[ACERCA DE ESTE LIBRO 2](#_Toc327364503)

[INTRODUCCIÓN 4](#_Toc327364504)

[I. UNIDAD ARITMÉTICA VECTORIAL: RAYTRAC. 7](#_Toc327364505)

[II. BLOQUES DE MEMORIA DE ENTRADA Y SALIDA. 13](#_Toc327364506)

[III. PIPELINE Y ARQUITECTURA DE OPERACIONES ARITMÉTICAS. 18](#_Toc327364507)

[*IV.* CONTROLADOR DE CAMINO DE DATOS Y SINCRONIZACIÓN. **¡Error! Marcador no definido.**](#_Toc327364508)

[*V.* Interfase de Programación y Sistema de Interrupciones. 39](#_Toc327364509)

[VI. SIMULACIÓN, VERIFICACIÓN Y VALIDACIÓN FUNCIONAL Y ANÁLISIS DE TIEMPOS 49](#_Toc327364510)

ACERCA DE ESTE LIBRO

INTRODUCCIÓN.

***Objetivo General***

***Objetivos Específicos***

***Background, Herramientas y Plataformas Tecnológicas para el desarrollo del sistema.***

***Background***

* ***Trazado de Rayos***
* Reflexión Refracción Absorción.
* Algoritmo de Trazado de Rayos.
* Trazado de Rayos vs. Rastering
* ***Arquitectura***
* El modelo de programación del RayTrac es SIMD, cifras en inglés de *Single Instruction Multiple Data.*
* El RayTrac es un *Streaming Processor[[1]](#footnote-1),* el modelo de programación impone restricciones, pero estas restricciones también ofrece ventajas. Localidad,
* Cómo el Paralelismo ofrecido por la arquitectura favorece al trazado de rayos y cómo el modelo de programación favorece la portabilidad entre sistemas.
* Kernels y flujos. Modelo de algoritmo de raytracing sencillo basado en kernels y flujos.

PRIMERA PARTE: RAYTRAC

***Capítulo Primero***, se realiza una descripción general del RayTrac y los elementos que la componen y la manera como interactúan. Se muestra un diagrama de bloques. Por último se lista las características funcionales de la unidad.

***Capítulo Segundo***, se encuentra la descripción de todos los elementos compuestos por memoria: la cola de instrucciones de entrada, el bloque de registros de operandos de entrada y la cola de resultados de salida. Se detalla los mecanismos de escritura y lectura de cada uno de estos elementos y los mecanismos de sincronización que los componen.

***Capítulo Tercero***, se encuentran detallados los aspectos de la arquitectura del *pipeline* aritmético. Los detalles del diseño desarrollo e implementación por tipo de operación, se encuentran descritos en esta sección.

***Capítulo Quinto(5)*** se explica en detalle el control de flujo de datos, el cual es controlado por una *máquina de estados* *para el control de flujo,* apoyados en la *interconexión decodificada de circuitos aritméticos* y *el sistema de control del flujos de entrada y salida*.

***Capítulo Cuarto (4)***, describe la interfase de programación implementada para realizar aplicaciones con el Rt Engine. La interfaz de programación tiene dos aspectos principales: la configuración de una función especializada o *kernel* y la programación de las fuentes de flujo de entrada, la cantidad de datos que vienen en este flujo y la configuración del destino de los resultados que van en los flujos de salida*.*

***Capítulo Sexto***, Pruebas hechas en simulación. Para realizar la simulación del RayTrac y su pipeline aritmético, se desarrolló un par de scripts que permiten la observación de ciertas señales, que se encuentran en los diseños RTL de jerarquía más bajo y automatizan el proceso de despliegue de resultados en archivos tipo *.csv* , o cómo lo indican sus siglas en inglés, archivos separado por comas. El objetivo de estas herramientas es simular una secuencia de instrucciones enviadas al RayTrac, emulando ***funcionalmente*** el comportamiento del bus externo (Avalon) y verificar los resultados.

Adicionalmente se realiza la simulación de tiempos, la cual es importante para saber si la segmentación de los circuitos combinatorios que componen el RayTrac es suficiente y soportan la frecuencia de reloj que se planea implementar con estos.

SEGUNDA PARTE: RTENGINE

***Capítulo Séptimo*** Descripción Gral del RtEngine. Diagrama de Bloques. Ejemplos de posibles aplicaciones. Descripción breve de la aplicación a implementar.

***Capítulo Octavo*** Integración e Implementación del Sistema. Se detalla el procedimiento de integración de los componentes del RtEngine, para obtener el sistema final integrado, haciendo uso de las herramientas Quartus II, Qsys y otras.

Se realiza un resumen a partir de la información generada en los reportes sobre los recursos utilizados, la organización del pool de memoria, los dominios de reloj, etc, etc.

Las herramientas por último generan la capa de abstracción de hardware, mediante la generación de código de alto nivel C. Esta capa provee el *kernel* o sistema operativo del sistema, los *drivers* de los componentes y una *API (Aplication Programming Interfaz)* para el desarrollo de aplicaciones. A este software se le llama *BSP*, (*Board Support Package*) y para los sistemas embebidos constituye el componente fundamental para el desarrollo de aplicaciones.

***Capítulo Noveno***, describe la implementación del algoritmo de trazado de rayos, el algoritmo de *Shading* (cálculo de color en cada pixel) y en general la aplicación de *render*.

***Capítulo Decimo,*** Describe las aplicaciones externas realizadas. Para la comunicación entre la tarjeta y el computador, se necesita realizar una aplicación host que se comunique con esta. Adicionalmente se describen las herramientas que sirven de interfase para configurar y diseñar las escenas que se quieren renderizar.

TERCERA PARTE: CIERRE

***Capítulo Undécimo***, Se detallan resultados generales y se presentan las conclusiones del trabajo en general. Se enuncian posibles trabajos, por hacer a partir de este, entre los que se encuentran mejoras, aplicaciones y correcciones.

INTRODUCCIÓN

*Objetivo General.*

Diseñar e implementar una arquitectura de trazado de rayos para síntesis de imágenes.

*Objetivos Específicos.*

* Diseñar, simular e implementar los circuitos aritméticos que soporten la resolución geométrica de un sistema de trazado de rayos.
* Implementar una interfase operativa, humano – máquina, mediante codiseño y herramientas propietarias.
* Probar la implementación de la arquitectura de trazado de rayos, a partir de software.

*Herramientas y Plataformas Tecnológicas para el desarrollo del sistema.*

Software.

* Quartus II. Suite de Diseño, Simulación y Programación de Altera. Con esta herramienta se compilan y verifican los diseños producto de este trabajo. Adicionalmente con esta herramienta se integran otras *IP* a el sistema RayTrac: el procesador NIOS II, el sistema de manejo de memorias externas, el TCP/IP NicheStack, el controlador de pantalla VGA y el bus Avalon, todo esto con la herramienta QSYS. Adicionalmente se verifica la idoneidad de la frecuencia de reloj a la que opera el sistema con la herramienta TimeQuest Analyzer.

* ModelSim ASE. Software de Simulación de Mentor Graphics. El propósito de esta herramienta es simular funcionalmente el diseño resultado de este trabajo. Adicionalmente la herramienta provee mecanismos para depurar errores y mejorar la funcionalidad del sistema. La herramienta puede leer, compilar y ejecutar varios tests, cuyos resultados pueden ser desplegados en diversos formatos.
* Eclipse IDE+SIGASI: Suite de desarrollo con el plugin de SIGASI como suite de desarrollo para la escritura de los RTL en VHDL.
* Blender3D: Suite de Animación 3D con posibilidad de interpretar, leer y ejecutar scripts en Python, para generar entradas para el Rt Engine, ya sea en tiempo de simulación o en tiempo de ejecución.
* TestBench compiler: Herramienta creada en el marco de este trabajo para automatizar la escritura de testbenches y poder verificar el funcionamiento de señales que no pertenecen a la entidad principal como puertos de salida de esta o *top entity.*

Hardware .

* PC / Windows 7 64 Bits / Intel Core I7-2600K Sandy Bridge / CPU @ 3.40 GHz / 8 Gigs RAM
* Terasic Altera NEEK Development Kit / VGA 800x600 Touchscreen / JTAG / Ethernet / 32K External Ram / Cyclone III EP3C25F324C6 25K LE’s. / Intel P30 Flash 16Mbytes Flash / Synchronous Static RAM ADP2560 1 MByte / AD7556 DDRAM 32 MBytes

1. UNIDAD ARITMÉTICA VECTORIAL: RAYTRAC.

El *Rt Engine* es un sistema de trazado de rayos, cuyo motor aritmético es el *RayTrac*. Este motor aritmético es una Unidad Aritmética Vectorial.

Es similar a una *ALU (Unidad Aritmético Lógica)* regular pero especializada en vectores y en ella se configuran operaciones como, productos punto, productos cruz, productos escalares, suma, resta y normalización de vectores.

# Descripción General.

El RayTrac se compone de los siguientes elementos:

* Pipeline aritmético.
* Interconexión decodificada de circuitos aritméticos.
* Adaptador de formato para los flujos de datos de entrada y salida.
* Máquina de estados para el control de flujo de datos de entrada y salida.
* Bloque de registros cómo interfaz de programación y control.

El *pipeline aritmético* (*PA)* es el circuito que implementa las operaciones aritméticas.

Se compone de circuitos que llevan a cabo las operaciones de multiplicación, suma, resta, raíz cuadrada e inversión y al interconectar estos circuitos se pueden llevar a cabo operaciones vectoriales más complejas como producto punto, producto cruz, normalización, cálculo de magnitud, etc.

El acceso a los circuitos aritméticos está restringido exclusivamente al interior del *RayTrac,* no existe un esquema de direccionamiento o acceso para el programador.

La especificación y detalle de este circuito se explica en el *Capítulo III* PIPELINE Y ARQUITECTURA DE OPERACIONES ARITMÉTICAS.

La *interconexión decodificada de circuitos aritméticos (IDC)* es un circuito combinatorio encargado de interconectar las salidas y las entradas de los circuitos aritméticos que se encuentran en el *pa*.

El programador tiene la posibilidad de configurar durante la ejecución de un programa la interconexión. Al programar la interconexión, se estará configurando el *PA* para que ejecute una función especializada o *kernel* en el contexto de un *PF*. Esto quiere decir que el programador puede programa el *pipe* en modo producto punto, ejecutar *n* cantidad de operaciones de este tipo y después configurar un producto cruz y así sucesivamente.

Por ejemplo, una operación producto cruz, entre dos vectores de 3 componentes, consiste en realizar 6 multiplicaciones y con cada par de productos, 3 restas, esta secuencia difiere de la secuencia de la operación producto punto, la cual efectuaría con los mismos vectores 3 multiplicaciones, una suma con el primer par de productos y el resultado de esta se sumaría con el ultimo producto. Por lo tanto es el *interconexión decodificada de circuitos aritméticos* la encargado de conectar la secuencia de operaciones del *kernel* que se configure.

*El Adaptador de formato de flujos de datos (AFF)*, es un circuito secuencial en pipe encargado de descargar los datos que vienen desde el exterior desde el bus de datos de entrada (*32 bits de ancho*), en los registros que corresponden a las entradas del *pipeline aritmético* y también de cargar en el bus de salida (*ídem, 32 bits de ancho),* los 128 bits o 4 datos de 32 bits, que corresponden al resultado entregado por el *pipeline aritmético*.

