16. Signal generation using DDS

A technique known as Direct Digital Synthesis (DDS) will be implemented to demonstrate the use of interrupts and DAC to generate a signal with the desired frequency.

16.1. DDS technique

Consider a 16-bit wide register. Its content is increased periodically (the period is given as T_P) by a factor *K*. Obviously the time *T* needed to overflow the register equals:

$$T = T_P \cdot \frac{2^{16}}{K}$$

If we keep increasing the content of the register then the overflows will repeat at regular time intervals, and the frequency of overflows *f* can be calculated as:

$$f = \frac{1}{T} = \frac{1}{T_P} \cdot \frac{K}{2^{16}} = f_P \cdot \frac{K}{65536}$$

where f_P represents the number of increases per unit of time. If we consider the content of the register as the output signal, and the factor *K* is small, then we get a sawtooth generator; its frequency is directly proportional to factor *K*. However, we can use the content of the register as a pointer to a table, which can be filled by any waveshape during the initialization process, like one period of a sinewave. If we consider the entry from a table pointed to by the content of the register as the output signal we have a sinewave generator. The frequency of the sinewave signal can be adjusted by the same small steps, and the amplitude can be adjusted by a multiplication of the retrieved entry by a constant before it gets passed to the DAC.

It does not seem rational to prepare a huge table with 65536 entries, since the neighboring entries would be quite close in value for a slowly varying signal like sinewave. It is more rational to prepare a smaller table, and use only the bits from the upper part of the register as a pointer. The decision on the table length depends on the required precision, and we arbitrary decide to use a table with 4096 entries in this example. It would be possible to reduce the size of this table to ¼ by exploiting the symmetry of a sinewave, but we will not do this for the reason of simplicity. The block diagram of the

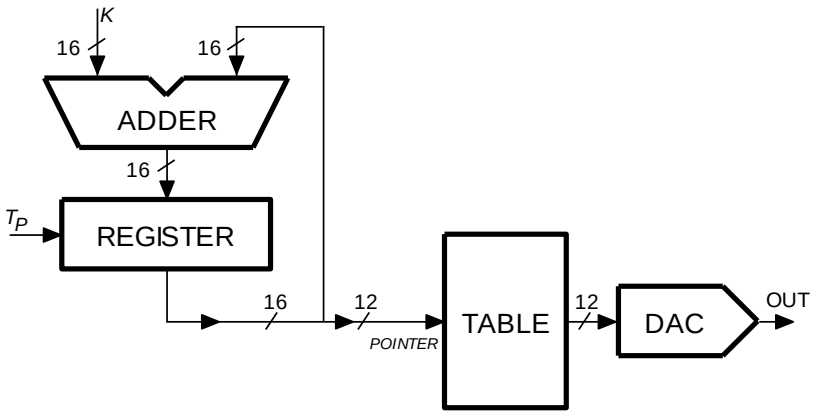

Figure 16.1: A hardware implementation of the DDS technique

hardware that could be used as a DDS generator is shown in Fig. 16.1.

The same technique is widely used in contemporary digital receiver units, and width of registers is increased to about 40 bits as is the frequency of increasing to some hundreds of MHz to achieve a stable output signal with a frequency of above 100 MHz that can be adjusted by few mHz! By expanding this technique it is very simple to generate a modulated signal (AM, FM, PM, …), and also to define new shapes of signals as well as to generate multiple signals with different frequencies. There are specialized integrated circuits available to do just this.

16.2. The software implementation

Such structure is easy to implement in a microcontroller. One only needs a timer and associated interrupt function. The timer defines the period T_P, and periodically calls an interrupt function, where a variable declared as an unsigned short integer (16 bits) gets increased by a factor *K*.

If we select the frequency of interrupt requests from the timer (f_P) at 100 kHz and a register with 16 bits, then the frequency of overflows is defined by:

$$f = 1.525879\ \mathrm{K}$$

Changing factor *K* by one changes the frequency of the generated signal for about 1.5Hz. If we intend to generate audio signals with frequencies from 20 Hz to 20 kHz then the factor K must vary from 13 to 13107. The listing of the main part of the program is given below.

