Inexpensive RLC meter

Jesus Arias-Alvarez

Dpto. E. y Electrónica, E.T.S.I. Telecomunicación. 47011 Valladolid. Spain

1 Introduction

This design started some time ago as a metal detector. In that case the reactance on the sensing inductor had to be measured along with its resistance. Both components are of interest because different metals can alter the impedance of the coil in different ways. For instance iron increases the inductance a little, but the increase in the resistance is higher. Non magnetic metals, like copper, decreases the inductance but increases the resistance. This last component also depends on the conductivity of the metal: lead increases the resistance more than aluminium. Thus, a circuit to measure the two components of a complex impedance is required, and that is, basically, an RLC meter.

One common way to measure an impedance is to put a constant AC current through it and to measure the voltage drop across its terminals. This voltage is proportional to the impedance, with the in-phase part being the resistance and the in-quadrature part the reactance. That was the way I followed for the first prototype, and soon I found it easier to implement in the mind than on a board. It required operational amplifiers with a lot of bandwidth, precision resistors, and even cascode transistors to avoid building a parasitic oscillator. There must be a simpler way to measure a complex impedance. In this design a simple voltage divider between a known resistor and the measured impedance is used instead of a current source. The output voltage still depends on the impedance but their relationship is no longer linear. This isn't a big problem: just let a microcontroller to do the math.

Also, the AC signal has to be a pure sine wave because the reactance depends on the frequency and, therefore, if harmonics are present the measured amplitude will include an error component.

The measuring process will include several steps, like sine-wave generation, amplification, mixing, filtering, digitization, and complex variable math. In this design the AC signal is digitized before the mixing, so, the microcontroller has to deal with an intensive signal processing. This, on the other hand, has the advantage of getting rid of a lot of analog components with all their non-idealities: there are no phase errors with digitally generated sine waves, nor channel imbalances in the digital mixer, digital integrators no not leak, etc. There are still some required analog parts, like a DAC for the generation of a sampled sine-wave, a reconstruction filter to remove the harmonics due to the sampling, some amplifiers to increase the measured voltage when the impedance is low, and an ADC with an input multiplexer to digitize the resulting signals. Today many microcontrollers include an ADC inside the chip, so this critical part comes at no additional cost. A DAC is not so common, but a simple DAC can be built using resistors, again a very cheap part. The remaining analog blocks are now the filter and the amplifiers.

A key trick used in this design is to have a sampling rate that is exactly 4 times the frequency of the AC signal. This eases a lot the signal generation and processing because the sample sequence of a such sine-wave is just 0, 1, 0, -1, and so on. Therefore, the DAC only has to generate three different levels, and this can be accomplished with just two identical resistors. Also, the mixing doesn't require any multiplication in the

microcontroller, it is merely the addition or subtraction of sample values, something any CPU is quite capable to do. A fixed frequency of 50kHz was chosen for the AC signal because a selectable frequency would also imply to have a tunable reconstruction filter. With this frequency the sampling rate of the ADC has to be 200kS/s, or said with another words: in addition to have an appropriate ADC, all the mixing and integration has to be done in less than 5μs for each sample.

The frequency chosen for the test signal, along with the voltage divider resistance, Z_0, set the measuring range of the meter. Z_0 was chosen as 120Ω from peak current considerations. From a practical point of view the meter stops to take accurate measurements when the output voltage of the divider is too low or too close to the input amplitude. Setting an upper and lower bounds of 1/32 of the full scale range of the ADC will result in a relative quantization error about $\pm 1.6\%$ and a measurement range of 3.84Ω to 3720Ω, but low voltages can be amplified up to $\times 121$, so the minimum measurable resistance is 32mΩ. When measuring reactances, the respectives ranges are 100nH to 11.8mH for inductors and 850pF to 100μF for capacitors.

The RLC meter was designed to be portable and powered with batteries. Instead of using a sophisticated power regulation electronics in order to have several accurate supply voltages (maybe, with some of them negative), everything is powered directly by two AA cells. This implies a variable voltage supply that decreases as the cells are drained. The voltage can range from 3.1V when batteries are new to about 2.2V when they are almost exhausted. The microcontroller, operational amplifiers, and everything else must be able to operate within this range of voltage.

Finally, as a human interface, a typical alphanumeric LCD display was considered. The problem with this kind of display is the high voltage required for the biasing of the LCD, around 5V, but the current consumption is low (about 0.3mA). Therefore, a charge pump was designed with capacitors and diodes to generate a negative voltage for its V_{EE} pin (The LCD bias is $(V_{DD} - V_{EE})$), while the logic of the display is powered directly with the 3V of the batteries. The LCD bias voltage also has to be adjustable in order to get the best contrast in the display. A reflective LCD model was chosen in order to avoid the current consumption of a backlight.

2 The circuit

The simplified block diagram of the RLC meter is shown in figure 1. The microcontroller is a cheap ARM Cortex-M0 made by NXP, the LPC1112. This microcontroller runs with a 48MHz clock generated from a reference crystal oscillator of 12MHz (It can also run from an internal oscillator, but it isn't very accurate). It includes a 10-bit ADC capable of more than 300kS/s, along with several timers, 4kB of internal RAM and 16kB of internal Flash. An interesting aspect of this microcontroller is its low current consumption, less than 10mA, which is very desirable for a portable system. Its package only includes 20 pins, making it easy to solder, yet, it is a bit short of I/O pins and as a consequence some of them are used for more than a single purpose in this design.

Along with the microcontroller comes an EEPROM memory with an I2C bus intended for the storage of calibration data. The internal Flash of the microcontroller could also be used for this but it only allows the erasing of 4kB blocks, thus resulting in a considerable waste of memory that is also much needed for program, specially when floating point routines have to be included into the code. Almost any EEPROM could be used, a 128 byte model is more than enough. The only consideration comes from the software point of view: an EEPROM with more than 256 bytes in a single bank will require two bytes for the address instead on one.

For the generation of the sine-wave two pins with a matching to a timer function are used. Both pins toggle each 10μs, but one toggles 5μs before the other, resulting in two square waves, 90º of phase apart. Connecting identical resistors between each pin and a common output node we have the desired 3-level DAC. The same pins are also driving the charge pump for the generation of the negative voltage needed by the LCD. Notice that the output amplitude of the DAC doesn't require to be very accurate because the actual amplitude is going to