La *Máquina de estados para el control de flujo (MECF)* es un circuito que controla y abstrae en hardware el mecanismo de control de flujos de datos. El *RayTrac* debe verse como una banda procesadora de datos. Ciclo tras ciclo, en cada etapa del *pipeline aritmético,* van transitando datos correspondientes a los valores intermedios de una operación en particular, una tras otro. Todos los datos que entran por un mismo punto y salen por un mismo punto, de ahí el nombre *pipe* o en español *tubo*.

No obstante este tubo se puede atorar, para que esto no ocurra se debe controlar el transito o *flujo* de los datos que por el tubo transitan. El *PA* y los circuitos *AFF* cuentan con señales de sincronización que son utilizadas por la *máquina de estados*, controlando la cantidad de datos que hay en el pipe: cuidando que no hayan muchos, provocando perdida de datos y que no hayan pocos, provocando la perdida de desempeño.

En conjunto con el *bloque de registros de control (BREG),* la *MECF* sabe la cantidad de datos que entran en el tubo y cuantos deben salir. De hecho es la *MECF* la encargada de ejecutar las operaciones de lectura de datos en memoria externa *–a modo de etapa fetch-* y escribir los resultados de igual forma, en una memoria externa *–a modo de etapa write-.*

El *BREG,* la interfaz de programación de bajo nivel del RayTrac, en conjunto con el *IDC,* el *PA* que hace de este un procesador de flujos y un elemento programable,

El *registro de bloques de control,* básicamente almacena los datos que controlan la ejecución de las operaciones, entre otros datos encontramos en este registro las direcciones de lectura y escritura externas, la cantidad de datos que se van a descargar, el modo de ejecución etc.

En los *BREG* se configura el *kernel* a ejecutar en el *PA.* Adicionalmente en los registros del *BREG*  se configura las direcciones de los bloques de memoria donde se encuentran los datos con los vectores a procesar y las direcciones donde se deben escribir esos resultados.

# *Es un procesador de flujos (Streaming Processor).*

El RayTrac es un procesador, desde el punto de vista conceptual.

Un procesador posee una etapa de *fetch* encargada de leer las instrucciones de la memoria a partir de un registro que le indica en que lugar de una memoria de instrucciones se encuentra la siguiente instrucción a procesar.

Un procesador posee una etapa de *decode* que básicamente señala que salida de algún circuito aritmético o lógico se debe conectar a algún tipo de memoria donde se escriben los resultados.

Un procesador posee una etapa de *execute* para realizar las operaciones que la instrucciones indiquen.

Un procesador posee una etapa de *write* para señalizar que un dato se encuentra listo, o es válido y colocar las señales que habilitan su escritura en algún tipo de memoria de resultados o bus externo etc.

El RayTrac posee todas las etapas mencionadas en el párrafo anterior:

La etapa *fetch* está a cargo de la *máquina de estados* y *el bloque de registros de control.*

La etapa *decode* la ejecuta la *interconexión decodificada de circutios aritméticos*.

La etapa execute la ejecuta el *pipeline aritmético*.

La etapa *write* la ejecuta la *máquina de estados* y *el bloque de registros de control*.

Sin embargo el *RayTrac* es un *procesador de flujos*, puesto que su operación consiste en la configuración de una función especializada o *kernel* de procesamiento, en el que se opera un conjunto de datos *extenso[[2]](#footnote-2).*

Este conjunto de datos de entrada corresponde a un *flujo de entrada (FE)* y el conjunto de resultados calculados por el *kernel* son un *flujo de salida.(FS)*

En este documento se usará la palabra *kernel* para hacer referencia a la función especializada, que ejecutaría el *pipeline aritmético,* dependiendo de la interconexión de sus circuitos a través de la *interconexión decodificada de circuitos aritméticos*. Dicho *kernel* puede ser una operación de 7, las cuales son mencionadas en la sección *1.4.*

# Diagrama de Bloques del Sistema

A continuación una breve descripción del sistema:

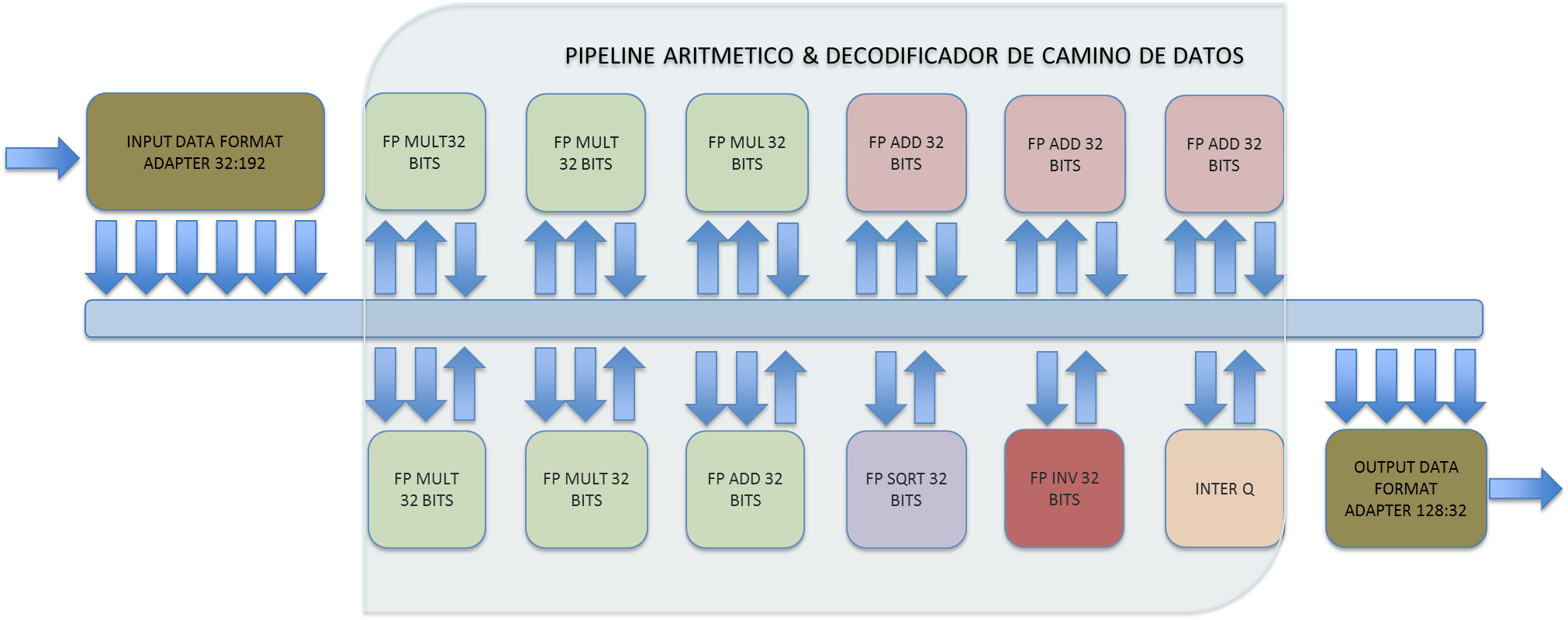


Ilustración 1. Diagrama General de Bloques del RayTrac. RTL fuente: raytrac.vhd[[3]](#footnote-3)

Los datos del exterior entran en series de 32 bits por ciclo de entrada. Estos datos entran al bloque *adaptador de formato de flujos de entrada (AFFe),* los datos que vayan entrando se agrupan en 2 vectores de 3 componentes.

Estos 2 vectores entran al *pipeline aritmético* vía la *interconexión decodificada de circuitos aritméticos,* representado en la ilustración mediante el bloque azul de la mitad y de ahí los datos se van procesando y transformando, saliendo y entrando a los distintos bloques aritméticos usando la *interconexión decodificada*.

Por ejemplo, al ejecutar un producto punto, los vectores entran a los 3 de los 6 multiplicadores del sistema, dos de los 3 productos entran a uno de los 3 sumadores, el otro producto entra a la cola de resultados intermedios. El resultado del sumador y el resultado en la cola intermedia entran a otro sumador, el resultado de esa suma entran al bloque de formateo de datos de salida, para ser entregado al exterior del *RayTrac.*

La unidad, como tal se puede ver como una línea de producción en *serie*: la materia prima, transportada y procesada a través de una *serie* de procesos, máquinas y procedimientos. Y al igual que en una línea de producción si cualquier etapa suspende su funcionamiento, las etapas previas deben suspender su funcionamiento también.

En la arquitectura del *RayTrac* sucede lo mismo, la máquina de estados que la controla debe garantizar que el pipeline aritmético no se llene y evitando el descarte de datos en el bloque de salida que corresponde al bloque de formateo de datos de salida. La máquina de estados también debe garantizar que el *pipeline aritmético* no quede vacío*,* para no perder desempeño.

Cada vez que se termine de procesar un conjunto de datos el sistema produce una interrupción que le indica a algún proceso ejecutado en el exterior que los datos se han procesado y hay resultados disponibles para ser leídos.

# Características Funcionales del Sistema.

* Funciones o *Kernels*, para vectores en punto flotante:
  + Producto Punto.
  + Producto Cruz.
  + Suma.
  + Resta.
  + Magnitud de vector.
  + Normalización de vector.
  + Multiplicación componente a componente.
* Ejecución SIMD (*Single Instruction Multiple Data).*
* Arquitectura en pipeline,
* 16x32 bits registros de control, operación y depuración.
* Interfaz compatible con bus avalon de Altera\*\*.
* API en C, para programar sobre el NIOS II.

1. BLOQUES DE ADAPATACION DE FORMATO PARA LA ENTRADA Y SALIDA DE FLUJOS DATOS.

La unidad aritmética vectorial en punto flotante, *RayTrac* tiene una interfase de entrada y salida: los bloques de adaptación de formato. Los flujos de datos de entrada son poseen datos que en su forma, distintos a los que opera el *kernel* configurado en el *RayTrac* y también es distinto el formato de los datos que conforman los flujos de salida que el de los resultados que entrega el *kernel* configurado en el *RayTrac.*

La adaptación es necesaria porque la entrada de datos al *RayTrac* es a través de un único bus de datos de 32 bits de ancho. Pero las operaciones que realiza el *RayTrac* son con vectores los cuales poseen mínimo 3 componentes de cada uno y en general se utilizan dos vectores por operación. Para que una operación pueda “arrancar” o ser “valida” deben encontrarse todos los datos de ambos vectores presentes.

La necesidad de la adaptación también corre por cuenta del flujo de datos de salida. Ya que el resultado entregado por el *RayTrac* es en general un vector de 3 o 4 componentes de 32 bits cada uno y el bus de datos de salida es de 32 bits de ancho. Lo que hace necesario un control de flujo para serializar cada uno de los componentes del vector.

Son 2 los elementos de adaptación de flujos que posee el *RayTrac*:

* Adaptador 32:192 (Flujo de Entrada).
* Adaptador 128:32 (Flujo de Salida).

En este capítulo se detalla la implementación realizada. La función de estos bloques es servir de interfaz de entrada y salida al *RayTrac*.

Los mecanismos de interfaz de entrada y salida del *RayTrac* son transparentes al usuario que desarrolla software para el *RayTrac,* por lo tanto la descripción de la implementación hecha es de una interfaz de *Hardware* y no corresponde a una interfaz de *Software.*

* 1. Adaptador de formato de flujos de salida (AFFs).

