Introducción.

En el 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.

En el 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.

En el Capítulo Tercero, se encuentran detallados los aspectos de la arquitectura del *pipeline* aritmético

En el Capítulo Cuarto, se encuentran detallados el controlador de camino de datos y el sistema de sincronización del RayTrac

El Capítulo Quinto, describe la arquitectura de la interfase de programación implementada para realizar aplicaciones sobre el Rt Engine. Y la máquina de estados que controla el RayTrac. Para poder implementar un programa en el sistema se necesita de una capa de abstracción de hardware y encima de ella un sistema operativo, kernel o programa de control, etc. La capa de abstracción de hardware es definitiva para la interfase de programación y dicha abstracción se realiza plenamente en el procedimiento que se describe en el Capítulo Séptimo.

El Capítulo Sexto, describe la implementación de pruebas del Rt Engine, los resultados obtenidos, principalmente en términos de desempeño y como este desempeño se encuentra relacionado con la arquitectura del sistema.

En el Capítulo Séptimo, se encuentra detallado el proceso de uso de la *SOPC Builder,* para interconectar los componentes del RtEngine, mediante el bus Avalon de ALTERA. La salida del proceso descrito en este capítulo, es la entrada que utiliza el compilador de ALTERA para hacer la síntesis y *fitting* del sistema en la FPGA física. La información contenida en los archivos que se producen, tiene entre muchas otras cosas, por ejemplo, los encabezados para la programación del sistema, que es la esencia de la capa de abstracción de hardware, o cómo sus siglas en inglés lo indican *HAL (hardware abstraction layer).*

El Capítulo Octavo, 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*.

En el Capítulo Noveno, se exponen 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.

*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 Tecnologías a usar en el desarrollo.*

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. Capítulo 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 (Unidad Aritmética Vectorial). Es similar a una ALU regular pero especializada en vectores y en ella se implementan operaciones como, productos punto, productos cruz, productos escalares, suma, resta y normalización de vectores.

# Descripción general y arquitectura.

El RayTrac se compone de los siguientes elementos:

* Bloque de registros de operandos de entrada.
* Cola de instrucciones de entrada.
* Colas de resultados de salida.
* Pipeline aritmético.
* Controlador de camino de datos y sincronización.
* Máquina de estados e interfase de programación.

El *bloque de registros de operandos de entrada*, es una memoria de acceso aleatorio, a la que se escribe datos desde fuera del RayTrac y se lee internamente por el RayTrac. Esta memoria posee los valores de los operandos de las diferentes operaciones vectoriales a ejecutar.

La *cola de instrucciones de entrada*, es una memoria configurada en modo SCIFIFO\*\* en la que se guarda las instrucciones que se van a realizar con los operandos almacenados en el bloque de registros de operandos. Se escribe externamente y se lee internamente por el RayTrac.

Las *colas de resultados de salida*, son las colas donde se almacenan los valores de los resultados de las operaciones vectoriales ejecutadas en la unidad. Se escriben internamente los datos y externamente estos son leídos.

Los 3 elementos hasta ahora brevemente descritos se encuentran detallados y especificados en el capítulo Bloques de Memoria de I/O.

El *pipeline aritmético* es el circuito que implementa las operaciones aritméticas para llevar a cabo las instrucciones leídas en la cola de instrucciones. Se compone de circuitos que llevan a cabo las operaciones de multiplicación, suma, resta, raíz cuadrada e inversión y de manera compuesta operaciones vectoriales más complejas como producto punto, producto cruz, normalización, cálculo de magnitud, etc. Las entradas y salidas de este pipe están restringidas al interior del RayTrac y no son accesibles directamente desde el exterior. La especificación y detalle del pipeline aritmético se explica en el capítulo pipeline aritmético.

El *controlador de camino de datos y sincronización* es un circuito combinatorio encargado de interconectar los componentes que se encuentran en el pipeline aritmético, por ejemplo, interconecta los operandos, con los circuitos de suma y la cola de resultados cuando la operación es una suma o interconecta los multiplicadores y la salida de estos a los bloques de suma y después a la cola de resultados. Este circuito también se encarga de sincronizar la escritura y la lectura de los diferentes elementos de memoria que posee el RayTrac, lo hace a través de una cadena de sincronización compuesta de una secuencia de registros, que señalan en que etapa del pipe se encuentra un dato valido para lectura o para escritura.

La *máquina de estados e interfase de programación* es un circuito que controla y abstrae en hardware el mecanismo de programación del RayTrac, que hace de este un procesador, un elemento programable. Básicamente este elemento lee las instrucciones del bloque de registros de operandos de entrada y genera las interrupciones para señalizar cuando el RayTrac está listo para ser leído y/o escrito. Posee dos estado FlushToNextInstruction y ExecuteInstruction.

Es un procesador.

El RayTrac no es más que 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. El fetch está a cargo de la máquina de estados e interfase de programación en conjunto con la cola de instrucciones de entrada y el bloque de registros de operandos de entrada. El decode lo ejecuta el controlador de camino de datos y sincronización. El execute lo ejecuta el pipeline aritmético y el write lo ejecuta la máquina de estados e interfase de programación en conjunto con la cola de resultados de salida.

Por lo tanto el RayTrac es un elemento programable y para ser usado requiere que se codifiquen instrucciones: o sea se deben escribir pequeños fragmentos de código poder explotar su funcionalidad.

# Diagrama de Bloques del Sistema

memblock.vhd

arithblock.vhd

sqrt32.vhd

invr32.vhd

memblock.vhd

fadd32.vhd.vhd

fmul32.vhd

sm.vhd

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

El RayTrac posee una variedad de bloques funcionales que cumplen pequeñas tareas, muy especializadas. Si se mira al nivel de etapas combinatorias, separadas por registros, se encuentran funciones atómicas que ejecutan etapas de una operación en punto flotante. Sin embargo al mirar al nivel aritmético –un poco más “arriba” en la abstracción- encontramos sumadores, multiplicadores, etc., que son circuitos en pipeline de varias etapas. Sin embargo sus salidas y entradas pueden estar a la entrada y salida de uno o varios multiplexores, esto con el fin de crear distintos caminos de datos (datapaths), según la operación que se desee ejecutar: es por eso que en la ilustración xxxx, entre los bloques funcionales del pipeline aritmético encontremos el controlador de caminos de datos.

