

# Embedded electronics: exam

January 9, 2023

## 1 AD9954 DDS driver

We wish to develop the software framework for controlling an Analog Devices Direct Digital Synthesizer (DDS) AD9954. Based on an initial inspiration from <https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/drivers/iio/dac/ad8801.c> we aim at implementing the communication protocol over a Serial Peripheral Interface (SPI) bus and implement some of the control register access interfaces. The AD9954 chip outputs a radiofrequency sine wave whose frequency is determined by the (fixed) system clock and a user-supplied frequency-tuning word (FTW).

First the connection of the AD9954 on the SPI bus is described thanks to the Device Tree and more precisely through an Overlay available at <http://jmfriedt.free.fr/ad9954.dts>

1. compile the devicetree binary overlay and add its functionality to the Red Pitaya kernel: where can you check in the GNU/Linux userspace filesystem that the overlay has been activated?
2. compile the kernel module available at <http://jmfriedt.free.fr/ad9954.c> using a Makefile you will provide, and insert the IIO-compatible kernel module providing the functionality needed to communicate with an AD9954 if it were connected to the SPI port whose pins are available on the E2 connector of the Red Pitaya. How can you check that the driver has been loaded? What message confirms that the driver is active?
3. What is the size (in bytes?) of an “int” variable for the Linux kernel? Modify the kernel module to display this value.
4. what mechanism/field in the devicetree and the driver relates them to each other, i.e. how was the driver activated by the devicetree overlay?
5. a new directory has been created giving access to the new peripheral registers: where is it located? How do you check that it has been indeed created by the devicetree overlay?
6. the IIO-compatible driver implements a voltage-type interface with as many channels as there are registers in the AD9954. Find the available interface types in the Linux kernel source code tree: where are the possible IIO interfaces (e.g. IIO\_VOLTAGE) defined and why did we not select a better suited interface type for the AD9954 radiofrequency synthesizer?
7. at the moment only a single byte can be sent to the AD9954 whereas most communication requires 3 to 6 byte communication (the register index followed by the argument): modify the kernel module to transfer 3 to 5 bytes depending on the register index. At the moment the number of bytes to be sent is given as the last argument of `ad9954_spi_write()`.
8. the DDS core clock is set to 400 MHz and the relation between the output frequency and the clock frequency is given page 34 of the datasheet: compute the Frequency Tuning Word (FTW0) for the DDS to output a 10 MHz sine wave: probe the MOSI and CLK signals of the SPI bus and demonstrate the ability to output the right signals to the DDS.
9. What is the behaviour of the Chip Select pin associated with each communication sequence? How does it compare with your expectation?
10. What is the fastest rate at which the frequency output of the AD9954 can be reprogrammed from userspace by communicating the new setting to the chip through the SPI bus? Justify.
11. What is the base address of the SPI interface accessible through the AXI bus of the ARM processors of the Zynq processor on the Red Pitaya? How did you find the information?

Answers:

1. On the PC:

```
$BR/output/host/usr/bin/dtc -@ -I dts -o ad9954.dtbo ad9954.dts
```

and on the Red Pitaya:

```
mkdir /sys/kernel/config/device-tree/overlays/ad9954
cat ad9954.dtbo > /sys/kernel/config/device-tree/overlays/ad9954/dtbo
```

2. the module is compiled on the PC using the following Makefile

```
PATH:=$(PATH):$BR/output/host/usr/bin/

obj-m += ad9954.o
all:
make ARCH=arm CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf- -C \
$BR/output/build/linux-xilinx-v2019.1/ M=$(PWD) modules
```

and the resulting kernel object is transferred to the Red Pitaya for insertion

```
insmod ad9954.ko
```

resulting in

```
ad9954: loading out-of-tree module taints kernel.
ad9954 spi0.0: Linked as a consumer to regulator.2
```

3. We can check the size of an integer by adding a `printk("%d", sizeof(int));` in the `probe` method of the kernel module which displays 4 in the `dmesg` output
4. the “compatible” entry of the devicetree overlay must match the “name” of the kernel module and its “id” when using the openfirmware devicetree framework.
5. the IIO directory providing the pseudo-files for access the hardware is at

```
/sys/bus/iio/devices/iio\:device1/
```

6. the following command on the PC

```
grep -r IIO_V0 $BR/output/build/linux-headers-4.19.215/include/*
```

tells us that most probably the IIO peripherals are defined in

```
$BR/output/build/linux-headers-4.19.215/include/uapi/linux/iio/types.h: IIO_VOLTAGE,
and indeed we see in this header file that there is no “FREQUENCY” peripheral on any closely
matching description of a DDS.
```

