Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

IMPLEMENTACIÓN DE UN RELOJ DIGITAL EN FPGA

Alumno: José Gómez Rueda

Tutor: Prof. D. Pedro J. Casanova Peláez
Dpto: Ingeniería Electrónica y Automática

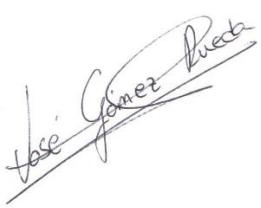
JUNIO, 2019

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Electrónica y Automática

Don PEDRO CASANOVA PELÁEZ, tutor del Trabajo Fin de Grado titulado: IMPLEMENTACIÓN DE UN RELOJ DIGITAL EN FPGA, que presenta JOSÉ GÓMEZ RUEDA, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, JUNIO de 2019

El alumno: El tutor:



CASANOVA PELAEZ PEDRO JOSE - 26000096E

Firmado digitalmente por CASANOVA PELAEZ PEDRO JOSE - 26000096E
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-26000096E, givenName=PEDRO JOSE, sn=CASANOVA PELAEZ, cn=CASANOVA PELAEZ PEDRO JOSE - 26000096E
Fecha: 2019.06.12 19:28:06 +02'00'

JOSÉ GÓMEZ RUEDA PEDRO CASANOVA PELÁEZ

AGRADECIMIENTOS

Especialmente a la persona con la que he compartido todo en mi vida, mi hermano, es la persona que mejor me conoce y siempre me ha ayudado a tomar las mejores decisiones. Acordarme de mi padre y mi madre, por su esfuerzo para que pudiera llegar a lo que soy, son mis referencias en todos los apartados de la vida.

También a los buenos profesores que he tenido durante estos años, especialmente a mi tutor, Pedro Casanova, porque he tenido la posibilidad de realizar este trabajo y también cursar dos asignaturas durante la carrera y siempre ha estado disponible para cualquier tipo de consulta.

Por último, como jiennense, creo que es obligatorio estar agradecido a la Universidad de la ciudad que me vió crecer.

Índice

Ilustraciones

Tablas

1. INTRODUCCIÓN

En este primer punto explicaré las características básicas de este Trabajo de fin de grado con el objetivo de aportar un conocimiento general sobre lo que se desea plasmar, asimismo explicaré los motivos que me han llevado a elegir este trabajo y los objetivos que se persiguen.

1.1. Introducción al proyecto

Este proyecto consistirá en la realización de un reloj digital y sus múltiples funcionalidades con FPGA. Las funciones que estarán disponibles son el propio reloj, el cronómetro, la cuenta atrás y una alarma.

Para la realización del proyecto se ha utilizado la FPGA de la familia MAX10 de Intel incluida en el kit de evaluación y desarrollo DE10-Lite de Terasic. En el desarrollo del mismo se utilizará el software Quartus, más concretamente el Quartus Prime 17.1, al cual podemos tener acceso directamente desde la propia plataforma de Intel.

En nuestro proyecto se expondrán los distintos elementos externos para el montaje hardware del reloj debido a que con las características de la propia placa no tenemos suficiente.

1.2. Motivación

Actualmente el uso de la electrónica digital es muy frecuente en el día a día, eso no es una noticia, pero hay muchas utilidades que se desconocen y pueden ser muy usadas habitualmente. Debido a esto, escogí realizar algo tan común como un reloj digital con diversas aplicaciones.

Para la realización de este trabajo he utilizado el leguaje de codificación “VHDL”, para el cual es imprescindible tener un conocimiento bastante amplio, tanto por parte del autor para poder desarrollar el código, como por la del lector para poder interpretarlo correctamente. El hecho de tener que aprender un lenguaje nuevo era una motivación añadida.

1.3. Objetivos

Los objetivos principales que se persiguen con la realización de este Trabajo de fin de grado son los siguientes:

- Desarrollo individualizado del reloj.
- Desarrollo individualizado del cronómetro.
- Desarrollo individualizado de la cuenta atrás.
- Desarrollo de la alarma en función de la hora marcada por el reloj.
- Implementación de todos los bloques en uno principal.

2. CONCEPTOS BÁSICOS

2.1. Introducción a la FPGA

Las FPGA, “field programmable gate array”, son un tipo de dispositivo, que como su nombre indica, pueden programarse para la creación de diferentes aplicaciones digitales de forma interna como puertas lógicas, sistemas combinacionales y otros de mucha más complejidad.

Se pueden realizar gran variedad de proyectos debido a que el número de servicios que pueden llegar a ofrecer los diferentes elementos lógicos del sistema es tan elevado que es muy difícil hacer uso de todas las posibilidades que permite realizar. Este número varía en función de la FPGA que se use. El propio software se encargará de usar internamente el número de elementos que considere más óptimo.

Una de las características principales de las FPGAs es que pueden ser programados las veces que sea necesarios.

El VHDL es un lenguaje de descripción de hardware, el cual usaremos para nuestro proyecto y su uso es muy frecuente, principalmente en Europa. Existen otros lenguajes como puede ser el Verilog que al igual que el VHDL tiene gran uso en ciertos lugares.

2.2. Utilidad de las FPGAs

En la actualidad la diversidad de uso de las FPGAs está tan extendida que son diferentes los campos de actuación en los que podemos encontrar el uso de estos dispositivos. A continuación, en la Ilustración 2 hago una breve clasificación de los diferentes campos de actuación de los mismos.

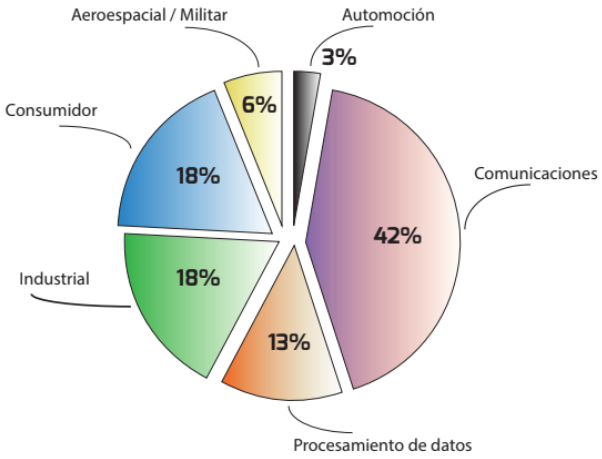


Ilustración 1. Aplicaciones de los FPGAs

- Aplicaciones aeroespaciales, la ESA “Agencia Espacial Europea” usa los FPGAs en aplicaciones críticas y substituyen a los ASIC de manera regular. Un ejemplo de su uso puede ser los codificadores telemetría.
- Aplicaciones en automoción, unas de las aplicaciones que se espera que tenga más influencia es en la seguridad vial, concretamente intervienen en procesamiento y control, permitiendo la toma de decisiones basadas en las entradas procesadas debido a que los FPGA ofrecen una ventaja crucial en el procesamiento de señales digitales al proporcionar un procesamiento paralelo pueden dar respuestas más rápidas a posibles peligros.

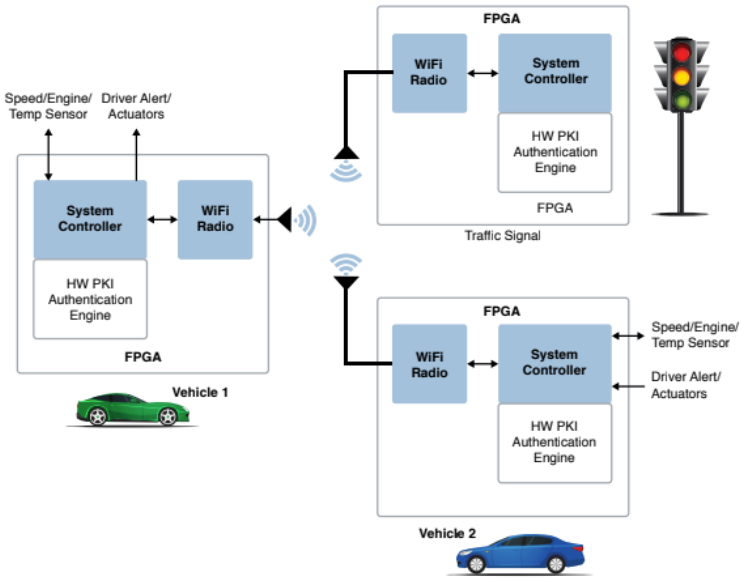


Ilustración 2. FPGA en aplicaciones de automoción

3. HARDWARE UTILIZADO.

3.1. Elementos DE10-Lite.

La placa que usaremos, como ya se ha comentado previamente, es la DE10-Lite la cual cuenta con una plataforma de diseño de hardware alrededor de Intel MAX 10 FPGA, cuenta con un número importante de elementos, que analizaremos uno a uno.

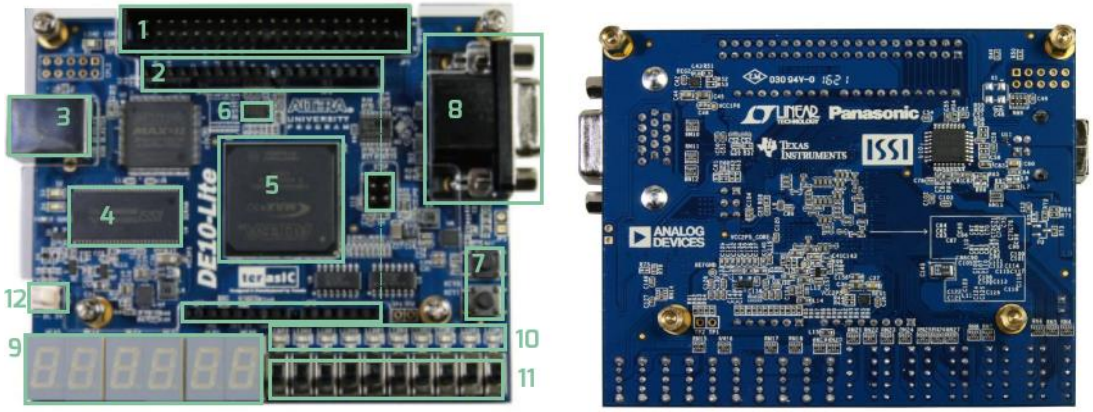


Ilustración 3. FPGA DE10-Lite

A partir de la numeración mostrada en la ilustración 4, voy a detallar las características de la placa utilizada.

1. 2x20 GPIO Headers, en total son 40 pines, de los cuales 36 están directamente conectados a la FPGA, dos son pines de tierra y los otros dos restantes proporcionan, uno, una tensión 3,3V y el otro 5V, ambos de corriente continua.
2. Conectores de Arduino Uno R3, consiste en una serie de pines que son compatibles con el modelo de Arduino mencionado, más concretamente tiene 16 pines GPIO y 1 pin de reset. La entrada analógica se realiza a través de 6 pines que son conectados directamente al convertidor analógico digital, también dispone de tres pines de tierra, uno de tensión de 5V y dos de 3,3V, ambos de corriente continua.
3. Puerto USB Blaster, con conector USB tipo B.

4. SDRAM, concretamente 64MB con un chip. Tiene una línea de datos de 16 bits, una línea de control y una de dirección. Este chip utiliza el estándar de señalización 3,3V LVCMOS.
5. Intel MAX 10, 10M50DAF484C7G es la FPGA incluida en la tarjeta.
6. Acelerómetro, consiste en un módulo de sensor acelerómetro, ADXL345, conocido como G-sensor.
7. 2 botones, que pueden evitar el ruido utilizando la activación Schmitt Trigger.
8. Puerto VGA, conector D-SUB que cuenta con 15 pines de salida. Se utiliza un convertidor digital analógico de 4 bits que usa una red de resistencias para producir señales de datos analógicas, los colores rojo, verde y azul.
9. 6 Displays de 7 segmentos cada uno y de ánodo común.
10. 10 LEDs, cada uno de ellos se activa directamente y de forma individualizada, se encenderán cuando reciban un nivel lógico alto.
11. 10 Interruptores, cada interruptor proporciona un nivel lógico bajo cuando dicho interruptor está en posición “abajo”.
12. 2 Pin Headers, uno de 5V y otro de toma de tierra.

En cuanto a las características de la FPGA MAX 10 10M50DAF484C7G:

- Convertidores analógico digital “ADCs” Integrados dual, cada convertidor admite 1 entrada analógica dedicada y 8 pines de doble función.
- 50000 elementos lógicos programables.
- La memoria integrada consiste en bloques de memoria M9K, este tipo de bloques, en dispositivos Intel MAX10 proporcionan 9 Kb de memoria, pudiendo llegar a funcionar a 284 MHz, en las FPGAs del grupo 10M50 puede llegar a 1638 Kb como máximo.
- Bloque de memoria flash de usuario (UFM), cuyo valor máximo posible es 5888 Kb, incluida la memoria flash del usuario y la memoria flash de configuración.
- Los dispositivos Intel MAX 10 admiten hasta 144 bloques multiplicadores incrustados, cada bloque admite un multiplicador individual de 18x18 bits o dos multiplicadores de 9x9 bits.
- 4 PLLs, bucles de seguimiento de fase.

- 64MB de memoria SDRAM.

Una vez hemos podido conocer las características generales de la placa que vamos a utilizar, vamos a describir a fondo los elementos que usaremos para nuestro proyecto en función de las necesidades.

3.1.1. Circuitos de Reloj.

Para nuestro proyecto es muy importante la señal de reloj, para ello utilizaremos una de las señales de reloj de 50 MHz, para posteriormente realizar un divisor de frecuencia con el objetivo de conseguir la frecuencia que deseamos para cada modalidad de nuestro proyecto.

En nuestro caso para la asignación de pines usaremos el siguiente,

Nombre señal	Nº PIN	Descripción	E/S estándar
MAX10_CLK1_50	PIN_P11	50MHz de reloj de entrada	3.3-V LVTTL

Tabla 1. Asignación de pines del reloj

3.1.2. Botones

La placa incluye dos botones ambos necesarios para la realización de mi trabajo, concretamente serán usados de la siguiente manera en función de cada modalidad:

- En el modo reloj, estos botones servirán para modificar el valor de la hora, uno de los botones, concretamente el KEY0, hará que el valor ascienda y el KEY1 que haga la acción contraria.
- En el modo cronómetro, la función del KEY0 será tanto la de iniciar la cuenta como la de hacer que el reloj esté parado el tiempo que se considere oportuno hasta que se vuelva a pulsar, en cambio con el botón KEY1 hará que el valor presente en el momento de ser pulsado, se detenga para mostrarse, pero no pare la cuenta y una vez soltado muestre la misma.
- En el modo cuenta atrás, la función de KEY0 será la del ajuste de minutos en modo ascendente y KEY1 hará la función descendente.

- En el modo alarma, los botones tienen la misma función que en el modo reloj, KEY0 ajustará la hora en manera ascendente y KEY1 de forma descendente.

Una observación importante es que estos botones están normalmente activos.

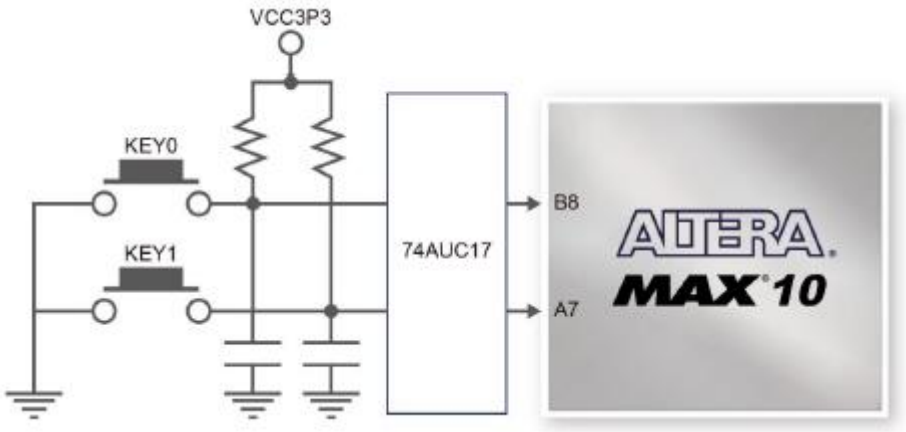


Ilustración 4. Botones Altera MAX10

Como se observa en la ilustración nº6, estos botones cuentan con un antirrebote, también cabe destacar que la función de activación “Schmitt trigger” provoca que el ruido no interfiera en la señal que generan estos botones.

La asignación de pines de los botones KEY0 y KEY1 es la siguiente,

Nombre señal	Nº PIN	Descripción	E/S estándar
KEY0	PIN_B8	Botón 0	3.3-V SCHMITT TRIGGER
KEY1	PIN_A7	Botón 1	3.3-V SCHMITT TRIGGER

Tabla 2. Asignación de pines de los botones

3.1.3. Interruptor

A continuación, voy a realizar mediante una tabla el nombre de señal para cada interruptor:

Nombre señal	Nº PIN	Descripción	E/S estándar
SW0	PIN_C10	Interruptor 0	3.3-V LVTTL
SW1	PIN_C11	Interruptor 1	3.3-V LVTTL
SW7	PIN_A14	Interruptor 7	3.3-V LVTTL
SW8	PIN_B14	Interruptor 8	3.3-V LVTTL
SW9	PIN_F15	Interruptor 9	3.3-V LVTTL

Tabla 3. Asignación de pines de los interruptores

En nuestro planteamiento, los interruptores tendrán una función muy importante, asignaremos los diez que disponemos, aunque al final solo vayamos a utilizar los siguientes:

- Para seleccionar el modo, al tener 4 modalidades (reloj, cronometro, cuenta atrás y alarma) utilizaremos dos interruptores, SW(7) y SW(8), que nos darán todas las combinaciones necesarias.
- Si la combinación de los SW(7) y SW(8) es “00”, estaremos actuando en el modo reloj. Entonces utilizaremos el SW(1) como ajuste, entendiendo como tal, la necesidad que esté activado para poder modificar los valores tanto de la hora como de los minutos.
- Si la combinación de los SW(7) y SW(8) es “11” , estaremos actuando en el modo cuenta atrás. Se utilizará SW(1) en modo ajuste, del mismo modo que el caso anterior. Si el valor de ajuste está proporcionando un nivel lógico alto, se podrá configurar los minutos y segundos desde donde queremos que comience la cuenta regresiva.
- Si la combinación de los SW(7) y SW(8) es “01” , estaremos actuando en el modo cronómetro.
- Si la combinación de los SW(7) y SW(8) es “10” , estaremos actuando en el modo Alarma.
- Se utilizará el SW(9) para resetear cualquiera de las funciones, excepto la hora que solo podrá resetearse en caso de estar modificándose la hora.
- Se utilizará el SW(0) para poder activar o desactivar la alarma.

3.1.4. Conectores de Arduino Uno R3

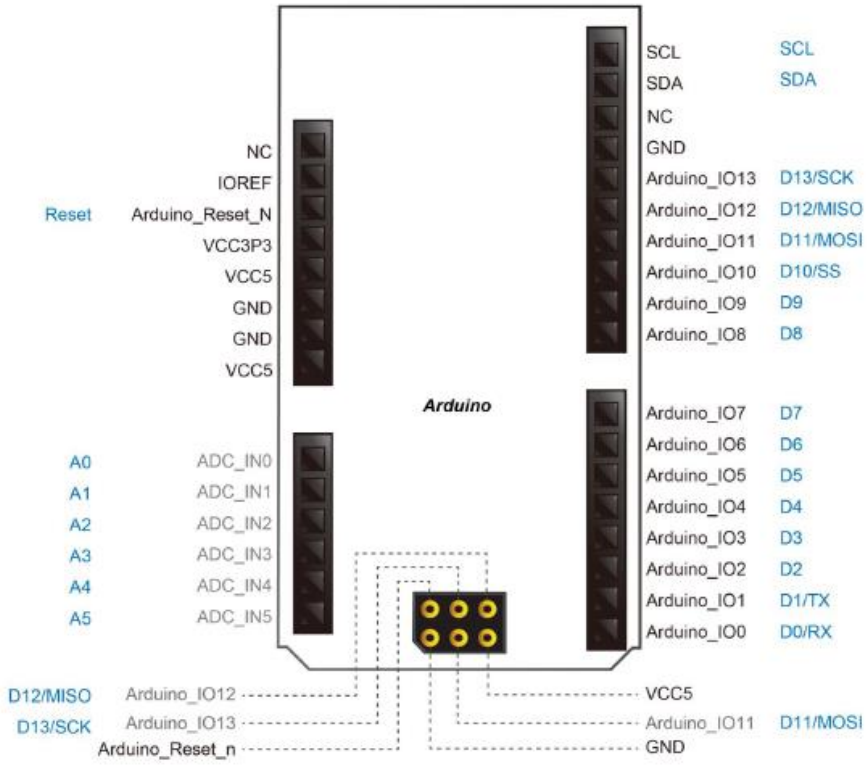


Ilustración 5. Conectores Arduino Uno R3

Como aspectos generales, indicar que esta parte es compatible con la placa de expansión de Arduino Uno R3, la cual dispone de 17 pines, mostrados en la ilustración 17, conectados directamente al MAX10. Las entradas analógicas se conectan a los 6 pines de ADC. La placa también consta de alimentación en continua de +5V (VCC5), de +3,3V (VCC3P3 y IOREF) y tres pines de toma a tierra. Para entender mejor la situación de los pines podemos observar la ilustración Nº7.

Debido a que en nuestra placa no dispone de suficientes elementos hardware para satisfacer las necesidades creadas por el proyecto, necesitamos un zumbador para hacerlo sonar una vez la hora fijada en el modo alarma coincida con la hora marcada por el modo reloj, dos pulsadores para modificar tanto de modo ascendente como descendente la parte de los minutos en el modo alarma y reloj, como modificar la configuración de los segundos en la cuenta atrás y un botón para dar comienzo a dicha cuenta, para utilizarlos, al igual que los botones integrados en el hardware, es importante conocer que están normalmente activos pero a diferencia de ellos, estos

no disponen ni de pull-up ni de antirrebotes, que se realizarán directamente en el código.

A continuación, en la siguiente tabla se observará los pines que se utilizarán.

Nombre señal	Nº PIN	Descripción	E/S estándar
ARDUINO_IO_IN(13)	PIN_AB20	Botón 4	3.3-V LVTTL
ARDUINO_IO_IN(12)	PIN_Y19	Botón 3	3.3-V LVTTL
ARDUINO_IO_IN(10)	PIN_AB19	Botón 2	3.3-V LVTTL
ARDUINO_IO_OUT(7)	PIN_AA12	Zumbador	3.3-V LVTTL

Tabla 4. Asignación de pines Arduino Uno R3

3.1.5. Displays de 7 segmentos

Como información complementaria de los elementos de nuestro FPGA, se puede destacar que tiene 6 displays, cada uno de ellos constan de siete segmentos de ánodo común, que se encenderán o apagarán en función les llegue un nivel lógico alto o bajo.

En nuestro proyecto, estos displays serán transcendentales debido a que en ellos se mostrarán los valores tanto del reloj como de cada una de las funciones.

Con la siguiente ilustración y la tabla podemos saber que pines le corresponde cada segmento.

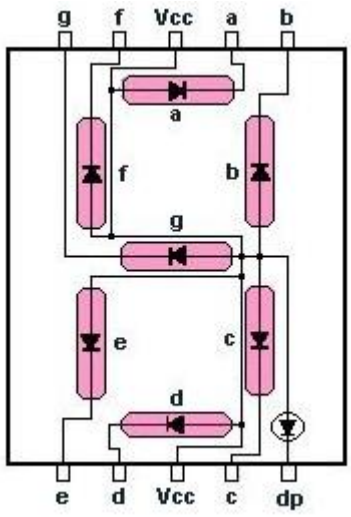


Ilustración 6. Display de ánodo común

Nombre señal	Nº PIN	Descripción	E/S estándar
HEX00	PIN_C14	Display 0 segmento A	3.3-V LVTTL
HEX01	PIN_E15	Display 0 segmento B	3.3-V LVTTL
HEX02	PIN_C15	Display 0 segmento C	3.3-V LVTTL
HEX03	PIN_C16	Display 0 segmento D	3.3-V LVTTL
HEX04	PIN_E16	Display 0 segmento E	3.3-V LVTTL
HEX05	PIN_D17	Display 0 segmento F	3.3-V LVTTL
HEX06	PIN_C17	Display 0 segmento G	3.3-V LVTTL
HEX07	PIN_D15	Display 0 segmento DP	3.3-V LVTTL
HEX10	PIN_C18	Display 1 segmento A	3.3-V LVTTL
HEX11	PIN_D18	Display 1 segmento B	3.3-V LVTTL
HEX12	PIN_E18	Display 1 segmento C	3.3-V LVTTL
HEX13	PIN_B16	Display 1 segmento D	3.3-V LVTTL
HEX14	PIN_A17	Display 1 segmento E	3.3-V LVTTL
HEX15	PIN_A18	Display 1 segmento F	3.3-V LVTTL
HEX16	PIN_B17	Display 1 segmento G	3.3-V LVTTL
HEX17	PIN_A16	Display 1 segmento DP	3.3-V LVTTL
HEX20	PIN_B20	Display 2 segmento A	3.3-V LVTTL
HEX21	PIN_A20	Display 2 segmento B	3.3-V LVTTL
HEX22	PIN_B19	Display 2 segmento C	3.3-V LVTTL
HEX23	PIN_A21	Display 2 segmento D	3.3-V LVTTL
HEX24	PIN_Y19	Display 2 segmento E	3.3-V LVTTL
HEX25	PIN_AB19	Display 2 segmento F	3.3-V LVTTL
HEX26	PIN_AB20	Display 2 segmento G	3.3-V LVTTL
HEX27	PIN_Y19	Display 2 segmento DP	3.3-V LVTTL
HEX30	PIN_F21	Display 3 segmento A	3.3-V LVTTL
HEX31	PIN_E22	Display 3 segmento B	3.3-V LVTTL
HEX32	PIN_E21	Display 3 segmento C	3.3-V LVTTL
HEX33	PIN_C19	Display 3 segmento D	3.3-V LVTTL
HEX34	PIN_C20	Display 3 segmento E	3.3-V LVTTL
HEX35	PIN_D19	Display 3 segmento F	3.3-V LVTTL
HEX36	PIN_E17	Display 3 segmento G	3.3-V LVTTL
HEX37	PIN_D22	Display 3 segmento DP	3.3-V LVTTL
HEX40	PIN_F18	Display 4 segmento A	3.3-V LVTTL
HEX41	PIN_E20	Display 4 segmento B	3.3-V LVTTL
HEX42	PIN_E19	Display 4 segmento C	3.3-V LVTTL
HEX43	PIN_J18	Display 4 segmento D	3.3-V LVTTL
HEX44	PIN_H19	Display 4 segmento E	3.3-V LVTTL
HEX45	PIN_F19	Display 4 segmento F	3.3-V LVTTL
HEX46	PIN_F20	Display 4 segmento G	3.3-V LVTTL
HEX47	PIN_F17	Display 4 segmento DP	3.3-V LVTTL
HEX50	PIN_J20	Display 5 segmento A	3.3-V LVTTL
HEX51	PIN_K20	Display 5 segmento B	3.3-V LVTTL
HEX52	PIN_L18	Display 5 segmento C	3.3-V LVTTL
HEX53	PIN_N18	Display 5 segmento D	3.3-V LVTTL

HEX54	PIN_M20	Display 5 segmento E	3.3-V LVTTL
HEX55	PIN_N19	Display 5 segmento F	3.3-V LVTTL
HEX56	PIN_N20	Display 5 segmento G	3.3-V LVTTL
HEX57	PIN_L19	Display 5 segmento DP	3.3-V LVTTL

Tabla 5. Asignación de pines de los displays

4. CODIFICACIÓN.

Una vez explicados los elementos que vamos a usar en nuestro trabajo, vamos a describir cómo será la configuración que necesitarán los elementos descritos en el punto anterior.

En nuestro caso, aunque internamente dentro del código está formado por muchos componentes que tendrán que ser instanciados a su vez por otros, al final todo se concreta en un único módulo cuyas entradas y salidas serán las que tengan que estar conexionadas con el hardware de nuestra FPGA.

4.1. CONFIGURACIÓN DE LA CONEXIÓN ENTRE FPGA Y CÓDIGO.

En este punto trataremos como se deben configurar tanto las entrada y salidas que se usarán. La información que se abordará a continuación deberá ir incluida en el Assignament Editor.

4.1.1. Reloj

ESTATUS	A	NOMBRE ASIGNACIÓN	VALOR	HABILITADO	ENTIDAD
OK	MAX10_CLK1_50	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	MAX10_CLK1_50	LOCATION	PIN_P11	YES	-

Tabla 6. Configuración de la asignación del reloj

Para configurar el conexionado del reloj que usaremos, generará una señal de reloj de 50 MHz, se deberán generar dos filas como se puede observar en la tabla nº6.

En la primera, se indicará la asignación entrada/salida estándar “I/O STANDARD” y el valor que se usará será 3.3V LVTTL (Low Voltage Transistor-transistor logic).

En la segunda fila, se indicará la información de la localización, se usará la asignación “LOCATION”, introduciendo el pin que le corresponde al reloj en el valor.

Para ambas filas, se necesitarán complementar las columnas tanto para indicar el estatus, como para conocer si está habilitado o no. En el primer caso se indicará con un “OK” si está correctamente definido el elemento y el segundo con un “YES” si

está habilitado. Por último, la columna de entidad se completará cuando la asignación ha sido la I/O ESTÁNDAR, teniendo que introducir el nombre de la entidad donde se definen estos elementos, en nuestro caso es el RELOJ_FINAL.

