ALU LOGARÍTMICA

T. G. 0432

MANUEL ALBERTO CARRERA OSORIO
EFRÉN OCTAVIO LÓPEZ URIBE
JUAN DIEGO NARANJO LÓPEZ

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
CARRERA DE INGENIERÍA ELECTRÓNICA
BOGOTÁ DC
2005

ALU LOGARÍTMICA

T. G. 0432

MANUEL ALBERTO CARRERA OSORIO
EFRÉN OCTAVIO LÓPEZ URIBE
JUAN DIEGO NARANJO LÓPEZ

Informe Final

Directores
Alejandra María González Correal
Ingeniera Electrónica
Francisco Viveros Moreno
Ingeniero Electrónico

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
CARRERA DE INGENIERÍA ELECTRÓNICA
BOGOTÁ DC
2005

PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD DE INGENIERÍA

CARRERA DE INGENIERÍA ELECTRÓNICA

RECTOR MAGNIFICO:	R.P. GERARDO REMOLINA S.J.
DECANO ACADÉMICO:	Ing. FRANCISCO JAVIER REBOLLEDO MUÑOZ
DECANO DEL MEDIO UNIVERSITARIO:	R.P. ANTONIO JOSÉ SARMIENTO NOVA S.J.
DIRECTOR DE CARRERA:	Ing. JUAN CARLOS GIRALDO CARVAJAL
DIRECTOR DEL PROYECTO:	Ing. ALEJANDRA MARIA GONZÁLEZ
	Ing. FRANCISCO VIVEROS

ARTICULO 23 DE LA RESOLUCIÓN No. 13 DE JUNIO DE 1946

"La universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado.

Sólo velará porque no se publique nada contrario al dogma y la moral católica y porque los trabajos no contengan ataques o polémicas puramente personales. Antes bien, que se vea en ellos el anhelo de buscar la verdad y la justicia".

CLAVELES ROJOS

De mis cabras guardar, volví al cortijo
Saltando entre peñas y matojos
cuando mi moza me llamó y me dijo.
Que del amo de la hacienda vino el hijo
y ella quería lucir claveles rojos.

Al monte volví como a la guerra,
con valentía, ceguedad y arrojo.
Porque al pasar con mis cabras y mi perra, vi allá en lo alto de la sierra
Una mata de claveles rojos.

Pero al llegar a lo alto de la sierra
y entre las hojas de una gran chumbera
vi a la madre clavelera, que lloraba
por sus hijos los claveles rojos.
Sus hijos le robaron?...Quién sería?..

Al cortijo volví lleno de hinojos...
triste porque llevar ya no podía
lo que mi moza amorosa me pedía.
Donde encontrar claveles rojos...?

De pronto me acordé que allá en la ermita
los mozos del cortijo de los hinojos llevaban a la virgen la ciervita
unos ramos de margaritas, rosas,
pensamientos y claveles rojos.

Corriendo iba como un gamo
como un galgo saltando los rastrojos.
En el camino me encontré a mi amo
Quien me llamó y me dijo ¡zagal véndeme el ramo!
como venderle los claveles rojos?

Me lo perdona el amo...hube de decirle.
Fatigado, mal herido y cojo
Quedó este pobre cabrerillo humilde,
para llevarle a su Matilde
este ramillete de claveles rojos.

¡Para tu Matilde haz dicho!..
¡Anda y reposa!..Que si por ella te has quedado cojo..
vete a curar a tu choza...
Porque... ¡óyelo bien! solo a esa moza,
yo he de llevar claveles rojos.

Pasó por mi no se cosa mala...
El señorito me clavó los ojos,...
se lanzó hacia mi como una bala, y
me desalojó el ramo de claveles rojos.

Al cortijo volví...Salté tres bancos
Bañados en lágrimas iban mis ojos
Allí vi una mata de claveles blancos.
los tomé y me los llevé al campo
para transformarlos en claveles rojos.

De ideas malas llevaba un enjambre
Al ver a mi amo, una nube cegó mis ojos
mi puñal en su pecho hundí con hambre
los claveles blancos empapé en su sangre
y a mi moza le llevé claveles rojos.

Autor desconocido

“A mis Padres, a mis hermanos, a Pipe, y a todos aquellos que creyeron en mi...”

Juan Diego Naranjo L.

***

Un maestro de la sabiduría paseaba por un bosque con su fiel discípulo, cuando vio a lo lejos un sitio de apariencia pobre y decidió hacer una breve visita al lugar.

Durante la caminata comentó al aprendiz sobre la importancia de las visitas; también de conocer personas y de las oportunidades de aprendizaje que tenemos de estas experiencias.

Llegando al lugar constató la pobreza del sitio; los habitantes, una pareja y sus tres hijos, la casa de madera, sus ropas sucias y rasgadas y sin calzado. Se aproximó entonces al señor, aparentemente el padre de familia y le pregunto: ¿En este lugar no existen posibilidades de trabajo para sobrevivir?

El señor calmadamente respondió: "Amigo mío, nosotros tenemos una vaquita que nos da varios litros de leche todos los días. Una parte del producto la vendemos o la cambiamos por otros géneros alimenticios en la ciudad vecina, y con la otra parte producimos queso, manteca, etc., para nuestro consumo, y así vamos sobreviviendo".

El sabio le agradeció la información, contemplo el lugar por un momento, y luego se despidió.

En el medio del camino se volvió hacia su fiel discípulo y le ordenó: "Busca la vaquita, llévala al precipicio y empújala al barranco".

El joven espantado miro al maestro y le cuestionó sobre el hecho que la vaquita era el único medio de subsistencia de aquella familia. Más, como percibió el silencio absoluto del maestro, fue a cumplir la orden.

Empuja la vaquita por el precipicio y la vio morir.
Aquella escena quedó grabada en la memoria del joven durante muchos años.

Un bello día, este joven agobiado por la culpa resolvió abandonar todo lo aprendido y regresar a aquel lugar, contarle todo a la familia, pedir perdón y ayudarlos. Así lo hizo y, a medida que se aproximaba al lugar vio todo muy bonito, con árboles floridos, todo habitado, un auto de un garaje de una linda casa, y algunos niños jugando.

Aceleró el paso y pregunto por la familia que vivía allí hacía unos cuatro años.

El señor respondió que allí seguían viviendo. Espantado el joven entro corriendo a la casa y confirmó que era la misma familia que visitara con su maestro. Elogió el lugar y le pregunto al señor: "¿Cómo hizo para mejorar este lugar y cambiar la vida?".

El señor entusiasmado le respondió: "Teníamos una vaquita que cayó por el precipicio y murió; de ahí en adelante nos vimos en la necesidad de hacer otras cosas y desarrollar otras habilidades que no sabíamos que teníamos; fue de esa manera como alcanzamos el éxito que sus ojos vislumbran ahora".

Todos nosotros tenemos una vaquita que nos proporciona alguna cosa básica para nuestra supervivencia, la cual se convierte en rutinaria. Nos hace dependientes y el mundo se reduce a lo que esta vaquita nos brinda.

Autor Anónimo

**** Este trabajo de grado se lo dedico al esfuerzo de mi papá y mí mamá, a mis hermanos por su apoyo, a mi abuela Yolanda por su entrega, y a Eduardo y Nancy por sus consejos ****

Manuel Alberto Carrera

Este trabajo es especialmente dedicado a Teresita, quien con su paciencia, apoyo, consejos, y monumental esfuerzo ha logrado darme las herramientas para labrar mi destino y enfrentar la vida. Madre, has sido un excelente ejemplo de tenacidad y honestidad, todo el trabajo que has realizado para soportar esta hermosa familia y formarnos como personas a mi y a mi hermana, quedará grabado en mi memoria para estar inmensamente agradecido contigo y con la vida por haberme otorgado el honor de ser tu hijo.

A Ivonne, quien también ha sido un apoyo incondicional en los buenos y en los malos momentos y más que una hermana, he encontrado en ella una amiga.

A Octavio, que de una manera u otra me motivó a seguir adelante. Padre, aunque en algunas circunstancias de la vida tengamos dificultades, siempre seré tu hijo y estaré presente para brindarte mi apoyo.

A mi familia, que es la parte fundamental de mi vida.…

Efrén Octavio López

AGRADECIMIENTOS

Queremos agradecer a nuestros directores de Trabajo de Grado, Alejandra y Francisco, por su colaboración y comprensión.

Agradecemos a nuestro asesor, Alejandro Forero.

También agradecemos a John Freddy Sánchez, a Caliche a Marlon y a Leo y técnicos de laboratorio.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

Este trabajo de grado es el desarrollo de una Unidad Aritmética Lógica en base logarítmica, en otras palabras es un sistema digital combinatorio que realiza operaciones aritméticas entre dos datos, donde cada dato es representado por su logaritmo en base dos. De esta manera se facilitan algunas operaciones complejas, como ejemplo, para realizar una multiplicación, por propiedad de los logaritmos, se deben sumar los datos.

El formato de cada número está conformado por una trama de 16 bits: 14 bits que conforman un número en complemento a dos, divididos en 5 bits para parte entera y 9 bits para la parte fraccionaria, a estos 14 bits se le agregan 2 bits para representar el signo y números que no tienen representación logarítmica, como el cero, indeterminaciones e indefiniciones.

Las operaciones que puede realizar la ALU logarítmica son: multiplicación, división, suma, resta potencia al cuadrado y raíz cuadrada.

La implementación de la ALU logarítmica está constituida por un FPGA ACEX1K100 de Altera y dos memorias EPROM AM27C256, para facilitar la comprobación del sistema y observar su comportamiento se desarrollo una tarjeta interfaz, esta tarjeta está constituida por un PIC18F8720 de Microchip, y un circuito integrado MAX232, para comunicación serial con PC.

La tarjeta de interfaz es controlada desde un PC por una aplicación programada en lenguaje JAVA, esta aplicación se llama “controlador.java” y permite suministrar operaciones a la ALU logarítmica y analizar el resultado de esta operación.

INTRODUCCIÓN

En todo sistema computacional, existen unidades funcionales encargadas de realizar las operaciones aritméticas, éstas son llamadas unidades de aritmética y lógica (ALU). Comúnmente trabajan numeración de punto fijo o punto flotante, utilizando la representación binaria de los números de base decimal. El funcionamiento de una unidad aritmética convencional se basa en la utilización de sumadores binarios, que están compuestos por compuertas que cumplen funciones lógicas sencillas.

Las operaciones que se pueden desarrollar en las ALU's descritas anteriormente son sumas, con la representación adecuada de los dígitos binarios (en complemento a dos), con la misma unidad sumadora se pueden implementar las restas. Para el desarrollo de multiplicaciones es necesaria la implementación de un algoritmo y de hardware adicional (incrementando el retardo de esta operación). Caso similar ocurre con la implementación de las divisiones. De esta manera al utilizar estas unidades en la solución de sistemas complejos que involucran un uso mayor que el usual de instrucciones aritméticas, tales como filtros digitales y transformadas, el tiempo requerido para la ejecución de los algoritmos se eleva.

El sistema de numeración con base logarítmica permite realizar multiplicaciones, divisiones y raíces de una manera más simple que un sistema de numeración de punto fijo pero con la dificultad que la realización de operaciones de suma y resta, se vuelve más compleja.

Un beneficio obtenido al utilizar un sistema de numeración con base logarítmica, es que el rango numérico puede incrementarse en comparación con el sistema de números en punto fijo, al igual que el punto flotante. Se debe encontrar el número de bits de palabra adecuado, el cual permita tener un intervalo numérico que produzca resultados acordes a la aplicación que se le esté dando al sistema.

OBJETIVOS

Objetivo general

- Desarrollar e implementar la arquitectura de una unidad aritmética lógica en base logarítmica en un dispositivo de lógica programable.

Objetivo específicos

- Determinar el número de bits que se necesitan para la representación de los números en base logarítmica.

- Describir la ALU en lenguaje VHDL.

- Implementar en un FPGA la arquitectura de la ALU.

- Construir una tarjeta de desarrollo que programe el dispositivo FPGA utilizado y que permita al usuario interactuar con la ALU de forma que pueda realizar las operaciones aritméticas de suma, resta, multiplicación y división.

1. MARCO TEÓRICO

1.1 NÚMEROS EN BASE LOGARÍTMICA

Para la representación de un número “A” en una base numérica logarítmica, se destina un bit del formato para el signo del número, este bit es S_a, el cual toma el valor de cero cuando el número es positivo y uno cuando es negativo, m bits que representan (en complemento a dos) el logaritmo en base dos de “A”, llamado “RA”, estos m bits están compuestos por k+l bits, de los cuales k es la parte entera de “RA” y l es la precisión o parte fraccionaria. Es decir:

$$A = (-1)^{SA} \cdot 2^{RA} \quad (1)$$

Donde:

$$R_a = r_{k-1} r_{k-2} \ldots r_0 , r_1 r_2 \ldots r_l \quad (2)$$

Dado que “RA” es el logaritmo del número, existe un conflicto cuando el número “A” es cero, por esto se agrega otro bit que determina si el número es cero o no, este bit no solo determina si el número es cero sino que permite representar otros números. Esto da un total de *m*+2 bits como formato de la representación numérica logarítmica.

Para obtener el número C que es la multiplicación de un número A por uno B, los cuales se encuentran representados en el formato descrito anteriormente, se debe realizar el siguiente procedimiento:

$$S_c = S_a \oplus S_b \quad (3)$$

$$R_c = R_a + R_b \quad (4)$$

El paso descrito en la ecuación *(3)* es para obtener el signo del número resultante, y el paso descrito por la ecuación *(4)* es para obtener la multiplicación de los dos números en su representación logarítmica. Para realizar la división, se realiza el primer paso igual que en la ecuación *(3)*, pero en el paso *(4)* se realizaría la diferencia entre el divisor y el dividendo para obtener el resultado de la división.

Ahora, para realizar las operaciones de suma o de resta entre dos números *A* y *B*, y obtener un dato *C,* tenemos que:

$$C = A \pm B \quad (5)$$

Lo que se debe hacer para el caso que |A| > |B|:

$$C = A \cdot \left(1 \pm \frac{B}{A}\right) \Rightarrow R_c = \log_2\left(A \cdot \left(1 \pm \frac{B}{A}\right)\right) \quad (6)$$

Por propiedades de los logaritmos se tiene

$$R_c = \log_2(A) + \log_2\left(1 \pm \frac{B}{A}\right) \quad (7)$$

De otro modo

$$R_c = R_a + \Phi(R_a - R_b) \quad (8)$$

Donde

$$\Phi(R_a - R_b) = \log_2(1 \pm 2^{-(R_a - R_b)}) \quad (9)$$

Para el caso en que |B| > |A|, en el procedimiento anterior se deben intercambiar el dato A por el dato B. Esto se hace con el fin de buscar simetría al realizar las operaciones, es decir que las operaciones de suma y resta sean conmutativas.

Para realizar la operación de potencia al cuadrado se obtiene según las ecuaciones (7) a la (10).

$$C = A^2 = ((-1)^{SA} \cdot 2^{RA})^2 \quad (7)$$

$$C = (-1)^{2 \cdot SA} \cdot 2^{2 \cdot RA} \quad (8)$$

$$SC = 0 \quad (9)$$

$$RC = 2 \cdot RA \quad (10)$$

El dato de entrada *RA* se debe multiplicar por dos para obtener la potencia del número, dado que *RA* es un número en complemento a dos, la operación de potencia se reduce a un corrimiento de los bits a la izquierda. Además el signo del resultado *SC,* siempre será positiv*o.*

De forma similar la operación de raíz al cuadrado se puede deducir de las ecuaciones (11) a la (14).

$$C = \sqrt{A} = ((-1)^{SA} \cdot 2^{RA})^{1/2} \quad (11)$$

$$C = (-1)^{SA/2} \cdot 2^{RA/2} \quad (12)$$

$$SC = SA/2 \quad (13)$$

$$RC = RA/2 \quad (14)$$

Así la raíz cuadrada es reducida a un corrimiento a la derecha del dato *RA*. En esta operación el signo no puede calcularse con alguna operación, debe revisarse el signo del dato de entrada, si es positivo o negativo, para determinar si el resultado es real o imaginario.

1.2 NÚMEROS BINARIOS EN PUNTO FLOTANTE

La representación de números en punto flotante de un número *n* consiste en dos partes, la mantisa M y el exponente E. El número flotante F representado por el par (*M, E*) tiene el valor

$$F = M \cdot \beta^{E} \quad (10)$$

En donde β es la base del exponente.

El intervalo de la representación en punto flotante es mucho mayor comparado con la representación de punto fijo, la desventaja de este sistema de representación es su menor precisión comparada con la de punto fijo.

La menor precisión se evidencia cuando algún número real se encuentre entre dos números consecutivos de punto flotante, en cuyo caso éste es aproximado a cualquiera de esos dos números. Por lo tanto una distancia muy grande entre dos números consecutivos resulta en una representación de baja precisión.

El exponente es usualmente un entero con sesgo, mientras la mantisa es representada en alguna de las siguientes dos formas: la primera como fracción pura, y la segunda como un número en el intervalo [1,2) (para $\beta = 2$).

El formato de numeración en punto flotante para un número *n* consiste en: un bit *S* para el signo, *e* bits para el exponente *E*, y *m* bits sin signo para la mantisa *M*, lo que satisface *m* + *e* + 1 = *n,* tal como se visualiza en la figura 1.

S	Exponente *E*	Mantisa *M*

Figura 1. Formato de número en punto flotante

El valor del número de punto flotante queda determinado por las variables (*S, E, M),* con lo que nos conduce a:

$$F = (-1)^S \cdot M \cdot \beta^E \quad (11)$$

En donde *F* es positivo o negativo dependiendo del valor de *S*. El máximo valor de *M* es $M_{max} = 1 - ulp$, en donde $ulp = 2^{-m}$. La base β es restringida a una potencia entera de una raíz *r* = 2, entonces $\beta = 2^k$ en donde *k* = 1, 2,.... Esta base se usa debido a que es un método simple para decrementar la mantisa e incrementar el exponente (o viceversa) al mismo tiempo, de manera que el número en punto flotante no sufra ninguna alteración.
El menor incremento en el exponente *E* es 1, lo que lleva como resultado a la siguiente ecuación:

$$M \cdot \beta^E = \left(\frac{M}{\beta}\right) \cdot \beta^{E+1} \quad (12)$$

La expresión (M / β) nos lleva a realizar un corrimiento hacia la derecha, pero debido a que $\beta = 2$ el corrimiento es de una posición a la derecha, lo cual se ve compensado por la suma de 1. Pueden existir diferentes maneras de representar un número, pero puede llegar a darse el caso en el que se realice un corrimiento hacia la derecha y como resultado se pierda el dígito menos significativo.

De todas las representaciones posibles se prefiere la que posea en la mantisa un 1 en la posición del bit más significativa, que se llama forma "normalizada". Por ejemplo, la forma normalizada de $0.00000110 \cdot 16^{101}$ es $0.01100000 \cdot 16^{100}$.

Cuando se trabaja de manera normalizada el rango de la mantisa se encuentra entre [0, 1-*ulp*]. Los valores mínimos y máximos permitidos son:

$$M_{\min} = \frac{1}{\beta} \qquad M_{\max} = 1 - ulp \quad (13)$$

En este rango no se incluye el valor de cero, pero es necesaria una representación para éste. Una posible representación para cero es la mantisa $M = 0$, y cualquier valor para el exponente E, pero se prefiere $E = 0$. La representación de cero en punto flotante es idéntica a la representación de cero en punto fijo.

Antes del año 1980 cada computador tenía su propia forma de representar los números en punto flotante, lo que hacia difícil el transporte de la información entre un computador y otro. Pero a partir de ese año se solucionó este problema por medio del estándar IEEE 754.

El estándar de punto flotante IEEE 754 posee cuatro formatos de numeración. El primero está formulado para un formato de precisión simple de 32 bits y el segundo para un formato de doble precisión de 64 bits, los otros dos son formatos extendidos para operaciones parciales, el primero, es un formato simple extendido de por lo menos 44 bits y el segundo, un formato doble extendido de por lo menos 80 bits.

En el formato de precisión simple, la base del exponente que utiliza es dos, debido a la facilidad para el manejo de los números, ya que al dividir por dos es equivalente a realizar un corrimiento hacia la derecha, la longitud del exponente es 8 bits. La trama en bits del formato se observa según la figura 2.

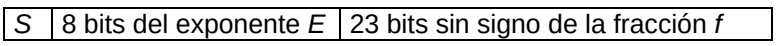


Figura 2. Formato de número en punto flotante IEEE 754

1.3 OPERACIONES EN PUNTO FLOTANTE

Se tienen dos números de entrada $F_1 = (-1)^{S_1} \cdot M_1 \cdot \beta^{E_1}$ y $F_2 = (-1)^{S_2} \cdot M_2 \cdot \beta^{E_2}$, se necesita calcular el resultado de las operaciones aritméticas básicas que den como resultado $F_3 = (-1)^{S_3} \cdot M_3 \cdot \beta^{E_3}$.

Las mantisas de los dos datos se multiplican como si fueran números en punto fijo, mientras los exponentes se suman. Estas dos operaciones se realizan en paralelo; para determinar el signo del resultado se realiza una **XOR** entre los signos de los datos como se expuso anteriormente. La división es muy parecida a la multiplicación, porque se dividen las mantisas y se restan los exponentes en lugar de multiplicar y sumar respectivamente.

La adición y la sustracción son un caso especial porque para su operación es necesario que los exponentes de los dos datos sean iguales. Solo cuando $E_1 = E_2$ se puede factorizar el término β^{E_1} y las mantisas M_1 y M_2 se pueden operar, ya sea una adición o sustracción. Para lograr que los dos exponentes sean iguales se hace un corrimiento hacia la derecha de la mantisa más pequeña, incrementado su exponente al mismo tiempo, hasta que se iguale con el otro.

2. ESPECIFICACIONES

El diagrama en bloques del sistema global se observa en la figura 3.

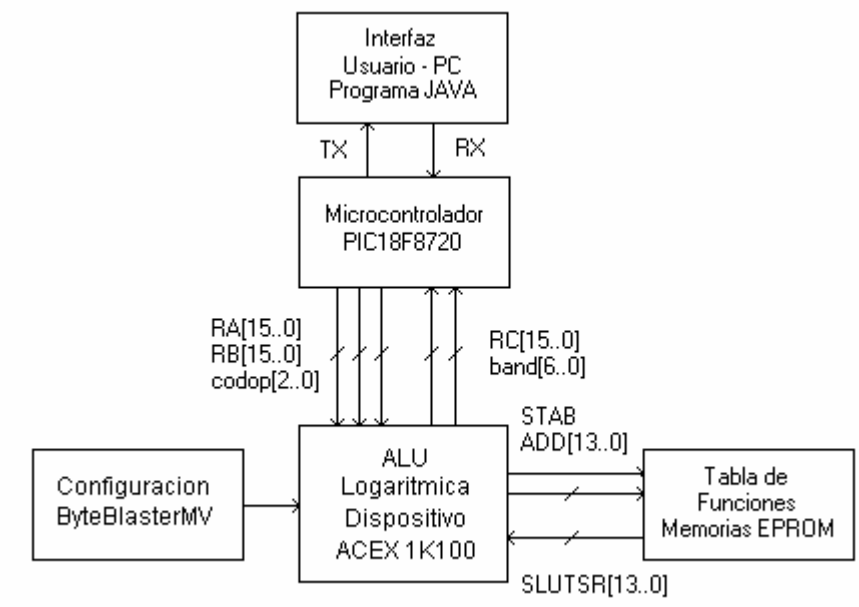


Figura 3. Diagrama en bloques del sistema global

2.1 ESPECIFICACIONES DE HARDWARE

2.1.1 Unidad aritmética lógica en base logarítmica. La ALU logarítmica soporta seis operaciones. Estas operaciones son: suma, resta, multiplicación, división, raíz cuadrada y potencia de dos. La unidad recibe dos números cada uno compuesto de 16 bits, como resultado arroja otro número también de 16 bits.

El diagrama de bloques de la unidad aritmética lógica se muestra en la figura 4.

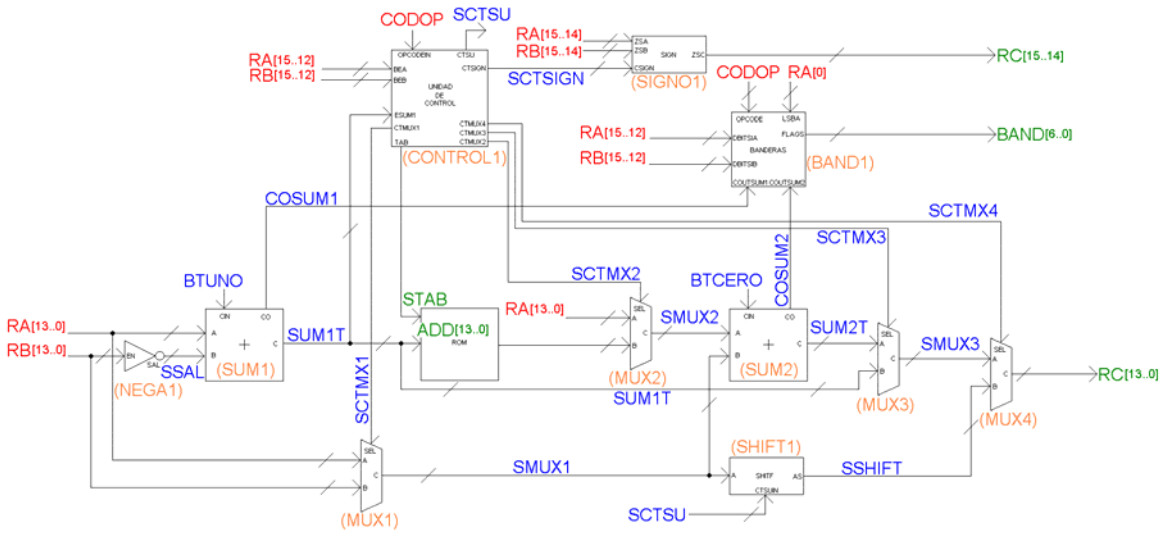


Figura 4. Diagrama en bloques de la ALU logarítmica

2.1.1.1 Formato de número. Los números son representados por 16 bits, de los cuales uno corresponde al bit de cero, otro bit es el bit de signo, los 14 bits restantes se reparten en 5 para la parte entera y 9 para la parte fraccionaria. El formato se visualiza de la siguiente manera:

$$Z_a S_a \;\; r_4 r_3 r_2 r_1 r_0 \;\; r_{-1} r_{-2} r_{-3} r_{-4} r_{-5} r_{-5} r_{-6} r_{-7} r_{-8} r_{-9}$$

Los siguientes son ejemplos de números representados en este formato:

-8 01 00011 000000000

El bit de signo es uno, por lo cual el número es negativo, el logaritmo en base dos de ocho es tres, por lo cual la parte fraccionaria es cero y la parte entera es tres.

12 00 00011 100101100

El número es positivo por lo cual el bit de signo es cero, se ve además que el logaritmo en base dos de doce es mayor que 3 y menor que 4 por lo tanto su parte entera es tres y su parte fraccionaria es diferente de cero.

1 00 00000 000000000

El número es positivo entonces el bit de signo es cero. El logaritmo en base dos de uno es cero, la parte entera del número y la parte fraccionaria son cero.

-0.0625 01 11100 000000000

El bit de signo del número anterior muestra que éste es negativo. El número aunque es fraccionario posee un logaritmo en base dos entero, y es igual a –4 por lo tanto la parte entera es –4 en complemento dos, 11100, la parte fraccionaria es cero.

Dado que los números anteriores encuentran representaciones en formato logarítmico, es decir, podemos obtener el logaritmo de su valor absoluto, el bit de cero es cero.

Los números especiales como cero, +infinito, -infinito e indefinido no se pueden representar con su logaritmo por lo tanto poseen representaciones especiales, en formato logarítmico se definen así:

Cero 10 00000 000000000
+ Infinito 10 01111 111111111
- infinito 11 01111 111111111
Indefinido 11 11111 111111111

2.1.1.2 Códigos de operación. La señal de código de operación es proporcionada a la ALU por medio de su puerto de entrada “CODOP”, los códigos están especificados en la tabla 1.

CÓDIGO DE OPERACIÓN CODOP	OPERACIÓN
000	Suma
001	Resta
010	Multiplicación
011	División
101	Potencia
110	Radicación
100 y 111	No especificados

Tabla 1. Códigos de operación

2.1.1.3 Banderas. La señal de salida de esta unidad (y del sistema global) denominada “BAND” se compone de siete (7) bits, a los cuales les corresponde un evento ocurrido en el proceso de computo de los datos, estos eventos especiales están definidos en la tabla 2.

BAND	EVENTO
Bit 0	Desbordamiento de algún sumador
Bit1	Resultado es + infinito
Bit 2	Resultado es –infinito
Bit 3	Resultado es NaN[1]
Bit 4	Cambio de signo en corrimiento
Bit 5	División por cero
Bit 6	Resultado es número imaginario

Tabla 2. Numeración de los bits de banderas.

2.1.2 Microcontrolador. Se utilizó el microcontrolador PIC18F8720 de la compañía Microchip[2], debido al número de I/O que tiene, esto es necesario porque cada bus de datos de la unidad aritmética lógica es de 16 bits.

Las características principales de este microcontrolador se resumen en la tabla 3.

Dispositivo	Memoria de Programa		Memoria de Datos		I/O	10-bit	CCP	MSSP		USART			Max
	Bytes	# Instrucciones de palabra	SRAM (bytes)	EEPROM (bytes)		A/D (ch)		SPI	I^2C		TIMERS 8-bit/16-bit	Ext Bus	Fosc (MHz)
PIC18F8720	128 K	65536	3840	1024	68	16	5	S	S	2	2//3	S	25

Tabla 3. Especificaciones del microcontrolador PIC18F8720 de Microchip

Los recursos utilizados del microcontrolador son los puertos de propósito general y un puerto USART. El microcontrolador, por medio de interrupciones espera una cadena de cinco bytes enviados por puerto serial RS 232 a la USART del microcontrolador, éste ubica la información en los puertos “A”, “B”, “D”, “E” y “F” configurados como salidas hacia la unidad aritmética lógica. La ALU realiza la operación indicada, y el resultado de ésta operación es colocada en los puertos “H”, “J”, “C” y “G” del microcontrolador, éste envía de regreso la información al PC, por puerto serial, organizado en una cadena de tres bytes. Todo el proceso de envío y captura de datos es realizado por el microcontrolador es en forma secuencial, excepto el proceso por medio del cúal la ALU realiza las operaciones. La figura 5 muestra el diagrama de flujo del funcionamiento del programa del microcontrolador.

[1] NaN, acrónimo de la frase en ingles “Not a Number”, que quiere decir número indefinido.
[2] “Microchip PIC 18F8720 Datasheet”. Microchip, 2004.

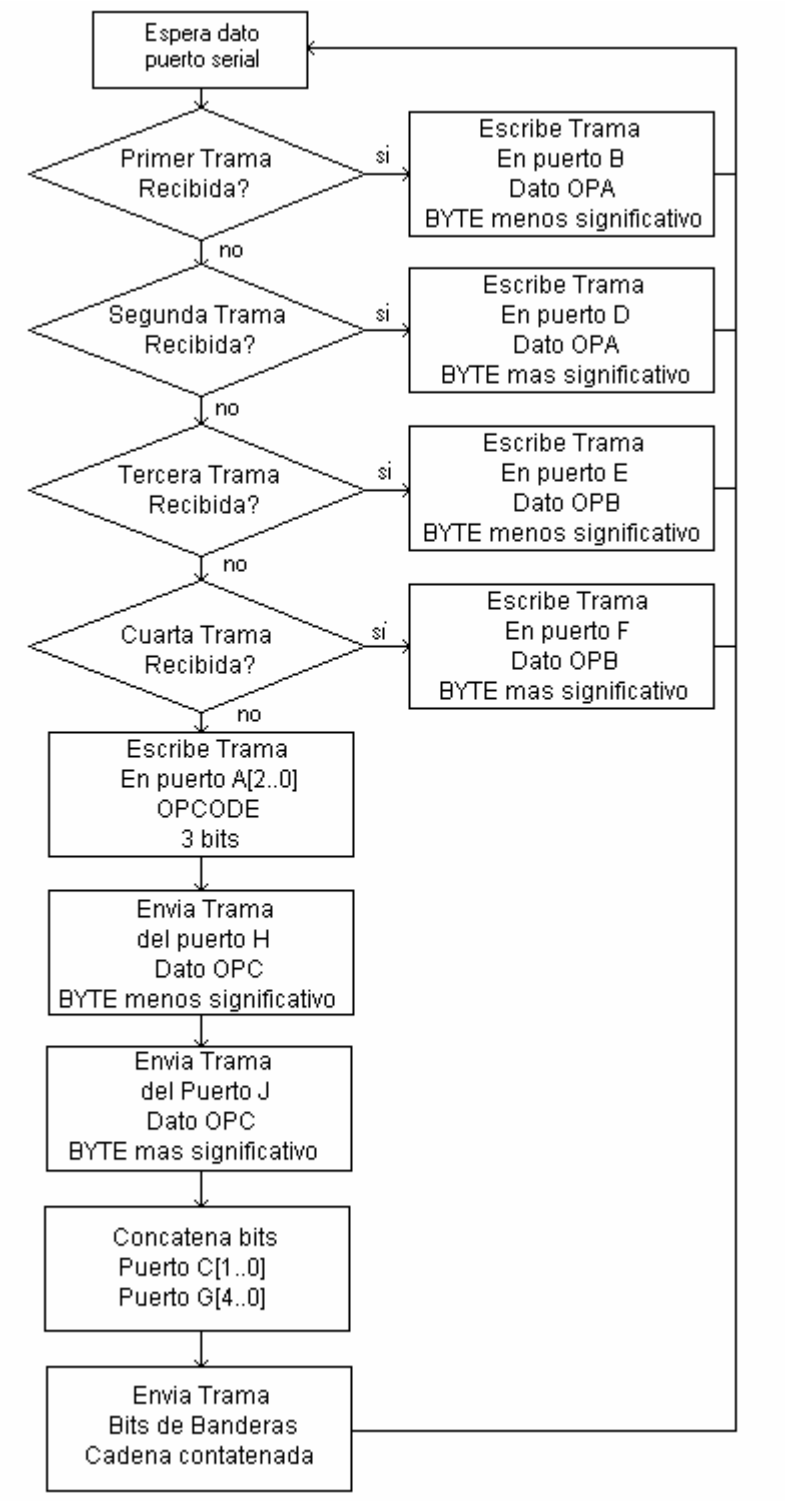


Figura 5. Diagrama de flujo del programa implementado en el microcontrolador

2.1.3 Tarjeta de desarrollo con dispositivo FPGA. Se hace uso de una tarjeta de desarrollo elaborada en el trabajo de grado titulado "Implementación del Procesador Alteric"[3], dado que ésta utiliza un dispositivo FPGA ACEX 1K100 de Altera[4], este proyecto no tiene ninguna relación con Alteric (se usó esta tarjeta de desarrollo por facilidad, ya que se disponía del circuito). Esta tarjeta posee puertos de configuración, puertos de poder y puertos de propósito general. Las características principales de este FPGA se resumen en la tabla 4.

CARACTERÍSTICAS	EP1K100
Compuertas típicas	1'000.000
Compuertas máximas del sistema	257.000
Elementos Lógicos	4.992
Bloque de Arreglos Embebidos (EAB)	12
Total de bits RAM	49.152
Máximo de pines entrada/salida	333

Tabla 4. Características del FPGA ACEX EP1K100-208 de Altera

Como se observa en la tabla 4, las características más importantes que se deben tener en cuenta de un FPGA ACEX1K100, es el número de elementos lógicos y el número de Bloques EAB[5], estos bloques sirven para implementar bloques de memoria, este FPGA en particular posee 12 EABs, en la figura 6 se observan diferentes configuraciones que puede tomar un EAB.

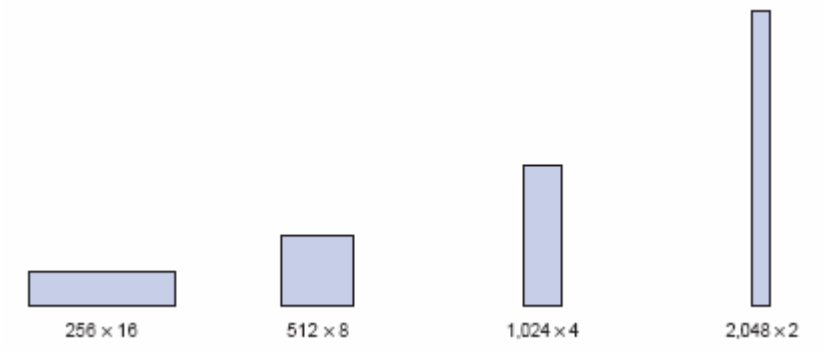


Figura 6. Posibles configuraciones para un EAB de un FPGA ACEX1K

El FPGA ACEX1K100 se organiza en sectores de 12 filas por 52 columnas, cada sector es llamado LAB[6], existen otros sectores especiales donde se encuentran ubicados los 12 bloques EAB, como se muestra en la figura 7.

[3] BELTRAN, Diego; HERRERA, Moisés & MAYOLO, Marco. "Implementación del procesador Alteric". Bogota, 2004. Trabajo de grado (Ingeniero Electrónico). Pontificia Universidad Javeriana. Facultad de Ingeniería. Carrera de Ingeniería Electrónica.
[4] "ACEX 1K, Programmable logic device family". San Jose CA: Altera, 2003.
[5] EAB, acrónimo de la frase en ingles Embeded Array Block, que traduce Bloque de Arreglo Integrado.
[6] LAB, acrónimo de la frase en ingles Logic Array Block, que traduce Bloque de Arreglo Lógico.

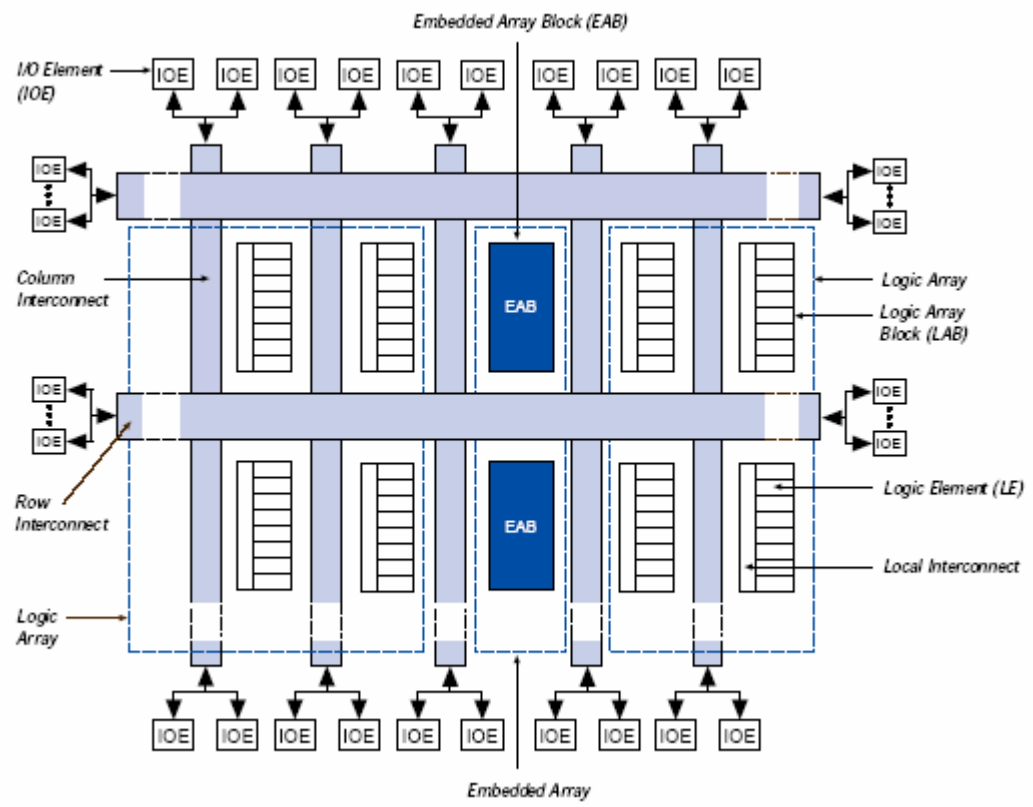


Figura 7. Organización interna de un FPGA ACEX1K

Cada LAB contiene 8 unidades de lógica básica, como se observa en la figura 8, estas unidades lógicas son llamadas Elementos Lógicos, cada elemento lógico posee una LUT[7], este permite implementar una ecuación lógica con cuatro bits de entrada y un bit de salida, posee también un Flip Flop para implementar lógica secuencial y registros. Posee también múltiples recursos de interconexión con otros elementos lógicos, a nivel global o a nivel local con elementos lógicos en el mismo LAB, los recursos de interconexión locales permiten realizar ecuaciones lógicas con entradas de más de cuatro bits o implementación de unidades aritméticas en la figura 9 se observa un elemento lógico.

[7] LUT, acrónimo de la frase en ingles Look Up Table, que traduce Tabla de Función.

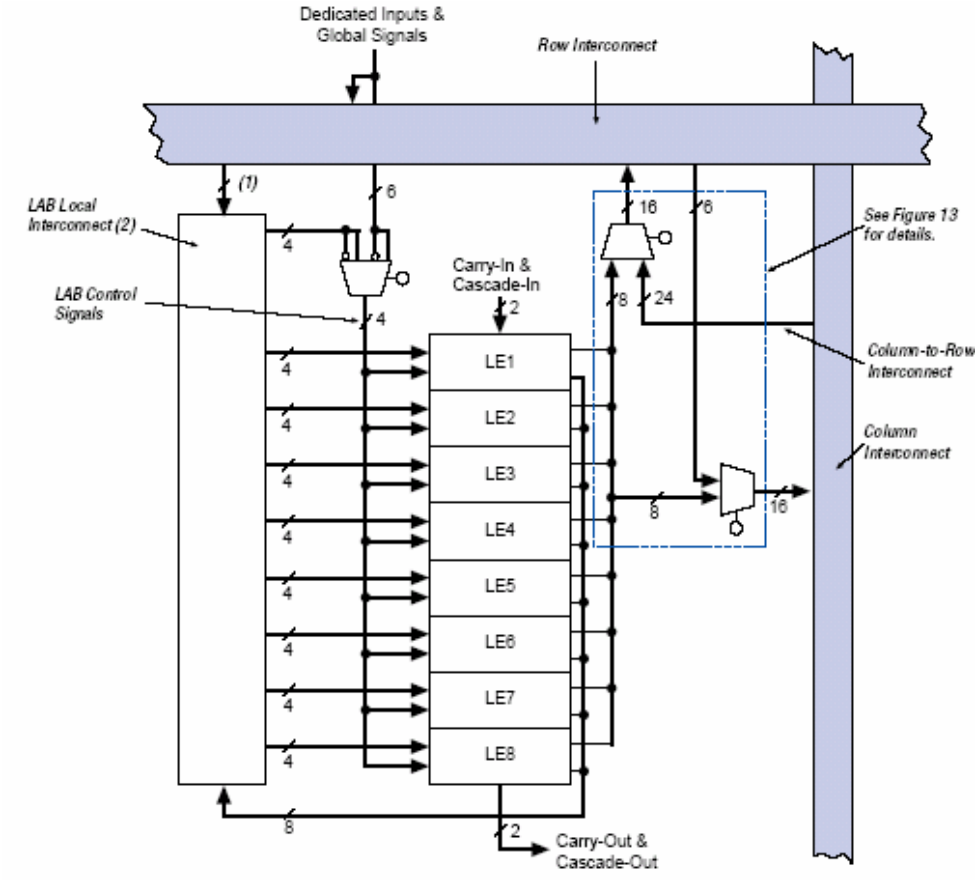


Figura 8. LAB Bloque de Arreglo Lógico de un FPGA ACEX1K

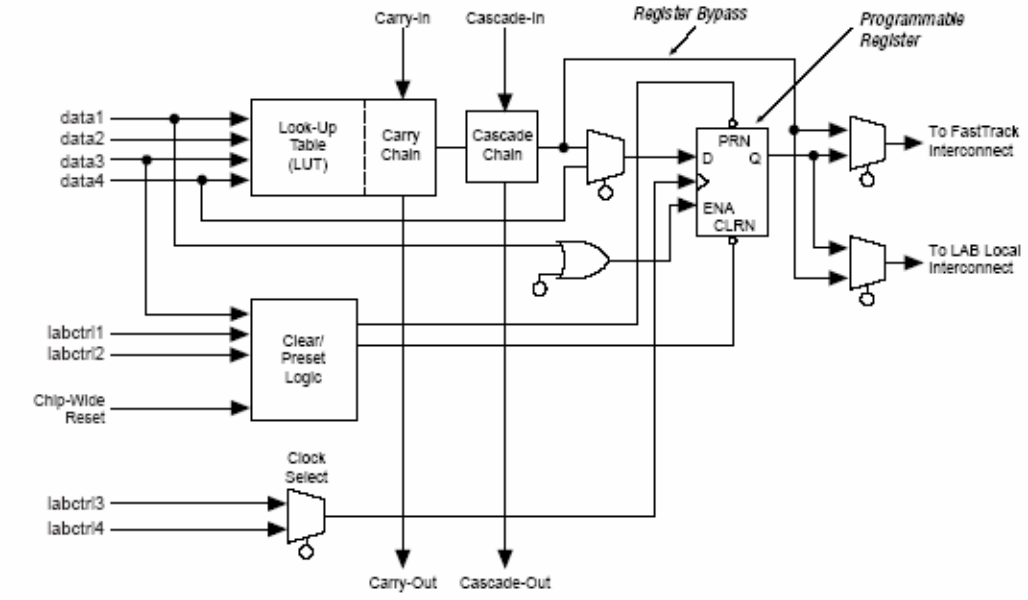


Figura 9. Elemento Lógico de un FPGA ACEX1K

La arquitectura de interconexión entre los Elementos Lógicos, EABs, y las terminales entrada/salida del dispositivo son proporcionados por la estructura denominada “FastTrack”, que es una serie de canales horizontales y verticales continuos que atraviesan el dispositivo.

