Cocket Nova Development Board Programming Guide C/C++

Release 0.1.0

Cesar Bautista

Jun 24, 2024

CONTENTS

Note: This project is under active development.

The Coket Nova guide is an instructional manual for utilizing the compiler SDCC. It serves various purposes, allowing users to explore different features and obtain new projects and configurations. The Cocket Nova CH552 boards are incredibly easy to use. The projects focus on innovation and present various alternatives for usage.

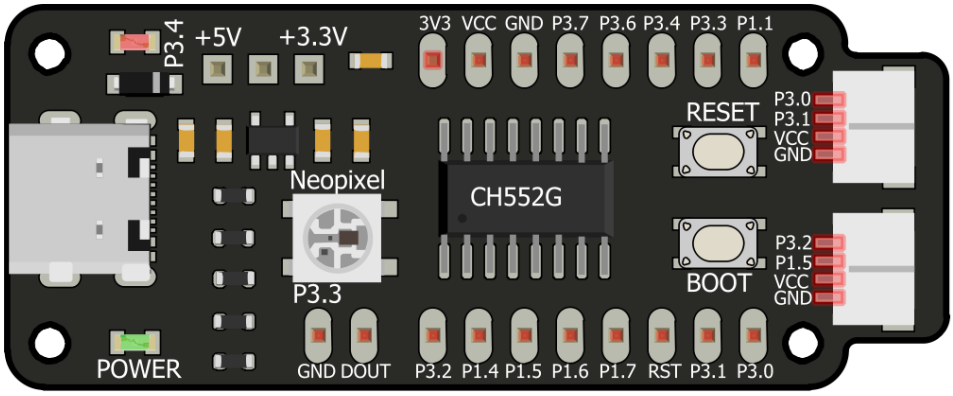

Fig. 1: Cocket Nova CH552 Desing graphic

Tip: The codes examples in github are available in the following link: CH55x_SDCC_Examples Github

CHAPTER

ONE

COCKET NOVA DEVELOPMET BOARD CH552 GUIDE

This is an excellent guide for beginner programmers, focused on using the SDCC compiler in both Windows and Linux environments. Here, you can find excellent references and examples along with comprehensive documentation focusing on developing technology for embedded systems. This course covers everything from installation and setting up the compiler to managing project dependencies and developing code. It's a valuable resource that will guide you through the development process using high-quality technology, ensuring long-lasting and robust projects.

1.1 Why Use the SDCC Compiler?

The Small Device C Compiler (SDCC) is a highly regarded tool in the field of embedded systems development. Here are several reasons why you might choose to use the SDCC compiler for your projects:

1. **Free and Open-Source**: SDCC is freely available and open-source, which means you can use it without licensing fees and contribute to its development if you wish.
2. **Wide Microcontroller Support**: It supports a broad range of microcontrollers, including popular ones like the CH552, making it a versatile choice for various projects.
3. **Ease of Use**: SDCC is known for its user-friendly interface and straightforward setup, which helps developers get started quickly.
4. **Active Community and Documentation**: With an active community and extensive documentation, you can find support and resources to help you solve any issues you encounter.
5. **Compatibility**: SDCC is compatible with many other tools and environments, allowing for seamless integration into existing workflows.

1.2 Understanding Programming Languages in Embedded Systems

To develop effective embedded systems, it's crucial to understand the different types of programming languages used:

1.2.1 Machine Language

Machine language, also known as machine code, is the most fundamental level of programming. Instructions are written in binary bit patterns, which are combinations of 1s and 0s. These patterns correspond to HIGH and LOW voltage levels that the microcontroller or microprocessor can directly interpret. This language is the most difficult for humans to use due to its complexity and lack of readability.

1.2.2 Assembly Language

Assembly language is a step above machine language, providing a more human-readable format. It uses mnemonics and hexadecimal codes to represent machine language instructions. For instance, the 8051 microcontroller assembly language includes a combination of English-like words called mnemonics and hexadecimal numbers. Despite being more readable than machine language, it still requires an in-depth understanding of the microcontroller's architecture.

1.2.3 High-level Language

High-level languages simplify programming by abstracting away the intricate details of the microcontroller's architecture. These languages use familiar words and statements, making them easier to learn and use. Examples of high-level languages include BASIC, C, Pascal, C++, and Java. Programs written in high-level languages are translated into machine code by a compiler, bridging the gap between human-friendly code and machine-understandable instructions.

By understanding these different levels of programming languages , you can choose the most appropriate one for your project, balancing ease of use with the level of control you need over the hardware.

1.3 Requirements

- Python (Package Installation and Environments)
- Utilization of Operating System Controllers
- Understanding of Basic Electronics

Tip: This guide is tailored for individuals with a basic understanding of programming and electronics. It's ideal for those interested in diving into embedded systems and programming microcontrollers.

1.4 PinOut

The CH552 microcontroller development board has a total of 16 pins, each with a specific function. The following is a list of the pins and their respective functions:

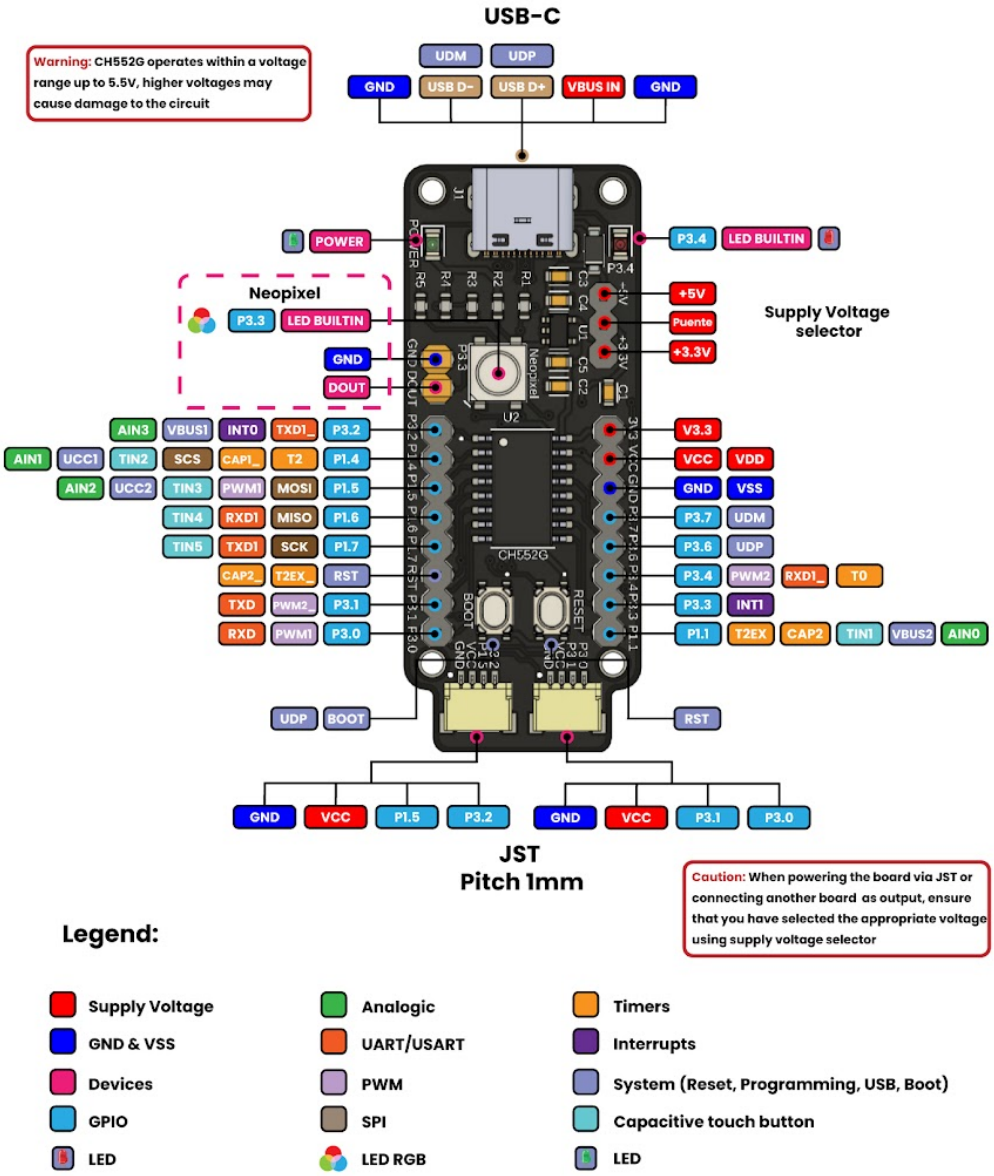

Fig. 1.1: Cocket Nova CH552 PinOut

CHAPTER

TWO

ENVIRONMENT SETUP ON UBUNTU

Caution: This project is under active development. The information provided here is subject to change.

Update the operating system:

```
sudo apt update
```

Install *make*, *binutils* and *sdcc*:

```
sudo apt install make
sudo apt install binutils
sudo apt install sdcc
```

Clone the examples with the main code:

```
git clone https://github.com/UNIT-Electronics/CH55x_SDCC_Examples.git
```

Navigate to the path:

```
cd CH55x_SDCC_Examples/Software/examples/Blink
```

Execute the command:

```
make help
```

You will see:

```
Use the following commands:
make all     compile, build, and keep all files
make hex     compile and build blink.hex
make bin     compile and build blink.bin
make clean   remove all build files
```

Connect a device with the BOOT button pressed:

```
lsusb
```

The device will be shown with this description:

```
pc@LAPTOP:~$ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
```

(continues on next page)

(continued from previous page)

```
Bus 001 Device 002: ID 4348:55e0 WinChipHead
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

CHAPTER

THREE

ENVIRONMENT SETUP ON WINDOWS

This section provides a step-by-step guide to setting up the SDCC compiler on Windows operating systems. It also includes instructions for installing the necessary tools and configuring the environment variables. Additionally, it covers the installation of the pyusb library and updating the driver using Zadig.

3.1 Compiler Installation

Follow the steps below to install the necessary tools:

- **Installing Git for Windows** Download and install Git for Windows from the official Git website.
- **Installing SDCC** Download and install the latest version of SDCC. You can find the latest version on the SDCC downloads page.
- **Installing MinGW** Install MinGW, which is a set of tools for software development on Windows. You can download the installer from the official MinGW website.
- **Installing Zadig** Download the latest version of Zadig. You can download it from the official website.

Tip: It is recommended to install the tools in the order listed above.

Caution: Remember to restart your computer after installing the tools.

3.2 Environment Variable Configuration

Remember that for Windows operating systems, an extra step is necessary, which is to open the environment variable -> Edit environment variable:

```
C:\MinGW\bin
```

3.3 Locate the file

After installing MinGW, you will need to locate the *mingw32-make.exe* file. This file is typically found in the *C:MinGWbin* directory. Once located, rename the file to *make.exe*.

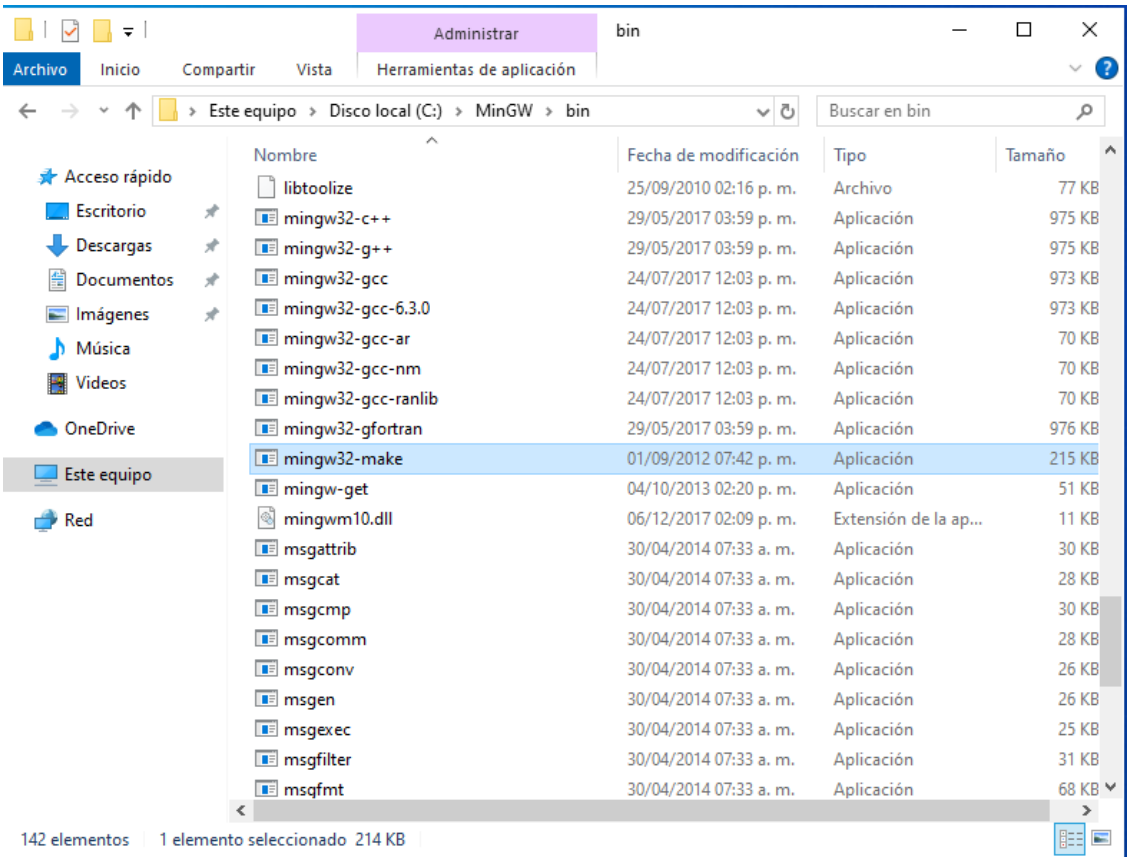

Fig. 3.1: Locating the *mingw32-make.exe* file

3.4 Rename it

After locating *mingw32-make.exe*, rename it to *make.exe*. This change is necessary for compatibility with many build scripts that expect the command to be named *make*.

> **Warning:** If you encounter any issues, create a copy of the file and then rename the copy to *make.exe*.

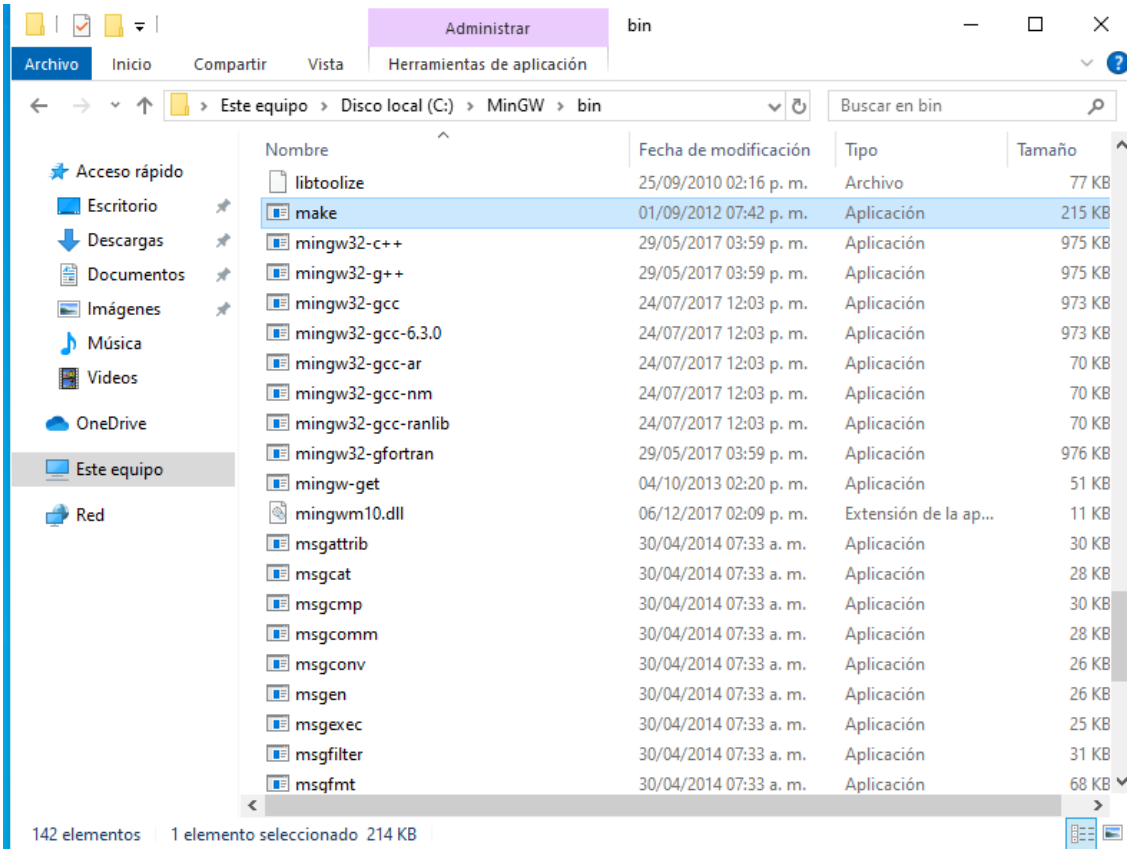

Fig. 3.2: Renaming *mingw32-make.exe* to *make.exe*

3.5 Add the path to the environment variable

Next, you need to add the path to the MinGW bin directory to your system's environment variables. This allows the *make* command to be recognized from any command prompt.

1. Open the Start Search, type in "env", and select "Edit the system environment variables".
2. In the System Properties window, click on the "Environment Variables" button.
3. In the Environment Variables window, under "System variables", select the "Path" variable and click "Edit".
4. In the Edit Environment Variable window, click "New" and add the path *C:MinGWbin*.

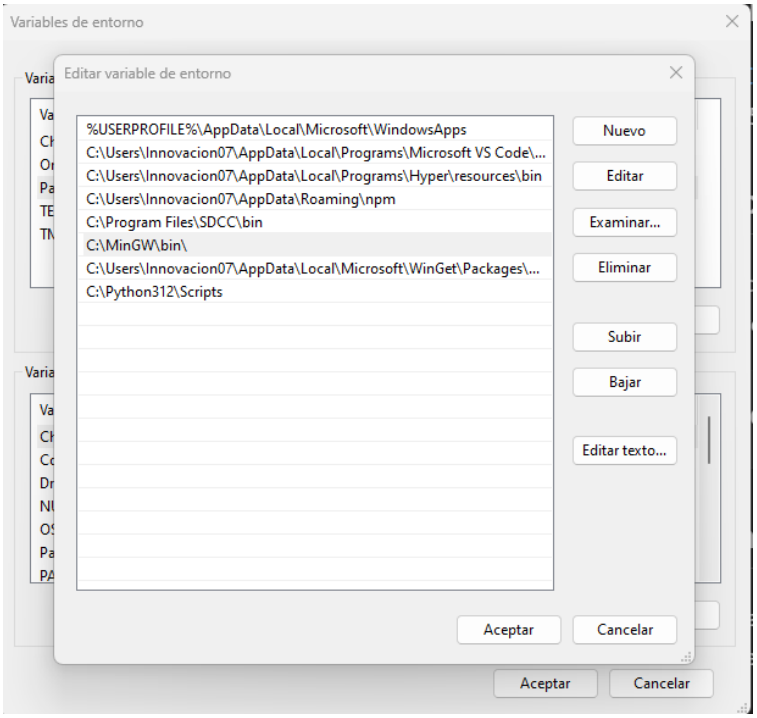

Fig. 3.3: Adding MinGW bin directory to environment variables

3.6 Verify the installation

To verify that the *make* command is correctly set up, open a new command prompt and type:

```
make --version
```

You should see the version information for *make*, indicating that it is correctly installed and recognized by the system.

```
$ make --version
GNU Make 3.82.90
Built for i686-pc-mingw32
Copyright (C) 1988-2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

Fig. 3.4: Verifying the installation of *make*

3.7 Update driver

The current loading tool can utilize the default driver and coexist with the official WCHISPTool. In case the driver encounters issues, it is advisable to switch the driver version to libusb-win32 using Zadig.

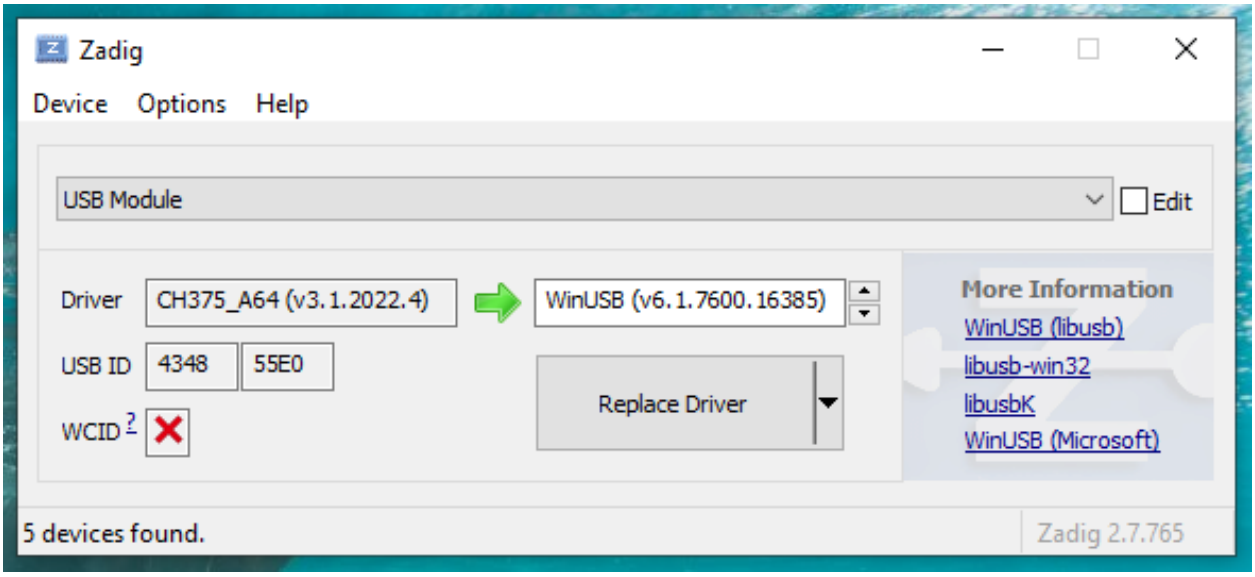

Fig. 3.5: driver

Warning: The use of Zadig is at your own risk. if you are not familiar with the tool, it is recommended to seek assistance from someone who is. In the case of changing the driver any device , it is important to have the original driver available to revert the changes.

CHAPTER

FOUR

COMPILE AND FLASH CH55X WITH SDCC

4.1 Running a Program

To run the program, you will need to use a bash terminal. Follow these steps to clone the examples repository, navigate to the appropriate directory, and compile the project.

1. **Clone the Examples Repository**

   Begin by cloning the examples repository which contains the main code. Open your bash terminal and execute the following command:

   ```
   git clone https://github.com/UNIT-Electronics/CH55x_SDCC_Examples
   ```

2. **Navigate to the Example Path**

   Once the repository is cloned, navigate to the path where the example programs are located. Use the following command to change the directory:

   ```
   cd ~/CH55x_SDCC_Examples/Software/examples/Blink/
   ```

3. **Connect the Device**

   Connect your CH55x device to your computer. Ensure that you press and hold the BOOT button while connecting the device. This is essential for the device to enter programming mode.

4. **Compile the Project**

   To compile the project and generate the necessary files, execute the following command in your terminal:

   ```
   make all
   ```

   This command will compile the project, resulting in the generation of files with various extensions necessary for flashing the microcontroller.

