

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

REPRESENTACIÓN DE FUNCIONES CON VHDL

DOCUMENTO DE PROYECTO



TRABAJO PRESENTADO POR

AITOR ALONSO LORENZO
VÍCTOR ADOLFO GALLEGOS ALCALÁ
ANA MARÍA MARTÍNEZ GÓMEZ

2014

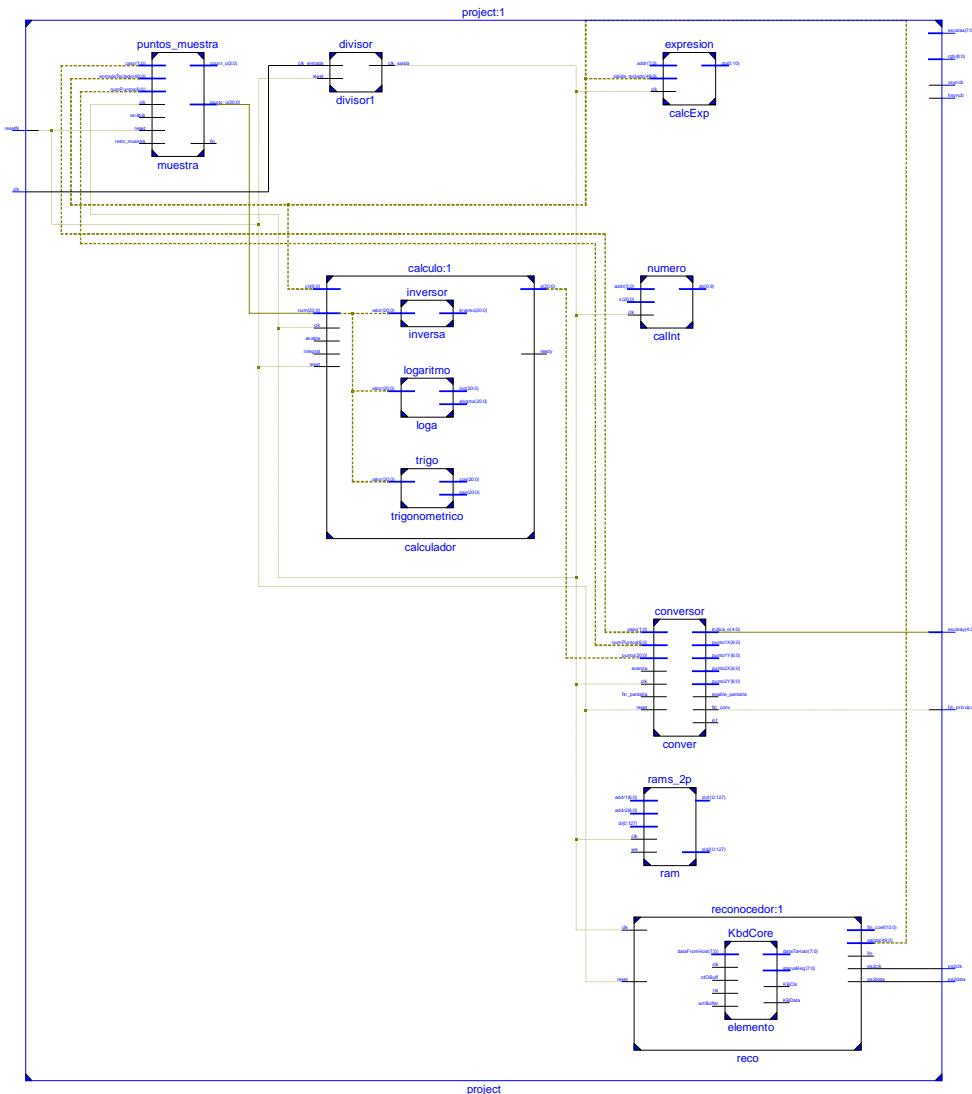
Tecnología y Organización de Computadores

Índice general

1. Ruta de datos	1
2. Controladores	2
3. Uso de la FPGA	7
4. ¿Por qué somos los mejores?	8
5. Trabajo realizado. Éxitos y dificultades	9
6. Reparto de trabajo	13
7. Programas y bibliografía	14
Programas utilizados	14
Bibliografía	14
Anexo	15
Ver la función sin FPGA	15
Problemas con el reloj	17
Desbordamientos	18
Programas en Matlab	18

1. Ruta de datos

El diagrama de la ruta de datos de nuestro proyecto es el siguiente:

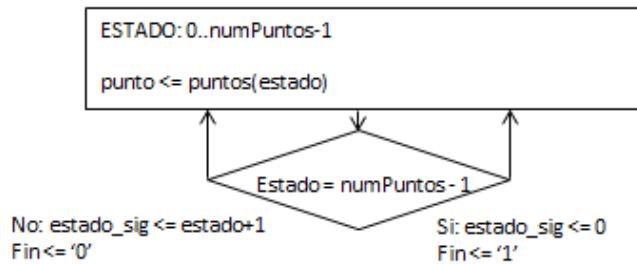


Con él queremos reflejar los distintos componentes que componen el proyecto, así como las conexiones entre ellos. En la siguiente sección se puede encontrar un detalle sobre la finalidad y funcionamiento de cada parte. Asimismo, en el diagrama se muestran los pines de entrada y salida. En el documento de Guía de Utilización (Pines y significado) se puede encontrar más información sobre ellos.

2. Controladores

En esta sección se presentan los diversos controladores de nuestro proyecto. Para cada uno de ellos se muestra un diagrama ASM y una breve descripción de estados y funcionamiento en los casos en los que sea preciso.

