# Reception and Transmission of Infrared Signals

# LCD Display

written by Bacher Jimmi and Ahmed Hayat

Content

[Reception and Transmission of Infrared Signals 1](#_Toc156918935)

[LCD Display 1](#_Toc156918936)

[Abstract: 3](#_Toc156918937)

[1. Introduction 4](#_Toc156918938)

[1.1 NEC protocol overview 4](#_Toc156918939)

[2. Implementation 5](#_Toc156918940)

[2.1 38 kHz Pulse Function 5](#_Toc156918941)

[2.2 Record Function 5](#_Toc156918942)

[2.3 Signal Timing and Storage 5](#_Toc156918943)

[2.4 Playback Function 6](#_Toc156918944)

[2.5 Error Handling 7](#_Toc156918945)

[3. LCD DISPLAY INTRODUCTION 8](#_Toc156918946)

[4. LCD DISPLAY IMPLEMENTATION 8](#_Toc156918947)

[4.1 Assigning the Buttons 8](#_Toc156918948)

[4.2 Button Function 8](#_Toc156918949)

[4.3 UI\_get\_selection Function 9](#_Toc156918950)

[4.4 Alphabet function 10](#_Toc156918951)

[5. table of contents for the buttons 11](#_Toc156918952)

[6. Conclusion 12](#_Toc156918953)

## Abstract:

[Replay](https://drive.google.com/file/d/1eCGgxZm4FBbogJcM-cKBm01AxfpTdt77/view?usp=sharing) (Video, google drive, 29.06.2023)

[Timeout](https://drive.google.com/file/d/1eCGgxZm4FBbogJcM-cKBm01AxfpTdt77/view?usp=sharing) (Video, google drive, 29.06.2023)

In this project, an advanced code has been developed to capture infrared signals utilizing the highly sophisticated TSOP series of integrated receivers from Vishay and the ATmega328p microcontroller. This code facilitates the accurate recording and storage of these signals in an intricate data structure, . Additionally, the code is designed to enable the playback of these recorded signals through the use of an infrared LED. The implementation of this project required the meticulous development of functions dedicated to both the capture and transmission of IR signals, and the resolution of numerous complex challenges encountered throughout the development phase.

## Introduction

The objective of this project was to implement a program for signal reception and transmission of Infrared Signals using an NEC transmission protocol controller and a microcontroller. The NEC protocol is a widely used standard for infrared communication, particularly in remote controls. The project aims to capture infrared signals and decode them according to the NEC protocol (although the finished code could, through minor changes, adapt to any transmission protocol).

All of this will be controllable by the user via a LCD, which allows for quick and easy navigation through various captured commands. These commands include the choice of three menu options and the choosing of one’s own name for a specific recorded signal.

### NEC protocol overview

The NEC protocol is based on pulse-distance coding and operates at a carrier frequency of 38 kHz.

The communication starts with a leading pulse burst of approximately 9ms, followed by a space of approximately 4.5ms. The following bits are encoded as pulse bursts and spaces, where the pulse duration represents a logical 1 and the space duration represents a logical 0. A bit is encoded using a 560us pulse burst followed by a 560us space for a logical 1 or a 560us pulse burst followed by a 1.69ms space for a logical 0.

## Implementation

### 38 kHz Pulse Function

Since the NEC transmission protocol operates at a frequency of 38kHz, we would need to be able to send a signal at that frequency.

After careful consideration and consultation of the ATmega328p microcontroller datasheet, the integrated “timer0”-interrupt was chosen and utilized to achieve said frequency.

An interrupt in the context of microcontrollers is an integrated mechanism by which the normal flow of a program is interrupted in response to an external or internal event, which allows the microcontroller to address the event and handle time-sensitive tasks. When an interrupt occurs, the microcontroller pauses the current ongoing operations, save the state of its work and then executes a special function known as the interrupt service routine (short: “ISR”). Those events include input from various sensors, timers, communication events or other peripheral activities.

