PERIPHERAL EQUIPMENT
2023 – 2024

PRACTICE 1

By Adrián Sánchez López

1. Microcontroller’s resources used

GPIOS

As GPIOs, the ports PG2, PG3, PG9, PD4, PD5, PA0 and PA4 have been used. PG2 and PG3 have been used as output ports through which PWM is displayed, port PG9 is the input auxiliary port used together with the button, which is PA0, ports PD4 and PD5 are the input ports that listen to PG2 and PG3 PWMs and finally PA4 is the analog output displaying Vc. PA0 has been selected because a button is needed and is the only usable button in the board, PA4 has been used as it is a DAC output port and the rest have been used as they are available GPIOs in the expansion connectors that do not have other features.

Timers

All general purpose timers have been used(TIM2 to TIM5), they have been used, as said, because they are general purpose ones and no special timer needed to be used. TIM2 and TIM5 are 32-bits long and TIM3 and TIM4 are 16-bits long. They can count up which is what I required. They have autoreload which is key for the TIM5 which has been used as the free-running timer that counts the time between rising edges from the PWMs and they can generate interrupts for different types of events, the only one i used is the overflow event, which is when the timer reaches the period assigned.

As said before, TIM5 has been used as the free-running timer, it means that it keeps counting up, which implies that it will eventually reach the maximum, when that happens extra operations need to be done to properly calculate the time since the last edge and the current edge, as the last would be a number close to 65535 and the current a value closer to 0. For example, imagine a last time value of 65035 and a current value of 300. If calculated as normal(current time - last time) we will get -64735, when the difference is actually 800. Therefore, before calculating the period it is checked if last time is bigger than current time (which only happens when the clock reaches its maximum and is reset). If it is, then the perdio is (65535 - last time + currentTime).

For the rest of the timers, TIM2 and TIM3 have been used to count up to Tp right and Tp left and throw and interrupt when overflown, to generate the PWMs. Finally, TIM4 has been used to count up to 1 ms and throw an interrupt when overflown, to toggle the status led.

DAC

The digital to analog converter has been used to transform the calculated value based on the current wheel's state to an analog output displayed through PA5, as said before.

EXTI

External interrupt/event controller has been to generate interrupts when the button is pressed or when the inputs listening to the PWMs detect a rising edge.

2. Interrupt structure

As previously said, EXTI interrupt and timer interrupts have been used. As there are different interrupts configured, there will be a lot of occasions where 2 interrupts are generated at the same time or 1 just after the other, which might make the one being execute second to not work as intended or measure wrong data as it is not being executed as soon as the interrupt is generated. To avoid these problems, priorities and sub-priorities have been arranged so in those cases the interrupt who depends more on being executed as soon as the interrupt is generated is executed first. The priorities have been the following, priority 0 has been given to the timers that generate the interrupts to generate the right and left sensor PWMs, the time for the left sensor has been given a subpriority of 0 and the one for the right of 1, so in case those 2 collide 1 is executed first. As the speed is generated in a way that makes the left wheel be faster or equal to the right wheel speed unless the button is pressed several times, a slightly superior priority is given to the left wheel timer as when testing it will be the fastest usually. Then, for PD4 and PD5 which generate interrupts when they detect a rising edge from the PWMs, they priority has been set to 1 and subpriority to 0 for the left and 1 for the right, again this subpriority follow the same logic as with the PWM timers. Finally, the timer that counts up to 1ms has been configured with a priority of 2 and sub priority of 0 and the interrupt generated when pressing the button with a priority of 3 and sub priority of 0.

The lower the number assigned the higher the priority, therefore the ones with the lowest priority are the timers used to generate the PWMs, followed by the interrupts generated at each PWM rising edge to count its period, the 1 ms timer and finally the button interrupt. This has been done like that as delaying the interrupt used to generate the PWM would result in a wrong PWM, then the second most important interrupt is the one used to coin the period of each PWM, as if delayed it will result in a longer or shorter period. Note that both the interrupt to generate the PWM and count it period require 0 delay, but if the interrupt to generate the PWM is delayed, not only the PWM will be generate wrong, but also the period counted for that PWM will be wrong, as it is being counted from a wrong PWM. Therefore, interrupt to generate the PWM has higher priority than the one to count its period, as the first affects the second, but not the other way around. Then we have the interrupt of the 1ms timer and the interrupt for the button. The one to coin up to 1 ms comes first as if delayed it will not count up to 1ms, but the PWM and its period are more important to be accurate than the 1ms counter as the PWMare key for the development of the project, but the 1ms counter is only used to toggle a led each 200 ms, therefore if it gets delayed 1,2 or 3 times each 200 ms it won't matter that much, as first its function is less important and it counts up to 200 ms, so the delay generated for executing another ISR first won’t affect that much this functionality as in the PWMas PWM has smaller times. Finally, the button has the worst priority as it is just required that when pressed the speed or correlation between wheel speeds is updated, as far as this is achieved, even if it is done with a small delay it will not affect the functioning of the system and we humans won't be able to perceive it.
: the interrupts used, relative priorities, etc.

3. Answers

Execution time of each ISR.

To get the execution time of each ISR, a GPIO has been set in the first line of each ISR and reset at the end. Then, the execution time is taken by measuring the time from the rising edge until the falling edge of that GPIO.

Execution time TIM2_IRQHandler(): 1 us
Execution time TIM3_IRQHandler(): 1us
Execution time TIM4_IRQHandler(): 1,12 us
Execution time EXTI0_IRQHandler(): 6,96 us
Execution time EXTI4_IRQHandler(): 166 ns
Execution time EXTI2_IRQHandler(): 166 ns

Rotation speed sensor ISR latency.

To be able to assess the latency of each wheel rotation speed sensor, a GPIO was set when the interrupt that generates the pwm sets the GPIO of the pwm and then in the interrupt of the rotation speed the GPIO set is reset.

The latency for the left wheel sensor has been 687 ns, while the latency for the right wheel has been 837 ns. There is a difference between the left and the right, which is about 150 ns which is the execution time of EXTI4. Which makes sense as the right wheel has the latency of the left one(as the process producing the latency is the same for both) + the execution time of the left one, as the left one has more priority so the right one has to wait for the left one to finish.

CPU Bandwidth dedicated to interrupt management.

Bandwidth = Lavg * (toh + tisr)

The bandwidth may change depending on the user actions, therefore min and max bandwidth will be calculated. Min bandwidth is the bandwidth of TIM4 which interrupt is executed each 1ms. EXTI0 (button int) is not counted as the minimum is when the user does not press the button. Also EXTI4, EXTI2 are not executed as the pwm is not generated as speed is 0. TIM2 & 3 are executed but no pwm is generated.

MiniBandwidth = bandwidthTIM4 + 2*bandwidthTIM2
BandwidthTIM4 = 1000 * (0 + 1,12*10-6) = 1,12*10-3
BandwidthTIM2 = 1000 * (0 + 1*10-6) = 1*10 -3

Lavg is 1000 as TIM4 generates interrupt each ms and TIM2 generates interrupt each ms when speed is 0. TIM3 is not calculated as it is the same as TIM2, so instead TIM 2 is multiplied by 2

