22. PLL and frequency demodulation

The PLL (Phase Locked Loop) describes the common way for generating a signal with a frequency which is in a fixed relation with the frequency of the reference signal (like multiple). It can also be used for measuring of frequency and the frequency demodulation. The example demonstrates the implementation of such block in software. The digital input signal f_{in} is connected to port E, pin 15, and the microcontroller generates a sine wave signal with the same frequency and phase DAC1 output. The output from DAC2 is used as a frequency demodulated output.

22.1. The theory of PLL

A Phase Locked Loop (PLL) is another commonly used block in digital electronics. The PLL block is capable of generating a signal f_{VCO} with a frequency which is the same as the frequency of the input signal f_{IN}; it is given in Fig. 22.1 in its basic form. By adding two dividers (one in series with each of the input signals to the phase comparator) the block generates a signal f_{VCO} which is the ratio of the two division factors multiplied by the frequency of the input signal f_{IN}. The same block can be used for frequency demodulation of the input signal f_{IN}; the signal T_P is proportional to the frequency of the input signal f_{IN}.

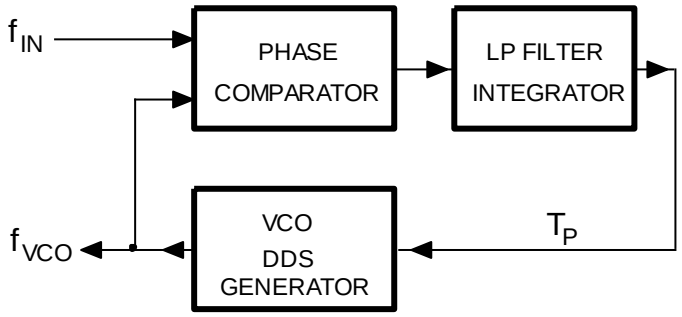

Figure 22.1: Graphical representation of the PLL block

In the following example we will program the PLL into the microcontroller and obtain a frequency meter and frequency demodulator at the same time. For those with experience in digital electronics: there are basically two types of phase comparators, XOR gate and a memory circuit (referred as Type I and Type II in typical PLL chip CD4046); we will implement the memory circuit in this example since it prevents PLL to lock on harmonics of the input signal.

The frequency of the local oscillator (VCO, DDS GENERATOR) f_{VCO} depends on the control signal Tp, and can be the same as the frequency of the input signal f_{IN}, for a certain value of control signal Tp, as shown in Fig. 22.2. In this case the PLL block is locked to the input signal in frequency and phase, and no further action is needed; signal T_p does not need to be adjusted. Arrows indicate moments of sampling of the two signals, and numbers their current value.

When phases (and frequencies) of the two signals differ, two scenarios are possible. The local signal f_{VCO} can be delayed compared to the input signal f_{IN}, as shown in Fig. 22.3, left; in this case the frequency of the local signal f_{VCO} must be increased to align the edges, therefore the control signal T_p

must be increased. Alternatively, the local signal f_{VCO} can come ahead of the input signal f_{IN} as shown in the same figure, right; in this case the frequency of the local signal must be decreased (and T_p as well) to match the edges.

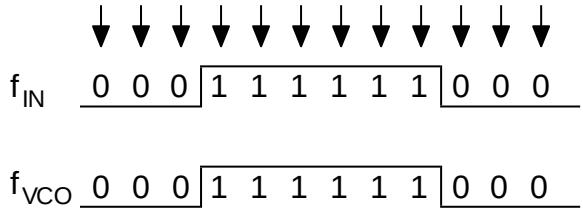

Figure 22.2: PLL block is locked to the input signal;
f_{VCO} is equal to f_{IN} in frequency and phase, no adjustments of T_p are needed

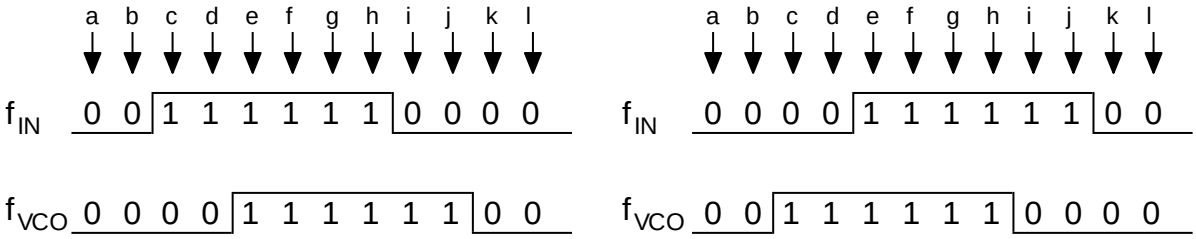

Figure 22.3: Two possible scenarios: left => frequency of signal f_{VCO} must be increased to match the phase;
right => frequency of the signal f_{VCO} must be decreased to match the phase

From Fig. 22.3, left, we can deduce to increase the frequency (therefore control signal T_p) while the input signal f_{IN} is high and the local signal f_{VCO} is low, and also while the input signal f_{IN} is low and the local signal f_{VCO} is high. The same can be deduced for decreasing the frequency from Fig. 22.3, right. What to do, then?

Consider the current and past samples of the input signal. The consecutive samples can be arranged to form an array of bits having value of either zero or one. Such array, when short, can comprise an integer variable; bit 0 (LSB) of the variable belongs to the current sample of the input signal, bit 1 belongs to the previous sample, bit 2 to the pre-previous sample... , like 001111110000 for the Fig. 22.3, left, top. When two digital signals (for f_{IN} and f_{VCO}) are sampled simultaneously, consecutive samples can be arranged in a common integer variable in such a way that odd bits (bit 1, bit 3,…) of the integer variable represent signal f_{IN}, and even bits (bit 0, bit 2,…) signal f_{VCO}. Additionally, let the least significant two bits represent the current values of both signals, and adjacent two more significant bits past values of the same signals, Fig. 22.4. Only four bits are important; for instance, at time **c** (the third sample, Fig. 22.3, left) the corresponding integer number reads 0010_b.

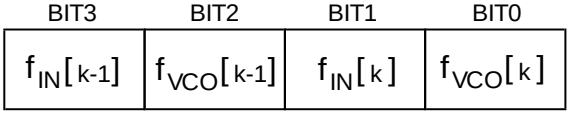

Figure 22.4: The arrangement of bits in variable InPat for the phase detection

Using this construct we can now identify the following situations:

- When signal f_{IN} and f_{VCO} have equal values we should not change the frequency of the VCO since we have no significant information on the relation of two frequencies (control signal T_p). The relation is hidden in the position of the edges of both signals, and we should concentrate on those.

- When signals have different values, we should change frequency of the VCO:
  - The frequency should start increasing when the constructed integer *InPat* becomes 0010b, and keep increasing until both signals become equal. This situation means that the leading edge at f_{IN} came earlier than the leading edge at f_{VCO}, so the frequency of f_{VCO} should be increased (T_p increased).
  - The frequency should also start increasing when the constructed integer *InPat* becomes 1101b, and keep increasing until both signals become equal. This situation means that the falling edge at f_{IN} came earlier than the falling edge at f_{VCO}, so the frequency of f_{VCO} should be increased (T_p increased).
  - The frequency should start decreasing when the constructed integer *InPat* becomes 0001b, and should keep decreasing until both signals become equal. This situation means that the leading edge at f_{VCO} came earlier than the falling edge at f_{IN}, so the frequency of f_{VCO} is too big and should be decreased (T_p decreased).
  - The frequency should also start decreasing when the constructed integer *InPat* becomes 1110b, and should keep decreasing until both signals become equal. This situation means that the falling edge at f_{VCO} came earlier than the falling edge at f_{IN}, so the frequency of f_{VCO} is too high and should be decreased (T_p decreased).

The unit performing this functions (constructing the integer value and calculating the required value of T_p) replaces both the phase comparator and the LP filter / integrator in the block diagram in Fig. 22.1.

22.2. The implementation of the PLL

The program to implement for PLL unit is based on the program for the DDS, chapter 16. The structure of the program is the same: the timer TIM5 is used to trigger periodic interrupts every 10 μs. All PLL processing is performed within the interrupt function for timer TIM5. The main program is used only to display current frequency of the DDS generator which is the same as the frequency of the input signal, when PLL is locked.