In this specific case, the microcontroller is supposed to send a signal at 38kHz which can be achieved by the chosen interrupt. The timer0 in the ATmega328p is an 8-bit timer, meaning it can count up to 2^8-1 (=255). The rate at which this timer counts is determined by the system clock frequency and the “prescaler” setting. The prescaler is used to divide the clock frequency to a slower rate, making the timer more versatile for a range of timing needs.

For this instance, the ATmega328p is running at a clock speed of 16MHz (given in datasheet). By selecting a prescaler value of 8, resulting in a tick every 0.5 us, the desired interval was calculated. With 26 ticks, an interval of 13 us was achieved, allowing for toggling the logic level and generating a 38kHz pulsating signal.

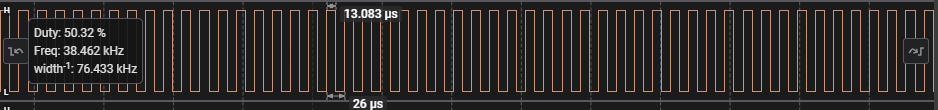
Initially, there were logic errors when calculating the reload value, but these were resolved rather quickly.

Figure 1: 38 kHz pulse signal, sent on PD7 using the timer0 interrupt

### Record Function

The record function presented challenges during its initial implementation. The “measure()” function was attempted, utilizing the ICF1 flag. However, I have encountered issues. Upon reviewing the provided guidelines and consulting the Timer1 interrupt pages in the datasheet, the Input Capture Pin was discovered.

Initiating Timer1 with prescaler settings (CS11 and CS10 = 64) was successful.

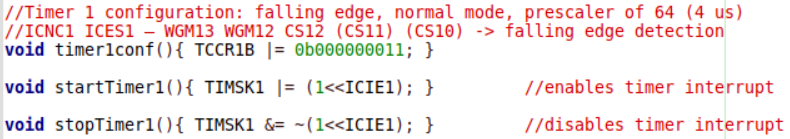
The startTimer1 and stopTimer1 functions were created to enable/disable the Timer1 Input Capture Interrupt.

Figure : Initialization of Timer1

### Signal Timing and Storage

The timings of an IR controller were detected and stored by saving the ICR1 register and using the microcontroller lecturer’s guidance.

The logic changes were captured and the resulting times between logic changes were stored in an uint16\_t array called “ir\_timings”.

Initially, there were challenges with the “int2str” function, as the output format for the 16-bit values was not clear. After discussing the issues with a colleague, it was realized that the ASCII values needed to be converted from hexadecimal and the maximum size of the input value had to be considered.

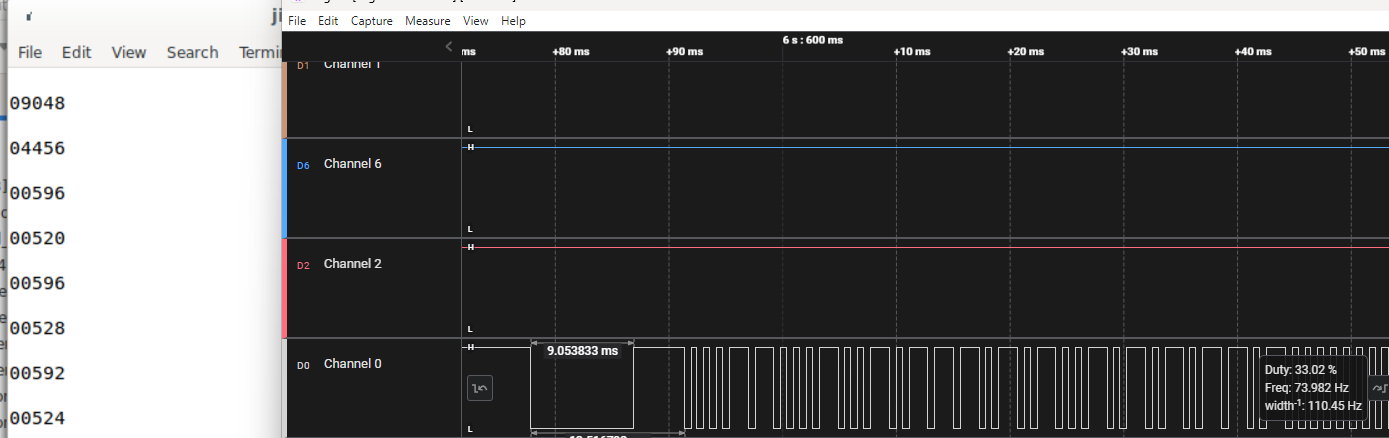
Additionally, the captured values needed to be multiplied by 4 to account for the 4 us tick.

