ELEC-240 Lab3

Basic Timer Interrupts on the STM32F429 Nucleo-144 Development Board

1 Introduction

In previous labs you have seen how to generate a delay using `blocking' techniques where the micro-controller simply wastes time performing a pointless task such as incrementing a variable to a very large value. This lab exercise introduces how to use a hardware timer interrupt to achieve the same result.

All micro-controllers have at least one hardware timer which, when enabled, counts without any intervention from the program counter and can generate an interrupt when the count reaches a pre-determined value (overflows).
Using interrupts is much more efficient as it allows the micro-controller to perform other tasks while the delay is being generated, or if there is no other tasks to perform it can be put into low power mode during this time thus greatly decreasing power consumption. Timer interrupts are also much more accurate which makes them ideal for discrete-time sampling Digital Signal Processing systems. Blocking delays cannot produce such accuracy as they can be influenced by other processes on the micro-controller.

1.1 Learning Outcomes

- ❖ By the end of this lab exercise you should be able to:
  - ✓ Demonstrate an understanding of hardware timer operation and the advantages over blocking delays
  - ✓ Calculate the necessary auto-reload value for a desired delay period
  - ✓ Configure the timer registers to implement the hardware timers on the micro-controller and demonstrate an understanding of each register’s operation

2 Hardware

The hardware timer derives its clock source from the main core clock. Optimally the core clock runs at 180MHz with the PLL enabled. Assuming no software division has been applied the following is true:

PLL Config	**Optimal Core Clock Speed (F_{AHB})**	**Timer clock Speed (F_{APB})**
Disabled	16Mhz	16MHz (div 1)
Enabled	180Mhz	90MHz (div 2)

and the value to be loaded into the Auto Reload Register (**ARR**) for a desired delay time is given by

$$ARR = \frac{F_{APB}}{PSC+1} \times T_{delay} \qquad (1)$$

Register details are available in Table 1.

The Timer works by incrementing a binary counter on every clock pulse of the Advanced Peripheral Bus (**APB**) clock. The instantaneous value of this counter is reflected in the count (**CNT**) register. When this value becomes equal to the **ARR** register value an Update event occurs. The counter is reset to zero and an Update Interrupt is generated if enabled in the timer configuration registers.
Timer2 & Timer5 have a 32bit ARR registers while Timer3 & Timer4 have 16bit ARR registers. A 32bit Timer can count to much higher values and hence produce longer delays for a given input clock frequency. If the desired delay requires the ARR value to exceed the maximum value then incoming clock frequency **F_{APB}** can be divided using a Pre-SCaler (**PSC**). Each timer has its own pre-scaler so each one can be divided by different amounts if necessary.

3 Software

There are several registers involved when configuring the hardware timer as it can control many peripheral features. This exercise uses Timer 2 which is one of several general-purpose hardware timers available on this micro-controller. Section 18.4 of the RM0090 Reference manual in Table 2 outlines the various registers used to configure the general purpose hardware timers.
The basic steps for enabling a timer interrupt are:

1. The timer clock must be enabled in the peripheral clock enable register
2. The update interrupt enable must be enabled in the timer DMA/Interrupt enable register
3. The pre-scaler and auto-reload registers must be loaded as using Equation (1) to generate the correct time delays
4. The global interrupt enable must be set in the interrupt controller register
5. The timer counter must be enabled in the timer control register

Please see Table 1 below for the register locations in the Reference Manual.

3.1 Interrupts

When any interrupt is triggered the program counter will immediately jump to the Interrupt Service Routine (ISR) associated with that interrupt source and execute the code contained therein. An interrupt flag for that interrupt is also asserted in the Status Register (**SR**) of the peripheral which generated the interrupt. When the code inside the ISR has been run the program counter will then return to the section of code which was being run at the moment the interrupt was invoked.

Before exiting any ISR the afore-mentioned interrupt flag must be cleared by software unless it is specifically stated in the Reference Manual that the flag is auto-cleared by hardware.
Failure to clear the interrupt flag will result in no further interrupts being generated by the peripheral even though it is still running.

Register Name	Reference Manual Section
Hardware Clock Enable Register (AHB1ENR)	6.3.10
Peripheral Clock Enable Register (APB1ENR)	6.3.13
Interrupt Controller Vector table (NVIC)	12.3
Timer Control Register (CR1)	18.4.1
Timer DMA/Interrupt Enable Register (DIER)	18.4.4
Timer Status Register (SR)	18.4.5
Timer Count Register (CNT)	18.4.10
Timer Prescale Register (PSC)	18.4.10
Timer Auto-Reload Register (ARR)	18.4.10

Table 1: Table of relevant control registers for generating a basic hardware timer interrupt
The Reference Manual can be accessed through Table 2.