La estructura de interconexión FastTrack consiste en los canales de la interconexión de la fila y de la columna que atraviesan el dispositivo entero. Cada fila de LABs es servida por una interconexión dedicada de la fila. La interconexión de la fila puede conducir los terminales de entrada/salida y alimentar otros LABs en la fila. Las interconexiones de fila pueden alimentar tanto señales de entrada/salida como otros LABs en la fila.

Los canales de la fila sirven a una interconexión local en los LAB o EAB. La señal de la fila se encuentra desacoplada en cada LAB o EAB para reducir el efecto de retraso por fan-out. Un canal de la fila se puede conducir por un LE o por uno de tres canales de la columna.

2.1.4 Tabla de funciones. El criterio más importante para la selección de las memorias fue la velocidad de lectura. Además la entrada a la tabla es un dato de 15 bits determinada por la ALU, obteniendo una salida de 14 bits de donde se excluyen los dos bits mas significativos (que ya fueron determinados por la unidad aritmética lógica), que corresponden al bit de cero y al bit de signo respectivamente, la cantidad de memoria requerida es de $2^{15} \times 14$. En donde 2^{15} son las posibles direcciones de memoria, y 14 son los bits de información alojados en cada posición de memoria.

Teniendo en cuenta esto y la disponibilidad de las mismas en el mercado se llegó a la selección de dos memorias AM27C256-45, que poseen una velocidad de lectura de 45 ns. En el capítulo 3.6 se detalla la descripción de la tabla de funciones.

2.2 ESPECIFICACIONES DE SOFTWARE

La aplicación desarrollada en el lenguaje de programación JAVA, con la herramienta Visual J, llamada “controlador ALU”, permite la interacción entre el usuario y la unidad de aritmética y lógica en base logarítmica, allí se visualizan los datos que el usuario desea operar, el resultado de la operación, el código de operación, y las banderas que se activaron, además permite la manipulación directa de los bits de los datos. En la figura 10 se observa la interfaz entre el usuario y el PC.

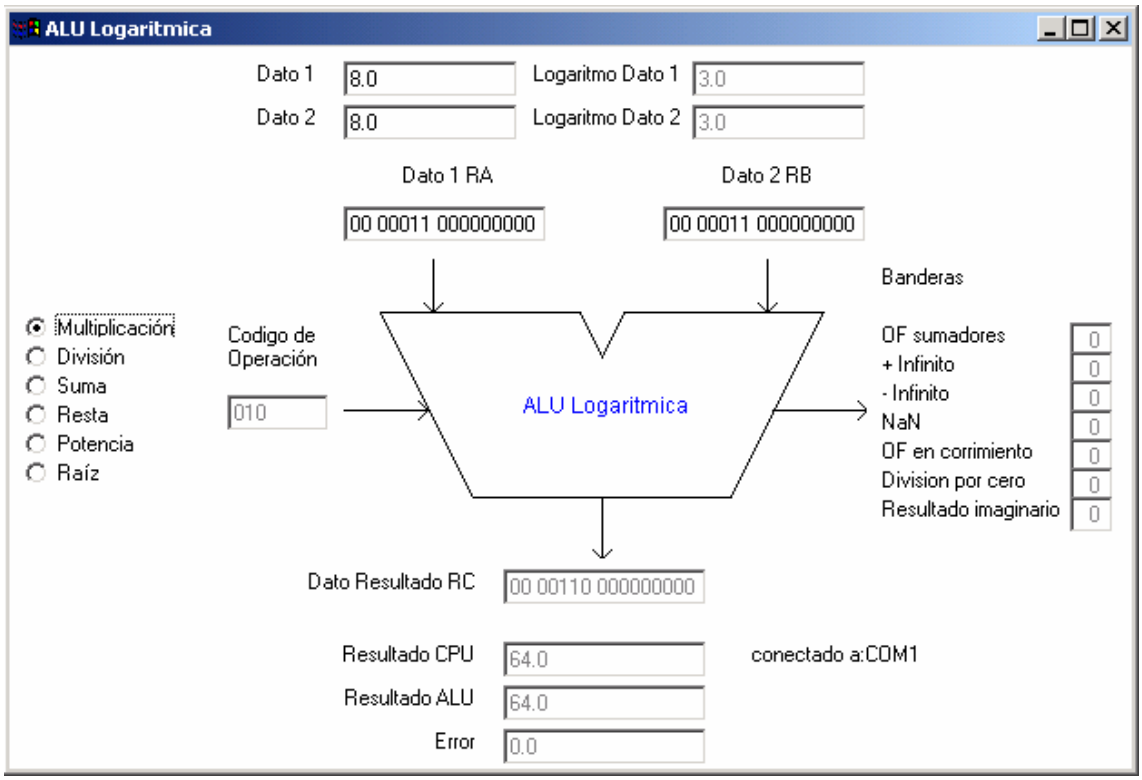


Figura 10. Interfaz didáctica con el usuario

3. DESARROLLOS

Los desarrollos realizados se basaron en el esquema del sistema global de la figura 1. De allí se dedujo cuales eran las etapas de desarrollo más importantes: interacción usuario - PC, modulo de control, ALU logarítmica y tablas de función. Adicionalmente hubo una etapa de búsqueda de bibliografía, una etapa sobre elección de la longitud del formato de bits, y una de desarrollo de circuitos impresos, necesarios para la implementación del sistema. Ciertas etapas permitieron que se trabajaran en paralelo para acelerar el proceso de desarrollo.

3.1 DOCUMENTACIÓN

La etapa de documentación se basó en la recopilación de información sobre los dispositivos de lógica programable, los microcontroladores, las memorias, lenguajes de descripción y programación, y el software que iba a ser utilizado, para finalmente realizar una evaluación y selección de las herramientas más adecuadas para este proyecto.

Se recopiló información acerca del lenguaje de descripción de hardware VHDL[8], el cual fue escogido desde el inicio del proyecto, y sobre el programa de Altera QUARTUS II. Así mismo de la manera de programar VHDL sobre QUARTUS II[9].

Se buscó información sobre los microcontroladores de Microchip, teniendo en cuenta el número de pines de propósito general que poseen, este aspecto es importante debido al número de pines en los buses de datos de la ALU Logarítmica[10].

Para la interfaz entre usuario - PC se investigó la posibilidad de desarrollar esta aplicación en C++ o en JAVA. Finalmente se decidió que el programa para realizar esta interfaz, sería JAVA, debido a la portabilidad de las aplicaciones desarrolladas en este lenguaje[11].

También se investigó las diferentes hojas de especificaciones de los dispositivos de lógica programable de ALTERA, teniendo en cuenta la memoria de cada dispositivo, para evaluar la posibilidad de incluir en la memoria la tabla de funciones para las operaciones de suma y resta[12]. En la sección 2.1.3 se realizó el análisis del dispositivo FPGA utilizado.

[8] ARMSTRONG, James R. “Structured logic design with VHDL”. New Jersey: Prentice Hall, 1993.
[9] “QUARTUS II Handbook”. San Jose CA: Altera, 2004.
[10] “Microchip PIC 18F8720 Datasheet”. Microchip, 2004.
[11] LEMAY, Laura; PERKINS, Charles & MORRISON, Michael. “Teach yourself JAVA in 21 days”. Sams net, 1996.
[12] “ACEX 1K, Programmable logic device family”. San Jose CA: Altera, 2003.

De la información obtenida de ALTERA, se analizó la manera de configurar el dispositivo FPGA, dos de los métodos más utilizados son JTAG y configuración serial pasiva por medio de cable de configuración y programación ByteBlasterMV. Fue elegido el segundo método. En las hojas de datos de ALTERA se describe la manera de conectar el puerto paralelo del PC, el cable ByteBlasterMV y los pines del dispositivo de lógica programable[13].

3.2 PROGRAMACIÓN INTERFAZ USUARIO - PC

La programación de la interfaz usuario - PC se realizó en el lenguaje de programación JAVA con la herramienta **Visual J++** Versión 6. Adicionalmente se utilizó la expansión de librería "**Communication API**", que contiene las definiciones para las funciones que manejan la comunicación serial RS-232.

El resultado final de la interfaz se muestra en la figura 10, en ésta se visualizan los dos datos de entrada con su representación logarítmica en decimal y binaria, los seis botones de operación (suma, resta, multiplicación, división, potencia y raíz), el resultado de la operación del computador, el código de operación, el resultado de la operación de la unidad aritmética lógica, las banderas y el error obtenido en la operación.

Hay dos posibilidades de introducir los datos, la primera de éstas es la introducción en forma decimal, pero existe la limitación que no permite números con decimales, y la segunda es manipulando directamente los bits del cuadro de texto donde aparece en formato binario cada dato. Los números al ser introducidos a la interfaz son convertidos a formato logarítmico por el programa, en la ALU Logarítmica no está implementada la función de conversión de datos.

La información de los datos y el código de operación son enviados en forma serial al microcontrolador por medio del puerto RS-232, en el siguiente orden: primero se envía el byte de código de operación, seguido por dos bytes que conforman el primer dato, y por último dos bytes que conforman el segundo dato.

Cuando se obtiene respuesta a una operación determinada, la interfaz muestra el resultado de la unidad aritmética lógica en el campo "Respuesta ALU", y compara con la respuesta generada por el PC, el error resultante se observa en el campo "Error", y es la diferencia de estas dos repuestas.

[13] "Configuration Handbook". San Jose CA: Altera, 2004.

El ANEXO B posee el código fuente de la aplicación que permite la interacción entre la ALU logarítmica y el usuario, escrita en lenguaje de programación JAVA.

3.3 MICROCONTROLADOR

El microcontrolador es el dispositivo de comunicación entre la interfaz usuario – PC y la unidad aritmética lógica. La interfaz envía 5 tramas de 1 byte por puerto serial RS-232, el microcontrolador se encarga de colocar esta información en los puertos de salida. La manera en que el microcontrolador coloca estos cinco bytes en sus puertos de E/S, se observa en la tabla 5.

	BITS[0..7]	**BITS [8..15]**
DATO A	Puerto B	Puerto D
DATO B	Puerto E	Puerto F
CÓDIGO OP	Puerto A[0..2]	

Tabla 5. Datos de Salida del microcontrolador a la ALU

El orden en que es enviada la información de la interfaz hacia el microcontrolador se aprecia en la tabla 6.

BYTE #	**DATO ENVIADO**
Byte 1	Código de operación
Byte 2	Dato A MSB
Byte 3	Dato A LSB
Byte 4	Dato B MSB
Byte 5	Dato B LSB

Tabla 6. Orden del envío de datos de la aplicación “Controlador ALU” al microcontrolador

La unidad aritmética lógica opera esta información, y da como resultado, una respuesta de 16 bits, según los dos datos recibidos y la operación seleccionada, y 7 bits de bandera. Ésta información es colocada por la ALU en el microcontrolador, que a su vez, retorna esta información por puerto serial RS-232 a la aplicación “Controlador ALU” implementada en JAVA. La manera en que la ALU coloca estos bits en los puerto de E/S del microcontrolador se observa en la tabla 7 y el orden en que esta información es enviada a la interfaz se observa en la tabla 8.

	BITS [0..7]	**BITS[8..15]**
RESULTADO	Puerto H	Puerto J
BANDERAS	Puerto G[0..6]	

Tabla 7. Datos de Salida de la ALU al microcontrolador

BYTE #	DATO ENVIADO
Byte 1	Banderas
Byte 2	Dato C MSB
Byte 3	Dato C LSB

Tabla 8. Orden de envío de datos del microcontrolador a la aplicación “Controlador ALU”

En el ANEXO C se encuentra el código fuente en lenguaje ensamblador, de la rutina programada en el microcontrolador PIC18F8720 de Microchip.

3.4 FORMATO DE NÚMERO Y LONGITUD DE LA TRAMA

Una parte importante del desarrollo del sistema digital ALU logarítmica, es la elección del formato de los números y la elección de la longitud de la trama de bits de cada dato. En este caso, dado que el proyecto es sobre todo demostrativo, existe más libertad en este tipo de decisión, dado que no existe un requerimiento predeterminado de precisión o exactitud en el resultado arrojado por la ALU logarítmica que se deba cumplir. Pero esto no significa que se deba pasar por alto esta característica del sistema. Según la aplicación que se le vaya a dar, se debe analizar si el error introducido por las operaciones cumple con los requisitos de la aplicación.

Por este motivo debe existir otro criterio en el cual se debe basar la elección del formato y su longitud, el cual fue el de generar un sistema con la precisión más alta posible, es decir, crear un sistema con el formato de número más largo que se pueda implementar. En donde cada bit que se le agrega a la parte fraccionaria del formato aumenta la precisión del número. Además de esto, son necesarios dos bits; uno de ellos representa el signo del número, y el otro bit representa el número cero, que no tiene representación en base numérica logarítmica. La representación de los números especiales se muestra en el capítulo 2.1.1.1.

Para implementar las tablas de funciones dentro de los EABs del dispositivo FPGA, el formato máximo de datos que se hubiese podido representar es de 8 bits para cada número, más 2 bits de cero y signo. Con memorias externas, implementado con dispositivos EPROM que se consiguen en el mercado, se puede hacer tablas de mayor tamaño, y el formato de número puede alcanzar a 14 bits, más 2 bits de cero y signo. Aunque existía un deterioro en la rapidez de la respuesta del sistema, se optó por implementar un sistema con memorias externas, para obtener una mayor precisión en el formato de datos.

Para repartir los 14 bits de la trama en una parte entera y una parte fraccionaria se predeterminó que el formato logarítmico pudiera abarcar los números que se pueden representar en formato entero binario en complemento dos de 16 bits, es decir los números de -32768 a 32767. El logaritmo en base dos del mayor número es 15, por lo cual la parte entera debe poder representar el número 15, en binario es 1000, al igual que –15, en binario y complemento a dos es 10001, por lo tanto el mínimo número de bits que requiere la parte entera es 5 bits. Ahora dado que cada bit en la parte fraccionaria que se le añada al formato aumenta la precisión del sistema no se le agregan más bits a la parte entera y se destinan los bits restantes a la parte fraccionaria, es decir 9 bits para la parte fraccionaria del número.

3.5 DESCRIPCIÓN DE LA UNIDAD ARITMÉTICA LÓGICA

La ALU logarítmica fue descrita en el lenguaje de descripción VHDL, en el ANEXO A se observa el código de descripción de este sistema y de cada bloque que lo conforma. La entidad “alulogaritmica.vhd” integra todas las unidades de la ALU, presta la conectividad de las señales externas hacia las unidades funcionales y sirve de interfaz hacia el sistema usuario. La figura 11 muestra un diagrama jerárquico de los componentes en la ALU logarítmica.

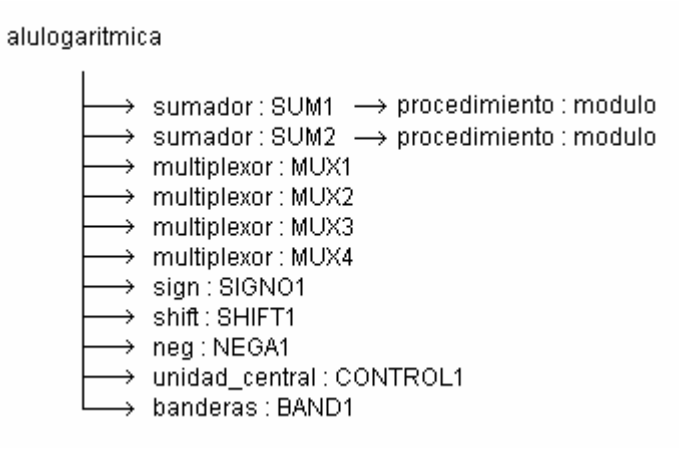


Figura 11. Diagrama jerárquico de la ALU logarítmica

El sistema en general utiliza dos componentes de tipo “sumador.vhd”, cuatro componentes de tipo “multiplexor.vhd”, un componente de tipo “sign.vhd” un componente de tipo “shift.vhd”, un componente de tipo “banderas.vhd”, un componente de tipo “neg.vhd” , y un componente de tipo “unidad_central.vhd”.

La entidad “alulogaritmica.vhd” posee 3 buses de entrada y dos buses de salida, los buses de entrada son “RA” y “RB” cada uno de 16 bits, son los datos de entrada a la ALU logarítmica, y el

bus “CODOP”, de 3 bits, este bus indica la operación que debe realizar la unidad. Los dos buses de salida son “RC”, de 16 bits, es el dato resultante de la operación realizada por la unidad, y el bus “BAND”, de 7 bits, que indica diferentes características sobre la operación que fue realizada.

Tres señales especiales son los buses “ADD” de salida y “SLUTSB” de entrada, ambos de catorce bits, y la señal “STAB” de salida, estas señales sirven para el manejo de las memorias que contienen las tablas de función requeridas para las operaciones de suma y resta, las señales “ADD” y “STAB” proporcionan la dirección que se quiere acceder de la memoria, la señal “SLUTSB” sale del bus de datos de las memorias y regresa al sistema.

En el figura 12 se observa un diagrama en bloques del sistema, este diagrama creado en Quartus II, se asemeja en funcionalidad al de la figura 4, y es una representación gráfica de las conexiones y declaraciones realizadas en la entidad “alulogaritmica.vhd”. En el ANEXO G se encuentra una imagen ampliada de la figura 12.

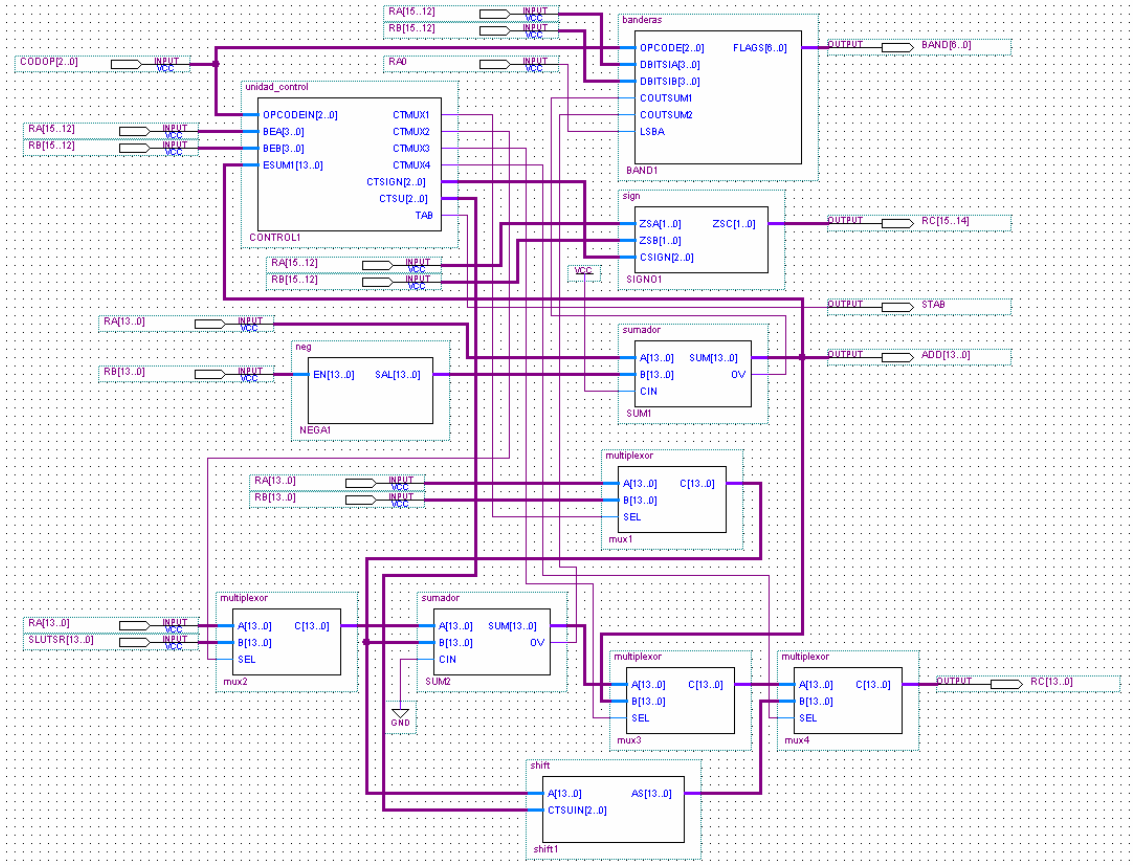


Figura 12. Diagrama en bloques de “alulogaritmica.vhd” creado en Quartus II

3.5.1 Unidad sumadora. La entidad “sumador.vhd” realiza una suma de dos datos de 14 bits, para ello unifica tres módulos de tipo "sumacla", cada modulo “sumacla” es una unidad sumadora de tipo carry-look-ahead[14], se utilizan 2 sumadores de 4 bits y uno de 6 bits, el sumador de 6 bits es utilizado para sumar los bits de datos ubicados en el centro del formato (del 4 al 9 bit), y los sumadores de 4 bits son utilizados para realizar la suma de los bits de los extremos del formato (del bit 13 al 10 y del bit 3 al 0). Esta entidad utiliza el Paquete "sumacla.vhd".

Esta entidad posee tres entradas, dos buses de 14 bits para datos llamados “A” y “B”, un bit llamado “CIN” de entrada para el bit de acarreo entrante y dos salidas, un bus de 14 bits para el resultado llamado “SUM”, y una señal para el bit de acarreo saliente llamado “OV”, este bit es utilizado para determinar si existe desbordamiento de los sumadores. En la figura 13 se observa un bloque funcional de esta entidad.

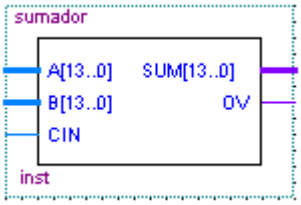


Figura 13. Bloque funcional de “sumador.vhd” creado en Quartus II

El procedimiento del paquete “sumacla.vhd”, realiza la suma de dos vectores de bits de entrada sin importar la longitud de bits de los vectores. Para esto identifica la longitud del vector que se le ha conectado y por medio de un ciclo “FOR”[15] genera las funciones lógicas necesarias. La suma es realizada implementando unidades de tipo carry-look-ahead.

Este procedimiento, recibe dos buses llamados “A” y “B”, y retorna un bus de resultado llamado “C” y dos señales “Gx” y “Px” que indican la generación y propagación de un bit de acarreo. Este procedimiento es utilizado por la entidad “sumador.vhd”

3.5.2 Unidad negadora. La entidad “neg.vhd” tiene la tarea de invertir la señal de entrada. Junto con la entidad "sumador.vhd" se utiliza parra crear una unidad de resta, esta unidad se utiliza en las operaciones de suma, resta y división. Posee dos buses, cada uno de 14 bits, como señales de entrada y salida, llamados “EN” y “SAL” respectivamente. En la figura 14 se observa un bloque funcional de esta entidad.

[14] KOREN, Israel. “Computer arithmetic algorithms”. Natiks: AK Peters. 2002
[15] ARMSTRONG, James R. “Structured logic design with VHDL”. New Jersey: Prentice Hall, 1993.

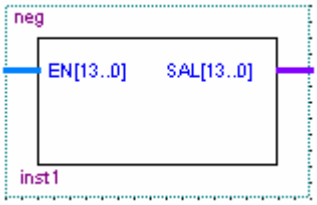


Figura 14. Bloque funcional de “neg.vhd” creado en Quartus II

3.5.3 Unidad de banderas. La entidad “banderas.vhd” genera los bits de banderas, de acuerdo a los datos de entrada del sistema, a los eventos especiales de las unidades funcionales internas y a la operación que se le indica realizar.

Como señales de entrada posee “OPCODE”, este es directamente conectado de las señales de entrada del sistema que indica el código de operación, “DBITSIA” y “DBITSIB” llega de los 4 bits más significativos de los datos de entrada “RA” y “RB”, “COUTSUM1” y “COUTSUM2” son los bits de acarreo generados por los sumadores e indican si algún sumador se desborda, y “LSBA” que es el bit menos significativo del dato “RA”. La salida de esta entidad es el bus “FLAGS” que va directamente conectado a la salida del sistema “BAND”. En la figura 15 se observa un bloque funcional de esta entidad.

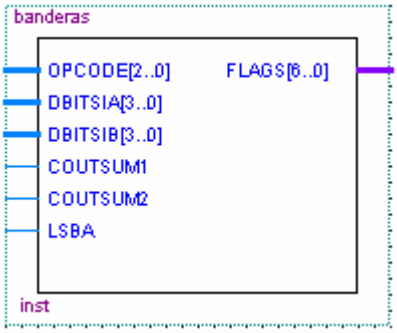


Figura 15. Bloque funcional de “banderas.vhd” creado en Quartus II

3.5.4 Unidad de signo. La entidad “sign.vhd” genera los bits de cero y signo (bits 15 y 14) para el dato resultante de la ALU logarítmica “RC”. Para calcular estos bits, esta entidad se basa en la información proporcionada por los bits de cero y signo de los datos de entrada y de señales de control proporcionadas por la unidad de control, de donde la entidad infiere la operación que se está realizando. En la figura 16 se observa un bloque funcional de esta entidad.

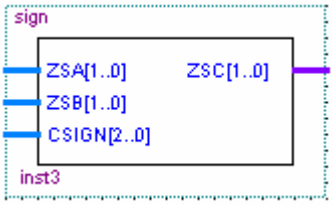


Figura 16. Bloque funcional de “sign.vhd” creado en Quartus II

La tabla 9 muestra la correspondencia entre la señal de control de tres bits “CSIGN” proporcionada por la unidad de control, con el signo de la señal de salida del sistema.

CSIGN	MANEJO
000	La salida es el signo del dato A
001	La salida es el signo del dato B
010	La salida es A15 NOT A14
011	La salida es NOT A15 y A14
100	La salida es B15 y NOT B14
101	La salida es NOT A15 y NOT A14
11X	La salida es A15 y (A14 XOR B14)

Tabla 9. Códigos de entrada de la unidad de signo

3.5.5 Unidad de corrimiento. La entidad “shift.vhd” cumple varias tareas, la principal de ellas es realizar las operaciones de raíz cuadrada y potencia al cuadrado; adicionalmente permite manejar las respuestas de operaciones cuyo resultado sean: + INFINITO, - INFINITO y NaN[16], las cuales dependen únicamente de los datos de entrada y de las operaciones indicadas por el usuario. En la figura 17 se observa un bloque funcional de esta entidad.

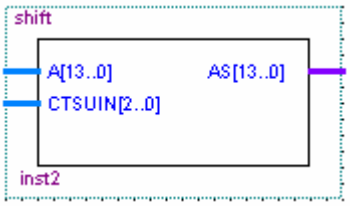


Figura 17. Bloque funcional de “shift.vhd” creado en Quartus II

La tabla 10 muestra la correspondencia de la señal de control de tres bits “CTSUIN” proporcionada por la unidad de control, con la operación de salida de esta unidad.

[16] NaN, acronimo de la frase en ingles “Not a Number”, que quiere decir número indefinido.

CTSUIN	OPERACIÓN
000	La salida es el dato de entrada
001	La salida es la negación de la entrada
010	La salida es infinito
011	La salida es el bit 13 negado
101	La salida es el corrimiento a la izquierda
100, 110 y 111	No esta especificado

Tabla 10. Códigos de entrada de unidad de corrimiento

3.5.6 Unidad multiplexora. La entidad “multiplexor.vhd” cumple la función de mostrar en la salida “C” los datos de entrada “A” o “B”, de acuerdo a la señal de control “SEL”. En la figura 18 se observa un bloque funcional de esta entidad.

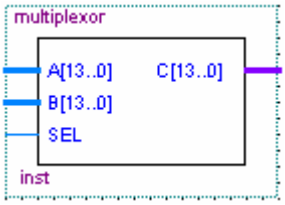


Figura 18. Bloque funcional de “multiplexor.vhd” creado en Quartus II

En la tabla 11 se aprecia el funcionamiento de esta entidad.

SEL	DATOS DE SALIDA
0	A
1	B

Tabla 11. Selección de la unidad multiplexora

3.5.7 Unidad central. La entidad “unidad_central.vhd” cumple la función de generar las señales de control de los multiplexores, de la unidad "shift", de la unidad "sign" y de las tablas de funciones. En la figura 19 se observa un bloque funcional de esta entidad.

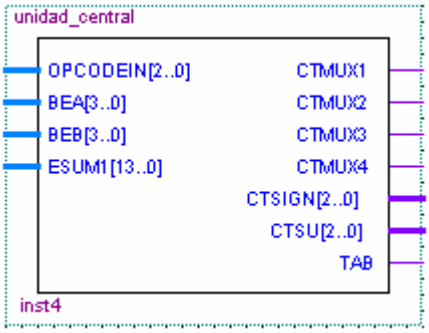


Figura 19. Bloque funcional de “unidad_central.vhd” creado en Quartus II

Posee como señales de entrada 4 buses, “OPCODE”, de 3 bits, el cual viene de la entrada del sistema “CODOP”. “BEA” y “BEB”, de 4 bits cada bus, y son los cuatro bits más significativos de los datos “RA” y “RB” respectivamente. “ESUM1”, de 14 bits, es el resultado del primer sumador, de esta manera la entidad identifica cuando los dos datos de entrada “RA” y “RB” tienen la misma magnitud.

Las salidas de esta entidad son las señales de control para la unidad de corrimiento “CTSU”, la unidad de signo “CTSIGN”, la señal de selección de la función de la tabla “TAB”, y de las señales de selección de los cuatro multiplexores “CTMUX1”, “CTMUX2”, “CTMUX3” y “CTMUX4”.

El bit “TAB” se conecta a la salida del sistema “STAB”, funciona como un bit extra de direccionamiento para las tablas de funciones, Más adelante se describe el funcionamiento de este bit.

3.6 MEMORIA Y TABLA DE FUNCIONES

Para la realización de la función mostrada en la ecuación 9, dada la complejidad de las operaciones, es necesario que se implemente por medio de valores en una tabla que es guardada en una memoria EPROM, de tal forma que las direcciones que se quieran acceder corresponden a los elementos del dominio de la función, y los datos guardados en esas posiciones corresponden a los elementos en el rango de la función.

$$\Phi(R_a - R_b) = \log_2(1 \pm 2^{-(R_a - R_b)}) \quad (9)$$

El formato de datos que trabaja esta tabla, tanto el dato de entrada como el de salida, es el mismo que el formato de números logarítmicos, es decir, se utilizan 5 bits para representar la parte entera

y 9 bits para representar la parte fraccionaria, sin necesidad de los bits de representación de negativo o cero. Además, como se ve en la ecuación 9, es necesario implementar dos ecuaciones diferentes, para esto se utiliza un bit adicional en la dirección, llamado "STAB", generado por la unidad de control de la ALU logarítmica, este bit selecciona una de las dos tablas de función guardadas en la memoria. El valor de este bit depende de la operación que se realice y de los signos de los datos “RA” y “RB”, en la tabla 12 se describe la relación de estas variables.

OPERACIÓN	SA	SB	BIT DIR STAB
Suma	0	0	0
Suma	0	1	1
Suma	1	0	1
Suma	1	1	0
Resta	0	0	1
Resta	0	1	0
Resta	1	0	0
Resta	1	1	1

Tabla 12. Valor del bit de direccionamiento de tabla según la operación y signos de los datos

Los datos y los archivos de programación de las memorias fueron generados a partir de una rutina en MATLAB, esta rutina procesa un vector de datos binarios, que tiene igual tamaño que el número de posiciones de la memoria, toma cada dirección de memoria y la identifica como un número que se puede representar en el formato logarítmico, y genera a partir de la función de la ecuación 9, el dato que debe guardarse en esa posición de memoria. En la tabla guardada en la EPROM se encuentra implícito el esquema de redondeo debido al error de cuantización presentado por el formato del número, en el cual el resultado se aproxima al número más cercano.

Así se requiere una memoria que posea 15 bits de dirección y 14 bits de datos, se utilizaron dos memorias EPROM con palabra de dirección de 15 bits y palabra de datos de 8 bits, las dos posiciones de datos más significativas se rellenaron con cero.

En el ANEXO D se encuentra el código de la rutina que tabula los datos para las memorias y crea los archivos en formato hexadecimal, estos archivos sirven para programar las memorias desde cualquier programador universal.

3.7 CIRCUITOS REALIZADOS

Fueron diseñados dos circuitos, el primer circuito posee un microcontrolador PIC18F8720 de Microchip y un Circuito integrado MAX232 para comunicación serial, y el segundo contiene dos memorias EPROM 27C256. Los esquemáticos de estos circuitos se encuentran en el ANEXO E.

Fueron fabricados dos circuitos impresos de doble capa y con tecnología “Through Hole”. La primera tarjeta posee dimensiones 6,3cm x 11,1cm, y el segundo con dimensiones 7,8cm x 6,3cm. Las figuras 20 y 21 muestran los circuitos impresos.

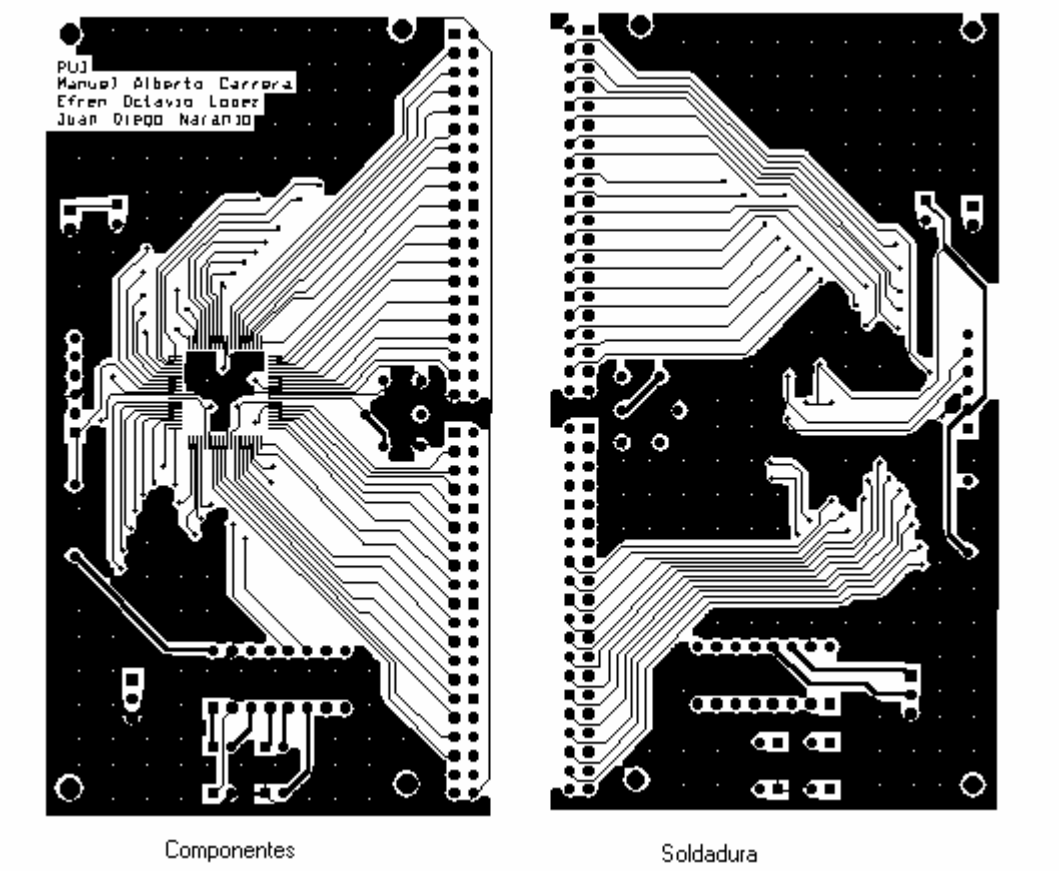


Figura 20. Circuito impreso de la tarjeta con el microcontrolador

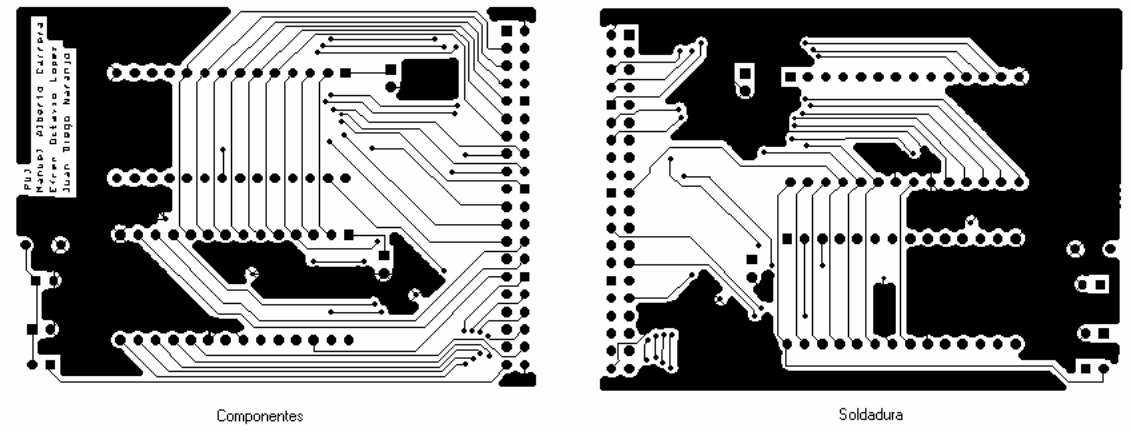


Figura 21. Circuito impreso de la tarjeta con las memorias EPROM

Cada impreso se conecta a la tarjeta de desarrollo por medio de buses de 40 líneas, las tarjetas se conectan de la manera indicada en la tabla 13.

PUERTO TARJETA	PUERTO ALTERIC
MCU JP1	Alteric JP2
MCU JP2	Alteric JP3
Memoria JP1	Alteric JP$

Tabla 13. Conexión entre los puertos de las tarjetas

En las tablas 14 y 15 se puede observar la función de cada uno de los pines de las tarjetas del microcontrolador y de memorias, y la asignación de estas señales con los pines del dispositivo FPGA, las asignaciones se hicieron teniendo en cuenta su localización en el puerto.

PUERTOS TARJETA MICROCONTROLADOR							
PUERTO JP1				PUERTO JP2			
PIN	ASIGNACIÓN	PIN	ASIGNACIÓN	PIN	ASIGNACIÓN	PIN	ASIGNACIÓN
1	NC	2	GND	1	NC	2	NC
3	A9(192)	4	BAND4(193)	3	A7(89)	4	NC
5	A10(195)	6	BAND3(196)	5	NC	6	NC
7	A11(197)	8	BAND2(198)	7	NC	8	NC
9	A12(199)	10	BAND1(200)	9	NC	10	NC
11	A13(202)	12	BAND0(203)	11	NC	12	NC
13	A14(204)	14	B0(205)	13	OUT15(101)	14	NC
15	A15(206)	16	B1(207)	15	OUT14(103)	16	B8(104)
17	OUT8(208)	18	OUT3(7)	17	OUT13(111)	18	B9(112)
19	OUT9(8)	20	OUT2(9)	19	OUT12(113)	20	OUT4(114)
21	OUT10(10)	22	OUT1(11)	21	BAND5(115)	22	OUT5(116)
23	OUT11(12)	24	OUT0(13)	23	BAND6(119)	24	OUT6(120)
25	A0(14)	26	B2(15)	25	NC	26	OUT7(122)
27	A1(16)	28	B3(17)	27	NC	28	B10(126)
29	A2(18)	30	B4(24)	29	OPC0(127)	30	B11(128)
31	A3(40)	32	B5(41)	31	OPC1(131)	32	B12(132)
33	A4(44)	34	B6(45)	33	OPC2(133)	34	B13(134)
35	A5(29)	36	B7(30)	35	NC	36	B14(136)
37	A6(31)	38	A8(46)	37	NC	38	B15(140)
39	GND	40	NC	39	NC	40	GND

Tabla 14. Localización y asignación de pines de la tarjeta con el MCU

En la tabla 14 se observa que los pines 31, 32, 33, 34 y 38 del puerto 1 de la tarjeta con el microcontrolador, no tienen la asignación correspondiente al puerto 2 de la tarjeta Alteric, debido que en un principio se concibió que se conectaría este puerto con el puerto 1 de la tarjeta Alteric, pero este puerto resultó defectuoso por lo cual se procedió a conectar al puerto 2 que era el puerto de mayor similitud con el puerto 1, sobre todo por la localización de los pines de tierra.

PUERTO TARJETA MEMORIAS			
PUERTO JP1			
PIN	**ASIGNACIÓN**	**PIN**	**ASIGNACIÓN**
1	NC	2	GND
3	AD0(141)	4	NC
5	AD1(143)	6	D0(4)
7	AD2(147)	8	D1(8)
9	AD3(149)	10	D2(10)
11	AD4(157)	12	D3(12)
13	AD5(159)	14	D4(14)
15	AD6(161)	16	D5(16)
17	AD7(163)	18	D6(18)
19	AD8(166)	20	D7(20)
21	AD9(168)	22	NC
23	AD10(170)	24	D8(24)
25	AD11(173)	26	D9(26)
27	AD12(175)	28	D10(28)
29	AD13(177)	30	D11(30)
31	AD14(180)	32	D12(32)
33	NC	34	D13(34)
35	NC	36	D14(36)
37	NC	38	D15(38)
39	NC	40	GND

Tabla 15. Localización y asignación de pines de la tarjeta con las memorias

4. ANÁLISIS DE RESULTADOS

4.1 PRUEBAS REALIZADAS

Se realizaron dos clases de pruebas y un análisis al formato de números; la primera clase de pruebas que se realizó, fue introducir una serie de operaciones a la ALU logarítmica y analizar el error generado en cada operación, esta serie de operaciones se realizó de diferentes formas para reflejar el comportamiento de las operaciones.

La segunda prueba realizada fue un análisis de tiempo de las señales que se generan en la ALU logarítmica. En esta prueba se trató de recrear el caso en el cual se generará los mayores tiempos de propagación.

4.1.1 Análisis del error generado en las operaciones. Se sabe que cada dato introducido a la ALU logarítmica, es el logaritmo en base dos de su respectiva cantidad a operar, esta situación se refleja en la ecuación 24, donde *y* es dato y *x* es la cantidad a operar.

$$y = \log_2 x \quad (24)$$

De la ecuación 24 podemos deducir el procedimiento planteado de la ecuación 25 a la 28.

$$\frac{dy}{dx} = \frac{1}{x\ln(2)} \quad (25)$$

$$dy = \frac{dx}{x\ln(2)} \quad (26)$$

$$dx = x\ln(2)dy \quad (27)$$

$$\Delta x \approx x\ln(2)\Delta y \quad (28)$$

Como cada dato puede incrementarse solo en un bit, el incremento menos significativo se observa en las ecuaciones 29 y 30.

$$\Delta y_{\min} = \frac{1}{2^9} \quad (29)$$

$$\Delta x \approx \frac{x \ln(2)}{2^9} \quad (30)$$

El mínimo incremento en el dato es constante en todo el intervalo de números representables, pero según las ecuaciones anteriores esto no es cierto para el intervalo de las cantidades a operar, y este intervalo crece linealmente con la magnitud del dato. Si los números que se quieren operar son de magnitudes pequeñas el error generado por la cuantización es pequeño, pero este error crece a medida que crece la magnitud de los datos.

Debido al esquema de redondeo, en el cual se aproxima al número representable más cercano, el error de cuantización de un dato, introducido por la aplicación "controlador.java" se puede calcular con la ecuación *31*.

$$e_{cuantización} \approx \frac{x \ln(2)}{2^{10}} \quad (31)$$

Fueron analizadas tres series de datos, cada serie fue diseñada para reflejar las características de la operación, el parámetro que se medía era la diferencia entre el resultado según la ALU logarítmica y el resultado según el PC, es decir el error introducido por la operación. En cada serie se probaron los datos con error de cuantización y sin error de cuantización en los datos.

Las operaciones realizadas en la primera serie de datos fueron: suma, resta y división. Se desarrolló de la manera descrita en las ecuaciones 32, 33 y 34.

$$opa = z + 200 \quad (32)$$

$$opb = z \quad (33)$$

$$z = z + 200 \quad (34)$$

Las operaciones realizadas en la segunda serie de datos fueron suma y resta, y se desarrolló según el procedimiento de las ecuaciones 35, 36 y 37.

$$opa = z \quad (35)$$

$$opb = 300 \quad (36)$$

$$z = z + 400 \quad (37)$$

La operación realizada en la tercera serie de datos fue el producto y se desarrollo según el procedimiento de las ecuaciones 38, 39 y 40.

$opa = z$ *(38)*

$opb = 5$ *(39)*

$z = z + 400$ *(40)*

Para la primera serie de datos se obtuvieron los siguientes resultados: En las figuras 22, 23 y 24 se ven los errores generados por la suma, la resta y la división respectivamente, las líneas rojas muestran $\pm \Delta x$ en función de *x,* de esta manera se observa que los datos tratan de mantenerse dentro del área de error de cuantización, dado que los datos, al igual que los resultados, son afectados por este tipo de error, existen algunas operaciones en las cuales el error conjunto supera la región de tolerancia, estos se ve sobre todo en la suma, donde un error de cuantización en teoría podría alcanzar $2\Delta x$, según el procedimiento de las ecuaciones 41 a la 47.

$opc = (opa + e_a) + (opb + e_b) + e_c$ *(41)*

$opa = opb + 200$ *(42)*

$opa \approx opb$ *Para opa>>200 y opb >>200 (43)*

$opc \approx 2opa \approx 2opb$ *(44)*

$e_c \approx 2e_a \approx 2e_b$ *(45)*

$\Delta x = 2e_a$ *(46)*

$e_{total} = e_a + e_b + e_c \approx 2\Delta x$ *(47)*

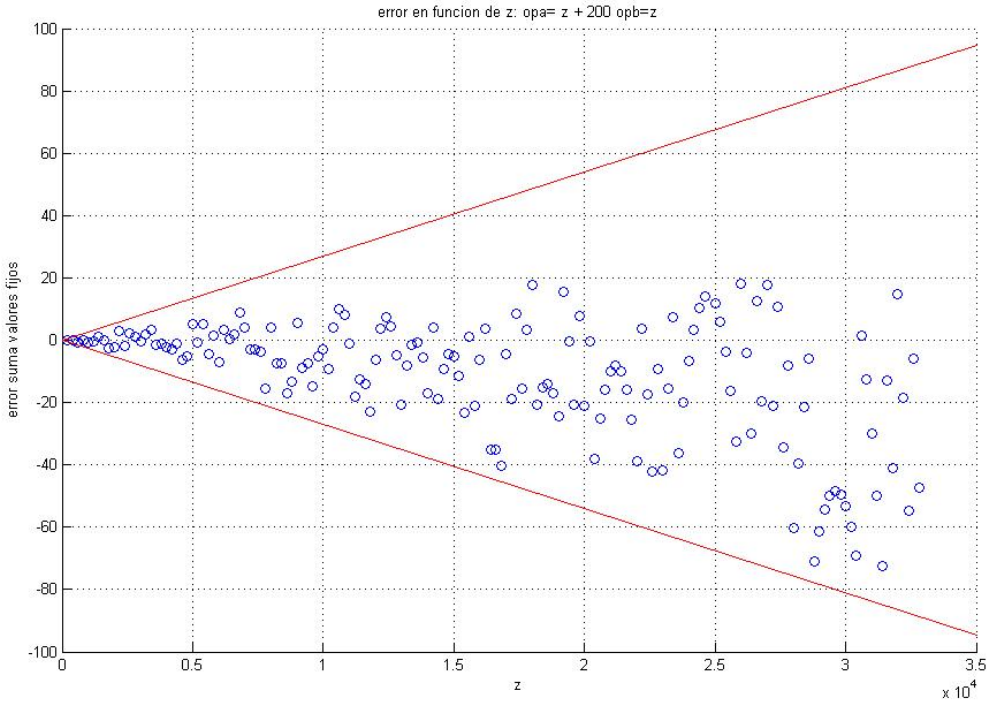


Figura 22. Error generado por la suma, datos A y B poseen error de cuantización

Para la resta se aplica un procedimiento similar, pero dado que el resultado siempre es de la misma magnitud, $opc \approx 200$, el error del resultado es muy bajo y su contribución al error total es despreciable, por lo tanto $e_{total} \approx 2e_a = \Delta x$.

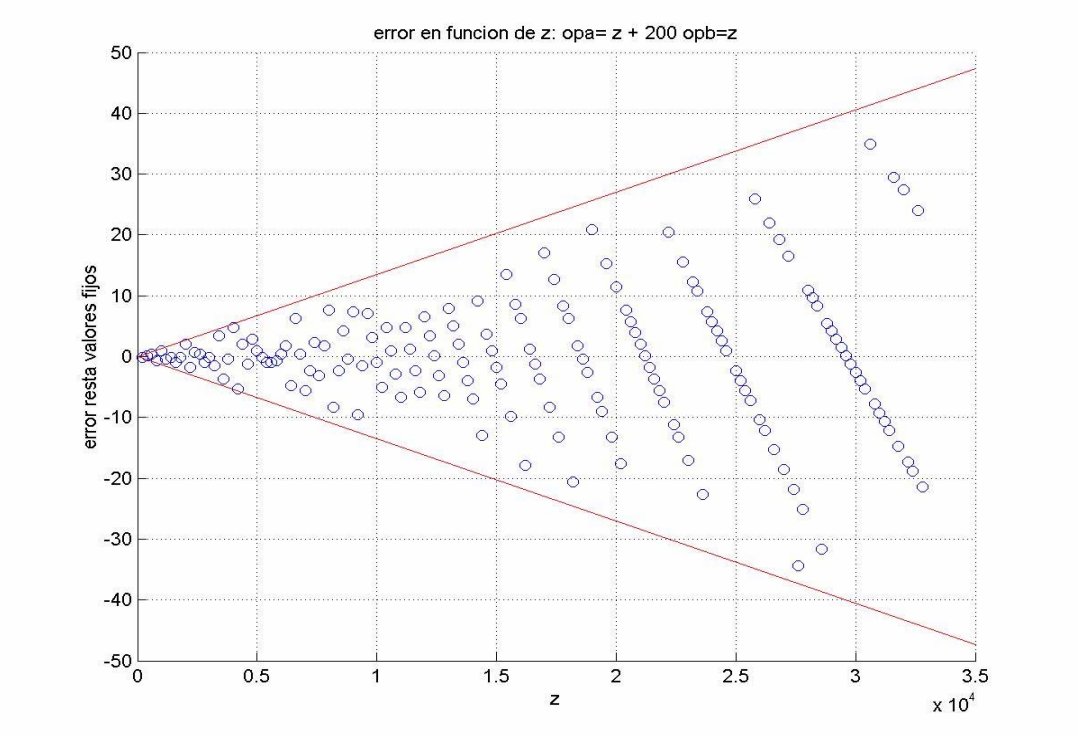


Figura 23. Error generado por la resta, datos A y B poseen error de cuantización

Para la división, el análisis es diferente y no es tan trivial como con la suma o la resta, pero se puede estimar el error de cuantización basado en el error del resultado, $opa \approx opb \Rightarrow opc \approx 1$, en esta magnitud de *x* el error es muy pequeño, tal como lo refleja la figura 24.

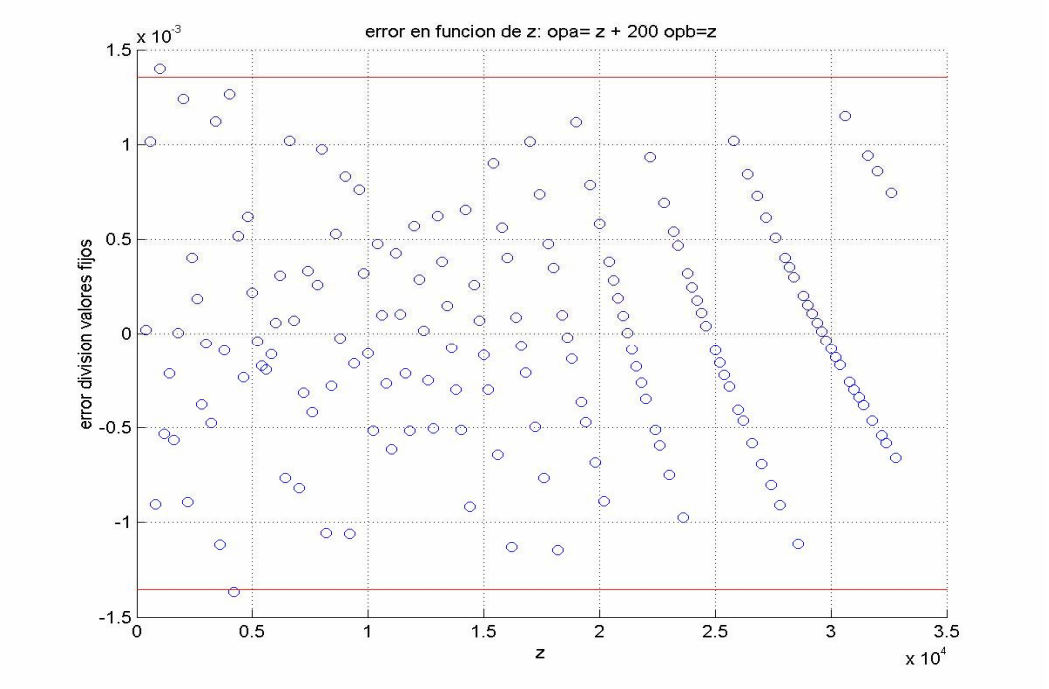


Figura 24. Error generado por la división, datos A y B poseen error de cuantización

En las figuras 25, 26 y 27 se observan los errores generados para las operaciones de suma resta y división, con los mismos datos que se utilizaron para los tres análisis anteriores, pero ahora los datos no poseen error de cuantización, por lo tanto $e_a = e_b = 0$, y el error obtenido en estas pruebas solo corresponden al error de cuantización del resultado.

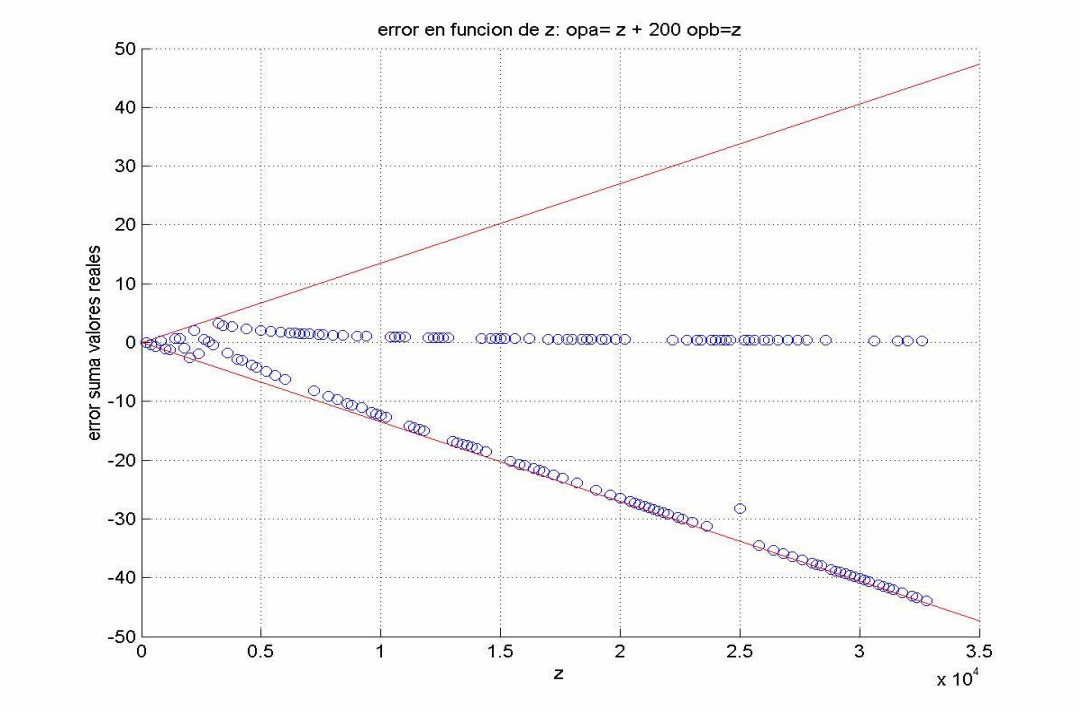


Figura 25. Error generado por la suma, datos A y B no poseen error de cuantización

Para la suma el error de cuantización se determina según las ecuaciones 48 a la 54.

$$opc = opa + opb + e_c \quad (48)$$

$$opa = opb + 200 \quad (49)$$

$$opa \approx opb \quad (50)$$

$$opc \approx 2opa \approx 2opb \quad (51)$$

$$e_c \approx 2e_a \quad (52)$$

$$\Delta x = 2e_a \quad (53)$$

$$e_{total} = e_c \approx \Delta x \quad (54)$$

Se debe aclarar que aunque el error de cuantización de los datos de entrada es nulo, e_a representa el caso crítico en caso de que este error existiera.

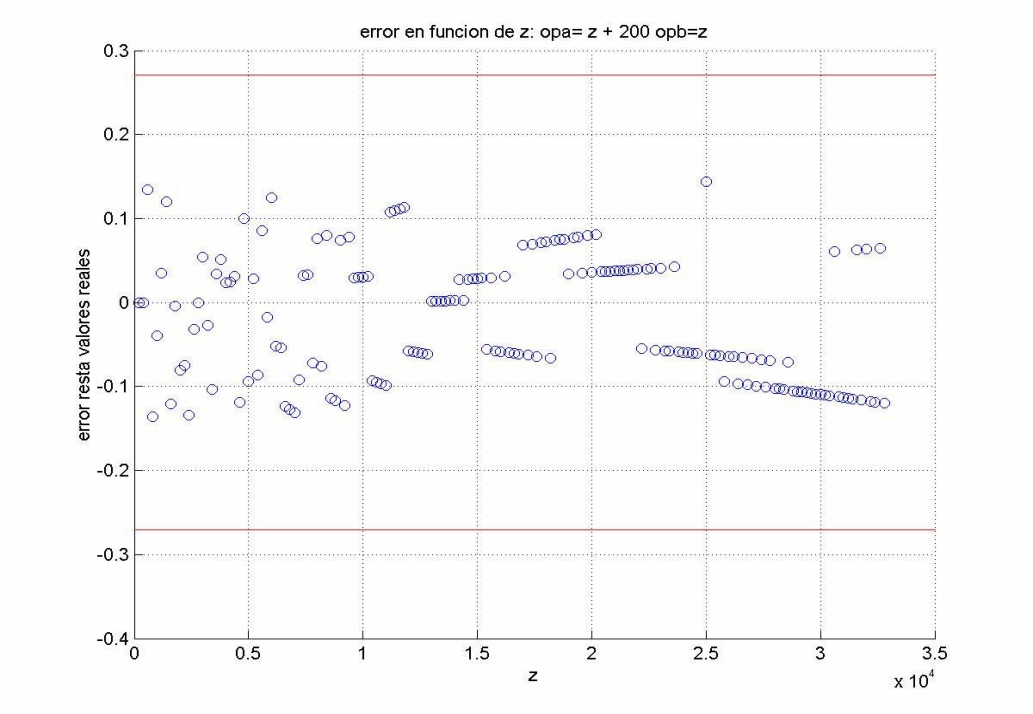


Figura 26. Error generado por la resta, datos A y B no poseen error de cuantización

Para la resta $opc = 200$, entonces e_c es muy pequeño. Un fenómeno similar ocurre con la división.

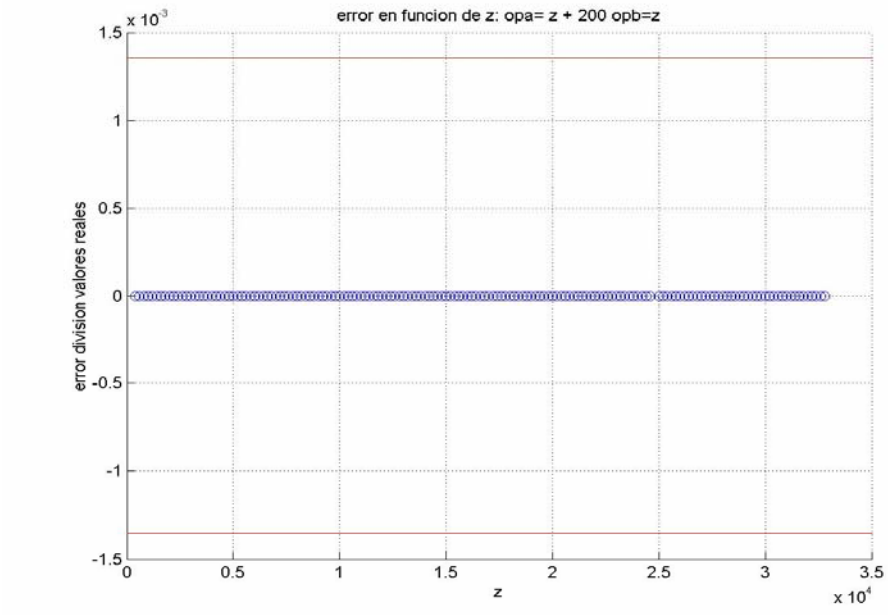


Figura 27. Error generado por la división, datos A y B no poseen error de cuantización

En la segunda prueba el dato opb se deja fijo en 300 y se incrementa el dato opa. Las figuras 28, 29, 30 y 31 corresponden a los mismos análisis de las figuras 22, 23, 25, 26.

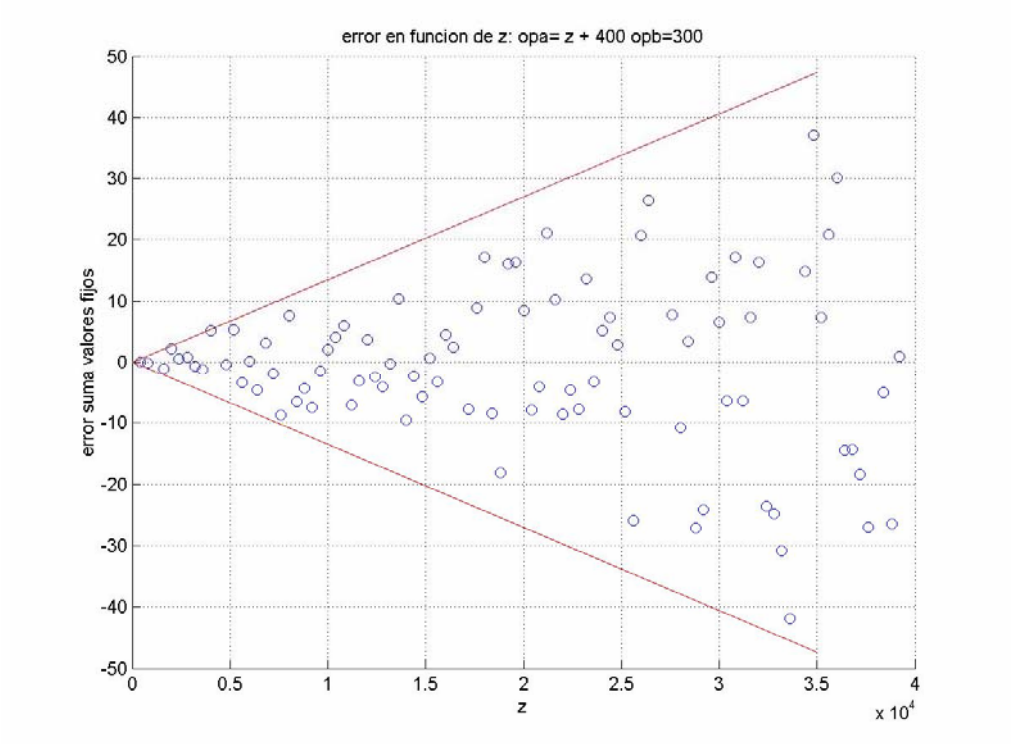


Figura 28. Error generado por la suma, segunda prueba, datos A y B poseen error de cuantización

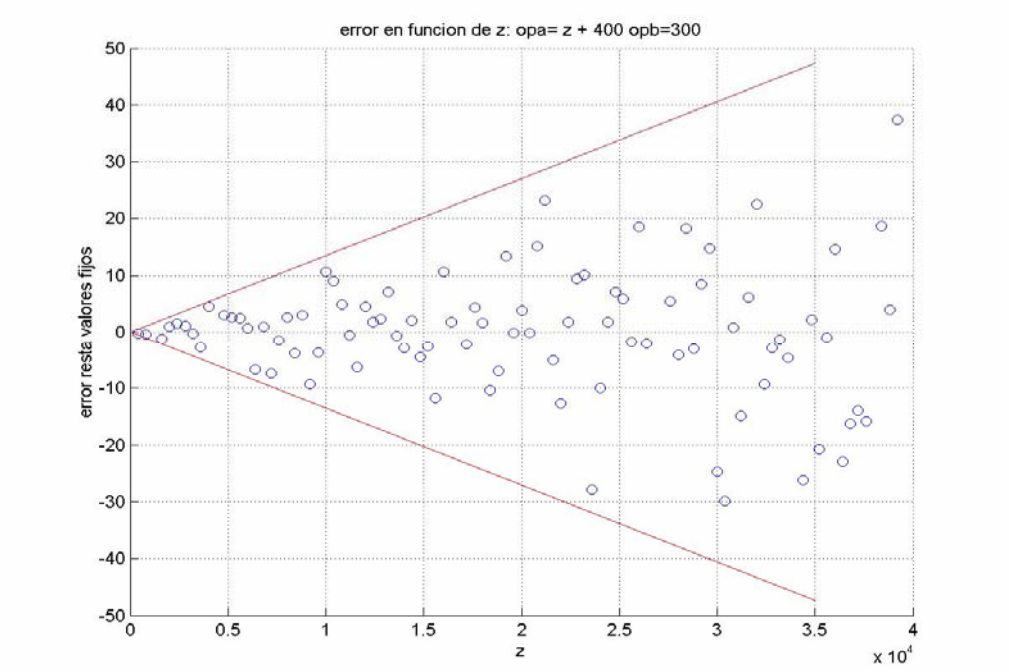


Figura 29. Error generado por la resta, segunda prueba, datos A y B poseen error de cuantización

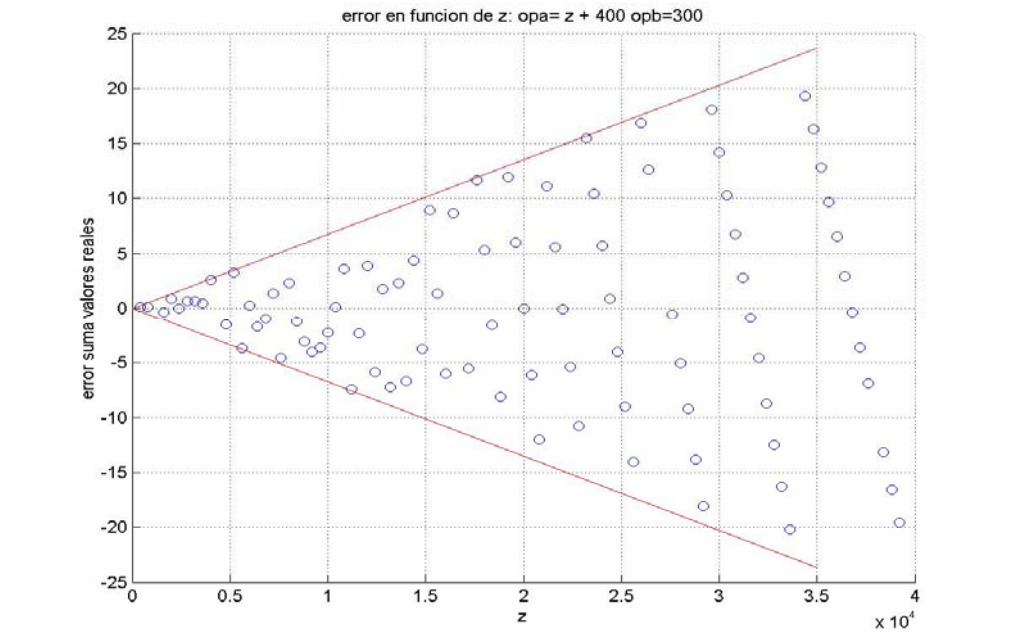


Figura 30. Error generado por la suma, segunda prueba, datos A y B no poseen error de cuantización

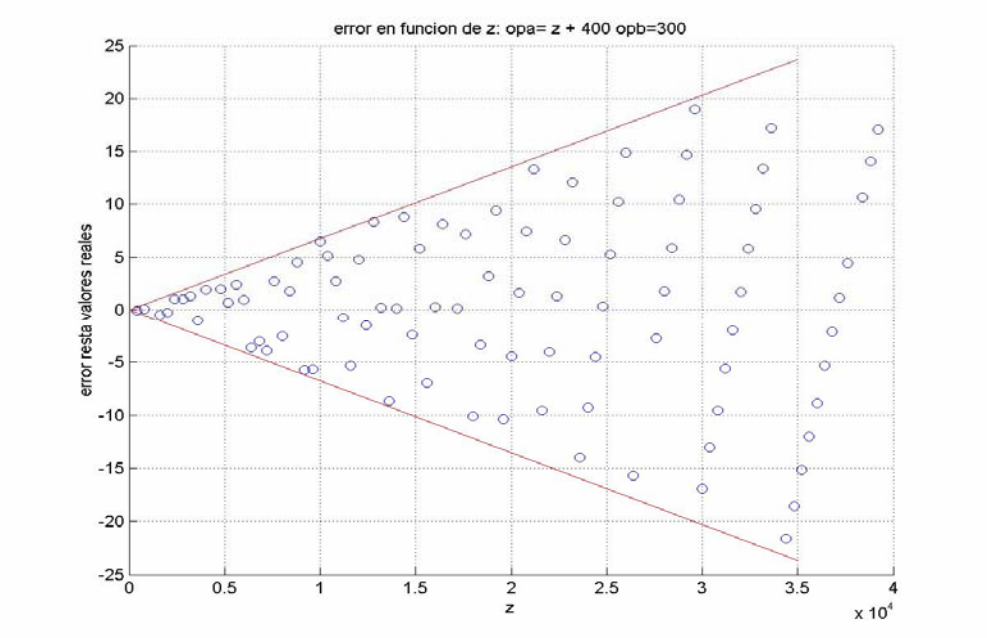


Figura 31. Error generado por la resta, segunda prueba, datos A y B no poseen error de cuantización

En las figuras 32 y 33 se observa los resultados de la tercera prueba, donde se examina los errores generados por la operación de multiplicación. El dato opb se deja fijo en 5 mientras el dato opa es aumentado. Dada la ecuación *58* se observa que el error de cuantización puede llegar a ser mucho más alto que en otras operaciones.

$$opc = (opa + e_a) \times (opb + e_b) + e_c \quad (55)$$

$epb = 5 \Rightarrow e_b \approx 0$ Según la ecuación *31* *(56)*

$$opc \approx (opa + e_a) \times (opb) + e_c \quad (57)$$

$$opc \approx opa \times opb + e_a \times opb + e_c \quad (58)$$

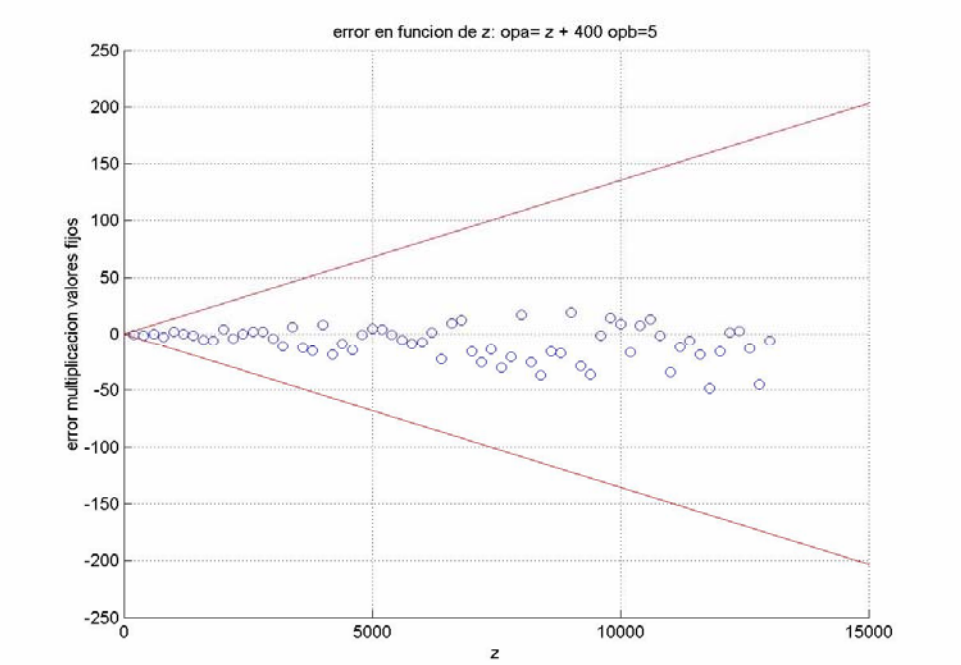


Figura 32. Error generado por la multiplicación, datos A y B no poseen error de cuantización

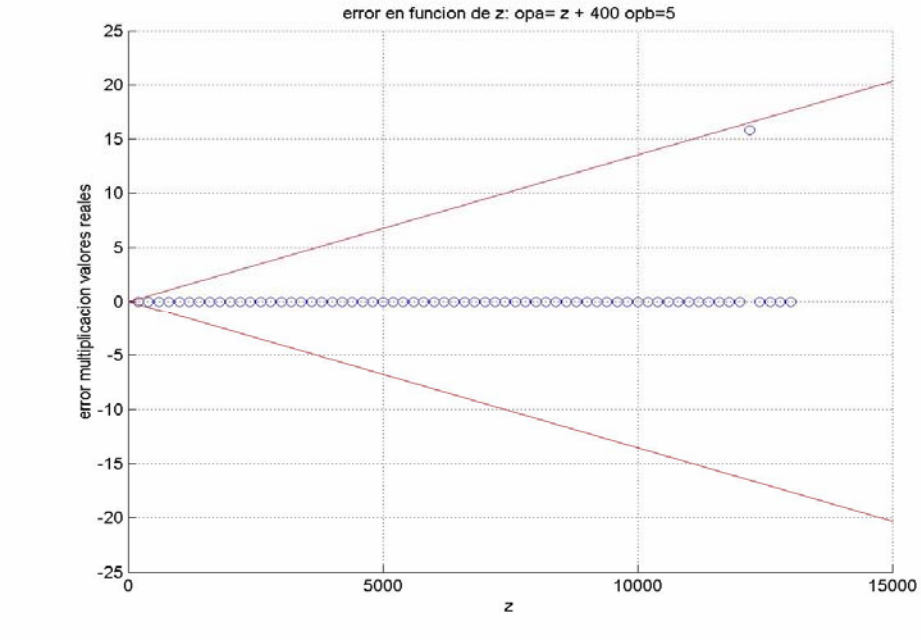


Figura 33. Error generado por la multiplicación, datos A y B no poseen error de cuantización

4.1.2 Medición del tiempo de retardo. Para medir el tiempo de retardo se debieron enfrentar varias dificultades técnicas, la principal era la velocidad con la cual el microcontrolador suministraba los datos a la ALU, dado que el microcontrolador ponía un byte a la vez y el tiempo entre byte y byte dependía del tiempo de reloj del microcontrolador, de esta manera los datos nunca eran introducidos al mismo tiempo en la ALU logarítmica.

Se tomó una serie de medidas que permitían obtener el retardo máximo de las señales con el cual el sistema respondería, la primera era con respecto al orden en que los bytes eran suministrados a la ALU, por esto el byte menos significativo del dato b es el ultimo en ser transmitido, de esta manera se obtiene el peor de los casos en el momento que la unidad sumadora de la entrada genera la propagación de bit de acarreo más demorada.

En la figura 33 se ve un diagrama de tiempos en el cual se puede medir el tiempo de retardo generado por la operación suma bajo condiciones críticas, es decir, éste es uno de los mayores tiempos de propagación que se va a obtener de la ALU con respecto al acarreo generado por la unidad de resta que se encuentra a la entrada. El retardo obtenido es 687.97 ns, este dato se basa en una sola medicion La operación realizada para obtener este diagrama de tiempos fue la suma de los siguientes datos:

Op A 0.9986 00 11111 111111111
OP B -1 01 00000 000000000

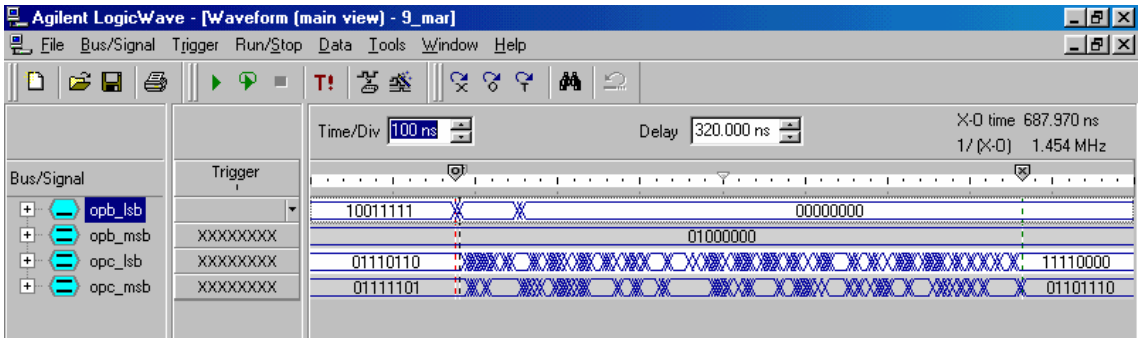


Figura 34. Medición tiempo de propagación de la operación suma

En la figura 34 se observa otro diagrama de tiempos, pero en éste la operación que se evalúa es una multiplicación, también bajo condiciones criticas, como resultado el retardo obtenido es 91.79 ns. Este diagrama de tiempos se obtiene de la multiplicación de los siguientes datos:

Op A 0.00389 00 10111 111111110

OP B -1 01 00000 000000000

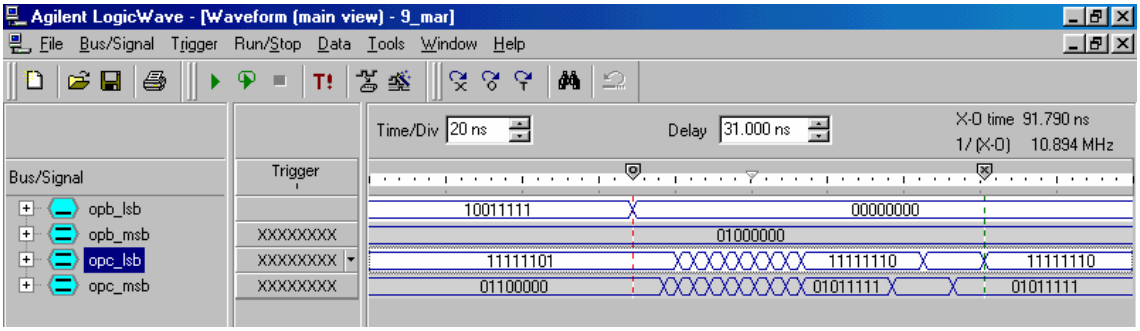


Figura 35. Medición tiempo de propagación de la operación multiplicación

Dada la similitud entre las operaciones de resta y suma, el resultado observado en la figura 33 también puede ser obtenido para la operación de resta. Lo mismo ocurre con las operaciones de producto y división.

La figura 35 muestra el mayor tiempo de propagación que nos indica la herramienta de diseño, el tiempo que nos muestra es de 109.3ns entre la señal RB [9] hasta BAND [0], estos tiempos son calculados por Quartus II. En el ANEXO G se observa una tabla donde quedan registrados los 200 mayores tiempos de propagación, y se muestra cada señal desde el terminal de entrada hasta el terminal de salida.

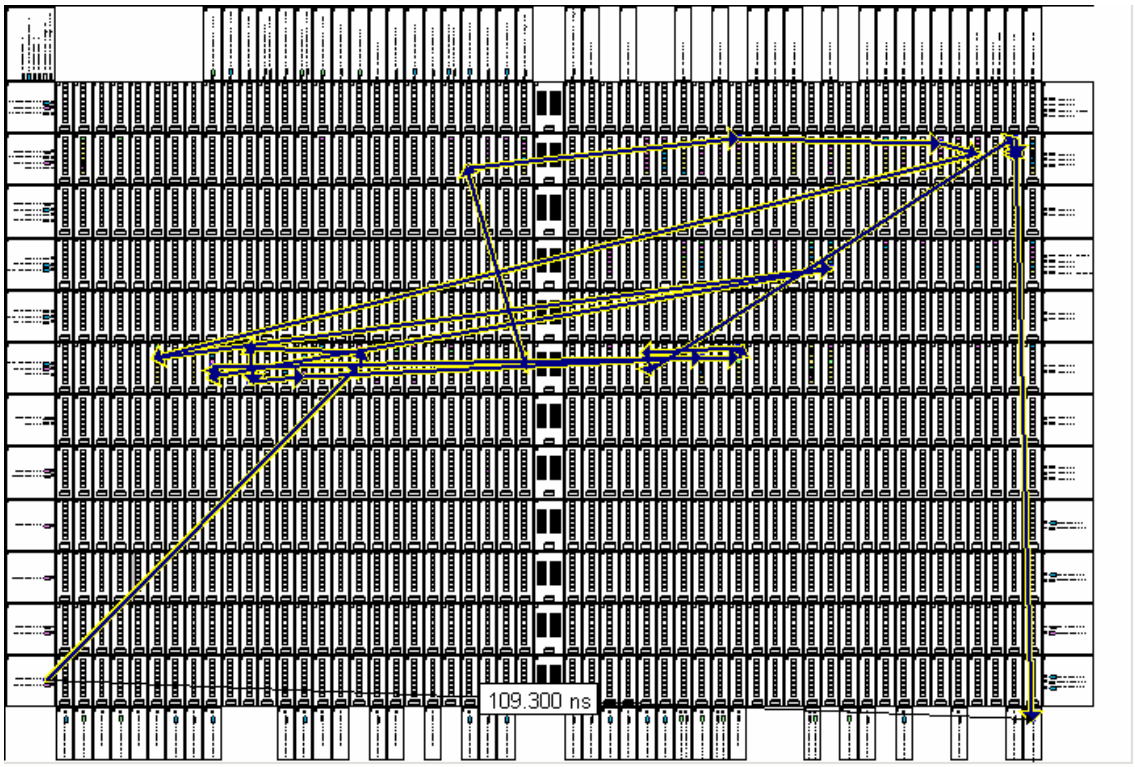


Figura 36. Camino de mayor tiempo de propagación en el Floor Plan.

Aunque este tiempo es el mayor reportado por Quartus II, hay que tener en cuenta que existe una salida del sistema que regresa, por lo tanto el retardo debe ser mayor. Para poder calcular un tiempo de retardo más exacto, debemos tener en cuenta tres componentes de este tiempo, el primero es el mayor tiempo de propagación de cualquier entrada a cualquier terminal de salida a la tarjeta de memorias, este tiempo lo llamamos t_{dra}, el segundo componente es el tiempo de retardo de las memorias, lo llamamos t_{dmem}, y el mayor tiempo que registra la entrada de las memoria a cualquier salida, y lo llamamos t_{dsr}, el primer y tercer componente de este tiempo se puede deducir de la tabla en el ANEXO G, dado que no están registrados estos tiempos, no son mayores a 84.3 ns, el segundo componente según las hojas de especificaciones de las memorias AM27C256-120 es 120 ns.

$$T_{total} = t_{dra} + t_{dmem} + t_{dsr} \qquad (59)$$

$$t_{dra} < 84.3ns \quad (60)$$

$$t_{dmem} = 120ns \quad (61)$$

$$t_{dsr} < 84.3ns \quad (62)$$

$$T_{total} < 84.3ns + 120ns + 84.3ns = 288.6ns \quad (63)$$

4.2 ANÁLISIS DE RECURSOS UTILIZADOS

La ALU logarítmica utiliza un total de 313 elementos lógicos de 4992, es decir el 6% de las celdas lógicas, de los cuales en realidad son 311, pero debido a que la asignación de terminales utiliza dos celdas lógicas extra, y utiliza 87 pines de 147. En la tabla 16 se muestra en detalle como es la distribución de elementos lógicos por unidad de diseño en VHDL en donde la primera fila se observa el total de 311 elementos lógicos que fueron necesarios para la unidad aritmética lógica.

Nodo de compilación jerárquica	Celdas Lógicas
|alulogaritmica|	311
|banderas:BAND1|	41
|multiplexor:MUX1|	14
|multiplexor:MUX2|	13
|multiplexor:MUX4|	66
|sign:SIGNO1|	1
|sumador:SUM1|	39
|sumador:SUM2|	32
|unidad_control:CONTROL1|	69

Tabla 16. Elementos lógicos utilizados por componente

En la tabla 16 se observa que componentes de la misma entidad, son constituidos por números de elementos lógicos diferentes. Como por ejemplo el “MUX4” ocupa 66 elementos lógicos mientras el “MUX1” ocupa 14, una posible razón es que ocupa mucho más dado que por la síntesis lógica que realiza Quartus II el “MUX3” queda adherido a la lógica del “MUX4”.

En la tabla 17 se muestra en detalle los recursos de interconexión de fila que fueron necesarios para el diseño de la unidad aritmético lógica, donde fueron utilizados 147 de 2496 de estos recursos.

La tarjeta de desarrollo está sobredimensionada porque inicialmente no se tenía conocimiento sobre la cantidad total de recursos utilizados en el diseño de la ALU en el FPGA. Luego de la descripción y compilación del proyecto en el software de desarrollo Quartus II, se llegó a un porcentaje de utilización de recursos de solo el 6%.

Fila	Interconexiones Usadas	Interconexiones Usadas de la Mitad Izquierda	Interconexiones Usadas de la Mitad Derecha
A	3 / 208 (1 %)	0 / 104 (0 %)	0 / 104 (0 %)
B	46 / 208 (22 %)	3 / 104 (2 %)	34 / 104 (32 %)
C	3 / 208 (1 %)	0 / 104 (0 %)	0 / 104 (0 %)
D	20 / 208 (9 %)	0 / 104 (0 %)	29 / 104 (27 %)
E	3 / 208 (1 %)	0 / 104 (0 %)	0 / 104 (0 %)
F	59 / 208 (28 %)	22 / 104 (21 %)	10 / 104 (9 %)
G	0 / 208 (0 %)	1 / 104 (< 1 %)	0 / 104 (0 %)
H	4 / 208 (1 %)	2 / 104 (1 %)	0 / 104 (0 %)
I	1 / 208 (< 1 %)	1 / 104 (< 1 %)	1 / 104 (< 1 %)
J	1 / 208 (< 1 %)	1 / 104 (< 1 %)	1 / 104 (< 1 %)
K	3 / 208 (1 %)	0 / 104 (0 %)	1 / 104 (< 1 %)
L	4 / 208 (1 %)	0 / 104 (0 %)	1 / 104 (< 1 %)
Total	**147 / 2496 (5 %)**	**30 / 1248 (2 %)**	**77 / 1248 (6 %)**

Tabla 17. Recursos de interconexión de fila utilizados

La tabla 18 muestra en detalle los recursos de interconexión de columna utilizados en el diseño. Estos corresponden a las interconexiones de los LABs. Esta tabla fue segmentada para una mejor visualización.

Col.	Conexiones Usadas	Col.	Conexiones Usadas	Col.	Conexiones Usadas	Col.	Conexiones Usadas
1	1 / 24 (4 %)	14	4 / 24 (16 %)	27	3 / 24 (12 %)	40	3 / 24 (12 %)
2	1 / 24 (4 %)	15	2 / 24 (8 %)	28	1 / 24 (4 %)	41	2 / 24 (8 %)
3	0 / 24 (0 %)	16	3 / 24 (12 %)	29	2 / 24 (8 %)	42	5 / 24 (20 %)
4	1 / 24 (4 %)	17	1 / 24 (4 %)	30	2 / 24 (8 %)	43	4 / 24 (16 %)
5	0 / 24 (0 %)	18	1 / 24 (4 %)	31	2 / 24 (8 %)	44	3 / 24 (12 %)
6	2 / 24 (8 %)	19	2 / 24 (8 %)	32	5 / 24 (20 %)	45	3 / 24 (12 %)
7	1 / 24 (4 %)	20	1 / 24 (4 %)	33	4 / 24 (16 %)	46	3 / 24 (12 %)
8	2 / 24 (8 %)	21	1 / 24 (4 %)	34	3 / 24 (12 %)	47	1 / 24 (4 %)
9	3 / 24 (12 %)	22	2 / 24 (8 %)	35	3 / 24 (12 %)	48	2 / 24 (8 %)
10	2 / 24 (8 %)	23	2 / 24 (8 %)	36	0 / 24 (0 %)	49	3 / 24 (12 %)
11	1 / 24 (4 %)	24	2 / 24 (8 %)	37	0 / 24 (0 %)	50	0 / 24 (0 %)
12	2 / 24 (8 %)	25	3 / 24 (12 %)	38	0 / 24 (0 %)	51	5 / 24 (20 %)
13	2 / 24 (8 %)	26	2 / 24 (8 %)	39	1 / 24 (4 %)	52	5 / 24 (20 %)
Total: 109 / 1248 (8%)							

Tabla 18. Recursos de interconexión de columna utilizados

De las tablas 16, 17 y 18, se puede observar que el diseño de esta unidad aritmético lógica puede implementarse sin necesidad de recursos lógicos ni de conexión extras, para el FPGA ACEX1K100-208.

4.3 COSTOS

4.3.1 Costos estimados. En la tabla 19 se observa los costos estimados en el anteproyecto del trabajo de grado, estos costos fueron calculados con base en trabajo de 4 meses. Para el cálculo del costo total son incluidos los conceptos de desarrollo, asesoría de los directores de trabajo de grado, costos de materiales, costos de equipos, etc.

COSTOS PROPIOS			
Costos de Ingeniería	**N meses**	**Salario Mensual**	**Salario Total**
Manuel Carrera	4	$1.500.000	$6.000.000
Efrén López	4	$1.500.000	$6.000.000
Juan Diego Naranjo	4	$1.500.000	$6.000.000
Subtotal			**$18.000.000**
Materiales e Insumos			**Costos**
Dispositivo FUGA			$60.000
Circuito impreso			$300.000
Componentes varios			$140.000
Subtotal			**$500.000**
Servicios	**N meses**	**Costo Mensual**	**Total**
Transporte	4	$200.000	$800.000
Servicios de papelería (fotocopias, impresión, anillado, empastado, CDS)			$240.000
Subtotal			**$1.040.000**
Subtotal Costos de Ingeniería			$18.000.000
Subtotal Materiales e Insumos			$500.000
Subtotal Servicios			$1.040.000
TOTAL PROPIOS			**$19.540.000**
COSTOS UNIVERSIDAD			
Costos de Asesoría	**N meses**	**Servicio Mensual**	**Servicio Total**
Ing. Francisco Viveros	4	$2.500.000	$10.000.000
Ing. Alejandra González	4	$2.000.000	$8.000.000
Subtotal			**$18.000.000**
Equipo necesario	**N meses**	**Costo Mensual**	**Costo Total**
Alquiler de Computador Pentium IV, 1.0GHz, 128MB de RAM, 20GB de Disco Duro. Software: Microsoft Office 2000	4	$700.000	$2.800.000
Fuente de Voltaje	4	$50.000	$200.000
DVM	4	$30.000	$120.000
Osciloscopio digital	4	$200.000	$800.000
Analizador de estados lógicos	4	$400.000	$1.600.000
Subtotal			**$5.520.000**
Servicios	**N meses**	**Costo Mensual**	**Total**
Internet	4	$100.000	$400.000
Energía	4	$80.000	$320.000
Subtotal			**$720.000**

Subtotal Costos de Ingeniería Asesoría	$18.000.000
Subtotal Alquiler de Equipos	$5.520.000
Subtotal Servicios	$720.000
TOTAL UNIVERSIDAD	**$24.240.000**
COSTO DEL PROYECTO	
Total propios	$19.540.000
Total universidad	$24.240.000
TOTAL PROYECTO	**$43.780.000**

Tabla 19. Costos estimados del proyecto

4.3.2 Costos reales. En la tabla 20 se observan los costos reales de este trabajo de grado, y se observa un aumento de $10.478.050, debido al mes adicional de trabajo que fue necesario dedicar al proyecto.

COSTOS PROPIOS			
Costos de Ingeniería	**N meses**	**Salario Mensual**	**Salario Total**
Manuel Carrera	5	$1.500.000	$7.500.000
Efrén López	5	$1.500.000	$7.500.000
Juan Diego Naranjo	5	$1.500.000	$7.500.000
Subtotal			**$22.500.000**
Materiales e Insumos			**Costos**
Dispositivo FUGA			$60.000
Circuito impreso			$147.000
Componentes varios			$75.350
Subtotal			**$282.350**
Servicios	**Cantidad**	**Costo unitario**	**Total**
Transporte	5 meses	$200.000 por mes	$1.000.000
Fotocopias	274 hojas	$50 por hoja	$13.700
Impresión	300 hojas	$300 por hoja	$90.000
Anillado	9	$4.000	$36.000
Empastado	3	$10.000	$30.000
CDS	3	$2.000	$6.000
Subtotal			**$1.175.700**
Subtotal Costos de Ingeniería			$22.500.000
Subtotal Materiales e Insumos			$282.350
Subtotal Servicios			$1.175.700
TOTAL PROPIOS			**$23.958.050**
COSTOS UNIVERSIDAD			
Costos de Asesoría	**N meses**	**Servicio Mensual**	**Servicio Total**
Ing. Francisco Viveros	5	$2.500.000	$12.500.000
Ing. Alejandra González	5	$2.000.000	$10.000.000
Subtotal			**$22.500.000**
Equipo necesario	**N meses**	**Costo Mensual**	**Costo Total**
Alquiler de Computador Pentium IV, 1.0GHz, 128MB de RAM, 20GB de Disco Duro. Software: Microsoft Office 2000	5	$700.000	$3.500.000
Fuente de Voltaje	5	$50.000	$250.000
DVM	5	$30.000	$150.000
Osciloscopio digital	5	$200.000	$1.000.000

Analizador de estados lógicos	5	$400.000	$2.000.000
Subtotal			**$6.900.000**
Servicios	N meses	**Costo Mensual**	**Total**
Internet	5	$100.000	$500.000
Energía	5	$80.000	$400.000
Subtotal			**$900.000**
Subtotal Costos de Ingeniería Asesoría			$22.500.000
Subtotal Alquiler de Equipos			$6.900.000
Subtotal Servicios			$900.000
TOTAL UNIVERSIDAD			**$30.300.000**
COSTO DEL PROYECTO			
Total propios			$23.958.050
Total universidad			$30.300.000
TOTAL PROYECTO			**$54.258.050**

Tabla 20. Costos reales del proyecto

4.4 DIFICULTADES

El poco soporte que el lenguaje de programación JAVA tiene para el uso de puertos de entrada y salida de un PC, impidió que se diera un mejor acabado a la aplicación “Controlador.class”. Aunque existía la librería de extensión para el trabajo con puertos serial y paralelo, llamada “javax.comm”, diferentes formas de empaquetamiento de la aplicación fallaban, debido a que no encontraban la extensión, cuando ésta había sido declarada de forma correcta. Una de esta formas de integración de archivos es “jar.exe”, que permite crear un paquete comprimido, al cual se le puede dar doble clic desde Windows u otro sistema operativo para ejecutar la aplicación comprimida, e incluso la misma aplicación desarrollada en Visual J no permite ser ejecutada fuera del entorno de depuración. Por esta razón la aplicación creada en JAVA “Controlador.class” solo permite ser ejecutada desde el comando de DOS.

La obtención de un microcontrolador con suficientes pines de entrada y salida, en el mercado colombiano, hizo que fuera necesario adquirir este componente en el exterior.

Para el trabajo de un proyecto jerarquizado en varios códigos fuentes, y que estos códigos fueran reutilizables, la herramienta de diseño digital Max Plus II, mostró ser inapropiada dado que su compilador de código VHDL, no soporta llamados a entidades externas.

Una de las mayores dificultades fue la adquisición de un soporte de integrado tipo PLCC de 20 pines para la memoria de configuración EPC2, dado que este tipo de empaque no se ha difundido con fuerza en el mercado colombiano. Esta memoria permitía que el sistema funcionara de manera independiente de un PC. Aunque uno de los objetivos específicos se refería a una implementación que pudiera funcionar independientemente, se decidió que el dispositivo de configuración no fuera

incluido, dado que esto no nos desviaba de nuestro objetivo general, el cual era el desarrollo de una ALU logarítmica.

Un problema que detuvo el desarrollo del trabajo de grado fue un defecto de la tarjeta ALTERIC, En un principio se había decidido utilizar los puertos uno y tres de la tarjeta Alteric para la conexión con la tarjeta que lleva el microcontrolador, el puerto uno de esta tarjeta tenía problemas de continuidad, por lo cual existían ciertos pines del puerto que no hacían conexión con el FPGA. Este problema se corrigió reasignando los pines que se utilizaron, la conexión se hizo entonces con los puertos dos y tres de la tarjeta ALTERIC.

Pero aún así persistían algunos problemas, los dos puertos, el uno y el dos, no poseen la misma configuración de pines con respecto a los pines de fuente y tierra, y era necesario tener cuidado en la conexión de las cintas de comunicación, para que no se hiciera un corto indeseado. Por esto las dos cintas que comunican las dos tarjetas tienen algunas líneas compartidas, estas son las líneas que no podían llegar al puerto dos de la tarjeta ALTERIC debido a que llegaban a pines de potencia, y se enviaron a pines disponibles en el puerto tres.

Un defecto menor de la tarjeta controladora, fue que los puertos de esta tarjeta se situaron muy cerca en el diseño del impreso, de tal manera que una vez puesto un conector el otro no tenía espacio para ser conectado, este problema se arreglo recortando un poco los conectores.

4.5 EVALUACIÓN DE OBJETIVOS

Satisfactoriamente fue desarrollada una arquitectura de una Unidad Aritmética y Lógica, que se apoya en las propiedades de los logaritmos, a este sistema digital le llamamos ALU logarítmica.

Dado que el sistema era más que todo demostrativo, no se cumplió con un criterio técnico para determinar la longitud de la trama, se decidió dar la mejor precisión a los resultados, para observar con mayor detalle el comportamiento de las operaciones.

El sistema fue descrito en VHDL dando como resultado un proyecto de códigos jerarquizados, de tal manera que partes del código fueran reutilizables. Esta descripción de un sistema digital fue implementada en un FPGA de Altera, el dispositivo utilizado es el ACEX EP1K100-208.

Se utilizó una tarjeta ya construida en un trabajo de grado anterior, esta tarjeta trae el dispositivo FPGA, reguladores de 2.5 v y 3.3 v que necesita el dispositivo, un puerto de configuración y varios puertos que permiten la conexión de señales digitales con el dispositivo, a esta tarjeta se le adaptó dos tarjetas, una de ellas lleva las memorias con las tablas de funciones, y la otra tarjeta trae un

microcontrolador, que permite la interacción entre un usuario y la ALU logarítmica. Para facilitar el uso de este sistema se desarrolló una aplicación que permite ver los datos que entran a la unidad y los resultados que esta unidad realiza.

CONCLUSIONES

- Debido a la utilización de exponentes para representar los números en formato de punto flotante y en formato logarítmico, un número en base logarítmica es una representación simplificada de formato en punto flotante, en este caso un número se representa por un bit de signo y un exponente, omitiendo la mantisa. Dado que se omite la mantisa se requiere un bit adicional para representar el cero. Los dos formatos presentan mucha similitud, sobre todo en las operaciones de producto y división donde los exponentes se suman y restan respectivamente.
- Existe un error en el resultado de las operaciones debido al fenómeno de cuantización, el margen de error de las operaciones, es mayor cuando los datos son números más grandes, entre mayor es el número del formato logarítmico el intervalo entre el número que éste representa y el siguiente se hace más grande. Para procesamiento de señales debe analizarse si este fenómeno afecta o no a la aplicación.
- No es recomendable desarrollar aplicaciones en el lenguaje de programación JAVA, que requieren el uso de puertos serial o paralelo, dado que las aplicaciones, aunque pueden funcionar, solo pueden ser ejecutadas desde el comando de DOS.
- La unidad final utiliza los recursos del FPGA ACEX1K100-208, mostrados en la tabla 21.

Recursos Utilizados	Numero de recursos
Celdas Lógicas	313 / 4.992 (6 %)
Celdas Lógicas con registro	0 / 4.992 (0 %)
Terminales entrada y salida	87 / 147 (59 %)
Interconexiones de columna	109 / 1.248 (8 %)
Interconexiones de fila	147 / 2.496 (5 %)
Bits de memoria	0 / 49.152
Cadenas en cascada	9 de longitud 2

Tabla 21. Sumario de recursos utilizados

- El tiempo de retardo estimado es 288.6 ns y el tiempo de retardo medido es 687.97 ns, el tiempo de retardo medido es 138% mayor que el tiempo estimado, teniendo en cuenta que en el tiempo de estimado no se incluyen los retardos causados por los caminos en los circuitos impresos y las cintas.
- La velocidad de operación de la unidad aritmética lógica, en el peor de los casos es 687.97 ns. La velocidad es afectada por la utilización de dos memorias externas para la implementación de las tablas de función LUT (look up Table).

BIBLIOGRAFÍA

"ACEX 1K, Programmable logic device family". San Jose CA: Altera, 2003.

ANGARITA PRECIADO, Fabián Enrique; GUERRA RODRÍGUEZ, Ruy Pelayo. "Tarjeta de desarrollo para el procesador RIC-2000". Bogota, 2001. Trabajo de grado (Ingeniero Electrónico). Pontificia Universidad Javeriana. Facultad de Ingeniería. Carrera de Ingeniería Electrónica.

ARMSTRONG, James R. "Structured logic design with VHDL". New Jersey: Prentice Hall, 1993.

ARNOLD, Mark G. "A VLIW Architecture for Logarithmic Arithmetic", Lehigh University, 2003.

ARNOLD, Mark G. & RUAN, Jie. "Combined Adder/Substractors for DCT Hardware", Lehigh University,

BELTRAN, Diego; HERRERA, Moisés & MAYOLO, Marco. "Implementación del procesador Alteric". Bogota, 2004. Trabajo de grado (Ingeniero Electrónico). Pontificia Universidad Javeriana. Facultad de Ingeniería. Carrera de Ingeniería Electrónica.

"Configuration Handbook". San José CA: Altera, 2004.

CHACÓN, Diego Joaquín; MERCADO, Oneida; NAVARRO, Fausto Enrique & PALLARES, Jorge Iván. "Procesador didáctico E-RIC". Bogota, 2000. Trabajo de grado (Ingeniero Electrónico). Pontificia Universidad Javeriana. Facultad de Ingeniería. Carrera de Ingeniería Electrónica.

FERGUSON, M. I. & ERCEGOVAC, Milos. "A Multiplier with Redundant Operands", Computer Science Department, University of California, Los Angeles. 1999.

"Introduction to QUARTUS II". San José CA: Altera, 2004.

KADLEC J., HERMANEK A., SOFTLEY Ch., MATOUSEK R., & LICKO M. "32-bit Logarithmic ALU for Andel C 2.1 and Celoxica DK1", University of Newcastle.

KADLEC J., HERMANEK A., MATOUSEK R., & LICKO M. "FPGA implementation of logarithmic unit", Katedra Telekomunikaci FEL CVUT, Praha.

KOREN, Israel. “Computer arithmetic algorithms”. Natiks: AK Peters. 2002

KOU, Weidong. “Digital image compression: algorithms and standards”. Boston: Kluwer Academic, 1995.

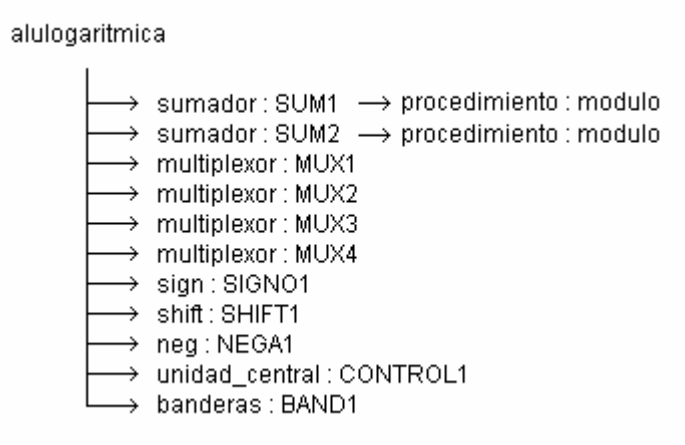
LEMAY, Laura; PERKINS, Charles & MORRISON, Michael. “Teach yourself JAVA in 21 days”. Sams net, 1996.

“Microchip PIC 18F8720 Datasheet”. Microchip, 2004.

“QUARTUS II Handbook”. San Jose CA: Altera, 2004.

ANEXO A DESCRIPCIÓN EN VHDL DE LA ALU LOGARÍTMICA

Jerarquía de llamado de entidades:



ALULOGARITMICA.VHD

```
-- ********************************************************************************
-- * TÍTULO:          ALU LOGARÍTMICA                                             *
-- * T.G.:            0432                                                        *
-- *                                                                              *
-- * INTEGRANTES:     MANUEL ALBERTO CARRERA OSORIO                               *
-- *                  EFRÉN OCTAVIO LÓPEZ URIBE                                   *
-- *                  JUAN DIEGO NARANJO LÓPEZ                                    *
-- *                                                                              *
-- * DIRECTORES:      ING. ALEJANDRA MARIA GONZÁLEZ                               *
-- *                  ING. FRANCISCO VIVEROS                                      *
-- *                                                                              *
-- * ENTIDAD:         alulogaritmica.vhd                                          *
-- *                                                                              *
-- * DESCRIPCIÓN:     Entidad que cumple la función integrar todas las unidades   *
-- *                  del la ALU, prestar la conectividad de las señales externas*
-- *                  hacia las unidades funcionales y servir de interfaz hacia   *
-- *                  el sistema usuario.                                         *
-- *                                                                              *
-- * MODIFICACIONES:                                                              *
-- *                                                                              *
--********************************************************************************


LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;
USE work.sumador.ALL;
USE work.neg.ALL;
USE work.multiplexor.ALL;
USE work.banderas.ALL;
USE work.shift.ALL;
USE work.sign.ALL;
USE work.unidad_ central.ALL;

ENTITY alulogaritmica IS
       PORT(
```

```
                -- DATOS DE ENTRADA

                -- Entrada del microcontrolador dato.
                RA      : IN STD_LOGIC_VECTOR(15 DOWNTO 0);

                -- Entrada del microcontrolador dato.
                RB      : IN STD_LOGIC_VECTOR(15 DOWNTO 0);

                -- Entrada del microcontrolador codigo de operación.
                CODOP   : IN STD_LOGIC_VECTOR(2 DOWNTO 0);

                -- Entrada de la memoria operación de la suma o resta.
                SLUTSR  : IN STD_LOGIC_VECTOR(13 DOWNTO 0);

                -- DATOS DE SALIDA

                -- Salida de señal de control de la LUT.
                --CTROM : OUT STD_LOGIC;

                -- Salida para direccionar datos en la memoria (LUT).
                ADD     : OUT STD_LOGIC_VECTOR(13 DOWNTO 0);
                STAB    : OUT STD_LOGIC;

                -- Salida de banderas del sistema.
                BAND    : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);

                -- Salida de la ALU.
                RC      : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
        );
END alulogaritmica;

ARCHITECTURE alulog OF alulogaritmica IS


        -- Instancias de los componentes que interconectan a las entidades declaradas en
        -- el directorio de trabajo (work) con el sistema global de la ALU.

        -- Unidad sumadora
        COMPONENT puerto_sumador
                PORT (
                        A,B     : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
                        CIN     : IN STD_LOGIC;
                        SUM     : OUT STD_LOGIC_VECTOR(13 DOWNTO 0);
                        OV      : OUT STD_LOGIC
                        );
        END COMPONENT;

        -- Unidad negadora
        COMPONENT puerto_negador
                PORT(
                        EN      : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
                        SAL     : OUT STD_LOGIC_VECTOR(13 DOWNTO  0)
        );
        END COMPONENT;

-- Unidad multiplexora
-- ****************************************
-- *    SEL     *      DATO DE SALIDA       *
-- ****************************************
-- *     0      *           A               *
-- *     1      *           B               *
-- ****************************************

        COMPONENT puerto_mux
                PORT(
                        A,B     : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
                        SEL     : IN STD_LOGIC;
                        C       : OUT STD_LOGIC_VECTOR(13 DOWNTO  0)                    );
        END COMPONENT;
```

```
-- Unidad de banderas
-- ********************************************************
-- *    FLAGS   *            EVENTO                       *
-- ********************************************************
-- *      0     * Overvlow de alguno de los sumadores      *
-- *      1     * Resultado es + INFINITO                  *
-- *      2     * Resultado es - INFINITO                  *
-- *      3     * Resultado es NON                         *
-- *      4     * Cambio de signo en corrimiento           *
-- *      5     * Bandera de división por cero             *
-- *      6     * Bandera de número imaginario             *
-- ********************************************************

        COMPONENT puerto_banderas
                PORT(
                        OPCODE          : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
                        DBITSIA         : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                        DBITSIB         : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                        COUTSUM1        : IN STD_LOGIC;
                        COUTSUM2        : IN STD_LOGIC;
                        LSBA            : IN STD_LOGIC;
                        FLAGS           : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
        );
        END COMPONENT;

-- Unidad shift
-- ********************************************************************************
-- *CTSUIN * OPERACIÓN                                                            *
-- ********************************************************************************
-- *  000  * La salida es el dato de entrada                                      *
-- *  001  * La salida es la negación de la entrada                               *
-- *  010  * La salida corresponde a INFINITO                                     *
-- *  011  * La salida es el bit 13 negado y los demás bits de entrada            *
-- *  101  * La salida corresponde al corrimiento a izquierda con LSB(out) = MSB(in) *
-- *  110  * La salida corresponde al corrimiento a derecha con MSB(out) = MSB-1(in) *
-- ********************************************************************************

        COMPONENT puerto_shift
                PORT(
                        A               : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
                        CTSUIN          : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
                        AS              : OUT STD_LOGIC_VECTOR(13 DOWNTO 0)
                        );
        END COMPONENT;

-- Unidad de signo
-- ************************************************
-- *   CSIGN   *      MANEJO                      *
-- ************************************************
-- *    000    * La salida es el signo del dato A  *
-- *    001    * La salida es el signo del dato B  *
-- *    010    * La salida es A15 y NOT A14        *
-- *    011    * La salida es NOT A15 y A14        *
-- *    100    * La salida es B15 y NOT B14        *
-- *    101    * La salida es NOT A15 y NOT A14    *
-- *    11X    * La salida es A15 y (A14 XOR B14)  *
-- ************************************************

        COMPONENT puerto_sign
                PORT(
                        ZSA             : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                        ZSB             : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                        CSIGN           : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
                        ZSC             : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)                );
        END COMPONENT;

        -- Unidad de control
        COMPONENT puerto_unidad_central
                PORT(
                        OPCODEIN        : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
                        BEA             : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
```

```
                        BEB             : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                        ESUM1           : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
                        CTMUX1          : OUT STD_LOGIC;
                        CTMUX2          : OUT STD_LOGIC;
                        CTMUX3          : OUT STD_LOGIC;
                        CTMUX4          : OUT STD_LOGIC;
                        CTSIGN          : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                        CTSU            : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                        TAB             : OUT STD_LOGIC
                        );
        END COMPONENT;

        -- Declaración necesaria para identificar que entidad se interconecta con cada
        -- componente declarado anteriormente.
        FOR ALL : puerto_sumador USE ENTITY work.sumador;
        FOR ALL : puerto_negador USE ENTITY work.neg;
        FOR ALL : puerto_mux USE ENTITY work.multiplexor;
        FOR ALL : puerto_banderas USE ENTITY work.banderas;
        FOR ALL : puerto_shift USE ENTITY work.shift;
        FOR ALL : puerto_sign USE ENTITY work.sign;
        FOR ALL : puerto_unidad_central USE ENTITY work.unidad_central;



                -- Declaración de constantes
                CONSTANT BTUNO          : STD_LOGIC := '1';
                CONSTANT BTCERO         : STD_LOGIC := '0';


                -- Señales de interconexión de datos dentro de la ALU.
                SIGNAL SSAL             : STD_LOGIC_VECTOR(13 DOWNTO 0);
                SIGNAL SUM1T            : STD_LOGIC_VECTOR(13 DOWNTO 0);
                SIGNAL SUM2T            : STD_LOGIC_VECTOR(13 DOWNTO 0);
                SIGNAL SMUX1            : STD_LOGIC_VECTOR(13 DOWNTO 0);
                SIGNAL SMUX2            : STD_LOGIC_VECTOR(13 DOWNTO 0);
                SIGNAL SMUX3            : STD_LOGIC_VECTOR(13 DOWNTO 0);
                SIGNAL SSHIFT           : STD_LOGIC_VECTOR(13 DOWNTO 0);

                -- Señales de acarreo de los sumadores.
                SIGNAL COSUM1           : STD_LOGIC;
                SIGNAL COSUM2           : STD_LOGIC;

                -- Señales de control.
                        -- De los multiplexores.
                SIGNAL SCTMX1           : STD_LOGIC;
                SIGNAL SCTMX2           : STD_LOGIC;
                SIGNAL SCTMX3           : STD_LOGIC;
                SIGNAL SCTMX4           : STD_LOGIC;

                -- De la unidad de signo (SIGN).
                SIGNAL SCTSIGN          : STD_LOGIC_VECTOR(2 DOWNTO 0);

                -- De la unidad de corrimiento (SHIFT).
                SIGNAL SCTSU            : STD_LOGIC_VECTOR(2 DOWNTO 0);


                        BEGIN

                        ADD(13 DOWNTO 0)        <= SUM1T(13 DOWNTO 0);

                        NEGA1           : puerto_negador
                                PORT MAP( EN(13 DOWNTO 0) => RB(13 DOWNTO 0), SAL(13 DOWNTO 0)
=> SSAL(13 DOWNTO 0) );

                        SUM1            : puerto_sumador
                                PORT MAP( A(13 DOWNTO 0) => RA(13 DOWNTO 0), B(13 DOWNTO 0) =>
SSAL(13 DOWNTO 0), CIN => BTUNO, SUM => SUM1T, OV => COSUM1 );

                        SUM2            : puerto_sumador
                                PORT MAP( A(13 DOWNTO 0) => SMUX2(13 DOWNTO 0), B(13 DOWNTO 0)
=> SMUX1(13 DOWNTO 0), CIN => BTCERO, SUM => SUM2T, OV => COSUM2 );
```

```
                        MUX1            : puerto_mux
                                PORT MAP( A(13 DOWNTO 0) => RA(13 DOWNTO 0), B(13 DOWNTO 0)
        => RB(13 DOWNTO 0), SEL => SCTMX1, C(13 DOWNTO 0) => SMUX1(13 DOWNTO 0) );

                        MUX2            : puerto_mux
                                PORT MAP( A(13 DOWNTO 0) => RA(13 DOWNTO 0), B(13 DOWNTO 0)
        => SLUTSR(13 DOWNTO 0), SEL => SCTMX2, C(13 DOWNTO 0) => SMUX2(13 DOWNTO 0) );

                        MUX3            : puerto_mux
                                PORT MAP( A(13 DOWNTO 0) => SUM2T(13 DOWNTO 0), B(13 DOWNTO 0)
=> SUM1T(13 DOWNTO 0), SEL => SCTMX3, C(13 DOWNTO 0)       => SMUX3(13 DOWNTO 0) );

                        MUX4            : puerto_mux
                                PORT MAP( A(13 DOWNTO 0) => SMUX3(13 DOWNTO 0), B(13 DOWNTO 0)
=> SSHIFT(13 DOWNTO 0), SEL => SCTMX4, C(13 DOWNTO 0) => RC(13 DOWNTO 0) );

                        SHIFT1          : puerto_shift
                                PORT MAP( A(13 DOWNTO 0) => SMUX1(13 DOWNTO 0), CTSUIN(2
DOWNTO 0) => SCTSU(2 DOWNTO 0), AS(13 DOWNTO 0) => SSHIFT(13 DOWNTO 0)    );

                        BAND1           : puerto_banderas
                                PORT MAP( OPCODE(2 DOWNTO 0) => CODOP(2 DOWNTO 0), DBITSIA(3
DOWNTO 0) => RA(15 DOWNTO 12), DBITSIB(3 DOWNTO 0)  => RB(15 DOWNTO 12), COUTSUM1 => COSUM1,
COUTSUM2 => COSUM2, LSBA => RA(0), FLAGS(6 DOWNTO 0) => BAND(6 DOWNTO 0) );

                        SIGNO1          : puerto_sign
                                PORT MAP( ZSA(1 DOWNTO 0) => RA(15 DOWNTO 14), ZSB(1 DOWNTO 0)
=> RB(15 DOWNTO 14), CSIGN(2 DOWNTO 0) => SCTSIGN(2 DOWNTO 0), ZSC(1 DOWNTO 0) => RC(15
DOWNTO 14) );

                        CONTROL1        : puerto_unidad_central
                                PORT MAP( OPCODEIN(2 DOWNTO 0) => CODOP(2 DOWNTO 0), BEA(3
DOWNTO 0) => RA(15 DOWNTO 12), BEB(3 DOWNTO 0) => RB(15 DOWNTO 12), ESUM1(13 DOWNTO 0) =>
SUM1T(13 DOWNTO 0), CTMUX1 => SCTMX1, CTMUX2 => SCTMX2, CTMUX3 => SCTMX3, CTMUX4 => SCTMX4,
CTSIGN(2 DOWNTO 0) => SCTSIGN(2 DOWNTO 0), CTSU(2 DOWNTO 0) => SCTSU(2 DOWNTO 0), TAB =>
STAB );

END alulog;
```

UNIDAD_CENTRAL.VHD

```
-- ********************************************************************************
-- * TÍTULO:           ALU LOGARÍTMICA                                            *
-- * T.G.:             0432                                                       *
-- *                                                                              *
-- * INTEGRANTES:      MANUEL ALBERTO CARRERA OSORIO                              *
-- *                   EFRÉN OCTAVIO LÓPEZ URIBE                                  *
-- *                   JUAN DIEGO NARANJO LÓPEZ                                   *
-- *                                                                              *
-- * DIRECTORES:       ING. ALEJANDRA MARIA GONZÁLEZ                              *
-- *                   ING. FRANCISCO VIVEROS                                     *
-- *                                                                              *
-- * ENTIDAD:          unidad_central.vhd                                         *
-- *                                                                              *
-- * DESCRIPCIÓN:      Entidad que cumple la función de controlar las señales de  *
-- *                   todos los multiplexores de la ALU, de la unidad "shift",   *
-- *                   de la unidad "sign" y de las tablas de LOOK-UP.            *
-- *                                                                              *
-- * MODIFICACIONES:                                                              *
-- *                                                                              *
-- ********************************************************************************

LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;

ENTITY unidad_central IS
        PORT(
```

```
-- DATOS DE ENTRADA
-- Codigo de operación que será realizada por la ALU
-- ************************************************
-- *   OPCODE  *    OPERACIÓN                     *
-- ************************************************
-- *    000    *      SUMA                        *
-- *    001    *      RESTA                       *
-- *    010    *      MULTIPLICACIÓN              *
-- *    011    *      DIVISIÓN                    *
-- *    101    *      CORR_IZQ_RED                *
-- *    110    *      CORR_DER_SIN_RED            *
-- ************************************************

               OPCODEIN      : IN STD_LOGIC_VECTOR(2 DOWNTO 0);

-- Bits mas significativos de los datos de entrada del dato A.
-- ************************************************************************
-- *   BEA     *      3      *      2      *      1      *      0      *
-- *   A       *      15     *      14     *      13     *      12     *
-- *    ?      *    BIT CERO *   BIT SIGNO *    BIT DATO *    BIT DATO *
-- ************************************************************************

               BEA           : IN STD_LOGIC_VECTOR(3 DOWNTO 0);

-- Bits mas significativos de los datos de entrada del dato B.
-- ************************************************************************
-- *   BEB     *      3      *      2      *      1      *      0      *
-- *   B       *      15     *      14     *      13     *      12     *
-- *    ?      *    BIT CERO  *   BIT SIGNO  *    BIT DATO *    BIT DATO  *
-- ************************************************************************

               BEB           : IN STD_LOGIC_VECTOR(3 DOWNTO 0);

-- Datos de la salida del sumador 1, usados para reconocer el cero y
-- que indica el resultado de la resta de RA - RB.
-- ***************************
-- * ESUM1(13) *   0   *   1   *
-- * Resultado *  POS  *  NEG  *
-- ***************************

               ESUM1         : IN STD_LOGIC_VECTOR(13 DOWNTO 0);

-- DATOS DE SALIDA
-- Salidas de señales de control de los multiplexores MUX1, MUX2, MUX3 y MUX4;
--
-- ***************************
-- * CASO      *   0   *   1   *
-- * SALIDA    *   A   *   B   *
-- ***************************
-- La siguiente tabla corresponde a las señales de la entrada SEL de cada uno
-- de los multiplexores.
--
-- ******************************************************
-- * OPERACIÓN        *    MUX1      * MUX2  * MUX3  * MUX4  *
-- ******************************************************
-- * SUMA             *   MSBSUM1    *   1   *   0   *   0   *
-- * RESTA            *   MSBSUM1    *   1   *   0   *   0   *
-- * MULT.            *     1        *   0   *   0   *   0   *
-- * DIV.             *     x        *   x   *   1   *   0   *
-- ******************************************************
-- * CASOS            *    0 ó 1     *   x   *   x   *   1   *
-- * ESPECIALES       *              *       *       *       *
-- ******************************************************
-- NOTA : Los casos especiales surgen cuando se los datos de entrada son + INF,
--  -INF, NON y/o CERO, y de acuerdo a la operación indicada para con los mismo.
--  MSBSUM1 corresponde al bit mas significativo a la salida del sumador 1, el mismo
--  ESUM(13).

               CTMUX1        : OUT STD_LOGIC;
               CTMUX2        : OUT STD_LOGIC;
```

```
                CTMUX3          : OUT STD_LOGIC;
                CTMUX4          : OUT STD_LOGIC;

                -- Salida de señal de control de la unidad SIGN.
                CTSIGN          : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);

                -- Salida de señal de control de la unidad SHIFT.
                CTSU            : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);

                -- Salida de señal de control acceso a la correspondiente tabla de las
        memorias.
                TAB             : OUT STD_LOGIC

                );
END unidad_ central;

ARCHITECTURE alucontrol OF unidad_central IS

        BEGIN
        PROCESS(OPCODEIN,BEA,BEB,ESUM1(13))

-- DECLARACIÓN Y DESCRIPCIÓN DE VARIABLES
-- Las variables "MASINFA" y "MASINFB" indican si alguno de los datos de entrada es un
-- + INFINITO; "MENOSINFA" y "MENOSINFB" indican si alguno de los datos de entrada es un
-- - INFINITO; "ZEROA" y "ZEROB" indican si alguno de los datos de entrada es un CERO;
-- "NONA" y "NONB" indican si alguno de los datos de entrada es un "not-a-number";
-- "POSA" y "POSB" indican si los datos de entrada son positivos; "NEGA" y "NEGB" indican
-- si los datos de entrada son negativos; "CES" indica si la entrada es un caso especial
-- para la suma; "CER" indica si la entrada es un caso especial para la resta; y "MAX"
-- indica cuando los datos |A| > |B| o |A| < |B|.
-- NOTA : Todas las señales indican el correspondiente evento descrito anteriormente
-- con un uno (1) lógico.

        VARIABLE MASINFA, MASINFB           : STD_LOGIC;
        VARIABLE MENOSINFA, MENOSINFB       : STD_LOGIC;
        VARIABLE ZEROA, ZEROB               : STD_LOGIC;
        VARIABLE NONA, NONB                 : STD_LOGIC;
        VARIABLE POSA, POSB                 : STD_LOGIC;
        VARIABLE NEGA, NEGB                 : STD_LOGIC;
        VARIABLE CES, CER, MAX              : STD_LOGIC;
        VARIABLE ZSUM                       : STD_LOGIC;

                BEGIN

                MASINFA         := BEA(3) AND NOT(BEA(2) OR BEA(1)) AND BEA(0);
                MASINFB         := BEB(3) AND NOT(BEB(2) OR BEB(1)) AND BEB(0);
                MENOSINFA       := BEA(3) AND BEA(2) AND (NOT BEA(1)) AND BEA(0);
                MENOSINFB       := BEB(3) AND BEB(2) AND (NOT BEB(1)) AND BEB(0);
                ZEROA           := BEA(3) AND NOT (BEA(2) OR BEA(1) OR BEA(0));
                ZEROB           := BEB(3) AND NOT (BEB(2) OR BEB(1) OR BEB(0));
                NONA            := BEA(3) AND BEA(1) AND (NOT BEA(0));
                NONB            := BEB(3) AND BEB(1) AND (NOT BEB(0));
                POSA            := NOT(BEA(3) OR BEA(2));
                POSB            := NOT(BEB(3) OR BEB(2));
                NEGA            := NOT(BEA(3)) AND BEA(2);
                NEGB            := NOT(BEB(3)) AND BEB(2);
                CES             := NONA OR (MASINFA AND (MENOSINFB OR MASINFB OR POSB OR NEGB
OR ZEROB)) OR (MENOSINFA AND (POSB OR NEGB OR ZEROB)) OR (ZEROB AND (POSA OR NEGA));
                CER             := NONB OR (MENOSINFB AND (MENOSINFA OR POSA OR NEGA OR ZEROA
OR MASINFA)) OR (MASINFB AND (MASINFA OR POSA OR NEGA OR ZEROA)) OR (ZEROA AND (POSB OR
NEGB));
                MAX             := (POSA OR NEGA) AND (POSB OR NEGB);
                ZSUM            := ESUM1(13) OR ESUM1(12) OR ESUM1(11) OR ESUM1(10) OR
ESUM1(9) OR ESUM1(8) OR ESUM1(7) OR ESUM1(6) OR ESUM1(5) OR ESUM1(4) OR ESUM1(3) OR ESUM1(2)
OR ESUM1(1) OR ESUM1(0);

                TAB             <= (OPCODEIN(0) AND ((BEA(2) AND BEB(2)) OR ( (NOT BEA(2)) AND
(NOT BEB(2))))) OR ((NOT OPCODEIN(0)) AND (BEA(2) XOR BEB(2)));

                CASE OPCODEIN IS
                        -- Operación de suma
```

```
                    WHEN "000" =>
                        IF ((NOT ZSUM) AND (NOT CES)  AND ((POSA AND NEGB) OR (POSB
AND NEGA))) = '1' THEN
                            CTMUX1        <= 'X';
                            CTMUX2        <= 'X';
                            CTMUX3        <= '1';
                            CTMUX4        <= '0';
                            CTSIGN(0)     <= '1';
                            CTSIGN(1)     <= POSA;
                            CTSIGN(2)     <= NOT POSA;
                            CTSU(0)       <= '0';
                            CTSU(1)       <= '0';
                            CTSU(2)       <= '0';
                        ELSE
                            CTMUX1        <= ((NOT MAX) AND (NOT CES)) OR (MAX AND
ESUM1(13));
                            CTMUX2        <= MAX;
                            CTMUX3        <= NOT MAX;
                            CTMUX4        <= NOT MAX;
                            CTSIGN(0)     <= ((NOT MAX) AND (NOT CES)) OR (MAX AND
ESUM1(13));
                            CTSIGN(1)     <= '0';
                            CTSIGN(2)     <= '0';
                            CTSU(0)       <= (MASINFA AND MENOSINFB) OR (MENOSINFA
AND MASINFB);
                            CTSU(1)       <= '0';
                            CTSU(2)       <= '0';
                        END IF;

                    -- CTLUT<=
                    -- Operación de resta
                    WHEN "001" =>
                        IF ((NOT ZSUM) AND (NOT CER) AND ((POSA AND POSB) OR (NEGA AND
NEGB))) = '1' THEN
                            CTMUX1 <= 'X';
                            CTMUX2 <= 'X';
                            CTMUX3 <= '1';
                            CTMUX4 <= '0' OR (MASINFB AND MENOSINFA);
                            CTSIGN(0)     <= '1' AND (NOT (MASINFB AND
MENOSINFA));
                            CTSIGN(1)     <= POSA AND (NOT NEGA);
                            CTSIGN(2)     <= (NOT POSA) AND NEGA;
                            CTSU(0)<= '0';
                            CTSU(1)<= '0';
                            CTSU(2)<= '0';
                        ELSE
                            CTMUX1 <= ((NOT MAX) AND CER) OR (MAX AND ESUM1(13));
                            CTMUX2 <= MAX;
                            CTMUX3 <= NOT MAX;
                            CTMUX4 <= NOT MAX;
                            CTSIGN(0)     <= (CER AND (NOT (MASINFA AND
MENOSINFB))) AND ((((NOT MAX) AND ( CER)) OR (MAX AND ESUM1(13))) AND (NOT (ZEROA AND
(MASINFB OR MENOSINFB OR POSB OR NEGB)))) AND (NOT (MASINFB AND NEGA)) AND (NOT (MENOSINFB
AND POSA));
                            CTSIGN(1)     <= '0';
                            CTSIGN(2)     <= ((POSA OR NEGA OR ZEROA) AND (MASINFB
OR MENOSINFB)) OR ((ZEROA OR ESUM1(13)) AND (POSB OR NEGB));
                            CTSU(0)<= (MASINFA AND MASINFB) OR (MENOSINFA AND
MENOSINFB);
                            CTSU(1)<= '0';
                            CTSU(2)<= '0';
                        END IF;

                    -- CTLUT<=
                    -- Operación de multiplicación
                    WHEN "010" =>
                        CTMUX1 <= NOT (NONA OR (MASINFA AND (POSB OR NEGB)) OR (ZEROA
AND (MASINFB OR MENOSINFB OR POSB OR NEGB OR ZEROB)) OR (MENOSINFA AND (POSB OR NEGB OR
MENOSINFB)));
                        CTMUX2 <= NOT ((POSA OR NEGA) AND (POSB OR NEGB));
                        CTMUX3 <= NOT ((POSA OR NEGA) AND (POSB OR NEGB));
```

```
                                CTMUX4  <= NOT ((POSA OR NEGA) AND (POSB OR NEGB));
                                CTSU(0)<= (MASINFA AND MENOSINFB) OR (MENOSINFA AND MASINFB);
                                CTSU(1)<= '0';
                                CTSU(2)<= '0';
                                        IF (NEGB AND (MASINFA OR MENOSINFA)) = '1' THEN
                                                CTSIGN(2 DOWNTO 0) <= "010";
                                        ELSIF (NEGA AND (MASINFB OR MENOSINFB)) = '1' THEN
                                                CTSIGN(2 DOWNTO 0) <= "100";
                                        ELSIF ((POSA OR NEGA) AND (POSB OR NEGB)) = '1' THEN
                                                CTSIGN(2 DOWNTO 0) <= "11X";
                                        ELSIF (NONA OR (MASINFA AND (POSB OR MASINFB)) OR
(ZEROA AND (MASINFB OR MENOSINFB OR POSB OR NEGB OR ZEROB)) OR (MENOSINFA AND (POSB OR
MENOSINFB))) = '1' THEN
                                                CTSIGN(2 DOWNTO 0) <= "000";
                                        ELSE
                                                CTSIGN(2 DOWNTO 0) <= "001";
                                        END IF;
                        -- CTLUT<=
                        -- Operación de división
                        WHEN "011" =>
                                CTMUX1  <= NONB OR ((MENOSINFA OR POSA OR NEGA OR ZEROA) AND
(MASINFB OR MENOSINFB)) OR (ZEROB AND (POSA OR NEGA));
                                CTMUX2  <= 'X';
                                CTMUX3  <= (POSA OR NEGA) AND (POSB OR NEGB);
                                CTMUX4  <= NOT ((POSA OR NEGA) AND (POSB OR NEGB));
                                        IF (ZEROA AND ZEROB) = '1' THEN
                                                CTSU(2 DOWNTO 0) <= "011";
                                        ELSIF (ZEROB AND (POSA OR NEGA)) = '1' THEN
                                                CTSU(2 DOWNTO 0) <= "010";
                                        ELSIF ((MENOSINFB OR MASINFB) AND (MASINFA OR MENOSINFA
OR POSA OR NEGA OR ZEROA)) = '1' THEN
                                                CTSU(2 DOWNTO 0) <= "001";
                                        ELSE
                                                CTSU(2 DOWNTO 0) <= "000";
                                        END IF;

                                        IF ((MASINFA OR MENOSINFA) AND NEGB) = '1' THEN
                                                CTSIGN(2 DOWNTO 0) <= "010";
                                        ELSIF (NEGA AND ZEROB) = '1' THEN
                                                CTSIGN(2 DOWNTO 0) <= "100";
                                        ELSIF ((POSA OR NEGA) AND (POSB OR NEGB)) = '1' THEN
                                                CTSIGN(2 DOWNTO 0) <= "11X";
                                        ELSIF (NONB OR ((MENOSINFA OR POSA OR NEGA OR ZEROA)
AND (MASINFB OR MENOSINFB)) OR (POSA AND ZEROB)) = '1' THEN
                                                CTSIGN(2 DOWNTO 0) <= "001";
                                        ELSE
                                                CTSIGN(2 DOWNTO 0) <= "000";
                                        END IF;

                        -- CTLUT<=
                        -- Operación de corrimiento a izquierda con redondeo de MSB
                        WHEN "101" =>
                                CTMUX1  <= '0';
                                CTMUX2  <= 'X';
                                CTMUX3  <= 'X';
                                CTMUX4  <= '1';

                                        IF NEGA = '1' THEN
                                                CTSIGN(2 DOWNTO 0)    <= "010";
                                        ELSE
                                                CTSIGN(2 DOWNTO 0)    <= "000";
                                        END IF;

                                        IF (MASINFA OR MENOSINFA OR NONA) = '1' THEN
                                                CTSU(2 DOWNTO 0) <= "000";
                                        ELSE
                                                CTSU(2 DOWNTO 0) <= "101";
                                        END IF;

                        -- CTLUT<=
                        -- Operación de corrimiento a derecha con inclusión de MSB.
```

```
                        WHEN "110" =>
                                CTMUX1                          <= '0';
                                CTMUX2                          <= 'X';
                                CTMUX3                          <= 'X';
                                CTMUX4                          <= '1';
                                CTSIGN(2 DOWNTO 0)      <= "000";

                                        IF (MASINFA OR MENOSINFA OR NONA) = '1' THEN
                                                CTSU(2 DOWNTO 0) <= "000";
                                        ELSE
                                                CTSU(2 DOWNTO 0) <= "110";
                                        END IF;
                        WHEN OTHERS =>
                                CTMUX1  <= '0';
                                CTMUX2  <= 'X';
                                CTMUX3  <= 'X';
                                CTMUX4  <= '1';
                                CTSIGN(2 DOWNTO 0)      <= "000";
                                CTSU(2 DOWNTO 0)        <= "000";
                                -- CTLUT<=
                END CASE;
        END PROCESS;

END alucontrol;
```

SUMADOR.VHD

```
-- ********************************************************************************
-- * TÍTULO:          ALU LOGARÍTMICA                                             *
-- * T.G.:            0432                                                        *
-- *                                                                              *
-- * INTEGRANTES:     MANUEL ALBERTO CARRERA OSORIO                               *
-- *                  EFRÉN OCTAVIO LÓPEZ URIBE                                   *
-- *                  JUAN DIEGO NARANJO LÓPEZ                                    *
-- *                                                                              *
-- * DIRECTORES:      ING. ALEJANDRA MARIA GONZÁLEZ                               *
-- *                  ING. FRANCISCO VIVEROS                                      *
-- *                                                                              *
-- * ENTIDAD:         sumador.vhd                                                 *
-- *                                                                              *
-- * DESCRIPCIÓN:     Entidad que unifica tres módulos de tipo "sumacla" para     *
-- *                  la implementación de la unidad sumadora de tipo carry-look-*
-- *                  ahead, se utiliza 2 módulos de 4 bits y uno de 6 bits, los  *
-- *                  módulos de 6 bits son utilizados para sumar los bits de     *
-- *                  datos ubicados en el centro del formato (del 4 al 9 bit),   *
-- *                  y los módulos de 4 bits son utilizados para relizar la suma*
-- *                  de los bits de los extremos del formato (del 13 al 10 bit y*
-- *                  del 3 al 0). Esta entidad utiliza el PACKAGE "sumacla".     *
-- *                                                                              *
-- * MODIFICACIONES:                                                              *
-- *                                                                              *
-- ********************************************************************************


LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;
USE work.sumacla.ALL;

ENTITY sumador IS

        PORT (
                -- DATOS DE ENTRADA
                -- Datos de entrada
                A,B     : IN STD_LOGIC_VECTOR(13 DOWNTO 0);

                -- Bit de acarreo de entrada al sumador
                CIN     : IN STD_LOGIC;
```

```
                -- DATOS DE SALIDA
                -- Dato resulatdo de la suma de los datos de entrada.
                SUM     : OUT STD_LOGIC_VECTOR(13 DOWNTO 0);

                -- Bit que indica un evento de overflow.
                OV      : OUT STD_LOGIC );

END sumador;

ARCHITECTURE suma16 OF sumador IS

        BEGIN
        PROCESS(A,B,CIN)

-- DECLARACIÓN Y DESCRIPCIÓN DE VARIABLES.
-- La variable "suma", almacena parcialmente el resultado de los tres módulos de suma,
-- la variable "acarreo" interconecta los bits de acarreo entre los módulos,
-- las variables "G" y "P" son vectores de 3 bits para almacenar los bits generados
-- y propagados que arrojan los módulos para posteriormente generar los bits del vector
-- "acarreo" que alimentan los módulos con los acarreos, y las variables restantes son
-- utilizadas para almacenar parcialmente los datos de 4 bits y de 6 bits de entrada
-- para que su correspondiente índice (ej. dunoa(0)) no presente problema al utilizar el
-- PACKAGE correspondiente que realiza el cómputo de la suma.

        VARIABLE suma         : STD_LOGIC_VECTOR(13 DOWNTO 0);
        VARIABLE acarreo      : STD_LOGIC_VECTOR(2 DOWNTO 0);
        VARIABLE G,P          : STD_LOGIC_VECTOR(2 DOWNTO 0);
        VARIABLE dunoa,dunoc  : STD_LOGIC_VECTOR(3 DOWNTO 0);
        VARIABLE ddosa,ddosc  : STD_LOGIC_VECTOR(3 DOWNTO 0);
        VARIABLE dunob,ddosb  : STD_LOGIC_VECTOR(5 DOWNTO 0);

                BEGIN

                dunoa          :=A(3 DOWNTO 0);
                dunob          :=A(9 DOWNTO 4);
                dunoc          :=A(13 DOWNTO 10);
                ddosa          :=B(3 DOWNTO 0);
                ddosb          :=B(9 DOWNTO 4);
                ddosc          :=B(13 DOWNTO 10);

                modulo(CIN,dunoa,ddosa,G(0),P(0),suma(3 DOWNTO 0));
                modulo(acarreo(0),dunob,ddosb,G(1),P(1),suma(9 DOWNTO 4));
                modulo(acarreo(1),dunoc,ddosc,G(2),P(2),suma(13 DOWNTO 10));

                acarreo(0)     :=G(0) OR (CIN AND P(0));
                acarreo(1)     :=G(1) OR (G(0) AND P(1)) OR (CIN AND P(0) AND P(1));
                --acarreo(2)   :=G(2) OR (G(1) AND P(2)) OR (G(0) AND P(1) AND P(2)) OR (CIN
AND P(0) AND P(1) AND P(2));

                SUM            <= suma;
                OV             <=  (A(13) AND B(13) AND NOT(suma(13))) OR (NOT(A(13)) AND
NOT(B(13)) AND suma(13));
        END PROCESS;

END suma16;
```

SUMACLA.VHD

```
-- ********************************************************************************
-- * TÍTULO:          ALU LOGARÍTMICA                                             *
-- * T.G.:            0432                                                        *
-- *                                                                              *
-- * INTEGRANTES:     MANUEL ALBERTO CARRERA OSORIO                               *
-- *                  EFRÉN OCTAVIO LÓPEZ URIBE                                   *
-- *                  JUAN DIEGO NARANJO LÓPEZ                                    *
-- *                                                                              *
-- * DIRECTORES:      ING. ALEJANDRA MARIA GONZÁLEZ                               *
-- *                  ING. FRANCISCO VIVEROS                                      *
```

```
-- *                                                                            *
-- * PAQUETE:          sumacla.vhd                                              *
-- *                                                                            *
-- * DESCRIPCIÓN:      Función que realiza la suma de cualesquier vectores de   *
-- *                   bits de entrada sin importar el tamaño de los vectores.  *
-- *                   La suma es realizada implementando unidades de tipo      *
-- *                   carry-look-ahead, y retorna la suma mas dos señales      *
-- *                   Gx y Px que indican la generación de un bit de acarreo   *
-- *                   o la propagación de uno respectivamente.                 *
-- *                                                                            *
-- * MODIFICACIONES:                                                            *
-- *                                                                            *
-- ******************************************************************************



LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;

PACKAGE sumacla IS

        -- Declaración del procedimiento que llama el PACKAGE.
        PROCEDURE modulo (

                -- SEÑALES DE ENTRADA
                -- Bit de acarreo de entrada.
                CIN             : IN STD_LOGIC;
                -- Datos de entrada a operar (cualquier longitud).
                A, B            : IN STD_LOGIC_VECTOR;

                -- SEÑALES DE SALIDA
                -- Señales de tipo BIT, Gx corresponde al bit de acerreo
                -- generado, y Px corresponde al bit de acarreo que será
                -- propagado.
                Gx,Px           : OUT STD_LOGIC;

                - Resultado de la suma de los datos de entrada.
                C               : OUT STD_LOGIC_VECTOR
                );

END sumacla;

PACKAGE BODY sumacla IS

        -- Declaración del comportamiento del procedimiento.
        PROCEDURE modulo (

                CIN     : IN STD_LOGIC;
                A, B    : IN STD_LOGIC_VECTOR;
                Gx,Px   : OUT STD_LOGIC;
                C       : OUT STD_LOGIC_VECTOR  ) IS

                -- Declaración de variables para la utilizarlas en el
                -- comportamiento del algoritmo.
                VARIABLE SUM, Gi, Pi, Ci      : STD_LOGIC_VECTOR(A'HIGH DOWNTO A'LOW);
                VARIABLE Go, Po               : STD_LOGIC;

                BEGIN

                Go:='0';
                Po:='1';
                Ci(Ci'LOW)     :=CIN;

                FOR K IN 1 TO A'HIGH LOOP
                        Gi(K-1)         :=A(K-1) AND B(K-1);
                        Pi(K-1)         :=A(K-1) OR B(K-1);
                        Ci(K)           :=Gi(K-1) OR (Ci(K-1) AND Pi(K-1));
                        SUM(K-1)        :=A(K-1) XOR B(K-1) XOR Ci(K-1);
                        Go              :=Gi(K-1) OR (Go AND Pi(K-1));
                        Po              :=Po AND Pi(K-1);
                END LOOP;
```

```
                SUM(SUM'HIGH)  :=A(A'HIGH) XOR B(B'HIGH) XOR Ci(Ci'HIGH);
                Gi(Gi'HIGH)    :=A(A'HIGH) AND B(B'HIGH);
                Pi(Pi'HIGH)    :=A(A'HIGH) OR B(B'HIGH);
                Go             :=Gi(Gi'HIGH) OR (Go AND Pi(Pi'HIGH));
                Po             :=Po AND Pi(Pi'HIGH);
                C              :=SUM;
                Gx             :=Go;
                Px             :=Po;

        END modulo;

END sumacla;
```

SIGN.VHD

```
-- ********************************************************************************
-- * TÍTULO:          ALU LOGARÍTMICA                                             *
-- * T.G.:            0432                                                        *
-- *                                                                              *
-- * INTEGRANTES:     MANUEL ALBERTO CARRERA OSORIO                               *
-- *                  EFRÉN OCTAVIO LÓPEZ URIBE                                   *
-- *                  JUAN DIEGO NARANJO LÓPEZ                                    *
-- *                                                                              *
-- * DIRECTORES:      ING. ALEJANDRA MARIA GONZÁLEZ                               *
-- *                  ING. FRANCISCO VIVEROS                                      *
-- *                                                                              *
-- * ENTIDAD:         sign.vhd                                                    *
-- *                                                                              *
-- * DESCRIPCIÓN:     Entidad que cumple la función de manejar el signo del dato  *
-- *                  de salida (bits 15 y 14).                                   *
-- *                                                                              *
-- * MODIFICACIONES:                                                              *
-- *                                                                              *
-- ********************************************************************************


LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;

ENTITY sign IS
        PORT(

                -- DATOS DE ENTRADA
                -- Bits 15 y 14 del vector de entrada A.
                ZSA     : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                -- Bits 15 y 14 del vector de entrada B.
                ZSB     : IN STD_LOGIC_VECTOR(1 DOWNTO 0);

-- Señal de control que indica el manejo al signo de salida.
-- ******************************************************
-- *   CSIGN   *       MANEJO                           *
-- ******************************************************
-- *    000    * La salida es el signo del dato A       *
-- *    001    * La salida es el signo del dato B       *
-- *    010    * La salida es A15 y NOT A14             *
-- *    011    * La salida es NOT A15 y A14             *
-- *    100    * La salida es B15 y NOT B14             *
-- *    101    * La salida es NOT A15 y NOT A14         *
-- *    11X    * La salida es A15 y (A14 XOR B14)       *
-- ******************************************************

                CSIGN   : IN STD_LOGIC_VECTOR(2 DOWNTO 0);

                -- DATOS DE SALIDA
                -- Bits 15 y 14 del vector de salida C.
                ZSC     : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
                );
```

```
END sign;

ARCHITECTURE alusign OF sign IS
       BEGIN
       PROCESS(ZSA,ZSB,CSIGN)
              BEGIN
              CASE CSIGN(2 DOWNTO 0) IS
                     -- Caso salida ->A15 y (A14 xor B14), usado en la multiplicación y
división.
                     WHEN "111"  =>
                            ZSC(0) <= ZSA(0) XOR ZSB(0);
                            ZSC(1) <= ZSA(1);
                     -- Caso salida ->A15 y (A14 xor B14), usado en la multiplicación y
división.
                     WHEN "110"  =>
                            ZSC(0) <= ZSA(0) XOR ZSB(0);
                            ZSC(1) <= ZSA(1);
                     -- Caso salida ->B15 y !B14
                     WHEN "100" =>
                            ZSC(0) <= NOT ZSB(0);
                            ZSC(1) <= ZSB(1);
                     -- Caso salida ->A15 y !A14
                     WHEN "010" =>
                            ZSC(0) <= NOT ZSA(0);
                            ZSC(1) <= ZSA(1);
                     -- Caso salida ->A15 y A14
                     WHEN "000" =>
                            ZSC(1 DOWNTO 0) <= ZSA(1 DOWNTO 0);
                     -- Caso salida ->B15 y B14
                     WHEN "001" =>
                            ZSC(1 DOWNTO 0) <= ZSB(1 DOWNTO 0);
                     -- Caso salida ->!A15 y A14
                     WHEN "011" =>
                            ZSC(0) <= ZSA(0);
                            ZSC(1) <= NOT ZSA(1);
                     -- Caso salida ->!A15 y !A14
                     WHEN "101" =>
                            ZSC(0) <= NOT ZSA(0);
                            ZSC(1) <= NOT ZSA(1);
              END CASE;
       END PROCESS;
END alusign;
```

SHIFT.VHD

```
-- ********************************************************************************
-- * TÍTULO:          ALU LOGARÍTMICA                                             *
-- * T.G.:            0432                                                        *
-- *                                                                              *
-- * INTEGRANTES:     MANUEL ALBERTO CARRERA OSORIO                               *
-- *                  EFRÉN OCTAVIO LÓPEZ URIBE                                   *
-- *                  JUAN DIEGO NARANJO LÓPEZ                                    *
-- *                                                                              *
-- * DIRECTORES:      ING. ALEJANDRA MARIA GONZÄLEZ                               *
-- *                  ING. FRANCISCO VIVEROS                                      *
-- *                                                                              *
-- * ENTIDAD:         shift.vhd                                                   *
-- *                                                                              *
-- * DESCRIPCIÓN:     Entidad que cumple varias tareas, la principal de ellas es  *
-- *                  la de realizar las operaciones de raíz cuadrada y potencias*
-- *                  al cuadrado; adicionalmente permite manejar las respuestas  *
-- *                  a operaciones cuyo resultado sean : + INFINITO, - INFINITO  *
-- *                  y NaN, las cuales dependen únicamente de los datos de       *
-- *                  entrada y de las operaciones indicadas por el usuario.      *
-- *                                                                              *
-- * MODIFICACIONES:                                                              *
-- *                                                                              *
-- ********************************************************************************
```

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;

ENTITY shift IS
	PORT(

		-- DATOS DE ENTRADA.
		-- Vector de datos.
		A	: IN STD_LOGIC_VECTOR(13 DOWNTO 0);

-- Vector de control.
-- ********************************************************************************
-- * CTSUIN  * OPERACIÓN                                                          *
-- ********************************************************************************
-- *   000   * La salida es el dato de entrada                                    *
-- *   001   * La salida es la negación de la entrada                             *
-- *   010   * La salida corresponde a INFINITO                                   *
-- *   011   * La salida es el bit 13 negado y los demás bits de entrada          *
-- *   101   * La salida corresponde al corrimiento a izquierda con LSB(out) = MSB(in) *
-- *   110   * La salida corresponde al corrimiento a derecha con MSB(out) = MSB-1(in) *
-- ********************************************************************************
		CTSUIN  : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
		-- DATOS DE SALIDA.
		AS	: OUT STD_LOGIC_VECTOR(13 DOWNTO 0)
		);
END shift;

ARCHITECTURE alushift OF shift IS
	BEGIN
	PROCESS(A,CTSUIN)
	BEGIN
	CASE CTSUIN IS
			-- Caso corrimiento a derecha con inclusión del MSB del dato
			WHEN "110" =>
				AS(13) <= A(13);
				AS(12 DOWNTO 0) <= A(13 DOWNTO 1);
			-- Caso corrimiento a izquierda.
			WHEN "101" =>
				AS(0) <= '0';
				AS(13 DOWNTO 1) <= A(12 DOWNTO 0);
			-- Caso !A13 y ->A12 .. A0 (División)
			WHEN "011" =>
				AS(13) <= NOT A(13);
				AS(12 DOWNTO 0) <= A(12 DOWNTO 0);
			-- Caso GINF (Generación de infinito a partir de la entrada)
			WHEN "010" =>
				AS(13) <= A(13);
				AS(12 DOWNTO 0) <= NOT A(12 DOWNTO 0);
			-- Caso NEG (negar la entrada)
			WHEN "001" =>
				AS(13 DOWNTO 0) <= NOT A(13 DOWNTO 0);
			-- Caso ->A
			WHEN OTHERS =>
				AS(13 DOWNTO 0) <= A(13 DOWNTO 0);
		END CASE;
	END PROCESS;
END alushift;
```

NEG.VHD

```
-- ********************************************************************************
-- * TÍTULO:          ALU LOGARÍTMICA                                             *
-- * T.G.:            0432                                                        *
-- *                                                                              *
-- * INTEGRANTES:     MANUEL ALBERTO CARRERA OSORIO                               *
-- *                  EFRÉN OCTAVIO LÓPEZ URIBE                                   *
-- *                  JUAN DIEGO NARANJO LÓPEZ                                    *
-- *                                                                              *
-- * DIRECTORES:      ING. ALEJANDRA MARIA GONZÄLEZ                               *
-- *                  ING. FRANCISCO VIVEROS                                      *
```

```
-- *                                                                          *
-- * ENTIDAD:          neg.vhd                                                *
-- *                                                                          *
-- * DESCRIPCIÓN:      Entidad que cumple la tarea de invertir su entrada para    *
-- *                   desempeñar (junto con la entidad "sumador") la tarea de la *
-- *                   substracción que es utilizada en las operaciones de suma,  *
-- *                   esta y división.                                       *
-- *                                                                          *
-- * MODIFICACIONES:                                                          *
-- *                                                                          *
-- ****************************************************************************

LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;

ENTITY neg IS
        PORT(
                -- DATOS DE ENTRADA
                EN      : IN STD_LOGIC_VECTOR(13 downto 0);
                -- DATOS DE SALIDA
                SAL     : OUT STD_LOGIC_VECTOR(13 downto 0)
        );
END neg;

ARCHITECTURE aluneg OF neg IS
        BEGIN
        PROCESS(EN)
                BEGIN
                SAL(13 downto 0) <= NOT EN(13 downto 0);
        END PROCESS;
END aluneg;
```

MULTIPLEXOR.VHD

```
-- ****************************************************************************
-- * TÍTULO:           ALU LOGARÍTMICA                                        *
-- * T.G.:             0432                                                   *
-- *                                                                          *
-- * INTEGRANTES:      MANUEL ALBERTO CARRERA OSORIO                          *
-- *                   EFRÉN OCTAVIO LÓPEZ URIBE                              *
-- *                   JUAN DIEGO NARANJO LÓPEZ                               *
-- *                                                                          *
-- * DIRECTORES:       ING. ALEJANDRA MARIA GONZÁLEZ                          *
-- *                   ING. FRANCISCO VIVEROS                                 *
-- *                                                                          *
-- * ENTIDAD:          multiplexor.vhd                                        *
-- *                                                                          *
-- * DESCRIPCIÓN:      Entidad que cumple la tarea de seleccionar los datos de    *
-- *                   salida A o B, de acuerdo a la señal de control.            *
-- *                                                                          *
-- * MODIFICACIONES:                                                          *
-- *                                                                          *
-- ****************************************************************************

LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;

ENTITY multiplexor IS
        PORT(

                -- DATOS DE ENTRADA

                -- Señales para seleccionar
                A,B     : IN STD_LOGIC_VECTOR(13 downto 0);

-- Señal de control de selección
--*****************************************
--*       SEL  *       DATO DE SALIDA        *
--*****************************************
--*        0   *            A               *
```

```
--*        1   *            B               *
--*****************************************
               SEL     : IN STD_LOGIC;

               -- DATO DE SALIDA
               C       : OUT STD_LOGIC_VECTOR(13 downto 0)
       );
END multiplexor;

ARCHITECTURE mux OF multiplexor IS
       BEGIN
       WITH SEL SELECT
              C <= A   WHEN '0',
                   B   WHEN '1';
END mux;
```

BANDERAS.VHD

```
-- ********************************************************************************
-- * TÍTULO:          ALU LOGARÍTMICA                                             *
-- * T.G.:            0432                                                        *
-- *                                                                              *
-- * INTEGRANTES:     MANUEL ALBERTO CARRERA OSORIO                               *
-- *                  EFRÉN OCTAVIO LÓPEZ URIBE                                   *
-- *                  JUAN DIEGO NARANJO LÓPEZ                                    *
-- *                                                                              *
-- * DIRECTORES:      ING. ALEJANDRA MARIA GONZÁLEZ                               *
-- *                  ING. FRANCISCO VIVEROS                                      *
-- *                                                                              *
-- * ENTIDAD:         banderas.vhd                                                *
-- *                                                                              *
-- * DESCRIPCIÓN:     Entidad que realiza el cálculo de las banderas de acuerdo   *
-- *                  a los datos de entrada al sistema, los eventos especiales   *
-- *                  de las unidades funcionales internas y con respecto a la    *
-- *                  operación indicada por el usuario.                          *
-- *                                                                              *
-- * MODIFICACIONES:                                                              *
-- *                                                                              *
-- ********************************************************************************


LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;

ENTITY banderas IS
       PORT(

-- SEÑALES DE ENTRADA
-- Codigo de operación que será realizada por la ALU
-- ***********************************
-- *  OPCODE   *    OPERACIÓN          *
-- ***********************************
-- *    000    *    SUMA               *
-- *    001    *    RESTA              *
-- *    010    *    MULTIPLICACIÓN     *
-- *    011    *    DIVISIÓN           *
-- *    100    *    CORR_DER_RED       *
-- *    101    *    CORR_IZQ_RED       *
-- *    110    *    CORR_DER_SIN_RED   *
-- ***********************************
               OPCODE         : IN STD_LOGIC_VECTOR(2 DOWNTO 0);

               -- Bits mas significativos de los datos de entrada del dato A, A(1) bit de
cero y A(0) es el bit de signo
               DBITSIA        : IN STD_LOGIC_VECTOR(3 DOWNTO 0);

               -- Bits mas significativos de los datos de entrada del dato B, B(1) bit de
cero y B(0) es el bit de signo
               DBITSIB        : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
```

```
                -- Acarreo de la unidad sumadora SUM1
                COUTSUM1        : IN STD_LOGIC;

                -- Acarreo de la unidad sumadora SUM2
                COUTSUM2        : IN STD_LOGIC;

                -- Entrada de MSB, LSB y de bit 12 de dato A
                LSBA            : IN STD_LOGIC;

-- VECTOR DE SALIDA
-- Señal de salida que indica el evento especial ocurrido
-- ******************************************************
-- *   FLAGS   *              EVENTO                        *
-- ******************************************************
-- *     0     * Overvlow de alguno de los sumadores       *
-- *     1     * Resultado es + INFINITO                   *
-- *     2     * Resultado es - INFINITO                   *
-- *     3     * Resultado es NON                          *
-- *     4     * Cambio de signo en corrimiento            *
-- *     5     * Bandera de división por cero              *
-- *     6     * Bandera de número imaginario              *
-- ******************************************************

                FLAGS   : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
                );
END banderas;

ARCHITECTURE alubanderas OF banderas IS

        BEGIN
        PROCESS(OPCODE,DBITSIA,DBITSIB,COUTSUM1,COUTSUM2,LSBA)

-- DECLARACIÓN Y DESCRIPCIÓN DE VARIABLES
-- Las variables "MASINFA" y "MASINFB" indican si alguno de los datos de entrada es un
-- + INFINITO; "MENOSINFA" y "MENOSINFB" indican si alguno de los datos de entrada es un
-- - INFINITO; "ZEROA" y "ZEROB" indican si alguno de los datos de entrada es un CERO;
-- "NONA" y "NONB" indican si alguno de los datos de entrada es un "not-a-number";
-- "POSA" y "POSB" indican si los datos de entrada son positivos; y "NEGA" y "NEGB" indican
-- si los datos de entrada son negativos.
-- NOTA : Todas las señales indican el correspondiente evento descrito anteriormente
-- con un uno (1) lógico.

        VARIABLE MASINFA, MASINFB     : STD_LOGIC;
        VARIABLE MENOSINFA, MENOSINFB : STD_LOGIC;
        VARIABLE ZEROA, ZEROB         : STD_LOGIC;
        VARIABLE NONA, NONB           : STD_LOGIC;
        VARIABLE POSA, POSB           : STD_LOGIC;
        VARIABLE NEGA, NEGB           : STD_LOGIC;

                BEGIN
                MASINFA         := DBITSIA(3) AND NOT(DBITSIA(2) OR DBITSIA(1)) AND
DBITSIA(0);
                MASINFB         := DBITSIB(3) AND NOT(DBITSIB(2) OR DBITSIB(1)) AND
DBITSIB(0);
                MENOSINFA       := DBITSIA(3) AND DBITSIA(2) AND (NOT DBITSIA(1)) AND
DBITSIA(0);
                MENOSINFB       := DBITSIB(3) AND DBITSIB(2) AND (NOT DBITSIB(1)) AND
DBITSIB(0);
                ZEROA           := DBITSIA(3) AND NOT (DBITSIA(2) OR DBITSIA(1) OR
DBITSIA(0));
                ZEROB           := DBITSIB(3) AND NOT (DBITSIB(2) OR DBITSIB(1) OR
DBITSIB(0));
                NONA            := DBITSIA(3) AND DBITSIA(1) AND (NOT DBITSIA(0));
                NONB            := DBITSIB(3) AND DBITSIB(1) AND (NOT DBITSIB(0));
                POSA            := NOT(DBITSIA(3) OR DBITSIA(2));
                POSB            := NOT(DBITSIB(3) OR DBITSIB(2));
                NEGA            := NOT(DBITSIA(3)) AND DBITSIA(2);
                NEGB            := NOT(DBITSIB(3)) AND DBITSIB(2);


                CASE OPCODE IS
```

```
                        -- Operación de suma
                        WHEN "000" =>
                                FLAGS(0) <= (COUTSUM1 OR COUTSUM2) AND NOT(NONA OR NONB OR
MASINFA OR MASINFB OR MENOSINFA OR MENOSINFB);
                                FLAGS(1) <= ((MASINFA OR MASINFB) AND (POSA OR POSB OR NEGA OR
NEGB OR ZEROA OR ZEROB)) OR (MASINFA AND MASINFB);
                                FLAGS(2) <= ((MENOSINFA OR MENOSINFB) AND (POSA OR POSB OR
NEGA OR NEGB OR ZEROA OR ZEROB)) OR (MENOSINFA AND MENOSINFB);
                                FLAGS(3) <= NONA OR NONB OR (MASINFA AND MENOSINFB) OR
(MENOSINFA AND MASINFB);
                                FLAGS(4) <= '0';
                                FLAGS(5) <= '0';
                                FLAGS(6) <= '0';

                        -- Operación de resta
                        WHEN "001" =>
                                FLAGS(0) <= (COUTSUM1 OR COUTSUM2) AND NOT(NONA OR NONB OR
MASINFA OR MASINFB OR MENOSINFA OR MENOSINFB);
                                FLAGS(1) <= ((MASINFA XOR MENOSINFB) AND (POSA OR POSB OR NEGA
OR NEGB OR ZEROA OR ZEROB)) OR (MASINFA AND MENOSINFB);
                                FLAGS(2) <= ((MENOSINFA XOR MASINFB) AND (POSA OR POSB OR NEGA
OR NEGB OR ZEROA OR ZEROB)) OR (MENOSINFA AND MASINFB);
                                FLAGS(3) <= NONA OR NONB OR (MASINFA AND MASINFB) OR
(MENOSINFA AND MENOSINFB);
                                FLAGS(4) <= '0';
                                FLAGS(5) <= '0';
                                FLAGS(6) <= '0';

                        -- Operación de multiplicación
                        WHEN "010" =>
                                FLAGS(0) <= COUTSUM2;

                                FLAGS(1) <= (MASINFA AND (POSB OR MASINFB)) OR (MASINFB AND
(POSA OR MASINFA)) OR (MENOSINFA AND (MENOSINFB OR NEGB)) OR (MENOSINFB AND NEGA);
                                FLAGS(2) <= (MENOSINFA AND POSB) OR (MASINFA AND NEGB) OR
(POSA AND MENOSINFB) OR (MASINFB AND NEGA);
                                FLAGS(3) <=  NONA OR NONB OR (MENOSINFA AND MASINFB) OR
(MASINFA AND MENOSINFB);
                                FLAGS(4) <= '0';
                                FLAGS(5) <= '0';
                                FLAGS(6) <= '0';

                        -- Operación de división
                        WHEN "011" =>
                                FLAGS(0) <= COUTSUM1;
                                FLAGS(1) <= ((POSA OR MASINFA) AND ZEROB) OR (MENOSINFA AND
NEGB) OR (MASINFA AND POSB);
                                FLAGS(2) <= ((NEGA OR MENOSINFA) AND ZEROB) OR (MASINFA AND
NEGB) OR (MENOSINFA AND POSB);
                                FLAGS(3) <= (ZEROA AND (MASINFB OR MENOSINFB)) OR ((POSA OR
NEGA) AND (MASINFB OR MENOSINFB)) OR (ZEROA AND ZEROB) OR NONB OR NONA OR ((MASINFA OR
MENOSINFA) AND (MASINFB OR MENOSINFB));
                                FLAGS(4) <= '0';
                                FLAGS(5) <= ZEROB;
                                FLAGS(6) <= '0';

                        -- Operación de corrimiento a izquierda con redondeo de MSB
                        WHEN "101" =>
                                FLAGS(0) <= '0';
                                FLAGS(1) <= MASINFA;
                                FLAGS(2) <= MENOSINFA;
                                FLAGS(3) <= NONA;
                                FLAGS(4) <= (DBITSIA(1) XOR DBITSIA(0)) AND (NOT(MASINFA OR
MENOSINFA OR NONA));
                                FLAGS(5) <= '0';
                                FLAGS(6) <= '0';

                        -- Operación de corrimiento a derecha con inclusión de MSB
                        WHEN "110" =>
                                FLAGS(0) <= '0';
```

```
                                FLAGS(1) <= MASINFA;
                                FLAGS(2) <= MENOSINFA;
                                FLAGS(3) <= NONA;
                                FLAGS(5 DOWNTO 4) <= "00";
                                FLAGS(6) <= DBITSIA(2);

                            WHEN OTHERS =>
                                FLAGS(6 DOWNTO 0) <= "1111111";
                    END CASE;
            END PROCESS;

    END alubanderas;
```

ANEXO B CÓDIGO FUENTE DE INTERFAZ DIDÁCTICA

CONTROLADOR.JAVA

```
//  ------------------------------------------------------------------------------
//  | TÍTULO:         ALU LOGARÍTMICA                                            |
//  | T.G.:           0432                                                       |
//  |                                                                            |
//  | INTEGRANTES:    MANUEL ALBERTO CARRERA OSORIO                              |
//  |                 EFRÉN OCTAVIO LÓPEZ URIBE                                  |
//  |                 JUAN DIEGO NARANJO LÓPEZ                                   |
//  |                                                                            |
//  | DIRECTORES:     ING. ALEJANDRA MARIA GONZÁLEZ                              |
//  |                 ING. FRANCISCO VIVEROS                                     |
//  |                                                                            |
//  | CODIGO:         controlador.java                                           |
//  |                                                                            |
//  | DESCRIPCIÓN:    Este código esta escrito en lenguaje de programación JAVA, |
//  |                 es la implementación de una interfaz grafica que permite,  |
//  |                 de manera didáctica, a un usuario interactuar con las      |
//  |                 operaciones que realiza la ALU Logarítmica.                |
//  |                                                                            |
//  | MODIFICACIONES:                                                            |
//  |                                                                            |
//  ------------------------------------------------------------------------------
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;
import javax.comm.*;
import java.io.*;


/**
 * Esta clase puede tomar un número variable de parámetros
 * en la línea de comandos. La ejecución del programa comienza
 * con el método main(). La llamada al constructor de clase no tiene lugar a menos que se
cree un objeto del tipo 'Form1'
 * en el método main().
 */
public class Form1 extends Form implements SerialPortEventListener{


        //---variables de puerto---
        CommPortIdentifier portIDE;
        SerialPort Puerto;
        OutputStream outputbuffer;
        InputStream inputbuffer;
        boolean bandera=false;

        //---variables importantes;
        double error1=0;
        int opa3=1536;
        int opb3=1536;
        int opc3=0;
        int opd3=0;
        double opa1=8, opb1=8, opc1=0, opd1=0;
        double opa2=3, opb2=3, opc2=0, opd2=0;
        int Banderas=0;

        public Form1()
        {
                // Requerido para la compatibilidad con el Diseñador de formularios de Visual
J++
```

```
        initForm();

        // TODO: agregar el código del constructor después de la llamada a initForm.
        //---buscar el puerto COM1---

        try{
                portIDE = CommPortIdentifier.getPortIdentifier("COM1");
                label19.setText("conectado a: "+portIDE.getName());
        }catch(NoSuchPortException e){
                label19.setText("err1: no existe puerto");
        }

        //---apertura del puerto COM1---

        try {
                Puerto = (SerialPort) portIDE.open("controlador", 2000);
        }catch (PortInUseException e) {
                label19.setText("err2: puerto en uso");
        }


        //---fijando parametros de trasmision---

        try {
                Puerto.setSerialPortParams(9600,
                                SerialPort.DATABITS_8,
                                SerialPort.STOPBITS_2,
                                SerialPort.PARITY_NONE);
        } catch (UnsupportedCommOperationException e) {
                label19.setText("err3: params not set");
        }


        //---linking objeto OutputStream & InputStream---

        try {
                outputbuffer = Puerto.getOutputStream();
        } catch (IOException e) {}
        try {
                inputbuffer = Puerto.getInputStream();
        } catch (IOException e) {}

        //---agregar listener---

        try {
                Puerto.addEventListener(this);
                Puerto.notifyOnDataAvailable(true);
        } catch (Exception e) {
                label19.setText("err4: no se pudo agregar listener");
        }

}

/**
 * Form1 reemplaza el método "dispose" para poder
 * limpiar la lista de componentes.
 */
public void dispose()
{
        super.dispose();
        components.dispose();
}


private void edit1_textChanged(Object source, Event e)
{
        try{
                opa1=(double)Integer.parseInt(edit1.getText());
                msgerror.setText(" ");
                if(opa1 > 32767){
                        opa1=32767;
```

```
                        edit1.setText(Double.toString(opa1));
                        }
                if(opa1 < -32768){
                        opa1=-32768;
                        edit1.setText(Double.toString(opa1));
                        }
                if(opa1==0){
                        opa2=0;
                        edit3.setText("-inf");
                        opa3=32768;
                        edit5.setText(conversion(opa3));
                }else{
                        opa2=Math.log((double) Math.abs(opa1))/Math.log(2);
                        opa3=Math.round((float)(opa2*Math.pow(2,9)));
                        if(opa1<0)
                                opa3=opa3+16384;
                                edit3.setText(Double.toString(opa2));
                                edit5.setText(conversion(opa3));
                        }
        }catch(NumberFormatException er){
                msgerror.setText("err5: Datos 1 no es un numero valido");
        }
}

private void edit2_textChanged(Object source, Event e)
{
        try{
                opb1=(double)Integer.parseInt(edit2.getText());
                msgerror.setText(" ");
                if(opb1 > 32767){
                        opb1=32767;
                        edit2.setText(Double.toString(opb1));
                        }
                if(opb1 < -32768){
                        opb1=-32768;
                        edit2.setText(Double.toString(opb1));
                        }
                if(opb1==0){
                        opb2=0;
                        edit4.setText("-inf");
                        opb3=32768;
                        edit6.setText(conversion(opb3));
                }else{
                        opb2=Math.log((double) Math.abs(opb1))/Math.log(2);
                        opb3=Math.round((float)(opb2*Math.pow(2,9)));
                        if(opb1<0)
                                opb3=opb3+16384;
                                edit4.setText(Double.toString(opb2));
                                edit6.setText(conversion(opb3));
                        }
        }catch(NumberFormatException er){
                msgerror.setText("err5: Dato 2 no es un numero valido");
        }
}

private void radioButton1_checkedChanged(Object source, Event e)
{
                if(radioButton1.getChecked()){
                        opd1=opa1*opb1;
                        res_cpu.setText(Double.toString(opd1));
                        msgerror.setText(" ");
                        sendOperacion(opa3,opb3,(byte)2);
                        codigo.setText("010");
                }
}

private void radioButton2_checkedChanged(Object source, Event e)
{
                if(radioButton2.getChecked()){
                        if(opb1==0){
                                opd1=0;
```

```
                        msgerror.setText("division por cero");

                }else{
                        opd1=opa1/opb1;
                        msgerror.setText(" ");
                }
                res_cpu.setText(Double.toString(opd1));
                sendOperacion(opa3,opb3,(byte)3);
                codigo.setText("011");
        }
}

private void radioButton3_checkedChanged(Object source, Event e)
{
        if(radioButton3.getChecked()){
                opd1=opa1+opb1;
                res_cpu.setText(Double.toString(opd1));
                msgerror.setText(" ");
                sendOperacion(opa3,opb3,(byte)0);
                codigo.setText("000");
        }
}

private void radioButton4_checkedChanged(Object source, Event e)
{
        if(radioButton4.getChecked()){
                opd1=opa1-opb1;
                res_cpu.setText(Double.toString(opd1));
                msgerror.setText(" ");
                sendOperacion(opa3,opb3,(byte)1);
                codigo.setText("001");
        }
}

private void radioButton5_checkedChanged(Object source, Event e)
{
        if(radioButton5.getChecked()){
                opd1=Math.pow(opa1,2);
                res_cpu.setText(Double.toString(opd1));
                msgerror.setText(" ");
                sendOperacion(opa3,opb3,(byte)5);
                codigo.setText("101");
        }
}

private void radioButton6_checkedChanged(Object source, Event e)
{
        if(radioButton6.getChecked()){
                opd1=Math.sqrt(Math.abs(opa1));
                if(opa1<0){
                        res_cpu.setText("im "+Double.toString(opd1));
                        msgerror.setText("resultado imaginario");
                }else{
                        res_cpu.setText(Double.toString(opd1));
                        msgerror.setText(" ");
                }
                sendOperacion(opa3,opb3,(byte)6);
                codigo.setText("110");
        }
}

private void edit5_textChanged(Object source, Event e)
{
        int var1=conversion2(edit5.getText());
        if(var1==65536){
                msgerror.setText("cadena invalida RA");
        }else{
                opa3=var1;
                double ops[]=tratamiento(opa3);
                opa1=ops[0];
                opa2=ops[1];
```

```
                        edit1.setText(Double.toString(opa1));
                        edit3.setText(Double.toString(opa2));
                        edit5.setText(conversion(opa3));
                        msgerror.setText(" ");
                }
}

private void edit6_textChanged(Object source, Event e)
{
                int var1=conversion2(edit6.getText());
                if(var1==65536){
                        msgerror.setText("cadena invalida RB");
                }else{
                        opb3=var1;
                        double ops[]=tratamiento(opb3);
                        opb1=ops[0];
                        opb2=ops[1];
                        edit2.setText(Double.toString(opb1));
                        edit4.setText(Double.toString(opb2));
                        edit6.setText(conversion(opb3));
                        msgerror.setText(" ");
                }
}


private void Acerca_click(Object source, Event e)
{

}

/**
 * NOTA: el código siguiente es necesario para el diseñador de
 * formularios de Visual J++ y puede modificarse con el editor de formularios.
 * No lo modifique desde el editor de código.
 */
Container components = new Container();
Label label1 = new Label();
Label Label2 = new Label();
Edit edit4 = new Edit();
Edit edit2 = new Edit();
Label label5 = new Label();
Label label6 = new Label();
Edit edit5 = new Edit();
Edit edit6 = new Edit();
Label label3 = new Label();
Label label4 = new Label();
PictureBox pictureBox1 = new PictureBox();
Edit edit3 = new Edit();
Edit edit1 = new Edit();
RadioButton radioButton1 = new RadioButton();
RadioButton radioButton2 = new RadioButton();
RadioButton radioButton3 = new RadioButton();
RadioButton radioButton6 = new RadioButton();
RadioButton radioButton4 = new RadioButton();
RadioButton radioButton5 = new RadioButton();
Label label2 = new Label();
Edit codigo = new Edit();
Edit RC = new Edit();
Label label7 = new Label();
Label label8 = new Label();
Label label9 = new Label();
Edit res_alu = new Edit();
Edit res_cpu = new Edit();
Label label10 = new Label();
Edit edit11 = new Edit();
Label label11 = new Label();
Label label12 = new Label();
Label label13 = new Label();
Label label14 = new Label();
Label label15 = new Label();
Label label16 = new Label();
```

```
Label label17 = new Label();
Label label18 = new Label();
Edit band1 = new Edit();
Edit band2 = new Edit();
Edit band3 = new Edit();
Edit band4 = new Edit();
Edit band5 = new Edit();
Edit band6 = new Edit();
Edit band7 = new Edit();
Label label19 = new Label();
Label msgerror = new Label();
MainMenu mainMenu1 = new MainMenu();
MenuItem Acerca = new MenuItem();
MenuItem Ayuda = new MenuItem();
Label label20 = new Label();
Label label21 = new Label();

private void initForm()
{
        // NOTA: este formulario almacena información en un
        // archivo externo. No modifique el parámetro de cadena para ninguna
        // llamada a la función resources.getObject(). Por ejemplo, no
        // modifique "ubicacion_abc1" en la siguiente línea de código
        // aunque sí cambie el nombre del objeto Abc:
        //      abc1.setLocation((Point)resources.getObject("ubicacion_abc"));

