# INFORME DEL TRABAJO DE PRÁCTICA PROFESIONAL SUPERVISADA

Caracteristicas de la SDR ADALM-PLUTO


Jonathan Armando Patiño

# INTRODUCCIÓN

El objetivo de la Práctica Profesional Supervisada es evaluar el desempeño de algoritmos de procesamiento de señales aplicados a sistemas de radares en una plataforma de radio definida por software. Para alcanzar este objetivo se utilizaron radios definidas por software implementadas en una placa Adalm-Pluto (SDR) conectada a un servidor y con una Virtual Private Network (VPN) que permite el acceso remoto desde cualquier parte del mundo. El código con el cual se generó el archivo ejecutable fue realizado en C. Luego se definen los requisitos que debe tener el pulso para poder ser transmitido y la configuración del hardware. También se utiliza un entorno virtual local corriendo en Docker, lo cual permite tener un contexto de la SDR.

Cabe destacar que que el proyecto se realiza de manera stand alone, es decir, de manera independiente de otros softwares. Para poder realizar esto, se tiene en cuenta el contexto creado en Docker con la versión de compilador que se necesita en la SDR, luego se lo compila y al ejecutable se lo envía mediante Secure Copy (SCP) para ser ejecutado en la placa. Los recursos de la SDR son limitados por lo que se deben tener en cuenta a la hora de utilizar ciertas variables.

# DESARROLLO

**Software**

El proyecto se lleva a cabo en Windows como sistema operativo principal, pero opcionalmente se puede utilizar Linux. Se deben considerar los siguientes pasos para poder ejecutar el programa:

***Para usuario:***

Primer paso:

descargar la imagen de Docker desde con el comando

***docker pull jonathan684/radar-project:2.7***

la cual es la última hasta el momento.

Segundo paso:

Conectarse a la VPN de Cisco y crear el contenedor

***docker run -it -p 8001:8001 --net=host –name sdr_proj id_imagen***

se puede visualizar el contenedor con ***docker ps -a***

```
jonathan@jonathan:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND   CREATED              STATUS                          PORTS     NAMES
186cdeaf002b   7c243fd64e17   "bash"    About a minute ago   Exited (0) About a minute ago             sdr_proj
```

luego se inicia el contenedor con:

```
jonathan@jonathan:~$ docker start sdr_proj
sdr_proj
```

y a continuación se ejecuta la bash del contenedor con

***docker exec -it sdr_proj bash***

```
jonathan@jonathan:~$ docker exec -it sdr_proj bash
root@jonathan:/project/init#
```

Tercer paso:

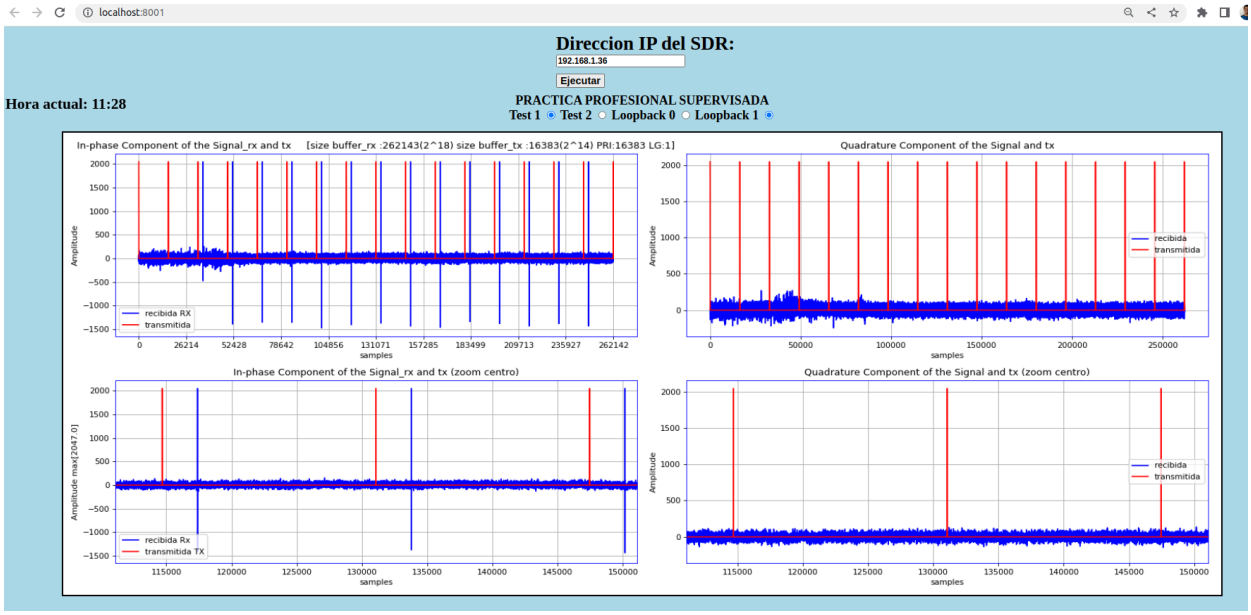
Para ejecutar la aplicación se debe ejecutar:

**bash run.sh**

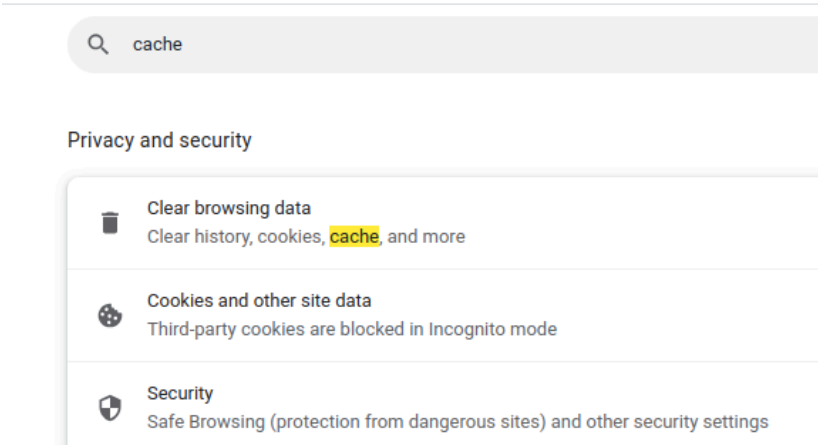
```
jonathan@jonathan:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND   CREATED         STATUS
     PORTS     NAMES
c5feba30862b   93b9e154dd1d   "bash"    4 minutes ago   Exited (130) 5 seconds a
go             sdr_proj
jonathan@jonathan:~$ docker start sdr_proj
sdr_proj
jonathan@jonathan:~$ docker exec -it sdr_proj bash
root@jonathan:/project/app# bash run.sh
Servidor en ejecución en http://localhost:8001
127.0.0.1 - - [23/Jul/2023 18:23:12] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [23/Jul/2023 18:23:12] "GET /config.js HTTP/1.1" 200 -
127.0.0.1 - - [23/Jul/2023 18:23:12] "GET /script.js HTTP/1.1" 200 -
127.0.0.1 - - [23/Jul/2023 18:23:12] "GET /styles.css HTTP/1.1" 200 -
127.0.0.1 - - [23/Jul/2023 18:23:12] "GET /test.png HTTP/1.1" 200 -
127.0.0.1 - - [23/Jul/2023 18:23:12] "GET /rx_config.txt HTTP/1.1" 200 -
127.0.0.1 - - [23/Jul/2023 18:23:12] "GET /valores.json HTTP/1.1" 200 -
127.0.0.1 - - [23/Jul/2023 18:23:12] code 404, message File not found
127.0.0.1 - - [23/Jul/2023 18:23:12] "GET /favicon.ico HTTP/1.1" 404 -
```

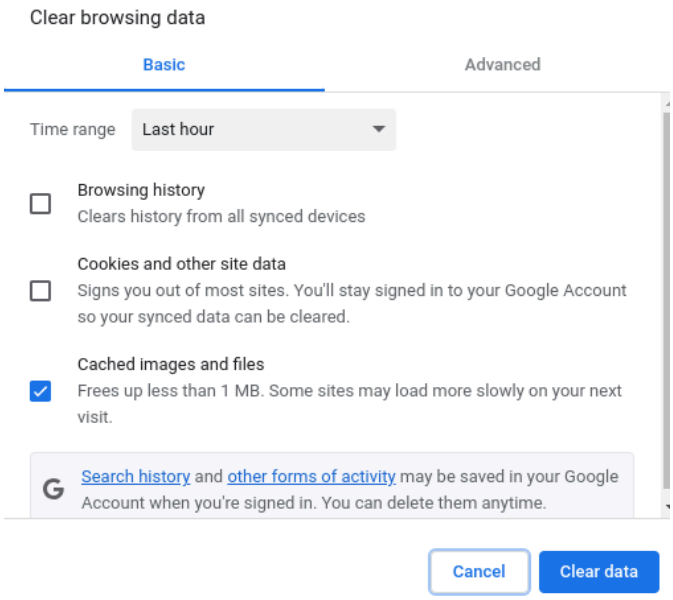
Cuarto paso:

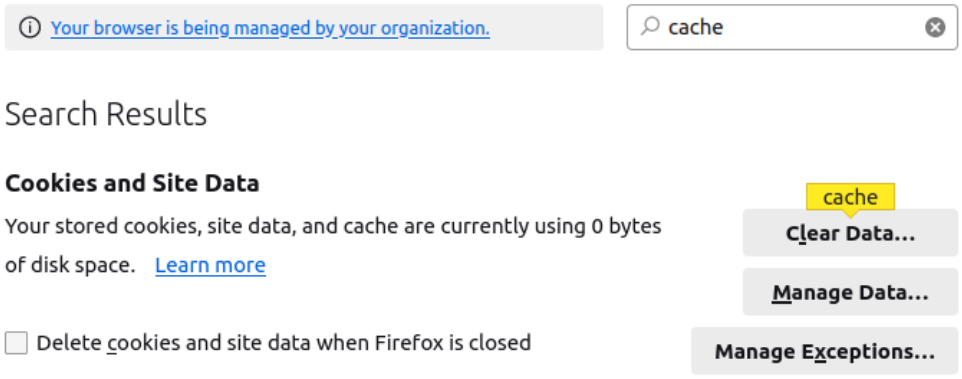
Abrir el http://localhost:8001/, el cual se debe observar como se muestra en la siguiente figura:

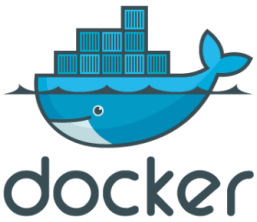


Listo. Para observar la transmisión y recepción la figura muestra las dos componentes, una columna en fase y en cuadratura del buffer de recepción. También se tiene la posibilidad de elegir entre test1 y test2. Para test1 las muestras de la recepción son de una lectura del buffer, mientras que en el test2 se lee tres veces el buffer. Será necesario recargar la página cada vez que se ejecute una simulación, y en caso de no observar cambios, también será necesario borrar la caché del browser. En Mozilla se puede borrar la caché de files en configuraciones, como se observa en las imágenes:





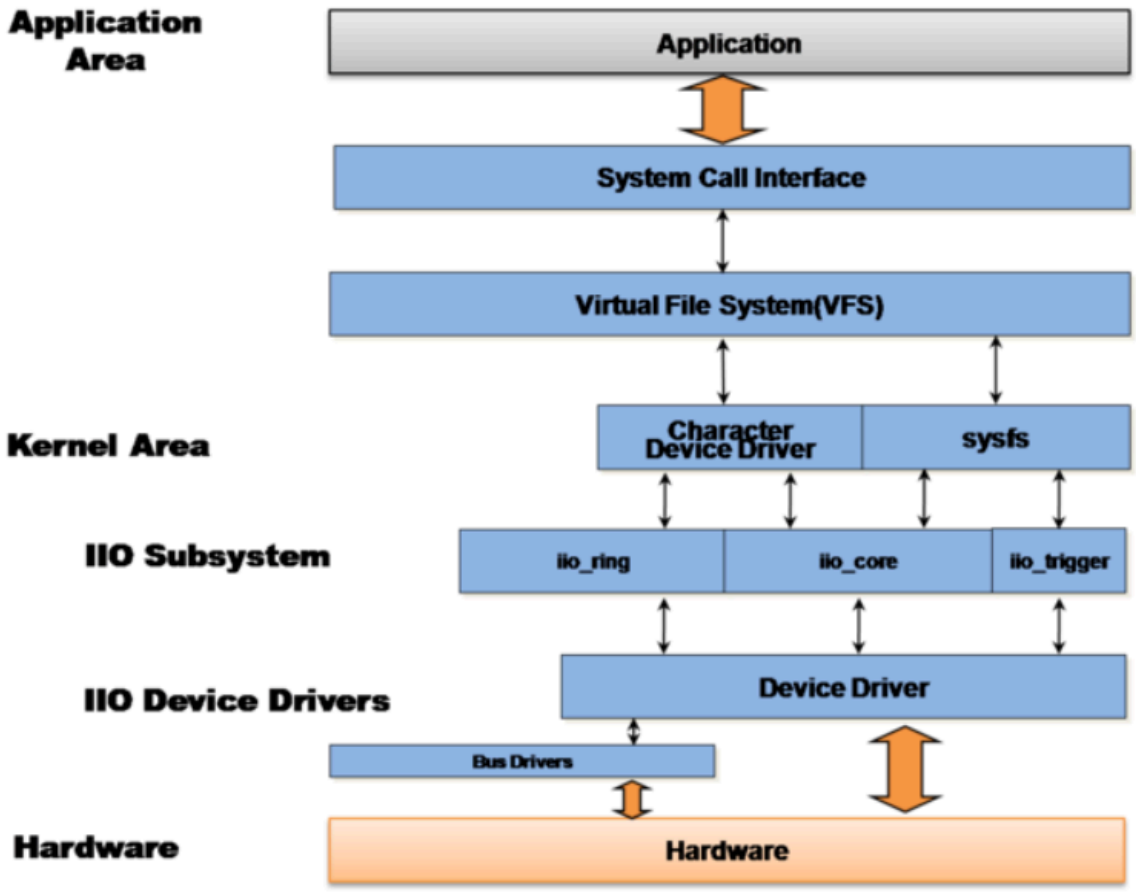





**Hardware**:

La SDR posee como sistema operativo Linux iio, el cual es un subsistema de E/S industrial que está diseñado para proporcionar soporte para dispositivos que, en cierto sentido, son convertidores de analógico a digital o de digital a analógico (ADC, DAC).

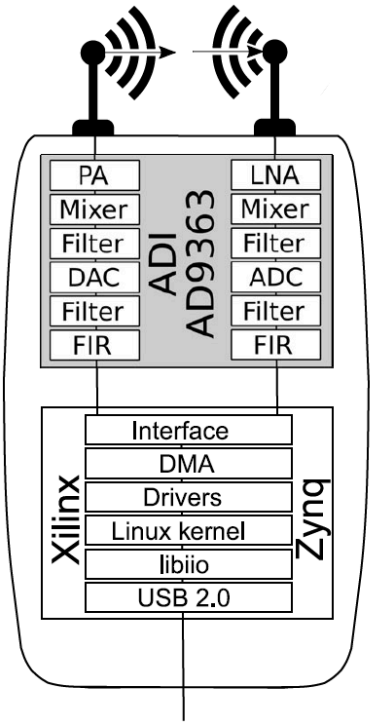
## Descripción general del subsistema IIO




Por otra parte, el hardware de la SDR está compuesto por:

- Una sección de RF(Radio frecuencia) analógica (antena, filtros de RF, mux de entrada, LNA, ganancia, atenuación, mezclador)
- Una sección de banda base analógica (filtros analógicos, ADC y DAC)
- Algunas unidades de procesamiento de señales (filtros fijos dentro de un transceptor o filtros definidos por el usuario) dentro de un FPGA o DSP, o procesador de propósito general).

En la siguiente imagen se muestra la distribución de los componentes:




La sección de banda base analógica (filtros analógicos, ADC o DAC) se implementa en el AD9363, transceptor ágil de RF integrado.

El procesamiento de la señal se divide entre:

- ➔ Partes del procesamiento de la señal se implementan en el *AD9363*, Integrado Transceptor ágil de RF. Esto incluye la decimación de media banda fija y filtros de interpolación y filtros FIR programables de 128-tap.
- ➔ Opcionalmente, el filtrado y la decimación deben ser realizados en Xilinx Zynq' FPGA .