El *adaptador de formato de flujos de salida* tiene la función de tomar los flujos que vienen como resultados de 128 bits, desde el *PA* transformarlos en flujos de 32 bits, mediante la creación de una secuencia de 4 flujos de 32 bits cada uno. El problema es que la salida del adaptador es el bus de datos de salida externo al *RayTrac* y al ser este un bus arbitrado, no es accesible todo el tiempo. El mecanismo de arbitramento no es determinístico, o sea, no se puede predecir en que momento el acceso al bus será otorgado. Por lo tanto los datos se deben “encolar” para ello se utiliza una cola de 32 bits de ancho, en donde se van almacenando secuencialmente los flujos 32 bits secuenciales. Una vez el mecanismo de arbitramento otorga el acceso al bus se activa la señal de lectura de la cola.

El *adaptador de formato de flujos de salida* se compone de una cola de resultados vectoriales, que emite flujos de 128 bits de ancho, una máquina de estados de transición y un cola de salida de datos que permite flujos de 32 bits de ancho.

Cola de Resultados .

La cola de resultados, es un *fifo* de 128 bits de ancho. Almacena vectores de 4 componentes y su entrada esta conectada a la salida del *pipeline aritmético,* desde las señales de resultado de los bloques aritméticos que correspondan según el *kernel*  que se encuentre configurado al momento. La escritura de los datos en la cola o  *push in* está a cargo de los mecanismos de sincronización el *pipeline aritmético.* La lectura o

*push out* está a cargo del sistema de adaptación.

La cola de resultados provee la señal *empty* para indicar al resto del *RayTrac* que se encuentra vacía. Esto lo utiliza el adaptador de formato de flujos de datos de salida, pero también la *máquina de estados* del *RayTrac,* para la detección de ciertos eventos, como el final del procesamiento de un flujo de datos a través del *kernel* configurado.

Para implementar la descripción de la cola se utilizó la librería *altera*\_*mf*, la megafunción *scififo.* A continuación el siguiente bloque define las características de la implementación:

|  |  |
| --- | --- |
| **Ítem** | **Result Q** |
| Width | 128 |
| Num Words | 256 |
| Sync Signals | Wr, Ack, Empty. |
| M9k Blocks / LCells | M9K |

Tabla 1. Configuración de la cola de resultados del Adaptador de Formato de Flujos de Salida.

Máquina de estado adaptación de flujos.

Las solución al problema de la adaptación de los flujos de datos de salida, es escoger secuencialmente los componentes del vector en la salida de la cola de resultados.

La máquina de adaptación de flujos de salida, es una máquina de estados de 4 estados. Estos estados son VX,VY,VZ,SC. El objetivo de cada estado es decodificar que dato debe transitar hacia la interfaz externa. La máquina se activará cada vez que haya un resultado en la cola de resultados.

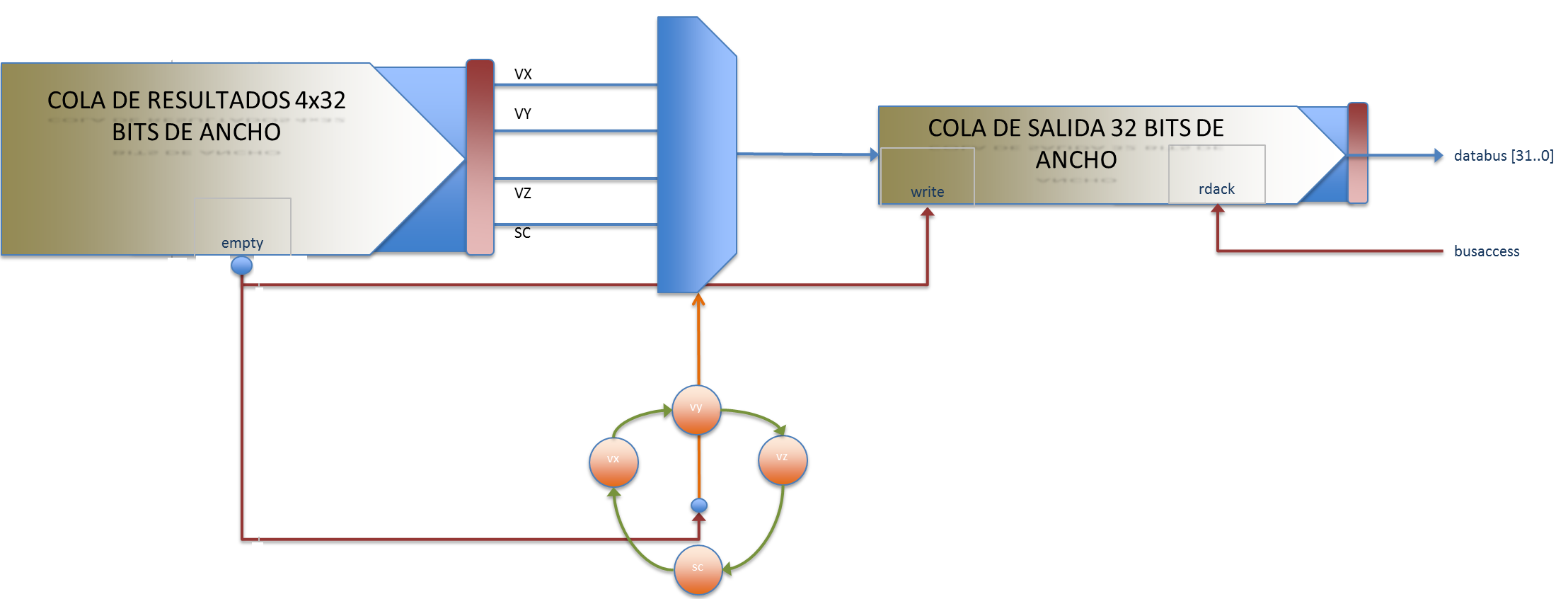


Ilustración 2. Adaptador de Formato de Flujos de Salida. La máquina de estados, según el estado en que se encuentre insertara los datos Vx, Vy, Vz y Sc, provenientes de la cola de resultados en la cola de salidas, transformando el flujo de 128 bits en un flujo de 32.

Se mencionó la interfaz externa en el párrafo anterior. Esa interfaz no es propiamente el bus de interconexión externo al *RayTrac.* Naturalmente se puede pensar en primera instancia que se le entregue el dato al bus externo, pero esto no es real, porque para esto deben estar sincronizados dos eventos: el evento que señaliza que la cola de resultados no está vacía y el evento que señaliza que el bus de datos a la salida se encuentra disponible. Por lo tanto y para sincronizar estos dos eventos se utiliza una cola de salida extra.

Cola de Salida.

La cola de salida en cumple dos funciones: soporte para el control de flujo y tránsito de datos a lo largo del *pipeline aritmético* y sincronizar los eventos *empty* de la cola de resultados y la disponibilidad del bus externo de datos de salida, hacia donde deben transitar el flujo.

Los datos a los que se hace *push in,* provienen de la cola de resultados y el orden con que estos datos se cargan en la cola de salida está controlado por la máquina de estados de adaptación de flujos de salida.

Para leer los datos de la cola de salida, se debe esperar a que la *MEFC,* inicie una *transacción*  de carga en el bus de datos de salida. El bus externo de datos de salida notifica el acceso a la cola de salida, para que esta entregue el dato.

|  |  |
| --- | --- |
| **Ítem** | **Result Q** |
| Width | 32 |
| Num Words | 256 |
| Sync Signals | Wr, Ack, Empty, UsedW |
| M9k Blocks / LCells | M9K |

Tabla 2. Configuración de la cola de salida del Adaptador de Formato de Flujos de Salida.

* 1. Descripción del bloque adaptador de flujos de entrada.

Los flujos de datos que provienen del bus de dato exterior, son de 32 bits. El *pipeline aritmético* procesa vectores de 3 componentes de 32 bits cada uno. Por lo tanto ántes de que se carguen efectivamente los datos dentro del *pipeline aritmético,* es necesario primero precargar por los menos 6 de estos datos para operaciones de dos vectores y 3 de estos datos para operaciones unarias. Este bloque controla la secuencia de entrada de datos y carga de los vectores una vez se encuentran todos enganchados dentro del *RayTrac.*

Máquina de Estados de Sincronización de Carga.

*La máquina de Estados de Sincronización de Carga sirve para distribuir los datos que entran al* RayTrac *en los registros que forman los vectores que operará el* kernel *conforme van llegando.*

*Si el* kernel *configurado es una operación unaria (p.ej. Magnitud de un Vector), entonces la máquina transitará por los estados AXBX, AYBY,AZBZ, cargando tres valores por vector, si el* kernel *configurado es una operación de dos vectores (p.ej Suma de Vectores), entonces la máquina irá por los estados AX,AY,AZ,BX,BY,BZ, cargando seis valores por cada para de vectores.*

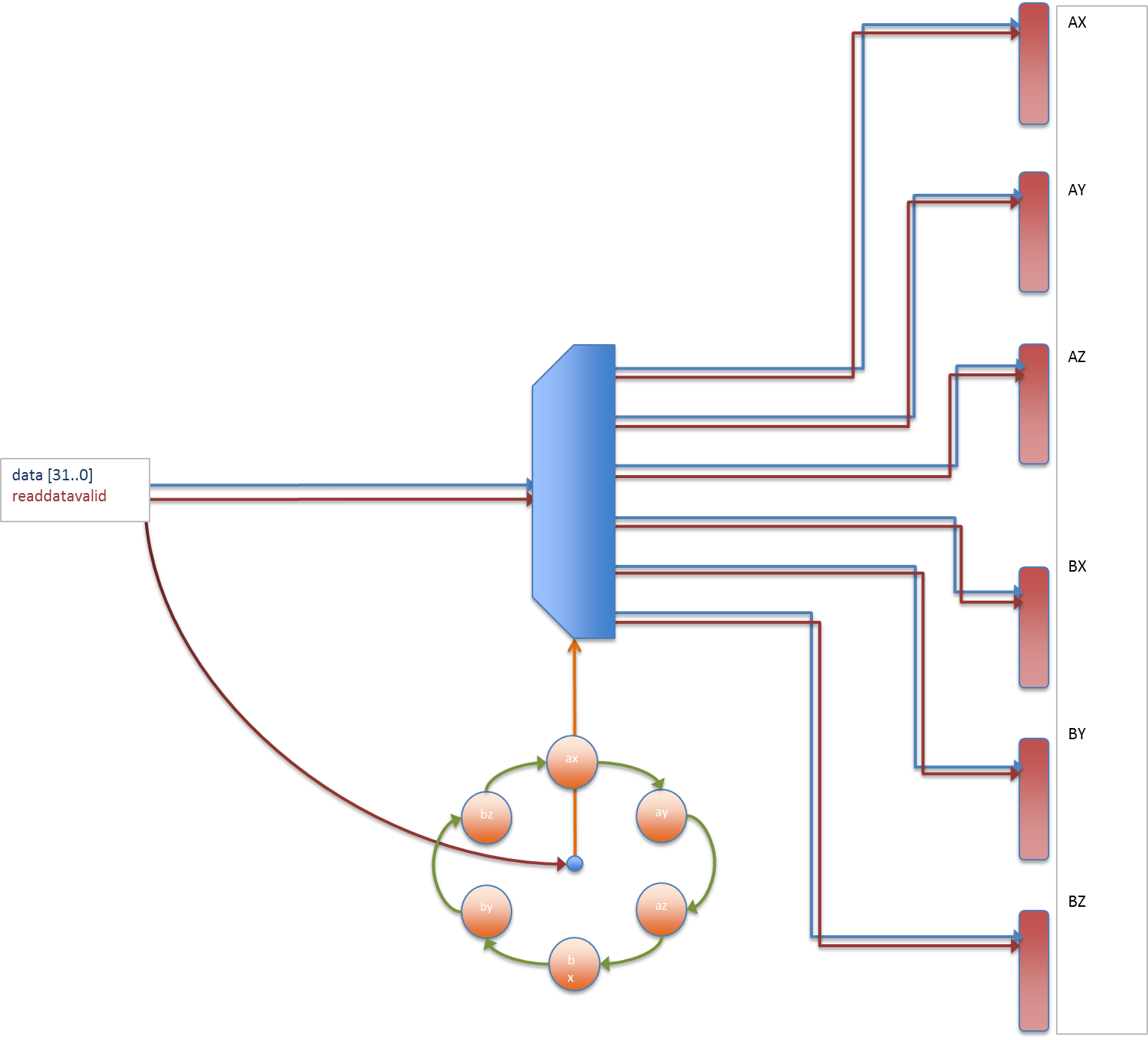
******

Ilustración 3. Adaptador de Formato de Flujos de Entrada. La máquina de estados determina en que registro se escribe los datos validos que vienen de un bus externo.

