Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

IMPLEMENTACIÓN DE APLICACIONES EN FPGA

Alumno: Raúl Sánchez Narváez

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

Junio, 2018

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Pedro Casanova Peláez , tutor del Proyecto Fin de Carrera titulado: Implementación de aplicaciones en FPGA, que presenta Raúl Sánchez Narváez, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, JUNIO de 2018

El alumno:

Raúl Sánchez Narváez

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: 2018.06.21 19:05:13 +02'00'

Pedro J. Casanova Peláez

A mis padres y a mi hermana,

Por haberme criado y educado para ser lo que soy en la actualidad. Por haber confiado siempre en mí sin un atisbo de duda no solo en los buenos momentos sino en los malos, cuando ni yo confiaba en mis facultades. Por haberme aconsejado siempre y aun así haber respetado cada una de mis decisiones incluso ni cuando tenía la edad para tomarlas. Hoy me convierto en Ingeniero y es principalmente gracias a vosotros.

A mis profesores que de verdad han sentido vocación por su profesión y me han transmitido su pasión por ello, pero especialmente a Óscar y Antonio, mis profesores de Tecnología. Vosotros me formasteis, me mostrasteis la belleza y atracción del camino de la Ingeniería cuando no encontraba mi futuro camino laboral en la adolescencia. Siempre os estaré agradecido.

Y especialmente a mi tutor Pedro Casanova y a mi profesor Luis Miguel Nieto, por estar siempre a mi disposición sin una mala cara o mal gesto ante mis errores, preguntas o muestras de cabezonería que me caracterizan. Este proyecto ha sido un verdadero reto para mí al ser algo diferente a todo lo que había visto y sin vosotros, no sería posible. Muchas gracias.

Raúl, junio 2018

Índice

RESUMEN

Este proyecto se basa en el diseño e implementación de distintas aplicaciones en FPGA enfocadas en el objetivo final de obtención de una calculadora decimal y funcional con las operaciones básicas. Nuestro código irá descrito en lenguaje VHDL

Para ello ha sido necesario evaluar las capacidades y posibilidades de una placa donde estará integrada la FPGA y una evaluación de los distintos recursos de los que disponemos para un diseño con la máxima extensión posible.

Las decisiones finales adoptadas serán las de diseñar e implementar un teclado matricial de 4 columnas por 4 filas y una disposición visual de dos displays de 4 dígitos por 7 segmentos.

Finalmente se mostrará la codificación de la calculadora con sus correspondientes instanciación y funcionalidad de manera explicativa tomando como base el propio código.

ABSTRACT

This Project is based on the design and implementation of various applications in FPGA. The main objective is obtain a decimal calculator with the functionality in basic operations. The descriptive code will be based on VHDL.

Firstly, it will be necessary to make an evaluation of the different capacities and possibilities of our shield, where the FPGA will be integrated. In addition, it will evaluated the resources available for external connections for the purpose of making the design as extensive as possible.

Therefore, the final decisions of this project will be make the design and the implementation of a matrix keyboard with 4 columns by 4 rows. Moreover, a hardware assembly with two displays of 4 digits by 7 segments each one.

Finally, it will be show the descriptive code of the calculator with an explication step by step of all the instances and functionalities of our code.

1. INTRODUCCIÓN Y OBJETIVOS PRINCIPALES

Este TFG se basa en el diseño y obtención de diferentes aplicaciones digitales mediante FPGA. Para este fin se han implementado 3 principales aplicaciones.

- Un teclado matricial con su correspondiente multiplexación en diferentes posibilidades de disposición.
- Un controlador para displays de 7 segmentos óptimamente diseñado analógicamente y con su correspondiente multiplexación.
- Una calculadora funcional de operaciones básicas sobre la que serán integrados las 2 anteriores aplicaciones para completar un proyecto conjunto y unitario.

Los objetivos a alcanzar de este trabajo fin de grado se basan en el diseño, implementación y obtención de una calculadora decimal funcional en la FPGA XC7A35T de la familia Artix desarrollada por Xilinx a nivel tanto de software como de hardware.

Como inicio, se evaluarán las condiciones y capacidades de nuestra shield donde la FPGA está integrada, en este caso la “Basys 3” y se decidirá la disposición adecuada del montaje hardware de nuestra calculadora, la cual será la aplicación en FPGA elegida.

La plataforma para el desarrollo de dicho proyecto será el software Vivado HLx Editions 2017.1 proporcionado por Xilinx de manera gratuita en su página web siguiendo este LINK. El lenguaje de descripción utilizado para poner en funcionamiento esta calculadora será “VHDL”.

Para el desarrollo de este proyecto se entiende que el lector posee unos conocimientos básicos tanto en electrónica, en sus modalidades analógica y digital, así mismo como una base en el lenguaje de codificación “VHDL” por lo que durante el avance del mismo proyecto, se expondrá el código y el diseño de manera explicativa, pero no así a modo de tutorial.

2. CONCEPTOS BÁSICOS.

2.1. FPGA: Breve explicación.

Las FPGA, “Field Programmable Gate Array” son dispositivos que permiten describir distintos circuitos digitales con un lenguaje especificado cuyo código es cargado en el integrado y provoca que se cree “físicamente” en el chip.

Esto es debido a que este chip está formado por bloques lógicos de forma propia en su arquitectura interna que actúan como cables, puertas lógicas, biestables, etc.

Todos ellos preparados para ser unidos entre ellos ya que se encuentran inicialmente sin conexión entre ellos, dispuestos a cualquier unión que se le cargue por el archivo “Bitstream” que se genera tras la correcta descripción de nuestro circuito en el código.

La FPGA se puede reprogramar tantas veces como se desee permitiendo todo tipo de conexión mientras no se exceda el número de elementos que tiene internos nuestro chip, los cuales dependen de la FPGA en cuestión pero por lo general son de un número muy elevado que permita diseños de gran complejidad con considerables enlaces.

Lo más importante a destacar en este caso es que a la hora de diseñar nuestro proyecto, en una FPGA no se efectúa lenguaje de programación, aunque el término esté permitido, sino lenguaje de descripción, los más utilizados son:

- VHDL
- Verilog.

2.2. Principales beneficios.

Como acabamos de comentar, una FPGA tendrá una gran capacidad de aceptar circuitos con conexionado de todo tipo con gran cantidad de enlaces entre ellos pero principalmente sus beneficios serán:

- Rendimiento: La potencia de las FPGA supera en gran medida a la de los procesadores digitales de señales, al permitir el control de las diferentes entradas y salidas a nivel de hardware se consigue obtener respuestas mucho más rápidas
- Actualización: Esta tecnología está tomando una gran importancia y está obteniendo una creciente rapidez en el desarrollo de diferentes prototipos y productos, eso unido a que la cantidad de desarrolladores a nivel amateur está creciendo de manera exponencial, permite general un gran interés esta plataforma.
- Fiabilidad: Esta se podría catalogar como el beneficio más importante y aclamado de todos, los circuitos FPGA son implementaciones seguras de la ejecución de un programa. Un sistema basado en procesadores frecuentemente implica varios niveles de abstracción a la hora del desarrollo de distintas tareas. Estas tareas son administradas por recursos hardware que gestionan la memoria y el ancho de banda por un procesador, sin embargo este procesador solo puede ejecutar una instrucción a la vez, lo cual provoca un riesgo a posibles obstrucciones o detenciones. Esto no ocurre en las FPGA, los cuales no necesitan de un sistema operativo, ejecutándose de forma paralela ya que poseen un hardware únicamente dedicado a la ejecución de cada tarea.
- Mantenimiento: debido a su capacidad reprogramable, permite las continuas modificaciones y mejoras funcionales sin la necesidad de un rediseño hardware o en la placa.

2.3. Principales aplicaciones en la actualidad.

Actualmente y tal como acabamos de comentar, las FPGA están tomando una creciente importancia en nuestra sociedad hasta el punto de estar incluidos en aplicaciones varias de campos muy variados de la actualidad.

- Sistemas de visión artificial: Ejemplos de estas aplicaciones puede ser cámaras de video vigilancia, robots, etc. Estas aplicaciones requieres de sistemas para conocer la posición así como los objetos de su entorno o cualidades como el reconocimiento de rostros. Esto requiere el manejo de una gran cantidad de imágenes y una necesidad de tratados de filtros sobre dichas imágenes la mayoría en tiempo real, así como una gran fiabilidad en sus funciones por lo que la FPGA es un interesante elemento para este campo.
- Sistemas de imágenes médicas: Las FPGA cada vez son más utilizadas en la obtención y el adecuamiento de imágenes biomédicas obtenidas mediante procesos PET, escáner CT, rayos X, imágenes tridimensionales, etc. Estos sistemas de visión médica requieres de una gran resolución y capacidad de procesado e incluso en algunos casos el desarrollo en tiempo real. Las FPGA serán una gran opción a este campo debido a su capacidad de procesamiento paralelo.
- Codificación y encriptación: La seguridad en el envío de mensajes está tomando una importancia fundamental en la actualidad, a la hora de recibir un email o los datos electrónicos que podamos aportar a la hora de comprar en internet, sino incluso también en ámbito militar, aeronáutico y gubernamental donde el espionaje es una realidad. Por tanto una encriptación eficiente y segura se vuelve algo esencial y posible por FPGA gracias a su amplio terreno de capacidad a la hora de gestionar información y realizar operaciones aritméticas en el encriptado de la misma.
- Aeronáutica y defensa: Existen gran multitud de aplicaciones en estos campos donde las FPGA son utilizadas de manera asidua gracias a sus buenas características como la fiabilidad de sus componentes.

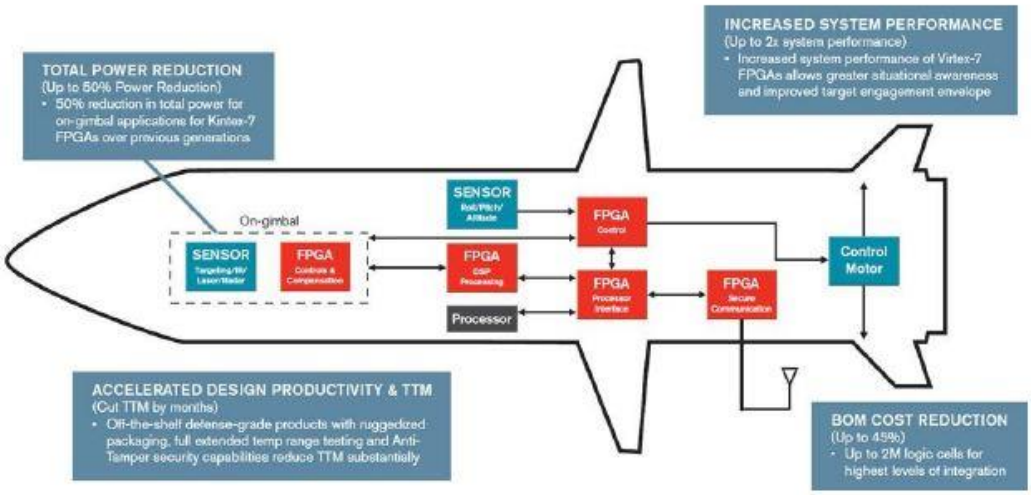

Ilustración 2.1

De hecho la aeronáutica es uno de los principales campos donde las FPGA reciben gran interés y uso debido a su gran capacidad de restricción ante posibles errores aportando una inmensa fiabilidad al control de la nave.

No solo a nivel aeronáutico, sino incluso aeroespacial donde ha estado integrada en diferentes misiones espaciales tripuladas o incluso en el propio Rover de marte.

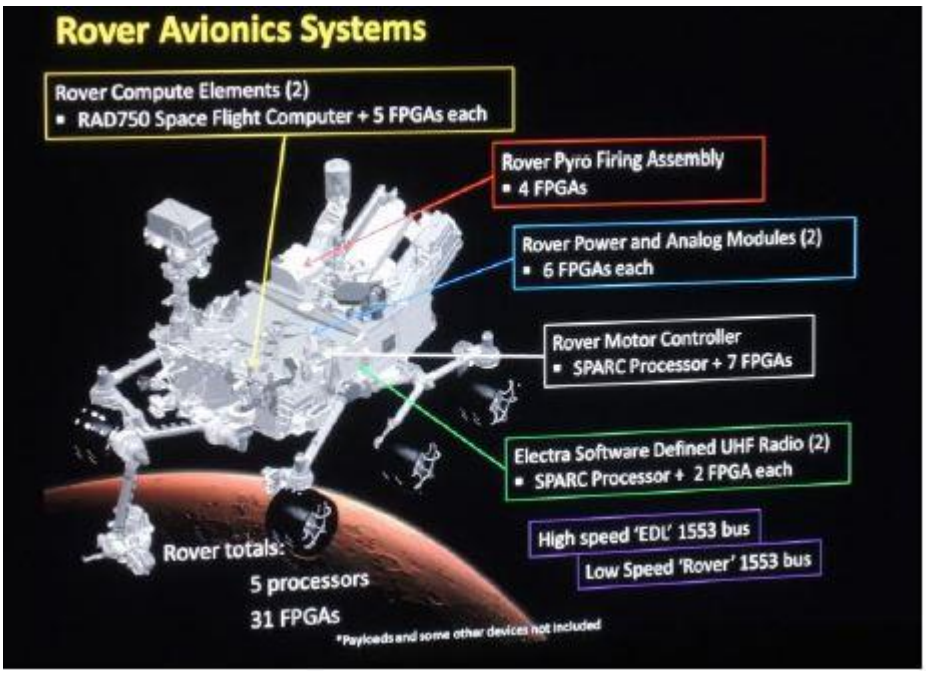

Ilustración 2.2

3. CONEXIONADO HARDWARE.

3.1. Análisis de elementos de la placa.

Como en cualquier diseño electrónico donde nuestro microcontrolador o microprocesador está integrado en una shield, nuestro primer objetivo será el análisis de los elementos de los que disponemos verificando si son suficientes y/o necesarios para el proyecto que se desea llevar a cabo, o si por consiguiente, necesitaremos dotarlo de unos elementos externos que acompañen a nuestra placa para poder llegar a nuestro fin deseado, en este caso el diseño y funcionamiento de una calculadora aritmética básica.

Nuestra FPGA, en este caso la “XC7A35T” de la familia Artix-7 desarrollada por Xilinx estará integrada en la shield “Basys 3”.

Ilustración 3.1

La FPGA Artix-7 está optimizado para la lógica de alto rendimiento y ofrece más capacidad, mayor rendimiento y más recursos que los diseños anteriores. Las características de Artix-7 35T incluyen:

- 33,280 celdas lógicas en 5200 rebanada (cada rebanada contiene cuatro LUT, posibles tablas de verdad, de 6 entradas y 8 flip-flops)
- 1,800 Kbits de RAM.
- Cinco fichas de gestión de reloj posibles para gestionar.
- 90 DSP, procesador digital de señales.
- Posibles velocidades de reloj interno superiores a 450 MHz.
- Conversor Analógico-Digital (XADC).

Así mismo, nuestra shield “Basys3” tiene incluidos estos distintos puertos y periféricos:

- 16 conmutadores.
- 16 LEDs activos a nivel alto.
- Display de 4 dígitos y 7 segmentos.
- Puerto VGA para comunicación con monitor de 12 bits por pixel.
- 1 puerto USB-JTAG.
- 3 bancos de conexión de 8 bits para elementos externos (PMOD).
- Un puente de transferencia USB-UART.
- 1 puerto USB HID Host para ratones, teclados y memorias flash.
- 5 pulsadores activos a nivel alto.
- 1 banco exclusivo de conexión para el CAD.

De estos diferentes puertos y periféricos, podremos analizar si varios de ellos podrían tener la funcionalidad necesaria que deseamos, de este modo podríamos ahorrar elementos externos que configurar para lograr nuestro diseño.

3.1.1. Display de 4 dígitos y 7 segmentos.

Como acabamos de observar, en nuestra placa, tenemos un display cuya colocación en la shield será la siguiente.

Ilustración 3.2

Gran parte de diseños ya elaborados sobre diferentes tipos de calculadora, son normalmente efectuados sobre este tipo de display, interno y con pocos valores ya que basta con efectuar los diferentes tipos de operaciones y mostrarlos en pantalla.

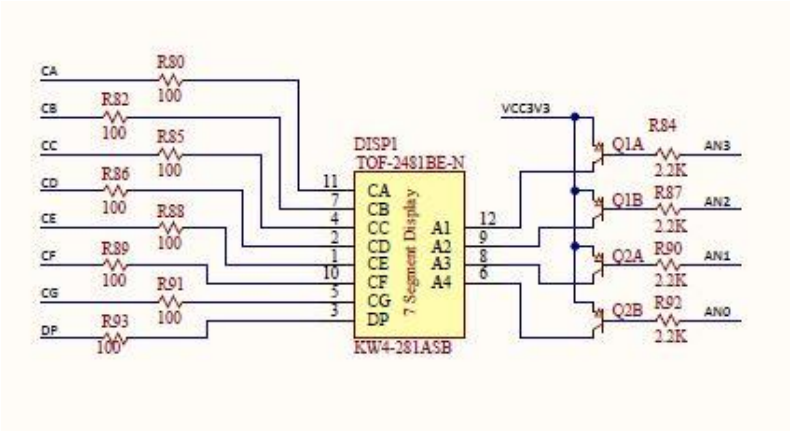

Ilustración 3.3

Se puede observar por su tipo de configuración con transistores PNP los cuales reciben el voltaje directamente desde VCC, que estamos ante una disposición de display de ánodo común , los cuales tienen las consiguiente resistencias limitadoras para evitar excesos de corriente por los LEDs.

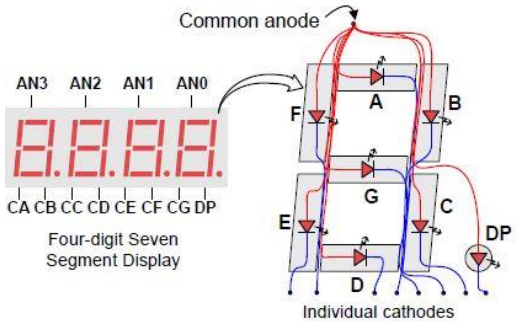

Ilustración 3.4

En este tipo de diseño, donde los ánodos están conectados de forma común y los cátodos están de forma individual, debemos tener en consideración de que los diferentes leds de cada segmento lucirán a nivel bajo, o lo que es lo mismo, siempre que reciban un “0”.

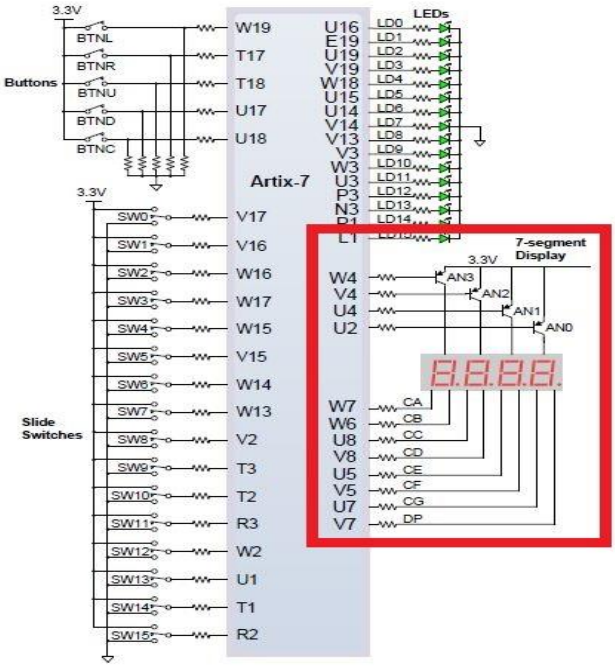

Ilustración 3.5

Se puede observar también, cuál será la colocación de cada uno de ellos a la hora de configurarlos para su uso en el fichero “Constraints”. A los cuales se asignarán cada ánodo (dígito) o cada segmento.

El problema de este tipo de display, es la poca visibilidad que ofrece, al tratarse de únicamente 4 dígitos, solo permite mostrar en pantalla escasamente las decenas de los valores, no permitiendo así mostrar ni la operación o el signo igual antes de implementar el resultado en los diferentes displays.

Este problema se soluciona refrescando la pantalla cada vez que se introduce la operación para así solo mostrar los valores que estamos introduciendo, siguiendo por tanto el proceso.

1. Introducir valor (solo hasta 4 dígitos).
2. Pulsar operación (refresco de display).
3. Introducir nuevo valor (de nuevo únicamente hasta 4 dígitos).
4. Pulsar igual (refresco de display).
5. Muestra del resultado (solo hasta 4 dígitos).

