28. Coin detector

This example demonstrates the use of a microprocessor to measure small changes of signal phase in a resonant LC circuit when a metal object is inserted into the coil. The setup can be used to detect small metal objects like coins. Basic data: the coil has a diameter of 160 mm and 15 turns of wire ϕ = 1 mm, the capacitor has the capacitance of 2.2 μF; the resonant frequency if this LC circuit is about 11 kHz.

28.1 The theory

Consider a simple circuit from figure 28.1 consisting of a parallel resonant connection of a coil *L* and capacitor *C*; here RL represents the resistance of the wire in the coil. When such a resonant circuit is excited by a sinusoidal signal *REF* with constant amplitude and known frequency, the output signal of the circuit *Z* is also a sinusoidal signal, but its amplitude and phase depends on the resonant frequency of the LC circuit and the frequency of excitation signal *REF*. The relation between *Z* and *REF* is given by the formula:

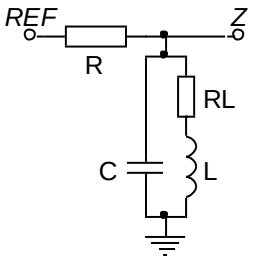

Figure 28.1: A resonant circuit with excitation

$$Z = REF\ \frac{R_L + i\omega L}{R_L + R + i\omega(RR_LC + L) - RLC\omega^2}$$

From here the *Gain* of the circuit ($Gain = {|Z|}/{|REF|}$) and the phase relation φ between the output and the input signal can be derived as:

$$Gain = \sqrt{\frac{\omega^2L^2 + R_L^2}{\omega^2R^2C^2(\omega^2L^2 + R_L^2) - 2\omega^2R^2LC + \omega^2L + R^2 + 2RR_L + R_L^2}}$$

$$\tan\varphi = \frac{\omega R\left(L - C(L^2\omega^2 + R_L^2)\right)}{\omega^2L^2 + R_L(R + R_{lL})}$$

Both are shown in diagrams in Fig. 28.2, solid lines, for selected values of components. The frequency where the phase angle between the output and the excitation signal is zero is called the resonant frequency ω_R:

$$\omega_R = \sqrt{\frac{1}{LC} - \frac{R_L^2}{L^2}}$$

From former equations (and experience) we know that the resonant frequency depends on inductance *L* and capacitance *C* in the circuit from Fig. 1. Increasing either the capacitance or inductance shifts both curves towards lower frequencies, Fig, 28.2, dashed lines. Additionally, the

equations (and experience) tell us that the value of the resistor R_L defines the width of the resonant curve and simultaneously the steepness of the phase change, when the capacitance and the inductance are kept constant; smaller value of this resistor results in narrower resonant curve.

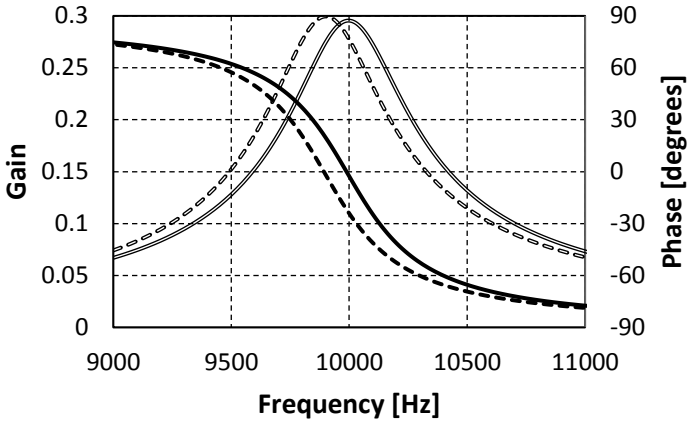

Figure 28.2: Gain (gray) and phase (black) versus frequency for the circuit at Fig. 28.1; scales are normalized.

If the frequency of the excitation *REF* is kept constant at the resonant frequency of the original circuit, the increase in inductance as depicted in fig. 28.2 will cause the reduction of the output amplitude by about 10 % and the increase of the phase angle to about -30 degrees. Such effects, especially the change of the phase, can be easily measured and the results used to detect the insertion of a coin into the coil (for instance).

The measurement of the phase angle might be quite a complex task, as described in chapter 24. However, when the microprocessor is used for both excitation and detection, there are simpler ways. Consider the signals presented at Figure 28.2; both are sinusoidal, and the dashed one is phase shifted for about -30 degrees. The solid line represents the signal *Z* without anything inserted into the coil, and the dashed signal is the signal *Z* obtained after the insertion. Instantaneous value of the signal is measured four times per period: at 0, 90, 180 and 270 degrees to obtain four results Z_0, Z_{90}, Z_{180} and Z_{270}.

