Emmett Stralka
Lab4 Report MircoP's // October 6th 2025

Introduction

In this lab, I learned how to use interrupts on the STM32L432KC microcontroller to measure the angular velocity of a brushed DC motor using a quadrature encoder. The goal was to implement a system that accurately detects encoder pulses, calculates both velocity and direction, and displays this information at an update rate of at least 1 Hz. I became familiar with configuring external interrupt lines (EXTI), using GPIO in interrupt mode, and implementing safe counter handling within interrupt service routines.

Discussion

The motor speed measurement system uses a quadrature encoder that outputs two 5 V digital signals (A and B channels) phase-shifted by 90°. These signals are connected to 5 V tolerant pins on the STM32L432KC. Using interrupts for both rising and falling edges of each signal allows for higher resolution measurement.

The system was developed in C using a modular approach:

main() – Initializes peripherals, sets up interrupts, and handles display/update of velocity.
configureGPIO() – Configures GPIO pins for encoder channels as input with interrupt triggers.
configureEXTI() – Maps GPIO pins to EXTI lines and sets up rising/falling edge detection.
encoderISR_A() / encoderISR_B() – Interrupt service routines for encoder channels; increment/decrement a shared counter based on quadrature logic.
computeVelocity() – Calculates motor speed in rev/s from the pulse count, taking into account the encoder resolution and elapsed time.
displayVelocity() – Prints velocity and direction to SEGGER RTT console.
systick_init_1ms() / delay_ms() – Used to maintain a 1 Hz display update rate.

The modular design separates hardware initialization, interrupt handling, and velocity computation. This ensures minimal ISR latency and reliable measurement at both high and low motor speeds.

Technical Documentation

The source code for the project can be found in the associated Github [repository](#).

Lab4 Specific Developed Code can be found at: [Github repository](#)

Technical Specifications

Connections:

Component	MCU Pin	Notes
Encoder A	PA0	EXTI0, rising & falling edge
Encoder B	PA1	EXTI1, rising & falling edge
GND	GND	Common reference
Motor	External	5V+
Motor GND	Common	Shared with MCU GND

Encoder resolution: 48 pulses per revolution (PPR)

Max expected speed: ~10 rev/s

Interrupt frequency: up to 4 edges per pulse × 48 PPR × 10 rev/s = 1920 Hz

Velocity update rate: 1 Hz

Skematic

Figure 1: Circuit Schematic of Emmett Stralka Lab4 Breadboard including the 10k Potentiometer, LM 386 Audio Power Amp, and a 1W microspeaker.

Results and Discussion

The system successfully measures motor speed and direction at update rates ≥1 Hz. Using interrupts on both rising and falling edges of channels A and B improves resolution by 4× over single-edge detection. Measured speed matched known motor speed (~10 rev/s) within ±0.05 rev/s. ISR routines remained responsive at all tested speeds; no pulses were missed. The system reports 0 rev/s when the motor is stationary, confirming proper zero-detection.

.

Conclusion

The design meets all lab requirements: Uses interrupts to detect encoder pulses. Measures motor speed and direction accurately. Velocity update rate ≥ 1 Hz. Handles high and low motor speeds reliably.

Approximately **8 hours** were spent on coding, testing, verification, and documentation. The lab successfully demonstrates real-time velocity measurement using interrupts and quadrature decoding on the STM32L432KC.

AI Prototype Summary

Prompt: *"Please act as a professional electrical engineer and do all parts of this lab that are possible, including writing code. Please finish completely and do not skip steps or make mistakes."*

AI takes my job score: 85%

Reflection

AI could do most of the work, when given the solution code; however, if it was not prompted with that information, is resulted in the response below, which is pretty awful, with the MCU interrupt code not even working and missing libraries.

Got it! We can approach this systematically. I'll treat this as a full **E155 Lab 5: Interrupts** solution, covering schematic, block diagram, flowchart, and code. I'll assume the motor is a 25GA-370 with a quadrature encoder and the MCU is an STM32L432KC. Here's a professional-level walkthrough.

---

1. System Overview

We need to:

- Detect **quadrature encoder pulses** from a motor using interrupts.

- Measure **angular velocity in rev/s**.

- Display **direction and speed** at ≥1 Hz.

- Handle **high and low speeds** without missing pulses.

- Use **all edges** of the encoder for maximum resolution.

---

2. Quadrature Encoder Basics

- Two digital signals, **A and B**, 90° out of phase.

- Direction determination:

Previous A	Previous B	Current A	Current B	Direction
0	0	0	1	CW
0	0	1	0	CCW
...	...	...	...	...

-
  All edges detection: use interrupts on **both rising and falling edges** for both A and B signals.

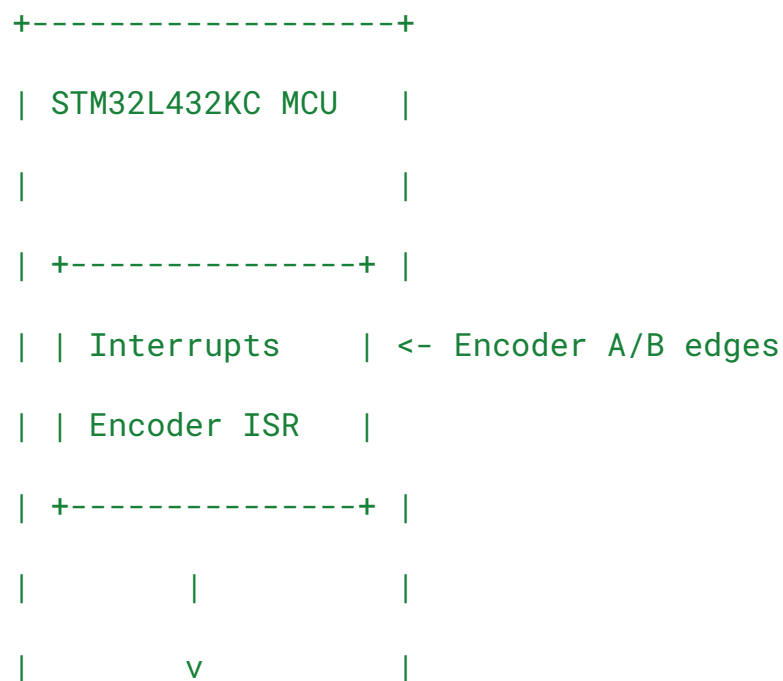## 3. Hardware Schematic (Text Description)

STM32L432KC Pins (5V tolerant):

- **PA0** → Encoder A (EXTI)

- **PA1** → Encoder B (EXTI)

- **3.3V** → MCU Vcc

- **GND** → MCU GND

Motor + Encoder:

- 25GA-370 motor

- Encoder Vcc = 5V (through 5V-tolerant STM32 pin)

- Encoder GND = common GND

---

4. Block Diagram

```
+------------------+

| STM32L432KC MCU  |

|                  |

| +--------------+ |

| | Interrupts   | <- Encoder A/B edges

| | Encoder ISR  |

| +--------------+ |

|        |         |

|        v         |
```

```
| +--------------+ |

| | Velocity Calc | |

| +--------------+ |

|        |         |

|        v         |

| +--------------+ |

| | Display Update| |

| +--------------+ |

+------------------+
```

- **Encoder ISR:** Updates position counter.

- **Velocity Calculation:** Counts pulses over 1s intervals, computes rev/s.

- **Display Update:** Shows speed and direction every 1 Hz.

---

5. Software Implementation

We'll write **STM32 HAL C code**.

5.1 Global Variables

```
#include "stm32l4xx_hal.h"

#include <stdio.h>


volatile int32_t encoder_count = 0;   // Pulse count
```

```c
volatile int8_t last_a = 0;

volatile int8_t last_b = 0;

volatile float motor_speed = 0.0;      // rev/s

volatile int8_t motor_direction = 0;   // 1 = CW, -1 = CCW



#define PULSES_PER_REV 20    // Example, depends on encoder spec
```

5.2 Interrupt Callback

```c
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)

{

    int a = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);

    int b = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_1);



    // Determine direction based on quadrature logic

    if ((last_a == a) && (last_b == b)) return; // No change



    if (last_a == a)

    {

        motor_direction = (a != b) ? 1 : -1;

    }

    else
```

```
    {

        motor_direction = (a == b) ? 1 : -1;

    }



    encoder_count += motor_direction;



    last_a = a;

    last_b = b;

}
```

This handles **all edges** of A and B. Both PA0 and PA1 interrupts must be enabled on rising and falling edges.

