
-- CONTADOR STATE
-------------------

PROCESS(clk,reset)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
state <=1;-- Ponemos el contador a Cero
ELSIF(CL_STATE = '1')THEN
                state <=1;
ELSIF(INC_STATE = '1')THEN
                state <= state +1;-- Contamos
ENDIF;
ENDIF;
ENDPROCESS;
STAT <= '1' when(state >2)else '0';
-------------------
-- REGISTRO TIEMPO ACTIVACION + MULTIPLEXOR
-------------------
PROCESS(clk,reset)
BEGIN
IF(clk'EVENTAND clk='1')THEN
```

```
IF reset = '1' THEN
                t_activA <=30;-- Ponemos el valor por defecto
ELSIF(LD_T_ACT = '1')THEN
if(t_act_def='1')then-- Ponemos el valor por defecto
                 t_activA <=30;
else
                 t_activA <= t_activA_aux;
endif;
ENDIF;
ENDIF;
ENDPROCESS;
-------------------
-- REGISTRO TIEMPO DESACTIVACION + MULTIPLEXOR
-------------------
PROCESS(clk,reset)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                t_desacA <=30;-- Ponemos el valor por defecto
ELSIF(LD_T_DESC = '1')THEN
if(t_desc_def='1')then-- Ponemos el valor por defecto
                 t_desacA <=30;
else
                 t_desacA <= t_desacA_aux;
endif;
ENDIF;
ENDIF;
ENDPROCESS;
-------------------
-- REGISTRO COMANDO
-------------------
PROCESS(clk,reset)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                com_tx <="000";-- Ponemos el valor por defecto
ELSIF(LD_COM = '1')THEN
                com_tx <= comando_rx;
ENDIF;
ENDIF;
ENDPROCESS;
comando_tx <= com_tx;
-------------------
-- REGISTRO COMANDO AUXILIAR
-------------------
PROCESS(clk,reset)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                cmx <="000";-- Ponemos el valor por defecto
ELSIF(LD_CXM = '1')THEN
                cmx <= comando_rx;
ENDIF;
ENDIF;
ENDPROCESS;

-------------------
-- REGISTRO ACTIVAR ALARMA
-------------------
PROCESS(clk,reset)
BEGIN
```

```
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                AS <= '0';-- Ponemos el valor por defecto
ELSIF(LDAS = '1')THEN
                AS <= as_a;
ENDIF;
ENDIF;
ENDPROCESS;
--------------------
-- REGISTRO CONF AUX
--------------------
PROCESS(clk,reset)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                t_desacA_aux <=30;
t_activA_aux <=30;-- Ponemos el valor por defecto
Qpassword_aux <= X"30303030";
                timestamp_tx_aux2 <=0;
ELSIF(LD = '1')THEN
                t_desacA_aux <=t_desac;
                t_activA_aux <=t_activ;
                Qpassword_aux <= password;
                timestamp_tx_aux2 <= timestamp_rx;
ENDIF;
ENDIF;
ENDPROCESS;
--------------------
-- REGISTRO ALARMA ACTIVADA
--------------------
PROCESS(clk,reset)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                activado <= '0';-- Ponemos el valor por defecto
ELSIF(LD_ACTIVADO_A = '1')THEN
                activado <= activado_A;
ENDIF;
ENDIF;
ENDPROCESS;
--------------------
-- REGISTRO TIEMPO RESTANTE ACTIVAR ALARMA (despues de activarla)
--------------------
PROCESS(clk,reset)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                res_act <= res_actA;-- Ponemos el valor por defecto
ELSIF(LD_RES_ACTA = '1')THEN
                res_act <= res_actA;
ENDIF;
ENDIF;
ENDPROCESS;
--------------------
-- REGISTRO INTRUSION
--------------------
PROCESS(clk,reset)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                intrusion <= '0';-- Ponemos el valor por defecto
```

```
ELSIF(LD_INTRUSOA = '1')THEN
                intrusion <= intrusoA;
ENDIF;
ENDIF;
ENDPROCESS;
-------------------
-- REGISTRO TIEMPO RESTANTE ACTIVAR ALARMA (despues de que se detecte
intrusion)
-------------------
PROCESS(clk,reset)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                t_res <= t_res_A;-- Ponemos el valor por defecto
ELSIF(LD_T_RES_A = '1')THEN
                t_res <= t_res_A;
ENDIF;
ENDIF;
ENDPROCESS;
-------------------
-- REGISTRO ALARMA
-------------------
PROCESS(clk,reset)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                alarma <= '0';-- Ponemos el valor por defecto
ELSIF(LD_ALARMA = '1')THEN
                alarma <= alarma_A;
ENDIF;
ENDIF;
ENDPROCESS;
-------------------
-- REGISTRO PASS + MULTIPLEXOR
-------------------
PROCESS(clk,reset)
BEGIN
IF(clk'EVENTAND clk='1')THEN
IF reset = '1' THEN
                Qpassword <= X"30303030";
-- Ponemos el valor por defecto
ELSIF(LD_PASS = '1')THEN
if(pass_def='1')then-- Ponemos el valor por defecto
Qpassword <= X"30303030";
else
                  Qpassword <= Qpassword_aux;
endif;
ENDIF;

ENDIF;
ENDPROCESS;