4.2 Install pyusb

Verify the installation with *python –version*. If not installed, run:

```
sudo apt install python3-pip
```

Then verify the installation:

```
(.venv) server@admin:~/Documents/CH55x_SDCC_Examples/Software/examples/0. Blink$ ls
blink.bin  blink.c  Makefile  src  uploader
(.venv) server@admin:~/Documents/CH55x_SDCC_Examples/Software/examples/0. Blink$ make all
Compiling blink.c ...
Compiling src/delay.c ...
Building blink.ihx ...
Building blink.bin ...
Building blink.hex ...
packihx: read 11 lines, wrote 14: OK.
-----------------
FLASH: 130 bytes
IRAM:  0 bytes
XRAM:  256 bytes
-----------------
```

Fig. 4.1: Compilation output files

```
python3 -m pip show pyusb
```

4.3 Error with pip

If you encounter this error, we recommend installing the Python environment:

```
sudo apt install python3-venv
```

Create an environment:

```
python3 -m venv .venv
```

Activate the environment:

```
source .venv/bin/activate
```

And install *pyusb*:

```
pip install pyusb
```

4.4 Flashing the Program

Once the project is compiled, you need to flash the program onto the CH55x device. Follow these steps:

1. **Connect the Device**

   Ensure your CH55x device is connected and the BOOT button is pressed, as done during the compilation step.

2. **Flash the Program**

   Execute the following command to flash the compiled program onto the microcontroller:

   ```
   make flash
   ```

   If the flashing process is successful, the code will generate a blinking effect on the connected LED, indicating that the program is running correctly.

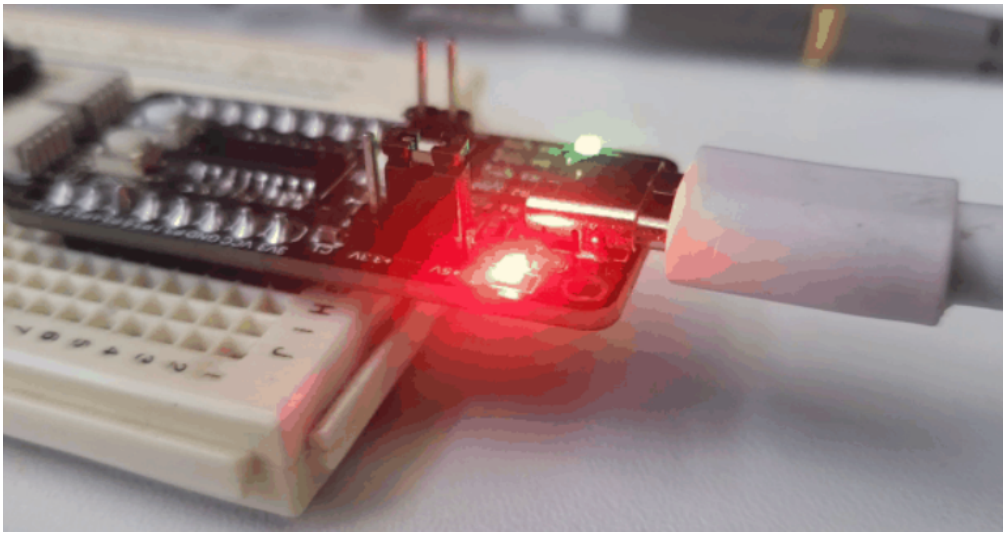

Fig. 4.2: LED blinking effect

CHAPTER

FIVE

SOFTWARE DEVELOPMENT PROTOTYPE

5.1 Loadupch

The Loadupch is a software development prototype designed to facilitate the uploading of code to the CH552 microcontroller. It is a user-friendly tool that provides a graphical interface, making it easier for users to upload their code. The Loadupch tool is compatible with both Windows and Linux operating systems.

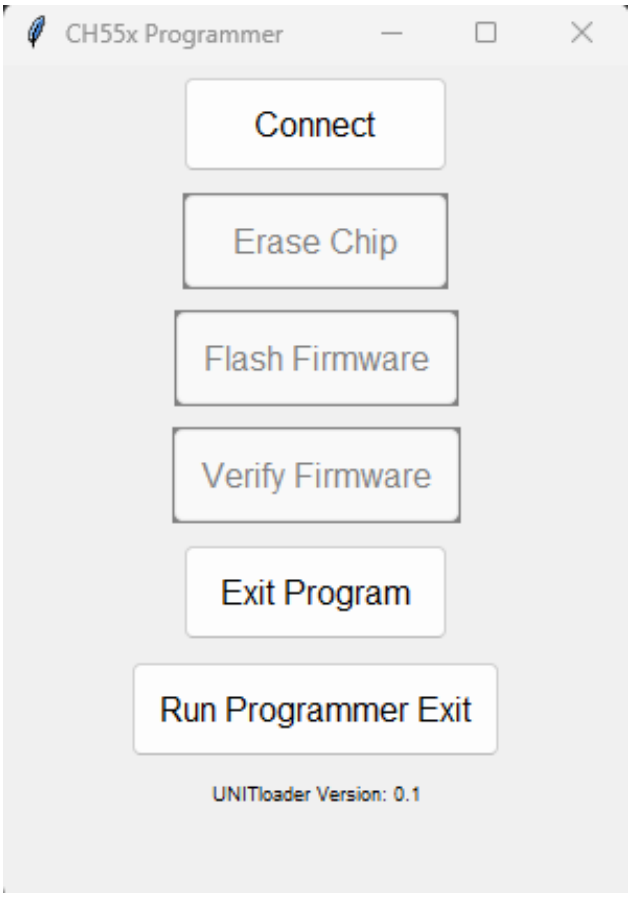

Fig. 5.1: Loadupch tool interface

5.1.1 Installing Loadupch

> **Warning:** The Loadupch tool is currently under development and may contain bugs. Use it at your own risk.

To install the Loadupch tool, you can use *pypi*. Follow these steps:

1. **Install Loadupch**

   Use the following command to install the Loadupch tool via pip:

   ```
   pip install loadupch
   ```

2. **Run Loadupch**

   After installation, you can run the Loadupch tool with the following command:

   ```
   python -m loadupch
   ```

   This will launch the graphical interface of the Loadupch tool, allowing you to upload code to your CH552 microcontroller easily.

Tip: If you need to uninstall the Loadupch tool for any reason, use the following command:

```
pip uninstall loadupch
```

CHAPTER

SIX

GENERAL BOARD CONTROL

The CH552, characterized by its compact size, native USB connectivity, and 16 KB memory (with 14 KB usable), enables the creation of simple yet effective programs. This allows for greater control in implementing various applications. The choice of this microcontroller is based on its affordability, ease of connection, and compatibility with various operating systems.

6.1 Recommended Operating Conditions

Table 6.1: Recommended Operating Conditions

Symbol	Description	Range
VUSB	Voltage supply via USB	3.14 to 5.255 V
VIn	Voltage supply from pins	2.7 to 5.5 V
Top	Operating temperature	-40 to 85 °C

6.2 Voltage Selector

The development board utilizes a clever voltage selector system consisting of three pins and a jumper switch. The configuration of these pins determines the operating voltage of the board. By connecting the central pin to the +5V pin via the jumper, the board operates at 5V. On the other hand, by connecting the central pin to the +3.3V pin, the APK2112K regulator is activated, powering the board at 3.3V. It is crucial to ensure that the jumper switch is in the correct position according to the desired voltage to avoid possible damage to modules, components, and the board itself.

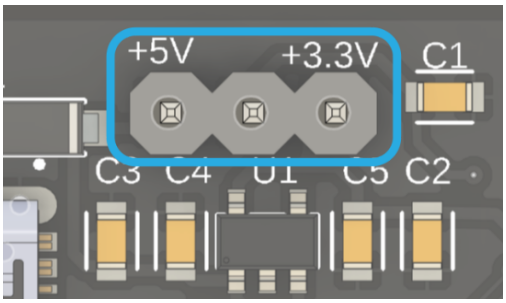

Fig. 6.1: USB Connector

6.3 JST Connectors

The board features two 1mm JST connectors, linked to different pins. The first connector directly connects to GPIO 3.0 and 3.1 of the microcontroller, while the second one is linked to pins 3.2 and 1.5. Both connectors operate in parallel to the selected power supply voltage via the jumper switch. These connectors are compatible with QWIIC, STEMMA QT, or similar pin distribution protocols. It is essential to verify that the selector voltage matches the system voltage to avoid circuit damage. Additionally, these connectors allow the board to be powered and offer functionalities such as PWM and serial communication.

