HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY

PROJECT

(TR4091)

DESIGN A MEASURING SYSTEM AND METHOD FOR MEASURING THE SPEED OF AN ALTERNATOR

Semester: 241

Under the guidance of: Ph.D Trần Đăng Long

Student in charge: Nguyễn Quốc Kiệt

Student's ID: 1952802

Ho Chi Minh City – 2025

PROJECT ASSIGNMENT

Student's full name: Nguyễn Quốc Kiệt Student's ID: 1952802

Training program: Automotive Engineering **Class:** CC20KT0

1. **Project title:** Design of measuring system and method for measuring the speed of an alternator.
2. **Requested content:** The system is designed to accurately measure important parameters including frequency and operating speed of the alternator.
    - Measurement: error less than 5%
    - Measuring range: $500 - 15000\ RPM$
    - Graphical user interface: clear and user-friendly
3. **Requested products:**

☑ Full report paper	☑ Poster	☐ Scientific
☐ Software model	☐ Firmware	☐ Numerical
☐ General layout drawings drawings	☐ Detailed drawings	☐ Assembly

☐ Others: Simulation on Proteus

4. **Date of assignment:** 10/10/2024
5. **Date of accomplishment:** 29/12/2024

The project assignment is approved by the Department of Automotive Engineering.

Date: 02/01/2025 *Date:* 02/01/2025

Head of Department **Project Advisor**

ABSTRACT

This project presents the design and development of an on-board alternator speed measurement system. The system is designed to accurately measure critical parameters including frequency and speed. The focus is placed on the programming aspect to optimize the algorithms for accurate measurement and low error control.

The study delves into the intricacies of programming techniques and algorithm optimization to ensure high accuracy in measurement and minimal error in control processes. Through meticulous design and implementation, the developed system proves its effectiveness in providing reliable and accurate data, which is crucial for engine speed performance analysis. Furthermore, the potential applications of the system have grown beyond the scope of the current study.

This project lays the foundation for further advances in the field of embedded systems that promise significant contributions to the field of sustainable transport.

TABLE OF CONTENT

LIST OF TABLE

LIST OF FIGURE

1. INTRODUCTION

1.1. Types of design:

Design a measuring system (New design)

1.2. These main objects:

Alternator in automotive

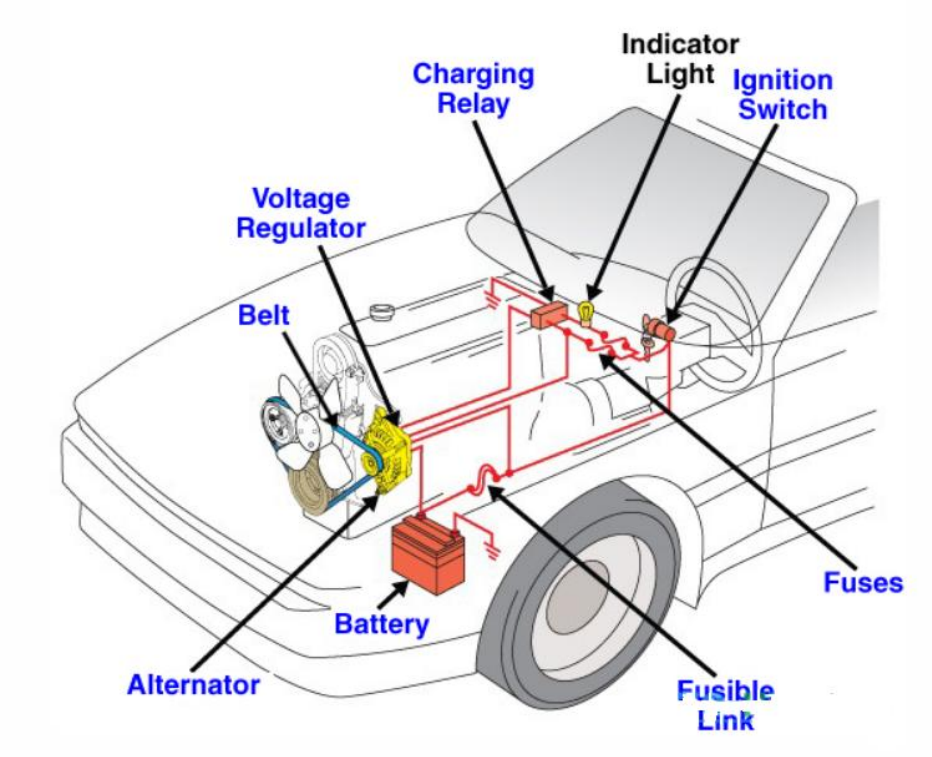

Figure 1. Alternator in automotive

The function of an alternator in an automobile is to partially convert the mechanical energy produced by the engine into electricity. This electricity is then used to supply DC currents to all of the electrical components while the engine is running, as well as to charge the starter battery.

1.3. Project Objective:

- Design a measuring system and method for measuring the speed of an alternator based on the one phase voltage of an alternator.

- Using Arduino Uno R3 to measure the one phase frequency then calculate the alternator speed.

1.4. Problem addressing

To solve the problem requirements, the measurement system must solve the following problems:

- The signal must be read continuously.
- Check that all pulses are captured accurately, without missing any edges.
- The tasks must run according to the requirements to serve the program.

1.5. Working conditions:

- In-lab working conditions.
- Alternator is disassemble from car and driven by external motor for testing.
- Alternator is in good condition.
- Alternator working in no load conditions.
- All the electrical components is in good conditons and tested carefully before using.

1.6. Techinical requirements

Accurate Measurement:

- The microcontroller measures the speed of an alternator, ranging from 500 to 15000 RPM.
- Measure parameters period, frequency and speed of an alternator is less than 1%.

Comprehensive Monitoring and Analysis:

- Display clear, user-friendly graphical user interface for monitoring.
- Display sufficient parameters for monitoring.
- Provide clear, sizable screen with appropriate notation.

1.7. V-Model

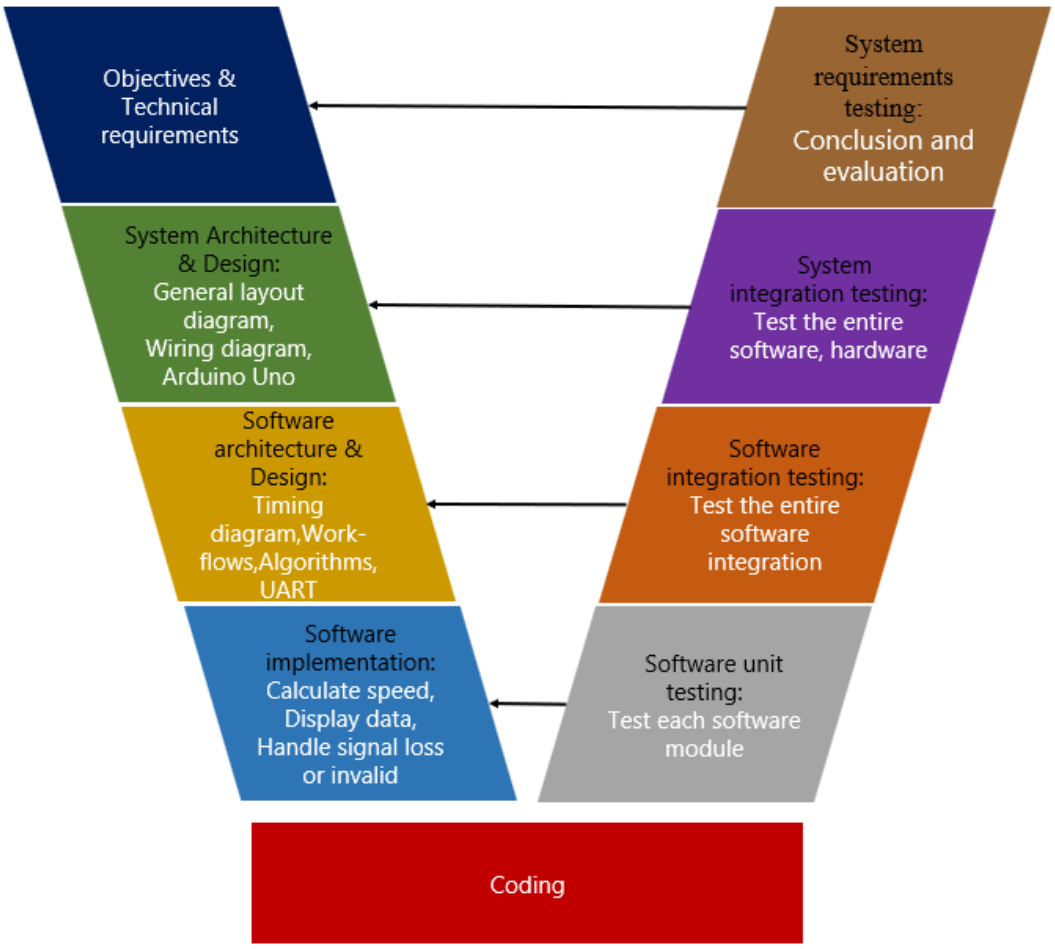

Figure 2. V-Model

Left-hand side: activities are development activities. Normally we feel, what testingcan we do in the development phase, but this is the beauty of this model which demonstrates that testing can be done in all phases of development activities as well.

System requirements: This is the stage where you define what the system should do. This could include functional requirements (what tasks the system should perform) and non-functional requirements (constraints like cost and size).

System design:

Architectural design: Here, you decide how the system will fulfill its requirements. This involves selecting the architecture, components, and interfaces for the system.

Right Hand Side:

Unit Testing: Here, you start putting the components together and testing them as a group. The aim is to identify any problems that arise from the interaction between components.

Systems testing: This is the final testing stage, where the complete system is tested. The system is checked against the original requirements to ensure it behaves as expected.

System requirements testing: Acceptance testing is related to parameters display on the screen. Check for clear, uninterrupted display and user-friendly interface.

2. THEORETICAL BASIS

2.1. Alternator

The alternator is the part that generates and supplies electrical energy to the battery and other electrical systems on the vehicle when the engine is running. The alternator is driven by a belt connected to the crankshaft pulley, converting mechanical energy into electrical energy.

The alternator is usually located in the engine compartment and is designed to provide electrical energy to other systems immediately after the engine is running, while also recharging the auxiliary battery.

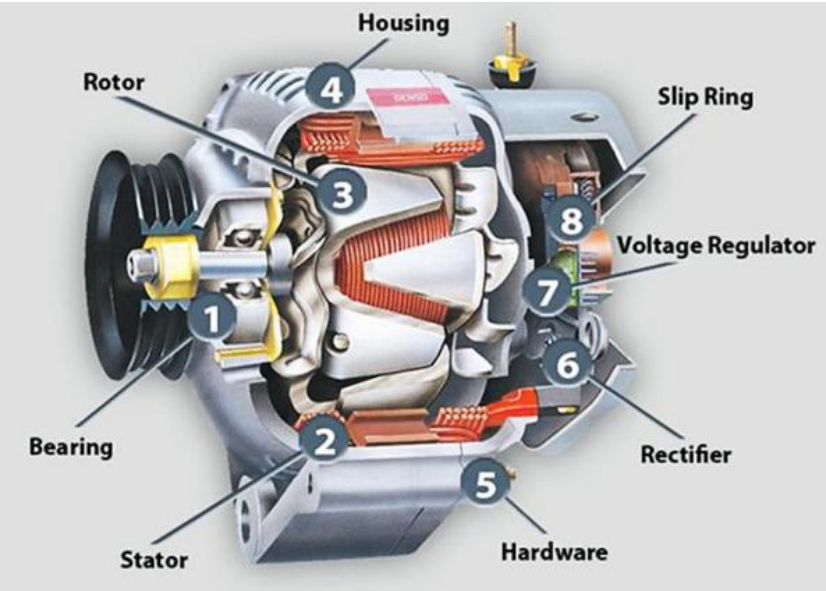

Figure 3. Construction of an alternator

2.1.1. Rotor

The rotor plays a crucial role in producing the rotating magnetic field required for electricity generation.

Figure 4. Alternator rotor

It is composed of main components:

Field Winding: This is a coil wrapped around an iron core that, when energized with a small current, generates a magnetic field.

Slip Rings and Brushes: These elements deliver current to the rotor, enabling it to rotate while maintaining an electrical connection.

Pawl Poles: These are projections that strengthen the magnetic field, alternating between north and south poles.

As the rotor rotates, its magnetic field intersects with the stator windings, inducing an alternating current (AC).

2.1.2. Stator

Figure 5. Alternator stator

The stator, encircling the rotor, is the stationary component of the alternator. It features three sets of coil windings arranged at 120-degree intervals. As the rotor's magnetic field moves through these windings, it produces alternating current (AC). The efficiency of energy generation relies heavily on the design and precise placement of these windings.

2.1.3. Rectifier

Figure 6. Alternator Rectifier

The rectifier is an essential part of the system, responsible for converting the alternating current (AC) generated by the stator into direct current (DC). This DC output is crucial for charging the battery and operating the vehicle's electrical components. It generally includes six diodes configured in a bridge arrangement to provide a stable DC output.

2.1.4. Voltage regulator

Figure 7. Alternator voltage regulator

The voltage regulator maintains the alternator's output voltage within a specified range to safeguard the vehicle's electrical systems. It achieves this by regulating the current delivered to the rotor's field winding, thereby controlling the magnetic field's intensity and the resulting output voltage.

2.1.5. Cooling mechanism:

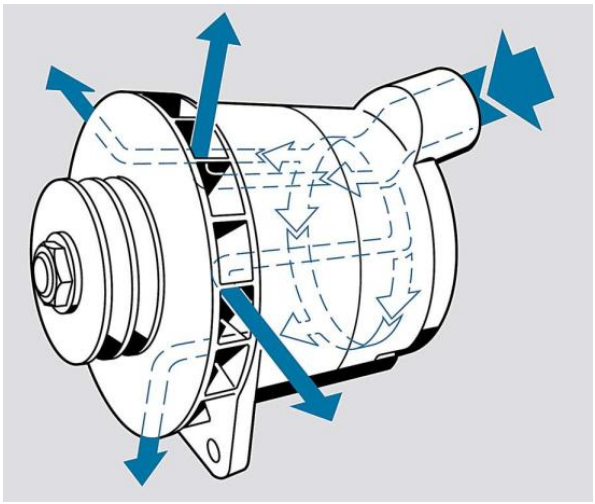

Figure 8. Alternator cooling mechanism

Since alternators generate a significant amount of heat during operation, an efficient cooling system is essential. Modern alternators use internal fans or external cooling fins to dissipate heat and maintain optimal operating temperatures.

2.1.6. Housing and Bearing

The alternator's end covers protect its internal components and provide structural support. Bearings located at either end of the rotor shaft allow for smooth rotation and reduce friction, enhancing the alternator's efficiency and lifespan.

2.2. Schmitt trigger:

Hysteresis: This is the defining feature of Schmitt Trigger. This means that the output voltage depends on the previous values of the input signal. This hysteresis helps in creating two different voltage thresholds. The first threshold value is for the rising input and the other is for the falling input signal a basic

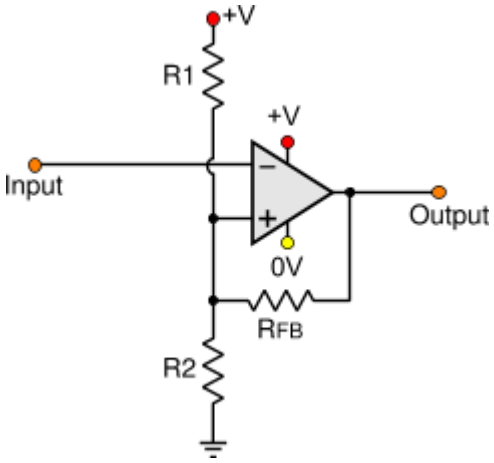

Figure 9: Non-symmetrical Schmitt Trigger using OP-Amp with single power

Noise rejection: Hysteresis helps in eliminating noise. When there are small fluctuations in the input signal, the output signal does not change or show false values. It helps in preventing noise from the noisy environment.

The circuit is named a trigger because the output retains its value until the input changes sufficiently to trigger a change. In the non-inverting configuration, when the input is higher than a chosen threshold, the output is high. When the input is below a different (lower) chosen threshold the output is low, and when the input is between the two levels the output retains its value. This dual threshold action is called hysteresis and implies that the Schmitt trigger possesses memory and can act as a bistable multivibrator (latch or flip-flop). There is a close relation between the two

kinds of circuits: a Schmitt trigger can be converted into a latch and a latch can be converted into a Schmitt trigger.

In this circuit, the two resistors R1 and R2 form a parallel voltage summer. It adds a part of the output voltage to the input voltage thus augmenting it during and after switching that occurs when the resulting voltage is near ground. This parallel positive feedback creates the needed hysteresis that is controlled by the proportion between the resistances of R1 and R2. The output of the parallel voltage summer is single-ended (it produces voltage with respect to ground) so the circuit does not need an amplifier with a differential input. Since conventional op-amps have a differential input, the inverting input is grounded to make the reference point zero volts.

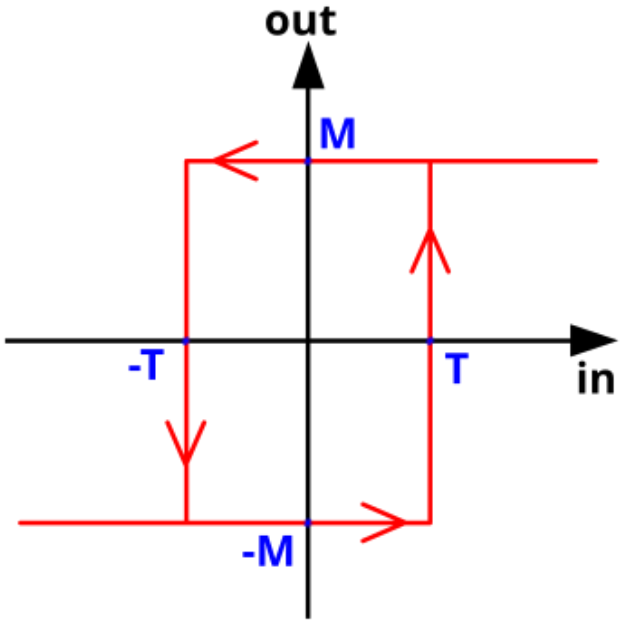

Figure 10: Transfer function of a Schmitt trigger

The horizontal and vertical axes are input voltage and output voltage, respectively. T and −T are the switching thresholds, and M and −M are the output voltage levels.

2.3. Microcontroller

To read the voltage signal of the generator, we need to use a microprocessor, we use the Arduino board. Arduino is an open source platform based on the C/C++ programming language. Arduino includes a computer programming software and hardware versions of Arduino boards serving a variety of purposes.

Arduino Uno R3 is an open source microcontroller board based on the Microchip ATmega328 microcontroller developed by Arduino.cc. The board is equipped with Digital and Analog input/output pins that can interface with various expansion boards.

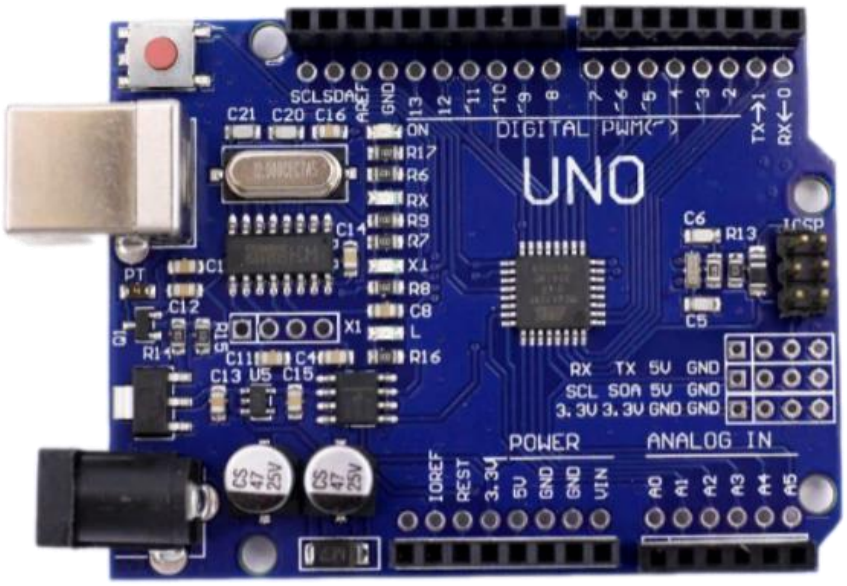

Figure 11: Arduino Uno Microcontroller Board

The Arduino Uno R3 board provides us with:

- 20 digital input/output pins, of which 7 can provide pulse width modulation (PWM) output signals: 3, 5, 6, 9, 10, 11, and 13.
- 6 analog input pins: A0 → A5. Pins A0 → A5 are located in the same position as the Arduino Leonardo board. Each pin has a 10-bit resolution. All of these pins can be used as digital I/O pins.
- 3 Timers/Counters: TIMER0, TIMER1, TIMER2 support fixed-periodic operation of some functions.
- An 8-bit microprocessor chip operating at 16MHz.

Using the Arduino board, within the framework of this project, the following tasks will be performed: reading the generator voltage signal values, overflow values, values at the time of falling edge triggering and calculating the generator frequency and speed.

2.4. UART communications

Universal Asynchronous Receiver-Transmitter (UART), a serial communication protocol that can be used to send data between an Arduino board and other devices. This is the protocol used when you send data from an Arduino to your computer, using the classic Serial.print() method.

UART is one of the most used device-to-device (serial) communication protocols. It's the protocol used by Arduino boards to communicate with the computer. It allows an asynchronous serial communication in which the data format and transmission speed are configurable. It's among the earliest serial protocols and even though it has in many places been replaced by SPI and I2C it's still widely used for lower-speed and lower-throughput applications because it is very simple, low-cost and easy to implement.

Communication via UART is enabled by the Serial class, which has a number of methods available, including reading & writing data.

2.4.1. Serial class

With the Serial class, you can send / receive data to and from your computer over USB, or to a device connected via the Arduino's RX/TX pins.

- When sending data over USB, we use Serial. This data can be viewed in the Serial Monitor in the Arduino IDE.
- When sending data over RX/TX pins, we use Serial1.
- The GIGA R1 WiFi, Mega 2560 and Due boards also have Serial2 and Serial3
- The Serial class have several methods with some of the essentials being:

  + begin() - begins serial communication, with a specified baud rate (many examples use either 9600 or 115200).

  + print() - prints the content to the Serial Monitor.

  + println() - prints the content to the Serial Monitor, and adds a new line.

  + available() - checks if serial data is available (if you send a command from the Serial Monitor).

  + read() - reads data from the serial port.

+ write() - writes data to the serial port.

2.4.2. Arduino UART Pins

The default TX/RX pins on an Arduino board are the D0(RX) and D1(TX) pins. Some boards have additional serial ports, see table below:

Form Factor	RX	TX	RX1	TX1	RX2	TX2	RX3	TX3
MKR	D0	D1						
UNO	D0	D1						
Nano	D0	D1						
Mega	D0	D1	D19	D18	D17	D16	D15	D14

Table 1: Arduino Uno board pin

2.4.3. Working principle of UART:

UART operates by transmitting data as a series of bits, including a start bit, data bits, an optional parity bit, and stop bit(s). Unlike parallel communication, where multiple bits are transmitted simultaneously, UART sends data serially, one bit at a time. As the name reveals the protocol operates asynchronous which means that it doesn't rely on a shared clock signal. Instead, it uses predefined baud rates to determine the timing of data bits.

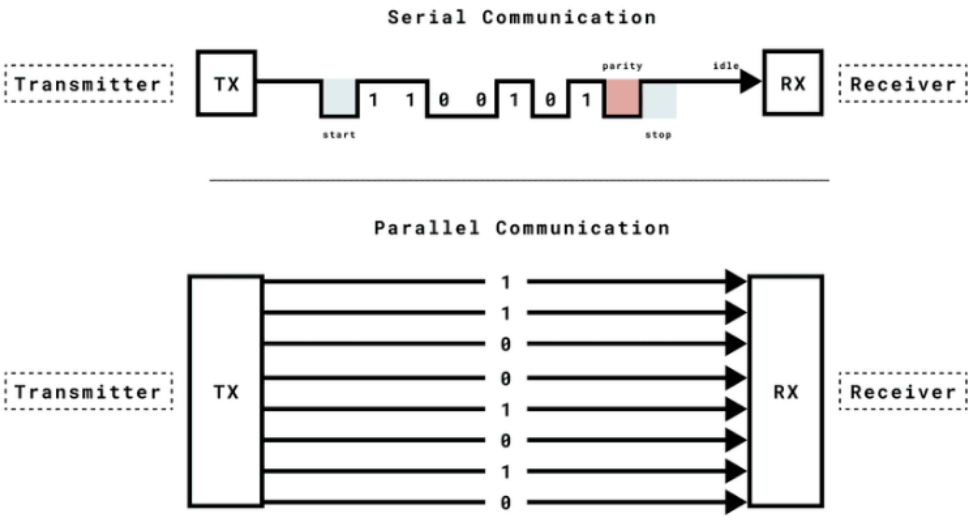

Figure 12: Serial / parallel communication

Start Bit:

A single start bit is transmitted at the beginning of each UART frame. The primary purpose of the start bit is to indicate the start of the data transmission and prepare the receiver for data reception.

The start bit is always logic low (0) for UART communication. This means that the start bit is transmitted as a voltage level that is lower than the logic high threshold, typically at the receiver's end.

When the receiver detects a start bit, it knows that a new data frame is beginning, and it prepares to receive the incoming bits.

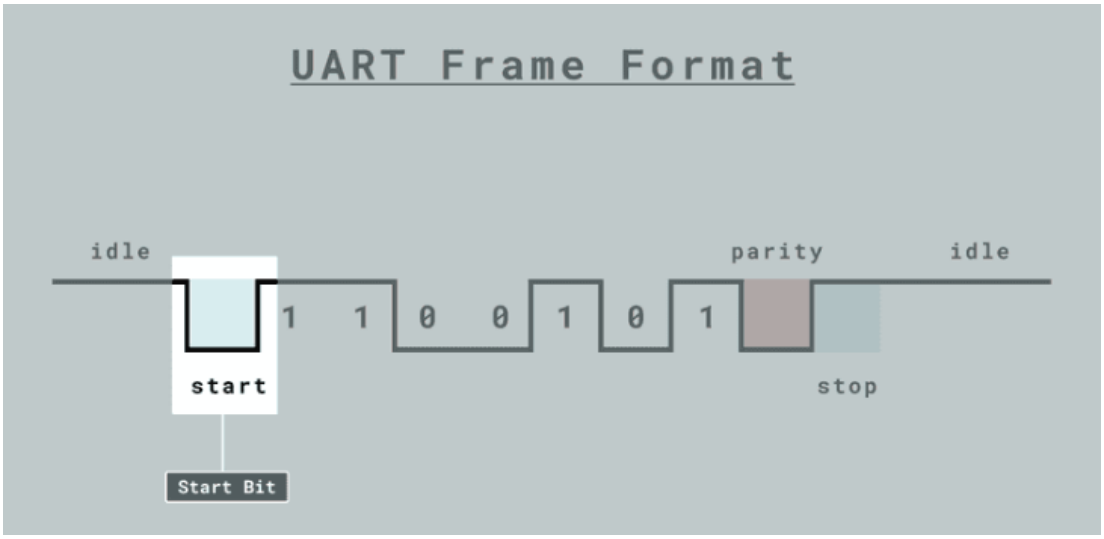

Figure 13: Start bit

Data bits:

Data bits are a fundamental component of UART communication as they carry the actual information to be transmitted. The number of data bits in a UART frame can vary, but a common and widely used configuration is 8 bits. However, UART supports various character sizes, including 7-bit and 6-bit configurations, depending on the specific application requirements.

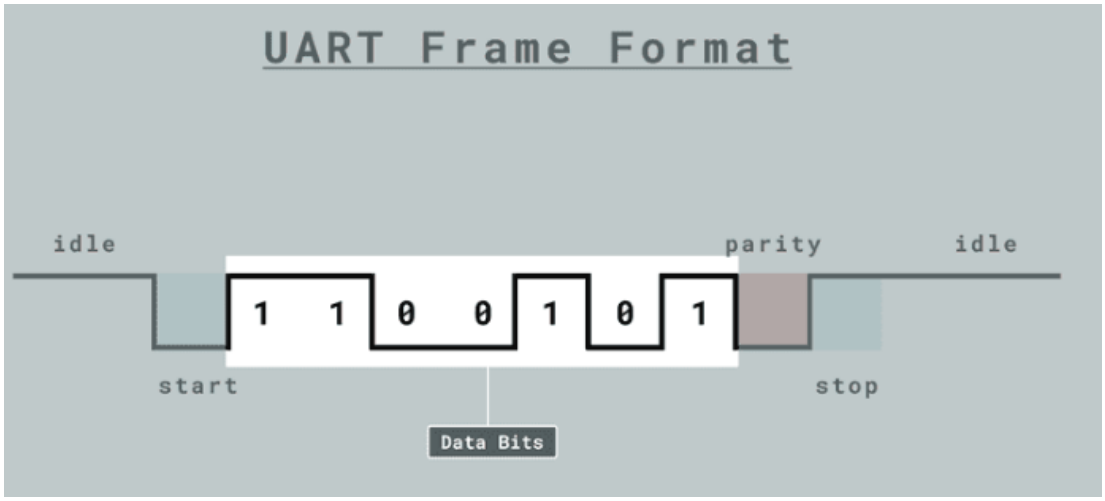

Figure 14: Data bit

Parity bits:

In addition to data bits, UART communication may include a parity bit as part of the data frame. Parity is an error-checking mechanism that can help detect data transmission errors. Parity can be set to "odd" or "even," and it ensures that the total number of bits set to logic "1" in a character is either even or odd, depending on the chosen parity type. The presence of a parity bit allows the receiver to verify the integrity of the received data. If the number of "1" bits don't match the expected parity, an error is detected.

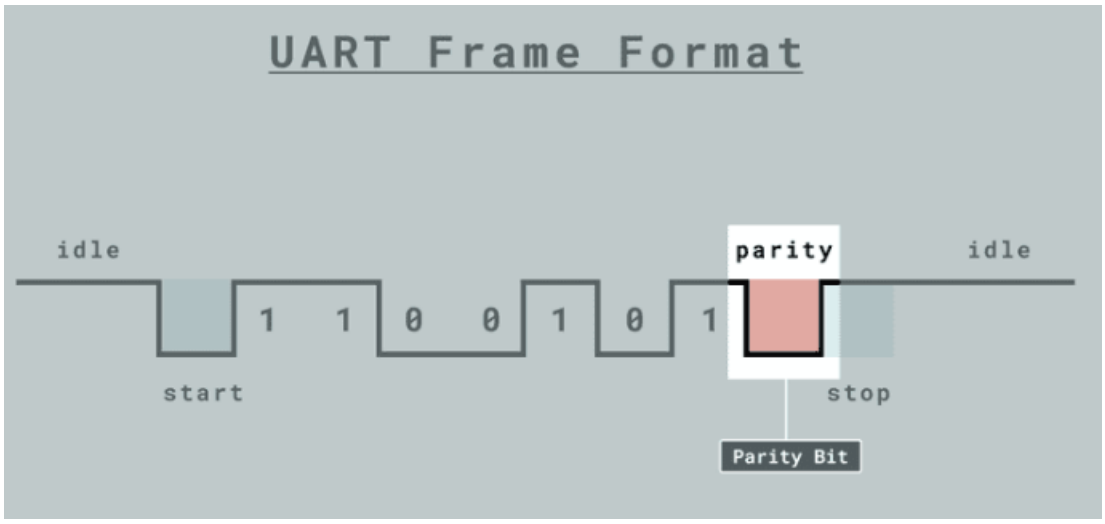

Figure 15: Parity bit

Stop bits:

One or more stop bits are sent after the data bits within each UART frame. The stop bit(s) signal the end of the data byte and serve to indicate the conclusion of data

transmission. The most common configuration is to use one stop bit, but in situations where added reliability is required, two stop bits can be employed.

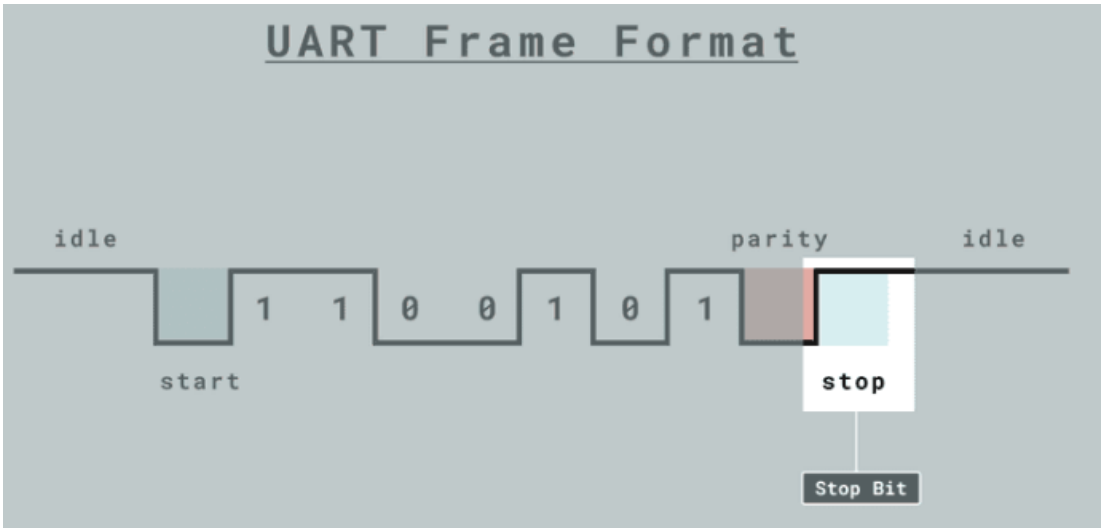

Figure 16: Stop bit

3. GENERAL LAYOUT DESIGN

After identifying the main components of the system, we divide the main components into functional clusters that undertake specific tasks.

To solve the problem in the topic, the generator speed measurement system is determined to include four functional clusters:

- Signal generation cluster
- Signal conditioning cluster
- Signal reading and frequency calculation cluster
- Display cluster

With the functional clusters arranged, we also need to determine the path of the signals and the interactions between the functional clusters.

In this section, we will solve the issues: General layout diagram.

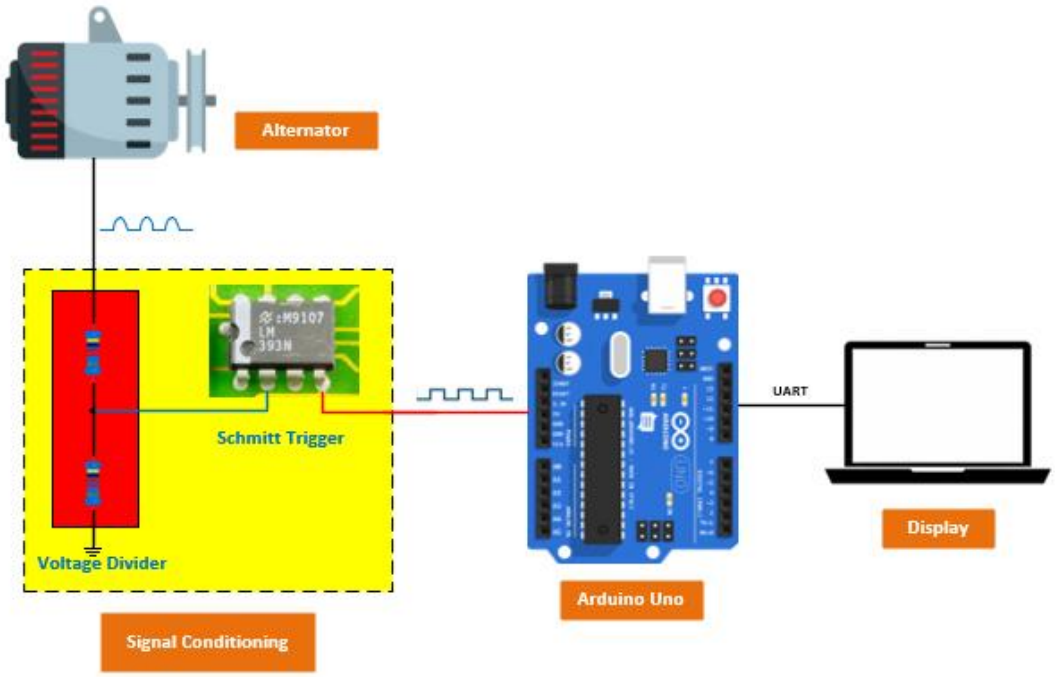

Figure 17: General layout diagram

- **Signal generation cluster:**
  - Generates a 3-phase AC voltage signal when the rotor rotates.
  - Output signal: AC (Analog) sine wave, containing information about the frequency (f), depending on the rotor rotation speed
- **Signal conditioning cluster:**
- Using a voltage divider circuit to match the input voltage level of the Arduino (0–5V).
- Convert the AC signal from the generator into a digital signal that the microcontroller can read.
- Use a Schmitt Trigger circuit to convert the sine wave signal into a square wave**.**
- **Signal reading and frequency calculation cluster:**
- Input signal: Square pulse signal (Digital Signal) from the signal conditioning block.
- Use Timer1 to measure the number of values that the Timer/Counter 1 has counted between two falling edges of the pulse.
- Calculate the rotational speed (RPM) based on the formula: $S = \frac{60 \times f}{P}$
- Output signal: Speed, T1, T2, Overf, f data as Serial Data.

- Transmit to: Display Block

- **Display cluster:**

- Input signal: Processed data from Microcontroller.
- Display measured parameters including: Speed, Frequency, Overflow, T1, T2
- Help users observe and evaluate measurement results.

4. TECHNICAL DESIGN

4.1. Voltage divider design

Because the Arduino UNO R3 maximum input voltage is 5V, so we have to use it in order to reduce the phase voltage, from nearly 14V to this value

❖ **Calculation:**

Choose $R_1 = 10K\ \Omega;\ R_2 = 6.8\ K\ \Omega$

Apply the formula for calculating $\boldsymbol{V_{out}}$**:**

$$V_{out} = V_{in} \times \frac{R_2}{R_1 + R_2}$$

4.2. Schmitt trigger circuit design

The output signals which is generated by the alternator are the sine wave form. However, the microcontroller is only capable of reading the digital signals, so that we have to convert the input signal to digital signal by using this circuit.

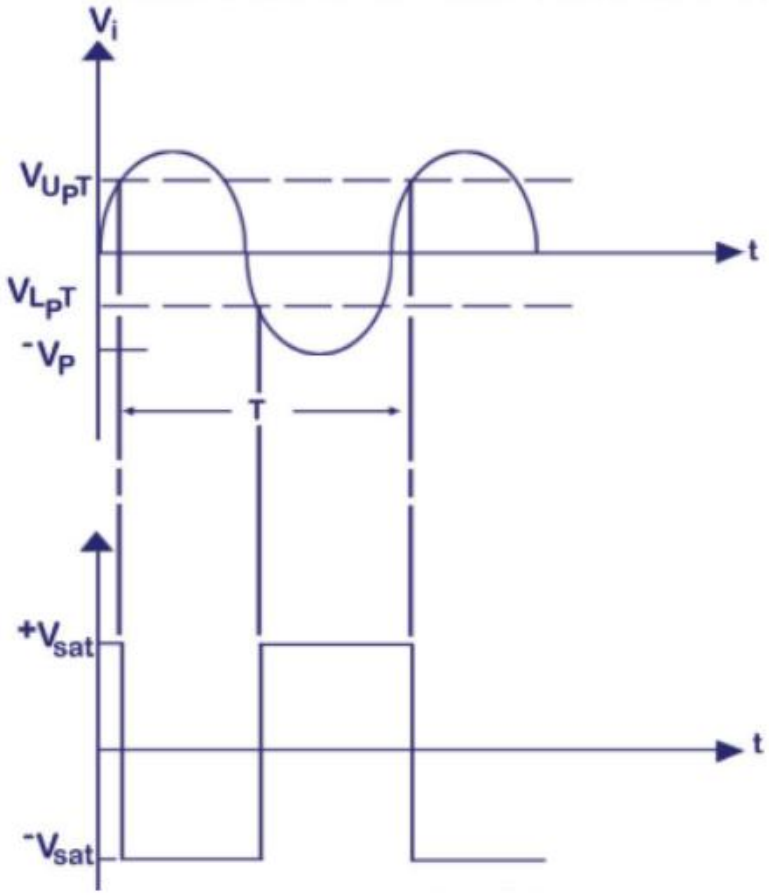

Figure 18: Schmitt trigger input and output waveform

When $V_{out} = +V_{sat}$, the voltage across R_{div1} becomes the Upper Threshold Voltage (V_{upt}). The input voltage, V_{in}, must be slightly more positive than V_{upt} to cause the output V_0 to switch from $+V_{sat}$ to $-V_{sat}$. When the input voltage is less than V_{upt}, the output voltage V_{out} is at $+V_{sat}$.

Upper Threshold Voltage: $V_{upt} = + V_{sat}\ (R_{div1} / [R_{div1} + R_{div2}])$

When $V_{out} = -V_{sat}$, the voltage across R_{div1} is called the Lower Threshold Voltage (V_{lpt}). The input voltage, V_{in}, must be slightly more negative than V_{lpt} to cause the output V_0 to switch from $-V_{sat}$ to $+V_{sat}$. When the input voltage is less than V_{lpt}, the output voltage V_{out} is at $-V_{sat}$.

Lower Threshold Voltage: $V_{lpt} = -V_{sat}\ (R_{div1} / [R_{div1} + R_{div2}])$

❖ **Calculation:**

Choose $R_3 = 10K\ \Omega$; $R_4 = 10K\ \Omega$; $R_5 = 10K\ \Omega$

Choose $V_{ref} = 5V; V_S = 5V$

Calculate the high and low threshold.

- High threshold:

$$R_{TOT1} = \frac{R_1 \times R_{FB}}{R_1 + R_{FB}}$$

$$V_{High\ thres} = V_S \times \frac{R_2}{R_2 + R_{TOT1}}$$

- Low threshold:

$$R_{TOT2} = \frac{R_2 \times R_{FB}}{R_2 + R_{FB}}$$

$$V_{Low\ thres} = V_S \times \frac{R_{TOT2}}{R_1 + R_{FB}}$$

4.3. Wiring diagram

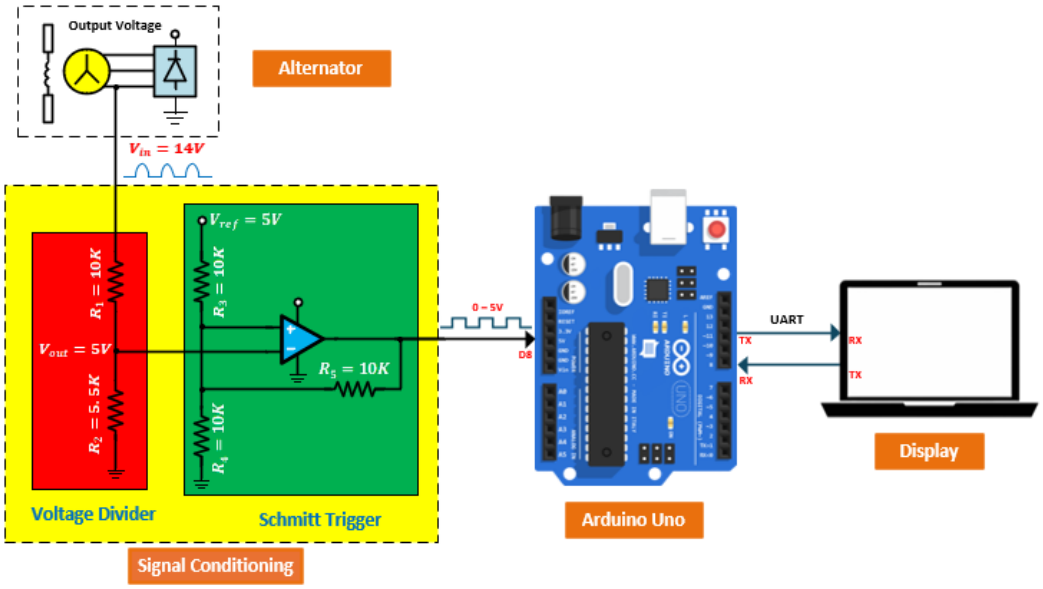

Figure 19: Wiring diagram of measuring system the speed of an alternator

First, the generator operates and generates a signal in the form of a sine wave, with an operating voltage of about 14V. Since the microcontroller only receives signals from 0 to 5V, this signal is passed through a signal processing circuit to convert the voltage from 14 to 5V. Here, I have passed it through a voltage regulator with two resistors of 10 ohms and 5.5 ohms respectively.

Since the generator signal is a sine wave, we need to process this signal into a square wave to suit the microcontroller. We will pass it through a schmitt trigger circuit to convert the signal from a sine wave to a square wave with an amplitude ranging from 0V to 5V.

Then, the microprocessor will read the signal and calculate the frequency, speed of this square wave and then send the value it reads to the screen for the user.

4.4. Timing diagram

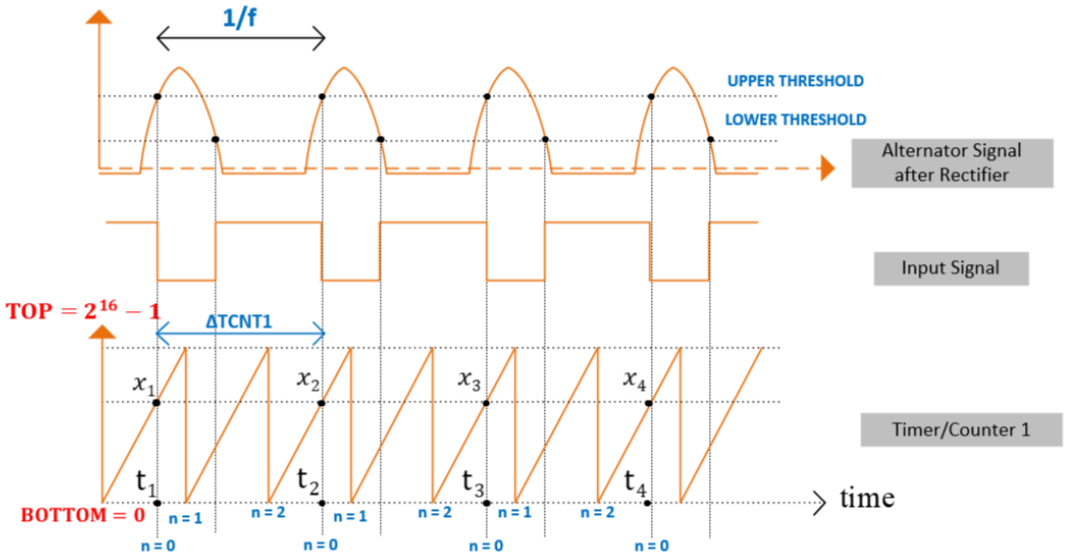

Figure 20. Timing diagram

The timing diagram describes the counting process of the Timer/Counter1 counter. Where:

$T_{cycle} = \frac{1}{f}$: the period of Alternator signal.

f: the frequency of Alternator signal

TOP: the peak value at which the Timer/Counter1 changes state.

BOTTOM: the lowest value that a Timer/Counter 1 reaches.

ΔTCNT1 : count value of Timer/Counter1 that counted from x_1 to x_2.

n: number of overflow.

t_1: at the falling edge of the speed signal.

t_2: at the next falling edge of the speed signal

x_1: count value of Timer/Counter 1 at t_1.

x_2: count value of Timer/Counter1 at t_2.

From the timing diagram above, we can calculate as follows:

Count value of Timer/Counter1:

$$\Delta TCNT1 = n \times TOP + x_2 - x_1$$

Frequency of alternator:

$$f = \frac{F}{Prescaler \times \Delta TCNT1}$$

Rotational speed of alternator:

$$S = \frac{60 \times f}{P}$$

Where:

$\Delta TCNT1$: count value of Timer/Counter1 that counted from x_1 to x_2.

$T_{cycle} = \frac{1}{f}$: the period of Alternator signal.

f: the frequency of Alternator signal

F: clock speed of Arduino ($= 16$ MHz)

Prescaler: frequency divider (= 64)

$\frac{F}{Prescaler} = 25 \times 10^4$(Hz): the frequency of Timer/Counter 1.

S: Rotational speed of an alternator

P: number of pair poles

4.5. Work flows

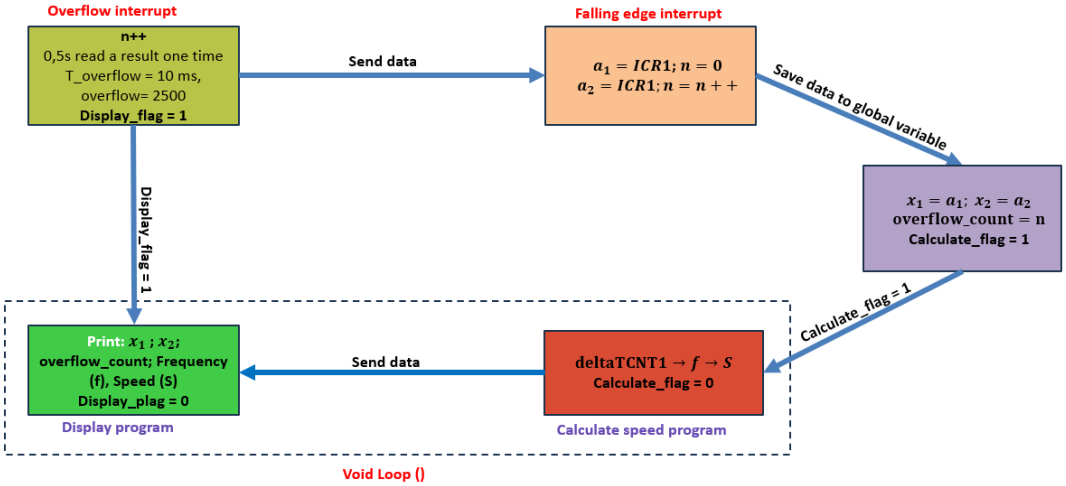

Figure 21. Work-flows

❖ **Priority and workload of each task when implementing the program:**

Fixed periodic repetition work:

- Check flag status
- Check if data from Timer is complete
- Display calculated data on screen

Random repetitive work:

- At t_1: Jump to interrupt program of the falling edge: $a_1 = ICR1$ and set $n = 0$.
- After, exit interrupt program and the program continues execution. Until Timer/Counter1 overflow, jump to interrupt program Overflow and n++
  (if n > 1 → n =0)
- At t_2: Jump to interrupt program of the next falling edge:

$a_2 = ICR1.2$ and after that, save $a_1 = ICR1; a_2 = ICR1$ and n, then save value into global variables.

calculate the value of: $\Delta TCNT1$ → Calculate f → Calculate S → Serial display (Delay 1000ms).

Reset: When: n != 0 → $a_1 = a_2$, n = 0

n = 0 → $a_1 = ICR1.1$ (ko update)

- Continues program, at t_3: Save x_1; a_3=$ICR1$. Save $a_2 = a_3$ and n.

Then, calculate:

$\Delta TCNT1$→ Calculate f →Calculate S → Serial display (Delay 1000ms)

Reset: When n != 0 → $a_1 = a_2$, n = 0

n = 0 → $a_1 = ICR1.2$ (ko update)

❖ **Advantage of work-flows:**

Workflow organization: Helps programmers and development teams understand the steps involved in a software project from start to finish. This includes planning, coding, testing, and deployment.

Increase productivity: A clear workflow helps avoid unnecessary repetition, reduces errors, and speeds up project completion.

Quality assurance: Workflows often include quality checks such as code review, automated testing, and continuous integration testing, ensuring that the source code meets certain standards.

Workflow assignment: For team projects, workflows help to properly divide work among members, define responsibilities, and track progress.

Version control: Workflows are often integrated with version control tools such as Git to track source code changes and handle conflicts when multiple people work on the same project.

Easy to extend and maintain: Clear workflows make it easier to extend and maintain the software, as the steps are standardized and documented.

Better collaboration: Workflows support effective communication between team members, reducing misunderstandings and enhancing coordination.

4.6. Algorithms

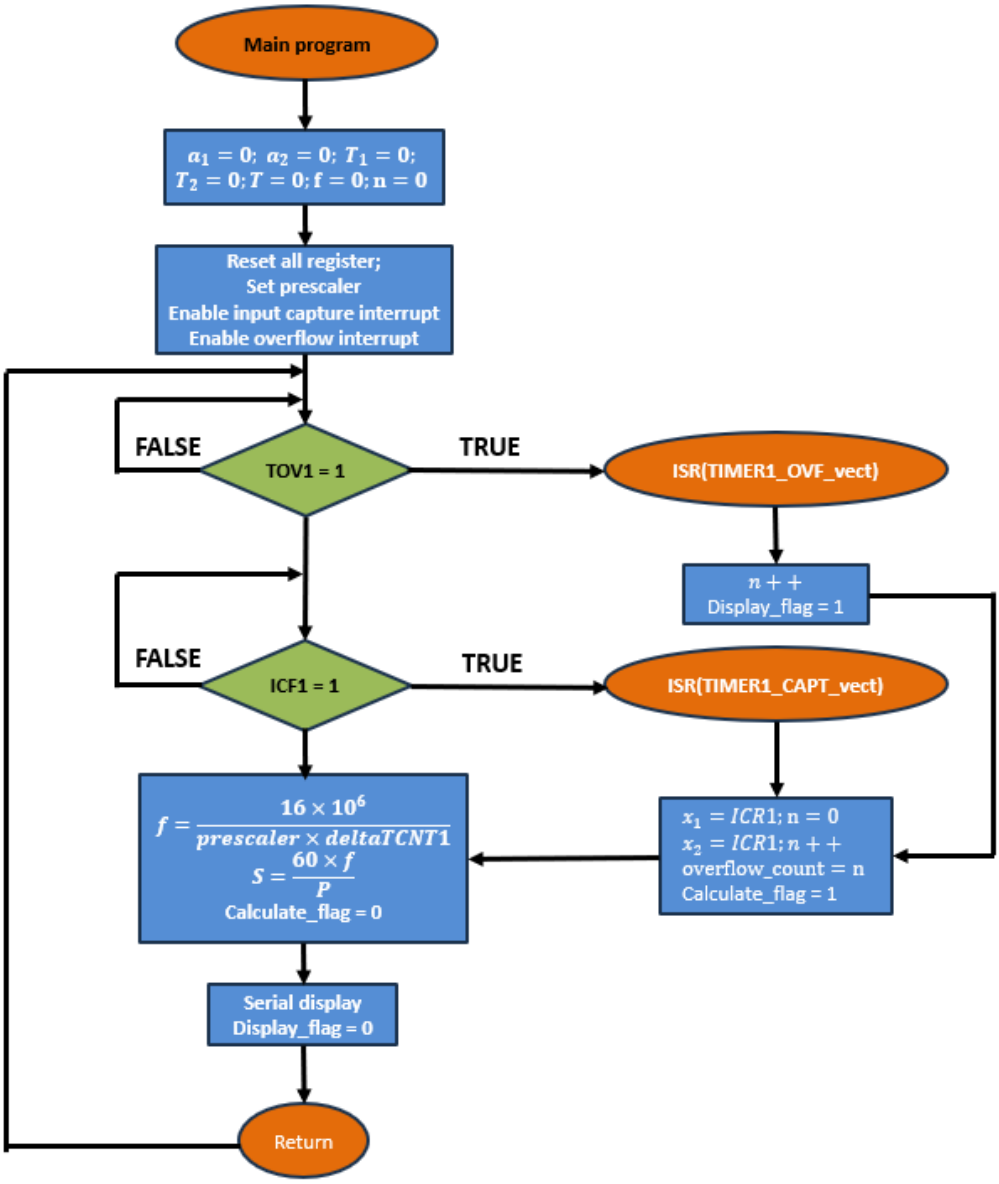

Figure 22. Algorithms

First is main program initialization

Setup function: (Void setup)

System initialization.

Timer1 configuration

Reset all register

Set prescaler

Enable input capture interrupt

Enable overflow interrupt

Interrupt function.

ISR: TIMER1_CAPT_vect

Activated automatically and occurs when there is a falling edge interrupt of the input signal.

Recording the time of the falling edge interrupt, it will record the value of a_1, a_2.

ISR: TIMER1_OVF_vect

Activated automatically and occurs when the Timer1 counter overflows, that is, when counting to 65535.

Loop function: (Void loop)

Check if there are enough triples of numbers x_1, x_2 and n to calculate the speed

Check the signal flag to display the value on the screen

Reset the variables to serve the calculation for the next cycle.

The function executes continuously throughout the system.

5. SIMULATIONS AND RESULTS

5.1. Const frequency case:

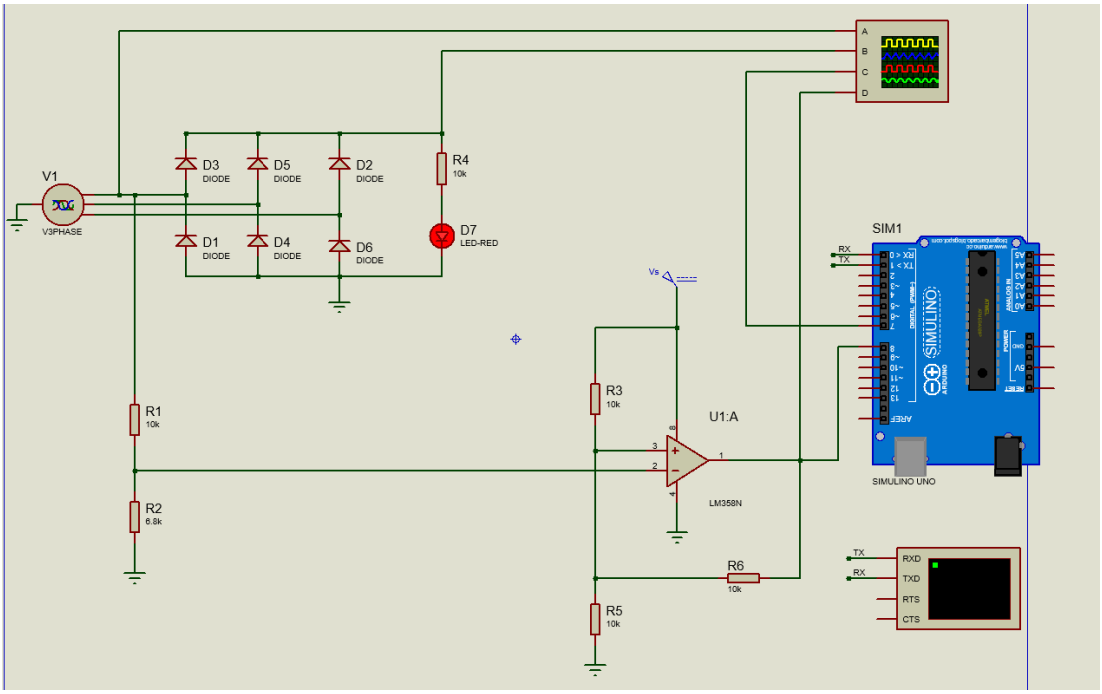

Figure 23. Program simulation on Proteus

Using the Proteus Software to simulate. In real life, alternator's RPM and frequency vary depending on the engine load. However, 3- phase generator model in Proteus just can run at a constant frequency, so it is hard to prove that our design running smoothly in practice.

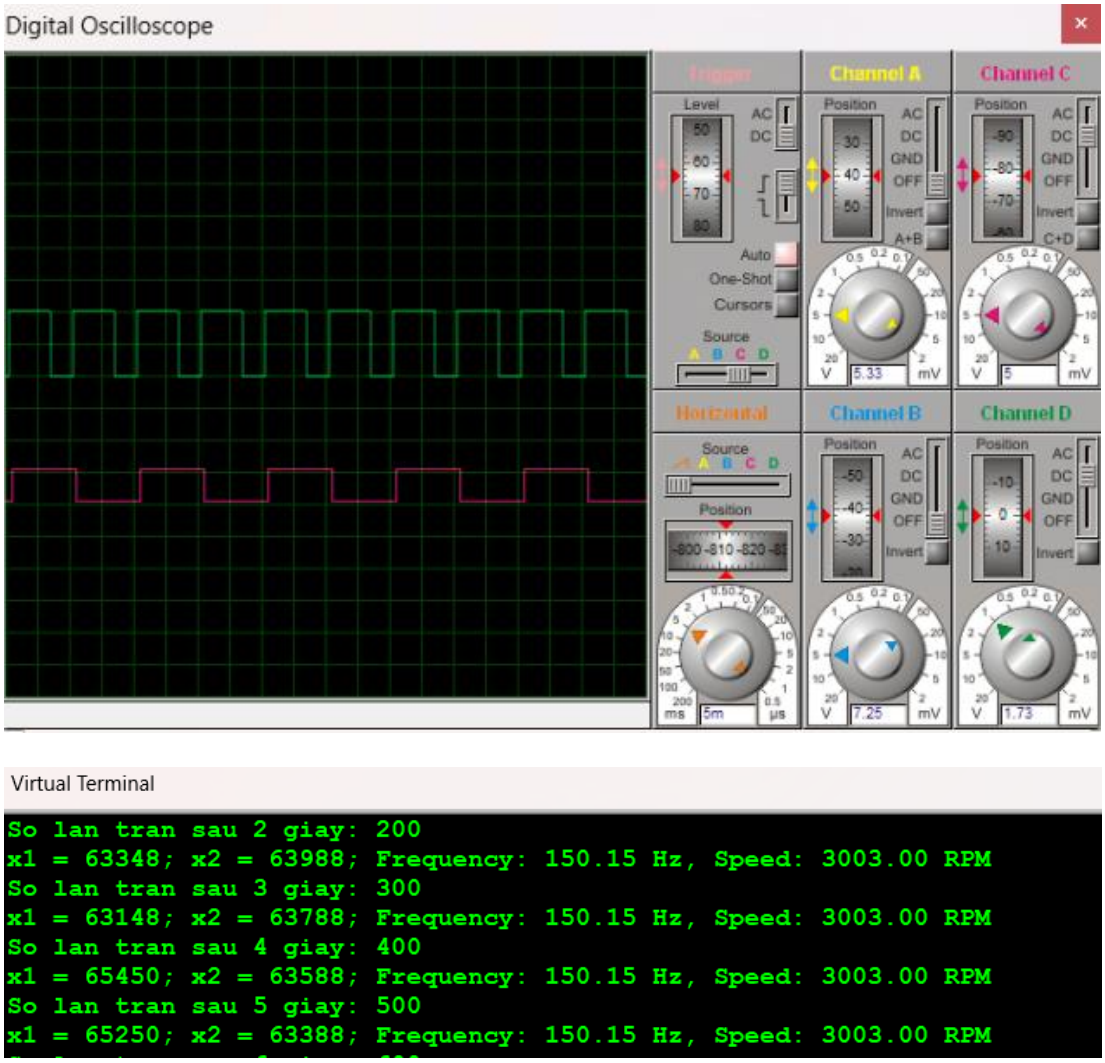

Figure 24. Results in const frequency case

Check timer overflow cycle:

The simplest way is to print the overflow count every second to observe and check whether the displayed overflow count is correct with the analyzed result.

Cycle of Timer/Counter1:

$$T = \frac{Prescaler \times (2^{16} - 1)}{16\ MHz}$$

Time for Timer/Counter1 to count from 0 to 65535:

$$T = \frac{64 \times 65535}{16\ MHz} = 0{,}262\ s$$

But the time we need to create is 0.01s so that overflow.

Therefore, we need $\frac{0.01s}{4\mu s} = 2500$ counts. Initial value of TCNT1 = 65536 - 2500 = 63035.

```
const unsigned int MAX_TIMER_COUNT = 2500;

ISR(TIMER1_OVF_vect)
{
    n++;
    overflow_count++;
    TCNT1 = 63035;
    toggle_state = !toggle_state;
    digitalWrite(7, toggle_state);

    if (overflow_count % 100 == 0) {
        elapsed_seconds++;
        display_flag = true;
    }
}
```

Figure 25. Code in Timer Interrupt

Here, I have set the Timer to overflow after 10ms, which means that after 1s it will overflow 100 times.

For example, overflow count displayed on Virtual Terminal:

after 2 seconds → 

Conclusion: timer operates with correct required overflow cycle.

Add one task:

In the program, I have added 1 task to check if the program runs stably or not.

Every time an overflow interrupt occurs, I will control the activation of pin D7 (Digital Output), if it is at level 1 then it will go down to level 0 and opposite.

If it happens as the above condition, then the program I organized is correct.

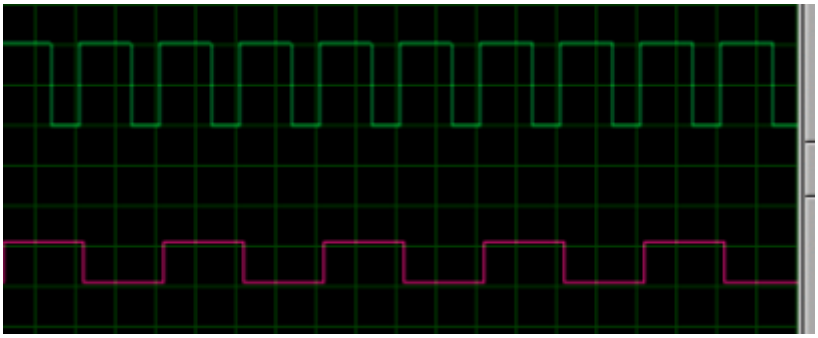

Figure 26. Program simulation on Proteus

Conclusion: The program run correct as requested.

Check the results:

F: Input Frequency (Hz).

$V = \frac{60 \times F}{P}$ (RPM), P = 3: number of pole pairs

F_2: measured frequency in fixed case

$V_2 = \frac{60 \times F_2}{P}$ (RPM)

$\%\Delta_2 = \frac{|F_2 - F|}{F}$: error when measuring in fixed case

Input frequency/Input Speed (F/S)	Real-time Frequency	
	Result display (F_1/S_1)	Error $\%\Delta_1$
100/2000	100.04/2000.80	0.04 %
120/2400	120.02/2400.38	0.02 %
150/2600	150.15/3003.00	0.06 %

Table 2. Results for the fixed frequency case show errors

The fixed frequency case: in this case, we will stop the simulation and change the fixed frequency to test whether our code run well.

5.2. Real-time frequency case:

In this case, we will use an activepotentiometer in Proteus to simulate the variation of frequency by changing the input voltage. The ADC function of Arduino will read and calculate the input voltage, then using that value to create a square wave by Arduino CTC mode for Timer1. Then we will recalculate our wave's frequency to test if the code is fit for practical or not.

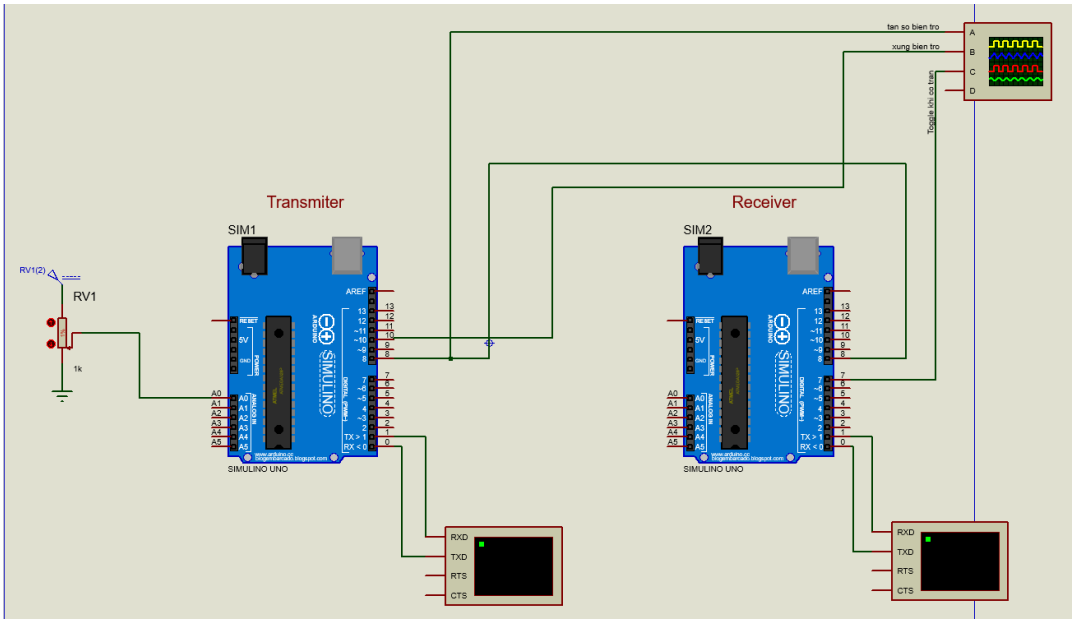

Figure 27. Program simulation on Proteus

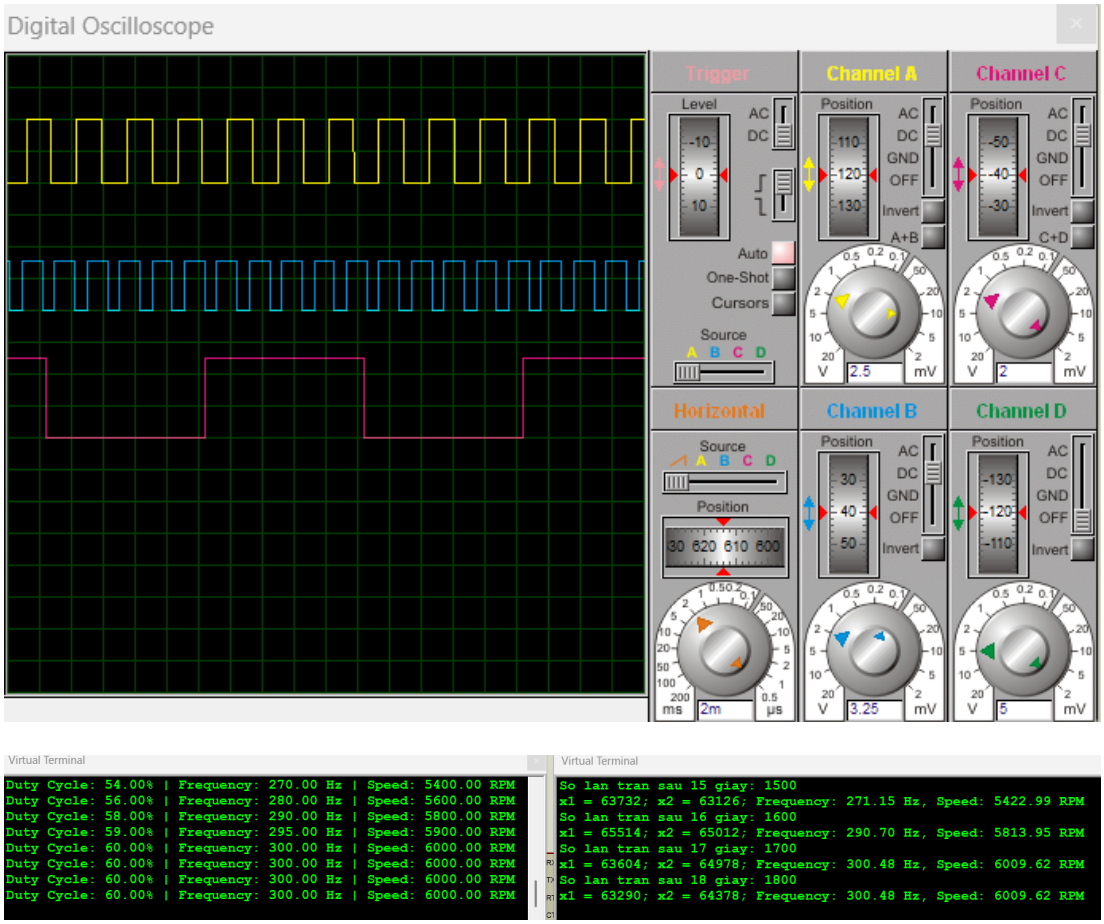

Figure 28. Results in real-time frequency case

Check the results:

Input frequency/Input Speed (F/S)	Real-time Frequency	
	Result display (F_1/S_1)	Error %Δ_1
150/3000	150.24/3004.81	0.13%
220/4400	220.07/4401.41	0.03%
300/6000	300.48/6009.62	0.009%

Table 3. Results for the real-time frequency case show errors

Discussion:

When executing this program, it is necessary to disregard the initial results, as the program requires 1-2 seconds to stabilize. After this period, the results can be read and utilized. Occasionally, the microcontroller may incorrectly capture the TCNT1 value,

leading to inaccurate calculations. This issue could stem from code imperfections or the microcontroller's limitations. However, if the differences between results are minor and their average aligns closely with the correct value, we can consider the program to be functioning reliably.

Based on the simulation, the program demonstrates a low error margin (<5%). While the error tends to increase at higher frequencies, within the measurement range of the alternator, the program performs effectively and can be deemed reliable.

6. CONCLUSION:

From the results obtained, we believe our design fulfills the required criteria.

Throughout the design process, we gained valuable insights.We learned to select and study appropriate materials for the tasks, apply theoretical knowledge to practical scenarios, and independently identify and correct errors during testing. Additionally, we developed a better understanding of the design workflow, formal report writing, and creating clear and effective presentations. However, we also encountered various challenges that required significant time to resolve. These issues arose from the simulation software, the program code, or other factors. Overcoming these obstacles has boosted our confidence, proving that with persistence and practice, we can tackle complex problems. This subject has been extremely beneficial for both our knowledge and personal growth.

Lastly, we extend our sincere gratitude to our instructor, Dr. Tran Dang Long, for his invaluable guidance throughout the design process. His advice and encouragement were instrumental in helping us overcome difficulties. Without his support, our results would not have been as successful.

7. REFERENCE

[1] ATmega48A/PA/88A/PA/168A/PA/328/P _ Mega AVR Data Sheet

[2] Trần Đăng Long _ AEES Material Powerpoint

[3] Wolfgang Ewald (2022), Timer and PWM – Part 2 (16 Bit Timer1), https://wolles-elektronikkiste.de/en/timer-and-pwm-part-2-16-bit-timer1

[4] Khaled Magdy (2020), Arduino Counter Timer Mode Tutorial & Code Examples, https://deepbluembedded.com/arduino-counter-timer/