Universidad de Buenos Aires

Facultad de Ingeniería

1° Cuatrimestre de 2022

88.41

Sistemas Digitales

Trabajo práctico final

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

Integrantes:
Sanchez, Marcelo Fernando <marce_chez@msn.com>

Padrón:
87685

17 de marzo de 2024

Índice

1. Objetivo **3**

2. Especificaciones **4**

3. Diseño **5**
3.1. Dual Port RAM - memoria de video 5
3.2. Número de iteraciones Cordic 5
3.3. Representación de las coordenadas 6
3.4. Procesamiento de las coordenadas 7

4. Arquitectura **8**
4.1. gral_ctrl 9
4.2. uart2sram 10
4.3. sram_ctrl 11
4.4. sram2cordic 12
4.5. rotador3d 13
4.6. xilinx_dual_port_ram_sync 15
4.7. vga_ctrl 16
4.8. Resumen de dispositivos utilizados 16

5. Conclusiones **17**

6. Referencias **18**

7. Apéndice **19**
7.1. Repositorio con código fuente 19
7.2. Scripts en Octave para conversión de coordenadas 19
7.3. Script en Python para envío de coordenadas por UART 20

1. Objetivo

En el presente Trabajo Práctico el alumno desarrollará una arquitectura de rotación de objetos 3D basada en el algoritmo CORDIC. El objetivo principal es desarrollar tanto la unidad aritmética de cálculo como así también el controlador de video asociado. Para la realización completa del Trabajo Práctico se cargarán en memoria externa las coordenadas correspondientes a un objeto tridimensional predefinido mediante una interfaz serie UART. A partir de los valores de las componentes, se rotará el objeto alrededor de cada uno de los ejes de coordenadas según el valor que adquieran las entradas del sistema, y por último las componentes rotadas serán presentadas en un monitor VGA mediante la aplicación de una proyección plana. En la Figura 1.1 puede observarse un diagrama en bloques del sistema completo. Deberá determinarse la cantidad mínima de bits de ancho de palabra (bits de precisión) para alcanzar las especificaciones requeridas

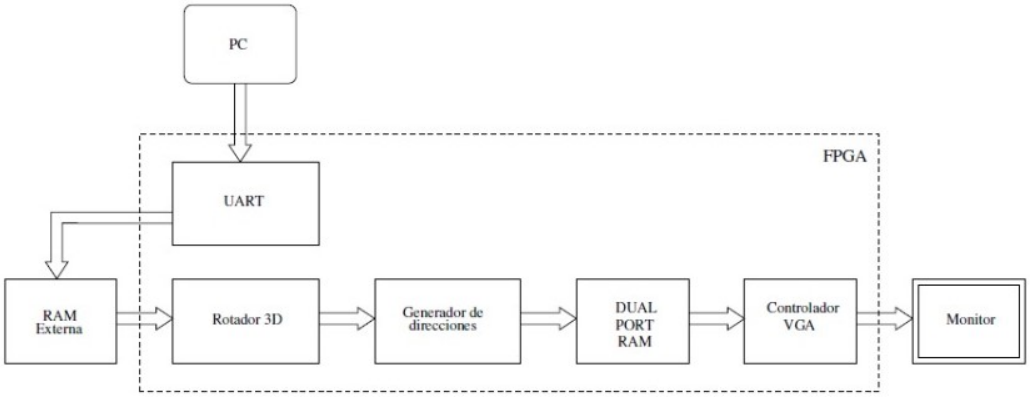


Figura 1.1 – Esquema completo del sistema.

2. Especificaciones

- Resolución de video: 640 x 480 1 bit monocromo 50 Hz.
- Velocidad angular de rotación mínima en c/u de los ejes: 35.15625 grados/seg.
- Paso angular Δ_Φ: 0.703125 grados.
- Dispositivo: Cyclone II EP2S35 (Kit de desarrollo DE2 Board) o Spartan 3E-500 (Kit Nexys 2 Board).

3. Diseño

En esta sección se muestran los cálculos realizados para dimensionar cada uno de los componentes utilizados. En principio se calculó el tamaño de la memoria RAM interna para alojar las coordenadas de video a graficar. A partir de este se hizo una estimación de la precisión requerida del ángulo resultante y así se determinó la cantidad de iteraciones del algoritmo para finalmente decidir el formato y longitud de las coordenadas de entrada.

En el último apartado se muestra una breve explicación del camino que recorren los datos desde su adquisición hasta su visualización, detallando los pasos intermedios de transporte, transformación y almacenamiento.

3.1. Dual Port RAM - memoria de video

Como se muestra en el esquema general del sistema (figura 1.1), una vez rotadas las coordenadas se utiliza un componente denominado Dual Port Ram integrado en la FPGA, para almacenar las coordenadas que finalmente se muestran en la pantalla VGA. Específicamente el dispositivo Spartan 3E-500 (Kit Nexys 2 Board), utilizado para este trabajo, tiene una capacidad de almacenamiento de 368640 bits.

Dada la resolución de video de 640 X 480 píxeles, se optó por representar la porción cuadrada del centro de la pantalla (320 x 320 píxeles). De esta manera se utilizaron 9 bits por cada eje, quedando un vector de direccionamiento de 18 bits X 1 bit de datos, como se ilustra en la figura 3.1.

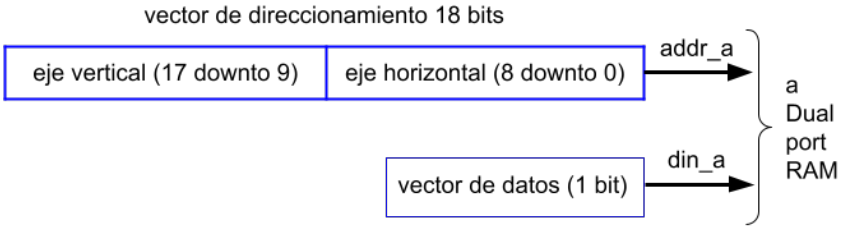


Figura 3.1 – Vectores de direcionamiento y datos para memoria de video

3.2. Número de iteraciones Cordic

Siendo la resolución de video elegida de 320 X 320, se puede representar el contorno de la figura a rotar como un círculo unitario normalizado a 160 píxeles. De aquí que el paso mínimo representable en la pantalla cumple con la siguiente ecuación:

$$\Phi_{min-video} = \arctan \frac{1}{160} = 6{,}25 \times 10^{-3}\,\text{rad} \tag{3.1}$$

Luego, la precisión mínima requerida para para diferenciar entre 2 píxeles debe cumplir con:

$$\Phi_{min-representable} < \frac{\Phi_{min-video}}{2} = 3{,}12 \times 10^{-3}\,\text{rad} \tag{3.2}$$

Para lograr esta precisión se puede ver de la tabla 3.2 que se necesitan nueve o más iteraciones del algoritmo Cordic.