La unidad, como tal se puede ver como una línea de producción en *serie*: la materia prima, es clasificada, transportada, procesada y distribuida 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, si la cola de resultados se llena, el pipeline aritmético debe suspender su funcionamiento y la máquina de estados debe quedar suspendida y no leer más operaciones del bloque de registros de operandos de entrada ni de la cola de instrucciones de entrada. Así mismo si no hay materia prima en la línea, o sea, no hay instrucciones en la cola de instrucciones, en RayTrac debe “parar”, Es por esto que en el RayTrac se implementan mecanismos de sincronización, pero la sincronización puede ser muy costosa en términos de recursos de silicio utilizado y por ello debe ser una sincronización sencilla, confiable y que incluso sacrifique el desempeño del sistema en general.

La sincronización no está completa si no existe un mecanismo que señale la disponibilidad de datos validos en la salida del RayTrac. Por ello el sistema genera interrupciones. Pero también se debe señalar que el sistema esta congestionado y no puede seguir operando hasta que externamente se realice una lectura de los datos disponibles. En ese caso se puede generar una interrupción distinta que los dispositivos externos al RayTrac atiendan con mayor prioridad.

# Características funcionales del sistema.

* Instrucciones SIMD para vectores en punto flotante:
  + Producto Punto.
  + Producto Cruz.
  + Suma.
  + Resta.
  + Magnitud de vector.
  + Normalización de vector.
  + Multiplicación por un valor escalar.
* Arquitectura en pipeline, que permite hasta 100 millones de cálculos de producto punto por segundo y hasta 50 millones en las otras instrucciones.
* Interfase con bus *Avalon Altera*.\*\*.
* Cola de instrucciones, de hasta 32 instrucciones de 32 bits.
* 4 Bloque de registros de operandos para almacenar hasta 512 vectores, de 3 componentes, de 32 bits cada uno, en formato punto flotante.
* Mecanismos de sincronización mediante interrupciones y los mecanismos automáticos que provee el bus Avalon Altera.\*\*.
* API en C, para programar sobre el NIOS II.

En resumen, el sistema funciona como una línea de procesamiento de vectores en serie, programable.

1. Capítulo 2, Bloques de Memoria de Entrada y Salida.

La unidad aritmética vectorial en punto flotante, RayTrac tiene una interfase de entrada y salida fundamental, los bloques de memoria.

Son 3 los elementos de memoria que posee el RayTrac:

* Bloque de registros de operandos de entrada.
* Cola de instrucciones de entrada.
* Colas de resultados de salida.

En este capítulo se detalla los tipos de memoria y la implementación realizada. La importancia de estos bloques es que fundamentan la manera como el RayTrac se comunica con el resto del Rt Engnine. Estas memorias no son mas que los bloques fundamentales de la interfase de entrada y salida del RayTrac.

La organización de la entrada y salida, crea un concepto simple para la aplicación que utilice el RayTrac como recurso de computo: puesto que las memorias tienen que tener un rango de direcciones definido, resulta muy fácil especificar en qué direcciones se escriben los operandos o las entradas al sistema y en qué direcciones se leen los resultados obtenidos. Es fácil, sencillo y cualquier usuario desarrollador de aplicaciones, novato o experto encontrará simple este mecanismo.

Descripción de los bloques de memoria.

Bloques de registros de operandos.

Los bloques de registros de operandos de entrada, se componen de 12 bloques de memoria de acceso aleatorio, cada bloque se llama bloque de componentes vectoriales. Cada bloque de componentes cuenta con 512 palabras de 32 bits de ancho y con una interfase independiente de escritura y una interfase independiente de lectura:

0 [31..0]

1 [31..0]

2 [31..0]

3 [31..0]

4 [31..0]

5 [31..0]

.

.

.

MEMORY

BLOCK

.

.

.

.

.

.

509 [31..0]

510 [31..0]

511 [31..0]

WRITE INTERFACE

data[31..0]

add[8..0]

clk

ena

READ

INTERFACE

q[31..0]

add[8..0]

clk

ena

**Ilustración 2. Memoria de interfase dual, una de escritura y otra de lectura. 512x32 bits. Archivo RTL memblock.vhd.** [[2]](#footnote-2)

Al agrupar en conjuntos de 3 bloques de componentes, se obtiene la abstracción de 4 bloques vectoriales de 512 vectores cada uno:

AZ

BZ

CZ

DZ

AY

BY

CY

DY

AX

BX

CX

DX

Ilustración . Bloque de memorias de Operandos de Entrada. Archivo RTL: memblock.vhd

En estos registros de entrada solo podrán escribir datos los dispositivos externos programables, pero no leerlos. Los dispositivos internos del RayTrac pueden leer los datos consignados en estos, pero no escribirlos.

AZ

BZ

CZ

DZ

AY

BY

CY

DY

AX

BX

CX

DX

CompSel / Blocksel decoder

EXTERNAL ADDRESS

EXTERNAL BUS DATA

Ilustración 4. Escritura de Operandos de Entrada desde la interfase externa Avalon. Archivo RTL: memblock.vhd.

En el capítulo de interfase de programación se detalla las reglas de escritura de operandos, según la operación que se desee ejecutar. Sin embargo es importante entender que en estos bloques de memoria, se alojarán *única y exclusivamente* los *operandos*, de cada instrucción particular.

Las instrucciones que ingresan en el RayTrac están codificadas de tal manera que se abstrae el nombre de los vectores operados como A, B, C y D. Por ejemplo una instrucción puede ser sumar el vector A con el vector B. Los detalles y la manera cómo se encuentran codificadas las instrucciones se explicarán profundamente en el capítulo de interfase de programación.

AZ

BZ

CZ

DZ

AY

BY

CY

DY

AX

BX

CX

DX

CompSel

BlockSel

INTERNAL RAYTRAC DATA

Ilustración . Interfase de lectura de operandos desde el RayTrac. Archivo RTL: memblock.vhd

La Ilustración 4, muestra el mecanismo de decodificación para seleccionar la salida del bloque de operandos que se va a leer.