-------------------
-- COMPARADORES
-------------------
  PASSOK <= '1' when password = Qpassword else '0';
  firmware <= X"41444D33";
END a;
```

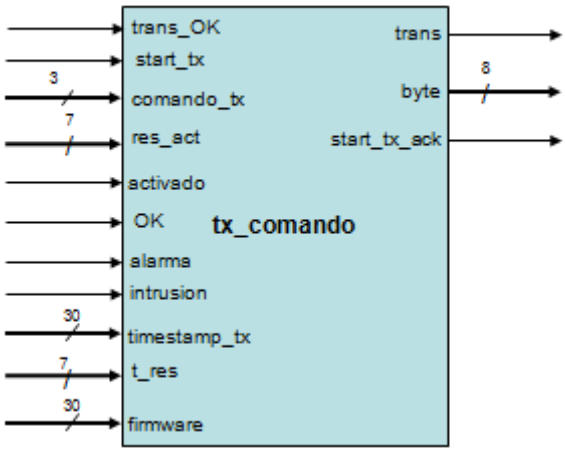
e. Simulación

La comprobación de este módulo se basó en varias pruebas realizadas con el programa Quartus y la propia placa DE2 de Altera. (Sin antes haberlo simulado con el ModelSim debido a la complejidad, número de entradas y salidas que se tienen que generar…)

Para está realización de las pruebas, conectamos este módulo realizado, con el resto de módulos, que anteriormente ya fueron comprobados, es decir que este fue el último modulo del que se comprobó el funcionamiento

Para ver el funcionamiento de este módulo se precisa de una muestra en directo con la altera o de un video ya que con imágenes no podríamos demostrar su correcto funcionamiento

5.7. TX_COMANDO



a. Descripción

El propósito de este módulo es la construcción del comando entero a enviar a la aplicación de Mirakonta. Este módulo está conectado por una parte al gestor de comandos y por otra parte al tx_byte. Al recibir la señal de transmitir, con los datos recibidos creará el comando oportuno para enviarlo.

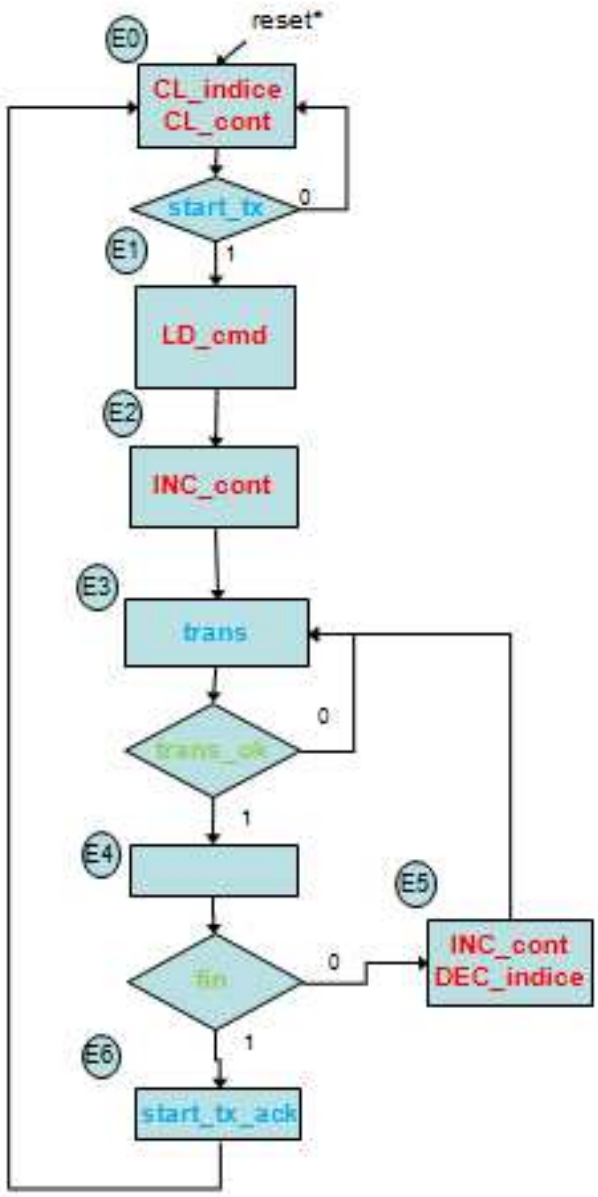
Señales de entrada:

- **trans_OK**: Indica que debe comenzar la transmisión de un nuevo byte.
- **start_tx**: Indica que tiene que empezar a transmitir el comando.
- **comando_tx**: Indica que comando es el que hay que transmitir.
- **res_act**: Tiempo restante para la activación de la alarma.
- **activado**: Indica si está o no activada la alarma.
- **OK**: Indica si el procesamiento del comando ha ido bien o mal.
- **alarma**:Indica si ha saltado la alarma o no.
- **intrusion**: Indica si ha habido alguna intrusión o no.
- **timestamp_tx:**Es la fecha y hora actual representado en 9 caracteres**.**
- **t_res**: Tiempo restante para la activación de la señal alarma.
- **firmware**: Versión de la alarma.

Señales de salida:

- **trans**: Indica que hay que transmitir el byte al modulo inferior.
- **byte**: Dato de 8 bits a transmitir.
- **start_tx_ack**: Indica al módulo superior que ha recibido con éxito la señal start_tx y que comenzará a enviar el comando seleccionado.

b. Unidad de Control



Primeramente, estaremos en el estado E0, en el cual haremos iniciaremos el índice que utilizaremos para ir determinando la posición de cada byte del comando a enviar y el contador que contará los bytes que vamos enviando. En este estado, preguntaremos por la señal start_tx que nos la enviará el gestor de los comandos que indicará que debemos empezar a enviar un comando. Si esta señal está activada, pasaremos al estado E1, sino seguiremos en el mismo estado.

En el estado E1, activaremos la señal LD_cmd que cargará en una señal interna auxiliar el número de comando a transmitir. De este estado pasaremos al estado E2 incondicionalmente.

En el siguiente estado, el E2, activaremos la señal INC_cont e iremos al estado E3 posteriormente sin ninguna condición.

En el estado E3, activaremos la señal trans para hacerle saber al módulo tx_byte de que hay un byte disponible para transmitir. En este estado preguntaremos por la señal trans_ok que nos la activará el módulo tx_byte que indicará que se ha transmitido el byte correctamente. Cumplida esta condición, pasaremos al estado E4, sino seguiremos en el estado E3.

En el estado E4, únicamente comprobaremos si la señal fin esta activada, la cual se activará una vez enviados todos los bytes. Cumplida esta condición pasaremos al estado E6, sino iremos al estado E5.

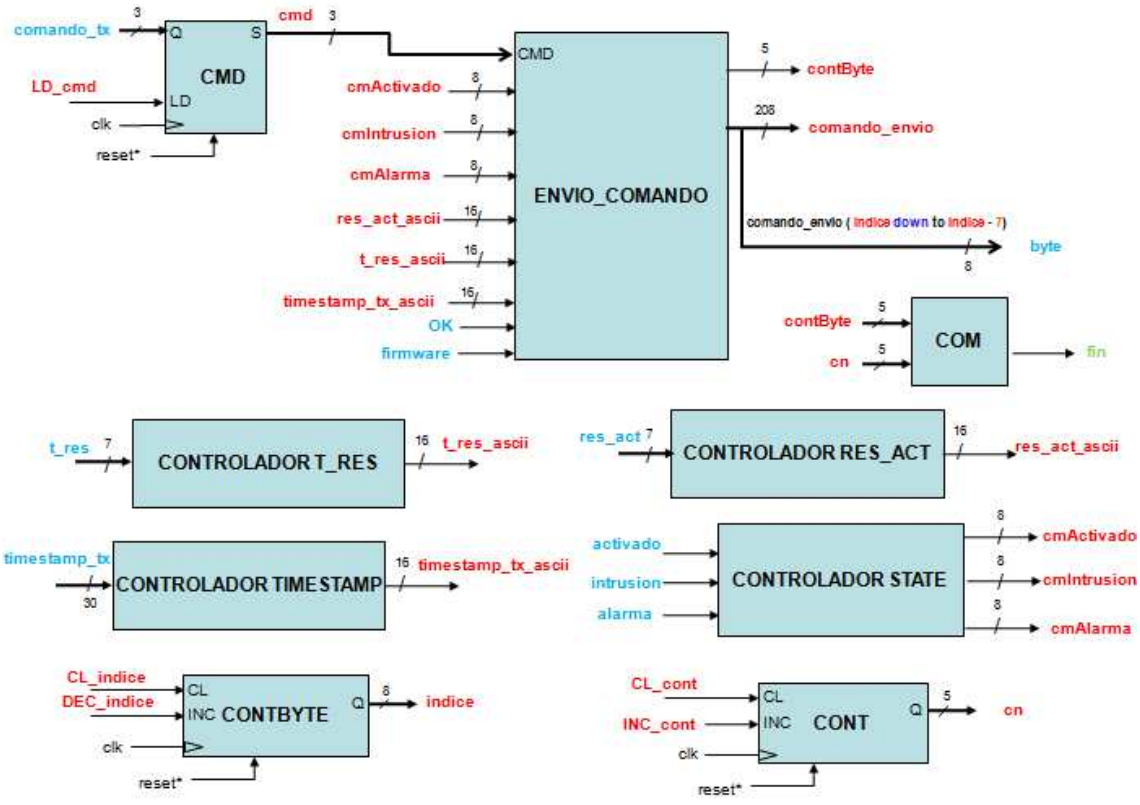
En el estado E5, incrementaremos el contador de bytes transmitidos, y decrementaremos el índice, ya que la transmisión del comando se empieza desde el byte de más peso. Incondicionalmente pasaremos al estado E3.

Por último, en el estado E6, activaremos la señal start_tx_ack, para avisarle al gestor de comando que se ha enviado el comando completo y regresaremos incondicionalmente al primer estado, el E0.

c. Unidad de Proceso

La tabla que implementará el bloque combinacional llamado **ENVIO_COMANDO** será:

OK / comando_tx	1	0
000	+MK:,OK	+MK:,ERROR
001	+V:...OK	+V:...,ERROR
010	+CONF:,OK	+CONF:,ERROR
011	+PASS:,OK	+PASS:,ERROR
100	+ACT:,OK	+ACT:,ERROR
101	+D&H:...,OK	+D&H...,ERROR
110	+STATE: ...,OK	+STATE: ...,ERROR
111	+DEACT:,OK	+DEACT:,ERROR



d. Código VHDL

```
--  Library Clause
LIBRARYIEEE;
LIBRARYwork;


--  Use Clause
USEIEEE.STD_LOGIC_1164.ALL;
USEIEEE.std_logic_arith.all;
USEIEEE.std_logic_unsigned.all;


entity tx_comando IS
port
(
    reset, clk            :instd_logic;
    trans_OK              :instd_logic;
    start_tx              :instd_logic;
    comando_tx            :inunsigned(2downto0);
    timestamp_tx          :innaturalrange0to999999999;
OK                    :instd_logic;
    activado              :instd_logic;
    res_act               :innaturalrange0to99;
intrusion             :instd_logic;
      t_res                    :innaturalrange0to99;
alarma                :instd_logic;
      firmware               :instd_logic_vector(31downto0);

    byte                  :outstd_logic_vector(7downto0);
trans                 :outstd_logic;
    start_tx_ack          :outstd_logic

);
end tx_comando;

architecture a of tx_comando is

-- señales internas de UP
signal contByte         :unsigned(4downto0);
signal cn               :unsigned(4downto0);
signal cmd              :unsigned(2downto0);
signal comando_envio    :std_logic_vector(207downto0);
signal indice           :naturalrange0to207:=207;

--Señales para concatenar los tiempos
signal timestamp_tx_ascii :std_logic_vector(71downto0);
signal t_res_ascii  :std_logic_vector(15downto0);
signal res_act_ascii  :std_logic_vector(15downto0);

--Señales para la conversión a ascii de las señales alarma, intrusión
y activado
signal cmActivado :std_logic_vector(7downto0);
signal cmIntrusion :std_logic_vector(7downto0);
signal cmAlarma :std_logic_vector(7downto0);

--Señales para la traducción de binario a ascii del timestamp
signal digito1              :std_logic_vector(7downto0);
signal digito2              :std_logic_vector(7downto0);
signal digito3              :std_logic_vector(7downto0);
signal digito4              :std_logic_vector(7downto0);
```

