9. Timer – counting pulses

The microcontroller houses several timers that can be used for different purposes. This example shows the use of timer for counting of pulses that are applied to the pin of the microcontroller.

9.1. The hardware – a Timer

There are 14 timers included in the STM32F407VG. Their key properties are listed in Fig. 9.1 (DM00037051, pg. 30, 31). Detailed properties and the description of each group of timers can be found in RM0090. We will use Timer2 in this experiment to count pulses applied to a pin on the microcontroller and show current content of the counter on the LCD. There will be three pushbuttons used to initialize the content of the counter, to start counting, and to stop counting.

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complementary output	Max interface clock (MHz)	Max timer clock (MHz)
Advanced-control	TIM1, TIM8	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	Yes	84	168
General purpose	TIM2, TIM5	32-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM3, TIM4	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM9	16-bit	Up	Any integer between 1 and 65536	No	2	No	84	168
	TIM10, TIM11	16-bit	Up	Any integer between 1 and 65536	No	1	No	84	168
	TIM12	16-bit	Up	Any integer between 1 and 65536	No	2	No	42	84
	TIM13, TIM14	16-bit	Up	Any integer between 1 and 65536	No	1	No	42	84
Basic	TIM6, TIM7	16-bit	Up	Any integer between 1 and 65536	Yes	0	No	42	84

Figure 9.1: The timers included in STM32F407 and their key properties

Timers TIM1 and TIM8 use 16-bit counters and are the most complex timers of all timers included in the microcontroller. Timers TIM2 and TIM5 are 32-bit versions of TIM1/TIM8, but have less

hardware included and therefore less options. Timers TIM3 and TIM4 are 16-bit versions of TIM2/TIM5. Timers TIM9, TIM10 and on are stripped-down versions of timer TIM4, and so on.

A simplified block diagram for timer TIM2 is given in Fig. 9.2 (reference manual RM0090, Figure 134, page 577). The content of the counter CNT (yellow) is available in register TIM2_CNT.