Dados estos resultados se decidió trabajar con trece iteraciones para así obtener tres dígitos significativos en el resultado final del ángulo rotado.

i	radianes	grados
0	0.78539816	45
1	0.46364761	26.5650512
2	0.24497866	14.0362435
3	0.12435499	7.12501635
4	0.06241881	3.57633437
5	0.03123983	1.78991061
6	0.01562373	0.89517371
7	0.00781234	0.44761417
8	0.00390623	0.2238105
9	0.00195312	0.11190568
10	0.00097656	0.05595289
11	0.00048828	0.02797645
12	0.00024414	0.01398823
13	0.00012207	0.00699411
14	6.1035E-05	0.00349706
15	3.0518E-05	0.00174853

Figura 3.2 – Tabla de grados rotación en función de iteraciones Cordic.

3.3. Representación de las coordenadas

Habiendo elegido trece iteraciones fijas para el algoritmo. Las coordenadas a rotar deben tener al menos doce dígitos por la naturaleza misma del algoritmo, ya que en cada iteración se '*shiftea*' un dígito hacia la derecha para sumar/restar (ver figura 3.3). Para este trabajo se optó por trabajar con trece dígitos en punto fijo $Q_{1,12}$ con bit de signo, resultando un total de catorce bits para representar las coordenadas.

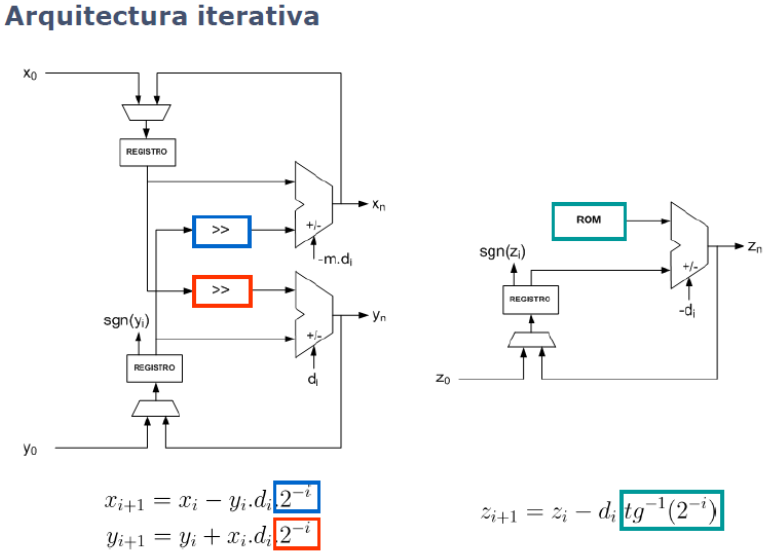


Figura 3.3 – Arquitectura iterativa del algoritmo Cordic.

3.4. Procesamiento de las coordenadas

En este último apartado se muestra la transformación de las coordenadas en cada etapa del diseño, desde su fuente original (PC) hasta llegar a la memoria de video en la FPGA para su visualización.

Primero se analizó la capacidad de almacenamiento de la RAM externa del Kit Nexys 2. El manual dice que tiene incorporada una RAM de 128 Mbit modelo *Micron M45W8MW16 Cellular RAM pseudo-static DRAM* organizada en 8 Mbytes x 16 bits, con un bus de direccionamiento de 24 bits y un bus de datos de 16 bits [1]. Por lo que se optó por transformar las coordenadas de cada eje en 16 bits con el formato descripto en la sección anterior, siendo solo los 14 bits menos significativos los utilizados para el procesamiento. Se utilizó un script en Octave para transformar las coordenadas originales a 16 bits.

Teniendo las coordenadas transformadas se envían vía UART hacia la memoria RAM externa del Kit. Una vez almacenadas, se pasa a la lectura y rotación individual de cada coordenada, transformación final en coordenadas VGA y almacenamiento en la Dual Port RAM. En forma paralela estos datos son leídos por el controlador VGA y transportados a una pantalla para su visualización.

Se muestra el camino completo de estas transformaciones en la figura 3.4.

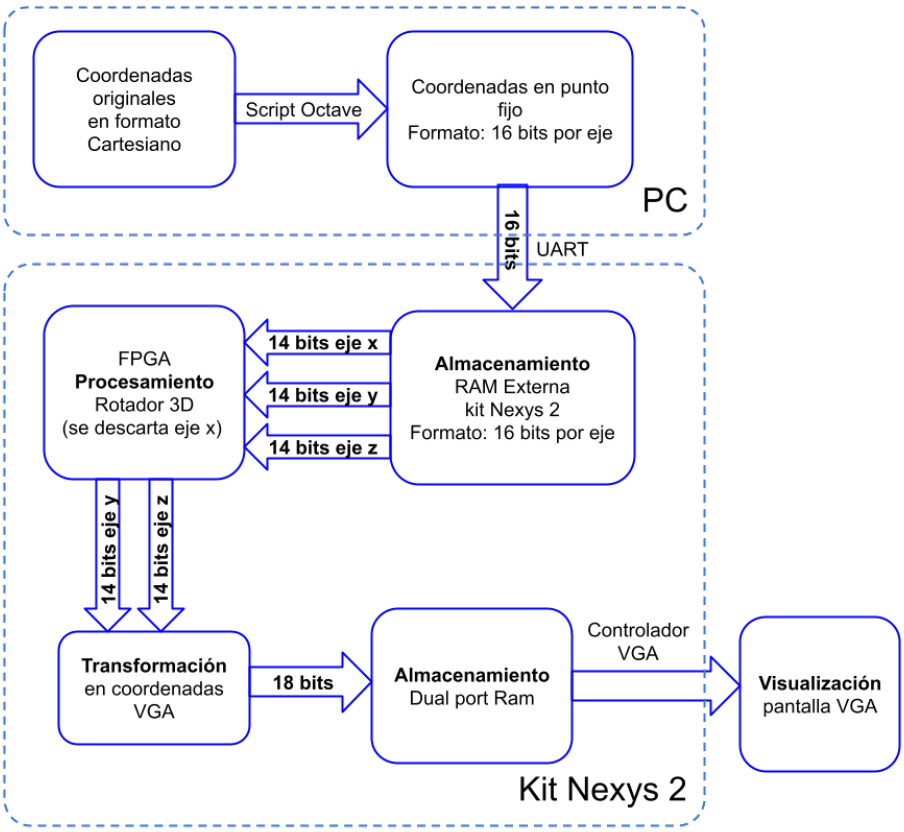


Figura 3.4 – Esquema del recorrido y transformación de las coordenadas desde su adquisición hasta la visualización.

4. Arquitectura

En esta sección se describen los componentes principales que forman la arquitectura dentro de la FPGA. En la figura 4.1 se puede ver la arquitectura completa.

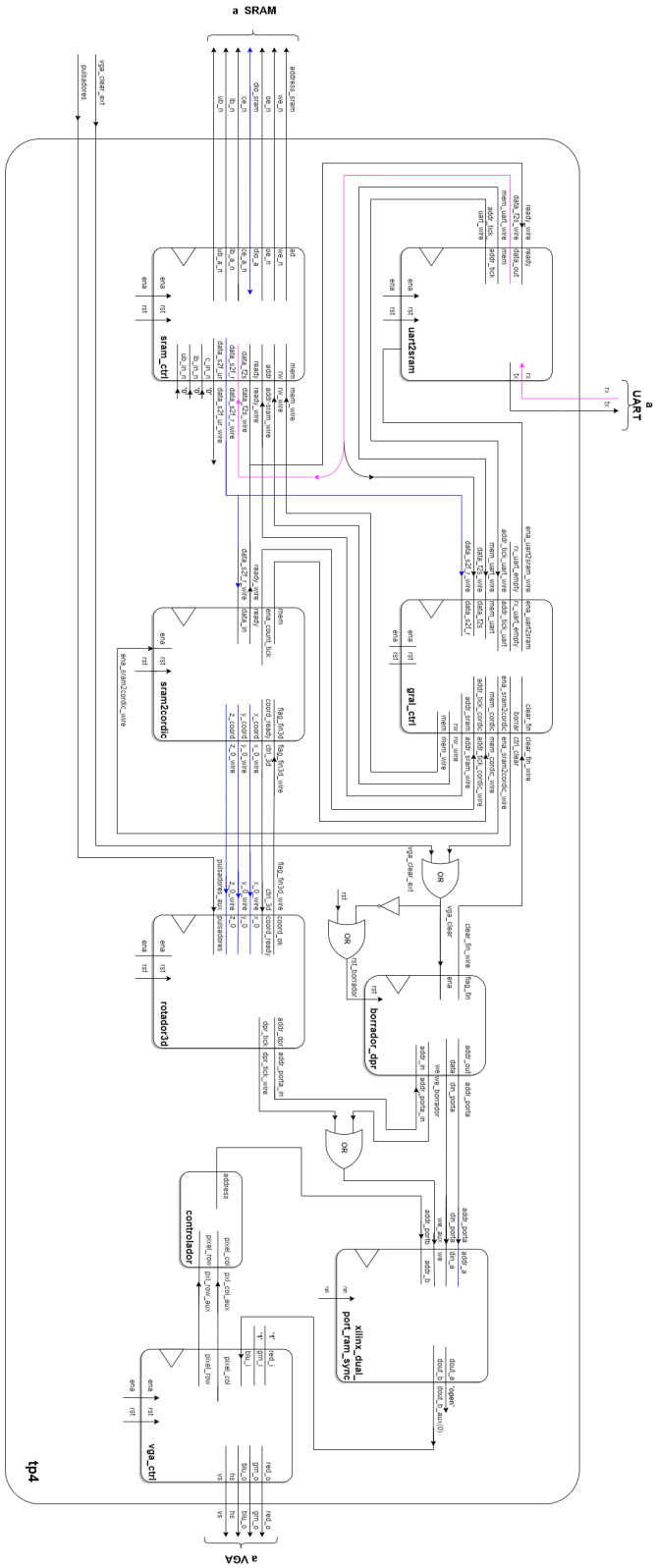


Figura 4.1 – Arquitectura del sistema.

El componente `gral_ctrl` es el encargado de controlar las tareas principales dentro de la FPGA. Se diseñó en base a una maquina de estados. Luego se asignó un componente para cada tarea:

- **uart2sram.** Su tarea es leer desde el puerto de entrada `rx` las componentes enviadas en serie con protocolo uart y almacenar 2 bytes (2 palabras en este protocolo), para poder ser escrito en la memoria externa.
- **sram_ctrl.** Es el componente encargado de escribir y leer las componentes desde la RAM externa hacia la FPGA y viceversa. Sus principales tareas comprenden direccionamiento, almacenamiento temporal, generación de señales de escritura/-lectura y *flags* de fin de operación.
- **sram2cordic.** Su principal tarea es preparar las coordenadas que son leídas desde la SRAM. Obtiene de a una coordenada a la vez y se encarga de almacenarlas hasta recibir las tres coordenadas. Una vez completada esta tarea da aviso al `rotador3d`. Luego espera que se termine la operación actual de rotación para comenzar de nuevo con la lectura.
- **rotador3d.** Es el procesador del sistema, el que se encarga de rotar todas las componentes, también lleva incorporado el `generador_direcciones` que mapea las coordenadas resultantes en una dirección para ser almacenado en la Dual Port RAM por el puerto A.
- **borrador_dpr.** Se encarga de hacer un barrido de borrado por todas las direcciones de la Dual Port RAM antes de la próxima rotación.
- **dual_port_ram y vga_ctrl.** `vga_ctrl` se encarga enviar hacia el puerto VGA del Kit las señales para su visualización. Utiliza el puerto B de la `dual_port_ram` para tomar las coordenadas rotadas.

En las secciones siguientes se describe con mayor detalle el funcionamiento de cada componte.

4.1. gral_ctrl

Su funcionamiento se basa en una maquina de estados. Se definieron cinco estados: REPOSO, CARGA_DATOS, ROTACION, ESPERA_ANG, REFRESH_DPR. Una vez habilitado con la señal `ena = '1'`, el sistema comienza a operar en estado REPOSO.