La idea es cargar secuencialmente los datos en seis registros con la información de los siguientes vectores a operar dentro del *kernel* configurado. Una vez se encuentren cargados los seis registros con datos *“validos”*, se procede a procesarlos en la siguiente etapa del *RayTrac,*  la cual corresponde al *pipeline aritmético*. El último registro de la cadena de sincronización de carga servirá de señal para dar como valido los seis datos que se encuentran cargados en los registros de datos.

La máquina de estados se activa cada vez que hay un dato valido en la interfaz externa de entrada, la cual es un bus de datos que permite flujos con un ancho de 32 bits. Entre los 6 registros de 32 bits cada uno hay 192 bits, una vez están cargados los datos en los registros se tiene entonces el flujo rearmado pero el ancho ahora es de 192 bits. La relación del adaptador es de 32 a 192. El diseño implementado es un demultiplexor con su controlador gobernado por la máquina de estados.

El *AFFe* se constituye como el circuito de entrada de flujos en un contexto de procesamiento de flujos o como la entrada desde el bus de datos externo de entrada en el contexto de la arquitectura del *RayTrac*.

1. PIPELINE Y ARQUITECTURA DE OPERACIONES ARITMÉTICAS.

El pipeline aritmético es un conjunto de circuitos combinatorios conectados de manera secuencial mediante registros. Cada circuito tiene las entradas y las salidas registradas. Dado que los circuitos están conectados de manera secuencial, las salidas registradas de un circuito, son las entradas registradas del siguiente. A cada circuito combinatorio con entradas y salidas registradas, se le denomina *etapa.*

La misión principal *de las etapas* en el pipeline es ejecutar operaciones de naturaleza aritmética, tales operaciones son:

* Suma.
* Resta.
* Multiplicación.
* Inversión.
* Raíz Cuadrada.

# Arquitectura del Pipeline Aritmético.

El pipeline aritmético cuenta con once bloques que realizan diversas operaciones aritméticas en punto flotante:

* 3 Sumadores.
* 6 Multiplicadores.
* 1 Raíz Cuadrada
* 1 Inversor.

Cada circuito tiene una latencia específica que se detalla más adelante en esta sección, la latencia está determinada en ciclos de reloj. La salidas y las entradas de cada circuito están conectadas a la *IDC.*

El pipeline cuenta con 6 registros de entradas de operandos. Una vez entran los operandos al pipeline estos se van transformado etapa tras etapa, dentro de cada circuito aritmético. Cada vez que salen de la última etapa de cualquier circuito aritmético entran a la *interconexión decodificada de circuitos aritméticos* y de ahí continúan al siguiente circuito aritmético que corresponda o al bloque de *adaptación de formato de flujos de salida*, si el resultado ya está listo.

Dicho control se encuentra detallado y especificado en el *Capítulo* ***¡Error! No se encuentra el origen de la referencia.******¡Error! No se encuentra el origen de la referencia.****.*

Como la *interconexión decodificada de circuitos aritméticos* puede configurar distintas funciones especializadas o *kernels,* la longitud o extensión del camino de datos resultante puede cambiar y a lo largo de este puede variar el número de circuitos aritméticos que procesarían los vectores de entrada.

La interconexión resultante de las diferentes operaciones resulta en *kernels*  u operaciones compuestas, que no son más que operaciones vectoriales y estas son:

* Producto Punto
* Producto Cruz
* Suma de Vectores
* Resta de Vectores
* Multiplicación de un Vector por un número Escalar.
* Magnitud de Vectores
* Normalización de Vectores

Estas operaciones son las que realmente se especifican en la interfaz de programación del *RayTrac*. En el *Capítulo V* Interfase de Programación y Sistema de Interrupciones.

Cada *kernel* configurado supone pues un *pipeline “distinto”* . El detalle de los *kernels* configurados para cada operación se encuentra detallado en el *Capítulo IV, Interconexión Decodificada de Circuitos Aritméticos.*

# Representación de Datos en Punto Flotante.

Los flujos de entrada y de salida en el Raytrac, están representados en el sistema aritmético base 2 y sus valores aritméticos están representados en punto flotante sencillo, usando el estándar IEEE 754[[4]](#footnote-4).

Este estándar de representación viene dado por un trama de 32 bits. Esta trama tiene 3 campos que define el estándar:

* Signo: Es un campo de un bit, que en general, define si el valor del número a representar es menor que cero. El estándar permite representar el pseudo - valor “-0”.
* Exponente: Es un campo de 8 bits y su propósito es representar un exponente con offset.
* Mantisa: Es un campo de 23 bits. Representa un factor por el que se debe multiplicar un valor potencia de 2, para obtener el valor representado. Los valores de la mantisa son fijos y pertenecen al rango [1.0,2.0). La representación de la mantisa en punto fijo que se denota A(1,23). La notación indica que la mantisa se representa con 24 bits, uno más que el campo de la trama correspondiente, la razón es que el bit de la parte entera siempre debe ser 1.

Los bloques que componen las operaciones soportan el estándar IEEE 754 de representación. Por lo tanto si se requiere operar 2 valores enteros, estos deben ser convertidos en el código a estándar IEEE 754, mediante software y esto causa un overhead en el performance.

## Punto Fijo vs. Punto Flotante.

Cuando se implementa en hardware los bloques aritméticos, usando punto flotante, se garantiza que el tamaño de palabra del resultado y los parámetros son idénticos. Una arquitectura de punto fijo debe truncar el valor de los bits menos significativos para poder garantizar esto. Si la arquitectura en hardware no los truncará y simplemente asumiera el valor de palabra distinto en la salidas, sería imposible reutilizar los mismos bloques para hacer distintas operaciones, por ejemplo sería imposible usar los mismos bloques de suma, para implementar producto punto y suma de vectores.

b0

b1

b2

b3

b4

b5

b6

b7

b0

b1

b2

b3

b4

b5

b6

b7

b8

b9

b10

b11

b12

b13

b14

b15

b0

b1

b2

b3

b4

b5

b6

b7

Ilustración . Suma en punto fijo. Los 2 sumandos poseen 8 bits cada uno y el resultado de la suma deriva en un resultado de 16.

El problema de mayor repercusión en la arquitectura de punto fijo es que el tamaño de los operandos y de los resultados, los cuales difieren, provocando que no se puedan realizar sumas acumulativas, puesto que en cada iteración de dicha suma, se requeriría de un circuito que soporte parámetros con el doble de ancho que la iteración anterior.

Finalmente si la arquitectura optará por el truncamiento para mantener fijo el tamaño de los operandos, es inminente la perdida de precisión y por lo tanto perdida de información. Esto no es aceptable.

Sin embargo existen razones para preferir la implementación de una unidad de punto fijo en lugar de punto flotante y la más importante de ellas es la cantidad de celdas lógicas a usar.

Sin embargo el uso de circuitos de punto flotante, resultaría favorable, a pesar que dicha implementación utilizaría más celdas lógicas que el circuito de punto fijo, es claro que resulta mucho más funcional y tiene grandes ventajas en la precisión.

En la siguiente ilustración se observa la suma de 2 números en punto flotante, los cuales poseen un bit para representar el signo, 8 para el exponente y 23 para la mantisa. Cada componente, -signo, exponente o mantisa-, debe ser operado con su contraparte en el otro parámetro, aunque esto conlleve un mayor número de celdas lógicas al final se obtiene un resultado del mismo ancho que los parámetros, lo cual resulta una ventaja en términos de diseño bastante importante.

b8

b9

b10

b11

b12

b13

b14

b15

b0

b1

b2

b3

b4

b5

b6

b7

b17

b18

b19

b20

b21

b22

b16

b29

b23

b24

b25

b26

b27

b28

b30

b31

b8

b9

b10

b11

b12

b13

b14

b15

b0

b1

b2

b3

b4

b5

b6

b7

b17

b18

b19

b20

b21

b22

b16

b29

b23

b24

b25

b26

b27

b28

b30

b31

b8

b9

b10

b11

b12

b13

b14

b15

b0

b1

b2

b3

b4

b5

b6

b7

b17

b18

b19

b20

b21

b22

b16

b29

b23

b24

b25

b26

b27

b28

b30

b31

Ilustración . Suma en punto fijo. Los 2 sumandos poseen 8 bits cada uno y el resultado de la suma deriva en un resultado de 16.

# Suma IEEE 754, 32 Bits Floating Point, Single Precision.

## Abstracción algebraica de la operación.

Se tiene los valores F0 y F1 cuya representación es en punto flotante:

F0 = s0 2e0 f0

F1 = s12e1 f1

Donde si toma el valor de -1 o 1 dependiendo del signo, ei es un valor entero y fi es una fracción en el rango [1.0,2.0).

F0 + F1 = s0 2e0 f0 + s12e1 f1

Por razones prácticas en la ilustración de este ejemplo, se supone que e0 > e1. Se tiene entonces:

Se puede reemplazar el valor de la diferencia de los exponentes, multiplicando y dividiendo al mismo tiempo por una potencia de 2 elevada a :

Y reemplazamos e0 en la expresión:

Finalmente multiplicar por una potencia de 2 elevada a un numero negativo, en representación binaria, significa hacer un corrimiento de bits a la derecha:

Sin embargo la fracción resultado *fres’*puede que no haya quedado normalizada, dentro del rango [1.0,2.0). Así que se debe normalizar, realizando corrimientos a la izquierda o a la derecha, los bits de *fres’* para obtener la fracción final y restando o sumando respectivamente, a e0, el número de corrimientos hechos, obteniendo finalmente *2res*.

## Diagrama de Flujo.

|  |  |
| --- | --- |
|  | Etapa 0   * Cálculo del signo del segundo operando (s1). * Cálculo de la diferencia de los exponentes (delta). * Si el exponente e0 es menor e1, se intercambian todos los operandos. |
| Etapa 1   * Decodificación “One Hot” de la diferencia de exponentes (deltaOneHot). * Correr los bits de la mantisa de menor exponente. * Signar la mantisa de mayor exponente. |
| Etapa 2   * Signar la mantisa corrida. |
| Etapa 3   * Sumar las mantisas. |
| Etapa 4   * Sacar el signo de la mantisa resultante. |
| Etapa 5   * Calcular el corrimiento que requerirá la mantissa (delta). |

Tabla . Diagrama de flujo de la suma flotante. La columna de la izquierda ilustra el diagrama, la columna en la derecha, detalla los bloques correspondientes en la columna izquierda. Cada etapa se encuentra precedida o sucedida por flipflops.