        IResourceManager resources = new ResourceManager(this, "Form1");
        label1.setLocation(new Point(136, 8));
        label1.setSize(new Point(40, 16));
        label1.setTabIndex(8);
        label1.setTabStop(false);
        label1.setText("Dato 1");

        Label2.setLocation(new Point(136, 32));
        Label2.setSize(new Point(40, 16));
        Label2.setTabIndex(9);
        Label2.setTabStop(false);
        Label2.setText("Dato 2");

        edit4.setEnabled(false);
        edit4.setLocation(new Point(376, 32));
        edit4.setSize(new Point(96, 20));
        edit4.setTabIndex(26);
        edit4.setText("3");

        edit2.setLocation(new Point(184, 32));
        edit2.setSize(new Point(96, 20));
        edit2.setTabIndex(27);
        edit2.setText("8");
        edit2.addOnTextChanged(new EventHandler(this.edit2_textChanged));

        label5.setLocation(new Point(216, 64));
        label5.setSize(new Point(56, 16));
        label5.setTabIndex(4);
        label5.setTabStop(false);
        label5.setText("Dato 1 RA");

        label6.setLocation(new Point(392, 64));
        label6.setSize(new Point(64, 16));
        label6.setTabIndex(2);
        label6.setTabStop(false);
        label6.setText("Dato 2 RB");

        edit5.setLocation(new Point(184, 88));
        edit5.setSize(new Point(112, 20));
        edit5.setTabIndex(22);
        edit5.setText("00 00011 000000000");
        edit5.addOnTextChanged(new EventHandler(this.edit5_textChanged));

        edit6.setLocation(new Point(360, 88));
        edit6.setSize(new Point(112, 20));
```

```
                edit6.setTabIndex(20);
                edit6.setText("00 00011 000000000");
                edit6.addOnTextChanged(new EventHandler(this.edit6_textChanged));

                label3.setLocation(new Point(288, 8));
                label3.setSize(new Point(88, 16));
                label3.setTabIndex(3);
                label3.setTabStop(false);
                label3.setText("Logaritmo Dato 1");

                label4.setLocation(new Point(288, 32));
                label4.setSize(new Point(88, 16));
                label4.setTabIndex(0);
                label4.setTabStop(false);
                label4.setText("Logaritmo Dato 2");

                pictureBox1.setLocation(new Point(184, 112));
                pictureBox1.setSize(new Point(288, 176));
                pictureBox1.setTabIndex(28);
                pictureBox1.setTabStop(false);
                pictureBox1.setText("pictureBox1");
                pictureBox1.setImage((Bitmap)resources.getObject("pictureBox1_image"));
                pictureBox1.setSizeMode(PictureBoxSizeMode.CENTER_IMAGE);

                edit3.setEnabled(false);
                edit3.setLocation(new Point(376, 8));
                edit3.setSize(new Point(96, 20));
                edit3.setTabIndex(25);
                edit3.setText("3");

                edit1.setLocation(new Point(184, 8));
                edit1.setSize(new Point(96, 20));
                edit1.setTabIndex(23);
                edit1.setText("8");
                edit1.setAcceptsReturn(false);
                edit1.addOnTextChanged(new EventHandler(this.edit1_textChanged));

                radioButton1.setLocation(new Point(8, 144));
                radioButton1.setSize(new Point(100, 23));
                radioButton1.setTabIndex(29);
                radioButton1.setText("Multiplicación");
                radioButton1.addOnCheckedChanged(new
EventHandler(this.radioButton1_checkedChanged));

                radioButton2.setLocation(new Point(8, 160));
                radioButton2.setSize(new Point(100, 23));
                radioButton2.setTabIndex(30);
                radioButton2.setText("División");
                radioButton2.addOnCheckedChanged(new
EventHandler(this.radioButton2_checkedChanged));

                radioButton3.setLocation(new Point(8, 176));
                radioButton3.setSize(new Point(100, 23));
                radioButton3.setTabIndex(34);
                radioButton3.setText("Suma");
                radioButton3.addOnCheckedChanged(new
EventHandler(this.radioButton3_checkedChanged));

                radioButton6.setLocation(new Point(8, 224));
                radioButton6.setSize(new Point(100, 23));
                radioButton6.setTabIndex(31);
                radioButton6.setText("Raíz");
                radioButton6.addOnCheckedChanged(new
EventHandler(this.radioButton6_checkedChanged));

                radioButton4.setLocation(new Point(8, 192));
                radioButton4.setSize(new Point(100, 23));
                radioButton4.setTabIndex(33);
                radioButton4.setText("Resta");
                radioButton4.addOnCheckedChanged(new
EventHandler(this.radioButton4_checkedChanged));
```

```
                radioButton5.setLocation(new Point(8, 208));
                radioButton5.setSize(new Point(100, 23));
                radioButton5.setTabIndex(32);
                radioButton5.setText("Potencia");
                radioButton5.addOnCheckedChanged(new
EventHandler(this.radioButton5_checkedChanged));

                label2.setLocation(new Point(120, 152));
                label2.setSize(new Point(56, 32));
                label2.setTabIndex(35);
                label2.setTabStop(false);
                label2.setText("Codigo de Operación");

                codigo.setEnabled(false);
                codigo.setLocation(new Point(120, 192));
                codigo.setSize(new Point(56, 20));
                codigo.setTabIndex(17);
                codigo.setText("");

                RC.setEnabled(false);
                RC.setLocation(new Point(272, 288));
                RC.setSize(new Point(112, 20));
                RC.setTabIndex(18);
                RC.setText("");

                label7.setLocation(new Point(160, 288));
                label7.setSize(new Point(96, 16));
                label7.setTabIndex(1);
                label7.setTabStop(false);
                label7.setText("Dato Resultado RC");
                label7.setTextAlign(HorizontalAlignment.RIGHT);

                label8.setLocation(new Point(176, 328));
                label8.setSize(new Point(80, 16));
                label8.setTabIndex(6);
                label8.setTabStop(false);
                label8.setText("Resultado CPU");
                label8.setTextAlign(HorizontalAlignment.RIGHT);

                label9.setLocation(new Point(176, 352));
                label9.setSize(new Point(80, 16));
                label9.setTabIndex(7);
                label9.setTabStop(false);
                label9.setText("Resultado ALU");
                label9.setTextAlign(HorizontalAlignment.RIGHT);

                res_alu.setEnabled(false);
                res_alu.setLocation(new Point(272, 352));
                res_alu.setSize(new Point(112, 20));
                res_alu.setTabIndex(24);
                res_alu.setText("");

                res_cpu.setEnabled(false);
                res_cpu.setLocation(new Point(272, 328));
                res_cpu.setSize(new Point(112, 20));
                res_cpu.setTabIndex(21);
                res_cpu.setText("");

                label10.setLocation(new Point(176, 376));
                label10.setSize(new Point(80, 16));
                label10.setTabIndex(5);
                label10.setTabStop(false);
                label10.setText("Error");
                label10.setTextAlign(HorizontalAlignment.RIGHT);

                edit11.setEnabled(false);
                edit11.setLocation(new Point(272, 376));
                edit11.setSize(new Point(112, 20));
                edit11.setTabIndex(19);
                edit11.setText("");
```

```
label11.setLocation(new Point(520, 112));
label11.setSize(new Point(48, 16));
label11.setTabIndex(36);
label11.setTabStop(false);
label11.setText("Banderas:");

label12.setBackColor(Color.WHITE);
label12.setLocation(new Point(480, 136));
label12.setSize(new Point(104, 32));
label12.setTabIndex(43);
label12.setTabStop(false);
label12.setText("OF sumadores");

label13.setBackColor(Color.WHITE);
label13.setLocation(new Point(480, 152));
label13.setSize(new Point(104, 32));
label13.setTabIndex(42);
label13.setTabStop(false);
label13.setText("+ Infinito");

label14.setBackColor(Color.WHITE);
label14.setLocation(new Point(480, 168));
label14.setSize(new Point(104, 32));
label14.setTabIndex(41);
label14.setTabStop(false);
label14.setText("- Infinito");

label15.setBackColor(Color.WHITE);
label15.setLocation(new Point(480, 184));
label15.setSize(new Point(104, 32));
label15.setTabIndex(40);
label15.setTabStop(false);
label15.setText("NaN");

label16.setBackColor(Color.WHITE);
label16.setLocation(new Point(480, 200));
label16.setSize(new Point(104, 32));
label16.setTabIndex(39);
label16.setTabStop(false);
label16.setText("OF en corrimiento");

label17.setBackColor(Color.WHITE);
label17.setLocation(new Point(480, 216));
label17.setSize(new Point(104, 32));
label17.setTabIndex(38);
label17.setTabStop(false);
label17.setText("Division por cero");

label18.setBackColor(Color.WHITE);
label18.setLocation(new Point(480, 232));
label18.setSize(new Point(104, 32));
label18.setTabIndex(37);
label18.setTabStop(false);
label18.setText("Resultado imaginario");

band1.setEnabled(false);
band1.setLocation(new Point(584, 136));
band1.setSize(new Point(24, 20));
band1.setTabIndex(16);
band1.setText("");
band1.setTextAlign(HorizontalAlignment.CENTER);

band2.setEnabled(false);
band2.setLocation(new Point(584, 152));
band2.setSize(new Point(24, 20));
band2.setTabIndex(15);
band2.setText("");
band2.setTextAlign(HorizontalAlignment.CENTER);

band3.setEnabled(false);
```

```
band3.setLocation(new Point(584, 168));
band3.setSize(new Point(24, 20));
band3.setTabIndex(14);
band3.setText("");
band3.setTextAlign(HorizontalAlignment.CENTER);

band4.setEnabled(false);
band4.setLocation(new Point(584, 184));
band4.setSize(new Point(24, 20));
band4.setTabIndex(13);
band4.setText("");
band4.setTextAlign(HorizontalAlignment.CENTER);

band5.setEnabled(false);
band5.setLocation(new Point(584, 200));
band5.setSize(new Point(24, 20));
band5.setTabIndex(12);
band5.setText("");
band5.setTextAlign(HorizontalAlignment.CENTER);

band6.setEnabled(false);
band6.setLocation(new Point(584, 216));
band6.setSize(new Point(24, 20));
band6.setTabIndex(11);
band6.setText("");
band6.setTextAlign(HorizontalAlignment.CENTER);

band7.setEnabled(false);
band7.setLocation(new Point(584, 232));
band7.setSize(new Point(24, 20));
band7.setTabIndex(10);
band7.setText("");
band7.setTextAlign(HorizontalAlignment.CENTER);

label19.setLocation(new Point(408, 352));
label19.setSize(new Point(96, 16));
label19.setTabIndex(45);
label19.setTabStop(false);
label19.setText("Conectando...");

msgerror.setLocation(new Point(408, 376));
msgerror.setSize(new Point(192, 16));
msgerror.setTabIndex(44);
msgerror.setTabStop(false);
msgerror.setText("Hola!");

Acerca.setText("Acerca de...");
Acerca.addOnClick(new EventHandler(this.Acerca_click));

Ayuda.setMenuItems(new MenuItem[] {
                                  Acerca});
Ayuda.setText("Ay&uda");

mainMenu1.setMenuItems(new MenuItem[] {
                                         Ayuda});
/* @designTimeOnly mainMenu1.setLocation(new Point(16, 8)); */

this.setBackColor(Color.WHITE);
this.setText("ALU Logaritmica");
this.setAutoScaleBaseSize(new Point(5, 13));
this.setClientSize(new Point(618, 398));
this.setMenu(mainMenu1);

label20.setLocation(new Point(408, 328));
label20.setSize(new Point(88, 16));
label20.setTabIndex(46);
label20.setTabStop(false);
label20.setText("Mensajes:");

label21.setLocation(new Point(72, 112));
label21.setSize(new Point(64, 16));
```

```
                label21.setTabIndex(48);
                label21.setTabStop(false);
                label21.setText("Operaciones:");

                this.setNewControls(new Control[] {
                                                        label21,
                                                        label20,
                                                        msgerror,
                                                        label19,
                                                        band7,
                                                        band6,
                                                        band5,
                                                        band4,
                                                        band3,
                                                        band2,
                                                        band1,
                                                        label18,
                                                        label17,
                                                        label16,
                                                        label15,
                                                        label14,
                                                        label13,
                                                        label12,
                                                        label11,
                                                        edit11,
                                                        label10,
                                                        res_cpu,
                                                        res_alu,
                                                        label9,
                                                        label8,
                                                        label7,
                                                        RC,
                                                        codigo,
                                                        label2,
                                                        radioButton6,
                                                        radioButton5,
                                                        radioButton4,
                                                        radioButton3,
                                                        radioButton2,
                                                        radioButton1,
                                                        edit1,
                                                        edit3,
                                                        pictureBox1,
                                                        label4,
                                                        label3,
                                                        edit6,
                                                        edit5,
                                                        label6,
                                                        label5,
                                                        edit2,
                                                        edit4,
                                                        Label2,
                                                        label1});
        }

//-------------------------------------------------------------
        //Punto de entrada principal para la aplicación.

        public static void main(String args[])
        {
                Application.run(new Form1());
        }

//-------------------------------------------------------------
        //---funcion conversion de dato int log a String---

        public String conversion(int num){
                StringBuffer tex = new StringBuffer(16);
                int mascara=1<<15;
                int i=0;
```

```
                for(i=0;i<16;i++){
                   ((num & mascara) == 0){
                                tex.append('0');
                        }
                        else{
                                tex.append('1');
                        }
                        if(i==6 || i==1){
                                tex.append(' ');
                        }
                        num <<= 1;
                }
                return tex.toString();
        }

//-------------------------------------------------------------
        //---funcion conversion especial para Banderas---

        public String conversion3(int num){
                StringBuffer tex = new StringBuffer(8);
                int mascara=1<<6;
                int i=0;

                for(i=0;i<7;i++){
                        if((num & mascara) == 0){
                                tex.append('0');
                        }
                        else{
                                tex.append('1');
                        }
                        num <<= 1;
                }
                return tex.toString();
        }
//-------------------------------------------------------------
        //---funcion conversion2 de dato String a int log---

        public int conversion2(String str){
                if(str.length() == 18){
                        int acum=0, i;
                        char[] texto=new char[18];
                        int k=15;

                        str.getChars(0,18,texto,0);
                        for(i=0;i<18;i++){
                                if(i==2 || i==8){
                                        ++i;
                                }

                                if(texto[i]=='1'){
                                        acum+=(int)Math.pow(2,k);
                                        --k;
                                }else{
                                        if(texto[i]=='0'){
                                                --k;
                                        }else{
                                                return 65536;
                                        }
                                }
                        }
                        return acum;
                }
                return 65536;
        }
//-------------------------------------------------------------

        //---funcion tratamiento de dato int log op_1 y op_2---
        public double[] tratamiento(int valor){
                double op[]=new double[2];
                double a=10;
                double b=-10;
```

```
                double c=0;

                if(valor>=32768){
                        if(valor==40959){     // 10 01111 111111111
                                op[0]=a/c;
                                op[1]=0;
                                return op;
                        }
                        if(valor==57343){     // 11 01111 111111111
                                op[0]=b/c;
                                op[1]=0;
                                return op;
                        }
                        if(valor==40960 || valor==57344){    // 1x 10000 000000000
                                op[0]=c/c;
                                op[1]=0;
                                return op;
                        }
                        // detectando como cero
                        op[0]=c;
                        op[1]=b/c;
                        return op;
                }
                int mascara1=16383;   //00 11111 111111111
                int mascara2=8192;

                op[1]=(double)(mascara1 & valor);

                if( ( valor & mascara2 ) != 0 ){
                        op[1]=op[1]-2*mascara2;
                }

                op[1]=op[1]/Math.pow(2,9);
                op[0]=Math.pow(2,op[1]);

                if( ( ( mascara1 + 1 ) & valor ) != 0 ){
                        op[0]=-op[0];
                }

                return op;
        }

//-------------------------------------------------------------

        //---funcion sendOperacion de envio de operacion---

        public void sendOperacion(int op1, int op2, byte operand){
                byte[] writebuf=new byte[5];

                int mascara1=255;                    // 0000 0000 1111 1111
                int mascara2=mascara1 << 8;   // 1111 1111 0000 0000
                int termino=0;

                writebuf[0]=operand;

                writebuf[2]=(byte)(mascara1 & op1);
                termino=(mascara2 & op1)>>8;
                writebuf[1]=(byte)(termino);

                writebuf[4]=(byte)(mascara1 & op2);
                termino=(mascara2 & op2)>>8;
                writebuf[3]=(byte)(termino);

                try {
                        outputbuffer.write(writebuf);
                } catch (IOException e) {
                        msgerror.setText("err 6: no se pudo escribir buffer de salida");
                }
                return;
        }
```

```
//---------------------------------------------------------
//---------------------------------------------------------

        //---handler listener---

        public void serialEvent(SerialPortEvent event) {
                switch (event.getEventType()) {
                case SerialPortEvent.BI:
                case SerialPortEvent.OE:
                case SerialPortEvent.FE:
                case SerialPortEvent.PE:
                case SerialPortEvent.CD:
                case SerialPortEvent.CTS:
                case SerialPortEvent.DSR:
                case SerialPortEvent.RI:
                case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
                        break;
                case SerialPortEvent.DATA_AVAILABLE:
                        byte[] readbuf = new byte[3];

                        try{
                                int numBytes=inputbuffer.read(readbuf);
                                int mascara1=255; // 1111 1111
                                int termino1=0;
                                int termino2=0;

                                termino1=(int)readbuf[2];
                                Banderas=termino1 & mascara1;

                                String ban=conversion3(Banderas);

                                band1.setText(" "+ban.charAt(6));
                                band2.setText(" "+ban.charAt(5));
                                band3.setText(" "+ban.charAt(4));
                                band4.setText(" "+ban.charAt(3));
                                band5.setText(" "+ban.charAt(2));
                                band6.setText(" "+ban.charAt(1));
                                band7.setText(" "+ban.charAt(0));

                                termino1=(int)readbuf[1];
                                termino1=termino1 & mascara1;
                                termino2=(int)readbuf[0];
                                termino2=termino2 & mascara1;
                                termino1=termino1 << 8;
                                opc3=termino1 | termino2;

                                double ops[]=tratamiento(opc3);
                                RC.setText(conversion(opc3));
                                opc1=ops[0];
                                opc2=ops[1];
                                res_alu.setText(Double.toString(opc1));
                                error1=opd1-opc1;
                                edit11.setText(Double.toString(error1));
                        }
                        catch(IOException e){
                                RC.setText(" ");
                                res_alu.setText(" ");
                                edit11.setText(" ");
                                msgerror.setText("err6: no se pudo recibir");
                        }
                }
        }
}
```

ANEXO C CÓDIGO DE PROGRAMACIÓN PARA EL MICROCONTROLADOR

CODIGO_INTERFAZ.ASM

```
;  ------------------------------------------------------------------------------
;  | TÍTULO:          ALU LOGARÍTMICA                                           |
;  | T.G.:            0432                                                      |
;  |                                                                            |
;  | INTEGRANTES:     MANUEL ALBERTO CARRERA OSORIO                             |
;  |                  EFRÉN OCTAVIO LÓPEZ URIBE                                 |
;  |                  JUAN DIEGO NARANJO LÓPEZ                                  |
;  |                                                                            |
;  | DIRECTORES:      ING. ALEJANDRA MARIA GONZÁLEZ                             |
;  |                  ING. FRANCISCO VIVEROS                                    |
;  |                                                                            |
;  | CODIGO:          codigo_interfaz.asm                                       |
;  |                                                                            |
;  | DESCRIPCIÓN:     Este código fue desarrollado en lenguaje ensamblador,     |
;  |                  para el pic18f8720, el microcontrolador realiza la función |
;  |                  de proveer operaciones a la ALU logarítmica, que a su vez |
;  |                  recibe de un computador por medio de un protocolo de      |
;  |                  comunicación serial asíncrono RS-232.                     |
;  |                                                                            |
;  | MODIFICACIONES:                                                            |
;  |                                                                            |
;  ------------------------------------------------------------------------------

title "PIC18F8720 proyecto final"
list p=18f8720,f=inhx32
#include <p18f8720.inc>

VAR1    equ     0x00    ;variable que lleva la cuenta de los bytes que ha recibido USART
VAR2    equ     0x01    ;variable que guarda temporalmente el codigo de bandera concatenado
VAR3    equ     0x02
MASK1   equ     0x03
MASK2   equ     0x1F

        org     00h
        goto    start

        org     08h
        goto    rutina

        org     18h
        goto    rutina

        org     20h
start

        ;-------configuracion del dispositivo----------------------

        ;inicializacion

        bcf     WDTCON,SWDTEN  ;evitar activacion del wdt
        clrf    BSR                             ;situa el banco en 0x00

        movlw   0x05           ;carga 5 -> var1
        movwf   VAR1

        movlw   0x55           ;número de prueba
        movwf   PORTB          ;llena con número de prueba el puerto b
        movwf   PORTD          ;llena con número de prueba el puerto d
        movwf   PORTE          ;llena con número de prueba el puerto e
        movwf   PORTF          ;llena con número de prueba el puerto f
```

```
        clrf    PORTA           ;llena de ceros el puerto a

        ;configuracion entradas

        clrf    TRISB           ;declara puerto b como salida del sistema op1[7..0]
        clrf    TRISD           ;declara puerto d como salida del sistema op1[15..8]
        clrf    TRISE           ;declara puerto e como salida del sistema op2[7..0]
        clrf    TRISF           ;declara puerto f como salida del sistema op2[15..8]

        bcf     TRISA,0         ;
        bcf     TRISA,1         ;
        bcf     TRISA,2         ;

        ;configuracion salidas

        setf    TRISH           ;declara puerto h como entrada del sistema opOUT[7..0]
        setf    TRISJ           ;declara puerto f como entrada del sistema opOUT[15..8]
        setf    TRISG           ;declara puerto g como entrada del sistema band[4..0]

        bsf     TRISC,0         ;band[5]
        bsf     TRISC,1         ;band[6]

        ;congiguracion perifericos einterrupciones

        bsf     TRISC,7         ;configurar puerto c para uso con usart
        bcf     TRISC,6         ;

        movlw   0x19            ;declara la tasa de tx rx, 9.6 kbauds en modo de alta
frecuencia
        movwf   SPBRG1

        movlw   0x90            ;configura modo de rx
        movwf   RCSTA1

        movlw   0x26            ;configura modo de tx, habilitado de una vez
        movwf   TXSTA1

        bsf     ADCON1, PCFG0   ;configura conversor ad
        bsf     ADCON1, PCFG1   ;
        bsf     ADCON1, PCFG2   ;
        bsf     ADCON1, PCFG3   ;

        bcf     RCON, IPEN      ;deshabilita prioridades de interrupcion
        bsf     PIE1, RC1IE     ;habilita int usart rx
        bsf     INTCON, PEIE    ;habilita int periferico
        bsf     INTCON, GIE     ;habilita int global

        ;-------programa principal-------------------------------------

loop
        nop                                 ;loop sin utilidad
        nop
        nop
        nop
        goto    loop


        ;-------rutina de interrupcion---------------------------------

rutina
        movff   VAR1,VAR3
        decf    VAR1,F

        dcfsnz  VAR3,F          ;ciclo que funciona como un while, que determina byte i rx
        goto    byte4
        dcfsnz  VAR3,F
        goto    byte3;
        dcfsnz  VAR3,F
        goto    byte2;
        dcfsnz  VAR3,F
        goto    byte1;
```

```
        dcfsnz  VAR3,F
        goto    byte0;
        retfie

byte0
        movff   RCREG1,PORTA    ;mueve dato recibido al puerto b op1[7..0]
        retfie

byte1
        movff   RCREG1,PORTD    ;mueve dato recibido al puerto d op1[15..8]
        retfie

byte2
        movff   RCREG1,PORTB    ;mueve dato recibido al puerto e op2[7..0]
        retfie

byte3
        movff   RCREG1,PORTF    ;mueve dato recibido al puerto f op2[15..8]
        retfie

byte4
        movff   RCREG1,PORTE    ;mueve dato recibido al puerto a opcode

        movlw   0x05            ;carga de nuevo var1 con 5
        movwf   VAR1

        movff   PORTH,TXREG1    ;envia dato del puerto h opOUT[7..0]
        nop
        nop
        nop
        movff   PORTJ,TXREG1    ;envia dato del puerto j opOUT[15..8]

        ;concatenando banderas RC[1..0] y RG[4..0]

        movf    PORTC,W         ;mueve el puerto C a W
        andlw   MASK1           ;aplica mascara 1
        movwf   VAR2            ;mueve a var2
        rlncf   VAR2,F          ;rota datos a la izq 5 posiciones
        rlncf   VAR2,F
        rlncf   VAR2,F
        rlncf   VAR2,F
        rlncf   VAR2,F
        movf    PORTG,W         ;mueve datos del puerto G a W
        andlw   MASK2           ;aplica mascara 2
        iorwf   VAR2,F          ;hace or con los datos de Var2 y W

        ;confirmar que el txreg esta vacio para enviar el tercer byte, banderas, un loop

loop2
        nop
        nop
        btfss   PIR1,TX1IF
        goto    loop2

        movff   VAR2,TXREG1     ;envia dato banderas
        retfie

        end
```

ANEXO D CÓDIGO EN MATLAB PARA GENERAR LAS TABLAS DE FUNCIONES

LUT_ARCHIVO.M

```
%  ------------------------------------------------------------------------------
%  | TÍTULO:           ALU LOGARÍTMICA                                          |
%  | T.G.:             0432                                                     |
%  |                                                                            |
%  | INTEGRANTES:      MANUEL ALBERTO CARRERA OSORIO                            |
%  |                   EFRÉN OCTAVIO LÓPEZ URIBE                                |
%  |                   JUAN DIEGO NARANJO LÓPEZ                                 |
%  |                                                                            |
%  | DIRECTORES:       ING. ALEJANDRA MARIA GONZÁLEZ                            |
%  |                   ING. FRANCISCO VIVEROS                                   |
%  |                                                                            |
%  |                                                                            |
%  | CODIGO:           lut_archivo.m                                            |
%  |                                                                            |
%  | DESCRIPCIÓN:      Este código esta escrito en lenguaje de programación de  |
%  |                   MATLAB, el código calcula las tablas de funciones y genera |
%  |                   los archivos necesarios para programar las memorias EPROM |
%  |                   que implementaran las funciones.                         |
%  |                                                                            |
%  | MODIFICACIONES:                                                            |
%  |                                                                            |
%  ------------------------------------------------------------------------------

%tabla para la suma log2(1+2^-abs(in))
clc
clear
tic

L=5;    %número de bits parte entera
M=9;    %número de bits parte fraccionaria
N=L+M;  %número de bits total

%------------------------------------------------------------------------------
%vector de peso [-16 8 4 2 1 1/2 1/4 1/8 1/16 1/32 1/64 1/128 1/256 1/512]
%el producto punto de un número binario en complemento 2 con el vector de
%peso, genera el número en decimal que representa.

vec_peso=-(L-1):M;
vec_peso=2.^-vec_peso;
vec_peso(1)=-vec_peso(1);

disp(['termino fase 1 tiempo transcurrido: ' num2str(toc)])

%------------------------------------------------------------------------------
%creacion d1,t1, dominio de las funciones.
%d1 es una cadena decimal entera
%t1 es una matriz binaria que representa a d1
%d3 es la representacion en decimal de los datos en t1 segun el vector de
%peso
%d4 es una cadena que indica el signo del número

d1=(0:2^N-1)';
t1=dec2bin(d1,N);
d2=double(double(t1)==49);
d3=d2*(vec_peso)';
d4=sign(d3)==-1;

disp(['termino fase 2 tiempo transcurrido: ' num2str(toc)])

%------------------------------------------------------------------------------
```

```
%creacion tabla sb con la funcion log2(1+2^-|x|)

d5_sb=log2(1+2.^-abs(d3));   %funcion suma que se implementa en la tabla
d6_sb=round(d5_sb*2^M);
d6_sb=d6_sb+2^(N).*(sign(d6_sb)==-1);
t2_sb=dec2bin(d6_sb,16);
disp(['termino fase 3 tiempo transcurrido: ' num2str(toc)])

%-------------------------------------------------------------------------
%creacion tabla con la funcion log2(1-2^-|x|)

d5_db=log2(1-2.^-abs(d3));   %funcion resta que se implementa en la tabla
d6_db=round(d5_db*2^M);
d6_db=d6_db+2^(N).*(sign(d6_db)==-1);
t2_db=dec2bin(d6_db,16);
t2_db(1,:)='0100000000000000';
disp(['termino fase 4 tiempo transcurrido: ' num2str(toc)])

%-------------------------------------------------------------------------
%organizacion de los datos en dos memorias separadas, partiendo la tabla t2
%t2 representa los datos utiles, la organizacion esta hecha de tal forma,
%que ya se tuvo en cuenta la direccion que representa cada dato.

t2=[t2_sb;t2_db];
msb=t2(:,1:8);
msb=bin2dec(msb);
lsb=t2(:,9:16);
lsb=bin2dec(lsb);

%-------------------------------------------------------------------------
%escritura de los archivos
%datos_texto.rtf contiene informacion de las memorias en formato de texto
%lut_msb.hex contiene informacion memoria con byte mas significativo
%lut_lsb.hex contiene informacion memoria con byte menos significativo
%datos_texto.rtf contiene la informacion  en texto de la tabla

d1a=(0:2^(N+1)-1)';
t1a=dec2bin(d1a,N+1);
espacio=char(32*ones(2^(N+1),1));
texto=[t1a espacio t2 espacio dec2hex(bin2dec(t1a)) espacio dec2hex(bin2dec(t2))];
fid=fopen('datos_texto.rtf','w+');
for i=1:2^(N+1)
    fprintf(fid,texto(i,:));
    fprintf(fid,'\n');
end
fclose(fid);

%archivo que contiene la informacion que guarda la memoria con el
%byte mas significativo
fid=fopen('lut_msb.hex','w+');
fwrite(fid,msb,'uint8');
fclose(fid);

%archivo que contiene la informacion que guarda la memoria con el
%byte menos significativo
fid=fopen('lut_lsb.hex','w+');
fwrite(fid,lsb,'uint8');
fclose(fid);

clear
disp(['termino fase final tiempo transcurrido: ' num2str(toc)])
```

ANEXO E ESQUEMÁTICOS

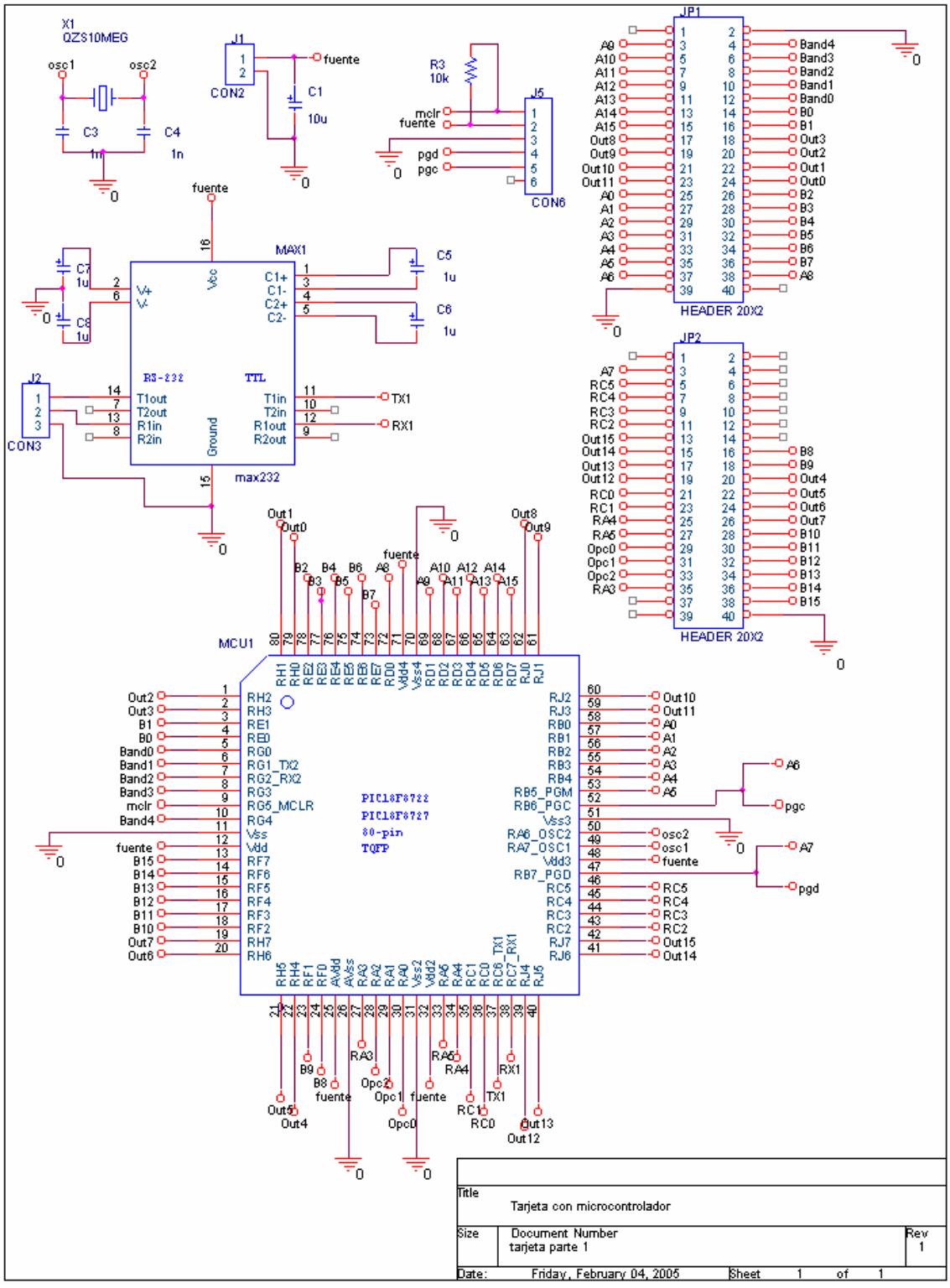


Figura 1. Tarjeta con microcontrolador

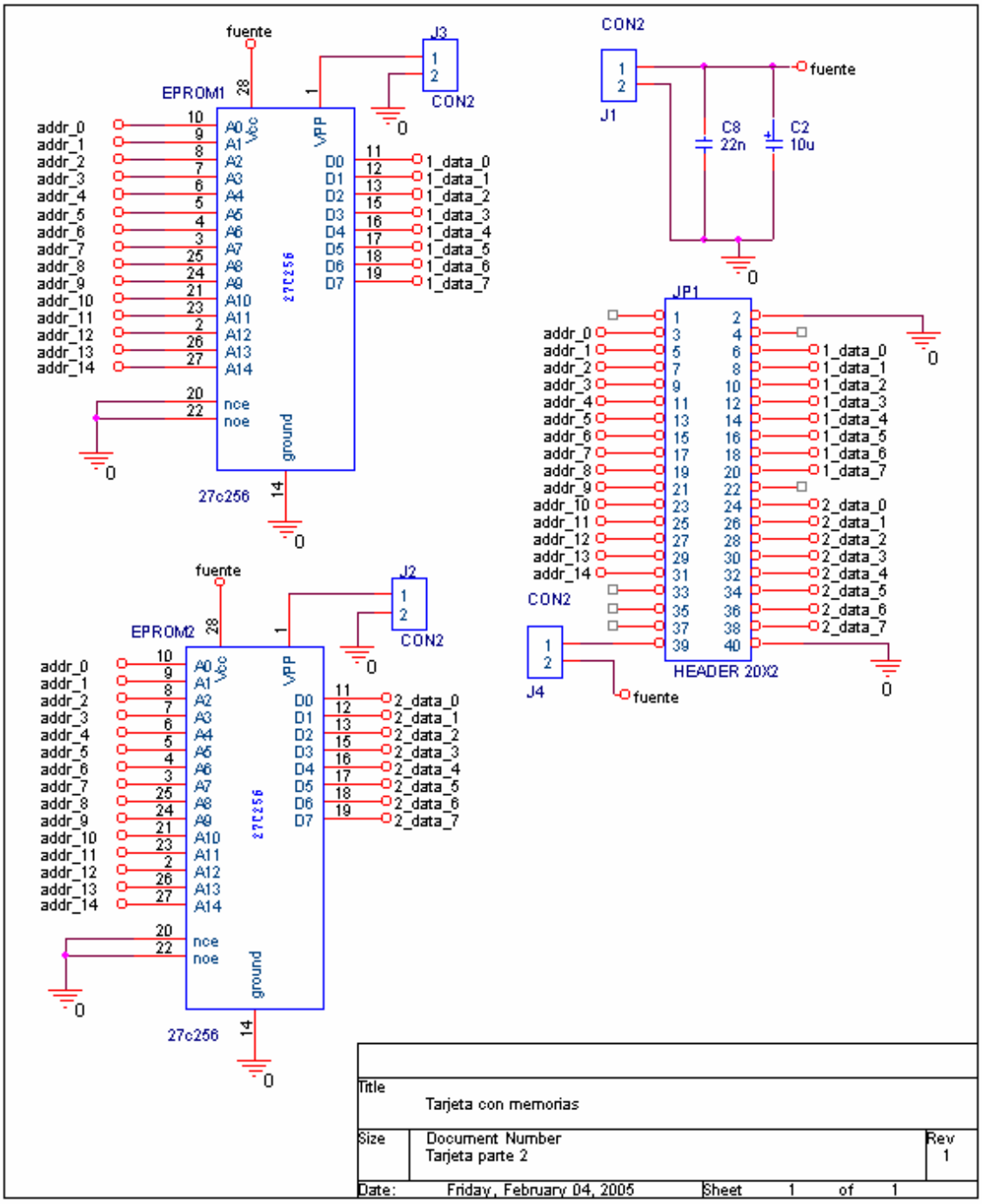


Figura 2. Tarjeta con memorias

ANEXO F GUÍA RÁPIDA DE QUARTUS II WEB EDITION SOFTWARE

¿QUÉ ES QUARTUS II?

Quartus II es una herramienta de desarrollo y análisis de proyectos de técnicas digitales por medio de la utilización de diferentes lenguajes de descripción de hardware como lo son VHSIC (Very High Speed Integrated Circuit) Hardware Description Language (VHDL por sus siglas en inglés) o Verilog, junto con otras herramientas EDA.

Por medio de la utilización de este software se pueden crear, compilar, simular y configurar en dispositivos de lógica programable los diferentes proyectos para los cuales sean necesarios involucrar dispositivos de alta densidad de elementos lógicos y elementos de memoria.

A continuación, se presentará al lector una guía rápida para el uso de esta herramienta de desarrollo, los ítems que aparecen a continuación se redactaron emulando el proceso en el desarrollo de cualquier proyecto para aquellos lectores que ya están familiarizados con este tipo de herramientas.

Las diferentes funcionalidades presentan (generalmente), tres maneras de acceder a ellas las cuales son: acceso por íconos de escritorio, por medio de la barra de menú y combinación de teclas de acceso rápido, en lo posible serán indicadas las tres.

En la figura 1 se muestra el ambiente de trabajo de esta herramienta de desarrollo.

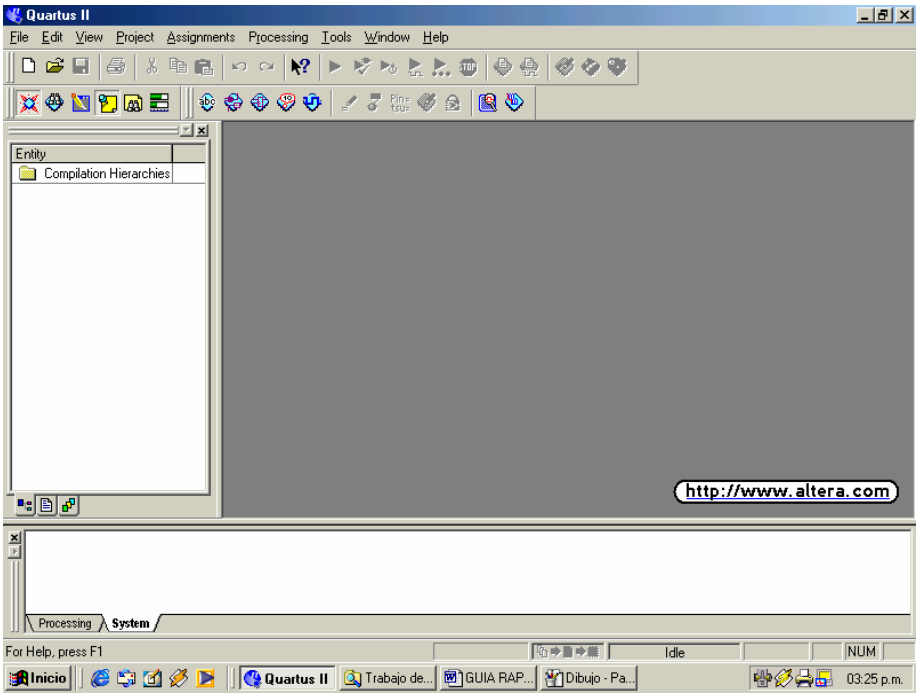


Figura 1.

REDACCIÓN DE CÓDIGO

Para abrir un archivo de texto, donde se hará la redacción del código, se puede realizar por medio de la herramienta de menú siguiendo el siguiente proceso:

- Clic en la opción *"File"*.
- Se elige la opción *New*, la cual despliega la ventana en la *Figura 2*.

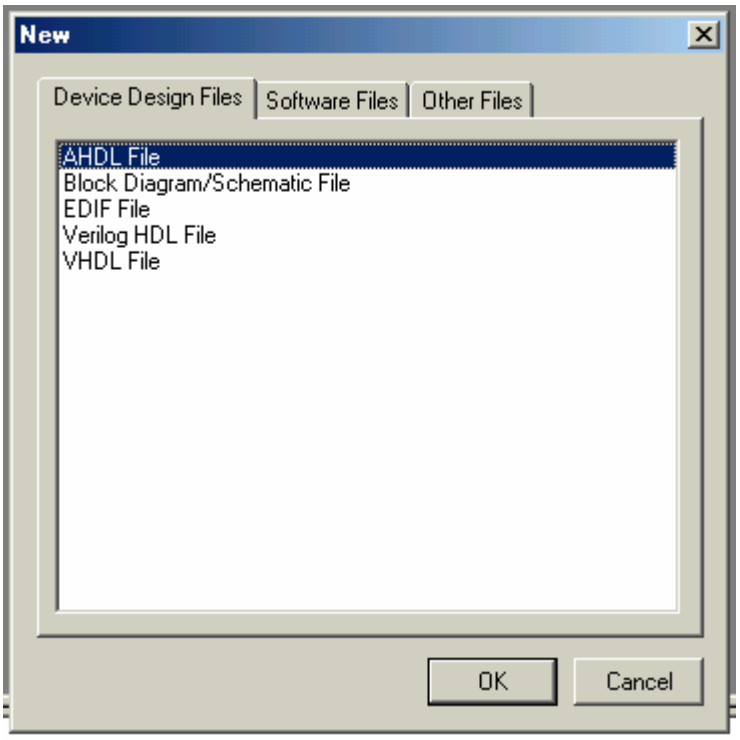


Figura 2.

- Se selecciona el tipo de archivo a trabajar entre las siguientes opciones: *"Device Design Files", "Software Files"* o *"Other Files"*; en nuestro caso elegiremos *"VHDL File"* de la primera opción.
- También es posible tener acceso a un editor de texto por medio del icono de la Figura 3 y guardarlo como el correspondiente tipo de archivo deseado, operación que se pueden localizar en la opción File de la barra de menú, como se muestra en la Figura 4.

Figura 3.

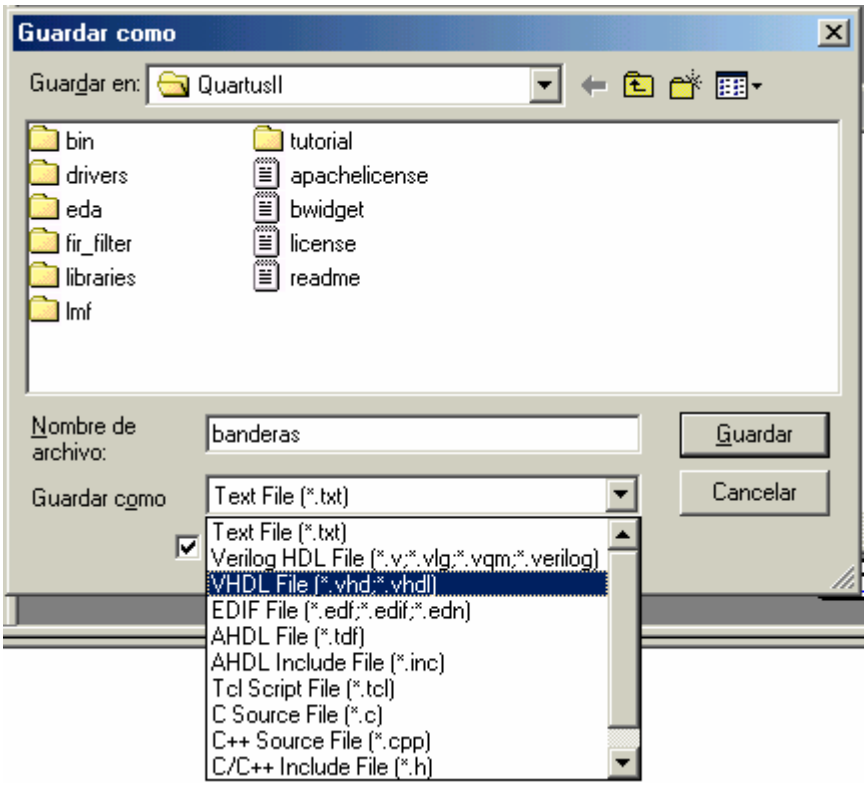


Figura 4.

CREACIÓN DEL PROYECTO

Para la creación de un proyecto el cual incluya uno o varios archivos de VHDL el(los) cual(es) contenga(n) el(los) código(s) que haga(n) la descripción del hardware deseado, es necesario realizar los siguientes pasos.

- Clic en la opción "*File*" de la barra de menú.
- Elegir la opción "*New Proyect Wizard"* como es mostrado en la *Figura 5*, la cual abrirá un ayudante, mostrado en la *Figura 6*, para especificar el directorio en donde se guardará el proyecto y se generará la base de datos asociada a este proyecto. También es indicado el nombre del proyecto, y por último la entidad de nivel jerárquico superior.

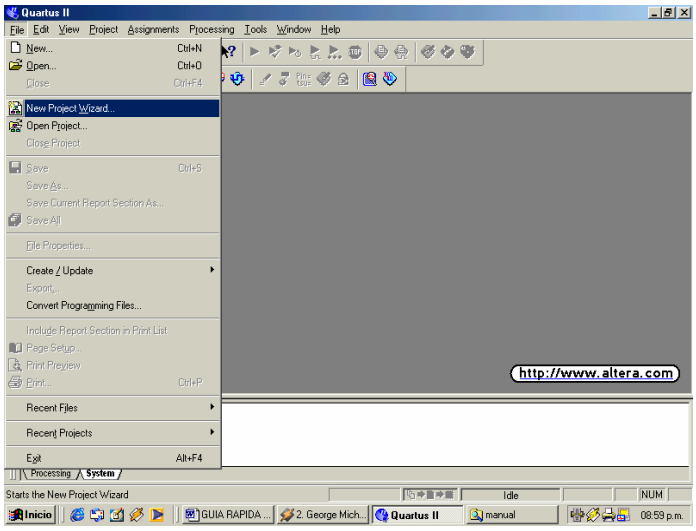


Figura 5.

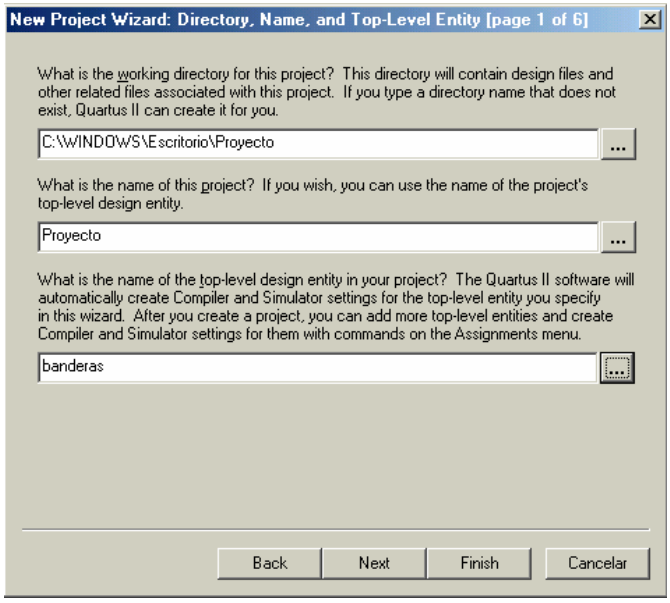


Figura 6.

- Posteriormente se agregan al proyecto todos los archivos VHDL generados para este proyecto, esto se logra dando clic en el botón que presenta puntos suspensivos y buscando los archivos deseados en los directorios donde se encuentren alojados, señalarlos y dar la opción abrir para que queden alojados en el cuadro en blanco, tal y como es mostrado en la Figura 7.

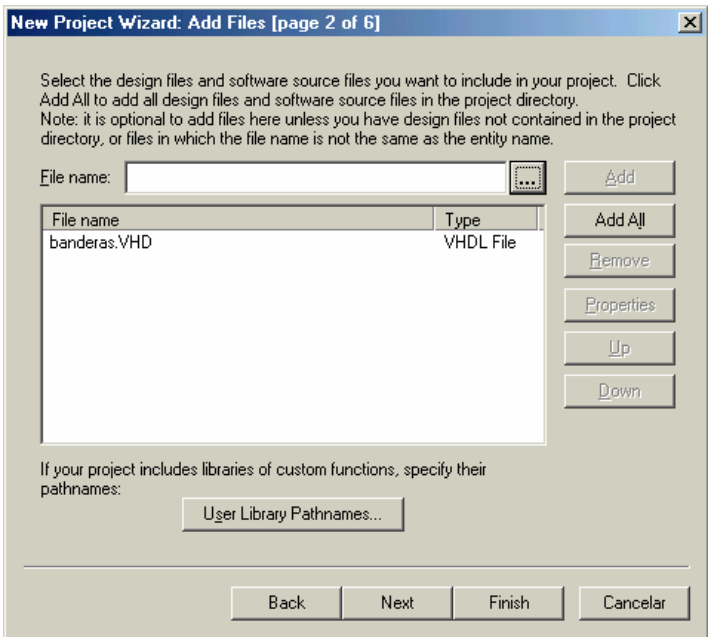


Figura 7.

- Si se utilizará una herramienta de tipo EDA, se selecciona el tipo de herramienta y en el botón desplegable *"Tool Name"* es seleccionada, tal como se muestra en la Figura 8.

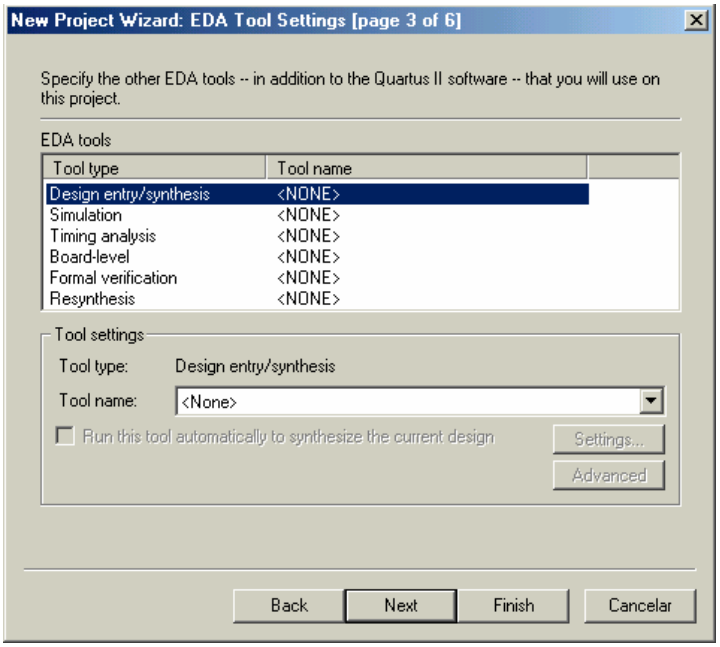


Figura 8.

- El siguiente paso a seguir es escoger el tipo de familia de dispositivo de lógica programable que se utilizará, o dejar que el software elija el dispositivo, tal y como es mostrado en la Figura 9.

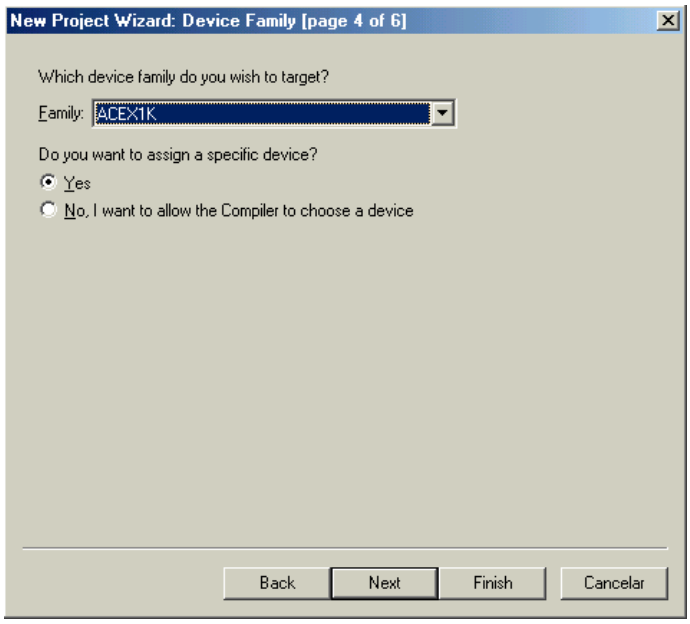


Figura 9.

- Posteriormente, se indica el dispositivo específico dentro de la familia elegida, para esto el software nos presenta unos parámetros de filtraje para ubicar más rápidamente el dispositivo que se va a trabajar, estos son *"Package", "Pin count"* o *"Speed grade"*; o simplemente dejar estas opciones en *"Any"* y el software elegirá un dispositivo dentro de la familia, estas opciones se pueden detallar más en la Figura 10.

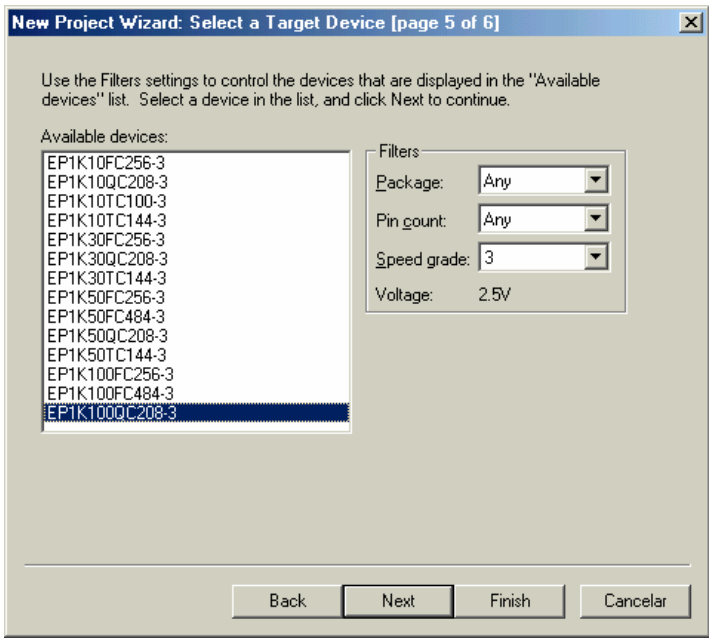


Figura 10.

- En el último paso de este ayudante, el software presenta un resumen de los parámetros elegidos en las ventanas anteriores, y, al dar clic en el botón *"Finish"* se creará el proyecto, tal y como es mostrado en la Figura 11.

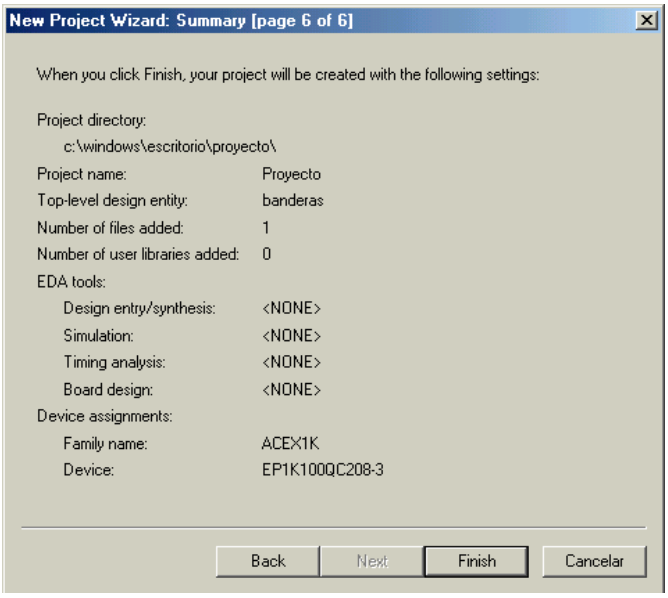


Figura 11.

COMPILACIÓN Y PARÁMETROS DE COMPILACIÓN

Para ingresar los parámetros de compilación los cuales serán aplicados al proyecto creado, es necesario realizar los pasos que se describen a continuación:

- Para definir los parámetros de compilación por primera vez con la ayuda del *"Compiler settings wizard"*, es necesario dar clic en *"Assignments"* y elegir la opción *"Compiler settings wizard",* como es mostrado en la Figura 12.

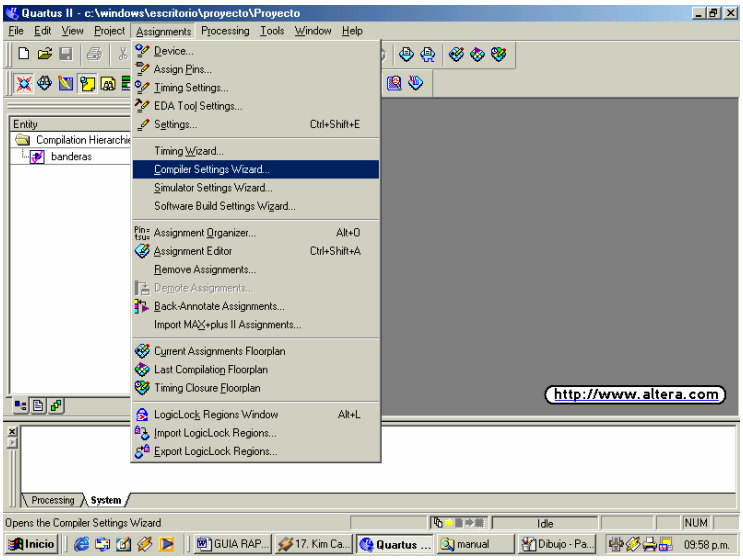


Figura 12.

- En el primer paso de los parámetros de compilación el software nos pide un *"Focus point"* en el cual se le indica al programa la entidad de nivel superior o una entidad de menor nivel jerárquico sobre la cual se desea realizar la compilación del proyecto (se recomienda ir compilando desde entidades de nivel jerárquico inferior y posteriormente compilar las inmediatamente superiores para que se generen los datos en la base de datos y así, al compilar las entidades de nivel superior, se tenga la información requerida de las entidades inferiores en la correspondiente base de datos). En el cuadro denominado *"Settings name"* se escribe el nombre que se le desea dar a los parámetros elegidos o elegir un nombre de la lista del menú desplegable al Figura 13.

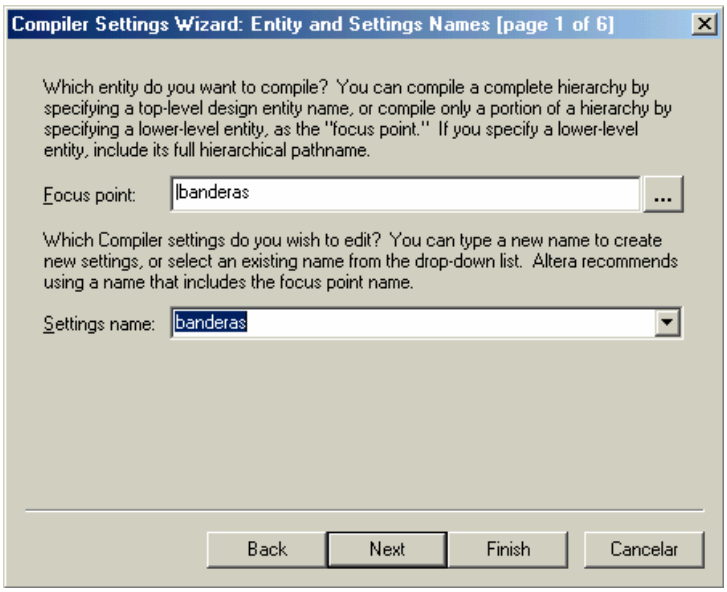


Figura 13.

- Posteriormente, el software nos permite seleccionar unos parámetros de velocidad de compilación versus el espacio que se utilizará en el disco, como se muestra en la Figura 14, como también el número de nodos creados (que se reflejan en la utilización de espacio en el disco donde se crea la base de datos) para realizar la compilación, simulación, y otros procesos que se ejecutan en la utilización de esta herramienta.

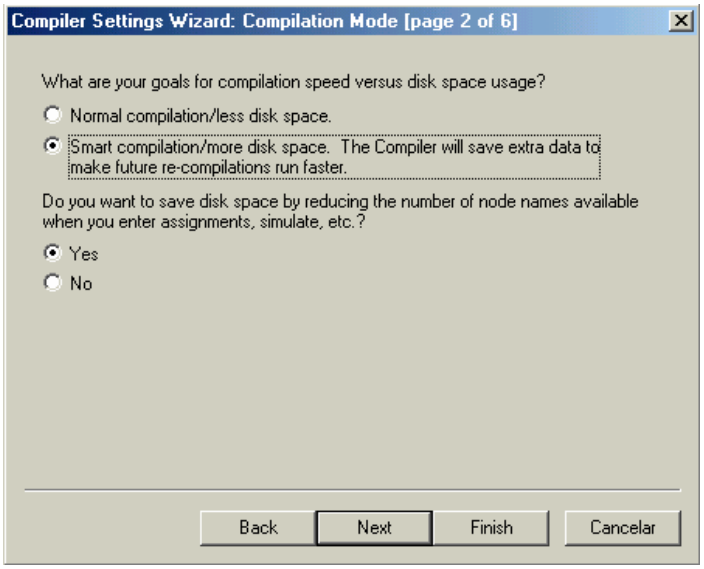


Figura 14.

- Los siguientes dos pasos son para definir el tipo de familia y de dispositivo (si se desea se modifican, pero afecta también los parámetros definidos en el sistema global), y el siguiente

nos pregunta si deseamos una lista de los nodos creados en un archivo *"Verilog Quartus Mapping File"*, pero se puede obviar si no se crea y solo se trabaja en VHDL.

- En el último paso de este ayudante, el software nos presenta un resumen de los parámetros elegidos en las ventanas anteriores, y, al dar clic en el botón *"Finish"* se habrá definido los parámetros de compilación, tal y como se muestra en la Figura 15.

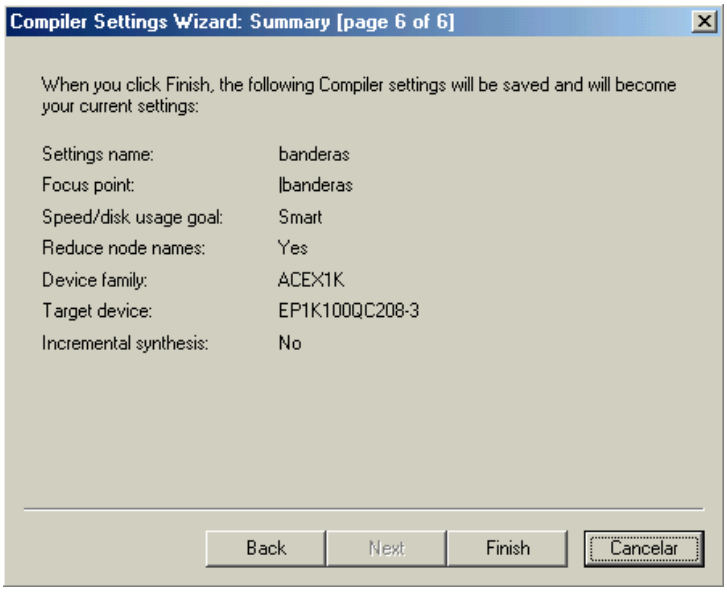


Figura 15.

- Al realizar los pasos anteriores, ya podemos compilar nuestro proyecto, lo cual se hace dando clic en *"Processing"* de la barra de menú y posteriormente seleccionando la opción *"Start Compilation"*, presionando las teclas Ctrl+J, o simplemente presionando el ícono de compilación (triángulo de color morado). Al realizar la compilación del proyecto, el software nos indicará el término del proceso como se muestra en la Figura 16.

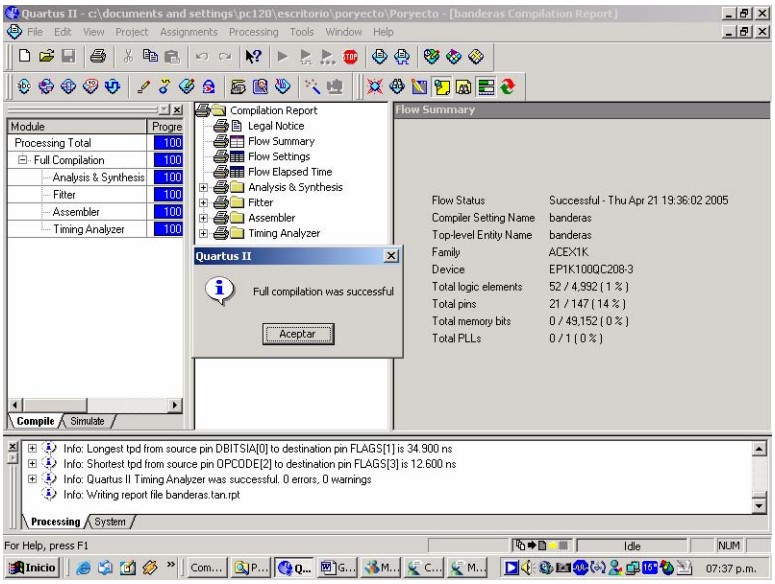


Figura 16.

ELEMENTOS DE ANÁLISIS

Esta herramienta presenta varios elementos de análisis, los cuales nos indican varios parámetros dentro de los cuales podemos realizar diversas operaciones en el proyecto; uno de los elementos es el análisis gráfico, por medio del cual podemos ingresar vectores de prueba y obtener la salida tanto funcional como con análisis de tiempos.

A continuación se detallará de una manera rápida y sencilla la forma de realizar simulaciones para el proyecto creado.

- Para generar el correspondiente archivo en el cual se ingresarán los estímulos exteriores a simular, es necesario dar clic en *"File"*, posteriormente en *"New"* lo que nos mostrará la ventana mostrada en la Figura 17, en donde se elije la opción *"Vector Waveform File"* de la pestaña *"Other Files"*.

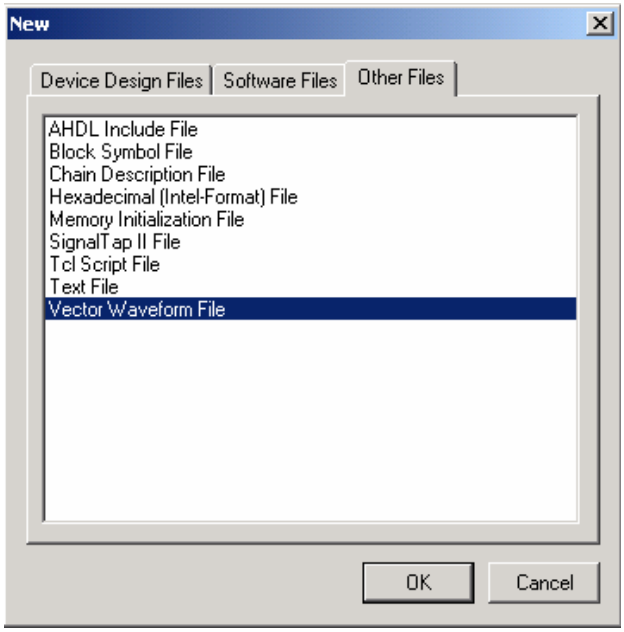


Figura 17.

- Para agregar los parámetros de entrada al sistema y observar las señales de salida a los estímulos, es necesario dar clic derecho sobre el *"Vector Waveform File"* que se acaba de crear (paso anterior), seleccionar la opción *"Insert Node or Bus"* como se muestra en la Figura 18 lo que causará que aparezca el recuadro mostrado en la *Figura 19*, se da clic en el botón *"Node Finder"*. En el cuadro que aparece se selecciona en la casilla *"Filter"* seleccionamos la opción *"Pins: all"* y damos clic en el botón *"Start"* para obtener todas las señales en el recuadro de la izquierda (el de la derecha aparece en blanco), en el cual se muestra las señales que se pueden tomar para posterior simulación tal y como se muestra en la Figura 20.

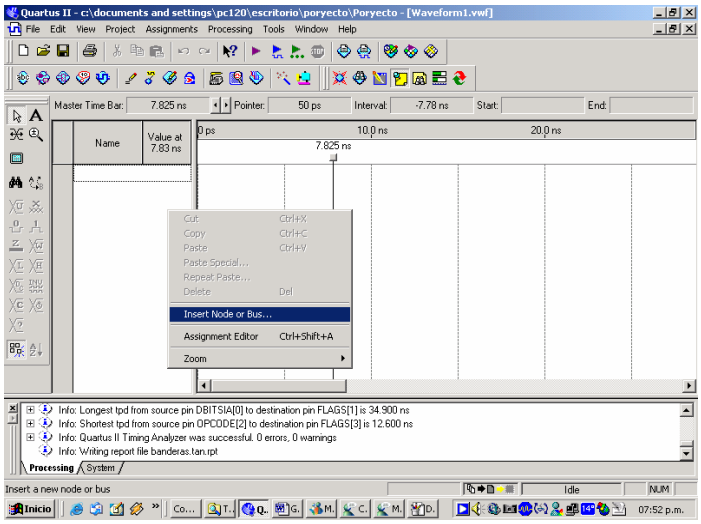


Figura 18.

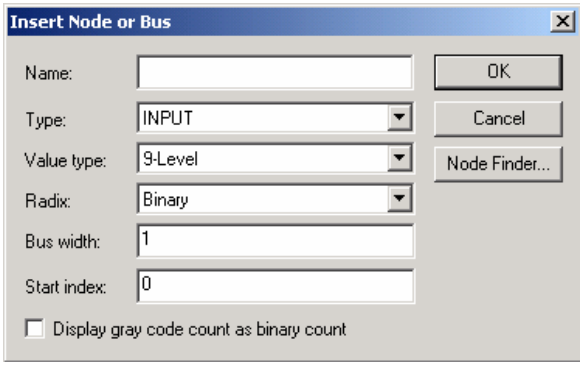


Figura 19.

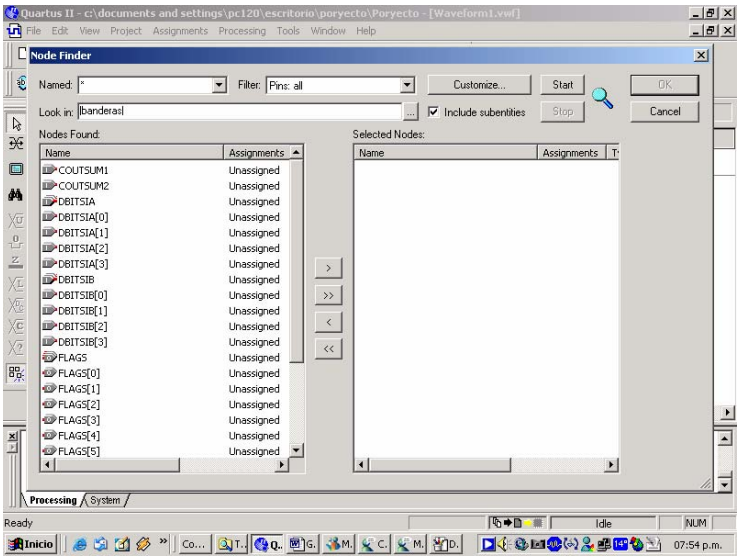


Figura 20.

- Para seleccionar las señales que se desean observar, le damos clic en la señal del recuadro izquierdo y presionamos el botón que se asemeja a una flecha que apunta hacia el cuadro que aparece a la derecha (hay que notar que cuando se tiene un bus aparecen las señales individuales y agrupadas), y realizamos esta operación hasta tener en el recuadro de la derecha todas las señales que se utilizan en nuestro proyecto, tal y como se muestra en la Figura 21, aceptamos dando clic en el botón *"OK"* hasta obtener la visualización mostrada en la Figura 22 (con las señales del proyecto seleccionadas).

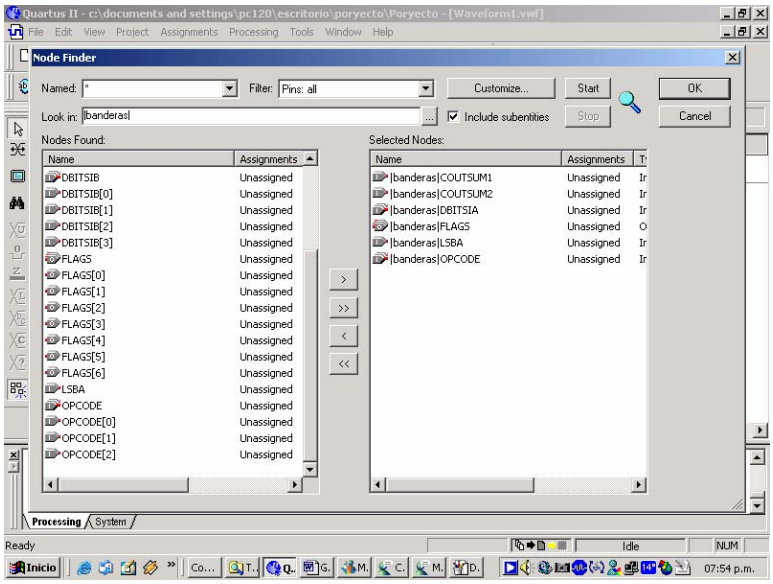


Figura 21.

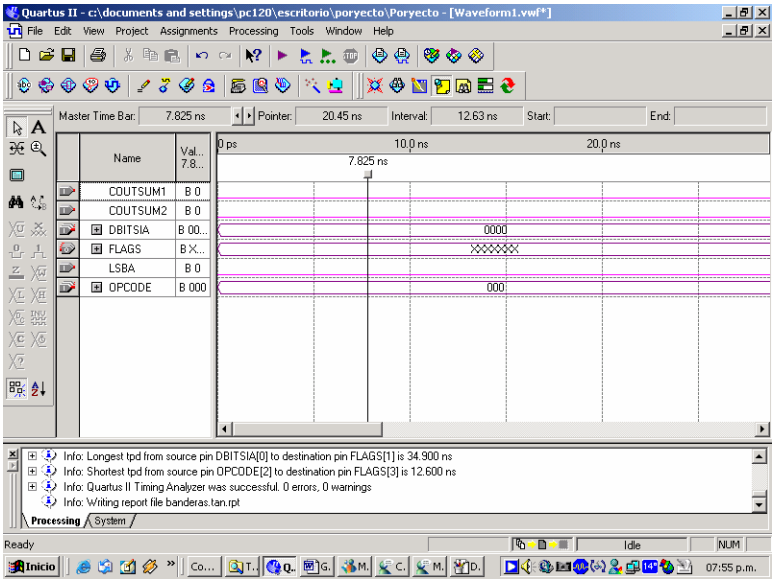


Figura 22.

- Los valores que entran al sistema pueden ingresarse en el vector de simulación de una manera gráfica, dando clic en la señal deseada y arrastrando hasta donde se desea en la línea de tiempo para dejar sombreada el área donde se aplicará el estímulo, y posteriormente dando clic en el botón con el símbolo de interrogación para obtener la pantalla mostrada en la Figura 23, en donde se puede elegir la base en que se desea trabajar (recuadro denominado *"Radix"* Binaria, Hexadecimal, Octal, Decimal con signo o Decimal sin signo) y en el recuadro *"Numeric or named value"* se ingresa el estímulo que se desea dar a la señal seleccionada por el tiempo indicado.

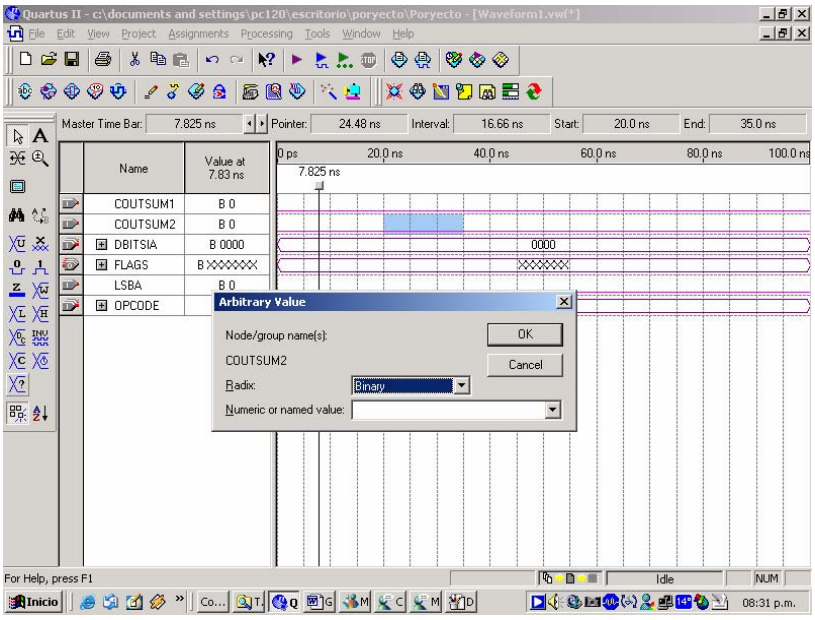


Figura 23.

- Para guardar el archivo, damos clic en el icono con la figura de diskette, acción que causa que aparezca el cuadro indicado en la Figura 24, en donde podemos buscar la locación donde se va a guardar el archivo de vector y su nombre, y, seleccionando la opción *"Add file to current proyect"* se adiciona el correspondiente archivo al proyecto.

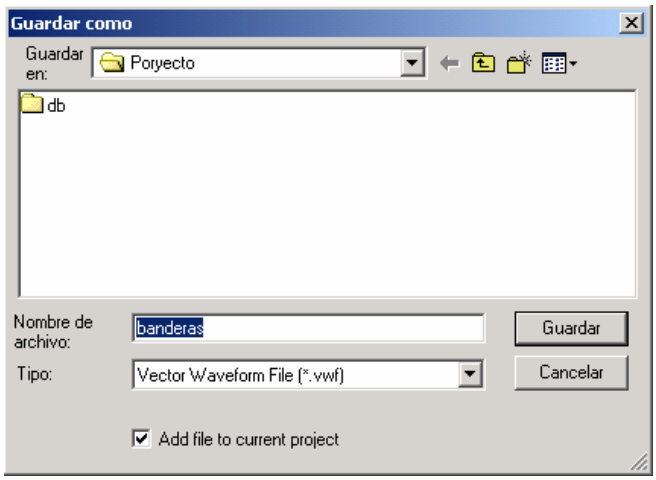


Figura 24.

- Ahora, para ingresar los parámetros de simulación e indicarle al software qué archivo utilizar para la simulación, damos clic en *"Assignments"* y seleccionamos la opción *"Simulator Settings Wizard"*, tal y como se muestra en la Figura 25.

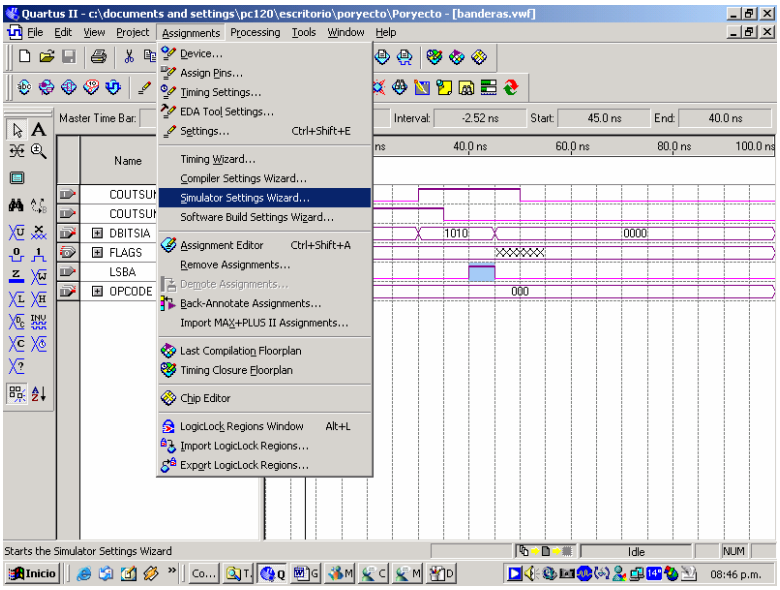


Figura 25.

- Posteriormente, se obtiene la ventana mostrada en la Figura 26, en donde simplemente se selecciona en el primer recuadro la entidad de nivel superior denominada *"Focus Point"*, y en el segundo recuadro podemos asignarle un nombre específico a la simulación que se va a realizar.

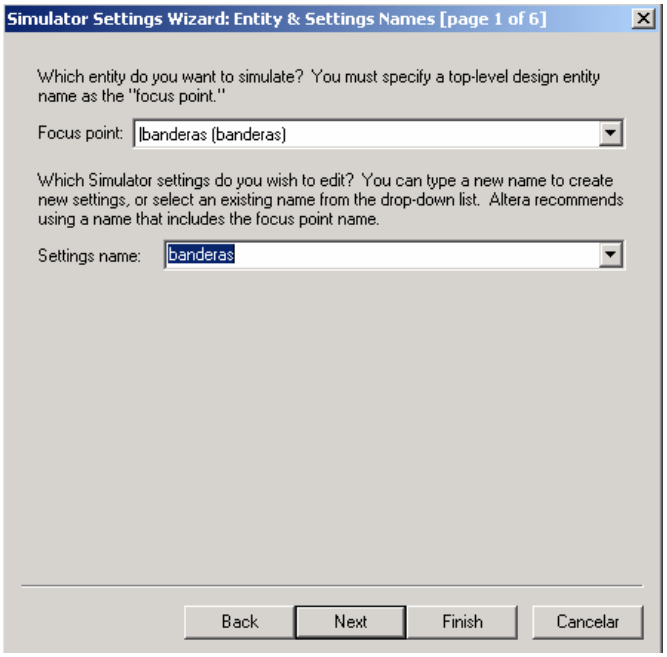


Figura 26.

- En la siguiente ventana podemos elegir entre una simulación de tipo funcional, la cual nos indicará en las variables de salida los valores correspondientes a los estímulos que son puestos en las variables de entrada; o una simulación temporizada, en la cual el simulador además de proporcionar las salidas a los vectores de entrada, también presenta una estimación de los tiempos de propagación de las señales a través del sistema. Esta ventana es la mostrada en la Figura 27.

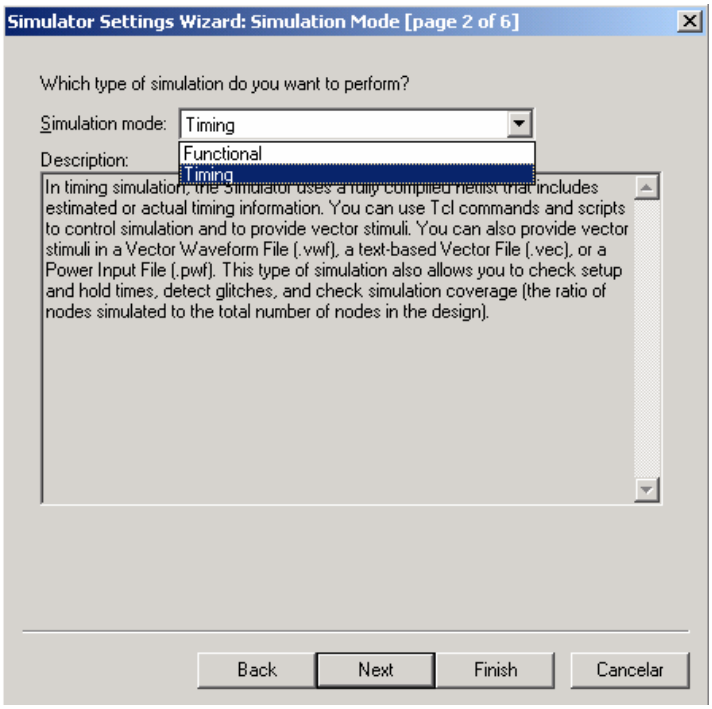


Figura 27.

- En el siguiente paso se le indica al software el archivo de donde se encuentran los vectores de estímulo del sistema (Vector Waveform File, .vwf que se creó en pasos anteriores), seleccionando en la primera opción *"Yes, use this file"*, y dando clic en el botón de los puntos suspensivos en donde se busca el correspondiente archivo a simular. Esta ventana es la mostrada en la Figura 28.

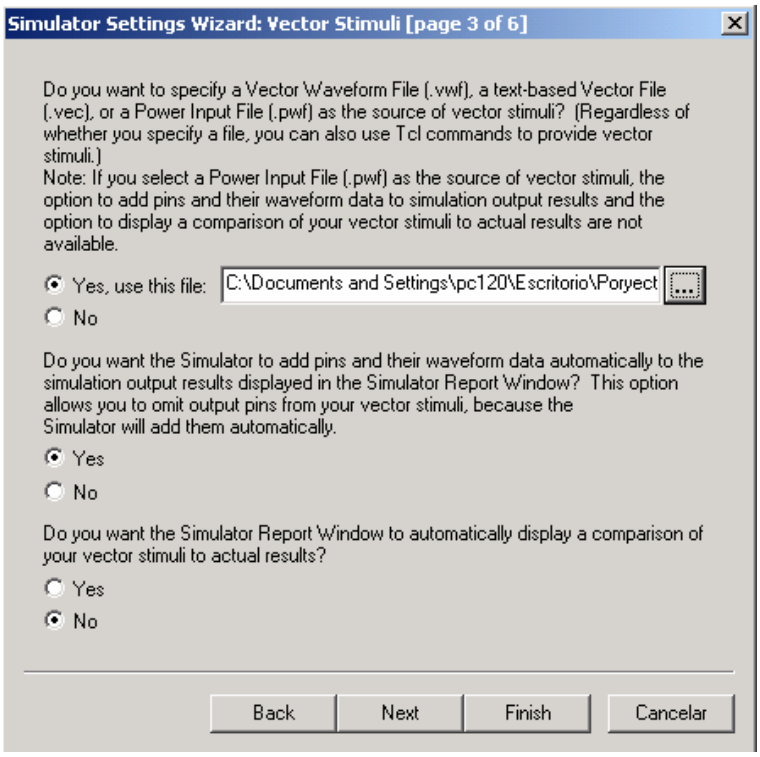


Figura 28.

- En la última ventana correspondiente a la Figura 29, se observa un resumen de los parámetros elegidos para la simulación.

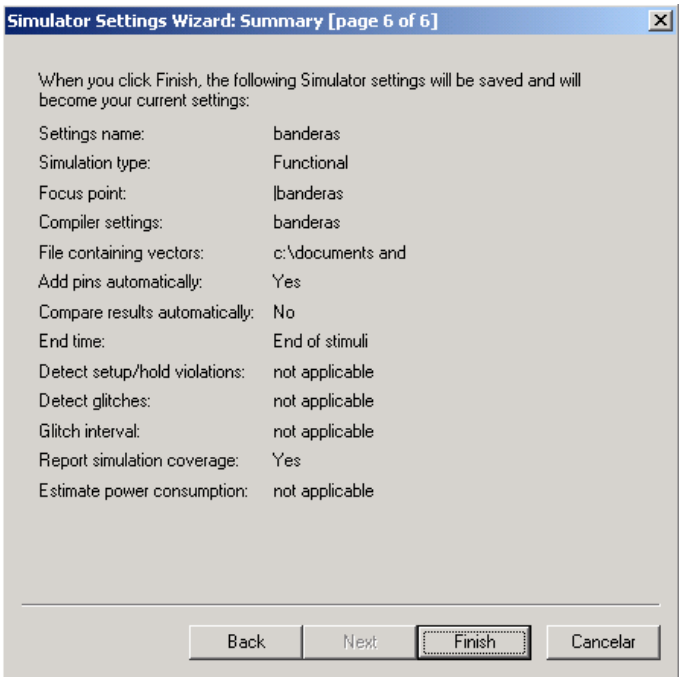


Figura 29.

- Por último, para realizar la simulación, se presiona el botón que tiene un triángulo azul con un "reloj digital" en el inferior. Al finalizar la simulación, se obtiene un reporte de esta, junto con las formas de onda obtenidas en una ventana aparte del archivo .vwf denominada "Simulation Waveforms".

NOTA: Para editar los parámetros seleccionados, se da clic en *"Assignments"* y posteriormente se selecciona la opción *"Settings"*, en donde se pueden editar las opciones que se desean para todo el proyecto.

CONFIGURACIÓN EN LOS DISPOSITIVOS DE LÓGICA PROGRAMABLE

Los esquemas de configuración que tiene el software son muy variados, van desde la simple utilización de un dispositivo que provee el fabricante (EPC) hasta la opción de utilizar microcontroladores para desempeñar esta función. Se debe tener en cuenta que no todos los esquemas de configuración están disponibles para todas las familias de dispositivos, es necesario entonces revisar los que son soportados para la familia utilizada, esta información se encuentra en las hojas de especificaciones de las familias de dispositivos que se pueden descargar de la página web de Altera (www.altera.com).

A continuación se explicará, por simplicidad, el esquema más sencillo que permite el fabricante, que es la utilización del software.

- Para configurar los proyectos que son realizados en el dispositivo de lógica programable por medio del puerto paralelo del computador, es necesario proporcionar el esquema de configuración que se va a utilizar, con el método *"Passive Parallel Asinchronous"*, se puede utilizar el protocolo JTAG que maneja la opción mencionada; esto se puede lograr haciendo clic en *"Assignments"* y posteriormente se selecciona la opción *"Settings"*, una vez allí, de la opción *"Compiler Settings",* elegimos el parámetro *"Device"*, del cual damos clic en *"Device & Pin Options"* lo cual despliega la ventana mostrada en la Figura 30, de donde, de la pestaña *"Programming Files"*, se selecciona la opción *"Passive Parallel Asinchronous"* para que el compilador genere el correspondiente archivo con el nombre del proyecto y sufijo .sof (*SRAM Object File*), el cuál será utilizado en el proceso de configuración del dispositivo de lógica programable.

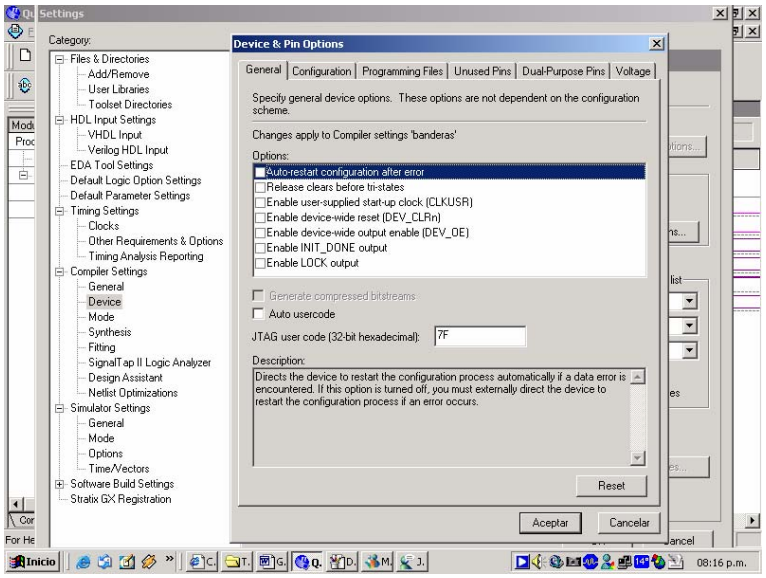


Figura 30.

- Una vez realizado el paso anterior, podemos iniciar el proceso de compilación; se selecciona la opción *"Programmer"* de la opción "Tools" de la barra de menú del software, tal y como es mostrado en la Figura 31.

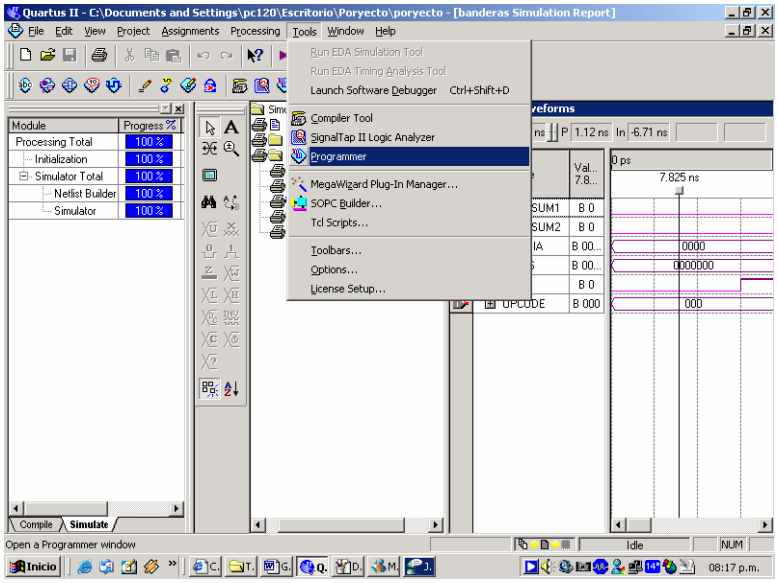


Figura 31.

- Posteriormente, es necesario seleccionar la opción JTAG del menú desplegable *"Mode"*, como se muestra en la Figura 32.

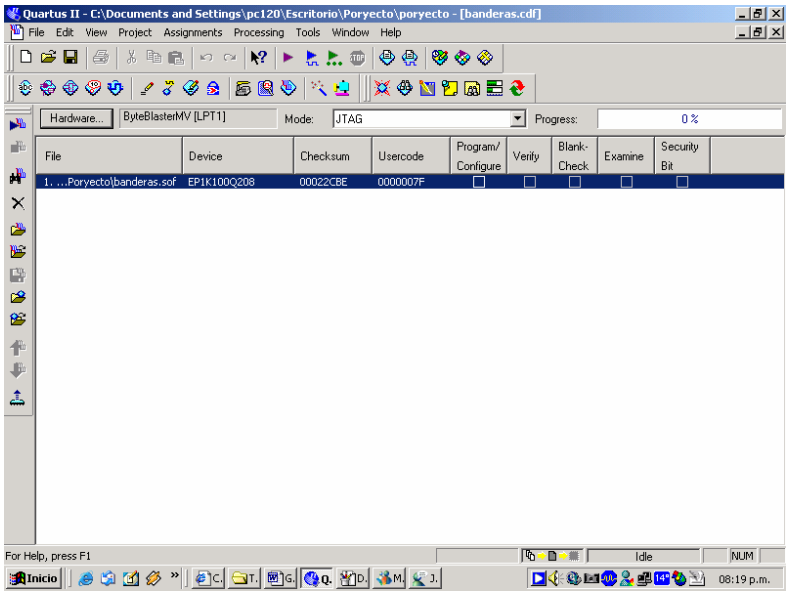


Figura 32.

- El siguiente paso a seguir es indicarle al software el puerto por medio del cual se realizará la tarea de configuración del proyecto en el dispositivo de lógica programable (este paso puede ser obviado si las indicaciones dadas en este paso ya habían sido realizadas), esto se realiza dando clic en el botón “Hardware” que se muestra en la Figura 32, lo que despliega la ventana mostrada en la Figura 33 en la cual adicionamos el hardware a utilizar, ByteBlaster(LPT1) para el puerto paralelo, esta operación debe ser realizada en la pestaña *“Hardware Settings”*.

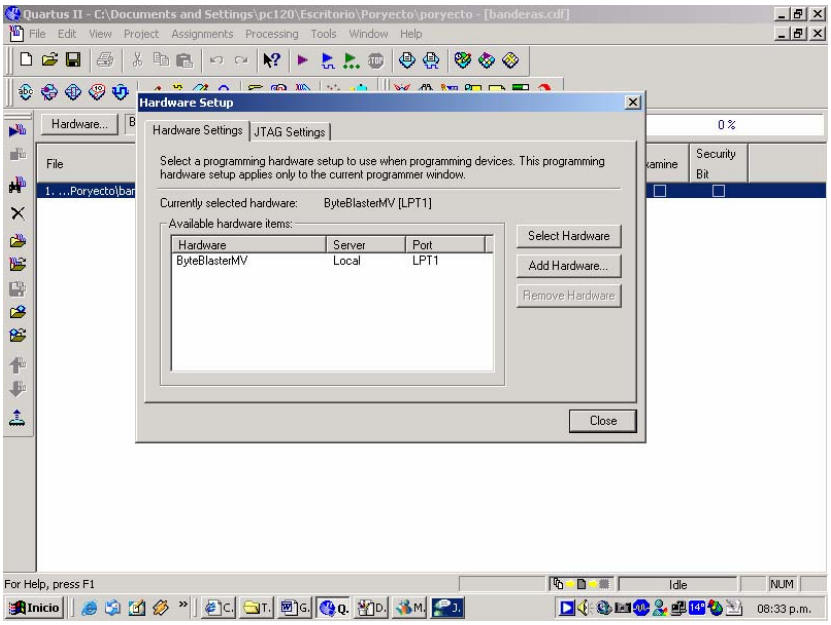


Figura 33.

- Ya habiendo seleccionado el puerto por medio del cual se realizará la programación del dispositivo, solamente nos queda indicarle al software el archivo que debe utilizar para realizar la configuración, esta operación se realiza dando clic en el botón mostrado en la Figura 34 que se encuentra en la parte izquierda de la pantalla actualmente activa, lo que despliega la ventana mostrada en la Figura 35, en donde se selecciona el archivo que tiene el nombre del proyecto junto con el sufijo.*sof* y se da clic en *"Aceptar"*.

Figura 34.

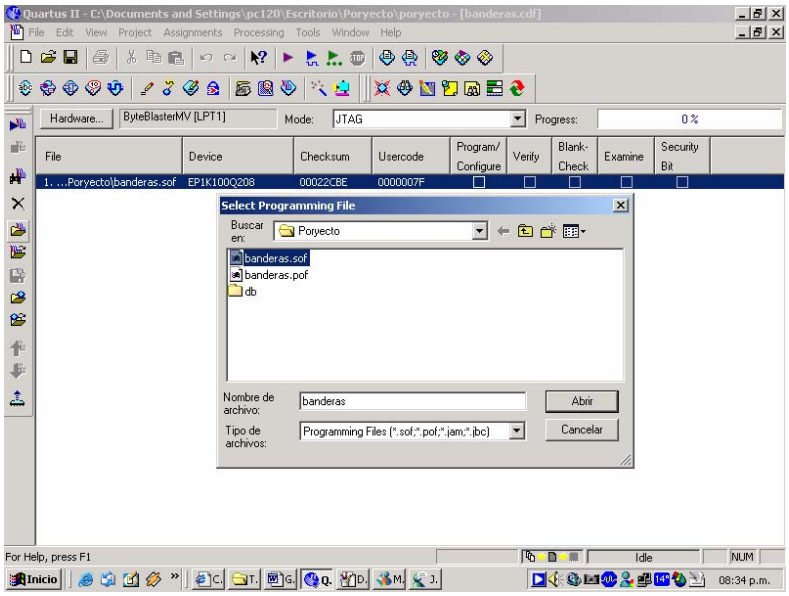


Figura 35.

- Por último, cerciorándonos previamente que el cable de programación "ByteBlaster" esté conectado y que el dispositivo este correctamente polarizado, damos clic en el "cuadrito" bajo el indicador marcado como *"Program/Configure"* para que aparezca marcado (tal y como se muestra en la Figura 36); y por último se da clic en el botón que se muestra en la Figura 37.

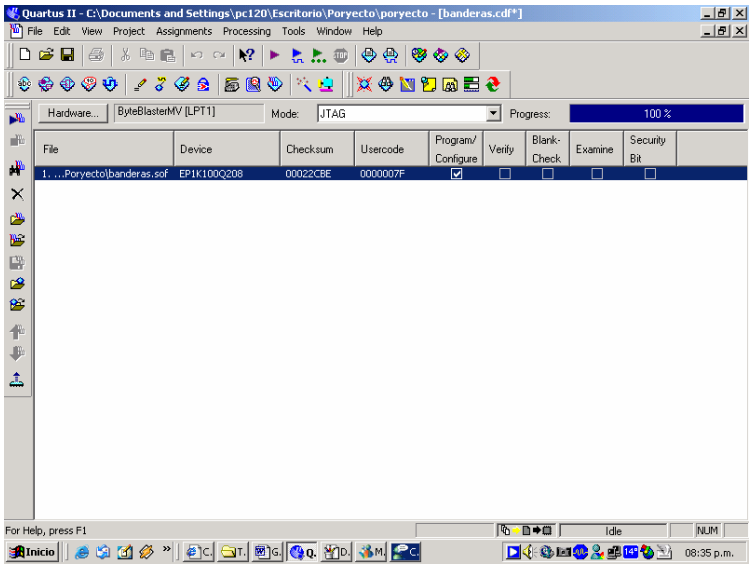


Figura 36.

Figura 37.

REALIZACIÓN DE CÓDIGO EN VHDL

Para la descripción de un diseño lógico en VHDL, se debe entender la sintaxis de este lenguaje. En el Ejemplo 1 se observa la forma de redacción.

En el encabezado, es necesario indicar las librerías que se van a utilizar, en el Ejemplo 1 se indica que se utilizará la librería IEEE, de la cual se utilizará el “package” std_logic_1164 en el cual están definidos los tipos de señales que son posible utilizar, estas son “0”, “1”, “X”, entre otras.

Ejemplo 1:

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;

ENTITY neg IS
	PORT(
		-- DATOS DE ENTRADA
		EN	: IN STD_LOGIC_VECTOR(13 downto 0);

		-- DATOS DE SALIDA
		SAL	: OUT STD_LOGIC_VECTOR(13 downto 0)
	);
END neg;

ARCHITECTURE aluneg OF neg IS
	BEGIN
		SAL(13 downto 0) <= NOT EN(13 downto 0);
```

```
END aluneg;
```

Las unidades de diseño en VHDL son principalmente “entidades” (ENTITY) en las cuales se les define un nombre (debe ser igual al nombre como se guarda el archivo), en estas entidades es necesario definir los parámetros de salida y de entrada, esto se realiza con el “*Keyword*” PORT, en donde se define el nombre de las señales, el tipo de señal y el sentido en el cual operan las señales (IN, OUT o BIDIR).

Posteriormente, se debe definir el cuerpo de la unidad de diseño o arquitectura tal y como se muestra en el Ejemplo 1. En esta es necesario darle un nombre y correlacionarlo con la entidad de diseño, posteriormente, se pueden definir variables, señales y/o constantes que pueden ser necesarias en el diseño propuesto, y posteriormente, se realiza la descripción de las funciones que desarrolla la entidad posterior al “*Keyword*” BEGIN.

Como en cualquier lenguaje de programación, VHDL permite una sintaxis especial para desarrollar un modelo jerárquico en el diseño del proyecto, opción que permite definir claramente funciones de cada archivo de descripción de hardware, y otorga una organización del sistema global. Esto es posible por medio de la sintaxis del Ejemplo 2.

Ejemplo 2:

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;
USE work.neg.ALL;

ENTITY impreso IS
       PORT(

              RA            : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
              RC            : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
       );
END impreso;

ARCHITECTURE temp OF impreso IS

 COMPONENT puerto_negador
  PORT(
       EN     : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
       SAL    : OUT STD_LOGIC_VECTOR(13 DOWNTO  0) );
 END COMPONENT;

 FOR ALL       : puerto_negador USE ENTITY work.neg;

 BEGIN

  NEGA1        : puerto_negador
 PORT MAP(EN(13 DOWNTO 0)=>RB(13 DOWNTO 0), SAL(13 DOWNTO 0)=>RC(13 DOWNTO 0) );

END temp;
```

Tomando como unidad jerárquicamente inferior la declarada en el Ejemplo 1, se llamará en la entidad del Ejemplo 2 que será considerada como la jerárquicamente de orden superior.

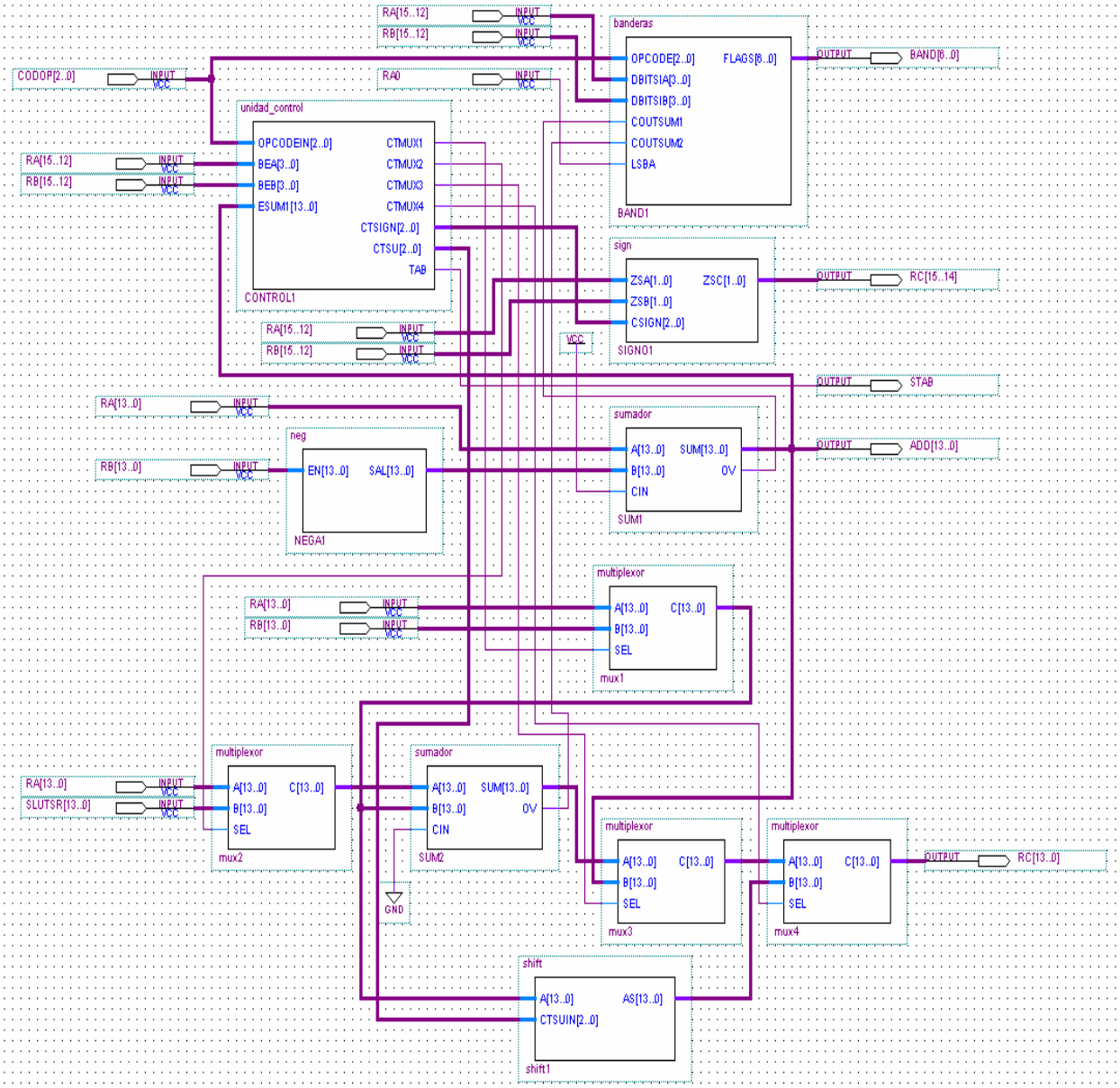
En primera instancia, es necesario hacer el llamado a la entidad de orden inferior, y esto se logra definiendo por medio de la instancia: USE work.neg.ALL; en la cual se indica que la unidad se encuentra alojada en el directorio de trabajo (work), y que se utilizarán todos sus parámetros (entrada y salida).

Posteriormente, (en la arquitectura del sistema) se define un componente (COMPONENT), en el cual se le indica un nombre cualquiera. Este componente, realizando una analogía, es como el socket en un circuito impreso que conecta el “chip” (entidad neg en este caso) con el circuito impreso (impreso, en el ejemplo). Hay que tener en cuenta que el puerto de este componente se deben definir las señales tal y como se definieron en el puerto de la unidad jerárquica de nivel inferior (en este caso *neg*).

Adicionalmente se debe correlacionar la entidad jerárquica superior con la inferior, esto se hace por medio de la línea FOR ALL : puerto_negador USE ENTITY work.neg;.

En este lenguaje se puede definir un “label” para el componente que se desea utilizar, el cual nos permite tener un orden en el diseño, e diferenciar varias unidades del mismo tipo (si existe más de una). Por último se realiza el mapeo de puertos, que debe ser realizado de la forma en que se muestra el ejemplo, de izquierda a derecha y en el orden indicado.

ANEXO G DIAGRAMA EN BLOQUES DE “ALULOGARITMICA.VHD”



ANEXO H TABLA DE TIEMPOS DE PROPAGACIÓN DE TERMINAL A TERMINAL

Fuente	Destino	Tiempo de retardo	Fuente	Destino	Tiempo de retardo	Fuente	Destino	Tiempo de retardo	Fuente	Destino	Tiempo de retardo
RB[9]	BAND[0]	109.300 ns	RB[4]	BAND[0]	96.500 ns	RA[7]	RC[9]	92.700 ns	RA[7]	RC[7]	89.100 ns
RA[9]	BAND[0]	108.200 ns	RB[2]	RC[12]	96.300 ns	RB[7]	RC[9]	92.700 ns	RB[7]	RC[7]	89.100 ns
RB[0]	BAND[0]	106.800 ns	RA[8]	RC[12]	96.200 ns	RB[3]	RC[11]	92.700 ns	RA[8]	RC[6]	89.000 ns
RA[1]	BAND[0]	106.600 ns	RA[9]	RC[11]	96.100 ns	RB[0]	RC[7]	92.200 ns	RA[5]	RC[12]	88.700 ns
RB[8]	BAND[0]	106.500 ns	RA[4]	BAND[0]	96.000 ns	RA[1]	RC[7]	92.000 ns	RB[0]	RC[5]	88.400 ns
RA[3]	BAND[0]	106.400 ns	RB[0]	RC[9]	95.800 ns	RA[7]	RC[8]	92.000 ns	RB[3]	RC[10]	88.400 ns
RA[0]	BAND[0]	106.400 ns	RA[1]	RC[9]	95.600 ns	RB[7]	RC[8]	92.000 ns	RA[1]	RC[5]	88.200 ns
RB[6]	BAND[0]	106.300 ns	RB[8]	RC[9]	95.500 ns	RB[4]	RC[13]	92.000 ns	RB[3]	RC[6]	88.200 ns
RB[1]	BAND[0]	106.300 ns	RA[3]	RC[9]	95.400 ns	RB[8]	RC[7]	91.900 ns	RB[8]	RC[5]	88.100 ns
RA[6]	BAND[0]	106.200 ns	RA[0]	RC[9]	95.400 ns	RA[3]	RC[7]	91.800 ns	RB[5]	RC[12]	88.100 ns
RA[2]	BAND[0]	105.900 ns	RB[3]	RC[12]	95.400 ns	RA[0]	RC[7]	91.800 ns	RB[12]	BAND[0]	88.100 ns
RB[2]	BAND[0]	105.700 ns	RB[6]	RC[9]	95.300 ns	RA[9]	RC[10]	91.800 ns	RA[3]	RC[5]	88.000 ns
RA[8]	BAND[0]	105.600 ns	RB[1]	RC[9]	95.300 ns	RB[6]	RC[7]	91.700 ns	RA[0]	RC[5]	88.000 ns
RB[9]	RC[13]	104.800 ns	RA[10]	BAND[0]	95.300 ns	RB[1]	RC[7]	91.700 ns	RB[6]	RC[5]	87.900 ns
RB[3]	BAND[0]	104.800 ns	RA[6]	RC[9]	95.200 ns	RA[9]	RC[6]	91.600 ns	RB[1]	RC[5]	87.900 ns
RA[9]	RC[13]	103.700 ns	RB[0]	RC[8]	95.100 ns	RA[6]	RC[7]	91.600 ns	RA[6]	RC[5]	87.800 ns
RA[7]	BAND[0]	103.700 ns	RA[1]	RC[8]	94.900 ns	RA[7]	RC[11]	91.600 ns	RA[2]	RC[5]	87.500 ns
RB[7]	BAND[0]	103.700 ns	RA[2]	RC[9]	94.900 ns	RB[7]	RC[11]	91.600 ns	RA[12]	BAND[0]	87.500 ns
RB[0]	RC[13]	102.300 ns	RB[8]	RC[8]	94.800 ns	RA[4]	RC[13]	91.500 ns	RB[2]	RC[5]	87.300 ns
RA[1]	RC[13]	102.100 ns	RB[9]	RC[7]	94.700 ns	RA[2]	RC[7]	91.300 ns	RA[7]	RC[10]	87.300 ns
RB[8]	RC[13]	102.000 ns	RA[3]	RC[8]	94.700 ns	RB[11]	BAND[0]	91.300 ns	RB[7]	RC[10]	87.300 ns
RA[3]	RC[13]	101.900 ns	RA[0]	RC[8]	94.700 ns	RB[2]	RC[7]	91.100 ns	RA[8]	RC[5]	87.200 ns
RA[0]	RC[13]	101.900 ns	RB[2]	RC[9]	94.700 ns	RA[8]	RC[7]	91.000 ns	RA[7]	RC[6]	87.100 ns
RB[6]	RC[13]	101.800 ns	RB[0]	RC[11]	94.700 ns	RB[9]	RC[5]	90.900 ns	RB[7]	RC[6]	87.100 ns
RB[1]	RC[13]	101.800 ns	RB[6]	RC[8]	94.600 ns	RA[10]	RC[13]	90.800 ns	RA[5]	RC[9]	87.100 ns
RA[6]	RC[13]	101.700 ns	RB[1]	RC[8]	94.600 ns	RB[10]	BAND[0]	90.600 ns	RB[4]	RC[12]	87.100 ns
RA[2]	RC[13]	101.400 ns	RA[8]	RC[9]	94.600 ns	RB[0]	RC[10]	90.400 ns	RB[15]	BAND[0]	87.100 ns
RB[2]	RC[13]	101.200 ns	RA[6]	RC[8]	94.500 ns	RB[0]	RC[6]	90.200 ns	RB[14]	BAND[0]	87.000 ns
RA[8]	RC[13]	101.100 ns	RA[1]	RC[11]	94.500 ns	RB[3]	RC[7]	90.200 ns	RB[11]	RC[13]	86.800 ns
RB[3]	RC[13]	100.300 ns	RB[8]	RC[11]	94.400 ns	RA[1]	RC[10]	90.200 ns	RA[4]	RC[12]	86.600 ns
RB[9]	RC[12]	99.900 ns	RA[3]	RC[11]	94.300 ns	RB[8]	RC[10]	90.100 ns	RB[5]	RC[9]	86.500 ns
RA[7]	RC[13]	99.200 ns	RA[0]	RC[11]	94.300 ns	RA[1]	RC[6]	90.000 ns	RB[3]	RC[5]	86.400 ns
RB[7]	RC[13]	99.200 ns	RA[7]	RC[12]	94.300 ns	RA[3]	RC[10]	90.000 ns	RA[5]	RC[8]	86.400 ns
RA[9]	RC[12]	98.800 ns	RB[7]	RC[12]	94.300 ns	RA[0]	RC[10]	90.000 ns	RB[10]	RC[13]	86.100 ns
RB[9]	RC[9]	98.300 ns	RA[2]	RC[8]	94.200 ns	RA[11]	BAND[0]	90.000 ns	RA[5]	RC[11]	86.000 ns
RA[5]	BAND[0]	98.100 ns	RB[6]	RC[11]	94.200 ns	RB[8]	RC[6]	89.900 ns	RA[10]	RC[12]	85.900 ns
RB[9]	RC[8]	97.600 ns	RB[1]	RC[11]	94.200 ns	RB[6]	RC[10]	89.900 ns	RB[5]	RC[8]	85.800 ns
RB[5]	BAND[0]	97.500 ns	RA[6]	RC[11]	94.100 ns	RB[1]	RC[10]	89.900 ns	RB[4]	RC[9]	85.500 ns
RB[0]	RC[12]	97.400 ns	RB[2]	RC[8]	94.000 ns	RA[9]	RC[5]	89.800 ns	RA[11]	RC[13]	85.500 ns
RA[9]	RC[9]	97.200 ns	RA[8]	RC[8]	93.900 ns	RA[3]	RC[6]	89.800 ns	RB[13]	BAND[0]	85.500 ns
RB[9]	RC[11]	97.200 ns	RB[3]	RC[9]	93.800 ns	RA[0]	RC[6]	89.800 ns	RA[13]	BAND[0]	85.500 ns
RA[1]	RC[12]	97.200 ns	RA[2]	RC[11]	93.800 ns	RA[6]	RC[10]	89.800 ns	RB[5]	RC[11]	85.400 ns
RB[8]	RC[12]	97.100 ns	RA[9]	RC[7]	93.600 ns	RB[6]	RC[6]	89.700 ns	RB[9]	RC[4]	85.300 ns
RA[3]	RC[12]	97.000 ns	RB[2]	RC[11]	93.600 ns	RB[1]	RC[6]	89.700 ns	RA[7]	RC[5]	85.300 ns
RA[0]	RC[12]	97.000 ns	RA[5]	RC[13]	93.600 ns	RA[6]	RC[6]	89.600 ns	RB[7]	RC[5]	85.300 ns
RB[6]	RC[12]	96.900 ns	RA[8]	RC[11]	93.500 ns	RA[2]	RC[10]	89.500 ns	RA[4]	RC[9]	85.000 ns
RB[1]	RC[12]	96.900 ns	RB[3]	RC[8]	93.100 ns	RA[2]	RC[6]	89.300 ns	RB[4]	RC[8]	84.800 ns
RA[6]	RC[12]	96.800 ns	RB[5]	RC[13]	93.000 ns	RB[2]	RC[10]	89.300 ns	RA[14]	BAND[0]	84.800 ns
RA[9]	RC[8]	96.500 ns	RB[9]	RC[10]	92.900 ns	RA[8]	RC[10]	89.200 ns	RB[4]	RC[11]	84.400 ns
RA[2]	RC[12]	96.500 ns	RB[9]	RC[6]	92.700 ns	RB[2]	RC[6]	89.100 ns	RA[4]	RC[8]	84.300 ns

ANEXO I GUÍA DE USUARIO “ALU LOGARÍTMICA”

Para hacer uso de la Unidad Aritmético Lógica es necesario instalar la aplicación de JAVA que nos permite utilizar el entorno de operación creado para el sistema, realizar la configuración del dispositivo de lógica programable e interconectar las tarjetas elaboradas; estas operaciones son explicadas a continuación.

1. INSTALACIÓN DE SOFTWARE

Para el uso de la aplicación “Controlador” es necesario instalar un ambiente de desarrrollo de **JAVA**, el ambiente de desarrollo mas reciente es el **Java Developement Kit 1.5**, que se puede encontrar en la pagina de Sun Microsystems www.sun.com. Esta herramienta se utiliza desde el comando de DOS.

2. INSTALACIÓN DE API COMM

API es el acrónimo de **Application Programming Interface**, y son librerías de funciones que sirven como extensiones del ambiente JAVA. En este caso es necesario utilizar una extensión llamada **Communication API**, que contiene las definiciones de funciones para trabajar con el puerto serial RS-232 de un computador. Esta extensión también se puede encontrar en la dirección de Internet www.sun.com.

La instalación de este **API** se realiza después de la instalación del **Java Developement Kit**, y consiste en copiar y pegar ciertos archivos en la carpeta donde quedo instalado el JDK. Los archivos que se deben copiar y pegar son los siguientes:

win32com.dll en el directorio <JDK>\bin.

comm.jar en el directorio <JDK>\lib.

javax.comm.properties en el directorio <JDK>\lib.

Para que la extensión **Communication API** sea reconocida por el ambiente de desarrollo, es necesario que el archivo comm.jar sea incluído en el **CLASSPATH** de JAVA, para esto, en el comando de DOS se teclea la siguiente instrucción:

```
C:\>set CLASSPATH=c:\<JDK>\lib\comm.jar;%classpath%
```

Si el compilador javac.exe o el depurador java.exe no reconocen los archivos que se deben compilar o correr, se debe tener cuidado que no se halla alterado el **CLASSPATH** original, si esto se presentó se puede escribir el que se muestra a continuación, de tal forma que el **CLASSPATH** incluya la carpeta donde se encuentra la extensión **Communication API** y la carpeta donde se encuentra la aplicación “controlador.class”.

```
C:\>set CLASSPATH=c:\<JDK>\lib\comm.jar;c:\<JDK>\bin\
```

3. COMPILACIÓN Y EJECUCIÓN DEL ARCHIVO CONTROLADOR.JAVA

El archivo “controlador.java”, se copia y pega en la carpeta `c:\<JDK>\bin\`

Se debe posicionar el comando de DOS en la carpeta c:\<JDK>\bin, en esta carpeta se encuentran las utilidades javac.exe y java.exe, como también las demás herramientas de JDK para el desarrollo de aplicaciones JAVA.

Los códigos fuente de JAVA se guardan con la extensión .java, el archivo fuente del trabajo de grado se llama “controlador.java”. La utilidad javac.exe se encarga de compilar estos archivos fuente, y crea uno o varios archivos con extensión .class, los cuales conforman objetos que representan la aplicación en JAVA. Con la siguiente instrucción desde DOS se puede compilar la aplicación controlador:

```
C:\<JDK>\bin\>javac controlador.java
```

Una vez compilado el código “controlador.java”, son generados dos archivos de extensión .class, “controlador.class” y “controlador$rutina.class”.

Para ejecutar el objeto resultante, se utiliza la aplicación java.exe. Aunque es necesario hacer uso de estas herramientas desde DOS, el objeto “controlador.class” es ejecutado en una ventana de Windows. La instrucción necesaria para realizar esta función se muestra a continuación:

```
C:\<JDK>\bin\>java controlador
```

4. CREACIÓN DE UN ARCHIVO BATCH

Para crear un archivo que permita la ejecución de “Controlador.java” sin necesidad de teclear instrucciones desde el comando de DOS, se puede crear un archivo BATCH con las instrucciones anteriores, para ello se abre un nuevo archivo de texto, se escribe las líneas de comando que se muestran a continuación y se guarda este archivo con la extensión .bat; no es necesario que este archivo sea guardado en alguna carpeta especial o que tenga un nombre específico.

```
@ECHO OFF
set CLASSPATH=.;c:\jdk15\bin\;c:\jdk15\lib\comm.jar
cd\
cd <jdk>
cd bin
java controlador
```

5. CONFIGURACIÓN EN LA TARJETA DE DESARROLLO

Para la configuración del dispositivo FPGA de Altera se hace uso de la aplicación Quartus II, esta aplicación permite compilar código en VHDL, simular sistemas digitales, programar y configurar dispositivos lógicos de Altera.

Es necesario que este programa esté instalado para configurar el dispositivo por medio del cable ByteBlasterMV, este cable se conecta al puerto paralelo del PC y va directo al puerto JTAG de la tarjeta de desarrollo que contiene el FPGA.

Para la configuración en el FPGA, es necesario tener ubicado el archivo *“pruebaalu.sof”*, en el cual se encuentra compilado el trabajo y también se tienen los parámetros de asignación de pines para la tarjeta de desarrollo utilizada. Para esto, es necesario abrir Quartus II y ejecutar los pasos indicados en la sección “Configuración en los dispositivos de lógica programable” del ANEXO F y utilizar el archivo .sof anteriormente mencionado, adicionalmente es necesario haber polarizado previamente la tarjeta de desarrollo que contiene el FPGA.

6. INTERCONEXIÓN DEL HARDWARE

El Hardware de este sistema esta constituido por tres tarjetas. La primera es una tarjeta de desarrollo que contiene un FPGA ACEX 1K100-208 de Altera, la segunda es una tarjeta controladora la cual contiene un microcontrolador PIC 18F8720 de Microchip. La tercera tarjeta

contiene dos memorias EPROM 27C256. La interconexión de las tres tarjetas se observa en la figura 1.

Las tres tarjetas son alimentadas con un voltaje de 5 voltios. La tarjeta controladora va conectada al puerto Serial COM1 del PC donde la aplicación “controlador.class” fue instalada.

La tarjeta de desarrollo es conectada a un PC por medio de un cable de configuración ByteBlasterMV este se conecta al puerto Paralelo y al Puerto JTAG de la tarjeta de desarrollo.

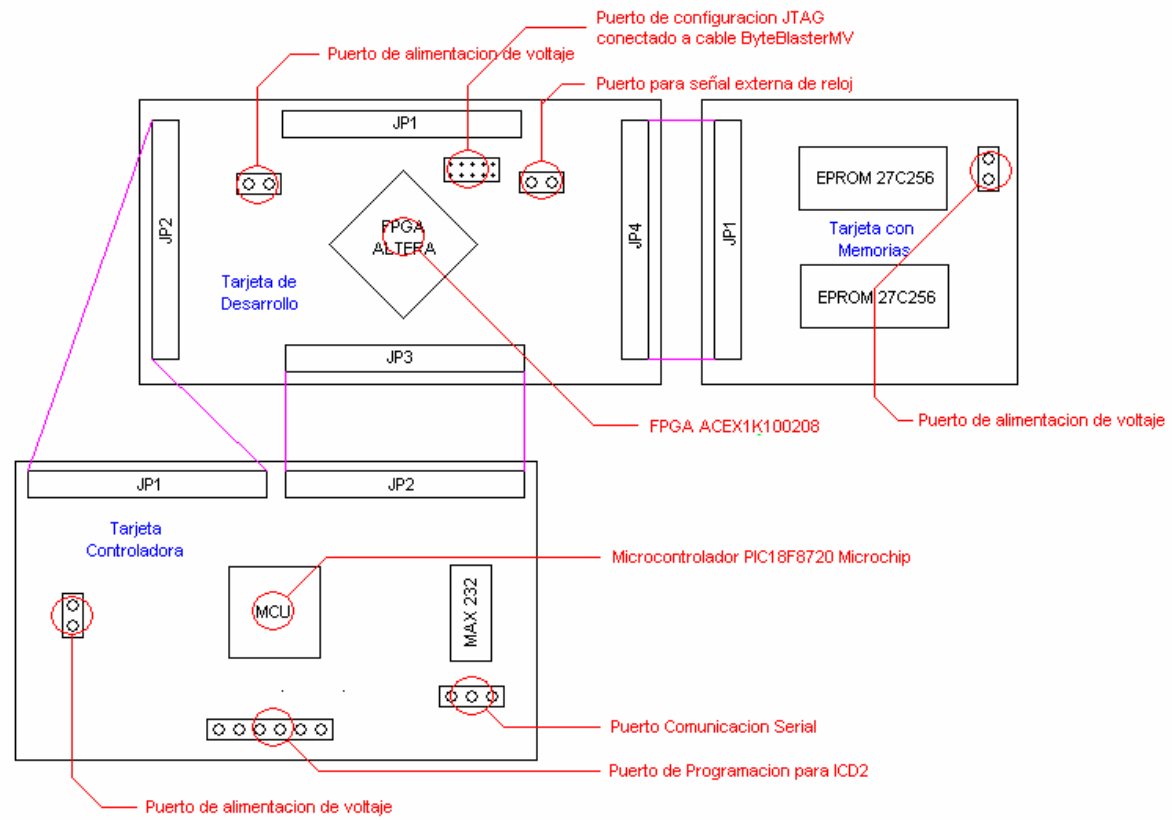


Figura 1. Interconexión del sistema.

Para la conexión entre la tarjeta de desarrollo y la tarjeta controladora, se utilizan dos cintas de 40 líneas cada una con una construcción especial dado que inicialmente se había pensado conectar el puerto uno de la tarjeta controladora con el puerto uno de la tarjeta de desarrollo, y posteriormente fue necesaria hacer un arreglo a las cintas para que el puerto uno de la tarjeta controladora se pudiera conectar con el puerto 2 de la tarjeta de desarrollo.

7. ENTORNO DE LA APLICACIÓN

En la figura 2 se observa el entorno de la aplicación “controlador.class”, en este sencillo entorno se observa los campos dato 1 y dato 2 donde se ingresan las cantidades a operar, posteriormente se elije la operación deseada a partir de las opciones localizadas a la izquierda de la ventana, el resultado de la operación realizada por la ALU logarítmica es mostrado en el campo “resultado ALU”, y el resultado obtenido por el CPU es mostrado en el campo “resultado CPU”. El campo “Error” registra la diferencia entre el resultado de la ALU logarítmica y el CPU.

Existen además los campos “Dato 1 RA”, “Dato 2 RB” y “Dato resultado RC” en donde se observan en formato binario los datos; adicionalmente también se observan los bits de banderas y el código de operación que le ingresan y/o salen de la ALU logaritmica. Los bits de los campos “Dato 1 RA” y “Dato 2 RB” también pueden ser manipulados en formato binario, para ingresar un número específico.

Al lado derecho de la aplicación se encuentra registradas las banderas de la última operación realizada.

Por último, cabe notar que es necesario tener datos en los dos campos para seleccionar con el ratón la operación que se desea realizar, a excepción de las operaciones de potencia y de raíz que toman solamente el dato ingresado en el campo “Dato 1” o en “Dato 1 RA” que es la representación binaria del anteriormente mencionado.

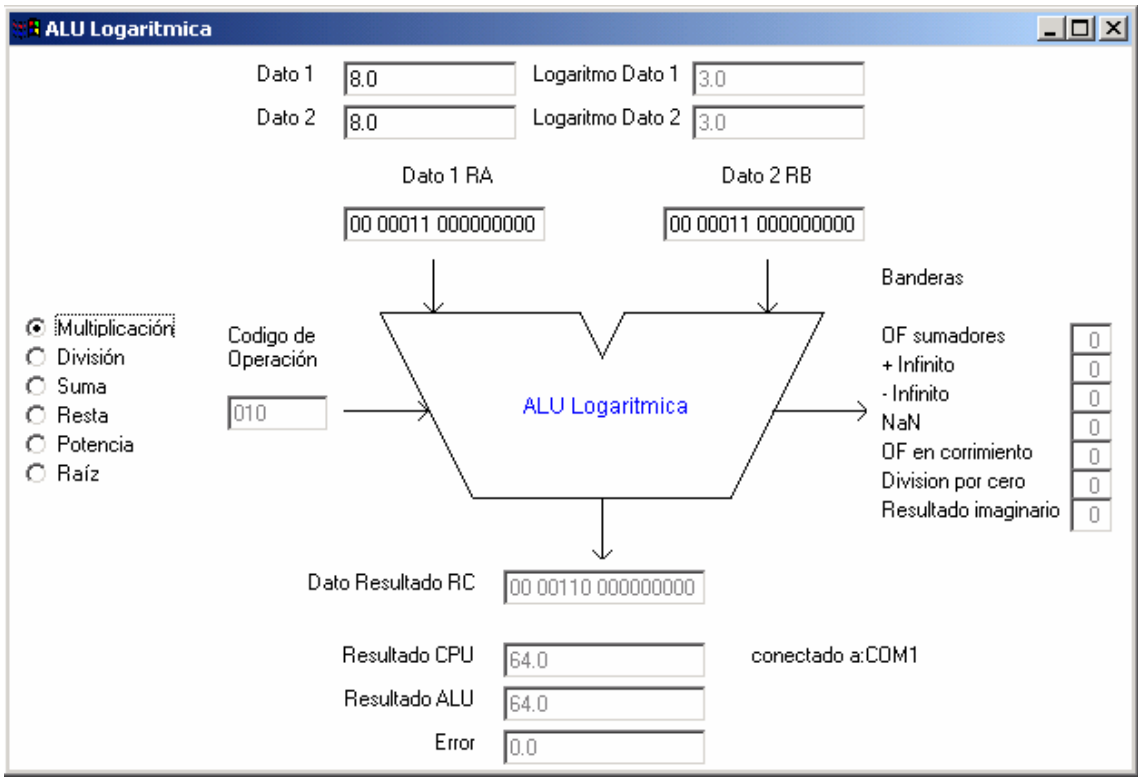


Figura 2. Entorno de la aplicación controlador.class