```
signal digito5              :std_logic_vector(7downto0);
signal digito6              :std_logic_vector(7downto0);
signal digito7              :std_logic_vector(7downto0);
signal digito8              :std_logic_vector(7downto0);
signal digito9              :std_logic_vector(7downto0);

signal resto1  :naturalrange0to99999999;
signal resto2  :naturalrange0to9999999;
signal resto3  :naturalrange0to999999;
signal resto4  :naturalrange0to99999;
signal resto5  :naturalrange0to9999;
signal resto6  :naturalrange0to999;
signal resto7  :naturalrange0to99;
signal resto8  :naturalrange0to9;

signal trozo1 :naturalrange0to9;
signal trozo2 :naturalrange0to9;
signal trozo3 :naturalrange0to9;
signal trozo4 :naturalrange0to9;
signal trozo5 :naturalrange0to9;
signal trozo6 :naturalrange0to9;
signal trozo7 :naturalrange0to9;
signal trozo8 :naturalrange0to9;
signal trozo9 :naturalrange0to9;

--Señales para la traducción de binario a ascii del t_res
signal t_res_digito1  :std_logic_vector(7downto0);
signal t_res_digito2  :std_logic_vector(7downto0);
signal t_res_resto1  :naturalrange0to9;
signal t_res_trozo1 :naturalrange0to9;
signal t_res_trozo2 :naturalrange0to9;

--Señales para la traducción de binario a ascii del res_act
signal res_act_digito1  :std_logic_vector(7downto0);
signal res_act_digito2  :std_logic_vector(7downto0);
signal res_act_resto1  :naturalrange0to9;
signal res_act_trozo1 :naturalrange0to9;
signal res_act_trozo2 :naturalrange0to9;


-- estados presentes y siguiente de UC
type estado is(e0,e1,e2,e3,e4,e5,e6);
signal ep,es : estado;

-- señales UC->UP
signal CL_cont, CL_indice, INC_cont, LD_cmd, DEC_indice:std_logic;

-- señales UP->UC
signal fin :std_logic;

begin
--====================================
-- Unidad de Control
--====================================
-------------------------------------
-- Transicion de estados
process(ep, start_tx, fin, trans_OK)
begin
case ep is
when e0 =>
if start_tx = '1' then es <= e1;
```

```
else es <= e0;
endif;
when e1 =>
         es <= e2;
when e2 =>
        es <= e3;
when e3 =>
if trans_OK = '1' then es <= e4;
else es <= e3;
endif;
when e4 =>
if fin = '1' then es <= e6;
else  es <= e5;
endif;
when e5 =>
        es <= e3;
when e6 =>
es <= e0;


endcase;
endprocess;

----------------------------------------
-- REGISTRO de estados
process(clk, reset)
begin
if(reset='1')then
          ep <= e0;
elsif(clk'EVENTand clk='1')then
ep <= es;
endif;
endprocess;

----------------------------------------
-- salidas de la unidad de control
CL_indice <='1' when ep = e0 else '0';
  CL_cont <= '1' when ep = e0 else '0';
  LD_cmd <= '1'  when ep = e1 else '0';
  trans <='1' when(ep = e3)else '0';
  DEC_indice <= '1' when ep = e5 else '0';
  INC_cont <= '1'  when ep = e2 or ep=e5 else '0';
  start_tx_ack <= '1' when ep = e6 else '0';

--====================================
-- Unidad de Proceso
--====================================
------------------------------
-- CMD
process(clk, reset)
begin
if reset ='1' then
        cmd <="000";
elsif clk'eventand clk='1' then
if LD_cmd = '1' then
        cmd <= comando_tx;
endif;
endif;
endprocess;

-------------------------
```

```
-- Comparador
   fin <='1' when contByte = cn else '0';

-------------------------------
-- Contador
process(clk, reset)
begin
if reset = '1' then
      cn <="00000";
elsif clk'eventand clk='1' then
if CL_cont = '1' then
          cn <="00000";
elsif INC_cont = '1' then
              cn <= cn+1;
endif;
endif;
endprocess;

-------------------------------
-- ContadorByte
process(clk, reset)
begin
if reset ='1' then
      indice <=207;
elsif clk'eventand clk='1' then
if CL_indice ='1' then
            indice <=207;
endif;
if DEC_indice ='1' then
        indice <= indice -8;
endif;
endif;
endprocess;

-------------------------------
-- controladorTimestamp
process(clk, reset)
begin

-------------------------------------------------------------------------
---
-------Paso del tiempo a su valor hexadecimal ascii correspondiente---
---
-------------------------------------------------------------------------
---

        trozo1 <= timestamp_tx/100000000;
        resto1  <= timestamp_tx -(trozo1*100000000);

        trozo2 <=(resto1/10000000);
        resto2  <= resto1 -(trozo2*10000000);

        trozo3 <= resto2 /1000000;
        resto3  <= resto2 -(trozo3*1000000);

        trozo4 <= resto3 /100000;
        resto4  <= resto3 -(trozo4*100000);

        trozo5 <= resto4 /10000;
        resto5  <= resto4 -(trozo5*10000);
```

```
        trozo6 <= resto5 /1000;
        resto6  <= resto5 -(trozo6*1000);

        trozo7 <= resto6 /100;
        resto7  <= resto6 -(trozo7*100);

        trozo8 <= resto7 /10;
        resto8  <= resto7 -(trozo8*10);

        trozo9 <= resto8;

-- PASAR CADA TROZO A SU VALOR HEXADECIMAL DEL ASCII PARA QUE LO PUEDA
REPRESENTAR LA APLICACION
case trozo1 is
when9=> digito1 <= X"39";
when8=> digito1 <= X"38";
when7=> digito1 <= X"37";
when6=> digito1 <= X"36";
when5=> digito1 <= X"35";
when4=> digito1 <= X"34";
when3=> digito1 <= X"33";
when2=> digito1 <= X"32";
when1=> digito1 <= X"31";
when0=> digito1 <= X"30";
endcase;
case trozo2 is
when9=> digito2 <= X"39";
when8=> digito2 <= X"38";
when7=> digito2 <= X"37";
when6=> digito2 <= X"36";
when5=> digito2 <= X"35";
when4=> digito2 <= X"34";
when3=> digito2 <= X"33";
when2=> digito2 <= X"32";
when1=> digito2 <= X"31";
when0=> digito2 <= X"30";
endcase;
case trozo3 is
when9=> digito3 <= X"39";
when8=> digito3 <= X"38";
when7=> digito3 <= X"37";
when6=> digito3 <= X"36";
when5=> digito3 <= X"35";
when4=> digito3 <= X"34";
when3=> digito3 <= X"33";
when2=> digito3 <= X"32";
when1=> digito3 <= X"31";
when0=> digito3 <= X"30";
endcase;
case trozo4 is
when9=> digito4 <= X"39";
when8=> digito4 <= X"38";
when7=> digito4 <= X"37";
when6=> digito4 <= X"36";
when5=> digito4 <= X"35";
when4=> digito4 <= X"34";
when3=> digito4 <= X"33";
when2=> digito4 <= X"32";
when1=> digito4 <= X"31";
when0=> digito4 <= X"30";
endcase;
```

```
case trozo5 is
when9=> digito5 <= X"39";
when8=> digito5 <= X"38";
when7=> digito5 <= X"37";
when6=> digito5 <= X"36";
when5=> digito5 <= X"35";
when4=> digito5 <= X"34";
when3=> digito5 <= X"33";
when2=> digito5 <= X"32";
when1=> digito5 <= X"31";
when0=> digito5 <= X"30";
endcase;
case trozo6 is
when9=> digito6 <= X"39";
when8=> digito6 <= X"38";
when7=> digito6 <= X"37";
when6=> digito6 <= X"36";
when5=> digito6 <= X"35";
when4=> digito6 <= X"34";
when3=> digito6 <= X"33";
when2=> digito6 <= X"32";
when1=> digito6 <= X"31";
when0=> digito6 <= X"30";
endcase;
case trozo7 is
when9=> digito7 <= X"39";
when8=> digito7 <= X"38";
when7=> digito7 <= X"37";
when6=> digito7 <= X"36";
when5=> digito7 <= X"35";
when4=> digito7 <= X"34";
when3=> digito7 <= X"33";
when2=> digito7 <= X"32";
when1=> digito7 <= X"31";
when0=> digito7 <= X"30";
endcase;
case trozo8 is
when9=> digito8 <= X"39";
when8=> digito8 <= X"38";
when7=> digito8 <= X"37";
when6=> digito8 <= X"36";
when5=> digito8 <= X"35";
when4=> digito8 <= X"34";
when3=> digito8 <= X"33";
when2=> digito8 <= X"32";
when1=> digito8 <= X"31";
when0=> digito8 <= X"30";
endcase;
case trozo9 is
when9=> digito9 <= X"39";
when8=> digito9 <= X"38";
when7=> digito9 <= X"37";
when6=> digito9 <= X"36";
when5=> digito9 <= X"35";
when4=> digito9 <= X"34";
when3=> digito9 <= X"33";
when2=> digito9 <= X"32";
when1=> digito9 <= X"31";
when0=> digito9 <= X"30";
endcase;
```

