23. Lock-in detection

Lock-in detection is a technique for the detection of the amplitude and phase of a sinusoidal signal with a known frequency. Its distinctive advantage over other detection techniques is that it can selectively determine the amplitude and phase of a signal at just the desired frequency (within a narrow band), and reject signals with other frequencies. This significantly boosts the signal to noise ratio of the detection system. The example demonstrates the implementation of a lock-in detection in software.

The microcontroller first generates a sinusoidal signal at the DAC output using the DDS technique described in Chapter 16. This signal drives the externally connected RC network that alters its amplitude and phase. The resulting signal is measured by the ADC within the microcontroller. The software calculates the amplitude of the measured signal and its phase using the lock-in detection algorithm in real time.

23.1. The theory of lock-in detection

The block diagram of the lock-in detection algorithm is depicted in Fig. 23.1. The tested system is driven by a sinusoidal signal *REF*, having a predefined amplitude and frequency. The output signal from the tested system is fed to an ADC input, where it gets sampled into signal *X*. Next the signal *X* gets multiplied, once by in-phase version (R0) and once by 90-degree out-of-phase version (R90) of the excitation signal.

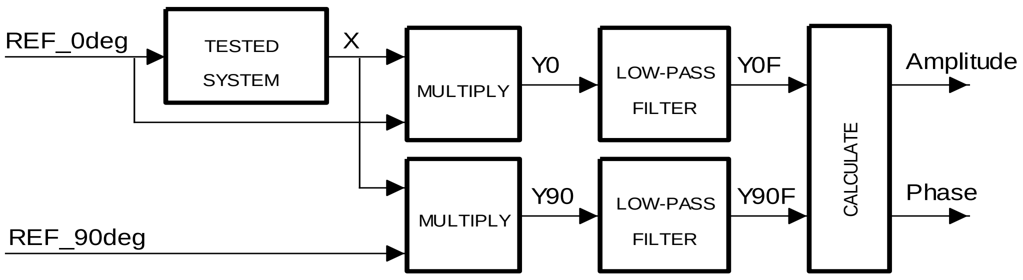

Figure 23.1: The lock-in detection: block diagram

Let the excitation signal *REF* be:

$$REF = R\sin\omega t$$

Then the input signal *X* is:

$$X = A\sin(\omega t + \varphi)$$

The input signal *X* has the same frequency as the excitation signal *REF*, but may differ in amplitude and phase. The multiplication of the input signal *X* with the two versions of the excitation signal gives the intermediate signals Y_0 and Y_{90}:

$$Y_0 = R\sin\omega t \;\cdot\; A_1\sin(\omega t + \varphi) = \frac{RA_1}{2}\,[\cos(\varphi) - \cos(2\omega t + \varphi)]$$

$$Y_{90} = R\cos\omega t \cdot A\sin(\omega t + \varphi) = \frac{RA}{2}\,[\sin(\varphi) + \sin(2\omega t + \varphi)]$$

Each of these intermediate signals is composed of two parts. One part is constant in time, the other oscillates with double the frequency of the excitation signal. Signals Y_0 and Y_{90} are sent through two low-pass filters to get rid of the AC components, and get signals Y_{0F} and Y_{90F}; we want to set the corner frequency of the low pass filter as low as practically possible to reject as much of AC component as possible.

$$Y_{0F} = \frac{RA}{2}\cos\varphi \qquad\qquad Y_{90F} = \frac{RA}{2}\sin\varphi$$

These two signals contain the information on the amplitude *A* of the signal *X* and its phase φ, we only need to calculate them using simple mathematics:

$$\sqrt{Y_{0F}^2 + Y_{90F}^2} = \frac{R\cdot A}{2} \qquad \rightarrow \qquad A = \frac{2}{R}\sqrt{Y_{0F}^2 + Y_{90F}^2}$$

$$\frac{Y_{90F}}{Y_{0F}} = \frac{\sin\varphi}{\cos\varphi} \qquad \rightarrow \qquad \varphi = \tan^{-1}\frac{Y_{90F}}{Y_{0F}}$$

Calculating the geometrical sum of signals Y_{0F} and Y_{90F} gives a weighted amplitude of the detected signal, and calculation of the arc tangent of the quotient of the two signals gives the phase angle.