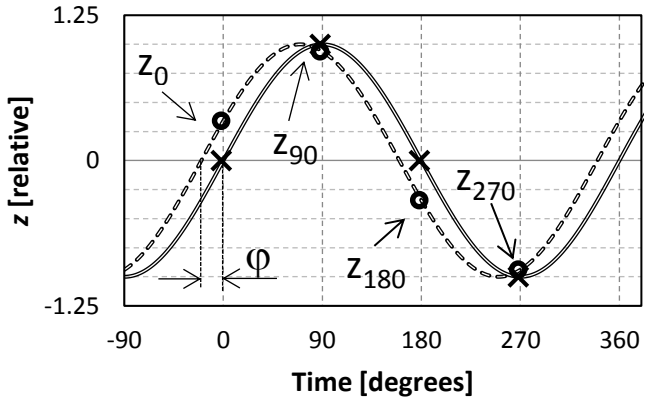

Figure 28.3: Original signal Z (solid) and phase shifted version (dashed) after the insertion of a ferromagnetic material into the coil.

Obviously, the difference $Z_0 - Z_{180}$ is directly proportional to the phase angle φ for small phase angles, since the sinusoidal shape can then be approximated by a straight line. However, the slope of this line is proportional to the amplitude of the signal, and this can be calculated from the difference $Z_{90} - Z_{270}$ to normalize the calculated angle with the amplitude. The calculated phase angle from four consecutive points on the signal is therefore (approximately, for small angles) given by:

$$\varphi = -const \cdot \frac{Z_0 - Z_{180}}{Z_{90} - Z_{270}}$$

The driving signal *REF* for the circuit can be sinusoidal, but can also be a square-wave. A square-wave is composed of an infinite number of sinusoidal signals having odd multiples of the fundamental frequency (Fourier!); it looks like we are driving the circuit with an infinite number of different sinusoidal signals. However, when a square-wave signal is used to excite a resonant circuit, all consistent odd harmonics fall at frequencies where the impedance of the LC circuit is very small, and

are therefore filtered out. When signal *REF* is a square-wave with a frequency equal to the resonant frequency of the LC circuit, the output signal *Z* is almost pure sinusoidal in shape.

28.2 The Hardware

A square-wave signal to drive the resonant circuit is simple to make with a microcontroller. All we need is a timer having its repetition rate and PWM output configured properly. We still need four triggering signals for the ADC to determine the value of signal *Z* at 0, 90, 180 and 270 degrees of the generated square-wave.

Consider the timing diagram at Figure 28.4. The first square-wave signal SQ runs with a frequency that is four times the resonant frequency of the LC circuit. This signal is divided by four in frequency to obtain a square-wave for driving *REF*. The rising edges of the signal SQ can be used to trigger the ADC as we are already used to from previous experiments.

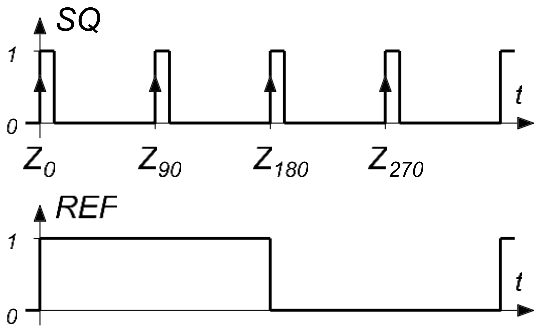

Figure 28.4: Signals need for this experiment

Such pair of signals can be generated by two timers connected serially. The signal SQ can be derived for instance by timer TIM1 from the main clock of the microcontroller (168 MHz) by Compare channel 1. It does not need to be available at a pin of the microcontroller since we need it only to start a conversion (SC) at the ADC and to get it divided by four. A second timer, say timer TIM4, can be used for this additional division. This should count 0, 1, 2, 3, 0, 1, 2, 3 …, therefore divide its input signal by four in frequency. The PWM output of this timer (say Compare channel 3) should be configured to stay high when the content of the counter is 0 or 1, and change low for the other two contents of 2 and 3. This output *REF* is used to drive the LC circuit, and it must be available at a pin of the microcontroller. The connection of both timers is shown at Figure 28.5. For the sake of demonstration, the output SQ will also be available at a pin of the microcontroller. The compare channels of both timers were selected since they are available at the BaseBoard, connector K406, pins 1 and 5 respectively.