4.1.2. Interruptores

ESTATUS	A	NOMBRE ASIGNACIÓN	VALOR	HABILITADO	ENTIDAD
OK	SW[0]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	SW[1]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	SW[2]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	SW[3]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	SW[4]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	SW[5]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	SW[6]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	SW[7]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	SW[8]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	SW[9]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	SW[0]	LOCATION	PIN_C10	YES	-
OK	SW[1]	LOCATION	PIN_C11	YES	-
OK	SW[2]	LOCATION	PIN_D12	YES	-
OK	SW[3]	LOCATION	PIN_C12	YES	-
OK	SW[4]	LOCATION	PIN_A12	YES	-
OK	SW[5]	LOCATION	PIN_B12	YES	-
OK	SW[6]	LOCATION	PIN_A13	YES	-
OK	SW[7]	LOCATION	PIN_A14	YES	-
OK	SW[8]	LOCATION	PIN_B14	YES	-
OK	SW[9]	LOCATION	PIN_F15	YES	-

Tabla 7. Configuración de la asignación de los interruptores

En el caso de los interruptores, como se observa en la ilustración nº7 ocurre algo similar al del reloj, tenemos que crear dos filas por cada interruptor que se desee habilitar. En la asignación habrá que indicar la localización, con su pin correspondiente en el apartado del valor, y en la entrada/salida estándar con el valor escogido, para nuestro trabajo, 3,3V LVTTL.

4.1.3. Pines Arduino.

ESTATUS	A	NOMBRE ASIGNACIÓN	VALOR	HABILITADO	ENTIDAD
OK	ARDUINO_IO_IN	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	ARDUINO_IO_OUT	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL

OK	ARDUINO_IO_IN[8]	WEAK PULL-UP RESISTOR	ON	YES	RELOJ_FINAL
OK	ARDUINO_IO_IN[9]	WEAK PULL-UP RESISTOR	ON	YES	RELOJ_FINAL
OK	ARDUINO_IO_IN[10]	WEAK PULL-UP RESISTOR	ON	YES	RELOJ_FINAL
OK	ARDUINO_IO_IN[11]	WEAK PULL-UP RESISTOR	ON	YES	RELOJ_FINAL
OK	ARDUINO_IO_IN[12]	WEAK PULL-UP RESISTOR	ON	YES	RELOJ_FINAL
OK	ARDUINO_IO_IN[13]	WEAK PULL-UP RESISTOR	ON	YES	RELOJ_FINAL
OK	ARDUINO_IO_IN[14]	WEAK PULL-UP RESISTOR	ON	YES	RELOJ_FINAL
OK	ARDUINO_IO_IN[15]	WEAK PULL-UP RESISTOR	ON	YES	RELOJ_FINAL
OK	ARDUINO_IO_OUT[0]	LOCATION	PIN_AB5	YES	-
OK	ARDUINO_IO_OUT[1]	LOCATION	PIN_AB6	YES	-
OK	ARDUINO_IO_OUT[2]	LOCATION	PIN_AB7	YES	-
OK	ARDUINO_IO_OUT[3]	LOCATION	PIN_AB8	YES	-
OK	ARDUINO_IO_OUT[4]	LOCATION	PIN_AB9	YES	-
OK	ARDUINO_IO_OUT[5]	LOCATION	PIN_Y10	YES	-
OK	ARDUINO_IO_OUT[6]	LOCATION	PIN_AA11	YES	-
OK	ARDUINO_IO_OUT[7]	LOCATION	PIN_AA12	YES	-
OK	ARDUINO_IO_IN[8]	LOCATION	PIN_AB17	YES	-
OK	ARDUINO_IO_IN[9]	LOCATION	PIN_AA17	YES	-
OK	ARDUINO_IO_IN[10]	LOCATION	PIN_AB19	YES	-
OK	ARDUINO_IO_IN[11]	LOCATION	PIN_AA19	YES	-
OK	ARDUINO_IO_IN[12]	LOCATION	PIN_Y19	YES	-
OK	ARDUINO_IO_IN[13]	LOCATION	PIN_AB20	YES	-
OK	ARDUINO_IO_IN[14]	LOCATION	PIN_AB21	YES	-
OK	ARDUINO_IO_IN[15]	LOCATION	PIN_AA20	YES	-

Tabla 8. Configuración de la asignación de los pines Arduino

En este apartado, es importante conocer que la mitad los pines del 0 al 7 estarán dispuestos como salidas y del 8 al 15 como entradas.

En esta configuración, como se puede comprobar en la tabla nº8, habrá diferencia con lo visto hasta ahora debido a que los pines configurados como entradas, se les tendrán que añadir una resistencia Pull-up, para obtener como resultado que cuando no esté pulsado de una señal lógica alta y en el momento que se presiona se consiga un nivel lógico bajo, con esto se evita que el ruido pueda interferir en los valores que puedan darnos distintos elementos como puede ser un pulsador.

Para conseguir obtener la resistencia Pull-up, se introducirá una línea por cada pin asignado como entrada, donde asignaremos la opción “Weak Pull-up resistor” con el valor “on” para activarlo.

Por otra parte, todos los pines deberán tener las dos líneas que hemos tenido que aplicar a cada elemento. Una de ellas con la localización de los pines y la otra indicando en la asignación la entrada/salida estándar con el valor 3,3V LVTTL. Debido a que tanto los pines ARDUINO_IO_IN como ARDUINO_IO_OUT se agrupan en vectores, por lo que se pueden configurar todos directamente, como se observa en la tabla.

4.1.4. Botones

ESTATUS	A	NOMBRE ASIGNACIÓN	VALOR	HABILITADO	ENTIDAD
OK	KEY0	I/O STANDARD	3.3 V Schmitt Trigger	YES	RELOJ_FINAL
OK	KEY1	I/O STANDARD	3.3 V Schmitt Trigger	YES	RELOJ_FINAL
OK	KEY0	LOCATION	PIN_B8	YES	-
OK	KEY1	LOCATION	PIN_A7	YES	-

Tabla 9. Configuración de la asignación de los botones

En los botones, como aspecto diferenciador al resto de elementos, podemos comprobar en la tabla nº9 que, en una de las líneas usadas para configurarlos, el valor de la asignación entrada/salida estándar se usa 3,3 V Schmitt Trigger. Su función es la de prevenir que el ruido interfiera y cambie los niveles lógicos que nos proporcionan ambos botones.

La otra fila es igual que lo visto en elementos anteriores, pero con el pin correspondiente a cada uno de los dos botones integrados en el hardware.

4.1.5. Displays

ESTATUS	A	NOMBRE ASIGNACIÓN	VALOR	HABILITADO	ENTIDAD
OK	HEX0[0]	LOCATION	PIN_C14	YES	-
OK	HEX0[1]	LOCATION	PIN_E15	YES	-
OK	HEX0[2]	LOCATION	PIN_C15	YES	-
OK	HEX0[3]	LOCATION	PIN_C16	YES	-
OK	HEX0[4]	LOCATION	PIN_E16	YES	-
OK	HEX0[5]	LOCATION	PIN_D17	YES	-
OK	HEX0[6]	LOCATION	PIN_C17	YES	-
OK	HEX0[7]	LOCATION	PIN_D15	YES	-
OK	HEX1[0]	LOCATION	PIN_C18	YES	-
OK	HEX1[1]	LOCATION	PIN_D18	YES	-
OK	HEX1[2]	LOCATION	PIN_E18	YES	-
OK	HEX1[3]	LOCATION	PIN_B16	YES	-
OK	HEX1[4]	LOCATION	PIN_A17	YES	-
OK	HEX1[5]	LOCATION	PIN_A18	YES	-
OK	HEX1[6]	LOCATION	PIN_B17	YES	-
OK	HEX1[7]	LOCATION	PIN_A16	YES	-
OK	HEX2[0]	LOCATION	PIN_B20	YES	-
OK	HEX2[1]	LOCATION	PIN_A20	YES	-
OK	HEX2[2]	LOCATION	PIN_B19	YES	-
OK	HEX2[3]	LOCATION	PIN_A21	YES	-
OK	HEX2[4]	LOCATION	PIN_B21	YES	-
OK	HEX2[5]	LOCATION	PIN_C22	YES	-
OK	HEX2[6]	LOCATION	PIN_B22	YES	-
OK	HEX2[7]	LOCATION	PIN_A19	YES	-
OK	HEX3[0]	LOCATION	PIN_F21	YES	-
OK	HEX3[1]	LOCATION	PIN_E22	YES	-
OK	HEX3[2]	LOCATION	PIN_E21	YES	-
OK	HEX3[2]	LOCATION	PIN_C19	YES	-
OK	HEX3[4]	LOCATION	PIN_C20	YES	-
OK	HEX3[5]	LOCATION	PIN_D19	YES	-
OK	HEX3[6]	LOCATION	PIN_E17	YES	-
OK	HEX3[7]	LOCATION	PIN_D22	YES	-
OK	HEX4[0]	LOCATION	PIN_F18	YES	-
OK	HEX4[1]	LOCATION	PIN_E20	YES	-
OK	HEX4[2]	LOCATION	PIN_E19	YES	-
OK	HEX4[3]	LOCATION	PIN_J18	YES	-
OK	HEX4[4]	LOCATION	PIN_H19	YES	-
OK	HEX4[5]	LOCATION	PIN_F19	YES	-
OK	HEX4[6]	LOCATION	PIN_F20	YES	-
OK	HEX4[7]	LOCATION	PIN_F17	YES	-
OK	HEX5[0]	LOCATION	PIN_J20	YES	-
OK	HEX5[1]	LOCATION	PIN_K20	YES	-
OK	HEX5[2]	LOCATION	PIN_L18	YES	-
OK	HEX5[3]	LOCATION	PIN_N18	YES	-
OK	HEX5[4]	LOCATION	PIN_M18	YES	-
OK	HEX5[5]	LOCATION	PIN_N19	YES	-

OK	HEX5[6]	LOCATION	PIN_N20	YES	-
OK	HEX5[7]	LOCATION	PIN_L19	YES	-
OK	HEX0[0]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX0[1]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX0[2]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX0[3]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX0[4]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX0[5]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX0[6]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX0[7]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX1[0]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX1[1]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX1[2]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX1[3]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX1[4]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX1[5]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX1[6]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX1[7]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX2[0]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX2[1]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX2[2]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX2[3]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX2[4]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX2[5]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX2[6]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX2[7]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX3[0]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX3[1]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX3[2]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX3[2]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX3[4]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX3[5]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX3[6]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX3[7]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX4[0]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX4[1]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX4[2]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX4[3]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX4[4]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX4[5]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX4[6]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX4[7]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX5[0]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX5[1]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX5[2]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX5[3]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX5[4]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX5[5]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL

OK	HEX5[6]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL
OK	HEX5[7]	I/O STANDARD	3.3-V LVTTL	YES	RELOJ_FINAL

Tabla 10. Configuración de la asignación de los displays

Por último, nos queda por configurar los displays. Estos elementos, no se diferencian de la configuración de los interruptores o la señal de reloj, por lo que, como hemos visto anteriormente, se crearan dos líneas por cada segmento de los 6 displays que dispone nuestra FPGA. En una de ellas se asignará la localización con los pines que le corresponde a cada uno y en el otro el voltaje cuando la asignación es la entrada/salida estándar.

4.2. LIBRERIAS

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
```

Ilustración 7. Librerías usadas

Cuando se va a realizar un proyecto con una FPGA, a la hora de realizar la implementación del código, es muy importante conocer las distintas posibilidades que nos pueden aportar el uso de distintas librerías, a través de su uso podemos conseguir funcionalidades que nos ayuden a realizar un mejor trabajo.

Un problema importante es que las librerías están normalmente desarrolladas por distintas empresas, por lo que el uso de una librería a la hora de realizar un código para una determinada placa puede provocar que dicho código no funcione en otra FPGA desarrollada por distinta empresa.

Es por esa razón que para la realización del trabajo he escogido solamente dos librerías que se utilizan en todas las FPGAs.

Para poder hacer hacer uso de las librerías, tenemos que instanciarlas, para ello se utiliza el código “library IEEE”, que nos permitirá usar las librerías que

posteriormente estén definidas previamente con la palabra “use” y posteriormente con “.all”.

En el caso de nuestro proyecto se ha usado solo dos:

- IEEE.STD_LOGIC_1164: Será utilizada para poder utilizar el tipo de dato STD_LOGIC, que puede tomar valores ‘U’, ‘X’, ‘0’, ‘1’, ‘Z’, ‘W’, ‘L’, ‘H’ y ‘-’.
- IEEE.NUMERIC_STD: Nos permite realizar operaciones aritméticas con signed (puede tomar valores enteros) y unsigned (puede tomar valores positivos), por lo que, si no tenemos este tipo de dato, debemos de realizar conversiones para poder hacer dichas operaciones.

4.3. ENTIDAD PRINCIPAL

En este punto describiremos la entidad principal con sus respectivas señales, puertos y componentes. Todos estos elementos quedan reflejados en la ilustración nº8.

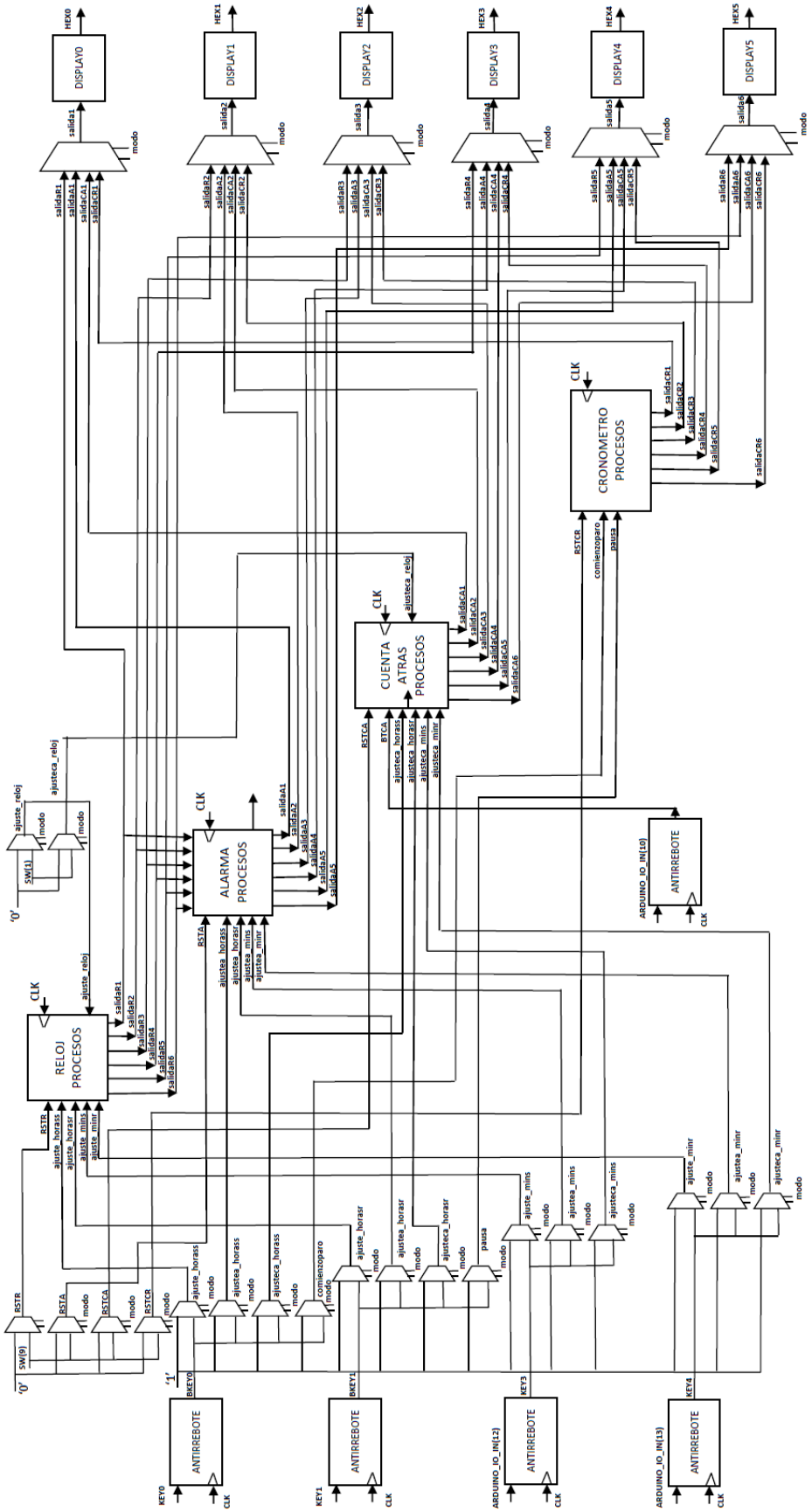


Ilustración 8. RTL entidad principal

4.3.1. Descripción de los puertos

```
entity RELOJ_FINAL is
   port     (
            SW : in std_logic_vector(9 downto 0);
            MAX10_CLK1_50 : in std_logic;
            KEY0 : in std_logic:='1';
            KEY1 : in std_logic;
            HEX0 : out std_logic_vector (7 downto 0);
            HEX1 : out std_logic_vector (7 downto 0);
            HEX2 : out std_logic_vector (7 downto 0);
            HEX3 : out std_logic_vector (7 downto 0);
            HEX4 : out std_logic_vector (7 downto 0);
            HEX5 : out std_logic_vector (7 downto 0);
            ARDUINO_IO_IN : in STD_LOGIC_VECTOR (15 downto 8);
            ARDUINO_IO_OUT : out STD_LOGIC_VECTOR (7 downto 0)
            );
end RELOJ_FINAL;
```

Ilustración 9. Entidad RELOJ_FINAL

Como entidad principal tendremos el RELOJ_FINAL, para la declaración de los puertos tendremos que usar la estructura que se observa en la ilustración nº9. Utilizando la palabra “port”, seguido de las entradas y salidas que tendrá la entidad entre paréntesis.

Aquí se tendrá que indicar si son entradas o salidas, utilizando las palabras reservadas “in” y “out”, al igual que se tendrá que indicar el tipo de dato que es. En nuestra entidad principal, podemos diferenciar:

- SW: Está configurada como entrada. Se trata de un vector binario y servirá para que puedan ser utilizados todos los interruptores que dispone nuestra FPGA.
- MAX10_CLK1_50: Está configurada como entrada de valor binario y nos proporcionará la señal de reloj que nos servirá como referencia.
- KEY0 y KEY1: Dos botones independientes de valor binario que están configurados como entradas.
- HEX0, HEX1, HEX2, HEX3, HEX4 y HEX5: Son independientes cada uno. Están configurados como salidas y se declaran como un vector binario para representar los 8 diodos de los que disponen cada uno de los 6 diplays.

- ARDUINO_IO_IN: Se trata de vector binario de 8 valores con los pines que, en este caso se han configurado como entradas.
- ARDUINO_IO_OUT: Se trata de vector binario de 8 valores con los pines que, en este caso se han configurado como salidas.

4.3.2. Señales

Una vez se han declarado las entradas y salidas de la entidad tendremos que diseñar cómo será la arquitectura y todas las interconexiones internas que utilizarán.

Para comenzar, en la ilustración nº8, se puede ver la distribución de los componentes que forman nuestro diseño en la entidad principal, las interconexiones internas se realizan mediante señales las cuales se pueden diferenciar en la ilustración ya mencionada.

En la ilustración nº 10 se puede ver todas las señales utilizadas.

```
signal salida1 : std_logic_vector(3 downto 0);
signal salida2 : std_logic_vector(3 downto 0);
signal salida3 : std_logic_vector(3 downto 0);
signal salida4 : std_logic_vector(3 downto 0);
signal salida5 : std_logic_vector(3 downto 0);
signal salida6 : std_logic_vector(3 downto 0);
signal salidaR1 : std_logic_vector(3 downto 0);
signal salidaR2 : std_logic_vector(3 downto 0);
signal salidaR3 : std_logic_vector(3 downto 0);
signal salidaR4 : std_logic_vector(3 downto 0);
signal salidaR5 : std_logic_vector(3 downto 0);
signal salidaR6 : std_logic_vector(3 downto 0);
signal salidaCR1 : std_logic_vector(3 downto 0);
signal salidaCR2 : std_logic_vector(3 downto 0);
signal salidaCR3 : std_logic_vector(3 downto 0);
signal salidaCR4 : std_logic_vector(3 downto 0);
signal salidaCR5 : std_logic_vector(3 downto 0);
signal salidaCR6 : std_logic_vector(3 downto 0);
signal salidaCA1 : std_logic_vector(3 downto 0);
signal salidaCA2 : std_logic_vector(3 downto 0);
signal salidaCA3 : std_logic_vector(3 downto 0);
signal salidaCA4 : std_logic_vector(3 downto 0);
signal salidaCA5 : std_logic_vector(3 downto 0);
signal salidaCA6 : std_logic_vector(3 downto 0);
signal salidaA1 : std_logic_vector(3 downto 0);
signal salidaA2 : std_logic_vector(3 downto 0);
signal salidaA3 : std_logic_vector(3 downto 0);
signal salidaA4 : std_logic_vector(3 downto 0);
signal salidaA5 : std_logic_vector(3 downto 0);
signal salidaA6 : std_logic_vector(3 downto 0);
signal ajuste_reloj : std_logic;
signal ajuste_horass : std_logic;
signal ajuste_mins : std_logic;
signal ajuste_horasr : std_logic;
signal ajuste_minr : std_logic;
signal comienzoparo : std_logic;
signal pausa : std_logic;
signal ajusteca_reloj : std_logic;
signal ajusteca_horass : std_logic;
signal ajusteca_mins : std_logic;
signal ajusteca_horasr : std_logic;
signal ajusteca_minr : std_logic;
signal ajustea_reloj : std_logic;
signal ajustea_horass : std_logic;
signal ajustea_mins : std_logic;
signal ajustea_horasr : std_logic;
signal ajustea_minr : std_logic;
signal RSTR : std_logic;
signal RSTCR : std_logic;
signal RSTCA : std_logic;
signal RSTA : std_logic;
signal BTCA : std_logic;
signal KEY3 : std_logic;
signal KEY4 : std_logic;
signal BKEY0 : std_logic;
signal BKEY1 : std_logic;
signal modo : std_logic_vector(1 downto 0);
```

Ilustración 10. Señales entidad principal

4.3.3. ANTIRREBOTE: Descripción, declaración e instanciación.

La función de esta entidad es la de evitar que el ruido interfiera en el nivel lógico que se desea obtener debido a la alta velocidad a la que se procesan los datos.

Para entender el módulo vamos a comenzar con la explicación de su código. En la siguiente ilustración vemos las entradas y salidas que tendrá la entidad.

```
entity ANTIRREBOTE is
   port  (
      CLK : in std_logic;
      TEC : in std_logic;
      BT : out std_logic
      );
end ANTIRREBOTE;

architecture ANTIRREBOTE_arch of ANTIRREBOTE is

   signal boton : std_logic;
   signal actboton : std_logic;
```

Ilustración 11. Entidad ANTIRREBOTE

Para la realización de la entidad hemos usado dos señales:

- botón: Será la señal que hará de enlace entre el proceso y la entrada BT.
- actboton: Será la señal que servirá de enlace entre el proceso que se definirá a continuación y la salida TEC.

Tras haber definido los elementos y las señales que se utilizarán en este proceso, tendremos que atender a cómo funciona la entidad en cuestión.

En toda entidad, las funciones que son un diseño secuencial se pueden abordar a través de procesos, los cuales nos permiten que estos componentes no se ejecuten de manera paralela como ocurre con los componentes combinacionales.

Para realizar cualquier función se tendrá que incluir la palabra “process” seguida de una lista de sensibilidad. Esta lista tendrá la utilidad de llamar al proceso en cuestión cuando las variables incluidas en ella cambien debido a que los procesos no se están ejecutando continuamente. En el proceso que se puede ver en la ilustración nº12 se ha utilizado la señal de reloj CLK.

En los procesos se pueden utilizar señales internas dentro del propio proceso, reciben el nombre de variables, también se puede declarar constantes que no variarán su valor durante el proceso.

- maxcount: Constante de tipo entero, cuyo valor es 5e7/10.
- count: Variable de tipo entero cuyo rango será de 0 a un número menos del definido en la constante maxcount.
- wait_rebote: Variable de tipo booleano. Se inicializa con el valor FALSE.

Una vez visto estos detalles de importancia, procedemos a explicar la función que realiza. Como hemos comentado previamente se introducirá en el proceso una vez que la señal CLK cambie su valor. Utilizando “rising_edge” se introducirá en el “if” únicamente cuando el flanco de reloj sea positivo.

Con esta función se pretende hacer un proceso secuencial que tarde un tiempo previamente calculado en procesar la información. En nuestro caso hemos querido hacer una espera de 100 milisegundos.

```
begin

BT<=boton;
actboton<=TEC;

process (CLK)
constant maxcount : integer := 5e7/10;
variable count : integer range 0 to maxcount-1 :=0;
variable wait_rebote : boolean :=false;
begin
   if rising_edge(CLK) then
      if not wait_rebote then
         if boton/=actboton then
            boton<=actboton;
            wait_rebote:=true;
            count:=0;
         end if;
      else
         count:=count+1;
         if count=maxcount-1 then
            wait_rebote:=false;
         end if;
      end if;
   end if ;
end process;

end ANTIRREBOTE_arch;
```

Ilustración 12. Proceso ANTIRREBOTE

Esta entidad se tendrá que declarar como componente dentro de otra que necesita realizar el proceso que hemos comentado previamente y cuyo objetivo era el de evitar que el ruido interfiera en el nivel lógico que se desea obtener debido a la alta velocidad a la que se procesan los datos.

La representación RTL del componente se puede ver en la ilustración nº8.

El componente se incluye en la entidad principal con la palabra “port” e introduciendo las entradas y salidas. Posteriormente, se instanciará los componentes donde se incluirán las señales para combinarlos con los elementos.

```
component ANTIRREBOTE is
   port     (
            CLK : in std_logic;
            TEC : in std_logic;
            BT : out std_logic
            );
end component;
```

Ilustración 13. Declaración componente ANTIRREBOTE

```
ANTIRREBOTE_BTCA : ANTIRREBOTE
   port map (MAX10_CLK1_50,not ARDUINO_IO_IN(10),BTCA);
ANTIRREBOTE_KEY1 : ANTIRREBOTE
   port map (MAX10_CLK1_50,KEY0,BKEY0);
ANTIRREBOTE_KEY2 : ANTIRREBOTE
   port map (MAX10_CLK1_50,KEY1,BKEY1);
ANTIRREBOTE_KEY3 : ANTIRREBOTE
   port map (MAX10_CLK1_50,ARDUINO_IO_IN(12),KEY3);
ANTIRREBOTE_KEY4 : ANTIRREBOTE
   port map (MAX10_CLK1_50,ARDUINO_IO_IN(13),KEY4);
```

Ilustración 14. Instanciación componentes ANTIRREBOTE

Este componente se usará en varias ocasiones debido a que en cada pulsador de la entidad principal lo utilizará, por lo que debemos realizar una instanciación por cada una de las veces que se use, es importante saber que cada instanciación es independiente a otra, aunque se use el mismo componente.

Para realizar una instanciación, debemos saber que se utilizará para definir la instanciación las palabras “port map”, seguido de la relación de las entradas y salidas declaradas en el componente y las señales descritas en la entidad.

Las relaciones explicadas de un modo general serían las siguientes:

- MAX10_CLK1_50 se asigna a CLK, ambas son de tipo lógico, esta asignación da la casualidad que es igual para todos los casos, y nos aporta la señal de reloj que nos proporciona la FPGA, en nuestro caso 50 MHz.
- KEY1, KEY0, ARDUINO_IO_IN(10), ARDUINO_IO_IN(12) y ARDUINO_IO_IN(13) se asignan a TEC, son de tipo lógico. Todas ellas son los botones definidos como entradas que se harán pasar por el ANTIRREBOTE.
- BTCA, BKEY0, BKEY1, KEY3 y KEY4 se asignan a BT, son de tipo lógico y serán la salida del ANTIRREBOTE, estas señales serán usadas por distintos componentes dentro de la entidad.

4.3.4. DISPLAY: Descripción, declaración e instanciación.

La función de este componente tendrá como función recibir un vector binario de 4 valores y dar otro de 8, cada uno de estos valores representa el valor de un led de los 8 que componen a cada uno de los displays.

En cuanto a la explicación del código de la entidad DISPLAY comenzaremos con conocer las entradas y salidas que tendrá esta entidad, como se muestra en la siguiente ilustración.

```
entity DISPLAY is
   port  (
         BIN : in STD_LOGIC_VECTOR (3 downto 0);
         D7SEG : out STD_LOGIC_VECTOR (7 downto 0)
         );
end DISPLAY;

architecture DISPLAY_arch of DISPLAY is
```

Ilustración 15. Entidad DISPLAY

Esta entidad no necesitará de señales intermedias debido a que se utilizarán directamente las entradas y salidas.

En este caso, la entidad es un proceso combinacional que consiste en la asignación de números binarios a los LEDs que componen un display. Los números que se pueden representar en un display es hasta el 9 por lo que en otro caso como se puede comprobar en la ilustración nº16 daría todos los LEDs a '1', por lo que estarían apagados.

```
begin
   with BIN select D7SEG <=
      "11000000" when "0000",
      "11111001" when "0001",
      "10100100" when "0010",
      "10110000" when "0011",
      "10011001" when "0100",
      "10010010" when "0101",
      "10000010" when "0110",
      "11111000" when "0111",
      "10000000" when "1000",
      "10010000" when "1001",
      "11111111" when others;

end DISPLAY_arch;
```

Ilustración 16. Proceso DISPLAY

Este componente se usará para cada uno de los displays de los que consta nuestra FPGA, recordemos que constaba de 6.

La representación RTL del componente se puede ver en la ilustración nº8.

Como se ha explicado para el componente ANTIRREBOTE, para declarar este componente, se hará de la misma manera, como podemos contemplar con las ilustraciones que se muestran a continuación.

```
component DISPLAY is
   port     (
            BIN : in STD_LOGIC_VECTOR (3 downto 0);
            D7SEG : out STD_LOGIC_VECTOR (7 downto 0)
            );
end component;
```

Ilustración 17. Declaración componente DISPLAY

```
Disp0: display
   port map (salida1,HEX0);
Disp1: display
   port map (salida2,HEX1);
Disp2: display
   port map (salida3,HEX2);
Disp3: display
   port map (salida4,HEX3);
Disp4: display
   port map (salida5,HEX4);
Disp5: display
   port map (salida6,HEX5);
```

Ilustración 18. Instanciación componentes DISPLAY

Para la instanciación del componente DISPLAY ocurre algo parecido con el de ANTIRREBOTE, debido a que se tendrá que hacer una instanciación para cada uno de los displays de los que dispone nuestra FPGA, con la ilustración nº17 y nº18 podemos ver las relaciones que se utilizarán:

- Salida1, Salida2, Salida3, Salida4, Salida5, Salida6 se asigna a BIN, son vectores de tipo lógico y se encargan de introducir el número binario que se pretende mostrar en el display.
- HEX1, HEX2, HEX3, HEX4, HEX5, HEX6 se asigna a D7SEG, cada uno representa un número de 8 bits para poder mostrar los segmentos necesarios dependiendo del número.

4.3.5. RELOJ_PROCESOS: Declaración e instanciación

Debido a la complejidad de esta entidad la descripción se explicará en una sección aparte.

Uno de los principales componentes de los que está compuesto la entidad principal es el RELOJ_PROCESO, aquí se llevará a cabo el proceso de contaje del reloj, también se podrá cambiar el valor mostrado correspondiente a las horas y a los minutos según lo desee el usuario.

La representación RTL del componente se puede ver en la ilustración nº8.

Como cualquier declaración de los componentes y como ya hemos visto en componentes anteriores se realizará como se indica en la ilustración nº19.

```
component RELOJ_PROCESOS is
   port     (
            AJ : in std_logic;
            RST : in std_logic;
            CLK : in std_logic;
            KEY0 : in std_logic;
            KEY1 : in std_logic;
            KEY2 : in std_logic;
            KEY3 : in std_logic;
            SALIDA0 : out std_logic_vector (3 downto 0);
            SALIDA1 : out std_logic_vector (3 downto 0);
            SALIDA2 : out std_logic_vector (3 downto 0);
            SALIDA3 : out std_logic_vector (3 downto 0);
            SALIDA4 : out std_logic_vector (3 downto 0);
            SALIDA5 : out std_logic_vector (3 downto 0)
            );
end component;
```

Ilustración 19. Declaración componente RELOJ_PROCESOS

```
cuentareloj : RELOJ_PROCESOS
   port map (ajuste_reloj,RSTR,MAX10_CLK1_50,ajuste_horass,ajuste_horasr,ajuste_mins,ajuste_minr,salidaR1,salidaR2,salidaR3,salidaR4,salidaR5,salidaR6);
```

Ilustración 20. Instanciación componentes RELOJ_PROCESOS

Para la instanciación, como se observa en la ilustración nº20, se realizan las siguientes asignaciones, estas se pueden entender mejor con la ayuda de la ilustración nº19:

- ajuste_reloj se asigna a AJ, ambas son de tipo lógico. A partir de esta señal se puede modificar la hora o a continuar con la secuencia.
- RSTR se asigna a RST, son de tipo lógico y su función es la de borrar cuando la señal de ajuste_reloj está dando un nivel lógico alto.
- MAX10_CLK1_50 se asigna a CLK, de tipo lógico, aporta la señal de reloj escogida en nuestra FPGA a la entidad, recordemos que era 50 MHz.
- ajuste_horass se asigna a KEY0, ambas de tipo lógico y a partir de ella se puede modificar de forma ascendente la parte de las horas.
- ajuste_horasr se asigna a KEY1, ambas de tipo lógico y a través de ella se puede modificar de forma descendente la parte de las horas.
- ajuste_mins se asigna a KEY2, ambas de tipo lógico y al igual que las señales utilizadas para modificar las horas, esta permite modificar de manera ascendente la parte de los minutos.
- ajuste_minr se asigna a KEY3, ambas de tipo lógico permiten modificar de manera descendente la parte de los minutos.
- SalidaR1 se asigna a SALIDA1, son vectores binarios y se encargan de dar a la salida1 la señal de salida.

- SalidaR2 se asigna a SALIDA2, son vectores binarios y se encargan de dar a la salida2 la señal de salida.
- SalidaR3 se asigna a SALIDA3, son vectores binarios y se encargan de dar a la salida3 la señal de salida.
- SalidaR4 se asigna a SALIDA4, son vectores binarios y se encargan de dar a la salida4 la señal de salida.
- SalidaR5 se asigna a SALIDA5, son vectores binarios y se encargan de dar a la salida5 la señal de salida.
- SalidaR6 se asigna a SALIDA6, son vectores binarios y se encargan de dar a la salida6 la señal de salida.

4.3.6. CUENTA_ATRAS_PROCESOS: Declaración e instanciación

Este componente realizará otra de las funciones importantes de nuestro reloj, será la de la cuenta atrás, mediante este componente se conseguirá que se pueda configurar tanto los minutos como los segundos que deseemos y comenzar la cuenta regresiva una vez se termine.

Esta entidad también es de gran complejidad por lo que se explicará en una sección aparte.

El diseño RTL se puede ver en la ilustración nº8.

En la ilustración nº21, podemos comprobar el componente irá incluido en la entidad principal.

```
component CUENTAATRAS_PROCESOS is
   port      (
             RST : in std_logic;
             CLK : in std_logic;
             AJUSTE : in std_logic;
             KEY0 : in std_logic;
             KEY1 : in std_logic;
             KEY3 : in std_logic;
             KEY4 : in std_logic;
             KEY2 : in std_logic;
             salida0 : out std_logic_vector (3 downto 0);
             salida1 : out std_logic_vector (3 downto 0);
             salida2 : out std_logic_vector (3 downto 0);
             salida3 : out std_logic_vector (3 downto 0);
             salida4 : out std_logic_vector (3 downto 0);
             salida5 : out std_logic_vector (3 downto 0)
             );
end component;
```

Ilustración 21. Declaración componente CUENTAATRAS_PROCESOS

```
cuentaatras : CUENTAATRAS_PROCESOS
   port map (RSTCA,MAX10_CLK1_50,ajusteca_reloj,ajusteca_horass,ajusteca_horasr,ajusteca_mins,ajusteca_minr,
             BTCA,salidaCA1,salidaCA2,salidaCA3,salidaCA4,salidaCA5,salidaCA6);
```

Ilustración 22. Instanciación componentes CUENTAATRAS_PROCESOS

En cuando a instanciar este componente se realizará de la misma manera que los anteriores. El proceso y la utilidad de los botones será muy parecido al RELOJ_PROCESOS.

- RSTCA se asigna a RST, son de tipo lógico y su función es la de poner a 0 todos los datos cuando la señal sea un nivel lógico alto.
- MAX10_CLK1_50 se asigna a CLK, de tipo lógico. Aporta la señal de reloj escogida en nuestra FPGA a la entidad, recordemos que era 50 MHz.
- ajusteca_reloj se asigna a AJUSTE, ambas son de tipo lógico. A partir de esta señal se puede modificar la parte de los minutos o habilitar el comienzo de la cuenta regresiva en el caso de no estar a cero.
- ajuste_horass se asigna a KEY0, ambas de tipo lógico y a partir de ella se puede modificar de forma ascendente la parte de las horas.
- ajuste_horasr se asigna a KEY1, ambas de tipo lógico y a través de ella se puede modificar de forma descendente la parte de las horas.
- ajuste_mins se asigna a KEY2, ambas de tipo lógico y al igual que las señales utilizadas para modificar las horas, esta permite modificar de manera ascendente la parte de los minutos.
- ajuste_minr se asigna a KEY3, ambas de tipo lógico permiten modificar de manera descendente la parte de los minutos.
- BTCA se asigna a KEY2, ambas de tipo lógico permiten iniciar la cuenta.
- SalidaCA1 se asigna a SALIDA1, son vectores binarios y se encargan de dar a la salida1 la señal de salida.
- SalidaCA2 se asigna a SALIDA2, son vectores binarios y se encargan de dar a la salida2 la señal de salida.
- SalidaCA3 se asigna a SALIDA3, son vectores binarios y se encargan de dar a la salida3 la señal de salida.
- SalidaCA4 se asigna a SALIDA4, son vectores binarios y se encargan de dar a la salida4 la señal de salida.

- SalidaCA5 se asigna a SALIDA5, son vectores binarios y se encargan de dar a la salida5 la señal de salida.
- SalidaCA6 se asigna a SALIDA6, son vectores binarios y se encargan de dar a la salida6 la señal de salida.

4.3.7. CRONOMETRO_PROCESOS: Declaración e instanciación

La descripción de entidad al igual que los otros dos grandes bloques que llevamos vistos hasta ahora se explicará en una sección aparte.

Este otro gran componente, tendrá como finalidad la de contar hasta los minutos, aquí irá incluido el proceso que realizará, así como las distintas posibilidades que se pueden hacer con él.

El diseño RTL se puede ver la ilustración nº8.

A continuación, en la siguiente ilustración se puede observar el componente que estará declarado en la entidad principal.

```
component CRONOMETRO_PROCESOS is
   port      (
             RST : in STD_LOGIC;
             CLK : in std_logic;
             KEY0 : in std_logic;
             KEY1 : in std_logic;
             SALIDA0 : out std_logic_vector (3 downto 0);
             SALIDA1 : out std_logic_vector (3 downto 0);
             SALIDA2 : out std_logic_vector (3 downto 0);
             SALIDA3 : out std_logic_vector (3 downto 0);
             SALIDA4 : out std_logic_vector (3 downto 0);
             SALIDA5 : out std_logic_vector (3 downto 0)
             );
end component;
```

Ilustración 23. Declaración componente CRONOMETRO_PROCESOS

```
cuentacrono : CRONOMETRO_PROCESOS
   port map (RSTCR,MAX10_CLK1_50,comienzoparo,pausa,salidaCR1,salidaCR2,salidaCR3,salidaCR4,salidaCR5,salidaCR6);
```

Ilustración 24. Instanciación componentes CUENTAATRAS_PROCESOS

Observando las dos ilustraciones anteriores para instanciar este componente:

- RSTCR se asigna a RST, son de tipo lógico y su función es la de poner los valores de salida a 0 cuando está dando un nivel lógico alto.

- MAX10_CLK1_50 se asigna a CLK, de tipo lógico. Aporta la señal de reloj escogida en nuestra FPGA a la entidad, recordemos que era 50 MHz.
- comienzoparo se asigna a KEY0, ambas de tipo lógico y a partir de ella se puede iniciar la cuenta y pararla cuando se desee hasta que se vuelva a activar esta señal.
- pausa, se asigna a KEY1, ambas de tipo lógico y a través de ella se detener la cuenta de forma temporal cuando reciba una señal de nivel lógico positivo y una vez lo de negativo mostrará la cuenta que nuca se ha detenido.
- SalidaCR1 se asigna a SALIDA0, son vectores binarios y se encargan de dar a la salida0 el valor de lo que será las centésimas de segundo.
- SalidaCR2 se asigna a SALIDA1, son vectores binarios y se encargan de dar a la salida1 el valor bde lo que será las décimas de segundo.
- SalidaCR3 se asigna a SALIDA2, son vectores binarios y se encargan de dar a la salida2 el valor de lo que será las unidades de segundo.
- SalidaCR4 se asigna a SALIDA3, son vectores binarios y se encargan de dar a la salida3 el valor de lo que será las decenas de segundo.
- SalidaCR5 se asigna a SALIDA4, son vectores binarios y se encargan de dar a la salida3 el valor de lo que será las unidades de minuto.
- SalidaCR6 se asigna a SALIDA5, son vectores binarios y se encargan de dar a la salida3 el valor de lo que será las decenas de minuto.

4.3.8. ALARMA_PROCESOS: Declaración e instanciación

Debido a la complejidad de esta entidad se describirá en una sección aparte.

El último gran bloque consistirá en ALARMA_PROCESOS, en este componente se puede configurar la hora a la que sonará nuestra alarma y una vez que coincida la hora del PROCESO_RELOJ con la que se ha configurado, se activará una señal que nos dará una melodía predeterminada.

Este componente también está incluido en el RTL que hemos estado mencionado en los componentes anteriores en la ilustración nº8.

Por último, en la ilustración nº25 podemos ver la declaración del componente que estamos tratando, recordemos que después tendremos que instanciarlo.

```
component ALARMA_PROCESOS is
    port      (
              clk : in std_logic;
              ACT : in std_logic;
              RST : in std_logic;
              ajuste_hhS : in std_logic;
              ajuste_mmS : in std_logic;
              ajuste_hhR : in std_logic;
              ajuste_mmR : in std_logic;
              datoMin01in : in std_logic_vector (3 downto 0);
              datoMin10in : in std_logic_vector (3 downto 0);
              datoHr01in : in std_logic_vector (3 downto 0);
              datoHr10in : in std_logic_vector (3 downto 0);
              datoSg01out : out std_logic_vector (3 downto 0);
              datoSg10out : out std_logic_vector (3 downto 0);
              datoMin01out : out std_logic_vector (3 downto 0);
              datoMin10out : out std_logic_vector (3 downto 0);
              datoHr01out : out std_logic_vector (3 downto 0);
              datoHr10out : out std_logic_vector (3 downto 0);
              MUSICA : out std_logic
              );
end component;
```

Ilustración 25. Declaración componente ALARMA_PROCESOS

```
alarm : ALARMA_PROCESOS
   port map (MAX10_CLK1_50,SW(0),RSTA,ajustea_horass,ajustea_mins,ajustea_horasr,ajustea_minr,
             salidaR3,salidaR4,salidaR5,salidaR6,salidaA1,salidaA2,salidaA3,salidaA4,salidaA5,salidaA6,
             ARDUINO_IO_OUT(7));
```

Ilustración 26. Instanciación componente ALARMA_PROCESOS

Para instanciar este componente que hemos definido en la ilustración nº26, definimos las siguientes asignaciones con la ayuda de la ilustración nº25.

- MAX10_CLK1_50 se asigna a CLK, de tipo lógico, aporta la señal de reloj escogida en nuestra FPGA a la entidad, recordemos que era 50 MHz.
- SW(0) se asigna a ACT, ambas de tipo lógico nos permite activar la alarma para que suene en el momento que coincide la hora marcada por el reloj con la fijada por este componente.
- RSTA se asigna a RST, son de tipo lógico y su función es la de inicializar todos los contadores a cero cuando haya un nivel lógico alto en esta señal.
- ajustea_horass se asigna a ajuste_hhS, ambas de tipo lógico y a partir de ella se puede modificar de forma ascendente la parte de las horas.
- ajustea_horasr se asigna a ajuste_mmS, ambas de tipo lógico y a través de ella se puede modificar de forma descendente la parte de las horas.

- ajustea_mins se asigna a ajuste_hhR, ambas de tipo lógico y al igual que las señales utilizadas para modificar las horas, esta permite modificar de manera ascendente la parte de los minutos.
- ajustea_minr se asigna a ajuste_mmR, ambas de tipo lógico permiten modificar de manera descendente la parte de los minutos.
- SalidaR1 se asigna a datoMin01in, son vectores binarios y se encargan de introducir en este componente el valor de las unidades de los minutos que nos proporciona el componente RELOJ_PROCESOS.
- SalidaR2 se asigna a datoMin10in, son vectores binarios y se encargan de introducir en este componente el valor de las decenas de los minutos que nos proporciona el componente RELOJ_PROCESOS.
- SalidaR3 se asigna a datoHr01in, son vectores binarios y se encargan de introducir en este componente el valor de las unidades de las horas que nos proporciona el componente RELOJ_PROCESOS.
- SalidaR4 se asigna a datoHr10in, son vectores binarios y se encargan de introducir en este componente el valor de las decenas de las horas que nos proporciona el componente RELOJ_PROCESOS.
- SalidaA1 se asigna a datoSg01out, son vectores binarios y se encargan de dar el valor de las unidades de los segundos que nos proporciona este componente.
- SalidaA2 se asigna a datoSg10out, son vectores binarios y se encargan de dar el valor de las decenas de los segundos que nos proporciona este componente.
- SalidaA3 se asigna a datoMin01out, son vectores binarios y se encargan de dar el valor de las decenas de los segundos que nos proporciona este componente.
- SalidaA4 se asigna a datoMin10out, son vectores binarios y se encargan de dar el valor de las decenas de los segundos que nos proporciona este componente.
- SalidaA5 se asigna a datoHr01out, son vectores binarios y se encargan de dar el valor de las decenas de los segundos que nos proporciona este componente.
- SalidaA6 se asigna a datoHr10out, son vectores binarios y se encargan de dar el valor de las decenas de los segundos que nos proporciona este componente.

4.4. Función RELOJ_PROCESOS

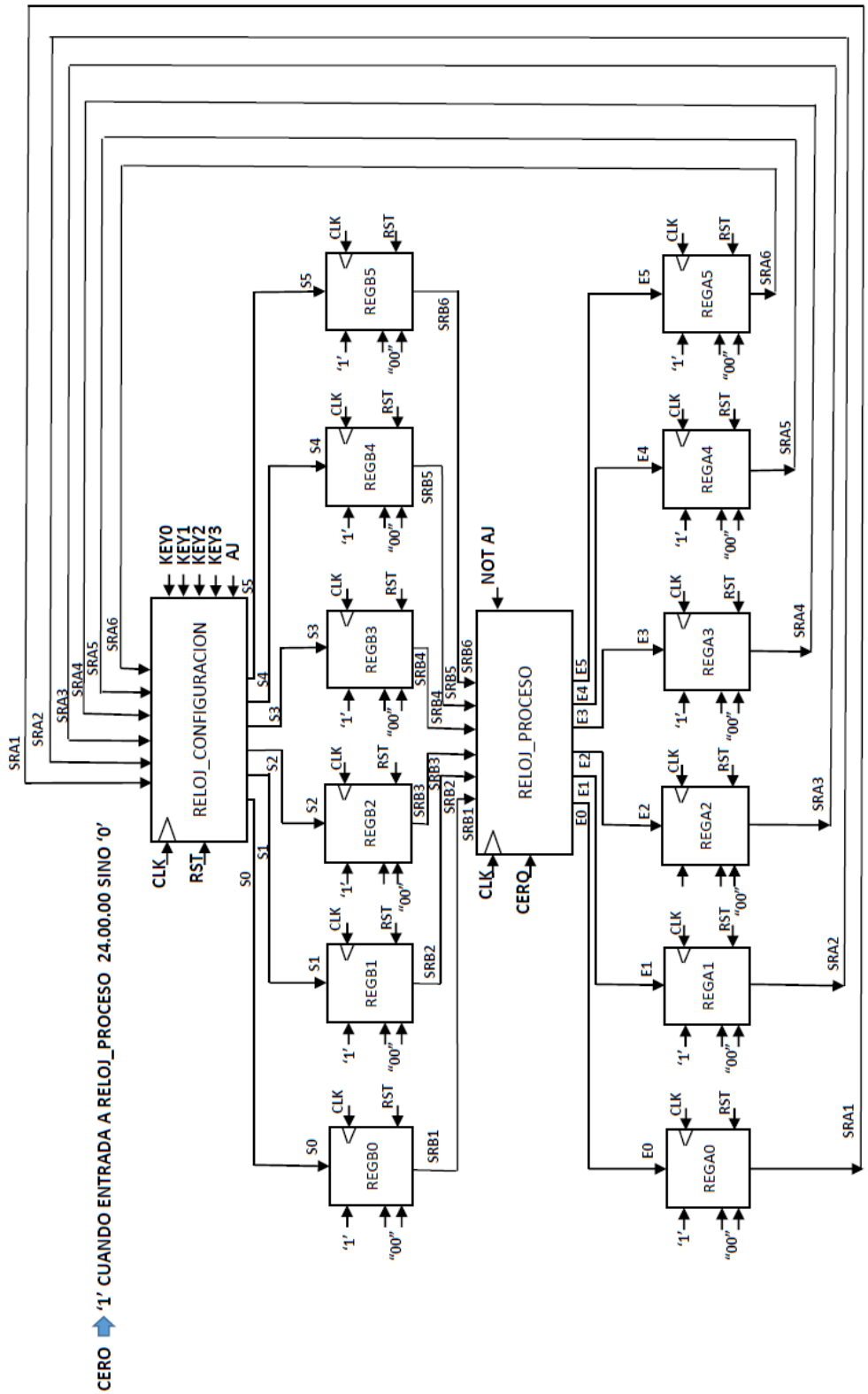


Ilustración 27. RTL entidad RELOJ_PROCESOS

En este diseño RTL de la entidad RELOJ_PROCESOS se puede ver de una forma más general las señales que a continuación profundizaremos un poco más.

```
entity RELOJ_PROCESOS is
   port     (
            AJ : in std_logic;
            RST : in std_logic;
            CLK : in std_logic;
            KEY0 : in std_logic;
            KEY1 : in std_logic;
            KEY2 : in std_logic;
            KEY3 : in std_logic;
            SALIDA0 : out std_logic_vector (3 downto 0);
            SALIDA1 : out std_logic_vector (3 downto 0);
            SALIDA2 : out std_logic_vector (3 downto 0);
            SALIDA3 : out std_logic_vector (3 downto 0);
            SALIDA4 : out std_logic_vector (3 downto 0);
            SALIDA5 : out std_logic_vector (3 downto 0)
            );
end RELOJ_PROCESOS;

architecture RELOJ_PROCESOS_arch of RELOJ_PROCESOS is

   signal E0 : std_logic_vector (3 downto 0);
   signal E1 : std_logic_vector (3 downto 0);
   signal E2 : std_logic_vector (3 downto 0);
   signal E3 : std_logic_vector (3 downto 0);
   signal E4 : std_logic_vector (3 downto 0);
   signal E5 : std_logic_vector (3 downto 0);
   signal SRA1 : std_logic_vector (3 downto 0);
   signal SRA2 : std_logic_vector (3 downto 0);
   signal SRA3 : std_logic_vector (3 downto 0);
   signal SRA4 : std_logic_vector (3 downto 0);
   signal SRA5 : std_logic_vector (3 downto 0);
   signal SRA6 : std_logic_vector (3 downto 0);
   signal SRB1 : std_logic_vector (3 downto 0);
   signal SRB2 : std_logic_vector (3 downto 0);
   signal SRB3 : std_logic_vector (3 downto 0);
   signal SRB4 : std_logic_vector (3 downto 0);
   signal SRB5 : std_logic_vector (3 downto 0);
   signal SRB6 : std_logic_vector (3 downto 0);
   signal S0 : std_logic_vector (3 downto 0);
   signal S1 : std_logic_vector (3 downto 0);
   signal S2 : std_logic_vector (3 downto 0);
   signal S3 : std_logic_vector (3 downto 0);
   signal S4 : std_logic_vector (3 downto 0);
   signal S5 : std_logic_vector (3 downto 0);
   signal CERO : std_logic;
```

Ilustración 28. Declaración entidad RELOJ_PROCESOS

En la ilustración anterior podemos ver las señales utilizadas para esta entidad, en las que cada una tiene una función específica:

- E0, E1, E2, E3, E4, E5: Estas señales son de tipo std_logic_vector de 4 bits. Los valores que representan se irán modificando en la medida que el componente encargado de ello vaya modificando dichos valores.
- SRA0, SRA1, SRA2, SRA3, SRA4, SRA5: Estas señales son de tipo std_logic_vector de 4 bits, cambiarán su valor conforme cambie el valor de entrada al registro.
- S0, S1, S2, S3, S4, S5: Estas señales son de tipo std_logic_vector de 4 bits, cambiarán su valor conforme se cambien en el componente encargado de modificar estos valores manualmente.
- SRB0, SRB1, SRB2, SRB3, SRB4, SRB5: Estas señales son de tipo std_logic_vector de 4 bits y al igual que las señales anteriores, al provenir de un registro cambian su valor conforme se modifique la entrada de dicho registro.
- CERO, señal de tipo lógico cuyo valor se puede ver reflejado en la ilustración nº27.

4.4.1. RELOJ_CONFIGURACIÓN: Declaración e instanciación

Debido a la complejidad de esta entidad, la descripción se realizará en una sección específica.

Este componente estará incluido en la entidad de RELOJ_PROCESOS, su función será la de modificar los valores de los minutos y horas de forma ascendente y descendente.

Para entender mejor como está conectado se puede ver el diseño RTL de la ilustración nº27.

```
component RELOJ_CONFIGURACION is
   port     (
            AJ : in std_logic;
            RST : in std_logic;
            CLK : in std_logic;
            KEY0 : in std_logic;
            KEY1 : in std_logic;
            KEY3 : in std_logic;
            KEY4 : in std_logic;
            EDAT00 : in std_logic_vector (3 downto 0);
            EDAT01 : in std_logic_vector (3 downto 0);
            EDAT02 : in std_logic_vector (3 downto 0);
            EDAT03 : in std_logic_vector (3 downto 0);
            EDAT04 : in std_logic_vector (3 downto 0);
            EDAT05 : in std_logic_vector (3 downto 0);
            SDAT00 : out std_logic_vector (3 downto 0);
            SDAT01 : out std_logic_vector (3 downto 0);
            SDAT02 : out std_logic_vector (3 downto 0);
            SDAT03 : out std_logic_vector (3 downto 0);
            SDAT04 : out std_logic_vector (3 downto 0);
            SDAT05 : out std_logic_vector (3 downto 0)
            );
end component;
```

Ilustración 29. Componente RELOJ_CONFIGURACION

```
confi : RELOJ_CONFIGURACION
   port map (AJ,RST,CLK,KEY0,KEY1,KEY2,KEY3,SRA1,SRA2,SRA3,SRA4,SRA5,SRA6,S0,S1,S2,S3,S4,S5);
```

Ilustración 30. Instanciación componente RELOJ_CONFIGURACION

La declaración de este componente dentro de una entidad de realiza como ya se ha explicado y se ve en la ilustración nº29.

Para la instanciación de éste componente recordemos que se utilizará para definir la instanciación las palabras “port map” seguido de las relaciones de las entradas y salidas declaradas en el componente y las señales descritas en la entidad como se puede ver en la ilustración nº30.

Las relaciones serían las siguientes:

- AJ se asigna a AJ, ambas son de tipo lógico. AJ es una entrada de la entidad RELOJ_CONFIGURACION.
- RST se asigna a RST, ambas son de tipo lógico, al igual que el caso anterior, RST es una entrada de la entidad RELOJ_CONFIGURACION.
- CLK se asigna a CLK, son de tipo lógico. CLK también es una entrada de la entidad RELOJ_CONFIGURACION.

- KEY0 se asigna a KEY0, ambas de tipo lógico, enlaza la entrada de la entidad con la entrada del este componente.
- KEY1 se asigna a KEY1, ambas de tipo lógico, enlaza la entrada de la entidad con la entrada del este componente.
- KEY2 se asigna a KEY3, ambas de tipo lógico, enlaza la entrada de la entidad con la entrada del este componente.
- KEY3 se asigna a KEY4, ambas de tipo lógico, enlaza la entrada de la entidad con la entrada del este componente del registro A.
- SRA1 se asigna a EDATO1, ambas son vectores binarios y es una de las señales que recibe este componente del registro A.
- SRA2 se asigna a EDATO2, ambas son vectores binarios y es una de las señales que recibe este componente del registro A.
- SRA3 se asigna a EDATO3, ambas son vectores binarios y es una de las señales que recibe este componente del registro A.
- SRA4 se asigna a EDATO4, ambas son vectores binarios y es una de las señales que recibe este componente del registro A.
- SRA5 se asigna a EDATO5, ambas son vectores binarios y es una de las señales que recibe este componente del registro A.
- SRA6 se asigna a EDATO6, ambas son vectores binarios y es una de las señales que recibe este componente del registro A.
- S0 se asigna a SDATO0, ambas son vectores binarios y son las señales que se proporcionarán al registro B.
- S1 se asigna a SDATO1, ambas son vectores binarios y son las señales que se proporcionarán al registro B.
- S2 se asigna a SDATO2 ambas son vectores binarios y son las señales que se proporcionarán al registro B.
- S3 se asigna a SDATO3, ambas son vectores binarios y son las señales que se proporcionarán al registro B.
- S4 se asigna a SDATO4, ambas son vectores binarios y son las señales que se proporcionarán al registro B.
- S5 se asigna a SDATO5, ambas son vectores binarios y son las señales que se proporcionarán al registro B.

4.4.2. RELOJ_PROCESO: Declaración e instanciación

Debido a la complejidad de la entidad, la descripción de la misma se realizará en una sección específica.

Este componente estará incluido en la entidad de RELOJ_PROCESOS, pero a diferencia del anterior, es el encargado de realizar la secuencia que se puede observar en cualquier reloj, también mostrará el cambio de valores del componente anterior cuando se produzca.

Para comprender mejor la situación de este componente en la entidad a la que pertenece y como se ha mencionado previamente, podemos ver el diseño RTL de la ilustración nº27.

```
component RELOJ_PROCESO is
   port     (
            CLK : in std_logic;
            AJUSTE : in std_logic;
            DATOSG01IN : in std_logic_vector (3 downto 0);
            DATOSG10IN : in std_logic_vector (3 downto 0);
            DATOMIN01IN : in std_logic_vector (3 downto 0);
            DATOMIN10IN : in std_logic_vector (3 downto 0);
            DATOHR01IN : in std_logic_vector (3 downto 0);
            DATOHR10IN : in std_logic_vector (3 downto 0);
            DATOSG01OUT : out std_logic_vector (3 downto 0) :=(others=>'0');
            DATOSG10OUT : out std_logic_vector (3 downto 0) :=(others=>'0');
            DATOMIN01OUT : out std_logic_vector (3 downto 0) :=(others=>'0');
            DATOMIN10OUT : out std_logic_vector (3 downto 0) :=(others=>'0');
            DATOHR01OUT : out std_logic_vector (3 downto 0) :=(others=>'0');
            DATOHR10OUT : out std_logic_vector (3 downto 0) :=(others=>'0');
            CERO : in std_logic
            );
end component;
```

Ilustración 31. Componente RELOJ_PROCESO

```
Proceso : RELOJ_PROCESO
   port map (CLK,not AJ,SRB1,SRB2,SRB3,SRB4,SRB5,SRB6,E0,E1,E2,E3,E4,E5,CERO);
```

Ilustración 32. Instanciación componente RELOJ_PROCESO

La declaración del componente dentro de la entidad donde se pretende usar, se realizará conforme se puede ver en la ilustración nº31.

Una vez declarado el componente se necesitará instanciarlo para ello con la ayuda de la ilustración nº32, se puede ver que se ha realizado las siguientes asignaciones:

- CLK se asigna a CLK, ambas son de tipo lógico. La señal CLK instanciada nos dará la señal de reloj a la que trabajará este componente.
- not AJ se asigna a AJUSTE, ambas son de tipo lógico, es una entrada que nos permitirá copiar los valores del componente RELOJ_CONFIGURACION o realizar la secuencia que realiza un reloj común.
- SRB1 se asigna a DATOSG01IN, ambas son vectores binarios y es una de las señales que recibe este componente del registro B0.
- SRB2 se asigna a DATOSG10IN, ambas son vectores binarios y es una de las señales que recibe este componente del registro B1.
- SRB3 se asigna a DATOMIN01IN, ambas son vectores binarios y es una de las señales que recibe este componente del registro B2.
- SRB4 se asigna a DATOMIN10IN, ambas son vectores binarios y es una de las señales que recibe este componente del registro B3.
- SRB5 se asigna a DATOHR01IN, ambas son vectores binarios y es una de las señales que recibe este componente del registro B4.
- SRB6 se asigna a DATOHR10IN, ambas son vectores binarios y es una de las señales que recibe este componente del registro B5.
- E0 se asigna a DATOSG01OUT, ambas son vectores binarios y es la señal que recibirá el registro A0.
- E1 se asigna a DATOSG10OUT, ambas son vectores binarios y es la señal que recibirá el registro A1.
- E2 se asigna a DATOMIN01OUT, ambas son vectores binarios y es la señal que recibirá el registro A2.
- E3 se asigna a DATOMIN10OUT, ambas son vectores binarios y es la señal que recibirá el registro A3.
- E4 se asigna a DATOHR01OUT, ambas son vectores binarios y es la señal que recibirá el registro A4.
- E5 se asigna a DATOHR10OUT, ambas son vectores binarios y es la señal que recibirá el registro A5.
- CERO asigna a CERO, ambas son de tipo lógico. Esta señal, como se indica en el RTL de la entidad en la ilustración nº27, dará un ‘1’ cuando en las salidas de los registros A, se tenga un 240000, sabiendo que SRA0

en el menos significativo y el SRA6 es el más significativo. Esta relación se muestra en la siguiente ilustración

```
cero <='1' when (SRA1="1001" and SRA2="0101" and SRA3="0101" and SRA4="1001" and SRA5="0011" and SRA6="0010") else '0';
```

Ilustración 33. Activación cero

4.4.3. REG_N: Descripción, declaración e instanciación

El uso esta entidad será muy diversa, concretamente dependerá de donde se vaya a ser instanciado posteriormente, debido a que en el interior puede realizar varias acciones, entre ellas estará la que usaremos en todas nuestras entidades, será la de copiar datos de entrada y sacarlos cuando se quiera por la salida.

```
entity REG_N is
   generic  (
            n_bits : integer := 4
            );
   port     (
            CLK : in STD_LOGIC;
            RST : in STD_LOGIC;
            WR : in STD_LOGIC;
            OP : in STD_LOGIC_VECTOR (1 downto 0);
            DIN : in  STD_LOGIC_VECTOR (n_bits-1 downto 0);
            DOUT : out  STD_LOGIC_VECTOR (n_bits-1 downto 0)
            );
end REG_N;

architecture REG_N_arch of REG_N is

   signal data : STD_LOGIC_VECTOR (n_bits-1 downto 0) := (others=>'0');
```

Ilustración 34. Entidad REG_N

Con la ilustración anterior vemos las entradas y salidas de la entidad y la señal que se utilizará.

- data, señal de tipo vector binario de tantos valores (por defecto será 4) como se indiquen en el generic una vez se instancie y su valor se modificará en el proceso.

```
process (CLK,RST)
begin
   if RST='1' then
      data<=(others=>'0');
   elsif  rising_edge(CLK) then
      if WR='1' then
         data<=DIN;
      else
         if OP="01" then
            data<=(others=>'0');
         elsif OP="10" then
            data<=std_logic_vector(unsigned(data)+1);
         elsif OP="11" then
            data<=std_logic_vector(unsigned(data)-1);
         end if;
      end if;
   end if;
end process;
DOUT<=data;

end REG_N_arch;
```

Ilustración 35. Proceso entidad REG_N

En la ilustración nº35 se observa que la lista de sensibilidad necesaria para introducirse en el proceso estará compuesta por CLK y RST.

El proceso realizará distintas operaciones dependiendo de la operación indicada (OP), siempre y cuando WR no esté a uno porque entonces únicamente copiaría el valor que recibe.

Para la instanciación en otras entidades, necesitaremos declararlo como componente tal y como se refleja en la ilustración nº36.

```
component REG_N is
   generic  (
            n_bits : integer := 4
            );
   port     (
            CLK : in STD_LOGIC;
            RST : in STD_LOGIC;
            WR : in STD_LOGIC;
            OP : in STD_LOGIC_VECTOR (1 downto 0);
            DIN : in  STD_LOGIC_VECTOR (n_bits-1 downto 0);
            DOUT : out  STD_LOGIC_VECTOR (n_bits-1 downto 0)
            );
end component;
```

Ilustración 36. Componente REG_N

```
RegA0 : REG_N
   generic map (4)
   port map (CLK,RST,'1',"00",E0,SRA1);
RegA1 : REG_N
   generic map (4)
   port map (CLK,RST,'1',"00",E1,SRA2);
RegA2 : REG_N
   generic map (4)
   port map (CLK,RST,'1',"00",E2,SRA3);
RegA3 : REG_N
   generic map (4)
   port map (CLK,RST,'1',"00",E3,SRA4);
RegA4 : REG_N
   generic map (4)
   port map (CLK,RST,'1',"00",E4,SRA5);
RegA5 : REG_N
   generic map (4)
   port map (CLK,RST,'1',"00",E5,SRA6);


RegB0 : REG_N
   generic map (4)
   port map (CLK,RST,'1',"00",S0,SRB1);
RegB1 : REG_N
   generic map (4)
   port map (CLK,RST,'1',"00",S1,SRB2);
RegB2 : REG_N
   generic map (4)
   port map (CLK,RST,'1',"00",S2,SRB3);
RegB3 : REG_N
   generic map (4)
   port map (CLK,RST,'1',"00",S3,SRB4);
RegB4 : REG_N
   generic map (4)
   port map (CLK,RST,'1',"00",S4,SRB5);
RegB5 : REG_N
   generic map (4)
   port map (CLK,RST,'1',"00",S5,SRB6);
```

Ilustración 37. Instanciación componente REG_N

La instanciación de este componente se realiza como se ve en la ilustración anterior, pero para podernos hacer una idea gráfica de su utilidad podemos ver el RTL de la ilustración nº27.

Estas instanciaciones tendrán en común las siguientes asignaciones:

- CLK asigna a CLK, ambas son de tipo lógico. Esta asignación servirá para que se use el reloj de entrada de la entidad en los componentes REG_N.
- RST asigna a RST, ambas de tipo lógico. Con esta asignación conseguiremos que la señal RST se active cuando lo haga el RST de entrada de la entidad.

- ‘1’ asigna a WR, ambas de tipo lógico. Conseguiremos que siempre le entre una señal lógica activa.
- “00” asigna a OP, ambas son vectores binarios y servirá para indicar un número específico binario en la entrada OP del registro en cuestión.

En cambio, cada una tiene una entrada y salida específica que será el número que desea ingresar en el registro.

- E0 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente RELOJ_PROCESO en el RegA0.
- E1 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente RELOJ_PROCESO en el RegA1.
- E2 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente RELOJ_PROCESO en el RegA2.
- E3 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente RELOJ_PROCESO en el RegA3.
- E4 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente RELOJ_PROCESO en el RegA4.
- E5 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente RELOJ_PROCESO en el RegA5.
- SRA1 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará el RegA0 a la SALIDA0 y a la entrada del componente RELOJ_CONFIGURACIÓN.
- SRA2 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará el RegA1 a la SALIDA1 y a la entrada del componente RELOJ_CONFIGURACIÓN.
- SRA3 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará RegA2 a la SALIDA2 y a la entrada del componente RELOJ_CONFIGURACIÓN.
- SRA4 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará RegA3 a la SALIDA3 y a la entrada del componente RELOJ_CONFIGURACIÓN.

- SRA5 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará el RegA4 a la SALIDA4 y a la entrada del componente RELOJ_CONFIGURACIÓN.
- SRA6 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará el registro a la SALIDA5 y a la entrada del componente RELOJ_CONFIGURACIÓN.
- S0 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente RELOJ_CONFIGURACION en el RegB0.
- S1 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente RELOJ_ CONFIGURACION en el RegB1.
- S2 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente RELOJ_ CONFIGURACION en el RegB2.
- S3 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente RELOJ_ CONFIGURACION en el RegB3.
- S4 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente RELOJ_ CONFIGURACION en el RegB4.
- S5 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente RELOJ_ CONFIGURACION en el RegB5.
- SRB1 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará el RegA0 a la entrada del componente RELOJ_PROCESO.
- SRB2 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará el RegA1 a la entrada del componente RELOJ_PROCESO.
- SRB3 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará RegA2 y a la entrada del componente RELOJ_ PROCESO.
- SRB4 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará RegA3 a la entrada del componente RELOJ_ PROCESO.
- SRB5 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará el RegA4 a la entrada del componente RELOJ_PROCESO.
- SRB6 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará el RegA5 a la entrada del componente RELOJ_PROCESO.

4.5. Función RELOJ_CONFIGURACION

La función principal de esta entidad como se ha comentado brevemente será la de modificar los valores correspondientes a las horas y minutos cuando sea posible mediante los procesos que componente esta entidad. Dependiendo de cómo estén configuradas las entradas también podrá dar como salida los valores que recibe esta entidad.

```
entity RELOJ_CONFIGURACION is
   port      (
             AJ : in std_logic;
             RST : in std_logic;
             CLK : in std_logic;
             KEY0 : in std_logic;
             KEY1 : in std_logic;
             KEY3 : in std_logic;
             KEY4 : in std_logic;
             EDATO0 : in std_logic_vector (3 downto 0);
             EDATO1 : in std_logic_vector (3 downto 0);
             EDATO2 : in std_logic_vector (3 downto 0);
             EDATO3 : in std_logic_vector (3 downto 0);
             EDATO4 : in std_logic_vector (3 downto 0);
             EDATO5 : in std_logic_vector (3 downto 0);
             SDATO0 : out std_logic_vector (3 downto 0);
             SDATO1 : out std_logic_vector (3 downto 0);
             SDATO2 : out std_logic_vector (3 downto 0);
             SDATO3 : out std_logic_vector (3 downto 0);
             SDATO4 : out std_logic_vector (3 downto 0);
             SDATO5 : out std_logic_vector (3 downto 0)
             );
end RELOJ_CONFIGURACION;

architecture RELOJ_CONFIGURACION_arch of RELOJ_CONFIGURACION is

   signal uu_s : std_logic_vector (3 downto 0);
   signal dd_s : std_logic_vector (3 downto 0);
   signal E2 : std_logic_vector (3 downto 0);
   signal E3 : std_logic_vector (3 downto 0);
   signal E4 : std_logic_vector (3 downto 0);
   signal E5 : std_logic_vector (3 downto 0);
   signal S2 : std_logic_vector (3 downto 0);
   signal S3 : std_logic_vector (3 downto 0);
   signal S4 : std_logic_vector (3 downto 0);
   signal S5 : std_logic_vector (3 downto 0);
```

Ilustración 38. Entidad RELOJ_CONFIGURACION

Esta entidad contará con las entradas y salidas que pueden verse en la ilustración nº38. En cuanto a las señales utilizadas podemos destacar:

- uu_s, señal de tipo vector binario de 4 elementos representa el valor de las unidades de segundo.
- dd_s, señal de tipo vector binario de 4 elementos representa el valor de las decenas de los segundos.

- E2, E3, ambas son de tipo vector binario de 4 elementos y serán las señales de entrada del componente PARTE_59.
- E4, E5, ambas son de tipo vector binario de 4 elementos y serán las señales de entrada del componente PARTE_23.
- S2, S3, ambas son de tipo vector binario de 4 elementos y serán las salidas del componente PARTE_59.
- S4, S5, ambas son de tipo vector binario de 4 elementos y serán las salidas del componente PARTE_23.

4.5.1. Proceso valores segundos

Esta entidad contará con un proceso para los valores correspondientes a los segundos debido a que en los componentes instanciados como se verá más adelante, no podrán modificarse los segundos.

```
process (RST)
begin
   if RST ='0' then
   uu_s <= EDATO0;
   dd_s <= EDATO1;
   else
   uu_s <="0000";
   dd_s <="0000";
   end if;
SDATO0 <= uu_s;
SDATO1 <= dd_s;
end process;
```

Ilustración 39. Proceso de uu_s y dd_s

En esta ilustración se muestra que la lista de sensibilidad del proceso es el RST, por lo que cambiará cada vez que esta modifica su valor. En cuanto al proceso que realiza, podemos comprobar que si RST es ‘0’ la señales uu_s y dd_s copian el valor y sino ambas señales dan un ‘0’, una vez cerrados todos los “if” se copia el valor a las salidas de la entidad.

4.5.2. PARTE_23: Descripción, declaración e instanciación.

Para modificar los valores de las unidades y decenas correspondiente a las horas se instanciará el bloque PARTE_23. esta entidad consigue sumar hasta 23 y en el caso de la resta una vez pase 0 vuelve a 23.

La entradas y salidas que compondrán nuestra entidad se ven en la ilustración siguiente.

```
entity PARTE_23 is
   port      (
             CLK : in std_logic;
             RST : in std_logic;
             AJUSTE : in std_logic;
             AJUSTE_S : in std_logic;
             AJUSTE_R : in std_logic;
             EDECENAS : in std_logic_vector (3 downto 0);
             EUNIDADES : in std_logic_vector (3 downto 0);
             SDECENAS : out std_logic_vector (3 downto 0) :=(others=>'0');
             SUNIDADES : out std_logic_vector (3 downto 0) :=(others=>'0')
             );
end PARTE_23;

architecture PARTE_23_arch of PARTE_23 is

   signal UNIDADES : integer range 0 to 9:=0;
   signal DECENAS : integer range 0 to 9:=0;
   signal clk_U : std_logic := '0';
   signal CLK_D : std_logic;
   signal SRU0,SRU1 : std_logic;
   signal SRD0,SRD1 : std_logic;
   signal SRD : std_logic_vector(1 downto 0);
   signal SRU : std_logic_vector(1 downto 0);
   signal CLK2,CLK3 : std_logic := '0';
   signal RESET : std_logic := '0';
   signal RESETD : std_logic := '0';
```

Ilustración 40. Entidad PARTE_23

En cuanto a las señales utilizadas podemos destacar:

- UNIDADES, se trata de un valor entero de hasta 10 valores, será la señal utilizada para cambiar el valor de las unidades durante los procesos.
- DECENAS, se trata de un valor entero de hasta 10 valores, será la señal utilizada para cambiar el valor de las unidades durante los procesos.
- CLK_U, se trata de una señal tipo lógico nos la proporcionará el componente CONFIGURACION_FSM y generará un pulso de reloj en el proceso de suma y resta de las unidades.
- CLK_D, se trata de una señal tipo lógico que generará a través del componente CONFIGURACION_FSM un pulso de reloj en el proceso de suma y resta de las decenas.
- SRD0, SRD1, se trata de dos señales de tipo lógico procedentes del componente CONFIGURACION_FSM, que nos proporcionará los valores necesarios para obtener la señal SRD

- SRU0, SRU1, se trata de dos señales de tipo lógico procedentes del componente CONFIGURACION_FSM, que nos proporcionará los valores necesarios para obtener la señal SRU
- SRD, se trata de una señal tipo vector binario de dos elementos cuyo bit más significativo es SRD1 y el menos significativo SRD0 como se observa en la ilustración nº41.
- SRU, se trata de una señal tipo vector binario de dos elementos cuyo bit más significativo es SRU1 y el menos significativo SRU0 como se observa en la ilustración nº41.
- CLK2, se trata de una señal tipo lógico, que dará un nivel lógico positivo cuando las unidades sean 0, sino dará un '0', se puede ver con claridad en la ilustración nº41.
- CLK3, se trata de una señal tipo lógico, y será una señal de entrada a la máquina de estados. Esta señal variará en los procesos de la entidad.

```
CLK2 <= '1' when UNIDADES=0 else '0' when UNIDADES/=0 else '0';
SRU <= "10" when SRU1='1' and SRU0='0' else "01" when SRU1='0' and SRU0='1' else "11" when SRU1='1' and SRU0='1'else "00";
SRD <= "10" when SRD1='1' and SRD0='0' else "01" when SRD1='0' and SRD0='1' else "11" when SRD1='1' and SRD0='1'else "00";
```

Ilustración 41. Señales CLK2, SRU y SRD

4.5.2.1. CONFIGURACION_FSM: Definición, declaración e instanciación

Para el control de las entradas y salidas de las entidades PARTE_23 y PARTE_59 requeriremos de una máquina de estados. En la ilustración nº42 podemos ver las entradas y salidas que tendrá nuestro controlador a través de la máquina de estados y cuyas salidas se enlazarán con los procesos, en la ilustración nº43 se muestra como quedarían implementadas en la entidad.

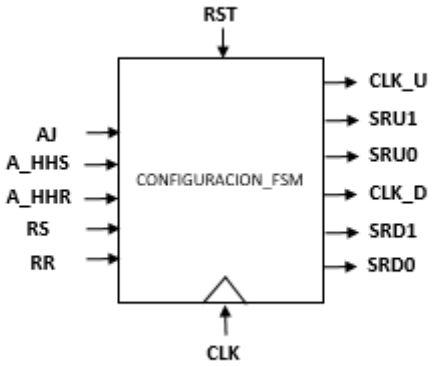


Ilustración 42. RTL CONFIGURACION_FSM

```
entity CONFIGURACION_FSM is
   port     (
            CLK : in STD_LOGIC;
            RST : in STD_LOGIC;
            AJ : in STD_LOGIC;
            A_HHS : in STD_LOGIC;
            A_HHR : in STD_LOGIC;
            RS : in STD_LOGIC;
            RR : in STD_LOGIC;
            CLK_U : out STD_LOGIC;
            SRU1 : out STD_LOGIC;
            SRU0 : out STD_LOGIC;
            CLK_D : out STD_LOGIC;
            SRD1 : out STD_LOGIC;
            SRD0 : out STD_LOGIC
            );
end CONFIGURACION_FSM;

architecture CONFIGURACION_FSM_arch of CONFIGURACION_FSM is

   TYPE ESTADOS is (Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9,Q10);
   signal ESTADO : ESTADOS := Q0;
   signal funcion : std_logic_vector (5 downto 0);
```

Ilustración 43. Entidad CONFIGURACION_FSM

En las señales que se observan podemos destacar:

- ESTADO, la función de esta señal será la de asignar los valores que se han definido en la línea “TYPE”.
- funcion, se trata de vector binario de 6 elementos cuyo valor irá asignado a los diferentes estados definidos en “TYPE”.

Antes de explicar que realizará la máquina de estados es importante saber que las entradas y salidas de cada estado son las que podemos ver en la ilustración nº44, que ya hemos definido al comienzo de la entidad.

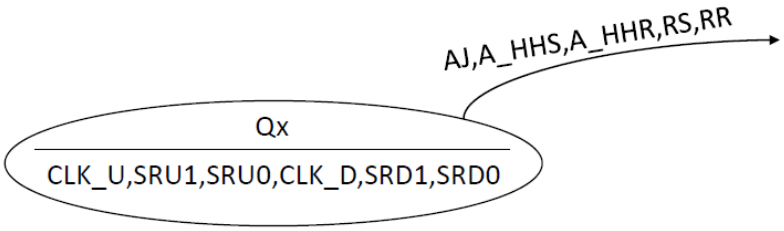


Ilustración 44. Entradas y salidas máquina Moore CONFIGURACION_FSM

Ilustración 45. Máquina de Moore CONFIGURACION_FSM

En la máquina de estados podemos distinguir un total de 11 estados, explicaremos las salidas de cada uno en función de las entradas que recibe.

- Q0, es el estado inicial, en este estado no se activa las señales CLK_U y CLK_D, pero se configura SRU y SRD como “11”. Se llegará a él siempre que AJ o RST sean ‘0’ y ‘1’ respectivamente.
- Q1, en este estado se activarán las señales de reloj que anteriormente estaban desactivadas, manteniendo la configuración de SRU y SRD. Siempre llegará desde Q0 sin condición ninguna.
- Q2 y Q9, a estos estados se puede llegar dependiendo si se pulsan los botones de sumar o restar, se preparará la señal SRU para que en el siguiente estado junto con la señal CLK_U se activen.
- Q3, llegamos a este estado y se activa la señal de reloj con el SRU que se ha configurado previamente.
- Q4, a este estado se llega si la señal RS se activa, entonces se activará la señal CLK_D con la configuración SRD que en el estado anterior se ha preparado.
- Q5, a este estado se llegará cuando el ajuste desde el estado Q4 se pone a 0 o se suelta el botón y desde Q3 si la señal RS es 0. Desde este estado se puede ir a Q0 si se pone a 0 la señal AJ, a Q2 si se pulsa el botón de restar o Q9 si es el de sumar.
- Q6, en este estado ocurre lo mismo que en el estado Q3, pero con la preparación de SRU para restar.
- Q10, en este estado, se da la señal de reloj que junto con la configuración de SRU puesta en el estado anterior se resta uno.
- Q7, en este estado se llega cuando se activa la señal CLK_D por lo que se restará con la configuración de SRD que se ha configurado en el estado anterior.
- Q8, a este estado se llega desde Q10 cuando SR está a 0 y está soltado el botón de restar o desde Q7 cuando se suelta el botón.

En la ilustración nº46 podemos ver implementada la máquina de Moore de nuestro controlador en VHDL.

```
process (CLK,RST)
begin
   if RST = '1' then
      ESTADO<=Q0;
   elsif rising_edge (CLK) then
      case ESTADO is
         when Q0 =>
               ESTADO<=Q1;
         when Q1 =>
            if AJ='1' and A_HHS='1' and A_HHR='0'then
               ESTADO<=Q9;
            elsif AJ='1' and A_HHS='0' and A_HHR='1'then
               ESTADO<=Q2;
            elsif AJ='0' then
               ESTADO<=Q0;
            else
               ESTADO<=Q1;
            end if;
         when Q2 =>
            if AJ='1' and A_HHS='0' and A_HHR='1'then
               ESTADO<=Q6;
            elsif AJ='0' then
               ESTADO<=Q0;
            else
               ESTADO<=Q2;
            end if;
         when Q9 =>
            if AJ='1' and A_HHS='1' and A_HHR='0' then
               ESTADO<=Q3;
            elsif AJ='0' then
               ESTADO<=Q0;
            else
               ESTADO<=Q9;
            end if;
         when Q3 =>
            if A_HHS='0' and RS='0' then
               ESTADO<=Q5;
            elsif RS='1' then
               ESTADO<=Q4;
            else
               ESTADO<=Q3;
            end if;
         when Q4 =>
            if A_HHS='0' then
               ESTADO<=Q5;
            else
               ESTADO<=Q4;
            end if;
         when Q5 =>
            if AJ='1' and A_HHS='1' and A_HHR='0' then
               ESTADO<=Q9;
            elsif AJ='1' and A_HHS='0' and A_HHR='1'then
               ESTADO<=Q2;
            elsif AJ='0' then
               ESTADO<=Q0;
            else
               ESTADO<=Q5;
            end if;
```

```
            when Q6 =>
                if RR='0' then
                    ESTADO<=Q10;
                elsif RR = '1' then
                    ESTADO<=Q7;
                else
                    ESTADO<=Q6;
                end if;
            when Q10 =>
                if A_HHR='0' and RR='0' then
                    ESTADO<=Q8;
                else
                    ESTADO<=Q10;
                end if;
            when Q7 =>
                if A_HHR='0'then
                    ESTADO<=Q8;
                else
                    ESTADO<=Q7;
                end if;
            when Q8 =>
                if AJ='1' and A_HHS='1' and A_HHR='0' then
                    ESTADO<=Q9;
                elsif AJ='1' and A_HHS='0' and A_HHR='1' then
                    ESTADO<=Q2;
                elsif AJ='0' then
                    ESTADO<=Q0;
                else
                    ESTADO<=Q8;
                end if;
        end case;
    end if;
end process;

CLK_U <= funcion(5);
SRU1 <= funcion (4);
SRU0 <= funcion (3);
CLK_D <= funcion(2);
SRD1 <= funcion (1);
SRD0 <= funcion (0);


with ESTADO select funcion <=

    "011011" when Q0,
    "111111" when Q1,
    "001000" when Q2,
    "110010" when Q3,
    "100110" when Q4,
    "011011" when Q5,
    "101001" when Q6,
    "100101" when Q7,
    "011011" when Q8,
    "010000" when Q9,
    "011011" when Q10;

end CONFIGURACION_FSM_arch;
```

Ilustración 46. Proceso CONFIGURACION_FSM

La lista de sensibilidad del proceso estará compuesta por la entrada CLK y RST de la entidad. En cuanto al propio proceso cabe destacar lo intuitivo del mismo debido a que en cada caso, definidos por la palabra “when” indicamos el estado que vamos a definir para que con los “if” se indiquen los estados siguientes.

```
component CONFIGURACION_FSM is
   port     (
            CLK : in STD_LOGIC;
            RST : in STD_LOGIC;
            AJ : in STD_LOGIC;
            A_HHS : in STD_LOGIC;
            A_HHR : in STD_LOGIC;
            RS : in STD_LOGIC;
            RR : in STD_LOGIC;
            CLK_U : out STD_LOGIC;
            SRU1 : out STD_LOGIC;
            SRU0 : out STD_LOGIC;
            CLK_D : out STD_LOGIC;
            SRD1 : out STD_LOGIC;
            SRD0 : out STD_LOGIC
            );
end component;
```

Ilustración 47. Componente CONFIGURACION_FSM

```
FSM_MIN : CONFIGURACION_FSM
 port map (CLK,RST,AJUSTE,AJUSTE_S,AJUSTE_R,CLK2,CLK3,CLK_U,SRU1,SRU0,CLK_D,SRD1,SRD0);
```

Ilustración 48. Instanciación componente CONFIGURACION_FSM

En la ilustración nº47 se puede observar cuáles serán las entradas y salidas del componente CONFIGURACION_FSM, con este componente se pretenderá controlar las señales que enlazarán con los procesos que están incluidos en esta entidad.

En la ilustración nº48 se observa cuáles serán las señales instanciadas que ahora analizaremos más detenidamente:

- CLK se asigna a CLK, ambas de tipo lógico, con esto se consigue que el reloj con el que ejecutará la máquina será el de la entidad.
- RST se asigna a RST, ambas de tipo lógico, con esta instanciación se consigue que está entrada del componente tenga el mismo valor que la entrada RST de la entidad.
- AJUSTE se asigna a AJ, ambas de tipo lógico, será una de las entradas que modifiquen los estados de nuestra máquina de estados.

- AJUSTE_S se asigna a A_HHS, ambas de tipo lógico, será una de las entradas que modifiquen los estados de nuestra máquina de estados.
- AJUSTE_R se asigna a A_HHR, ambas de tipo lógico, será una de las entradas que modifiquen los estados de nuestra máquina de estados.
- CLK2 se asigna a RS, ambas de tipo lógico, será una de las entradas que modifiquen los estados de nuestra máquina de estados, su valor de entrada se ha comentado previamente.
- CLK3 se asigna a RR, ambas de tipo lógico, será una de las entradas que modifiquen los estados de nuestra máquina de estados, su valor se modifica en el proceso de suma resta de las unidades que se hablará en el siguiente punto.
- CLK_U se asigna a CLK_U, ambas de tipo lógico, será una de las salidas de nuestra máquina de estados, su valor dará la señal de reloj por la que se modificará el proceso suma y resta de unidades.
- SRU1 se asigna a SRU1, ambas de tipo lógico, será una de las salidas de nuestra máquina de estados, su valor será parte de la señal SRU como ya se ha comentado.
- SRU0 se asigna a SRU0, ambas de tipo lógico, será una de las salidas de nuestra máquina de estados, su valor será parte de la señal SRU como ya se ha comentado.
- CLK_D se asigna a CLK_D, ambas de tipo lógico, será una de las salidas de nuestra máquina de estados, su valor dará la señal de reloj por la que se modificará el proceso suma y resta de las decenas.
- SRD1 se asigna a SRD1, ambas de tipo lógico, será una de las salidas de nuestra máquina de estados, su valor será parte de la señal SRD como ya se ha comentado.
- SRD0 se asigna a SRD0, ambas de tipo lógico, será una de las salidas de nuestra máquina de estados, su valor será parte de la señal SRD como ya se ha comentado.

4.5.2.2. Proceso suma y resta de unidades

En este proceso se realizará la suma y resta de las unidades, hasta el número que se quiera obtener pues este proceso será exactamente el mismo que se usará

para los minutos y horas, variando unos detalles que comentaremos. El proceso se puede ver en la ilustración nº49.

```
process (RST,CLK_U,SRU)
begin
if RST = '1' then
   UNIDADES <= 0;
elsif  CLK_U'event and CLK_U = '1' then
   if SRU = "11" then
   UNIDADES <=TO_INTEGER(unsigned(EUNIDADES));
   elsif SRU = "10" then
   UNIDADES <= UNIDADES + 1;
      if DECENAS = 2 and UNIDADES = 3 then
         UNIDADES <= 0;
         RESET <= '1';
      elsif UNIDADES = 9 then
         UNIDADES <= 0;
         RESET <= '0';
      end if;
   elsif SRU = "01"then
      if UNIDADES /= 0 then
      UNIDADES <= UNIDADES - 1;
         CLK3 <= '0';
         RESETD <= '0';
      elsif UNIDADES = 0 and DECENAS = 0 then
         UNIDADES <= 3;
         RESETD <= '1';
         CLK3 <= '1';
      elsif UNIDADES = 0 then
         UNIDADES <= 9;
         RESETD<='0';
         CLK3 <= '1';
      end if;
   else
   UNIDADES <=TO_INTEGER(unsigned(EUNIDADES));
   end if;
end if;
SUNIDADES <= std_logic_vector(to_unsigned(UNIDADES,4));
end process;
```

Ilustración 49. Proceso suma y resta de unidades

La lista de sensibilidad, que recordemos que sirve para realizar el proceso una vez uno de los componentes de la lista de sensibilidad cambia de valor, estará compuesta por las señales RST, CLK_U y SRU que en los puntos anteriores hemos conocido cuando cambian de valor.

En el proceso descrito en la ilustración nº49 se puede ver que cuando recibimos una señal de reloj con flanco ascendente, dependiendo del valor SRU podemos copiar el valor que se reciba de las entradas de la entidad, sumar el valor y una vez llegue a 9 se volverá a 0 activando con ello la señal CLK3 que será entrada en el componente CONFIGURACION_FSM para el proceso suma y resta de decenas.

4.5.2.3. Proceso suma y resta de decenas

```
process (RST,CLK_D,SRD)
begin
   if RST = '1' then
      DECENAS <= 0;
   elsif CLK_D'event and CLK_D = '1' then
      if SRD = "11"then
      DECENAS <=TO_INTEGER(unsigned(EDECENAS));
      elsif SRD = "10"then
      DECENAS <= DECENAS + 1;
         if RESET ='1' then
            DECENAS <= 0;
         end if;
      elsif SRD = "01" then
      DECENAS <= DECENAS - 1;
         if RESETD = '1' then
         DECENAS <= 2;
         elsif UNIDADES = 0 and RESETD = '0' then
         DECENAS <= 9;
         end if;
      end if;
   end if;
SDECENAS <= std_logic_vector(to_unsigned(DECENAS,4));
end process;
```

Ilustración 50. Proceso suma y resta de decenas

Al igual que el proceso de suma y resta de unidades, la lista de sensibilidad en este proceso estará compuesta por las señales RST, CLK_D, SRD, que ya se han definido cuando hemos hablado de todas las señales de la entidad.

Este proceso es muy similar al anterior, ya que cuando se reciba un flanco de reloj positivo, dependiendo del valor de SRD, se podrá copia el valor que recibimos como entrada, sumar o cuando se llegue al valor 2 en las decenas y 3 en las unidades se ponga a 0.

```
component PARTE_23 is
   port     (
            CLK : in std_logic;
            RST : in std_logic;
            AJUSTE : in std_logic;
            AJUSTE_S : in std_logic;
            AJUSTE_R : in std_logic;
            EDECENAS : in std_logic_vector (3 downto 0);
            EUNIDADES : in std_logic_vector (3 downto 0);
            SDECENAS : out std_logic_vector (3 downto 0);
            SUNIDADES : out std_logic_vector (3 downto 0)
            );
end component;
```

Ilustración 51. Componente PARTE_23

```
cambioHORAS : PARTE_23
   port map (CLK,RST,AJ,not KEY0,not KEY1,E5,E4,S5,S4);
```

Ilustración 52. Instanciación componente PARTE_23

El componente definido en la ilustración nº51 se utilizará durante el proyecto cuando se quiera cambiar de forma manual (con los botones) pudiendo ser este cambio de forma ascendente y descendente los valores hasta 23, normalmente se usará en las horas debido a que son las que llegan hasta dicho valor.

En cuanto a la instanciación que se ve en la ilustración nº52 se contemplan las siguientes asignaciones:

- CLK se asigna a CLK, ambos de tipo lógico y servirá para utilizar el reloj declarado como entrada en la entidad.
- RST se asigna a RST, ambas de tipo lógico y al igual que ocurre con CLK, utilizará el valor de RST declarado a la entrada.
- AJ se asigna a AJUSTE, ambas de tipo lógico. AJ es una entrada de la entidad que dará valor el valor correspondiente.
- Not KEY0 se asigna a AJUSTE_S, ambas de tipo lógico. La idea de complementar el valor de KEY0, es para que dé un valor lógico alto cuando de como salida un ‘1’, conectará este componente con la entrada de la entidad.
- Not KEY1 se asigna a AJUSTE_R, ambas de tipo lógico. La idea de complementar el valor de KEY1, al igual que el caso anterior, es para dar un valor lógico alto cuando de como salida un ‘1’, conectará este componente con la entrada de la entidad.
- E5 se asigna a EDECENAS, ambas son un vector de tipo binario que serán la entrada de las decenas del componente PARTE_23 y cuyo valor puede variar dependiendo del valor de, como se observa en la ilustración nº 53.
- E4 se asigna a EUNIDADES, ambas son un vector de tipo binario que serán la entrada de las unidades del componente PARTE_23, cuyo valor puede variar al igual que en caso anterior con la señal AJ.

- S5 se asigna a SDECENAS, ambas son un vector de tipo binario y serán la salida de la parte decenas del componente PARTE_23, como puede verse la ilustración nº53, puede volver a ser parte de la entrada de dicho componente.
- S4 se asigna a SUNIDADES, ambas son un vector de tipo binario y serán la salida de la parte unidades del componente PARTE_23, como puede verse la ilustración nº53, puede volver a ser parte de la entrada de dicho componente.

```
E4<= EDATO4 when AJ='0' else S4 when AJ='1' else EDATO4;
E5<= EDATO5 when AJ='0' else S5 when AJ='1' else EDATO5;
```

Ilustración 53. Valor de E4 y E5

4.5.3. PARTE_59: Descripción, declaración e instanciación

Esta entidad contiene los mismos elementos que la entidad PARTE_23, explicada en el punto anterior, por lo que en este punto no se volverán a explicar. La diferencia estará en los procesos debido a que el contaje de este componente estará configurado para que la suma se ponga a 0 una vez pasa el valor 59 y la resta una vez pase de 0 se ponga a 59.

Estas diferencias con la entidad anterior se podrán ver en los procesos que se muestran en la siguiente ilustración.

```
process (RST,CLK_U,SRU)
begin
if RST = '1' then
   UNIDADES <= 0;
elsif  CLK_U'event and CLK_U = '1' then
   if SRU = "11" then
   UNIDADES <=TO_INTEGER(unsigned(EUNIDADES));
   elsif SRU = "10" then
   UNIDADES <= UNIDADES + 1;
      if DECENAS = 9 and UNIDADES = 9 then
         UNIDADES <= 0;
         RESET <= '1';
      elsif UNIDADES = 9 then
         UNIDADES <= 0;
         RESET <= '0';
      end if;
   elsif SRU = "01"then
      if UNIDADES /= 0 then
      UNIDADES <= UNIDADES - 1;
         CLK3 <= '0';
         RESETD <= '0';
      elsif UNIDADES = 0 and DECENAS = 0 then
         UNIDADES <= 9;
         RESETD <= '1';
         CLK3 <= '1';
      elsif UNIDADES = 0 then
         UNIDADES <= 9;
         RESETD<='0';
         CLK3 <= '1';
      end if;
   else
   UNIDADES <=TO_INTEGER(unsigned(EUNIDADES));
   end if;
end if;
SUNIDADES <= std_logic_vector(to_unsigned(UNIDADES,4));
end process;


process (RST,CLK_D,SRD)
begin
   if RST = '1' then
      DECENAS <= 0;
   elsif CLK_D'event and CLK_D = '1' then
      if SRD = "11"then
      DECENAS <=TO_INTEGER(unsigned(EDECENAS));
      elsif SRD = "10"then
      DECENAS <= DECENAS + 1;
         if RESET ='1' then
            DECENAS <= 0;
         end if;
      elsif SRD = "01" then
      DECENAS <= DECENAS - 1;
         if RESETD = '1' then
         DECENAS <= 9;
         end if;
      end if;
   end if;
SDECENAS <= std_logic_vector(to_unsigned(DECENAS,4));
end process;
```

Ilustración 54. Proceso PARTE_59

```
component PARTE_59 is
   port      (
             CLK : in std_logic;
             RST : in std_logic;
             AJUSTE : in std_logic;
             AJUSTE_S : in std_logic;
             AJUSTE_R : in std_logic;
             EDECENAS : in std_logic_vector (3 downto 0);
             EUNIDADES : in std_logic_vector (3 downto 0);
             SDECENAS : out std_logic_vector (3 downto 0);
             SUNIDADES : out std_logic_vector (3 downto 0)
             );
end component;
```

Ilustración 55. Componente PARTE_59

```
cambioMINUTOS : PARTE_59
   port map (CLK,RST,AJ,not KEY3,not KEY4,E3,E2,S3,S2);
```

Ilustración 56. Instanciación componente PARTE_59

En la ilustración se observa que este componente contiene las mismas entradas y salidas que el componente PARTE_23. La instanciación también será muy parecida, en las asignaciones podemos destacar:

- Las asignaciones CLK, RST y AJ son las mismas que en el punto 4.7.1 debido a que tienen exactamente las mismas entradas.
- Not KEY3 asigna a AJUSTE_S, ambas de tipo lógico, la idea de complementar el valor de KEY3, es para que dé un valor lógico alto cuando de como salida un ‘1’, conectará este componente con la entrada de la entidad.
- Not KEY4 asigna a AJUSTE_R, ambas de tipo lógico, la idea de complementar el valor de KEY4, es para que dé un valor lógico alto cuando de como salida un ‘1’, conectará este componente con la entrada de la entidad.
- E3 asigna a EDECENAS, ambas son un vector de tipo binario que serán la entrada de las decenas del componente PARTE_59 y cuyo valor puede variar dependiendo del valor de E3, las condiciones que para definir la señal E3 se puede ver en la ilustración nº57.
- E2 asigna a EUNIDADES, ambas son un vector de tipo binario que serán la entrada de las unidades del componente PARTE_59, cuyo valor puede variar al igual que en caso anterior con la señal AJ, las condiciones que para definir la señal E2 se puede ver en la ilustración nº57.

- S3 asigna a SDECENAS, ambas son un vector de tipo binario y serán la salida de la parte decenas del componente PARTE_59, puede volver a ser parte de la entrada de dicho componente.
- S2 asigna a SUNIDADES, ambas son un vector de tipo binario y serán la salida de la parte unidades del componente PARTE_59, puede volver a ser parte de la entrada de dicho componente.

```
E2<= EDATO2 when AJ='0' else S2 when AJ='1' else EDATO2;
E3<= EDATO3 when AJ='0' else S3 when AJ='1' else EDATO3;
```

Ilustración 57. Valor de E2 y E3

4.6. RELOJ_PROCESO

La función principal de esta entidad será la de realizar un contaje como la de cualquier reloj. Dependiendo de la configuración de las entradas de nuestra entidad realizará el proceso o copiará los valores que recibe de entrada para mostrarlos en la salida.

```
entity RELOJ_PROCESO is
   port      (
             CLK : in std_logic;
             AJUSTE : in std_logic;
             DATOSG01IN : in std_logic_vector (3 downto 0);
             DATOSG10IN : in std_logic_vector (3 downto 0);
             DATOMIN01IN : in std_logic_vector (3 downto 0);
             DATOMIN10IN : in std_logic_vector (3 downto 0);
             DATOHR01IN : in std_logic_vector (3 downto 0);
             DATOHR10IN : in std_logic_vector (3 downto 0);
             DATOSG01OUT : out std_logic_vector (3 downto 0) :=(others=>'0');
             DATOSG10OUT : out std_logic_vector (3 downto 0) :=(others=>'0');
             DATOMIN01OUT : out std_logic_vector (3 downto 0) :=(others=>'0');
             DATOMIN10OUT : out std_logic_vector (3 downto 0) :=(others=>'0');
             DATOHR01OUT : out std_logic_vector (3 downto 0) :=(others=>'0');
             DATOHR10OUT : out std_logic_vector (3 downto 0) :=(others=>'0');
             CERO : in std_logic
             );
end RELOJ_PROCESO;

architecture RELOJ_PROCESO_arch of RELOJ_PROCESO is

   signal uu_s : integer range 0 to 9:=0;
   signal dd_s : integer range 0 to 9:=0;
   signal uu_m : integer range 0 to 9:=0;
   signal dd_m : integer range 0 to 9:=0;
   signal uu_h : integer range 0 to 9:=0;
   signal dd_h : integer range 0 to 9:=0;
   signal clk0 : std_logic;
   signal clk1 : std_logic;
   signal clk2 : std_logic;
   signal clk3 : std_logic;
   signal clk4 : std_logic;
   signal clk5 : std_logic;
```

Ilustración 58. Descripción entidad RELOJ_PROCESO

Para la realización de los procesos se han utilizado una serie de señales que a continuación explicaremos la función que tendrá cada una.

- uu_s, señal de tipo entero que representará las unidades de los segundos, su valor variará durante los procesos de la entidad.
- dd_s, señal de tipo entero que representará las decenas de los segundos, su valor variará durante los procesos de la entidad.
- uu_m, señal de tipo entero que representará las unidades de los minutos, su valor variará durante los procesos de la entidad.
- dd_m, señal de tipo entero que representará las decenas de los minutos, su valor variará durante los procesos de la entidad.
- uu_h, señal de tipo entero que representará las unidades de las horas, su valor variará durante los procesos de la entidad.
- dd_h, señal de tipo entero que representará las unidades de los segundos, su valor variará durante los procesos de la entidad.
- clk0, señal de tipo lógico recibida del divisor de frecuencia y necesaria para la realizar un proceso.
- clk1, señal de tipo lógico recibida del proceso donde varía el valor de las unidades de segundo y necesaria para la realizar un proceso.
- clk2, señal de tipo lógico recibida del proceso donde varía el valor de las decenas de segundo y necesaria para la realizar un proceso.
- clk3, señal de tipo lógico recibida del proceso donde varía el valor de las unidades de minuto y necesaria para la realizar un proceso.
- clk4, señal de tipo lógico recibida del proceso donde varía el valor de las decenas de minuto y necesaria para la realizar un proceso.
- clk5, señal de tipo lógico recibida del proceso donde varía el valor de las unidades de hora y necesaria para la realizar un proceso.

4.6.1. Proceso RELOJ_PROCESO

En la siguiente ilustración procederemos a describir más a fondo los distintos procesos que componen a la entidad que estamos tratando.

Esta entidad cuenta con un proceso por cada señal que represente un valor de nuestro reloj, todos los procesos son muy parecidos debido a solo se modificará el número final al que cada valor debe reestablecerse a 0.

La lista de sensibilidad de cada proceso será una señal del proceso anterior excepto en el primer caso que la señal será la señal que conecta la salida del divisor de frecuencia con el proceso de las unidades de segundo.

En el primer proceso, siempre que las entradas de la entidad cero y ajuste sean respectivamente ‘0’ y ‘1’ y la señal de reloj se ponga a ‘1’, entonces la señal uu_s copiará el valor que le llega a la entrada de la entidad y ésta dependiendo de si es igual a ‘9’ tomará como valor un ‘0’ y activará la señal clk1 para el siguiente proceso, en caso de ser distinto sumará uno. En caso de ser la señal cero un ‘1’ la señal uu_s tomará el valor 0, y en caso de ser el valor de ajuste un ‘0’ se copiará el valor de la entrada.

El resto de procesos será igual solo que el valor de dd_s será hasta 5, uu_m será hasta 9, dd_m será hasta 5. En el caso de las horas también varía un poco debido a que hay que condicionar la señal uu_h a si esta es igual a 9 o si uu_h igual a 2 y dd_h a 23, para las dd_h será igual que el primer proceso, pero hasta que llegue a 2, en este proceso no se activarán señales debido a que es el último dato.

Todo lo explicado anteriormente se ve claramente en el código, expuesto en la ilustración nº59.

```
process (clk0)
begin
   if cero ='0' then
      if ajuste ='1' then
         if clk0 ='1'then
            uu_s <=TO_INTEGER(unsigned(datoSg01IN));
            if uu_s = 9 then
               uu_s<=0;
               clk1 <='1';
            else
               uu_s <=uu_s +1;
               clk1<='0';
            end if;
         end if;
      else
      uu_s <=TO_INTEGER(unsigned(datoSg01IN));
      end if;
   datoSg01OUT <= std_logic_vector(to_unsigned(uu_s,4));
   else
   uu_s<=0;
   end if;
end process;

process (clk1)
begin
   if cero ='0' then
      if ajuste ='1' then
         if clk1 ='1' then
         dd_s <=TO_INTEGER(unsigned(datoSg10IN));
            if dd_s = 5 then
               dd_s<=0;
               clk2 <='1';
            else
               dd_s <=dd_s +1;
               clk2<='0';
            end if;
         end if;
      else
      dd_s <=TO_INTEGER(unsigned(datoSg10IN));
      end if;
   datoSg10OUT <= std_logic_vector(to_unsigned(dd_s,4));
   else
   dd_s<=0;
   end if;
end process;
```

```
process (clk2)
begin
   if cero ='0' then
      if ajuste ='1' then
         if (clk2'event and clk2 ='1') then
         uu_m <=TO_INTEGER(unsigned(datoMin01IN));
            if uu_m = 9 then
               uu_m<=0;
               clk3 <='1';
            else
               uu_m <=uu_m +1;
               clk3<='0';
            end if;
         end if;
      else
      uu_m <=TO_INTEGER(unsigned(datoMin01IN));
      end if;
   datoMin01OUT <= std_logic_vector(to_unsigned(uu_m,4));
   else
   uu_m<=0;
   end if;
end process;

process (clk3)
begin
   if cero ='0' then
      if ajuste ='1' then
         if (clk3'event and clk3 ='1') then
         dd_m <=TO_INTEGER(unsigned(datoMin10IN));
            if dd_m = 5 then
               dd_m<=0;
               clk4 <='1';
            else
               dd_m <=dd_m +1;
               clk4<='0';
            end if;
         end if;
      else
      dd_m <=TO_INTEGER(unsigned(datoMin10IN));
      end if;
   datoMin10OUT <= std_logic_vector(to_unsigned(dd_m,4));
   else
   dd_m<=0;
   end if;
end process;
```

```
process (clk4)
begin
   if cero ='0' then
    if ajuste ='1' then
         if (clk4'event and clk4 ='1') then
         uu_h <=TO_INTEGER(unsigned(datoHr01IN));
            if uu_h = 3 and dd_h=2 then
               uu_h <=0;
               clk5<='1';
            elsif uu_h = 9 then
               uu_h <=0;
               clk5<='1';
            else
               uu_h <= uu_h +1;
               clk5<='0';
            end if;
         end if;
      else
      uu_h <=TO_INTEGER(unsigned(datoHr01IN));
      end if;
   datoHr01OUT <= std_logic_vector(to_unsigned(uu_h,4));
   else
   uu_h<=0;
   end if;
end process;

process (clk5)
begin
   if cero ='0' then
      if ajuste ='1' then
         if (clk5'event and clk5 ='1') then
         dd_h <=TO_INTEGER(unsigned(datoHr10IN));
            if dd_h = 2 then
               dd_h <=0;
            else
               dd_h <= dd_h +1;
            end if;
         end if;
      else
      dd_h <=TO_INTEGER(unsigned(datoHr10IN));
      end if;
   datoHr10OUT <= std_logic_vector(to_unsigned(dd_h,4));
   else
   dd_h<=0;
   end if;
end process;
```

Ilustración 59. Proceso entidad RELOJ_PROCESO

4.6.2. DIVISOR: Descripción, declaración e instanciación

La función principal de esta entidad será la de conseguir una división de frecuencia para que nuestro circuito trabaje de la manera que deseamos.

```
entity DIVISOR is
   generic  (
            intC : integer
            );
   port  (
         MAX10_CLK1_50 : in STD_LOGIC;
         CLK : out STD_LOGIC
         );
end DIVISOR;

architecture DIVISOR_arch of DIVISOR is

   signal ckin : STD_LOGIC;
   signal ckout : STD_LOGIC :='0';
```

Ilustración 60. Entidad DIVISOR

Esta entidad necesitará del uso de dos señales,

- ckin, señal de tipo lógico cuyo valor en este proceso será igual que la señal de reloj de entrada.
- Ckout, señal de tipo lógico cuyo valor será el de salida del bloque.

Este proceso cuenta con el uso de contantes y variables,

- max_count, señal de tipo entero cuyo valor será igual a la mitad del generic que recibe la entidad.
- Count, señal de tipo entero cuyo rango va desde 0 al valor de max_count.

```
begin

ckin<=MAX10_CLK1_50;
CLK<=ckout;

process (ckin)
   constant max_count : integer := intC/2;
   variable count : integer range 0 to max_count :=0;
begin
   if rising_edge(ckin) then
      count:=count+1;
      if count=max_count then
         count:=0;
         ckout<=not ckout;
      end if;
   end if;
end process;

end DIVISOR_arch;
```

Ilustración 61. Proceso entidad DIVISOR

La lista de sensibilidad que tendrá este proceso está compuesta por la señal ckin, por lo que cada vez que esta cambie su valor nos introduciremos en el proceso.

En el proceso de la ilustración nº61 se puede comprobar que cada vez que hay un flanco positivo de la señal ckin, la variable count suma uno. Una vez sea igual la variable a la constante se inicializa la cuenta y da un flanco positivo ckout.

```
component DIVISOR is
   generic  (
            intC : integer
            );
   port     (
            MAX10_CLK1_50 : in STD_LOGIC;
            CLK : out STD_LOGIC
            );
end component;
```

Ilustración 62. Componente DIVISOR

```
div : DIVISOR
   generic map (50000000)
   port map (CLK,clk0);
```

Ilustración 63. Instanciación componente DIVISOR

En cuanto a la declaración del componente que estará en el interior de la entidad, podemos comprobar las entradas y salidas que se instanciarán con las señales.

En cuanto a la instanciación hay que indicar que en esta instanciación al haber un generic en la declaración habrá que usar la palabra “generic map” para indicar el valor que se desea, en este caso es 50000000. Este valor puede modificarse en cada instanciación debido a que son independientes.

En cuanto a las asignaciones con el port map podemos destacar:

- CLK se asigna a MAX_CLK1_50, ambas de tipo lógico, con esto se consigue que el reloj declarado en la entrada sea el que use la señal CLK.
- Clk0 se asigna a CLK, ambas de tipo lógico, con esta instanciación se consigue que la señal clk0 tenga el valor deseado a la salida del divisor de frecuencia.

4.7. FUNCIÓN CRONOMETRO_PROCESOS

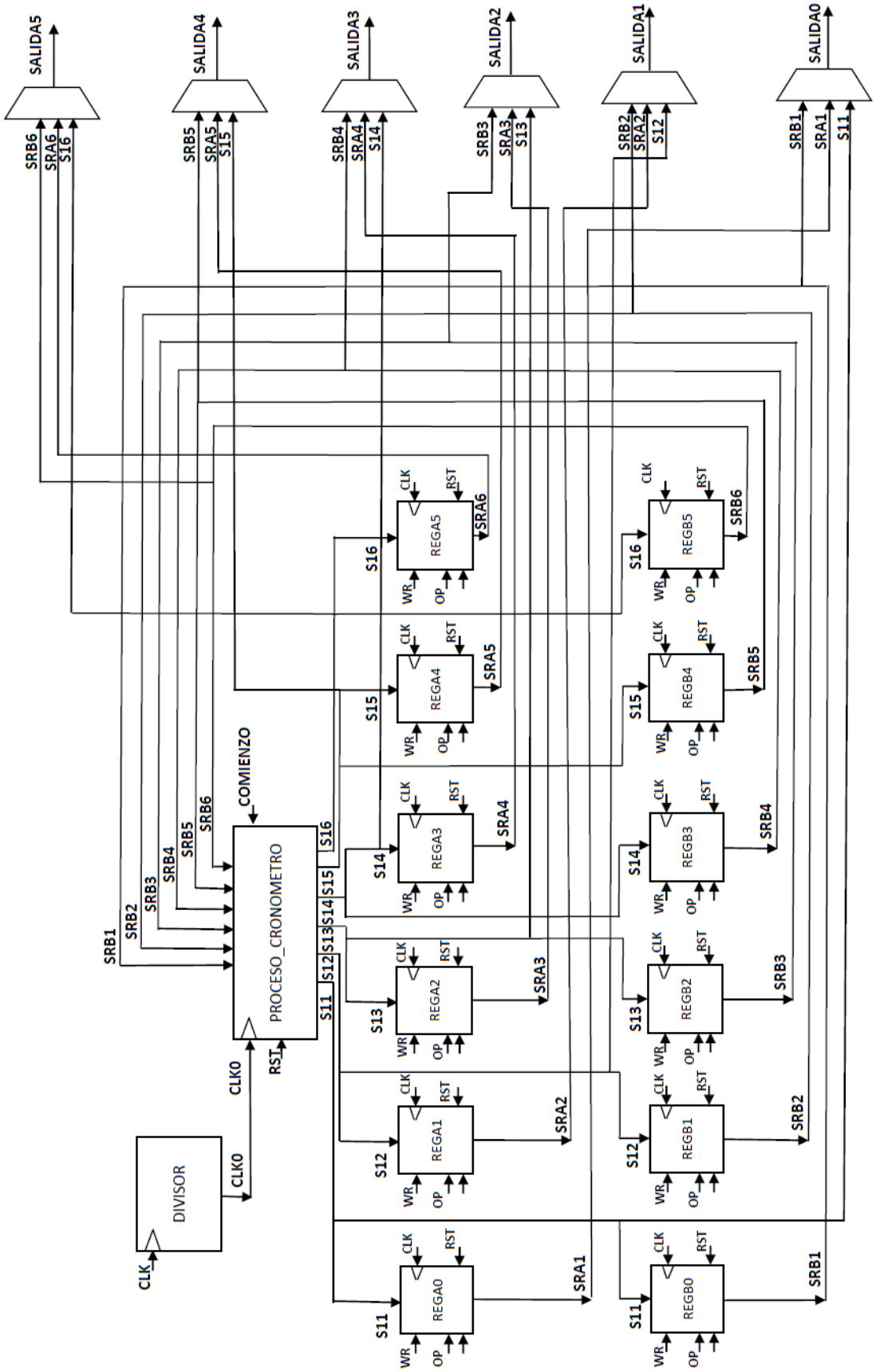


Ilustración 64. RTL CRONOMETRO_PROCESOS

Esta entidad cuyo RTL donde se puede ver las señales y los componentes internos de los que está compuesto la entidad se puede comprobar en la ilustración nº 64.

```
entity CRONOMETRO_PROCESOS is
   port      (
             RST : in STD_LOGIC;
             CLK : in std_logic;
             KEY0 : in std_logic;
             KEY1 : in std_logic;
             SALIDA0 : out std_logic_vector (3 downto 0);
             SALIDA1 : out std_logic_vector (3 downto 0);
             SALIDA2 : out std_logic_vector (3 downto 0);
             SALIDA3 : out std_logic_vector (3 downto 0);
             SALIDA4 : out std_logic_vector (3 downto 0);
             SALIDA5 : out std_logic_vector (3 downto 0)
             );
end CRONOMETRO_PROCESOS;

architecture CRONOMETRO_PROCESOS_arch of CRONOMETRO_PROCESOS is

   signal clk0 : std_logic;
   signal COMIENZO : std_logic;
   signal PAUSA : std_logic;
   signal DETENCION : std_logic;
   signal SRA1 : std_logic_vector (3 downto 0);
   signal SRA2 : std_logic_vector (3 downto 0);
   signal SRA3 : std_logic_vector (3 downto 0);
   signal SRA4 : std_logic_vector (3 downto 0);
   signal SRA5 : std_logic_vector (3 downto 0);
   signal SRA6 : std_logic_vector (3 downto 0);
   signal SRB1 : std_logic_vector (3 downto 0);
   signal SRB2 : std_logic_vector (3 downto 0);
   signal SRB3 : std_logic_vector (3 downto 0);
   signal SRB4 : std_logic_vector (3 downto 0);
   signal SRB5 : std_logic_vector (3 downto 0);
   signal SRB6 : std_logic_vector (3 downto 0);
   signal S11 : std_logic_vector (3 downto 0);
   signal S12 : std_logic_vector (3 downto 0);
   signal S13 : std_logic_vector (3 downto 0);
   signal S14 : std_logic_vector (3 downto 0);
   signal S15 : std_logic_vector (3 downto 0);
   signal S16 : std_logic_vector (3 downto 0);
   signal WA : std_logic;
   signal WB : std_logic;
```

Ilustración 65. Entidad CRONOMETRO_PROCESOS

En la ilustración anterior cuando se declara la entidad se puede ver las entradas y salidas que tendrá y que más tarde veremos cómo se instancian con las señales.

Las señales descritas son,

- Clk0, señal de tipo lógico que enlaza la salida del divisor de frecuencia con el componente que realiza el proceso.
- COMIENZO, señal de tipo lógico que será de entrada a componente donde se incluye el proceso.
- PAUSA, señal de tipo lógico que condicionará las salidas como se muestra en la ilustración nº 66.
- DETENCION, señal de tipo lógico que condicionará las salidas como se muestra en la ilustración nº 66.
- WA, señal de tipo lógico que enlazará una salida de la máquina de estados con una entrada en los registros A.
- WB, señal de tipo lógico que enlazará una salida de la máquina de estados con una entrada en los registros B.
- SRA1, SRA2, SRA3, SRA4, SRA5, SRA6: Estas señales son de tipo std_logic_vector de 4 bits. Estos valores conectarán la salida de los registros A con una de las entradas de los multiplexores que nos proporcionarán la salida de cada dato de nuestro cronómetro.
- SRB1, SRB2, SRB3, SRB4, SRB5, SRB6: Estas señales son de tipo std_logic_vector de 4 bits. La señal conectará la salida de los registros B con otra de las entradas del multiplexor y será la entrada del componente principal CRONOMETRO_PROCESO.
- S11, S12, S13, S14, S15, S16: Estas señales son de tipo std_logic_vector de 4 bits, servirán de enlace entre la salida del componente CRONOMETRO_PROCESO y las entradas de todos los registros A y B.

```
SALIDA0 <= SRB1 when PAUSA = '1' else SRA1 when DETENCION = '1' else S11;
SALIDA1 <= SRB2 when PAUSA = '1' else SRA2 when DETENCION = '1' else S12;
SALIDA2 <= SRB3 when PAUSA = '1' else SRA3 when DETENCION = '1' else S13;
SALIDA3 <= SRB4 when PAUSA = '1' else SRA4 when DETENCION = '1' else S14;
SALIDA4 <= SRB5 when PAUSA = '1' else SRA5 when DETENCION = '1' else S15;
SALIDA5 <= SRB6 when PAUSA = '1' else SRA6 when DETENCION = '1' else S16;
```

Ilustración 66. Señal SALIDA0, SALIDA1, SALIDA2, SALIDA3, SALIDA4, SALIDA5

4.7.1. CRONOMETRO_PROCESO: Descripción, componente, instanciación

La función principal que realizará esta entidad será exactamente igual que la realizada por el RELOJ_PROCESO, pero con la salvedad de que los flancos de reloj son más continuos y se reciben directamente como entrada de la entidad.

```
entity CRONOMETRO_PROCESO is
   port     (
            clk : in std_logic;
            RST : in std_logic;
            comienzo : in std_logic;
            DATOSG01IN : in std_logic_vector (3 downto 0);
            DATOSG10IN : in std_logic_vector (3 downto 0);
            DATOMIN01IN : in std_logic_vector (3 downto 0);
            DATOMIN10IN : in std_logic_vector (3 downto 0);
            DATOHR01IN : in std_logic_vector (3 downto 0);
            DATOHR10IN : in std_logic_vector (3 downto 0);
            DATOSG01OUT : out std_logic_vector (3 downto 0);
            DATOSG10OUT : out std_logic_vector (3 downto 0);
            DATOMIN01OUT : out std_logic_vector (3 downto 0);
            DATOMIN10OUT : out std_logic_vector (3 downto 0);
            DATOHR01OUT : out std_logic_vector (3 downto 0);
            DATOHR10OUT : out std_logic_vector (3 downto 0)
            );
end CRONOMETRO_PROCESO;
architecture CRONOMETRO_PROCESO_arch of CRONOMETRO_PROCESO is

   signal centesimas_s : integer range 0 to 9;
   signal decimas_s : integer range 0 to 9;
   signal unidades_s : integer range 0 to 9;
   signal decenas_s : integer range 0 to 9;
   signal unidades_m : integer range 0 to 9;
   signal decenas_m : integer range 0 to 9;
   signal clk1 : std_logic := '0';
   signal clk2 : std_logic := '0';
   signal clk3 : std_logic := '0';
   signal clk4 : std_logic := '0';
   signal clk5 : std_logic := '0';
```

Ilustración 67. Entidad CRONOMETRO_PROCESO

Para la realización de los procesos se han utilizado una serie de señales que a continuación explicaremos la función que tendrá cada una.

- centesimas_s, señal de tipo entero que representará las centésimas de segundo, su valor variará durante los procesos de la entidad.
- decimas_s, señal de tipo entero que representará las décimas de segundo, su valor variará durante los procesos de la entidad.
- unidades_s, señal de tipo entero que representará las unidades de segundos, su valor variará durante los procesos de la entidad.
- decenas_s, señal de tipo entero que representará las decenas de segundos, su valor variará durante los procesos de la entidad.
- unidades_m, señal de tipo entero que representará las unidades de minutos, su valor variará durante los procesos de la entidad.

- decenas_m, señal de tipo entero que representará las unidades de minutos, su valor variará durante los procesos de la entidad.
- clk1, señal de tipo lógico recibida del proceso donde varía el valor de las unidades de segundo y necesaria para la realizar un proceso.
- clk2, señal de tipo lógico recibida del proceso donde varía el valor de las decenas de segundo y necesaria para la realizar un proceso.
- clk3, señal de tipo lógico recibida del proceso donde varía el valor de las unidades de minuto y necesaria para la realizar un proceso.
- clk4, señal de tipo lógico recibida del proceso donde varía el valor de las decenas de minuto y necesaria para la realizar un proceso.
- clk5, señal de tipo lógico recibida del proceso donde varía el valor de las unidades de hora y necesaria para la realizar un proceso.

En la siguiente ilustración procederemos a describir más a fondo los distintos procesos que componen a la entidad que estamos tratando.

Esta entidad cuenta con un proceso por cada señal que represente un valor de nuestro reloj, todos los procesos son muy parecidos debido a solo se modificará el número final al que cada valor debe reestablecerse a 0.

La lista de sensibilidad de cada proceso será una señal del proceso anterior excepto en el primer caso que la señal será la señal que conecta la salida del divisor de frecuencia con el proceso de las unidades de segundo.

En el primer proceso, siempre que las entradas de la entidad RST y comienzo sean respectivamente ‘0’ y ‘1’ y la señal de reloj se ponga a ‘1’, entonces la señal centesimas_s copiará el valor que le llega a la entrada de la entidad y ésta, dependiendo de si es igual a ‘9’ tomará como valor un ‘0’ y activará la señal clk1 para el siguiente proceso, en caso de ser distinto sumará uno. En caso de ser la señal cero un ‘1’ la señal centesimas _s tomará el valor 0, y en caso de ser el valor de ajuste un ‘0’ se copiará el valor de la entrada.

El resto de procesos será igual solo que el valor de decimas_s será hasta 9, unidades_s será hasta 9, decenas_s será hasta 5. En el caso de las horas también varía un poco debido a que hay que condicionar la señal unidades_m a si esta es igual

a 9 o si unidades_m igual a 9 y decenas_m a 5, para las decenas_m será igual que el primer proceso, pero hasta que llegue a 5, en este proceso no se activarán señales debido a que es el último dato.

Todo lo explicado anteriormente se ve claramente en el código, expuesto en la ilustración nº68.

```
process (CLK)
begin
   if RST ='0' then
      if comienzo = '1' then
         if CLK ='1' then
            centesimas_s <=TO_INTEGER(unsigned(DATOSG01IN));
            if centesimas_s=9 then
               centesimas_s<=0;
               clk1<='1';
            else
               centesimas_s <= centesimas_s + 1;
               clk1<='0';
            end if;
         end if;
      else
      centesimas_s <=TO_INTEGER(unsigned(DATOSG01IN));
      end if;
   else
   centesimas_s<=0;
   end if;
DATOSG01OUT <= std_logic_vector(to_unsigned(centesimas_s,4));
end process;

process (CLK1)
begin
   if RST ='0' then
      if comienzo = '1' then
         if CLK1 ='1' then
         decimas_s <=TO_INTEGER(unsigned(DATOSG10IN));
            if decimas_s=9 then
               decimas_s<=0;
               clk2<='1';
            else
               decimas_s <= decimas_s + 1;
               clk2<='0';
            end if;
         end if;
      else
      decimas_s <=TO_INTEGER(unsigned(DATOSG10IN));
      end if;
   else
   decimas_s<=0;
   end if;
DATOSG10OUT <= std_logic_vector(to_unsigned(decimas_s,4));
end process;
```

```
process (CLK2)
begin
   if RST ='0' then
      if comienzo = '1' then
         if CLK2 ='1' then
         unidades_s <=TO_INTEGER(unsigned(DATOMIN01IN));
            if unidades_s=9 then
               unidades_s<=0;
               clk3<='1';
            else
               unidades_s <= unidades_s + 1;
               clk3<='0';
            end if;
         end if;
      else
      unidades_s <=TO_INTEGER(unsigned(DATOMIN01IN));
      end if;
   else
   unidades_s<=0;
   end if;
DATOMIN01OUT <= std_logic_vector(to_unsigned(unidades_s,4));
end process;

process (CLK3)
begin
   if RST ='0' then
      if comienzo = '1' then
         if CLK3 ='1' then
         decenas_s <=TO_INTEGER(unsigned(DATOMIN10IN));
            if decenas_s=5 then
               decenas_s<=0;
               clk4<='1';
            else
               decenas_s <= decenas_s + 1;
               clk4<='0';
            end if;
         end if;
      else
      decenas_s <=TO_INTEGER(unsigned(DATOMIN10IN));
      end if;
   else
   decenas_s<=0;
   end if;
DATOMIN10OUT <= std_logic_vector(to_unsigned(decenas_s,4));
end process;
```

```
process (CLK4)
begin
   if RST ='0' then
      if comienzo = '1' then
         if CLK4 ='1' then
         unidades_m <=TO_INTEGER(unsigned(DATOHR01IN));
            if unidades_m=9 then
               unidades_m<=0;
               clk5<='1';
            else
               unidades_m <= unidades_m + 1;
               clk5<='0';
            end if;
         end if;
      else
      unidades_m <=TO_INTEGER(unsigned(DATOHR01IN));
      end if;
   else
   unidades_m<=0;
   end if;
DATOHR01OUT <= std_logic_vector(to_unsigned(unidades_m,4));
end process;

process (CLK5)
begin
   if RST ='0' then
      if comienzo = '1' then
         if CLK5 ='1' then
         decenas_m <=TO_INTEGER(unsigned(DATOHR10IN));
            if decenas_m=5 then
               decenas_m<=0;
            else
               decenas_m <= decenas_m + 1;
            end if;
         end if;
      else
      decenas_m <=TO_INTEGER(unsigned(DATOHR10IN));
      end if;
   else
   decenas_m<=0;
   end if;
DATOHR10OUT <= std_logic_vector(to_unsigned(decenas_m,4));
end process;
```

Ilustración 68. Proceso entidad CRONOMETRO_PROCESO

4.7.2. CRONOMETRO_FSM: Descripción, componente, instanciación

Para el control de las entradas del bloque CRONOMETRO_PROCESO con dos botones tendremos que realizar una máquina de estados en el que sus entradas y salidas sean como se muestran en la siguiente ilustración.

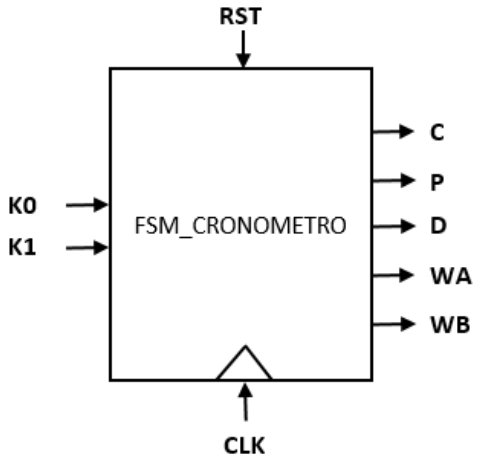


Ilustración 69. Entradas y salidas CONFIGURACION_FSM

```
entity CRONOMETRO_FSM is
   port     (
            CLK : in STD_LOGIC;
            RST : in STD_LOGIC;
            K0 : in STD_LOGIC;
            K1 : in STD_LOGIC;
            C : out STD_LOGIC;
            P : out STD_LOGIC;
            D : out STD_LOGIC;
            WA : out STD_LOGIC;
            WB : out STD_LOGIC
            );
end entity;

architecture CRONOMETRO_FSM_arch of CRONOMETRO_FSM is

   TYPE ESTADOS is (Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8);
   signal ESTADO : ESTADOS := Q0;
   signal funcion : std_logic_vector (4 downto 0);
```

Ilustración 70. Entidad CRONOMETRO_FSM

En las señales que se observan podemos destacar:

- ESTADO, la función de esta señal será la de asignar los valores que se han definido en la línea “TYPE”.
- funcion, se trata de vector binario de 4 valores cuyo valor irá asignado a los diferentes estados definidos en “TYPE”.

Antes de explicar que realizará la máquina de estados podemos observar en la ilustración nº70 las entradas y salidas de un estado.

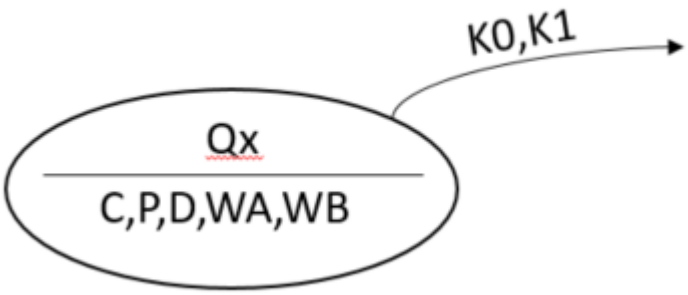


Ilustración 71. Estado CONFIGURACION_FSM

En la máquina de estados podemos distinguir un total de 9 estados, explicaremos las salidas de cada uno en función de las entradas que recibe.

- Q0, es el estado inicial, en este estado todas las salidas se encuentran a '0'.
- Q1, en este estado se pondrá a '1' la salida D, debido a que se ha recibido un '1' de la entrada K0
- Q2, sigue igual que el estado anterior cuando se ha recibido un '0' en K0.
- Q3, siempre se llega a este estado debido a que no hay condición con las entradas que lo impida y su salida son todas con valor '0' excepto WA que se pone a '1'.
- Q4, en este estado se activa la salida WA y se ponen a '0' el resto.
- Q5, llegamos a este estado si está K0 a '0' y K1 a '1', se pone a '1' la salida C y WB a la vez que se mantiene la D como en el estado anterior.
- Q6, siempre se llega a este estado sin ningún tipo de condición y la salida es igual que en el estado anterior únicamente con C tomando el valor '0'
- Q7, se llega a este estado cuando ambas entradas son '0' y las salidas que resultan de este estado son todas '0' excepto la D cuyo valor sigue valiendo '1'
- Q8, llegamos a este estado si está K0 a '1' y K1 a '0', se pone a '1' la salida P y se mantiene la D como en el estado anterior.

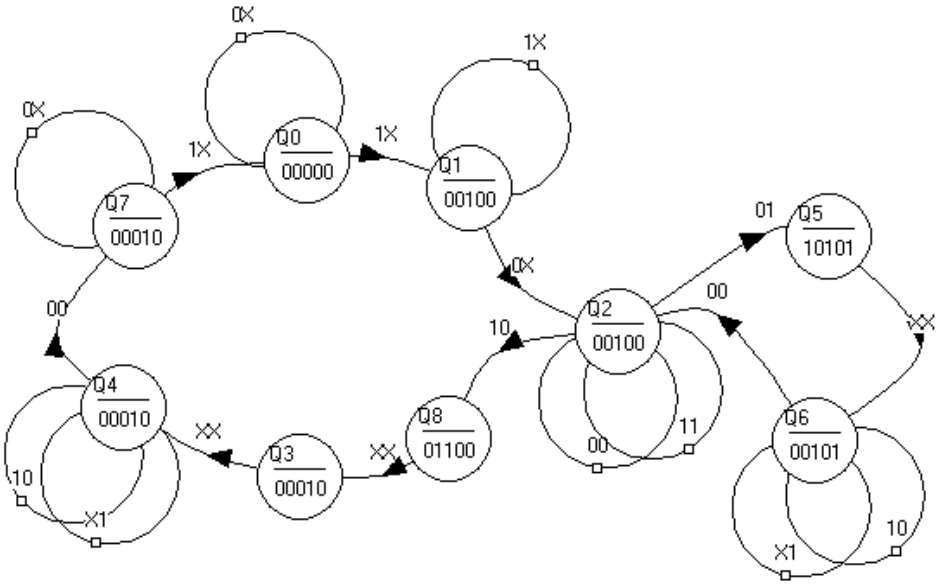


Ilustración 72. Máquina de estados CRONOMETRO_FSM

En la ilustración nº73 podemos ver implementada la máquina de Moore de nuestro controlador en VHDL.

```
process (CLK,RST)
begin
   if RST = '1' then
      ESTADO<=Q0;
   elsif rising_edge (CLK) then
      case ESTADO is
         when Q0 =>
             if K0='1'then
               ESTADO<=Q1;
            else
               ESTADO<=Q0;
            end if;
         when Q1 =>
            if K0='0' then
               ESTADO<=Q2;
            else
               ESTADO<=Q1;
            end if;
         when Q2 =>
            if  K0='1' and k1='0' then
               ESTADO<=Q8;
            elsif K0='0' and k1='1' then
               ESTADO<=Q5;
            else
               ESTADO<=Q2;
            end if;
```

```
            when Q8 =>
                ESTADO<=Q3;
            when Q3 =>
                ESTADO<=Q4;
            when Q4 =>
               if k0='0' and k1='0' then
                  ESTADO<=Q7;
               else
                  ESTADO<=Q4;
               end if;
            when Q5 =>
               ESTADO<=Q6;
            when Q6 =>
               if k0='0' and k1='0' then
                  ESTADO<=Q2;
               else
                  ESTADO<=Q6;
               end if;
            when Q7 =>
               if k0 ='1' then
                  ESTADO<=Q0;
               else
                  ESTADO<=Q7;
               end if;
         end case;
      end if;
   end process;
   WA <= funcion (4);
   WB <= funcion (3);
   C <= funcion (2);
   P <= funcion (1);
   D <= funcion (0);

   with ESTADO select funcion <=

      "00000" when Q0,
      "00100" when Q1,
      "00100" when Q2,
      "00010" when Q3,
      "00010" when Q4,
      "10101" when Q5,
      "00101" when Q6,
      "00010" when Q7,
      "01100" when Q8;

   end CRONOMETRO_FSM_arch;
```

Ilustración 73. Proceso CRONOMETRO_FSM

La lista de sensibilidad y la estructura del proceso es exactamente igual que la máquina de Moore explicada en el punto 4.5.2.1

```
component CRONOMETRO_FSM is
   port     (
            CLK : in STD_LOGIC;
            RST : in STD_LOGIC;
            K0 : in STD_LOGIC;
            K1 : in STD_LOGIC;
            C : out STD_LOGIC;
            P : out STD_LOGIC;
            D : out STD_LOGIC;
            WA : out STD_LOGIC;
            WB : out STD_LOGIC
            );
end component;
```

Ilustración 74. Componente CRONOMETRO_FSM

```
FSM_cuenta : CRONOMETRO_FSM
   port map(clk,rst,not KEY0,not KEY1,COMIENZO,PAUSA,DETENCION,WA,WB);
```

Ilustración 75. Instanciación componente CRONOMETRO_FSM

Tanto en la ilustración nº74 como en la nº71 se pueden observar cuáles serán las entradas y salidas del componente CRONOMETRO_FSM, con este componente se pretenderá controlar las señales que enlazarán con los procesos que están incluidos en esta entidad.

En la ilustración nº75 se observamos las asignaciones realizadas:

- CLK se asigna a CLK, ambas de tipo lógico, con esto se consigue que el reloj con el que ejecutará la máquina será el de la entidad.
- RST se asigna a RST, ambas de tipo lógico, con esta instanciación se consigue que está entrada del componente tenga el mismo valor que la entrada RST de la entidad.
- Not KEY0 se asigna a K0, ambas de tipo lógico, será una de las dos entradas de nuestra máquina de estados.
- Not KEY1 se asigna a K1, ambas de tipo lógico, es la entrada restante fundamental para el cambio de estado.
- COMIENZO asigna a C, ambas de tipo lógico, será una de las salidas mostradas a la entidad donde se instancie.
- PAUSA asigna a P, ambas de tipo lógico, será una de las salidas mostradas a la entidad donde se instancie.

- DETENCIÓN asigna a D, ambas de tipo lógico, será una de las salidas mostradas a la entidad donde se instancie.
- WA asigna a WA ambas de tipo lógico, será una de las salidas mostradas a la entidad donde se instancie.
- WB asigna a WA, ambas de tipo lógico, será una de las salidas mostradas a la entidad donde se instancie.

4.7.3. REG_N: Descripción, componente, instanciación

La descripción de la entidad y la declaración del componente es exactamente igual que la realizada en el punto 4.4.3.

```
RegA0 : REG_N
   generic map (4)
   port map (clk,rst,WA,"00",S11,SRA1);
RegA1 : REG_N
   generic map (4)
   port map (clk,rst,WA,"00",S12,SRA2);
RegA2 : REG_N
   generic map (4)
   port map (clk,rst,WA,"00",S13,SRA3);
RegA3 : REG_N
   generic map (4)
   port map (clk,rst,WA,"00",S14,SRA4);
RegA4 : REG_N
   generic map (4)
   port map (clk,rst,WA,"00",S15,SRA5);
RegA5 : REG_N
   generic map (4)
   port map (clk,rst,WA,"00",S16,SRA6);

RegB0 : REG_N
   generic map (4)
   port map (clk,rst,WB,"00",S11,SRB1);
RegB1 : REG_N
   generic map (4)
   port map (clk,rst,WB,"00",S12,SRB2);
RegB2 : REG_N
   generic map (4)
   port map (clk,rst,WB,"00",S13,SRB3);
RegB3 : REG_N
   generic map (4)
   port map (clk,rst,WB,"00",S14,SRB4);
RegB4 : REG_N
   generic map (4)
   port map (clk,rst,WB,"00",S15,SRB5);
RegB5 : REG_N
   generic map (4)
   port map (clk,rst,WB,"00",S16,SRB6);
```

Ilustración 76. Instanciación componente REG_N

En cuanto a las asignaciones sí que hay bastantes variaciones, las asignaciones serán las siguientes.

- CLK se asigna a CLK, ambas son de tipo lógico. Esta asignación servirá para que se use el reloj de entrada de la entidad en los componentes REG_N.
- RST se asigna a RST, ambas de tipo lógico. Con esta asignación conseguiremos que la señal RST se active cuando lo haga el RST de entrada de la entidad.
- "00" se asigna a OP, ambas son vectores binarios y servirá para indicar un número específico binario en la entrada OP del registro en cuestión.

En cambio, cada una tiene una entrada y salida específica que será el número que desea ingresar en el registro.

- WA se asigna a WR, ambas de tipo lógico. Sirve para enlazar la salida de la máquina de estados con una entrada de los registros A aportándole el valor que la señal tiene.
- WB se asigna a WR, ambas de tipo lógico. Sirve para enlazar la salida de la máquina de estados con una entrada de los registros B aportándole el valor que la señal tiene.
- S11 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente CRONOMETRO_PROCESO en el RegA0.
- S12 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente CRONOMETRO_PROCESO en el RegA1.
- S13 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente CRONOMETRO_PROCESO en el RegA2.
- S14 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente CRONOMETRO_PROCESO en el RegA3.
- S15 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente CRONOMETRO_PROCESO en el RegA4.
- S16 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente CRONOMETRO_PROCESO en el RegA5.
- SRA1 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará el RegA0 al multiplexor de una de las salidas.
- SRA2 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará el RegA1 al multiplexor de una de las salidas.

- SRA3 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará RegA2 al multiplexor de una de las salidas.
- SRA4 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará RegA3 al multiplexor de una de las salidas.
- SRA5 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará el RegA4 al multiplexor de una de las salidas.
- SRA6 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará el registro al multiplexor de una de las salidas.
- S11 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente CRONOMETRO_PROCESO en el RegB0.
- S12 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente CRONOMETRO_PROCESO en el RegB1.
- S13 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente CRONOMETRO_PROCESO en el RegB2.
- S14 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente CRONOMETRO_PROCESO en el RegB3.
- S15 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente CRONOMETRO_PROCESO en el RegB4.
- S16 se asigna a DIN, ambas son vectores binarios y es la señal que se recibe del componente CRONOMETRO_PROCESO en el RegB5.
- SRB1 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará el RegB0 a la entrada del componente CRONOMETRO_PROCESO.
- SRB2 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará el RegB1 a la entrada del componente CRONOMETRO_PROCESO.
- SRB3 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará RegB2 y a la entrada del componente CRONOMETRO_PROCESO.
- SRB4 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará RegB3 a la entrada del componente CRONOMETRO_PROCESO.

- SRB5 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará el RegB4 a la entrada del componente CRONOMETRO_PROCESO.
- SRB6 se asigna a DOUT, ambas son vectores binarios y es la señal que proporcionará el RegB5 a la entrada del componente CRONOMETRO_PROCESO.

4.7.4. DIVISOR: Descripción, componente, instanciación

En nuestra entidad volveremos a necesitar la instanciación de la entidad divisor la cual se ha descrito previamente en el punto 4.6.2.

```
div : divisor
  generic map (500000)
  port map (clk,clk0);
```

Ilustración 77. Instanciación componente DIVISOR

Como se observa en la ilustración, la instanciación también es la misma que la comentada al comienzo de esta sección con la salvedad de que el valor del generic map es menos, esto tendrá una repercusión en la frecuencia de la señal de reloj generada por este divisor de frecuencia.

4.8. FUNCIÓN CUENTAATRAS_PROCESOS

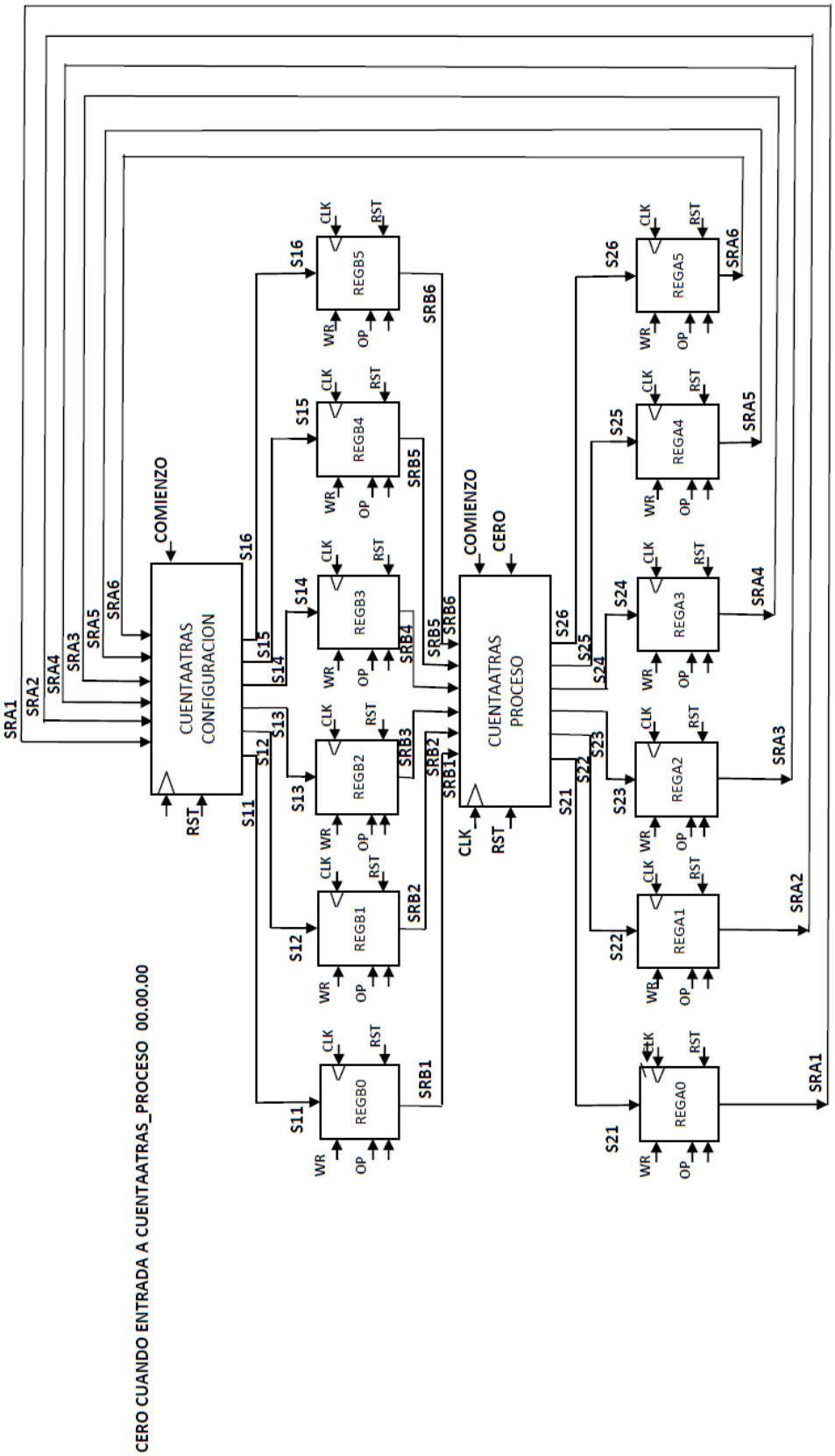


Ilustración 78. RTL CUENTAATRAS_PROCESO

Otro de los grandes bloques que se ha instanciado en la entidad principal es la CUENTAATRAS_PROCESOS, el RTL de esta entidad se puede observar en la ilustración nº78. Las entradas y salidas de la entidad se pueden ver en la siguiente ilustración.

```
entity CUENTAATRAS_PROCESOS is
   port     (
            RST : in std_logic;
            CLK : in std_logic;
            AJUSTE : in std_logic;
            KEY0 : in std_logic;
            KEY1 : in std_logic;
            KEY3 : in std_logic;
            KEY4 : in std_logic;
            KEY2 : in std_logic;
            salida0 : out std_logic_vector (3 downto 0);
            salida1 : out std_logic_vector (3 downto 0);
            salida2 : out std_logic_vector (3 downto 0);
            salida3 : out std_logic_vector (3 downto 0);
            salida4 : out std_logic_vector (3 downto 0);
            salida5 : out std_logic_vector (3 downto 0)
            );
end CUENTAATRAS_PROCESOS;

architecture CUENTAATRAS_PROCESOS_arch of CUENTAATRAS_PROCESOS is

   signal CER : std_logic;
   signal COMIENZO : std_logic;
   signal CERO : std_logic;
   signal STOP : std_logic;
   signal AJ : std_logic;
   signal S11: std_logic_vector (3 downto 0);
   signal S12: std_logic_vector (3 downto 0);
   signal S13: std_logic_vector (3 downto 0);
   signal S14: std_logic_vector (3 downto 0);
   signal S15: std_logic_vector (3 downto 0);
   signal S16: std_logic_vector (3 downto 0);
   signal S21: std_logic_vector (3 downto 0);
   signal S22: std_logic_vector (3 downto 0);
   signal S23: std_logic_vector (3 downto 0);
   signal S24: std_logic_vector (3 downto 0);
   signal S25: std_logic_vector (3 downto 0);
   signal S26 : std_logic_vector (3 downto 0);
   signal SRA1 : std_logic_vector (3 downto 0);
   signal SRA2 : std_logic_vector (3 downto 0);
   signal SRA3 : std_logic_vector (3 downto 0);
   signal SRA4 : std_logic_vector (3 downto 0);
   signal SRA5 : std_logic_vector (3 downto 0);
   signal SRA6 : std_logic_vector (3 downto 0);
   signal SRB1 : std_logic_vector (3 downto 0);
   signal SRB2 : std_logic_vector (3 downto 0);
   signal SRB3 : std_logic_vector (3 downto 0);
   signal SRB4 : std_logic_vector (3 downto 0);
   signal SRB5 : std_logic_vector (3 downto 0);
   signal SRB6 : std_logic_vector (3 downto 0);
```

Ilustración 79. Entidad CUENTAATRAS_PROCESO

Las señales que conectarán los distintos componentes instanciados en la entidad CUENTAATRAS_PROCESOS serán,

- CER, señal de tipo lógico que será entrada de la máquina de estados y cuyo valor será el mostrado por la ilustración nº80.
- COMIENZO, señal de tipo lógico que enlaza una de las salidas de la máquina de estados con una entrada del proceso de la cuenta atrás.
- CERO, señal de tipo lógico que enlaza una de las salidas de la máquina de estados con una entrada del proceso de la cuenta atrás.
- STOP, señal de tipo lógico que enlaza una de las salidas de la máquina de estados con una entrada del proceso de la cuenta atrás.
- AJ, señal de tipo lógico que será una de las entradas de la CUENTAATRAS_CONFIGURACION y CUENTAATRAS_PROCESO.
- S11, S12, S13, S14, S15, S16: Estas señales son de tipo std_logic_vector de 4 bits, servirán de enlace entre la salida del componente CUENTAATRAS_CONFIGURACION con los registros B.
- SRB1, SRB2, SRB3, SRB4, SRB5, SRB6: Estas señales son de tipo std_logic_vector de 4 bits. La señal conectará las salidas de los registros B con CUENTAATRAS_PROCESO.
- S21, S22, S23, S24, S25, S26: Estas señales son de tipo std_logic_vector de 4 bits, servirán de enlace entre la salida del componente CUENTAATRAS_PROCESO y las entradas de todos los registros A.
- SRA1, SRA2, SRA3, SRA4, SRA5, SRA6: Estas señales son de tipo std_logic_vector de 4 bits. Estas señales serán las salidas de los registros A conectarán con el componente CUENTAATRAS_CONFIGURACION.

```
CER<='1' when (S21="0000" and S22="0000" and S23="0000" and S24="0000" and S25="0000" and S26="0000") else '0';
```

Ilustración 80. Señal CER

4.8.1. CUENTAATRAS_FSM: Descripción, componente, instanciación

Para el control de las entradas del bloque CUENTAATRAS_PROCESO y CUENTAATRAS_CONFIGURACION a partir de tres entradas, tendremos que realizar una máquina de estados en el que sus entradas y salidas sean como se muestran en la siguiente ilustración.

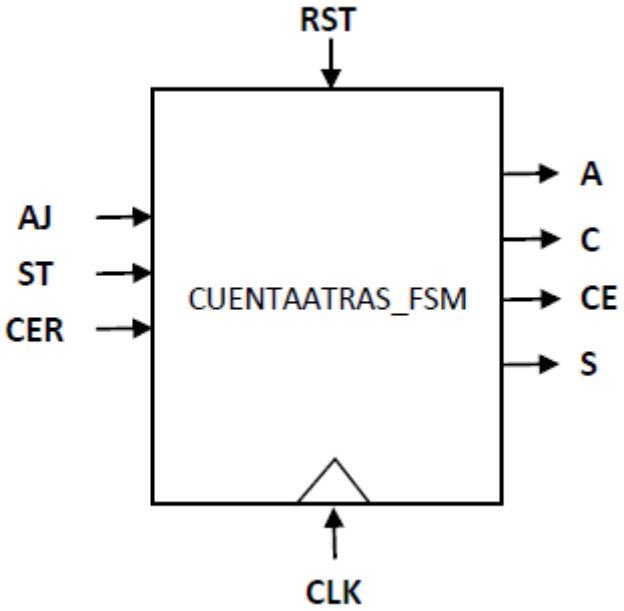


Ilustración 81. Entradas y salidas CUENTAATRAS_FSM

```
entity CUENTAATRAS_FSM is
   port     (
            CLK : in STD_LOGIC;
            RST : in STD_LOGIC;
            AJ : in STD_LOGIC;
            ST : in STD_LOGIC;
            CER : in STD_LOGIC;
            A : out STD_LOGIC;
            C : out STD_LOGIC;
            CE : out STD_LOGIC;
            S : out STD_LOGIC
            );
end CUENTAATRAS_FSM;

architecture CUENTAATRAS_FSM_arch of CUENTAATRAS_FSM is

   TYPE ESTADOS is (Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8);
   signal ESTADO : ESTADOS := Q0;
   signal funcion : std_logic_vector (3 downto 0);
```

Ilustración 82. Entidad CUENTAATRAS_FSM

En las señales que se observan podemos destacar:

- ESTADO, la función de esta señal será la de asignar los valores que se han definido en la línea “TYPE”.
- funcion, se trata de vector binario de 4 valores cuyo valor irá asignado a los diferentes estados definidos en “TYPE”.

Antes de explicar que realizará la máquina de estados podemos observar en la ilustración nº81 las entradas y salidas de un estado.

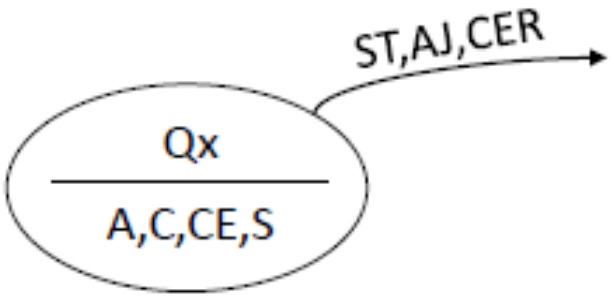


Ilustración 83. Estado CUENTAATRAS_FSM

En la máquina de estados podemos distinguir un total de 9 estados, explicaremos las salidas de cada uno en función de las entradas que recibe.

- Q0, es el estado inicial, en este estado todas las salidas se encuentran a ‘0’.
- Q1, en este estado se pondrá a ‘1’ la salida A, debido a que se ha recibido un ‘1’ de la entrada AJ
- Q2, se vuelve a poner todas las salidas a ‘0’ cuando se ha recibido un ‘0’ en AJ, desde este estado se puede ir a Q3 si ST y AJ es ‘1’ y ‘0’ respectivamente o Q1 si esas entradas tienen el valor opuesto.
- Q3, en este estado todas las salidas se encuentran a ‘0’.
- Q4, en este estado la salida C estará a ‘1’ y se llegará cuando ST se ponga a ‘0’, a partir de este estado se puede ir a Q5 si ST está a ‘0’ y CER a ‘1’, también se puede ir a Q6 si estas mismas entradas tienen el valor opuesto.
- Q5, en este estado únicamente estará a ‘1’ la salida CE.
- Q6, en este estado todas las salidas estarán a ‘0’.
- Q7, se llega a este estado cuando la entrada ST es ‘0’ y todas las salidas de este estado son todas ‘0’ excepto S cuyo valor vale ‘1’
- Q8, llegamos a este estado si AJ es ‘0’, se ponen todas las salidas a ‘0’ y desde este estado se va al estado inicial cuando AJ vuelve a valer ‘1’.

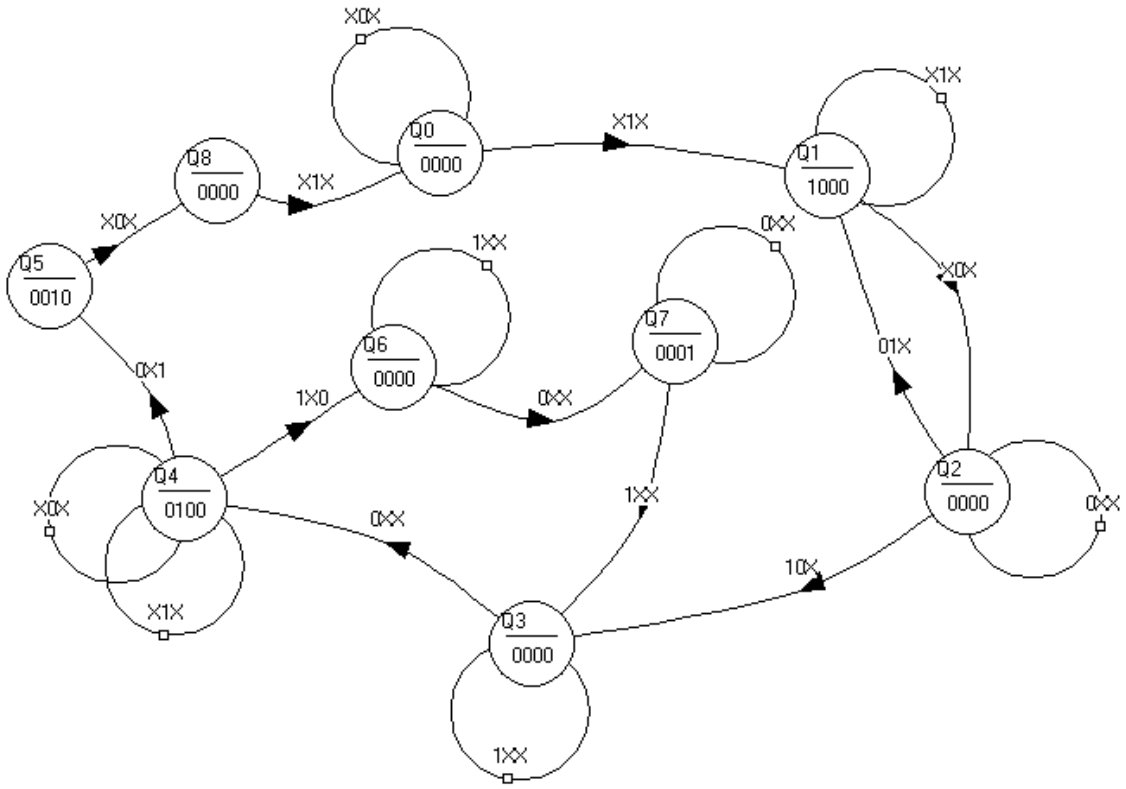


Ilustración 84. Máquina de estados CUENTAATRAS_FSM

En la ilustración nº85 podemos ver implementada la máquina de Moore de nuestro controlador en VHDL.

```
process (CLK,RST)
begin
   if RST = '1' then
      ESTADO<=Q0;
   elsif rising_edge (CLK) then
      case ESTADO is
         when Q0 =>
              if AJ='1'then
                 ESTADO<=Q1;
            else
               ESTADO<=Q0;
            end if;
         when Q1 =>
            if AJ ='0' then
               ESTADO<=Q2;
            else
               ESTADO<=Q1;
            end if;
         when Q2 =>
            if ST ='1' and AJ ='0' then
               ESTADO<=Q3;
            elsif AJ ='1' and ST ='0'then
               ESTADO<=Q1;
            else
               ESTADO<=Q2;
            end if;
         when Q3 =>
            if ST ='0' then
               ESTADO<=Q4;
            else
               ESTADO<=Q3;
            end if;
         when Q4 =>
            if ST='1' and CER ='0' then
               ESTADO<=Q6;
            elsif ST='0' and CER ='1' then
               ESTADO<=Q5;
            else
               ESTADO<=Q4;
            end if;
```

```
            when Q5 =>
                if AJ='0'then
                  ESTADO<=Q8;
               else
                  ESTADO<=Q5;
               end if;
            when Q6 =>
               if ST ='0'then
                  ESTADO<=Q7;
               else
                  ESTADO<=Q6;
               end if;
            when Q7 =>
               if ST ='1' then
                  ESTADO<=Q3;
               else
                  ESTADO<=Q7;
               end if;
            when Q8 =>
               if AJ ='1' then
                  ESTADO<=Q1;
               else
                  ESTADO<=Q8;
               end if;
         end case;
      end if;
   end process;

   A <= funcion (3);
   C <= funcion (2);
   CE <= funcion (1);
   S <= funcion (0);


   with ESTADO select funcion <=

      "0000" when Q0,
      "1000" when Q1,
      "0000" when Q2,
      "0000" when Q3,
      "0100" when Q4,
      "0010" when Q5,
      "0000" when Q6,
      "0001" when Q7,
      "0000" when Q8;

   end CUENTAATRAS_FSM_arch;
```

Ilustración 85. Proceso CUENTAATRAS_FSM

La lista de sensibilidad y la estructura del proceso es exactamente igual que la máquina de moore explicada en el punto 4.5.2.1.

```
component CUENTAATRAS_FSM is
   port     (
            CLK : in STD_LOGIC;
            RST : in STD_LOGIC;
            AJ : in STD_LOGIC;
            ST : in STD_LOGIC;
            CER : in STD_LOGIC;
            A : out STD_LOGIC;
            C : out STD_LOGIC;
            CE : out STD_LOGIC;
            S : out STD_LOGIC
            );
end component;
```

Ilustración 86. Componente CUENTAATRAS_FSM

```
FSM_cuenta : CUENTAATRAS_FSM
   port map(clk,rst,AJUSTE,KEY2,CER,AJ,COMIENZO,CERO,STOP);
```

Ilustración 87. Instanciación componente CUENTAATRAS_FSM

Con este componente se pretenderá controlar las distintas señales de entrada de los componentes que forman la entidad.

En la ilustración nº87 se observamos las asignaciones realizadas:

- CLK se asigna a CLK, ambas de tipo lógico, con esto se consigue que el reloj con el que ejecutará la máquina será el de la entidad.
- RST se asigna a RST, ambas de tipo lógico, con esta instanciación se consigue que está entrada del componente tenga el mismo valor que la entrada RST de la entidad.
- AJUSTE se asigna a AJ, ambas de tipo lógico, será una de las dos entradas de nuestra máquina de estados.
- KEY2 se asigna a ST, ambas de tipo lógico, es la entrada restante fundamental para el cambio de estado.
- CER asigna a CER, ambas de tipo lógico, será una de las salidas mostradas a la entidad donde se instancie.
- AJ asigna a A, ambas de tipo lógico, será una de las salidas mostradas a la entidad donde se instancie.
- COMIENZO asigna a C, ambas de tipo lógico, será una de las salidas mostradas a la entidad donde se instancie.
- CERO asigna a CE ambas de tipo lógico, será una de las salidas mostradas a la entidad donde se instancie.
- STOP asigna a S, ambas de tipo lógico, será una de las salidas mostradas a la entidad donde se instancie.

4.8.2. CUENTAATRAS_CONFIGURACIÓN: Descripción, declaración e instanciación

En esta sección no vamos a explicar la entidad de CUENTAATRAS_CONFIGURACION debido a que es exactamente igual que la entidad RELOJ_CONFIGURACION exceptuando que uno de los bloques cambia de valores, ese componente en concreto sí que se estudiará en una sección específica.

Este componente estará incluido en la entidad de CUENTAATRAS_PROCESOS, su función será la de modificar los valores de los minutos y segundos de forma ascendente y descendente.

Para entender mejor como está conectado se puede ver el diseño RTL de la ilustración nº78.

```
component CUENTAATRAS_CONFIGURACION is
   port     (
            AJ : in std_logic;
            RST : in std_logic;
            CLK : in std_logic;
            KEY0 : in std_logic;
            KEY1 : in std_logic;
            KEY3 : in std_logic;
            KEY4 : in std_logic;
            EDATO0 : in std_logic_vector (3 downto 0);
            EDATO1 : in std_logic_vector (3 downto 0);
            EDATO2 : in std_logic_vector (3 downto 0);
            EDATO3 : in std_logic_vector (3 downto 0);
            EDATO4 : in std_logic_vector (3 downto 0);
            EDATO5 : in std_logic_vector (3 downto 0);
            SDATO0 : out std_logic_vector (3 downto 0);
            SDATO1 : out std_logic_vector (3 downto 0);
            SDATO2 : out std_logic_vector (3 downto 0);
            SDATO3 : out std_logic_vector (3 downto 0);
            SDATO4 : out std_logic_vector (3 downto 0);
            SDATO5 : out std_logic_vector (3 downto 0)
            );
end component;
```

Ilustración 88. Componente CUENTAATRAS_CONFIGURACION

```
PARTE1 : CUENTAATRAS_CONFIGURACION
   port map(AJ,rst,clk,KEY0,KEY1,KEY3,KEY4,SRA1,SRA2,SRA3,SRA4,SRA5,SRA6,S11,S12,S13,S14,S15,S16);
```

Ilustración 89. Instanciación componente CUENTAATRAS_CONFIGURACION

La declaración de este componente dentro de una entidad de realiza como ya se ha explicado y se ve en la ilustración nº88.

Para la instanciación de éste componente recordemos que se utilizará para definir la instanciación las palabras “port map” seguido de las relaciones de las entradas y salidas declaradas en el componente y las señales descritas en la entidad como se puede ver en la ilustración nº89.

Las relaciones serían las siguientes:

- AJ se asigna a AJ, ambas son de tipo lógico. AJ es una entrada de la entidad RELOJ_CONFIGURACION.

- RST se asigna a RST, ambas son de tipo lógico, al igual que el caso anterior, RST es una entrada de la entidad RELOJ_CONFIGURACION.
- CLK se asigna a CLK, son de tipo lógico. CLK también es una entrada de la entidad RELOJ_CONFIGURACION.
- SRA1 se asigna a EDATO1, ambas son vectores binarios y es una de las señales que recibe este componente.
- SRA2 se asigna a EDATO2, ambas son vectores binarios y es una de las señales que recibe este componente.
- SRA3 se asigna a EDATO3, ambas son vectores binarios y es una de las señales que recibe este componente.
- SRA4 se asigna a EDATO4, ambas son vectores binarios y es una de las señales que recibe este componente.
- SRA5 se asigna a EDATO5, ambas son vectores binarios y es una de las señales que recibe este componente.
- SRA6 se asigna a EDATO6, ambas son vectores binarios y es una de las señales que recibe este componente.
- S11 se asigna a SDATO0, ambas son vectores binarios y son las señales que se proporcionarán al registro B.
- S12 se asigna a SDATO1, ambas son vectores binarios y son las señales que se proporcionarán al registro B.
- S13 se asigna a SDATO2 ambas son vectores binarios y son las señales que se proporcionarán al registro B.
- S14 se asigna a SDATO3, ambas son vectores binarios y son las señales que se proporcionarán al registro B.
- S15 se asigna a SDATO4, ambas son vectores binarios y son las señales que se proporcionarán al registro B.
- S16 se asigna a SDATO5, ambas son vectores binarios y son las señales que se proporcionarán al registro B.

4.8.2.1. PARTE_99: Descripción, declaración e instanciación

En esta entidad solo habrá una diferencia en el proceso que la forma por lo que no hace falta explicar la declaración ni la instanciación debido a que se realiza de la misma manera que RELOJ_CONFIGURACION.

```
process (RST,CLK_U,SRU)
begin
if RST = '1' then
   UNIDADES <= 0;
elsif  CLK_U'event and CLK_U = '1' then
   if SRU = "11" then
   UNIDADES <=TO_INTEGER(unsigned(EUNIDADES));
   elsif SRU = "10" then
   UNIDADES <= UNIDADES + 1;
      if DECENAS = 9 and UNIDADES = 9 then
         UNIDADES <= 0;
         RESET <= '1';
      elsif UNIDADES = 9 then
         UNIDADES <= 0;
         RESET <= '0';
      end if;
   elsif SRU = "01"then
      if UNIDADES /= 0 then
      UNIDADES <= UNIDADES - 1;
         CLK3 <= '0';
         RESETD <= '0';
      elsif UNIDADES = 0 and DECENAS = 0 then
         UNIDADES <= 9;
         RESETD <= '1';
         CLK3 <= '1';
      elsif UNIDADES = 0 then
         UNIDADES <= 9;
         RESETD<='0';
         CLK3 <= '1';
      end if;
   else
   UNIDADES <=TO_INTEGER(unsigned(EUNIDADES));
   end if;
end if;
SUNIDADES <= std_logic_vector(to_unsigned(UNIDADES,4));
end process;


process (RST,CLK_D,SRD)
begin
   if RST = '1' then
      DECENAS <= 0;
   elsif CLK_D'event and CLK_D = '1' then
      if SRD = "11"then
      DECENAS <=TO_INTEGER(unsigned(EDECENAS));
      elsif SRD = "10"then
      DECENAS <= DECENAS + 1;
         if RESET ='1' then
            DECENAS <= 0;
         end if;
      elsif SRD = "01" then
      DECENAS <= DECENAS - 1;
         if RESETD = '1' then
         DECENAS <= 9;
         end if;
      end if;
   end if;
SDECENAS <= std_logic_vector(to_unsigned(DECENAS,4));
end process;
```

Ilustración 90. Proceso entidad PARTE_99

Como se observa en la ilustración anterior los dos procesos constan de la misma estructura que las entidades PARTE_23 y PARTE_59, con la diferencia que en este caso el proceso podrá sumar hasta 99 y se reiniciaría la cuenta a 0 y en el caso de restar una vez se llegue a 0 el siguiente número seria el 99.

```
component PARTE_59 is
   port     (
            CLK : in std_logic;
            RST : in std_logic;
            AJUSTE : in std_logic;
            AJUSTE_S : in std_logic;
            AJUSTE_R : in std_logic;
            EDECENAS : in std_logic_vector (3 downto 0);
            EUNIDADES : in std_logic_vector (3 downto 0);
            SDECENAS : out std_logic_vector (3 downto 0) :=(others=>'0');
            SUNIDADES : out std_logic_vector (3 downto 0) :=(others=>'0')
            );
end component;
```

Ilustración 91. Componente PARTE_59

```
cambioHORAS : PARTE_99
   port map (CLK,RST,AJ,not KEY0,not KEY1,E5,E4,S5,S4);
```

Ilustración 92. Instanciación componente PARTE_59

La declaración es igual que las entidades declaradas PARTE_23 y PARTE_59, debido a que se ha procurado seguir un patrón por facilitar en medida de los posible la interpretación.

Para la instanciación se han usado las siguientes asignaciones:

- CLK se asigna a CLK, ambos de tipo lógico y servirá para utilizar el reloj declarado como entrada en la entidad.
- RST se asigna a RST, ambas de tipo lógico y al igual que ocurre con CLK, utilizará el valor de RST declarado a la entrada.
- AJ se asigna a AJUSTE, ambas de tipo lógico. AJ es una entrada de la entidad que dará valor el valor correspondiente.
- Not KEY0 se asigna a AJUSTE_S, ambas de tipo lógico. La idea de complementar el valor de KEY0, es para que dé un valor lógico alto cuando de como salida un ‘1’, conectará este componente con la entrada de la entidad.

- Not KEY1 se asigna a AJUSTE_R, ambas de tipo lógico. La idea de complementar el valor de KEY1, al igual que el caso anterior, es para dar un valor lógico alto cuando de como salida un '1', conectará este componente con la entrada de la entidad.
- E5 se asigna a EDECENAS, ambas son un vector de tipo binario que serán la entrada de las decenas del componente PARTE_99 y cuyo valor puede variar dependiendo del valor de.
- E4 se asigna a EUNIDADES, ambas son un vector de tipo binario que serán la entrada de las unidades del componente PARTE_99, cuyo valor puede variar al igual que en caso anterior con la señal AJ.
- S5 se asigna a SDECENAS, ambas son un vector de tipo binario y serán la salida de la parte decenas del componente PARTE_99, puede volver a ser parte de la entrada de dicho componente.
- S4 se asigna a SUNIDADES, ambas son un vector de tipo binario y serán la salida de la parte unidades del componente PARTE_99, puede volver a ser parte de la entrada de dicho componente.

```
E4<= EDATO4 when AJ='0' else S4 when AJ='1' else EDATO4;
E5<= EDATO5 when AJ='0' else S5 when AJ='1' else EDATO5;
```

Ilustración 93. Valor de E4 y E5

4.8.2.2. PARTE_59: Descripción, declaración e instanciación

Es exactamente el mismo que la entidad usada en el punto 4.5.3.

Para la declaración e instanciación ocurre lo mismo, es exactamente igual debido a que el bloque de configuración tanto en RELOJ_PROCESOS como en CUENTAATRAS_PROCESOS es igual.

4.8.3. CUENTAATRAS_PROCESO: Declaración e instanciación

Debido a la complejidad de la entidad se la descripción de la misma se realizará en una sección específica.

Este componente, a diferencia del anterior, es el encargado de realizar la cuenta regresiva que se puede ver en cualquier cuenta atrás.

Para comprender mejor la situación de este componente en la entidad, podemos ver el diseño RTL de la ilustración nº78.

```
component CUENTAATRAS_PROCESO is
   port     (
            clksignal : in std_logic;
            RST : in std_logic;
            comienzo : in std_logic;
            pausa : in std_logic;
            DATOSG01IN : in std_logic_vector (3 downto 0);
            DATOSG10IN : in std_logic_vector (3 downto 0);
            DATOMIN01IN : in std_logic_vector (3 downto 0);
            DATOMIN10IN : in std_logic_vector (3 downto 0);
            DATOHR01IN : in std_logic_vector (3 downto 0);
            DATOHR10IN : in std_logic_vector (3 downto 0);
            DATOSG01OUT : out std_logic_vector (3 downto 0);
            DATOSG10OUT : out std_logic_vector (3 downto 0);
            DATOMIN01OUT : out std_logic_vector (3 downto 0);
            DATOMIN10OUT : out std_logic_vector (3 downto 0);
            DATOHR01OUT : out std_logic_vector (3 downto 0);
            DATOHR10OUT : out std_logic_vector (3 downto 0)
            );
end component;
```

Ilustración 94. Componente CUENTAATRAS_PROCESO

```
PARTE2 : CUENTAATRAS_PROCESO
   port map(clk,CERO,COMIENZO,STOP,SRB1,SRB2,SRB3,SRB4,SRB5,SRB6,S21,S22,S23,S24,S25,S26);
```

Ilustración 95. Instanciación componente CUENTAATRAS_PROCESO

La declaración del componente dentro de la entidad donde se pretende usar, se realizará conforme se puede ver en la ilustración nº94.

Una vez declarado el componente se necesitará instanciarlo para ello con la ayuda de la ilustración nº95, se puede ver que se ha realizado las siguientes asignaciones:

- CLK se asigna a CLK, ambas son de tipo lógico. La señal CLK instanciada nos dará la señal de reloj a la que trabajará este componente.
- not AJ se asigna a AJUSTE, ambas son de tipo lógico, es una entrada que nos permitirá copiar los valores del componente RELOJ_CONFIGURACION o realizar la secuencia que realiza un reloj común.

- SRB1 se asigna a DATOSG01IN, ambas son vectores binarios y es una de las señales que recibe este componente del registro B0.
- SRB2 se asigna a DATOSG10IN, ambas son vectores binarios y es una de las señales que recibe este componente del registro B1.
- SRB3 se asigna a DATOMIN01IN, ambas son vectores binarios y es una de las señales que recibe este componente del registro B2.
- SRB4 se asigna a DATOMIN10IN, ambas son vectores binarios y es una de las señales que recibe este componente del registro B3.
- SRB5 se asigna a DATOHR01IN, ambas son vectores binarios y es una de las señales que recibe este componente del registro B4.
- SRB6 se asigna a DATOHR10IN, ambas son vectores binarios y es una de las señales que recibe este componente del registro B5.
- S21 se asigna a DATOSG01OUT, ambas son vectores binarios y es la señal que recibirá el registro A0.
- S22 se asigna a DATOSG10OUT, ambas son vectores binarios y es la señal que recibirá el registro A1.
- S23 se asigna a DATOMIN01OUT, ambas son vectores binarios y es la señal que recibirá el registro A2.
- S24 se asigna a DATOMIN10OUT, ambas son vectores binarios y es la señal que recibirá el registro A3.
- S25 se asigna a DATOHR01OUT, ambas son vectores binarios y es la señal que recibirá el registro A4.
- S26 se asigna a DATOHR10OUT, ambas son vectores binarios y es la señal que recibirá el registro A5.
- CERO asigna a CERO, ambas son de tipo lógico. Esta señal indicará cuando llega a 0 nuestra cuenta.

4.9. CUENTAATRAS_PROCESO

La función principal como ya se ha comentado es restar una unidad el valor que se haya fijado cada cierto periodo que se configura a través del componente DIVISOR.

```
entity CUENTAATRAS_PROCESO is
port(
     clksignal : in std_logic;
     RST : in std_logic;
     comienzo : in std_logic;
     pausa : in std_logic;
     DATOSG01IN : in std_logic_vector (3 downto 0);
     DATOSG10IN : in std_logic_vector (3 downto 0);
     DATOMIN01IN : in std_logic_vector (3 downto 0);
     DATOMIN10IN : in std_logic_vector (3 downto 0);
     DATOHR01IN : in std_logic_vector (3 downto 0);
     DATOHR10IN : in std_logic_vector (3 downto 0);
     DATOSG01OUT : out std_logic_vector (3 downto 0);
     DATOSG10OUT : out std_logic_vector (3 downto 0);
     DATOMIN01OUT : out std_logic_vector (3 downto 0);
     DATOMIN10OUT : out std_logic_vector (3 downto 0);
     DATOHR01OUT : out std_logic_vector (3 downto 0);
     DATOHR10OUT : out std_logic_vector (3 downto 0)
     );
end CUENTAATRAS_PROCESO;

architecture CUENTAATRAS_PROCESO_arch of CUENTAATRAS_PROCESO is

   signal centesimas_s : integer range 0 to 9;
   signal decimas_s : integer range 0 to 9;
   signal unidades_s : integer range 0 to 9;
   signal decenas_s : integer range 0 to 9;
   signal unidades_m : integer range 0 to 9;
   signal decenas_m : integer range 0 to 9;
   signal clk0 : std_logic;
   signal clk1 : std_logic;
   signal clk2 : std_logic;
   signal clk3 : std_logic;
   signal clk4 : std_logic;
   signal clk5 : std_logic;
```

Ilustración 96. Descripción entidad CUENTAATRAS_PROCESO

Para la realización de los procesos se han utilizado una serie de señales que a continuación explicaremos la función que tendrá cada una.

- centesimas_s, señal de tipo entero que representará las centésimas de segundo, su valor variará durante los procesos de la entidad.
- decimas_s, señal de tipo entero que representará las décimas de segundo, su valor variará durante los procesos de la entidad.
- unidades_s, señal de tipo entero que representará las unidades de segundos, su valor variará durante los procesos de la entidad.
- decenas_s, señal de tipo entero que representará las decenas de segundos, su valor variará durante los procesos de la entidad.
- unidades_m, señal de tipo entero que representará las unidades de minutos, su valor variará durante los procesos de la entidad.

- decenas_m, señal de tipo entero que representará las unidades de minutos, su valor variará durante los procesos de la entidad.
- Clk0, señal de tipo lógico recibida del proceso donde varía el valor de las centésimas de segundo y necesaria para la realizar un proceso.
- Clk1, señal de tipo lógico recibida del proceso donde varía el valor de las décimas de segundo y necesaria para la realizar un proceso.
- Clk2, señal de tipo lógico recibida del proceso donde varía el valor de las unidades de segundo y necesaria para la realizar un proceso.
- Clk3, señal de tipo lógico recibida del proceso donde varía el valor de las decenas de segundo y necesaria para la realizar un proceso.
- Clk4, señal de tipo lógico recibida del proceso donde varía el valor de las unidades de minuto y necesaria para la realizar un proceso.

4.9.1. Proceso CUENTAATRAS_PROCESO

En la siguiente ilustración procederemos a describir más a fondo los distintos procesos que componen a la entidad que estamos tratando.

Todos los procesos se pueden contemplar en la ilustración nº94.

En el primer proceso serán donde se activen las señales de reloj que harán que se ejecuten los diferentes procesos que componen la entidad, se diferenciarán mediante las palabras reservadas “if” y “elsif” las distintas posibilidades de activación que estas señales tendrán. Cada vez que las centésimas lleguen a ‘0’ siempre hay que valorar la posibilidad de que haya un número distinto de cero a la izquierda, en caso de haberlo se activarán todas las señales hasta dicho número, con la finalidad de que se resten todos hasta él, de esta forma todos los valores irán tomando el valor de ‘0’ progresivamente.

El resto de los procesos tienen una estructura muy parecida al CRONOMETRO_PROCESO, con la diferencia de que restan y una vez el dato tenga el valor cero se reestablecerá el 9. Dependiendo de la unidad que se trate este valor será distinto, en concreto el único caso excepcional es con las decenas de segundo que será un 5.

```
process (clk0)
begin
   if RST = '0' then
      if comienzo = '1' then
         if PAUSA ='0' then
            if clk0 ='1' then
            centesimas_s <=TO_INTEGER(unsigned(DATOSG01IN));
               if centesimas_s /= 0 then
                  centesimas_s <= centesimas_s - 1;
                  CLK1 <= '0';
               elsif centesimas_s = 0 and decimas_s = 0 and unidades_s = 0 and decenas_s = 0 and unidades_m = 0 and decenas_m /= 0 then
                  centesimas_s <= 9;
                  CLK1 <= '1';
                  CLK2 <= '1';
                  CLK3 <= '1';
                  CLK4 <= '1';
                  CLK5 <= '1';
               elsif centesimas_s = 0 and decimas_s = 0 and unidades_s = 0 and decenas_s = 0 and unidades_m /= 0 then
                  centesimas_s <= 9;
                  CLK1 <= '1';
                  CLK2 <= '1';
                  CLK3 <= '1';
                  CLK4 <= '1';
                  CLK5 <= '0';
               elsif centesimas_s = 0 and decimas_s = 0 and unidades_s = 0 and decenas_s /= 0 then
                  centesimas_s <= 9;
                  CLK1 <= '1';
                  CLK2 <= '1';
                  CLK3 <= '1';
                  CLK4 <= '0';
                  CLK5 <= '0';
               elsif centesimas_s = 0 and decimas_s = 0 and unidades_s /= 0 then
                  centesimas_s <= 9;
                  CLK1 <= '1';
                  CLK2 <= '1';
                  CLK3 <= '0';
                  CLK4 <= '0';
                  CLK5 <= '0';
               elsif centesimas_s = 0 and decimas_s /= 0 then
                  centesimas_s <= 9;
                  CLK1 <= '1';
                  CLK2 <= '0';
                  CLK3 <= '0';
                  CLK4 <= '0';
                  CLK5 <= '0';
               end if;
            end if;
         DATOSG01OUT <= std_logic_vector(to_unsigned(centesimas_s,4));
         else
         centesimas_s <=TO_INTEGER(unsigned(DATOSG01IN));
         end if;
      DATOSG01OUT <= std_logic_vector(to_unsigned(centesimas_s,4));
      else
      centesimas_s <=TO_INTEGER(unsigned(DATOSG01IN));
      end if;
   DATOSG01OUT <= std_logic_vector(to_unsigned(centesimas_s,4));
   else
   centesimas_s<=0;
   end if;
end process;
```

```
process (clk1)
begin
   if RST = '0' then
      if comienzo = '1' then
         if PAUSA ='0' then
            if clk1'EVENT AND clk1 ='1' then
            decimas_s <=TO_INTEGER(unsigned(DATOSG10IN));
               if decimas_s /= 0 then
                  decimas_s <= decimas_s - 1;
               elsif decimas_s = 0 then
                  decimas_s <= 9;
               end if;
            end if;
         DATOSG10OUT <= std_logic_vector(to_unsigned(decimas_s,4));
         else
         decimas_s <=TO_INTEGER(unsigned(DATOSG10IN));
         end if;
      DATOSG10OUT <= std_logic_vector(to_unsigned(decimas_s,4));
      else
      decimas_s <=TO_INTEGER(unsigned(DATOSG10IN));
      end if;
   DATOSG10OUT <= std_logic_vector(to_unsigned(decimas_s,4));
   else
   decimas_s<=0;
   end if;
end process;

process (clk2)
begin
   if RST = '0' then
      if comienzo = '1' then
         if PAUSA='0' then
            if clk2'EVENT AND clk2 ='1' then
            unidades_s <=TO_INTEGER(unsigned(DATOMIN01IN));
               if unidades_s /= 0 then
                  unidades_s <= unidades_s - 1;
               elsif unidades_s = 0 then
                  unidades_s <= 9;
               end if;
            end if;
         DATOMIN01OUT <= std_logic_vector(to_unsigned(unidades_s,4));
         else
         unidades_s <=TO_INTEGER(unsigned(DATOMIN01IN));
         end if;
      DATOMIN01OUT <= std_logic_vector(to_unsigned(unidades_s,4));
      else
      unidades_s <=TO_INTEGER(unsigned(DATOMIN01IN));
      end if;
   DATOMIN01OUT <= std_logic_vector(to_unsigned(unidades_s,4));
   else
   unidades_s<=0;
   end if;
end process;


process (clk3)
begin
   if RST = '0' then
      if comienzo = '1' then
         if PAUSA ='0' then
            if clk3'EVENT AND clk3 ='1' then
            decenas_s <=TO_INTEGER(unsigned(DATOMIN10IN));
               if decenas_s /= 0 then
                  decenas_s <= decenas_s - 1;
               elsif decenas_s = 0 then
                  decenas_s <= 5;
               end if;
            end if;
         DATOMIN10OUT <= std_logic_vector(to_unsigned(decenas_s,4));
         else
         decenas_s <=TO_INTEGER(unsigned(DATOMIN10IN));
         end if;
      DATOMIN10OUT <= std_logic_vector(to_unsigned(decenas_s,4));
      else
      decenas_s <=TO_INTEGER(unsigned(DATOMIN10IN));
      end if;
   DATOMIN10OUT <= std_logic_vector(to_unsigned(decenas_s,4));
   else
   decenas_s<=0;
   end if;
end process;
```

```
process (clk4)
begin
   if RST = '0' then
      if comienzo = '1' then
         if PAUSA ='0' then
            if clk4'EVENT AND clk4 ='1' then
            unidades_m <=TO_INTEGER(unsigned(DATOHR01IN));
               if unidades_m /= 0 then
               unidades_m <= unidades_m - 1;
               elsif unidades_m = 0 then
                  unidades_m <= 9;
               end if;
            end if;
         DATOHR01OUT <= std_logic_vector(to_unsigned(unidades_m,4))
         else
         unidades_m <=TO_INTEGER(unsigned(DATOHR01IN));
         end if;
      DATOHR01OUT <= std_logic_vector(to_unsigned(unidades_m,4));
      else
      unidades_m <=TO_INTEGER(unsigned(DATOHR01IN));
      end if;
   DATOHR01OUT <= std_logic_vector(to_unsigned(unidades_m,4));
   else
   unidades_m<=0;
   end if;
end process;

process (clk5)
begin
   if RST = '0' then
      if comienzo = '1' then
         if PAUSA ='0' then
            if clk5'EVENT AND clk5 ='1' then
               decenas_m <=TO_INTEGER(unsigned(DATOHR10IN));
               if decenas_m /= 0 then
               decenas_m <= decenas_m - 1;
               elsif decenas_m = 0 then
                  decenas_m <= 9;
               end if;
            end if;
         DATOHR10OUT <= std_logic_vector(to_unsigned(decenas_m,4));
         else
         decenas_m <=TO_INTEGER(unsigned(DATOHR10IN));
         end if;
      DATOHR10OUT <= std_logic_vector(to_unsigned(decenas_m,4));
      else
      decenas_m <=TO_INTEGER(unsigned(DATOHR10IN));
      end if;
   DATOHR10OUT <= std_logic_vector(to_unsigned(decenas_m,4));
   else
   decenas_m<=0;
   end if;
end process;

end CUENTAATRAS_PROCESO_arch;
```

Ilustración 97. Proceso entidad CUENTAATRAS_PROCESO

4.9.2. DIVISOR

En este caso, no hará falta describir la entidad DIVISOR porque no hay ninguna diferencia con la explicada en el punto 4.6.2.

En cuanto a la declaración e instanciación tampoco hay necesidad de explicarla porque también es la misma debido a que la frecuencia de reloj que se requiere es la misma y se utilizan señales con el mismo nombre para realizar el mismo trabajo.

4.10. FUNCIÓN ALARMA

La función de esta entidad será la de hacer sonar una melodía, una vez coincida una hora fijada por este modo de uso y la hora del reloj.

```
entity ALARMA_PROCESOS is
   port      (
             clk : in std_logic;
             ACT : in std_logic;
             RST : in std_logic;
             ajuste_hhS : in std_logic;
             ajuste_mmS : in std_logic;
             ajuste_hhR : in std_logic;
             ajuste_mmR : in std_logic;
             datoMin01in : in std_logic_vector (3 downto 0);
             datoMin10in : in std_logic_vector (3 downto 0);
             datoHr01in : in std_logic_vector (3 downto 0);
             datoHr10in : in std_logic_vector (3 downto 0);
             datoSg01out : out std_logic_vector (3 downto 0);
             datoSg10out : out std_logic_vector (3 downto 0);
             datoMin01out : out std_logic_vector (3 downto 0);
             datoMin10out : out std_logic_vector (3 downto 0);
             datoHr01out : out std_logic_vector (3 downto 0);
             datoHr10out : out std_logic_vector (3 downto 0);
             MUSICA : out std_logic
             );
end ALARMA_PROCESOS;

architecture ALARMA_PROCESOS_arch of ALARMA_PROCESOS is

   signal SA : std_logic;
   signal Sg01 : std_logic_vector (3 downto 0);
   signal Sg10 : std_logic_vector (3 downto 0);
   signal Min01 : std_logic_vector (3 downto 0);
   signal Min10 : std_logic_vector (3 downto 0);
   signal Hr01 : std_logic_vector (3 downto 0);
   signal Hr10 : std_logic_vector (3 downto 0);
   signal SALIDA : std_logic;
   signal reloj : std_logic;
```

Ilustración 98. Entidad CUENTAATRAS_PROCESO

Las entradas y salidas de esta entidad se muestran en la ilustración nº98. Otra información importante para entender el funcionamiento de esta entidad son las señales usadas por esta entidad serán las siguientes

- SA, señal de tipo entero que representará las centésimas de segundo, su valor variará conforme varíen los valores de los componentes presentes en la entidad, se muestra en la ilustración nº99.
- Sg01, señal de tipo vector binario que representará las unidades de segundo, su valor variará durante el componente ALARMA_CONFIGURACION.

- Sg10, señal de tipo vector binario que representará las decenas de segundo, su valor variará durante el componente ALARMA_CONFIGURACION.
- Min01, señal de tipo vector binario que representará las decenas de segundos, su valor variará durante el componente ALARMA_CONFIGURACION.
- Min10, señal de tipo vector binario que representará las unidades de minutos, su valor variará durante el componente ALARMA_CONFIGURACION.
- Hr01, señal de tipo vector binario que representará las unidades de minutos, su valor variará durante el componente ALARMA_CONFIGURACION.
- Hr10, señal de tipo vector binario que representará las unidades de minutos, su valor variará durante el componente ALARMA_CONFIGURACION.
- SALIDA, señal de tipo lógico recibida del componente ALARMA_FSM y servirá para condicionar la señal reloj.
- reloj, señal de tipo lógico que estará condicionada como se muestra en la ilustración nº100.

```
SA <= '1' when datoMin01in = Min01 and datoMin10in = Min10 and datoHr01in = Hr01 and datoHr10in = Hr10 else '0';
```

Ilustración 99. Señal SA

```
reloj<=clk when SALIDA = '1' else '0' when SALIDA = '1' else '0';
```

Ilustración 100. Señal reloj

4.10.1. ALARMA_FSM: Descripción, componente, instanciación

Para el control de la activación de la alarma se usará una pequeña máquina de estados únicamente de dos salidas y una entrada.

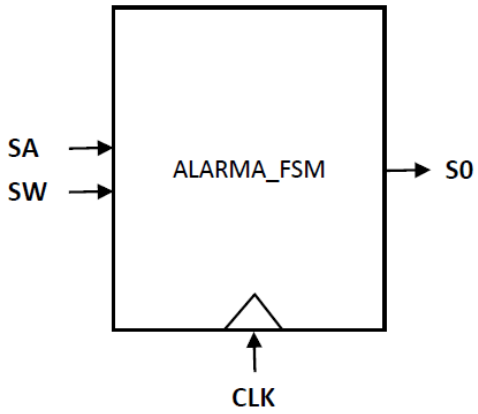


Ilustración 101. Entradas y salidas ALARMA_FSM

```
entity ALARMA_FSM is
   port     (
            CLK : in STD_LOGIC;
            SA : in STD_LOGIC;
            SW : in STD_LOGIC;
            SO : out STD_LOGIC
            );
end ALARMA_FSM;

architecture ALARMA_FSM_arch of ALARMA_FSM is

TYPE ESTADOS is (Q0,Q1);
signal ESTADO : ESTADOS := Q0;
signal funcion : std_logic;
```

Ilustración 102. Entidad ALARMA_FSM

En las señales que se observan podemos destacar:

- ESTADO, la función de esta señal será la de asignar los valores que se han definido en la línea “TYPE”.
- funcion, se trata de vector binario de 4 valores cuyo valor irá asignado a los diferentes estados definidos en “TYPE”.

Antes de explicar que realizará la máquina de estados podemos observar en la ilustración nº100 las entradas y salidas de un estado.

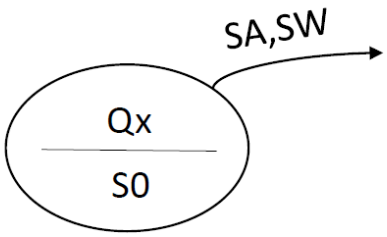


Ilustración 103. Estado ALARMA_FSM

En la máquina de estados podemos distinguir un total de 9 estados, explicaremos las salidas de cada uno en función de las entradas que recibe.

- Q0, es el estado inicial, en este estado la salida se encuentra a '0'.
- Q1, en este estado se pondrá a '1' la salida, debido a que se ha recibido un '1' de ambas entradas, para volver a Q0 es necesario que SW esté a 0.

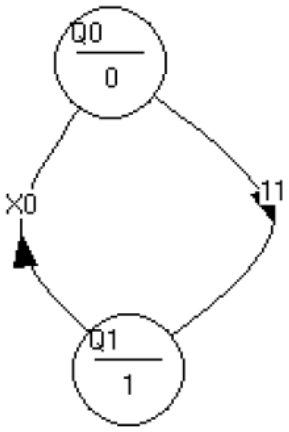


Ilustración 104. Máquina de estados ALARMA_FSM

En la ilustración nº105 podemos ver implementada la máquina de Moore de nuestro controlador en VHDL.

```
process (CLK)
begin
   if rising_edge (CLK) then
      case ESTADO is
         when Q0 =>
             if SA='1' and SW='1' then
               ESTADO<=Q1;
            else
               ESTADO<=Q0;
            end if;
         when Q1 =>
            if SW ='0' then
               ESTADO<=Q0;
            else
               ESTADO<=Q1;
            end if;
      end case;
   end if;
end process;

SO <= funcion;
with ESTADO select funcion <=

   '0' when Q0,
   '1' when Q1;

end ALARMA_FSM_arch;
```

Ilustración 105. Proceso ALARMA_FSM

La lista de sensibilidad está compuesta únicamente por la señal de reloj el proceso que muestra en la ilustración anterior.

```
component ALARMA_FSM is
   port      (
             CLK : in STD_LOGIC;
             SA : in STD_LOGIC;
             SW : in STD_LOGIC;
             SO : out STD_LOGIC
             );
end component;
```

Ilustración 106. Componente ALARMA_FSM

```
FSM_ALARM : ALARMA_FSM
   port map(clk,SA,ACT,SALIDA);
```

Ilustración 107. Instanciación componente ALARMA_FSM

En la ilustración nº107 se observamos las asignaciones realizadas:

- CLK se asigna a CLK, ambas de tipo lógico, con esto se consigue que el reloj con el que ejecutará la máquina será el de la entidad.

- SA se asigna a SA, ambas de tipo lógico, es la señal que se activa como hemos comentado anteriormente.
- ACT se asigna a SW, ambas de tipo lógico, cogerá el valor de una de las entradas de la entidad.
- SALIDA se asigna a SO, ambas de tipo lógico, es la señal que sale de la única salida de la máquina de estados.

4.10.2. ALARMA_CONFIGURACION: Descripción, componente, instanciación

La función principal de esta función de esta entidad será la de poder modificar las horas y minutos en esta entidad con el fin de compararlo con la hora que nos marca la entidad RELOJ_PROCESOS.

```
entity ALARMA_CONFIGURACION is
   port      (
             AJ : in std_logic;
             RST : in std_logic;
             CLK : in std_logic;
             KEY0 : in std_logic;
             KEY1 : in std_logic;
             KEY3 : in std_logic;
             KEY4 : in std_logic;
             SDATO0 : out std_logic_vector (3 downto 0);
             SDATO1 : out std_logic_vector (3 downto 0);
             SDATO2 : out std_logic_vector (3 downto 0);
             SDATO3 : out std_logic_vector (3 downto 0);
             SDATO4 : out std_logic_vector (3 downto 0);
             SDATO5 : out std_logic_vector (3 downto 0)
             );
end ALARMA_CONFIGURACION;

   signal E2 : std_logic_vector (3 downto 0);
   signal E3 : std_logic_vector (3 downto 0);
   signal E4 : std_logic_vector (3 downto 0);
   signal E5 : std_logic_vector (3 downto 0);
   signal S2 : std_logic_vector (3 downto 0);
   signal S3 : std_logic_vector (3 downto 0);
   signal S4 : std_logic_vector (3 downto 0);
   signal S5 : std_logic_vector (3 downto 0);
```

Ilustración 108. Entidad ALARMA_CONFIGURACION

Esta entidad contará con las entradas y salidas que pueden verse en la ilustración nº108. En cuanto a las señales utilizadas podemos destacar:

- E2, E3, ambas son de tipo vector binario de 4 elementos y serán las señales de entrada del componente PARTE_59.

- E4, E5, ambas son de tipo vector binario de 4 elementos y serán las señales de entrada del componente PARTE_23.
- S2, S3, ambas son de tipo vector binario de 4 elementos y serán las salidas del componente PARTE_59.
- S4, S5, ambas son de tipo vector binario de 4 elementos y serán las salidas del componente PARTE_23.

Esta entidad a diferencia de la entidad RELOJ_CONFIGURACIÓN, no tiene señales de entrada, por lo que las salidas de SDATO0 y SDATO1 se les dará el valor “0000” por defecto.

4.10.2.1. PARTE_23 y PARTE_59

Estos componentes no se describirán, declararán ni instanciarán debido a que se realiza de la misma manera que hemos hecho en el RELOJ_CONFIGURACIÓN.

4.10.3. SONIDO: Descripción, declaración e instanciación

La función de esta entidad será la de realizar una melodía a través de distintas frecuencias generadas por un divisor de frecuencia que tendrá que ser instanciado cada una de las veces que se desee cambiar de nota.

```
entity SONIDO is
   port     (
            MAX10_CLK1_50 : in std_logic;
            SALIDAMUSICA : out STD_LOGIC
            );
   end SONIDO;

architecture SONIDO_arch of SONIDO is

   signal clk : std_logic:='0';
   signal sonido : std_logic;
   signal DO : std_logic;
   signal RE : std_logic;
   signal MI : std_logic;
   signal FA : std_logic;
   signal SOL : std_logic;
   signal LA : std_logic;
   signal SI : std_logic;
   signal t : integer range 0 to 130 :=0;
```

Ilustración 109. Entidad SONIDO

En este caso solo habrá una entrada y una salida como se refleja en la ilustración nº109. Las señales utilizadas son:

- clk, señal de tipo lógico que enlazarán la entrada de la entidad con las entradas de los distintos divisores.
- sonido, señal de tipo lógico cuyo valor va variando conforme muestra la ilustración nº110.
- DO, señal de tipo lógico que enlazarán los distintos divisores de frecuencia con el proceso que las gestiona.
- RE, señal de tipo lógico que enlazarán los distintos divisores de frecuencia con el proceso que las gestiona.
- MI, señal de tipo lógico que enlazarán los distintos divisores de frecuencia con el proceso que las gestiona.
- FA, señal de tipo lógico que enlazarán los distintos divisores de frecuencia con el proceso que las gestiona.
- LA, señal de tipo lógico que enlazarán los distintos divisores de frecuencia con el proceso que las gestiona.
- SI, señal de tipo lógico que enlazarán los distintos divisores de frecuencia con el proceso que las gestiona.
- t, señal de tipo lógico que indicará el tiempo que estará cada nota sonando, se controla mediante un proceso.

```
sonido <= LA when (t>0 and t<8) else
          RE when (t>8 and t<16) else
          FA when (t>16 and t<18) else
          SOL when (t>18 and t<20) else
          LA when (t>20 and t<28) else
          RE when (t>28 and t<36) else
          FA when (t>36 and t<38) else
          SOL when (t>38 and t<40) else
          MI when (t>40 and t<52) else
          SOL when (t>52 and t<60) else
          DO when (t>60 and t<68) else
          FA when (t>68 and t<70) else
          MI when (t>70 and t<72) else
          SOL when (t>72 and t<80) else
          DO when (t>80 and t<88) else
          FA when (t>88 and t<90) else
          MI when (t>90 and t<92) else
          RE when (t>92 and t<104) else
          '0';
```

Ilustración 110. Señal sonido

4.10.3.1. Proceso de t

```
process (clk)
begin
   if (clk'event and clk ='1') then
      t <= t + 1;
   end if;
end process;
```

Ilustración 111. Proceso entidad SONIDO

En la ilustración anterior podemos ver un proceso en el que va sumando una unidad a la señal t cada vez que le entra un flanco de reloj positivo, en la lista de sensibilidad podemos ver que solo tiene una señal, CLK procede de un divisor de frecuencia.

4.10.3.2. DIVISOR: Descripción, declaración e instanciación

Para este componente la descripción y declaración es igual que todas las que ya hemos realizado anteriormente por lo que no la volveremos a escribir, solo nos centraremos en la instanciación.

```
Tiempo : DIVISOR
   generic map (6250000)
   port map (MAX10_CLK1_50,clk);
nota1 : DIVISOR
   generic map (191571)
   port map (MAX10_CLK1_50,DO);
nota2 : DIVISOR
   generic map (170648)
   port map (MAX10_CLK1_50,RE);
nota3 : DIVISOR
   generic map (151975)
   port map (MAX10_CLK1_50,MI);
nota4 : DIVISOR
   generic map (143266)
   port map (MAX10_CLK1_50,FA);
nota5 : DIVISOR
   generic map (127877)
   port map (MAX10_CLK1_50,SOL);
nota6 : DIVISOR
   generic map (113636)
   port map (MAX10_CLK1_50,LA);
nota7 : DIVISOR
   generic map (101420)
   port map (MAX10_CLK1_50,SI);
```

Ilustración 112. Instanciación entidad SONIDO

En todas las instanciaciones que podemos ver en la ilustración nº112 comprobamos que tienen la misma asignación en la entrada del componente debido a que solo se usa la entrada de la entidad MAX10_CLK1_50, cada uno variará su proceso interno debido a que en el generic map cada una tiene el valor que le corresponde a su nota, por último, en la salida asignamos las señales mencionadas al comienzo de la entidad a la nota que le corresponde.

5. CONCLUSIONES

Una vez concluido el proyecto que teníamos planteado para nuestro trabajo de fin de grado, podemos obtener varias conclusiones.

- Se ha conseguido obtener todas las funcionalidades que se pretendían en el reloj, utilizando en gran medida los elementos del hardware del que dispone nuestra placa, para ello se ha recurrido al uso frecuente de distintas máquinas de estado y multiplexores.
- Otra conclusión que he sacado de conclusión es que pese a realizar una entidad por funcionalidad, a la hora de unirlos todos en un bloque principal surgen muchos problemas principalmente por la multiplexación debido a que en el momento que una no esté definida correctamente da muchos problemas en el resto del código.

Como mejoras que se le pueden aplicar a nuestro proyecto, destacaría:

- Aumentar de funcionalidades nuestro reloj, siempre que se pueda aumentar los elementos del hardware se puede conseguir un aumento de aplicaciones.

Bibliografía

[1] EL USO DE FPGAS REPROGRAMABLES EN EL ESPACIO. Disponible: https://m.esa.int/Our_Activities/Space_Engineering_Technology/Microelectronics/The_use_of_reprogrammable_FPGAs_in_space

[2] DE10-LITE User Manual. Disponible: https://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=English&No=1021&FID=a13a2782811152b477e60203d34b1baa

[3] INTEL MAX 10 FPGA DEVICE DATASHEET. Disponible: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/max-10/m10_datasheet.pdf

[4] INTEL MAX 10 FPGA DEVICE OVERVIEW. Disponible: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/max-10/m10_overview.pdf

[5] VHDL: Programming by example. Douglas L. Perry. Editorial McGraw-Hill.

[6] INTEL QUARTUS PRIME STANDARD EDITION USER GUIDE. Disponible: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug-qps-getting-started.pdf

[7] FPGA Design: Best Practices for Team-based Design. Simpson, Philip. Editorial: New York, NY : Springer Science+Business Media, LLC, 2010.