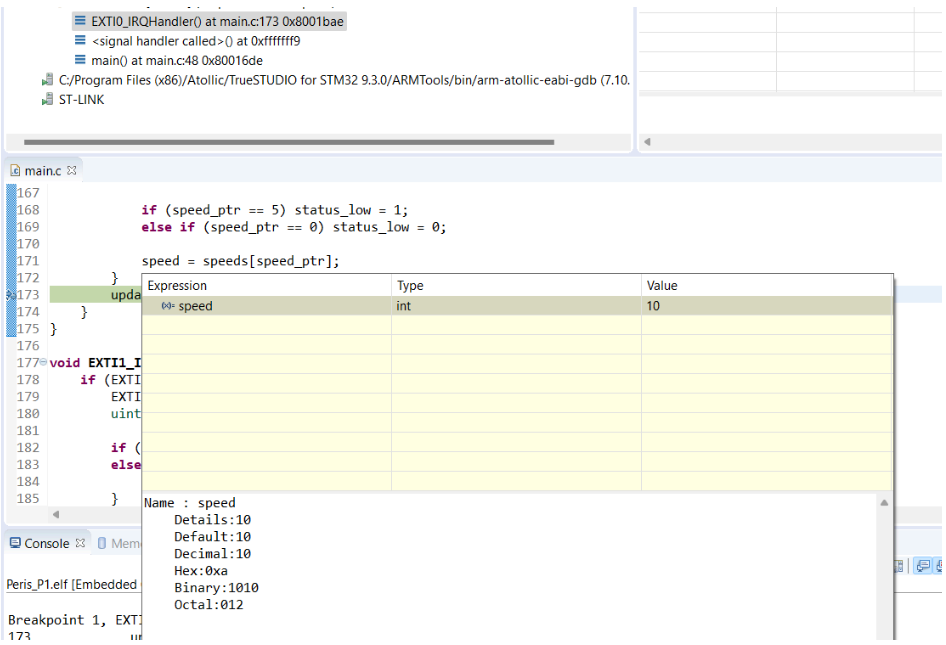
MinBandwidth = 2*10-3 + 1,12*10-3 = 0,00312.

Max bandwidth is bandwidth of TIM4 interrupt, pus TIM2 and TIM3 interrupts when speed is 270 and correlation 2,2 and BanEXTI0 + ban_EXTI4 and ban EXTI3
MaxBandwidth = Ban_TIM4 + Ban_TIM2 + Ban_TIM3 + Ban_EXTI0 + Ban_EXTI4 + Ban_EXTI3.
BanTIM4 = 0,001.
BanTIM2 = 5400 * 1*10-6 = 0,0054. 5400 is the the number of interrupts generated with speed 594 km/h (270 * 2,2) each second.
BanTIM3 = 2450 * 1*10-6 = 0,00245. 2450 is the number of interrupts generated with speed 270 each second.
BanEXTI0 = 1 * 6,96*10-6 = 0,00000696. Assume in the worst case the user will press the button 1 time per second.
Ban_EXTI4 = 2700 * 1,66*10-7 = 0,00044
Ban_EXTI3 = 1225 * 1,66*10-7 = 0,000203

MaxBandwidth = 0,001 + 0,0054 + 0,00245 + 0,00000696 + 0,00044 + 0,000203 = 0,0095.

Mechanism to visualize the contents of the variables.

When debugging and when in a breakpoint you can put the cursor over the variable/ expression you want and its result will be shown. As seen on the picture, when leaving the cursor over a variable/expression when in a breakpoint a new window appears with the variable/expression, its type and its value, pus then below the value in different numeric systems(binary, decimal, hexadecimal, octal).



```
EXTI0_IRQHandler() at main.c:173 0x8001bae
<signal handler called>() at 0xfffffff9
main() at main.c:48 0x80016de
C:/Program Files (x86)/Atollic/TrueSTUDIO for STM32 9.3.0/ARMTools/bin/arm-atollic-eabi-gdb (7.10.
ST-LINK
```

main.c

```
        if (speed_ptr == 5) status_low = 1;
        else if (speed_ptr == 0) status_low = 0;

        speed = speeds[speed_ptr];
    }
    upda
  }
}

void EXTI1_I
  if (EXTI
    EXTI
    uint

    if (
    else

    }
```

Expression	Type	Value
speed	int	10

```
Name : speed
    Details:10
    Default:10
    Decimal:10
    Hex:0xa
    Binary:1010
    Octal:012
```

Console Mem

Peris_P1.elf [Embedded

```
Breakpoint 1, EXTI
```

Alternatively, as explained in the Atollic guide, printf can be used with the proper set up explained in the guide to print variable values.

4. Mechanisms to modify the variables

Variables are modified as usual, no strange mechanism is needed, when variables need to be accessed from an ISR, the variables are declared globally, so the ISR that needs them can access them. Also when wanting to get 2 values as with the speed of right and left wheel, a function is called passing 2 variables(speed_right, speed_left) as reference, so with 1 call to the function it can get both variables without having to return an array or having to call the function 2 time(1 per each speed).