Este tipo de características, restringe considerablemente nuestro ratio de trabajo viéndonos constantemente censurados a operaciones que lleguen como máximo a la unidad de millar o a que nuestro resultado se vea “cortado” si este número excede esta unidad.

Por lo que debido a esto descartaremos la posibilidad de utilizar este display interno. Esto nos llevará a diseñar y configurar de manera adecuada unos displays externos con su correspondiente circuito que asegure su funcionamiento óptimo.

Esto además nos dará la posibilidad de gestionar el tamaño y configuración del mismo.

3.1.2.Pulsadores y conmutadores.

Tal y como hemos observado en el listado de los diferentes puertos y periféricos, de la placa, en la “Basys 3” dispondremos de una serie de pulsadores y conmutadores que nos puedan servir a la hora de generar nuestros valores de cálculo o nuestra operación a realizar.

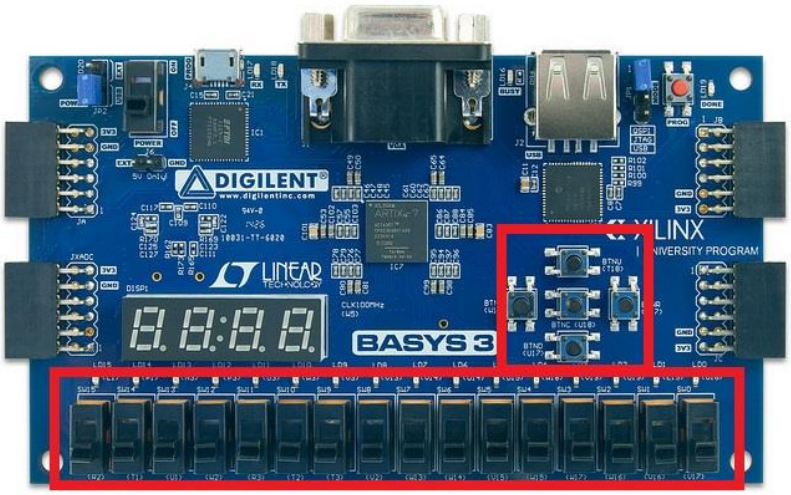

Ilustración 3.6

El planteamiento para el uso de estos elementos en nuestra calculadora podría basarse en que los 5 botones marcaran la operación a realizar con los botones ubicados en los 4 distintos puntos cardinales mientras que el botón central podría indicar que deseamos que nos muestre el resultado.

Además como observaremos en el siguiente esquema que muestra su configuración de conexión. Estos botones estarán planteados como activos a nivel alto.

Algo importante ya que en este tipo de shield suelen ir a nivel bajo, además, no se aprecia de forma aparente que en ningún caso tengan alguna especie de sistema antirrebote por lo que habría que tenerlo en cuenta posteriormente a la hora de diseñar el código de descripción.

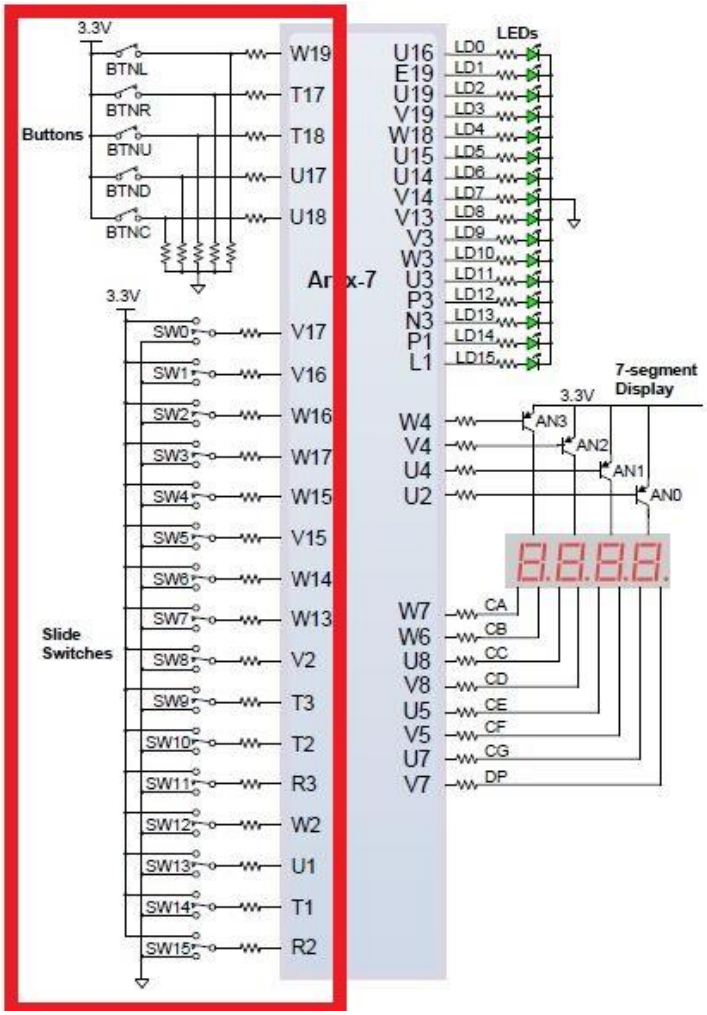

Ilustración 3.7

En cambio los 16 distintos conmutadores nos permitirían elegir en código binario para los dos distintos valores, sin embargo , debido a que la muestra de números no puede superar el valor del “9999” al no poder tener más de 4 cifras. Esto nos censura en gran medida, ya que podría llegar al valor de

$$2^{16} - 1 = 65.535$$

Esto nos hará plantearnos que tampoco será idóneo usar este tipo de configuración, a parte que para la selección del valor, la persona, tiene que tener un conocimiento de la numeración binaria, en este tipo de valores ocasiona que si se quiere colocar un valor muy alto, se tenga que recurrir a cualquier calculadora decimal-binaria para poder encontrar el valor necesario a colocar con los distintos conmutadores, haciendo el proceso bastante molesto y nada dinámico.

3.2. Diseño y proceso de configuración externa.

Tras analizar los distintos elementos que podríamos haber utilizado de la placa y haberlos descartado por los diferentes problemas que podrían haber causado, hemos llegado a la situación donde deberemos decidir el modo más adecuado de diseñar y colocar las diferentes partes del circuito general que decidamos ubicar.

3.2.1.Bancos para conexión de elementos externos

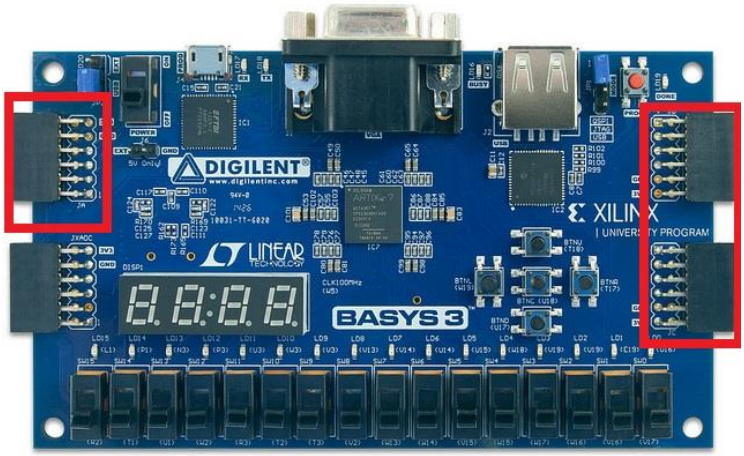

Ilustración 3.8

Este será nuestro primer paso a efectuar, para saber qué elementos vamos a integrar en nuestro circuito, primero tendremos que saber cuántos podríamos incluir en los bancos que tenemos disponibles para conectar a la placa.

Por desgracia, solo dispondremos de tres de los cuatro ubicados en la shield, ya que uno de ellos será únicamente para posibles conexiones con un Convertidor Analógico-Digital (CAD).

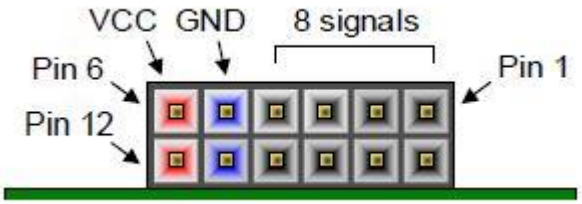

Ilustración 3.9

Como podemos apreciar, además, cada uno de los bancos, inicialmente con 12 diferentes conexiones, solo podrán ser utilizados 8 de ellos al estar 4 de los puntos de conexión guardados para únicamente conexiones a VCC o a la tierra en cuestión.

Esto nos limita bastante las conexiones pero intentaremos adecuarlo todo de manera correcta. Como añadido, únicamente decir que estas serán los conexionados adecuados para cada banco.

Pmod JA	Pmod JB	Pmod JC	Pmod XDAC
JA1: J1	JB1: A14	JC1: K17	JXADC1: J3
JA2: L2	JB2: A16	JC2: M18	JXADC2: L3
JA3: J2	JB3: B15	JC3: N17	JXADC3: M2
JA4: G2	JB4: B16	JC4: P18	JXADC4: N2
JA7: H1	JB7: A15	JC7: L17	JXADC7: K3
JA8: K2	JB8: A17	JC8: M19	JXADC8: M3
JA9: H2	JB9: C15	JC9: P17	JXADC9: M1
JA10: G3	JB10: C16	JC10: R18	JXADC10: N1

Tabla 3.1

Será muy importante destacar, así mismo que estos bancos de conexionado tienen una característica especial.

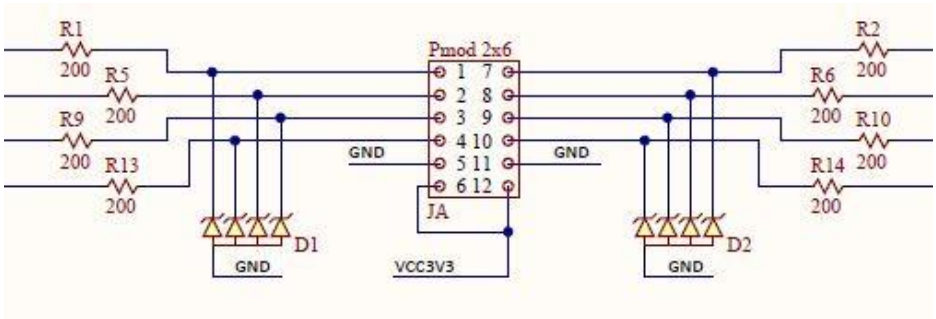

Ilustración 3.10

Y es que como observamos en este ejemplo del PMOD “JA” todas sus conexiones tienen integrada una resistencia de 200 Ω con la finalidad de aportar seguridad y alargar la vida útil de cualquier elemento que se conecte a ellos.

3.2.2. Teclado Matricial.

Aunque inicialmente planteamos la utilización de los diferentes conmutadores y pulsadores integrados en la placa, finalmente la opción más adecuada será la del diseño, conexionado y codificación para la lectura de los números necesarios de operación a partir de un teclado matricial de forma externa a través del último de los bancos (JA).

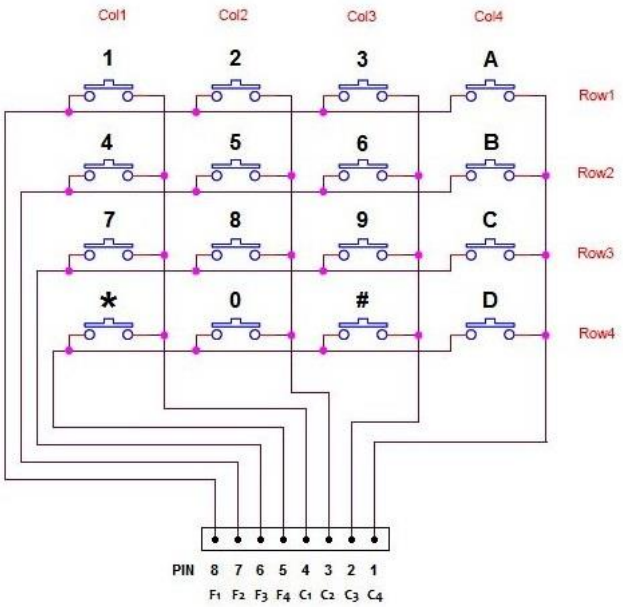

Ilustración 3.11

Como se puede apreciar, la configuración es idónea teniendo como entradas únicamente las 4 filas y las 4 columnas que irían a los 8 pines que quedarían libre en este “JA” , sin necesidad de añadir cualquier voltaje o tierra por la manera de configuración ni tampoco resistencia al tenerla ya integrada el propio banco.

La única diferencia entre este teclado y el utilizado finalmente serán cambios en cuanto a asignaciones.

A -> Suma “+”	D -> División “÷”
B-> Resta “-”	# -> Igualdad “=”
C-> Multiplicación “×”	* -> Limpieza “C”

Tabla 3.2

En las diferentes tiendas y demás distribuidores se podrán encontrar infinidad de teclados diseñados y preparados para este fin de la misma forma como “Teclado Matricial 4x4”, Sin embargo el utilizado en este caso se ha completado con una carcasa y teclas realizadas mediante impresión 3D.

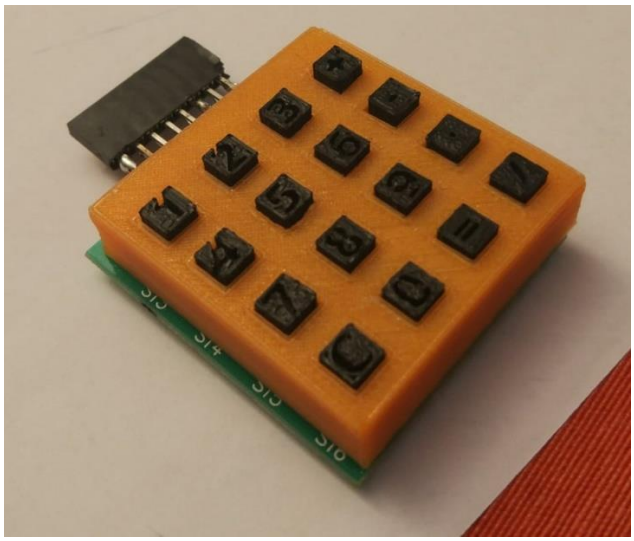

Ilustración 3.13

3.2.3. Display 8 Dígitos-7 Segmentos.

Tras analizar y finalmente descartar el uso del display interno de nuestra “Basys 3”, nos dispondremos a diseñar de una manera adecuada el uso y diseño de nuestros displays con la finalidad de que muestren tanto nuestros operandos, operaciones y resultados.

Para este display de 8 dígitos, elegiremos el uso de dos display de 4 dígitos de la marca “Kingbright”, en este caso el modelo “CC56-12SURKWA”.

Ilustración 3.14

Este display, de alrededor de 5 cm de largo se considerara favorablemente su diseño más robusto y mayor tamaño a la hora de mostrar los diferentes valores que hayamos introducido por teclado.

De este display, a parte de su tamaño y su correspondiente patillaje para cada segmento y dígitos, es también muy importante destacar que corresponderá a una configuración de cátodo común, el cual difiere del display interno que teníamos cuya configuración se basaba en el ánodo común.

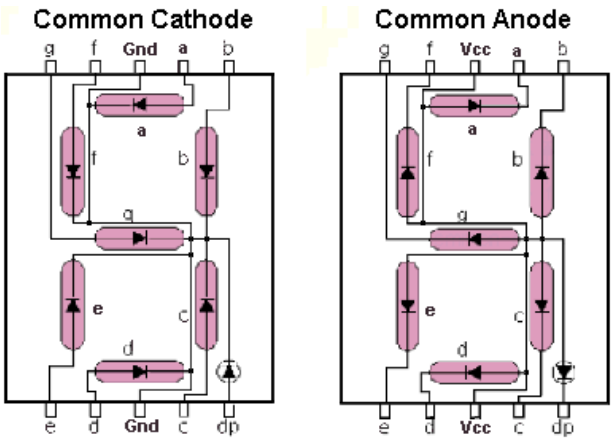

Ilustración 3.15

A diferencia del ejemplo anterior, esta vez, los leds de cada segmento estarán orientados de forma contraria y por tanto esta vez serán los cátodos los que estén formando una unión única mientras que los ánodos finalizan de forma habitual. Esto, aparentemente sin repercusión, genera un cambio de diseño, ya que esta vez los segmentos se activarán a nivel alto, por tanto cuando reciban un “1” y para ello deben cambiar las intensidades y la configuración del circuito.

En la “**Ilustración 3.5”** podemos apreciar como el display interno de ánodo común estaba comandado en la gestión de sus dígitos por transistores de tipo PNP cuyos colectores estaban recibiendo un voltaje constante desde VCC. Lo cual permitía que al asignar un “0” a la salida del cátodo del led cuyo segmento queríamos encender, la corriente fluyera y el led en cuestión se encendiera. Esto cambia con el cátodo común, como apreciaremos a continuación.

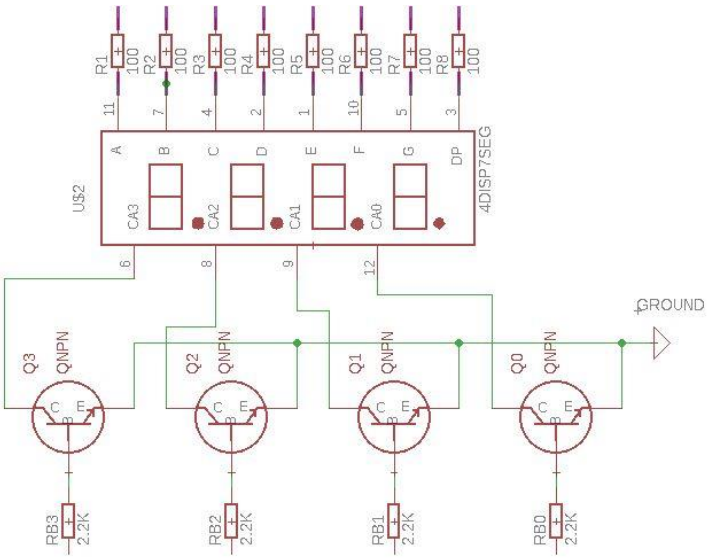

Ilustración 3.16

Como observamos, en cátodo común, esta vez el “1” se asignaría a la salida del ánodo individual de cada led, esto generará que debamos buscar que la corriente fluye desde el ánodo al cátodo, de esta forma deberemos cambiar el transistor a un NPN y cuyos emisores deban estar conectados a la tierra permitiendo este paso de corriente.

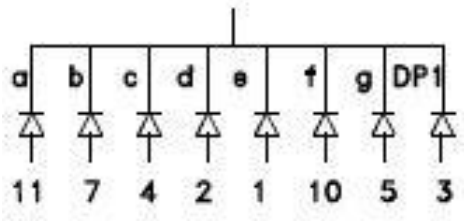

Ilustración 3.17

Aquí podemos apreciar, como en esta configuración de 4 dígitos, necesitaríamos asignar 8 puertos para los 7 segmentos y el punto respecto del patillaje del display, tal y como se aprecia en la imagen anterior, e igualmente 4 para los dígitos pertenecientes por las cuatro patillas restantes. Ante esto observamos que ocupamos el banco JA en su plenitud(a excepción de los VCC y los GND).

En cambio del campo JB, podremos conectar 4 más, lo que nos permite en este caso ubicar un nuevo display con la misma conexión, uniendo simplemente la salida de los diferentes segmentos quedando de la siguiente forma.

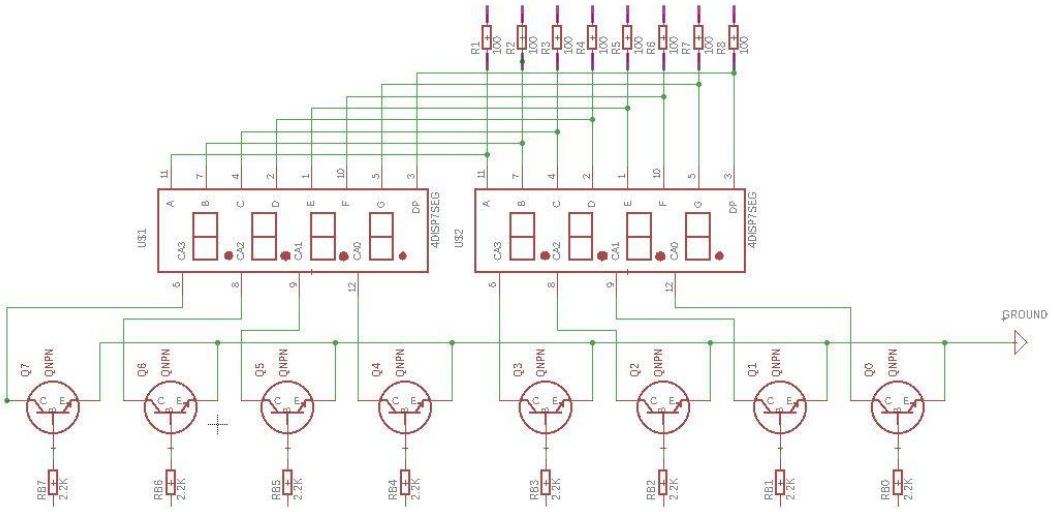

Ilustración 3.18

Esto finalmente ocuparía todos puertos disponibles de JB conectando igualmente todos los emisores a una de las conexiones a “Ground” de dicho banco, finalizando de esta forma con el diseño de la configuración Hardware de los displays, permitiendo de esta forma operar hasta con 8 dígitos llegando al “99.999.999” .

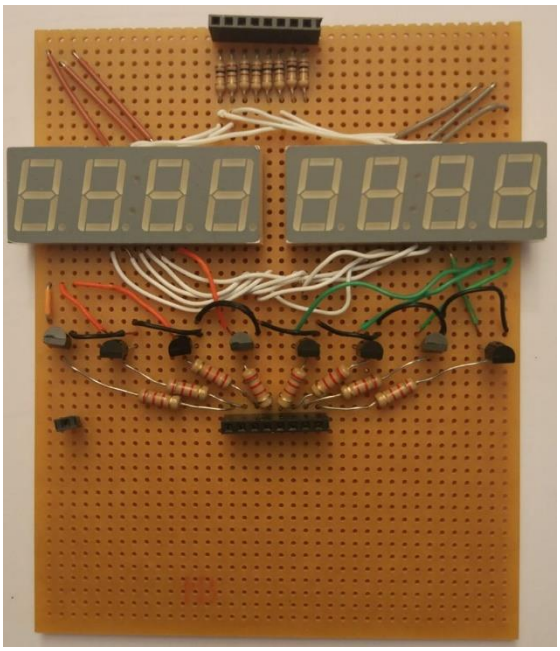

Ilustración 3.19

4. CODIFICACIÓN E IMPLEMENTACIÓN.

Tras los finalizar el diseño del circuito a utilizar para el correcto funcionamiento de una calculadora funcional. El objetivo inicial se basará en el correcto conexionado de enlace entre nuestro diseño y la shield correspondiente, en nuestro caso, la "Basys 3" de Xilinx.

4.1. Configuración Software de elementos externos.

Como ya hemos tomado en cuenta a la hora del diseño Hardware, esta configuración se efectuará a través de los distintos bancos de conexionado (PMOD) existente en nuestra placa, los cuales a partir de un diseño adecuado deberían ser ocupados en su plenitud. Toda esta configuración pertenecerá al archivo Constraints.

4.1.1. Constraints iniciales.

Debido a la forma de conexionado necesario si la implementación Software se efectúa mediante el programa "VIVADO", se deberá de codificar un fichero especial Constraits el cual especificará los pines de la FPGA que se utilizarán para las entradas y salidas codificadas en nuestro programa.

Este fichero Constraits llevará unas primeras líneas de código básicas para su funcionamiento correcto codificando entre otras partes la asignación de voltaje por los distintos módulos.

```
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]

set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]

set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]
```

Tabla 4.1

4.1.2.Constraints del reloj.

```
## Clock signal
## 100 MHz
set_property -dict [list PACKAGE_PIN W5 IOSTANDARD LVCMOS33] [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]
```

Tabla 4.2

Así mismo, será necesario inicializar el reloj interno del que dispone nuestra FPGA siendo esta su configuración idónea adaptándola para una frecuencia de 100MHz.

4.1.3.Constraints del Teclado.

```
#Pmod Header JA
#Sch name = JA1
set_property -dict [list PACKAGE_PIN J1 IOSTANDARD LVCMOS33 PULLUP true] [get_ports COL[0]]
#Sch name = JA2
set_property -dict [list PACKAGE_PIN L2 IOSTANDARD LVCMOS33 PULLUP true] [get_ports COL[1]]
#Sch name = JA3
set_property -dict [list PACKAGE_PIN J2 IOSTANDARD LVCMOS33 PULLUP true] [get_ports COL[2]]
#Sch name = JA4
set_property -dict [list PACKAGE_PIN G2 IOSTANDARD LVCMOS33 PULLUP true] [get_ports COL[3]]
#Sch name = JA7
set_property -dict [list PACKAGE_PIN H1 IOSTANDARD LVCMOS33] [get_ports FIL[0]]
#Sch name = JA8
set_property -dict [list PACKAGE_PIN K2 IOSTANDARD LVCMOS33] [get_ports FIL[1]]
#Sch name = JA9
set_property -dict [list PACKAGE_PIN H2 IOSTANDARD LVCMOS33] [get_ports FIL[2]]
#Sch name = JA10
set_property -dict [list PACKAGE_PIN G3 IOSTANDARD LVCMOS33] [get_ports FIL[3]]
```

Tabla 4.3

Tal y como habíamos analizado anteriormente, el teclado irá ubicado en el PMOD “JA” al cual se le asignan los diferentes pines a las 4 columnas y las 4 filas, reservando además los pines “JA5” y “JA11” para las conexiones a tierra y los “JA6” y “JA12” para las conexiones a VCC tal y como observamos en la **Ilustración 3.9.**

Estas diferentes localizaciones del puerto, irán luego enlazadas a sus correspondientes pines además de proporcionar información de que las conexiones se tratarán de entradas o salidas estándar “IOSTANDARD” así como asignarles que se encuentran en nivel alto , estos recibirán un voltaje de 3.3 V. Así mismo, con la idea de asegurar una correcta lectura del teclado, las 4 columnas, serán configuradas con una resistencia interna PULLUP la cual evitará posibles errores de funcionamiento al leer alguna tecla pulsada, o si se pulsan dos a la vez por error.

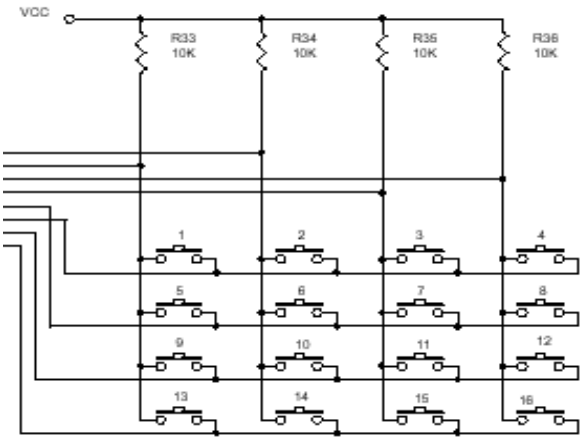

Ilustración 4.1

4.1.4.Constraints del Display.

```
##Pmod Header JB
##Sch name = a
set_property -dict [list PACKAGE_PIN A14 IOSTANDARD LVCMOS33] [get_ports seg[0]]
##Sch name = b
set_property -dict [list PACKAGE_PIN A16 IOSTANDARD LVCMOS33] [get_ports seg[1]]
##Sch name = c
set_property -dict [list PACKAGE_PIN B15 IOSTANDARD LVCMOS33] [get_ports seg[2]]
##Sch name = d
set_property -dict [list PACKAGE_PIN B16 IOSTANDARD LVCMOS33] [get_ports seg[3]]
##Sch name = e
set_property -dict [list PACKAGE_PIN A15 IOSTANDARD LVCMOS33] [get_ports seg[4]]
##Sch name = f
set_property -dict [list PACKAGE_PIN A17 IOSTANDARD LVCMOS33] [get_ports seg[5]]
##Sch name = g
set_property -dict [list PACKAGE_PIN C15 IOSTANDARD LVCMOS33] [get_ports seg[6]]
##Sch name = h
set_property -dict [list PACKAGE_PIN C16 IOSTANDARD LVCMOS33] [get_ports seg[7]]
```

```
#Pmod Header JC
#catodos
#Sch name = dig1
set_property -dict [list PACKAGE_PIN K17 IOSTANDARD LVCMOS33] [get_ports cat[7]]
#Sch name = dig2
set_property -dict [list PACKAGE_PIN M18 IOSTANDARD LVCMOS33] [get_ports cat[6]]
#Sch name = dig3
set_property -dict [list PACKAGE_PIN N17 IOSTANDARD LVCMOS33] [get_ports cat[5]]
#Sch name = dig4
set_property -dict [list PACKAGE_PIN P18 IOSTANDARD LVCMOS33] [get_ports cat[4]]
#Sch name = dig1-2
set_property -dict [list PACKAGE_PIN L17 IOSTANDARD LVCMOS33] [get_ports cat[3]]
#Sch name = dig2-2
set_property -dict [list PACKAGE_PIN M19 IOSTANDARD LVCMOS33] [get_ports cat[2]]
#Sch name = dig3-2
set_property -dict [list PACKAGE_PIN P17 IOSTANDARD LVCMOS33] [get_ports cat[1]]
#Sch name = Jdig4-2
set_property -dict [list PACKAGE_PIN R18 IOSTANDARD LVCMOS33] [get_ports cat[0]]
```

Tabla 4.4

Como se puede apreciar la configuración adecuada de nuestro display estará dividida en dos apartados perfectamente diferenciados igualmente por los dos bancos de conexión diferenciados del lado derecho de nuestra shield como se puede ver en la **Ilustración 3.8**.

- Inicialmente, en la **Tabla 4.3** asociamos los pines del PMOD “JB” a los distintos segmentos de nuestro display, en este caso los 7 segmentos y el punto. Igualmente reservamos los pines sucesivos para VCC y GROUND tal y como había hecho en el caso anterior del teclado. Estos segmentos y el punto irán interconectados entre todos los displays del circuito, los cuales en este caso son 2. Ya que a través de estos nos aseguraremos simplemente de su adecuado funcionamiento con el encendido o apagado de cada uno de los segmentos según cada situación, mientras que los cátodos serán los que seleccionen cada dígito a mostrar.
- Finalmente, en la **Tabla 4.4** asociamos los pines del PMOD “JC” para conectar a 0 V cada uno de los cátodos comunes de cada dígito perteneciente al display, en este caso 8 dígitos al incluir dos displays de 4 dígitos cada uno. De este modo por cada pin gestionaremos cada dígito indicando si debe estar encendido o no.

4.1.5.Constraints de los LEDs.

```
## LEDs

set_property -dict [list PACKAGE_PIN U16 IOSTANDARD LVCMOS33] [get_ports led[0]]
set_property -dict [list PACKAGE_PIN E19 IOSTANDARD LVCMOS33] [get_ports led[1]]
set_property -dict [list PACKAGE_PIN U19 IOSTANDARD LVCMOS33] [get_ports led[2]]
set_property -dict [list PACKAGE_PIN V19 IOSTANDARD LVCMOS33] [get_ports led[3]]
set_property -dict [list PACKAGE_PIN W18 IOSTANDARD LVCMOS33] [get_ports led[4]]
set_property -dict [list PACKAGE_PIN U15 IOSTANDARD LVCMOS33] [get_ports led[5]]
set_property -dict [list PACKAGE_PIN U14 IOSTANDARD LVCMOS33] [get_ports led[6]]
set_property -dict [list PACKAGE_PIN V14 IOSTANDARD LVCMOS33] [get_ports led[7]]
set_property -dict [list PACKAGE_PIN V13 IOSTANDARD LVCMOS33] [get_ports led[8]]
set_property -dict [list PACKAGE_PIN V3 IOSTANDARD LVCMOS33] [get_ports led[9]]
set_property -dict [list PACKAGE_PIN W3 IOSTANDARD LVCMOS33] [get_ports led[10]]
set_property -dict [list PACKAGE_PIN U3 IOSTANDARD LVCMOS33] [get_ports led[11]]
set_property -dict [list PACKAGE_PIN P3 IOSTANDARD LVCMOS33] [get_ports led[12]]
set_property -dict [list PACKAGE_PIN N3 IOSTANDARD LVCMOS33] [get_ports led[13]]
set_property -dict [list PACKAGE_PIN P1 IOSTANDARD LVCMOS33] [get_ports led[14]]
set_property -dict [list PACKAGE_PIN L1 IOSTANDARD LVCMOS33] [get_ports led[15]]
```

Tabla 4.5

Para mejorar la visibilidad de nuestro proyecto, incluiremos los LEDs internos de la placa en nuestro diseño, siendo posible ubicarlos en la **Ilustración 4.2.** De este modo nos permitirá mostrar a través de ellos distintos valores a nivel binario, como el valor introducido por el teclado, el resultado como valor binario o incluso el estado en el que nos encontremos.

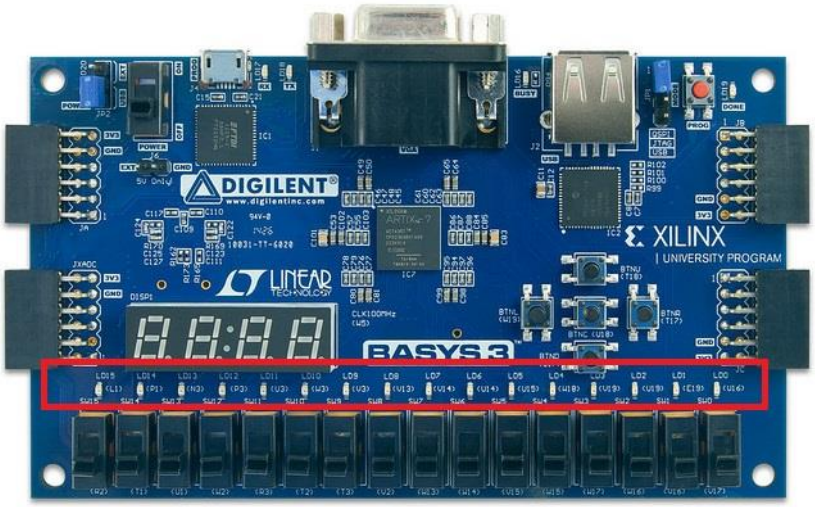

Ilustración 4.2

4.2. Librerías.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.all;
use IEEE.NUMERIC_STD.all;
use IEEE.STD_LOGIC_arith.all;
```

Tabla 4.6

Para un correcto funcionamiento y diseño Software de cualquier aplicación FPGA, es casi tan importante las distintas implementaciones durante el código descriptivo, como la selección de las distintas librerías. Los cuales permitirán ampliar las funcionalidades y capacidades de nuestra FPGA.

Esto es posible debido a que VHDL es un lenguaje muy restrictivo en sí, y para ampliar sus capacidades, las distintas empresas han diseñado ciertas librerías para mejorar sus posibilidades.

Inicialmente el código “library IEEE” será necesario ser descrito para poder acceder a la lista de librerías disponibles para luego ser usadas por la palabra reservada “use”. Estas serán las distintas librerías usadas.

- **IEEE.STD_LOGIC_1164**: será necesaria para poder soportar el tipo de dato “STD_LOGIC”.
- **IEEE.STD_LOGIC_unsigned**: nos indicará que cualquier vector con distintos valores binarios irán sin signo por lo que no habría que discriminar el valor menos significativo de ellos para dicho fin.
- **IEEE.NUMERIC_STD:** nos permitirá el uso de valores de tipo entero para cualquier operación aunque para eso tendremos que declarar el tipo de cada valor de forma adecuada o convertir un valor de un tipo a otro.
- **IEE.STD_LOGIC_ARITH:** nos permitirá el uso de las funciones aritméticas básicas (suma, resta, multiplicación y división), además será necesaria para la conversión entre los distintos tipos de valores.

Finalmente todas estas librerías deben ir acabadas por el término reservado “.ALL”.

4.3. Descripción de la entidad (puertos, señales y componentes).

4.3.1. Descripción de los puertos.

```
entity Calculadora is
    Port(
        led : out STD_LOGIC_VECTOR (15 downto 0);
        COL : in STD_LOGIC_VECTOR (3 downto 0);
        FIL : out STD_LOGIC_VECTOR (3 downto 0);
        cat : out STD_LOGIC_VECTOR (7 downto 0);
        seg : out STD_LOGIC_VECTOR (7 downto 0);
        clk: in STD_LOGIC

        );
end Calculadora;
```

Tabla 4.7

El primer paso, se basará en la declaración de la entidad global de nuestro proyecto, en este caso la Calculadora, representando de esta forma su interfaz general a partir de la palabra reservada “Port” para enlazar con el exterior los siguientes elementos ya configurados en los Constraints.

- “led”, SALIDA: se tratará de un vector de tipo binario de 16 valores, tantos como los LEDs ya configurados.
- COL, ENTRADA: vector de tipo binario con 4 valores, tantos como columnas posee el teclado diseñado.
- FIL, SALIDA: vector de tipo binario de 4 valores, tantos como filas posee el teclado diseñado.
- cat, SALIDA: vector de tipo binario de 8 valores, tantos como dígitos posee nuestro diseño, en este caso, dos displays de 4 dígitos cada uno.
- seg, SALIDA: vector de tipo binario de 8 valores, tantos como segmentos posea cada dígito de nuestros displays.
- clk, Entrada: valor de tipo binario, nos indicará el estado lógico de nuestro reloj; en nuestro caso de 100 MHz.

4.3.2.Descripción de las señales.

```
architecture Behavioral of Calculadora is

signal clk1kHz : STD_LOGIC;

signal tecla : STD_LOGIC_VECTOR (3 downto 0) :="0000";
signal pulsada : STD_LOGIC :='0';

signal d0 : STD_LOGIC_VECTOR (4 downto 0) :="10000";
signal d1 : STD_LOGIC_VECTOR (4 downto 0) :="10000";
signal d2 : STD_LOGIC_VECTOR (4 downto 0) :="10000";
signal d3 : STD_LOGIC_VECTOR (4 downto 0) :="10000";
signal d4 : STD_LOGIC_VECTOR (4 downto 0) :="10000";
signal d5 : STD_LOGIC_VECTOR (4 downto 0) :="10000";
signal d6 : STD_LOGIC_VECTOR (4 downto 0) :="10000";
signal d7 : STD_LOGIC_VECTOR (4 downto 0) :="10000";
```

Tabla 4.8

```
signal acumulador:STD_LOGIC_VECTOR(26 downto 0):=(others => '0');
signal numero:STD_LOGIC_VECTOR(26 downto 0):=(others => '0');
signal sumatoria:STD_LOGIC_VECTOR(26 downto 0):=(others => '0');
signal auxiliar:STD_LOGIC_VECTOR(26 downto 0):=(others => '0');
signal resultado:STD_LOGIC_VECTOR(26 downto 0):=(others => '0');
signal resultadobcd:STD_LOGIC_VECTOR(31 downto 0):=(others => '0');

signal cociente:STD_LOGIC_VECTOR(26 downto 0):=(others => '0');
signal multi:STD_LOGIC_VECTOR(26 downto 0):=(others => '0');
signal suma:STD_LOGIC_VECTOR(26 downto 0):=(others => '0');
signal resta:STD_LOGIC_VECTOR(26 downto 0):=(others => '0');

signal operacion: STD_LOGIC_VECTOR (1 downto 0):="00"; --
signal estado: STD_LOGIC_VECTOR (1 downto 0):="00"; --
```

Tabla 4.9

Tras la declaración de la entidad, será necesaria diseñar la funcionalidad de la arquitectura interna de dicha entidad. Así mismo, si en la declaración de la entidad, teníamos que hacer el enlace entre la entidad con el mundo exterior o distintos elementos de la shield donde está integrada la FPGA, en la declaración de la arquitectura será necesaria la descripción de cómo se interconectarán los elementos en la propia FPGA.

Para asegurar el funcionamiento, por lo general, será preferible trabajar en todo momento con señales “signal” y tras hacer las distintas acciones sobre ellas, enlazarlas ya directamente con las entradas o salidas de la entidad. De este modo evitamos el cambio sucesivo de las entradas y salidas por valores “intermedios” del proceso descriptivo de nuestro proyecto, siendo únicamente asignado al final del mismo.

La descripción de las distintas señales será explicada en los sucesivos apartados en los que sean usados.

Para un diseño más adecuado se procurará un diseño jerarquizado de nuestro proyecto, para esto se diseñará nuestro “Prescaler”, “Teclado” y “Displays” como componentes de nuestra entidad. Esto nos permitirá separarlo y de la misma forma volver a ser utilizado directamente en cualquier otra entidad sin necesidad de reiterar las mismas líneas de código.

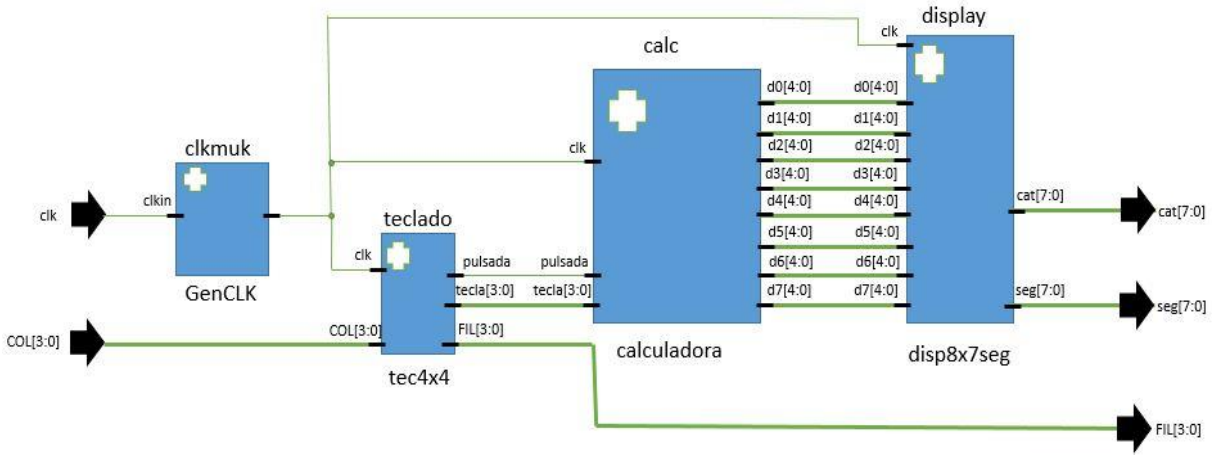

Ilustración 4.3

Por lo tanto nuestro diagrama de bloques, quedaría de una manera similar de la que podemos comprobar en este diagrama de bloques. Del cual analizaremos los componentes anteriormente indicados, siendo la calculadora la que irá definida en la zona “begin” de nuestra entidad general.

4.3.3. Descripción del componente "Prescaler".

Para comenzar, es necesario saber que un Prescaler es la división de una frecuencia de reloj por una serie de ciclos a determinar para una adecuada preparación de la señal de reloj para el fin que sea requerido.

Ilustración 4.4

De esta forma un componente no es más que la forma de diseño estructural en la que se puede dividir la arquitectura de la entidad, por tanto, estará incluido dentro del diseño de la funcionalidad de la arquitectura justo después de la declaración de las distintas señales.

```
component GenCLK
    Port (
    clkin: in STD_LOGIC;
    clkout: out STD_LOGIC
    );
end component;
```

Tabla 4.10

De igual manera que hemos hecho con la entidad, enlazaremos mediante "Port" nuestro componente con el exterior, el cual tendremos que instanciar interiormente con los distintos elementos que deseemos, en este caso al ser un componente, podremos enlazarlo con señales de la arquitectura.

4.3.4.Descripción del componente "Teclado".

Tras declarar nuestro reloj donde seleccionaremos su velocidad para un periodo de valor más viable a la hora de leer un valor introducido por nuestro teclado. Se dispondrá la declaración de dicho teclado cuya configuración será el siguiente paso a establecer.

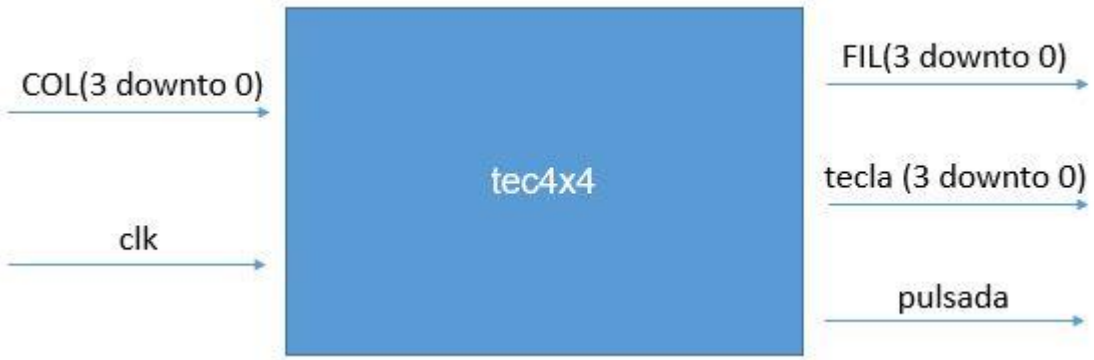

Ilustración 4.5

Tal y como hemos decidido en el anterior paso, el del Prescaler, este tipo de configuraciones los podemos representar de una manera óptima como componentes de nuestra entidad.

Esta configuración tendrá que ser efectuada de igual manera que el caso anterior, siendo necesario su enlace con el exterior mediante la palabra reservada "port", así como su correspondiente instanciación.

```
component tec4x4
    port (
    COL : in STD_LOGIC_VECTOR (3 downto 0);
    FIL : out STD_LOGIC_VECTOR (3 downto 0);
    tecla : out STD_LOGIC_VECTOR (3 downto 0);
    pulsada : out STD_LOGIC;
    clk: in STD_LOGIC
    );
end component;
```

Tabla 4.11

4.3.5.Descripción del componente "Displays"

Finalmente, será necesario igualmente la configuración adecuada del último componente con sus correspondientes declaraciones e instanciaciones, este será el de nuestros displays diseñados en el **punto** **3.2.3.**

Ilustración 4.6

```
component disp8x7seg
    port (
    cat : out STD_LOGIC_VECTOR (7 downto 0);
    seg : out STD_LOGIC_VECTOR (7 downto 0);
    d0 : in STD_LOGIC_VECTOR (4 downto 0);
    d1 : in STD_LOGIC_VECTOR (4 downto 0);
    d2 : in STD_LOGIC_VECTOR (4 downto 0);
    d3 : in STD_LOGIC_VECTOR (4 downto 0);
    d4 : in STD_LOGIC_VECTOR (4 downto 0);
    d5 : in STD_LOGIC_VECTOR (4 downto 0);
    d6 : in STD_LOGIC_VECTOR (4 downto 0);
    d7 : in STD_LOGIC_VECTOR (4 downto 0);
    clk: in STD_LOGIC
    );
end component;
```

Tabla 4.12

4.4. Funcionalidad del “Prescaler”.

```
entity GenCLK is
  Port (
       clkin: in STD_LOGIC;
       clkout: out STD_LOGIC
 );
end GenCLK;

architecture Behavioral of GenCLK is
signal clkdiv : STD_LOGIC;
```

Tabla 4.13

Ya de forma interna en el componente, este deberá de ser configurado como si de una entidad propia se tratara, por ello se enlazará de la misma forma con los elementos externos, tal y como acabamos de evaluar.

Posteriormente en la instanciación de la arquitectura de nuestra entidad, podremos definir nuestras señales, en este caso:

- clkdiv: señal de tipo lógico que nos servirá de valor intermedio que definirá nuestra señal a la salida tras convertir nuestra señal de entrada.

```
begin
-- divide 100MHz clock to 1kHz
div1kHz: process(clkin)
variable intCnt: integer range 0 to 50000-1;
begin
    if rising_edge(clkin) then
        if intCnt=50000-1 then
            intCnt:=0;
            clkdiv<=not clkdiv;
        else
            intCnt:=intCnt+1;
        end if;
    end if;
end process div1kHz;

clkout<=clkdiv;

end Behavioral;
```

Tabla 4.14

Tras la correcta definición y enlace de nuestros determinados elementos, podremos avanzar a la definición de la funcionalidad de nuestra entidad interna del componente.

Para ello incluiremos nuestra función en un “process” debido a que esta arquitectura, da la opción de diseño de circuitos secuenciales, ya que el principal problema de las FPGA es que su diseño es completamente combinacional, lo cual obliga a una capacidad de abstracción considerable al suceder todo de manera paralela. El “process” nos permite el control del avance de nuestro código de forma secuencial dentro de este diseño combinacional, lo cual por tanto nos será realmente útil. Además deberemos de destacar que este “process” solo será utilizado cuando los valores de su **lista de sensibilidad** cambien, en este caso “clkin”.

Tal y como sucede en otros tipos de arquitectura, esto nos permitirá el uso de señales internas, lo cual en un “process”, serán declaradas como “variables”, valores que solo existirán en el interior del “process”. En este caso:

- intCont: variable de tipo entero en un rango de 0 a 50.000, que llevará el contaje de los ciclos transcurridos de señal de reloj.

Tras esto, pasaremos a la definición de nuestra funcionalidad. Debido a que nuestro process será utilizado siempre que “clkin” cambie, esto nos genera el problema de que será utilizado tanto en el cambio de 0->1 como de 1->0, lo cual provocara no el cambio con cada periodo, sino con el semiperiodo.

Esto se soluciona con nuestro “if rising_edge” condición que evalúa únicamente el flanco ascendente de nuestro clkin. Además tal y como se ha apreciado, evaluaremos por tanto el periodo de nuestro, reloj, y necesitaremos adecuar por tanto nuestra señal de entrada, a la de salida deseada.

Nuestro reloj de entrada posee una señal de entrada de f=100MHz lo cual nos da un T=10 ns. Mientras nuestra señal seleccionada a la salida será de f=1kHz lo cual es un T=1 ms.

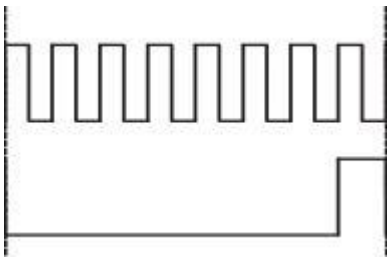

Ilustración 4.7

Será necesario hallar cuantos ciclos debemos de contar de flanco ascendente a la entrada para poder emitir un cambio de señal a la salida tal y como se puede observar en el ejemplo de Prescaler de la **Ilustración 4.3**.

$$intCnt = \frac{Tout}{Tin} = \frac{0{,}001}{0{,}00000001} = 50.000$$

Fórmula 4.1

Por tanto por cada cambio de señal de reloj a la entrada, evaluaremos si el contador ha llegado a su valor 49.999 (contando el 0), si así es, se cambiará el valor lógico a la salida del Prescaler, pero si no es así, se sumará un ciclo más al contaje hasta llegar a nuestro valor final.

Finalmente nuestra señal, será asignada como hemos indicado a nuestra salida.

4.5. Funcionalidad del “Teclado”.

```
entity tec4x4 is
  Port (
        COL : in STD_LOGIC_VECTOR (3 downto 0);
        FIL : out STD_LOGIC_VECTOR (3 downto 0);
        tecla : out STD_LOGIC_VECTOR (3 downto 0);
        pulsada : out STD_LOGIC;
        clk: in STD_LOGIC
   );
end tec4x4;

architecture Behavioral of tec4x4 is

signal soltada : STD_LOGIC :='1';
signal puls : STD_LOGIC :='0';
signal fila : integer range 0 to 3 :=0;
```

Tabla 4.15

```
multiplexacion_teclado:process(clk)
variable modo : STD_LOGIC :='1';     -- 0 => hexadecimal, 1 => calculadora
begin
    if rising_edge(clk) then
        if puls='1' then
            puls<='0';
            pulsada<='1';
        end if;
        if soltada='1' then
            if COL="1110" or COL="1101" or COL="1011" or COL="0111" then
                puls<='1';
                soltada<='0';
                if modo='0' then
                    if COL="1110" then
                        case fila is
                        when 3 =>
                            tecla<="1100";
                        when 2 =>
                            tecla<="1000";
                        when 1 =>
                            tecla<="0100";
                        when others=>
                            tecla<="0000";
                        end case;
```

```
76                  elsif COL="1101" then
77                      case fila is
78                      when 3 =>
79                          tecla<="1101";
80                      when 2 =>
81                          tecla<="1001";
82                      when 1 =>
83                          tecla<="0101";
84                      when others=>
85                          tecla<="0001";
86                      end case;
87                  elsif COL="1011" then
88                      case fila is
89                      when 3 =>
90                          tecla<="1110";
91                      when 2 =>
92                          tecla<="1010";
93                      when 1 =>
94                          tecla<="0110";
95                      when others=>
96                          tecla<="0010";
97                      end case;
98                  elsif COL="0111" then
99                      case fila is
100                     when 3 =>
101                         tecla<="1111";
102                     when 2 =>
103                         tecla<="1011";
104                     when 1 =>
105                         tecla<="0111";
106                     when others=>
107                         tecla<="0011";
108                     end case;
109                 end if;
110             else
111                 if COL="1110" then
112                     case fila is
113                     when 3 =>
114                         tecla<="1111";-- RETROCESO "C"
115                     when 2 =>
116                         tecla<="0111";-- 7
117                     when 1 =>
118                         tecla<="0100";-- 4
119                     when others=>
120                         tecla<="0001";-- 1
121                     end case;
```

```
                    elsif COL="1101" then
                        case fila is
                        when 3 =>
                            tecla<="0000";-- 0
                        when 2 =>
                            tecla<="1000";-- 8
                        when 1 =>
                            tecla<="0101";-- 5
                        when others=>
                            tecla<="0010";-- 2
                        end case;
                    elsif COL="1011" then
                        case fila is
                        when 3 =>
                            tecla<="1110";--IGUAL "="
                        when 2 =>
                            tecla<="1001";-- 9
                        when 1 =>
                            tecla<="0110";-- 6
                        when others=>
                            tecla<="0011";-- 3
                        end case;
                    elsif COL="0111" then
                        case fila is
                        when 3 =>
                            tecla<="1101";--DIVISIÓN "/"
                        when 2 =>
                            tecla<="1100";--MULTIPLICACION "X"
                        when 1 =>
                            tecla<="1011";--RESTA"-"
                        when others=>
                            tecla<="1010"; --SUMA"+"
                        end case;
                    end if;
                end if;
            end if;
        end if;
        if COL="1111" then
            soltada<='1';
            puls<='0';
            pulsada<='0';
            case fila is
            when 0 =>
                 fila<=1;
                 FIL<="ZZ0Z";
```

```
167             when 1=>
168                  fila<=2;
169                  FIL<="ZOZZ";
170             when 2 =>
171                  fila<=3;
172                  FIL<="OZZZ";
173             when others=>
174                  fila<=0;
175                  FIL<="ZZZO";
176             end case;
177         end if;
178     end if;
179 end process multiplexacion_teclado;
180
181 end Behavioral;
```

Tabla 4.16

La multiplexación de nuestro teclado es una de las partes primordiales de nuestro proyecto, ya que la lectura de los valores introducidos por teclado será la base de todo y si no hacemos una correcta gestión de ella, esta provocara errores a la hora de los distintos cálculos con nuestros diferentes operando.

Inicialmente declararemos las señales del componente:

- puls: valor lógico que nos indicará si alguna de las teclas ha sido pulsada, inicialmente parece tener la misma funcionalidad que pulsada, pero es un recurso necesario para aportar otro ciclo de reloj para que pulsada sea renovada de forma óptima a la par que se renueva tecla.
- soltada: valor lógico que nos indicará si la pieza ha sido pulsada, esta señal nos visibilizará de forma óptima el hecho de que una pieza sea pulsada y posteriormente soltada, evitando de esta forma estar evaluando constantemente pulsada o pulsada negada.
- fila: valor entero de 4 valores, nos servirá para rotar su valor de forma manual y gestionada por nosotros mismos, evitando así actuar constantemente sobre la salida FIL.

Como ya hemos comentado anteriormente, para evitar la ejecución simultánea de todo el circuito, incluiremos todo acto que necesite ser secuencial dentro de un “Process”.

En nuestro process cuyos elementos externos ya hemos configurado, únicamente tendremos una variable interna, denominada **"modo"**, la cual será únicamente un valor lógico que te indicará el tipo de disposición que pueda tener el teclado en función de las necesidades particulares. En nuestro caso utilizaremos la disposición que podemos apreciar en la **Ilustración 3.11.**

En nuestro código descriptivo iniciaremos de nuevo nuestro process con la condición de la **línea 55** de que nuestra señal de reloj reciba un flanco ascendente tal como hemos explicado anteriormente para que se ejecute tras cada periodo. Posteriormente, nuestra función se dividirá en tres condiciones ubicadas en las **líneas 56, 60 y 159.** Estas condiciones serán las que siempre que recibamos el flanco ascendente sean evaluadas:

- Condición línea 56: Nos evaluará la señal "puls" si cualquiera de nuestras teclas ha sido pulsada en el paso anterior se entrará en dicha condición donde se activará la salida pulsada, dando así un ciclo de reloj a su funcionamiento y evitando posibles problemas.
- Condición línea 60: Evaluará la situación de soltada, ya que puede ser precedida de la situación soltada, o de la que haya estado pulsada cualquiera de las columnas. **En el interior de esta condición estará la lectura de cada tecla** en función de la columna seleccionada y que evaluaremos más adelante**.**
- Condición línea 159: Evaluará **si ninguna de nuestras columna** ha sido activada, lo que indicará que no se ha pulsado ninguno de los botones de nuestro teclado, en cuyo caso, se activará soltada, se desactivará pulsada y puls y continuaremos rotando las filas a la espera de la llegada de una pulsación actuando igualmente sobre las distintas filas declarándolas como de **alta impedancia "Z"** con la finalidad de evitar cualquier posible pulsación doble.

Por lo que ya se habrá descrito de esta forma la funcionalidad cuando el teclado no reciba ninguna pulsación. Ahora nos introduciremos en la condición de la línea 60 para **la evaluación de la situación ante cualquier pulsación**.

Tras introducirnos en esta condición al estar activa soltada, evaluaremos si alguna de nuestras columnas ha sido pulsada en el ciclo anterior y por tanto renovado su valor en este ciclo. De esta forma descartaremos la situación que esté soltada porque ya estuviera anterior soltada.

Este tipo se evaluación se basará en que la columna seleccionada, deba encontrarse **activada a nivel bajo** debido a nuestro circuito con resistencias pullup que hemos diseñado anteriormente para nuestro teclado.

Tras esta verificación, sabremos que debemos leer cual de nuestras teclas ha sido la seleccionada. Inicialmente y tras activar la señal “puls” marcando de esta forma que se ha pulsado alguna de las teclas, y además renovar la señal “soltada”, tendremos que evaluar en qué tipo de modo de teclado nos encontramos, siendo el de calculadora, en nuestro caso (modo=1).

Esto conllevará dirigirnos a la **línea 110** directamente para posteriormente evaluar de forma independiente según la columna activada, cual es la fila por la que tendremos libre de la alta impedancia y guardamos el dato de cual será si tecla correspondiente como se puede apreciar en los comentarios de cada elección. De esta forma lograremos la lectura adecuada de una tecla introducida por el teclado.

4.6. Funcionalidad de los “Displays”.

