11.Interrupts and Timer

The interrupt mechanism is very important for a programmer, and is implemented in every microcontroller. An interrupt request can come from many sources, here it is generated periodically by Timer 5. Such approach can be used for instance for periodical sampling or generating of a signal, or for periodical switching between tasks in a multitasking system.

11.1. The hardware – NVIC, the Nested Vectored Interrupt Controller

The discussion on interrupt processing hardware was given in chapter 10 (Interrupts and ports). Here an example of the use of timer and interrupt processing will demonstrate how to periodically interrupt the execution of the regular program in such a way that the processor can do some work at precisely defined time intervals. In our case the processor will generate two consecutive pulses at port E, bit 8, for every interrupt request.

The block diagram representing the behavior of the microcontroller for this experiment is given in Fig. 11.1. The timer TIM5 is clocked by a signal CLK with frequency of 84MHz, and is configured to make periodical pulses, the period T being 1ms. Each of these pulses is passed to Nested Vectored interrupt Controller NVIC to issue an interrupt to the processor, and processor in turn makes a double pulse at port E, pin 8.

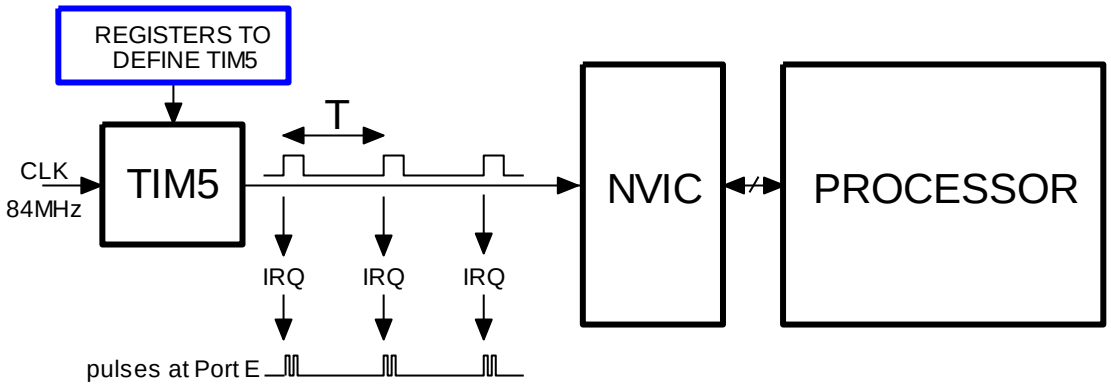

Figure 11.1: The chain used to implement periodic interrupt requests - simplified

The timer TIM5 has the same capabilities and block structure as previously described timer TIM2; it consist of a 32-bit counter, several multiplexors to select the clock source and additional hardware to control the counting. The clock signal CLK is one of the internal clocks of the microcontroller, and is connected to the input of the counter through the set of multiplexors by default.

The timer TIM5 counts up to a number stored in its auto-reload register and then resets back to zero; the return to zero is called an update event, and can be configured to trigger an interrupt. Setting the content of the auto-reload register to 84000 and using the clock signal with the frequency of 84 MHz assures the time interval T to be 1 ms. The controller NVIC must be configured to receive pulses (update events) from the timer TIM5.

11.2. The software – the making of double pulses once per millisecond

The complete configuration for the timer TIM5 and the NVIC can be put into a function and named "TIM5init_TimeBase_ReloadIRQ()". This function is called prior to the use of timer interrupts, and takes the time interval between consecutive interrupts as an argument; note that integer variable is 32 bits in length. The listing of the function is given below.

```
void TIM5init_TimeBase_ReloadIRQ (int interval)  {
TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;                                  // 2

  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM5,  ENABLE);                             // 4

  TIM_TimeBaseInitStructure.TIM_Prescaler = 0;                                      // 6
  TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up;                   // 7
  TIM_TimeBaseInitStructure.TIM_Period = interval;                                  // 8
  TIM_TimeBaseInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;                       // 9
  TIM_TimeBaseInitStructure.TIM_RepetitionCounter = 0;                              // 10
  TIM_TimeBaseInit(TIM5, &TIM_TimeBaseInitStructure);                               // 11

  NVIC_EnableIRQ(TIM5_IRQn);                 // Enable IRQ for TIM5 in NVIC         // 13
  TIM_ITConfig(TIM5, TIM_IT_Update, ENABLE); // Enable IRQ on update for Timer5     // 14

  TIM_Cmd(TIM5, ENABLE);                                                            // 16
}
```

The timer is configured using a CMSIS function "TIM_TimeBaseInit()", given in the source file "stm32f4xx_tim.c". The same function was already used in chapter on counting pulses using the timer TIM2. This function requires the use of data structure named 'TIM_TimeBaseInitStructure' described in the header file "stm32f4xx_tim.h", lines 55 to 78. The data structure is declared in the second line of the function, and the clock for the timer TIM5 is enabled in line 4. The members of the data structure are initialized in lines 6 to 10 as follows.

- The first member '.TIM_Prescaler' is set to 0, since we do not want to reduce the frequency of the clock signal.
- The second member '.TIM_CounterMode' is initialized to 'TIM_CounterMode_Up' to make the counter advance on clock pulse.
- The third member '.Tim_Period' is the number to be stored into the auto-reload register; we could use value 84000 in this example to define the time interval T to 1ms. However, in order to make this function more flexible the value is not fixed, but is sent to this function as an argument.
- The fourth member '.TIM_ClockDivision' is set to 0 since we do not want to reduce the frequency of the clock signal.
- The last member '.TIM_RepetitionCounter' is irrelevant for this example, but nevertheless initialized to 0.

The timer TIM5 is then configured by calling the function "TIM_TimeBaseInit()" in line 11; the timer has not been started yet. Next comes the configuration of the controller NVIC, which must be enabled to receive interrupt request pulses from timer TIM5. This is done by a call in line 13. The last step in configuration is to allow timer TIM5 to use the update events as a source of the interrupt request pulses, and this gets configured by a call to function "TIM_ITConfig()" in line 14. The full description of this function is given in the source file, lines 2372 to 2389, and possible names of events within the timer to be used as interrupt requests are defined in the header file, lines 553 to 560.

The last thing to do is to enable timer TIM5 to commence the counting, this is done in line 16.

The interrupt function to be executed on interrupt must be prepared and written as a stand-alone function taking and returning no arguments. In our case four statements are needed to toggle port E, bit 8, high-low-high-low. Additionally, the update event of timer TIM5 toggles a flag stored in status register of timer TIM5, and must be cleared within the interrupt function to prevent immediate repetition of the same interrupt request. The complete listing of the function is given below.

```
void TIM5_IRQHandler(void)       {
int i;
  TIM_ClearITPendingBit(TIM5, TIM_IT_Update); // clear interrupt flag                    // 3
  GPIOE->ODR |=  BIT_8;        for (i=0; i<10; i++) {};                                 // 4
  GPIOE->ODR &= ~BIT_8;        for (i=0; i<10; i++) {};                                 // 5
  GPIOE->ODR |=  BIT_8;        for (i=0; i<10; i++) {};                                 // 6
  GPIOE->ODR &= ~BIT_8;                                                                 // 7
}
```

Note the name of the interrupt function “TIM5_IRQHandler”, it must be the same as specified in the interrupt vector table, file “startup_stm32f4xx.s”, line 133.

The order of events is then as follows. When the timer TIM5 content reaches the predefined value stored in the reload register the content of the counter register CNT returns to zero triggering a reload event, and the counting continues from zero on. The reload event turns on a flag in status register of timer TIM5, and is simultaneously passed to the controller NVIC as a signal to trigger an interrupt request. Since the controller NVIC is enabled to respond to this particular interrupt request, it forces the processor to interrupt the execution of the regular program and starts executing the relevant interrupt function. Within the interrupt function the processor toggles the port E, bit 8, four times to make two consecutive pulses and clears the update event flag in the status register of timer TIM5 to acknowledge the execution of the interrupt function. After this the processor continues with the execution of the regular program as if nothing had happened.

The listing of the demo program is given below, it must be complemented with the two functions listed above and a function to initialize port E, pins 8 to 15 as outputs, see chapter 4 for details.

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

void main(void) {

  GPIOEinit ();                                                                         // 9
  TIM5init_TimeBase_ReloadIRQ (84000);           // 84000 == 1ms                       // 10

  while (1) {};
}
```

The listing starts with the include statements, here all relevant source files are added to the user program. The main section of the listing commences with a call to initialize port E and a call to initialize timer TIM5 and the controller NVIC, and then enters an infinite loop where the processor simply waits for an interrupt. On interrupt the processor executes the interrupt function “TIM_IRQHandler()”, and then returns to the infinite loop.