Módulo de calibración del receptor de holografía para el Radiotelescopio de 40m del Centro Astronómico de Yebes

D. Cordobés, J.A. López Pérez,
C. Almendros, J.A. Abad,
J. M. Yagüe, S. Henche

Informe Técnico IT - OAN 2008 - 12

CONTENIDO

I. Introducción

En este informe se presenta la integración y las medidas del módulo HOLOCAL, que permitirá determinar la estabilidad en amplitud y fase del receptor de holografía del Centro Astronómico de Yebes. Para ello, este módulo genera un tono a 11.2GHz que es inyectado simultáneamente en los dos canales del receptor mediante un acoplador direccional situado antes de los LNAs. De esta forma se podrán separar y evaluar las contribuciones de la atmósfera y del receptor al error del sistema.

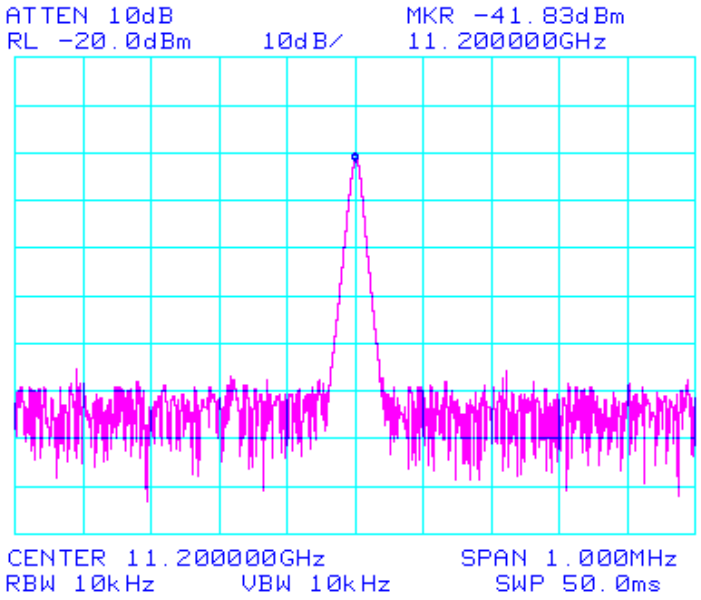


Figura 1. Tonos de 11.2 GHz empleados para calibrar el receptor de holografía

II. Montaje y medida del módulo

1) Montaje

El diagrama de bloques del módulo se muestra en la **Figura 2.** El tono de 11.2 GHz se obtiene a partir de uno de 2.24GHz generado por un sintetizador Herley BBS-2330, mediante un multiplicador de frecuencia basado en un diodo SRD [1]. El oscilador se engancha a una referencia externa de 10MHz y 0 ± 3dBm, de la cual se obtiene una copia que se envía hacia un conector de salida en el panel frontal.