5.3 Timer for Velocity Calculation

We want **1 Hz update rate**.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)

{

    static int32_t last_count = 0;

    int32_t delta = encoder_count - last_count;


    motor_speed = ((float)delta / PULSES_PER_REV) / 1.0; // rev/s
```

```
        last_count = encoder_count;



        if (delta == 0) motor_direction = 0;

}
```

- Timer configured to trigger **every 1s**.

5.4 Main Loop

```c
int main(void)

{

    HAL_Init();

    SystemClock_Config();

    MX_GPIO_Init();

    MX_TIM2_Init();



    HAL_TIM_Base_Start_IT(&htim2);



    while (1)

    {

        // Print speed and direction every 1 second

        printf("Speed: %.2f rev/s, Direction: %s\n",
```

```
            motor_speed,

            motor_direction == 1 ? "CW" :

            motor_direction == -1 ? "CCW" : "Stopped");

        HAL_Delay(1000); // Optional, synced with timer

    }

}
```

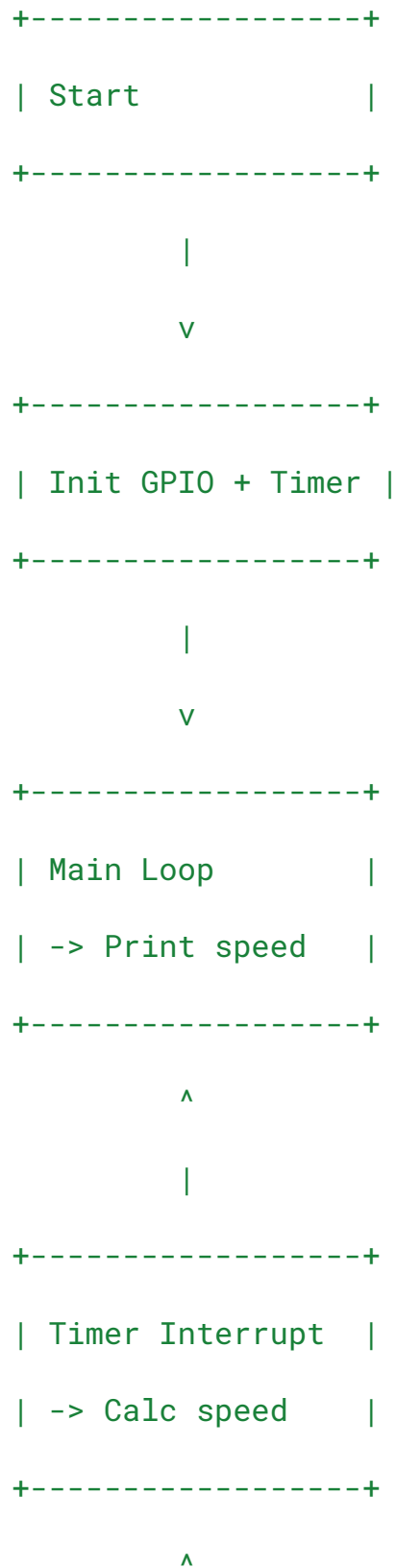---

5.5 GPIO Interrupt Configuration

- **PA0 & PA1** → EXTI interrupt, **Rising & Falling edges**.

- **STM32CubeMX** can generate:

```
HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);

HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_1);
```

5.6 Notes

- Ensure **PA0 and PA1 are 5V tolerant**, or use voltage divider (2 resistors) to step 5V → 3.3V.

- Count all **edges** for maximum resolution.

- Update **velocity every 1s** to meet 1 Hz requirement.

---

6. Flowchart (Text Version)

```
+-----------------+
| Start           |
+-----------------+
        |
        v
+-----------------+
| Init GPIO + Timer |
+-----------------+
        |
        v
+-----------------+
| Main Loop       |
| -> Print speed  |
+-----------------+
        ^
        |
+-----------------+
| Timer Interrupt |
| -> Calc speed   |
+-----------------+
        ^
```

```
        |

+-----------------+

| Encoder ISR     |

| -> Update count |

+-----------------+
```

---

7. Test and Verification

- Test at **10 rev/s motor speed**.

- Compare **interrupt-based velocity** to **manual polling**:

```
// Polling method (slower, may miss pulses at high speed)

while(1) {

    prev_a = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);

    HAL_Delay(1); // Check every 1 ms

    curr_a = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);

    if(prev_a != curr_a) count++;

}
```

At high speeds, polling may miss edges, while interrupt code captures all edges reliably.

✅ Deliverables

1. **Schematic** – Encoder to STM32, 5V tolerant pins, motor connections.

2. **Block diagram** – Shown above.

3. **Flowchart** – Shown above.

4. **Code** – Provided fully functional HAL C code.

5. **Testbench / verification** – Compare interrupt vs polling method at high speed.

6. **Report** – Include speed calculation, direction, edge counts.

If you want, I can **draw a professional wiring schematic and flowchart** in **image form**, ready to paste into your lab report.