Task 1

The example code available on the DLE uses a hardware timer interrupt (using Timer 2) to blink the Green LED (PB0) on and off every 100ms. Compile and download the code to the board and confirm using the oscilloscope that the pulse width is 100ms. This can be done by probing PB0 where it is available on the header.
If you're using a Pico-Scope please refer to Section 3.2.

Task 2

Modify the code to create separate functions that blink the LED every:

a) 1ms
b) 1s

confirm each using the oscilloscope. ***There should be no blocking delays!***

Task 3

Modify the code to create separate functions that use Timer 3 instead of Timer 2 and blink the LED every:

a) 100ms
b) 1ms
c) 1µs
d) 1s

confirm each using the oscilloscope. ***There should be no blocking delays!***
HINT:

- Copy the ***Init_Timer2()*** and rename ***Init_Timer3().***
- Replace all instances of ***TIM2*** with ***TIM3*** and adjust the **ARR** and **PSC** for 16bit operation
- Update Timer3 global interrupt enable by adjusting value ***NVIC->ISER[0]*** to set bit ***TIM3_IRQn*** (given in file ***stm32f429xx.h***)
- Create a ***TIM3_IRQHandler()***

Task 4

a) Modify the code to make the three LEDs blink together at different speeds using multiple timers interrupts. It is not necessary to confirm this operation with the oscilloscope, but the three LEDs should have visibly different blinking speeds. ***There should be no blocking delays!***

b) Ensure you achieve the above with an empty ***While(1)*** loop in the ***main();***
We can now reduce power by putting the CPU into standby mode whilst Waiting For Interrupts to occur by inserting ***__WFI();*** in the empty while loop. (note double underscore)

- If you are able, measure the change in current to confirm. This will require you to power the board from an external power supply (not the USB).

Task 5

Whilst interrupts are very good for long delays, they can be inefficient for creating very short delays. However, we can still use Timers to create **accurate** short delays.

a) Stop the Timer generating an interrupt by commenting out the line

- NVIC->ISER[0]|=(1u<<29);   //timer 3 global interrupt enabled

Note: the timer is still running just not generating interrupts!

b) Create a ***WaiT3(int delay_us)*** function that:

- Reads the Timer3 counter to find start value.
  start = TIM3->CNT;
- Create a loop that continues to read the Timer counter again and subtracts the ***start*** value until the desire number of timer clock cycles have occurred to achieve the desired ***delay_us***
- Prove by calculation why the **ARR** should be set to ***0xffff*** to allow for the count to overflow.

Task6

If you can't use a 32-bit timer, then there is the core cycles counter. Just enable it once and then read the value from DWT->CYCCNT.

a) Use the following code to initialise the Core Cycle Counter in ***Main()***

```
CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk;
DWT->CYCCNT = 0;
DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk;
```

Note that as it returns the elapsed processor cycles, it will overflow in a couple of seconds.

b) Create the following function and test.

```
void wait_us(uint32_t n) {
        uint32_t start, cnt;
        start = DWT->CYCCNT;
        cnt = n * (SystemCoreClock/1000000);
        while (((DWT->CYCCNT - start)) < cnt);
}
```

Task 7.

Copy **ALL** your appropriately named delay routines to a ***Delay.C*** file and create ***Delay.h*** that contains declarations for ALL your functions.
You will need these in future labs