Timers can be connected serially inside of the microcontroller. The first timer is called the “master”, and the second is a “slave”. The “master” timer must be configured to output a pulse for triggering of the slave timer, the corresponding output of this timer is called TRGO (trigger output), and the slave timer must be configured to use this signal as its clock, the corresponding input is called ITRx (internal trigger, there are four possible choices in total). However, not all combinations of timers are possible;

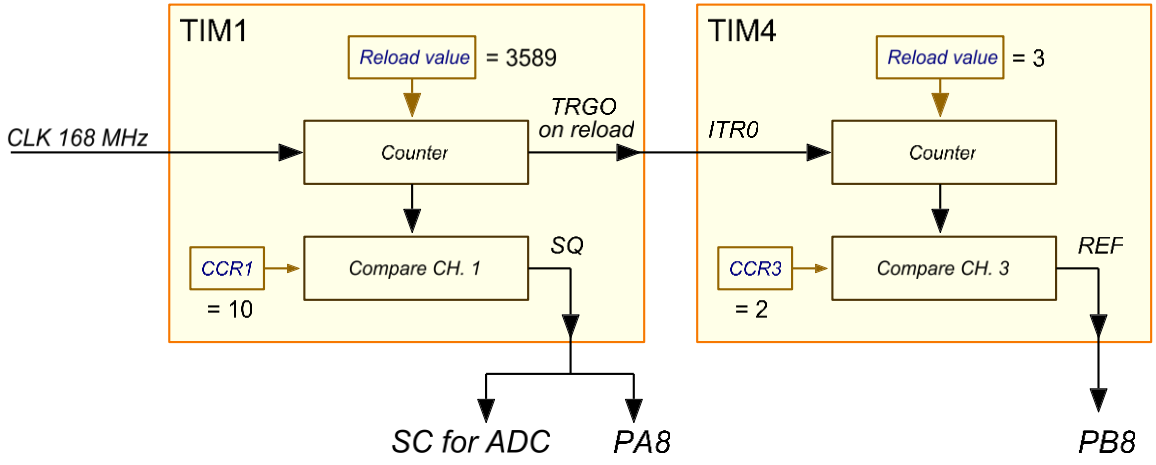

Figure 28.5: The use of two counters to generate the required signals

the RM0090 states that timer TIM 1 can be used as master for timer TIM4 when internal trigger ITR1 is selected as the clock source for timer TIM4, and with this the aforementioned selection of timers is justified.

The resonant frequency of the LC circuit used is about 11700 Hz. The "master" timer TIM1 must therefore divide the clock frequency of 168 MHz by:

$$Reload\ value_{TIM1} = {}^{168\ MHz}/_{11700 * 4} = 3589$$

And this is the reload value that must be programmed into a reload register of timer TIM1. Since the resonant frequency could change for different LC circuits, the reload value and with this the division ration should be made adjustable in software. The width of pulses at the PWM output is not important, and we can make this short by setting the CCR1 register to a small value, say 10.

The reload value for the "slave" timer TIM4 is fixed and equal to 3 (the timer then counts from including 0 to including 3, four steps altogether). Since the output of the Compare channel 3 should be high half of the time, the CCR3 register of this timer is set to 2 (the output should stay high for counter states 0 and 1, and change low for states 2 and 3).

27.1. The Software

The program commences with configuration routines for port E (switches for setting the frequency), LCD (displaying results of measurement), and continues with the configuration of ADC, TIM1 and TIM4, followed by commands for enabling both timers and by writing some introductory text on the LCD. Next the program enters an endless loop and waits for results from the interrupt routine; when results are available, it checks switches, corrects frequency if needed, and writes result on the LCD. All action takes place in interrupt routine.

a) ADC

The configuration for the ADC is almost the same as used in previous experiments, the only difference being the source of the start conversion signal (Time1, CCR1 in this case, line 22 below). The end of conversion will again trigger an interrupt. For the sake of simplicity of keeping the same configuration routine as used previously two ADCs are configured as in most of the previous examples, although one ADC would suffice for this experiment.

```
void ADCinit_T1_CC1_IRQ (void)      {
ADC_InitTypeDef         ADC_InitStructure;
GPIO_InitTypeDef        GPIO_InitStructure;
ADC_CommonInitTypeDef   ADC_CommonInitStructure;
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1  | RCC_APB2Periph_ADC2,  ENABLE);
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA | RCC_AHB1Periph_GPIOB, ENABLE);

  GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_1 | GPIO_Pin_2;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  ADC_CommonInitStructure.ADC_DMAAccessMode    = ADC_DMAAccessMode_Disabled;
  ADC_CommonInitStructure.ADC_Mode             = ADC_DualMode_RegSimult;
  ADC_CommonInitStructure.ADC_Prescaler        = ADC_Prescaler_Div2;
  ADC_CommonInit(&ADC_CommonInitStructure);

  ADC_InitStructure.ADC_Resolution             = ADC_Resolution_12b;
  ADC_InitStructure.ADC_ScanConvMode           = DISABLE;
  ADC_InitStructure.ADC_ContinuousConvMode     = DISABLE;
  ADC_InitStructure.ADC_ExternalTrigConvEdge   = ADC_ExternalTrigConvEdge_Rising;
```

```
  ADC_InitStructure.ADC_ExternalTrigConv       = ADC_ExternalTrigConv_T1_CC1;          // CCR1
  ADC_InitStructure.ADC_DataAlign              = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfConversion        = 1;
  ADC_Init(ADC1, &ADC_InitStructure);
  ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1, ADC_SampleTime_3Cycles);

  ADC_Init(ADC2, &ADC_InitStructure);
  ADC_RegularChannelConfig(ADC2, ADC_Channel_2, 1, ADC_SampleTime_3Cycles);

  ADC_Cmd(ADC1, ENABLE);
  ADC_Cmd(ADC2, ENABLE);

  NVIC_EnableIRQ(ADC_IRQn);                // Enable IRQ for ADC in NVIC
  ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);  // Enable IRQ generation in ADC
}
```

b) TIM1 – master timer

Timer TIM1 configuration is done in several steps: first the pin for outputting the signal *SW* is configured, then the main part of the master counter is set-up. This is followed by the configuration of the Compare Ch.1, and finally output trigger source for TRGO is selected. The routine follows:

```
void TIM1init_TimeBase_CC1 (void)  {
TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
TIM_OCInitTypeDef       TIM_OCInitStructure;
GPIO_InitTypeDef        GPIO_InitStructure;

  // configure output pin, TIM1 Compare Ch1 output can be connected to PA8, K406/pin1
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
  GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_8;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_25MHz;
  GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource8, GPIO_AF_TIM1);

  // configure main part of the master timer TIM1
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1,  ENABLE);
  TIM_TimeBaseInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
  TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInitStructure.TIM_Period = 3389;
  TIM_TimeBaseInitStructure.TIM_Prescaler = 0;
  TIM_TimeBaseInitStructure.TIM_RepetitionCounter = 0;
  TIM_TimeBaseInit(TIM1, &TIM_TimeBaseInitStructure);

  // configure Compare CH1 module of the master timer
  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStructure.TIM_OutputNState = TIM_OutputState_Disable;
  TIM_OCInitStructure.TIM_Pulse = 10;
  TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Reset;
  TIM_OCInitStructure.TIM_OCNIdleState =  TIM_OCIdleState_Set;
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
  TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_Low;
  TIM_OC1Init(TIM1, &TIM_OCInitStructure);                          // configure now
  TIM_CtrlPWMOutputs(TIM1, ENABLE);                                 // enable PWM outputs
```

```
  // select the source of the output signal TRGO from master timer
  TIM_SelectOutputTrigger(TIM1, TIM_TRGOSource_Update);
}
```

The listing is identical to listing for configuration of a timer from previous examples for sections 1 and 2. However, sections 3 (configure Compare Ch1 module) and 4 (select the source of the output signal) may need some clarification.

The reuired data structure for a call to configure Compare module was declared at the top of this routine, here members of the structure are initialized. First the PWM1 mode is selected for this Compare output, see RM0090 for details. Next the true output is enabled and the inverted is disabled, and the content of the Compare register CCR1 is set to 10 to produce narrow pulses of about 60 ns at the output of this module. The true output is set to produce positive pulses, and the false output is not important since it is not enabled. The data structure is then used to configure the Compare module 1, and the PWM output is enabled.

The section 4 consists of one command only, and this selects TRGO output to change when the content of the counter in timer TIM1 is changed from the maximum value back to zero.