7. the driver is updated with

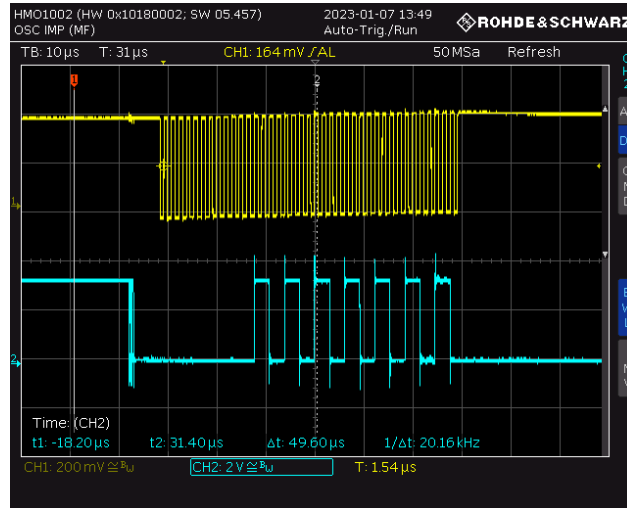
```
static int ad9954_spi_write(struct ad9954_state *state, u8 reg, unsigned int value, int n)
{unsigned char data[5];
  data[0] = reg;
  *(int*)&data[1] = (cpu_to_be32(value<<((5-n)*8)));
  return spi_write(state->spi, data, n);
}
```

so that all bytes of the 32-bit integer word are transferred through the SPI bus and not only a single byte.

8. The frequency tuning work  $\frac{10}{400} \times 2^{32}$  is 107374182 or 0x6666666. By sending this value on the SPI bus with

```
while true; do echo 107374182 > /sys/bus/iio/devices/iio\:device1/out_voltage0_raw ;done
```

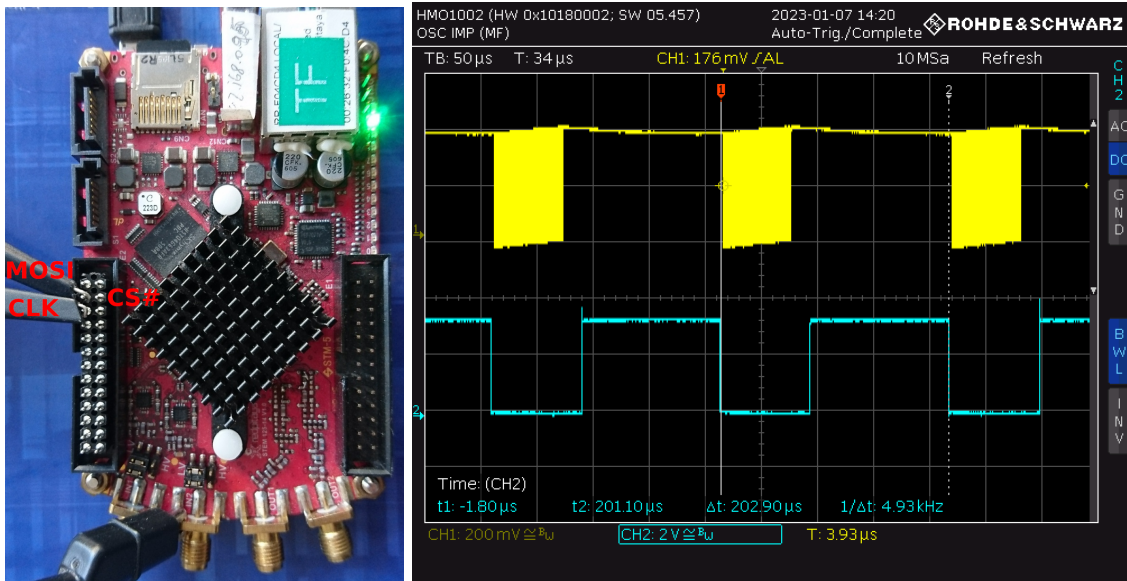
we observe the following oscilloscope traces



with the first byte defining the register index (0x00) and the 4 next bytes the FTW0 value (0x6666666).

Actually the register holding FTW0 is indexed 4 so the command should be `echo 107374182 > /sys/bus/iio/devi`

9. as expected, the CS# (blue) drops during each communication (yellow)



10. by repeating the measurement in a while loop and removing all `printk()` displays in the module, the fastest rate is about 4.9 kHz

11. decompiling the devicetree binary found in the first partition of the Red Pitaya shows that

```
$BR/output/host/usr/bin/dtc -I dtb -O dts zynq-red_pitaya.dtb | grep spi1 | tail -1
spi1 = "/amba/spi@e0007000";
```

the base address is 0xe0007000