Puntos muestra.vhd

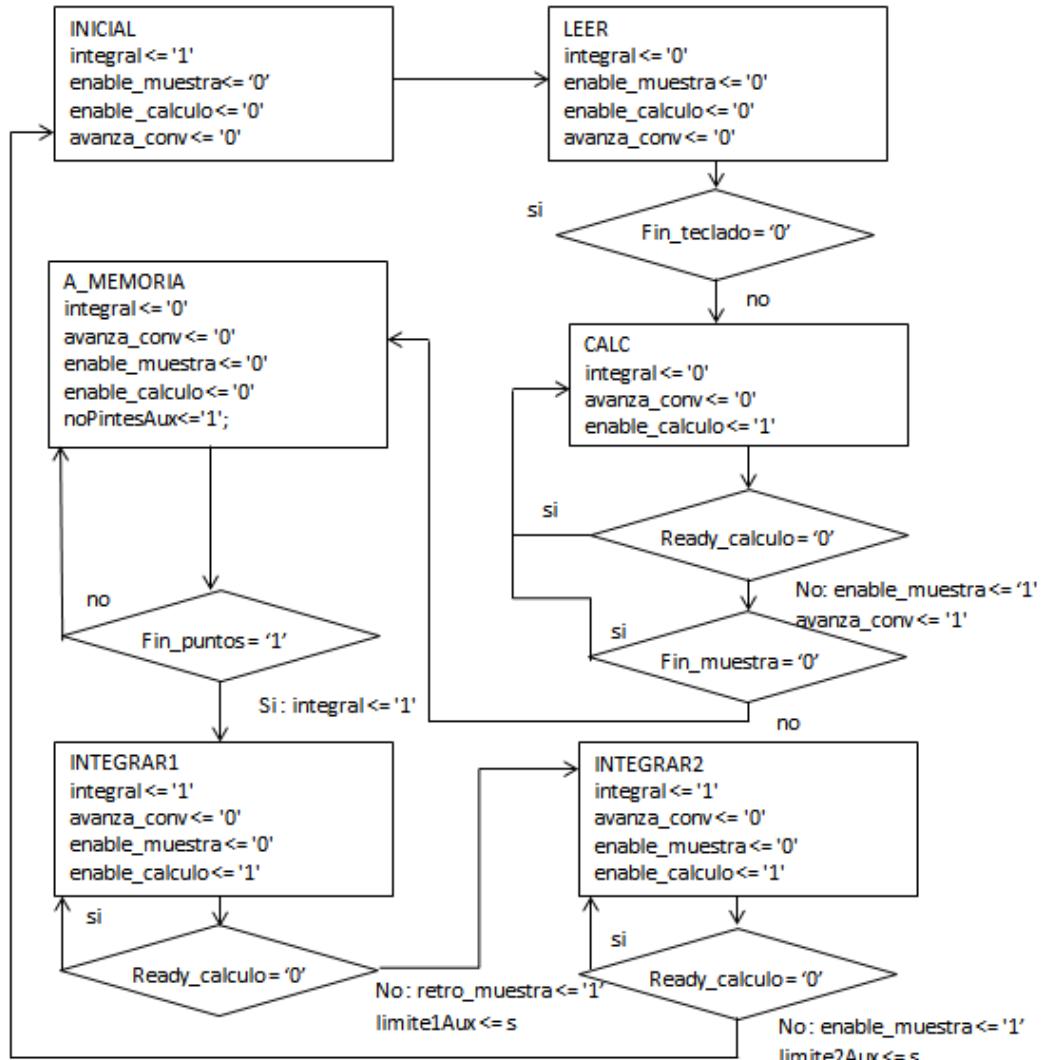


Se encarga de, una vez recibidos los coeficientes de la función introducida por el teclado, elegir la escala horizontal adecuada (señal count¹), así como escoger el muestreado de puntos a evaluar adecuado, con el fin de lograr un óptimo reconocimiento de la función a representar. Posee tres constantes de tipo matriz, una para cada set de puntos, donde se guardan los puntos en los que evaluaremos la función.

Vga.vhd

El controlador principal se encuentra en project (vga.vhd). Consiste en una máquina de estados (sincronoGen) asociada a la señal estadoGen. Los estados son: *Inicial*: estado de inicio. *Leer*: se obtienen los coeficientes de la función por medio del teclado. *Calc*: se calcula la imagen de los distintos puntos del muestreo. *A memoria*: se mandan pares de coordenadas de pantalla de puntos, con el fin de que se escriba en la RAM la recta que los une. *Integrar1* e *Integrar2*: se evalúa la primitiva en los dos extremos.

¹Process pcount



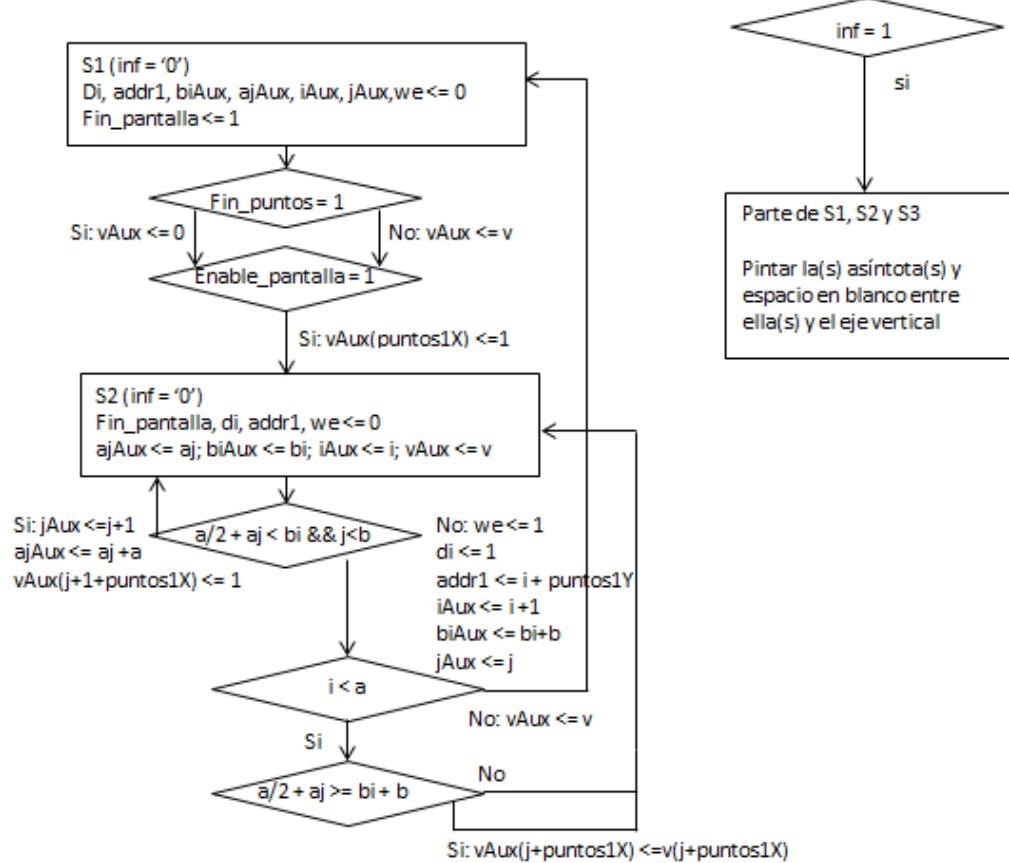
También contiene el controlador de la pantalla ², así como el resto de componentes. En él también aparece la algoritmia dedicada a pintar un segmento entre dos puntos y la representación de las asíntotas cuando la función tiende a ∞ ³. Además cuenta con dos máquinas de estados ⁴.

El diagrama de la algoritmia para pintar un segmento dados dos puntos es el siguiente:

²Process *pA*, *pB*, *C*, *D*, *colorear*, *pintar_fondo*, *pintar_ejes*, *pintar_funcion*, *pintar_expresion*

³Esto se realiza en el process *pinf1y2*

⁴*sincrono*, para la algoritmia del segmento; y *sincronoGen*, la descrita en primer lugar



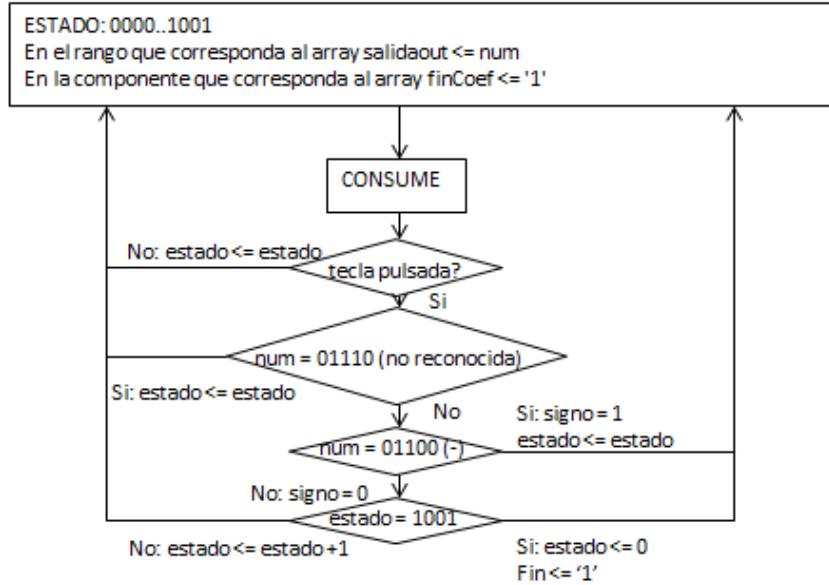
Se escriben todas las columnas entre dos puntos dados. Se tiene en cuenta que siempre se mandan parejas de puntos donde la coordenada X del primer punto es la del segundo del par de puntos anterior, menos en la primera pareja que es el de la izquierda de la pantalla. Además se ha de guardar en cada iteración el vector anterior debido a que en cada columna puede haber más de un punto (esto depende de la pendiente).

Reconocedor.vhd

El módulo del reconocedor de teclado. Contiene su máquina de estados⁵, además del módulo KbdCore (ver aquí). Se encarga de enviar como salida los coeficientes de la función introducidos por el teclado⁶

⁵process sinc y leer

⁶posiciones 49-45: log; 44-40: sen; 39-35: cos; 34-30: x^3 ; 29-25: x^2 ; 24-20: x ; 19-15:constante; 14-10: x^{-1} ; 9-5: x^{-2} ; 4-0: x^{-3} ; codificados en C2.



