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 transferring 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**
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://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 Analog Converter (DAC)

\[ f_{\text{out}} = f_{\text{ref}} \times \frac{\text{FTW}}{2^N} \]

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

Note that the exposed pad 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:
Use the AD9951 KiCAD part (similar to AD9954 except for two pins)

Decoupling capacitors (as many as supply voltage pins)
## Bill of Materials

<table>
<thead>
<tr>
<th>Ref</th>
<th>Qnty</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1, C4, C5, C7, C8, C13</td>
<td>6</td>
<td>100n</td>
</tr>
<tr>
<td>C2</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>C3</td>
<td>1</td>
<td>1u</td>
</tr>
<tr>
<td>C6</td>
<td>1</td>
<td>10n</td>
</tr>
<tr>
<td>C9</td>
<td>1</td>
<td>1n</td>
</tr>
<tr>
<td>C10</td>
<td>1</td>
<td>4.7u</td>
</tr>
<tr>
<td>C11, C12</td>
<td>2</td>
<td>27p</td>
</tr>
<tr>
<td>J1</td>
<td>1</td>
<td>Raspberry Pi 2 3 Connector²</td>
</tr>
<tr>
<td>J2, J3</td>
<td>2</td>
<td>Connector Coaxial (SMA)</td>
</tr>
<tr>
<td>J4, J5</td>
<td>2</td>
<td>Conn 01x02 Male</td>
</tr>
<tr>
<td>R1</td>
<td>1</td>
<td>3920</td>
</tr>
<tr>
<td>R2</td>
<td>1</td>
<td>243</td>
</tr>
<tr>
<td>R3, R4, R5, R6, R7</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>R8, R9</td>
<td>2</td>
<td>25</td>
</tr>
<tr>
<td>T1</td>
<td>1</td>
<td>Transformer 1P SS</td>
</tr>
<tr>
<td>TP1, TP2, TP3</td>
<td>3</td>
<td>TestPoint</td>
</tr>
<tr>
<td>U1</td>
<td>1</td>
<td>AD9951 400 MSPS DDS³</td>
</tr>
<tr>
<td>U2</td>
<td>1</td>
<td>LM1117-1.8 800 mA Linear Regulator⁴</td>
</tr>
<tr>
<td>Y2</td>
<td>1</td>
<td>20MHz Crystal</td>
</tr>
</tbody>
</table>

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 assignment 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
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 assignment 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

<table>
<thead>
<tr>
<th></th>
<th>5x5 mm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical package dimensions</td>
<td>5x5 mm</td>
</tr>
<tr>
<td>Maximum package thickness</td>
<td>1.5 mm</td>
</tr>
</tbody>
</table>

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 resonator (avoid parasitic 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 driver⁵: 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

⁵https://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):

```python
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.
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

```python
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

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

CFR1=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(7, 1)  # reset hi

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

pi.write(23, 1)  # IO_UPDATE

time.sleep(0.01)
pi.write(23, 0)
```