Mark	

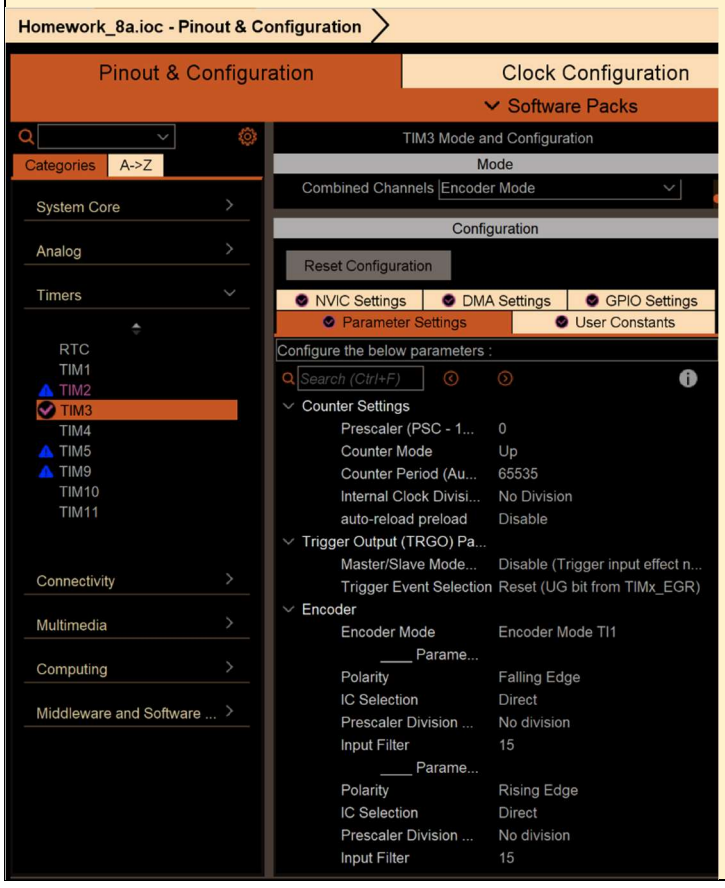
Team name:	*A1*		
Homework number:	*HOMEWORK 8*		
Due date:	26/11/2023		
Contribution	NO	Partial	Full
Pietro Monti			*x*
Alessia Moretto			*x*
Francesco Pallotto			*x*
Alessandro Perna			*x*
Ludovico Ventura			*x*
Notes:			

Project name	Encoder readout		
Not done	Partially done (major problems)	Partially done (minor problems)	Completed
			x

Explanation:
We successfully completed the homework.

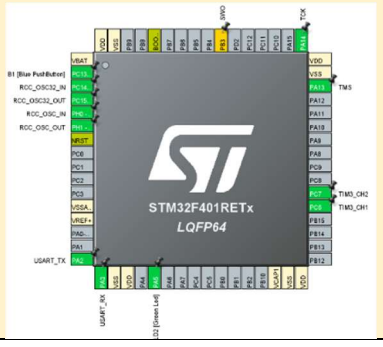
Part 1:

With this final project, the objective is to read the rotational speed (in rpm) of the encoder on the board.



1. Encoder configuration with TIM3:

- The incremental encoder operates discretely, producing 24 ticks per revolution. To achieve this, Timer TIM3 is interfaced with the encoder, operating in polling mode to keep track of the tick count. The rotational speed is subsequently calculated using the formula rpm = delta/24*60, where delta represents the change in tick count, given a specific interval of time. (Encoder mode and polarity choices are explained in the 3rd part).



2. TIM2 Callback:

- Within the TIM2 callback, set to trigger every 1 second, we define three variables: counts (current encoder ticks), old_counts (previous encoder ticks), and delta (the difference between counts and old_counts). These variables are of type int16_t, a choice motivated by reasons elaborated in subsequent sections (it solves over and underflow problems, as detailed in Part 2).

- Upon TIM2 triggering, the callback reads the TIM3 counter associated with the encoder. Subsequently, the rotational speed is computed using the previously described formula. Finally, the calculated rotational speed is transmitted to the terminal via UART (DMA, once activated the USART2 global interrupt from the NVIC table).

```c
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){

  static int16_t counts, old_counts, delta = 0;

  if (htim==&htim2){
    // Retrieve the current encoder counts and calculate the change in counts (delta), handling overflow/underflow.
    old_counts = counts;
    counts = __HAL_TIM_GET_COUNTER(&htim3);
    delta = counts-old_counts;
    // Calculate RPM (Revolutions Per Minute) from the delta counts, format it into a string, and transmit it via UART
    float rpms = (delta/24.000)*60;
    len = snprintf(str, sizeof(str), "%+.3f rpm \r\n", rpms);
    HAL_UART_Transmit_DMA(&huart2, (uint8_t *)str, len);
  }
}
```

Part 2:

- To fully understand the intricacies of using uint16_t variables as opposed to int16_t variables, we can assess the problems arising from the code seen during the lecture, which makes use of uint16_t for *old_counts* and *counts* and int for *delta.* In this case an incorrect handling of over and underflows can be noticed, as negative numbers cannot be interpreted as such. Any negative number (having MSB equal to 1, as per 2's complement encoding) is interpreted as a positive number, extending the range of positive values to 2^{16}-1. For this reason, whenever the variable *old_counts* and the variable *counts* contain values of opposite sign after crossing the 0 value, the subtraction causes an offset of an entire full scale range. Specifically, when the former is positive and the latter is negative, an entire FSR must be subtracted, and vice versa for the complementary situation (as seen in the code presented during class). This solution has the drawback of halving the sensing capability to values between -2^{16-1} and $+2^{16-1}$-1.