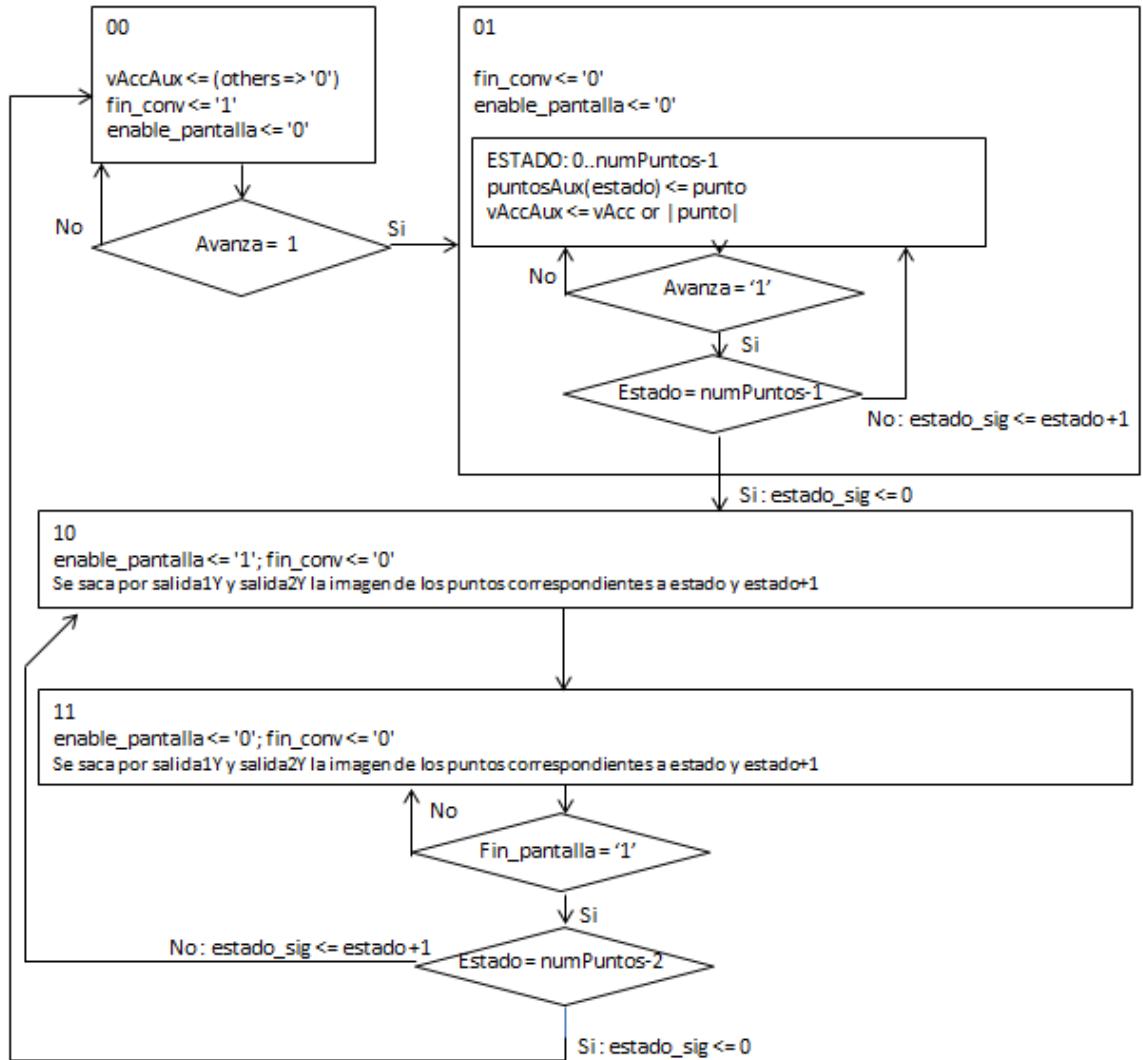
Calculo.vhd

Se encarga de calcular la imagen del punto que recibe como entrada (num) por la función introducida por el teclado (entrada c). Para ahorrar recursos de la FPGA, cuenta con un único multiplicador (21x21 bits), por lo que está controlado con una máquina de estados⁷, que tiene dos rutas en función de la señal integral. Si está desactivada, vamos calculando las potencias positivas de x, las negativas, el seno, el coseno y el logaritmo (multiplicando por sus coeficientes introducidos por el teclado) en ese orden, ayudándonos de acumuladores de resultados parciales (cumsum, acc, acc2), hasta que termina, y la salida s devuelve el resultado de la evaluación. De forma análoga, cuando el modo integral está activado, evaluamos en el punto dado mediante una primitiva de la función. Por este motivo, es necesario calcular hasta la cuarta potencia de x, así como emplear las constantes llamadas tres (almacena el valor de un tercio, para la primitiva de x^2); y pi (su inverso), para las primitivas de $\text{sen}(\pi x)$ y $\text{cos}(\pi x)$. Por último, contiene los módulos inversor.vhd, trigo.vhd y logaritmo.vhd, donde se guardan las tablas con los valores posibles⁸

⁷Process pestado y asicn

⁸funciones $\frac{1}{x}$, $\text{sen}(\pi x)$, $\text{cos}(\pi x)$, $\log(x)$, y $x\log(x) - x$. Consultar genera2.m para más información

Conversor.vhd



Este módulo cuenta con una máquina de cuatro estados: *00*: inicial; *01*: de guardado de puntos. Se guardan en puntos, y vAcc (en este último, como valor absoluto, con el fin de poder obtener el índice necesario realizar el reescalado a coordenadas de pantalla, viendo el primer bit no nulo con un codificador de prioridad⁹ (factor para la escala vertical)); *10* y *11*: estados de transferencia de puntos a la memoria/pantalla ¹⁰

⁹Process index

¹⁰Hay dos estados debido a que tal y como está diseñada la algoritmia de la pantalla, es necesario permanecer un ciclo con su enable a 1

3. Uso de la FPGA

En la última versión del proyecto, éstos son los porcentajes de uso de los recursos de la FPGA

```
Device utilization summary:  
-----  
Selected Device : 3s1000ft256-5  
  
Number of Slices: 4903 out of 7680 63%  
Number of Slice Flip Flops: 1464 out of 15360 9%  
Number of 4 input LUTs: 9280 out of 15360 60%  
Number of IOs: 29  
Number of bonded IOBs: 29 out of 173 16%  
Number of BRAMs: 6 out of 24 25%  
Number of MULT18X18s: 3 out of 24 12%  
Number of GCLRs: 2 out of 8 25%
```

Estos son los tiempos:

```
Minimum period: 40.431ns (Maximum Frequency: 24.733MHz)  
Minimum input arrival time before clock: 6.253ns  
Maximum output required time after clock: 27.949ns  
Maximum combinational path delay: No path found
```

Todos estos datos fueron tomados con las opciones por defecto de Xilinx de los laboratorios de la Facultad de Informática.

En cuanto a los mensajes de Warning, todos fueron identificados y eliminados, los que no, su permanencia está justificada a lo largo de la Sección 5.

4. ¿Por qué somos los mejores?

Se expone a continuación una lista a modo de resumen de lo mejor de nuestro proyecto. Todos los puntos se explican con más detalle en 4. Exitos y dificultades

- **Pintamos más funciones de las que fueron pedidas y usamos escala variable, aunque inicialmente se nos indicó que bastaba con que fuese fija.** Más información aquí.
- **Mucho tiempo invertido en algoritmia.** Más información aquí.
- **Implementamos la funcionalidad extra de integrar.** Más información aquí.
- **El tiempo necesario para pintar una función es tan pequeño que el proceso es imperceptible a la vista, a pesar de que se nos permitía pintarla despacio e incluso a trozos.** También borramos la función al hacer reset. Más información aquí.
- **Tratamos de usar el mínimo hardware posible** Rediseñamos el sistema para adaptarlo a la capacidad de la FPGA (más información aquí). Tratamos de reducir el número de multiplicadores y divisores (más información aquí). Esto afectó a la forma de calcular de imagen de cada punto (más información aquí). Redujimos la memoria (más información aquí).
- **Tuvimos que adaptarnos a la necesidad de avanzar el proyecto sin poder disponer de FPGA** Encontramos soluciones eficaces a este problema. Más información aquí.
- **Utilización de otros programas y la necesidad de aprender a usarlos o de aprender a usar algunas funciones** El ejemplo más significativo es Matlab, como se describe en la sección siguiente. También los programas de edición de vídeo y audio.
- **Finalizamos pronto la parte obligatoria del proyecto** El día 9 de enero teníamos implementadas funcionalidades más allá de lo acordado en la parte obligatoria (más información aquí).