Se tiene entonces el bloque A de vectores, conformado por los bloques 0 a 2 del bloque de registros de operandos, el grupo B por los bloques 3 a 5, el grupo C por los bloques 6 a 8 y el grupo D por los bloques 9 a 11.

La máquina de estados que controla la lectura de los registros de operandos y la lógica que escribe en ellos, deben seguir las siguientes reglas de direccionamiento:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BlockSel | | CompSel | | InBlockAddress | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

Tabla 1. 13 Bits de direccionamiento interno.

|  |  |  |  |
| --- | --- | --- | --- |
| **BlockSel /CompSel** | **00 – Z** | **01 – Y** | **10 – X** |
| **00 – Grupo A** | Az | Ay | Ax |
| **01 – Grupo B** | Bz | By | Bx |
| **10 – Grupo C** | Cz | Cy | Cx |
| **11 – Grupo D** | Dz | Dy | Dx |

Tabla 2. Organización de los operandos que entran al RayTrac por bloques

Los 9 bits correspondientes al bloque InBlockAddress direccionan una de las 512 palabras de un bloque de memoria en particular. Para escribir en una dirección de memoria específica se debe seleccionar en qué bloque de memoria, para esto se usa los campos BlockSel y CompSel. BlockSel selecciona uno de los 4 grupos A, B, C ó D y CompSel selecciona el componente en el que se va a escribir, X (“00”), Y(“01”) ó Z(“10”).

Cuando Compsel vale “11”, sirve para señalar que se planea escribir en la cola de instrucciones, al mismo tiempo que hace que los campos InBlockAddress y BlockSel sean ignorados.

*Especificaciones de la memoria de operandos*

Cola de Instrucciones.

La cola de instrucciones de entrada, como es natural, posee una única entrada y una única salida. Haciendo uso del mismo esquema de direccionamiento del bloque de registros de operandos de entrada se puede escribir datos en esta cola, al seleccionar en el campo Compsel el valor “11”.

Los datos dentro de la cola tienen codificadas las instrucciones que se quieren ejecutar en el RayTrac. La manera en que se encuentran codificadas está descrita y especificada en el capítulo interfase de programación.

El RayTrac lee internamente estas instrucciones y con ellas decodifica la manera en que se interconectarán los distintos elementos del pipeline aritmético y la dirección en la que se en encuentran los operandos de la instrucción, dentro de la memoria de operandos.

ILUSTRACIÓN DEL PARRAFO ANTERIOR AQUÍ.

El diagrama de bloques que describe el mecanismo de escritura y de lectura de datos dentro de la cola se describe a continuación.

ILUSTRACIÓN COLA DE INSTRUCCIONES.

*Especificación de cola de instrucciones.*

La cola de instrucciones viene provee mecanismos de sincronización. La cola cuenta con una señal “empty” que se activa justo cuando no hay elementos/instrucciones dentro de la cola indicándole a la lógica de control en el RayTrac que debe permanecer ociosa y a la espera de nuevas instrucciones.

La sincronización inicial de la lógica del RayTrac, ocurre cuando se escribe una instrucción en la cola de instrucciones. La lógica del control del RayTrac depende y se conecta a estas señales de sincronización para detectar el momento en el que debe “arrancar”.

Además de las señales de sincronización hay otra característica importante, que ayuda a simplificar la lógica de control en el RayTrac: la cola de instrucciones está configurada en modo “*show ahead*”, esto permite que en la salida de la cola *siempre* se encuentre el primer elemento que entró en ella, sin necesidad de colocar un ´1´ lógico en la señal de lectura *rd* de la cola.

El sentido funcional de la señal rd es la de un *read acknowledge,* cuyo propósito es “avisar” a la cola que el dato que se encuentra ya fue leído y puede ser descartado en el siguiente ciclo de reloj. Después de descartar el dato, si existen datos adicionales en la cola será el siguiente dato el que aparecerá en la salida, de lo contrario se mantiene el dato en la salida, pero se debe interpretar como un dato invalido puesto que la señal *empty* se actualizará al valor lógico ´1´.

Colas de Resultados.

Las colas de resultados cumplen con la función de almacenar los resultados de las instrucciones que se operan dentro del RayTrac. Son 8 colas cada una con dirección propia y almacenan en particular algún componente del resultado de la instrucción operada en el pipeline aritmético:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Base+ | OPERACIONES | | | | | |
| RES0 | 0 |  |  |  | |A| |  |  |
| RES1 | 256 |  | A+-Bi | AxBi |  |  |  |
| RES2 | 512 | A.B | A+-Bj | AxBj |  |  |  |
| RES3 | 768 |  | A+-Bk | AxBk |  |  |  |
| RES4 | 1024 | C.D |  |  |  |  |  |
| RES5 | 1280 |  |  |  |  | A/|A|i | kCi |
| RES6 | 1536 |  |  |  |  | A/|A|j | kCj |
| RES7 | 1792 |  |  |  |  | A/|A|k | kCk |

Tabla 3. Direcciones de lectura de las colas de resultados por dirección base y por operación.

Por ejemplo en la cola RES0 se almacenan los resultados de la instrucción de cálculo de magnitud de vectores. Si la operación que se está realizando es la normalización de un vector entonces se debe leer las colas RES5, RES6 y RES7.

A la salida de las colas de resultados, los datos que salen se encuentran conectados a un decodificador de dirección que selecciona de qué cola se debe leer. La dirección de lectura de las colas de resultados es una entrada al RayTrac, por lo tanto es responsabilidad del dispositivo que direcciona sobre el bus externo leer desde las direcciones adecuadas.

Al igual que en la cola de instrucciones, las colas de resultados están configuradas en modo *showahead*.

Esto es conveniente, porque desde el bus de direcciones externo hace falta codificar una de las ocho direcciones de la cola cuyo dato se desea leer en el bus externo de datos, lo cual tomaría un ciclo de reloj. Si no se configuran las colas de resultados en modo *showahead* entonces se deberían emplear 2 ciclos de reloj para realizar la lectura de un dato en particular: un ciclo para sacar los datos de una cola en particular y otro para codificar la conexión entre la cola escogida para ser leída y el bus externo.