## Diagrama de Bloques

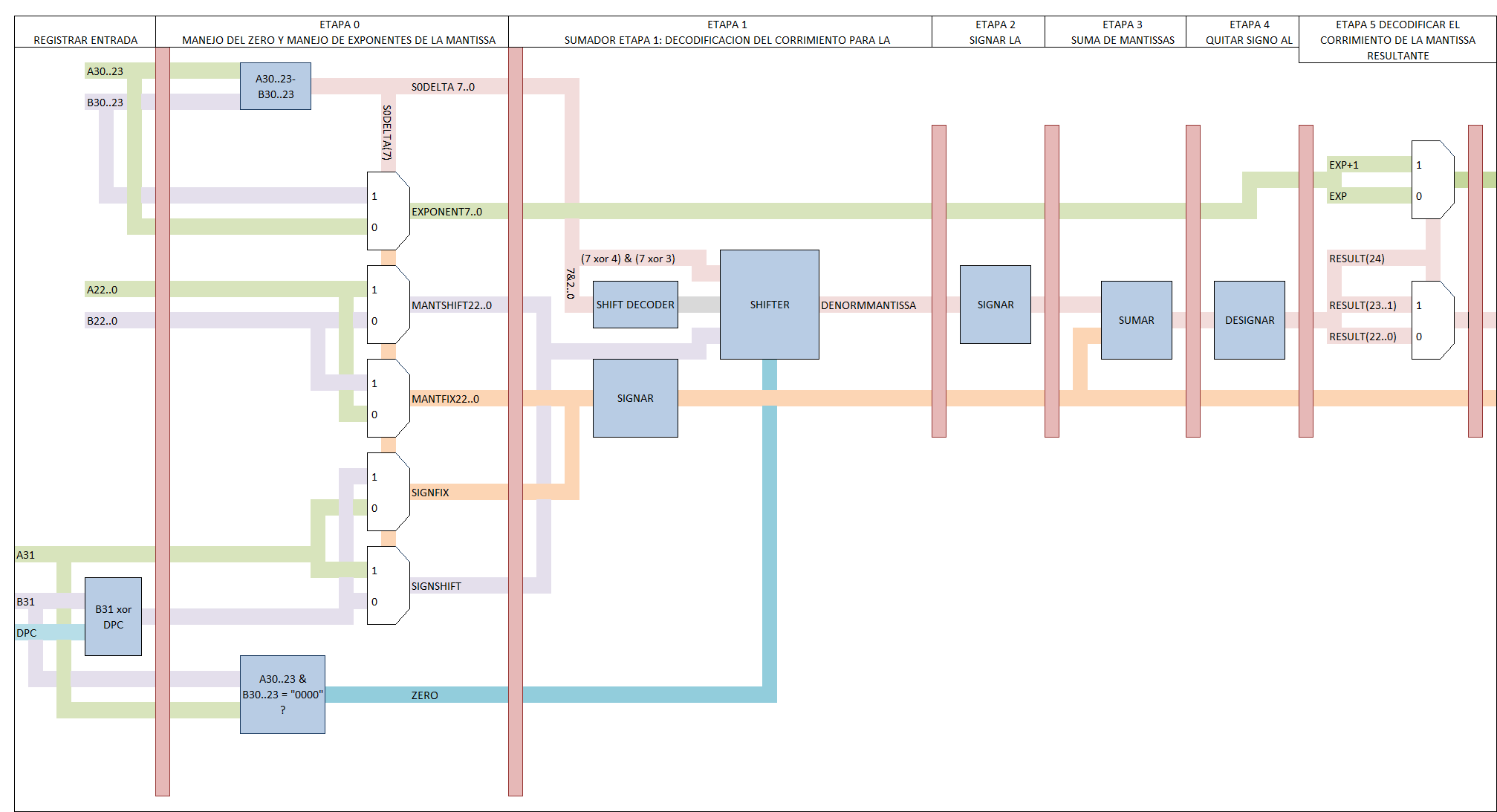


Ilustración . Diagrama de Bloques de la suma en punto flotante de 32 bits. Archivo RTL: fadd32.vhd

## Recursos Utilizados, Sumador en Punto Flotante 32 bits.

Los recursos utilizados por la implementación se encuentran detallados en el *Capítulo Octavo, Integración e Implementación del Sistema.*

# Multiplicación 32 Bits Floating Point IEEE 754, Single Precision.

## Abstracción Algebraica de la operación.

Se tiene los valores F0 y F1 cuya representación es en punto flotante:

*F0 = s0 2e0 f0*

*F1 = s12e1 f1*

Donde si toma el valor de -1 o 1 dependiendo del signo, ei es un valor entero y fi es una fracción en el rango [1.0,2.0).

El producto de los dos valores será el siguiente:

Si el valor de la fracción calculada *fres* es mayor o igual a 2.0, entonces se debe dividir por 2 y el valor del exponente se le debe sumar 1.

## Diagrama de Flujo.

|  |  |
| --- | --- |
|  | Etapa 0   * Suma de Exponentes * Multiplicación de Signos * Multiplicación de Mantisas |
| Etapa 1   * Normalización de la mantisa resultante. |
| Etapa 2   * Normalización del exponente resultante. |

Tabla . Diagrama de flujo que describe las etapas combinatorias del multiplicador en punto flotante.

## Diagrama de Bloques.

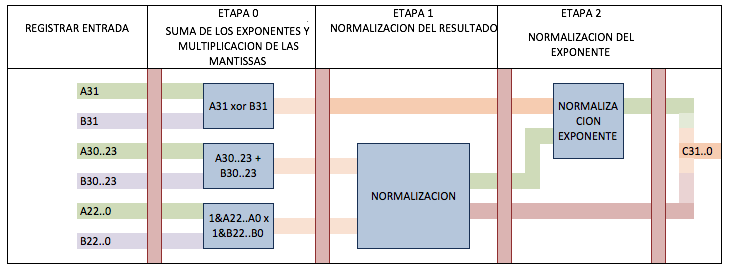


Ilustración . Diagrama de bloques del circuito de multiplicación en punto flotante IEEE754.

## Recursos Utilizados por el Multiplicador en Punto Flotante IEEE754.

Los recursos utilizados por la implementación se encuentran detallados en el *Capítulo Octavo, Integración e Implementación del Sistema.*

# Inversor, 32 Bits en Punto Flotante IEEE 754, Precisión Sencilla.

## Abstracción Algebraica de la operación,

Se tiene el valor Fi cuya representación es en punto flotante:

*Fi = si 2ei fi*

Donde si toma el valor de -1 o 1 dependiendo del signo, ei es un valor entero y fi es una fracción en el rango [1.0,2.0).

El valor elevado a la -1 potencia sería:

Dado que si es 1 o -1, el valor de si-1 es si. El valor de el inverso de Fi es entonces:

Si el valor de la fracción calculada *fi-1* es menor a 1, entonces se debe multiplicar por 2 y el valor del exponente se le debe restar 1.

El circuito está calculado usando lógica combinatoria, excepto las expresiones y , las cuales utilizan una tabla de búsqueda, implementada en una memoria ROM de 1024 direcciones y 18 bits de ancho de palabra. El direccionamiento de esa tabla es con una palabra de 10 bits de ancho, que corresponden a los 10 bits más altos de la mantisa *fi*. Se ignoran los 13 bits restantes de la mantisa y por lo tanto habrá una perdida de precisión. Usar mas bits habría requerido memorias más grandes y es posible aumentar la precisión aumentar la precisión mediante este método.

## Diagrama de Flujo.

er = -er

fr = memsqrimpar(fr[22..13])

Ilustración . Diagrama de Flujo Inversor 32 bits punto flotante IEEE754. Se cambia de signo al exponente y direccionando con los primeros 10 bits de la mantissa se calcula el valor de la fracción invertida.

## Diagrama de Bloques.

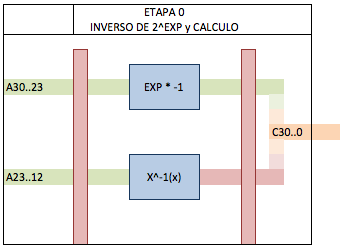


Ilustración . Diagrama de bloques del multiplicador flotante en 32 bits IEEE754. RTL: invr32.vhd

## Recursos Utilizados.

Los recursos utilizados por la implementación se encuentran detallados en el *Capítulo Octavo, Integración e Implementación del Sistema.*

# Raíz Cuadrada IEEE 754, 32 Bits Floating Point, Single Precision.

## Abstracción Algebraica de la operación,

Se tiene el valor F0 cuya representación es en punto flotante:

*Fi = si 2ei fi*

Al calcular la raíz cuadrada se asume que si vale 1, puesto que el dominio de la función raíz cuadrada es [0,∞) ei es un valor entero y fi es una fracción en el rango [1.0,2.0).

El valor elevado a la -1 potencia sería:

El valor del exponente resultante *ei*/2 debe ser un valor entero. Por lo tanto el valor de *ei* debe ser un número par. Si *ei* no es un número par, se requerirá de una normalización del exponente y la fracción. Finalmente el valor de la raíz cuadrada se calculará así:

El circuito está calculado usando lógica combinatoria, excepto las expresiones o y , las cuales utilizan una tabla de búsqueda, implementada en una memoria ROM de 1024 direcciones y 18 bits de ancho de palabra. El direccionamiento de esa tabla es con una palabra de 10 bits de ancho, que corresponden a el bit menos significativo del exponente y a los 9 bits más altos de la mantisa *fi*. Se ignoran los 14 bits restantes de la mantisa y por lo tanto habrá una perdida de precisión. Usar mas bits habría requerido memorias más grandes y es posible aumentar la precisión mediante este método.

## Diagrama de Flujo.

Expsq = Expr >> 1

fsq = memsqrimpar(fr[22..14])

Exp0 impar?

fsq = memsqrpar(fr[22..14])

Ilustración . Diagrama de Flujo Operación Raíz Cuadrada en formato IEEE754. El exponente se divide entre dos mediante el corrimiento de un bit a la derecha. Usado una de dos memorias, en función de la paridad del exponente de entrada, se cálcula la mantissa.

## Diagrama de Bloques.

## 

Ilustración . Diagrama de bloques del multiplicador en 32 bits de punto flotante IEEE754.RTL: sqrt32.vhd

## Recursos Utilizados.

Los recursos utilizados por la implementación se encuentran detallados en el *Capítulo Octavo, Integración e Implementación del Sistema.*

# Colas de Sincronización.

## Cola Producto Punto.

El pipeline aritmético soporta la operación vectorial Producto Punto. En ella se llevan a cabo 3 multiplicaciones y 2 sumas. Las 3 multiplicaciones se pueden realizar de manera simultánea, pues son 6 operandos y 3 multiplicadores. Los 3 resultados deben sumarse, sin embargo los bloques de sumas solo soportan 2 operandos. Por lo tanto debe sumarse los primeros 2 productos y el resultado se sumaría con el tercer producto.

Cuando se suman los primeros 2 productos, el tercer producto debe seguir moviéndose por el pipeline, para ello se implementa la cola de sincronización. La cola de sincronización de producto punto, es una cola que tiene tantas posiciones como etapas el sumador, seis. El tercer producto entra en esta cola mientras que los otros 2 productos son sumados en primera instancia y el resultado de esa suma se suma con el tercer producto que es enganchado fuera de la cola en ese momento.

Lo anterior se ilustra en el *Capitulo IV. Interconexión Decodificada de Circuitos Aritméticos.*

## Cola Normalización de Vectores.

Cuando se realiza la operación de Normalización de Vectores, lo primero que se hace es calcular la magnitud del vector, después se calcula el inverso de dicha magnitud, al que se debe, por último, multiplicar por cada uno de los componentes del vector.