```
            timestamp_tx_ascii <=
digito1&digito2&digito3&digito4&digito5&digito6&digito7&digito8&digito
9;
endprocess;


-------------------------------
-- controlador t_res
process(clk, reset)
begin

-------------------------------------------------------------------------
---
-------Paso del tiempo a su valor hexadecimal ascii correspondiente---
---
-------------------------------------------------------------------------
---

        t_res_trozo1 <= t_res/10;
        t_res_resto1  <= t_res -(t_res_trozo1*10);

        t_res_trozo2 <= t_res_resto1;

-- PASAR CADA TROZO A SU VALOR HEXADECIMAL DEL ASCII PARA QUE LO PUEDA
REPRESENTAR LA APLICACION
case t_res_trozo1 is
when9=> t_res_digito1 <= X"39";
when8=> t_res_digito1 <= X"38";
when7=> t_res_digito1 <= X"37";
when6=> t_res_digito1 <= X"36";
when5=> t_res_digito1 <= X"35";
when4=> t_res_digito1 <= X"34";
when3=> t_res_digito1 <= X"33";
when2=> t_res_digito1 <= X"32";
when1=> t_res_digito1 <= X"31";
when0=> t_res_digito1 <= X"30";
endcase;
case t_res_trozo2 is
when9=> t_res_digito2 <= X"39";
when8=> t_res_digito2 <= X"38";
when7=> t_res_digito2 <= X"37";
when6=> t_res_digito2 <= X"36";
when5=> t_res_digito2 <= X"35";
when4=> t_res_digito2 <= X"34";
when3=> t_res_digito2 <= X"33";
when2=> t_res_digito2 <= X"32";
when1=> t_res_digito2 <= X"31";
when0=> t_res_digito2 <= X"30";
endcase;

t_res_ascii <= t_res_digito1&t_res_digito2;
endprocess;

-------------------------------
-- controlador res_act
process(clk, reset)
begin

-------------------------------------------------------------------------
---
```

```
-------Paso del tiempo a su valor hexadecimal ascii correspondiente---
---
-------------------------------------------------------------------------
---

        res_act_trozo1 <= res_act/10;
        res_act_resto1  <= res_act -(res_act_trozo1*10);

        res_act_trozo2 <= res_act_resto1;

-- PASAR CADA TROZO A SU VALOR HEXADECIMAL DEL ASCII PARA QUE LO PUEDA
REPRESENTAR LA APLICACION
case res_act_trozo1 is
when9=> res_act_digito1 <= X"39";
when8=> res_act_digito1 <= X"38";
when7=> res_act_digito1 <= X"37";
when6=> res_act_digito1 <= X"36";
when5=> res_act_digito1 <= X"35";
when4=> res_act_digito1 <= X"34";
when3=> res_act_digito1 <= X"33";
when2=> res_act_digito1 <= X"32";
when1=> res_act_digito1 <= X"31";
when0=> res_act_digito1 <= X"30";
endcase;
case res_act_trozo2 is
when9=> res_act_digito2 <= X"39";
when8=> res_act_digito2 <= X"38";
when7=> res_act_digito2 <= X"37";
when6=> res_act_digito2 <= X"36";
when5=> res_act_digito2 <= X"35";
when4=> res_act_digito2 <= X"34";
when3=> res_act_digito2 <= X"33";
when2=> res_act_digito2 <= X"32";
when1=> res_act_digito2 <= X"31";
when0=> res_act_digito2 <= X"30";
endcase;

            res_act_ascii <= res_act_digito1&res_act_digito2;
endprocess;

--------------------------------
-- controladorState
process(clk,reset)
begin
if activado = '1' then
          cmActivado <= X"59";
else
          cmActivado <= X"4E";
endif;

if intrusion = '1' then
       cmIntrusion <= X"59";
else
       cmIntrusion <= X"4E";
endif;

if alarma = '1' then
         cmAlarma <= X"59";
else
         cmAlarma <= X"4E";
endif;
```

```
endprocess;

--------------------------------
-- Envio_comando
process(clk,cmd,OK)
begin
case cmd is
WHEN"000"=>
if OK = '1' then
-- Nº de bytes, incluyendo los dos bytes del fin de línea y el retorno
de carro: 9
                    contByte <="01001";
-- +MK:,OK
                    comando_envio <=
X"2B4D4B3A2C4F4B0A0D0000000000000000000000000000000000";
else
-- Nº de bytes, incluyendo los dos bytes del fin de línea y el retorno
de carro: 12
                              contByte <="01100";
-- +MK:,ERROR
                              comando_envio <=
X"2B4D4B3A2C4552524F520A0D0000000000000000000000000000";
endif;

WHEN"001"=>
if OK = '1' then
-- Nº de bytes, incluyendo los dos bytes del fin de línea y el retorno
de carro: 12
                            contByte <="01100";
-- +V:<ASCII>,OK
                            comando_envio <=
X"2B563A"&firmware&X"2C4F4B0A0D0000000000000000000000000000";
else
-- Nº de bytes, incluyendo los dos bytes del fin de línea y el retorno
de carro: 15
                            contByte <="01111";
-- +V:<ASCII>,ERROR
                            comando_envio <=
X"2B563A"&firmware&X"2C4552524F520A0D0000000000000000000000";
endif;

WHEN"010"=>
if OK = '1' then
-- Nº de bytes, incluyendo los dos bytes del fin de línea y el retorno
de carro: 11
                            contByte <="01011";
-- +CONF:,OK
                            comando_envio <=
X"2B434F4E463A2C4F4B0A0D0000000000000000000000000000000000";
else
-- Nº de bytes, incluyendo los dos bytes del fin de línea y el retorno
de carro: 14
                            contByte <="01110";
-- +CONF:,ERROR
                            comando_envio <=
X"2B434F4E463A2C4552524F520A0D0000000000000000000000000000";
endif;

WHEN"011"=>
if OK = '1' then
```

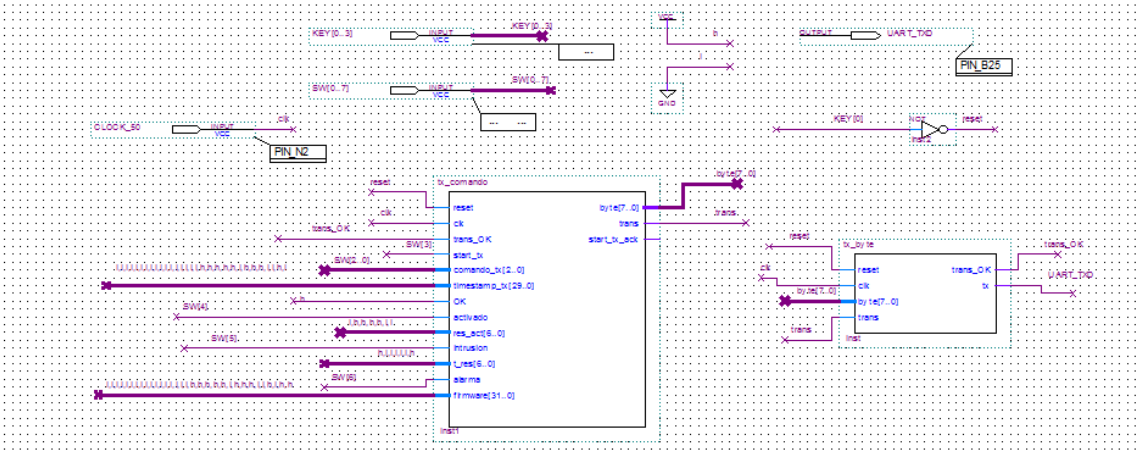
```
-- Nº de bytes, incluyendo los dos bytes del fin de línea y el retorno
de carro: 11
                                contByte <="01011";
-- +PASS:,OK
                                comando_envio <=
X"2B504153533A2C4F4B0A0D0000000000000000000000000000";
else
-- Nº de bytes, incluyendo los dos bytes del fin de línea y el retorno
de carro: 14
                                contByte <="01110";
-- +PASS:,ERROR
                                comando_envio <=
X"2B504153533A2C4552524F520A0D0000000000000000000000";
endif;