El transmisor Adalm-Pluto consta de:

- Rango de frecuencia:  325 a 3800 MHz
- Ancho de banda del canal: 200–20, 000 kHz  (20Mhz)
- Frecuencia de muestreo:  61,44 mega muestras por segundo (MSPS)
- Tamaño de paso LO (oscilador local): 2,4 Hz
- Tamaño de paso de frecuencia de muestreo: 5 Hz
- Precisión de modulación (EVM - Magnitud del vector de error): 40 dB (típico)

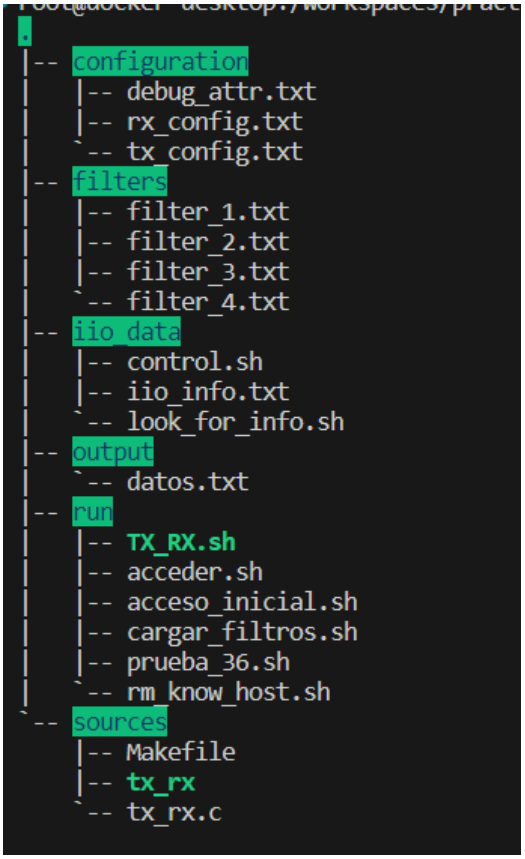
- DAC (convertidores de digital a analógico): resolución de 12 bits

Receptor (RX):

- Rango de frecuencia: 300-3800 GHz
- Ancho de banda del canal: 200-20 000 kHz
- Frecuencia de muestreo: 65,1-61440 kSPS
- Tamaño de paso LO (oscilador local): 2,4 Hz
- Tamaño de paso de frecuencia de muestreo: 5 Hz
- Precisión de modulación: no especificada.
- ADC (convertidores de analógico a digital): resolución de 12 bits

**Funcionamiento de la aplicación para la SDR**

El programa cuenta con distintos archivos *.sh* lo cual permite obtener de manera ágil los datos y las configuraciones. La siguiente imagen muestra cómo se distribuyen los archivos:




La carpeta llamada configuration contiene los parámetros de configuración para el

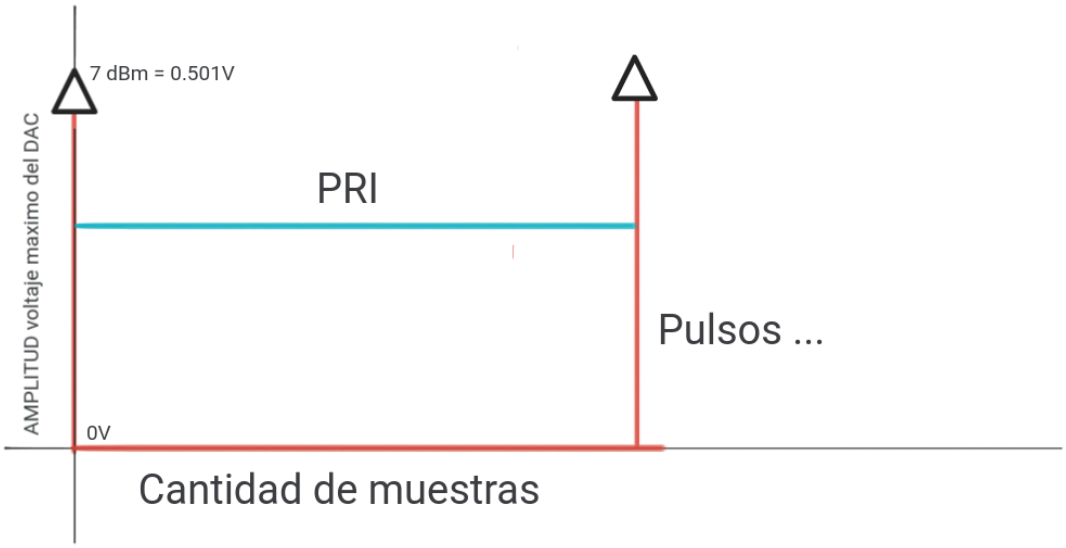
transmisor como para el receptor y la señal:

| tx_config.tx: | rx_config.txt: |
|---|---|
| sampling_frequency 60000000<br>rf_bandwidth 20000000<br>gain_control_mode fast_attack<br>hardwaregain -10<br>frequency 3500000000 | sampling_frequency 60000000<br>rf_bandwidth 20000000<br>gain_control_mode fast_attack<br>hardwaregain 73<br>frequency 3500000000 |

También cuenta con un archivo ***debug_attr.txt*** el cual contiene atributos necesarios para esta simulación:

```
Longitud_del_pulso 1
PRI 2_17
amplitud 0x0FFF
TxBufferSize 2_17
RxBufferSize 2_18
rx 1
loopback 1
```

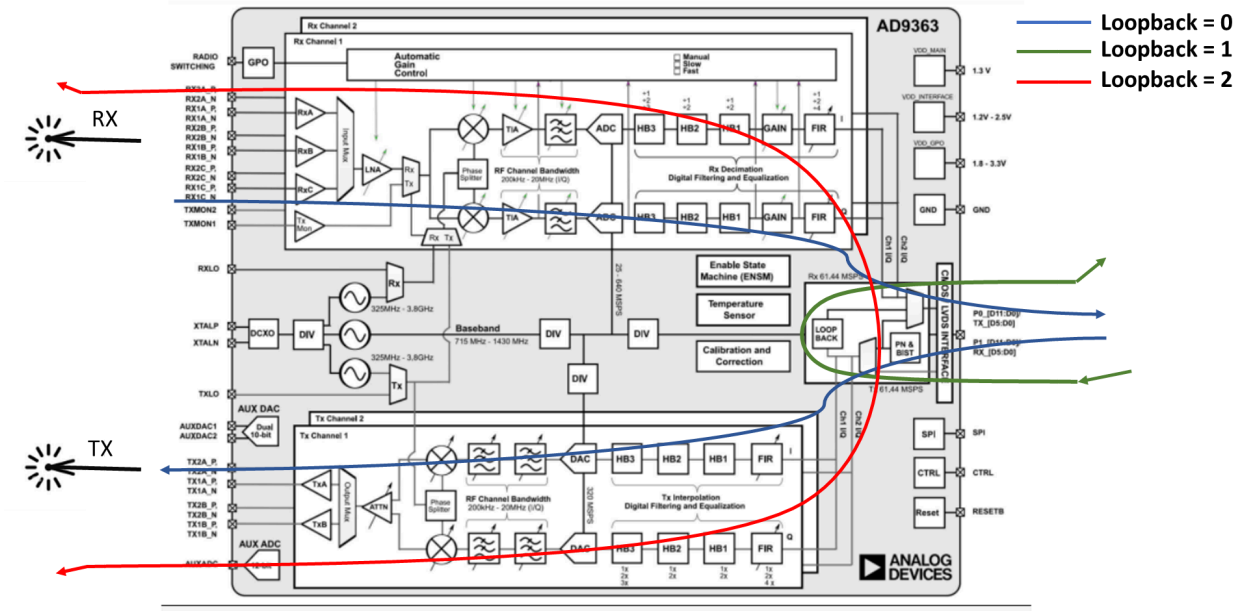
- *Longitud_del_pulso 1* significa que el pulso tendrá una longitud de 1/frecuencia de muestreo
- PRI $2^{17}$ significa que existen ***131072*** muestras en que el DAC entrega 0 volt.
- *amplitud 0x0FFF* es la máxima potencia que puede entregar el DAC, la cual dependerá de la frecuencia de la portadora. Para el caso de trabajar con 3.5 Ghz se observa en el datasheet del AD9361 una potencia de 7.0 dBm, como se muestra en la figura:




| | | | | |
|---|---|---|---|---|
| TRANSMITTERS, 3.5 GHz | | | | |
| Output Return Loss | S22 | −10 | dB | |
| Maximum Output Power | | 7.0 | dBm | 1 MHz tone into 50 Ω load |
| Modulation Accuracy (EVM) | | −34 | dB | 40 MHz reference clock |



- *TxBufferSize 2_17* es el tamaño del buffer de transmisión
- *RxBufferSize 2_18* es el tamaño del buffer de recepción
- *rx 1* significa que el receptor está disponible. Existen casos donde se desea que esto no ocurra, ya que se está analizando la recepción con el osciloscopio, por ejemplo.
- configuración *loopback* como se observa en la siguiente figura, la SDR puede trabajar en diferentes modos: como repetidor ***loopback 2***, con ***loopback 1*** se tiene loopback digital sin utilizar los filtros, y con ***loopback 0*** es posible sacar una señal por la antena transmisora y utilizar la antena receptora. En este caso no se utilizan los filtros.