En la etapa que se empieza a calcular la magnitud del vector, al mismo tiempo se inserta en la cola de sincronización. El vector permanece en la cola mientras se calcula la magnitud de este y se invierte. En el momento que se invierte se engancha el vector por fuera de la cola y el resultado de la inversión de la magnitud se multiplica por cada uno de los componentes.

Lo anterior se ilustra en el *Capitulo IV. Interconexión Decodificada de Circuitos Aritméticos.*

## Recursos Utilizados.

Los recursos de control de flujo y señales de sincronización asociados a las colas de sincronización del *pipeline aritmético*, están detallados en el *Capítulo VIII. Implementación e Integración del Sistema.*

Las cola de sincronización de la operación de normalización de vectores es una instancia de la biblioteca altera\_mf, que es de la IP de Altera. Dicha instancia se encuentra en el RTL del controlador de camino de datos y sincronización. El siguiente bloque resume la configuración de los parámetros de dichas funciones:

|  |  |  |
| --- | --- | --- |
| **Ítem** | **Dot Prod Fifo** | **Norm Fifo** |
| Width | 32 | 32 x 3 |
| Num Words | 6 | 19 |
| Sync Signals | Wr, RdAck | Wr, RdAck, Empty |
| M9k Blocks / LCells | LCells | M9K |

Tabla . Configuración de las colas de sincronización del pipeline aritmético.

1. INTERFASE DE PROGRAMACION DE BAJO NIVEL DEL RAYTRAC.

Como se ha mencionado se considera al *RayTrac* un *Procesador de Flujos.* La función que cumple un procesador de este tipo es la de una función especializada o *Kernel.* A este *kernel* entran y salen flujos de datos, con datos por procesar en los flujos de entrada y con datos procesador en los flujos de salida.

Dado lo anterior son dos los aspectos que se pueden programar en el *RayTrac:* el *kernel* o función especializada a ejecutar y la fuente y destino de los flujos de entrada y salida respectivamente. La *acto* de programar se lleva a cabo mediante la escritura en ciertos registros ubicados en el *bloque de registros de programación y control de flujo*.

# Bloque de Registros de Programación y Control de Flujo.

El bloque de registros consiste en un conjunto de *16 egistros de 32 bits de ancho*, que sirven para programar y controlar el *RayTrac.*

A continuación se enumeran y describen dichos registros.

|  |  |  |
| --- | --- | --- |
| *Nombre/s* | *Descripción* | *Dirección* |
| *Reg\_ctrl* | *Registro de control: En este registro se llevan a cabo las configuraciones que definen la operación o kernel a ejecutar, la operación de el adaptador de formato de flujos de salida, la operación del adaptador de flujos de datos de entrada y se almacenan algunas banderas para soportar el control de flujo.*  *Más adelante se detalla este registro.* | *BASE+0* |
| *Reg\_Vx/Reg\_Vy/Reg\_Vz/Reg\_Sc* | *Estos registros almacenan el último vector registrado en el adaptador de formato de flujos de salida.* | *BASE+4 / BASE+8 / BASE+12 / BASE+16* |
| *Reg\_Scratch00* | *Registro Scratch, para realizar pruebas almacenar datos etc. En general se utiliza para depuración del hardware.* | *BASE + 20* |
| *Reg\_OutputCounter* | *Registro que cuenta el número de datos que han salido hacia el bus de datos externo. El programador puede escribir cualquier valor en este registro, perdiendo la cuenta que el sistema lleve.* | *BASE + 24* |
| *Reg InputCounter* | *Registro que al igual que OutputCounter, cuenta datos, pero de entrada. Es escribible por el programador.* | *BASE + 28* |
| *Reg FetchStart* | *Registro en el que se programa la dirección externa donde se almacena el bloque de flujo de datos de entrada.* | *BASE + 32* |
| *Reg SinkStart* | *Registro en el que se programa la ubicación externa a donde se direccionará el flujo de salida de datos* | *BASE + 36* |
| *Reg\_Ax/Reg\_Ay/Reg\_Az / Reg\_Bx/Reg\_By/Reg\_Bz* | *Registros en los que se almacena el último flujo de entrada “formateado” por el adapatador de formatos de flujo de entrada, es decir en estos registros se cargan el último par de vectores que entraron en el sistema.* | *BASE + 40/ BASE + 44/*  *BASE + 48/*  *BASE + 52/*  *BASE + 56/*  *BASE + 60* |

Tabla . Nombre, descripción y direcciones del los registros del bloque de registros de programación y control de flujo.

Los registros que se utilizan para programar el *kernel* usando la *interconexión decodificada de circuitos aritméticos* son el registro de control, reg\_ctrl, el registro de dirección de fuente de flujo de datos de entrada, reg\_fetchstart y el registro de dirección de escritura de resultados del flujo de datos de salida, reg\_sinkstart.

# Programación del *kernel* o función especializada y direcciones de almacenamiento de flujos de entrada y salida.

El *RayTrac* utiliza un grupo de circuitos aritméticos para ejecutar las operaciones que el programador configura. Para ordenar el camino que los datos recorren desde la entrada, hasta la salida y por el cual se van transformando, el sistema utiliza la *interconexión decodificada de circuitos aritméticos.* Este circuito decodifica función especializada o *kernel* configurado por el programador y está decodificación determina entonces el camino de datos dentro del *pipeline aritmético.*

Haciendo uso del reg\_ctrl, el programador configura el *kernel* o función especializada a ejecutar. A continuación se detallan los campos de este registro.

|  |  |  |
| --- | --- | --- |
| *Bit No.* | *Nombre (rw)* | *Descripción* |
| *0* | *cmb (rw)* | *Este bit describe la manera en que el adaptador de flujos de entrada carga o formatea los flujos de entrada. Este campo aplica para operaciones que requieren dos vectores A y B. Se ignora para operaciones unarias como la normalización de vectores o el cálculo de magnitud de vectores.*  *0: Todos los flujos de entrada se cargan secuencialmente en seis componentes, que conforman los 2 vectores A y B, de la siguiente manera: Ax, Ay, Az, Bx, By, Bz, Ax, Ay ….. y así sucesivamente. Una vez exista otro dato disponible entrante, empieza de nuevo la secuencia.*  *1: Todos los flujos de entrada se cargan secuencialmente en los 3 componentes que conforman el vector B, salvo los primeros 3 datos del flujo de entrada que se cargarán en el vector A, de la siguiente manera: Ax, Ay, Az, Bx, By, Bz, Bx, By, Bz, Bx, By, Bz, Bx…… y así sucesivamente.* |
| *[3:1]* | *dcs (rw)* | *Kernel o función especializada a ejecutar.*  *El valor de estos 3 bits codifica la operación a realizar y es función de la interconexión decodificada de circuitos aritméticos decodificarlos, para interconectar las operaciones aritméticas que definen un kernel.*  *000: Suma de Vectores. A+B.*  *001: Resta de Vectores. A-B.*  *011: Producto Cruz. AxB.*  *100: Producto Punto. A.B.*  *110: Magnitud y Normalización de Vectores. |A| y A\*1/|A|.*  *111: Multiplicación Simple Componente a Componente.* |
| *[5:4]* | *vtsc (rw)* | *Este campo define la tasa de adaptación de flujos de salida.*  *00 ó 10: Solo leer el componente vectorial de la cola de resultados 96:32. Se debe usar este valor cuando se quiere configurar un kernel cuyos resultados solo sean vectores, ejemplo: Producto Cruz.*  *01: Solo leer el componente escalar de la cola de resultados 32:32. Se debe usar este valor cuando se quiere configurar un kernel cuyos resultados son exclusivamente valores escalares, por ejemplo el producto punto o la magnitud vectorial.*  *11: Leer el componente vectorial y escalar de la cola de resultados 128:32. Cuando se selecciona la operación Magnitud y Normalización Vectorial (dcs=110), se debe tener en cuenta que en la cola de resultados se almacenaran cuatro valores por resultados: Los 3 componentes del vector normalizado y la magnitud del vector original. Es deseable leer los cuatro datos y en ese caso se utiliza el valor vtsc=11.* |
| *6* | *dma (rw)* | *0: El sistema funciona como un stream processor, al que se le configura un kernel o función especializada para ejecutar y una fuente de flujos de entrada y un destino para los flujos de salida. A este modo de le llama modo RayTrac.*  *1: El sistema funciona como un direct memory access, escribiendo bulks de datos utilizando los registro REG\_FETCHSART como dirección externa de lectura de datos y REG\_SINKSTART como dirección externa de escritura de datos. El tamaño del bulk se escribe en el campo de bits nfetch de este mismo registro.* |
| *[13:7]* | *Banderas: (r)* | *[ap : dp : pl : pp : wp : dc : fc]*  *Consultar el capítulo V. Control de flujo de datos y sincronización externa. Para información más detallada sobre las banderas a continuación.*  *ap: address pending flag.*  *1: Está pendiente por ser enganchada la dirección y tamaño del bulk programado del flujo de datos de entrada.*  *0: No hay pendientes en el momento enganches de dirección de flujos de entrada.*  *dp: data pending flag.*  *1: En el momento hay una transmisión de flujos de datos de entrada y no ha terminado.*  *0: En el momento está pendiente la entrada de flujos de entrada al sistema.*  *pl: parameter loading flag.*  *0: La máquina de estados del adaptador de formato de flujos de entrada se encuentra en el estado Ax.*  *1: La máquina de estados del adaptador de formato de flujos de entrada se encuentra en la mitad de la carga de algún vector, por lo tanto puede estar en cualquier estado menos en Ax.*  *pp: pipeline pending flag.*  *0: No se encuentra en el momento dato alguno transitando entre los circuitos aritméticos ni la interconexión decodificada de circuitos aritméticos en el momento.*  *1: Existen datos transitando en las etapas de los circuitos aritméticos o en la interconexión decodificada de circuitos aritméticos.*  *wp: write on memory pending flag.*  *0: No existen flujos de salida para ser direccionados hacia una dirección en el exterior del RayTrac.*  *1: Hay flujos de salida ya formateados pendientes por ser escritos en una dirección externa al RayTrac.*  *fc: fetch condition flag.*  *0: No existen direcciones de flujos de entrada pendientes por ser enganchadas o la tasa de tránsito de datos por el RayTrac es muy alta y no se deben traer dentro más flujos de datos de entrada.*  *1: Existen direcciones de flujos de datos de entrada pendientes por ser enganchadas y la tasa de tránsito de datos por el RayTrac es baja.*  *dc: drain condition flag.*  *0: No es prioritario escribir flujos de salida, afuera del RayTrac.*  *1: En el momento las condiciones del RayTrac en general indican que es prioritario escribir flujos de salida, afuera del RayTarc.* |
| *14* | *rlsc (rw)* | *0: Una vez se escribe el ultimo flujo de datos de salida entonces no se reinicia el adaptador de formato de flujos de entrada. Esto quiere decir que sin importar en que estado se encuentre la máquina de estados del adaptador de formato de flujos de entrada, conservará su estado.*  *1: Una vez se escribe el último flujo de datos de salida entonces se reiniciaría el adaptador de formato de flujos de entrada. Esto quiere decir que sin importar en que estado se encuentre la máquina de estados del adaptador de formato de flujos de entrada, volverá al estado AX.* |
| *15* | *rom(r)* | *0: El registro se encuentra en modo lectura y escritura donde aplique.*  *1: El registro se encuentra en modo escritura. Esto ocurre cuando el kernel está ejecutando operaciones.* |
| *[30:16]* | *Nfetch (rw)* | *Número de datos que posee el siguiente flujo de entrada. Este valor se encuentra en cero todo el tiempo. Una vez se le asigne un valor distinto a cero, el bit rom se colocará en cero y el sistema comenzará a ingresar los flujos de entrada procesarlos y sus resultados escribirlos. Max Bulk Size=215-1.* |
| *31* | *Irq* | *Una vez se escriba el último dato del último flujo de resultados de salida, este bit se colocará en uno para señalar el evento del fin del procesamiento del flujo.* |

Tabla . Registro de Control y Programación REG\_CTRL.

