Práctica Sistemas Digitales
Alarma domótica

David Guerra Martín
Roman Valls Guimerà
Daniel Clemente Laboreo

SDMI 22 E

Abril 2005

Índice

Planteamiento del problema

Se pide diseñar e implementar una alarma domestica que monitorize varias señales governadas por sensores tales como detección de fuego, fugas de agua e intrusiones. Dicha alarma se especifica en el enunciado en forma de dos bloques generales: el primero se ocupa de detectar un código numérico tecleado por el usuario mientras que el segundo se encarga de gestionar las acciones de la propia alarma.

Se ha decidido partir el problema en varios sub-bloques para disponer de un diseño modular más limpio y mantenible facilitando así las tareas de depuración y mantenibilidad. Dichos bloques se describen en el apartado “diagrama de bloques”.

Cabe destacar en este punto los estados o modos que soporta la alarma y que se activan/desactivan en función de claves introducidas por el usuario. Dichos modos son:

- AV: Clave de activación del modo M1.
- AA: Clave de activación del modo M2.
- AAT: Clave de atraco.
- DA: Clave de desactivación de la alarma.

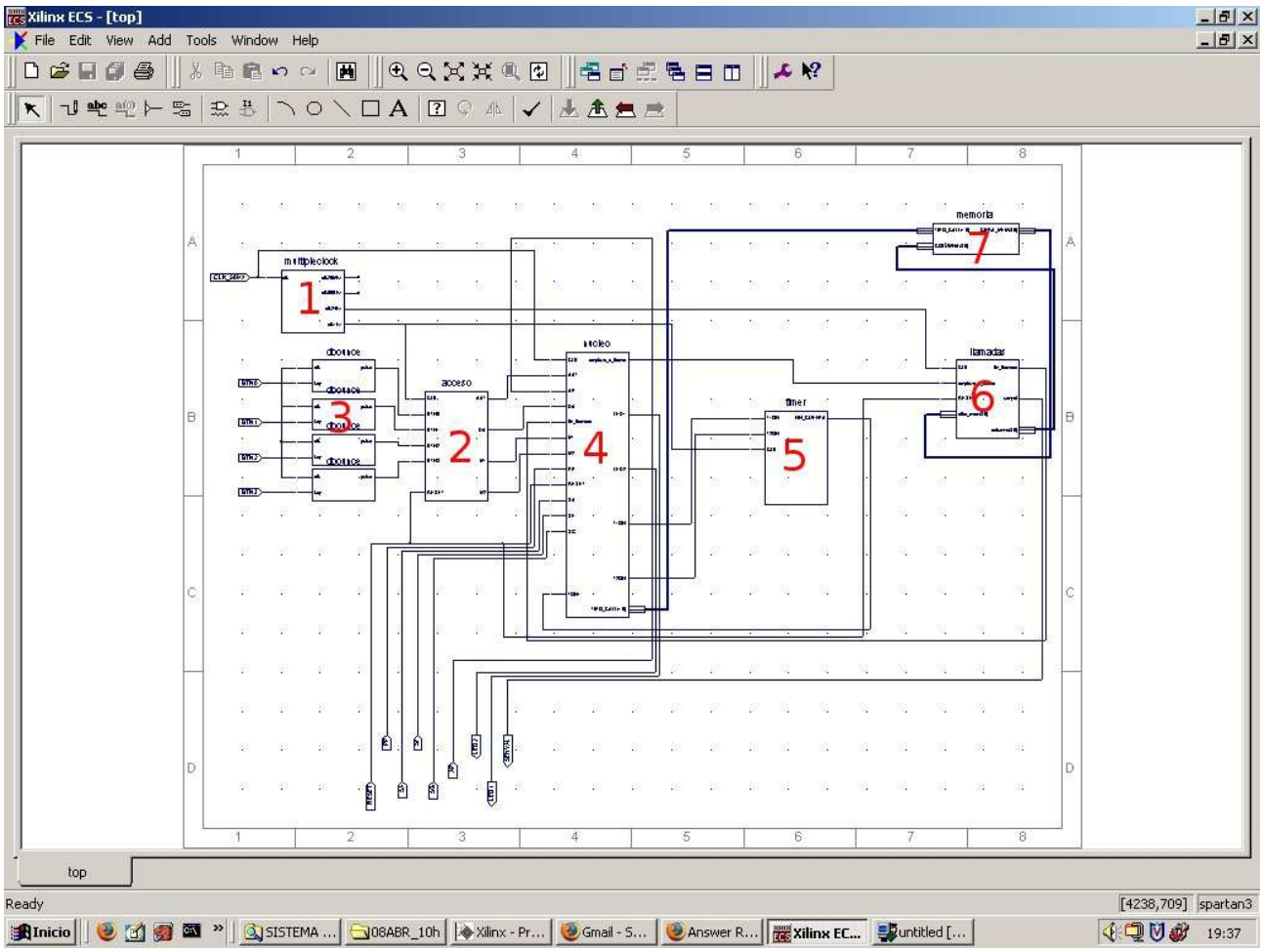
Por otra parte, las señales relevantes del sistema se enumeran y explican a continuación:

- PP: Indica si la puerta principal ha sido abierta
- AP: Vale “1” si alguna ventana o puerta diferente de la principal se encuentra abierta, 0 en caso contrario.
- SF: Se activa en caso de detectarse fuego.
- SG: Avisa sobre la presencia de una fuga de gas.
- SA: Informa en caso de detectarse un escape de agua.

El sistema debe ser capaz de gestionar una serie de situaciones que se especifican claramente en el enunciado. También se detalla en el enunciado el cómo realizar llamadas telefónicas una vez detectada una de las situaciones antes citadas.

Diagrama de bloques

La captura de pantalla siguiente muestra el “top” schematic del proyecto, con todos los módulos interconectados y los signals de entrada/salida mapeados por las contraints.



Enumeramos seguidamente una breve descripción de las funciones de cada módulo, así como el fichero de constraints resultante, cada módulo se verá com más detalle en la sección “Bloques usados”:

1) **Reloj de múltiples frecuencias:** El circuito dispone de necesidades variables en cuanto a tiempos de reloj. Este módulo proporciona 4 frecuencias de reloj distintas a partir de la que facilita la placa (50Mhz).
2) **Detección de código:** Sirve para permitir o denegar las claves de activación/desactivación de la alarma.
3) **Anti-rebotes:** Proporcionado ya implementado en la práctica, sirve para evitar señales residuales por malos contactos en los pulsadores.
4) **Núcleo:** Parte central del diseño, se encarga como se ha dicho antes, de toda la lógica de estados inherente de la alarma (control de modos, activación de senyales de alarma, activación de llamadas, etc..).
5) **Temporizador:** Hace posible cumplir con los requistos del enunciado referentes a esperas para activar/desactivar estados en la alarma (T1 & T2).
6) **Llamadas:** Ejecuta el tren de pulsos para realizar las llamadas según el tipo de alarma que ha sido activado, previo direccionado del número que se encuentra en el módulo memoria.
7) **Memoria:** Almacena los números de teléfono a los que llamar en caso de atraco, fuego, fuga de gas o de agua.