Figure 3: Captured values displayed via Minicom (on the left); Sent signal displayed via Logic Analyzer (on the right);

### Playback Function

To play back the recorded signals, a separate function named “pulse()” was created. This function contained a while loop that checked for the end of the “ir\_timings” array.

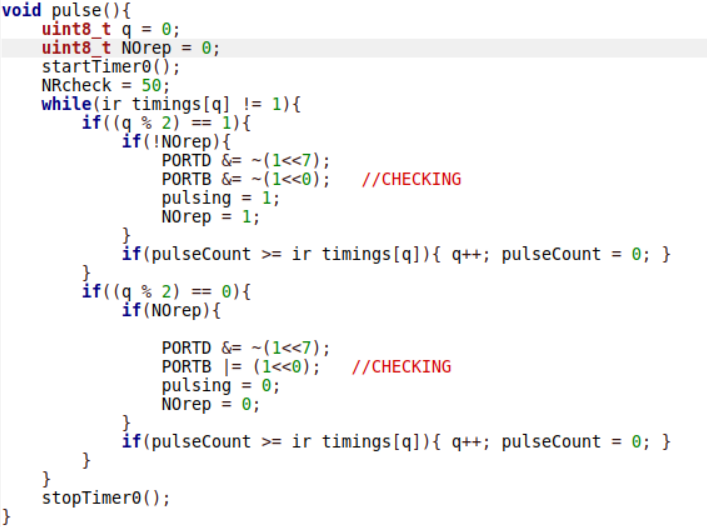
Inside the while loop, two if statements were crucial. One checked if the variable “q”, which was just the name of a simple counter, was positive or negative using modulus 2. If “q” was positive, the timer0 interrupt was triggered to generate pulses, incrementing the volatile integer “pulsecount” by 13 until it exceeded the current time value. If “q” was negative, the volatile integer “pauseCount” was incremented until the time value was surpassed. The process repeated by resetting the respective variables to zero.

Figure 4: pulse function that turns on and off the 38 kHz according to the value that was saved by previous functions

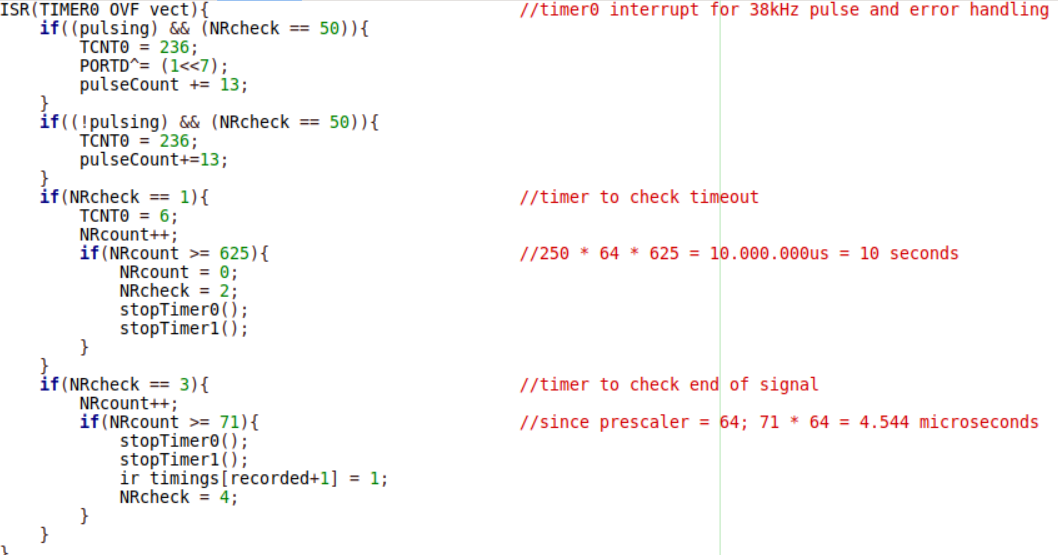


Figure 5: 38 kHz pulse that gets toggled according to the pulse function

### Error Handling

To incorporate error handling, a global volatile variable named “NRcheck” was introduced. NRcheck controlled the program’s flow, whether it be the timer1 interrupt, pulsation or other functionalities.

Setting NRcheck to 1 when calling the record function activated the timer0 and timer1 interrupts. When NRcheck was 1, ther timer0 interrupt incremented a volatile integer “NRcount” every 16ms with a prescaler of 1024. A function “timer0conf(uint8\_t mode)” was written to switch between two modes: the standard pulsation mode and the error handling mode. If NRcount reached a value over 625, indicating that 10 seconds had passed, an error code was returned.

During this 10-second interval, signals could still be detected. If a falling edge was detected, the timeout check was stopped by setting NRcheck to 3. When NRcheck was set to 2, indicating a timeout, both timers were stopped.

Additionally, if the recorded signal was larger in size than the given maximum infrared signal of 250, NRcheck was set to 5, indicating a bigger than usual IR signal.

In the record function after the waiting while loop, NRcheck was checked and a timeout code was returned if NRcheck was 2. If NRcheck was 3, indicating the end of the signal, NRcheck was set 4 and the recorded signals were outputted via the Minicom.

Ein Bild, das Text, Schrift enthält.