# Interconexión decodificada de circuitos aritméticos.

Una vez se programa el registro de control con el *kernel* que se quiere ejecutar en el *pipeline aritmético*, se configuran las conexiones de las salidas y las entradas de los circuitos del *pipeline aritmético.* A continuación se detalla la configuración de las interconexiones.

|  |  |
| --- | --- |
| DCS | Circuito Interconectado |
| 011: AxB |  |
| 000 y 001: A+B y A-B |  |
| 110:  |A| y A/|A| |  |
| 100:  A.B |  |
| 111:  A\*B |  |

Tabla . Detalle de los circuitos con la interconexión decodificada, a partir de los bits DCS del registro de control, que contienen el kernel a ejecutar condificado.

# Secuencia de programación, configuración de un kernel y flujos de entrada y salida.

El programador debe seguir la siguiente secuencia de pasos.

1. Escribir en el registro reg\_fecthstart, la dirección de memoria donde se encuentra el flujo de datos de entrada, o sea los parámetros para operar.
2. Escribir la dirección donde se van a guardar los resultados en el registro reg\_sinkstart. El programador es responsable por el segmento de memoria donde se vaya a escribir esté disponible, tenga capacidad suficiente y esté asegurado para evitar posibles *race conditions*.
3. Escribir en el registro de control reg\_ctrl: qué función o *kernel* configurar en el *pipeline aritmético* campo de bits *dsc*, si la función va a retornar resultados vectoriales, escalares o ambos, en el campo de bits *vtsc,* el modo de carga de operandos si es combinatorio o no en el bit *dsc.*
4. Escribir el número de datos a operar o tamaño del bulk, en el registro reg\_ctrl. Si son 2 vectores los que se van a ingresar el número es 6, si son 100 vectores los que se van a operar el número es 300. Tener en cuenta que el formato de los valores almacenados debe ser punto flotante sencillo IEEE754, lo que implica que cada uno ocupa 4 bytes de memoria. Para especificar este valor se debe utilizar el campo de bits nfetch. *El kernel comenzará a procesar datos tan pronto se especifique el bulk size.* Por eso debe ser este valor el último programado en la secuencia.

El siguiente código refleja la secuencia explicada. El objetivo es ejecutar 300 operaciones de producto punto.

*#define reg\_fetchstart 32*

*#define reg\_sinkstart 36*

*#define reg\_ctrl 0*

*#define dotprod 0x6*

*#define vectorial 0x10*

*#define scalar 0x08*

*#define vect\_n\_scalar 0x18*

*#define bulk\_sizer(x) (x<<16)*

*Uint \* ptr\_to\_raytrac = (uint\*) raytrac\_0\_base;*

*Float input\_stream[600];*

*Float output stream[200];*

*Uint index;*

*/\* Crear los vectores a operar \*/*

*for (i=0:i<600;i++)*

*input\_stream[i]=rand(); //Numero al azar entre 0 y 1.*

*/\*Paso Numero 1: Escribir la dirección de flujo de entrada en el registro fetch start \*/*

*ptr\_to\_raytrac[reg\_fetchstart] = input\_stream;*

*/\*Paso Numero 2: Escribir la dirección del flujo de salida en el registro sink start \*/*

*ptr\_to\_raytrac[reg\_sinkstart] = output\_stream;*

*/\*Paso Numero 3: Configurar el kernel y los adaptadores de formato : producto punto, devolviendo valores escalares únicamente \*/*

*ptr\_to\_raytrac[reg\_ctrl] = (dot\_prod+vectorial)<<1;*

*/\*Paso Numero 4: Configurar el bulk size\*/*

*ptr\_to\_raytrac[reg\_ctrl] += bulksizer(300);*

*/\*Paso 5 esperar a que se termine de ejecutar los 300 productos punto.\*/*

*while(¡(ptr\_to\_raytrac[reg\_ctrl] & 0x80000000));*

El ejemplo anterior es una forma muy básica de programar el *RayTrac.* Uno de los productos de este proyecto es una API que haga realmente transparente al usuario la secuencia de programación y se enfoque más en la aplicación que está realizando que en otra cosa.

La última línea hace referencia a un *polling* del bit de interrupción, esto no es una buena práctica de programación. Lo que se debe hacer es utilizar la interfaz de programación del NIOS II, procesador que soporta Interrupciones internas o controlador externo de interrupciones y escribir una *ISR o rutina de servicio de interrupción.*

En el capitulo diez que describe la implementación del trazador de rayos hay ejemplos de código, sin embargo junto con el software driver del RayTrac se añaden ejemplos. Driver y ejemplos son anexos a este documento.

1. CONTROL DE FLUJO DE DATOS Y SINCRONIZACION EXTERNA

# Descripción detallada del mecanismo de sincronización.

El mecanismo de sincronización está encargado de generar las señales que lee la máquina de estados (ver capítulo, máquina de estado), para que esta genere la lectura y escritura de los diferentes elementos que componen el RayTrac.

En él se describe, como cambia el pipeline aritmético en su interconexión dependiendo de la instrucción que se va a realizar. De igual manera el resultado de diferentes operaciones, no se guarda en una misma cola de resultados, sino en una cola dependiendo de la operación que se lleve a cabo.

En el capítulo Datapath Control, se asignó el “opcode” de cada operación que se lleva a cabo y ese “opcode” es la entrada para el circuito combinatorio que realiza la interconexión del pipe aritmético. Ahí se analizó como dependiendo de las operaciones el pipe puede cambiar de longitud y de elementos de procesamiento o etapas.

Deteniendo ó permitiendo el avance de los datos a través de un pipe

Tomemos por ejemplo el “opcode” x, para llevar a cabo una operación de producto cruz. El pipe tiene una longitud de 12 etapas y los componentes del producto cruz se escriben en las colas denominadas RES1, RES2 y RES3.

Se podría pensar que en el momento que se llenen estas colas –las 3 de manera simultánea-, la sincronización se puede llevar a cabo mediante una señal de “parada” que se propague por todas las señales de todos los registros que componen las 12 etapas del pipe.

Una señal de sincronización a partir del estado “full” de la cola es valido como criterio funcional, pero es costoso en términos de recursos. La primera etapa en el pipe del producto cruz es la primera etapa de la operación producto flotante y en su entrada contiene 64 registros, para 2 factores, cada uno de 32 bits. Dado que la operación de producto cruz requiere 6 multiplicadores en realidad son 384 registros a la entrada del pipe –solo la entrada- y aun falta otras 11 etapas para terminar la operación. Así que pensar en una señal de enable que sirva de freno para todo el pipeline aritmético es bastante engorroso, no solo por la cantidad de señales, los recursos de interconectividad necesarios, sino además el *fan out* necesitado para distribuir la señal.

El enfoque alternativo y económico es generar una señal de sincronización en la cola de resultados que indique cuando quedan 12 espacios disponibles y aun no se haya comenzado a generar ciclos de lectura de la cola de resultados. Esa sincronización genera *una única* señal de parada que se distribuye por el pipe aritmético, pero en varios ciclos de reloj.

Para entender a qué hace referencia “distribución de la señal de parada pero en varios ciclos de reloj”, se debe pensar en la manera que se pensaba distribuir la señal enable por todo el pipe aritmético: en un mismo ciclo de reloj se iban a cancelar o a deshabilitar el paso desde los puertos *d* a *q* a lo largo y ancho del pipeline aritmético, lo que se determinó como inconveniente. Entonces la distribución de una señal de parada es comenzar a deshabilitar la posibilidad de lectura progresivamente, al mismo ritmo que avanza una operación en particular oi en el pipe.

Entonces si el pipe tiene una longitud de 12 etapas, solo aquellos resultados que en ese momento se encuentran en dichas etapas alcanzarán a ser escritos dentro de la cola de resultados, en el peor de los casos, que es el evento en que no se realicen ciclos de lectura de la cola de resultados desde el exterior. Qué pasará en el ciclo número 13 después de generada la señal de cancelación? No se escribe en la cola de resultados, puesto que esta se encuentra llena.

El párrafo anterior describe el criterio de sincronización, o sea, de donde se va a sacar la información para saber cuándo se genera el evento de sincronización. Pero, cómo se implementa? La respuesta es una cadena de sincronización paralela al pipe:

Esta cadena de sincronización tiene un ancho de 1 bit y corre en paralelo al pipe aritmético. Al final de la cadena de sincronización, el ultimo q*i* va conectado a la señal de escritura de la cola de resultados. En el caso del producto punto va conectada a la señal de escritura de las 3 colas de resultados RES1, RES2 y RES3.

A continuación se describe al compas de los eventos que la definen, el funcionamiento de la cadena de sincronización:

Evento: Inicio del RayTrac asincrónico, más conocido como reset.

En el momento que se realiza un rst general asincrónico del RayTrac, todos los qi de la cadena de sincronización estarán en valor 0, por lo tanto la escritura en RES1, RES2 y RES3 estará deshabilitada.

Evento: Comienzo de la operación.

En el momento que la cola de resultados no se encuentre llena y dentro de ella queden más espacios que etapas en el pipe y exista una operación pendiente o por comenzar, se genera un evento de sincronización que genera un 1 en la entrada de la cadena de sincronización y con él, de manera paralela, a la entrada del pipeline aritmético, los operandos a operar.

Evento: Ejecución de escritura en la cola de resultados una vez han transcurrido 12 ciclos de reloj después del comienzo de la operación.

Conforme corren los ciclos de reloj los operandos se van ejecutando a lo largo del pipe, al mismo tiempo y ritmo, la cadena de sincronización va propagando, etapa tras etapa, el valor lógico 1.

En la última etapa del pipe, en la etapa 12, se obtendrá el valor de los 3 componentes del producto punto y estos estarán conectados a las entradas de datos de las colas RES1, RES2 y RES3. Pero hasta ese momento el valor de la señal de escritura de esas colas había sido 0: en ese momento comienza la señal de escritura de las 3 entradas a ser 1, lo cual es coherente puesto que ya existe un valor valido para escribir.

Evento: Generación de la señal de parada una vez quede menos espacio en la cola de resultados que la longitud del pipe.

La cola de resultados posee una capacidad de 512 x 32 bits. En el caso de la operación producto cruz, se debe tener en cuenta que su longitud es de 12 etapas de pipe. En el momento que se ha escrito 500 resultados en la cola de resultados y no se haya leído uno solo de estos, la cola tendrá espacio para 12 valores mas.

En el peor de los casos, esos 12 espacios, serán ocupados por los valores que se encuentran en proceso en las 12 etapas del pipe aritmético y adicionalmente no habrá ninguna petición de lectura por parte del bus externo. Dada la condición anterior, la escritura en la cola de resultados se debe detener una vez los 12 valores se hayan almacenado.

En general y para cualquier operación, cómo se generaría esa señal de “parada”, qué condiciones presenta ese evento? Son 2 las condiciones y entre ellas inclusivas (and):

* No existe una petición de lectura a la cola de resultados desde el bus externo y…
* Quedan tantos espacios o menos como la longitud en etapas que posea el pipe aritmético para una operación en particular.

Una vez se presentan estas dos condiciones el evento de “parada” se ha generado y se debe asegurar un 0 lógico a la entrada de la cadena de sincronización. En ese momento el RayTrac entra en estado IDLE.