Tener 2 ciclos de reloj para leer acarrea la implementación de un pipe de lectura de datos, lo cual no es deseable porque utiliza recursos extra del FPGA. Alternativamente se podría implementar la lectura de un dato cada 2 ciclos de reloj, lo que tampoco es deseable porque disminuye a la mitad la velocidad general del dispositivo.

Las colas de resultados cuentan con una señal *almost\_full*, donde le indican a los dispositivos de control externos que se encuentran casi llenas y los datos con resultados deben ser leídos y descartados.

El mecanismo de sincronización detiene el funcionamiento del RayTrac cuando se genera un evento de cola llena: detiene el pipeline aritmético y la lectura de la memoria de operandos. La cola de resultados es el extremo por el que salen los datos del pipeline aritmético, si este extremo se obstruye con datos correspondientes a resultados que no se leen, entonces todo el pipe se obstruye y se debe detener su funcionamiento a fin de que no ocurra perdida de datos.

Otros bloques.

Existen otros bloques de memoria implementados en el RayTrac, que no están relacionados con la entrada y salida de datos. Estos bloques sirven para calcular resultados intermedios en las operaciones Raíz Cuadrada e Inversión y proveen mecanismos de sincronización para la operación Producto Punto.

Para más información sobre estos bloques revisar el capítulo 3, Pipeline Aritmético.

1. Capítulo 3, 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 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.

Estas operaciones suelen tener uno o dos operandos de entrada y un resultado. Sin embargo este no es el caso del pipeline aritmético, puesto que a este llegan 12 operandos desde la salida de cada bloque del registro de operandos de entrada.

La razón para tener mas de 2 entradas es obvia: se ejecutan varias etapas en paralelo. El objetivo de la ejecución en paralelo es obtener desempeño.

Una arquitectura en paralelo implica varias “líneas” de pipe en paralelo. Implica también más silicio porque ya no es una sola suma, resta, multiplicación, etc., que se desea ejecutar: ahora son varias multiplicaciones (hasta 6 multiplicaciones al mismo tiempo) ó sumas ó restas.

# Arquitectura del Pipeline Aritmético.

El pipeline cuenta con 12 entradas de operandos, que entran en 6 pipes distintos, 2 operandos por cada pipe.

Desde la primera etapa combinatoria los datos que entran y salen de las etapas combinatorias, son propagadas a través de caminos de datos controlado por multiplexores y cuya lógica de selección viene determinada por el código de la operación que se está realizando.

Dicho control se encuentra detallado y especificado en el capítulo 4 *Controlador de camino de datos y sincronización.* Por lo tanto la manera en que se encuentran interconectadas las etapas del pipeline aritmético hace parte importante de los elementos de la arquitectura del pipe.

Dado lo anterior se puede decir que la interconexión de las etapas es distinta, según la operación que se ejecute en un momento en particular, esto significa también que algunas etapas pueden quedar por fuera del camino de datos, según la interconexión vigente, o el pipeline y los caminos de datos en ciertas operaciones pueden ser más extensos que en otras.

La longitud o extensión de los caminos de datos puede cambiar, porque los datos que fluyen por el pipe ejecutan varias etapas de varias operaciones, para realizar operaciones más complejas, operaciones compuestas si se quiere.

Adicionalmente el hecho de que haya varios caminos de pipe, al mismo tiempo operándose resulta conveniente para operaciones vectoriales, donde se pueden llevar a cabo hasta n operaciones al mismo tiempo, por ejemplo: al momento de realizar un producto punto se requiere multiplicar, en primera instancia, todos los componentes vectoriales, uno a uno, de cada vector.

ILUSTRACIÓN AQUÍ DE LOS BLOQUES DE SUMA RESTA MULTIPLICACION Y DIVISION, RAIZ CUADRADA CON EL DPC (DIAGRAMA DE BLOQUES DE LA ARQUITECTURA, MÁS COMO EN CAPAS QUE BLOQUES INTERCONECTADOS)

Cada bloque de operaciones tiene dentro un número de etapas secuenciales. Cada bloque de operaciones es pues un compendio de etapas y estos bloques se encuentran interconectados por el *Controlador de camino de datos y sincronización.*

La interconexión resultante de las diferentes operaciones resulta en 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 interfase de programación del Raytrac. Para detalles referentes a la manera en que se encuentran codificadas las instrucciones por favor leer el capítulo Quinto.

Explicado lo anterior se entiende que la arquitectura, desde el punto de vista de la conexión de componentes, es distinta con cada operación pues el camino de datos decodificado es distinto.

## Ejemplo Modo Producto Punto

## Ejemplo Modo Producto Cruz

## Ejemplo Modo Adición

# Representación de Datos en Punto Flotante.

Los número de entrada y de salida en el Raytrac, están representados en el sistema aritmético base 2 y sus valores están representados en punto flotante sencillo usando el estándar IEEE 754.

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” (CITAR).
* 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 6. 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 7. 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 4. 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 8. Diagrama de Bloques de la suma en punto flotante de 32 bits. Archivo RTL: fadd32.vhd

## 

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

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Nombre de la Unidad | Qty | lcells | | | multipliers | | | memory bits | | |
| unit size | total | ocuppied size | used by unit | total | usage | used by unit | total | usage |
| fadd32 (0) | 1 | 343 | 343 | 1.393% | 3 | 3 | 2.273% | 96 | 96 | 0.016% |
| fadd32 (1) | 1 | 339 | 339 | 1.377% | 3 | 3 | 2.273% | 0 | 0 | 0.000% |
| fadd32 (2) | 1 | 338 | 338 | 1.373% | 3 | 3 | 2.273% | 0 | 0 | 0.000% |
| fadd32 (3) | 1 | 303 | 303 | 1.231% | 3 | 3 | 2.273% | 0 | 0 | 0.000% |

Tabla 5. Recursos usados por los 4 sumadores del RayTrac.

