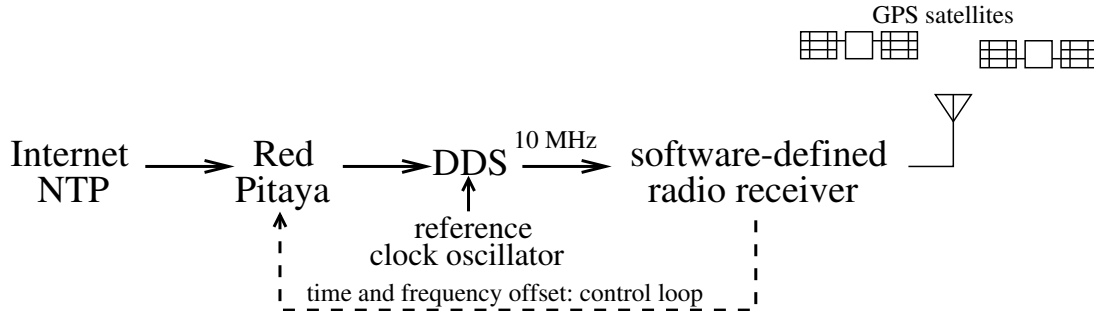


# M2 “embedded electronics” exam

J.-M. Friedt, January 15, 2024

We wish to connect a direct digital synthesizer (DDS) for generating a controlled, tunable frequency source, to clock a GPS global navigation satellite positioning receiver from a digital system. By driving the software defined radio GPS decoder with this flexible tunable source, the clock signal can be steered using a feedback loop to reproduce on the ground the stability of the spaceborne atomic clocks.

We furthermore wish this digital system to be time-synchronized to a common clock shared with other receivers over the Internet through a protocol named NTP (Network Time Protocol) as defined in RFC 5905. This protocol allows for any computer connected to the internet to be synchronized to within about 10 ms, or less than the GPS chip length of 20 ms, making sure all receivers are decoding the same GPS signal wherever they are located. We have selected the AD9834 DDS as meeting our requirements.



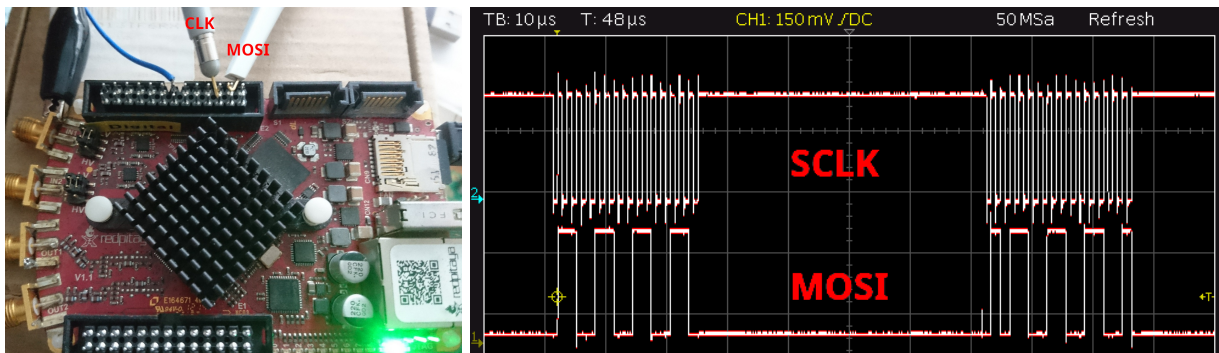
1. What is the benefit of controlling the DDS from a system running GNU/Linux – as opposed for example from a baremetal C programmed microcontroller – in the context of this project description? Justify.
2. We have selected the Red Pitaya as the digital platform for controlling the DDS. What pins of the Red Pitaya would you connect to the AD9834 for programming the DDS output frequency? Justify
3. The communication protocol between the Red Pitaya and the AD9834 leaves multiple degrees of freedom to be defined by the user. What are these tunable parameters and what are their settings in the case of the AD9834? Justify. Be careful not to blindly trust everything the datahsheet says, Analog Devices also writes mistakes !
4. Multiple software solutions are availabe for programming the communication signals between the digital system and the DDS. Cite two of them at least, with their benefits and drawbacks.
5. We have provided a kernel-space configuration file at <http://jmfriedt.free.fr/ad9834.dts>. How do you inform the kernel of this new configuration? Run the appropriate commands in order to configure the Red Pitaya kernel and check the success of the configuration of the new peripheral: how do you assess the successful addition of the DDS to the list of peripherals controllable by the Red Pitaya?
6. How can you access the AD9834 configuration? What kernel framework is used when adding this peripheral manufactured by Analog Devices?
7. What is the keyword in the configuration file we have provided that identified which driver is loaded. Identify the location of this driver source code and the associated keyword.
8. Demonstrate how you can generate the signal on the communication bus that would configure the DDS output frequency: what pins would you probe using an oscilloscope to check the communication signals? Configure the DDS to output a  $f = 10$  MHz sine wave: what are the communication signals and can you interpret their meaning to match the DDS configuration word? Remember that the frequency tuning word  $FTW$  configuring the output frequency is related to the master clock frequency  $f_{master}$  and the accumulator size of  $M$  bits as
$$FTW = \frac{f}{f_{master}} \times 2^M$$
9. With the proposed configuration, what is the frequency resolution when programming the clock driving the GPS software defined radio receiver?
10. Before selecting the AD9834, we had considered the AD9832 DDS as a viable option. Would you have been able to load the AD9832 driver using the same mechanism we used for loading the AD9834? Justify the answer based on the analysis of the source code of the AD9832 driver.

