



# Trabajo práctico 8

## Integración DSP y uP

---

Autores:

- Gerard Brian
- Raimondi Marcos

## Índice

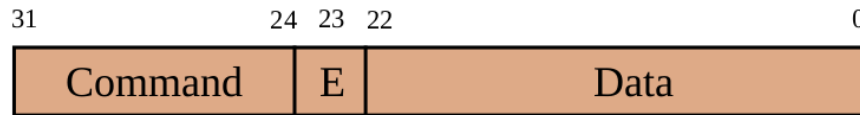
<b>Consigna</b>	<b>2</b>
<b>Resolución</b>	<b>3</b>
Memoria ram	3
File Register	4
Integración File Register - Memoria Ram	5
Pruebas del funcionamiento en la FPGA	7

## Consigna

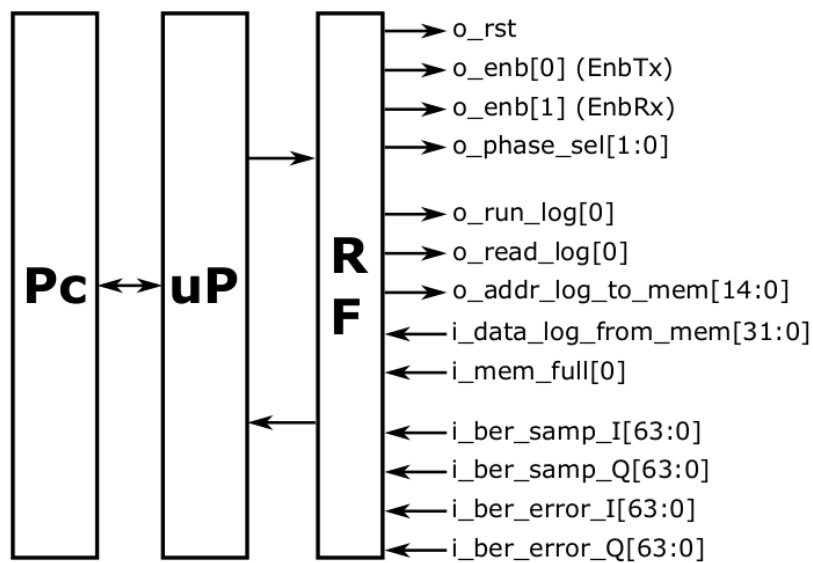
### - Ejercicio 1

Utilizando los desarrollos anteriores integrar el DSP y el uP tomando como referencia los siguientes lineamientos:

- El GPIO de salida se utiliza para escribir los comandos en el Register File (RF) particionando los 32bits en command[31:24], enable[23] y data[22:0] .

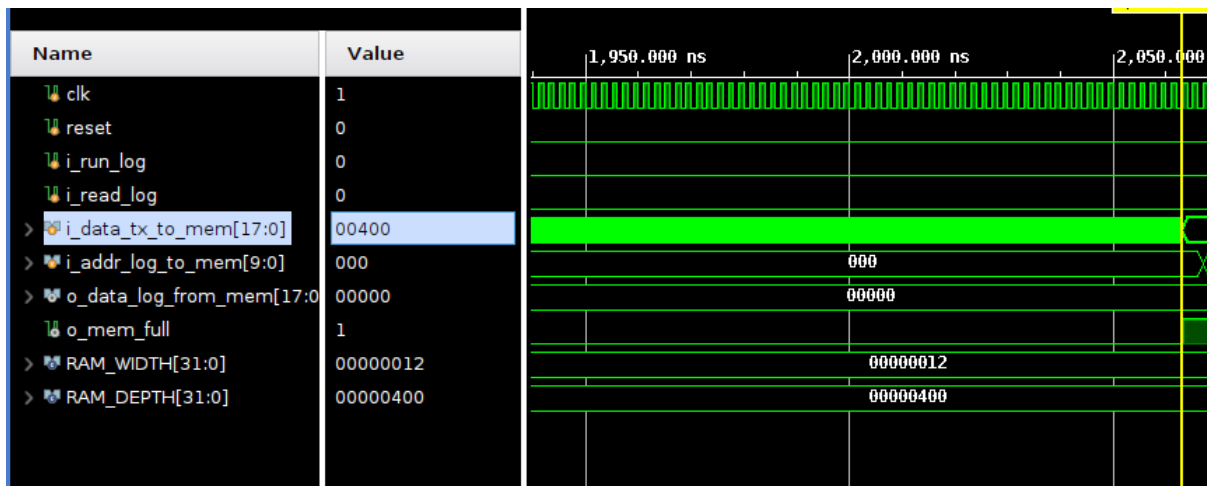


Se debe implementar una interfaz denominada Register File, la cual es un banco de registros que almacena los comandos enviados desde el uP.