The interrupt function is presented in the listing bellow.

```
void TIM5_IRQHandler(void)      // PLL takes approx 600 ns of CPU time!
{
  GPIOE->BSRRL =  BIT_8;        //
  TIM_ClearITPendingBit(TIM5, TIM_IT_Update); // clear interrupt flag                          // 4
  PortE = GPIOE->IDR & BIT_15;  // read input signal                                           // 5
  f_VCO = ptrTable   & 0x8000;  // this is locally generated signal                           // 6
  InPat = (InPat << 2) & 0x0c;  // construct pattern for phase detector                       // 7
  if (PortE) InPat += 2;        //                                                            // 8
  if (f_VCO) InPat += 1;        //                                                            // 9
  if (PortE == f_VCO)       dTp =  0;   // if equal signals (frequencies)                    // 10
  else { if (InPat == 0x02) dTp =  1;   // if frequency too low                              // 11
         if (InPat == 0x0d) dTp =  1;   //                                                  // 12
         if (InPat == 0x01) dTp = -1;   // if frequency too high                            // 13
         if (InPat == 0x0e) dTp = -1;   //                                                  // 14
  };
  Tp += dTp;                            // correct time interval                            // 16
  if (Tp > 0x8000) Tp = 0x8000;         // but not too much                                 // 17
  if (Tp < 0x0080) Tp = 0x0080;         //                                                  // 18
  ptrTable = (ptrTable + Tp + (dTp << 8)) & 0xffff;  // update pointer to table             // 19
  GPIOE->ODR = (GPIOE->ODR & ~BIT_10) | (f_VCO >> 5);// digital out - f_VCO                 // 20
  DAC->DHR12R1 = (Table[ptrTable >> 4]) + 2048;      // analog signal -> DAC                // 21
```

```
  DAC->DHR12R2 = K >> 2;                              // frequency -> DAC                        // 22
  GPIOE->BSRRH = BIT_8;        //
}
```

As already discussed the line 4 is used to clear the flag for the pending interrupt request in timer TIM5. The input signal f_{IN} is connected to port E, pin 15, the most significant bit of the short integer read into the processor by command in line 5; the AND function is used to remove the presence of any other bits and retain the result as *PortE*. Note that the pointer into the table *ptrTable* as used in the example on DDS generation in chapter 16 is also a short integer, and consecutive values of its most significant bit form a square wave signal with a frequency that equals the frequency of the generated sine wave. The digital signal *f_VCO* representing the sine wave from the DDS is constructed in line 6 using the AND function from the MSB of the table pointer.

The integer *InPat* is constructed in lines 7 to 9. Current value of the integer *InPat* is shifted left for two places, and only the least significant four bits are retained. The least significant two bits are then modified following the values of both input signal *PortE* and the locally generated signal *f_VCO.*

Once the variable *InPat* is constructed the software decides what to do with the frequency of the DDS generator in lines 10 to 14. If both signals *f_VCO* and *PortE* are the same the variable *dTp* (delta *Tp*, for this amount the frequency should change) is set to zero. However, when signals are different, all four possibilities from the former bulleted list are checked and the variable *dTp* is set accordingly.

The variable *dTp* is next used to update the control variable *Tp* (line 16) which define the frequency of the signal f_{VCO}. It might happen that the update pushes control variable *Tp* out of the acceptable range, so *Tp* is checked and bound in lines 17 and 18.

The next statement in line 19 updates the pointer *ptrTable* to table with samples of the output signal. The last term in the sum needs to be added to ensure the stability of the PLL loop (check the theory). The last three statements take care of generating the actual signals at pins of the microcontroller. The digital signal f_{VCO} is generated at port E, bit 10, and the analog version is generated at the DAC1. The second DAC2 is used to output the control variable *Tp*, therefore the analog voltage representing the frequency of the locally generated signal f_{VCO}. When a frequency-modulated signal is used ad f_{IN}, the output of the DAC2 is the demodulated version of the input signal.

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

```
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.c"
#include "stm32f4xx_gpio.c"
#include "stm32f4xx_tim.c"
#include "stm32f4xx_dac.c"
#include "LCD2x16.c"
#include "math.h"
#include "dd.h"

int Table[4096], ptrTable, Tp = 655, dTp = 0;
char InPat = 0;
int PortE, f_VCO;

int main ()  {

  // Table init for analog output
  for (ptrTable = 0; ptrTable <= 4095; ptrTable++)
    Table[ptrTable] = (int)(1850.0 * sin((float)ptrTable / 2048.0 * 3.141592653));

  LCD_init(); LCD_string("Fin=", 0x01); LCD_string("Hz", 13);
  GPIOEinit_8to11();
```

```
  DACinit();
  TIM5init_TimeBase_ReloadIRQ(840);    // 840 == 10us == 100kHz

  while (1) {
    LCD_uInt16((int)(Tp * 100000 / 65536),0x07,1);   // display frequency
    for (int i = 0; i < 5000000; i++) {};            // waste time
  };
}
```

This listing starts with the inclusion of CMSIS files and the declaration of the variables used. The function “main()” starts 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, DAC and timer TIM5. Note the function for the configuration of port E, which is the modified version of the function given in chapter 4. Only pins 8, 9, 10, and 11 are configured as outputs, other pins are inputs. The endless while loop is used only to output the current frequency of the DDS generator to the LCD display.