```
entity disp8x7seg is
    Port(
    cat : out STD_LOGIC_VECTOR (7 downto 0);
    seg : out STD_LOGIC_VECTOR (7 downto 0);
    d0 : in STD_LOGIC_VECTOR (4 downto 0);
    d1 : in STD_LOGIC_VECTOR (4 downto 0);
    d2 : in STD_LOGIC_VECTOR (4 downto 0);
    d3 : in STD_LOGIC_VECTOR (4 downto 0);
    d4 : in STD_LOGIC_VECTOR (4 downto 0);
    d5 : in STD_LOGIC_VECTOR (4 downto 0);
    d6 : in STD_LOGIC_VECTOR (4 downto 0);
    d7 : in STD_LOGIC_VECTOR (4 downto 0);
    clk: in STD_LOGIC
    );
end disp8x7seg;

architecture Behavioral of disp8x7seg is

signal digito : STD_LOGIC_VECTOR (4 downto 0);
signal catodos : STD_LOGIC_VECTOR (7 downto 0);
```

Tabla 4.17

```
59  multiplexacion_catodo:process(clk)
60  begin
61      if rising_edge(clk) then
62          case catodos is
63          when "00000001"=>
64              catodos<="00000010";
65          when "00000010"=>
66              catodos<="00000100";
67          when "00000100"=>
68              catodos<="00001000";
69          when "00001000"=>
70              catodos<="00010000";
71          when "00010000"=>
72              catodos<="00100000";
73          when "00100000"=>
74              catodos<="01000000";
75          when "01000000"=>
76              catodos<="10000000";
77          when others=>
78              catodos<="00000001";
79          end case;
80      end if;
81  end process multiplexacion_catodo;
82
83  cat<=catodos;
```

```
85  -- Colocación de los digitos en los displays
86  with catodos select digito<=
87      d7 when "10000000",
88      d6 when "01000000",
89      d5 when "00100000",
90      d4 when "00010000",
91      d3 when "00001000",
92      d2 when "00000100",
93      d1 when "00000010",
94      d0 when others;
95  -- Conversión de los digitos a 7 segmentos
96  with digito select seg<=
97      "00111111" when "00000",   --0
98      "00000110" when "00001",   --1
99      "01011011" when "00010",   --2
100     "01001111" when "00011",   --3
101     "01100110" when "00100",   --4
102     "01101101" when "00101",   --5
103     "01111101" when "00110",   --6
104     "00000111" when "00111",   --7
105     "01111111" when "01000",   --8
106     "01101111" when "01001",   --9
107     "00000000" when "01010",       -- (+)' '
108     "01000000" when "01011",       -- (-)'-'
109     "10000000" when "01100",       -- (X)'.'
110     "00000000" when "01101",       -- (/)' '
111     "01001000" when "01110",       -- (=)'calculo'
112 --    "01110001" when "01111",       -- (C)'limpieza'
113 -- Los próximos cuatro podrían ser implementados para más operaciones
114 --    "01110111" when "10000",       -- (A)'A'
115 --    "01111100" when "10001",       -- (B)'b'
116 --    "00111001" when "10010",       -- (C)'C'
117 --    "01011110" when "10011",       -- (D)'d'
118     "00000000" when others;        -- ' '
119
120
121 end Behavioral;
```

Tabla 4.18

Finalmente será esencial igualmente la multiplexación de nuestros displays, cuya finalidad será mostrar los valores que hemos ido operando en nuestra calculadora, una mala programación de dicha multiplexación conllevaría un falso muestreo y posibles problemas a la hora de detectar errores en el código descriptivo.

Inicialmente declararemos las señales de nuestro componente tras haber ya analizado las diferentes conexiones port:

- catodos: valor lógico de 8 valores, se ceñirá a la rotación de los 8 dígitos que compone nuestro diseño de display, de esta forma y dependiendo de cada flanco ascendente del reloj, hará que siempre se evalúen los 8 dígitos rotando su activación secuencial al nivel alto al ser de cátodo común.
- digito: valor lógico de 5 valores, su finalidad será recibir el valor a mostrar en cada uno de nuestros dígitos de cátodo común, a pesar de solo utilizar 16 valores, se implementará con la posibilidad de 32 valores con la finalidad de acomodar un diseño con más de 16 valores introducidos por teclado.

En este componente del display no será necesario incluir todo nuestro código en un process al no ser necesario un diseño secuencial en todo nuestro código, sino únicamente en la rotación de cátodos debido a que tanto el dígito como el seg, serán simple asignaciones que pueden efectuar de forma paralela sin problema ya que no se operará sobre ellos, sino que serán la simple representación de estos.

Nuestro código descriptivo se basará en tres partes claramente definidas.

- Process de multiplexación de los cátodos: En este y gracias a un case que se ejecutará en cada flanco ascendente de reloj, podremos ir haciendo un recorrido por todos nuestros dígitos del display para lograr de esta forma que todos estos sean evaluados en su valor. Habrá que tomar en consideración que al ser **activos a nivel alto debido a ser de cátodo común** esta será su correcta activación, si nuestro display fuera de ánodo común, su activación rotacional deberá ser a nivel bajo.
- Tabla de verdad de colocación de los dígitos: En el código VHDL, un “with-select” puede asemejarse de forma adecuada a la funcionalidad que puede tener una tabla de verdad. Debido a que por tanto será una asignación combinacional, esta NUNCA podrá ir dentro de un process, siendo detectado como un error si así se efectúa. En esta tabla se asignará cada dígito con su señal dX correspondiente, de esta forma

podremos llevar el muestreo adecuado de cada valor en su disposición deseada. Aquí solo se asignará su valor a nivel binario con cada posición del display, pero hay que recordar que la representación de dicho valor se efectúa con la siguiente tabla de verdad.

- Tabla de verdad de conversión de los dígitos a 7 segmentos: Igual que en el anterior caso, esta asignación será combinacional y se basará en función del valor introducido en cada dígito en valor binario, para luego convertirlo en la activación o desactivación de cada segmento con la siguiente disposición. Siendo el valor más significativo el “A” y el menos significativo el punto “Punto decimal”.

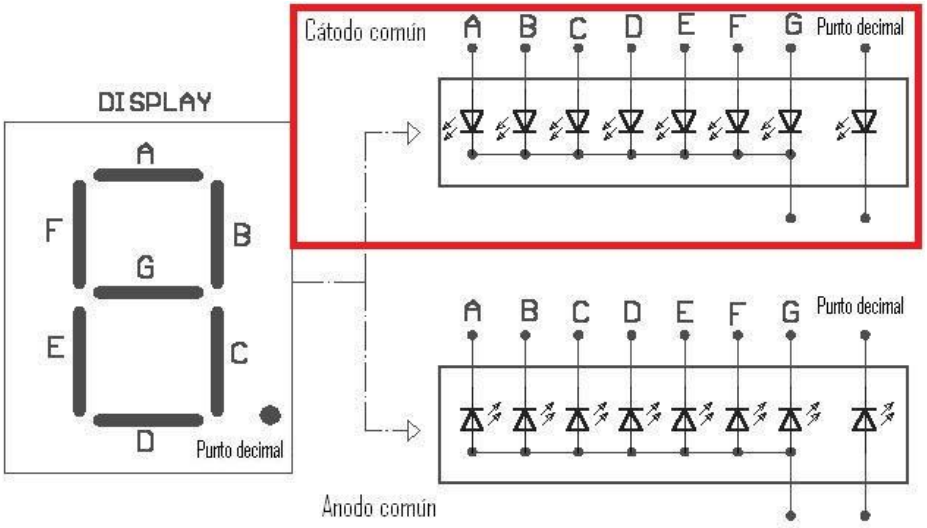

Ilustración 4.8

Como ya hemos comentado y se puede apreciar a partir de la línea de código 113, nuestro código está preparado para poder añadir otras 16 asignaciones más si así se desearan, siendo nuestro diseño fácilmente ampliable con la selección de otro teclado de distinta disposición.

4.7. Funcionalidad completa de la calculadora.

4.7.1. Instanciación "Prescaler".

```
begin
clkmux : GenCLK
    port map (
        clkin=>clk,
        clkout=>clk1kHz
        );
```

Tabla 4.19

Como podemos apreciar, nuestra instanciación del componente, deberá efectuarse tras el "begin" de la arquitectura, esto es debido de que usaremos señales pertenecientes a la entidad, el cual solo puede utilizarse en la definición de la funcionalidad de la entidad, razón por la que la instanciación de nuestro componente se haya después del "begin". Obviamente si se hiciera antes, la instanciación de una instanciación, generaría un error o una mala declaración en nuestro proyecto.

Así mismo en nuestro "port map" podemos observar.

- clkin se asigna a clk: Los dos elementos son de tipo lógico tal cual observamos en la **Tabla 4.10** y **Tabla 4.13**, se tratará del valor lógico de nuestra señal de reloj a la ENTRADA del Prescaler, en nuestro caso 100 MHz.
- clkout se asigna a clk1kHz: los dos elementos son de tipo lógico tal cual observamos en la **Tabla 4.10** y **Tabla 4.8,** se tratará del valor lógico de nuestra señal de reloj a la SALIDA del Prescaler, el cual prepararemos para 1 kHz.

4.7.2. Instanciación "Teclado".

```
teclado : tec4x4
    port map (
        COL(3 downto 0)=>COL(3 downto 0),
        FIL(3 downto 0)=>FIL(3 downto 0),
        tecla(3 downto 0)=>tecla(3 downto 0),
        pulsada=>pulsada,
        clk=>clk1kHz
        );
```

Tabla 4.20

Nuestro port map, estará definido por:

- COL se asigna a COL: los dos son vectores de tipo binario con 4 valores, tal cual observamos en la **Tabla 4.15** y **Tabla 4.11**, tantos como columnas posee el teclado diseñado. Puede parecer redundante pero es un paso necesario, al declararlos en el mismo nombre tanto en el Port de nuestra entidad general, como en el de nuestro componente.
- FIL se asigna a FIL: los dos son vectores de tipo binario con 4 valores, tal cual observamos en la **Tabla 4.15** y **Tabla 4.11**, tantos como filas posee el teclado diseñado.
- tecla se asigna a tecla: las dos son vectores de tipo binario con 4 valores, tal cual observamos en la **Tabla 4.15** y **Tabla 4.8**, permitiendo por tanto una lectura de hasta 16 valores distintos por teclado, este vector podría ampliarse si nuestro teclado fuera de mayores dimensiones y tuviera más botones.
- pulsada se asigna a pulsada: las dos son de tipo lógico, tal cual observamos en la **Tabla 4.15** y **Tabla 4.8**. Este dato, nos indicará cuando se ha hecho una nueva pulsación, evitando posibles errores como por ejemplo dos pulsaciones continuadas a la misma tecla del teclado, sin esta variable solo sería detectada la primera pulsación y al no detectar cambio, no reconocería la siguiente.
- clk se asigna a clk1kHz: las dos son de tipo lógico, tal cual observamos en la **Tabla 4.15** y **Tabla 4.8**. Asignando la frecuencia de reloj a la salida del Prescaler.

4.7.3. Instanciación "Display".

```
display : disp8x7seg
    port map (
        cat(7 downto 0)=>cat(7 downto 0),
        seg(7 downto 0)=>seg(7 downto 0),
        d0(4 downto 0)=>d0(4 downto 0),
        d1(4 downto 0)=>d1(4 downto 0),
        d2(4 downto 0)=>d2(4 downto 0),
        d3(4 downto 0)=>d3(4 downto 0),
        d4(4 downto 0)=>d4(4 downto 0),
        d5(4 downto 0)=>d5(4 downto 0),
        d6(4 downto 0)=>d6(4 downto 0),
        d7(4 downto 0)=>d7(4 downto 0),
        clk=>clk1kHz
        );
```

Tabla 4.21

Nuestro port map, estará definido por:

- cat se asigna a cat: los dos son vectores de tipo binario con 8 valores, tal cual observamos en la **Tabla 4.18** y **Tabla 4.7**, dígitos de cátodo común tiene nuestro circuito.
- seg se asigna a seg: los dos son vectores de tipo binario con 8 valores, tal cual observamos en la **Tabla 4.18** y **Tabla 4.7**, tantos como segmentos tiene cada dígito de nuestro display representando el valor introducido.
- dX se asigna a dX: los dos son vectores de tipo binario de 5 valores, han sido diseñados de este modo para permitir la entrada por ellos de hasta 32 valores distintos aunque inicialmente solo serían necesarios 16 por nuestro tipo de teclado (10 números más las 6 operaciones) pero diseñarlo de esta forma muestra la posibilidad de añadir muchas más operaciones con un teclado óptimo con más teclas.
- clk se asigna a clk1kHz: las dos son de tipo lógico, tal cual observamos en la **Tabla 4.14** y **Tabla 4.8**. Asignando la frecuencia de reloj a la salida del Prescaler.

4.7.4.Funcionalidad "Lectura por teclado".

Tras el correcto análisis y descripción de los distintos componentes que forman parte de nuestro proyecto, pasaremos a la codificación adecuada de nuestra calculadora, único bloque que nos falta por evaluar de nuestra **Ilustración 4.3**. Para ello, dividiremos nuestra calculadora en diferentes apartados de funcionalidades para hacer más simple la comprensión de nuestro código en función de la finalidad de cada apartado. Siendo el primero la lectura de los valores introducidos por el teclado.

4.7.4.1. Avance.

```
145 avance: process (pulsada)
146 variable carga,numtecla,numtemp: integer range 0 to 99999999;
147 begin
148 if tecla<="1001" then     -- si la tecla pulsada está entre 0 y 9
149     numtemp:= CONV_INTEGER(numero);
150     numtecla := CONV_INTEGER(tecla);
151     carga:=numtemp*10+numtecla;
152     sumatoria<= CONV_STD_LOGIC_VECTOR(carga,27);
153 else
154     sumatoria<=(others => '0');
155 end if;
156
157 end process avance;
```

Tabla 4.22

Se basará en no solo la representación de los valores entrantes en los displays. Sino igualmente asegurar el valor interno en nuestro proyecto, que llevará el valor que se está introduciendo actualmente, reciba una correcta conversión.

- Lista de sensibilidad: formada por la señal pulsada, visible su declaración en la **Tabla 4.9.** Lo cual provocará que con cada pulsación, se ejecute este process.
- Variables internas (carga, numtemp, numtecla): Nos permitirán su cambio instantáneo a la hora de hacer la operación para finalmente ser asignado a la señal de nuestra estructura la cual podrá ser accesible en distintos process a lo largo del proyecto.

- Señal final (sumatoria): será la señal donde se guardará el resultado que se refrescará al finalizar el process en cuestión. Su declaración puede ser visible en la **Tabla 4.9.**

Básicamente, en este process lo que nos aseguraremos de que nuestro valor actual, se multiplique por 10 para añadir un cero más a nuestro valor al que luego sumaremos el nuevo valor introducido.

Además habrá que considerar, que no todas las teclas pulsadas pudieran ser un valor a “cargar”, por eso usaremos una condición general donde nos aseguremos de que tecla deba estar entre 0 y 9. Sin esta condición, nuestra sumatoria podría recibir los valores asociados a cualquiera de las 6 operaciones como podemos apreciar en la **Tabla 3.2.**

4.7.4.2. Retroceso.

```
159 retroceso: process (pulsada)
160 variable numtemp: integer range 0 to 99999999;
161 begin
162 if tecla="1111" then     -- si la tecla pulsada es 15
163     numtemp:= CONV_INTEGER(numero);
164     numtemp:=numtemp/10;
165     auxiliar<= CONV_STD_LOGIC_VECTOR(numtemp,27);
166 else
167     auxiliar<=(others => '0');
168 end if;
169
170 end process retroceso;
```

Tabla 4.23

Se basará en no solo borrar el último valor introducido en el display, sino en dicho valor interno.

- Lista de sensibilidad: formada por la señal pulsada, visible su declaración en la **Tabla 4.9.** Lo cual provocará que con cada pulsación, se ejecute este process.
- Variables internas (numtemp): Nos permitirán su cambio instantáneo a la hora de hacer la operación para finalmente ser asignado a la señal de

nuestra estructura la cual podrá ser accesible en distintos process a lo largo del proyecto.

- Señal final (auxiliar): será la señal donde se guardará el resultado que se refrescará al finalizar el process en cuestión. Su declaración puede ser visible en la **Tabla 4.9.**

Esta solución de codificación será posible gracias a que la operación de la ALU "/" efectuará la división y se quedará con el cociente solo en sus valores enteros, lo que hará que cualquier valor dividido entre 10, sea el mismo valor, solo que siento eliminada las unidades de chico dato antes de la operación.

4.7.4.3. Lectura.

```
lectura: process(pulsada)
variable evaluador,numtecla: integer ;
begin
if rising_edge(pulsada) then
    numtecla := CONV_INTEGER(tecla);
    if numtecla<10 then     --estoy introduciendo un numero entre 0 y 9
        if estado="00" or estado="01" then  --si estamos en el estado de  escribir los numerandos
            d7<=d6;
            d6<=d5;
            d5<=d4;
            d4<=d3;
            d3<=d2;
            d2<=d1;
            d1<=d0;
            d0(3 downto 0)<=tecla;
            d0(4)<='0';
            numero<= sumatoria;
        end if;
    elsif numtecla=15 then   --retroceso
        if estado="10" then  -- si estoy en el estado de muestra de resultado, limpio todo
            d0<="10000";
            d1<="10000";
            d2<="10000";
            d3<="10000";
            d4<="10000";
            d5<="10000";
            d6<="10000";
            d7<="10000";
            operacion<="00";
            estado<="00";
            numero<=(others => '0');
            acumulador<=(others => '0');
        else              --si no, retrocedemos del numero que se escribe actualmente
            evaluador:=CONV_INTEGER(d0(3 downto 0));  --evaluamos el último valor introducido
            if evaluador>9 and evaluador<14 then -- si es una operación, retrocedemos al valor inicial
                estado<="00";
                numero<=acumulador;                  --cargandolo de nuevo en el numero actual
                d0<=d1;
                d1<=d2;
                d2<=d3;
                d3<=d4;
                d4<=d5;
                d5<=d6;
                d6<=d7;
                d7<="10000";
```

```
217             else          --si es un valor, retrocedemos con normalidad
218                 numero<=auxiliar;
219                 d0<=d1;
220                 d1<=d2;
221                 d2<=d3;
222                 d3<=d4;
223                 d4<=d5;
224                 d5<=d6;
225                 d6<=d7;
226                 d7<="10000";
227             end if;
228         end if;
229     elsif numtecla=14 then  --igual
230         if estado="01" then --la primera vez que pulsamos , mostramos el igual
231             d7<=d6;
232             d6<=d5;
233             d5<=d4;
234             d4<=d3;
235             d3<=d2;
236             d2<=d1;
237             d1<=d0;
238             d0(3 downto 0)<=tecla;
239             d0(4)<='0';
240             estado<="10";
241         elsif estado="10" then  -- al volver a pulsar, mostramos el valor ya convertido
242             d7(3 downto 0)<=resultadobcd(31 downto 28);
243             d7(4)<='0';
244             d6(3 downto 0)<=resultadobcd(27 downto 24);
245             d6(4)<='0';
246             d5(3 downto 0)<=resultadobcd(23 downto 20);
247             d5(4)<='0';
248             d4(3 downto 0)<=resultadobcd(19 downto 16);
249             d4(4)<='0';
250             d3(3 downto 0)<=resultadobcd(15 downto 12);
251             d3(4)<='0';
252             d2(3 downto 0)<=resultadobcd(11 downto 8);
253             d2(4)<='0';
254             d1(3 downto 0)<=resultadobcd(7 downto 4);
255             d1(4)<='0';
256             d0(3 downto 0)<=resultadobcd(3 downto 0);
257             d0(4)<='0';
258         end if;
259     elsif numtecla>9 and numtecla<14 then  --cualquier valor de operacion
260         if estado="00" then     -- siempre que estemos escribiendo el primer valor
261             acumulador<=numero; -- guardamos el valor y lo limpiamos para volver a escribir
262             numero<=(others => '0');
263             estado<="01";        --pasamos al siguiente valor a introducir
264             d7<=d6;
265             d6<=d5;
266             d5<=d4;
267             d4<=d3;
268             d3<=d2;
269             d2<=d1;
270             d1<=d0;
271             d0(3 downto 0)<=tecla;   --mostramos visualmente la operacion
272             d0(4)<='0';
```