Automatisch generierte Beschreibung

Figure 6: Description of which value in NRcheck indicates

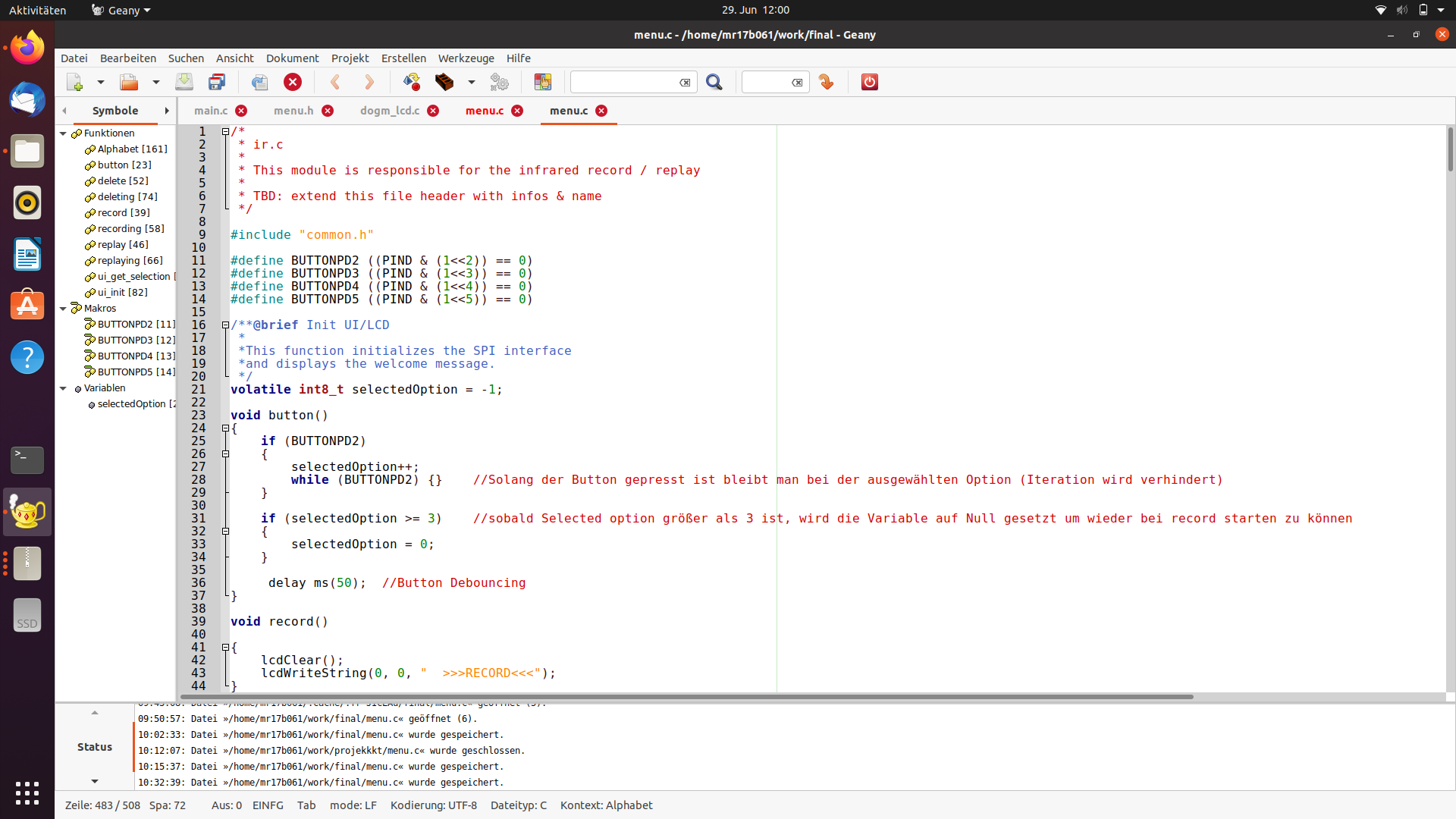
## LCD Display introduction

To be able to use the universal remote control, a menu is needed. This menu should be played on an LCD display and operated with the integrated buttons. The menu provides the opportunity to switch between different statements and to select them. Apart from this, names can also be selected independently by the user in the event of a “recording”.

## LCD DISPLAY IMPLEMENTATION

### Assigning the Buttons

The first step was to activate the buttons and define them. Button S1 in the top left corner of the LCD display is PD2 on the Arduino board, button S2 is right below it, PD3 on the board. Button S4 in the bottom corner, PD5 on the board and the last one right next to it, button S3, PD4 on the board.



### Button Function

To be able to switch between the individual statements in the menu, there is the variable selected option, which starts at minus 1, as soon as the button is pressed, the next statement is displayed. When you have reached the last statement, the variable is set to zero and you get to the first statement.