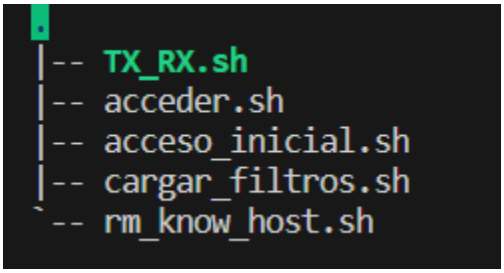

Los filtros en la carpeta ***filters*** son necesarios para el modo ***loopback 2***. A medida que se modifica la frecuencia de muestreo, se cambian los filtros. Estos filtros se cargan junto con el decimador y el interpolador dependiendo de la frecuencia de muestreo.

Además contiene una carpeta ***iio_data*** y un archivo **control.sh** que permite obtener de la SDR parámetros de la configuración actual para su observación, por ejemplo:

***echo -n "out_altvoltage1_TX_LO_frequency-:"; cat out_altvoltage1_TX_LO_frequency***

retorna la frecuencia de portadora del transmisor.

En la carpeta ***output*** se encuentra el archivo ***datos.txt*** que contiene los datos que obtuvo el receptor. La carpeta ***run*** contiene archivos *.sh*.

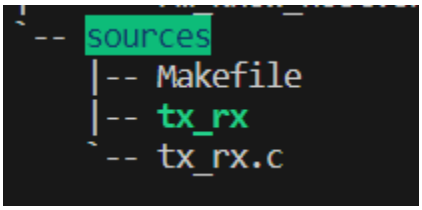



**TX_RX.sh** envía a la SDR el programa ***txt_rx*** y lo ejecuta, luego extrae los datos mediante ***scp***.

También en caso de ser necesario debemos darle un mayor privilegio a los archivos .sh

para que se ejecuten. Esto se hace mediante ***chmod +x TX_RX.sh.***

Finalmente en la carpeta *sources* se encuentran:




- **Makefile** que permite obtener el ejecutable para luego hacer la compilación cruzada
- ***tx_rx.c*** que contiene el programa principal

Para entender el código se debe tener en cuenta lo siguiente:

La Adalm-Pluto SDR contiene distintos dispositivos de configuración ***ad9361-phy xadc cf-ad9361-dds-core-lpc cf-ad9361-lpc***, todos configurables mediante canales ***spi***.

El dispositivo ***ad9361-phy*** contiene las configuraciones principales de la SDR como lo son: la frecuencia de la portadora, la frecuencia del muestreo, la ganancia y la configuración de los filtros. Mediante este dispositivo se puede configurar la mayor parte de la SDR. Se pueden observar los atributos que se pueden modificar siguiendo el siguiente path en la SDR. Este dispositivo corresponde a la configuración de *ad9361* desde la SDR.

***# cat /sys/bus/iio/devices/iio\:device0/name  ad9361-phy***

Atributos más comunes :

***# ls***

***out_altvoltage0_RX_LO_frequency***

***out_altvoltage1_TX_LO_frequency***

***in_voltage_rf_bandwidth***

***out_voltage_rf_bandwidth***

***out_voltage0_hardwaregain***

***in_voltage0_hardwaregain***

entre otras configuraciones.

Los demás dispositivos son:

***# cat /sys/bus/iio/devices/iio\:device1/name xadc*** necesario para la configuración de la FPGA

***# cat /sys/bus/iio/devices/iio\:device2/name cf-ad9361-dds-core-lpc*** dispositivo de transmisión

***# cat /sys/bus/iio/devices/iio\:device3/name cf-ad9361-lpc*** dispositivo de recepción

Para este caso de estudio, al modificar el primer dispositivo se modifican los demás ya que los atributos son dependientes. Cuando se esperan respuestas distintas entre el transmisor y el receptor, se modifican los otros dispositivos.

Para poder configurar cada dispositivo desde C, como es este caso, se debe crear un contexto con el dispositivo principal y luego configurar los dispositivos de transmisión y recepción. A continuación se muestra una parte del código para la configuración de los dispositivos:

```
#include <iio.h>

struct iio_context *ctx = NULL;

struct iio_device *main_dev;

struct iio_device *dev_tx;

struct iio_device *dev_rx;

struct iio_channel *tx0_i, *tx0_q;

struct iio_buffer *txbuf;

struct iio_channel *rx0_i, *rx0_q;

struct iio_buffer *rxbuf;

struct iio_channel *chnn_altvoltage1_output;

struct iio_channel *chnn_altvoltage0_output;

struct iio_channel *chnn_device_output;

struct iio_channel *chnn_device_input;
```

**Configuración del dispositivo Principal:**

```
main_dev = iio_context_find_device(ctx, "ad9361-phy");

chnn_device_output = iio_device_find_channel(main_dev, "voltage0", true);

 iio_channel_attr_write(chnn_device_output, “sampling_frequency”,“60000000”);
```

```
iio_channel_attr_write(chnn_device_output, “rf_bandwidth”, “20000000”);
iio_channel_attr_write(chnn_device_output,“hardwaregain”,“-10”);
iio_device_find_channel(main_dev, "altvoltage1", true);/
iio_channel_attr_write(chnn_altvoltage1_output,“frequency”,“3500000000”);
chnn_device_input=iio_device_find_channel(main_dev,"voltage0", false);
iio_channel_attr_write(chnn_device_input,“sampling_frequency”,“60000000”);
iio_channel_attr_write(chnn_device_input,“rf_bandwidth”, “20000000”);
iio_channel_attr_write(chnn_device_input,“gain_control_mode”,“fast_attack”);
chnn_altvoltage0_output=iio_device_find_channel(main_dev,"altvoltage0", true)
iio_channel_attr_write(chnn_altvoltage0_output,“frequency”,“3500000000”);
```

**Configuración del transmisor*:***

```
dev_tx=iio_context_find_device(ctx,"cf-ad9361-dds-core-lpc");//transmisor
tx0_i = iio_device_find_channel(dev_tx, "voltage0", true);
tx0_q = iio_device_find_channel(dev_tx, "voltage1", true);
iio_channel_enable(tx0_i);
iio_channel_enable(tx0_q);
txbuf = iio_device_create_buffer(dev_tx, TxBufferSize, true);
```

**Configuración del receptor:**

```
dev_rx = iio_context_find_device(ctx, "cf-ad9361-lpc");
 rx0_i = iio_device_find_channel(dev_rx, "voltage0", false);
rx0_q = iio_device_find_channel(dev_rx, "voltage1", false);
iio_channel_enable(rx0_i);
iio_channel_enable(rx0_q);
rxbuf = iio_device_create_buffer(dev_rx, RxBufferSize, false);
```

Luego de la configuración se procede a la transmisión y recepción del pulso:

```
/*TRANSMITIR*/
iio_buffer_push(txbuf);
```

```
p_inc = iio_buffer_step(txbuf);

p_end = iio_buffer_end(txbuf);

int count = 0;

    for (p_dat = (char *)iio_buffer_first(txbuf, tx0_i); p_dat < p_end; p_dat
+= p_inc) {

       if ( (count >= 0) && (count < Longitud_del_pulso)  ) {

            ((int16_t*)p_dat)[0] = amplitud;

            ((int16_t*)p_dat)[1] = amplitud;

        }

        else if ( (count >= Longitud_del_pulso ) && (count < PRI ) ) {

            ((int16_t*)p_dat)[0] =((int16_t)0x0000);

            ((int16_t*)p_dat)[1] =((int16_t)0x0000);

        }

        count ++;

        if (count == PRI) count = 0;

    }


/*RECIBIR*/

    if(rx == true){

            int I_rx = 0;

            double signal_i[N_rx];

            double signal_q[N_rx];

            iio_buffer_refill(rxbuf);

            p_inc = iio_buffer_step(rxbuf);

            p_end = iio_buffer_end(rxbuf);
```

```
            for (p_dat = (char *)iio_buffer_first(rxbuf, rx0_i); p_dat < p_end;
p_dat += p_inc) {

                if(I_rx < N_rx){

                    signal_q[I_rx] = (double) (((int16_t*)p_dat)[0] );

                    signal_i[I_rx] = (double) (((int16_t*)p_dat)[1]);}

                I_rx ++;

                if(I_rx == N_rx)I_rx = 0;

            }

      …signal_q y signal_i se escriben en datos.tx

    }
```

A continuación de haber recibido los datos en el archivo ***datos.txt*** se procederá a analizarlos para verificar si efectivamente está funcionando correctamente. Para este propósito se creó un *script* en Python, en el cual se pueden observar los datos recibidos de manera gráfica.

**Señal**

La señal que se eligió para transmitir es un sistema de radar pulsado monoestático muy simple ya que como primer paso se quiere observar el comportamiento del buffer.

Las aplicaciones de radar utilizan transmisores de pulsos de potencia relativamente alta y receptores sensibles, por lo que el radar funciona en bandas que no se utilizan para otros fines. La mayoría de las bandas de radar se encuentran en la parte de microondas del espectro, aunque ciertas aplicaciones importantes para la meteorología utilizan potentes transmisores en la banda UHF.

Para este caso, fue necesario situarse en la banda S definida en la tabla por la IEEE como bandas de radar con una frecuencia de 2 a 4 Ghz de onda corta de portadora.

La señal que se envía depende también de la frecuencia de muestreo, la cual da el alcance que tendrá el pulso, mientras más rápido pueda enviar esta señal más rápido se obtendrá una respuesta para analizar, de esta forma el tiempo entre lo que envio y lo que recibo puedo obtener la distancia del objetivo, para conocer el tiempo que se debe

esperar para recibir la respuesta del mismo, dependerá de la distancia a la cual se encuentra el objetivo. Para calcular el alcance mínimo del pulso transmitido se debe tener en cuenta:

$$Rmin = \frac{c * \tau p}{2}$$

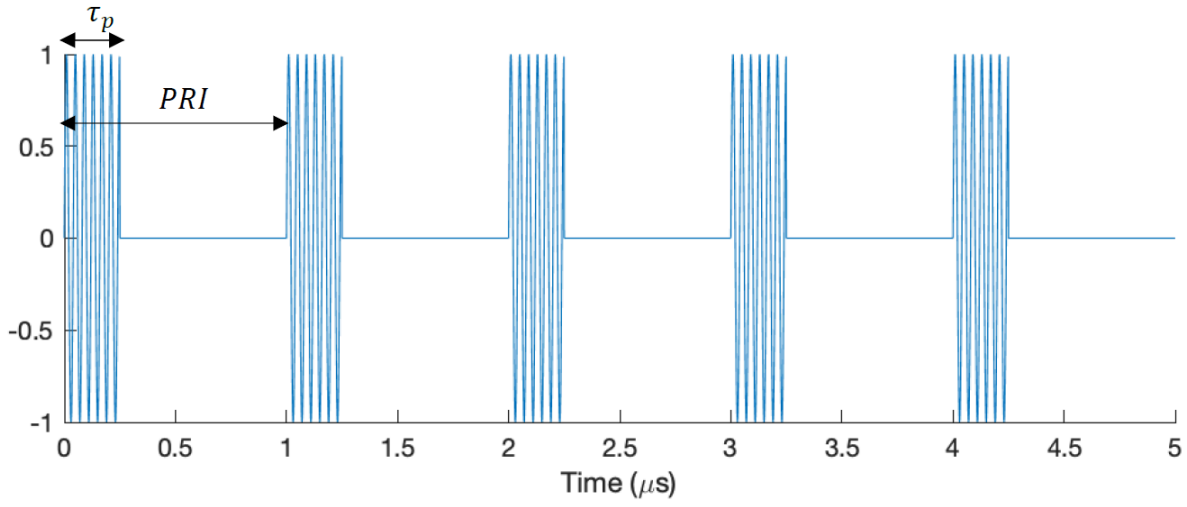
Siendo *Rmin* el rango mínimo, *c* la velocidad de la luz y $\tau p$ (tiempo mínimo del pulso)

El rango máximo dependerá del ***PRI*** (Intervalos de repetición de pulsos) que esté configurado en la señal, el cual se elige de acuerdo al error asociado con el pulso de retorno, es decir, un PRI muy grande acarrea una disminución de la amplitud del pulso de retorno y el máximo que se puede tener será cuando no sea posible distinguir entre la amplitud del pulso recibido y el ruido que se recibe.

$$Rmax = \frac{c * PRI}{2}$$

$$PRI = \frac{c * Rmax}{2}$$

se observa:




Rango para un caso particular:

Considerando,

$Frecuencia\ de\ portadora\ = 3.5\ Ghz$
$Frecuencia\ de\ muestreo\ = 60\ Mhz$

Rango mínimo:

$fp = 3500000000\ hz$ (frecuencia de portadora)
$fs = 60000000\ hz$ (frecuencia de muestreo)
$c = 2.998e + 8\ m/s$

$$\tau p = \frac{1}{fs} = \frac{1}{60000000} = 0.0000000166666667\ 1/s$$

$$Rmin = \frac{(c*\tau p)}{2} = 2.4983m$$

A continuación se tiene una tabla con variaciones de frecuencia de muestreo y sus respectivos alcances mínimos teóricos:

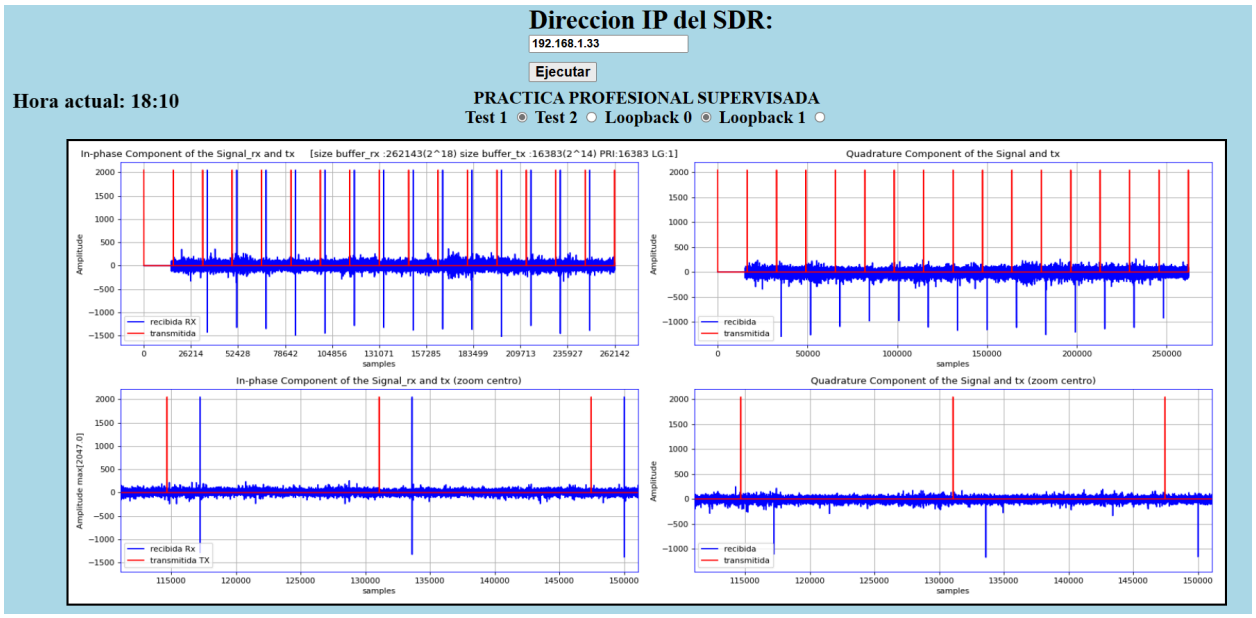
$step = 3150000\ hz$

| | $c(m/s)$ | $fs(hz)$ | $\tau p\ (1/s)$ | $Rmin\ (m)$ |
|---|---|---|---|---|
| 1 | 299.800.000,00 | 530000 | 0,000001886792453 | **282,83** |
| 2 | 299.800.000,00 | 3680000 | 0,0000002717391304 | **40,73** |
| 3 | 299.800.000,00 | 6830000 | 0,0000001464128843 | **21,95** |
| 4 | 299.800.000,00 | 9980000 | 0,0000001002004008 | **15,02** |
| 5 | 299.800.000,00 | 13130000 | 0,0000000761614623 | **11,42** |
| 6 | 299.800.000,00 | 16280000 | 0,00000006142506143 | **9,21** |
| 7 | 299.800.000,00 | 19430000 | 0,00000005146680391 | **7,71** |
| 8 | 299.800.000,00 | 22580000 | 0,00000004428697963 | **6,64** |
| 9 | 299.800.000,00 | 25730000 | 0,00000003886513797 | **5,83** |
| 10 | 299.800.000,00 | 28880000 | 0,00000003462603878 | **5,19** |
| 11 | 299.800.000,00 | 32030000 | 0,00000003122073057 | **4,68** |
| 12 | 299.800.000,00 | 35180000 | 0,00000002842524161 | **4,26** |
| 13 | 299.800.000,00 | 38330000 | 0,00000002608922515 | **3,91** |
| 14 | 299.800.000,00 | 41480000 | 0,00000002410800386 | **3,61** |
| 15 | 299.800.000,00 | 44630000 | 0,00000002240645306 | **3,36** |
| 16 | 299.800.000,00 | 47780000 | 0,0000000209292591 | **3,14** |
| 17 | 299.800.000,00 | 50930000 | 0,00000001963479285 | **2,94** |
| 18 | 299.800.000,00 | 54080000 | 0,00000001849112426 | **2,77** |
| 19 | 299.800.000,00 | 57230000 | 0,00000001747335314 | **2,62** |
| 20 | 299.800.000,00 | 60380000 | 0,00000001656177542 | **2,48** |

Por ejemplo eligiendo la siguiente configuración:

| *debug_attr* | *tx_config* | *rx_config* |
|---|---|---|
| **Longitud_del_pulso** 1<br>**PRI** 2_16<br>**amplitud** 0x0FFF<br>**TxBufferSize** 2_16<br>**RxBufferSize** 2_18<br>**rx** 1<br>**loopback** 1 | **sampling_frequency**<br>60000000<br>**rf_bandwidth**<br>20000000<br>**hardwaregain**<br>-10<br>**frequency**<br>3500000000 | **sampling_frequency**<br>60000000<br>**rf_bandwidth**<br>20000000<br>**gain_control_mode**<br>fast_attack<br>**hardwaregain**<br>73<br>**frequency**<br>3500000000 |

Se obtiene un análisis de la señal en fase y en cuadratura, como se observa a continuación,



**Direccion IP del SDR:**

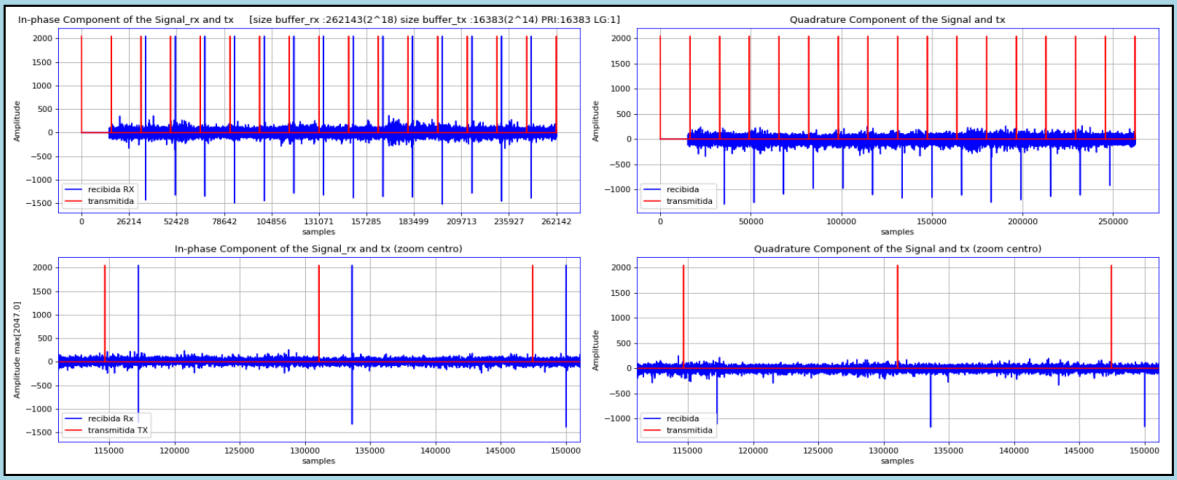
192.168.1.33

Ejecutar

**Hora actual: 18:12**

**PRACTICA PROFESIONAL SUPERVISADA**

**Test 1** ◉ **Test 2** ○ **Loopback 0** ○ **Loopback 1** ◉




**Direccion IP del SDR:**

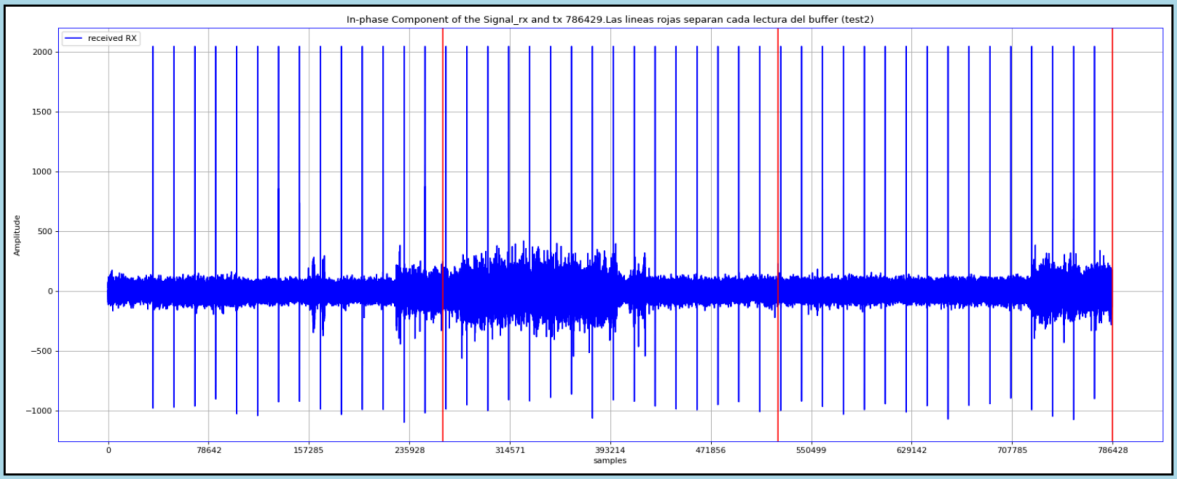
192.168.1.33

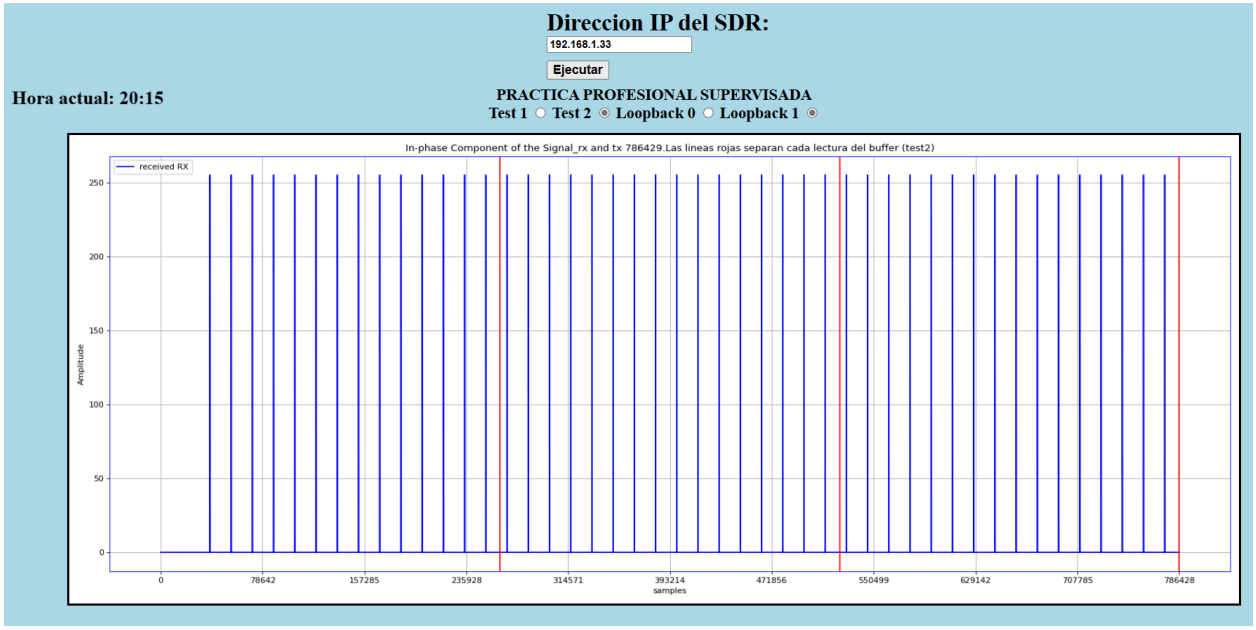
Ejecutar

**Hora actual: 20:10**

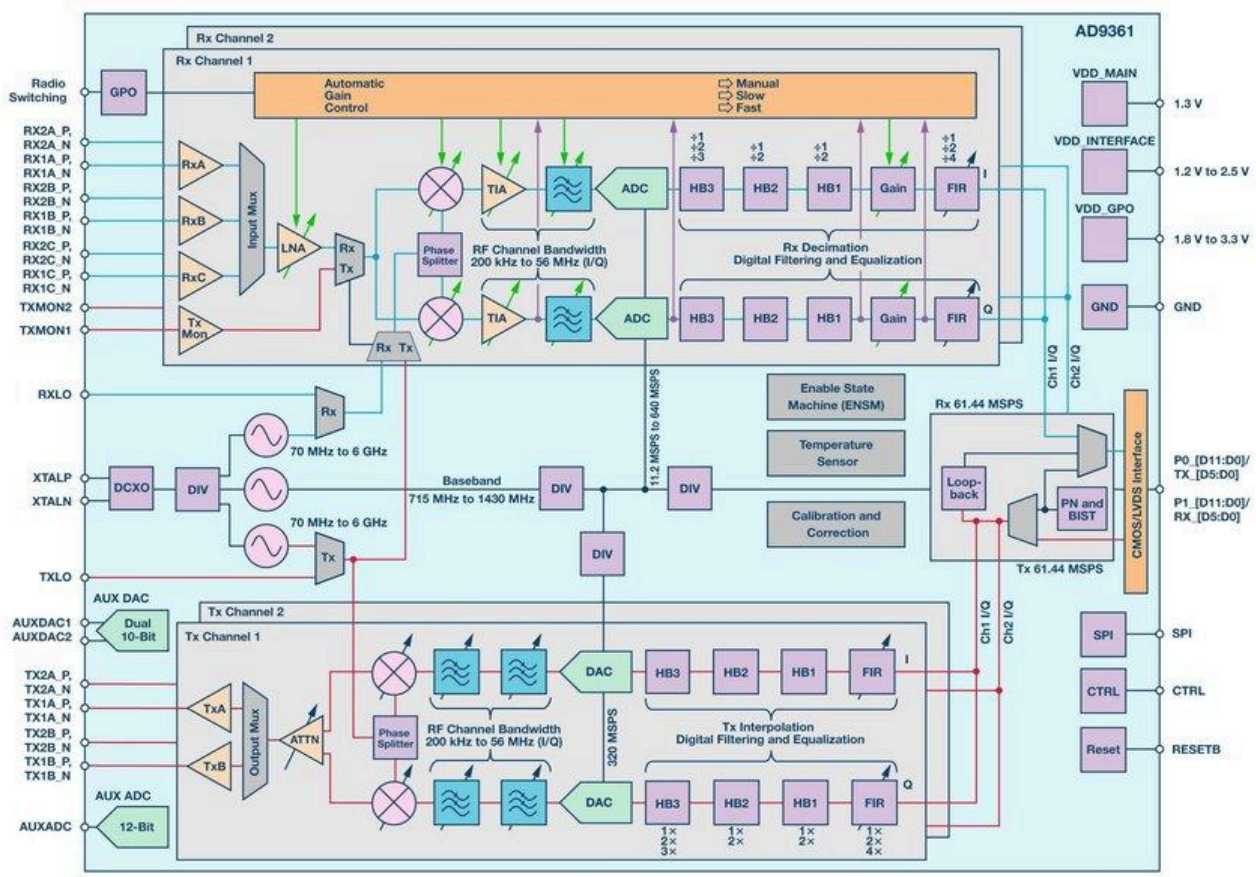
**PRACTICA PROFESIONAL SUPERVISADA**

**Test 1** ○ **Test 2** ◉ **Loopback 0** ◉ **Loopback 1** ○



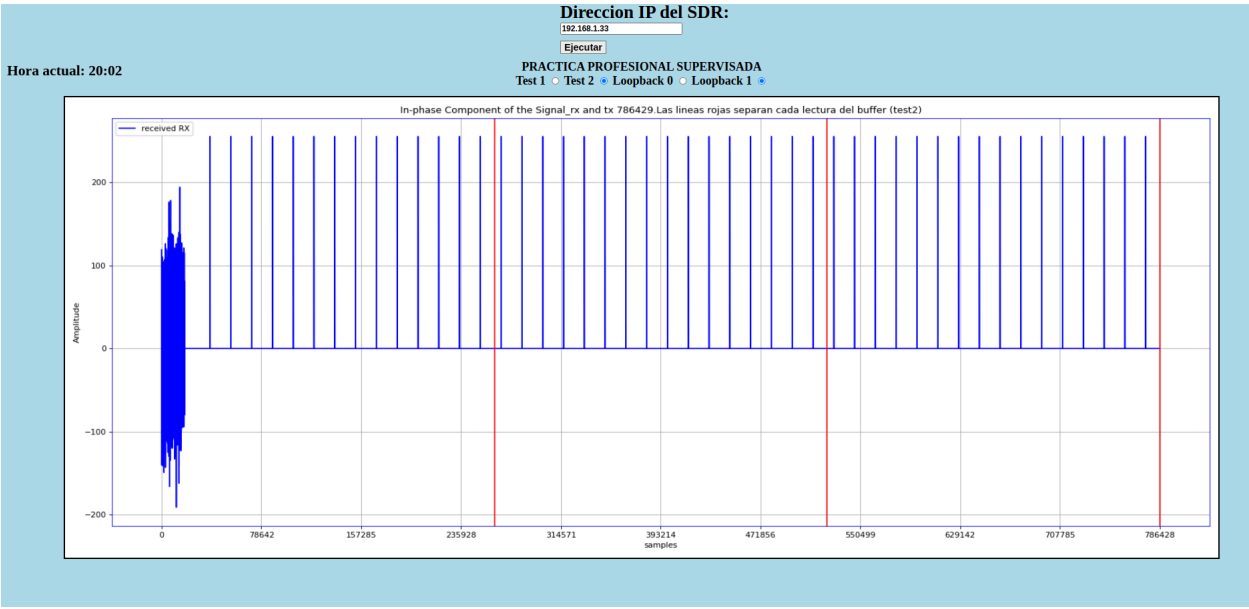



La SDR posee un transceptor AD9363 que contiene lo que se observa en la figura. Para más información, remitirse a la datasheet:



# RESULTADOS

El buffer cuenta con dificultades a la hora de su lectura. Una de ellas ocurre cuando se lee por primera vez y es posible no estar leyendo la información de la simulación actual, en vez de eso, se está leyendo información desactualizada, es decir, lo que tenía el buffer antes de la simulación. Por ejemplo, al visualizar la siguiente imagen se observan datos con ruido al comienzo del gráfico y esto es debido a que la simulación anterior fue realizada con loopback RF, de este modo los datos leídos son de esa simulación y la simulación actual fue con el modo loopback digital, por lo que al leer el buffer primero se está leyendo alguna muestra de la simulación en modo RF.