WHEN"100"=>
if OK = '1' then
-- Nº de bytes, incluyendo los dos bytes del fin de línea y el retorno
de carro: 10
                                contByte <="01010";
-- +ACT:,OK
                                comando_envio <=
X"2B4143543A2C4F4B0A0D0000000000000000000000000000";
else
-- Nº de bytes, incluyendo los dos bytes del fin de línea y el retorno
de carro: 13
                                contByte <="01101";
-- +ACT:,ERROR
                                comando_envio <=
X"2B4143543A2C4552524F520A0D0000000000000000000000";
endif;

WHEN"101"=>
if OK = '1' then
-- Nº de bytes, incluyendo los dos bytes del fin de línea y el retorno
de carro: 17
                                 contByte <="10001";
-- +D&H:<timestamp>,OK
                               comando_envio <=
X"2B4426483A"&timestamp_tx_ascii&X"2C4F4B0A0D00000000000000";
else
-- Nº de bytes, incluyendo los dos bytes del fin de línea y el retorno
de carro: 20
                                 contByte <="10100";
-- +D&H:<timestamp>,ERROR
                                 comando_envio <=
X"2B4426483A"&timestamp_tx_ascii&X"2C4552524F520A0D00000000";
endif;

WHEN"110"=>
if OK = '1' then
-- Nº de bytes, incluyendo los dos bytes del fin de línea y el retorno
de carro: 23
                                 contByte <="10111";
-- +STATE:<ACTIVADO YES O NO>, <TIEMPO RESTANTE PARA LA ACTIVACION DE
LA ALARMA>,
-- <INTRUSION Y O N>, <TIEMPO RESTANTE PARA SEÑAL DE ALARMA>, <ALARMA
Y O N>,OK
                                comando_envio <=
X"2B53544154453A"&cmActivado&X"2C"&res_act_ascii&X"2C"&cmIntrusion&X"2
C"&t_res_ascii&X"2C"&cmalarma&X"2C4F4B0A0D000000";
```

```
else
-- Nº de bytes, incluyendo los dos bytes del fin de línea y el retorno
de carro: 26
                            contByte <="11010";
-- +STATE:<ACTIVADO YES O NO>, <TIEMPO RESTANTE PARA LA ACTIVACION DE
LA ALARMA>,
-- <INTRUSION Y O N>, <TIEMPO RESTANTE PARA SEÑAL DE ALARMA>, <ALARMA
Y O N>,ERROR
                                comando_envio <=
X"2B53544154453A"&cmActivado&X"2C"&res_act_ascii&X"2C"&cmIntrusion&X"2
C"&t_res_ascii&X"2C"&cmAlarma&X"2C4552524F520A0D";
endif;

WHEN"111"=>
if OK = '1' then

-- Nº de bytes, incluyendo los dos bytes del fin de línea y el retorno
de carro: 12
                            contByte <="01100";
-- +DEACT:,OK
                            comando_envio <=
X"2B44454143543A2C4F4B0A0D00000000000000000000000000000000";
else
-- Nº de bytes, incluyendo los dos bytes del fin de línea y el retorno
de carro: 15
                            contByte <="01111";
-- +DEACT:,ERROR
                            comando_envio <=
X"2B44454143543A2C4552524F520A0D0000000000000000000000000";
endif;
WHENOTHERS=>
                contByte <="11111";
                comando_envio <=
X"00000000000000000000000000000000000000000000000000000000";
ENDCASE;
endprocess;

byte <=  comando_envio(indice downto indice -7);