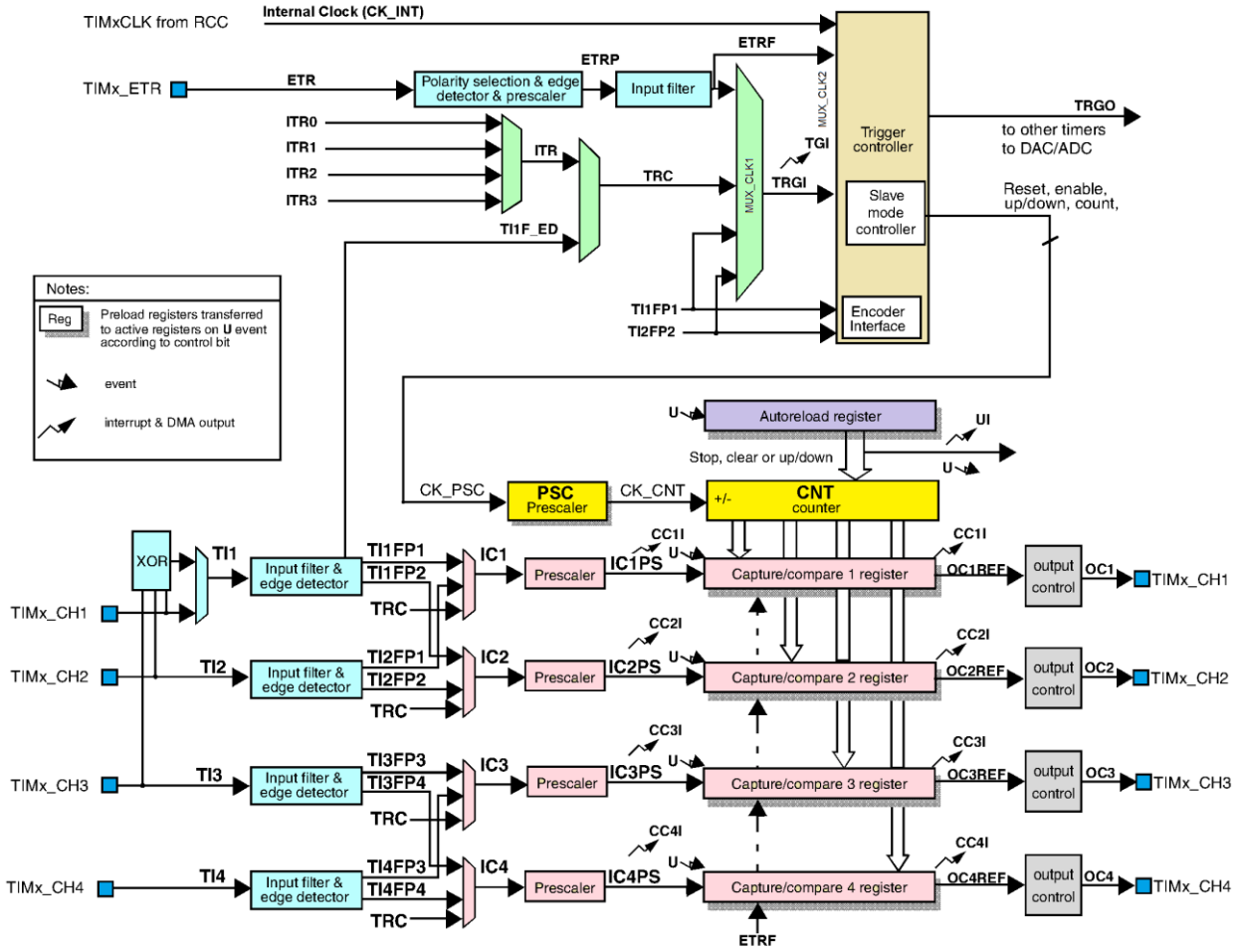

Figure 9.2: The block diagram for timers TIM2 and TIM5. Counter: yellow boxes, clock select: light green boxes, input signals conditioning: light blue boxes, output control: grey boxes.

The counter advances on transitions of the clock signal CK_PSC. This clock signal comes through a set of modules (light blue boxes and light green multiplexors) at the top of Fig. 9.2. Several signals are offered to become the source of the clock signal, and multiplexors select one of them as the 'CK_PSC'. The user can configure multiplexors to pass signals connected to pins marked TIMx_CH1 to TIMx_CH4 or TIMx_ETR. Alternatively, multiplexors can be configured to pass internally generated signals ITR0 to ITR3 or even internal clock signal CK_INT as a clock CK_PSC. The counter can also be fed by some internally derived signals like TI1FP1, TI2FP2, and then some. Details are given in reference manual RM0090, chapter 18.

Due to the availability of pins at the STM32F4-Discovery board and the limitations imposed by the BaseBoard we will use pin 15, port A, and designate it as a source of signal TIM2_ETR, which will in turn be routed through multiplexors to become the clock signal CK_PSC. This pin is available at the connector K406 at the edge of the BaseBoard.

The timer block houses additional sub-blocks, which will not be used in this experiment on counting. These blocks include prescaler modules, capture and compare modules, output modules and reload logic; this will be used in subsequent experiments.

9.2. The software to test the counting with Timer 2

Several bits distributed along registers within the block of Timer 2 are responsible for the configuration of the block and with this for its operation. Again, CMSIS functions stored in the source file “stm32f4xx_tim.c” are used to ease the correct configuration, and data structures and definitions are taken from the header file “stm32f4xx_tim.h”.

Two blocks are to be configured, these are the port A to pass the signal from its pin 15 to the Timer 2, and the Timer 2 to accept this signal, route it to the input of the counter and count it.

Let us deal with the configuration of the port first. A pin of a port is normally used to pass digital signals into or out of the microcontroller. It can be configured for analog signals, as we have seen in chapter on DAC and ADC. But a pin can also be used to pass special signals to or from peripherals which are built into the microcontroller; in this case we say the pin is assigned to an alternate function. Not all pins can pass signals from all peripherals, every pin can be assigned one of 16 alternate functions. The complete list of alternate functions available for every pin is given in table 9, pg. 60 and on, in the datasheet of the microcontroller (DM00037051). From this table one can see that port A, pin 15, can be used as a regular IO for digital signals, but it can also be used as input signal for Timer 2, ETR or as channel 1 (TIM2_CH1) if alternate function AF1 is assigned to this pin. Alternatively, this pin can be used to pass signals for SPI communication if alternate functions AF5 or AF6 are selected.

The complete code for the configuration of pin 15, port A, to serve as an input pin for signal ETR is put in a function “GPIOAinit_TIM2_ETR()” listed below.

```
void GPIOAinit_TIM2_ETR(void)   {
GPIO_InitTypeDef          GPIO_InitStructure;                                          // 2

  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);                                // 4

  GPIO_PinAFConfig(GPIOA, GPIO_PinSource15, GPIO_AF_TIM2);          // PA15:TIM2_Ch1/ETR  // 6

  GPIO_InitStructure.GPIO_Pin          = GPIO_Pin_15;                                  // 8
  GPIO_InitStructure.GPIO_Mode         = GPIO_Mode_AF;                                 // 9
  GPIO_Init(GPIOA, &GPIO_InitStructure);                                               // 10
}
```

The structure of this function is common for a configuration of a pin. The second line declares the data structure, and the fourth line enables the clock for port A. The sixth line allows pin 15 (‘GPIO_PinSource15’) of port A (‘GPIOA’) to be used as alternative function for Timer 2, therefore the ETR or CH1. All it takes is to call the function “GPIO_PinAFConfig()” with these parameters. The pin is not yet put into the alternate function mode, only one of all possible alternate functions has been selected. The rest of the function changes the mode for the selected pin from normal to alternate function. The data structure is initialized in lines 8 and 9, note the ‘GPIO_Mode_AF’ in line nine. Finally, the pin gets initialized by the call of function “GPIO_Init()” in line 10.

Now for the configuration of Timer 2 to accept the signal from port A and count it. All code is packed into a function “TIM2init_counter()” listed below.

```
void TIM2init_counter(void) {       // counting from ETR
TIM_TimeBaseInitTypeDef   TIM_TimeBaseInitStructure;                                 // 2

  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,  ENABLE);                              // 4

  TIM_TimeBaseInitStructure.TIM_Prescaler = 0;                                       // 6
  TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up;                    // 7
  TIM_TimeBaseInitStructure.TIM_Period = 100000;                                     // 8
  TIM_TimeBaseInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;                        // 9
```

```
   TIM_TimeBaseInit(TIM2, &TIM_TimeBaseInitStructure);                                    // 10

   TIM_ETRClockMode2Config(TIM2, TIM_ExtTRGPSC_OFF, TIM_ExtTRGPolarity_NonInverted, 0x00);      // 12

   TIM_Cmd(TIM2, ENABLE);                                                                 // 14
}
```

The configuration of Timer 2 follows the standard procedure briefly described in the source file “stm32f4xx_tim.c”; first the clock for the timer is enabled in line 4, second the timer is configured for counting direction, period and scaling in line 10; a CMSIS function call is used for easy configuration. Last the source of the signal to be counted is specified in line 12. A single data structure is needed for the second step, and it is declared at the beginning of the function, line 2.

The second step needs further discussion. The function “TIM_TimBaseInit()” is given in the source file “stm32f4xx_tim.c”, line 288, and the data structure to go with it is given in the header file “stm32f4xx_tim.h”, line 55 to 78.

- The first member ‘.TIM_Prescaler’ gives the prescaler value to be implemented in the block PSC in front of the counter CNT, see Fig. 9.2. The frequency of the input signal to the prescaler is simply divided by the value written as the prescaler value, or, in other words, if the value is 4 then only every fourth pulse will be passed to the input of the counter CNT. The prescaler value can be set to anything between 0 and 65535; since we want to count all pulses this member is initialized to 0.
- The second member ‘TIM_CounterMode’ defines the order of counting, is our case up-counting is selected by initializing this member with option ‘TIM_CounterMode_UP’. Other options are listed in the header file, lines 317 to 321.
- The third member ‘TIM_Period’ defines the content of the reload register; the content of the counter will reset back to 0 once the number in the reload register is reached. When a clock signal is connected to the input of the counter, this value specifies the period of counting. In out example we want to count up to a reasonable number before the content of the counter is reset back to 0, so this member is initialized to 100000.
- The fourth member ‘TIM_ClockDivision’ configures another divider of clock signal to be 1, ½ or ¼. In our case the divider is skipped by initializing this member to 0 (‘TIM_CKD_DIV1’).
- There is also a fifth member ‘TIM_RepetitionCounter’, but this option is not used in this experiment and the initialization of this member is not needed.

The third step to select the clock source also requires some additional discussion; the details are given in reference manual RM0090, chapter 18.3.3, pg. 587. Since we use ETR pin as a source of the signal to count, this requires the use of “External Clock mode 2”. There is a function in the CMSIS library for every possible clock source, and the one for the use of ETR signal as a clock is named “TIM_ETRClockMode2Config()”. The function requires following arguments.

- First argument is the name of the counter to be configured, TIM2 in this case.
- Second argument configures yet another prescaler inserted in the path of ETR signal. This prescaler can divide the frequency of the input signal by 2, 4 or 8, and it can also be disabled as in our case, where this parameter is set to ‘TIM_ExtTRGPSC_OFF’.
- The third argument configures the polarity of the ETR signal that is used to increment the counter, in this case it is set to true (‘TIM_ExtTRGPolarity_NonInverted’).
- The last argument configures the filter inserted into the path of ETR signal. Here we do not to filter the signal, and the value is set to 0 (it can range from 0 to 15), see the reference manual RM0090 for details.

The demo program is written to initialize the port and timer TIM2, and then enter the infinite loop where it periodically reads the content of the timer and writes it on the LCD. The complete program is given in the listing below.

```
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.c"
#include "stm32f4xx_gpio.c"
#include "stm32f4xx_tim.c"                                                    // 4
#include "LCD2x16.c"

int main ()  {
  GPIOAinit_TIM2_ETR();               // GPIO clock enable, digital pin definitions      // 8
  TIM2init_counter();                 // Timer 2 as counter: ETR input                  // 9
  LCD_init();                         // Init LCD                                       // 10
  LCD_string("TIM2 counter ETR", 0);                  // write title                    // 11

  while (1) {
    LCD_uInt32(TIM2->CNT, 0x40, 0x01);                // show counts                    // 14
    for (int i = 0; i < 1000000; i++)   {};           // waste time                     // 15
  };
}
```

The program starts with a set of include statements, note the fourth line to include the source file “stm32f4xx_tim.c”. The main function starts with three calls to initialize port A, Timer2 and LCD screen. The two functions listed above should be included in order for the program to compile. The last line before entering the infinite loop (11) writes some introductory text to the LCD screen.

Within the infinite loop the microcontroller simply reads the content of the counter register in timer TIM2 (32 bits), and writes the value on the LCD. Please note that the content of the counter could also be read by CMSIS function for better portability, but is not implemented here. The execution of the loop is slowed down by inclusion of an empty for loop.