Cuando se detecta una palabra en el buffer de la UART (señal `rx_empty = '0'`) se pasa al estado CARGA_DATOS que habilita el funcionamiento del componente `sram_ctrl` en modo escritura para la carga de datos en la SRAM externa a la FPGA.

Para determinar el fin de las coordenadas se definió una palabra denominada `EOF_WORD = "1111111111111111"`, una vez detectada se pasa al estado ROTACION. En este estado se habilita el componente `sram2cordic` y el componente `sram_ctrl` pasa a modo lectura.

Una vez leídas y rotadas todas las componentes (nuevamente monitoreando la lectura de `EOF_WORD`) se pasa al estado ESPERA_ANG. En este estado se espera un flag (`flag_rotnew = '1'`) que es habilitado cuando se detecta un cambio en el ángulo de rotación. Detectado el flag se pasa al estado REFRESH_DPR.

REFRESH_DPR se encarga de habilitar el componente `borrador_dpr` para hacer un barrido de borrado a la memoria interna (Dual Port Ram). Se espera a la finalización del borrado (señal `clear_fin = '1'`) para comenzar el ciclo ROTACION/ESPERA_ANG/REFRESH_DPR nuevamente.

Se muestra el diagrama de estados en la figura 4.2.

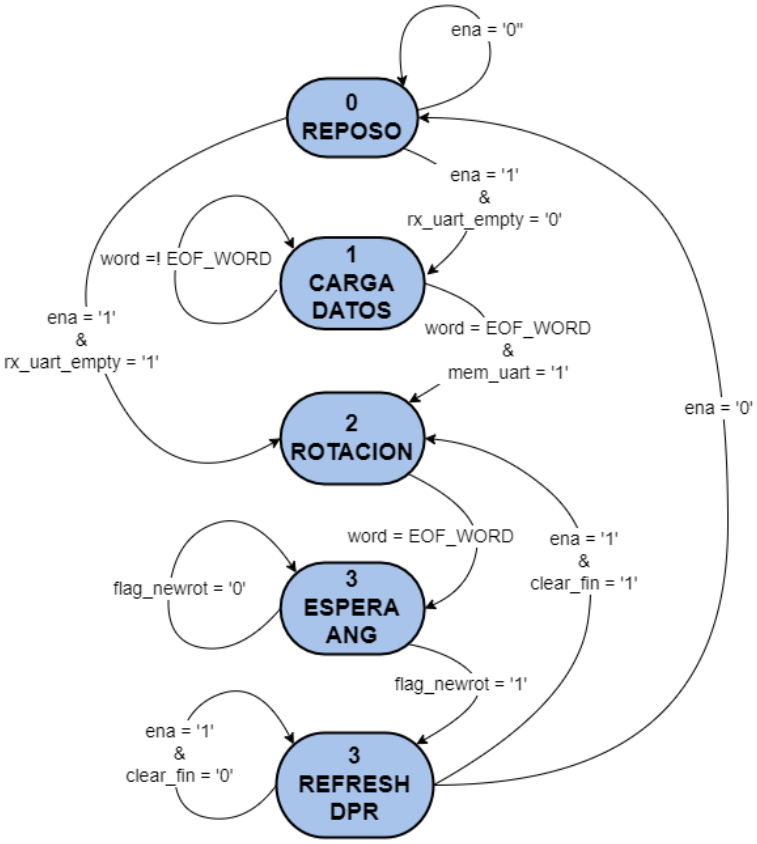


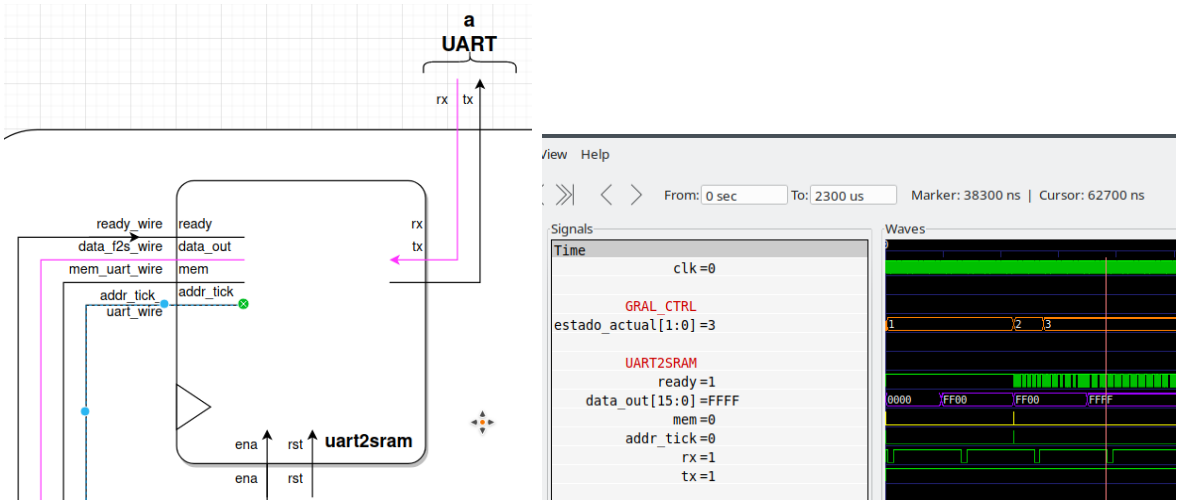
Figura 4.2 – Diagrama de estados de gral_ctrl

4.2. uart2sram

Se tomaron los componentes UART del libro [2](listing `7.1 uart`, `7.3 uart_tx`, `7.4 uart_rx`, `4.20 fifo`) y se realizó un *wrap* con el fin adaptar las señales al componente `sram_ctrl`.

Incorporando una maquina de estados, se leen 2 bytes desde la uart y se guardan temporalmente en dos registros, luego se escriben en la RAM externa. Se generan las señales hacia `sram_ctrl` para escritura. El proceso se repite hasta completar los datos requeridos (cuando llega la palabra `EOF_WORD`).

En la figura 4.3 se puede ver el componente, sus señales de entrada/salida y una simulación en la etapa de escritura. La figura 4.4 muestra el diagrama de estados.



(a) Componente **(b)** Simulación

Figura 4.3 – Simulación del componente uart2sram donde se muestran las señales involucradas en el proceso de carga de datos.