- Se debe implementar una BRAM
- Los datos se guardan a la frecuencia de clock.
- La memoria comienza a loguear cuando detecta el flanco ascendente de o run log y guarda datos en cada ciclo de clock hasta llegar a la posición final de la memoria.





```
run all
Memory full
Data read from memory: 31
Expected: 31
Data read from memory: 32
Expected: 32
Data read from memory: 33
Expected: 33
```

Para el diseño de la memoria ram se tuvo en cuenta que la cantidad de datos para guardar iba a ser muy pequeña (8 bits) en comparación de la RAM (32 bits). Por lo que se optó por guardar la salida de los 2 filtros en los bits [7:0] la señal filtrada de I y en [16:8] la señal filtrada de Q:

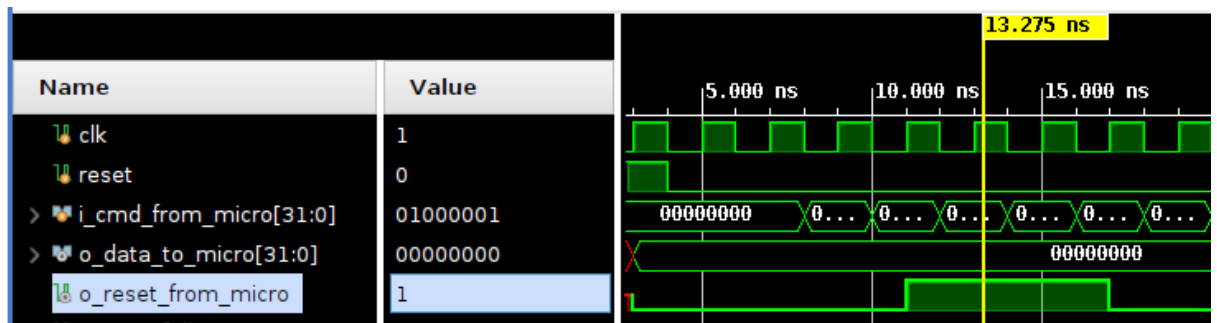
```
assign data_tx_to_mem = {{16{1'b0}} , filter_out_Q , filter_out_I};
```

## File Register

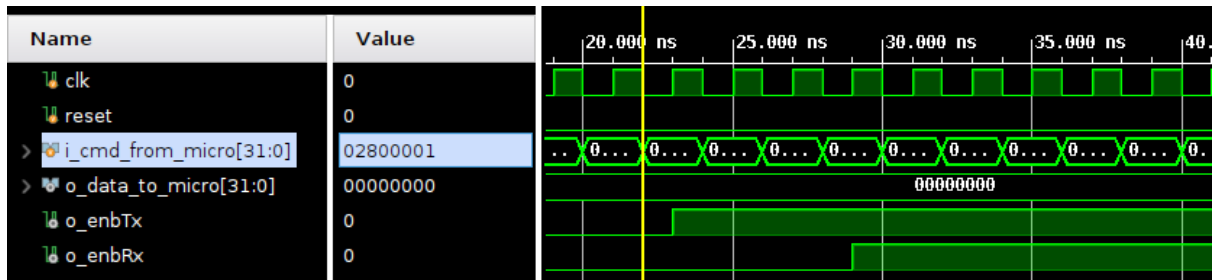
Para este caso se hizo más extensa la prueba ya que debíamos probar parte por parte cómo interactuaba el file register con los comandos que le llegaban.

- cmd: Reset

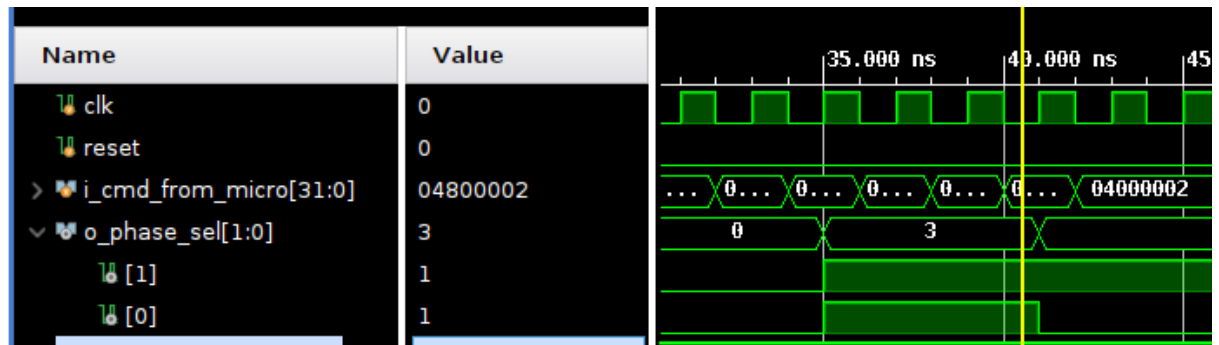
Se envía la señal para activar el reset, y luego se envía otra para desactivarlo.



- cmd: Enable\_TX y Enable\_RX

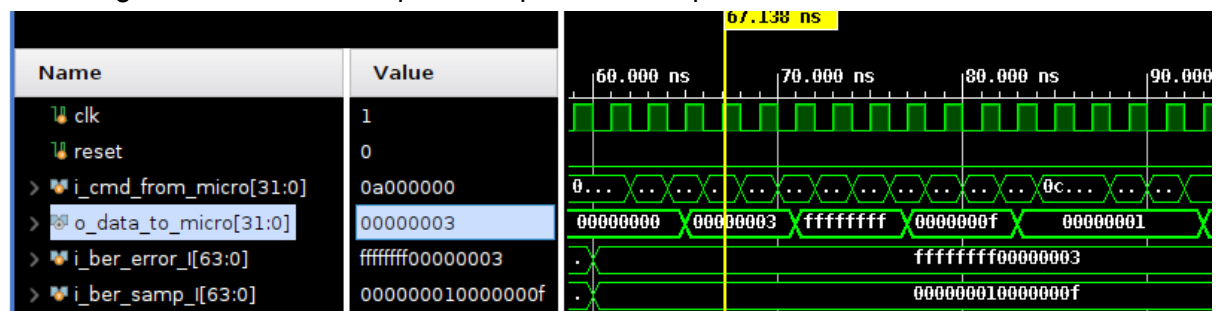


- cmd: phase\_sel



- cmd: Lectura\_BER

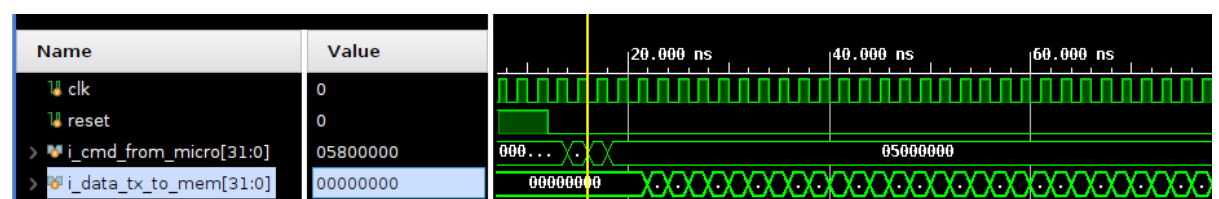
Se le asignó un valor a la ber, para comprobar si era posible la lectura



## Integración File Register - Memoria Ram

Se probó la integración de los dos módulos entre si y se obtuvo los resultados esperados