5. Trabajo realizado. Éxitos y dificultades

La parte más importante de nuestro proyecto fue la algoritmia, pues invertimos mucho tiempo en decidir las escalas, los puntos a calcular, como unir dos puntos, etc; tiempo que de otra forma hubiésemos invertido en realizar más código en vhdl. Primero tuvimos que decidir como escalar los ejes para que toda función fuese visible, que ninguna parte de la gráfica se saliese de la pantalla (ni de precisión por desbordamiento) pero que a su vez toda función fuese apreciable y no quedase reducida a 0 por la escala. Aunque al principio consideramos una escala fija en el eje Y y que lo que fuese demasiado grande valiese infinito lo que resolvimos fue para representar mejor las funciones emplear una escala variable en el eje Y, de forma que siempre hubiese un punto por encima de la mitad de la escala y todos cupiesen en su totalidad. Esto nos llevó también a ver que no servían escalas fijas en el eje X, debimos rehacer el código y permitir cambiar asimismo la escala del eje de las x, dado que si alcanzaba valores muy grandes la imagen de sus potencias positivas se dispara rápidamente, al igual que la imagen de sus potencias negativas en caso de que adopte valores muy pequeños, pero debíamos acercarnos mucho al 0 relativamente a la escala total para poder reflejar bien la curvatura. También tuvimos que emplear un set diferente de puntos en el caso del logaritmo pues para poder apreciar bien su gráfica debíamos acercarnos tanto al 0 que en caso de tener potencias negativas se desbordaría rápidamente.

El siguiente problema a la hora de representar las funciones fue como calcular la imagen de hecho de cada punto. Nuestro primer intento fue calcular una aproximación mediante la fórmula de Taylor, pero ni tan siquiera empleando 10 términos obteníamos aproximaciones razonables. Demostramos matemáticamente que en efecto tal aproximación era imposible empleando series y similares dado que $1/x$ tiene una singularidad en el 0, y como no queríamos usar un divisor porque retrasaría de forma considerable el cálculo de las imágenes de los puntos, algo que hacemos numerosas veces, decidimos cargar con una tabla la imagen inversa de todos los puntos, lo cual por supuesto restringe mucho que selección de puntos usaremos para calcular su imagen. Además esta solución sería fácilmente generalizable a cualquier tipo de función mucho más complicada que la inversa.

Luego tuvimos que calcular la recta que une dos puntos ya pintados en el plano. Lo primero que advertimos es que las funciones no tienen por qué ser inyectivas, por lo tanto para pintar entre dos puntos del plot tendríamos que cargar todos los anteriores. Viendo que esta opción se complicaba demasiado decidimos que dado que en una función cada punto tiene una única imagen dado un pixel del eje X con tan solo la información que ese pixel nos proporciona podemos determinar toda su columna, por lo tanto cambiamos la RAM para pintar por columnas, lo cual ya habíamos hecho, pero lamentablemente no

habíamos conservado la RAM por columnas y debimos rehacerla.

Asimismo el algoritmo que une dos puntos del plano fue complejo de idear pues al trabajar con pixels discretos no podemos trazar la recta analítica que pasa por esos puntos, sino que tenemos que ir saltando de un lado a otro de ella intentando acercarnos lo más posible. También tuvimos que dar la vuelta al problema para el caso de que la pendiente fuese negativa. Además esta parte fue realizada en navidad, cuando no disponíamos de FPGA, lo cual nos obligó a simular una memoria con 128 filas de 128 bits que estaban giradas y volteadas. El hecho de que estuviesen giradas y volteadas hacia más sencillo el código y la forma de mostrarlo por pantalla pero dificultaba realmente la simulación, aunque en un principio no pensamos que fuesemos a tener que simular la memoria. La mejor solución que encontramos fue hacer capturas de pantalla del contenido de la memoria, juntarlas, voltearlas y girarlas con el Photoshop para poder saber con seguridad como se vería en pantalla. Esto supuso un esfuerzo extra, sobretodo en tiempo. Ver fotos en el anexo Ver la función sin FPGA.

Además como no todos los miembros del equipo dominaban mirar el contenido de la RAM (dado que nos pareció más razonable que una sola persona se dedicase a esto y el resto avanzasen otras partes del proyecto), por lo que juntar las capturas con Photoshop y editarlas no solo había que hacerlo porque mejoraba la visualización, sino también para que lo vieran el resto de miembros. Esto fue un inconveniente cuando se estaba comprobando que otras partes del proyecto funcionasen y era necesario saber lo que se estaba pintando.

La parte obligatoria de nuestro proyecto, a falta de probarla, estaba terminada en navidades, cuando pudimos empezar con ampliaciones, y de hecho en cuanto volvimos de navidades tras dos días de pruebas en el laboratorio habíamos terminado la parte obligatoria, y continuamos con las ampliaciones.

Al principio del proyecto, tuvimos problemas de duplicados que nos hicieron perder mucho tiempo, pero los problemas fueron solucionados y gracias al tiempo invertido aprendimos como resolver este tipo de problemas, lo que fue muy útil en otras ocasiones. A pesar de las dificultades que nos encontramos con la pantalla conseguimos que las funciones se pintasen lo suficientemente rápido como para que fuese inapreciable a la vista que las columnas se van modificando una a una. Este es sin duda uno de nuestros grandes logros, pues en un principio se nos pidió que se pintases despacio e incluso a trozos, lo que se hubiese apreciado muchísimo. Esto lo conseguimos con una memoria de doble puerto. También logramos simular que se borrara la pantalla al hacer un reset, cosa que no se nos había pedido, dado que borrar la memoria era demasiado complicado. Por eso nos las ideamos para que pareciese que se borrase pero sin borrar realmente, solo dejando de pintar.

Otra mejora significativa fue la reducción de la RAM, pasando de tener 256 filas a 128. Cuando nos dimos cuenta que la pantalla hacía más grande una dimensión que la otra (prácticamente duplicaba una dimensión), nos pareció sensato duplicar la otra dimensión,

pasando a tener 256 filas x 128 columnas. En un principio guardamos las filas repetidas pero después nos dimos cuenta que esto era innecesario si pintábamos cada fila 2 veces, lo que nos ahorra una cantidad de memoria considerable (pasando a tener finalmente 128 x 128).

Llegó un momento en el que empleábamos más recursos de los que tenía la FPGA, IOB para ser exactos, porque para comunicar diferentes módulos pasábamos demasiada información de forma simultánea. Xilinx no daba aviso alguno de este hecho y nos costó darnos cuenta de lo que ocurría. Después como es lógico tuvimos que rediseñar todo nuestro sistema para que fuera calculando todo 1 a 1 para cada punto, tuvimos que poner una señal a todos los componentes que les indicase si debían estar trabajando o a la espera de que otros terminasen. No obstante fue imposible hacer esto para el conversor, pues necesita la imagen de cada uno de los puntos para calcular la escala y la escala para enviar a la VGA cada uno de los puntos con el número adecuado de bits, por lo que tuvimos que almacenar la información de todos los puntos, procesarla, y después enviarla 1 a 1.