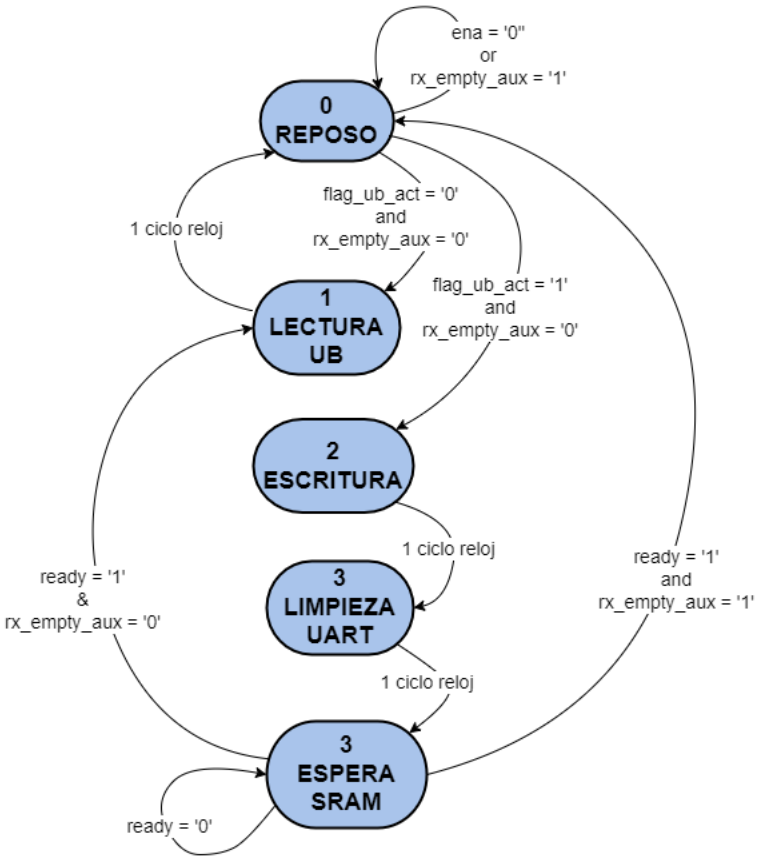


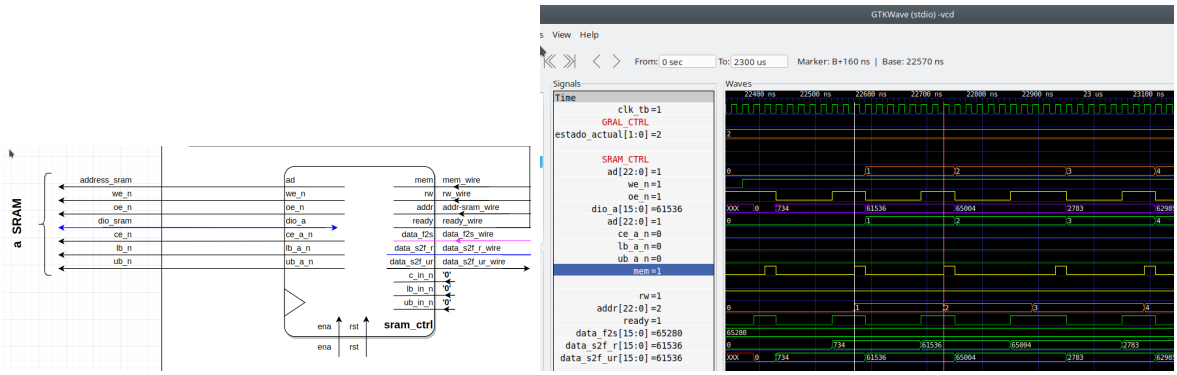
Figura 4.4 – Diagrama de estados de uart2sram

4.3. sram_ctrl

Se tomó el diseño del libro [2] (`Listing 10.1`) y se adaptaron los tiempos para generación de señales de lectura/escritura. Específicamente para este trabajo se realizan

las operaciones de forma asincrónica. Y dado que la SRAM provista por el Kit posee un ciclo de 70 ns [3], se cambió de 40 ns a 100 ns el tiempo para completar estas operaciones.

En las figuras 4.6 se muestra el componente con sus entradas/salidas y una simulación donde se ven las señales involucradas.



(a) Componente **(b)** Simulación

Figura 4.5 – Simulación del componente sram_ctrl, se muestran las señales involucradas en el proceso de lectura.

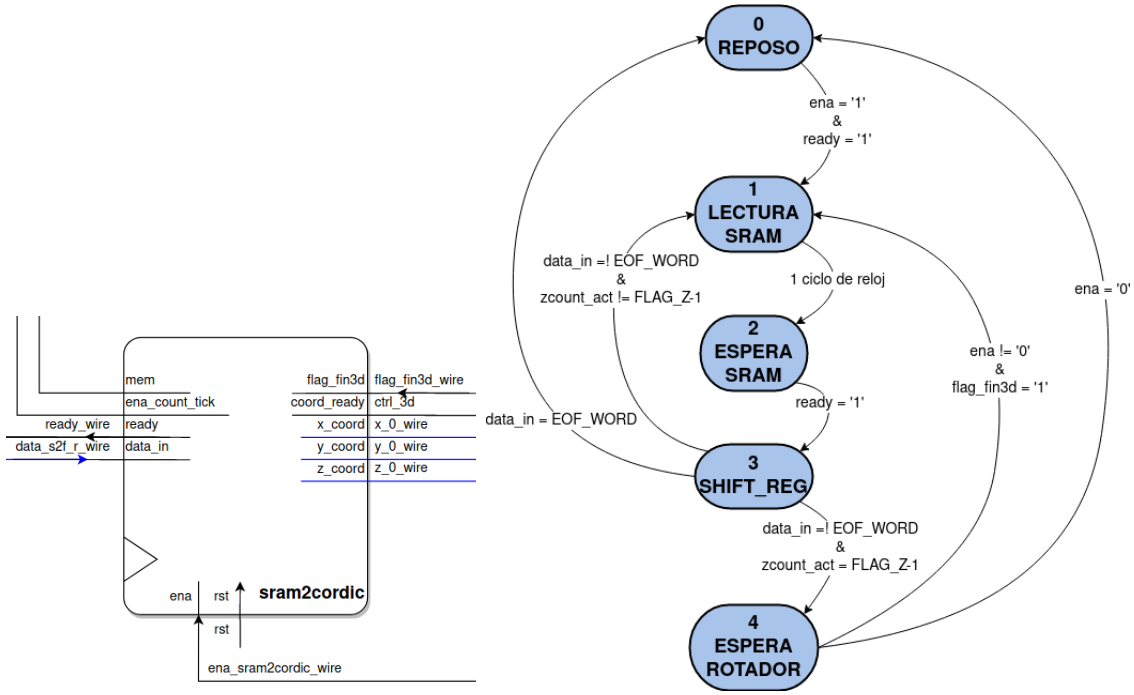
4.4. sram2cordic

Como su nombre lo indica este componente toma los datos de la RAM externa y los prepara para la etapa de rotación cordic. Su principal tarea es manejar el componente `sram_ctrl` en modo lectura y almacenar las tres coordenadas para que las tome el rotador3d e inicie la rotación.