```
273             case numtecla is       --marcamos que operacion hemos pulsado y lo guardamos
274                 when 10 =>
275                     operacion<="00";
276                 when 11 =>
277                     operacion<="01";
278                 when 12 =>
279                     operacion<="10";
280                 when 13 =>
281                     operacion<="11";
282                 when others =>
283                     operacion<="00";
284             end case;
285         end if;
286     end if;
287 end if;
288 end process lectura;
```

Tabla 4.24

Se basará en la lectura del valor que se ha pulsado e introducido por el teclado y las distintas casuísticas que se valorarán en función del valor introducido, ya sea un número, operación, limpieza o igualdad.

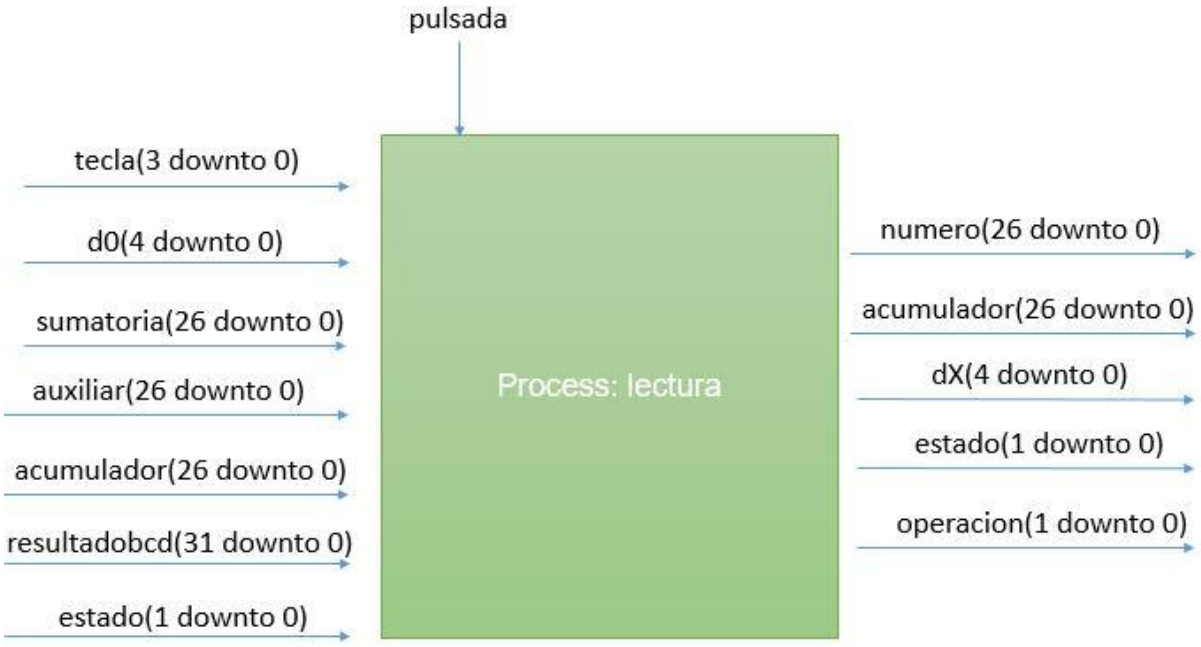

Ilustración 4.9

- Lista de sensibilidad: formada por la señal pulsada, visible su declaración en la **Tabla 4.9.** Lo cual provocará que con cada pulsación, se ejecute este process.

- Variables internas (evaluador, numtecla): Nos permitirán su cambio instantáneo, convirtiendo además valores binarios en enteros para visibilizar mejor las distintas condiciones.
- Señales final: será las señales de salida de nuestro process.
    - numero: apreciable su declaración en la **Tabla 4.8.** Mostrará el último valor generado tras el último dígito introducido por teclado.
    - acumulador: apreciable su declaración en la **Tabla 4.8.** Mostrará el primer valor de los dos que forman la operación.
    - dX: apreciable su declaración en la **Tabla 4.8.** Serán los valores a mostrar en cada dígito de nuestro display. En este caso, existirían 8 señales d (d7…d0), el cual incluirá ya el número a mostrar en formato BCD.
    - estado: apreciable su declaración en la **Tabla 4.9.** Nos ubicará la situación en la que se encuentra nuestro programa en cada ejecución de la misma y además evitará posibles errores de ejecución.
    - operacion: apreciable su declaración en la **Tabla 4.8.** Guardará la operación que se desea realidad y que será pulsada en la transición del estado 0 al estado 1.

Por tanto definiremos que la funcionalidad general de nuestra lectura, se basará principalmente en un avance de estados distintos donde expondremos la funcionalidad elegida a la hora de hacer el diseño del código descriptivo.

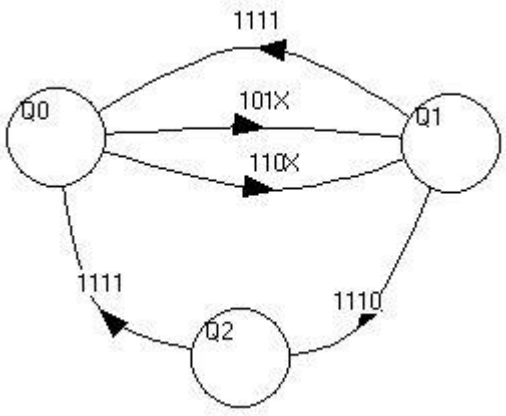

Ilustración 4.10

Según nuestro diagrama de estados regido el cambio entre los estados por la señal de entrada **"tecla"** podremos observar que nuestros estados serán.

- Q0: estado 0, en esta situación estaremos introduciendo el primero de los valores, donde siempre que el valor esté entre 0 y 9 nos mantendremos en este mismo estado, pero que al recibir cualquier operación pasaremos al siguiente estado.
- Q1: estado 1, situación donde se introduce el segundo número, del que solo se saldrá cuando se pulse limpieza y nuestro valor actual sea 0, o cuando se pulse el igual.
- Q2: estado 2, situación donde se mostrará el resultado final, del que solo se saldrá pulsando limpieza.

Finalmente, nuestro código descriptivo se basará en cuál es el valor introducido por teclado y el estado en el que nos encontremos como mostraremos en la **Tabla 4.23.**

- numtecla introducida es un número [0,9] y estado de escritura [Q0,Q1]: visible en las condiciones de las **líneas 177 y 178**, En este caso se efectuará un corrimiento de los dígitos del teclado para introducir el nuevo dígito y además cargaremos en número el nuevo valor "sumatoria" obtenido tras el process "Avance" visible en el **punto 4.7.4.1**. Además nos mantendremos en el mismo estado en el que nos encontrábamos.
- numtecla introducida es una operación [10,13] y estado de escritura [Q0]: visible en las condiciones de las **líneas 259 y 260.**La pulsación de una operación solo se efectuará en el estado 0.
    - **Línea 261 y 262,** Nos indicará que nuestro primer valor ha sido introducido por completo, por lo que se guardará en la señal "acumulador", limpiando la señal "número" a la espera del segundo valor.
    - **Línea 263**, avanzaremos al estado 1 para indicar que los siguientes dígitos a introducir pertenecerán a el segundo valor.

- **Líneas 264 a 272,** se efectuará un corrimiento de bits y se mostrará en display el signo de operación asignado a cada operación visible en el **punto 4.6.**
- **Finalmente,** se guardará la operación seleccionada para utilizarla a la hora del cálculo posterior efectuando las asignaciones.

Suma (+,10)	operacion=”00”
Resta (-,11)	operacion=”01”
Multiplicación(X,12)	operacion=”10”
División (/,13)	operacion=”11”

Tabla 4.25

- numtecla introducida es la igualdad [14]: visible en las condiciones de la **línea 229.**La funcionalidad de la igualdad dependerá del estado en que nos encontremos.
  - Q1, **líneas 230-240**: Se efectuará un corrimiento de los dígitos y se mostrará en pantalla el signo de la igualdad, en este paso además se dará tiempo al cálculo de la operación y su conversión a BCD. Así mismo se indicará el avance al estado 2.
  - Q2, **líneas 241-258**: Nos mostrará en los displays nuestro resultado final, por lo que asignaremos a cada dígito, su valor correspondiente al vector “resultadobcd”.
- numtecla introducida es la limpieza [15]: visible en las condiciones de la **línea 190.**La funcionalidad de la limpieza dependerá del estado en que nos encontremos.
  - Q2, **líneas 191-203**: Se efectuará la limpieza total del display, reinicializando el sistema, por lo que será necesaria la limpieza de todos los valores utilizados, así como el avance al estado 0.
  - Q0 y Q1**, líneas 204-228**: Se efectuará una evaluación de cuál es el último dígito que fue introducido que sea visible en los displays, de este modo.

- Evaluador es una operación: significará que nuestro segundo valor estará vacío, por lo que estaremos en el estado 1. Será necesario retroceder al estado 0, y se cargará el valor anterior “acumulador” en número para seguir borrando este valor en nuevas pulsaciones o pulsar otra operación. Haciendo además el sucesivo corrimiento de dígitos a la derecha.
- Evaluador es un número: en este caso, da igual si nos encontramos en el estado 0 o 1, simplemente se efectuará la carga del valor “auxiliar” efectuado en el process “retroceso” visible en el **punto 4.7.4.1** de esta memoria, y el sucesivo corrimiento a la derecha de los dígitos.

De esta forma se completaría la muestra y lectura de los valores necesarios para nuestras operaciones, así como guardar que operación hemos seleccionado, siendo explicado a continuación la forma en que estas operaciones son efectuadas

.

4.7.5.Funcionalidad "ALU".

Además, en nuestro bloque tendremos la ventaja de que nuestra FPGA contiene una ALU interna gracias a la elección de correctas librerías, que nos facilitará mucho la labor de las distintas operaciones. Aun así, evaluaremos por separado las distintas operaciones para mantener un correcto orden en su cálculo en estos distintos process:

4.7.5.1. Sumador.

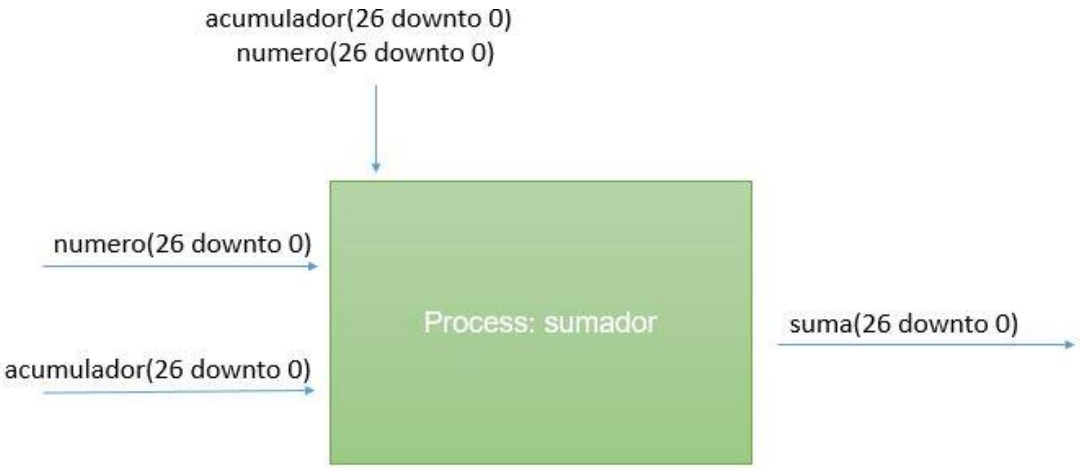

Ilustración 4.11

Comenzaremos por la operación de suma, la cual gracias a las librerías anteriormente introducidas se podrá usar las posibilidades de conversión que nos permite evitando así ciertos problemas que podrías provocarse si las operaciones fueran efectuadas a nivel binario. En nuestro process definiremos.

```
290 sumador: process (acumulador,numero)
291 variable sumando,sum1,sum2: integer range 0 to 99999999;
292 begin
293 sum1 := CONV_INTEGER(acumulador);
294 sum2 := CONV_INTEGER(numero);
295 sumando:=sum1+sum2;
296 suma<= CONV_STD_LOGIC_VECTOR(sumando,27);
297 end process sumador;
```

Tabla 4.26

- Lista de sensibilidad: formada por las señales acumulador y número, visible su declaración en la **Tabla 4.9.** Lo cual provocará que con cada cambio en estas señales, siempre sea operada la suma y su valor esté disponible para ser tomado.
- Variables internas (sumando, sum1, sum2): Nos permitirán su cambio instantáneo a la hora de hacer la operación para finalmente ser asignado a la señal de nuestra estructura la cual podrá ser accesible en distintos process a lo largo del proyecto.
- Señal final (suma): será la señal donde se guardará el resultado que se refrescará al finalizar el process en cuestión. Su declaración puede ser visible en la **Tabla 4.9.**

4.7.5.2. Restador.

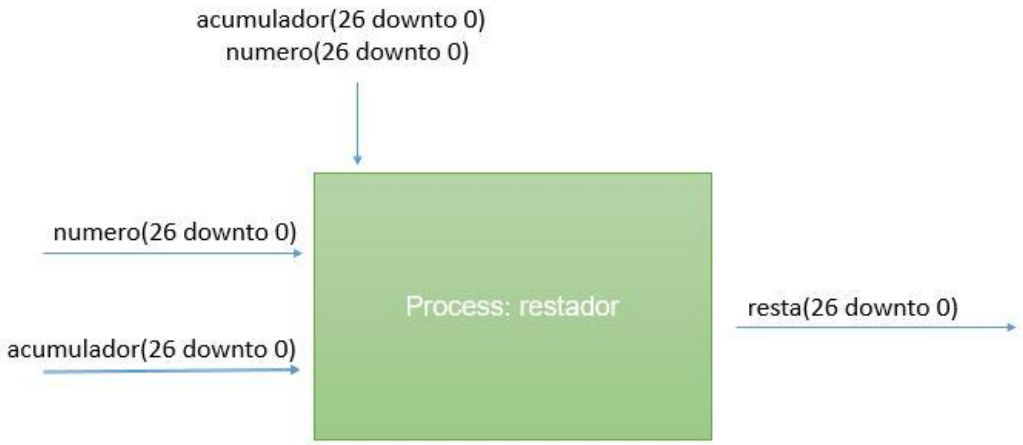

Ilustración 4.12

La siguiente operación a realizar donde se volverá a utilizarla posibilidad de conversión, evitando así ciertos problemas que podrías provocarse si las operaciones fueran efectuadas a nivel binario. En nuestro process definiremos.

```
299 restador: process (acumulador,numero)
300 variable restando,res1,res2: integer range 0 to 99999999;
301 begin
302 res1 := CONV_INTEGER(acumulador);
303 res2 := CONV_INTEGER(numero);
304 if res1>res2 then
305     restando:=res1-res2;
306 else
307     restando:=0;
308 end if;
309 resta<= CONV_STD_LOGIC_VECTOR(restando,27);
310 end process restador;
```

Tabla 4.27

- Lista de sensibilidad: formada por las señales acumulador y número, visible su declaración en la **Tabla 4.9.** Lo cual provocará que con cada cambio en estas señales, siempre sea operada la resta y su valor esté disponible para ser tomado.
- Variables internas (restando, res1, res2): Nos permitirán su cambio instantáneo a la hora de hacer la operación para finalmente ser asignado a la señal de nuestra estructura la cual podrá ser accesible en distintos process a lo largo del proyecto.
- Señal final (resta): será la señal donde se guardará el resultado que se refrescará al finalizar el process en cuestión. Su declaración puede ser visible en la **Tabla 4.9.**
- Caso específico: será evaluada la posibilidad de que el minuendo (primer valor) sea menor que el sustraendo (segundo valor), en dicho caso como nuestra calculadora solo evaluará valores positivos, se introducirá el valor 0 para indicar que ese será el menor valor disponible para la resta en ese caso.

4.7.5.3. Multiplicador.

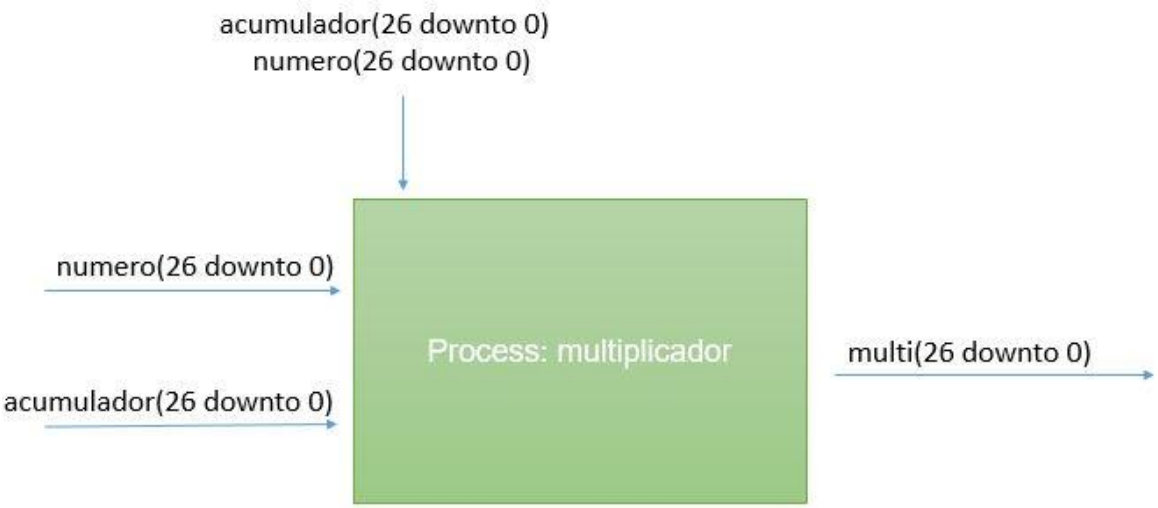

Ilustración 4.13

De igual forma podremos operar sobre la multiplicación, esta es el cálculo que más problemas nos habría causado si se hubiera operado a nivel binario ya que en la multiplicación binaria, el número de bits del resultado sería la suma de los bits de cada operando.

```
312  multiplicacion: process (acumulador,numero)
313  variable multiplicador,multil,multi2: integer range 0 to 99999999
314  begin
315  multil := CONV_INTEGER(acumulador);
316  multi2 := CONV_INTEGER(numero);
317
318  multiplicador:=multil*multi2;
319  multi<= CONV_STD_LOGIC_VECTOR(multiplicador,27);
320
321  end process multiplicacion;
```

Tabla 4.28

- Lista de sensibilidad: formada por las señales acumulador y número, visible su declaración en la **Tabla 4.9.** Lo cual provocará que con cada cambio en estas señales, siempre sea operada la multiplicación y su valor esté disponible para ser tomado.
- Variables internas (multiplicador, multi1, multi2): Nos permitirán su cambio instantáneo a la hora de hacer la operación para finalmente ser asignado a la señal de nuestra estructura la cual podrá ser accesible en distintos process a lo largo del proyecto.
- Señal final (multi): será la señal donde se guardará el resultado que se refrescará al finalizar el process en cuestión. Su declaración puede ser visible en la **Tabla 4.9.**

4.7.5.4. División.

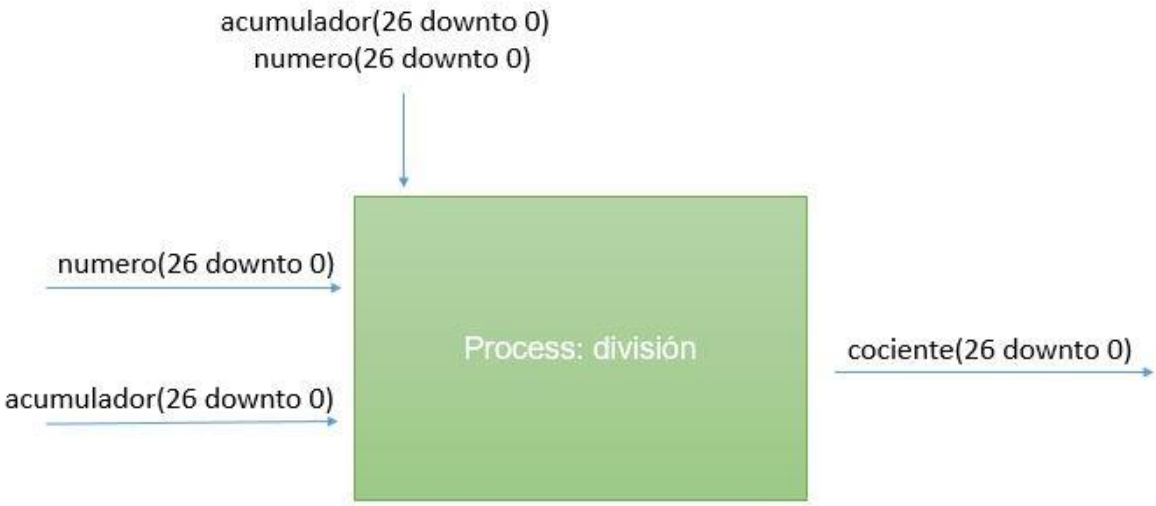

Ilustración 4.14

Finalmente con la división actuaremos de la misma forma pudiendo aplicar el mismo formato de process que en los anteriores casos pero con algunas especificaciones puntuales.