The advantage of the lock-in detection can be clearly seen when the input signal *X* is a sum of two components: the first one is the same as above, and the second is an unwanted signal with a frequency ω_Z ($\omega_Z \neq \omega$) and amplitude *Z.*

$$X = A\sin(\omega t + \varphi) + Z\sin(\omega_Z t + \varphi_Z)$$

Since the system is linear the processing of the first component remains as it was described above. The processing of the second component is analyzed below. The second component is multiplied with the excitation signal and its 90 degree sibling to get components Y_{0Z} and Y_{90Z}.

$$Y_{0Z} = R\sin\omega t \cdot Z\sin(\omega_Z t + \varphi_Z) = \frac{RZ}{2}\left[\cos\big((\omega - \omega_Z)t - \varphi_X\big) - \cos\big((\omega + \omega_Z)t + \varphi_X\big)\right]$$

$$Y_{90Z} = R\cos\omega t \cdot Z\sin(\omega_Z t + \varphi_Z) = \frac{RZ}{2}\left[\sin\big((\omega + \omega_Z)t + \varphi_X\big) - \sin\big((\omega - \omega_Z)t - \varphi_X\big)\right]$$

All components in both expressions are harmonic signals with frequency other than zero. The low-pass filter being the next stage in signal processing rejects AC components, and they do not affect the outputs of filters Y_{0F} and Y_{90F}. The outputs of both filters depend only on the input signal which has the same frequency as the excitation signal.

In practice, low pass filters have a finite corner frequency f_c and finite attenuation steepness in the transition region between pass-band and stop-band, and the lock-in detection system is then sensitive for signals having exactly the frequency of the excitation signal f_{REF} and signals having a frequencies close to the frequency of the excitation signal. However, the corner frequency f_c can be made small and the steepness can be increased to reject more interfering components at the expense of filter complexity and a slower response to the change of the signal at the excitation frequency. The Fig. 23.2 gives the frequency spectrum of the sensitivity for such a system.

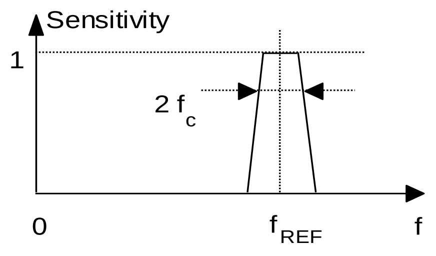

Figure 23.2: The relative sensitivity versus frequency of a lock-in detection system

23.2. The implementation of the lock-in detection

The lock-in detection will be tested on a simple passive electronic circuit that can change the amplitude and phase of a harmonic signal. The schematic diagram is given in Fig. 23.3. The microcontroller generates the driving signal *REF_0deg* at its analog output DAC1. The driving signal should have a frequency of about 1 kHz in this experiment since the values of resistor and capacitors are adjusted to this frequency to give an adequate amplitude and phase change. The potentiometer P1 is used to adjust the output amplitude, and the potentiometer P2 is used to adjust the output phase. The adjustment of phase and amplitude is not totally independent; changing the amplitude modestly affects the phase setting and vice versa. The output *X* of the circuit is applied to the analog input of the microcontroller, ADC_IN2, where it gets sampled and then processed.

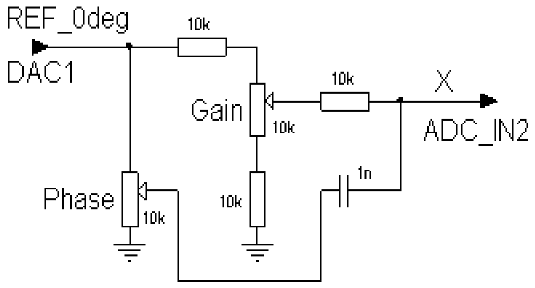

Figure 23.3: The external circuit to the BaseBoard for the lock-in detection experiment

The program is composed of two parts: one is the generation of a sinusoidal signal, and the other is the sampling and data processing of the acquired signals. The program is implemented as an interrupt routine, which is driven by timer TIM5, the time interval between successive interrupts is fixed to 20 μs.

The sinusoidal signal REF is best generated using a DDS technique, described in chapter 16.

The sampling of the input signal is implemented the same way as described in chapters 12 and 13. The timer overflow triggers the ADC to start the conversion, and the end-of-conversion signal from the ADC is used to interrupt the microcontroller and start the interrupt routine ADC_IRQhandler.