Expression	Type	Value
counts	uint16_t	65535
old_counts	uint16_t	1
delta	int	65534

Fig 1: a situation where two counterclockwise ticks produce a result of 65634 instead of -2

Expression	Type	Value
counts	uint16_t	1
old_counts	uint16_t	65535
delta	int	-65534

Fig 2: a situation where two clockwise ticks produce a result of -65634 instead of +2

A preliminary idea could involve using int16_t as the type for *counts* and *old_counts*. This solves the aforementioned problem, but shifts the issue elsewhere: specifically, when the accumulated value inside the TIM3 counter approaches the positive or negative extremes, a crossing of either limit produces exactly the same offset faced in the previous situation, requiring the same remedy. A simple way to solve all these issues is to replace the int *delta* (32 bits) with an int16_t *delta*, as truncation compensates for the

Name	Type	Value
rpms	float	163835
counts	int16_t	32766
old_counts	int16_t	-32768
len	int	18
delta	int	65534
str	char [32]	0x20017fc8

Fig 3: a situation where a single counterclockwise tick produces a result of 65534

Name	Type	Value
rpms	float	-163835
counts	int16_t	-32768
old_counts	int16_t	32766
len	int	18
delta	int	-65534
str	char [32]	0x20017fc8

Fig 4: a situation where a single clockwise tick produces a result of -65534

error, regardless of the types of *counts* and *old_counts*. This is the solution adopted in the above snippet.

However, this still does not overcome the limitation of halving the sensing capabilities of the 16-bit TIM3 counter, which could theoretically measure rotations spanning its entire range unidirectionally: it is theoretically possible (for very fast spinning) that the number of rotations occurring between consecutive polls ranges beyond FSR/2 of the TIM3 counter. In such cases, information is lost, as the new value placed in *count* will either reflect a slower rotation (when an entire FSR is exceeded) or a rotation in the wrong direction (when it is between FSR/2 and FSR). These considerations can be extended to all multiples of FSR/2 and FSR.

Part 3:

1) The role of the polarity in the encoder settings

Polarity in the encoder settings is a fundamental concept since it is at the base of the quadrature output idea. We have the possibility to exploit 2 encoder channels, which are 90° degrees out of phase. Setting the polarity of these two, we can decide in which situations the counter decreases or increases: in fact, when a transition of one channel happens, the system observes the level of the other signal, and we can act on it (i.e., setting its polarity) to infer the desired direction to the counter.

Let's make an example to better explain this. Let's assume to work in Encoder Mode TI1 (like in our project), and let's set the polarity of channel TI1 to *falling edge* and the one of channel TI2 to *rising edge*. In this way the counter increases if during TI2 rising edge TI1 is high, in fact its next transition is a falling edge. (i.e., if we are rotating the encoder clockwise), whereas it decreases if TI1 is low and is going to rise (i.e., if we are rotating the encoder counterclockwise). Changing the polarity of TI1 to *rising edge* (or symmetrically the polarity of TI2 to *falling edge*), the counter is increased for counterclockwise rotations, with a rising (falling) edge of TI1, which is low (high) during TI2 rising (falling) edge.

All the combinations are reported in table 49 of STM32F401RE reference manual

Active edge	Level on opposite signal (TI1FP1 for TI2, TI2FP2 for TI1)	TI1FP1 signal		TI2FP2 signal	
		Rising	Falling	Rising	Falling
Counting on TI1 only	High	Down	Up	No Count	No Count
	Low	Up	Down	No Count	No Count
Counting on TI2 only	High	No Count	No Count	Up	Down
	Low	No Count	No Count	Down	Up
Counting on TI1 and TI2	High	Down	Up	Up	Down
	Low	Up	Down	Down	Up

2) The working principle of the counter

The role of polarity is strongly related to the selected encoder mode. Using 2 signals TI1 and TI2 to interface with the encoder, the sequence of transitions of the two channels is evaluated and generates count pulses as well as the direction signal. There are 3 possible modes:

a. *Encoder Mode TI1* -> counter counts up/down on TI2 edge depending on TI1 level
b. *Encoder Mode TI2* -> counter counts up/down on TI1 edge depending on TI2 level
c. *Encoder Mode TI1 and TI2* -> counter counts up/down on both TI1 and TI2 edges depending on the level of the other channel

In the first two modes, the counter will accumulate 24 pulses per revolution, whereas in the third the number is 24 * 2 = 48, since the edges of both the two channels are relevant and counted.

The value stored in the counter represents the position of the encoder. If we want to compute the rotational speed, we can measure the time between two encoder events.

So when TIM3 is set Encoder Interface Mode, transitions on channels TI1 and TI2 act as an external clock source for the counter, with direction selection: the counter counts continuously between 0 and the auto-reload value.

Professor comments: