Objectives

Embedded digital radiofrequency network analyzer:

1. embedded operating system generated from the cross-compilation framework Buildroot
2. radiofrequency signal recording using GNU Radio running on the embedded board from an RTL-SDR DVB-T dongle + server on the embedded board for controlling the recording parameters
3. **signal source to probe the device under test**
4. programming the signal source and transfering to the PC to complete the measurement

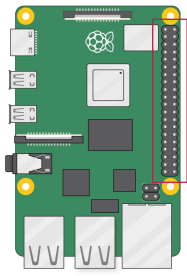
Tunable radiofrequency source:

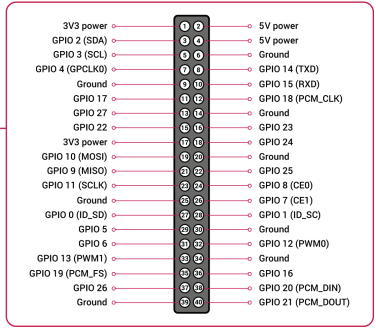
- Voltage Controlled Oscillator (VCO) requires a tuning voltage + non-linear response of frequency v.s tuning voltage requires calibration (linearization)
- Phase Locked Loop (**PLL**): poor resolution + settling time, **available on GPIO4 of the RPi4**
- Direct Digital Synthesizer (DDS): digital radiofrequency signal generation
- Selected signal source: **Analog Devices AD9954** requiring a dedicated board + **SPI communication**

Printed Circuit Board (PCB) design flow

Design a dedicated signal source based on the AD9954 DDS:

1. Datasheet analysis, evaluation board as example, passive peripheral components and link to signals available from the Raspberry Pi4 40-pin bus →
   passive external components, supply voltage, signals
2. Schematic: logical link between components through signals
3. Bill of Materials (BoM): list of components, supplier, reference and price
4. Custom footprint (SAW ceramic package)
5. Board: routing signals between components
6. Mechanical analysis (FreeCAD [1])



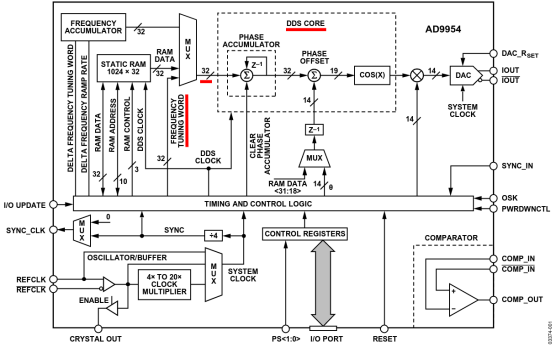


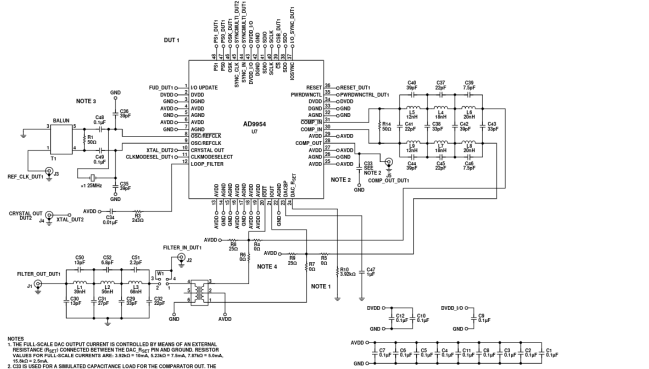
[1] https://www.raspberrypi.com/documentation/computers/images/GPIO-Pinout-Diagram-2.png
[1] https://wiki.freecadweb.org/KicadStepUp_Workbench/it

Datasheet analysis

DDS principle: a 32-bit phase accumulator counts up to the frequency tuning word (FTW), the phase counter is converted to a sine wave by a lookup table driving a Digital to Ananlog Converter (DAC)

$$f_{out} = f_{ref} \times FTW/2^N$$





Sample schematic from the evaluation board description:

Datasheet analysis

Pinout analysis: which are the relevant pins (beyond supply), logic function and acceptable voltages ⇒ power supply management

PIN CONFIGURATION AND FUNCTION DESCRIPTIONS

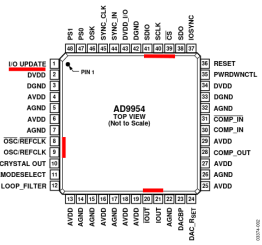
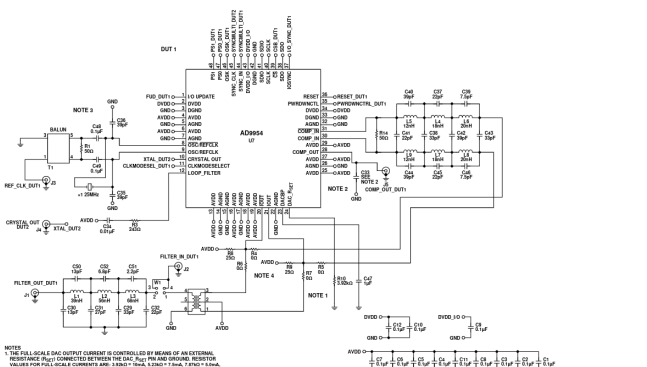


Figure 4. Pin Configuration

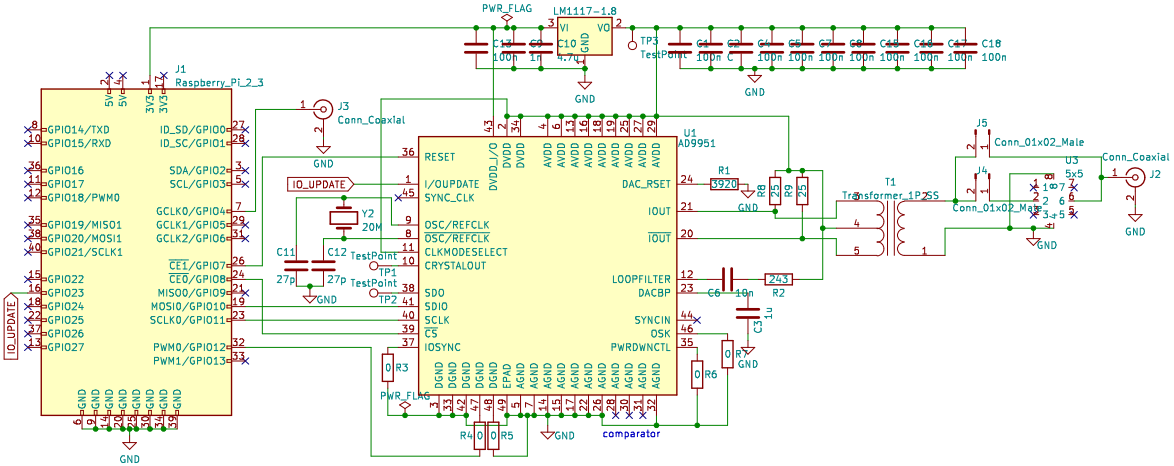
Note that the exposed paddle on the bottom of the package forms an electrical connection for the DAC and must be attached to analog ground. Note that Pin 43, DVDD_I/O, can be powered to 1.8 V or 3.3 V. The DVDD pins (Pin 2 and Pin 34) must be powered to 1.8 V.



Sample schematic from the evaluation board description:

Schematic

Use the AD9951 KiCAD part (similar to AD9954 except for two pins)



Decoupling capacitors (as many as supply voltage pins)

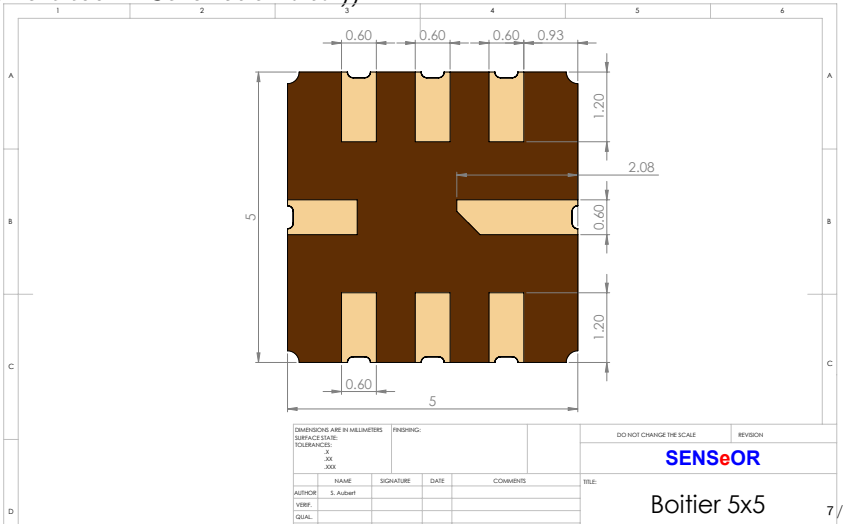
Bill of Materials

Ref	Qnty	Value	Reference	Vendor	Price
C1, C4, C5, C7, C8, C13	6	100n			
C2	1	C			
C3	1	1u			
C6	1	10n			
C9	1	1n			
C10	1	4.7u			
C11, C12	2	27p			
J1	1	Raspberry Pi 2 3 Connector[2]			
J2, J3	2	Connector Coaxial (SMA)			
J4, J5	2	Conn 01x02 Male			
R1	1	3920			
R2	1	243			
R3, R4, R5, R6, R7	5	0			
R8, R9	2	25			
T1	1	Transformer 1P SS			
TP1, TP2, TP3	3	TestPoint			
U1	1	AD9951 400 MSPS DDS[3]			
U2	1	LM1117-1.8 800 mA Linear Regulator[4]			
Y2	1	20MHz Crystal			

[4] https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/rpi_SCH_3bplus_1p0_reduced.pdf
[4] https://www.analog.com/static/imported-files/data_sheets/AD9951.pdf
[4] http://www.ti.com/lit/ds/symlink/lm1117.pdf

Drawing a custom footprint: SENSeOR SEAS10 filter ceramic package

- KiCAD separates the logic description of the component (schematic) and its footprint (board
- Under many circumstances, ability to describe a new component and associate with an existing footprint (*Footprint assignement tool* in Schematic Editor))
- Create a new component: Symbol Editor
- Create a new footprint: Footprint Editor
- Input = pin 2, output = pin 6, 4&8=GND



Drawing a custom footprint: SENSeOR SEAS10 filter ceramic package

- KiCAD separates the logic description of the component (schematic) and its footprint (board
- Under many circumstances, ability to describe a new component and associate with an existing footprint (*Footprint assignement tool* in Schematic Editor))
- Create a new component: Symbol Editor
- Create a new footprint: Footprint Editor
- Input = pin 2, output = pin 6, 4&8=GND

Mechanical specifications - Figure 1	
Typical package dimensions	5x5 mm
Maximum package thickness	1.5 mm
Connection to antenna - monopole type i.e. single ended configuration (pin n°)	Input and output: 6 Ground plane: 2,4,8 Not connected: 1,3,5,7
Connection to antenna - dipole type i.e. differential configuration (pin n°)	Input and output: 2,6 Ground plane: 4,8 Not connected: 1,3,5,7

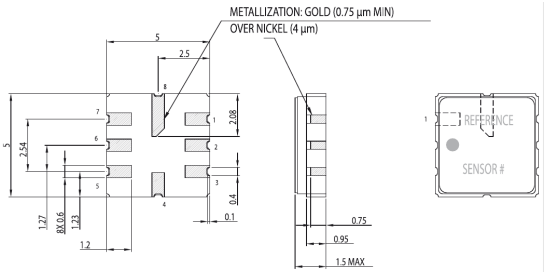
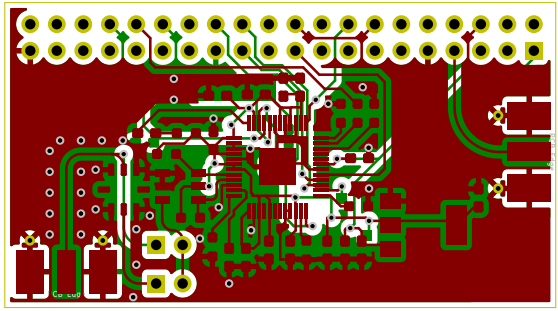
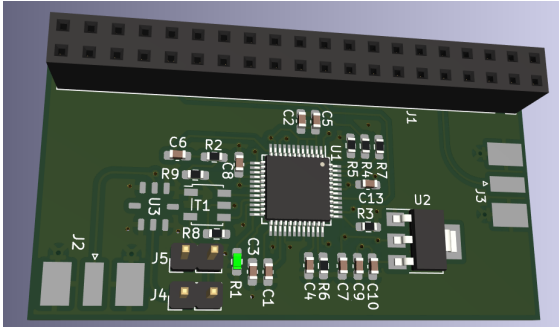


Figure 1: 5x5 mm ceramic package

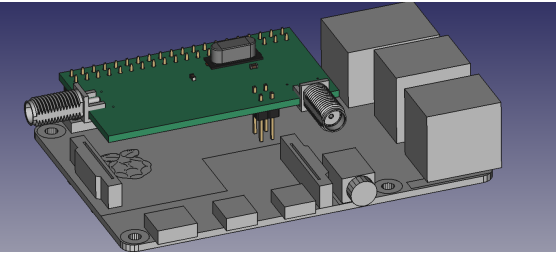
Board

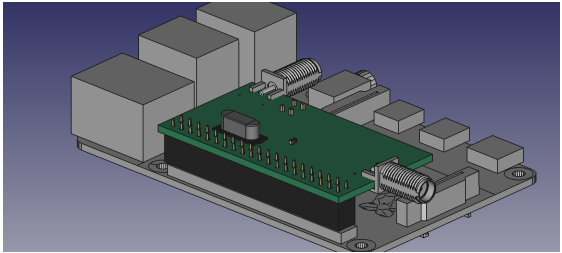
- Two layer board (top, bottom)
- Do not bother routing ground power supplies: **ground planes** on both sides
- shield radiofrequency tracks, avoid intersection with supply and digital signals on the opposite side
- short track length from radiofrequency signal generation to connectors
- decoupling capacitors as **close** as possible to each supply
- short track length from Osc pins to resontor (avoid parasiticy capacitance/inductance)

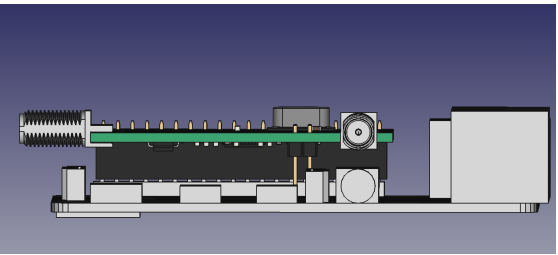




Mechanical layout: FreeCAD







- KiCAD plugin in FreeCAD
- 3D model of the Raspberry Pi4 (.step format)
- merge the two models to assess connector location and compatibility with existing mechanical constraint (e.g. Ethernet & USB connectors)
- screws? mechanical stability?

Activating SPI (DDS communication)

1. activate SPI Linux driver [5] : in the SD card first partition `config.txt`, add `dtaparam=spi=on`
   After reboot, check that `/dev/spidev0*` exists (.0 and .1 refer to CS0 and CS1). Also, check that the SPI related modules are loaded (`lsmod`)

```
spidev                 24576  0
spi_bcm2835            24576  0
```

2. In Buildroot, activate the packages `python-spidev` from `Target packages→Interpreter languages and scripting→Python3→External Modules` as well as `python-pigpio` from the same sub-menu
3. activate `pigpio` for the associated daemon in `Target packages→Hardware handling` (will be useful for testing IO_UPDATE): check the daemon is running

```
# ps aux | grep pigp
  199 root      /usr/bin/pigpiod
```

4. remember to toggle IO_UPDATE after each SPI transaction is completed and to handle the RESET signal

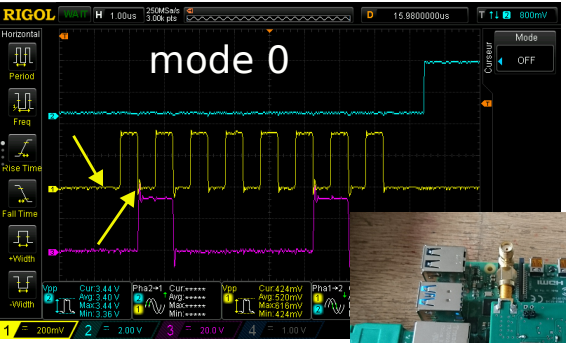
[5] `https://learn.sparkfun.com/tutorials/raspberry-pi-spi-and-i2c-tutorial/all`

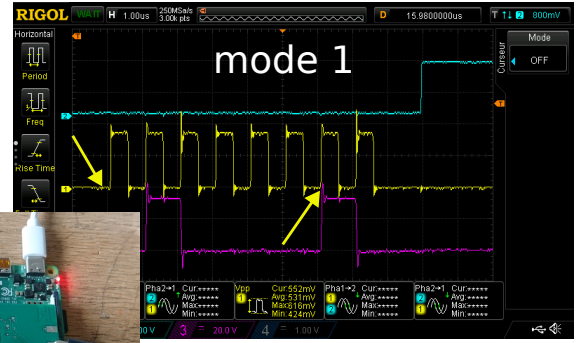
Testing SPI communication with the DDS

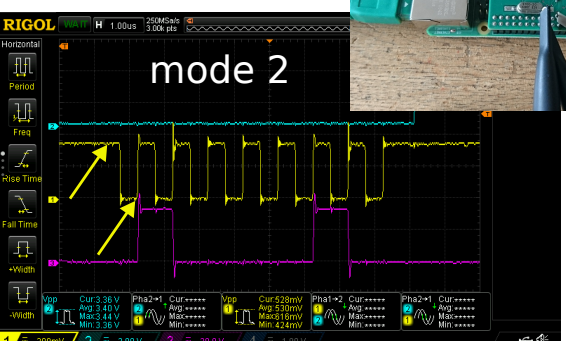
Basic sample program for SPI (check CK, MOSI and CS# with the oscilloscope):

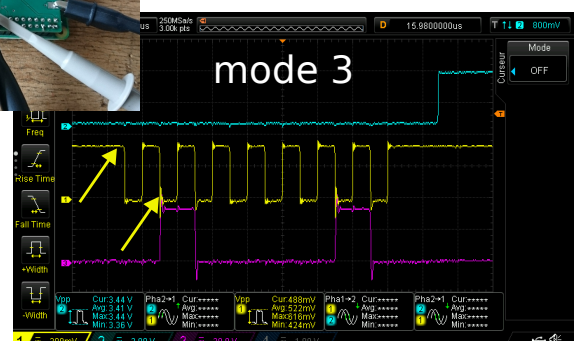
```
import time
import spidev

bus = 0        # SPI0
device = 0     # CS#
spi = spidev.SpiDev()
spi.open(bus, device)
spi.max_speed_hz = 500000
spi.mode = 0
msg = [0x42, 0x55, 0xAA]
spi.xfer2(msg)  # xfer should raise CS# between transactions but does not
```

Check how the various “mode” parameter affect CPHA and CPOL and select the one matching the DDS datasheet description.


RIGOL
mode 0

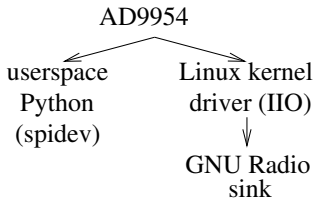

RIGOL
mode 1


RIGOL
mode 2


RIGOL
mode 3

Testing SPI communication with the DDS

- Test functional board: Python scripting to access SPI bus and GPIO
- Slow interpreted language ⇒ Linux kernel module
- Compliance with Linux API: IIO driver
- Use `gr-iio` or custom sink from GNU Radio to access the signal sink




```
# Datasheet AD9954 p.23:
# SPI CK rest state=0, change on falling edge => mode 0
# GPIO12 => 47
# GPIO23 => IO_UPDATE
import time
import spidev
import pigpio
bus = 0       # SPI0
device = 0    # CS#
spi = spidev.SpiDev()
spi.open(bus, device)
spi.max_speed_hz = 1000000
spi.mode = 0

pi=pigpio.pi()
pi.set_mode(12,pigpio.OUTPUT)
pi.write(12, 0)  # PS0 low

pi.set_mode(23,pigpio.OUTPUT)
pi.write(23, 0)  # IO_UPDATE low

pi.set_mode( 7,pigpio.OUTPUT)
pi.write( 7, 0)  # reset low
```

```
time.sleep(0.01)
pi.write( 7, 1)  # reset hi
time.sleep(0.01)
pi.write( 7, 0)  # reset low

CFR1=[0x00, 0x00, 0x00, 0x00, 0x40]; # CFR1 disable comparator
CFR2=[0x01, 0xC4, 0x02, 0x94];       # CFR2 REFCLK Multiplier
ASF =[0x02, 0x04, 0x55];             # Auto Ramp Rate Speed
ARR =[0x03, 0xff];                   # Amplitude Ramp Rate
POW0=[0x05, 0x00, 0x00];             # POW0 PHASE Offset World
FTW1=[0x06, 0x2C, 0x8B, 0x43, 0x95]; # FTW1 Frequency Tuning Word
FTW0=[0x04, 0x18, 0x2d, 0x82, 0xd8]; # FTW0 Frequency Tuning World
                                     # dec2hex(floor(34/360*2^32))

spi.xfer2(CFR1)
spi.xfer2(CFR2)
spi.xfer2(ASF)
spi.xfer2(ARR)
spi.xfer2(POW0)
spi.xfer2(FTW1)
spi.xfer2(FTW0)
pi.write(23, 1)  # IO_UPDATE
time.sleep(0.01)
pi.write(23, 0)
```