The theory behind the lock-in detection data processing is described above. The theory necessitates the calculation of two rather complex functions, namely the square root and the arc tangent. Both require processor time, in this case roughly 15 us. Since this seemed too much the calculation of both functions has been moved from the interrupt routine into the main program. Additionally, since the result of the calculation is shown to the user on an LCD screen, the software does not need to update the display and the calculation very often; this offers additional opportunity to average several consecutive results of measurement and thus improve the stability of the result displayed. The interrupt routine accumulates 64 results, and passes the accumulated version to the main program only on every 64th execution, this will be explained shortly.

The ADC interrupt function for the lock-in detection is presented in the listing below.

```
void ADC_IRQHandler(void)      {
  GPIOE->BSRRL = BIT_8;             // signal start, execution time is about 4us!            // 2

  // DDS generator
  ptrTable = (ptrTable + 655) & 0xffff;        // 655 ==> 1 kHz                             // 5
  DAC->DHR12R1 = Table[ptrTable >> 4] + 2048;   // next excitation                          // 6

  // Measurement
  X[k] = ADC1->DR;     X[k] = ADC2->DR;                                                     // 9

  // multiply with sin and cos components
  Y0[k]  = (float)X[k] * Table[((ptrTable >> 4) +    0)       ];                           // 12
  Y90[k] = (float)X[k] * Table[((ptrTable >> 4) + 1024) & 4095];                           // 13

  // LP, 1st order filter
  Y0F[k]  = 0.99999 * Y0F [(k - 1) & 63] + 0.00001 * Y0 [k];                               // 16
  Y90F[k] = 0.99999 * Y90F[(k - 1) & 63] + 0.00001 * Y90[k];                               // 17

  // some averaging to get smoother result
  R_im_acc += Y90F[k];          R_re_acc += Y0F[k];                                        // 20

  // report result every 64th time
  if (k == 0) {                                                                            // 23
    R_im_fin = R_im_acc;        R_re_fin = R_re_acc;                                       // 24
    R_im_acc = 0;               R_re_acc = 0;                                              // 25
  };                                                                                       // 26

  k++; k &= 63;                                                                            // 28
  GPIOE->BSRRH = BIT_8;                 // signal end                                      // 29
}
```

The function commences with setting pin 8, port E, high to signal the start of execution, line 2; the same pin is reset when the function ends, line 29. The generation of the excitation signal is given in lines 5 and 6. First the pointer to the table with sine samples is updated; the incrementing value of 655 gives the frequency of the excitation signal of 1 kHz. Next the entry from the sine table is read and sent to the DAC. The table has 4096 entries, and must be prepared in advance during the initialization process of the microcontroller.

Next the result is read from the ADC in line 9. Since we are using the same initialization routine for ADC throughout this text, and this routine initializes two ADCs, both results should be read one after another to clear their respective interrupt flags. The last result read representing the measured analog signal is stored into the circular buffer named *X* at position *k*. The position pointer *k* increments on every execution of the interrupt routine, line 28. This circular buffer has 64 elements; any number of elements above 2 would do, and 64 is selected arbitrarily.

As the lock-in detection requires the measured value is next multiplied by the sine and cosine components of the excitation signal. This is simple. The sine component is the same as used for the DDS generator, line 12, and the cosine component can be obtained from the same table using and offset of a quarter of the table length, namely 1024, line 13. Function “& 4095” must be used in line 13 to keep the pointer within the table boundaries. Also note that the calculation is performed by floating point numbers to increase precision; one of the arguments in multiplication is converted to “float” before the multiplication. The results are stored in two circular buffers named *Y0* and *Y90* at location *k*.

Low-pass filtering comes next. One might be tempted to implement the filter described in Chapter 20, the IIR filtering. However, in this case such filter might introduce additional problems. We want a low-pass filter with a very low corner frequency f_c, and such filters can be unstable due to the limited precision of calculations and due to the digitalization of filter coefficients, see the reference [1] for details. To elude such problems a simpler first order low-pass filter can be used. First order low-pass filter has inferior frequency characteristics, but is stable and can have very low corner frequency. Its difference equation is:

$$y_k = \alpha \cdot y_{k-1} + (1 - \alpha) \cdot x_k$$

Factor α determines the corner frequency with an approximate formula:

$$f_c = -\frac{f_s \cdot \ln \alpha}{2\pi}$$

In our case the sampling frequency f_s equals 10^5 Hz, and the factor α equals 0.99999; this gives the corner frequency of about 0.16 Hz. Filters for both signals Y0 and Y90 are implemented in lines 16 and 17 providing signals *FY0* and *FY90*; results reside in two circular buffers.