Elementos físicos de la placa (constraint.ucf)

El fichero de contraints (constraint.ucf) asocia las conexiones físicas de la placa Spartan3 con los nombres lógicos de las entradas y salidas. Después de cambiar alguno de estos alias y comentar los que no necesitábamos, queda de la siguiente forma:

```
#####################################
# Constraint file for Spartan3 Board #
#####################################
#--------------------#
# Seven Segments
#--------------------#
NET "SEL3"   LOC = "E13";   # Select one of the four 7 segments
NET "SEL2"   LOC = "F14";   #
NET "SEL1"   LOC = "G14";   #
NET "SEL0"   LOC = "D14";   #

NET "A"      LOC = "E14";   #            A
NET "B"      LOC = "G13";   #           ----
NET "C"      LOC = "N15";   #     F -> |    | <- B
NET "D"      LOC = "P15";   #          | G  |
NET "E"      LOC = "R16";   #           ----
NET "F"      LOC = "F13";   #     E -> |    | <- C
NET "G"      LOC = "N16";   #          | D  |
NET "DP"     LOC = "P16";   #           ----  . <- DP

#---------------------#
# Switches
#---------------------#
#NET "SW0"  LOC = "F12";
#NET "SW1"  LOC = "G12";
#NET "SW2"  LOC = "H14";
#NET "SW3"  LOC = "H13";
#NET "SW4"  LOC = "J14";
#NET "SW5"  LOC = "J13";
#NET "SW6"  LOC = "K14";
#NET "SW7"  LOC = "K13";

#---------------------#
# Push Buttons
#---------------------#
NET "BTN0" LOC = "M13";
NET "BTN1" LOC = "M14";
NET "BTN2" LOC = "L13";
NET "BTN3" LOC = "L14";   # User Reset button

#---------------------#
# LEDs
#---------------------#
#NET "LED7" LOC = "P11";
#NET "LED6" LOC = "P12";
#NET "LED5" LOC = "N12";
#NET "LED4" LOC = "P13";
#NET "LED3" LOC = "N14";
NET "LED2" LOC = "L12";
NET "LED1" LOC = "P14";
#NET "LED0" LOC = "K12";

#---------------------#
# Clocks
#---------------------#
NET "CLK"  LOC = "T9";    # 50Mhz clock
NET "SCLK" LOC = "D9";    # Socket clock
```

Descripción general

Para poder utilizar la alarma, el usuario ha de seleccionar mediante teclado el modo de trabajo en el que quiere que funcione (M1 o M2); si no, aunque introduzca los códigos de desactivar alarma (DA) o antiatraco (AAT) no se activará ninguna señal de alarma ni se realizará ninguna llamada telefónica.

El sistema se pone en marcha mediante pulsadores; la entrada que llega desde los pulsadores pasa por una batería de filtros antirebotes para evitar una detección errónea de las pulsaciones. Una vez filtrada la entrada se detecta el código introducido mediante un bloque de estados (ACCESO). En caso de ser un código correcto, se informa a otro bloque de estados (NUCLEO) que, en función del código introducido funcionará en uno de dos posibles modos de trabajo:

- En el modo de trabajo M1 la alarma controlará incidencias de tipo no intrusivo mediante sensores externos; en caso de ocurrir un evento (ya sea una fuga de gas, agua o un incendio) el sistema activará una alarma (AL1) que avisa al usuario. Además el sistema marcará el número de teléfono correspondiente a la incidencia mediante un tren de pulsos.
- En el modo de trabajo M2 la alarma controla todo tipo de incidencias, ya que se supone que el local esta vacío. Incluye todas las del modo M1.

En el momento de la activar el modo M2 el sistema espera durante una temporización (T1ON) en la que no se controla el sensor de apertura de puerta principal (PP), esto permite al usuario abandonar el local; por otro lado, si la alarma está funcionando en modo M2 y un usuario entra en el local, éste dispondrá de un tiempo (T2ON) para desactivarla, ya que sinó se pondrá en marcha la señal (AL2) que indica que alguien que no conoce el código de desactivación ha entrado; como puede darse el caso de que el usuario se equivoque al introducir el código de desactivación, el marcado de la llamada por robo no se produce (pero sí la señal AL2) hasta que el usuario introduce el código AAT.

En el hipotético caso de que el usuario que conoce la clave de DA fuera forzado a desactivar la alarma, éste puede intoducir el código AAT en cualquier momento (incluso si está activado ya AL2), con esto parará el AL2 pero se generará una llamada telefónica (NROB).

Si mientras el sistema está trabajando en cualquiera de los dos modos ocurre una indicencia (fuga de gas por ejemplo), se detecta el problema y se activa la marcación del número de teléfono correspondiente mediante un bloque de estados (LLAMADAS) que va generando pulsos en función de un número de teléfono que se encuentra almacenado en una memoria direccionada por los bloques LLAMADAS y NUCLEO.

Bloques usados

A continuación se detalla el proceso que hemos seguido para implementar cada bloque, de qué entradas/salidas disponen y por último que

funciones realizan en el circuito, como se comunican con otros bloques, etc:

Reloj de múltiples frecuencias

Este bloque alimenta a los demás bloques secuenciales con una entrada de clock. Además de la original de 50 MHz, tenemos salidas para 25 MHz, 50 kHz, 24 Hz y 1 Hz, que vienen directamente de determinados bits de un contador.

```
entity multipleclock is
    Port ( clk      : in std_logic;

           clk25Mhz : out std_logic;
                 clk50Khz : out std_logic;
                 clk24hz  : out std_logic;
                 clk1hz   : out std_logic);
end multipleclock;

architecture Behavioral of multipleclock is
        signal cnt1 : std_logic_vector(25 downto 0);
        signal cnt2 : std_logic_vector(26 downto 0);
        signal neg  : std_logic;
begin

        process(clk)
        begin
                if clk'event and clk = '0' then
                        cnt1 <= cnt1 + 1;
                        cnt2 <= cnt2 + 1;
                end if;
                if cnt1 = "10111110101111000010000000" then
                        cnt1 <= "00000000000000000000000000";
                        neg <= not neg;
                end if;
        end process;

        clk25Mhz <= cnt2(1);
        clk50Khz <= cnt2(10);
        clk24hz <= cnt2(21);
        clk1hz <= neg;

end Behavioral;
```

El núcleo de la alarma funciona a 50 MHz, porque es el que necesita mayor rapidez de respuesta y no perder ningún aviso.

En cambio, al temporizador (que tiene que medir tiempos en segundos) le va muy bien un reloj de 1 Hz, ya que cada ciclo durará aproximadamente un segundo. Por problemas con el programa en la salida de 1 Hz, también se puede poner a 24 Hz y contar 24 veces más.

El módulo de llamadas tiene que generar impulsos (un ciclo a 1 y otro a 0) continuamente, pero si lo hace muy rápido, el teléfono no conseguiría detectar los pulsos. 1 Hz funcionaría bien, pero al ser números de 9 cifras, puede que con algunos tardara mucho: en el caso peor (llamando al 999.999.999), son 10 impulsos para marcar un 9, o sea, 20 ciclos. Con los 5 de espera, 25, y para las 9 cifras, 225 ciclos, que a 1 Hz serían 225 segundos; demasiado lento. Por eso tendrá que funcionar a 24 Hz, por ejemplo (son 10 seg. en caso peor).

Pero es importante que la memoria trabaje más rápido que el módulo de llamadas para asegurar que los datos pedidos estarán preparados en el siguiente ciclo. No es un circuito secuencial (no recibe el clock); así que lo que

comprobamos es su latencia, que debería ser menor que los 1/24 = 42 ms. que dura un ciclo de llamadas. Si no fuera así, con escoger otra frecuencia se soluciona.