c) TIM4 – slave timer

Timer TIM4 configuration is also completed through several steps: first the pin for outputting the signal *REF* is configured, then the main part of the slave counter is set-up. This is followed by the selection of the clock signal for this counter, and finished by the configuration of the Compare Ch.3 for this timer. The routine follows:

```
void TIM4init_Slave_div4_CH3out (void)  {
TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
TIM_OCInitTypeDef       TIM_OCInitStructure;
GPIO_InitTypeDef        GPIO_InitStructure;

  // configure output pin, TIM4 Compare Ch3 output can be connected to PB8, K406/pin5
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
  GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_8;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_25MHz;
  GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOB, &GPIO_InitStructure);
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource8, GPIO_AF_TIM4);

  // configure main part of the slave timer TIM4
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4,  ENABLE);
  TIM_TimeBaseInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
  TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInitStructure.TIM_Period = 3;
  TIM_TimeBaseInitStructure.TIM_Prescaler = 0;
  TIM_TimeBaseInitStructure.TIM_RepetitionCounter = 0;
  TIM_TimeBaseInit(TIM4, &TIM_TimeBaseInitStructure);

  // select clock input and put the slave timer into correct mode
  TIM_SelectInputTrigger(TIM4, TIM_TS_ITR0);               // clock from TIM1
  TIM_SelectSlaveMode(TIM4, TIM_SlaveMode_External1);      // make this timer a “slave”

  // configure Compare Ch3 module of the slave timer
  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStructure.TIM_OutputNState = TIM_OutputState_Disable;
```

```
  TIM_OCInitStructure.TIM_Pulse = 2;
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
  TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_Low;
  TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Reset;
  TIM_OCInitStructure.TIM_OCNIdleState =  TIM_OCIdleState_Set;
  TIM_OC3Init(TIM4, &TIM_OCInitStructure);
}
```

Section three on selecting the clock input for this "slave" timer needs some additional explanation. It consists of two steps: first step selects ITR0 as the input of the clock, and the second step puts this counter in the "slave mode". Two steps are required due to the fact two multiplexers within timer TIM4 are responsible for selection of the driving clock, and each of them is adjusted in its own step.

The configuration in the last block is standard for a Compare module. This time the content of the compare register CCR3 is set to 2, making the output signal symetrical (half time high, half time low).

d) Interrupt routine for ADC

The listing of the interrupt routine for ADC is presented below. The microprocessor enters this routine about every 21 μs (four times per period of the driving signal *REF*).

```
void ADC_IRQHandler(void)      {
  ADCresult[TIM4->CNT] = ADC1->DR;
  if (TIM4->CNT == 3) {
    phase = ((float)(ADCresult[0] - ADCresult[2])) / ((float)(ADCresult[1] - ADCresult[3]));
    Avgs++;
    if (++Avgs < AVGSCOUNT) {
      PhaseSum += phase;
    } else {
      Avgs = 0;
      PhaseSafe = PhaseSum;      PhaseSum = 0;      MyFlag = 1;
    };
  };
}
```

An array named ADCresult with four members has been declared globally, and the result from the ADC is transferred into this array on every iteration of this routine. The position in the array is defined by the content of the counter in timer TIM4, and is therefore 0 to 3 including, which always points into the array.

When the content of the counter in timer TIM4 reaches 3 all four samples within a period are available, and the calculation of the phase angle commences following the formula given in "The theory". The individual calculated angle might be noisy, so an average over several (AVGSCOUNT) periods gets calculated. The calculated phase angled are accumulated in variable PhaseSum, and the accumulated version is periodically transferred to the main program to be presented at the LCD. The availability of a newly calculated value is signaled to the main program by setting the variable MyFlag to 1.

e) Main program

As stated the main program commences with several calls to configuration routines, followed by the infinite "while" loop.

```
int main ()  {
  GPIOE_Config();
  LCD_init();
  ADCinit_T1_CC1_IRQ();
  TIM1init_TimeBase_CC1();
  TIM4init_Slave_div4_CH3out();
```

```
  TIM_Cmd(TIM4, ENABLE);
  TIM_Cmd(TIM1, ENABLE);
  LCD_string("f[Hz]  = ", 0x40);
  LCD_string("Result = ", 0x00);

  while (1) {                          // endless loop
    if (MyFlag == 1) {
      MyFlag = 0;
      unsigned sw = GPIOE->IDR;
      if ((sw & S370) && (QuPer  < 4000)) QuPer++;
      if ((sw & S371) && (QuPer  > 3000)) QuPer--;
      TIM1->ARR = QuPer;
      LCD_uInt16(QuPer*11811/3585, 0x4a, 0x01);
      LCD_sInt16((int)(PhaseSafe / AVGSCOUNT * 5000), 0x09, 1);  // write result on LCD, 1st line
    };
  };
}
```

Within the loop the state of the flag MyFlag is checked, and when this becomes 1 switches are checked and the frequency of the driving signal *REF* is corrected (the line: TIM1->ARR = QuPer; updates the content of the auto-reload register ARR for the master timer TIM1). Additionally, the calculated sum of several measurements is normalized and shown at the LCD.