- Envío de comando de escritura en memoria



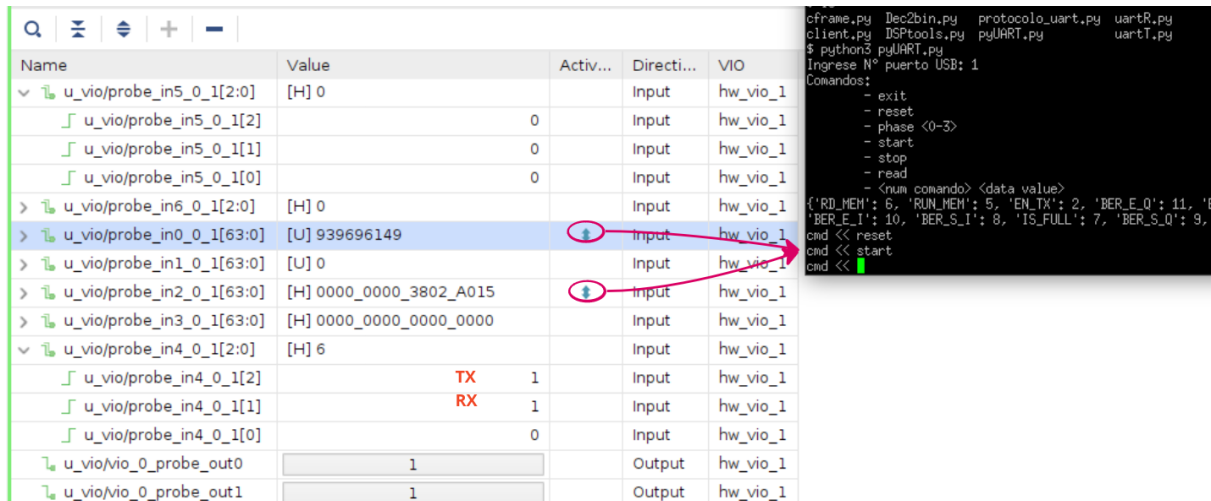
- Lectura de la memoria

---

```
.....  
addr_to_read = 1017  
data_to_micro =      1016  
.....  
addr_to_read = 1018  
data_to_micro =      1017  
.....  
addr_to_read = 1019  
data_to_micro =      1018  
.....  
addr_to_read = 1020  
data_to_micro =      1019  
.....  
addr_to_read = 1021  
data_to_micro =      1020  
.....  
addr_to_read = 1022  
data_to_micro =      1021  
.....  
addr_to_read = 1023  
data to micro =      1022
```

## Pruebas del funcionamiento en la FPGA

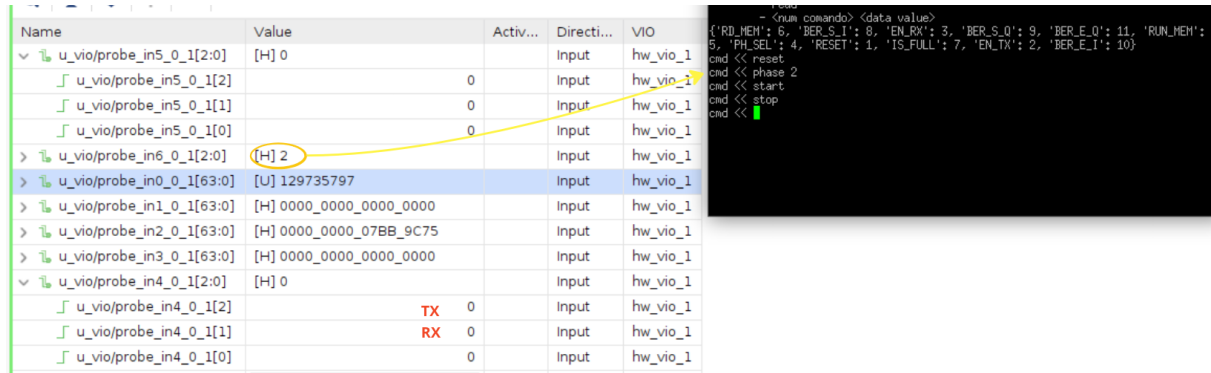
Para iniciar el sistema se escribe el comando start: El cual envía el comando EN\_TX y EN\_RX iniciando así el sistema.



