# Exam: embedded electronics (M1, semester 1)

J.-M Friedt, December 13, 2023

2 points/correct answer, negative points for erroneous answers copied from search engines or chatbots.

We are interested in understanding the use of the Analog Devices AD7293 [1] which includes Analog to Digital (ADC), Digital to Analog (DAC) and temperature sensing capability.

1. Considering a reference voltage of 1.25 V, what is the smallest voltage variation this chip can measure on its ADC inputs? Justify.

   12-bit ADC with values ranging from 0 to 4095 so 1.25 V/4096=300 $\mu$V

2. In order to reduce the noise on the measurement, we wish to accumulate 10 successive ADC measurements. What is the most appropriate variable type to store the successive values of the sum during the accumulation? Justifiy. How do you define such a variable in the C language?

   The sum is between 0 and 40950 so an **unsigned** short is needed since a (signed) short would overflow and a long would be inefficient

3. The user is not interested in a binary value resulting from the accumulation of the 10 successive acquisitions but wishes to be given a voltage (in V). How do you best convert the result of the acquisition of the previous question into a voltage displayed with 1 decimal accuracy (i.e. XY.Z V with Z a meaningful fraction of a volt)? Same question for 1 mV accuracy.

   The ADC word `w` is converted to voltage $V$ with $V = \frac{1.25}{4096} \times$ `w`. In order to keep one relevant decimal when computing with integers, we must compute $10V = \frac{12.5}{4096} \times$ `w`. Since the ADC measurement is encoded on 12 bits, then 10 accumulations is encoded on $12 + \log_2(10) = 16$ bits and multiplying the numerator with 12.5 requires an intermediate 32-bit variable. Since `w` is the accumulation of 10 measurements, the averaged voltage would be $w/10$ and keeping one relevant decimal requires computing $10V$ so the result is $10V = w \times 1.25/4096$ and the fractional part of the numerator is irrelevant so we are left with $w/4096$. For a 1 mV accuracy we must compute using integers $1000 \times V = (1250w)/4096$ with the numerator fitting again in a 32-bit variabe and $(625w)/2048 = (5^4 w)/2^{11}$ cannot be reduced further.

4. We have recorded a temperature measurement as a binary value resulting from reading the analog to digital converter and wish to display a temperature in °C. How do you convert the binary value to °C based on the information in the datasheet? How do you implement this calculation as a C function on a STM32F1 (Cortex M3 core) microcontroller?

   0.125 °C/bit so the word `w` measured on the ADC is converted to degrees by multiplying with 0.125 which is not efficient on a Cortex M3 but multiplying by 0.125 is also dividing by 8 but divisions are also inefficient. A division by 8 is a 3-fold binary shift so `w>>3` will efficiently provide the temperature in °C.

5. Table 27 of the datasheet describes the register organization for driving the voltage output of a DAC. Considering a variable `s` holding the binary sequence representing the voltage to be generated (what is the most appropriate storage type for `s`?), how to you fill the content of register located at address 0x30 with the content of `s` so that the output of the DAC is properly configured?

   The 4 least significant bits are used for data management purposes and do not reprensent the DAC output which is only held in the 12 most significant bits, so `s << 4` will shift `s` to the right bits of the register.

6. on which edge of the SPI bus is the DAC programmed? What is the rest state of the SPI clock? What are the resulting two configurations needed to properly define the SPI settings?

   - page 29 input register on the rising edge of the SCLK.

[1] `https://www.analog.com/en/products/ad7293.html`

- page 73: SCLK = low when CS drops
- CPOL=0, CPHA=0 or mode 0

7. what is the polarity of the Chip Select pin to activate the AD7293 during an SPI transaction?

   Active low

8. How do you check the state of bit $I_{sense}3$ *low* in regiser 0x15 as shown in Table 93? Provide an `if ... else ...` set of instructions in C language for calling `action1()` if this bit is set and `action2()` if cleared.

   Shift and mask to isolate the relevant bit:
   ```
   if (((((*unsigned short*)0x15)>>3)&0x01)==0) action2(); else action1();
   ```

9. We provide a set of programs at `https://github.com/jmfriedt/exam`. Demonstrate how you can compile and execute this program for the appropriate target: which is this target? Analyze the result of the execution: what do you observe? Is this the expected behaviour? If you see something abnormal, describe in detail.

   The `Makefile` rules are executed by using `make`. Since `gcc` is used, the target is the Intel x86 PC. The resulting executable is run with `./f` with the definition of the executable location in the current directory (".").

   The abnormal behaviour is that we wish to display the content of `message2` defined as "Hello" and yet we see "Hello world" displayed.

10. The author of this code is unskilled and left many warnings: update the source codes to remove all warnings during compilation. You might have to create an additional file.

    `#include <stdio.h>` and create `function2.h` included in `function1.c` with the prototype of `int my_display(char *);`

11. Dump the memory content of the 20 bytes starting at the memory location of `message2`: how do you achieve this result (provide the few lines of C code) and what is the result?

    `int k;for (k=0;k<20;k++) printf("%02hhx ",*(char*)(message2+k));` displays
    `48 65 6c 6c 6f 20 77 6f 72 6c 64 20 0c 00 00 00 01 00 00 00`
    clearly showing how "Hello " (ASCII 48 65 6c 6c 6f 20) is followed in memory by "World " (ASCII 77 6f 72 6c 64 20) and then the trailing 0s will stop the display as a side-effect.

12. If you found something awkward in the behaviour of the program, correct to make the program behave as expected. The output of the previous question might help. What is the expected behaviour and how do you correct the program to reach this result?

    A string (usage of `puts()` function) is defined in C as a char array terminated by the value 0. Here we "forgot" to terminate `message2` with `\0` so that the display function starts reading the content of `message2`, leaks into `message1` located in memory after `message2`, and stops displaying the memory content when it reaches 0 values at the end of message1. Had we defined more variables, the display would have continued until the first 0-byte value is reached, a dangerous side effect that would often lead to random crashes. The solution is to add `\0` at the end of each string definition.

13. Add a `Makefile` rule to remove all files generated during compilation and return the repository to a clean state.

    a rule to clean the directory:
    clean:
    	rm *.o f