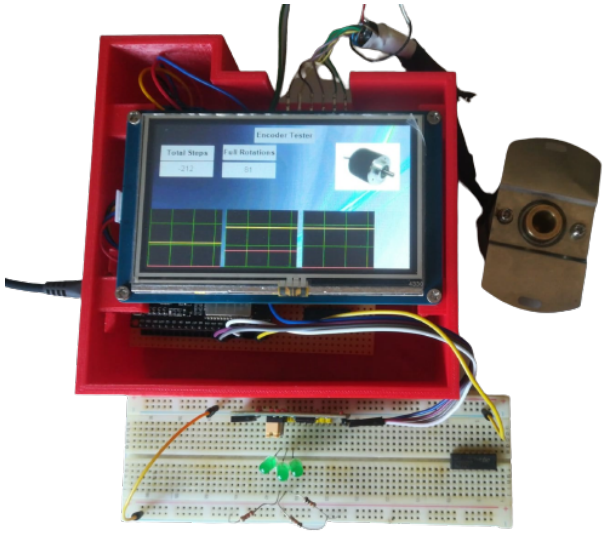
Universidade de Aveiro

Tecnologias de acionamento e comando

Encoder Tester - Final project of TAC



93068 - André Filipe Correira Cardoso

January 15, 2023

universidade de aveiro

Keywords

Encoder	ESP32	Arduino IDE
I2C	SPI	RS232

Abstract

Encoders are helpful devices to implement in all kinds of machines that move and need precision. For example, all CNCs machines require precision in the order of microns, autonomous vehicles and robots. While being a great piece of engineering they have to deal with harmful circumstances, oils, electrical noises and other harsh conditions. During a summer internship, a need to test many encoders was encountered without having a proper tool to test them other than an oscilloscope which isn't as portable. With this in mind, an encoder tester was developed that is able to read six channels of the encoder while counting steps and full rotations provided by the z channel.

Contents

1. Objectives **3**

2. Components **3**

3. Electrical Diagram **5**

4. Communications **5**

4.1 RS232 5

4.2 I2C 6

4.3 SPI 7

5. Usage Description **9**

6. Demonstration **9**

1. Objectives

Encoders are helpful devices to implement in all kinds of machines that move and need precision. For example, all CNCs (Computerized Numerical Control) machines that require precision in the order of microns, autonomous vehicles and robots. To achieve such accuracy, the motors used are usually servomotors that use an encoder to perceive their global position. While the encoders are of great benefit they require maintenance, since they work in a harsh environment with a lot of electrical noise, making it cumbersome to verify their work with tools such as an oscilloscope.

With this in mind, a new prototype for testing optical encoders was developed consisting of a microprocessor that will capture the signals and display this information on an LCD (Liquid Crystal Display). While displaying captured information, it also saves it into an SD (Secure Digital) card and shows it using LEDs (light-emitting diode) with the help of an I/O (Input/Output) extender for future use in testing also the movement of motors.

2. Components

For the development of this project, the components used in its build are:

- Esp32
- LCD `nx4827t043_011`
- Comparator lm339
- Expander Module I/O pcf8574
- MH-SD Card Module
- Any Encoder
- LED

This project based its build on the usage of existing microcontrollers that include communication interfaces such as RS-232 (Recommended Standard 232), I2C (Inter-Integrated Circuit) and SPI (Serial peripheral interface) already built in at low cost. The chosen microcontroller was ESP32, produced by Expressif Systems, which includes two high-speed Xtensa cores with up to 240 Mhz, which will be used to implement the interrupts in one of the cores while the other is printing the information to the LCD. This microcontroller also includes Wi-Fi (Wireless Fidelity) and Bluetooth, making an interesting addition in the future to use the phone as a visualizer. The Esp32 pinout can be seen in Figure 1, which describes the type of each pin.

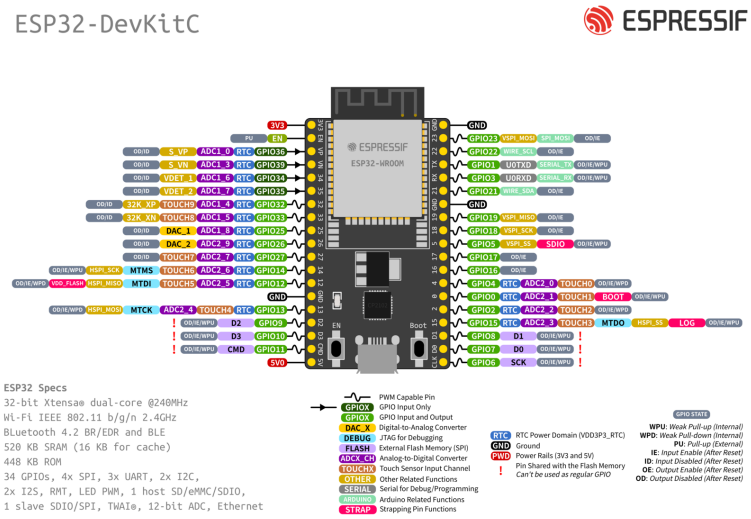


Figure 1: Esp32 Pin out [1]

An LCD was used to display the information, specifically the `nx4827t043_011` module made by Nextion. Nextion is a seamless HMI (Human Machine Interface) solution that provides a control and visualization interface between a human and a process machine application or appliance. Nextion is primarily used in the area of IoT (Internet of Things) or consumer electronics. Users can create and design their Nextion display interfaces with Nextion Editor software, which allows them to develop and design interfaces.

To read the channels of the encoder, two comparators were used, since the encoder can be powered with higher voltages than those that ESP32 supports. These quad-comparators offer high-gain and wide-bandwidth properties with offset voltage specifications as low as 2 mV max voltage. Each comparator has been explicitly designed to operate from a single power supply over a wide range of voltages. The operation of split power supplies is also possible.

The PCF8574 device provides general-purpose remote I/O expansion for most microcontroller families through the I2C interface, SCL (serial clock) and SDA (serial data). This component allows this application to control stepper motors when the encoder moves, testing both the encoder and stepper motor. This device turns on the LED corresponding to the respective encoder channel.

The MH-SD Card Module will provide the registered information storage in an SD card by using SPI communication, allowing the user to visualize later and replay the saved data to check the encoder functionality or even replay it with a motor. A standard microSD card has an operating voltage of 3.3 V. As a result, it is impossible to connect it directly to circuits that use 5V logic; in fact, any voltage above 3.6V may permanently damage the microSD card. That is why the module includes an onboard ultra-low dropout voltage regulator capable of regulating voltage to 3.3V. This module also comes with a logic level shifter chip that allows safe and easy communication with a 3,3 V or 5 V microcontroller without damaging the SD card.

The last essential component is the actual component to be tested. This can range from optical encoders that have two channels up to six channels and can have logic values higher than 3.3V due to the use of comparators.

3. Electrical Diagram

The components mentioned in Section 2. are connected following the diagram in Figure 6. It clearly shows the flow of the device in terms of where it begins and ends. Firstly, it starts by capturing the encoder's signals and processing them with the comparator sending them afterward to the ESP where all the logic part is. Afterward, processing the information, it presents it to the user by displaying an LED/controlling a motor, storing it in an SD card or displaying it in an LCD.

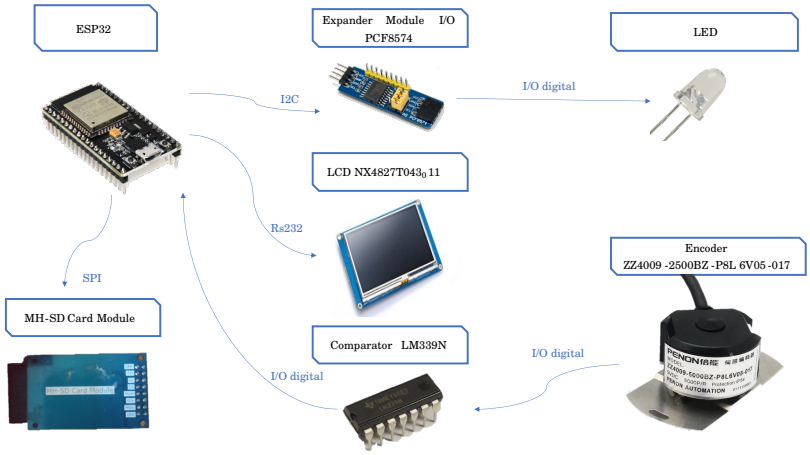


Figure 2: Diagram of connections between components

The more documented wiring diagram with all the details of each pin is presented in Appendix A Figure 6.

4. Communications

Communication is essential in this project, allowing two devices to communicate with each other with full-duplex or half-duplex communication. The ESP32 can have three types of communication, RS232, SPI, and I2C. In this case, RS232 communicates with the LCD, SPI interacts with the SD card, and I2C allows communication with the I/O expander.

4.1 RS232

RS232 is the most widely used interface between the computer and communications industries. It is defined as single-ended standards with many characteristics, such as a more considerable communication distance in low-speed serial communications, a moderate

price, and the good practicality of the system, so it is the most appropriate communication port [2]. For RS232 to work with computers, it needs to convert from a standard serial interface, and the computer to the TTL (Transistor-transistor logic) level [2].

LCD message consists of first sending where to write, following the value to write, and at the end, sending three bytes of 0xff. To make the code more concise and straightforward, a variable was created with the three last bytes, as seen in the following line of code.

```
String endChar = String(char(0xff)) + String(char(0xff)) + String(char(0
    xff));
```

To initialize the RS232 transmission, its baud rate needs to be declared in the Arduino initialization, as shown in the next lines of code.

```
Serial2.begin(9600);
while (!Serial2) {
    ; // wait for the serial port to connect.
}
```

After its initialization, it is possible to send information using serial write, as seen in the next lines of code.

```
Serial2.print("n0.val=" + String(encoderPos) + endChar);
Serial2.print("n1.val=" + String(FullRotations) + endChar);
Serial2.print("add 4,0," + String(ValueA*30) + endChar);
Serial2.print("add 4,1," + String((ValueB + 10)*30) + endChar);
Serial2.print("add 5,0," + String(Value_A*30) + endChar);
Serial2.print("add 5,1," + String((Value_B + 10)*30) + endChar);
Serial2.print("add 9,0," + String(ValueZ*30) + endChar);
Serial2.print("add 9,1," + String((Value_Z + 10)*30) + endChar);
```

4.2 I2C

I2C provides accessible communication without data loss, providing excellent speeds compared to other protocols [3]. It uses two wires to communicate, making it lightweight, economical, and omnipresent. Allows for communication with multiple devices using the same two wires by sending the address that wants to communicate with. Communication works by having an enslaver and multiple slaves, where the master sends a START bit to the bus, acting as a signal to all connected ICs that something is about to be transmitted. Afterwards, the message is followed by a STOP bit, which is the signal to all devices on the bus that it is available again [3]. I2C allows sending information to all of the slaves at the same time.

Expander I/0 needs a start bit followed by the address and a bit that dictates whether it will write or read. Afterwards, it sends the data to the slave, retrieving an acknowledgement followed by a stop bit, as seen in Figure 3.

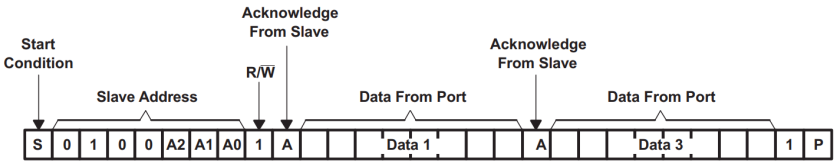


Figure 3: I2C communication message

To simplify the generation of the message seen in Figure 3 Wire library is used in the Arduino IDE. Firstly, the Wire library is initialized using the following line of code.

```
Wire.begin();
```

To generate the content of the message, firstly, it sends the start bit and the 7 bits of the address (0x20) with the method beginTransmission(). The write() function sends the last bit of the address and the actual data. In the end, the transmission terminates with the function endTransmission().

```
Wire.beginTransmission(Adress);  // Adress is 0x20
Wire.write(data);
Wire.endTransmission();
```

4.3 SPI

SPI is a protocol of four signal wires consisting of a clock signal (SCLK) sent from the bus master to all slaves. A slave select signal (SS) for each slave is used to select the slave to begin communication. Data are transmitted using two data lines, a Master-out-Slave In (MOSI) that transmits data to the slaves and a Master-In-Slave Out (MISO) that does the opposite. SPI is a single-master communication protocol. Meaning that one central device initiates all communications with the slaves. When the SPI master wishes to send data to a slave and request information from it, it selects a slave by pulling the corresponding SS line low and activates the clock signal at a clock frequency usable by the master and the slave. The master generates information on the MOSI line while it samples the MISO line [4].

SD card module requires SPI to communicate with the ESP32. Since this module is more challenging than previous modules, a library was utilized to aid in developing this project. To do so, it was first needed to import it using the following lines of code.

```
#include "FS.h"
#include "SD.h"
#include "SPI.h"
```

A helper function was devised to append information to the SD card.

```
bool appendFile(fs::FS &fs, const char * path, const char * message){
  //Serial.printf("Appending to file: %s\n", path);

  File file = fs.open(path, FILE_APPEND);
  if(!file){
```

```
    Serial.println("Failed to open file for appending");
    return false;
  }
  if(file.print(message)){
    delay(20);
      //Serial.println("Message appended");
  } else {
    Serial.println("Append failed");
    return false;
  }
  file.close();
  return true;
}
```

In order to send information to the SD card, the previous function is used as follows.

```
appendFile(SD, "/encoder.txt", " ;Channel A: ");
appendFile(SD, "/encoder.txt", itoa(ValueA,buffer,10));
appendFile(SD, "/encoder.txt", " ;Channel B: ");
appendFile(SD, "/encoder.txt", itoa(ValueB,buffer,10));
appendFile(SD, "/encoder.txt", " ;Channel _A: ");
appendFile(SD, "/encoder.txt", itoa(Value_A,buffer,10));
appendFile(SD, "/encoder.txt", " ;Channel _B: ");
appendFile(SD, "/encoder.txt", itoa(Value_B,buffer,10));
appendFile(SD, "/encoder.txt", " ;Channel Z: ");
appendFile(SD, "/encoder.txt", itoa(ValueZ,buffer,10));
appendFile(SD, "/encoder.txt", " ;Channel _Z: ");
appendFile(SD, "/encoder.txt", itoa(Value_Z,buffer,10));
appendFile(SD, "/encoder.txt", "\n");
```

Producing the information seen in Figure 4. The information store will be inside the file encoder.txt containing information of all channels encapsulated by semicolons to easily separate the information afterwards.

;Channel A: 0 ;Channel B: 0 ;Channel $_A : 1; Channel_B : 1; ChannelZ : 0; Channel_Z : 1$
;Channel A: 0 ;Channel B: 0 ;Channel $_A : 1; Channel_B : 1; ChannelZ : 0; Channel_Z : 1$
;Channel A: 0 ;Channel B: 0 ;Channel $_A : 1; Channel_B : 1; ChannelZ : 0; Channel_Z : 1$
;Channel A: 0 ;Channel B: 0 ;Channel $_A : 1; Channel_B : 1; ChannelZ : 0; Channel_Z : 1$
;Channel A: 0 ;Channel B: 0 ;Channel $_A : 1; Channel_B : 1; ChannelZ : 0; Channel_Z : 1$
;Channel A: 0 ;Channel B: 1 ;Channel $_A : 1; Channel_B : 0; ChannelZ : 0; Channel_Z : 1$
;Channel A: 1 ;Channel B: 1 ;Channel $_A : 0; Channel_B : 0; ChannelZ : 0; Channel_Z : 1$
;Channel A: 0 ;Channel B: 1 ;Channel $_A : 1; Channel_B : 0; ChannelZ : 0; Channel_Z : 1$
;Channel A: 1 ;Channel B: 1 ;Channel $_A : 0; Channel_B : 0; ChannelZ : 0; Channel_Z : 1$
;Channel A: 1 ;Channel B: 1 ;Channel $_A : 0; Channel_B : 0; ChannelZ : 0; Channel_Z : 1$
;Channel A: 1 ;Channel B: 1 ;Channel $_A : 0; Channel_B : 0; ChannelZ : 0; Channel_Z : 1$
;Channel A: 0 ;Channel B: 1 ;Channel $_A : 1; Channel_B : 0; ChannelZ : 0; Channel_Z : 1$
;Channel A: 1 ;Channel B: 1 ;Channel $_A : 0; Channel_B : 0; ChannelZ : 0; Channel_Z : 1$
;Channel A: 0 ;Channel B: 1 ;Channel $_A : 1; Channel_B : 0; ChannelZ : 0; Channel_Z : 1$
;Channel A: 1 ;Channel B: 1 ;Channel $_A : 0; Channel_B : 0; ChannelZ : 0; Channel_Z : 1$
;Channel A: 0 ;Channel B: 1 ;Channel $_A : 1; Channel_B : 0; ChannelZ : 0; Channel_Z : 1$
;Channel A: 0 ;Channel B: 0 ;Channel $_A : 1; Channel_B : 1; ChannelZ : 0; Channel_Z : 1$
;Channel A: 1 ;Channel B: 1 ;Channel $_A : 0; Channel_B : 0; ChannelZ : 0; Channel_Z : 1$
;Channel A: 1 ;Channel B: 1 ;Channel $_A : 0; Channel_B : 0; ChannelZ : 0; Channel_Z : 1$
;Channel A: 0 ;Channel B: 0 ;Channel $_A : 1; Channel_B : 1; ChannelZ : 0; Channel_Z : 1$

Figure 4: Text stored in the SD card inside the encoder.txt file

5. Usage Description

Encoders are complex and brittle equipment that work in harsh conditions, leading to problems that are hard to detect without testing all channels of the encoder simultaneously. A tool was developed with six wire connectors as seen in Figure 5 where the encoder channels are plugged. Before using the tool, plug an SD card into its appropriate place, followed by connecting the encoder in question to the tool. After plugging the tool into the computer, it displays all the information to verify the proper working order of the encoder. It is also possible to connect an H-bridge motor control to the equipment using the I/O expander to test a motor moving when the encoder moves. In the end, it is also possible to read all the encoder inputs by opening the text file inside the SD card, allowing the plot of graphs with previously recorded data.

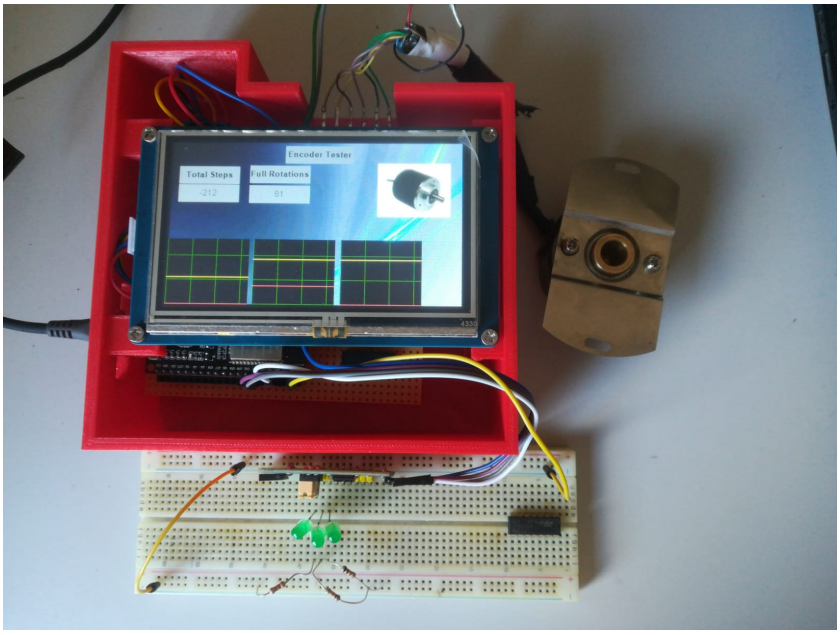


Figure 5: Project photo

6. Demonstration

A video detailing all the information and usage of the previously mentioned tool in Section 5. is available at https://youtu.be/D5HNI8b4qWk. It is also possible to consult all the project's code by following the GitHub page at https://github.com/andrefdre/Encoder-Tester.

References

[1] Expressif. *ESP32-DevKitC V4 Getting Started Guide*. Last accessed 09 January 2023. 2023. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html.

[2] X. Han and X. Kong. "The Designing of Serial Communication Based on RS232". In: *2010 First ACIS International Symposium on Cryptography, and Network Security, Data Mining and Knowledge Discovery, E-Commerce and Its Applications, and Embedded Systems*. 2010, pp. 382–384. DOI: 10.1109/CDEE.2010.80.

[3] J. Mankar et al. "Review of I2C protocol". In: *International Journal of Research in Advent Technology* 2.1 (2014).

[4] F. Leens. "An introduction to I2C and SPI protocols". In: *IEEE Instrumentation Measurement Magazine* 12.1 (2009), pp. 8–13. DOI: 10.1109/MIM.2009.4762946.

Appendix A

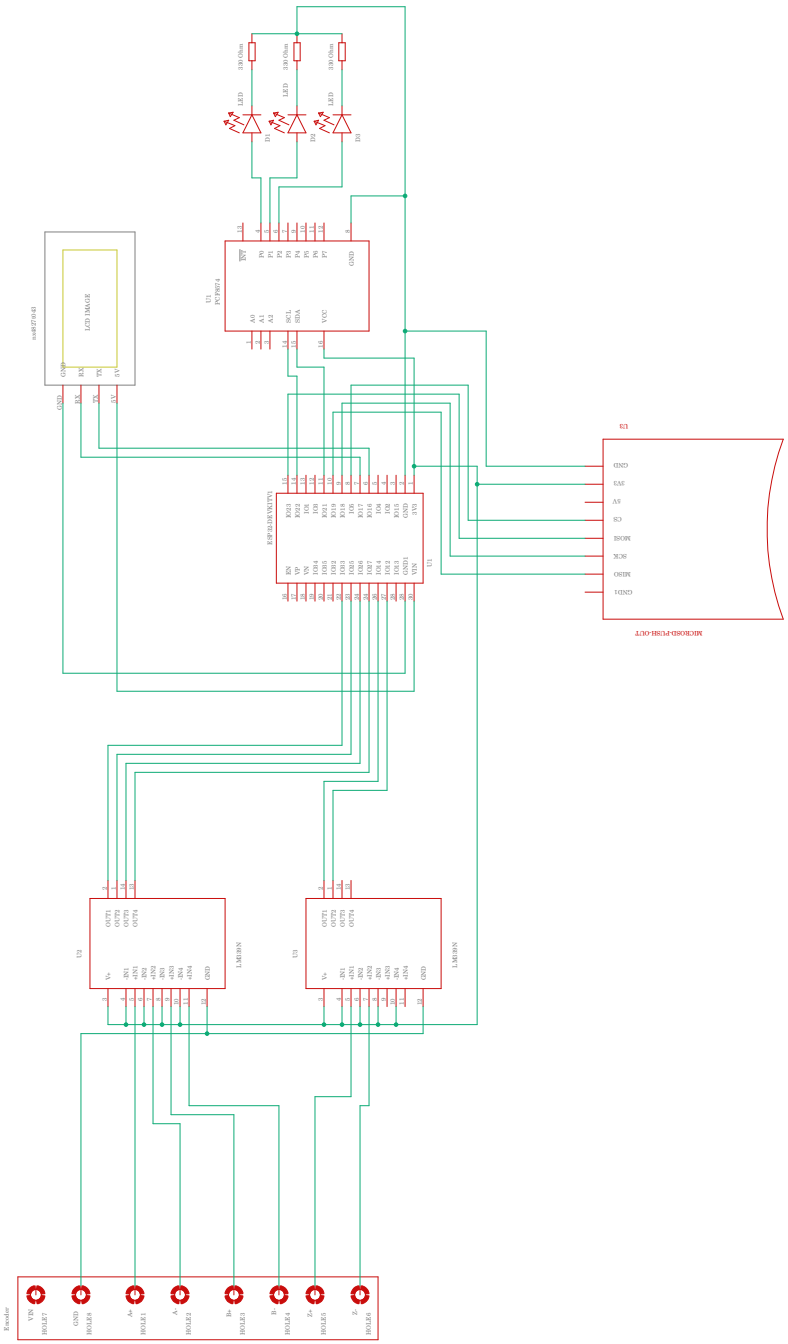


Figure 6: Electrical Diagram