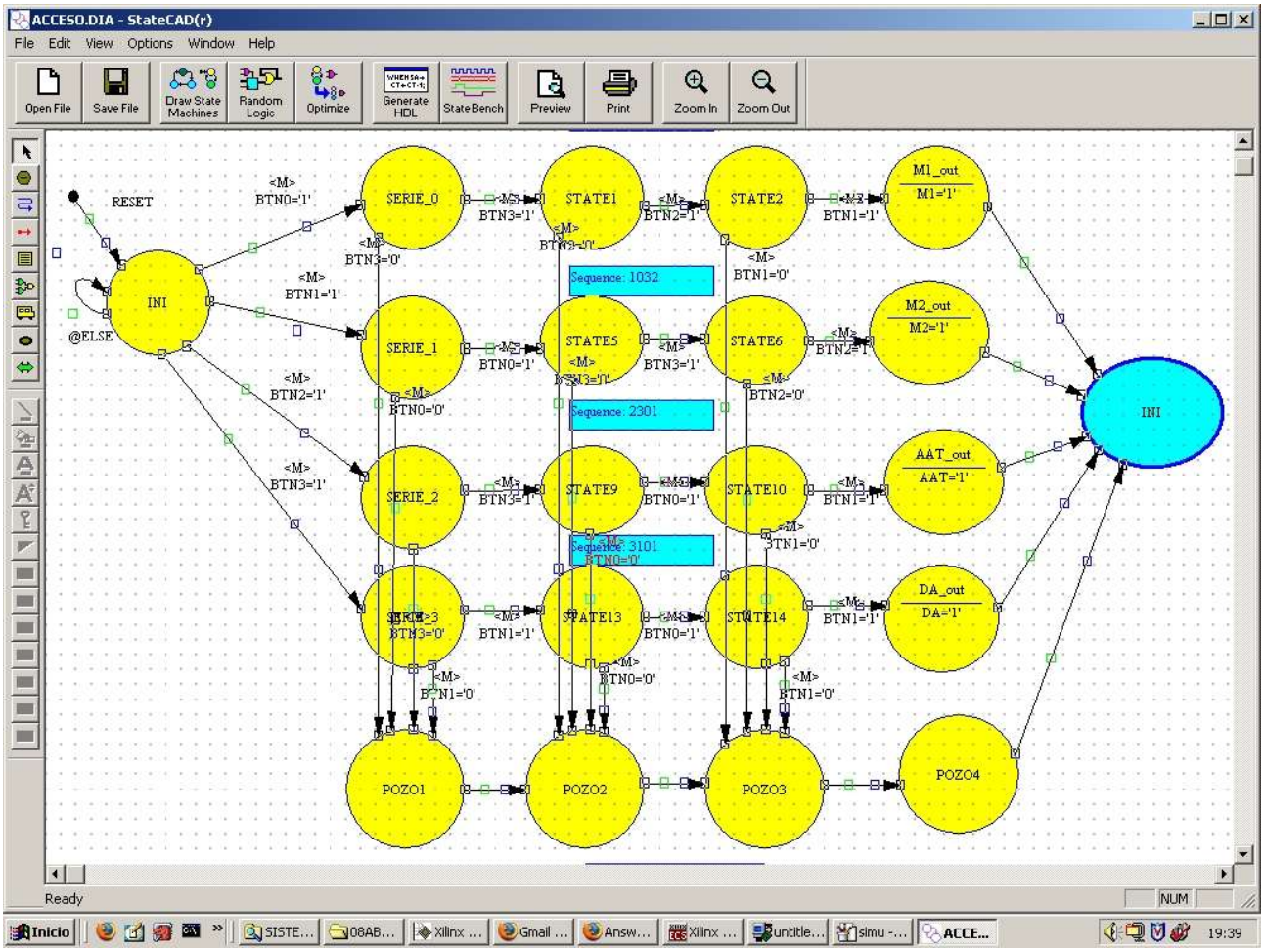
La detección del código que teclea el usuario se puede hacer a 24 Hz, porque parece suficiente. 1 Hz también funcionaría, pero el retardo entre apretar un pulsador y que sea detectado ya sería apreciable, y quizás molestaría. Para el anti-rebotes de detrás de cada pulsador, 24 comprobaciones por segundo también son apropiadas.

Detección de clave

Este módulo se encarga de activar el modo M1 & M2 mediante dos códigos diferentes especificados en las transiciones e indicados en los comentarios. También activa el modo antiatraco (AAT) y permite desactivar la alarma (DA).

De izquierda a derecha, el estado inicial espera la pulsación de uno de los cuatro botones de la spartan3. Cabe destacar que las entradas de los cuatro botones (de ahora en adelante BTN[0..3]) són mutuamente exclusivas, por tanto no hay lugar para ambiguedades en las transiciones.

Una vez pulsado uno de los BTN[0..3] disponibles, se pasa a comprobar cada una de las siguientes tres pulsaciones. En caso que el usuario introduzca uno de los códigos con éxito, se activará el output correspondiente, tal como se aprecia en los estados finales de la derecha (M1, M2, AAT o DA). Por el contrario, si el usuario introduce un dígito incorrecto para la secuencia o série de transiciones actual, se pasará automáticamente a los “estados pozo” de la parte inferior del grafo, que finalizan sin permitir ningún privilegio al usuario y vuelven finalmente al estado inicial INI.



Anti-rebotes

Ya implementado; su función es suavizar los cambios de estado que se generan en un pulsador cuando los cables entran en contacto, y en vez de tener una señal como 000001011011111, conseguir 0000000011111111 al apretar un botón.

```
entity dbounce is
    Port ( clk : in std_logic;
           key : in std_logic;
           pulse : out std_logic);
end dbounce;

architecture Behavioral of dbounce is
   signal cnt : std_logic_vector(7 downto 0);
begin

   process(clk)
   begin
      if key = '1' then
            cnt <= "00000000";
      elsif (clk'event and clk = '1') then
            if (cnt /= "11111111") then
               cnt <= cnt + 1;
         end if;
      end if;
         if (cnt > "10000000") and (key = '0') then
            pulse <= '0';
      else
            pulse <= '1';
      end if;
```