# Multiplicación 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).

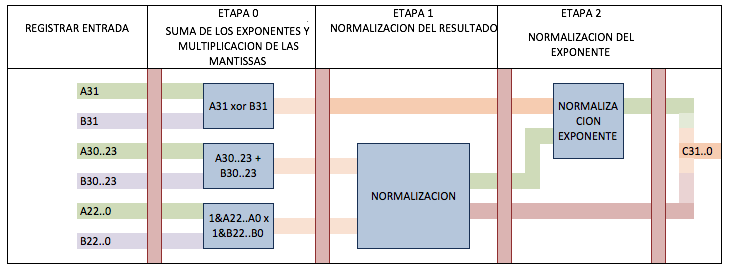
El producto de los dos valores será el siguiente:

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. |

## Diagrama de Bloques.



## Diagrama de Bloques, RTL producido, Recursos Utilizados.

# Valor elevando a la potencia -1, IEEE 754, 32 Bits en Punto Flotante, 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.

## RTL, Producido, Recursos Utilizados.

# 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, Diagrama de Bloques, RTL producido, Recursos Utilizados.

# Cola de Sincronización Producto Punto.

## Justificación,

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. 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 se debe encontrar a la salida de la cola para el momento.

## Diagrama de Flujo, Diagrama de Bloques, RTL producido, Recursos Utilizados.

# Cola de Sincronización de Normalización de Vectores.

## Justificación,

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.

El vector se lee la primera vez desde la memoria y debe continuar a través del pipe, mientras se calcula la magnitud de este y se invierte. Para ello se implementa una cola que tiene tantas etapas como la operación de cálculo e inversión de la magnitud tiene.

## Diagrama de Flujo, Diagrama de Bloques.

## RTL producido, Recursos Utilizados.

La cola de sincronización de la operación de normalización de vectores es una instancia de la biblioteca altsyncram, que es de la IP de Altera. Dicha instancia se encuentra en el RTL del controlador de camino de datos y sincronización.

1. Capítulo 4, Controlador de Camino de Datos y Sincronización.

# Sincronización del RayTrac, cola de salida.

La sincronización del RayTrac es muy importante porque la unidad aritmética está diseñada en pipeline y para poder explotar el desempeño potencial que este ofrece, la sincronización de los componentes en entrada al pipe, los componentes en el pipe y la salida del pipe es crucial.

Son 4 los elementos a sincronizar: los registros con los operandos, la cola de entrada de instrucciones, el pipeline aritmético y la cola de salida donde se escriben los resultados obtenidos.

La prioridad de la sincronización la tiene la cola de salida. Si esta cola se llena, los datos que marchan a través del pipeline aritmético, deben detenerse, dado que la cola de salida tiene un mecanismo de protección, que no permite la escritura en la cola una vez esta se encuentre llena. En ese momento el RayTrac se encontrará en estado ocioso, hasta que no ocurra un evento externo de lectura. Si esta cola está vacía puede recibir los resultados que vienen de la etapa posterior, el pipeline aritmético.

Conforme se escribe en la cola de resultados, el bus debe generar varios ciclos de lectura, para que se genere o se mantenga la capacidad de almacenamiento en la cola de salida y puedan transitar hacia esta los datos que vienen del pipe aritmético.

La cola de resultados debe generar por cada instrucción una interrupción que le permita sincronizar el comienzo de la lectura de datos al bus externo.

# Sincronización del RayTrac cola de entrada de instrucciones.

Es claro que la cola de salida de resultados es el extremo del sistema y que este debe tener espacio que permita que todas los datos que se encuentran en registros a lo largo de las etapas previas (aritmética, operandos e instrucciones), puedan seguir transitando.

Sin embargo el siguiente elemento en la sincronización a tener en cuenta es la cola de instrucciones. Los datos en esta cola le indican al RayTrac qué operaciones y con qué operandos se desea realizar la próxima operación en el RayTrac. Si la cola de instrucciones no tiene datos entonces el RayTrac se encontrará en estado ocioso.

La cola de instrucciones debe tener al menos un elemento para que el RayTrac comience a operar. Es importante anotar que antes que comience la operación del RayTrac con la primera instrucción los operandos, se deben encontrar escritos en el registro de entrada.

# Descripción detallada del mecanismo de sincronización.

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

Para comenzar a detallar el mecanismo de sincronización del pipeline entiéndase en RayTrac como un gran pipeline, la primera etapa es la cola de instrucciones, la segunda el pipe aritmético y la tercera la cola de resultados.

En el capítulo de (datapath control) se vio, 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. Capítulo 5, Máquina de Estados e Interfase de Programación y Sistema de Interrupciones.

La máquina de estados es el circuito que controla las operaciones que realiza el RayTrac. La interfase de programación son aquellos mecanismos que utiliza un desarrollador para programar y comunicarse con el RayTrac. Finalmente el Sistema de Interrupciones notifica a la aplicación usuaria del RayTrac dos eventos: que se terminó de ejecutar una instrucción en particular en el RayTrac o que la cola de resultados está llena, bloqueando el pipeline aritmético.

# Máquina de Estados.

La máquina de estados que controla la operación del RayTrac posee 3 estados:

* *Load\_Instruction*.
* *Execute\_Instruction*.
* *Flush\_Arith\_Pipeline.*

## Load\_Instruction.

El estado FlushToNextInstruction, es el estado en el que se inicializa el RayTrac. Básicamente este estado entra en ejecución, en el momento en que se termina de leer el último operando o par de operandos de la instrucción en ejecución.

El objetivo de este estado es esperar a que los datos de la operación que está terminando, finalicen su transitar por el pipeline aritmético y que la cola de resultados correspondiente a la siguiente instrucción tenga suficiente espacio para comenzar a recibir datos. Una vez se cumplen las condiciones mencionadas, la máquina de estados cambia al siguiente estado, ExecuteInstruction.

A pesar que el nombre del estado es ExecuteInstruction, haciendo alusión a que en este estado se ejecutan las instrucciones que entran en el RayTrac, no es esto lo que ocurre en primera instancia, puesto que si la cola de instrucciones está vacía, no existirá operación para ejecutar.