Task 8. Answer the following Questions

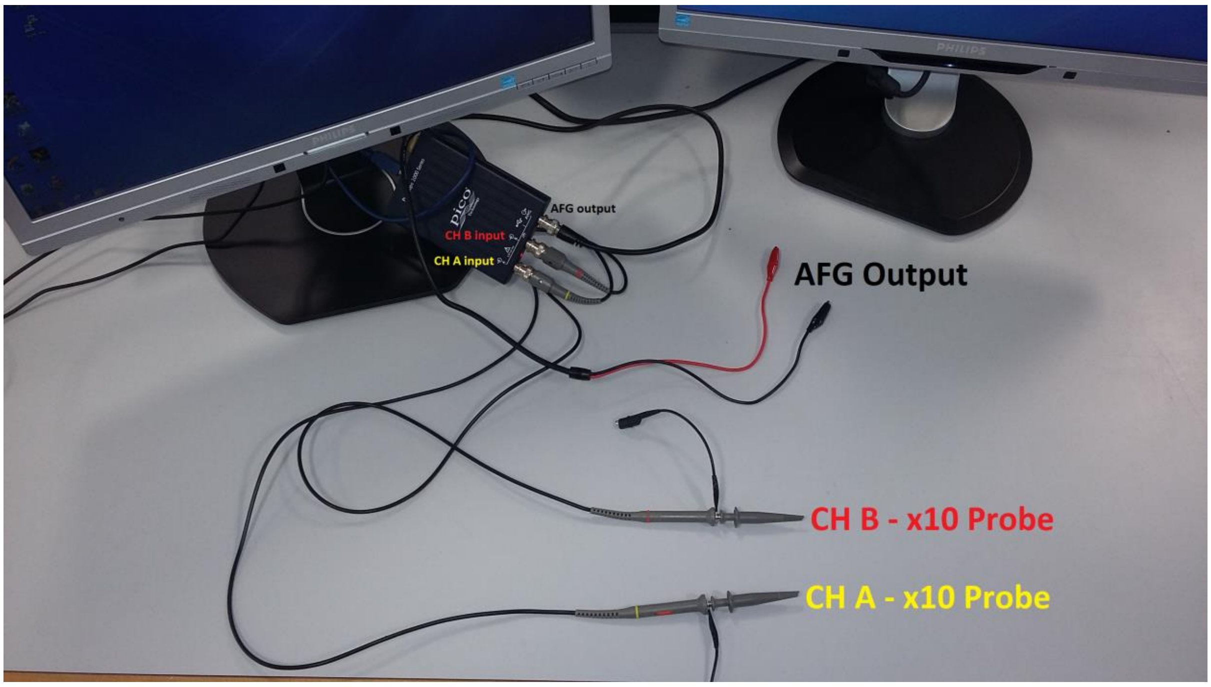
1. What are the highest values that can be programmed into the ***ARR*** registers for T
a) Timer2 ……………………
b) Timer3……………………

2. If the micro-controller has its PLL enabled and no pre-scale set (**PSC**=0) what is:
a) The clock speed of the micro-controller?..................
b) The time delay that will be produced by **Timer2** when the **ARR** registers are loaded with the highest possible value?...............
c) The time delay that will be produced by **Timer3** when the **ARR** registers are loaded with the highest possible value?...............

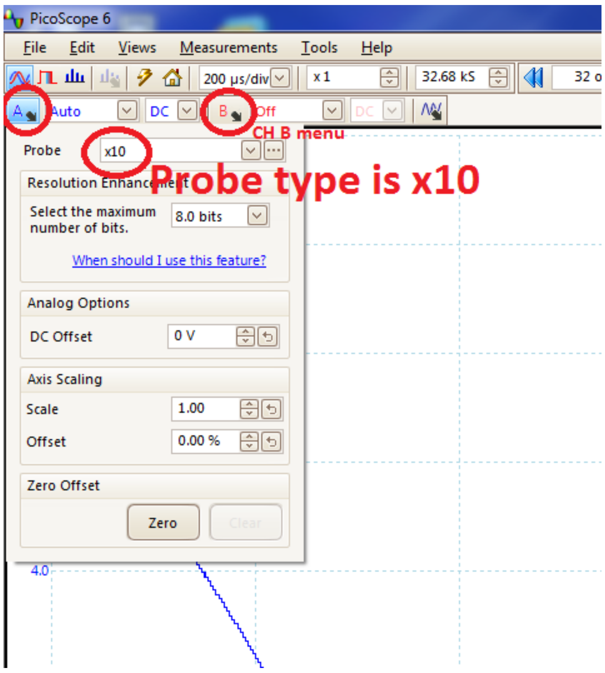
3. If the micro-controller has its PLL enabled and the Timer **PSC** registers are 16bit what is:
a) The largest value that can be programmed into the **PSC** register?.............
b) The longest time delay achievable by Timer2 and Timer3 respectively?.............