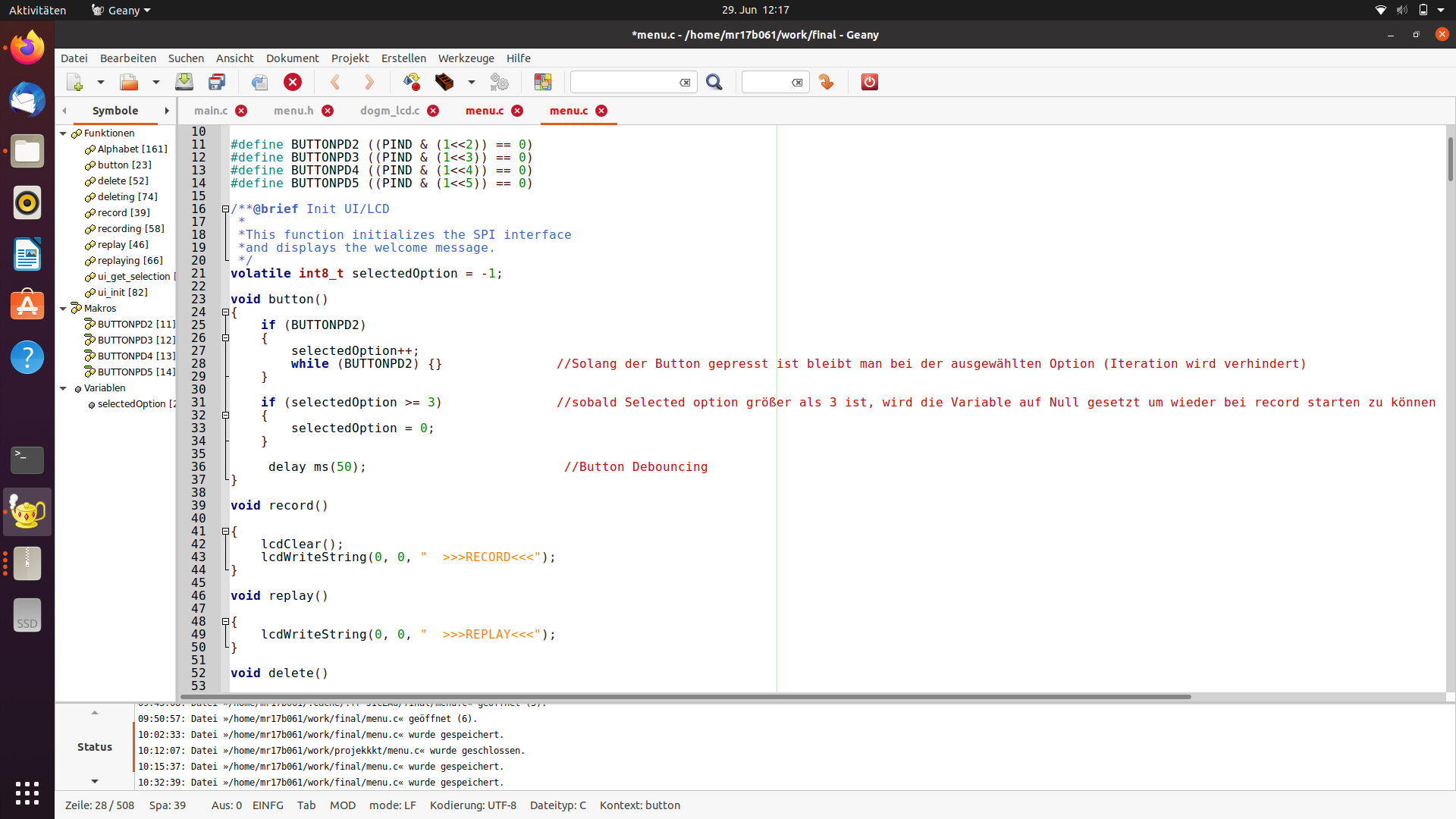


Figure : Button function

### UI\_get\_selection Function

With this function, the statements can then also be selected. As soon as the confirm button is pressed, the variable var is changed depending on what was selected. This variable is related to the switch cases in the main function.

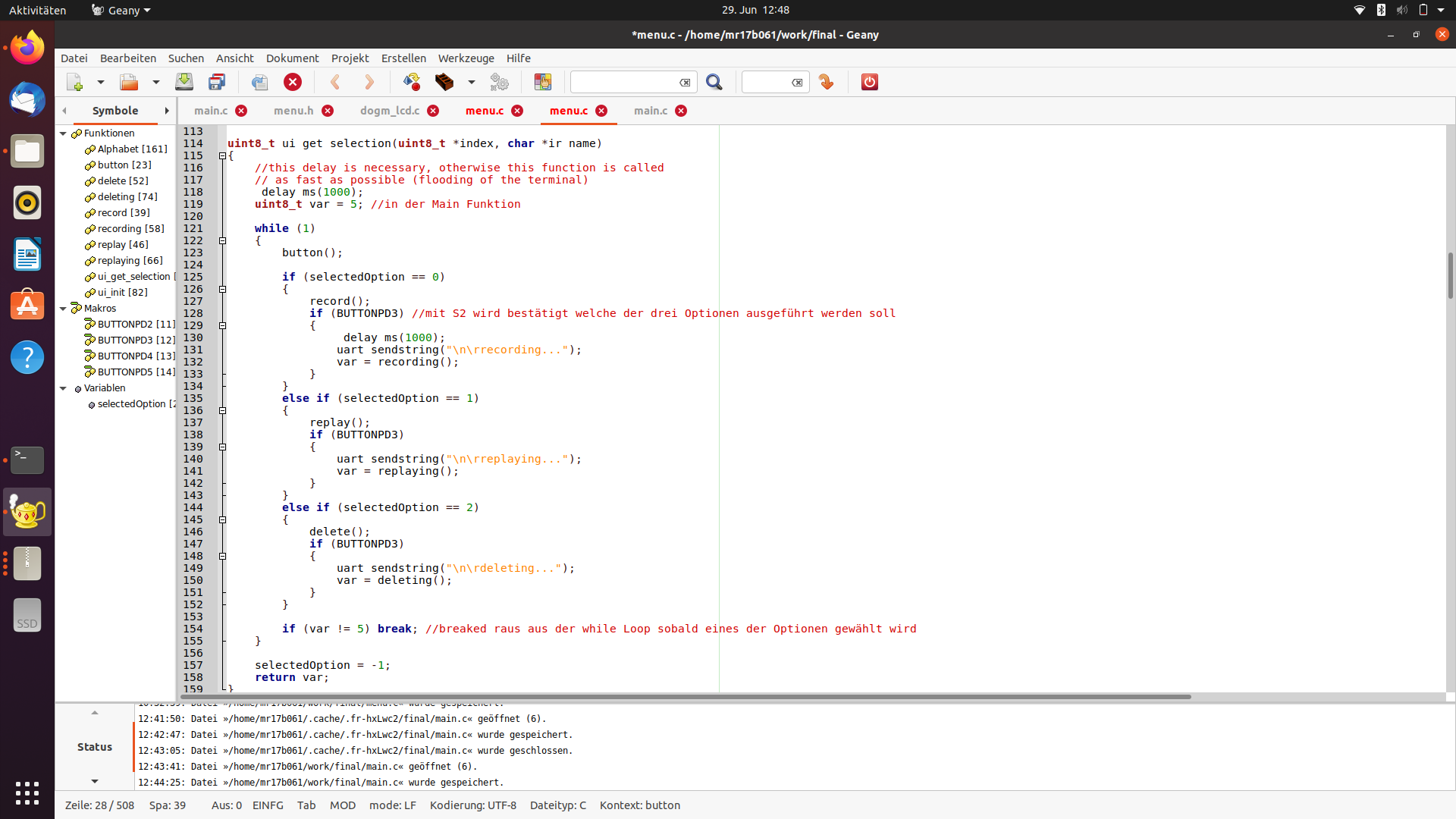


Figure : ui\_get\_selection function

### Alphabet function

This function appears as soon as you want to record. It has 3 tasks, it outputs the alphabet or the numbers, including a cursor. It stores the selected name in an array and then the stored name is output after a confirmation.

#### Cursor

The cursor is there to know where you are and to be able to move left and right as well as to the next "page".

|  |  |
| --- | --- |
| Figure : Cursor right | Ein Bild, das Text, Screenshot, Software, Computer enthält.  Automatisch generierte Beschreibung  Figure : Cursor left |

#### stored name

As each letter in the array has been stored by pressing a button, it can be confirmed at the end to output the whole name again.