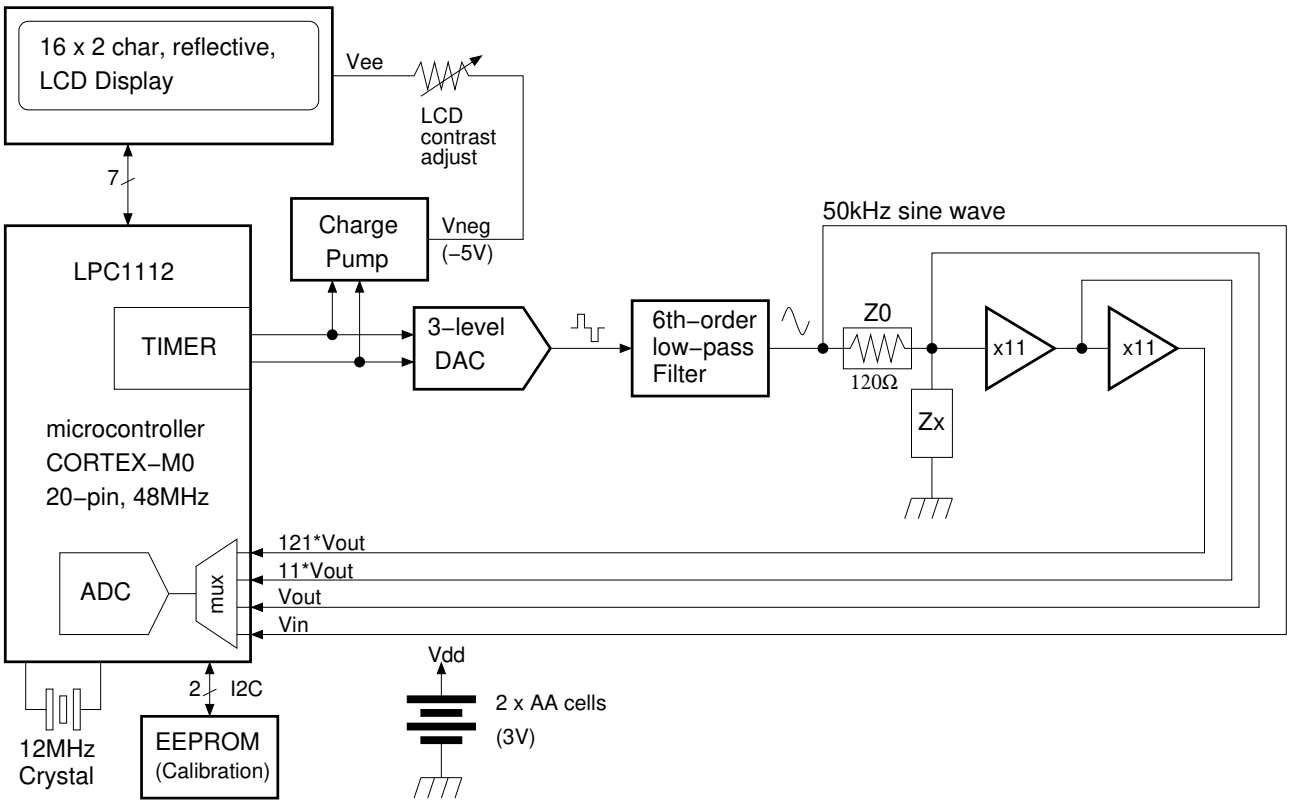


Figure 1: Block diagram of the RLC meter

be measured by the ADC.

After the DAC comes a filter block, the voltage divider, and some amplification. Three different gains can be selected for the output signal: $\times 1$, $\times 11$, and $\times 121$. The microcontroller must chose the output which has the higher amplitude, but without clipping. Only two amplification stages are used.

A more detailed description of the circuit follows.

2.1 Main sheet

In figure 2 the master sheet of the schematic shows its main component, the LPC1112 microcontroller, along with ancillary circuits, like the crystal oscillator or the in-system-programming connector, J1. The EEPROM for calibration data is U2, and the alphanumeric display is LCD1. The display uses a 4-bit data bus in order to reduce the required number of microcontroller pins. Even with this, the 4 data bits of the LCD bus have also some other functions. Three of these signals can be connected to ground through the series resistors R7, R8, or R9, if a short is placed in connector J6, which orders the processor to enter a calibration procedure. The bit D7 is connected to ground through a forward biased diode, and allows us to get a coarse estimation of the supply voltage (D7 is connected to PIO0_11, which can also be programmed as an analog input, AD0).

R2 and R3 forms the 3-level DAC. Its output resistance is the parallel equivalent of these resistors, namely $2.8k\Omega$ (This equivalent resistance is also part of the first stage of the filter). The output of this DAC is connected to another schematic block, ANALOGBLOCK, that also includes 4 outputs that are connected to the analog inputs AD1 to AD4 of the microcontroller.

The charge pump is built with capacitors C1, C2, and C3, and dual Schottky diodes D1 and D2 (only 3 diodes are actually used). Its output is connected to the V_{EE} pin of the display through the variable resistor RV1. Internally, the display has a $16.5k\Omega$ resistor between V_{DD} and V_{EE}. Therefore, the variable resistor RV1 allows us to apply a variable negative voltage to V_{EE} and to adjust the LCD contrast to its optimum value.

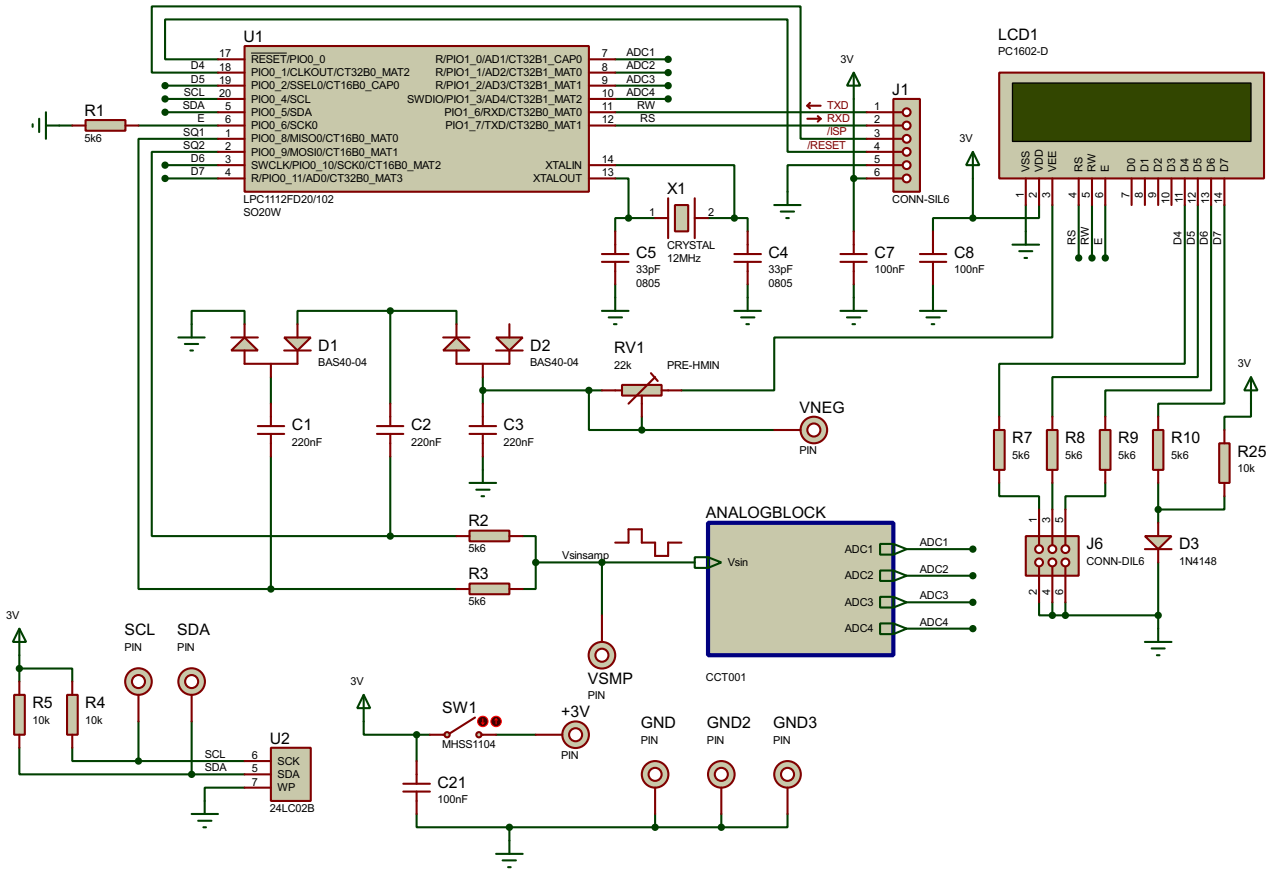


Figure 2: Main sheet of the schematic: microcontroller, display, DAC, charge pump, EEPROM.