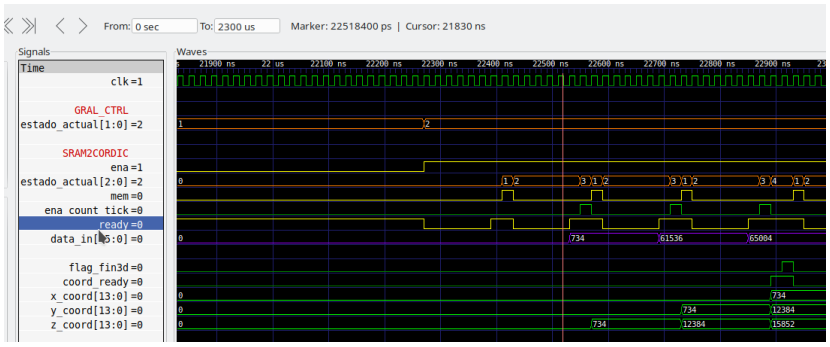
Además de manejar las señales para lectura desde la SRAM genera otros flags para avisar cuando los datos están listos para ser procesados y espera la señal de fin de rotación para comenzar nuevamente el proceso de lectura.

Nuevamente el proceso se diseñó con una maquina de estados. Se definieron cinco estados:

- REPOSO. Es el estado inicial, el cual cambia una vez que cambia la señal `ena = '1'` manejada por `gral_ctrl`.
- LECTURA_SRAM. Comienza el proceso de lectura enviando la señal `mem = '1'` a `sram_ctrl`. Dura solo un ciclo de reloj para generar esta señal.
- ESPERA_SRAM. En este estado se espera la señal `ready = '1'` desde `sram_ctrl` para poder almacenar la coordenada leída en un registro.
- SHIFT_REG. Dura solo un ciclo de reloj para generar la señal `wr_reg_tick = '1'` de guardado de registro. Luego pasa al estado LECTURA_SRAM y se repite el proceso dos veces mas. Una vez completado los tres registros se pasa al estado ESPERA_ROTADOR
- ESPERA_ROTADOR. En este estado las tres coordenadas quedan disponibles para lectura y rotación por parte del rotador. Se espera a la señal de fin de rotación (`flag_fin3d = '1'`) para comenzar el proceso nuevamente de la próxima terna de coordenadas.



(a) Componente **(b)** Máquina de estados



(c) Simulación

Figura 4.6 – Componente sram2cordic, maquina de estados y simulación. Se muestra en la simulación la carga de tres coordenadas y la espera de fin de rotación, pasando por todos los estados.

4.5. rotador3d

Este componente es el encargado de realizar la rotación. Tiene como entradas las coordenadas X Y Z, los pulsadores para rotación y como salida la dirección de memoria del vector rotado. Los ángulos de rotación los genera internamente con una tasa de incremento/decremento de 20 ms y un paso angular Δ_Φ: 0.703125 grados.

La arquitectura está conformada por: una maquina de estados que coordina las tareas, un generador de ángulos de rotación, un generador de direcciones para la memoria VGA y tres componentes Cordic para la rotación en cada eje.

Se definieron cinco estados: REPOSO, SHIFT_REG, ROTAR, SHIFT_REG_DPR, ESCRITURA_DPR. Para comenzar a operar (una vez habilitado desde `gral_ctrl`), se parte del estado REPOSO y se espera a que las coordenadas estén disponibles a la entrada (señal `coord_ready = '1'` desde `sram2cordic`) para pasar a SHIFT_REG. En este estado se genera la señal para guardar internamente las coordenadas y comenzar la rotación, dura un ciclo. Luego se pasa al estado ROTAR donde se ejecuta el proceso

de rotación, se espera la señal de fin de rotación para pasar a SHIFT_REG_DPR, donde se ejecuta el guardado de las coordenadas rotadas en tres registros locales. Luego se pasa a ESCRITURA_DPR que es la instancia donde se guarda en memoria RAM las coordenadas rotadas previa transformación. El proceso se repite para todos los vectores almacenados.

En la etapa de rotación el componente `cordic_3d` toma a su entrada los vectores de rotación α, β, γ calculados por `rotacion_ctrl` que a su vez tiene como entrada los pulsadores.

El cálculo por cada vector se realiza en cascada, es decir, se espera a que termine una rotación para comenzar la siguiente. Tiene una duración de 860 ns por vector, que resulta en un total de 10,2 ms para 11946 vectores de coordenadas.

En la figura 4.7 se muestra la arquitectura. Se puede ver que la maquina estados maneja las señales de habilitación del cordic3d y de los componentes externos `sram2cordic` y `dual_port_ram`. También se pueden observar los registros de entrada y salida de los vectores, el generador de angulos (`rotacion_ctrl`) a partir de los pulsadores de entrada y el habilitador cada 20 ms.

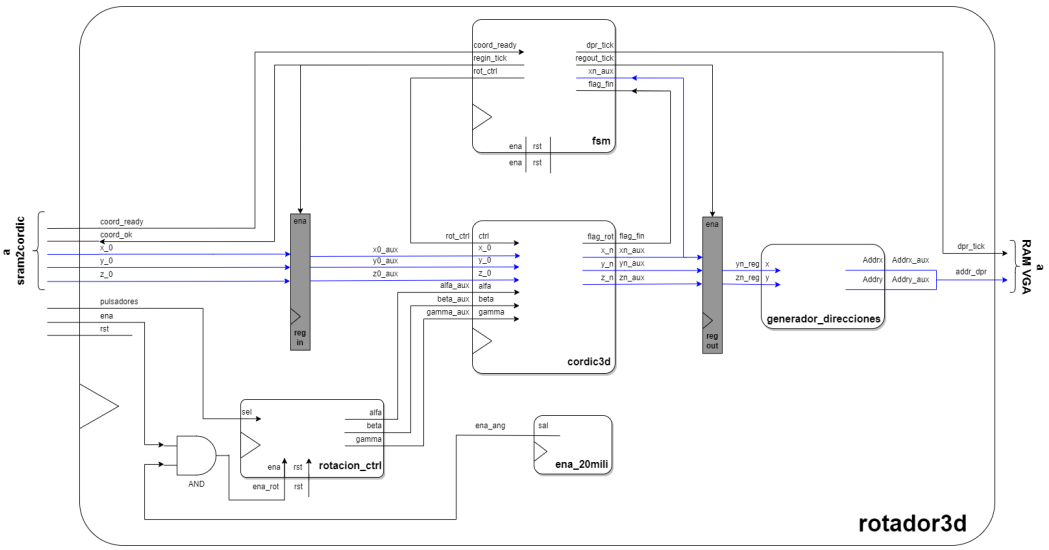


Figura 4.7 – Arquitectura del componente rotador3d.

En la figura 4.8 se muestra el diagrama de estados, luego en la figura 4.9 la simulación de rotación para un vector donde se pueden ver todas las señales y el tiempo total de 860 ns.

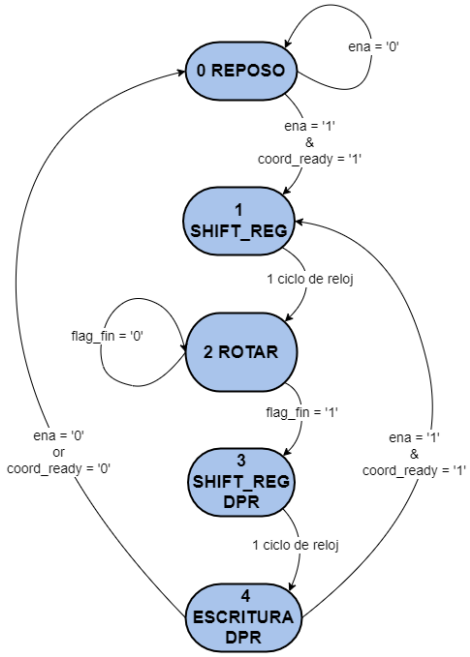


Figura 4.8 – Diagrama de estados dentro del componente rotador3d.

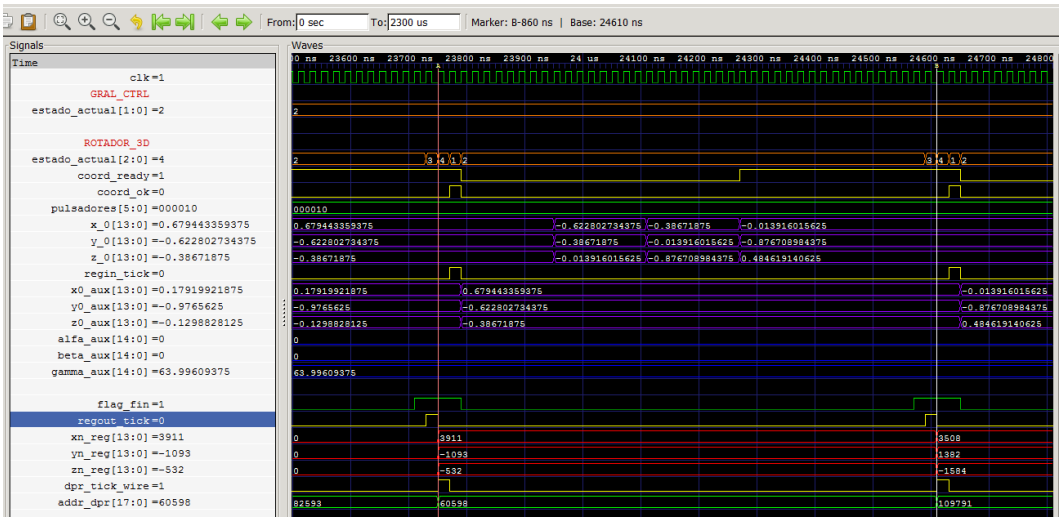


Figura 4.9 – Simulación del componente rotador3d.

4.6. xilinx_dual_port_ram_sync

Se tomó el diseño del libro [2] (`Listing 11.4`) que utiliza un *template* del manual *XST v8.1 i* provisto por el programa ISE. Este *template* está descripto por comportamiento e infiere automáticamente un módulo de memoria dentro de la FPGA. Solo es necesario elegir las dimensiones de direccionamiento y datos.

Para este proyecto se utilizó una memoria de dos puertos (Dual Port Ram), sincrónica, con 18 bits de direccionamiento y 1 bit de datos. Se muestra a continuación la declaración de entidad del componente, donde se pueden ver los purtos de entrada y salida.

Definición de puertos del componente Dual_Port_Ram

```
entity xilinx_dual_port_ram_sync is
  generic(
    ADDR_WIDTH: integer:=18;
    DATA_WIDTH: integer:=1
  );
  port(
    clk: in std_logic;
    we: in std_logic;
    rst: in std_logic;
    addr_a: in std_logic_vector (ADDR_WIDTH-1 downto 0);
    addr_b: in std_logic_vector (ADDR_WIDTH-1 downto 0);
    din_a: in std_logic_vector (DATA_WIDTH-1 downto 0);
    dout_a: out std_logic_vector (DATA_WIDTH-1 downto 0);
    dout_b: out std_logic_vector (DATA_WIDTH -1 downto 0)
  );
end;
```

4.7. vga_ctrl

Finalmente el componente `vga_ctrl` es el encargado de graficar los puntos almacenados en la Dual Port Ram. Para ello toma los datos por el puerto B con la ayuda del componente denominado `controlador`, que utiliza las señales de salida `pxl_row` y `pxl_col` para disponer los datos a la entrada de `vga_ctrl` en el tiempo correcto. Se le agregó a este último (el `controlador`) una máscara para que solo se grafique la porción central de 320 X 320 pixeles.

4.8. Resumen de dispositivos utilizados

En esta sección se muestran los dispositivos utilizados dentro de la FPGA luego de ser sintetizado con la herramienta ISE. Se puede ver en la tabla que efectivamente se sintetizaron 16 dispositivos de RAMB 16s, para la Dual Port Ram. También 8 multiplicadores, que fueron utilizados en las conversiones y escalamiento de las coordenadas.

Lista de dispositivos utilizados

Utilización Lógica	usados	Disponible	% Utilización
Flip Flops	592	9312	6 %
LUTs de 4 entradas	1820	9312	19 %
Slices	1032	4656	22 %
IOBs	99	232	42 %
RAMB 16s	16	20	80 %
BUFGMUXs	1	24	4 %
MULT18X18SIOs	8	20	40 %
Average Fanout of Non-Clock Nets	3,28		

Tabla 4.1 – Tabla resumen de síntesis del diseño completo.

5. Conclusiones

El trabajo permitió poner en práctica la teoría desarrollada a lo largo de la materia. Específicamente la rotación con el algoritmo Cordic se implementó de manera satisfactoria, demostrando que se puede realizar un rotador con solo dos operaciones básicas: suma y shifteo. También, el seleccionar un dispositivo específico permitió pensar en los recursos disponibles para luego dimensionar todas las variables presentes. Ejemplos de esto último fueron: capacidad de RAM interna y externa a la FPGA, velocidad de escritura/lectura de la RAM externa. Una elección importante, fue agregar al diseño máquinas de estado en casi todo el desarrollo. Esto presentó una ventaja, ya que muchas de las partes son procesos que se describen perfectamente con máquinas de estado, haciendo el código más compacto, legible y mantenible en el tiempo.

Con todas estas variables y restricciones se llegó al objetivo, logrando un rotador en los tres ejes en tiempo real, utilizando el protocolo UART para la carga de datos, una SRAM para el almacenamiento externo y visualización a través de una pantalla VGA.

6. Referencias

Referencias

[1] Digilent. (2008, january) Digilent nexys2 board reference manual. [Online]. Available: https://digilent.com/reference/programmable-logic/nexys-2/reference-manual

[2] P. P. Chu, *FPGA Prototyping by VHDL Examples*, three ed. Cleveland State University: WILEY, 2008.

[3] *Async/Page/Burst CellularRAMTM 1.5 MT45W8MW16BGX*, Micron Technology, Inc., 2004.

7. Apéndice

7.1. Repositorio con código fuente

El código completo se encuentra en el repositorio https://github.com/SanchezMarceloF/8641-sist-dig-tp4/.

7.2. Scripts en Octave para conversión de coordenadas

archivo_ptofijo.m

```
function archivo_ptofijo(file, ROW, COL, N_ROWS, DECIM)
% file: ruta a archivo
% ROW: numero de fila para comenzar a leer desde archivo
% COL: numero de fila para comenzar a leer desde archivo
% N_ROWS: numero de filas a leer del archivo
% DECIM: factor de decimaci n

  close all
  %clear all
  clc

  %datos = dlmread('../files/coordenadas.txt', '\t', 5, 0);
  datos = dlmread(file, '\t', ROW, COL);

  N = 16;    %longitud del numero
  M = 12;    %numero de decimales
  %N_ROWS = 11946;

  for i = 1:N_ROWS/DECIM
  for j=0 : 2
    datos_ptofijo(i , (j*N+1):((j+1)*N)) = decimal_a_ptofijo(N, M,
        datos(i*DECIM, j+1));
    datos_decimados(i, j+1) = datos(i*DECIM, j+1);
  endfor
  endfor
  % agregado del fin de archivo
  datos_ptofijo(N_ROWS/DECIM+1, 1:N*3) = dec2bin((2^(N*3))-1);

  % plot(datos_decimados(:,2), datos_decimados(:,3),".")

  % salida en punto fijo
  datos_ptofijo
  % salida en decimal
  % for i=1 : N_ROWS+1
    % for j=0 : 2
      % result = bin2dec(num2str(datos_ptofijo(i , (j*N+1):((j+1)*N)))
          );
      % printf("%i\t",result);
    % endfor
    % printf("\n");
  % endfor

  % guarda en archivo
  filename =  file(1:length(file)-4);
  printf("output file: ");

```

```
45   if (DECIM > 0)
46     output = [filename "_ptofijoDEC" num2str(DECIM) "-16.txt"]
47   else
48     output = [filename "_ptofijo-16.txt"]
49   endif
50
51   dlmwrite(output,   datos_ptofijo, "delimiter", "");
52
53 endfunction
```

decimal_a_ptofijo.m

```
1  function ret = decimal_a_ptofijo (N, M, x)
2  %convierte un numero entre 0 y 1 en pto fijo con bit de signo y:
3
4   %N longitud total del numero
5   %M numero de decimales
6   %x decimal a convertir
7
8    if (x<0)
9    % le sumo 2^N si es negativo porque dec2bin solo acepta numeros
10     % positivos
11     y = x*(2^M)+2^(N)+0.5;
12   else
13     y = x*(2^M);
14   endif
15
16   if (y >= (2**N))
17     ret = dec2bin(0,N);
18   else
19       ret = dec2bin(round(y),N);
20   endif
21
22 endfunction
```

7.3. Script en Python para envío de coordenadas por UART

bits2serial.py

```
1  import serial
2  import time
3  from os import listdir #, getcwd, chdir #mkdir,startfile
4  from os.path import isfile, join #, exists, dirname
5  from sys import argv
6
7  path = '../test_files/'
8  # ser = serial.Serial('/dev/ttyUSB0')  # open serial port linux
9  ser = serial.Serial('COM5')  # open serial port windows
10 ser.baudrate = argv[1]
11 ser.bytesize = 8
12 ser.stopbits = 1
13 parity = 'N'
14 print(ser)
15
16 txtfiles = [f for f in listdir(path) if isfile(join(path, f))\
17              and '-16.txt' in f]
18 n = 0
```

```
19 print('\nList of files:\n')
20 for file in txtfiles:
21   print(f'[{n}] {file}')
22   n = n+1
23
24 file = int(input('\nSelect number of file: '))
25
26
27 with open('../test_files/' + txtfiles[file]) as infile:
28 # with open('../test_files/coord_linea_ptofijo-16.txt') as infile:
29     for line in infile:
30         print(f'line\t\t: {line}, type : {type(line)}')
31         # Initialize a binary string
32         input_string=int(line[0:48], 2)
33         print(f'input_string\t: {input_string}, type : {type(
               input_string)}')
34         #Convert these bits to bytes
35         input_array = input_string.to_bytes(6, "big")
36         print(f'input_array\t: {input_array}, type : {type(input_array
               )}')
37         ser.write(input_array)
38
39 time.sleep(2)
40 ser.close()
```