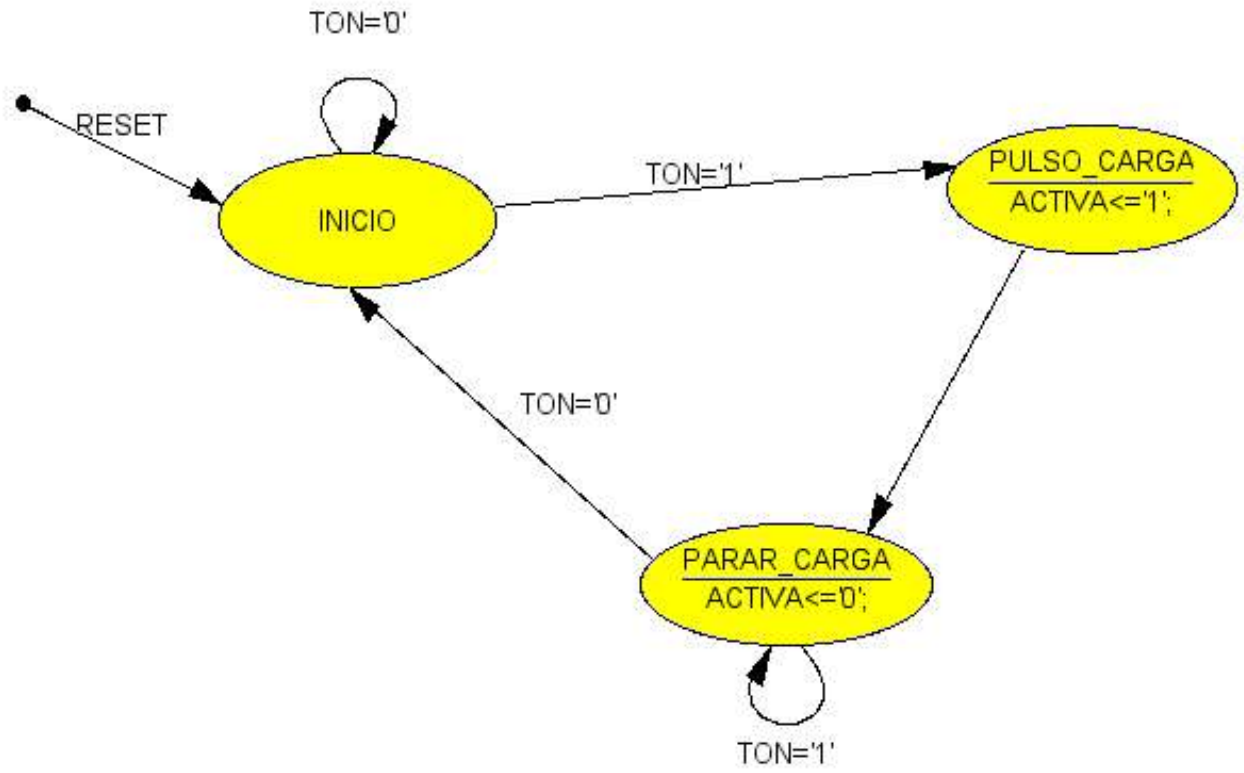
```
   end process;
end Behavioral;
```

De todas formas, esto soluciona parte del problema, ya que en los bloques normalmente queremos avisos que duren exactamente un ciclo de reloj; mientras que un pulsador (aún con anti-rebote) va a generar un 1 hasta que no suelte el botón. El módulo g_pulso soluciona este problema.

Generador de un pulso

Este bloque transforma una entrada como 0000111111110000 en 0000100000000000, quedándose sólo con el primer pulso.

Esto es muy útil para que un pulsador apretado durante unos segundos sólo genere una señal. También para que bloques que funcionan a distintas frecuencias puedan comunicarse poniendo una señal a 1 durante un rato (indefinido).



Compuesto por tres estados, descripción de cada uno de ellos:

- **INICIO:** Estado de espera en caso de que no se active ninguna temporización.

Entradas: Accedemos a este estado mediante un reset y nos mantenemos en él mientras no se solicite ningún pulso de activación mediante TON.

Salidas: Sólo se puede salir del estado inicial de espera mediante la activación de TON.

- **PULSO_CARGA:** Se ocupa de generar una salida a uno que se mantendrá durante un ciclo, de esta manera indicamos al siguiente bloque (TIMER) que ha de cargar el valor a decrementar (la salida T1ON y T2ON del bloque NUCLEO indican el valor a cargar para la temporización).

  Entradas: Solo podemos acceder a este estado si se activa la petición de temporización mediante TON.

  Salidas: Salimos en el siguiente ciclo de reloj pase lo que pase.

- **PARAR_CARGA:** Desactiva la petición, si se mantuviera a uno en el siguiente bloque estaríamos cargando continuamente el valor a decrementar.

  Entradas: Se accede sin ninguna condición de salto desde el estado PULSO_CARGA.

  Salidas: Nos mantendremos en este estado hasta que acabe la temporización solicitada y después retornaremos al estado de inicio.

Señales de entrada:

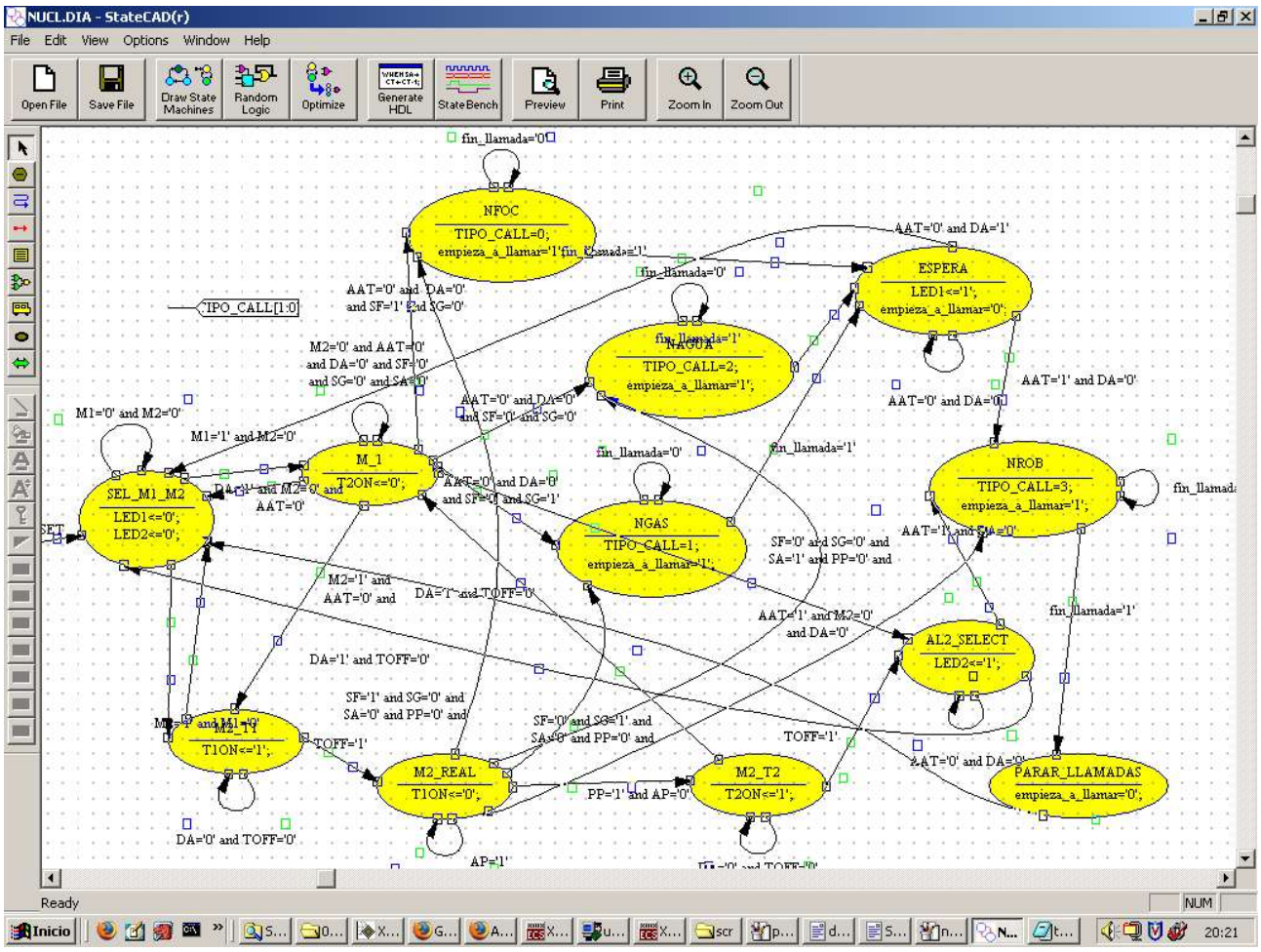
CLK: Señal de sincronismo.
RESET: Resetea y hace volver al estado inicial.
ON: Indica cuando empieza la temporización y cuando acaba..

Señales de salida:

ACTIVA: Genera un uno lógico durante un ciclo para indica al bloque de temporización que ha de cargar el valor a decrementar.

Núcleo

Es la parte principal de la alarma. Lleva la lógica de generación de alarmas y llamadas, y el control del modo actual.