2.2 Analog block

The analog block schematic is shown in figure 3. Here we see another two sub-blocks, FILTER, and AMPLIF, so the main purpose of this schematic sheet is to show how the voltage divider is actually implemented. First, a fixed resistor, R6, is connected in series with the output of the filter. This resistor is the known impedance of the divider, with Z_X being the impedance under test. R6 also limits the current that is applied to Z_X to about 10mA. This is important because this current has to be provided by an operational amplifier in the filter and we don't want to exceed its specs if a very low impedance is measured.

The output of the filter has a DC voltage about $V_{DD}/2$. Because of this the second terminal of Z isn't connected directly to ground, it is connected to a big capacitor, C6, instead. The resistor R11 is included to force the discharge of C6 if Z is an electrolytic capacitor. These capacitors will require attention to polarity, and they could be wrongly polarized during the negative semicycles of the test signal if R11 is omitted and a resistor or inductor was tested previously, leaving C6 charged.

The last thing we want to remark is the 4-lead connection for the impedance under test. This can make a difference when very low impedances (about 1Ω, or less) are measured because the impedance of the meter leads can be comparable to the impedance under test. Low impedances will require voltage amplification and, therefore, the amplifier block that follows the voltage divider must have a differential input. For high value impedances the error due to wires is negligible and a single-ended signal, ADC2, is taken directly from the divider and routed to the AD2 input of the microcontroller.

2.3 Filter

The reconstruction filter schematic is shown in figure 4. Here a 3-stage Sallen-Key active filter is built around 3 operational amplifiers, resulting in a 6^{th}-order filter, more or less of the Butterworth type. It has a cut-off

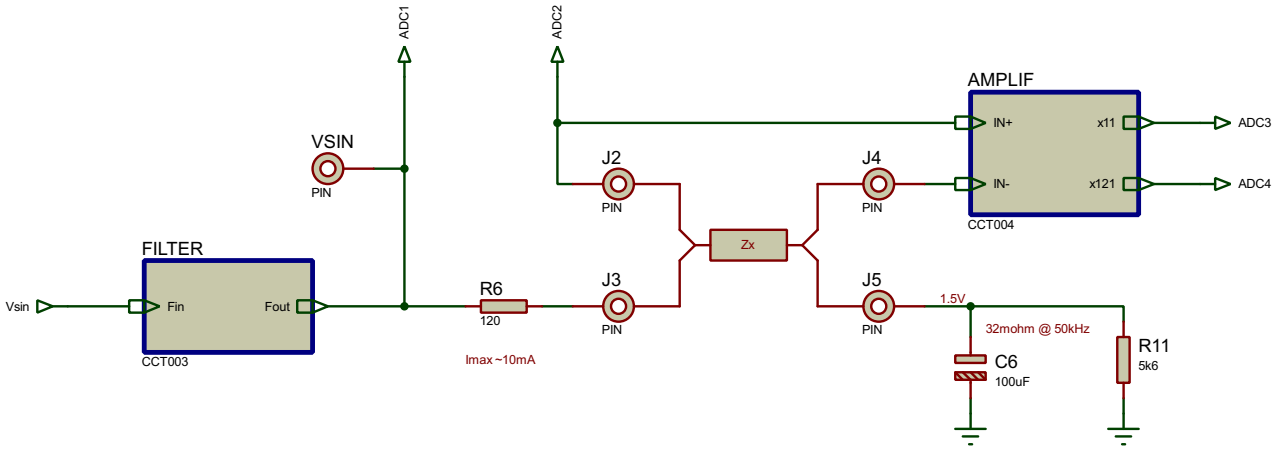


Figure 3: Analog block of the meter, showing the four-lead connection of the impedance under test.

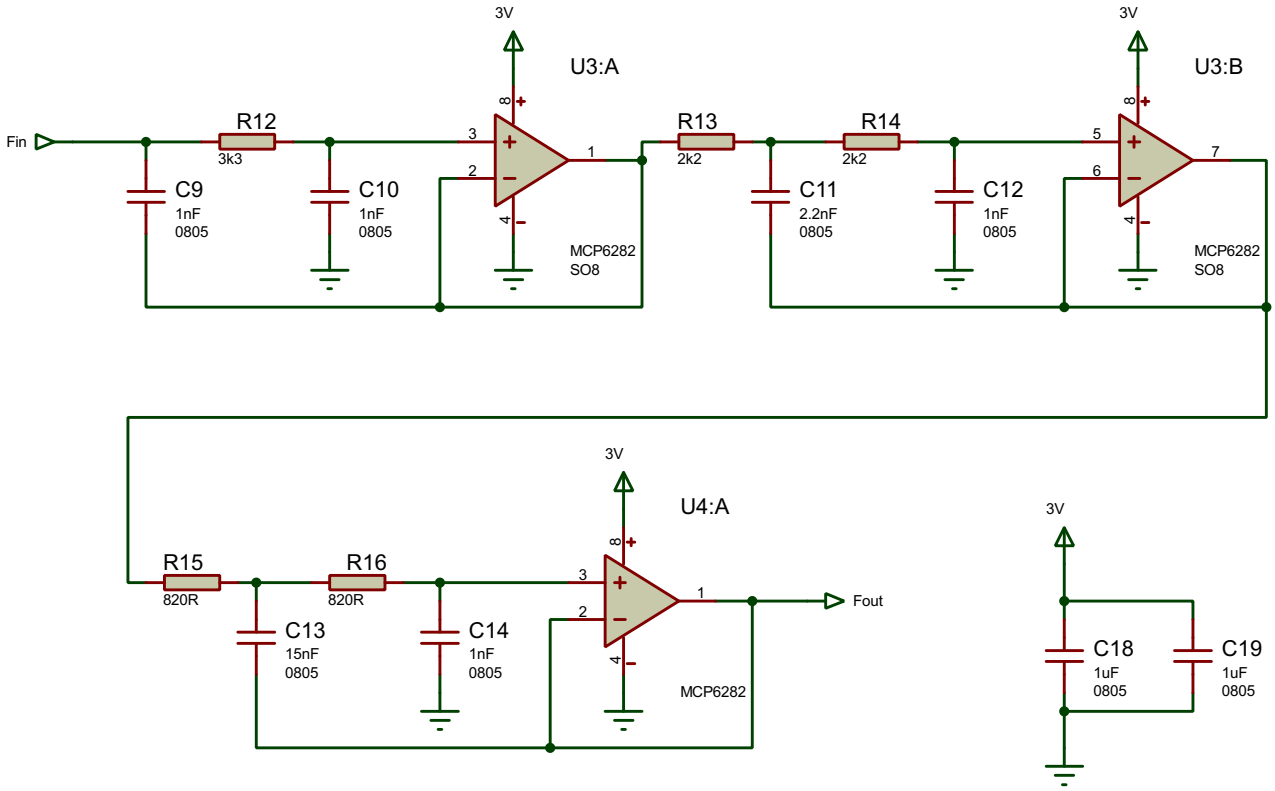


Figure 4: Schematic of the 6th-order, low-pass, active filter.