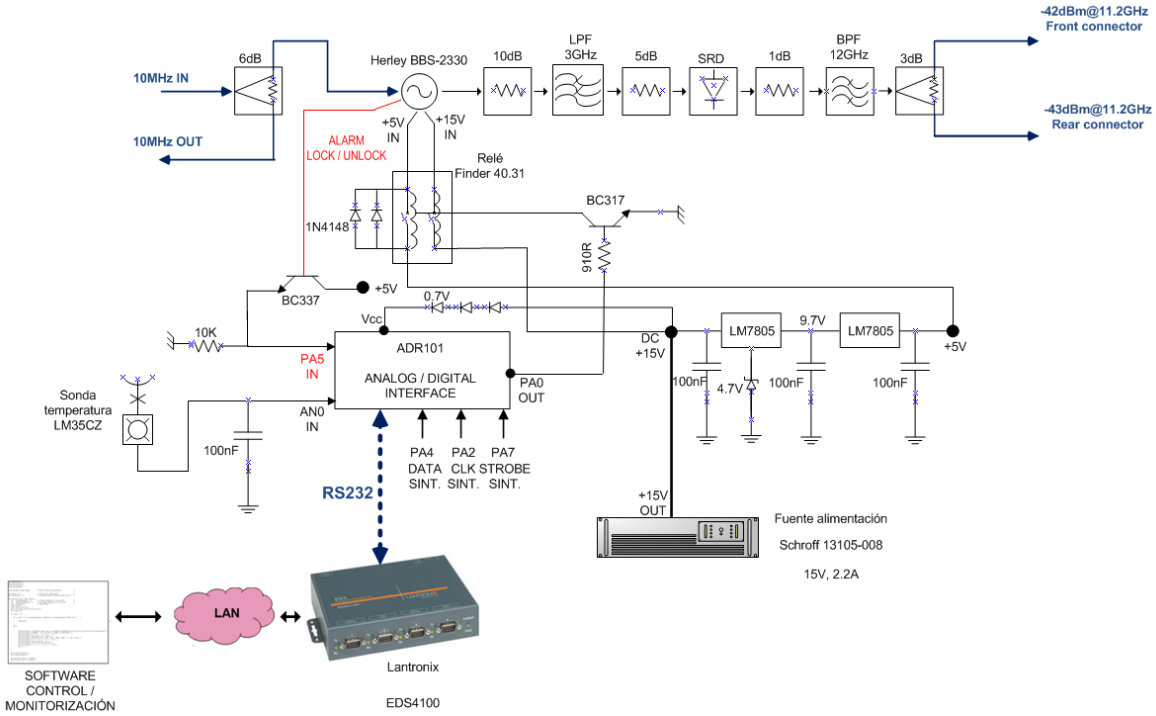


Figura 2. Diagrama de bloques del módulo

La señal de salida del oscilador, se filtra paso bajo para eliminar armónicos no deseados, se multiplica en frecuencia por el diodo SRD y se filtra paso banda para seleccionar el armónico quinto. Mediante un divisor de 3dB se obtienen dos muestras del armónico quinto (11.2 GHz), uno en el panel frontal y otro en el trasero. El del panel frontal

irá al acoplador del canal de test del receptor de holografía (TEST) y el del trasero al acoplador del canal de referencia (REF).

El control de la alimentación y del enganche del oscilador se realiza a través de una tarjeta Ontrak ADR101, con entradas y salidas digitales. En concreto, la alimentación se controla desde la salida digital PA0 y el enganche desde la entrada digital PA5. En la entrada analógica AN0 se monitoriza la temperatura medida por una sonda LM35CZ. El sintetizador requiere dos fuentes de alimentación, una de +15V y otra de +5V. Una fuente Schroff 13105-008 proporciona los +15V y los +5V se obtienen con una doble etapa de reguladores. El control de ambas fuentes se hace desde el ADR101 y se efectúa a través de un relé. Como la corriente que proporciona la salida PA0 es insuficiente, se ha utilizado un amplificador de corriente basado en el transistor BC317. Para la señal de enganche del sintetizador se empleó un transistor BC337 para adecuar los niveles a valores TTL.

El módulo se fabricó en un subrack Schroff y se puede ver en las **Figuras 4, 5 y 6.**

Figura 3. Vista general del módulo

Figura 4. Panel frontal del módulo

Figura 5. Panel trasero del módulo

2) Medida

2.1- Control de la tarjeta ADR101

La tarjeta ADR101 controla el relé a través del cual se alimenta el oscilador y monitoriza el estado del enganche del mismo. Se controla a través de una interfaz RS232 configurada de la siguiente forma:

9600baud – 8bit words – 1 stop bit – no parity – flux control Hw

La conexión de la tarjeta a un PC se realiza con este *pinout*

ADR101	PC
Pin 2: GND	Pin 5 :GND
Pin 3: Rx	Pin 3 :Tx
Pin 7: Tx	Pin 2: Rx

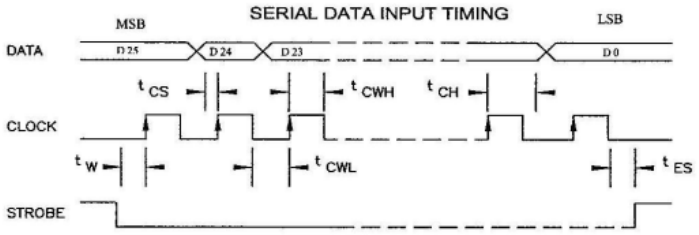
Los comandos que se han de utilizar son:

- *CPA00100000*: Configura PA0 como salida digital, PA5 como entrada y el resto salidas.
- Encendido del sintetizador: *MA1*
- Apagado del sintetizador: *MA0*
- Lectura de la alarma de enganche: *RPA5*. *0* indica desenganche, *1* enganche.

2.2- Control del sintetizador

El sintetizador tiene un rango de salida de 2.1GHz a 2.6GHz, en pasos de 10Hz. La selección de la frecuencia de salida se realiza a través de un protocolo de comunicación en el que intervienen tres señales: *DATA, CLOCK y STROBE*, de la forma en la que se muestra en la **Figura 6.** La forma de operar es la siguiente: la frecuencia que se desea obtener en GHz se resta de 2.1GHz y se obtiene el canal [0 a 50000000]. El canal se transforma a binario con 26bits y se manda en los flancos de subida de la señal *CLOCK*. La señal *STROBE* marca el inicio y el fin de la transmisión.

BBS-2330



t_{CS}	DATA TO CLOCK SETUP TIME	SEE DATA INPUT TIMING	50 US MINIMUM
t_{CH}	DATA TO CLOCK HOLD TIME	SEE DATA INPUT TIMING	50 US MINIMUM
t_{CWH}	CLOCK PULSE WIDTH HIGH	SEE DATA INPUT TIMING	50 US MINIMUM
t_{CWL}	CLOCK PULSE WIDTH LOW	SEE DATA INPUT TIMING	50 US MINIMUM
t_{ES}	CLOCK TO ENABLE SET UP TIME	SEE DATA INPUT TIMING	50 US MINIMUM
t_{W}	STROBE TO FIRST CLOCK	SEE DATA INPUT TIMING	1000 US MINIMUM

SERIAL DATA FORMAT — DATA IS SHIFTED MSB FIRST

		MSB																								LSB	
OUTPUT	CHANNEL	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
FREQUENCY	NUMBER	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
2100.00000 MHz	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2100.00001 MHz	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
2600.00000 MHz	50000000	1	0	1	1	1	1	1	0	1	0	1	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0

Figura 6. Diagrama de temporal de control del sintetizador BBS-2330

El control de las señales *DATA, CLOCK* y *STROBE*, también se realiza a través de la tarjeta ADR101. En concreto *DATA* está cableada a la salida digital

PA4, *CLOCK* a PA2 y *STROBE* a PA7. El control del sintetizador se hace a través de la red Ethernet a través de un terminal Lantronix.

2.3- Software

El control del módulo HOLOCAL se realiza a través de una interfaz de usuario manejada por un programa en Python (**Figura 7**).

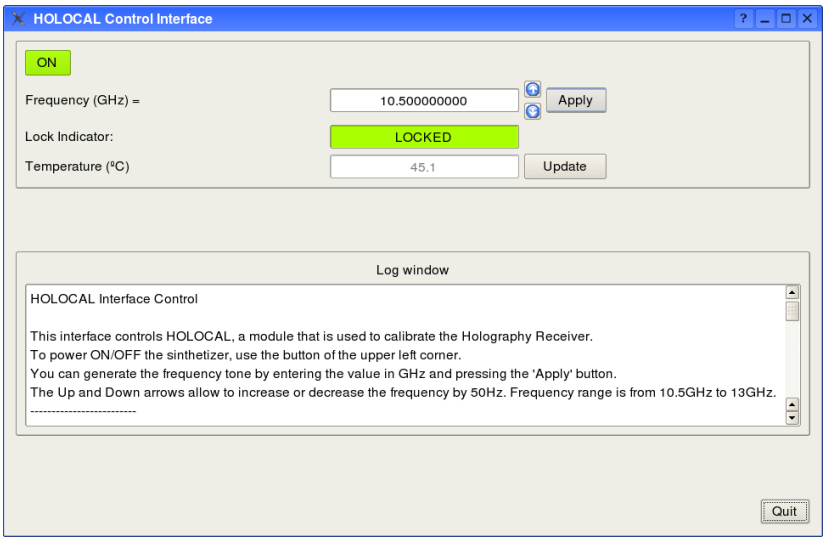


Figura 7. Interfaz gráfica de usuario para controlar HOLOCAL

Esta interfaz permite controlar el encendido y apagado del sintetizador, seleccionar la frecuencia de salida deseada y monitorizar la temperatura del entorno del sintetizador así como el estado del enganche en frecuencia. El programa en Python se muestra en el **Apéndice I**.