Descripción del diagrama de estados del núcleo:

Compuesto de 12 estados, descripción de cada uno de ellos:

- **SEL_M1_M2**: Estado de inicio (conocido) en caso de reset del sistema, sólo podemos salir de él mediante la activación de M1 o de M2; para poder activar AAT o DA antes se ha de haber activado uno de los dos estados principales de trabajo.

  Entradas: Podemos acceder a este estado mediante un reset, activando DA o tras activar el código de AAT y haber hecho la llamada correspondiente al NROB.

  Salidas: Para dejar este estado nos han de activar las señales M1 o M2, mientras no llegue ninguna de ellas nos mantendremos en espera en este estado.

- **M_1**: Alarma conectada sin control de robo, en este estado se comprueban las señales de SF, SG, SA, y, como consideramos que hay gente también tenemos en cuenta la señal de AAT.

  Entradas: Podemos acceder a este estado desde SEL_M1_M2 (estado de inicio), o tras desactivar el modo M2.

  Salidas: la activación de las señales SF, SG, SA, la desactivación de la alarma mediante DA, la introducción del código AAT o la activacion del modo M2 nos harían dejar este estado; en caso de pasar del estado M1 al

M2, no es posible volver directamente al estado M1, las dos vías posibles para regresar a M1 son:

- Desactivar M2 mediante DA antes de que pase la temporización, volver al estado inicial SEL_M1_M2 y entonces introducir el código de M1.
- Salir del local, dejar que pasen las temporizaciones, volver a entrar y desactivar el modo M2 mediante DA.

- **M2_T1:** Estado de temporización 1, se da tiempo al usuario que ha conectado la alarma para que deje el local, una vez pasado este tiempo se activa realmente el modo M2.
  Entradas: Podemos acceder a este estado mediante la activación del código M2 tanto desde M1, como desde SEL_M1_M2.
  Salidas: Introduciendo el código DA y volviendo entonces a SEL_M1_M2, o al acabar la temporización T1ON y activarse la señal TOFF.

- **M2_REAL:** Alarma conectada con control de robo, se controlan las señales de SF, SG, SA, PP, AP, mientras todas estén a cero nos mantendremos en este estado.
  Entradas: Solo podemos acceder a este estado tras esperar la temporizacion del estado M2_T1.
  Salidas: Mientras no se activen las señales SF, SG, SA, AP, PP nos mantendremos en este estado

- **M2_T2:** Estado de temporizacion 2, se da tiempo al usuario para que desconecte la alarma (modo M2).
  Entradas: Solo se puede acceder a este estado si estando en M2_REAL se activa la señal de PP.
  Salidas: Si el usuario no desactiva la alarma antes de que transcurra la temporización T2ON y se active la señal TOFF, o desactivando la alarma mediante la introducción del código DA, volviendo así al estado M1.

- **AL2_SELECT:** Activación de la señal de alarma por intrusión (LED2); nos mantendremos en este estado mientras no se desactive la alarma o se introduzca el código de antiatraco.
  Entradas: Al ser introducido el código de antiatraco en el estado M1, o tras haber pasado la temporizacion T2ON sin que el usuario haya desactivado M2 mediante DA.
  Salidas: parando la alarma y volviendo a SEL_M1_M2, o introduciendo el código AAT, con lo que la alarma dejara de “sonar” (se apagara el LED2) pero se hará la llamada NROB.

- **NROB:** Estado que activa la llamada por intrusión.
  Entradas: Se puede acceder a este estado de tres maneras diferentes; tras detectarse AP activado estando en el estado M2_REAL ya que se supone que no hay nadie en el local para abrir una ventana; si al estar sonando la alarma en AL2_SELECT se introduce el código AAT, ya que se podría estar obligando al usuario a desactivar la alarma y el atracador no tiene porque conocer el código DA; o si tras detectarse la activación de SF, SG, SA y estar en espera de desactivación de la alarma de no intrusión (LED1) se introduce el código AAT ,puesto que el incendio, fuga de gas o fuga de agua podría

haber sido provocado.

Salidas: En este estado enviaremos la señal de inicio de marcado, así que nos mantendremos en este estado mientras que el bloque de marcado de llamadas no nos de la confirmación de que la llamada al numero seleccionado mediante TIPO_CALL se ha realizado correctamente.

- **NFOC, NAGUA, NGAS:** Estados que activan las llamadas por incidencia.
  Entradas: A los tres estados se puede llegar tanto desde M_1 como desde M2_REAL, al detectarse la activación de SF, SG, SA, en cada uno de los estados seleccionaremos el numero de teléfono al que se ha de llamar dependiendo de la incidencia que haya ocurrido mediante la salida TIPO_CALL, y con la salida "empieza_a_llamar" activaremos el bloque de marcado; mientras el bloque de marcado no nos avise de que la llamada se ha realizado correctamente seguiremos en el estado en el que estemos.

- **ESPERA:** Activa la alarma por incidencia (LED1), nos mantendremos en este estado mientras no se desactive la alarma mediante DA o se active la AAT; en este estado además avisamos al bloque de marcado de que ya no es necesario hacer mas llamadas.
  Entradas: Accederemos a ESPERA desde los estados de llamada por incidencia NFOC, NGAS, NAGUA siempre y cuando haya llegado la señal de marcado correcto del bloque de marcado.
  Salidas: Tras la desactivación de la alarma (DA) o la introducción del codigo antiatraco.

- **PARAR_LLAMADAS:** Indica al bloque de marcado que ya no es necesario hacer mas llamadas.
  Entradas: Solo se puede llegar aquí desde el estado NROB tras llegar la señal de "fin_llamada" que indica que el bloque de marcado ha acabado.
  Salidas: Va directamente al estado inicial SEL_M1_M2.

Señales de Entrada/Salida en el schematic núcleo:

Señales de entrada:

- **CLK:** Señal de sincronismo.
- **AAT:** Nos llega desde el bloque ACCESO y nos indica que la clave de antiatraco ha sido pulsada.
- **AP:** Nos indica que una ventana ha sido abierta, esta señal solo se controla en el estado M2_REAL.
- **DA:** Desactiva la alarma, dependiendo del estado en el que estemos nos puede llevar al estado inicial SEL_M1_M2 o a M1. Esta señal nos llega desde el bloque ACCESO.
- **fin_llamada:** Indica que la llamada solicitada se ha realizado correctamente, esta señal viene del bloque LLAMADAS.
- **M1:** Activa el modo de trabajo M1 (sin control de intrusión), esta señal nos llega desde el bloque ACCESO.

- **M2:** Activa el modo de trabajo M2 (con control de intrusión), esta señal nos llega desde el bloque ACCESO.
- **PP:** Apertura de puerta principal, esta señal solo se controla en el estado M2_REAL.
- **RESET:** Resetea el bloque y retornamos al estado SEL_M1_M2.
- **SA, SF, SG:** Indica que se ha detectado una fuga de agua, un incendio o una fuga de gas; estas señales de entrada se detectan tanto en el modo de trabajo M1 como en el M2.
- **TOFF:** Indica que la temporización seleccionada ha acabado; esta señal nos llega del bloque TIMER.

Señales de salida:

- **Empieza_a_llamar:** Indica al bloque de LLAMADAS que ha de empezar a marcar.
- **LED1:** Señal de alarma por incidencia (agua, fuego, gas).
- **LED2:** Señal de alarma por intrusión.
- **T1ON, T2ON:** Seleccionan una temporizacion en el bloque TIMER.
- **TIPO_CALL:** Indican la fila de la memoria en la que se guardan los números de teléfono a los que se ha de llamar en cada caso.