Tras ensamblar todas las componentes del proyecto cuando sintetizábamos funcionaba perfectamente, pero al configurar la FPGA hacía cosas completamente diferentes, porque el reloj de la FPGA era demasiado rápido para los módulos empleados. Ver fotos en el anexo Problemas con el reloj. Buscando este error invertimos una gran cantidad de tiempo y esfuerzo, dado que para hacer pruebas era imprescindible probar con la FPGA y el tiempo requerido para sintetizar y configurarla era superior a 10 minutos. También tuvimos que considerar la opción de usar dos relojes diferentes, aunque al final al ver la velocidad con la que pintaba las funciones hicimos que toda la VGA funcionase con el reloj más lento para trabajar con solo un reloj. No obstante había algunas funciones específicas que se pintaban mal sin motivo aparente, y dado que el margen del reloj del módulo de cálculo y el de la pantalla era muy pequeño introdujimos un tercer reloj, cambiando todo el código. Al final el fallo no era ese, sino un desbordamiento porque al principio en un módulo usábamos la librería unsigned pero tras comprobar que funcionaba hicimos modificaciones y pasamos a usar la librería signed, en efecto comprobamos que siguiese sin haber problemas pero cometimos un fallo de cálculo y no nos dimos cuenta de que se desbordaba. Ver fotos de desbordamientos en el anexo Desbordamientos.

A lo largo de la práctica hemos debido realizar varias multiplicaciones y algunas divisiones, pero no disponemos de espacio suficiente para meter más de un multiplicador, por lo que decidimos emplear solo uno que estuviese en el módulo de cálculo, para dividir dado que solo realizamos divisiones entre unos pocos números desplazamos el vector y lo multiplicamos por una constante que es un tercio, y para realizar las multiplicaciones que no estaban en el calculador, por ejemplo a la hora de unir la imagen de dos puntos, lo que hicimos fue aprovechar la evolución de los estados para calcular la multiplicación con sumas sucesivas.

El Xilinx no reconoce que una variable estado esté acotada y no sintetiza al ser el

vector al que se accede con ella demasiado pequeño. Este fue un problema recurrente, su primera aparición y cuando se encontró su solución fue trabajando con el teclado. Se solucionó haciendo el vector más grande e ignorando sus bits más significativos. No obstante en determinadas ocasiones el Xilinx sí reconocía estas cotas y permitía tener vectores más pequeños, en algunos casos así está programado.

A la hora de representar las asíntotas de una función cuando dividimos por 0 dado que rellenamos nuestra RAM por columnas debemos escribir en ella un vector que tenga solo 1 hasta determinada posición y 0 en adelante, o viceversa. El tamaño del vector en cuestión es de 128 bits, por lo que un condicional en función de la variable de entrada no es factible, por lo que hicimos una declaración parcial del vector: hasta esa posición todo 1 y en las restantes todo 0. Eso fue implementado asignando solo a una parte del vector, y luego a la restante, pero el Xilinx no reconoce que estemos asignando un valor a todas las componentes del vector y coloca ahí un latch. Sabemos por qué se da ese fallo y tenemos opciones de cómo solucionarlo, pero son excesivamente costosas, es por una interpretación errónea del Xilinx y afecta de forma apenas apreciable al rendimiento de la aplicación.

Aunque ya lo hemos comentado en un apartado, el hecho de que tarde más de 10 minutos en sintetizar y configurar la FPGA retrasa considerablemente el trabajo en este proyecto, pues probamos algo para ver si hemos solucionado el fallo, suponemos cual va a ser la salida de la FPGA con el nuevo cambio y continuamos trabajando en consecuencia pues no podemos estar de brazos cruzados esperando, pero numerosas veces no acertamos con cual era el fallo y fue esfuerzo perdido. Esto también ocurrió aunque en menor medida en navidades al no poder bajar el código a la FPGA.

En cuanto al teclado los coeficientes han de introducirse relativamente despacio, sin pulsar varias teclas a la vez y sin pulsar una tecla demasiado tiempo. Esto se debe a que el teclado que descargamos de internet estaba pensado para este tipo de interacción, pausada, con el usuario. Escogimos este modelo porque para representar funciones no necesitamos una gran velocidad de interacción con el teclado.

Dado que hemos empleado código generado de forma automática por Matlab hay algunos ligeros fallos de precisión debido al epsilon máquina que aparece, por ejemplo algunos senos y cosenos que deberían evaluarse a 0 salen 2^{-10} . Para solucionarlo tan solo deberíamos sustituirlos uno a uno, pero dado que conocemos la causa y su poca significancia creemos que no merece la pena la sustitución.

6. Reparto de trabajo

El reparto de trabajo fue equitativo y tratamos que todos los miembros del equipo trabajasen en todas las partes del proyecto, aunque por la necesidad de avanzar más rápido Ana se centró en la parte de leer y escribir en memoria y escribir por pantalla, incluyendo el cálculo de la línea que une dos puntos, y el diseño gráfico; Aitor en la lectura por teclado, en el cálculo de la imagen de los puntos y en la algoritmia general (elección de las escalas y de los puntos de muestra, pintar la línea que une dos puntos, etc) y Víctor en el ensamblado, programación de los puntos de muestra y del reescalado, y el uso del Matlab.

Esto no significa que no trabajásemos en otras partes del proyecto. Además nos ayudamos cuando teníamos algún problema y revisábamos la parte de nuestros compañeros cuando había algún fallo o era necesario para implementar algo relacionado con esa parte, todos hemos revisado y programado en todas las partes del proyecto.

7. Programas y bibliografía

Programas utilizados

- **Xilinx:** Utilizado para desarrollar el proyecto.
- **Matlab 2013:** Utilizado para hacer pruebas sobre como representar las funciones, generar tablas, etc.
- **Photoshop CS5:** Utilizado para editar imágenes, como por ejemplo para juntar las capturas de la RAM.
- **NotePad++:** Utilizado cuando no disponíamos de Xilinx o por necesidad de algunas de sus funciones como la de comparar dos ficheros.