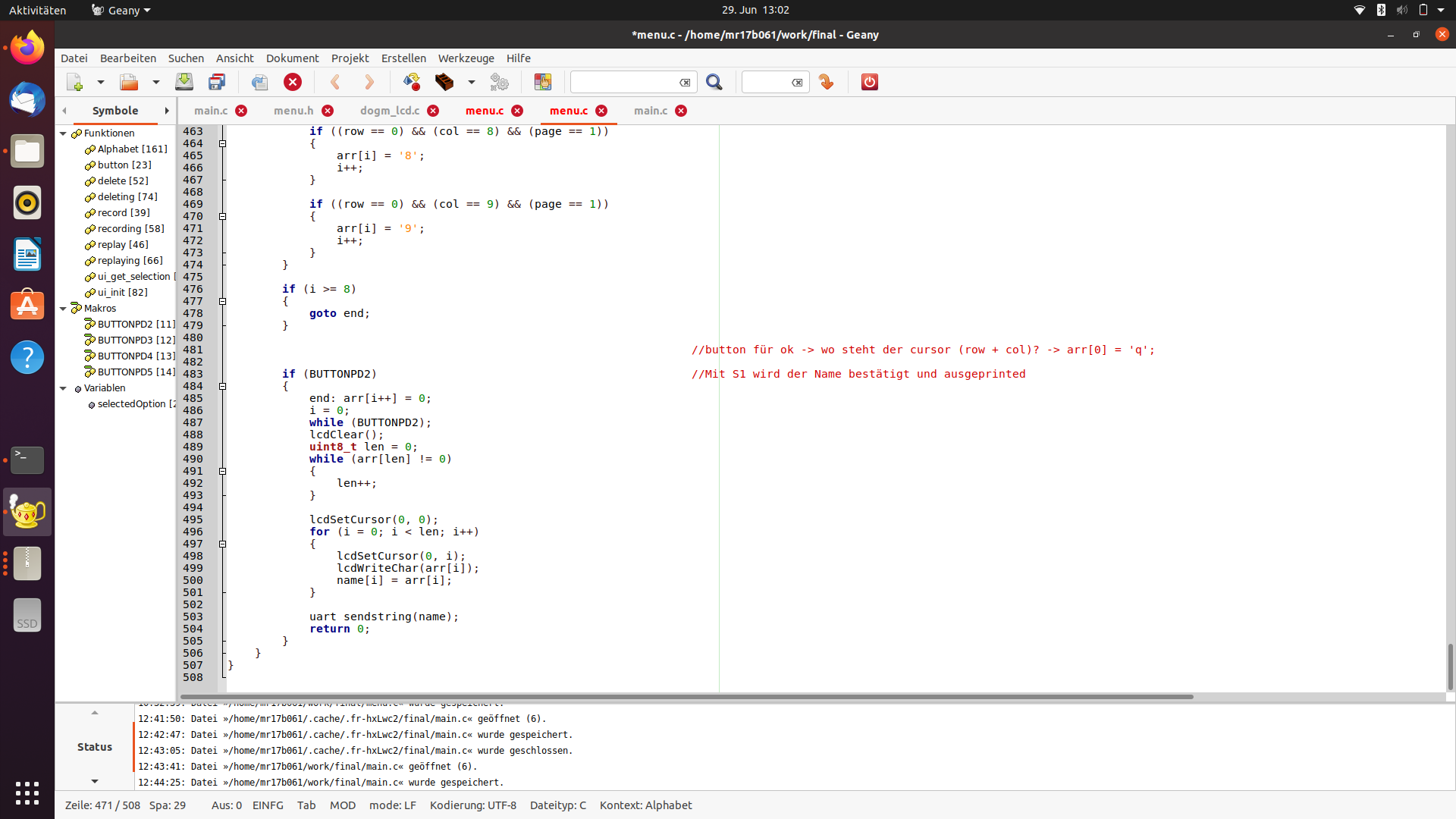


Figure : Stored name

## table of contents for the buttons

To switch between “replay”, “record” and “delete”: S1

To select “replay”, “record” or “delete”: S1

To go the right in the alphabet or when selecting a number: S3

To go the left in the alphabet or when selecting a number: S4

To store a letter or a number in the array: S2

To print the whole name: S1

To get back to the “replay”, “record” and “delete” options: S1

## Conclusion

In conclusion, the project demonstrated the successful implementation of the signal reception and transmission of any infrared transmission protocol using the ATmega328P microcontroller.

The developed system allowed for the capture and reproduction of IR signals.

Ein Bild, das Elektronik, Text, Computer, Schaltung enthält.

Automatisch generierte BeschreibungThrough careful handling of register, interrupts and timing calculations, the project achieved the desired functionality. Further improvements and optimizations can be made to enhance the preservation of stored signals, readability of the code and error handling capabilities.

Figure 7: Result; replay of a NEC transmission protocol control