1. NTP synchronization is running over IP so we benefit from the TCP/IP stack from the Linux kernel.
2. The AD9834 is controlled using an SPI bus so we need to identify the PS controlled SPI pins. <https://redpitaya.readthedocs.io/en/latest/developerGuide/hardware/125-14/extent.html> informs that all signals are on connector E2, with MOSI is on pin 3, SCK on pin 5 and CS on pin 6. Ground is implicit on pin 12, 21, 22 or 25 and 26.
3. CPHA and CPOL must be adjusted according to the datasheet description. We learn at <https://www.analog.com/media/en/technical-documentation/data-sheets/ad9834.pdf> on Fig. 5 that SCLK is high when FSYNC falls to ground, meaning the rest state of the clock is high and CPOL=1. The high rest state is consistent with the statement on page 27 but contradicts their definition that CPOL=0. If the data is valid on the falling edge, then CPHA=0, or with their definition of CPOL=0 then the falling edge is CPHA=1. FSYNC holds the role of the Chip Select signal.
4. We can select userspace access to the raw SPI bus and implementing manually the communication protocol, and kernel space using the Analog Devices driver. Userspace avoids understanding driver configuration and makes SPI access easy though interpreted languages such as Python, but requires learning and implementing the communication protocol. Kernel space requires documenting the devicetree to load the driver but then all communication methods are provided by Analog Devices.
5. `dtc -@ -I dts -O dtb -o ad9834.dtbo ad9834.dts` will generate the devicetree overlay which is appended to the existing devicetree with `mkdir /sys/kernel/config/device-tree/overlays/ad9834` and `cat ad9834.dtbo > /sys/kernel/config/device-tree/overlays/ad9834/dtbo`. When running these commands, `lsmod` tells us that

```
ad9834                16384  0
gpio_zynq             20480  2
industrialio_triggered_buffer 16384  1 xilinx_xadc
kfifo_buf            16384  1 industrialio_triggered_buffer
industrialio         61440  4 ad9834,xilinx_xadc,industrialio_triggered_buffer,kfifo_buf
```

meaning that the `ad9834` driver was loaded and so were its dependencies of the Industrial IO framework.

6. IIO has created the device tree structures at `/sys/bus/iio/devices/iio:device1` where `cat /sys/bus/iio/devices/iio:device1/name` answers `ad9834` so the right peripheral was loaded.
7. the `compatible` keyword in the `dts` file must match the definition in the source code. The driver source codes are located at `drivers/staging/iio/frequency/ad9834.*` of the Buildroot Linux source code.
8. `while true; do echo "10000000" > out_altvoltage0_frequency0 ;sleep 1;done` repeatedly reprograms the DDS with the frequency 10 MHz and allows monitoring the signals on the oscilloscope. Since the devicetree file informed `clock-frequency = <25000000>`; then the frequency tuning word is `dec2hex(floor(10/25*2^28))` (using GNU/Octave) answering `6666666` which is indeed the set of signals observed on MOSI.



9.  $25 \cdot 10^6 / 2^{28} = 0.037$  Hz for each bit of the frequency tuning word. However since the kernel only handles integers and the frequency is provided as a value in Hz, the driver limits the resolution to 1 Hz.
10. We are unable to identify any Open Firmware of `.*` command in the AD9832 driver, so it is probably not devicetree compatible. The Analog Devices documentation at <https://wiki.analog.com/resources/tools-software/linux-drivers/iio-dds/ad9832> indeed does not include any devicetree documentation or example.