## ExecuteInstruction.

En este estado se leen los operandos de la instrucción, que se ejecuta en el momento. Para seguir al estado FlushToNextInstruction, es necesario enviar el último par de operandos ó el último operando, por el pipeline aritmético. En este estado es necesario que la máquina realice una pausa si no hay espacio suficiente en la cola de resultados de la instrucción que corresponda.

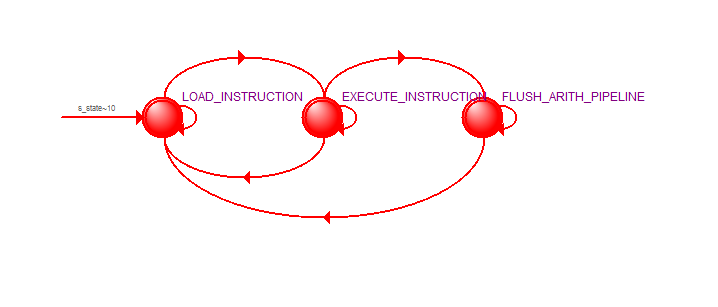


Ilustración 9. Máquina de estados que controla al RayTrac. La máquina de estado posee tres estados: LOAD\_INSTRUCTION, EXECUTE\_INSTRUCTION y FLUSH\_ARITH\_PIPELINE. RTL: sm.vhd[[3]](#footnote-3)

Puede que exista un sub estado IDLE, porque los 2 estados deben hacer una pausa si la cola de resultados, de la instrucción que se está ejecutando no tiene más espacio que el número de resultados que en ese momento se encuentren en el pipeline aritmético. Sin embargo este estado no se considera para simplificación del ejercicio.

# Interfase de Programación.

La interfase de programación está formada por el conjunto de elementos de Software y Hardware, que le permiten al desarrollador, programar aplicaciones que hacen uso del RayTrac.

Los recursos que se ofrecen para realizar un desarrollo sobre el RayTrac son apenas los suficientes, para mantener el alcance de este trabajo. Dichos recursos son:

* Set de Instrucciones SIMD.
* Interfase de Hardware mediante bus Avalon ® de Altera ®.
* Una Interfase de Programación de Aplicaciones, las siglas en inglés API, para el desarrollo de aplicaciones de Alto Nivel, en lenguaje C.

Instrucciones SIMD.

El RayTrac es una máquina para realizar SIMD. Sin embargo esto no es cierto para algunas configuraciones del Controlador de Camino de Datos. Dependiendo de la operación vectorial que se quiera realizar se puede decir que uno está realizando hasta 2 operaciones en paralelo ó una sola.

Sin embargo la limitante se da porque la FPGA usada en este proyecto no posee la capacidad suficiente para implementar una RayTrac más “grande”, que permita tener más pipelines al mismo tiempo.

El set de instrucciones que se detalla a continuación, permite ver desde el punto de vista de arquitectura cómo está concebido el hardware dentro del RayTrac. Sin embargo para efectos prácticos, es mejor programar las aplicaciones haciendo uso de la API en lenguaje C.

En total son cinco las instrucciones vectoriales que puede ejecutar en el RayTrac:

|  |  |  |
| --- | --- | --- |
| Instrucción | Descripción | Función y bloques de operandos que actúan. |
| DotProd | Producto Punto de un par de vectores | **A.B** y **C.D** |
| AddSub | Suma o Resta de un par de vectores. | **A±B** |
| CrossProd | Producto Cruz de un par de vectores. | **AxB** |
| MagNrm | Magnitud y Normalización de un vector. | **|A|** y **A/|A|** |
| SclMlt | Multiplicación de un escalar por un vector. | cx \* **D** |

Tabla 6. Tabla de Instrucciones, operandos y funciones.

Las operaciones vectoriales descritas en la tabla tienen cada una un espacio en los bloques de memoria donde residen sus respectivos operandos y los resultados de cada una de ellas también se encontrarán en espacios o direcciones específicas de los bloques de memoria de salida.

Bloques Vectoriales

Los bloques de entrada son 12, a estos bloques se les denomina bloques de componentes. 3 bloques de componentes por cada operando vectorial A, B, C y D. Cada conjunto de 3 bloques de componentes es 1 bloque vectorial, por lo tanto hay 4 bloques vectoriales. Según lo anterior los bloques vectoriales son bloques de memoria que contienen en sus registros un vector de 3 componentes en formato IEEE 754.

Cada bloque vectorial contiene hasta 512 vectores/palabras, de 96 bits de ancho cada uno. Para simplificar la lógica de decodificación de instrucciones cuando la máquina de estados se encuentre en el estado ExecuteInstruction, se segmenta los bloques vectoriales en 32 grupos de 16 vectores. Dentro del marco de la instrucción se hace referencia a los grupos más que a un vector en particular. Esto implica que las operaciones por lo menos se harán con 32 vectores siempre, mejor aún: esto implica que para la interfase de programación los operandos siempre van a ser bloques vectoriales, más que uno u otro vector en particular.

Por ejemplo, uno puede sacar la magnitud y el vector normal (MagNrm) de los 32 vectores que se encuentran en el segmento 0x09 del bloque vectorial A. El operando aquí es el segmento 0x09.

Por qué se asumió este criterio de diseño? Porque el RayTrac es una máquina hecha para Ray Tracing en primera instancia, pero en general sirve para realizar múltiples operaciones vectoriales por segundo y dichas operaciones tienen múltiples operandos para ser procesados y no 2, 3, etc., vectores en particular.

Los 16 segmentos de cada bloque vectorial, contiene 32 vectores cada uno pues y para referirse a un segmento en particular se denota así:

B[name]S[number]

Y para referirnos a un vector dentro de un segmento en particular se denota así:

B[name]S[number](vnumber)

Donde name es el nombre del bloque vectorial A, B, C o D. Y number es el número del segmento dentro del bloque vectorial referenciado mediante name. Vnumber es el número del vector dentro del segmento en particular y va desde el 0 al 31.

Modo Combinatorio ó Uno a Uno.

Una instrucción puede ser combinatoria o uno a uno. Lo primero es que una instrucción debe involucrar 2 bloques vectoriales distintos para que la selección del modo tenga sentido. Por lo tanto las operaciones SclMlt y MagNrm no hacen uso de un modo ó el otro.

Supongamos que la operación a aplicar es la siguiente:

AddSub (BAS0,BAS0,BBS3,BBS3,Comb)

La instrucción se lee así: “Sumar los 32 vectores que se encuentran en el segmento 0 y del bloque vectorial A, con los 32 vectores que se encuentran el segmento 3 del bloque vectorial B, de manera combinatoria”.

El algoritmo realizara las siguientes serie de sumas vectoriales:

BAS0(0)+BSS3(0), BAS0(0)+BSS3(1), …, BAS0(0)+BSS3(30), BAS0(0)+BSS3(31), BAS0(1)+BSS3(0), …, BAS0(1)+BSS3(30), BAS0(1)+BSS3(31),

BAS0(2)+BSS3(0), …, BAS0(2)+BSS3(30), BAS0(2)+BSS3(31),…,

BAS0(30)+BSS3(0),.., BAS0(30)+BSS3(30), BAS0(30)+BSS3(31),

BAS0(31)+BSS3(0),.., BAS0(31)+BSS3(30), BAS0(31)+BSS3(31).

Supongamos que la operación a aplicar es la siguiente:

AddSub (BAS0,BAS0,BBS3,BBS3,Uno)

La instrucción se lee así: “Sumar los 32 vectores que se encuentran en el segmento 0 y del bloque vectorial A, con los 32 vectores que se encuentran el segmento 3 del bloque vectorial B, de manera uno a uno”.

El algoritmo realizara la siguiente serie de sumas vectoriales:

BAS0(0)+BSS3(0), BAS0(1)+BSS3(1), …, BAS0(30)+BSS3(30), BAS0(31)+BSS3(31).

La combinatoria hace una combinación lineal de los vectores que hay en los segmentos escogidos para ser operados.

# Referencia de Instrucciones.

En esta sección se provee al lector con una guía de referencia de la manera como se codifican las instrucciones que ejecuta el RayTrac.

En general la trama de instrucciones es de 32 bits y posee 7 campos:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| UCA (3) | BLOCK START A/C (5) | BLOCK END A/C (5) | BLOCK START B/D (5) | BLOCK END B/D (5) | COMBINATORIAL (1) | RESERVED (8) |

Tabla 7. Trama de Instrucción.

El campo UCA tiene 4 bits, lo que da la posibilidad de tener 16 instrucciones en teoría, pero para simplificar la codificación solo se implementaron 7 operaciones:

|  |  |
| --- | --- |
| **Operación** | **Código UCA** |
| Mag/Nrm | 100/101 o 110 |
| Add / Sub | 001 / 011 |
| Dotprod | 000 |
| Crossprod | 010 |
| SclMlt | 111 |

Tabla 8. Tabla de Operaciones y Códigos UCA.

**MAG/NRM (*no*, *nf*)**

*Descripción:* Calcula la magnitud o la normal, de los vectores del bloque A entre los segmentos comprendidos desde el número, *no*, hasta el *nf*. Aunque son 2 operaciones distintas, estás se ejecutan en paralelo, pero con un desfase en ciclos de reloj: Primero queda calculado la magnitud del vector y quince ciclos de reloj después estará calculado el Vector Normal.

*Parámetros:* Los parámetros son los vectores que se encuentran el segmento inicial, no, y final, nf. Estos segmentos contienen cada uno 32 vectores, cuya magnitud será calculada y al mismo tiempo serán normalizados. Esta operación solamente lee vectores del bloque A desde el segmento 0 al segmento 15.

*Colas de Resultados:* El cálculo de la magnitud corresponde a un valor escalar, que se almacena en la cola de resultados *RES0*. El cálculo de la normalización es un vector y por lo tanto se compone de 3 valores escalares. Estos valores se escriben en las colas de resultados, *RES1, RES2, RES3*.

*Trama de Operación.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| UCA (3) | BLOCK START A/C (5) | BLOCK END A/C (5) | BLOCK START B/D (5) | BLOCK END B/D (5) | COMBINATORIAL (1) | RESERVED (8) |
| 100/101 o 110 | (00000-11111) | (00000-11111) | Don’t Care | Don’t Care | 0 | Don’t Care |

*Ejemplo:* NRM (09,13)

Calcula la normal de los vectores en los segmentos 09 al 13, al ser cinco segmentos y en cada uno 16 vectores, son 80 vectores normales las que se calcularían.

**DOTPROD (BAC*no*, BAC*nf*, BBD*no*, BBD*nf, Comb*)**

*Descripción:* Esta operación realiza 2 cálculos de productos punto al mismo tiempo: A.B y C.D sobre 2 pipes distintos.

*Parámetros:* Los parámetros son los vectores que se encuentran en los segmentos inicial y final, *no* y *nf*, respectivamente de los bloques A,B,C y D. En está operación se puede aplicar modo combinatorio, usando *Comb=1.* El modo combinatorio se aplicará ó no, a A.B tanto como a C.D.

*Colas de Resultados:* El cálculo de los productos punto corresponde a dos valores escalares, uno por cada operación, que se almacenan en la colas de resultados *RES2* y *RES4* para A.B y C.D Respectivamente.

*Trama de Operación.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| UCA (3) | BLOCK START A/C (5) | BLOCK END A/C (5) | BLOCK START B/D (5) | BLOCK END B/D (5) | COMBINATORIAL (1) | RESERVED (8) |
| 000 | (00000-11111) | (00000-11111) | Don’t Care | Don’t Care | 0/1 | Don’t Care |

Ejemplo: DOTPROD (0, 0, 0, 1, 0).

Este comando ejecuta productos punto uno a uno de la siguiente manera:

* 16 productos puntos entre A0 y B0: {A000⋅B000,.., A0i⋅B0i,.., A031⋅B031}.
* 16 productos puntos entre A0 y B1: {A000⋅B100,.., A0i⋅B1i,.., A031⋅B131}.
* 16 productos puntos entre C0 y D0: {C000⋅D000,.., D0i⋅D0i,.., C031⋅D031}.
* 16 productos puntos entre C0 y D1: {C000⋅D100,.., D0i⋅D1i,.., C031⋅D131}.

**CROSSPROD (BA*no*, BA*nf*, BB*no*, BB*nf, Comb*)**

*Descripción:* Esta operación realiza el cálculo del vector producto cruz de los vectores que se encuentran en los bloques A y B.

*Parámetros:* Los parámetros son los vectores que se encuentran los segmentos inicial y final, *no* y *nf*, respectivamente de los bloques A y B. En está operación se puede aplicar modo combinatorio, usando *Comb=1.*

*Colas de Resultados:* El cálculo de los productos punto corresponde a los 3 valores escalares que componen el vector resultante y se almacenan en la colas de resultados *RES1, RES2* y *RES3*.

*Trama de Operación.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| UCA (3) | BLOCK START A/C (5) | BLOCK END A/C (5) | BLOCK START B/D (5) | BLOCK END B/D (5) | COMBINATORIAL (1) | RESERVED (8) |
| 010 | (00000-11111) | (00000-11111) | Don’t Care | Don’t Care | 0/1 | Don’t Care |

Ejemplo: CROSSPROD (20, 21, 0, 0, 1)

Esta instrucción instruye al RayTrac a realizar 512 operaciones de producto cruz entre los vectores del bloque A del segmento 20 y 21, y los vectores del bloque B del segmento 0:

* 256 productos cruz entre A20 y B0 : {A2000 x B000, A2000 x B001, A2000 x B002, .., A2000 x B015, A2001 x B000, A2001 x B001, A2001 x B002, .., A2015 x B000, A2015 x B001, A2015 x B002, .., A2015 x B015}
* 256 productos cruz entre A21 y B0 : {A2100 x B000, A2100 x B001, A2100 x B002, .., A2100 x B015, A2101 x B000, A2101 x B001, A2101 x B002, .., A2115 x B000, A2115 x B001, A2115 x B002, .., A2115 x B015}

**ADD / SUB (BA*no*, BA*nf*, BB*no*, BB*nf, Comb*)**

*Descripción:* Estas operaciones realizan el cálculo de la suma o resta de vectores en los bloques A y B.

*Parámetros:* Los parámetros son los vectores que se encuentran los segmentos inicial y final, *no* y *nf*, respectivamente de los bloques A y B. En esta operación se puede aplicar modo combinatorio, usando *Comb=1.*

*Colas de Resultados:* El cálculo de los productos punto corresponde a los 3 valores escalares que componen el vector resultante y se almacenan en la colas de resultados *RES1, RES2* y *RES3*.

*Trama de Operación.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| UCA (3) | BLOCK START A/C (5) | BLOCK END A/C (5) | BLOCK START B/D (5) | BLOCK END B/D (5) | COMBINATORIAL (1) | RESERVED (8) |
| 010 | (00000-11111) | (00000-11111) | Don’t Care | Don’t Care | 0/1 | Don’t Care |

Ejemplo: SUB(20,21,0,0,1)

Esta instrucción instruye al RayTrac a realizar 512 operaciones de resta entre los vectores del bloque A del segmento 20 y 21, y los vectores del bloque B del segmento 0:

* 256 restas entre A20 y B0 : {A2000 - B000, A2000 - B001, A2000 - B002, .., A2000 - B015, A2001 - B000, A2001 - B001, A2001 - B002, .., A2015 - B000, A2015 - B001, A2015 - B002, .., A2015 - B015}
* 256 restas entre A21 y B0 : {A2100 - B000, A2100 - B001, A2100 - B002, .., A2100 - B015, A2101 - B000, A2101 - B001, A2101 - B002, .., A2115 - B000, A2115 - B001, A2115 - B002, .., A2115 - B015}

**SCLMLT (BCX*no*, BCX*nf*, BD*no*, BD*nf, Comb*)**

*Descripción:* Cálculo del producto vectorial entre un vector y un valor escalar.

*Parámetros:* Los parámetros son los vectores que se encuentran los segmentos inicial y final, *no* y *nf*, respectivamente de los bloques CX y D. En está operación se puede aplicar modo combinatorio, usando *Comb=1.*

*Colas de Resultados:* El cálculo de los productos punto corresponde a los 3 valores escalares que componen el vector resultante y se almacenan en la colas de resultados *RES1, RES2* y *RES3*.

*Trama de Operación.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| UCA (3) | BLOCK START CX (5) | BLOCK END CX (5) | BLOCK START D (5) | BLOCK END D (5) | COMBINATORIAL (1) | RESERVED (8) |
| 010 | (00000-11111) | (00000-11111) | Don’t Care | Don’t Care | 0/1 | Don’t Care |

Ejemplo: SUB (20, 21, 0, 0, 1)

Esta instrucción instruye al RayTrac a realizar 512 operaciones de resta entre los vectores del bloque CX del segmento 20 y 21, y los vectores del bloque D del segmento 0:

* 256 restas entre CX20 y D0 : {CX2000 - D000, CX2000 - D001, CX2000 - D002, .., CX2000 - D015, CX2001 - D000, CX2001 - D001, CX2001 - D002, .., CX2015 - D000, CX2015 - D001, CX2015 - D002, .., CX2015 - D015}
* 256 restas entre CX21 y D0 : {CX2100 - D000, CX2100 - D001, CX2100 - D002, .., CX2100 - D015, CX2101 - D000, CX2101 - D001, CX2101 - D002, .., CX2115 - D000, CX2115 - D001, CX2115 - D002, .., CX2115 - D015}

1. CAPÍTULO 6, 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 9. Frecuencia máxima calculada para el RayTrac.

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

Tabla 10. 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. **http://opencores.org/websvn,filedetails?repname=raytrac&path=%2Fraytrac%2Fbranches%2Ffp%2Fraytrac.vhd** [↑](#footnote-ref-1)
2. <http://opencores.org/websvn,filedetails?repname=raytrac&path=%2Fraytrac%2Fbranches%2Ffp%2Fmemblock.vhd> [↑](#footnote-ref-2)
3. **http://opencores.org/websvn,filedetails?repname=raytrac&path=%2Fraytrac%2Fbranches%2Ffp%2Fsm.vhd** [↑](#footnote-ref-3)