END a;
```

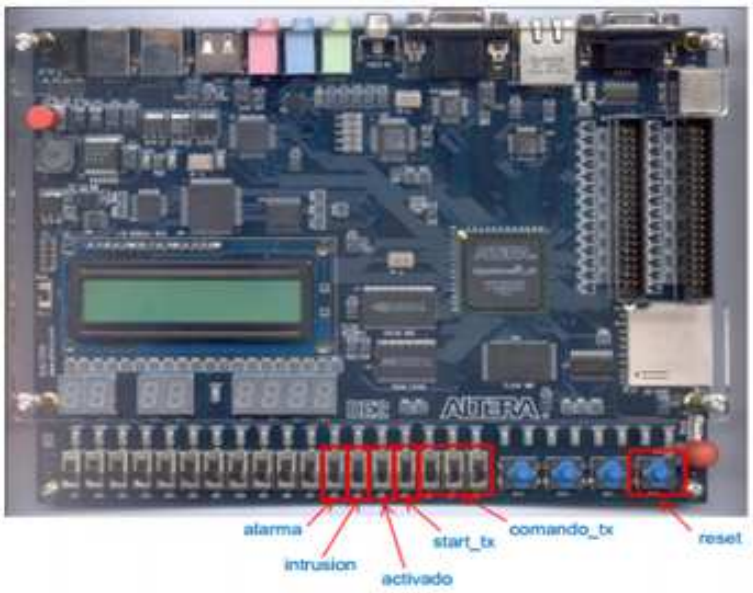
e. Simulación

La comprobación de este módulo se basó en varias pruebas realizadas con el programa Quartus y la propia placa DE2 de Altera.

Para está realización de las pruebas, conectamos este módulo realizado, tx_comando, y el módulo anteriormente diseñado, tx_byte, y los interconectamos entre sí. He aquí el diseño creado para testear el tx_comando en la herramienta Quartus.



Como se puede apreciar en la imagen, conectamos varias entradas del módulo tx_comando a ciertos pines específicos de la FPGA que están directamente conectados con varios componentes de la placa, como son los switches, los botones, el reloj de 50MHz y el puerto RS-232. La relación entre los nombres de estos componentes y sus respectivos pines de la FPGA la hicimos mediante el fichero de asignaciones proporcionado llamado *D2_pin_assignments.* Estos son los componentes que utilizamos para simular la entrada de varias señales a nuestro módulo:



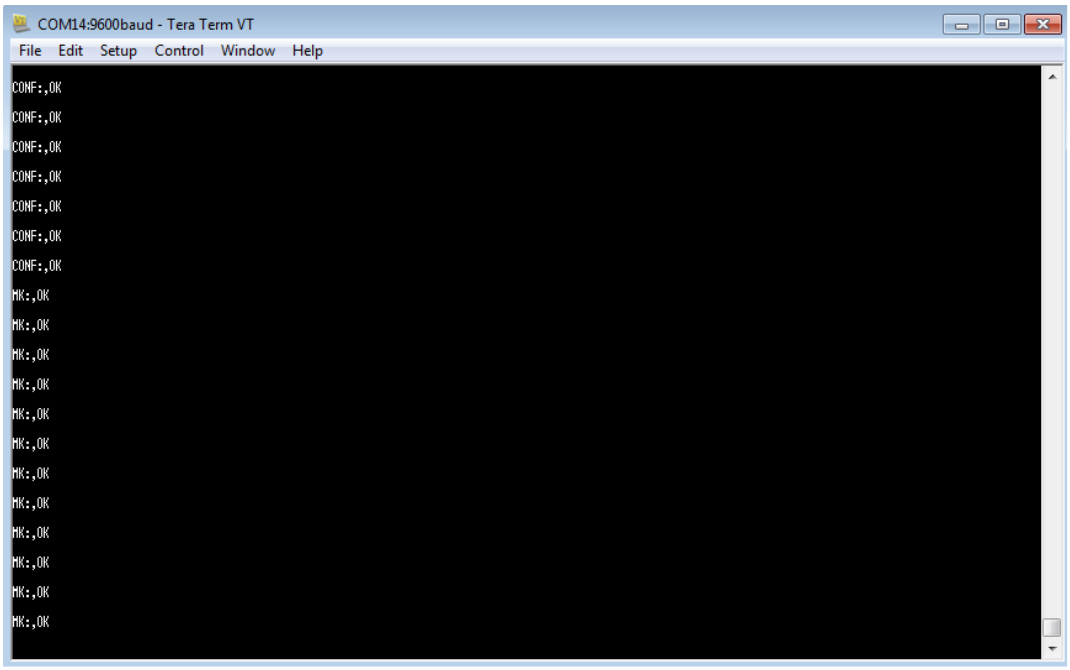
Otras señales:

La señal tx del módulo tx_byte la conectamos directamente al pin de la FPGA que está conectado con el puerto RS-232.

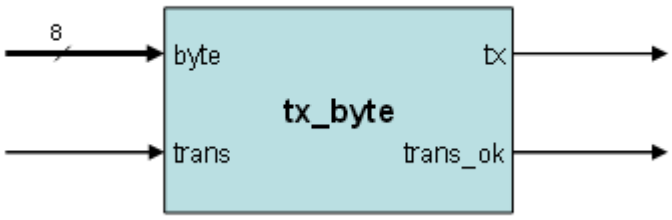
La señal clk está conectada directamente con el pin de la FPGA que está conectado al reloj de 50MHz de la placa.

Una vez creado el diseño entero, guardamos en la memoria de la FPGA el diseño creado y nos pusimos a probarlo, para ello pusimos en marcha el hyperterminal, configuramos para que la entrada fuera recibida por el puerto serie al cual estaba conectada la placa. Entonces, con los diferentes switches y botones que habíamos configurado conseguimos ver los resultados y pudimos así testear el módulo.

Vista desde el hyperterminal de las pruebas realizadas



5.8. TX_BYTE



a. Descripción

Al contrario que rx_byte, este módulo lo que hace es recibir un byte y transmitir ese byte, bit a bit a la aplicación de Mirakonta. Los byte que trasmitirá por la línea serie de salida serán caracteres correspondientes a un comando con o sin sus parámetros (Según el comando).

Señales de entrada:

- **byte:** Dato de 8 bits a transmitir.
- **trans:** Indica que debe comenzar la transmisión

Señales de salida:

- **tx:** Dato de 8 bits a transmitir por la línea serie de salida
- **trans_ok:** Indica que se ha terminado la transmisión. La señal se mantiene en '1' hasta que la entrada trans se ponga a '0' y hasta que no sucede esto no puede iniciarse una nueva transmisión.

b. Unidad de Control

Primeramente estaremos en el estado E0, en el cual activaremos las señales putbit y bit, que serán para enviar un 1 por el puerto serie. En este estado nos quedaremos esperando hasta que la señal trans este activada la cual nos enviará al estado E1, sino permaneceremos en el mismo estado E0.

En el estado E1 activaremos las señales ld_byte, putbit, cl_tmp y cl_bits. Se resetean el tiempo y los el contador de bits enviados, se sigue poniendo un 1 en el puerto serie y se carga el byte a enviar. De este estado pasaremos incondicionalmente al estado E2.

En el estado E2 activaremos las señales putbit e inc_tmp, la cual aumenta el tiempo de espera (hasta contar los 5208 ciclos). Nos mantendremos en este estado si la señal fin_tmp esta desactivada, en el otro caso pasaremos al estado E3.

En el estado E3 activaremos al señal cl_tmp para resetear el tiempo y pasaremos incondicionalmente al estado E4.

En el siguiente estado, el estado E4, se pone el primer bit del byte en el puerto serie y se hace una espera con la señal fin_tmp. Al terminar pasaremos al estado E5.

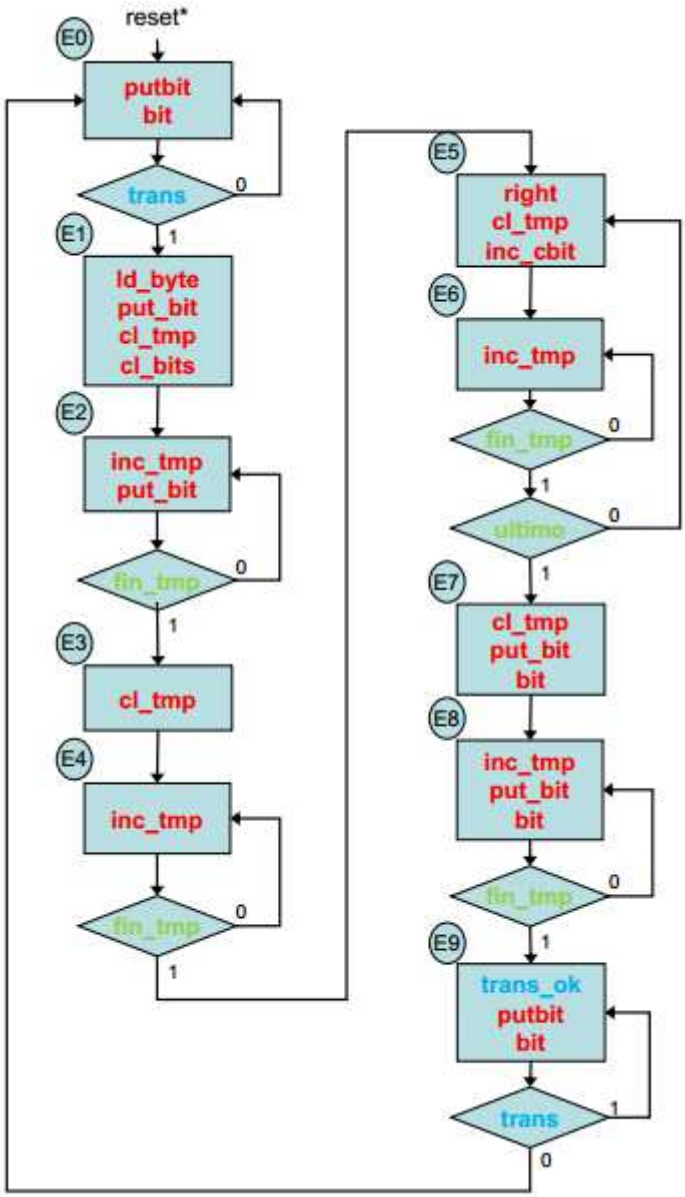
En el estado E5 se activarán las señales rigth (para desplazar el byte y enviar un nuevo bit por el puerto serie), cl_tmp e inc_cbit que aumentará el número de bits enviados. Posteriormente, pasaremos incondicionalmente al estado E6.

En el estado E6, activaremos al señal inc_tmp para incrementar el tiempo y haremos una espera condicional a la señal fin_tmp. Una vez activada esta señal, si la señal ultimo esta activada pasaremos al estado E7, sino pasaremos al estado E6 para volver a enviar un nuevo bit.

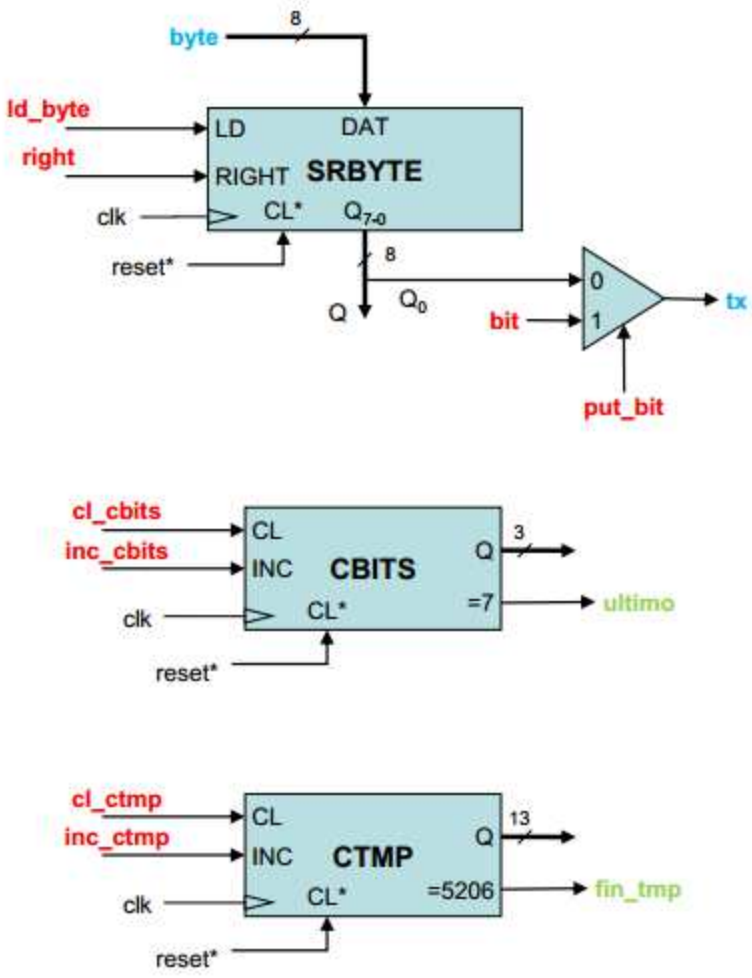
En el estado E7, activaremos la señales cl_tmp, putbit y bit para resetear el tiempo y poner un 1 por el puerto serie con las dos últimas. Pasaremos incondicionalmente de este estado al E8.

En este penúltimo estado, el E8, se activarán las señales inc_tmp, putbit y bit para seguir poniendo un 1 los ciclos necesarios (5208) realizando una espera activa sobre la señal fin_tmp. Terminada la espere pasaremos al estado E9.

En este último estado, el estado E9 activaremos las señales putbit, bit y trans_ok para decirle al tx_comando que se ha enviado el byte correctamente por el puerto serie. En este estado, permaneceremos bloqueados hasta que la señal trans se desactive volviendo así al primer estado E0 preparados para la transmisión de un nuevo byte.


reset*
E0
putbit
bit
trans
0
1
E1
ld_byte
put_bit
cl_tmp
cl_bits
E2
inc_tmp
put_bit
fin_tmp
0
1
E3
cl_tmp
E4
inc_tmp
fin_tmp
0
1
E5
right
cl_tmp
inc_cbit
E6
inc_tmp
fin_tmp
0
1
ultimo
0
1
E7
cl_tmp
put_bit
bit
E8
inc_tmp
put_bit
bit
fin_tmp
0
1
E9
trans_ok
putbit
bit
trans
1
0

c. Unidad de Proceso



Para contar Tbit contamos 104166ns/20ns=5208 ciclos de reloj (50 MHz)

d. Código VHDL

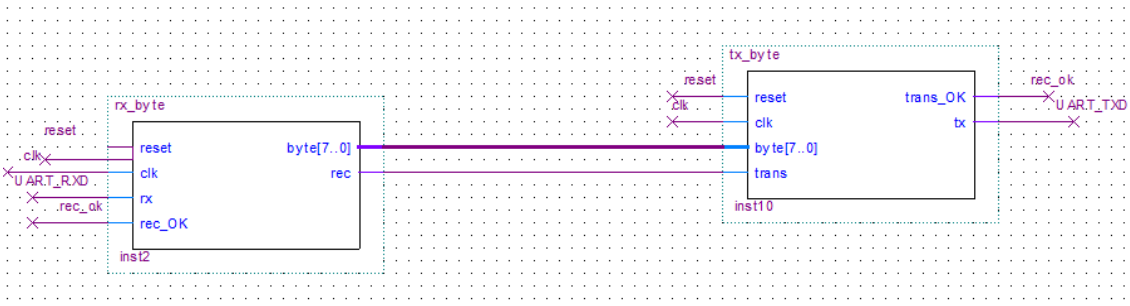
```
libraryieee;
useieee.std_logic_1164.all;
useieee.numeric_std.all;

entity tx_byte IS
port
(
    reset, clk     :instd_logic;
      byte         :inunsigned(7downto0);
    trans    :instd_logic;
    trans_OK    :outstd_logic;
    tx          :outstd_logic
);
end tx_byte;

architecture a of tx_byte is

-- señales internas de UP
signal Qcbits             :unsigned(2downto0);
signal Qctmp                  :unsigned(12downto0);
signal Qbyte                  :unsigned(7downto0);
-- estados presentes y siguiente de UC
type estado is(e0,e1,e2,e3,e4,e5,e6,e7,e8,e9);
signal ep,es : estado;
-- señales UC->UP
signalbit, putbit, ld_byte, cl_cbits, cl_ctmp, INC_ctmp,right,
INC_cbits :std_logic;
-- señales UP->UC
signal fintmp,ultimo :std_logic;
begin
--====================================
-- Unidad de Control
--====================================
----------------------------------------
-- Transicion de estados
process(ep, trans, fintmp, ultimo)
begin
case ep is
when e0 =>
if trans = '1' then es <= e1;
else  es <= e0;
endif;
when e1 =>
         es <= e2;
when e2 =>
if fintmp = '1' then es <= e3;
else  es <= e2;
endif;
when e3 =>
        es<=e4;
when e4 =>
if fintmp = '1' then es <= e5;
else  es <= e4;
endif;
when e5 =>
        es<=e6;
when e6 =>
if fintmp = '1' then
if ultimo = '1' then es <= e7;
```

```
else es<=e5;
endif;
else  es <= e6;
endif;
when e7 =>
        es<=e8;
when e8 =>
if fintmp = '1' then es <= e9;
else  es <= e8;
endif;
when e9 =>
if trans='1' then
es<=e9;
else es<=e0;
endif;
endcase;

endprocess;
-------------------------------------
-- REGISTRO de estados
process(clk, reset)
begin
if(reset='1')then
         ep <= e0;
elsif(clk'EVENTand clk='1')then
         ep <= es;
endif;
endprocess;
-------------------------------------
-- salidas de la unidad de control
bit<= '1' when(ep = e0 or ep=e7 or ep=e8 or ep=e9)else '0';
  putbit <= '1'  when(ep = e0 or ep = e1 or ep=e2 or ep=e7 or ep=e8 or
ep=e9)else '0';
  ld_byte <= '1' when ep = e1 else '0';
  cl_cbits <= '1' when ep = e1 else '0';
  cl_ctmp <= '1'  when(ep = e1 or ep=e3 or ep=e5 or ep=e7)else '0';
  INC_ctmp <= '1'  when(ep = e2 or ep=e4 or ep=e6 or ep=e8)else '0';
right<= '1'  when ep = e5 else '0';
  trans_OK <= '1' when ep = e9 else '0';
  INC_cbits <= '1' when ep = e5 else '0';

--====================================
-- Unidad de Proceso
--====================================
--------------------------------
-- registro desplazamiento
process(clk, reset)
begin
if reset = '1' then
      Qbyte <="00000000";
elsif clk'eventand clk='1' then
if ld_byte = '1' then
        Qbyte <= byte;
elsifright= '1' then
           Qbyte(6downto0)<= Qbyte(7downto1);
endif;
endif;
endprocess;
--------------------------------
-- MUX
  tx <=bitwhen putbit='1' else Qbyte(0);
```

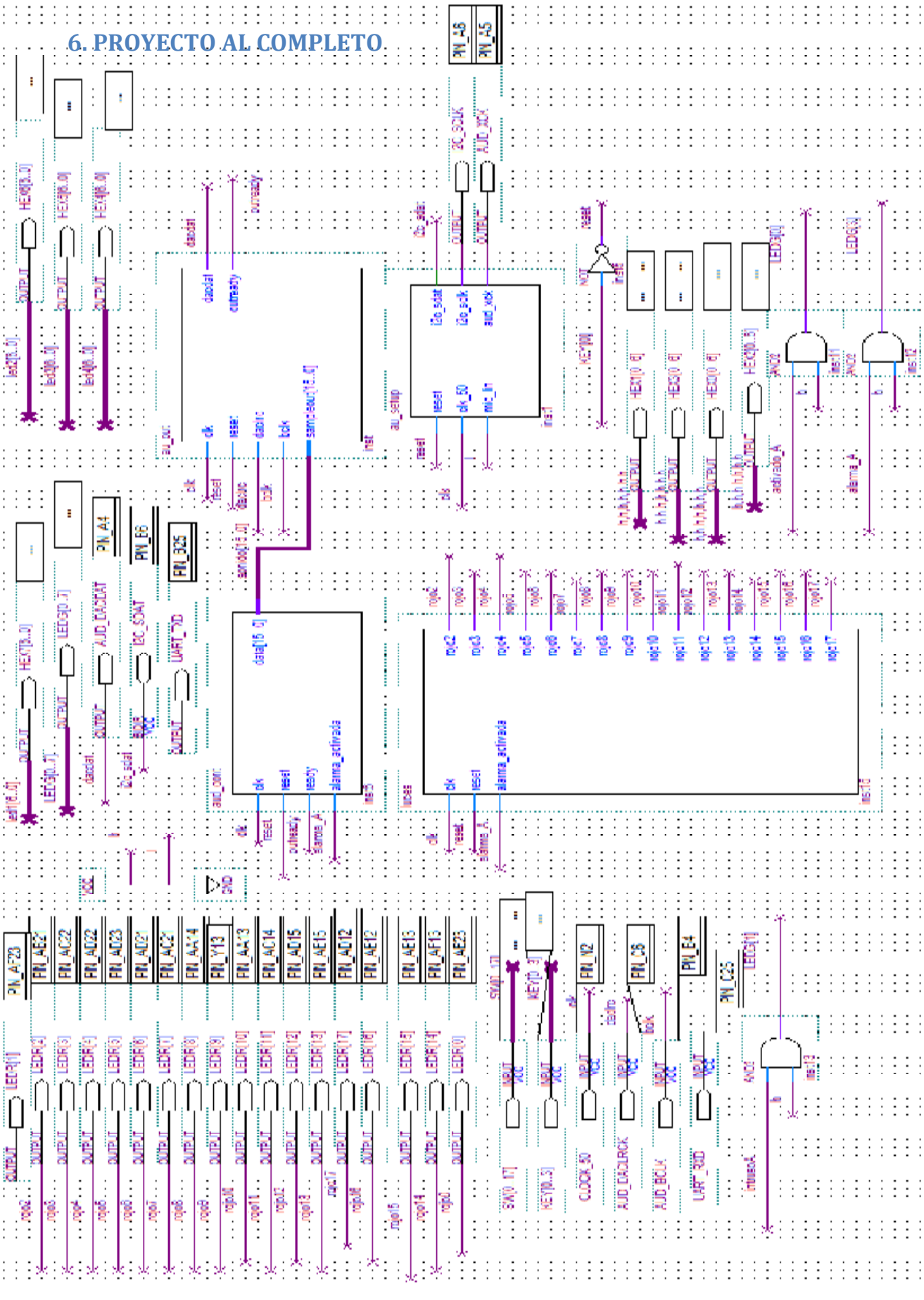
```
-------------------------------
-- CBITS
process(clk, reset)
begin
if reset = '1' then
      Qcbits <="000";
elsif clk'eventand clk='1' then
if cl_cbits = '1' then
          Qcbits <="000";
elsif INC_cbits = '1' then
              Qcbits <= Qcbits+1;
endif;
endif;
endprocess;

  ultimo <= '1' when Qcbits="111"else '0';
-------------------------------
-- CTMP
process(clk, reset)
begin
if reset = '1' then
      Qctmp <="0000000000000";
elsif clk'eventand clk='1' then
if cl_ctmp = '1' then
          Qctmp <="0000000000000";
endif;
if(INC_ctmp='1' and Qctmp<"1010001011000")then
              Qctmp <= Qctmp+1;
endif;
endif;
endprocess;


 fintmp <= '1' when Qctmp="1010001011000"else '0';