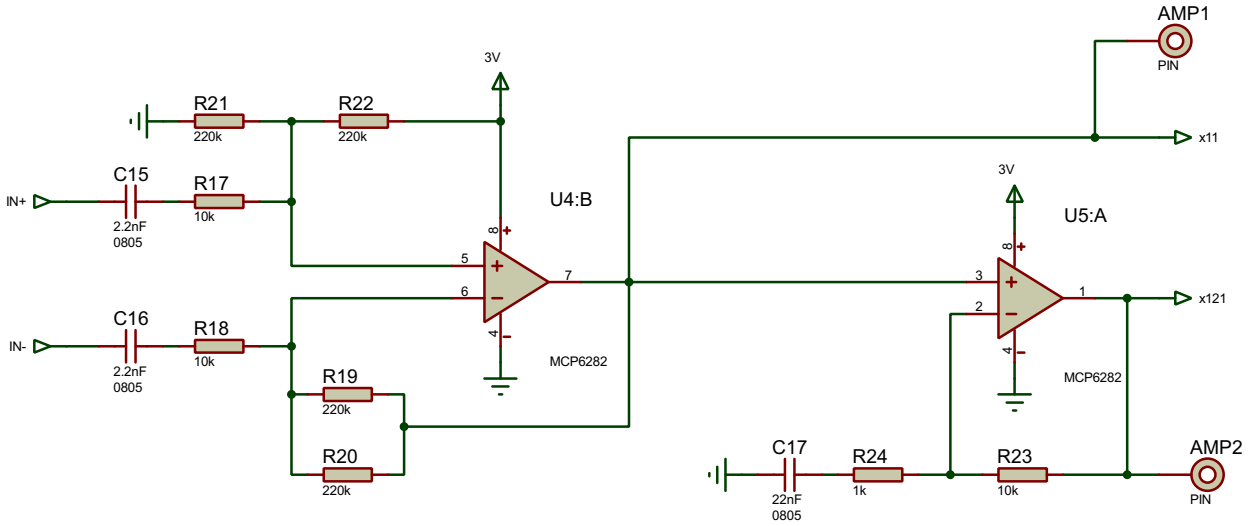


Figure 5: Schematic of the two amplifier stages

frequency of 50kHz and an out-of-band attenuation of -120dB/dec. Our objective here is to attenuate the third harmonic of the sampled sine-wave (no even harmonics are present in this signal) at least 50dB in order to reduce its amplitude below 1LSB of the ADC. A rough estimation of the attenuation is the ratio of frequencies to the power of the order of the filter, that in this case is $3^6 = 729$ or 57dB.

The filter has an unity gain in its pass-band because all operational amplifiers are connected as followers. The frequency response is thus controlled by the ratio of the capacitors in each stage. In the first stage the input resistor isn't included because it is the equivalent output resistance of the DAC.

The operational amplifier chosen for this circuit is the MCP6282, a cheap, dual rail-to-rail opamp with 5MHz gain-bandwidth product, FET inputs, and capable to operate with only 2.2V on its power supply. The only ugly thing about this opamp is its high offset voltage at the input, but for this circuit this specification isn't critical because the DC gain is only 1.

I want to remark that a previous prototype of the filter was built using LM741 opamps, just because we have lots of them in stock for teaching, and its performance was a disaster, with a total harmonic distortion about -33dBc, and a quite high second harmonic generated by the distortion of the LM741. On the other hand, the MCP6282 proved to be much more linear and the harmonic content of the output sine wave was negligible.

2.4 Amplifiers

The last part of the schematic of the RLC meter that we have to discuss is the amplifier. The corresponding schematic is shown in figure 5. It consists of a first stage that operates as a differential amplifier, and a second stage that is simply a non-inverting amplifier. Both stages have the same gain: $\times 11$, or 20.8dB. DC decoupling capacitors, C15, C16, and C17 are included with also a less clear function: Their values were chosen to make the phase difference at the output of each stage as close to zero as possible, taking into account the bandwidth of the opamps. Yet, this isn't really needed because any phase difference in the amplifier is going to be corrected thanks to calibration, but to have a proper value for capacitors doesn't hurt either.

3 PCB layout

The circuit layout was fit into a 80×70 mm^2, double side, PCB. The two layers are shown in figure 6. Most of the components are installed in the top side of the PCB, with only 4 devices on the bottom side. These are: the LCD display, the holder for the two AA cells, the variable resistor, RV1, for the contrast adjustment of the

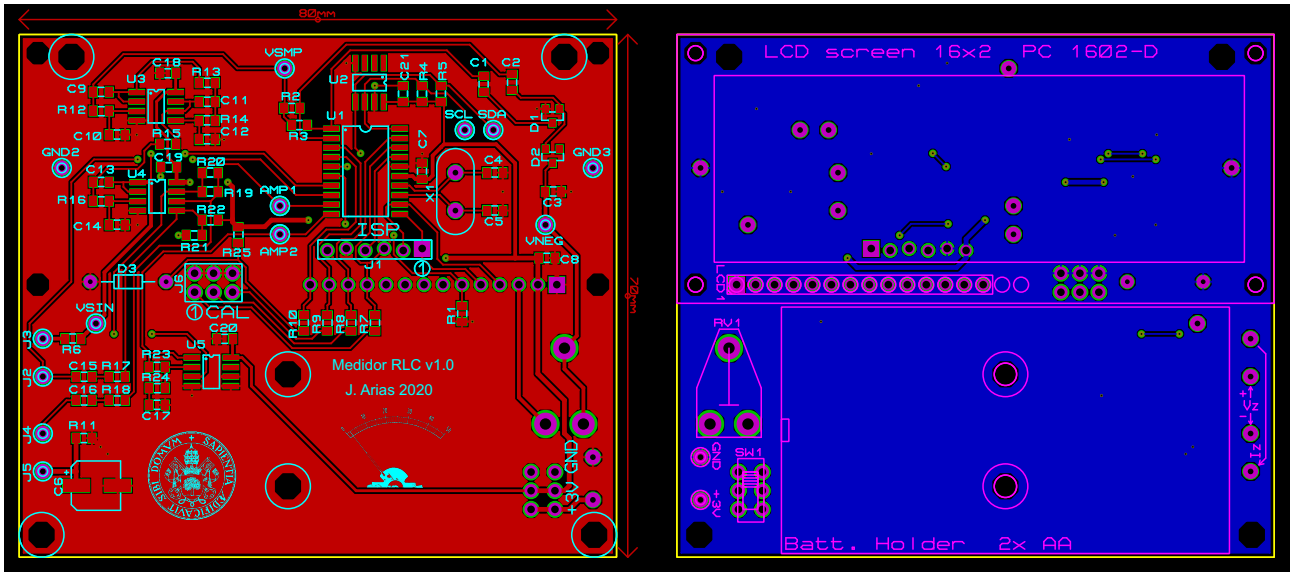


Figure 6: Top (red) and bottom (blue) sides of the PCB.

display, and the on-off switch, SW1.

The placing and routing was all done manually. The bottom layer was intended to be a ground plane and only a few short traces with other circuit nodes are present here.

4 Software

The firmware of the microcontroller has to cover a wide range of functions. First, there should be an initialization code with its reset and interrupt vector table, the setting of the initial values for the .data and .bss sections, and the setup of the peripherals to be used. This last part first includes the selection of the crystal oscillator and the setup of the PLL for the generation of the 48MHz clock. Then the function and direction of each microcontroller pin have to be selected.

Two timers are used. The 16-bit timer TMR16B0 is responsible for the generation of the 50kHz test signal. In order to do this, the MAT outputs CT16B0_MAT0 and CT16B0_MAT1 are programmed to toggle on the matching of the timer with their corresponding register values. MAT0 toggles when the counter reach the value 480 (10μs) and this matching also reset the counter to 0 (actually the counter counts from 0 to 479, so MAT registers are decremented by one). MAT1 toggles when the counter value is 240. In this way two 50kHz square waves with a 90º phase shift are generated. The CPU has to do nothing to generate these signals after the initialization of the timer.

The 32-bit timer TMR32B0 is also used. A 200kHz square wave is programmed to be generated in its MAT0 output in the same way as we did with the 16-bit timer. But in this case the MAT0 signal isn't available on any pin, it is used internally for the triggering of ADC conversions.

The ADC also has to be initialized and its interrupt enabled. The start of conversion is selected to be triggered by the CT32B0_MAT0 rising edges, and a 4MHz ADC clock is used. One ADC conversion takes 11 ADC clock cycles, or 2.75μs. In the vector table, entry number 40, the address of routine "ADC_IRQ" is also stored, and the bit 24 of the Interrupt Set Enable Register, ISER, is set, enabling the calling of the "ADC_IRQ" routine every time the ADC completes a conversion.

In addition to the internal peripherals, the LCD display also requires an initialization, along with a set of functions for its use that we can call a "driver". The same applies to the external EEPROM, that requires a driver for the I2C bus, where an internal hardware I2C controller is used, and for the memory itself.

After the initialization, the reading of calibration values from the EEPROM follows, and finally, the main

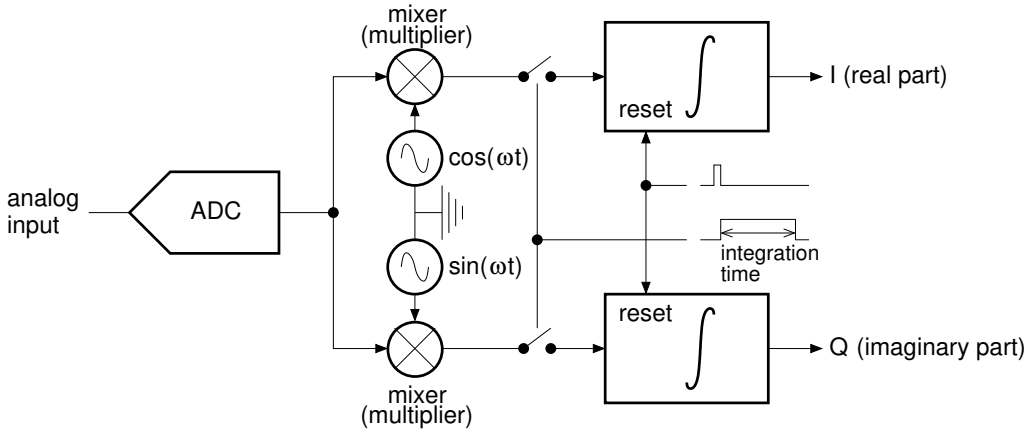


Figure 7: Equivalent block diagram of the signal processing.

loop of the code is reached. In addition to the main program loop, the ADC interrupt routine is performing a time-critical digital signal processing task that is detailed next.

4.1 Data acquisition, mixing, and integration

The signal processing is that of a synchronous detector whose block diagram is shown in figure 7. After the initial digitization in the ADC, everything is done by software inside the microcontroller. Each ADC sample is processed in the interrupt routine. This includes the mixing of the input signal with a cosine and a sine waves, that is merely the multiplication by the sequence of samples of the cosine and sine, that are:

Cosine	Sine
1	0
0	1
-1	0
0	-1
...	...

The result of the cosine product is accumulated (integrated) into the In-phase component of the signal, whereas the sine product is integrated into the In-quadrature component. A minimum of 4 samples (one signal cycle) have to be integrated, but we can integrate a lot more in order to reduce the effect of noise. But, first, by looking at the cosine and sine values, we can obtain a simple guideline for the code of the mixing and integration, that is:

- Read the sample value from the ADC.
- Look at the 2 low bits of a sample counter. Depending on their value do:
  - case 00: Add sample to the In-phase integrator variable.
  - case 01: Add sample to the In-quadrature integrator variable.
  - case 10: Subtract sample to the In-phase integrator variable.
  - case 11: Subtract sample to the In-quadrature integrator variable.
- After a lot of samples (integration time):
  - Copy the in-phase and in-quadrature values to their final destination
  - Reset the in-phase and in-quadrature values to 0.

The actual code is a little different because in our case the sample counter counts down, but the basic idea is the same. The integration time was chosen to be 20ms (4000 samples), or a multiple of this value, because I live in Europe and the mains frequency here is 50Hz (for America its better to integrate a multiple of 3332 samples). Choosing an integration time that is an integer number of cycles of the mains frequency we can lower the effect of the mains noise in the measurement. Samples are 10-bit long, and integrator variables are 32-bits wide. This means we can add up to 2^{22}, or roughly 4 million, samples before overflowing our integrators.

The IRQ routine is called every 5μs, or 240 CPU cycles. Time is tight, and in order to not waste any cycle, the code of the IRQ routine was written in assembly language, yet, still embedded into C code. This code is relocated to RAM during startup because the execution from flash is slowed down due to wait states (A read from flash takes 3 cycles, but 8 16-bit words are read, so, for sequential code execution one instruction every 8 takes two extra cycles to execute, but this has to be considered a best case scenario, non sequential code can have a longer time execution penalty that also depends on the code alignment to flash addresses). The trick here is to assign to the interrupt function the attribute `section(".data")`, a section typically used for static variables with non zero initial values whose content is copied from flash to RAM during startup. After all these optimizations for speed, this task takes about 30% of the CPU time.

The listing of the Interrupt routine and its variables follows:

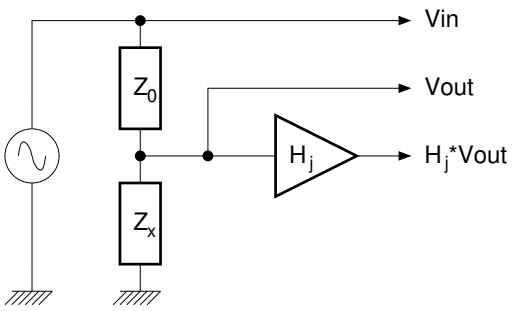
```
typedef unsigned int u32;
typedef signed int   s32;

////////////////////////////////////////////
//            INTERRUPT ROUTINES
////////////////////////////////////////////
volatile struct { // ISR Variables
    s32 tI;       // In-phase integrator
    s32 tQ;       // In-quadrature integrator
    u32 cnt;      // IRQ counter
    s32 I;        // In-phase result
    s32 Q;        // In-quadrature result
    u32 nresult;  // result counter
} demod;

////////////////////////////////////////////////////////////////////////////////
// ADC conversion done IRQ Service Routine
//   This ISR is called every 240 cycles (5us), and it last 76 cycles (worst
//   case), including 12 cycles for calling and 12 cycles for return. When
//   executed from flash times are longer. Thus, this code is copied to RAM
//   (section ".data") and executed there.
void __attribute__((naked,section(".data"))) ADC_IRQ(void)
{
    asm volatile (
"    push  {r4-r7}       \n" // R0-R3, R12, and LR, are already saved
"    ldr   r0,=ADCBAS+4  \n" // AD0GDR
"    ldr   r0,[r0]       \n" // R0: ADC result
"    lsl   r0,r0,#16     \n" // remove left bits
"    lsr   r0,r0,#22     \n" // remove right bits (0 <= sample <= 1023)
"    ldr   r7,=demod     \n" // Read data from "demod" struct:
"    ldmia r7!,{r1-r6}   \n" //   R1:tI, R2:tQ, R3:cnt R4:I, R5:Q, R6:nresult
"    lsr   r7,r3,#2      \n" // CY=cnt.1: 1: ADD, 0: SUB
"    bcs   1f            \n"
"    neg   r0,r0         \n" // change sample sign (SUB)
"1:  lsr   r7,r3,#1      \n" // CY=cnt.0: 1: add to tI, 0: add to tQ
"    bcc   2f            \n"
"    add   r2,r2,r0      \n"
"    b     3f            \n"
"2:  add   r1,r1,r0      \n"
"3:  sub   r3,#1         \n" // cnt--
"    bpl   4f            \n" // if (cnt<0)
"    mov   r4,r1         \n" //   I=tI
"    mov   r5,r2         \n" //   Q=tQ
"    mov   r1,#0         \n" //   tI=tQ=0;
"    mov   r2,#0         \n"
"    ldr   r3,=15999     \n" // 16000 samples/integration (4 50Hz cycles)
//"  ldr   r3,=7999      \n" //  8000 samples/integration (2 50Hz cycles)
//"  ldr   r3,=3999      \n" //  4000 samples/integración (1 50Hz cycle)
"    add   r6,#1         \n" // nresult++
"4:  ldr   r7,=demod     \n" // Store data to "demod" struct
"    stmia r7!,{r1-r6}   \n"
"    pop   {r4-r7}       \n" // restore registers
"    bx    lr            \n" // Return. R0-R3, R12, and LR, are also restored
"    .ltorg              \n" // local space for literals
    );
}
```

4.2 Impedance calculation and calibration

The interrupt routine processes the ADC samples but it doesn't select the source of the analog signal. In the main program we can chose the desired sine wave to measure by selecting the proper ADC input to digitize, and then wait until the "`demod.nresult` " variable changes. The in-phase and in-quadrature components of the result are assigned to the real and imaginary part of a complex variable, each component as a floating-point value.



This procedure can be used to measure both the input, V_{in}, and output, V_{out}, levels of the voltage divider where Z_x is the impedance under test. From these values, and knowing Z_0, the value of the impedance can be derived:

$$Z_x = \frac{V_{out}}{V_{in} - V_{out}} Z_o \tag{1}$$

Z_0 is mainly the resistor R6 (120Ω), but it can also include some reactance due to C6 and wires. Also R6 wasn't required to be a precision resistor, so its value isn't very accurate. The actual value of Z_0 is going to be measured by connecting a precision resistor, R_{cal}, in the place of Z_x and solving Equation 1 for Z_0 during the first calibration step:

$$Z_0 = \left(\frac{V_{in}}{V_{out}} - 1 \right) R_{cal} \tag{2}$$

With this calibration we can start our measurements, but, if Z_x is low we are going to digitize an amplified version of V_{out} instead of the actual signal. The amplifier changes the magnitude of V_{out}, but it can also change its phase, so, the amplifier gain also has to be treated as a complex number, H_j. There are two possible gains, H_1 and H_2, one per each amplifier output, and both of them need to be calibrated. The calibration involves the same procedure as before, but with smaller calibration resistors. Also, Z_0 needs to be calibrated before the gains. The gain value is:

$$H_j = \frac{V_{out.j}}{V_{in}} \left(\frac{Z_0}{R_{cal.j}} + 1 \right) \tag{3}$$

Where $R_{cal.j}$ is the precision resistor used for calibration, and $V_{out.j}$ the measured voltage at the output of the amplifier. With the known values of the gain we can measure small impedances by just replacing V_{out} by $(V_{out.j}/H_j)$ in equation 1.

I want to remark that the actual supply voltage doesn't play any role in these equations. Of course, with lower supply voltages we get less amplitude in all sine waves, but the measured impedance is going to be the same because voltage ratios remain constant.

4.3 Complex numbers

The firmware of the microcontroller was written in C language, with only the interrupt routine being directly written in assembler. Neither of these languages have support for complex numbers, but equations 1 to 3 all include complex variables, so, a few complex arithmetic functions had to be added to the code. These are:

```
typedef struct {float re,im;} complex;

// Addition
complex cadd(complex a,complex b)
{
        complex c;
        c.re=a.re+b.re;
        c.im=a.im+b.im;
        return c;
}

// Subtraction
complex csub(complex a,complex b)
{
        complex c;
        c.re=a.re-b.re;
        c.im=a.im-b.im;
        return c;
}

// Multiplication
complex cmul(complex a,complex b)
{
        complex c;
        c.re=a.re*b.re - a.im*b.im;
        c.im=a.re*b.im + a.im*b.re;
        return c;
}

// Multiplication by a real number
complex ckmul(float k,complex a)
{
        a.re*=k;
        a.im*=k;
        return a;
}

// Division
complex cdiv(complex a,complex b)
{
        complex c;
        float d;
        d = b.re*b.re + b.im*b.im;
        c.re=(a.re*b.re + a.im*b.im);
        c.im=(a.im*b.re - a.re*b.im);
        c.re/=d;
        c.im/=d;
        return c;
}
```

Using these functions the equation 1 is coded as:

```
zx=cmul(cal.z0,cdiv(v2,csub(vin,v2)));
```

Where "v2" can be either V_{out}, or $cdiv(V_{out,j}, H_j)$

Also, as another example, equation 3 is written as:

```
cal.h1=cmul(cdiv(v2,vin),cadd((complex){1,0},ckmul(1./RCAL2,cal.z0)));
```

In this example "v2" is $V_{out,1}$, the measured complex voltage at the output of the first amplification stage, and RCAL2 is a real number with a value of 10, because a 10Ω resistor is intended to be used for the calibration of the first stage gain.

5 Results

After testing the basic correctness of the circuit, that is: the power consumption is low, the microcontroller can be programmed, the 12MHz oscillator runs, the charge pump generates a negative voltage, and so on, the next

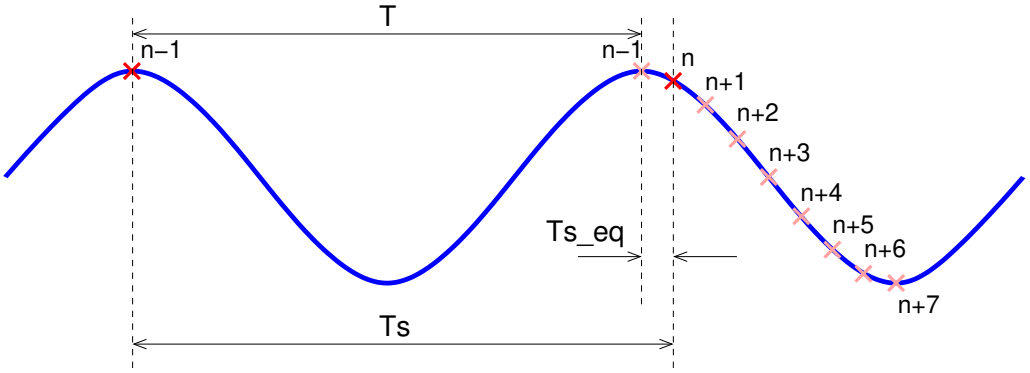


Figure 8: How to digitize a fast periodic signal with a slow ADC using subsampling.

thing I wanted to check was the linearity of the reconstruction filter. The tool for this test, that we already got integrated for free, is the ADC of the microcontroller, but if we only got four samples per each signal cycle is going to be impossible to check the harmonics of the signal. What we really need is a way faster sampling rate, but the ADC can't run much faster than 300kS/s. Fortunately, our signal may not be a pure sine but its going to be periodic for sure, and with periodic signals we can use the trick of subsampling.

The main idea behind subsampling is shown in figure 8. Here the sampling period is programmed to be slightly longer than the period of the signal. Because the signal level is the same after a time T, this is equivalent to have an effective sampling period that is the difference $(T_S - T)$, and in this way we can get a large number of samples per each signal cycle. In our case the CPU frequency is 48MHz, and this means a signal cycle is exactly 960 cycles of the CPU clock. The sampling frequency was obtained by dividing the clock by 962 (the divider had to be an even number). This is equivalent to having an effective sampling rate of 24MHz and 480 samples per each signal cycle, more than enough for a distortion analysis.

The results are shown in figure 9, were the acquired signal is shown along with its spectrum. Here the second harmonic is -60.5dB below the first, third harmonic is -61.5 dB, and the total harmonic distortion is about -56.0dBc. With these data we are really reaching the linearity limit of the ADC itself. The lower graph shows the difference between the actual samples and the best curve fit to a sine wave as a function of the signal level. An ideal sine wave in an ideal ADC would have that error confined to the -0.5 to +0.5 LSB range. Here, the visible pattern is telling us that the quantization error in the ADC is a significant part of the total difference. In any case the harmonic distortion in the filter output is low enough to be difficult to tell from the quantization error and the own nonlinearity of the ADC (the datasheet of the microcontroller states an integral nonlinearity error up to $\pm$1.5LSB), so for our purposes we can take the output of the filter as a pure sine wave.

Apart for this linearity check, that isn't required for the normal operation of the meter, the subsampling trick is also used in the code for the measurement of the peak to peak amplitude of signals. This amplitude is then used to decide how much amplification to use. The peak to peak amplitude is merely the difference between the highest subsample and the lowest during a signal cycle, and this value is used instead of the magnitude of the complex voltages because we don't want to go into clipping under any situation.

The first thing we have to do in order to have accurate measurements is to calibrate the meter. The firmware assumes default values for calibration: $Z_0 = (120 + j0)\ \Omega$, $H_1 = (11 + j0)$, $H_2 = (121 + j0)$, but these values have to be refined. The procedure to follow is:

- Connect a button switch between pins 5 and 6 of the calibration connector, J6.
- Connect an accurate 100Ω resistor to the meter leads and push the button. The measured value of Z_0 is stored into the EEPROM.
- Connect an accurate 10Ω resistor to the meter leads and push the button. The measured value of H_1 is stored into the EEPROM.

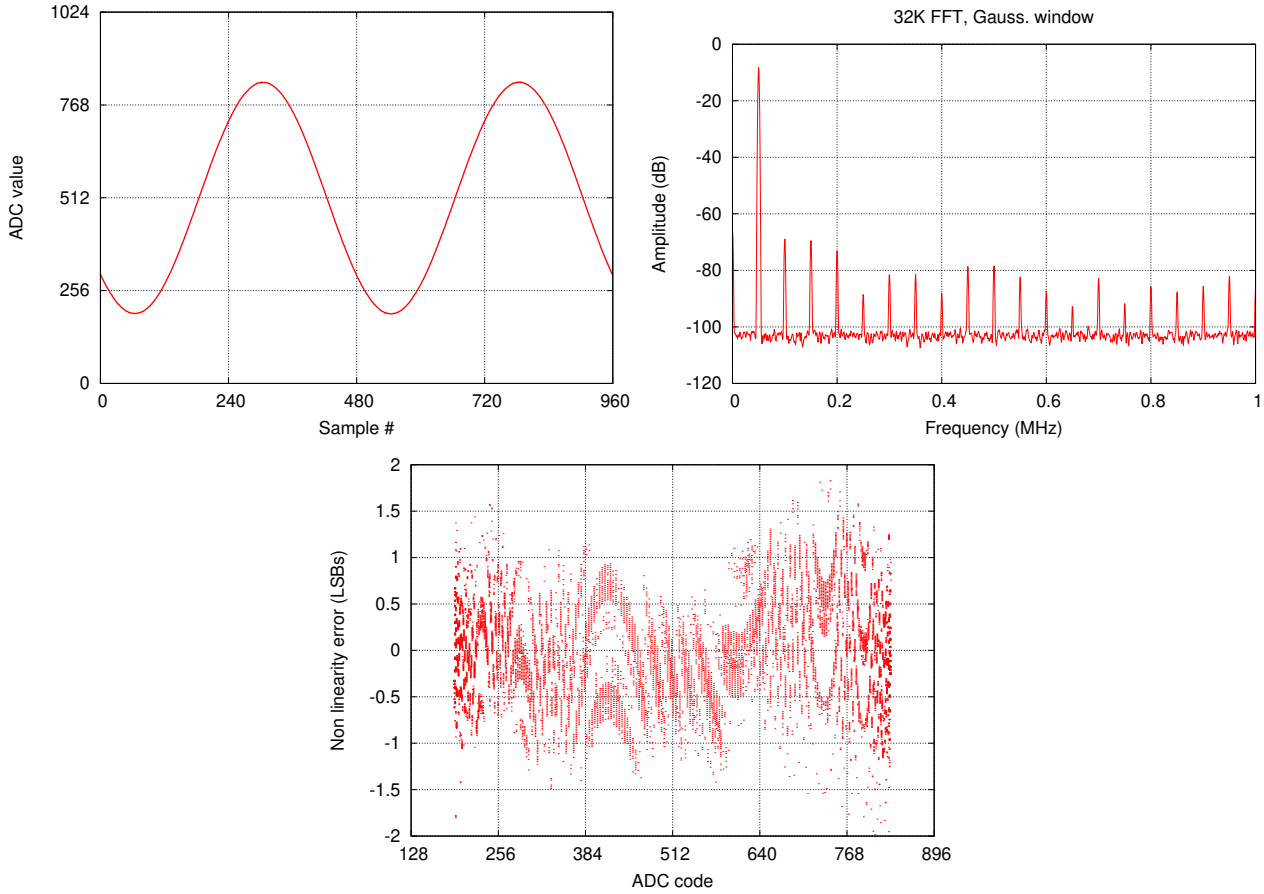


Figure 9: Upper curves: Acquired signal after the filter and its power spectrum. Lower graph: linearity error plot (10000 samples).

Figure 10: Actual calibration values

- Connect an accurate 1Ω resistor to the meter leads and push the button. The measured value of H_2 is stored into the EEPROM.

The stored values of the calibration variables are displayed after power on. In figure 10 the values obtained after the actual calibration are shown. As we can see from the photographs, the calibration values are close to the default ones, all with small imaginary parts. From the values of H_1 and H_2 we can conclude that the phase difference in the amplifiers of figure 5 is less than 1º.

We must show the actual measurements of some devices. Figure 11 (top) shows a 33μH inductor being measured, along with the way the four leads are connected to the device under test. In this measurement the RLC meter used its medium gain input ($\times$11). The inductor shows a series resistance about 0.170Ω, that is higher than the DC resistance (0.094Ω) due to the skin effect in the coil wire. This value is obtained for a 50kHz frequency, and is probably a better estimate for the inductor losses under real circumstances than the DC resistance value. This gives a quality factor for the inductor of $Q = 10.4\Omega/0.17\Omega = 61$ at 50kHz.

The measurement of a 47μF electrolytic capacitor is also shown in figure 11 (bottom). In this case the capacitance measurement isn't very accurate because the reactance is quite low (-64mΩ) and any stray inductance can alter its value significantly, but usually in these cases what we want to measure is the series resistance of the capacitor, that is 242mΩ, more than three times higher than the reactance. Still, this is a good capacitor. Faulty electrolytic capacitors usually have series resistances over 1Ω. The RLC meter used the high gain input

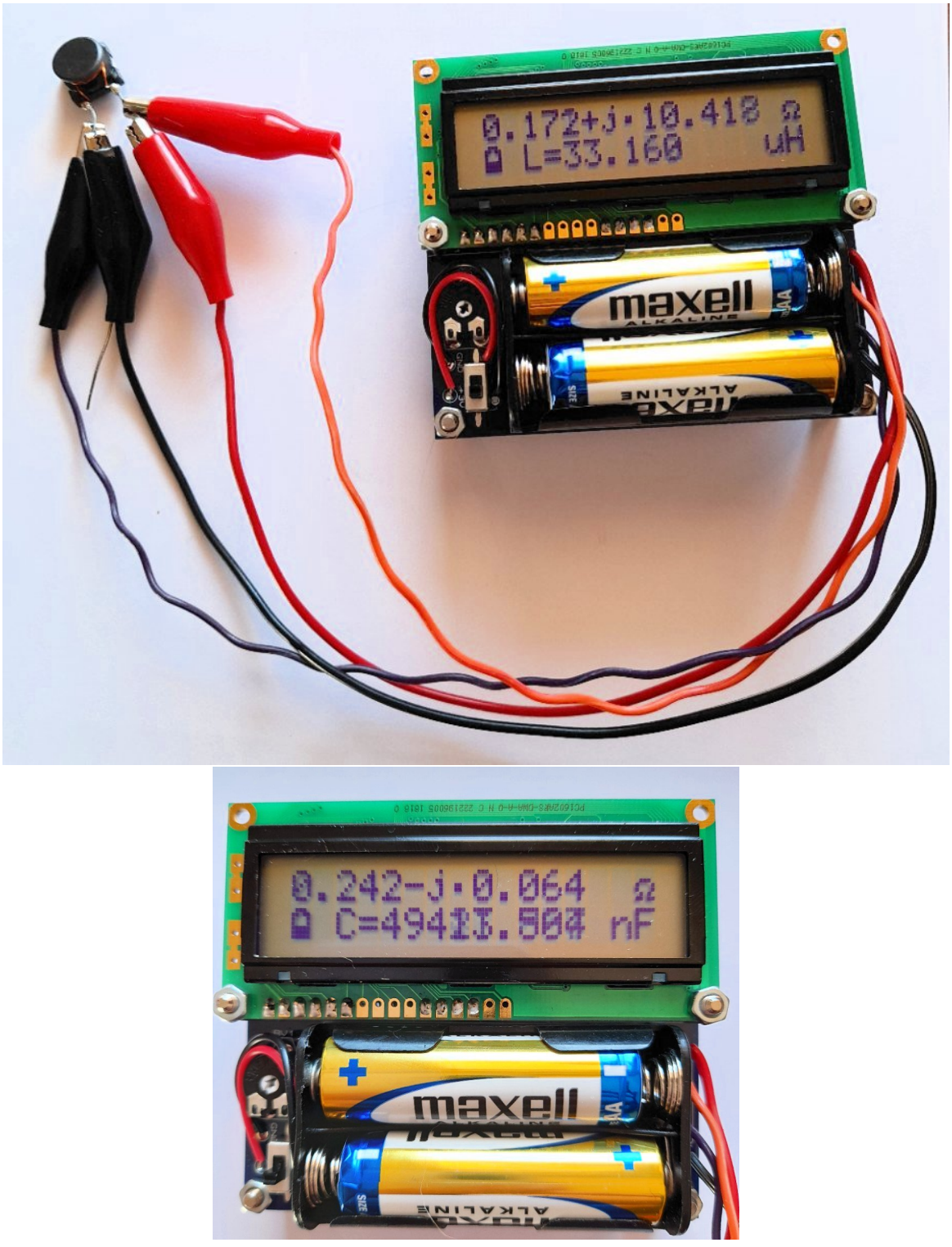


Figure 11: Top: a 33μH inductor being measured and detail of the 4-lead connection (Left to right: low sensing alligator, low driving alligator, high driving alligator, high sensing alligator). Bottom: a 47μF, electrolytic capacitor being measured for ESR.

($\times 121$) for this measurement. Also, the proper use of the four leads is mandatory if we want to avoid ending measuring an inductor instead of a capacitor.

Finally, the voltage of the batteries is measured and a single character icon is shown to display three levels of charge: full, half, and low. The way the supply voltage is measured is an indirect one: The forward voltage of the diode D3 shows little dependence with the supply voltage, but the reference voltage for the ADC is the supply itself. Thus, the value we get from the ADC increases as the supply voltage decreases. Two threshold values were calibrated using a variable power supply instead of the actual cells, one for 2.8V (full to half) and other for 2.5V (half to low). Unfortunately, the forward voltage of D3 also depends on temperature, so, the charge level shown isn't very indicative of the actual battery voltage if the meter is cold (precisely what happened with the photographs of figure 11). In fact, the contrast of the LCD screen is probably a better low battery indicator.

The current consumption of the meter is only 15mA and, therefore, the batteries are going to last at least 130 hours, assuming a cell capacity of 2000mAh, a parameter not a single cell manufacturer is willing to give you but that we can assume to be a rough estimate for an alkaline AA cell.

6 Conclusions

In this document, an RLC meter design is presented, along with the results of a prototype already tested. It is built with a low number of inexpensive components and is powered by two AA cells. No precision components, other than a quartz crystal, are included in the design of the meter, although some precision resistors are needed for calibration. Nevertheless, it still is an accurate device thanks, mainly, to the benefits of applying digital signal processing for synchronous detection. The microcontroller, a cheap ARM Cortex-M0, is powerful enough to process the incoming samples and to do all the math thanks to having a fixed sampling frequency that is exactly four times the frequency of the test signal. The fixed test signal frequency, of 50kHz, limits the measurement range of the meter, but it is still capable to measure impedances from less than 1Ω to more than 1kΩ, making it an useful tool for the testing of inductors, specially those found in switched power supplies, and also for testing the series resistance of electrolytic capacitors (In fact, a whole bunch of old electrolytic capacitors went to thrash when this RLC meter became operational).

And, with an appropriate sensing coil and another firmware, it can still be turned into an smart metal detector.