Temporizador

Este bloque sirve para contar tiempos; tanto para el intervalo T1 como para T2, ambos programables por el usuario. Se usan cuando el propietario de la alarma tiene que pasar por la puerta principal y necesita un margen de tiempo para desactivarla o cambiar de modo.

```
entity timer is
    Port (CLK : in std_logic;
          T1ON : in std_logic;
          T2ON : in std_logic;
          FIN_CUENTA : out std_logic);
end timer;

architecture Behavioral of timer is
  signal COUNT : std_logic_vector(7 downto 0);
begin

process (CLK)
begin
   if CLK='1' and CLK'event then
      COUNT <= COUNT - 1;
   end if;

   if T1ON='1' then COUNT<="00110011"; end if;
   if T2ON='1' then COUNT<="01000000"; end if;


   FIN_CUENTA<='0';
   if COUNT=0 then FIN_CUENTA<='1'; end if;

end process;
end Behavioral;
```

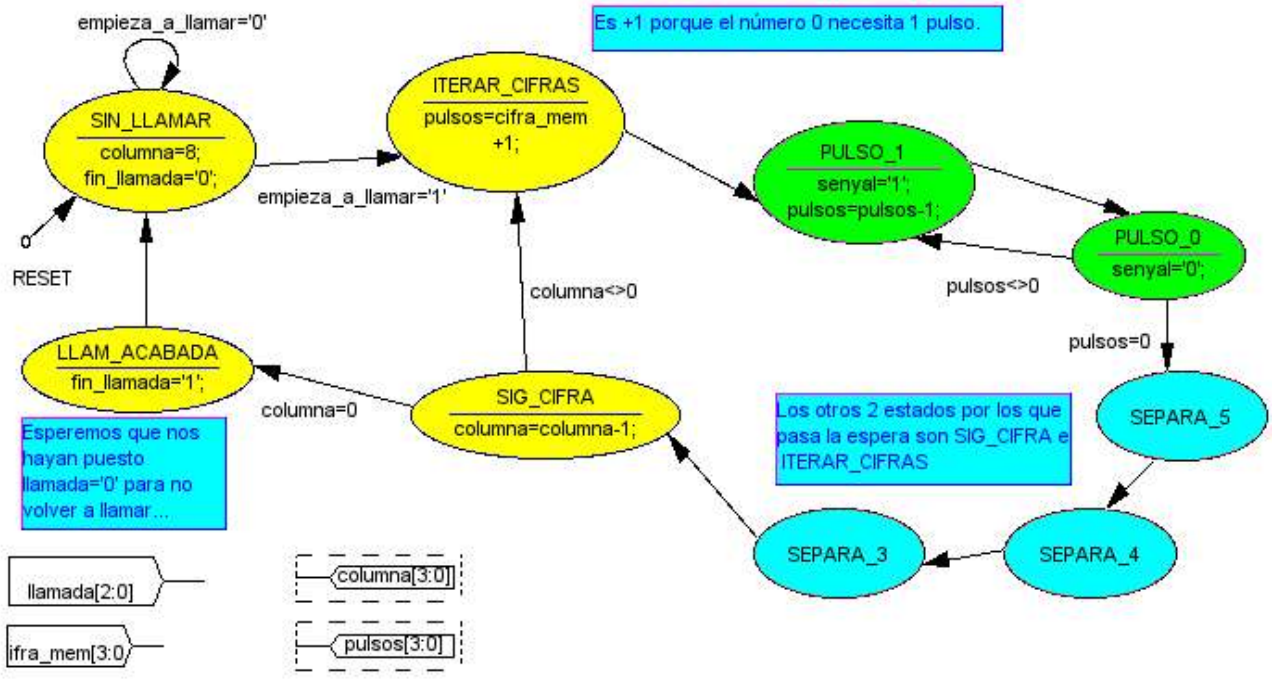
Le entran dos señales, T1ON y T2ON. La que se ponga a 1 indica el valor

con el que hay que empezar a contar. Al acabar, FIN_CUENTA se pone a 1, y el núcleo, que estaba esperando al contador, puede continuar. Si nadie está usando el contador, la señal FIN_CUENTA se puede ignorar.

La inicialización del contador no debe repetirse varias veces; sino sólo ejecutarse una vez por cada flanco de subida de T1ON o T2ON. No podemos poner varios 'event tanto en T1ON,T2ON y CLK por limitaciones de VHDL; pero sí que podemos filtrar la entrada usando un bloque que acorte los pulsos en los que el estado '1' dure más de un ciclo (ej: de 0000111110000 a 0000100000000).

Otra solución es acordar que el núcleo ponga la señal de aviso a 1 durante exactamente un ciclo, pero al trabajar a frecuencias distintas, se necesitaría un mecanismo de sincronización más complicado.

Llamadas



Este módulo es el encargado de marcar los números de teléfono apropiados mandando un tren de impulsos (101010...) por un cable de 1 bit, llamado “senyal” en el esquema. Entre cifra y cifra se esperan cinco ciclos.

Ejemplo: marcar el 931230120

```
10101010101010101010 00000 10101010 00000 1010 00000 101010 00000 10101010
00000 10 00000 1010 00000 101010 00000 10 00000
```

Los números están grabados en una memoria, de donde sacamos los números cifra a cifra. Puede haber números de teléfono distintos, pero el usuario no ha de informar del tipo de llamada a este módulo, sino a la memoria.

Empezamos pidiendo la primera de las 9 cifras, que está en la columna 8.

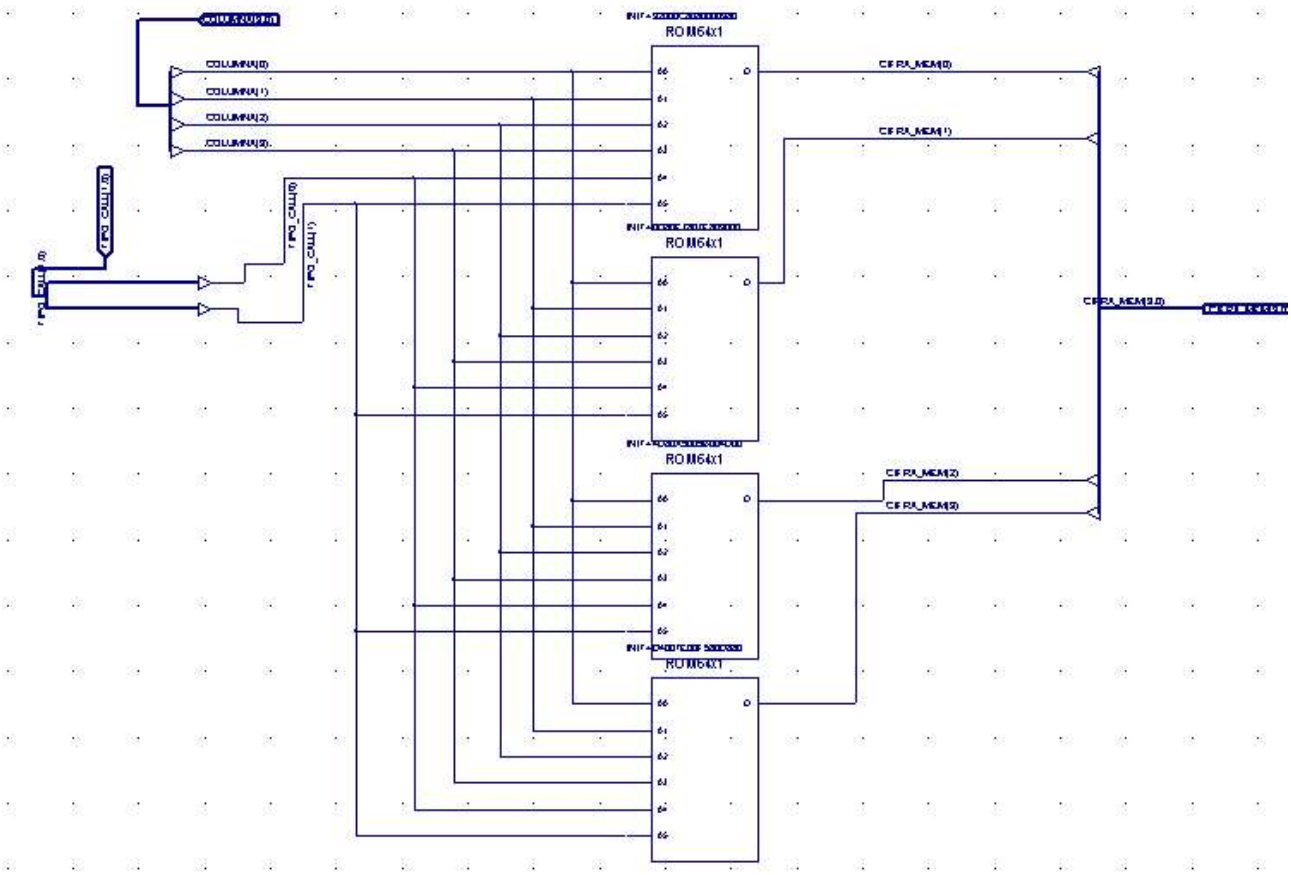
Será un número de 0 a 9 (4 bits), y tenemos que generar un impulso más de los que dice el número de la memoria, porque al 0 le corresponde 1 impulso (no 0), o sea, la serie 10.