2.4- Medidas en el conector del panel frontal

En la **Figura 8** se muestran medidas de la señal de salida del módulo en el panel frontal con anchos de resolución de 1 MHz y de 100Hz. Tal como se puede apreciar, el tono de salida está libre de espúreos y tiene un nivel de -42dBm.

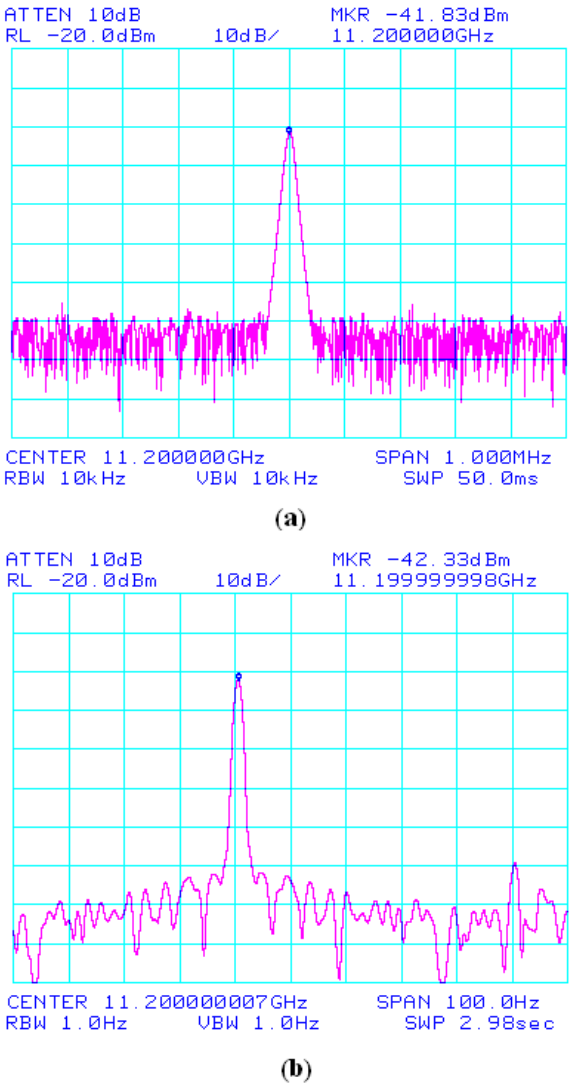


Figura 8. Pureza espectral en la señal de salida por el conector frontal.
Zoom alrededor de la portadora de (a) 1MHz (b) 100Hz

2.5- Medidas en el conector del panel trasero

De forma similar, en la **Figura 9** se muestran medidas de la señal de salida del módulo en el panel trasero con anchos de resolución de 1 MHz y de 100Hz. De nuevo se puede apreciar que el tono de salida está libre de espúreos y tiene un nivel de -43dBm.