Next comes the averaging, this is implemented in lines 20 to 26 including. There are two global variables *R_im_fin* and *R_re_fin* declared, and on every 64th execution (line 23) of the interrupt routine the content of these two variables gets updated in line 24. There are two variables *R_im_acc* and *R_re_acc*, these are used to accumulate the results in line 20, and when the accumulated values are passed to the main the accumulating variables are reset in line 25.

The listing of the rest of the program is given below.

```
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.c"
#include "stm32f4xx_adc.c"
#include "stm32f4xx_dac.c"
#include "stm32f4xx_gpio.c"
#include "stm32f4xx_tim.c"
#include "dd.h"
#include "LCD2x16.c"
#include "math.h"
#define pi 3.14159

short int X[64], k = 0;                                                                // 11
float Y0[64], Y90[64], float Y0F[64], Y90F[64], float Y1[64], Y2[64], Y3[64];          // 12
int Table[4096], ptrTable;                                                             // 13

int main ()  {                                                                         // 15
int del;
  for (ptrTable = 0; ptrTable <= 4095; ptrTable++)                                     // 17
    Table[ptrTable] = (int)(1850.0 * sin((float)ptrTable / 2048.0 * 3.14159265));      // 18

  LCD_init ();                                                                         // 20
  LCD_string("Pha=         dg", 0x00);                                                 // 21
  LCD_string("Amp=         mV", 0x40);                                                 // 22

  GPIOEinit ();                                                                        // 24
  ADCinit_T5_CC1_IRQ();                                                                // 25
  DACinit();                                                                           // 26
  TIM5init_TimeBase_CC1(840);           // 840 == 10 us == 100 kHz                     // 27

  while (1) {                            // endless loop                               // 29
    for (del = 0; del < 10000000; del++)  {};                                          // 30
    float Phi_fin = (atan(R_im_fin/R_re_fin)) * 180 / pi;                              // 31
    if (Phi_fin < 0)    {LCD_string("-", 0x06);    Phi_fin = -Phi_fin; }               // 32
    else                {LCD_string(" ", 0x06);                        };              // 33
    short Phi_int = (int)Phi_fin;                                                      // 34
    short Phi_frc = (int)((Phi_fin - Phi_int) * 10);                                   // 35
    LCD_sInt3DG((int)Phi_int , 0x07, 1);                                               // 36
    LCD_char('.');   LCD_char(Phi_frc + '0');                                          // 37
    float Amp_fin = (sqrt(R_im_fin * R_im_fin + R_re_fin * R_re_fin))/128*.838;        // 38
    LCD_sInt16((int)(Amp_fin/256), 0x47, 1);                                           // 39
  };                                                                                   // 40
}
```

This listing starts with the inclusion of CMSIS files and the declaration of the variables used. The function “main()” starts in line 17 with the initialization of the table to generate analog signal from as already explained in chapter 16, and continues to the configuration functions for LCD, port E, ADC, DAC and timer TIM5; all configuration functions were already described in previous chapters.

The endless while loop, lines 29 to 40, is used to calculate the amplitude and the phase angle of the signal *X* from the results passed by the ADC interrupt routine, and to display the calculated value at the LCD display.

The endless loop starts with a delay in line 30; we do not need to display results very often, human eyes cannot follow fast changing numbers anyway. The code in line 31 is used to calculate the phase angle *Phi_fin* in degrees following the formula given in the theory.

Lines 32 to 37 contain some wizardry to display a floating point number. The “LCD2x16.c” does not comprise a function for displaying a floating point number, and the “sprintf” function is not used in order to keep the final code short.

- The sign of the floating point number is dealt with first. If the number is negative, then it is made positive in line 32, and the negation sign is shown the the LCD, else the negation sign in cleared in line 33.
- The floating point number is split in integer (*Phi_int*) and fractional (*Phi_frc*) part, fractional part having only one digit, lines 34 and 35.
- The integer part is displayed in line 36, the position at the LCD is selected just after the minus sign, if it exists.
- The decimal point is displayed immediately following the integer part in line 37, followed by the fractional digit.

The amplitude of the signal X is calculated in line 38 by calculating the geometrical sum of the two components. The outcome of the square root calculation is too big for a factor of 8 due to the averaging and still some due to the amplitude of the excitation signal; both is adjusted in the same line by a simple multiplication. The amplitude is then displayed at the LCD screen using a function call in line 39.