```
323 division: process (acumulador,numero)
324 variable divisor,num,den: integer range 0 to 99999999;
325 begin
326 num := CONV_INTEGER(acumulador);
327 den := CONV_INTEGER(numero);
328 if den>0 then
329     divisor:=num/den;
330 else
331     divisor:=99999999;
332 end if;
333 cociente<= CONV_STD_LOGIC_VECTOR(divisor,27);
334 end process division;
```

323 division: process (acumulador,numero)

Tabla 4.29

- Lista de sensibilidad: formada por las señales acumulador y número, visible su declaración en la **Tabla 4.9.** Lo cual provocará que con cada cambio en estas señales, siempre sea operada la división y su valor esté disponible para ser tomado.
- Variables internas (divisor, num, den): Nos permitirán su cambio instantáneo a la hora de hacer la operación para finalmente ser asignado a la señal de nuestra estructura la cual podrá ser accesible en distintos process a lo largo del proyecto.
- Señal final (cociente): será la señal donde se guardará el resultado que se refrescará al finalizar el process en cuestión. Su declaración puede ser visible en la **Tabla 4.9.**
- Caso específico: será evaluada la posibilidad de que el numerador valga cero, en dicho caso Debido al que no es posible efectuar dicho cálculo directamente asignaremos el valor más alto posible, ya que cualquier número positivo dividido entre 0 es infinito, por lo que “99999999” será nuestro valor más cercano a infinito.

4.7.5.5. Gestión de la operación.

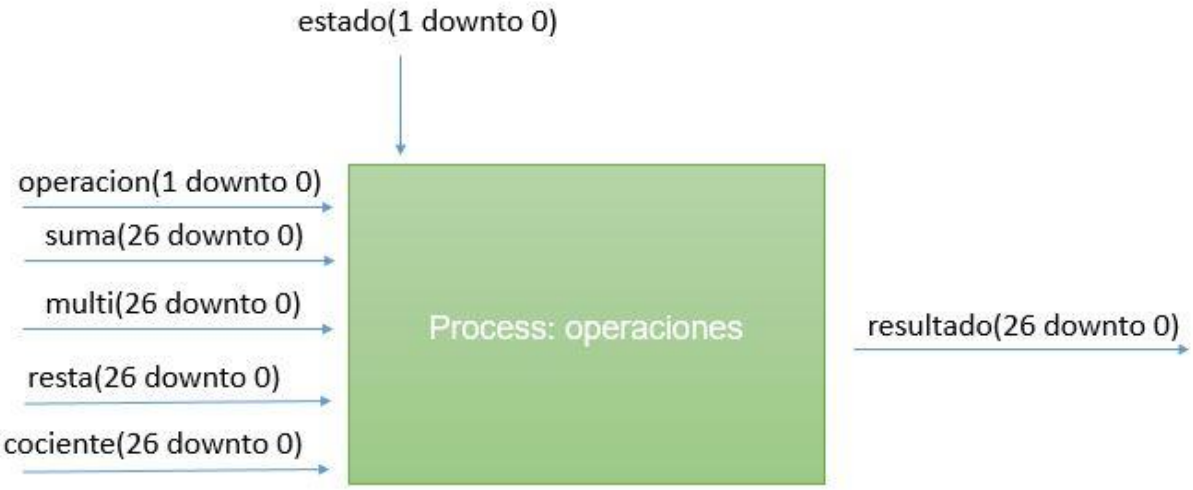

Ilustración 4.15

Tras la preparación de cada una de las operaciones posibles a realizar en los anteriores process finalmente diseñaremos uno donde en función de la discriminación según el estado en que nos encontremos, se seleccione la operación deseada introducida por teclado y que se verá posteriormente en la lectura del teclado.

```
336  operaciones: process(estado)
337  begin
338          if estado="10" then
339              case operacion is
340                  when "00" =>
341                       resultado<=suma;
342                  when "01" =>
343                       resultado<=resta;
344                  when "10" =>
345                       resultado<=multi;
346                  when "11" =>
347                       resultado<=cociente;
348              end case;
349          end if;
350  end process operaciones;
```

Tabla 4.30

- Lista de sensibilidad: formada por las señales acumulador y número, visible su declaración en la **Tabla 4.9.** Lo cual provocará que con cada cambio en estas señales, siempre sea operada la división y su valor esté disponible para ser tomado.
- Señal final (resultado): será la señal donde se guardará el resultado que se refrescará al finalizar el process en cuestión. Su declaración puede ser visible en la **Tabla 4.9.** Esta será la señal que se dirija a la posterior conversión en BCD que veremos a continuación y cómo podemos ver en la **Ilustración 4.3.**

La funcionalidad principal estará regida por la restricción de que solo cuando nos encontremos en el estado 2, podamos introducir en resultado los valores de la operación seleccionada anteriormente por el case principal del process.

4.7.6. Funcionalidad "Conversión a BCD".

Finalmente, y tras la correcta obtención del resultado obtenido por medio de nuestro process "operaciones" con los valores obtenidos del process "lectura" pasaremos a su conversión a formato BCD (Binary-Coded Decimal) ya que actualmente este valor se encuentra guardado en valor binario, no siendo el formato correcto para su muestra en los displays. Para eso sabremos que nuestro formato BCD se basará en la siguiente tabla.

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10-15	No importa

Tabla 4.31

Entre las muchas formas de convertir nuestro valor binario en BCD, elegiremos un método muy visual de cómo hacer la conversión. Se mostrará un ejemplo cualquiera donde representaremos el valor de "28071995" cuyo número binario de 27 bits sería "001101011000101100000111011".

ACCIÓN	BCD (4 BITS CADA UNO)								BINARIO (27 BITS)
	DEC. MILLÓN	UNI. MILLÓN	CENT. MILLAR	DEC. MILLAR	UNI. MILLAR	CENT.	DEC.	UNI.	NÚMERO BINARIO
Disposición inicial									001101011000101100000111011
Desplazar a la izq (3)								001	10101100010110000011101 1
Desplazar a la izq (4)								0011	0101100010110000011101 1
Desplazar a la izq (5)							0	0110	1011000101100000111011
+3 a U							0	1001	1011000101100000111011
Desplazar a la izq (6)							01	0011	0110001011000001110 11
Desplazar a la izq (7)							010	0110	1100010110000011101 1
+3 a U							010	1001	1100010110000011101 1
Desplazar a la izq (8)							0101	0011	1000101100000111011
+3 a D							1000	0011	1000101100000111011
Desplazar a la izq (9)						1	0000	0111	0001011000001110 11
+3 a U						1	0000	1010	0001011000001110 11
Desplazar a la izq (10)						10	0001	0100	0010110000011101 1
Desplazar a la izq (11)						100	0010	1000	0101100000111011
+3 a U						100	0010	1011	0101100000111011
Desplazar a la izq (12)						1000	0101	0110	1011000001110 11
+3 a C/D/U						1011	1000	1001	1011000001110 11
Desplazar a la izq (13)					1	0111	0001	0011	0110000011101 1
+3 a C					1	1010	0001	0011	0110000011101 1
Desplazar a la izq (14)					11	0100	0010	0110	1100000111011
+3 a U					11	0100	0010	1001	1100000111011
Desplazar a la izq (15)					110	1000	0101	0011	1000001110 11
+3 a Um/C/D					1001	1011	1000	0011	1000001110 11
Desplazar a la izq (16)				1	0011	0111	0000	0111	0000011101 1
+3 a C/U				1	0011	1010	0000	1010	0000011101 1
Desplazar a la izq (17)				10	0111	0100	0001	0100	000011101 1
+3 a Um				10	1010	0100	0001	0100	000011101 1
Desplazar a la izq (18)				101	0100	1000	0010	1000	00011101 1
+3 a Dm/C/U				1000	0100	1011	0010	1011	00011101 1
Desplazar a la izq (19)			1	0000	1001	0110	0101	0110	0011101 1
+3 a Um/C/D/U			1	0000	1100	1001	1000	1001	0011101 1
Desplazar a la izq (20)			10	0001	1001	0011	0001	0010	011101 1
+3 a Um			10	0001	1100	0011	0001	0010	011101 1
Desplazar a la izq (21)			100	0011	1000	0110	0010	0100	11101 1
+3 a Um/C			100	0011	1011	1001	0010	0100	11101 1
Desplazar a la izq (22)			1000	0111	0111	0010	0100	1001	1101 1
+3 a Cm/Dm/Um/U			1011	1010	1010	0010	0100	1100	1101 1
Desplazar a la izq (23)		1	0111	0101	0100	0100	1001	1001	101 1
+3 a Cm/Dm/D/U		1	1010	1000	0100	0100	1100	1100	101 1
Desplazar a la izq (24)		11	0101	0000	1000	1001	1001	1001	01 1
+3 a Cm/Um/C/D/U		11	1000	0000	1011	1100	1100	1100	01 1
Desplazar a la izq (25)		111	0000	0001	0111	1001	1001	1000	1 1
+3 a UM/Um/C/D/U		1010	0000	0001	1010	1100	1100	1011	1 1
Desplazar a la izq (26)	1	0100	0000	0011	0101	1001	1001	0111	1
+3 a Um/C/D/U	1	0100	0000	0011	1000	1100	1100	1010	1
Desplazar a la izq (27)	10	1000	0000	0111	0001	1001	1001	0101	
	2	8	0	7	1	9	9	5	

Tabla 4.32

Como podemos apreciar, esta conversión se basa en:

- Corrimiento a la izquierda de los bits hasta introducir todos y cada uno de los bits del número binario
- En el caso de que cualquiera de los valores de los posicionamientos de nuestro número decimal supere el 5- “0101” Sumarle el valor de 3- “0011”, esta será una forma de evitar que nuestro valor aunque un nuevo movimiento hacia la izquierda pase a ser el valor de 10-“1010” valor no existente en BCD, sino que tras la conversación nuestro valor sea “1000” el cual al hacer el corrimiento, pasará a ser “0001 0000” el cual en BCD será el número 10.

Este análisis será necesario de efectuarse en nuestro código de manera simultánea en todas las posiciones, y tras la visualización de su funcionamiento en la **Tabla 4.30**, Pasaremos a su código descriptivo.

```
352 bcd: process(resultado)
353     variable z: STD_LOGIC_VECTOR(58 downto 0);
354 begin
355     -- Inicialización de datos en cero.
356     if resultado>"10111110101111000001111111" then
357         resultadobcd<="100110011001100110011001100110011001";
358     else
359         z := (others => '0');
360         -- Se realizan los primeros tres corrimientos.
361         z(29 downto 3) := resultado;
362         for l in 0 to 23 loop
363             -- Unidades (4 bits).
364             if z(30 downto 27) > 4 then
365                 z(30 downto 27) := z(30 downto 27) + 3;
366             end if;
367             -- Decenas (4 bits).
368             if z(34 downto 31) > 4 then
369                 z(34 downto 31) := z(34 downto 31) + 3;
370             end if;
371             -- Centenas (4 bits).
372             if z(38 downto 35) > 4 then
373                 z(38 downto 35) := z(38 downto 35) + 3;
374             end if;
375             -- unidades de millar (4 bits).
376             if z(42 downto 39) > 4 then
377                 z(42 downto 39) := z(42 downto 39) + 3;
378             end if;
```