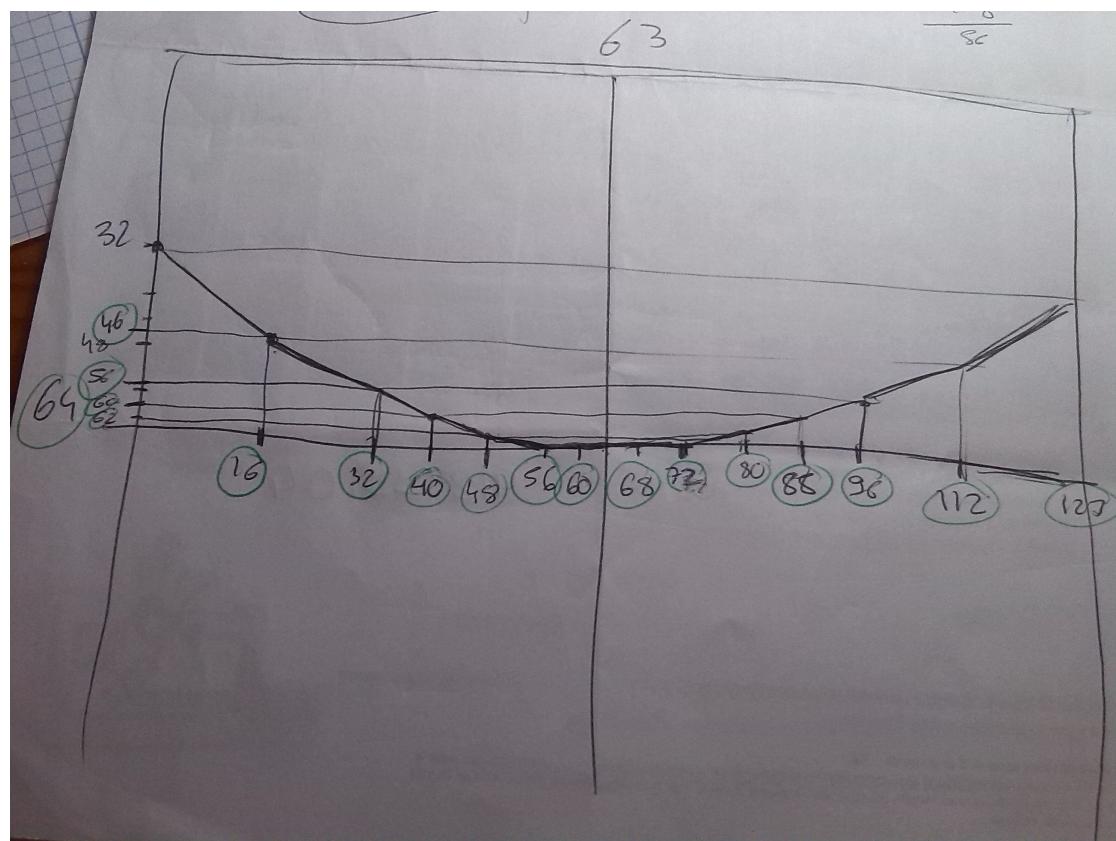
Bibliografía

- Diapositivas de teoría del Campus Virtual de la asignatura “Tecnología y Organización de Computadores”.
- RAM de doble puerto proporcionada por el profesor, con las debidas modificaciones.
- Vgacore.vhd (módulo original de la pantalla), proporcionado por el profesor, con las debidas modificaciones.
- El módulo del teclado fue modificado a partir de un teclado descargado de internet, no hemos realizado ningún componente de los que emplea reconocedor.vhd.
- <http://www.mathworks.es/es/help/matlab/index.html> : documentación de Matlab.