El estado IDLE se suspenderá cuando una de las dos condiciones del evento “parada” no se cumpla, por ejemplo, si no hay peticiones de lectura, pero el espacio disponible es mayor que la longitud en etapas de pipe aritmético para la operación en particular significa que podremos almacenar todos los valores que en ese momento se encuentren en el pipe aritmético, restableciendo el servicio de escritura en la cola de resultados, colocando un 1 lógico en la entrada de la cadena de sincronización.

Evento: Finalización de las operaciones descritas en la instrucción.

La finalización de las operaciones descritas en una instrucción están marcadas en el momento en que los 2 contadores de direcciones han llegado a las direcciones de la instrucción (ver capítulo de interfase de usuario). Una vez esa condición se ha generado, se revisa la siguiente instrucción se revisa la siguiente instrucción. Si el *opcode* de la siguiente instrucción es el mismo de la que se ejecuta, entonces se comienzan a leer los nuevos operandos y en general el pipe aritmético y la cola de resultados no se deben ver afectados por esto. Si se trata de un *opcode* diferente entonces se debe esperar a que se vacíe el pipeline aritmético, dado que la interconexión del pipe puede cambiar con la nueva instrucción. A continuación se ilustra lo anterior:

Descripción de la cadena de sincronización:

La cadena de sincronización contiene 27 etapas. Esto es porque la operación de normalización toma 27 etapas dentro del pipe aritmético para ser ejecutada y es la más extensa de todas.

Solo existe una cadena de sincronización dentro del RayTrac. Sin embargo la mayoría de las operaciones no utilizan la cadena en su totalidad, por ejemplo, la operación producto cruz utilizará 12 etapas de la cadena de sincronización, las 15 etapas posteriores son sencillamente ignoradas.

Se puede pensar en la cadena de sincronización como un indicador de dato valido en la etapa de pipe que corresponda, cuando el valor en la cadena es 1 ó invalido cuando este dato es cero:

Especificación del mecanismo de sincronización por operación:

Las ilustraciones a continuación evidencian cómo quedan interconectados el pipeline aritmético, la cola de resultados y la cadena de sincronización, según la operación que se esté realizando:

1. SIMULACIÓN, VERIFICACIÓN Y VALIDACIÓN FUNCIONAL Y ANÁLISIS DE TIEMPOS

# Simulación Funcional.

La verificación funcional del circuito se va a llevar a cabo haciendo uso de un conjunto de simulaciones. Este conjunto de simulaciones tiene por entrada un escenario 3D y por salida se fabricara un archivo JPEG, que representa un *render* de la escena de entrada.

Se simulará la generación de la imagen de un escenario 3D. En el escenario se encuentra un conjunto de entidades geométricas: un plano de proyección de 16x16 pixeles de resolución, esferas de radio y posición arbitraria y adicionalmente existirá en el escenario una fuente de luz omnidireccional. Adicionalmente habrá una cámara que representa la dirección y posición del observador que otea la escena.

El conjunto de simulaciones a realizar:

1. Normalización de 16x16 rayos que salen desde un observador al plano de proyección ergo existe un rayo por cada pixel del plano de proyección.
2. Cálculo de intersección entre los rayos mencionados y varias esferas en un escenario arbitrario.
3. Cálculo de la intensidad de la luz sobre los puntos donde los rayos y las superficies de las esferas se intersectan.
   1. Cálculo de las normales de las esferas en los puntos de intersección
   2. Cálculo de los vectores de incidencia de una luz puntual omnidireccional sobre los puntos de intersección.
4. Cálculo de la intensidad de la luz omnidireccional sobre los puntos de intersección haciendo uso de la operación producto punto entre las normales y los vectores hallados en 3a y 3b.

# Metodología de Simulación.

Las simulaciones se llevan a cabo con la entrada y salida de archivos, hacia y desde un *testbench*, el cual es ejecutado por el programa *MODELSIM Altera Starter Edition*.

Adicionalmente se crea una herramienta utilizando el lenguaje de programación PYTHON, cuyo objetivo es analizar los códigos fuente de los RTL que describen el RAYTRAC y compilar el código VHDL con el *testbench*.

Para generar la entrada de la simulación, la cual es una escena en 3D, se utiliza el programa Blender3D, en el que se puede modelar esta escena con esferas y mediante un script se genera un archivo de entradas para la simulación.

Para visualizar los resultados de la simulación se utilizan 2 herramientas:

1. Posterior a la ejecución del *testbench*, en un archivo CSV (*Comma Separated Values*) de valores separados por comas, se escriben los resultados de la simulación. Las filas poseerán señales o puertos de observación y las columnas avanzan hacia la derecha conforme la simulación avanza.
2. La ejecución del *testbench* adicionalmente genera un archivo gráfico JPEG o PNG, a partir de la información generada en la simulación 3c.

Para la generación de esta simulación funcional se escriben 3 herramientas concretas:

1. Script de generación de entradas: Este script corre en el ambiente Blender3D, está hecho en PYTHON y su objetivo es generar a partir de un escenario 3D las entradas o estímulos para la simulación.
2. Script de generación de *testbench*: Cuando se realiza una simulación funcional haciendo uso de un *testbench* es usual encontrar dos entidades en el *testbench*, la primera es la DUT (*Device Under Test*) o Dispositivo Bajo Test y la segunda es la entidad que controla y genera los estímulos o entradas para la simulación. El problema que surge es que la entidad que controla los estímulos solo puede acceder a los puertos de entrada y salida de la DUT, para escribirlos y leerlos respectivamente. Dado que la DUT es por lo general la entidad top en el circuito entero, las señales y puertos que se encuentran más abajo dejan de ser visibles para la simulación, lo que supone un problema para entender los posibles errores de diseño que surjan en aquellas entidades *bottom*. El script de generación *testbench* analiza todas las entidades instanciadas en el top *entity* y mediante reglas de comentarios (al estilo *doxygen*), detecta señales que el diseñador desea observar en la simulación y modifica los scripts de tal manera que esa señal sea visible en los puertos del top *entity* y de esta manera hacerlos observables para la entidad que controla y genera las entradas para la simulación.
3. El *testbench* compilado adicionalmente genera instrucciones para que en la ejecución de la simulación se genere un archivo con la imagen del *render*.

# Análisis de Resultados.

Los resultados de las simulaciones se anexan y se publican en la

Escenas Generadas

# Análisis de Tiempos.

Se realizó el análisis de tiempos para el RayTrac. El sistema posee un solo dominio de reloj de una frecuencia de 50 MHz, lo que simplifica el cuidado que se debe tener en el momento en que se realiza el diseño de los circuitos y la definición de restricciones o *constraints,* al momento de realizar el análisis de tiempos.

Sin embargo los circuitos externos al RayTrac, funcionan en un dominio de reloj de 100 MHz, por lo tanto en el capítulo de integración con los sistemas externos se realizará el análisis de tiempos, con restricciones y dominios de reloj adicionales.

Los reportes de tiempos fueron generados una vez realizada la compilación y el *fitting* del circuito en *Quartus* II.

El reporte de frecuencia máxima da por resultado 137.61 MHz. Este resultado indica que el periodo mínimo que puede tener el reloj del sistema es de 7.2 ns, tiempo satisfactorio para un sistema que se diseñó pensando para un reloj de 50 MHz cuyo periodo es de 20 ns.

El reporte de restricciones y *fmax* arrojado por la herramienta *TimeQuest* es el siguiente:

|  |  |  |
| --- | --- | --- |
| Fmax | Restricted Fmax | Clock Name |
| 137.61 MHz | 137.61 MHz | clko |

Tabla . Frecuencia máxima calculada para el RayTrac.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SDC Command | Name | Period | Waveform | Targets |
| create\_clock | clko | 20 | { 0.000 10.000 } | [get\_ports {clk}] |

Tabla . Restricciones creadas para el análisis de tiempo posterior a la operación de fitting que realiza Quartus II, que sirve como entrada para el TimeQuest Analyzer.

1. DESCRIPCION GENERAL DEL RTENGINE

Descripción Gral del RtEngine. Diagrama de Bloques. Ejemplos de posibles aplicaciones. Descripción breve de la aplicación a implementar.

El RtEngine es una plataforma de Hardware y Software, para desarrollar aplicaciones que basadas en matemática vectorial principalmente. El diseño e implementación de la plataforma del RtEngine es dominio de este trabajo. Adicionalmente el diseño de una aplicación que utiliza la plataforma es también parte del alcance de este trabajo, en particular, la aplicación propuesta dentro de los objetivos es la implementación de un sistema de síntesis de imágenes a partir de una técnica de *render* denominada *Ray Tracing* o en español Trazado de Rayos.

# Plataforma RtEngine

La plataforma del RtEngine tiene dos componentes principales, un componente hardware y componente software.

## Especificaciones Hardware

El *RtEngine* está implementado por una serie de componentes electrónicos, que en conjunto son programables y que para ello cuentan con interfaces de entrada y salida.

|  |  |  |
| --- | --- | --- |
| Tipo de Componente | Componente | Descripción |
| Memory | 32 Mbytes DDR SDRAM | Memoria para el uso general de las aplicaciones y para implementación de *DMA, Direct Memory Access,* para el Display y para la interfaz de comunicación TSE. |
| 1 Mbyte Synchronous SSRAM | Memoria de uso general, para el espacio de datos de las aplicaciones etc. |
| 16 Mbytes External Flash | Memoria de uso general, especialmente para el almacenamiento de instrucciones, sistema operativo, bootloaders. Etc. |
| Clocking and PLLs | 50 Mhz on board clock | Clock que viene con la tarjeta de desarrollo. |
| Altera Avalon PLL generated clock 100 Mhz | Clocks generados mediante PLL dentro de la FPGA 3C25F324C6 |
| Altera Avalon PLL generated clock 60 Mhz |
| Altera Avalon PLL generated clock 40 Mhz |
| DDR SDRAM generated clock 66.5 Mhz, 133.0 Mhz | Reloj generado por la memoria DDR SDRAM |
| Switches and Indicators | 4 User Controlled Buttons | Botones de uso general |
| 4 User Controlled Leds | Leds de uso general |
| Display | 800x480 LCD Touch Display | Display de hasta 24 bits por pixel e interfase Touch Screen. |
| Interfases de Comunicación | JTAG USB Cable | Cable para la programación del RtEngine y depuración. |
| RS232 Serial Interfaz | Interfase Serial. Hasta 115200 bps (baudios por segundo). |
| Triple Speed Ethernet, Network Adapter Interfaz | Interfase Ethernet 802.3, 10/100 mbps. |

Tabla . Componentes de Hardware Disponibles en el RtEngine.

ANEXO A: GLOSARIO DE SIGLAS EN ESTE DOCUMENTO.

*PA : Pipeline Aritmético.*

*IDC: Interconexión decodificada de circuitos.*

*PF: Procesador de Flujos.*

*AFF: Adaptador de formato de flujo.*

*MECF: Máquina de estados para el control de flujo.*

*BREG: Bloque de registros de control y programación.*

1. TIM PURCELL, Ray Tracing on a Stream Processor. [↑](#footnote-ref-1)
2. Tim Purcell. [↑](#footnote-ref-2)
3. **http://opencores.org/websvn,filedetails?repname=raytrac&path=%2Fraytrac%2Fbranches%2Ffp%2Fraytrac.vhd** [↑](#footnote-ref-3)
4. # IEEE Standard 754 Floating Point Numbers [Steve Hollasch](mailto:steve@hollasch.net?Subject=IEEE%20Floating%20Point%20Page)

   [↑](#footnote-ref-4)