Name	Value	Activ...	Directi...	VIO
u_vio/probe_in5_0_1[2:0]	[H] 0		Input	hw_vio_1
u_vio/probe_in5_0_1[2]		0	Input	hw_vio_1
u_vio/probe_in5_0_1[1]		0	Input	hw_vio_1
u_vio/probe_in5_0_1[0]		0	Input	hw_vio_1
u_vio/probe_in6_0_1[2:0]	[H] 0		Input	hw_vio_1
u_vio/probe_in0_0_1[63:0]	[U] 939696149		Input	hw_vio_1
u_vio/probe_in1_0_1[63:0]	[U] 0		Input	hw_vio_1
u_vio/probe_in2_0_1[63:0]	[H] 0000_0000_3802_A015		Input	hw_vio_1
u_vio/probe_in3_0_1[63:0]	[H] 0000_0000_0000_0000		Input	hw_vio_1
u_vio/probe_in4_0_1[2:0]	[H] 6		Input	hw_vio_1
u_vio/probe_in4_0_1[2]		TX 1	Input	hw_vio_1
u_vio/probe_in4_0_1[1]		RX 1	Input	hw_vio_1
u_vio/probe_in4_0_1[0]		0	Input	hw_vio_1
u_vio/vio_0_probe_out0	1		Output	hw_vio_1
u_vio/vio_0_probe_out1	1		Output	hw_vio_1

```
cframe.py Dec2bin.py protocolo_uart.py uartR.py
client.py DSPtools.py pyUART.py uartT.py
$ python3 pyUART.py
Ingrese N° puerto USB: 1
Comandos:
- exit
- reset
- phase <0-3>
- start
- stop
- read
- (num comando) <data value>
{'RD_MEM': 6, 'RUN_MEM': 5, 'EN_TX': 2, 'BER_E_0': 11, 'BER_E_1': 10, 'BER_S_1': 8, 'IS_FULL': 7, 'BER_S_0': 9, 'PH_SEL': 4, 'RESET': 1, 'IS_FULL': 7, 'EN_TX': 2, 'BER_E_1': 10}
cmd << reset
cmd << phase 2
cmd << start
cmd << stop
cmd <<
```

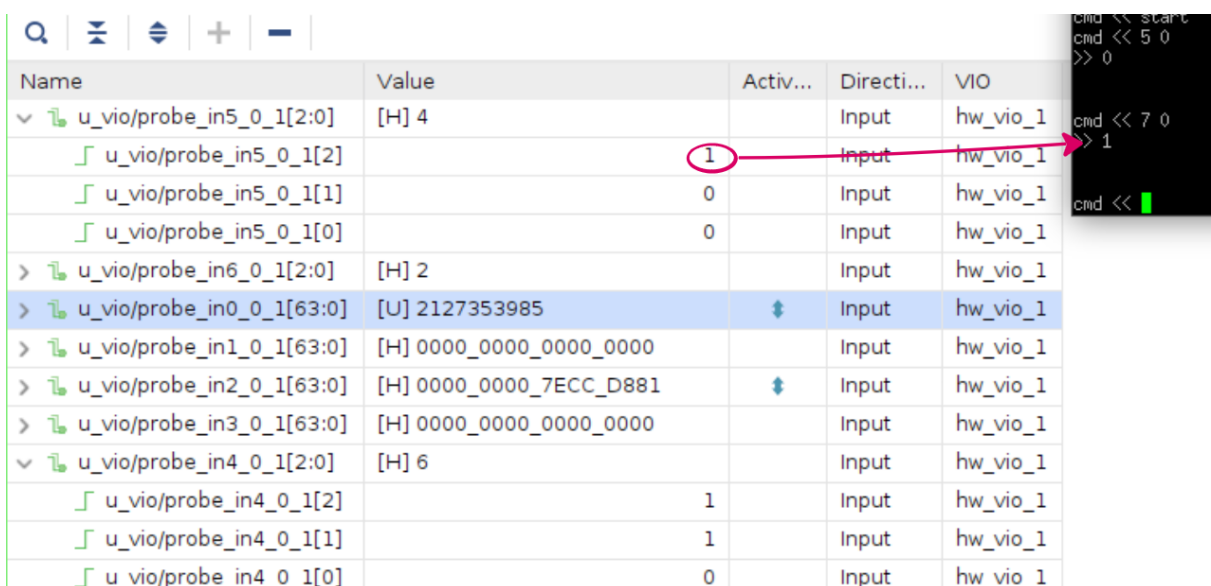
Seleccionamos la phase, deshabilitamos tx y rx con el comando stop



Name	Value	Activ...	Directi...	VIO
u_vio/probe_in5_0_1[2:0]	[H] 0		Input	hw_vio_1
u_vio/probe_in5_0_1[2]		0	Input	hw_vio_1
u_vio/probe_in5_0_1[1]		0	Input	hw_vio_1
u_vio/probe_in5_0_1[0]		0	Input	hw_vio_1
u_vio/probe_in6_0_1[2:0]	[H] 2		Input	hw_vio_1
u_vio/probe_in0_0_1[63:0]	[U] 129735797		Input	hw_vio_1
u_vio/probe_in1_0_1[63:0]	[H] 0000_0000_0000_0000		Input	hw_vio_1
u_vio/probe_in2_0_1[63:0]	[H] 0000_0000_07BB_9C75		Input	hw_vio_1
u_vio/probe_in3_0_1[63:0]	[H] 0000_0000_0000_0000		Input	hw_vio_1
u_vio/probe_in4_0_1[2:0]	[H] 0		Input	hw_vio_1
u_vio/probe_in4_0_1[2]		TX 0	Input	hw_vio_1
u_vio/probe_in4_0_1[1]		RX 0	Input	hw_vio_1
u_vio/probe_in4_0_1[0]		0	Input	hw_vio_1

```
- (num comando) <data value>
{'RD_MEM': 6, 'RUN_MEM': 5, 'EN_TX': 3, 'BER_S_0': 9, 'BER_E_0': 11, 'RUN_MEM': 5, 'PH_SEL': 4, 'RESET': 1, 'IS_FULL': 7, 'EN_TX': 2, 'BER_E_1': 10}
cmd << reset
cmd << phase 2
cmd << start
cmd << stop
cmd <<
```

Seleccionamos el comando 5 para guardar los datos en la memoria y luego con el comando 7 0 preguntamos si la memoria se encuentra llena



Name	Value	Activ...	Directi...	VIO
u_vio/probe_in5_0_1[2:0]	[H] 4		Input	hw_vio_1
u_vio/probe_in5_0_1[2]		1	Input	hw_vio_1
u_vio/probe_in5_0_1[1]		0	Input	hw_vio_1
u_vio/probe_in5_0_1[0]		0	Input	hw_vio_1
u_vio/probe_in6_0_1[2:0]	[H] 2		Input	hw_vio_1
u_vio/probe_in0_0_1[63:0]	[U] 2127353985		Input	hw_vio_1
u_vio/probe_in1_0_1[63:0]	[H] 0000_0000_0000_0000		Input	hw_vio_1
u_vio/probe_in2_0_1[63:0]	[H] 0000_0000_7ECC_D881		Input	hw_vio_1
u_vio/probe_in3_0_1[63:0]	[H] 0000_0000_0000_0000		Input	hw_vio_1
u_vio/probe_in4_0_1[2:0]	[H] 6		Input	hw_vio_1
u_vio/probe_in4_0_1[2]		1	Input	hw_vio_1
u_vio/probe_in4_0_1[1]		1	Input	hw_vio_1
u_vio/probe_in4_0_1[0]		0	Input	hw_vio_1

```
cmd << start
cmd << 5 0
>> 0
cmd << 7 0
>> 1
cmd <<
```

con el comando read leemos varios valores de la memoria para luego poderlos graficar



19A28998A

tb\_bram.v x tb\_file\_register.v x tb\_mem\_fregister.v x **hw\_vios** x

**hw\_vio\_1**

Dashboard Options

Name	Value	Activ...	Directi...	VIO
u_vio/probe_in5_0_1[2:0]	[H] 6		Input	hw_vio_1
u_vio/probe_in5_0_1[2]		1	Input	hw_vio_1
u_vio/probe_in5_0_1[1]		1	Input	hw_vio_1
u_vio/probe_in5_0_1[0]		0	Input	hw_vio_1

```
cmd << read
Leyendo memoria...
0 32895
1 32895
2 32895
3 32895
4 32895
5 51786
6 9216
7 27318
8 32642
9 22144
10 64640
11 42624
12 33410
13 42632
14 64648
15 22148
16 32388
17 32895
18 32895
19 32895
20 32895
21 51786
```

para leer un valor unico usamos el siguiente comando

```
cmd << 6 0
>> 32639

cmd << 6 3
>> 32639
```

para la lectura de la BER samples

u_vio/probe_in6_0_1[2:0]	[H] 2		Input	hw_vio_1
u_vio/probe_in0_0_1[63:0]	[U] 5639490974		Input	hw_vio_1

```
cmd << stop
cmd << 9 0
>> 5639490974
```

lectura BER errors

u_vio/probe_in3_0_1[63:0]	[H] 0000_0000_0000_0000		Input	hw_vio_1
u_vio/probe_in4_0_1[2:0]	[H] 0		Input	hw_vio_1

```
cmd << 11 0
>> 0
```

Una vez obtenemos las muestras se guardan en un archivo txt el cual con un script de python se lee y se grafica para lo cual el resultado es el siguiente:

