FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico Final
(Versión B)

Diseño de un motor de rotación gráfico 3D simplificado Basado en el algoritmo CORDIC

Integrantes:

Axelrud Federico 94395

Forte Martín 94378

Zomero Alejandro 94980

Objetivos

El presente trabajo tiene como objetivo el desarrollo de una arquitectura de rotación de un cubo en 3D basado en el algoritmo CORDIC. La misma será implementada en una FPGA, donde se utilizará la interfaz serie UART para enviar desde una PC (mediante el Matlab) las coordenadas del cubo a rotar, una serie de pulsadores que indicarán el sentido y el eje de la rotación, y un controlador de video que permitirá la visualización del cubo girando en tiempo real.

A partir del diseño y descripción de la arquitectura para su construcción, se desarrollará mediante el lenguaje VHDL y se simularán en ModelSim cada uno de los bloques que componen la arquitectura para corroborar su correcto funcionamiento. Una vez hecho esto se sintetizará mediante la herramienta ISE para luego cargar el programa en la FPGA a través del Adept.

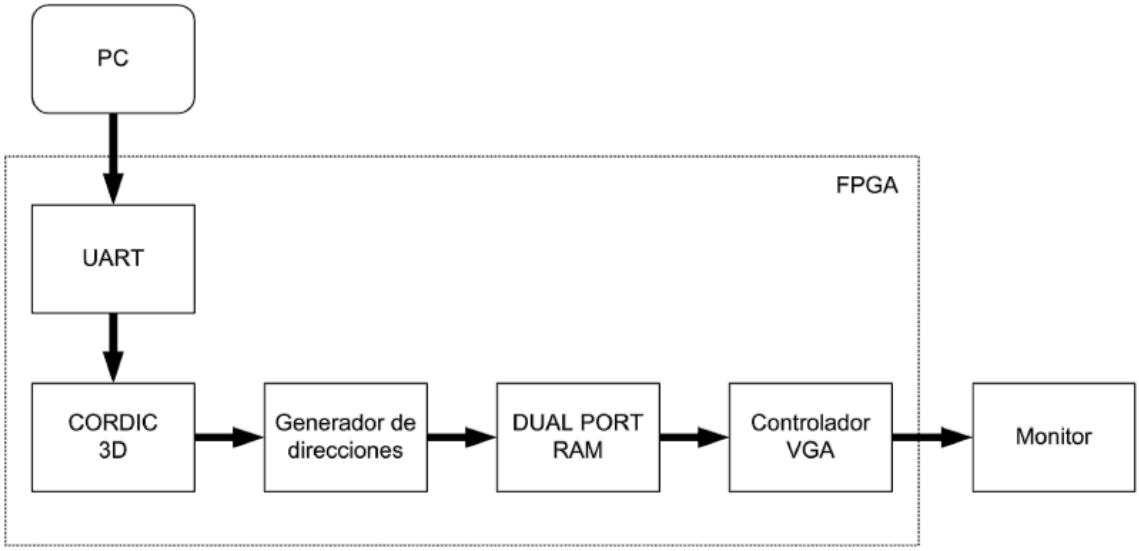

Figura 1. Esquema completo del sistema

Desarrollo

Para comenzar describiremos el comportamiento de la interfaz serie *UART*. Desde Matlab construimos las coordenadas correspondientes al cubo que queremos ver rotar en la pantalla luego de todo el trabajo que iremos detallando.

Eligiendo un baud rate de 1200 bps y conformando vectores de números enteros positivos (es decir, posicionando el cubo en el primer octante), el script de matlab se encarga de enviar una por una las coordenadas en vectores de 8 bits más un bit de start y uno de stop. Con un adaptador USB-RS232 y gracias a la interfaz serie UART construida, se irán cargando en un buffer, de aproximadamente 300 posiciones de 8 bits, que llamaremos "buffer de almacenamiento". El tamaño del buffer no es inconveniente ya que son pocos los puntos que utilizamos para formar el cubo (y la dual port RAM utilizada es de 480x480). El mismo está parametrizado con la letra "M" en el código. El cubo diseñado puede ser de

distintos tamaños pero debe ser menor a x<60, y<60, z<60 por cuestiones que están vinculadas con el controlador de video diseñado que luego explicaremos.

Dejaremos acá la explicación del buffer de almacenamiento para luego volver a llamarlo entremezclándolo con el comportamiento de los demás bloques del sistema.

Saltando al llamado "bloque de pulsadores", continuaremos con la explicación del funcionamiento del proyecto. El mismo está conformado por 3 "pulsadores" independientes que están conectados a 6 interruptores o pulsadores físicos en la placa Nexys 2 (2 por cada "pulsador"). Dado el abuso del lenguaje cabe destacar las diferencias entre "pulsador" y pulsador físico o interruptor. "Pulsador" hace mención al algoritmo implementado mientras que el pulsador físico o interruptor son los componentes físicos tangibles de la placa que presionamos o activamos.

Cuando un interruptor o pulsador físico es presionado, un contador cuenta, valga la redundancia, 1 millón de clocks y suma '1' al angulo correspondiente al interruptor asociado en el sentido apropiado. Este '1' se traduce a un paso angular de 0,703125 dadas las longitudes de vector y la representación que utilizamos para el acumulador de angulo: 11 bits (1 de signo, 8 de dato y 2 decimales - para tener mayor precisión). Por otro lado, dado que la frecuencia de reloj a la que trabaja la FPGA es de 50 MHZ, contando 1 millón de clocks entonces estarán transcurriendo 20 mili segundos para que sume 0,703125 grados al acumulador del ángulo, lo cual hace llegar a una velocidad angular de 35,15625 grados por segundo, tal como era necesario.

Cada interruptor o pulsador físico de los dos asociados a un eje está correlacionado a un contador independiente para que el sentido de giro sea el correcto y no se produzcan errores. Es decir, los ejes de rotación son independientes y puede ser presionado más de un pulsador físico a la vez, pero no dos que correspondan al mismo eje, ya que la lógica del algoritmo implementado no permite que se presionen los dos pulsadores que hacen girar en sentidos opuestos en un mismo eje de rotación.

Ante cualquier interruptor o pulsador físico presionado, este bloque enviará un flag a la dual port RAM. Como consecuencia, la misma comenzará a borrarse. Una vez que termine de limpiarse le avisará al buffer de almacenamiento, antes mencionado, para que le envíe las coordenadas correspondientes al CORDIC. Junto con los ángulos a rotar, que le provee el "bloque de pulsadores", el CORDIC (que es el corazón del proyecto), podrá rotar las coordenadas y entregar a la salida las componentes giradas en los ángulos indicados.

Tal como se observa en el diagrama en bloques, la salida del cordic la tomará un "buffer pre dual" que es el buffer ubicado antes del controlador de video para "clockear" correctamente el sistema y obtener los resultados deseados. El mismo es el responsable de hacer la proyección plana, ya que borra la coordenada z (simplemente la desprecia).

El CORDIC implementado es asincrónico y esto trajo algunos problemas ya que a veces su tiempo de cálculo es mayor a un ciclo de reloj, lo cual provoca una generación de puntos intermedios indeseados. Esto fue solucionado dominando los tiempos a los cuales el buffer almacenamiento le envía los datos al Cordic. En otras palabras, al buffer de almacenamiento le llega el flag de la dual port RAM indicando que está lista para que la

empiecen a cargar, y éste entonces comienza a enviar los datos (de a 3 coordenadas por vez – x,y,z), pero entre punto y punto deja pasar un tiempo prudente para que el Cordic pueda realizar los cálculos y llegar al resultado correcto en su salida. Para manejar este tiempo, el buffer controla un flag que le indica al "buffer pre dual" cuando ya está listo el siguiente dato para que lo levante y lo envíe al controlador de video ("bloque mux vga").

Adentrándonos en el "Bloque Cordic" se debe mencionar que el mismo posee 3 bloques "cordic pre-post", cuyo nombre hace referencia a que está compuesto a su vez por 3 módulos más pequeños encargados de hacer el pre procesamiento, el algoritmo cordic propiamente dicho y el post procesamiento (tal como se observa en la figura 2).

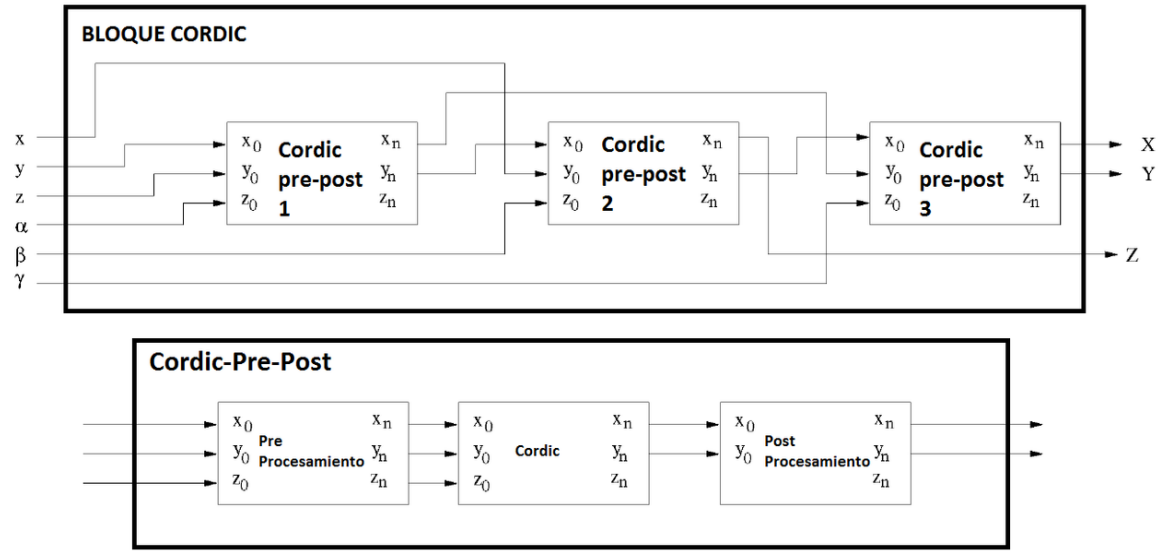

Figura 2: Diagrama del bloque Cordic

Por un lado, debemos relatar la razón del pre procesamiento y por otro lado la del post. El pre procesamiento está vinculado con una transformación de las coordenadas cuando el ángulo es mayor a 90 grados o menor a -90, dado que el cordic puede operar entre esos límites. Si el ángulo a girar no se encuentra allí dentro, entonces se deben pre procesar las coordenadas mediante una transformación lineal, para que queden en ese rango y haga la correcta operación. Por otro lado, el post procesamiento está relacionado con la ganancia inherente que posee el algoritmo Cordic. Luego de realizadas todas las operaciones, el vector resultante posee una ganancia que hay que ajustar para que el resultado sea el correcto y no esté amplificado.

Vale aclarar que el Cordic utilizado trabaja con un total de 11 bits, los cuales están compuestos por: 1 bit de signo + 8 + 2 decimales para precisión.

Por último, desarrollaremos una breve explicación del controlador de video. En principio, mencionaremos que la dual port ram se instanció a partir de una block RAM disponible en la FPGA.

La dual port RAM borrará todo su contenido cada vez que se presiona un pulsador y haya nuevos datos para cargar. Las posiciones X e Y de la salida que nos entrega el Cordic, y que son transferidas en el tiempo adecuado al "buffer pre dual" se mapearán en una RAM de 480x480. Para lograr esto, el algoritmo parte del centro de la RAM de 480x 480. La posición Y va a indicar la fila para arriba o para abajo según sea correcto, mientras que la

posición X va a colocar un ‘1’ ya sea más a la derecha o a la izquierda del centro apropiadamente. De esta manera se irán generando las direcciones de cada punto que se desea mostrar en pantalla. A su vez, la bien llamada dual port RAM, es leída continuamente por el controlador VGA que utilizamos en el trabajo práctico 2. Es decir que, mientras cargamos la memoria con ‘1’s mapeando las posiciones en las direcciones adecuadas, el controlador VGA lee continuamente la misma para exponer en el monitor los resultados del cubo 3D girado por el Cordic y proyectado en dos dimensiones.

Cabe aclarar si bien el cordic entrega un vector de tamaño 11 (1 bit de signo + 8 + 2 decimales de precisión) y el bit de signo es utilizado para generar la dirección de memoria adecuada, los dos bits más significativos de los 10 restantes no son tenidos en cuenta. Dicho de otro modo, los bits mapeados en la memoria son los 8 menos significativos de los 11 que nos entrega el Cordic. Este criterio fue seleccionado debido a que se prefirió mostrar una mayor precisión del cubo a costa de que sea más pequeño. A la vez, el tamaño de la memoria se consideró de 480x480, por lo cual, si nos posicionamos en el medio podemos lograr 240 bits para un lado y 240 para el otro. Teniendo en cuenta que 240 en binario es 11110000, y que de los 8 bits, 6 son de dato y 2 son decimales de precisión, como máximo se puede mostrar un cubo de 60 puntos (60 para un lado y 60 para el otro). El 60 se desprende de suprimir los últimos dos bits del 240, de acuerdo a lo mencionado recientemente.

En la Figura 3 se puede observar la cantidad de slices, Flip-Flops, LUTs utilizados y además se indica el porcentaje de utilización en base a las cantidades que hay disponibles.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	339	9,312	3%	
Number of 4 input LUTs	1,529	9,312	16%	
Number of occupied Slices	975	4,656	20%	
Number of Slices containing only related logic	975	975	100%	
Number of Slices containing unrelated logic	0	975	0%	
Total Number of 4 input LUTs	1,764	9,312	18%	
Number used as logic	1,529			
Number used as a route-thru	235			
Number of bonded IOBs	19	232	8%	
Number of RAMB16s	18	20	90%	
Number of BUFGMUXs	1	24	4%	
Number of MULT18X18SIOs	7	20	35%	
Average Fanout of Non-Clock Nets	2.91			

Figura 3: Device utilization summary

Diagrama en bloques

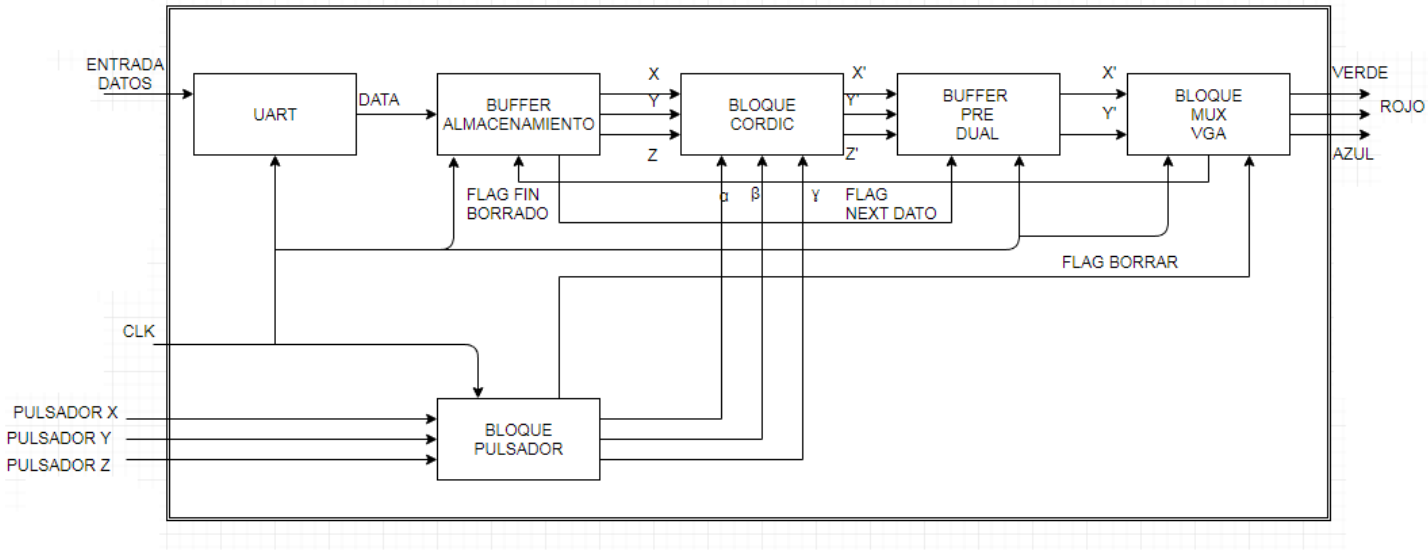

Figura 4: Diagrama en bloques completo

Conclusiones

Luego de todo el trabajo realizado, el esfuerzo aplicado y la investigación llevada a cabo, notamos la maravillosa y poderosa herramienta que es una FPGA. Captamos la dimensión de poder diseñar hardware abstrayéndonos de los problemas físicos que pueda traer apareados construir la arquitectura elaborada, para así focalizarnos en la sincronización, la optimización, y la obtención del resultado más eficiente que pueda lograrse. Nos conduce a gran cantidad de aplicaciones con diversos fines, nos permite ampliar nuestro espectro de posibilidades y entusiasmarnos con aplicaciones de alta complejidad, ya que, por ejemplo, programándose de manera remota uno podría estar diseñando hardware a la distancia y tener un dominio poderoso de hardware a la distancia (aplicaciones satelitales).

Por ello es que consideramos que este proyecto nos abre la puerta a un mundo de aplicaciones inimaginables de otra manera.