12. Periodical interrupts and ADC/DAC

The knowledge from previous chapters will be used here to prepare a program, which can periodically start a conversion at the ADC, wait for the result and pass the result to DAC.

The previously given demonstration program for ADC shows the initialization needed to use two of the built-in ADCs to sample two analog input signals. The results were displayed at the LCD. Here the results will be passed to the DAC giving a replica of the input signal, quantized in time. The picture in Fig. 1 shows one of the input signals and the corresponding quantized version at the output from a DAC. Similar program could be prepared using two former examples on ADC and DAC and inserting a software delay loop within the endless loop in regular program to define the time interval between two successive loop repetitions. However, using the software to define the time interval is wasting the processor time, and should be avoided.

Figure 1: The quantized-in-time version of the input signal

It has been shown that time intervals can be defined using a timer, and that the reload events of a timer can trigger interrupts. It is therefore only natural to define time intervals between two successive measurements by a timer, and to use interrupt function to start the conversion at ADC, wait for the result, and pass the result to DAC.

The complete listing of the program is given in Fig. 2. The program starts with a set of initializations, which were copied from previous examples:

- Ports are initialized first. Clocks are enabled, and three outputs are defined for two demo signals and a LED. One of the demo signals is used to mark the execution of the endless loop in the regular part of the program (bit 9, port E), and one is used to signal the execution of the interrupt function (bit 8, port E). The LED is used to confirm that the endless loop is running; when a pushbutton S370 is pressed, the LED is turned on.

- Next digital to analog converters are initialized. The initialization statements are copied from demo program “DACtest.c”.
- Next analog to digital converters are initialized. The initialization is copied from demo program “ADCtest.c”.
- Interrupt controller NVIC is initialized to receive interrupt requests from timer TIM2.
- Next timer TIM2 is initialized to generate interrupt requests every 100μs. The initialization statements are copied from demo program “Timer5_IRQ.c”, providing that the timer TIM2 is used instead of timer TIM5 and that the time interval is changed.

```c
#include "stm32f4xx.h"

int main ()  {
  // 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 |= 0x00040000;   // output pin PE09: toggle
  GPIOA->MODER |= 0x00001000;   // output pin PA06: LED D390

  // 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

  // ADCset-up
  RCC->APB2ENR |= 0x00000100;   // clock for ADC1
  RCC->APB2ENR |= 0x00000200;   // clock for ADC2
  ADC->CCR      = 0x00000006;   // Regular simultaneous mode only
  ADC1->CR2     = 0x00000001;   // ADC1 ON
  ADC1->SQR3    = 0x00000002;   // use PA02 as input
  ADC2->CR2     = 0x00000001;   // ADC1 ON
  ADC2->SQR3    = 0x00000003;   // use PA03 as input
  GPIOA->MODER |= 0x000000f0;   // PA02, PA03 are analog inputs

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

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

  // endless loop - indefinite
  while (1) {
    if (GPIOE->IDR & 0x0001) GPIOA->ODR |=  0x0040;   // LED on
    else                     GPIOA->ODR &= ~0x0040;   // else LED off
    GPIOE->ODR |=  0x0200;      // PE09 up
    GPIOE->ODR &= ~0x0200;      // PE09 down
  };
}

// IRQ function
void TIM2_IRQHandler(void)      // PASS takes approx 500ns of CPU time!
{
  GPIOE->ODR   |=  0x0100;      // PE08 up
  TIM2->SR     &= ~0x00000001;  // clear update event flag in TIM2
  DAC->DHR12R1  = ADC1->DR;     // pass ADC -> DAC, also clears EOC flag
  DAC->DHR12R2  = ADC2->DR;     // pass ADC -> DAC, also clears EOC flag
  ADC1->CR2    |= 0x40000000;   // simultaneous Start Conversion
  GPIOE->ODR   &= ~0x0100;      // PE08 down
}
```

Figure 2: A listing of the program to periodically sample input signals and pass them to the DAC

Following the initialization the program enters the endless loop to check the status of the pushbutton S370 and turn-on the LED on demand, as well as to generate a pulse for every execution of the loop.

The interrupt function starts with setting of the bit 8, port E, and ends with clearing the same bit. The corresponding signal can be checked using an oscilloscope to verify the execution of the interrupt function and to determine the time needed to execute it.

The body of the interrupt function starts by resetting of the interrupt request flag, as shown already in the demo program "Timer5_IRQ.c". Next the contents of both ADC data registers (ADCx_DR) are copied to two data holding registers (DAC_DHR12Rx) in DACs, and the next conversion is started by setting of the bit 30 in ADC control register 2 (ADC1_CR2).

Such implementation is straightforward, but may not be the best. It is known from theory than it is very important to take samples of the input signal at exact time intervals, or the signal to noise ratio of results will be impaired. The exact period is not guaranteed in the former example, since the processor can start executing the interrupt function only after it finishes the execution of the current instruction, and instructions can take different time to execute. Furthermore, the execution of the interrupt function for timer TIM2 itself might be delayed in a more complex program utilizing more than one interrupt function. It is therefore better to trigger a start of conversion directly by a timer without the intervention of the software, and use interrupt function only to copy the result of conversion to the DAC. Such interrupt function shall be initiated by end of conversion signal coming from the ADC, which can be used to trigger an interrupt. The listing of such version of the program is given in Fig. 3.

Most of the initialization statement are the same and are not repeated in listing in Fig. 3. Only the difference is given:

- Three statements are added to the initialization of the ADC regarding the source of the start conversion (SC) signal.
    - This time the signal SC is supplied by the timer TIM2, and this is selected by setting bits EXTSEL (ADC1_CR2) at the control input of the multiplexer within the ADC1 (see Fig. 1, Analog to digital converters).
    - External SC signals are enabled by setting the bit EXTEN (ADC1_CR2).
    - The ADC is enabled to issue an interrupt request signal by setting the least significant bit in control register 1 (ADC1_CR1).
- The interrupt controller must be allowed to respond to interrupts from the ADC, and the call to NVIC_EnableIRQ function is modified accordingly.
- The initialization of timer TIM2 is changed, and the timer is not allowed to issue interrupt requests. The formerly present statement (TIM2->CR2 =W 0x0020;) has been removed.

The interrupt function lacks two statements:

- The interrupts requests are not issued by timer anymore, so there is no need to clear the interrupt request flag within the timer; the corresponding statement has been removed. A flag to memorize the interrupt request from the ADC is present within the ADC, but gets cleared automatically when the content of the data register within the ADC is read by the software, so there is no need to insert a new statement to clear the flag.
- The statement to start the conversion (ADC1->CR2 |= 0x40000000;) has been removed.

```
#include "stm32f4xx.h"

int main ()  {
  // GPIO clock enable, digital pin definitions
  // the same as above and not shown here again

  // DAC set-up
  // the same as above and not shown here again

  // ADC set-up
  // the same as above and not shown here again

  ADC1->CR2    |= 0x06000000;   // use TIM2, TRG0 as SC source
  ADC1->CR2    |= 0x10000000;   // Enable external SC, rising edge
  ADC1->CR1    |= 0x00000020;   // Enable ADC Interrupt for EOC

  // NVIC IRQ enable
  NVIC_EnableIRQ(ADC_IRQn);     // Enable IRQ for ADC in NVIC

  // Timer 2 set-up
  RCC->APB1ENR |= 0x0001;       // Enable clock for Timer 2
  TIM2->ARR     = 8400;         // Auto Reload value: 8400 == 100us
  TIM2->CR2    |= 0x0020;       // select TRGO to be update event (UE)
  TIM2->CR1    |= 0x0001;       // Enable Counting

  // endless loop - indefinite
  // the same as above and not shown here again
}

// IRQ function
void ADC_IRQHandler(void)       // PASS takes approx 400ns of CPU time!
{
  GPIOE->ODR   |=  0x0100;      // PE08 up
  DAC->DHR12R1  = ADC1->DR;     // pass ADC -> DAC, also clears EOC flag
  DAC->DHR12R2  = ADC2->DR;     // pass ADC -> DAC, also clears EOC flag
  GPIOE->ODR   &= ~0x0100;      // PE08 down
}
```

Figure 3: A listing of the program to do the same as listing in Fig. 2, but using timer as source for a start conversion signal