Moviéndonos entre los estados de color verde se van generando los pulsos (primero a 1, luego a 0); esto se repite tantas veces como impulsos haya que generar. Luego vienen los 5 ciclos de espera, que pasan por los estados: SEPARA_5, SEPARA_4, SEPARA_3, SIG_CIFRA e ITERAR_CIFRAS.

Después se pide a la memoria la siguiente cifra del número. Como el módulo de llamadas va a una frecuencia mucho más lenta que la memoria, tendremos el resultado en el siguiente ciclo.

Si ya se ha marcado la novena cifra (columna 0) damos el aviso de llamada acabada (durante un ciclo).

Memoria



Este bloque se encarga de guardar 4 números de teléfono, de 9 cifras cada uno (cada cifra ocupa 4 bits). Le damos el tipo de llamada (0, 1, 2 ó 3) y la cifra que queremos (de 8 a 0) y nos da los 4 bits que codifican la cifra.

Como las memorias que tenemos guardan bits (no cifras de 4 bits), es difícil poder grabar los números de teléfono en binario en una sola memoria (de la forma normal, ej: 931… = 1001 0011 0001…), ya que para saber los 4 bits haría falta un “shift register” que fuera recordando los bits ya sacados, y

hacer 4 accesos a memoria.

Por eso nuestro diseño pide los 4 bits en paralelo: tenemos 4 bloques, en donde el bloque i es el encargado de grabar el i-ésimo bit de cada cifra (i va de 0 a 3).

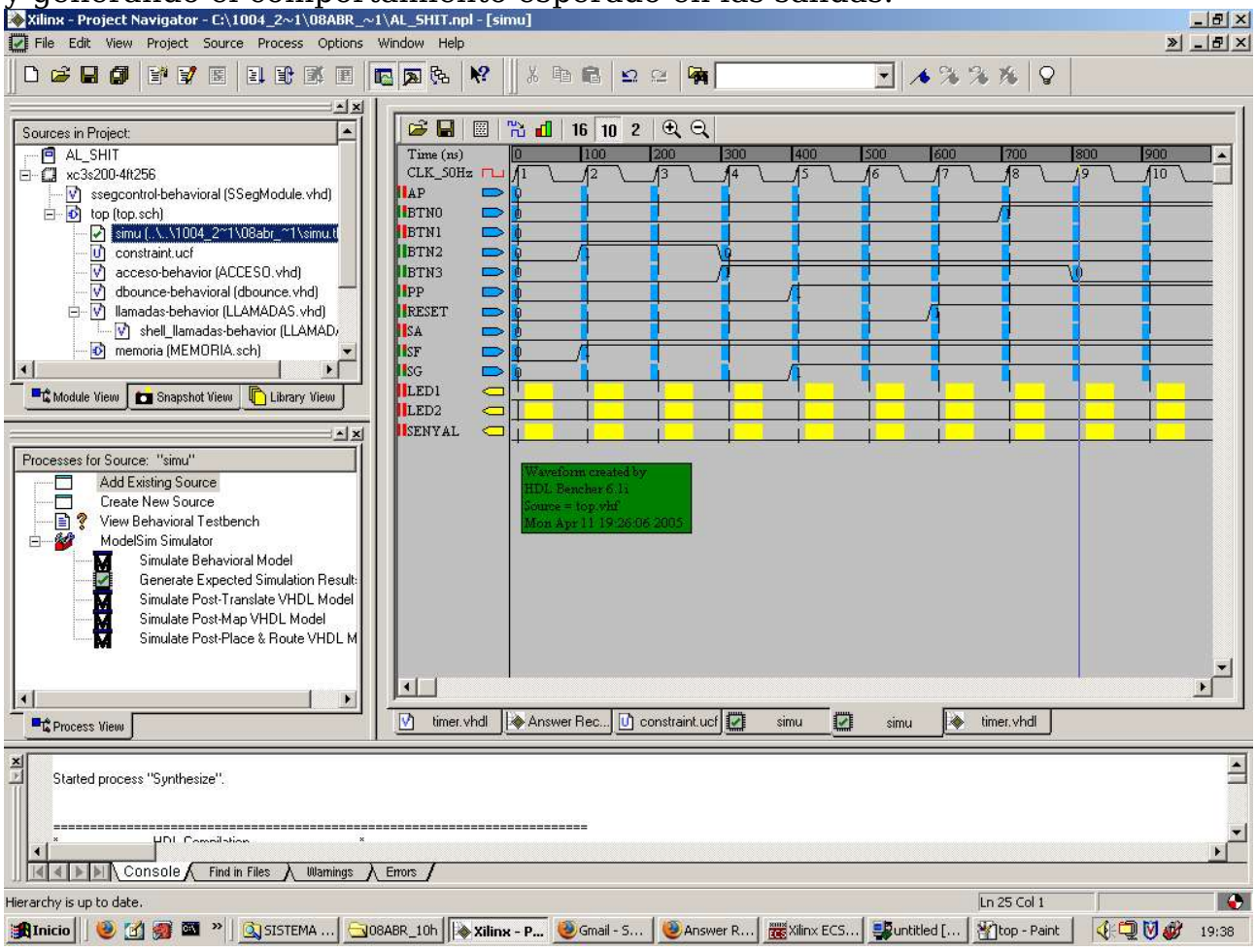
Cada bloque graba, por lo tanto, 4*9=36 bits, pero hemos usado memorias de 64 bits. Si fueran 8 cifras sí que serían 4*8=32, pero haría falta, además de la memoria de 32 bits, otra de 4 bits para guardar la cifra sobrante, y un multiplexor para decidir cuál de las dos memorias activar. Creemos que una sola memoria de 64 va a ser más eficiente, aunque esté medio vacía.

Grabar números en esta memoria es muy complicado para un técnico, así que hemos hecho un programa en C que genera el contenido que debe tener cada bloque para 4 números de teléfono concretos. Ver Anexo.

Simulación

Los diagramas de estado pueden ser simulados con el programa StateBench. En nuestras pruebas, los 4 diagramas (núcleo, detección de código, llamadas y g_pulso) nos han funcionado sin problemas y según lo planeado.

El resto de bloques puede simularse con ModelSim, añadiendo al proyecto un “Waveform”, dando valores a las entradas en función del tiempo, y generando el comportamiento esperado en las salidas:



Implementación

Al finalizar el diseño y compilar el proyecto mediante “implement design”, aparece en la ventana de log un resumen de los porcentajes de circuitería usada para implementar el proyecto. Se puede observar que en general el uso de recursos es bastante bajo (una media de ~9% del total de bloques configurables de la FPGA).

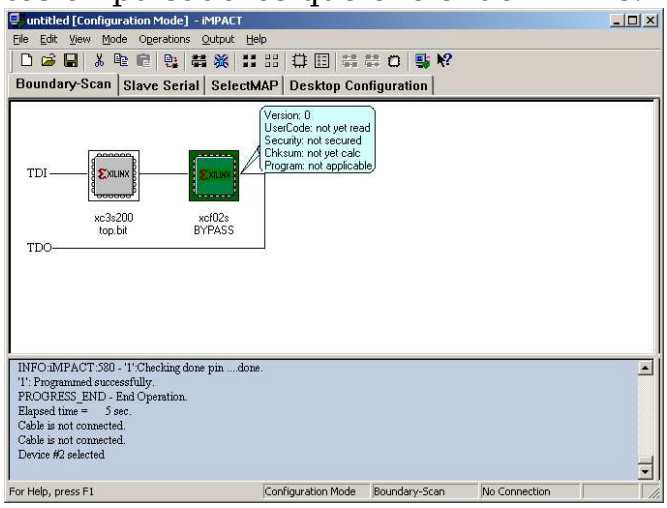
```
==============
*Final Report*
==============
Device utilization summary:
---------------------------
Selected Device : 3s200ft256-5
 Number of Slices:                      179    out of 1920   9%
 Number of Slice Flip Flops:            151    out of 3840   3%
 Number of 4 input LUTs:                323    out of 3840   8%
 Number of bonded IOBs:                 13     out of 173    7%
 Number of GCLKs:                       1      out of 8      12%
```

En el siguiente resumen, podemos apreciar cuatro parámetros interesantes, el primero indica la máxima frecuencia a la que podria trabajar nuestra implementación sin llegar a ser inestable (~181Mhz). Teniendo en cuenta que la Spartan3 usa un integrado de 50Mhz tenemos pues, un diseño bastante robusto.

Los dos timings siguientes hacen referencia al delay de los inputs/outputs, mientras que el último indica el máximo delay provocado por el circuito combinacional más largo (también llamado critical path). Se puede observar que el programa no ha sido capaz de encontrar dicho path (quizà porque el circuito es mayoritariamente secuencial).

```
Timig summary:
---------------
Speed Grade: -5
   Minimum period: 5.504ns (Maximum Frequency: 181.686MHz)
   Minimum input arrival time before clock: 5.605ns
   Maximum output required time after clock: 5.443ns
   Maximum combinational path delay: No path found
```

El resultado de la compilación se transfiere a la placa con iMPACT, pero para hacer pruebas satisfactorias ha habido que usar otros diseños más sencillos, consistentes en pulsadores que encienden LEDs.



Anexos

Escritura de teléfonos en memoria

Este programa facilita al técnico escribir los números de teléfono en los 4 bloques de memoria. También contiene el algoritmo para hacerlo a mano.

```
//
// Da el contenido de las 4 memorias a partir de los 4 números de teléfono
// Grupo 22 E de SDMI. Abril 2005.
//

typedef char BIT;
typedef int CIFRA;

int main(int argc, char *argv[]) {

  BIT mem[4][64];
  CIFRA nums[4][16]; // En realidad sólo se usan 9 cifras de las 16
  char hexa[4][16];

  int ci, ni, mi, bi; // Iteradores para cifra, número, memoria, bit

  if (argc!=1+4) exit(1); // Hay que pasar 4 números

  for(ni=0;ni<=3;ni++) {
    printf("Núm %d: %s\n",ni,argv[ni+1]);
    for(ci=0;ci<=8;ci++) { nums[ni][ci]=argv[ni+1][ci]-'0'; }
    for(ci=9;ci<=15;ci++) { nums[ni][ci]=0; }
  }

  for(mi=0;mi<=3;mi++) {
    for(ni=0;ni<=3;ni++) {
      for(ci=0;ci<=8;ci++) {
        mem[mi][ni*16+(8-ci)] = ( nums[ni][ci] & (1<<mi) )?1:0 ;
      }
      for(ci=9;ci<=15;ci++) mem[mi][ni*16+ci] = 0 ;
    }
  }

  for(mi=0;mi<=3;mi++) { for(ci=0;ci<=15;ci++) { hexa[mi][ci]=0; } }

  for(mi=0;mi<=3;mi++) {
    printf("\nMemoria %d: ",mi);
    for(bi=0;bi<=63;bi++) {
      printf("%d",mem[mi][bi]);
      if((bi+1)%16==0) printf(" ");
      hexa[mi][bi/4] |= mem[mi][bi] << (3-(bi%4));
    }
  }
  printf("\n");

  for(mi=0;mi<=3;mi++) {
    printf("\nMemoria %d: ",mi);
    for(ci=0;ci<=15;ci++) {
      printf("%x",hexa[mi][ci]);
      if((ci+1)%4==0) printf(" ");
    }
  }
  printf("\n");
```

```
  return 0;
}
```

Ejemplo de salida:

```
Núm 0: 936621070
Núm 1: 800142857
Núm 2: 111222333
Núm 3: 123456789

Memoria 0: 0101000110000000 1100010000000000 1110001110000000 1010101010000000
Memoria 1: 0100111100000000 1001000000000000 1111110000000000 0011001100000000
Memoria 2: 0100011000000000 1100100000000000 0000000000000000 0011110000000000
Memoria 3: 0000000010000000 0010000010000000 0000000000000000 1100000000000000

Memoria 0: 5180 c400 e380 aa80
Memoria 1: 4f00 9000 fc00 3300
Memoria 2: 4600 c800 0000 3c00
Memoria 3: 0080 2080 0000 c000
```

Implementaciones alternativas

Algunas opciones que hemos descartado durante el diseño son:

- Podríamos tener un bloque sólo para guardar el modo en el que estamos (1, 2, T1, T2, o apagado), pero no nos hace falta porque lo controlamos desde el módulo núcleo: el modo actual se sabe viendo en qué estado del diagrama estamos.
- En vez de hacer un algoritmo que genera impulsos a partir de los números de teléfono que hay en la memoria, podríamos grabar el tren de impulsos directamente en la memoria, en forma de una larga tira de bits para cada número de teléfono. Esto la simplificaría, pero sería poco eficiente.
- Módulos sencillos como g_pulso se podrían hacer en VHDL directamente, pero como no lo conocemos a fondo, hemos preferido diagramas.

Posibles mejoras

- El núcleo no tiene por qué esperar a que una llamada finalice correctamente (durante el tiempo que dura la llamada, no está detectando intrusiones).
- Para probar los módulos complejos en la placa de desarrollo, habría que añadir el módulo “display 7 segmentos” que teníamos de la primera práctica, usando como salidas las definidas en el fichero de “constraints”.
- Conectando la salida de los pulsadores al display 7 segmentos, se podría hacer que la clave se viera en pantalla mientras se teclea. Harían falta unos registros entre medio para que el valor no desapareciera al dejar de pulsar el botón.