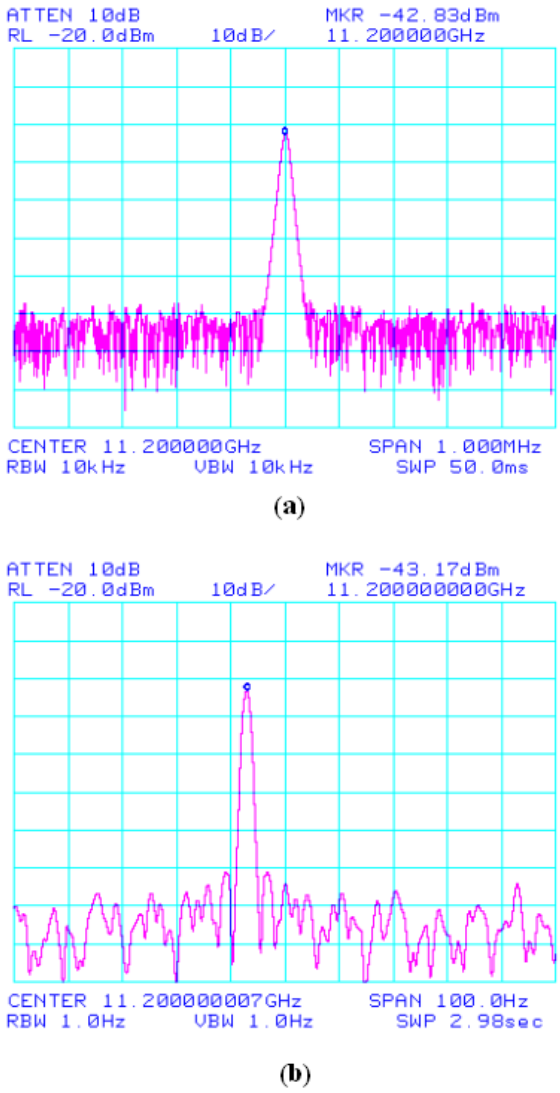


Figura 9. Pureza espectral en la señal de salida por el conector trasero.
Zoom alrededor de la portadora de (a) 1MHz (b) 100Hz

2.6- Medidas de rizado

El sintetizador genera frecuencias entre 2.1 y 2.6GHz, que tras ser multiplicadas por 5, se convierten al intervalo de 10.5 a 13GHz. Se han hecho medidas para ver el rizado de esa banda (**Figura 10**).

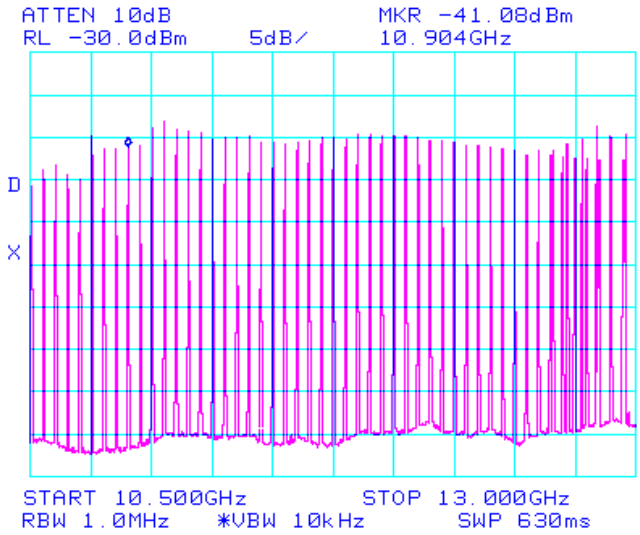


Figura 10. Rizado en la banda de salida del sintetizador

Como se puede ver, el rizado máximo pico-pico en la banda de 11.2 a 12.75GHz, es de 3dB y el nivel de potencia medio de -40dBm.

III. Referencias

[1] D. Cordobés, J. A. López Pérez, C. Almendros, J. A. Abad, J.M. Yagüe, S. Henche, *Montaje y medida de un multiplicador de frecuencia basado en un diodo SRD*, Informe Técnico IT- OAN 2007-10.

Apéndice I: Software en Python

```
#!/usr/bin/env python
# -*- coding: ISO-8859-15 -*

#/* HOLOCAL
# *
# * Copyright (C) 2004
# * Observatorio Astronómico Nacional, Spain
# *
# * This library is free software; you can redistribute it and/or modify it under
# * the terms of the GNU Library General Public License as published by the Free
# * Software Foundation; either version 2 of the License, or (at your option) any
# * later version.
# *
# * This library is distributed in the hope that it will be useful, but WITHOUT
# * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY FITNESS
# * FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more
# * details.
# *
# * You should have received a copy of the GNU Library General Public License
# * along with this library; if not, write to the Free Software Foundation, Inc.,
# * 675 Massachusetts Ave, Cambridge, MA 02139, USA. Correspondence concerning
# * APEX should be addressed as follows:
# *
# *  who              when        what
# * --------       ----------  --------------------------------
# * David Cordobés Gallo & José Antonio López Pérez    16/10/2008     created
# */



import sys, time, os
from time import sleep
from qt import *
from string import *
from holocal_gui import *
from math import sin, cos,pi
from socketclient import *


class surfaceChild(holocal_gui):

 def __init__(self):
  holocal_gui.__init__(self)


  self.internalTimer = QTimer(self) # create internal timer
  self.internalTimer.start(5000)    # Update temp every 5 sec
  self.__buffersize=1024
  self.__puerto=10003
  self.cliente = tcpSocketClient('192.168.0.121',self.__puerto)

  self.__cmd="CPA00100000\r\n" #La 6 es entrada de alarma, el resto salidas
  self.cliente.write(self.__cmd)
  sleep(0.1)

  self.updateTemp()

  self.__cmd="ma0\r\n" #Apago el sint
```

```
 self.cliente.write(self.__cmd)
 sleep(0.1)

 self.powerButton.setText("OFF")
 self.powerButton.setPaletteBackgroundColor(QColor(255,0,0))

 self.log_gui.clear()

  self.log_gui.clear()
 self.log_gui.append("-------------------------\n")
 self.log_gui.append("HOLOCAL Interface Control\n\n")
 self.log_gui.append("This interface controls HOLOCAL, a module that is used to calibrate the Holography Receiver.\n")
 self.log_gui.append("To power ON/OFF the sinthetizer, use the button of the upper left corner.\n")
 self.log_gui.append("You can generate the frequency tone by entering the value in GHz and pressing the 'Apply' button.\n")
 self.log_gui.append("The Up and Down arrows allow to increase or decrease the frequency by 50Hz. Frequency range is from 10.5GHz
to 13GHz.\n")
 self.log_gui.append("-------------------------\n\n")


def output_freq(self):

 # En esta función escribo la frecuencia deseada

 freq_gui= self.freq_input_gui.text().ascii()

 freq= float(freq_gui)*1e9
 freq=freq/5.0

 self.powerButton.setText("ON")
 self.powerButton.setPaletteBackgroundColor(QColor(170,255,0))
 self.__cmd="ma1\r\n" #Enciendo el sint
 self.cliente.write(self.__cmd)
 sleep(1)

 num_canal=freq-2100000000
 num_canal=num_canal/10 #RESTO entero
 num_canal=int(num_canal)

 count=26
 num_binario="".join([str((num_canal >> y) & 1) for y in range(count-1, -1, -1)]) #Convierte de decimal a binario

 for i in range(0,26):
        self.escribe_adr_101(num_binario[i])

 self.__cmd="ma129\r\n" #Pongo a 1 el enable (fin ciclo)
 self.cliente.write(self.__cmd)
 sleep(0.1)

 self.log_gui.append("HOLOCAL output frequency (GHz): %s\n" %(freq_gui))

 self.__cmd="RPA6\r\n" #Leo el enganche
 self.cliente.write(self.__cmd)
 sleep(0.1)

 lock=self.cliente.read(self.__buffersize)
 sleep(0.1)
 lock=atoi(lock)
 if lock == 1:
        self.SintLockValue.setText("LOCKED")
        self.SintLockValue.setPaletteBackgroundColor(QColor(170,255,0))
        self.log_gui.append("Sinthetizer locked\n" )

 else:
        self.SintLockValue.setText("UNLOCKED")
        self.SintLockValue.setPaletteBackgroundColor(QColor(255,0,0))
        self.log_gui.append("WARNING: Sinthetizer unlocked\n" )
```

```
