Industrial IoT, Factory of Things

Capstone Report

Viktor Shkoda

Nazarbayev University
Department of Electrical and Computer
Engineering School of Engineering and Digital
Sciences

This project report was created on TexStudio editing platform using LaTeX. All the figures were drawn using draw.io online software tool.

Electrical and Computer Engineering
Nazarbayev University
http://www.nu.edu.kz

Title:
Industrial IoT, Factory of things

Theme:
Capstone Project Final Report

Project Period:
Spring 2024

Project Group:
1

Participant(s):
Viktor Shkoda

Supervisor(s):
Refik Kizilirmak

Copies: 1

Page Numbers: 19

Date of Completion:
April 26, 2024

Abstract:

This capstone project investigates the optimization of the MCP2515 CAN bus module for intra-vehicle communications in smart automotive systems. Employing Arduino-based prototypes, the study explores the impact of varying SPI clock speeds and transmission delays on the system's efficiency and reliability. Through experimental trials, the research evaluates how different settings influence the throughput and channel utilization of the CAN network, focusing on clock speeds of 5MHz and 2.5MHz. Results indicate that specific configurations significantly enhance communication effectiveness, showcasing an optimal balance of speed and stability. This study not only provides a detailed understanding of the CAN bus capabilities but also offers valuable insights for automotive engineers seeking to improve the safety and functionality of modern vehicles through advanced communication technologies.

Preface

This capstone report is the culmination of a comprehensive study into the optimization of the MCP2515 CAN bus module for enhancing intra-vehicle communications in smart automotive systems. Harnessing the capabilities of Arduino-based prototypes, this project delves into the intricate relationship between varying Serial Peripheral Interface (SPI) clock speeds, transmission delays, and the subsequent effects on the efficiency and reliability of communication within the CAN network.

As the world strides into an era where the Internet of Things (IoT) and smart factories (Factory of Things) become the backbone of industrial innovation, the significance of reliable and secure vehicle communication systems has never been more pronounced. The work presented in this report is a reflection of that significance and the relentless pursuit of technological advancement within the Nazarbayev University's Department of Electrical and Computer Engineering.

This report is not merely an academic submission but a testament to the relentless pursuit of knowledge, a practical contribution to the field of automotive security, and a potential stepping stone to further research and development. It encapsulates the challenges and learning experiences encountered throughout the project period of Spring 2024 and offers insights into the potential applications of the findings in real-world scenarios.

I express my sincere gratitude to my supervisor, Refik Kizilirmak, whose guidance was instrumental in navigating the complex web of design, implementation, and analysis. The encouragement and support of the entire academic staff provided an environment conducive to rigorous study and innovation.

I invite you to peruse this report not just as a summary of findings but as an exploration of the possibilities that lie within the integration of traditional automotive mechanisms with emerging digital technologies. It is my hope that this work will inspire continued research and ignite a broader discussion on enhancing the safety and functionality of the vehicles that are becoming increasingly integral to our daily lives.

Viktor Shkoda
April 26, 2024
Nazarbayev University

Introduction

Electrical demands for smart cars (autonomous cars) and smart systems in the field of intra-vehicle communications have been growing very rapidly in recent years with the development of automobiles. This is because of its low cost, effective serial mechanism, simplicity of installation, and capacity to improve real-time communication[1]. Since cars are the most common means of transportation used by people, their security has always been considered to be of the utmost importance[2]. The development of modern smart cars is now very much tied to electronic communications. The ECUs connected to the CAN bus are susceptible to attackers getting access to the bus due to design constraints[3]. Car thieves are now able to acquire remote access to vehicles instead of just stealing physical keys by taking advantage of weaknesses in complex automotive systems[4]. Every modern car is now equipped with at least 80 electronic communication unit(ECU)[5]. All of this is basically sensors that read the vehicle's condition, and all of them must simultaneously communicate with each other and the centre of the system. To do this, the automaker's campaigns use Controller Area Network(CAN) bus transmission technology. CAN is an embedded network protocol that is used in modern-day automobiles[6]. The Control Area Network (CAN bus) facilitates communication between all body parts, much like the nervous system[7]. Nodes or ECUs are connected via the CAN bus, serving as a central networking architecture. Any part of an automobile, including the airbags, audio system, and engine control unit, is connected by the CAN bus. However, CAN is vulnerable to hostile security assaults due to its facilitation of illegal entry, making CAN bus communication dangerous. In this paper, building a CAN bus transmission between the two nodes will be examined. To build the CAN system with the tow ECUs, the Arduino UNO, Arduino Nano, DTH11 sensor, I2C display and 2 MCMP2515 CAN module will be used. The MCP2515 CAN bus chip was designed as a standalone Controller Area Network to simplify applications. These programs were required to communicate with the Controller Area Network. The SPI protocol block, control logic and registers, and the CAN module are all packaged in a single chip. Microchip Technology's MCP2515 is a standalone CAN controller that adheres to CAN standard 2.0B.

Literature Review

A system to ensure reliability was put forth by Tanasa et al. [8] that re-transmitted the messages on the FlexRay Segment. The authors of [9] provided a FlexRay communication that was optimized using a MILP-based method that reduced endto-end latency and the number of occupied slots. Zhu et al. presented a task mapping-based optimization framework to achieve the goals of minimizing end-to-end latencies [10]. Nevertheless, it has been shown that the CAN bus is susceptible to remote attacks, endangering the vehicle's functionality and safety[11].The suggested security goals are not guaranteed by these aforementioned schemes. Methods for achieving security and performance were proposed by certain researchers. Lin et al Zhao et al. suggested a performance- and security-aware architecture for Time Division Multiple Access (TDMA), such as the FlexRay[8], [12]. TDMA-based systems and CAN-based systems have various security characteristics. In contrast to TDMA-based systems, the CAN-based systems believe that secret keys are shared between them. Despite the remarkable successes in security and timing requirements, it is challenging to directly apply these techniques to CAN-based devices.

The authors of [12] presented a method employing hardware/software co-design that offered internal communication secrecy in distributed real-time embedded systems. By utilizing

cartographic methods, the scheme satisfies confidentiality requirements and time restrictions, although it necessitates hardware reconfiguration.

The authors of [13] described a bus-off attack that makes use of the CAN protocol's error-handling structure. Through experimental testing, Palanca et al. established their denial-of-service attack approach and gave countermeasures [14]. Froeschle et al. conducted an analysis of CAN attacker capabilities to aid in the creation of a more secure in-vehicle network [15]. In conclusion, a number of research teams have put up plans in response to several CAN protocol threats.

Methods

The methodology of this capstone project was built upon a controlled experimental setup intended to measure the performance of a CAN bus system, particularly focusing on throughput and channel utilization under different configurations of the MCP2515 CAN controller.

Experimental Setup:

The setup comprised two nodes representing ECUs in a vehicle communication system. The transmitter, equipped with an Arduino Nano and an MCP2515 CAN module, sent data sourced from a DHT11 sensor. The receiver, utilizing an Arduino UNO and a matching MCP2515 module, was tasked with receiving the transmitted data. This arrangement aimed to emulate the data exchange processes that occur in automotive CAN networks.

Configuration:

A series of tests were conducted using varying SPI clock speeds, adjusted through clock dividers to obtain frequencies of 5MHz, 2.5MHz, 625KHz, 156.25KHz, and 78.125KHz. Transmission delays were systematically introduced at intervals of 1000ms, 100ms, 10ms, and 1ms to investigate their impact on network efficiency.

Measurements:

Two key metrics were measured:

1. **Throughput**, defined as the total amount of data successfully transmitted from the transmitter to the receiver over a specified period, was calculated using the formula:

$$Throughput = \frac{Total\ data\ transfered(byte)}{Total\ time}$$

2. **Channel Utilization**, representing the proportion of time the bus was actively engaged in data transmission as opposed to being idle, was determined by the formula:

$$Channel\ Utilization = \frac{Time\ with\ data\ on\ the\ bus}{Total\ time}$$

Data Analysis:

The collected data were analyzed to determine the efficiency of the CAN bus network under various experimental conditions. The analysis involved a comparative assessment of throughput and channel utilization across different SPI clock speeds and transmission delays.

Hardware Description

Arduino UNO

The Arduino board contains several microprocessors and controllers. Connecting different expansion boards (shields) and extra circuitry to the board's digital and analog input/output (I/O) pins is possible. In addition to supporting several USB versions, the board's serial communication interface enables the loading of applications from a computer. Most often, functional versions of the C and C++ programming languages are used to program microcontrollers. An Integrated Development Environment (IDE) based on Processing language project and standard compiler tool-chain are part of the Arduino project. An implemented Arduino UNO board is shown in Fig. 1

Figure 1: Arduino UNO

Arduino NANO

While the Arduino Nano and Arduino UNO have certain similarities, they also have some distinctions. The processors utilized by the Nano and UNO are different; the Nano uses an Atmel Atmega328, while the UNO uses an Atmega328P, which has more functionality including hardware serial communication. Whereas the UNO depends entirely on an external power source, the Nano has more flexible power options, supporting both USB and external sources. With 32 KB of flash memory in contrast to the Nano's 16 KB, the UNO has more storage space than the Nano. This distinction may affect tasks that call for a lot of programming or data storage. With equivalent input/output pins and support for protocols like SPI and I2C, both boards have comparable connectivity. However, the UNO has more overall connectors and pins, which is useful for projects requiring many inputs and outputs. An implemented Arduino NANO board is shown in Fig. 2

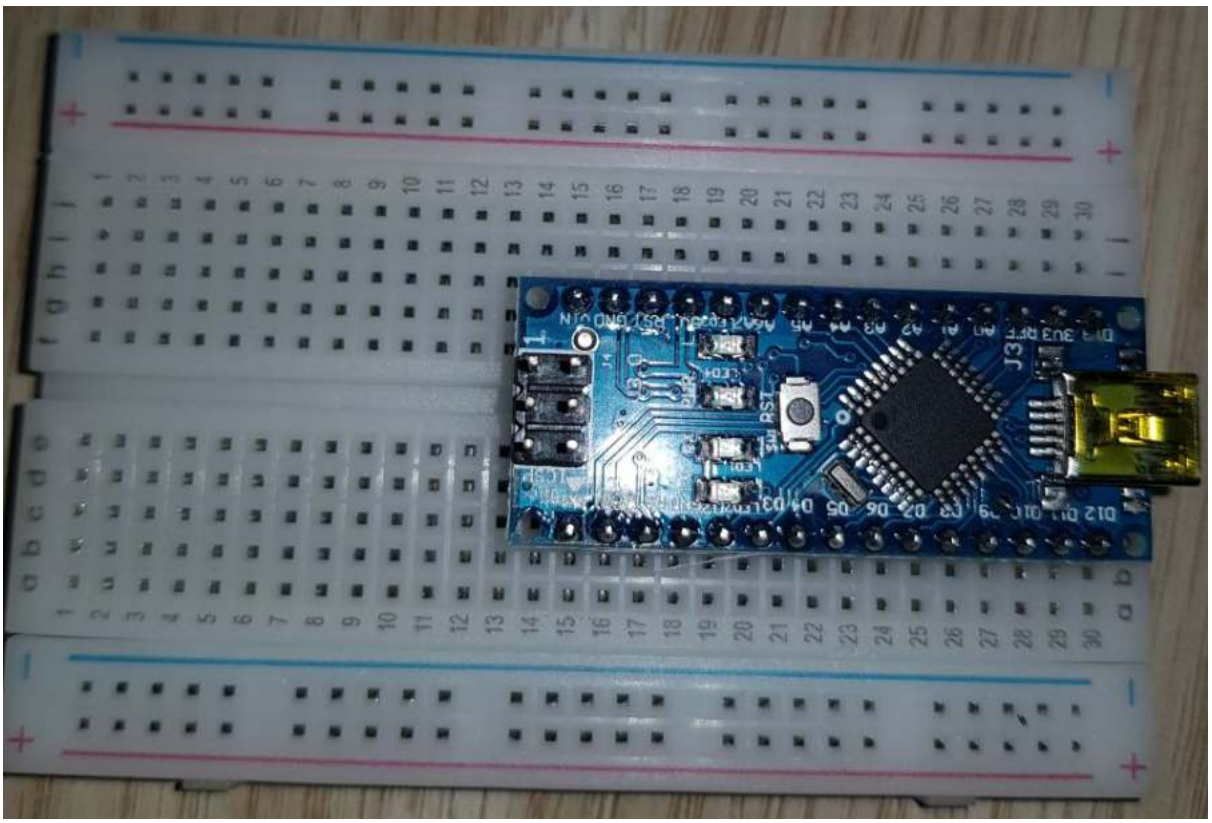

Figure 2: Arduino NANO

MCP2515 CAN module

This module enables the standard exchange of distant frames and extended data. Two acceptance masks and six acceptance filters assist the MCP2515 to reduce host MCU complexity and avoid unnecessary messages. The MCP2515 can communicate with industry-standard microcontrollers as well as Serial Peripheral Interface (SPI) devices. An implemented MCP2515 module is shown in Fig. 3

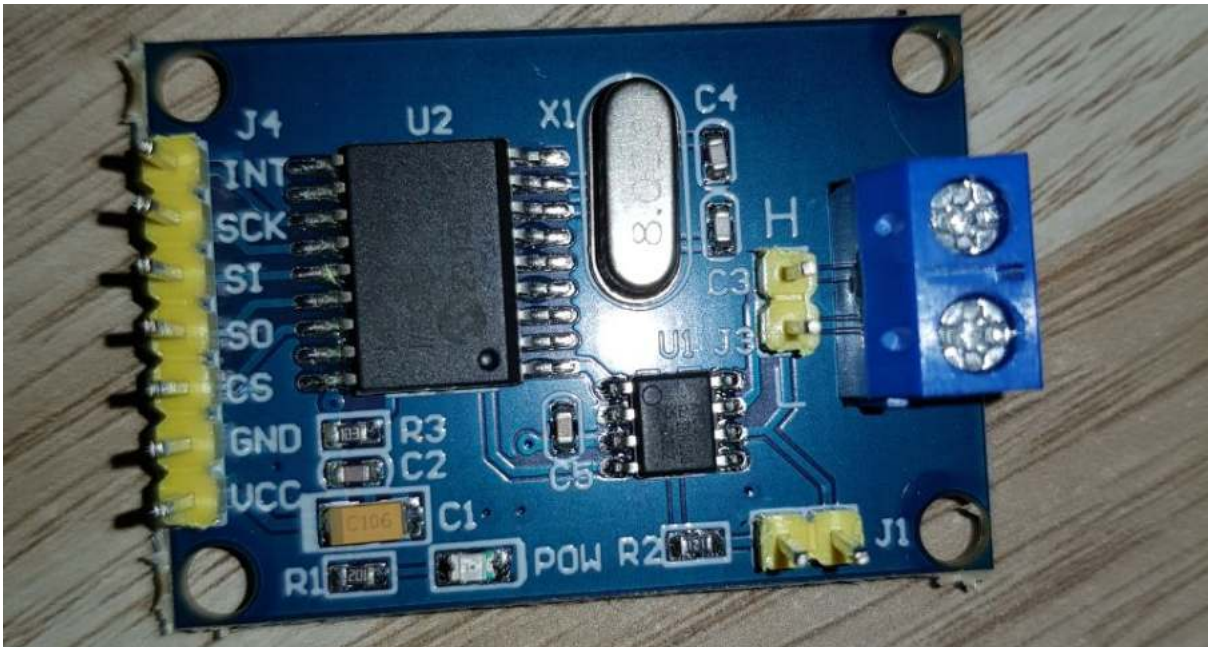

Figure 3: MCP2515

This module is composed of three main parts:

- The CAN module contains masks, filters, and transmit and receive buffers.

The CAN module manages all aspects of message transmission and reception on the CAN bus. Messages are sent once the appropriate message buffer and control registers have

been loaded. To begin transmission, either the control register bits over the SPI interface or the transmit enabling pins are used. To verify status and faults, the relevant registers can be read. Every CAN bus message is error-checked before being compared to the user-defined filters to see if it belongs in one of the two receive buffers. The CAN frame is show in the fig 4

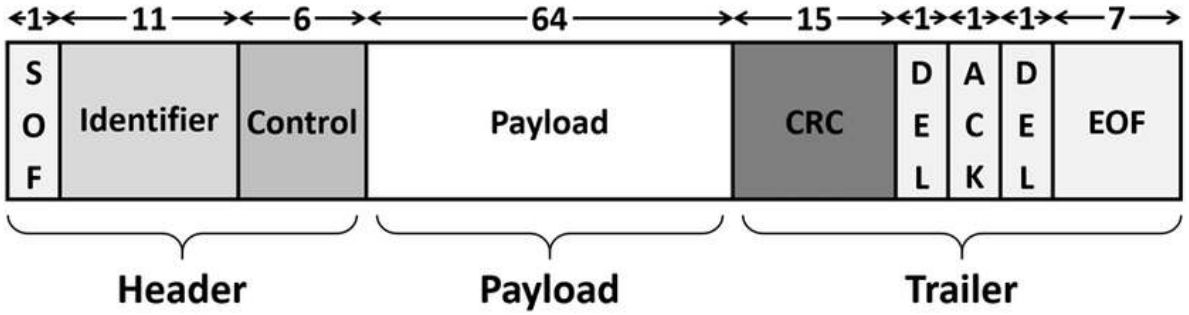

Figure 4: CAN frame

- Block of the SPI protocol

The SPI interface is responsible for connecting the MCU to the devices. To write to and read from all registers, standard SPI read and write instructions, as well as custom SPI commands, can be used.

- The control logic and registers

The control logic block configures and operates the MCP2515, interacting with the other blocks to convey control and information. Interrupt pins are introduced to the system to increase its adaptability. Each receive register has a single multipurpose interrupt pin (in addition to specific interrupt pins) that can be used to indicate that a valid message has been received and saved in a receive buffer.

I2C display

An I2C display is typically an OLED (Organic Light-Emitting Diode) or LCD (Liquid Crystal Display) with an I2C interface for straightforward integration with microcontrollers such as Arduino, Raspberry Pi, and other embedded systems. I2C is useful for displays because it requires fewer pins for communication than parallel interfaces, making it suitable for applications with a limited amount of pins. An implemented I2C display is shown in Fig. 5

Figure 5: I2C display

DHT11 sensor

An inexpensive, widely-used digital temperature and humidity sensor is the DHT11 sensor. Its dependability and simplicity make it a popular choice for a wide range of electronic

projects and Internet of Things (IoT) applications. An implemented DHT11 sensor is shown in Fig. 6.

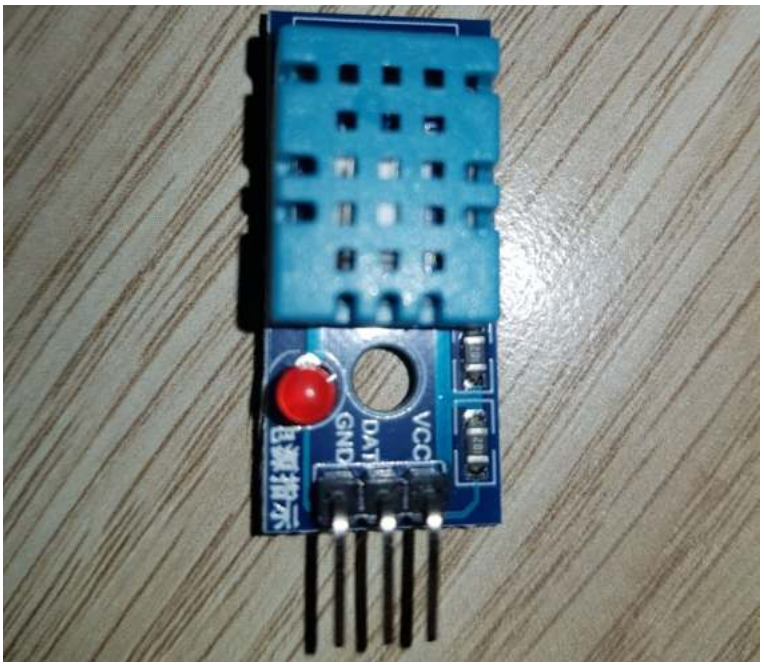

Figure 6: DHT11 sensor

Arduino codes for the receiver and transmitter ECU's are shown in the Appendix A.

Hardware implementation

The following modules are used in this work.

- Arduino UNO
- Arduino Nano
- Two MCP2515 modules
- I2C display
- DHT 11 sensor

The MCP2515 CAN Bus Controller is an integrated circuit that supports CAN Protocol 2.0 and data rates of up to 1Mbps. An inbuilt SPI interface allows microcontrollers to communicate with the independent CAN controller MCP2515 IC. The SPI Block, the Control Logic, and the CAN Module are the three main subcomponents that make up the MCP2515 IC, or principal controller. By connecting all the blocks, the Control Logic manages the configuration and functionality of the MCP2515. This integrated circuit is in charge of receiving data from the controller and sending it to the bus.

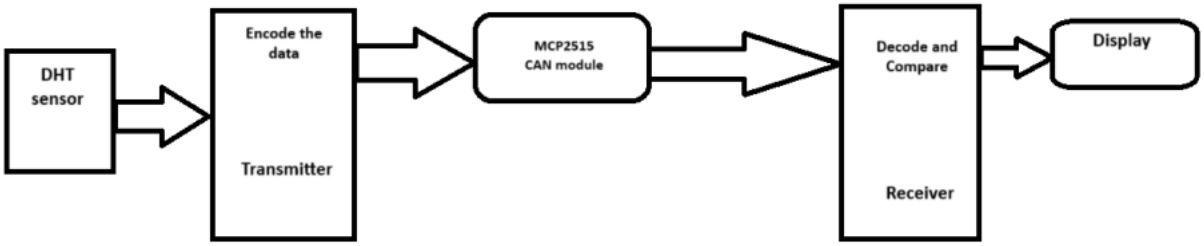

Figure 7: A Flow diagram of the CAN transmission system

A flow diagram of the implemented system is shown in Fig.7. There are two nodes in the system, and they both have unique properties. Node1 is configured as a receiver and consists of an Arduino UNO with an MCP2515 CAN module bus. Arduino NANO connected to the MCP2515 CAN module bus makes up Node2, which is configured as a transmitter. Node 1 receives the data from Node 2 after Node 2 transmits a different message to the CAN bus. All

the control units (nodes) are given a 5V DC supply at the time the system is turned on. The message is sent to the CAN bus by nodes that are connected to it (in this case, node 2). Node1 starts to receive the data from the CAN bus. Hardware implementation of the work is shown in the Fig.8..

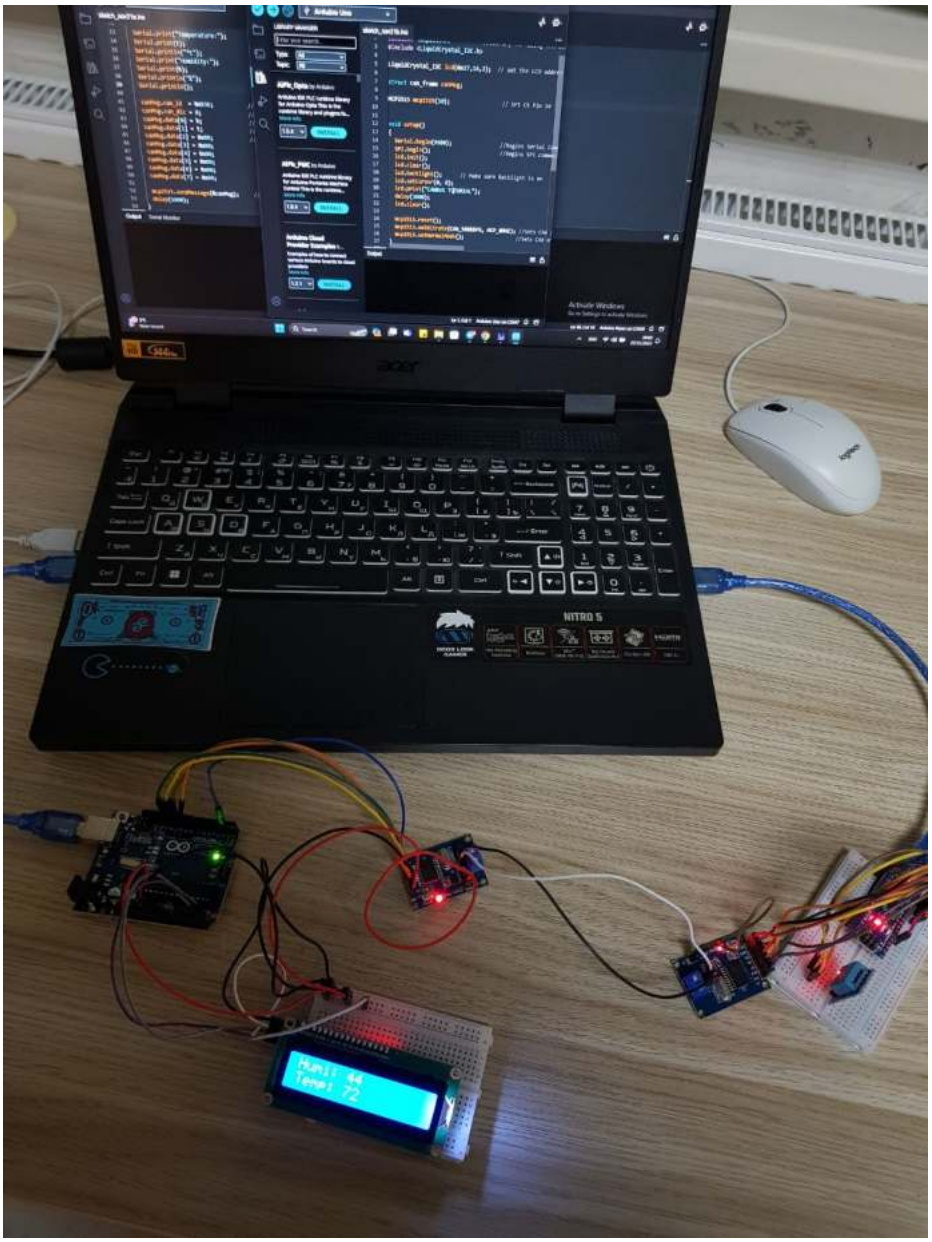

Figure 9: Hardware implementation picture

Results

The throughput of the CAN bus system was evaluated alongside channel utilization to provide a holistic view of the system's communication performance under various SPI clock speeds. Throughput was defined as the total data transferred (in bytes) over the total time (in seconds). The results depicted in the throughput measurement graph show an intriguing pattern across the different clock speeds. At a 1ms delay, the throughput readings reached the upper limit of 184 for all SPI clock speeds, indicating a possible saturation or measurement anomaly that warrants further investigation.

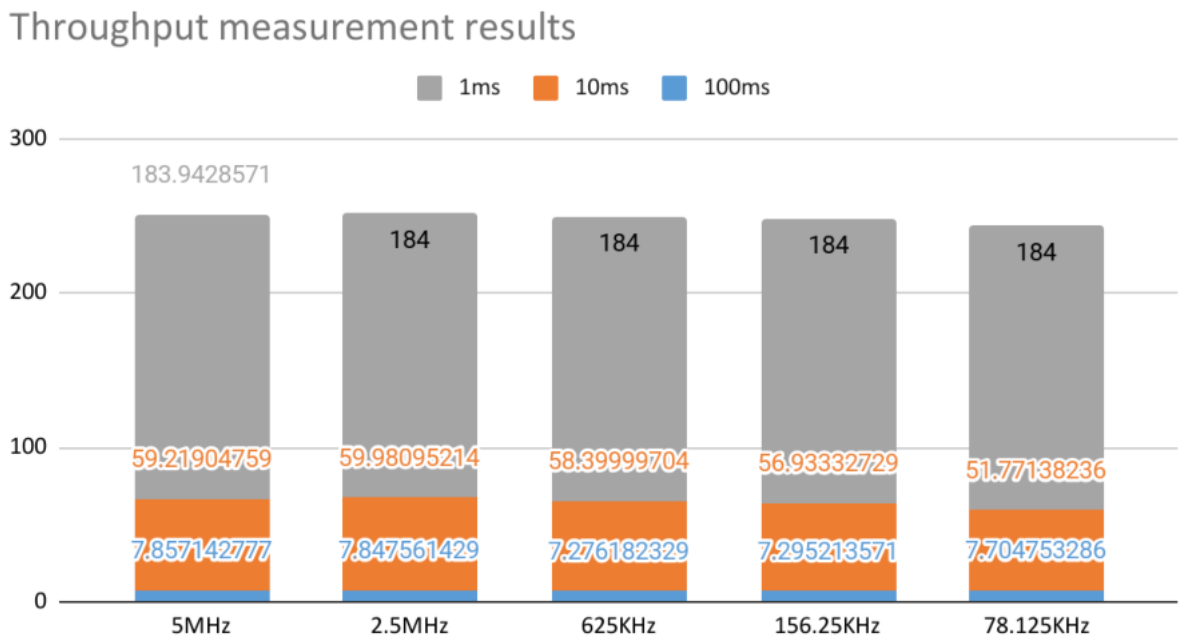

Figure 10:*Throughput measurement results*

In contrast, at 10ms and 100ms delays, the throughput values varied significantly with the SPI clock speed adjustments. For the highest clock speeds of 5MHz and 2.5MHz, the throughput recorded was substantially lower at approximately 59 bytes/sec and 7.8 bytes/sec for 10ms and 100ms delays, respectively. This could suggest that the CAN bus system is operating most efficiently at a moderate transmission delay where the balance between speed and data handling capability is optimized. As the SPI clock speed decreased, the throughput at 10ms delays displayed a gradual decline, dropping to its lowest at the 78.125KHz clock speed, with a measured value of approximately 51.77 bytes/sec. This decrease follows the expected trend where lower clock speeds typically result in reduced data transfer rates due to longer bit times.

Clock speed	**Channel utilization**
5MHz	62%
2.5MHz	62%
625KHz	59.8%
156.25KHz	58.8%
78.125KHz	58.8%

Table 1: Channel utilization results

The conducted experiment aimed to discern the relationship between SPI clock speeds and channel utilization within a CAN bus system. Channel utilization was determined as the ratio of the time the bus was active to the total time observed. The SPI clock speeds tested were 5MHz, 2.5MHz, 625KHz, 156.25KHz, and 78.125KHz. Contrary to expectations that a higher clock speed would result in greater channel utilization, the data revealed that both the 5MHz and 2.5MHz settings maintained a channel utilization rate of 62%. This suggests that within this

range, the CAN bus system can operate with high efficiency without being sensitive to changes in clock speed. A modest decrease in channel utilization was observed when the clock speed was reduced to 625KHz, with the channel utilization recording at 59.8%. The trend continued with further reductions in clock speed, with both the 156.25KHz and 78.125KHz frequencies showing a channel utilization of 58.8%. This indicates a plateau effect where the benefits of higher clock speeds do not significantly affect the efficiency of the network beyond a certain threshold. The experiment highlights the non-linear relationship between SPI clock speeds and channel utilization, demonstrating that within certain bounds, the CAN bus system's efficiency is not directly proportional to clock speed.

These findings underscore the importance of considering both throughput and channel utilization when optimizing a CAN bus system for specific applications. While channel utilization slightly increains to the across higher clock speeds, throughput displayed sensitivity to changes in both clock speed and transmission delay, highlighting a complex interplay between these parameters that affects the overall system performance.

Conclusion

This capstone project investigated the impact of various SPI clock speeds on the channel utilization and throughput of a CAN bus system, which is pivotal in modern automotive communication. The experiment revealed nuanced behaviors in the system performance, particularly highlighting the robustness of channel utilization across higher clock speeds and the intricate variations in throughput influenced by both clock speed and transmission delay. The channel utilization results showed stability at 62% when tested with SPI clock speeds of 5MHz and 2.5MHz, irrespective of the doubling of clock speed. This suggests an optimal operation range for the CAN bus system that could facilitate efficient data communication without a proportionate increase in channel activity. As the clock speed was further reduced, only a minor decrease in channel utilization was observed, indicating that lower speeds slightly compromise the system's efficiency. In terms of throughput, a surprising constant value was observed across all SPI clock speeds at the smallest transmission delay of 1ms, suggesting a possible measurement ceiling or system constraint. Meanwhile, at more considerable delays of 10ms and 100ms, throughput experienced expected decreases as clock speeds were lowered, reflecting the conventional understanding that higher speeds generally enable faster data transmission. These outcomes imply that while channel utilization can maintain a high level even when the clock speed is reduced by half, throughput is more sensitive to changes in both the clock speed and transmission delay. The implication for automotive system design is profound: optimizing CAN bus performance does not necessarily require the highest clock speeds, particularly when considering factors such as system cost and electromagnetic compatibility. Ultimately, the project's findings contribute valuable insights into the design and optimization of CAN bus systems for automotive engineers. While aiming for high channel utilization is crucial, understanding the limits of throughput across different operational settings can guide more nuanced and efficient designs. This balance is essential as the industry continues to evolve towards increasingly connected and autonomous vehicles.

Reference list

[1] H. Taslimasa *et al.*, “ImageFed: Practical privacy preserving intrusion detection system for in-vehicle can bus protocol,” *2023 IEEE 9th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, May 2023. doi:10.1109/bigdatasecurity-hpsc-ids58521.2023.00031

[2] P. Wei, B. Wang, X. Dai, L. Li, and F. He, “A novel intrusion detection model for the CAN bus packet of in-vehicle network based on attention mechanism and autoencoder,” *Digital Communications and Networks*, vol. 9, no. 1, pp. 14–21, Feb. 2023. doi:10.1016/j.dcan.2022.04.021

[3] C. B. de Melo and N. Dutt, “LOCoCAT: Low-overhead classification of CAN bus attack types,” *IEEE Embedded Systems Letters*, vol. 15, no. 4, pp. 178–181, Dec. 2023. doi:10.1109/les.2023.3299217

[4] J. A. Khan, D.-W. Lim, and Y.-S. Kim, “A deep learning-based ids for automotivetheft detection for in-vehicle can bus,” *IEEE Access*, vol. 11, pp. 112814–112829, 2023. doi:10.1109/access.2023.3323891

[5] AutoPi.io, “Can bus uncovered: The Ultimate Guide for Automotive Enthusiasts,” AutoPi.io, https://www.autopi.io/blog/can-bus-explained/ (accessed Nov. 22, 2023).

[6] L. E. Frenzel, “Chapter Eight - Controller Area Network (CAN),” *ScienceDirect*, Jan. 01, 2016. https://www.sciencedirect.com/science/article/pii/B9780128006290000085 (accessed Nov. 22, 2023)

[7] S. Lakshmi and R. H. Kumar, "Secure Communication between Arduinos using Controller Area Network(CAN) Bus," 2022 IEEE International Power and Renewable Energy Conference (IPRECON), Kollam, India, 2022, pp. 1-6, doi: 10.1109/IPRECON55716.2022.10059518.

[8] B. Tanasa, U. Dutta Bordoloi, P. Eles, and Z. Peng, “Reliability-aware frame packing for the static segment of flexray,” in Proc. 9th ACM Int. Conf. Embedded Softw., 2011, pp. 175–184.

[9] R. Zhao, G. Qin, H. Chen, J. Qin, and J. Yan, “Security-aware scheduling for flexray-based real-time automotive systems,” Math. Problems Eng., vol. 2019, 2019, Art. no. 4130756.

[10] C.-W. Lin, Q. Zhu, and A. Sangiovanni-Vincentelli, “Security-aware mapping for tdma-based real-time distributed systems,” in Proc. IEEE/ACM Int. Conf. Comput.-Aided Des., 2014, pp. 24–31.

[11] Devnath, Maloy Kumar. (2023). GCNIDS: Graph Convolutional Network-Based Intrusion Detection System for CAN Bus. 10.48550/arXiv.2309.10173.

[12] K. Jiang, P. Eles, and Z. Peng, “Co-design techniques for distributed real-time embedded systems with communication security constraints,” in Proc. Conf. Des., Autom. Test Eur.. EDA Consortium, 2012, pp. 947–952.

[13] K.-T. Cho and K. G. Shin, “Error handling of in-vehicle networks makes them vulnerable,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., 2016, pp. 1044–1055.

[14] A. Palanca, E. Evenchick, F. Maggi, and S. Zanero, “A stealth, selective, link-layer denial-of-service attack against automotive networks,” in Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment, 2017, pp. 185–206.

[15] S. Froschle and A. Stuhring, “Analyzing the capabilities of the can ¨ attacker,” in Proc. Eur. Symp. Res. Comput. Secur. Springer, 2017, pp. 464–482.

Appendix A

Transmitter code:

```
#include <SPI.h>          //Library for using SPI Communication

#include <mcp2515.h>      //Library for using CAN Communication

#include <DHT.h>          //Library for using DHT sensor

#define DHTPIN 8

#define DHTTYPE DHT11

struct can_frame canMsg;

MCP2515 mcp2515(10);

DHT dht(DHTPIN, DHTTYPE);      //initilize object dht for class DHT with DHT pin
with STM32 and DHT type as DHT11

void setup()

{

  while (!Serial);

  Serial.begin(9600);

  SPI.begin();               //Begins SPI communication

   dht.begin();                   //Begins to read temperature & humidity sesnor
value

  mcp2515.reset();

   mcp2515.setBitrate(CAN_500KBPS, MCP_8MHZ); //Sets CAN at speed 500KBPS and
Clock 8MHz

  mcp2515.setNormalMode();

}

void loop()

{

  int h = dht.readHumidity();       //Gets Humidity value

  int t = dht.readTemperature();    //Gets Temperature value



  Serial.print("Temperature:");

  Serial.print(t);

  Serial.println("*C");
```

```
  Serial.print("Humidity:");
  Serial.print(h);
  Serial.println("%");
  Serial.println();
  canMsg.can_id  = 0x036;           //CAN id as 0x036
  canMsg.can_dlc = 8;               //CAN data length as 8
  canMsg.data[0] = h;               //Update humidity value in [0]
  canMsg.data[1] = t;               //Update temperature value in [1]
  canMsg.data[2] = 0x00;            //Rest all with 0
  canMsg.data[3] = 0x00;
  canMsg.data[4] = 0x00;
  canMsg.data[5] = 0x00;
  canMsg.data[6] = 0x00;
  canMsg.data[7] = 0x00;
  mcp2515.sendMessage(&canMsg);     //Sends the CAN message
  delay(1000);
}
```

Receiver code:

```
#include <SPI.h>
#include <mcp2515.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27,16,2);
struct can_frame canMsg;
MCP2515 mcp2515(10);
void setup()
{
  Serial.begin(9600);
  SPI.begin();                       n
  lcd.init();
  lcd.clear();
```

```
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("CANBUS TUTORIAL");
  delay(3000);
  lcd.clear();
  mcp2515.reset();
  mcp2515.setBitrate(CAN_500KBPS, MCP_8MHZ);
  mcp2515.setNormalMode();
}
void loop()
{
  if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR_OK)
    int x = canMsg.data[0];
    int y = canMsg.data[1];
    lcd.setCursor(0, 0);
    lcd.print("Humi: ");
    lcd.print(x);
    lcd.setCursor(0, 1);
    lcd.print("Temp: ");
    lcd.print(y);
    delay(1000);
    lcd.clear();
  }
}
```

Appendix B

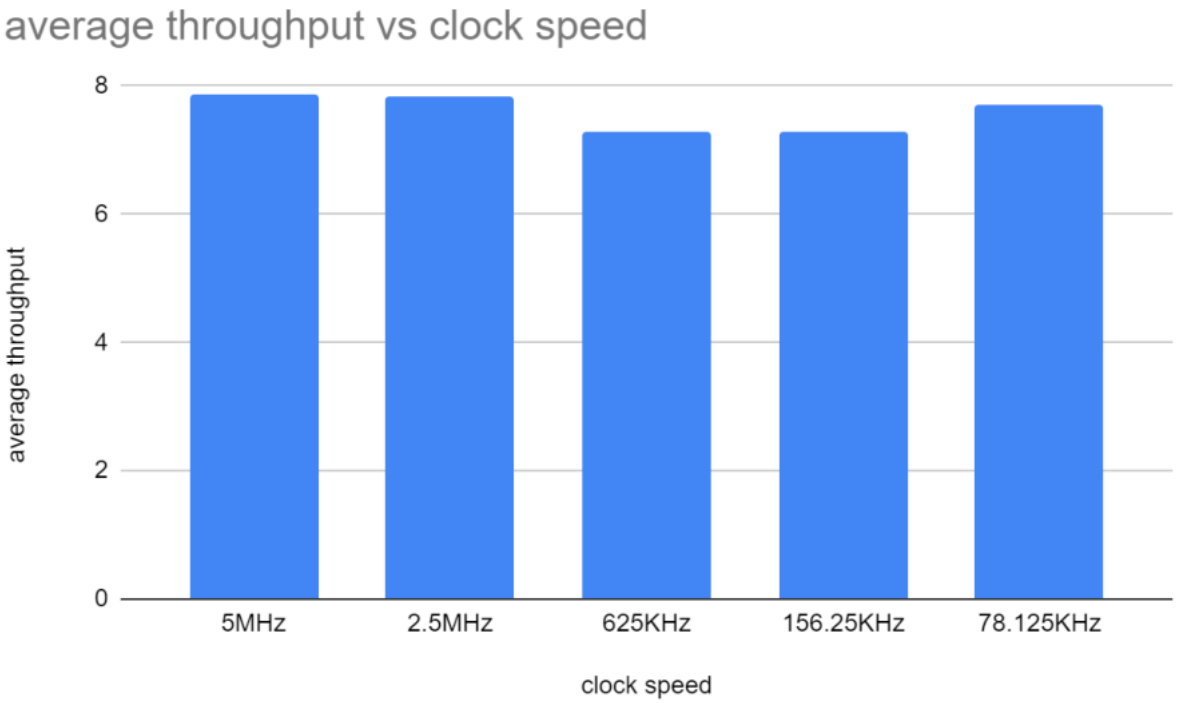

Figure B1: Throughput delay 100ms

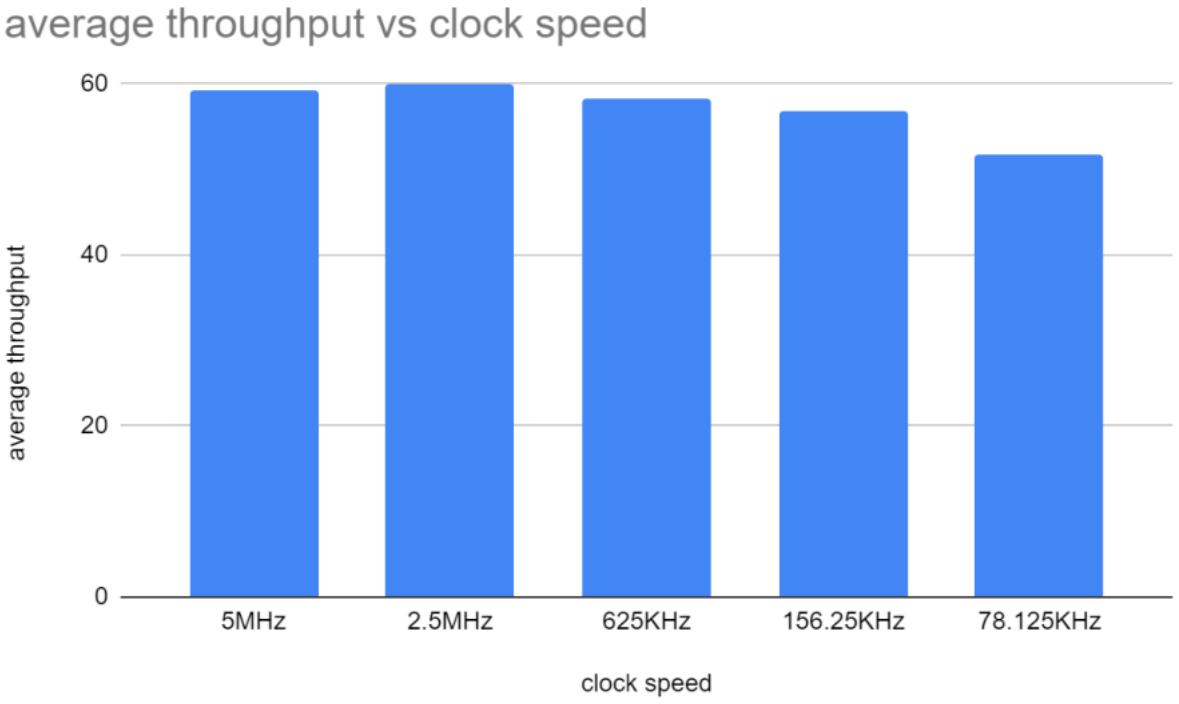

Figure B2: Throughput delay 10ms

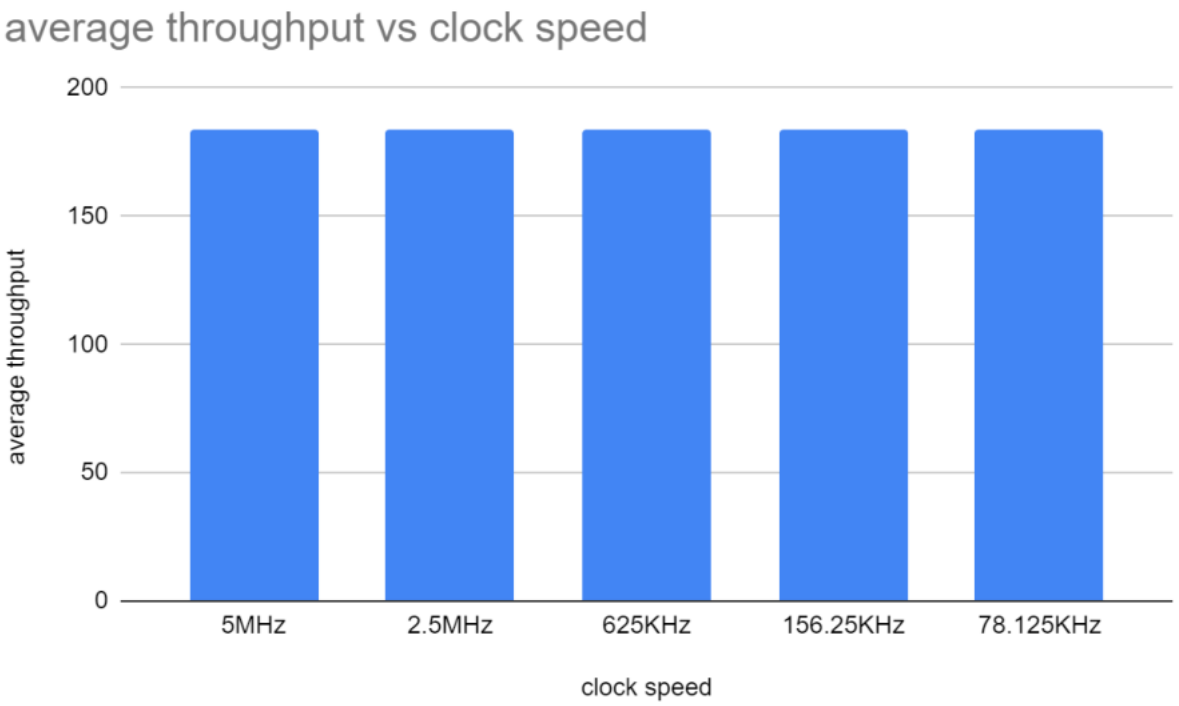

Figure B3: Throughput delay 1ms