```
379             -- decenas de millar (4 bits).
380             if z(46 downto 43) > 4 then
381                 z(46 downto 43) := z(46 downto 43) + 3;
382             end if;
383             -- centenas de millar (4 bits).
384             if z(50 downto 47) > 4 then
385                 z(50 downto 47) := z(50 downto 47) + 3;
386             end if;
387             -- unidades de millon (4 bits).
388             if z(54 downto 51) > 4 then
389                 z(54 downto 51) := z(54 downto 51) + 3;
390             end if;
391             -- decenas de millon (3 bits).
392             if z(58 downto 55) > 4 then
393                 z(58 downto 55) := z(58 downto 55) + 3;
394             end if;
395
396             -- Corrimiento a la izquierda.
397             z(58 downto 1) := z(57 downto 0);
398         end loop;
399         -- Pasando datos de variable Z, correspondiente a BCD.
400         resultadobcd <= z(58 downto 27);
401     end if;
402 end process bcd;
```

Tabla 4.33

- Lista de sensibilidad: formada por la señal resultado, visible su declaración en la **Tabla 4.9.** Lo cual provocará que con cada cambio de resultado, algo que solo ocurrirá tras una nueva operación, esta conversión se active.
- Variables internas (z): Nos permitirá el cambio instantáneo sobre este vector que integrará en él los 32 bits que formarán los 8 dígitos de 4 bits cada uno, en sus posiciones menos significativas, junto a los 27 bits del valor en formato binario.
- Señal final (resultadobcd): será la señal donde se guardará el resultado tras la conversión, y que pertenecerá a z del bit 27 al 58... Su declaración puede ser visible en la **Tabla 4.9.**

- Caso específico: será evaluada la posibilidad de que nuestro número binario sea mayor que “99999999” ya que este será nuestro último valor posible de ser mostrado, siendo en ese caso introducido de forma manual a resultadobcd, los valores necesarios para mostrar este número como situación alternativa, aunque igualmente podría haber sido seleccionado el número 0.

Esta conversión es más complicada al hacerla de forma manual a de forma codificada, ya que se basará únicamente.

1. Corrido inicial de 3 posiciones, para empezar a evaluar las posiciones con mínimo 3 bits introducidos.
2. Introducción en un “for” general de 24 vueltas, tantos como bits quedan del valor binario.
3. Evaluación de todas las posiciones para verificar si son igual o mayor al número 5.
4. Finalmente tras las evaluaciones y operaciones, corrimiento unitario a la izquierda para avanzar el vector antes de una nueva vuelta en el bucle.

De esta forma nuestro proyecto habrá finalizado en todas y cada una de sus partes.

5. CONCLUSIONES

Tras el desarrollo de esta aplicación implementada a nuestra FPGA, logramos la obtención de una calculadora funcional y óptima no solo optimizada para la shield “Basys 3” donde estará integrada nuestra FPGA, sino que se habrá optimizado tanto su diseño hardware en las máximas posibilidades permitidas, como software permitiendo así mismo la ampliación de las posibilidades de esta calculadora de una manera simple y adaptable en función del uso de shield con mayores posibilidades de conexión que la actual.

Como posibles mejoras podremos optar a:

- Uso de un teclado con posibilidad a más botones con los que añadir diferentes operaciones a nuestro proyecto.
- Uso de tantas operaciones matemáticas como se desee ampliando la señal operaciones y diseñando diferentes process como por ejemplo se pueden encontrar dos más en los **anexos 6.1 a partir de la línea 408** como es “cálculo de múltiplos” y el “cálculo de raíces”.
- Unificación del teclado y los displays en un mismo conjunto quizás cerrado mediante una carcasa por diseño 3D.

Por tanto se habrán conseguido los objetivos de lograr el diseño de una calculadora de manera explicativa y progreso de nuestro proyecto desde el análisis de los elementos disponibles, como de las máximas aportaciones posibles de forma externa a nuestra placa.

6. ANEXOS

6.1. Módulo “Calculadora”.

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24  use IEEE.STD_LOGIC_unsigned.all;
25  use IEEE.NUMERIC_STD.all;
26  use IEEE.STD_LOGIC_arith.all;
27
28  -- Uncomment the following library declaration if using
29  -- arithmetic functions with Signed or Unsigned values
30  --use IEEE.NUMERIC_STD.ALL;
31
32  -- Uncomment the following library declaration if instantiating
33  -- any Xilinx leaf cells in this code.
34  --library UNISIM;
35  --use UNISIM.VComponents.all;
36
37  entity Calculadora is
38      Port(
39          led : out STD_LOGIC_VECTOR (15 downto 0);
40          COL : in STD_LOGIC_VECTOR (3 downto 0);
41          FIL : out STD_LOGIC_VECTOR (3 downto 0);
42          cat : out STD_LOGIC_VECTOR (7 downto 0);
43          seg : out STD_LOGIC_VECTOR (7 downto 0);
44          clk: in STD_LOGIC
45
46          );
47  end Calculadora;
48
49  architecture Behavioral of Calculadora is
50
51  signal clk1kHz : STD_LOGIC;
52
53  signal tecla : STD_LOGIC_VECTOR (3 downto 0) :="0000";
54  signal pulsada : STD_LOGIC :='0';
55
56  signal d0 : STD_LOGIC_VECTOR (4 downto 0) :="10000";
57  signal d1 : STD_LOGIC_VECTOR (4 downto 0) :="10000";
58  signal d2 : STD_LOGIC_VECTOR (4 downto 0) :="10000";
59  signal d3 : STD_LOGIC_VECTOR (4 downto 0) :="10000";
60  signal d4 : STD_LOGIC_VECTOR (4 downto 0) :="10000";
61  signal d5 : STD_LOGIC_VECTOR (4 downto 0) :="10000";
62  signal d6 : STD_LOGIC_VECTOR (4 downto 0) :="10000";
63  signal d7 : STD_LOGIC_VECTOR (4 downto 0) :="10000";
64
```

```
65  signal acumulador:STD_LOGIC_VECTOR(26 downto 0):=(others => '0');
66  signal numero:STD_LOGIC_VECTOR(26 downto 0):=(others => '0');
67  signal sumatoria:STD_LOGIC_VECTOR(26 downto 0):=(others => '0');
68  signal auxiliar:STD_LOGIC_VECTOR(26 downto 0):=(others => '0');
69  signal resultado:STD_LOGIC_VECTOR(26 downto 0):=(others => '0');
70  signal resultadobcd:STD_LOGIC_VECTOR(31 downto 0):=(others => '0');
71
72  signal cociente:STD_LOGIC_VECTOR(26 downto 0):=(others => '0');
73  signal multi:STD_LOGIC_VECTOR(26 downto 0):=(others => '0');
74  signal suma:STD_LOGIC_VECTOR(26 downto 0):=(others => '0');
75  signal resta:STD_LOGIC_VECTOR(26 downto 0):=(others => '0');
76
77  signal operacion: STD_LOGIC_VECTOR (1 downto 0):="00"; --
78  signal estado: STD_LOGIC_VECTOR (1 downto 0):="00"; --
79
80  component GenCLK
81      Port (
82      clkin: in STD_LOGIC;
83      clkout: out STD_LOGIC
84      );
85  end component;
86
87  component tec4x4
88      port (
89      COL : in STD_LOGIC_VECTOR (3 downto 0);
90      FIL : out STD_LOGIC_VECTOR (3 downto 0);
91      tecla : out STD_LOGIC_VECTOR (3 downto 0);
92      pulsada : out STD_LOGIC;
93      clk: in STD_LOGIC
94      );
95  end component;
96
97
98  component disp8x7seg
99      port (
100     cat : out STD_LOGIC_VECTOR (7 downto 0);
101     seg : out STD_LOGIC_VECTOR (7 downto 0);
102     d0 : in STD_LOGIC_VECTOR (4 downto 0);
103     d1 : in STD_LOGIC_VECTOR (4 downto 0);
104     d2 : in STD_LOGIC_VECTOR (4 downto 0);
105     d3 : in STD_LOGIC_VECTOR (4 downto 0);
106     d4 : in STD_LOGIC_VECTOR (4 downto 0);
107     d5 : in STD_LOGIC_VECTOR (4 downto 0);
108     d6 : in STD_LOGIC_VECTOR (4 downto 0);
109     d7 : in STD_LOGIC_VECTOR (4 downto 0);
110     clk: in STD_LOGIC
111     );
112 end component;
```

```
113
114 begin
115 clkmux : GenCLK
116     port map (
117         clkin=>clk,
118         clkout=>clk1kHz
119         );
120
121 teclado : tec4x4
122     port map (
123         COL(3 downto 0)=>COL(3 downto 0),
124         FIL(3 downto 0)=>FIL(3 downto 0),
125         tecla(3 downto 0)=>tecla(3 downto 0),
126         pulsada=>pulsada,
127         clk=>clk1kHz
128         );
129

130 display : disp8x7seg
131     port map (
132         cat(7 downto 0)=>cat(7 downto 0),
133         seg(7 downto 0)=>seg(7 downto 0),
134         d0(4 downto 0)=>d0(4 downto 0),
135         d1(4 downto 0)=>d1(4 downto 0),
136         d2(4 downto 0)=>d2(4 downto 0),
137         d3(4 downto 0)=>d3(4 downto 0),
138         d4(4 downto 0)=>d4(4 downto 0),
139         d5(4 downto 0)=>d5(4 downto 0),
140         d6(4 downto 0)=>d6(4 downto 0),
141         d7(4 downto 0)=>d7(4 downto 0),
142         clk=>clk1kHz
143         );
144
145 avance: process (pulsada)
146 variable carga,numtecla,numtemp: integer range 0 to 99999999;
147 begin
148 if tecla<="1001" then      -- si la tecla pulsada está entre 0 y 9
149     numtemp:= CONV_INTEGER(numero);
150     numtecla := CONV_INTEGER(tecla);
151     carga:=numtemp*10+numtecla;
152     sumatoria<= CONV_STD_LOGIC_VECTOR(carga,27);
153 else
154     sumatoria<=(others => '0');
155 end if;
156
157 end process avance;
```

```
158
159 retroceso: process (pulsada)
160 variable numtemp: integer range 0 to 99999999;
161 begin
162 if tecla="1111" then     -- si la tecla pulsada es 15
163     numtemp:= CONV_INTEGER(numero);
164     numtemp:=numtemp/10;
165     auxiliar<= CONV_STD_LOGIC_VECTOR(numtemp,27);
166 else
167     auxiliar<=(others => '0');
168 end if;
169
170 end process retroceso;
171

172 lectura: process(pulsada)
173 variable evaluador,numtecla: integer ;
174 begin
175 if rising_edge(pulsada) then
176     numtecla := CONV_INTEGER(tecla);
177     if numtecla<10 then     --estoy introduciendo un numero entre 0 y 9
178         if estado="00" or estado="01" then --estado de  escribir los numerandos
179             d7<=d6;
180             d6<=d5;
181             d5<=d4;
182             d4<=d3;
183             d3<=d2;
184             d2<=d1;
185             d1<=d0;
186             d0(3 downto 0)<=tecla;
187             d0(4)<='0';
188             numero<= sumatoria;
189         end if;

190     elsif numtecla=15 then   --retroceso
191         if estado="10" then --estado de muestra de resultado, limpio todo
192             d0<="10000";
193             d1<="10000";
194             d2<="10000";
195             d3<="10000";
196             d4<="10000";
197             d5<="10000";
198             d6<="10000";
199             d7<="10000";
200             operacion<="00";
201             estado<="00";
202             numero<=(others => '0');
203             acumulador<=(others => '0');
204         else               --si no, retrocedemos del numero que se escribe actualmente
205             evaluador:=CONV_INTEGER(d0(3 downto 0));  --evaluamos el último valor introducido
206             if evaluador>9 and evaluador<14 then --operación, retrocedemos al valor inicial
207                 estado<="00";
208                 numero<=acumulador;--cargandolo de nuevo en el numero actual
209                 d0<=d1;
210                 d1<=d2;
211                 d2<=d3;
212                 d3<=d4;
213                 d4<=d5;
214                 d5<=d6;
215                 d6<=d7;
216                 d7<="10000";
```

```
217             else           --si es un valor, retrocedemos con normalidad
218                 numero<=auxiliar;
219                 d0<=d1;
220                 d1<=d2;
221                 d2<=d3;
222                 d3<=d4;
223                 d4<=d5;
224                 d5<=d6;
225                 d6<=d7;
226                 d7<="10000";
227             end if;
228         end if;
229     elsif numtecla=14 then  --igual
230         if estado="01" then --la primera vez que pulsamos , mostramos el igual
231             d7<=d6;
232             d6<=d5;
233             d5<=d4;
234             d4<=d3;
235             d3<=d2;
236             d2<=d1;
237             d1<=d0;
238             d0(3 downto 0)<=tecla;
239             d0(4)<='0';
240             estado<="10";

241         elsif estado="10" then  -- al volver a pulsar, mostramos el valor ya convertido
242             d7(3 downto 0)<=resultadobcd(31 downto 28);
243             d7(4)<='0';
244             d6(3 downto 0)<=resultadobcd(27 downto 24);
245             d6(4)<='0';
246             d5(3 downto 0)<=resultadobcd(23 downto 20);
247             d5(4)<='0';
248             d4(3 downto 0)<=resultadobcd(19 downto 16);
249             d4(4)<='0';
250             d3(3 downto 0)<=resultadobcd(15 downto 12);
251             d3(4)<='0';
252             d2(3 downto 0)<=resultadobcd(11 downto 8);
253             d2(4)<='0';
254             d1(3 downto 0)<=resultadobcd(7 downto 4);
255             d1(4)<='0';
256             d0(3 downto 0)<=resultadobcd(3 downto 0);
257             d0(4)<='0';
258         end if;

259     elsif numtecla>9 and numtecla<14 then  --cualquier valor de operacion
260         if estado="00" then     -- siempre que estemos escribiendo el primer valor
261             acumulador<=numero; -- guardamos el valor y lo limpiamos para volver a escribir
262             numero<=(others => '0');
263             estado<="01";        --pasamos al siguiente valor a introducir
264             d7<=d6;
265             d6<=d5;
266             d5<=d4;
267             d4<=d3;
268             d3<=d2;
269             d2<=d1;
270             d1<=d0;
271             d0(3 downto 0)<=tecla;   --mostramos visualmente la operacion
272             d0(4)<='0';
```

```
273             case numtecla is       --marcamos que operacion hemos pulsado y lo guardamos
274                 when 10 =>
275                    operacion<="00";
276                 when 11 =>
277                    operacion<="01";
278                 when 12 =>
279                    operacion<="10";
280                 when 13 =>
281                    operacion<="11";
282                 when others =>
283                    operacion<="00";
284             end case;
285         end if;
286     end if;
287 end if;
288 end process lectura;
289
290 sumador: process (acumulador,numero)
291 variable sumando,suml,sum2: integer range 0 to 99999999;
292 begin
293 suml := CONV_INTEGER(acumulador);
294 sum2 := CONV_INTEGER(numero);
295 sumando:=suml+sum2;
296 suma<= CONV_STD_LOGIC_VECTOR(sumando,27);
297 end process sumador;
298
299 restador: process (acumulador,numero)
300 variable restando,res1,res2: integer range 0 to 99999999;
301 begin
302 res1 := CONV_INTEGER(acumulador);
303 res2 := CONV_INTEGER(numero);
304 if res1>res2 then
305     restando:=res1-res2;
306 else
307     restando:=0;
308 end if;
309 resta<= CONV_STD_LOGIC_VECTOR(restando,27);
310 end process restador;
312 multiplicacion: process (acumulador,numero)
313 variable multiplicador,multil,multi2: integer range 0 to 99999999;
314 begin
315 multil := CONV_INTEGER(acumulador);
316 multi2 := CONV_INTEGER(numero);
317
318 multiplicador:=multil*multi2;
319 multi<= CONV_STD_LOGIC_VECTOR(multiplicador,27);
320
321 end process multiplicacion;
322
```

```
division: process (acumulador,numero)
variable divisor,num,den: integer range 0 to 99999999;
begin
num := CONV_INTEGER(acumulador);
den := CONV_INTEGER(numero);
if den>0 then
    divisor:=num/den;
else
    divisor:=99999999;
end if;
cociente<= CONV_STD_LOGIC_VECTOR(divisor,27);
end process division;

operaciones: process(estado)
begin
        if estado="10" then
            case operacion is
                when "00" =>
                     resultado<=suma;
                when "01" =>
                     resultado<=resta;
                when "10" =>
                     resultado<=multi;
                when "11" =>
                     resultado<=cociente;
            end case;
        end if;
end process operaciones;

bcd: process(resultado)
    variable z: STD_LOGIC_VECTOR(58 downto 0);
begin
    -- Inicialización de datos en cero.
    if resultado>"10111110101111000001111111" then
        resultadobcd<="100110011001100110011001100110011001";
    else
        z := (others => '0');
        -- Se realizan los primeros tres corrimientos.
        z(29 downto 3) := resultado;
        for l in 0 to 23 loop
            -- Unidades (4 bits).
            if z(30 downto 27) > 4 then
                z(30 downto 27) := z(30 downto 27) + 3;
            end if;
            -- Decenas (4 bits).
            if z(34 downto 31) > 4 then
                z(34 downto 31) := z(34 downto 31) + 3;
            end if;
            -- Centenas (4 bits).
            if z(38 downto 35) > 4 then
                z(38 downto 35) := z(38 downto 35) + 3;
            end if;
            -- unidades de millar (4 bits).
            if z(42 downto 39) > 4 then
                z(42 downto 39) := z(42 downto 39) + 3;
            end if;
```

```
division: process (acumulador,numero)
variable divisor,num,den: integer range 0 to 99999999;
begin
num := CONV_INTEGER(acumulador);
den := CONV_INTEGER(numero);
if den>0 then
    divisor:=num/den;
else
    divisor:=99999999;
end if;
cociente<= CONV_STD_LOGIC_VECTOR(divisor,27);
end process division;

operaciones: process(estado)
begin
        if estado="10" then
            case operacion is
                when "00" =>
                     resultado<=suma;
                when "01" =>
                     resultado<=resta;
                when "10" =>
                     resultado<=multi;
                when "11" =>
                     resultado<=cociente;
            end case;
        end if;
end process operaciones;

bcd: process(resultado)
    variable z: STD_LOGIC_VECTOR(58 downto 0);
begin
    -- Inicialización de datos en cero.
    if resultado>"10111110101111000001111111" then
        resultadobcd<="100110011001100110011001100110011001";
    else
        z := (others => '0');
        -- Se realizan los primeros tres corrimientos.
        z(29 downto 3) := resultado;
        for l in 0 to 23 loop
            -- Unidades (4 bits).
            if z(30 downto 27) > 4 then
                z(30 downto 27) := z(30 downto 27) + 3;
            end if;
            -- Decenas (4 bits).
            if z(34 downto 31) > 4 then
                z(34 downto 31) := z(34 downto 31) + 3;
            end if;
            -- Centenas (4 bits).
            if z(38 downto 35) > 4 then
                z(38 downto 35) := z(38 downto 35) + 3;
            end if;
            -- unidades de millar (4 bits).
            if z(42 downto 39) > 4 then
                z(42 downto 39) := z(42 downto 39) + 3;
            end if;
```

```
379             -- decenas de millar (4 bits).
380             if z(46 downto 43) > 4 then
381                 z(46 downto 43) := z(46 downto 43) + 3;
382             end if;
383             -- centenas de millar (4 bits).
384             if z(50 downto 47) > 4 then
385                 z(50 downto 47) := z(50 downto 47) + 3;
386             end if;
387             -- unidades de millon (4 bits).
388             if z(54 downto 51) > 4 then
389                 z(54 downto 51) := z(54 downto 51) + 3;
390             end if;
391             -- decenas de millon (3 bits).
392             if z(58 downto 55) > 4 then
393                 z(58 downto 55) := z(58 downto 55) + 3;
394             end if;
395
396             -- Corrimiento a la izquierda.
397             z(58 downto 1) := z(57 downto 0);
398         end loop;
399         -- Pasando datos de variable Z, correspondiente a BCD.
400         resultadobcd <= z(58 downto 27);
401     end if;
402 end process bcd;

403
404 led(1 downto 0)<=estado;
405 led(3 downto 2)<=operacion;
406 led(15 downto 12)<=tecla;
407
408 --multiplos: process (acumulador,numero)
409 --variable cargador,valor1,valorloop: integer range 0 to 99999999;
410 --begin
411 --valor1 := CONV_INTEGER(acumulador);
412 --cargador:=CONV_INTEGER(acumulador);
413 --valorloop := CONV_INTEGER(numero);
414
415 --for i in valorloop-1 downto 0 loop
416 --  cargador:=cargador*valor1;
417 --end loop;
418 ----valorfinal<= CONV_STD_LOGIC_VECTOR(cargador,27);
419 --end process multiplos;
420

421 --raices: process (acumulador,numero)
422 --variable comprobador,multiplo,valorraiz,valor: integer range 0 to 99999999;
423 --variable salida: STD_LOGIC := '0';
424 --begin
425 --valorraiz := CONV_INTEGER(acumulador);
426 --valor:= CONV_INTEGER(numero);
427 --for j in valor/2 downto 0 loop
428 --    if salida='0' then
429 --        multiplo:=j;
430 --        comprobador:=j;
431 --        for i in valorraiz-1 downto 0 loop
432 --          comprobador:=comprobador*multiplo;
433 --        end loop;
434
435 --        if comprobador=valor then
436 --            salida:='1';
437 --        end if;
438 --    end if;
439 --end loop;
440 ----raizfinal<= CONV_STD_LOGIC_VECTOR(comprobador,27);
441 --end process raices;
442
443 end Behavioral;
```

Tabla 6.1

6.2. Sub-módulo “GenCLK”.

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  -- Uncomment the following library declaration if using
26  -- arithmetic functions with Signed or Unsigned values
27  --use IEEE.NUMERIC_STD.ALL;
28
29  -- Uncomment the following library declaration if instantiating
30  -- any Xilinx leaf cells in this code.
31  --library UNISIM;
32  --use UNISIM.VComponents.all;
33
34  entity GenCLK is
35    Port (
36         clkin: in STD_LOGIC;
37         clkout: out STD_LOGIC
38   );
39  end GenCLK;
40
41  architecture Behavioral of GenCLK is
42  signal clkdiv : STD_LOGIC;
43  begin
44  -- divide 100MHz clock to 1kHz
45  div1kHz: process(clkin)
46  variable intCnt: integer range 0 to 50000-1;
47  begin
48      if rising_edge(clkin) then
49          if intCnt=50000-1 then
50              intCnt:=0;
51              clkdiv<=not clkdiv;
52          else
53              intCnt:=intCnt+1;
54          end if;
55      end if;
56  end process div1kHz;
57
58  clkout<=clkdiv;
59
60  end Behavioral;
```

Tabla 6.2

6.3. Sub-módulo “tec4x4”.

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  -- Uncomment the following library declaration if using
26  -- arithmetic functions with Signed or Unsigned values
27  --use IEEE.NUMERIC_STD.ALL;
28
29  -- Uncomment the following library declaration if instantiating
30  -- any Xilinx leaf cells in this code.
31  --library UNISIM;
32  --use UNISIM.VComponents.all;
33
34  entity tec4x4 is
35    Port (
36          COL : in STD_LOGIC_VECTOR (3 downto 0);
37          FIL : out STD_LOGIC_VECTOR (3 downto 0);
38          tecla : out STD_LOGIC_VECTOR (3 downto 0);
39          pulsada : out STD_LOGIC;
40          clk: in STD_LOGIC
41     );
42  end tec4x4;
43
44  architecture Behavioral of tec4x4 is
45
46  signal soltada : STD_LOGIC :='1';
47  signal puls : STD_LOGIC :='0';
48  signal fila : integer range 0 to 3 :=0;
49
50  begin
51
52  multiplexacion_teclado:process(clk)
53  variable modo : STD_LOGIC :='1';     -- 0 => hexadecimal, 1 => calculadora
54  begin
55      if rising_edge(clk) then
56          if puls='1' then
57              puls<='0';
58              pulsada<='1';
59          end if;
60          if soltada='1' then
61              if COL="1110" or COL="1101" or COL="1011" or COL="0111" then
62                  puls<='1';
63                  soltada<='0';
64                  if modo='0' then
```

```
64                if modo='0' then
65                    if COL="1110" then
66                        case fila is
67                        when 3 =>
68                            tecla<="1100";
69                        when 2 =>
70                            tecla<="1000";
71                        when 1 =>
72                            tecla<="0100";
73                        when others=>
74                            tecla<="0000";
75                        end case;
76                    elsif COL="1101" then
77                        case fila is
78                        when 3 =>
79                            tecla<="1101";
80                        when 2 =>
81                            tecla<="1001";
82                        when 1 =>
83                            tecla<="0101";
84                        when others=>
85                            tecla<="0001";
86                        end case;
87                    elsif COL="1011" then
88                        case fila is
89                        when 3 =>
90                            tecla<="1110";
91                        when 2 =>
92                            tecla<="1010";
93                        when 1 =>
94                            tecla<="0110";
95                        when others=>
96                            tecla<="0010";
97                        end case;
98                    elsif COL="0111" then
99                        case fila is
100                       when 3 =>
101                           tecla<="1111";
102                       when 2 =>
103                           tecla<="1011";
104                       when 1 =>
105                           tecla<="0111";
106                       when others=>
107                           tecla<="0011";
108                       end case;
109                   end if;
```

```
110                 else
111                     if COL="1110" then
112                         case fila is
113                         when 3 =>
114                             tecla<="1111";-- RETROCESO "C"
115                         when 2 =>
116                             tecla<="0111";-- 7
117                         when 1 =>
118                             tecla<="0100";-- 4
119                         when others=>
120                             tecla<="0001";-- 1
121                         end case;
122                     elsif COL="1101" then
123                         case fila is
124                         when 3 =>
125                             tecla<="0000";-- 0
126                         when 2 =>
127                             tecla<="1000";-- 8
128                         when 1 =>
129                             tecla<="0101";-- 5
130                         when others=>
131                             tecla<="0010";-- 2
132                         end case;
133                     elsif COL="1011" then
134                         case fila is
135                         when 3 =>
136                             tecla<="1110";--IGUAL "="
137                         when 2 =>
138                             tecla<="1001";-- 9
139                         when 1 =>
140                             tecla<="0110";-- 6
141                         when others=>
142                             tecla<="0011";-- 3
143                         end case;
144                     elsif COL="0111" then
145                         case fila is
146                         when 3 =>
147                             tecla<="1101";--DIVISIÓN "/"
148                         when 2 =>
149                             tecla<="1100";--MULTIPLICACION "X"
150                         when 1 =>
151                             tecla<="1011";--RESTA"-"
152                         when others=>
153                             tecla<="1010"; --SUMA"+"
154                         end case;
155                     end if;
156                 end if;
157             end if;
158         end if;
```

```
159         if COL="1111" then
160             soltada<='1';
161             puls<='0';
162             pulsada<='0';
163             case fila is
164             when 0 =>
165                  fila<=1;
166                  FIL<="ZZ0Z";
167             when 1=>
168                  fila<=2;
169                  FIL<="Z0ZZ";
170             when 2 =>
171                  fila<=3;
172                  FIL<="0ZZZ";
173             when others=>
174                  fila<=0;
175                  FIL<="ZZZ0";
176             end case;
177         end if;
178     end if;
179 end process multiplexacion_teclado;
180 
181 end Behavioral;
```

Tabla 6.3

6.4. Sub-módulo “disp8x7seg”.

```
24  library IEEE;
25  use IEEE.STD_LOGIC_1164.ALL;
26
27  -- Uncomment the following library declaration if using
28  -- arithmetic functions with Signed or Unsigned values
29  --use IEEE.NUMERIC_STD.ALL;
30
31  -- Uncomment the following library declaration if instantiating
32  -- any Xilinx leaf cells in this code.
33  --library UNISIM;
34  --use UNISIM.VComponents.all;
35
36  entity disp8x7seg is
37      Port(
38      cat : out STD_LOGIC_VECTOR (7 downto 0);
39      seg : out STD_LOGIC_VECTOR (7 downto 0);
40      d0 : in STD_LOGIC_VECTOR (4 downto 0);
41      d1 : in STD_LOGIC_VECTOR (4 downto 0);
42      d2 : in STD_LOGIC_VECTOR (4 downto 0);
43      d3 : in STD_LOGIC_VECTOR (4 downto 0);
44      d4 : in STD_LOGIC_VECTOR (4 downto 0);
45      d5 : in STD_LOGIC_VECTOR (4 downto 0);
46      d6 : in STD_LOGIC_VECTOR (4 downto 0);
47      d7 : in STD_LOGIC_VECTOR (4 downto 0);
48      clk: in STD_LOGIC
49      );
50  end disp8x7seg;
51
52  architecture Behavioral of disp8x7seg is
53
54  signal digito : STD_LOGIC_VECTOR (4 downto 0);
55  signal catodos : STD_LOGIC_VECTOR (7 downto 0);
56
57  begin
58
```

```
multiplexacion_catodo:process(clk)
begin
    if rising_edge(clk) then
        case catodos is
        when "00000001"=>
            catodos<="00000010";
        when "00000010"=>
            catodos<="00000100";
        when "00000100"=>
            catodos<="00001000";
        when "00001000"=>
            catodos<="00010000";
        when "00010000"=>
            catodos<="00100000";
        when "00100000"=>
            catodos<="01000000";
        when "01000000"=>
            catodos<="10000000";
        when others=>
            catodos<="00000001";
        end case;
    end if;
end process multiplexacion_catodo;

cat<=catodos;


-- Colocación de los digitos en los displays
with catodos select digito<=
    d7 when "10000000",
    d6 when "01000000",
    d5 when "00100000",
    d4 when "00010000",
    d3 when "00001000",
    d2 when "00000100",
    d1 when "00000010",
    d0 when others;

-- Conversión de los digitos a 7 segmentos
with digito select seg<=
    "00111111" when "00000",  --0
    "00000110" when "00001",  --1
    "01011011" when "00010",  --2
    "01001111" when "00011",  --3
    "01100110" when "00100",  --4
    "01101101" when "00101",  --5
    "01111101" when "00110",  --6
    "00000111" when "00111",  --7
    "01111111" when "01000",  --8
    "01101111" when "01001",  --9
    "00000000" when "01010",      -- (+)' '
    "01000000" when "01011",      -- (-)'-'
    "10000000" when "01100",      -- (X)'.'
    "00000000" when "01101",      -- (/)' '
    "01001000" when "01110",      -- (=)'calculo'
--    "01110001" when "01111",      -- (C)'limpieza'
-- Los próximos cuatro podrían ser implementados para más operaciones
--    "01110111" when "10000",      -- (A)'A'
--    "01111100" when "10001",      -- (B)'b'
--    "00111001" when "10010",      -- (C)'C'
--    "01011110" when "10011",      -- (D)'d'
    "00000000" when others;         -- ' '

end Behavioral;
```

Tabla 6.4

7. BIBLIOGRAFÍA

Smith, G. (2010). FPGAs 101: Everything You Need to Know to Get Started.

http://www.estadofinito.com/binario-bcd-7seg/

https://reference.digilentinc.com/_media/basys3:basys3_rm.pdf

https://reference.digilentinc.com/reference/programmable-logic/basys-3/start