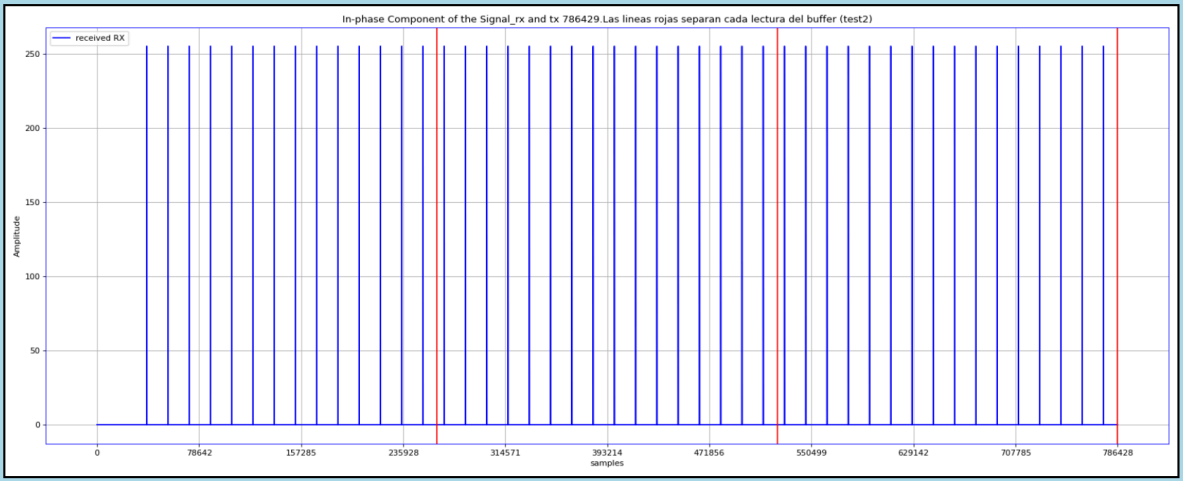



Otra simulación:

**Hora actual: 15:14**

**PRACTICA PROFESIONAL SUPERVISADA**

**Test 1** ○ **Test 2** ◉




**Direccion IP del SDR:**

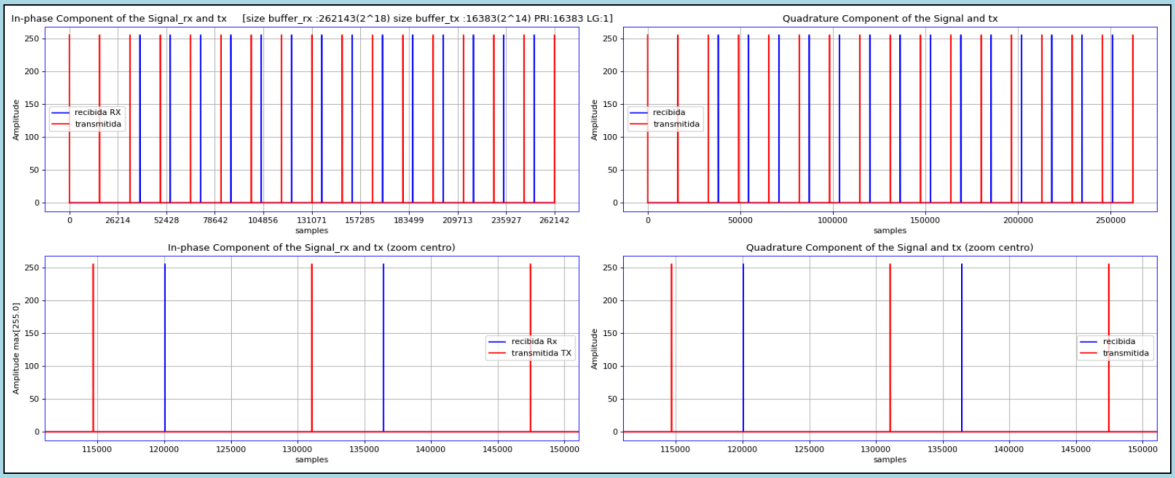
192.168.1.32

Ejecutar

**Hora actual: 21:21**

**PRACTICA PROFESIONAL SUPERVISADA**

**Test 1** ◉ **Test 2** ○ **Loopback 0** ○ **Loopback 1** ◉

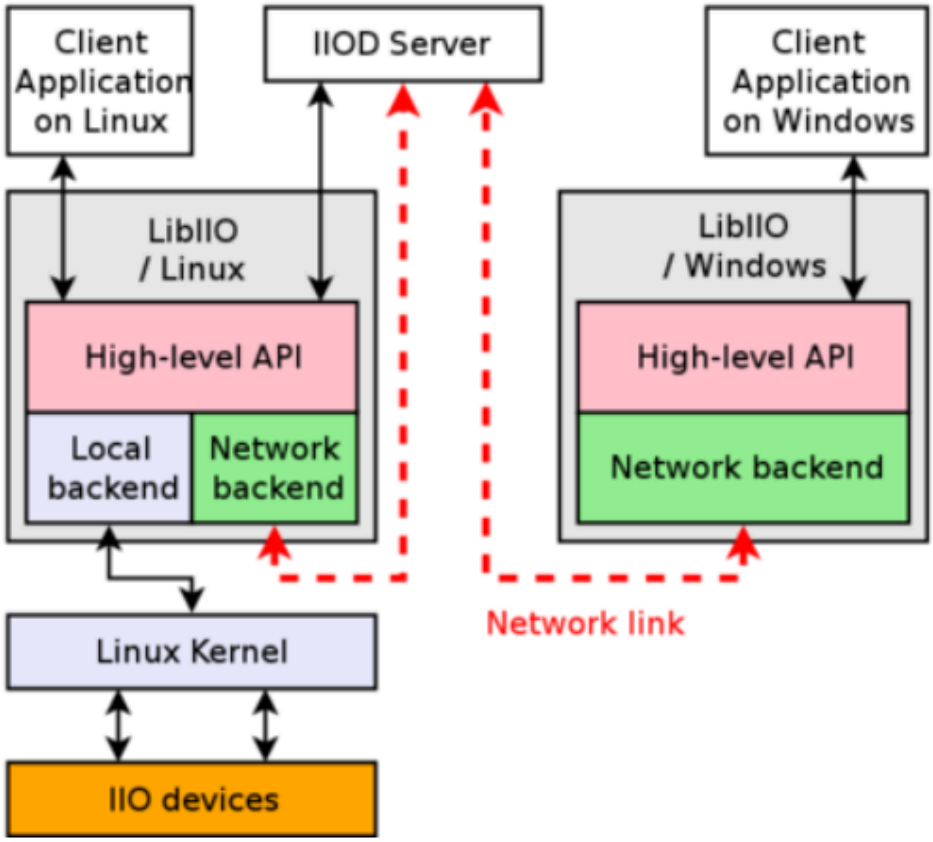


## CONCLUSIÓN

El proyecto presentó un gran desafío debido a que las SDRs funcionan de manera muy distinta de acuerdo al entorno donde se encuentran. Si se trabaja con loopback RF, como un primer paso, se utilizó IIO Oscilloscope que permite observar el tipo de señal que se transmite, también se utilizaron otras SDRs para recibir la señal enviada por la SDR. En un principio también se tuvo en cuenta la documentación de ***adi*** de Python para poder llegar a una similitud con la librería iio.h de C. C permite trabajar en un nivel más bajo que Python. El objetivo era tener un conocimiento de funcionamiento del buffer lo cual se logra con lo visto anteriormente.

Los próximos avances serán un procesamiento de mayor calidad de las señales recibidas y reducir la brecha entre el envío del pulso y la recepción.

Observación de la distribución del software:



# REFERENCIAS


1. Docker
2. Datasheet AD9363
3. iio-transceiver/ad9361
4. libiio/v0.24
5. IIO-Oscilloscope
6. Ejemplo_en_C