def escribe_adr_101(self, data):

        if data == "1":

                self.__cmd="ma1\r\n"
                self.cliente.write(self.__cmd)
                sleep(0.01)

                self.__cmd="ma17\r\n" #Pongo "1" en DATA
                self.cliente.write(self.__cmd)
                sleep(0.01)

                self.__cmd="ma21\r\n" #Pongo "1" en CLK
                self.cliente.write(self.__cmd)
                sleep(0.01)


        elif data == "0":

                self.__cmd="ma1\r\n" #Pongo "0" en DATA
                self.cliente.write(self.__cmd)
                sleep(0.01)

                self.__cmd="ma5\r\n" #Pongo "1" en CLK
                self.cliente.write(self.__cmd)
                sleep(0.01)


def updateTemp(self):


        self.__cmd="RD0\r\n"
        self.cliente.write(self.__cmd)
        sleep(1)

        temp=self.cliente.read(self.__buffersize)

        temp_float=atof(temp)

        temp_float=100*5*temp_float/255

        temp_float = "%3.1f" % (temp_float)

        self.temp_gui.setText(temp_float)
        self.log_gui.append("HOLOCAL internal temperature (ºC): %s\n" %(temp_float))



def sintPowerButton(self):


        self.__cmd="RPA0\r\n"
        self.cliente.write(self.__cmd)
        sleep(0.1)

        temp=self.cliente.read(self.__buffersize)

        sleep(0.1)
        power=atoi(temp)

        if power == 1:
                self.powerButton.setText("OFF")
                self.powerButton.setPaletteBackgroundColor(QColor(255,0,0))
                self.__cmd="ma0\r\n"
                self.cliente.write(self.__cmd)
```

```
                sleep(0.01)
                self.SintLockValue.setText("UNLOCKED")
                self.SintLockValue.setPaletteBackgroundColor(QColor(255,0,0))
                self.log_gui.append("Sinthetizer switched OFF\n" )

        else:
                self.powerButton.setText("ON")
                self.powerButton.setPaletteBackgroundColor(QColor(170,255,0))
                self.__cmd="ma1\r\n"
                self.cliente.write(self.__cmd)
                sleep(0.01)
                self.log_gui.append("Sinthetizer switched ON\n" )


def plusfreq(self):
        freq= self.freq_input_gui.text().ascii()
        f=float(self.freq_input_gui.text().ascii())
        f=f+0.00000005

        self.freq_input_gui.setText("%11.9f" %f)
        self.output_freq()


def minusfreq(self):

        f=float(self.freq_input_gui.text().ascii())
        f=f-0.00000005

        if f<10.50000000:
                f=10.50000000
                self.log_gui.append("WARNING: Minimum frequency value reached.\n")

        if f>13.00000000:
                f=13.00000000
                self.log_gui.append("WARNING: Maximum frequency value reached.\n")

        self.freq_input_gui.setText("%11.9f" %f)
        self.output_freq()



def quit_(self):

 print "EXIT"

 self.__cmd="ma0\r\n" #Apago el sint
 self.cliente.write(self.__cmd)
 sleep(0.1)
 self.powerButton.setText("OFF")
 self.powerButton.setPaletteBackgroundColor(QColor(255,0,0))

 os._exit(0)


def __del__(self):
        self.quit_()


def main(args):

 aplicacion = QApplication(sys.argv)

 sWindow = surfaceChild()
 aplicacion.setMainWidget(sWindow)
```

```
 sWindow.show()


 #sWindow.connect(sWindow.quitButton,SIGNAL("clicked()"),aplicacion,SLOT("quit()") )
 sWindow.connect(sWindow.powerButton,SIGNAL("clicked()"),sWindow.sintPowerButton)
 sWindow.connect(sWindow.applyButton,SIGNAL("clicked()"), sWindow.output_freq)
 sWindow.connect(sWindow.quitButton,SIGNAL("clicked()"), sWindow.quit_)
 sWindow.connect(sWindow.updateTempButton,SIGNAL("clicked()"), sWindow.updateTemp)
 sWindow.connect(sWindow.plusfreqButton,SIGNAL("clicked()"),sWindow.plusfreq)
 sWindow.connect(sWindow.minusfreqButton,SIGNAL("clicked()"),sWindow.minusfreq)
 #sWindow.connect(sWindow, SIGNAL("lastWindowClosed()"),sWindow.quit_)

  #sWindow.connect(sWindow.internalTimer,SIGNAL("timeout()"),sWindow.timeout)

  aplicacion.exec_loop()

if __name__=="__main__":
 main(sys.argv)
```