7. Anexo

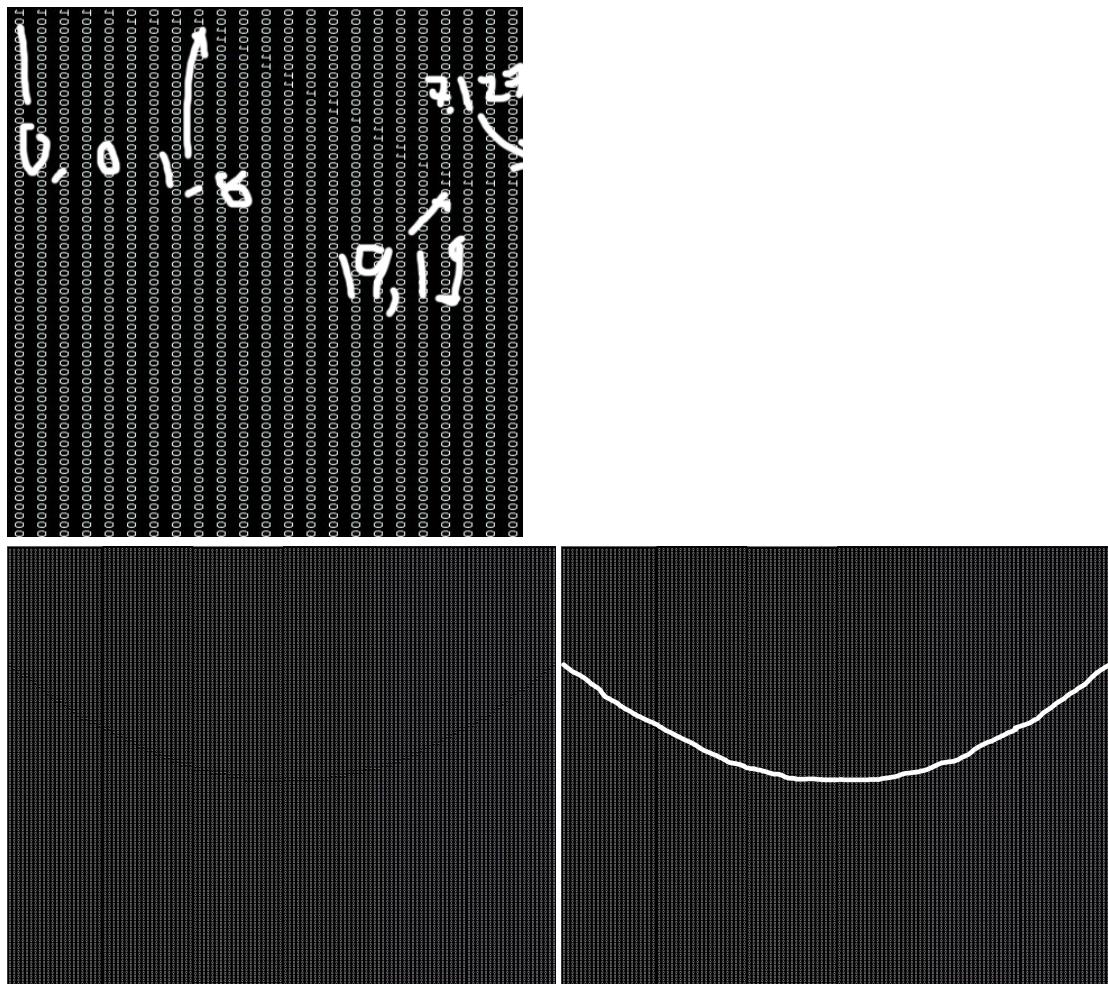
Ver la función sin FPGA

1. Realización en papel

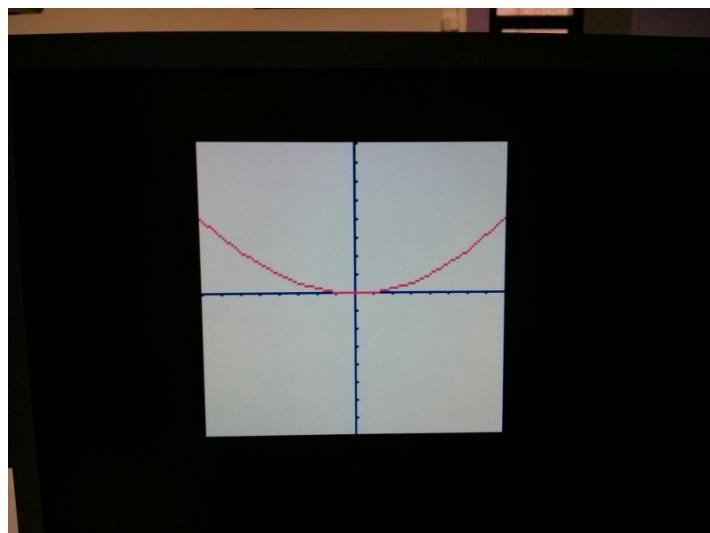


2. Contenido de la RAM, 128 filas de 128 bits giradas y volteadas

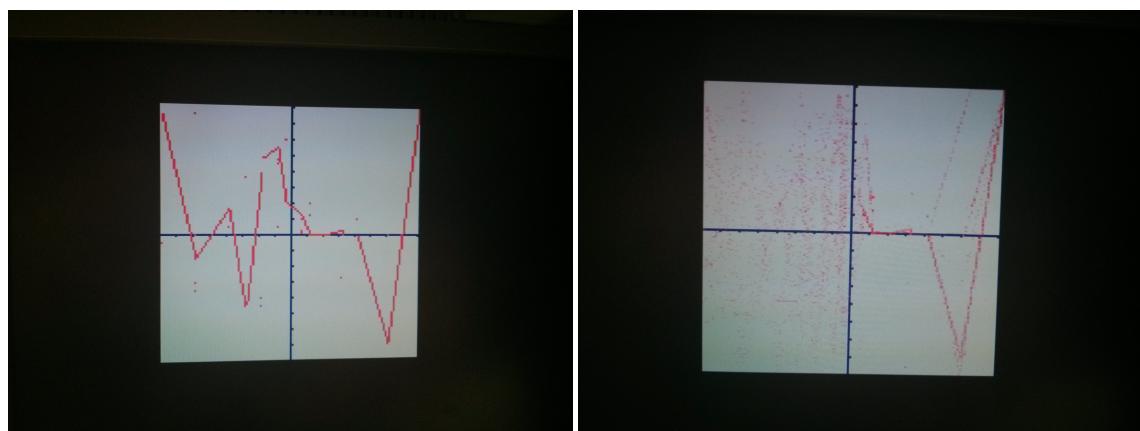
3. Capturas de la memoria juntadas con Photoshop



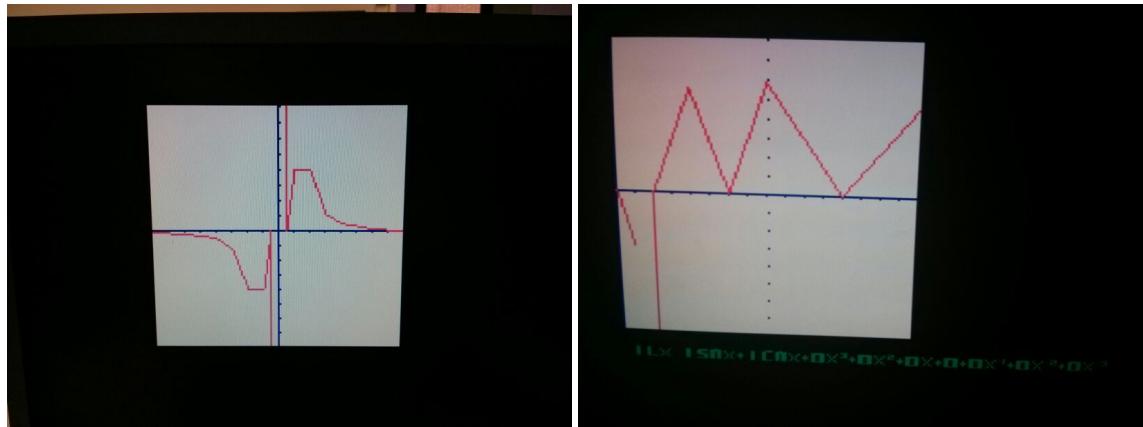
4. Resultado final en la FPGA (Coincidiendo con el esperado)



Problemas con el reloj



Desbordamientos



Programas en Matlab

Se incluyen los siguientes scripts de Matlab:

- **polplot.m**: permite plotear una función (dada por sus coeficientes) en pantalla, evaluando en unos puntos. Permitió ir realizando pruebas con las escalas
- **genera.m**: contiene un conversor de reales a nuestra representación en coma fija (modificable por el usuario), así como los diversos sets de puntos de muestra donde evaluamos la función, y funciones para generar las tablas de números que usa calculo.vhd