4. What will happen if the interrupt flag is not cleared before exiting the ISR?..........

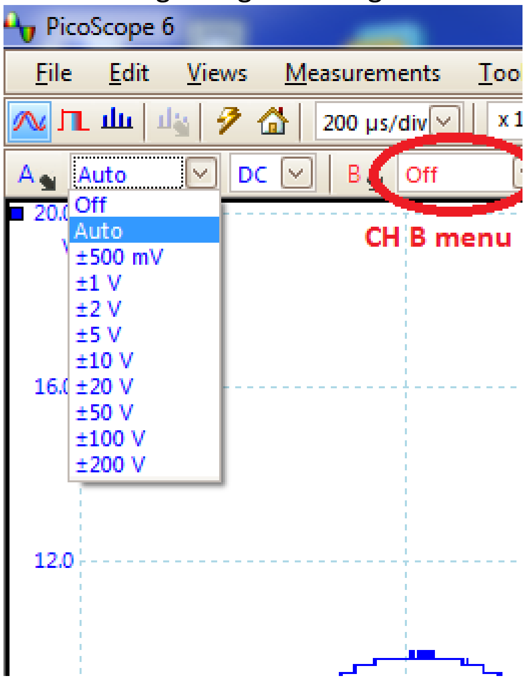
3.2 Pico-Scope for PC



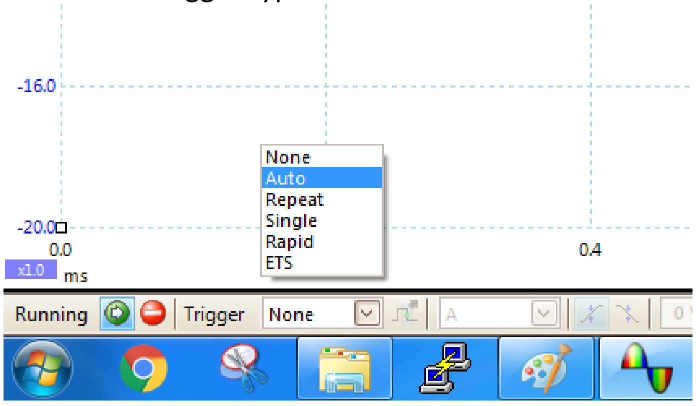
1. Open the Pico-Scope application by clicking the icon on the desktop.
2. The channels are configured here



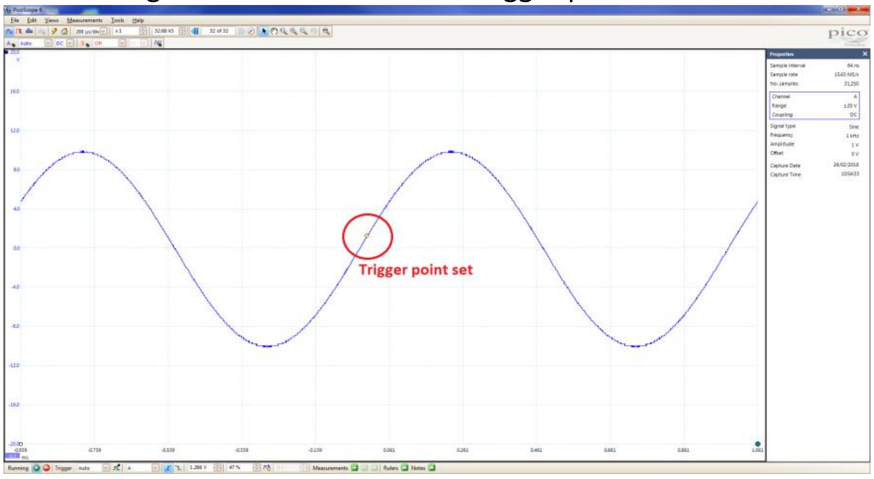
3. The voltage range is configured here



5. The trigger type is enabled here

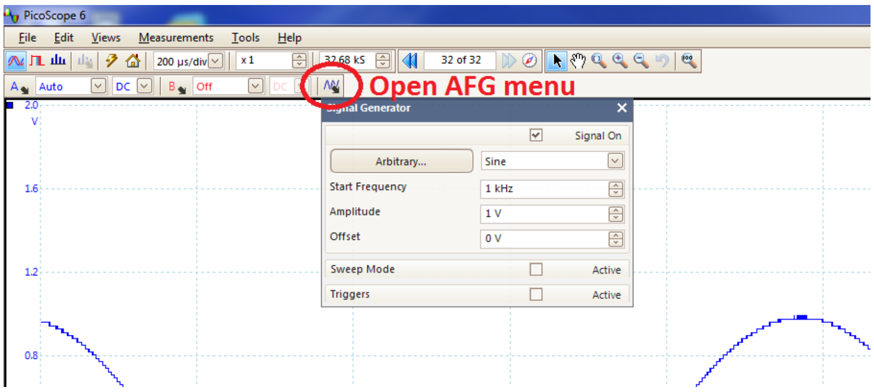


6. Drag the diamond to set the trigger point



3.2.1 Pico-Scope AFG

The AFG is enabled and configured here



Signal Generator

Signal On	✓
Arbitrary...	Sine
Start Frequency	1 kHz
Amplitude	1 V
Offset	0 V
Sweep Mode	☐ Active
Triggers	☐ Active

4 Support Documentation

Document Name	Contained Information
UM1974 User manual	• Pin identification and the supported special functions • Circuit schematics • Jumper and component identification • Header pinouts
RM0090 Reference manual	• MCU memory and peripherals architecture • Peripheral control registers, addresses and bit-fields

Table 2: Table of relevant support documentation for Nucleo-144 development boards (The document names are hyperlinks, please click on them to access the documents)