END a;
```

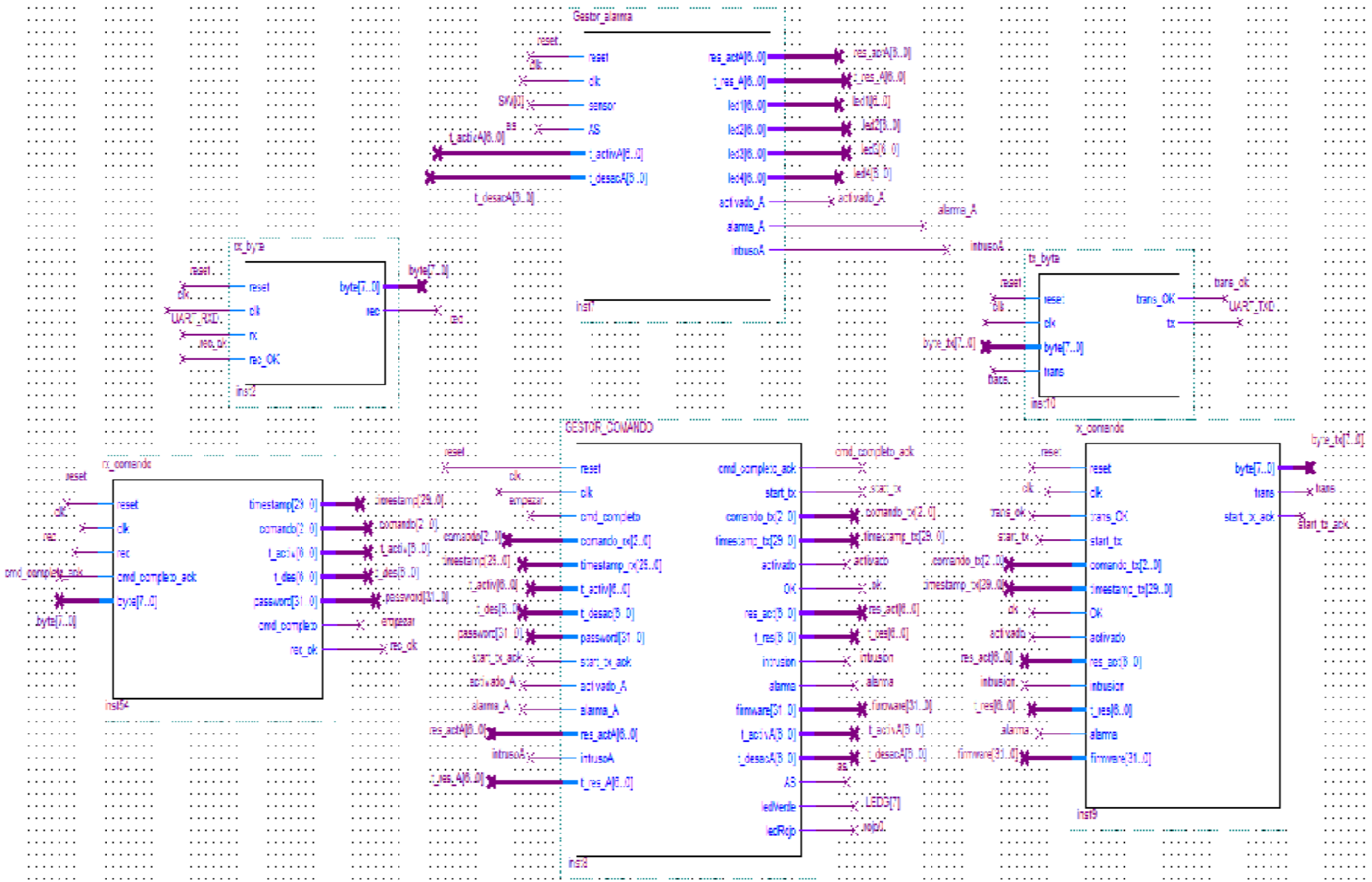
e. Simulación

Este módulo lo probamos en clase directamente con el Quartus utilizando el hyperterminal y el módulo rx_byte directamente, la prueba consistía en juntar los dos módulos y escribir en carácter por el hyperterminal y esperar la respuesta de la placa con el mismo carácter en el hyperterminal.



6. PROYECTO AL COMPLETO




Gestor_alarma
reset
clk
sensor
AS
t_activA[8..0]
t_desacA[3..0]
res_actA[8..0]
t_res_A[8..0]
led1[8..0]
led2[8..0]
led3[8..0]
led4[8..0]
activado_A
alarma_A
intrusoA
inst7
rx_byte
reset
clk
rx
rec_OK
byte[7..0]
rec
UART_RXD
inst2
tx_byte
reset
clk
byte[7..0]
trans
trans_OK
tx
trans_ok
UART_TXD
byte_tx[7..0]
inst10
GESTOR_COMANDO
reset
clk
empezar
cmd_completo
comando_rx[2..0]
timestamp_rx[25..0]
t_activ[8..0]
t_des[3..0]
password[31..0]
start_rx_ack
activado_A
alarma_A
res_actA[8..0]
intrusoA
t_res_A[8..0]
cmd_completo_ack
start_tx
comando_tx[2..0]
timestamp_tx[29..0]
activado
OK
res_act[8..0]
t_res[8..0]
intrusion
alarma
firmware[31..0]
t_activA[3..0]
t_desacA[3..0]
AS
LEDS[7]
inst8
rx_comando
reset
clk
rec
cmd_completo_ack
byte[7..0]
timestamp[29..0]
comando[2..0]
t_activ[8..0]
t_des[3..0]
password[31..0]
empezar
rec_ok
inst54
tx_comando
reset
clk
trans_ok
start_tx
comando_tx[2..0]
timestamp_tx[29..0]
OK
activado
res_act[8..0]
intrusion
t_res[8..0]
alarma
firmware[31..0]
byte[7..0]
trans
start_tx_ack
inst9

DCSD

7. CONCLUSIONES Y PROBLEMAS

Tanto los objetivos principales de la asignatura como los del proyecto los hemos cumplido satisfactoriamente, no obstante también hemos dedicado parte de nuestro tiempo para intentar mejorar aún más el sistema de alarma por lo que hemos añadido dos pequeñas funcionalidades mas.

La primera mejora ha sido a la hora de activarse la alarma y consiste en hacer un efecto visual con los leds rojos y para ello hemos utilizado un modulo extra.

La segunda y última mejora es la incorporación de sonido a la alarma, cuando esta se activa por la salida de audio transmitimos un pitido, y para esto hemos utilizado 3 módulos, uno construido por nosotros y los otros dos nos los proporcionaron los profesores.

En cuanto a los módulos extra, también cabe mencionar que nos costó comprender el funcionamiento de los módulos que intervenían en el funcionamiento del sonido.

A medida que se ha ido desarrollando el proyecto nos hemos encontrado diversos problemas de los que destacaremos los dos que consideramos más importantes.

El handshake entre los módulos no nos quedo muy claro desde un principio, por eso mismo a la hora de diseñar los algoritmos de los módulos no supimos hacer bien el handshake y estuvimos bastante tiempo parados intentando detectar el problema.

Otro gran problema que tuvimos fue a la hora de diseñar el gestor del comando, que al principio estaba unido con el gestor de alarma en un único modulo, debido a su complejidad y a nuestros escasos conocimientos al principio, no supimos como diseñar este modulo por lo que transmitimos este problema a los profesores y decidimos separarlo en dos módulos, **gestor comando**, encargado de las cosas relacionadas con los comandos y el **gestor alarma**, encargado de todo lo relacionado con la alarma.

Por último, ha sido un proyecto que nos ha gustado y por ello le hemos dedicado más tiempo para poder añadir estos dos módulos extra.

8. POSIBLES MEJORAS

En las posibles mejoras del proyecto, vemos un par de ideas que se podrían haber llevado a cabo:

- La utilización de la pantalla de texto de la placa para mostrar algún tipo de mensaje
- Introducir los sonidos de alarma mediante una tarjeta externa SD