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

int Table[4096], ptrTable, Am = 255, k = 655;                                   // 10

int main ()  {
int sw, Fp, j;
  for (ptrTable = 0; ptrTable <= 4095; ptrTable++)                              // 14
    Table[ptrTable] = (int)(1850.0 * sin((float)ptrTable / 2048.0 * 3.14159265));  // 15

  SWITCHinit ();                                                                // 17
  LCD_init (); LCD_string("Frq=", 0x01); LCD_string("Hz", 0x0d);                // 18
  LCD_string("Amp=", 0x41);                                                     // 19
  DACinit ();                                                                   // 20
  TIM5init_TimeBase_ReloadIRQ(840);           // 840 == 10us                    // 21

  while (1) {                                 // endless loop                   // 23
    sw = GPIOE->IDR;                                                            // 24
    if ((sw & S372) && (Am <   255)) Am++;                                      // 25
    if ((sw & S373) && (Am >     1)) Am--;                                      // 26
    if ((sw & S370) && (k  < 13107)) k++;                                       // 27
    if ((sw & S371) && (k  >    13)) k--;                                       // 28
    Fp = (int)(1.0e5 * (float)k / 65536.9);                                     // 29
    LCD_uInt16(Fp, 0x08, 1); LCD_uInt16(Am, 0x48, 1);                           // 30
    for (j = 0; j<200000; j++){};      // waste some time                       // 31
  };
}
```

The listing starts with the inclusion of necessary files. Note the file “math.c” needed for calculation of the sinusoidal wave shape. The declaration of the table ‘Table[]’ with 4096 integer elements, the pointer ‘ptrTable’ to this table, and some variables to define the amplitude *Am* and factor *K* follows in line 10. All of these must be declared as global since they are used in the interrupt function. The processor then fills the table with entries of one period of a sine wave, lines 14 and 15. The values range from -1850 to +1850, allowing for some headroom to accommodate the saturation voltage of the operational amplifier at the output from DACs. Floating point arithmetic must be used to calculate ‘sin’ function. Next is the configuration of all peripherals used: ports, DAC, timer, and LCD. The functions are taken from previous chapters and will not be discussed here again; note only that the time interval between successive interrupt request for timer TIM5 is set to 10μs (840).

The processor continues with the execution of the endless ‘while’ loop. In order to make program more flexible (and motivating) we shall make the amplitude and frequency of the generated signal changeable by means of four pushbuttons. Two can be used to increase/decrease the amplitude, and two to increase/decrease the frequency.

The processor first reads pushbuttons S370 to S373 in line 24, and then decides upon changing of values in four consecutive lines from 25 to 28. In line 25, for instance, the processor checks the state of pushbutton S372; if it is pressed and the current value of multiplier for the amplitude of the output signal is less than 255, the value of this multiplier is increased. The next line checks the pushbutton S373 and the current value of multiplication factor. If conditions are favorable, it decreases the value of multiplication factor. Similar operations are performed in lines 27 and 28 for the frequency.

Line 29 is used to calculate the frequency of the generated signal. The division factor seems strange as it should read 65536. However, this value was used due to the calibration of the readout. The commands in line 30 are used to write the calculated frequency and amplitude to the LCD screen.

The DDS technique is implemented in the interrupt function for timer TIM5, given below.

```
void TIM5_IRQHandler(void)       {
  TIM_ClearITPendingBit(TIM5, TIM_IT_Update); // clear interrupt flag                    // 2
  ptrTable = (ptrTable + k) & 0xffff;                                                   // 3
  DAC->DHR12R1 = (Am * Table[  ptrTable >> 4               ]) / 256 + 2048;             // 4
  DAC->DHR12R2 = (Am * Table[((ptrTable >> 4) + 1024) & 4095]) / 256 + 2048;            // 5
}
```

It is properly named as required by the interrupt vector table as “TIM2_IRQHandler”. The body starts by clearing the interrupt flag in timer TIM5 (line 2) and proceeds to calculating the new pointer value. A factor *K* is added to the pointer, and its value is bound to 16 bits by AND-ing it with value 65535. The pointer is used in the next line to retrieve correct element from the table, and only upper 12 bits of the pointer are used (this is the “ >> 4 ” operation). The retrieved element is multiplied by a variable Am (amplitude), and the resulting number divided by 256 to remain in the range of the DAC. The DAC can handle positive numbers only, so half of the DAC range (2048) is still added before sending the number to its destination.

In order to show the possibility of phase modulation (and also to allow further experiments with quadrature signals) the second DAC is used to generate a sine wave signal which is 90 degrees out of phase with the already generated signal. This is accomplished by retrieving the element from the table which is for ¼ of the table length away from the current pointer. Upper 12 bits of the pointer are taken, and 1024 is added to it. The sum might point outside of the table, so the sum is corrected to remain within the bounds of the table by AND-ing it with the pointer to the last valid element (4095), the rest is the same as for the first DAC. The complete interrupt function takes about 500ns to execute, much less than the available 10μs. The frequency of interrupts could be increased to allow the generation of smoother signals.