22. Phase Locked Loop

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. 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 accurately represents the frequency of the input signal f_{IN}.

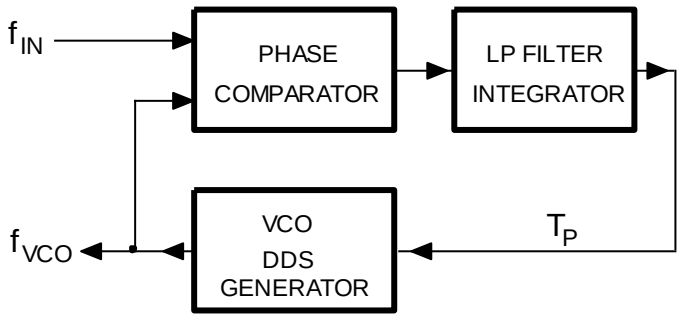

Figure 1: Graphical representation of the IIR filtering

In the following example we will program the PLL into the microcontroller obtaining a frequency meter and frequency demodulator at the same time. For those with experience in electronics: there are basically two types of phase comparators, XOR gate and a memory circuit (referred as Type I and Type II in classical PLL chip CD4046); we will implement the memory circuit, since it does not lock on harmonics of the input signal.

The frequency of the local oscillator f_{VCO} can be the same as the frequency of the input signal f_{IN}, as shown in Fig. 2. In this case the PLL is locked to the input frequency, and no further actions are needed. Arrows show the moment of sampling the two signals, and numbers their current value.

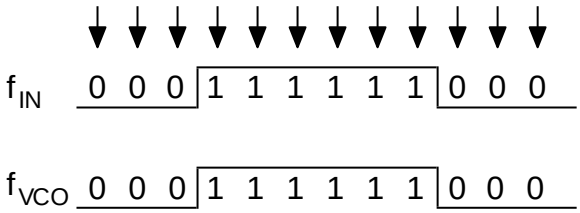

Figure 2: PLL is locked to the input signal;
f_{VCO} is equal to f_{IN} in frequency and phase, no adjustments are needed

When phases (and/or frequencies) of the two signals differ, two situations are possible. The local signal f_{VCO} can be delayed compared to the input signal f_{IN}, as shown in Fig. 3, left; in this case the frequency of the local signal f_{VCO} must be increased to align the edges. 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 to match the edges.

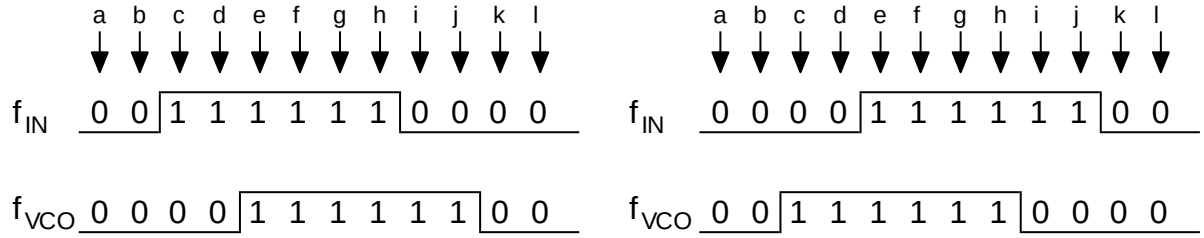

Figure 3: Two possible situations: left => frequency of signal f_{VCO} must be increased to match the phase; right => frequency of the signal f_{VCO} must be decreased to match the phase

If we use a DDS technique to generate the local signal f_{VCO} then increasing the frequency requires increasing the factor K (see chapter 16 on DDS, the frequency is proportional to factor K). From Fig. 3, left, we can deduce to increase the frequency (therefore factor K) 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. 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. 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 next two more significant bits past values of the same signals, Fig. 4. Only four bits are important; for instance, at time **c** (the third sample, Fig. 3, left) the corresponding integer number reads 0010_b.

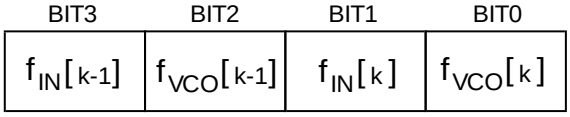

Figure 4: The arrangement of bits for the phase detection

Using this construct we can now isolate the following situations:

- When signal f_{IN} and f_{VCO} have equal values, we should not change the frequency (factor K).
- When signals have different values, we should change frequency of the DDS generator:
  - The frequency should start increasing when the constructed integer becomes 0010b, and keep increasing until both signals become equal.
  - The frequency should also start increasing when the constructed integer becomes 1101b, and keep increasing until both signals become equal.
  - The frequency should start decreasing when the constructed integer becomes 0001b, and should keep decreasing until both signals become equal.

- The frequency should also start decreasing when the constructed integer becomes 1110b, and should keep decreasing until both signals become equal.

The unit performing this functions (constructing the integer value and calculating the required frequency) replaces both the phase comparator and the LP filter / integrator in the block diagram in Fig. 1.

The program for PLL unit is given in figures 5 and 6. The main part of the program is the same as used for the DDS generator from chapter 16, and is given in Fig. 5. It starts with the declaration of variables for the DDS generator. Important variable to mention here is the variable dK which defines the change of the factor K; positive dK means the frequency of the signal f_{VCO} is increasing. The endless loop includes a write to the LCD; the frequency of the DDS generator is calculated using the known time interval between consecutive timer interrupt requests (10μs) and the width of the variable holding the table pointer (16 bits).

```
#include "stm32f4xx.h"
#include "LCD2x16.c"
#include "math.h"

int Table[4096], TablePtr, K = 655, dK = 0;
char InPat = 0;
int PortE, f_VCO;

int main ()  {

  // Table init
  for (TablePtr = 0; TablePtr <= 4095; TablePtr++)
    Table[TablePtr] = (int)(1850.0 * sin((float)TablePtr / 2048.0 * 3.14159265));

  // GPIO clock enable, digital pin definitions
  RCC->AHB1ENR |= 0x00000001;   // Enable clock for GPIOA
  RCC->AHB1ENR |= 0x00000010;   // Enable clock for GPIOE
  GPIOE->MODER |= 0x00010000;   // output pin PE08: time mark
  GPIOE->MODER |= 0x10000000;   // output pin PE14: f_out

  // LCD init
  LCD_init(); LCD_string("F in=", 0x01);

  // DAC set-up
  RCC->APB1ENR |= 0x20000000;   // Enable clock for DAC
  DAC->CR      |= 0x00010001;   // DAC control reg, both channels ON
  GPIOA->MODER |= 0x00000f00;   // PA04, PA05 are analog outputs

  // Timer 2 set-up
  RCC->APB1ENR |= 0x0001;       // Enable clock for Timer 2
  TIM2->ARR     = 840;          // Auto Reload: 8400 == 100us -> 100kHz
  TIM2->DIER   |= 0x0001;       // DMA/IRQ Enable Register - enable IRQ on update
  TIM2->CR1    |= 0x0001;       // Enable Counting

  // NVIC IRQ enable
  NVIC_EnableIRQ(TIM2_IRQn);    // Enable IRQ for TIM2 in NVIC

  // endless loop - display parameters
  while (1) {
    LCD_uInt16((int)(K * 100000 / 65536),0x08,1);     // display frequency
    for (int i = 0; i < 500000; i++) {};              // waste time
  };
}
```

Figure 5: The declaration of variables and initialization of hardware

The PLL is implemented in the interrupt function given in Fig. 6. As always all variables used in the interrupt function that should retain their values between interrupt function calls must be declared as global. The function starts by clearing the interrupt request flag in the timer TIM2, and then reads the value of the signal f_{IN}, connected to port E, bit 15, into variable f_IN. All other bits of port E are ignored. The current value of the locally generated signal f_{VCO} is also determined and stored in variable f_VCO. This is determined by taking the most significant bit of the 16-bit table pointer TablePtr. Next the integer variable gets constructed; its name is InPat. Its content is first shifted left for two places to move data from the two least significant bits to bits 2 and 3, and then least significant bits of variable InPat are filled with values of both signals f_IN and f_VCO.

Once the variable is constructed the software decides what to do with the frequency of the DDS generator. If both signals are the same the variable dK (delta K, 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 dK is set accordingly.

The variable dK is next used to update the factor K, therefore the frequency of the signal f_{VCO}. It might happen that the update of factor K pushes its value out of the acceptable range, so the factor K is next checked and bound.

The next statement updates the pointer 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 14, and the analog version is generated at the DAC1. The second DAC2 is used to generate the K factor, 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.

```
// IRQ function
void TIM2_IRQHandler(void)              // PLL takes approx 800 ns of CPU time!
{
  GPIOE->ODR |=  0x0100;                // PE08 up
  TIM2->SR &= ~0x00000001;              // clear update event flag in TIM2
  PortE = GPIOE->IDR & 0x8000;          // read input signal
  f_VCO = TablePtr   & 0x8000;          // this is locally generated signal
  InPat = (InPat << 2) & 0x0c;          // construct pattern for phase detector
  if (PortE) InPat += 2;                //
  if (f_VCO) InPat += 1;                //
  if (PortE == f_VCO)       dK =  0;    // if equal signals (frequencies)
  else { if (InPat == 0x02) dK =  1;    // if frequency too high
         if (InPat == 0x0d) dK =  1;    //
         if (InPat == 0x01) dK = -1;    // if frequency too low
         if (InPat == 0x0e) dK = -1;    //
  };
  K += dK;                              // correct time interval
  if (K > 0x8000) K = 0x8000;           // but not too much
  if (K < 0x0080) K = 0x0080;           //
  TablePtr = (TablePtr + K + (dK << 8)) & 0xffff;      // update pointer to table
  GPIOE->ODR = (GPIOE->ODR & ~0x4000) | (f_VCO >> 1);  // digital out - f_VCO
  DAC->DHR12R1 = (Table[TablePtr >> 4]) + 2048;        // analog signal -> DAC
  DAC->DHR12R2 = K >> 2;                               // frequency -> DAC
  GPIOE->ODR &= ~0x0100;                // PE08 down
}
```

Figure 3:The listing of the interrupt function - PLL

.