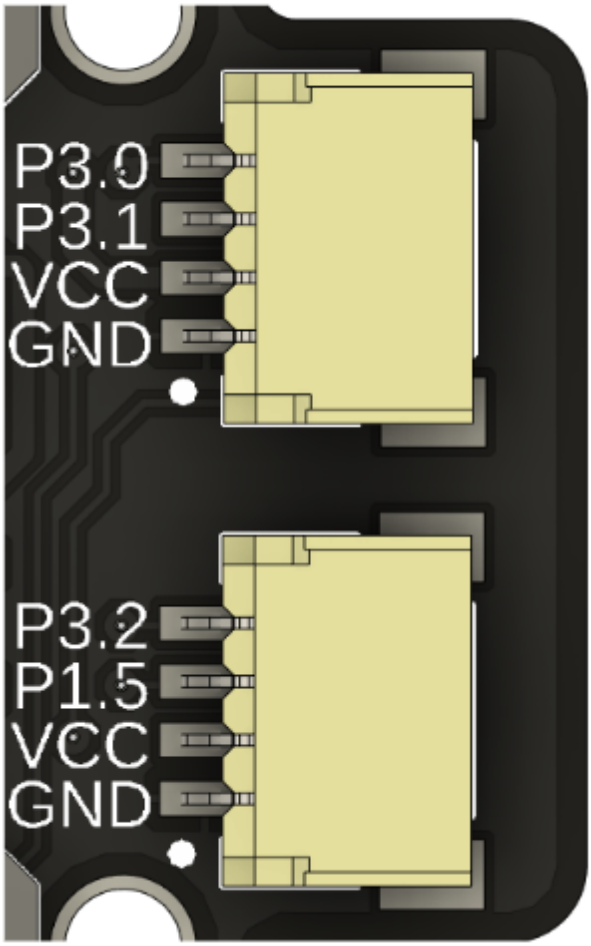

Fig. 6.2: JST Connectors

6.4 Built-In LEDs

The board features two LEDs directly linked to the microcontroller. The first one is connected to pin 3.4, while the second one is a Neopixel LED connected to pin 3.3. This Neopixel provides an output with two headers, one connected to the data output and the other to the board's ground, allowing for the external connection of more LEDs. To use this output, simply connect the DOUT pin to the DIN pin of the next LED in the row. As for power supply, you can use the VCC pin, provided that the external LEDs can operate at this voltage. Otherwise, it will be necessary to power them using an external source.

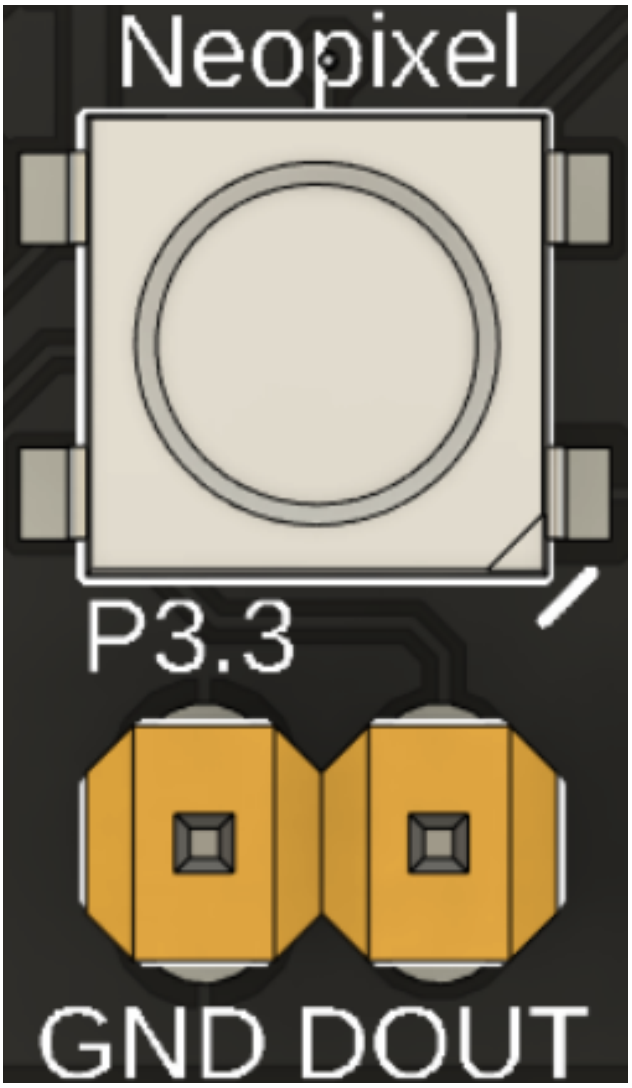

Fig. 6.3: Neopixel LED

CHAPTER

SEVEN

ANALOG TO DIGITAL CONVERTER (ADC)

The CH552 has four ADC channels, which can be used to read analog values from sensors. The ADC channels are multiplexed with the GPIO pins, so you can use any GPIO pin as an ADC input. the distribution of the ADC channels is as follows.

Table 7.1: Pin Mapping

PIN	ADC Channel
A0	P1.1
A1	P1.4
A2	P1.5
A3	P3.2

The ADC has a resolution of 8 bits, which means it can read values from 0 to 255. The ADC can be configured to read values from 0 to 5V, or from 0 to 3.3V, depending on the VCC voltage:

```
#include "src/config.h" // user configurations
#include "src/system.h" // system functions
#include "src/gpio.h"   // for GPIO
#include "src/delay.h"  // for delays

void main(void)
{
    CLK_config();
    DLY_ms(5);

    ADC_input(PIN_ADC);

    ADC_enable();
    while (1)
    {
        int data = ADC_read(); // Assuming ADC_read() returns an int
    }
}
```

CHAPTER

EIGHT

I2C (INTER-INTEGRATED CIRCUIT)

I2C is a serial communication protocol, so data is transferred bit by bit along a single wire (the SDA line). The SCL line is used to synchronize the data transfer. The I2C protocol is a master-slave protocol, which means that the communication is always initiated by the master device. A master device can communicate with one or multiple slave devices.

Note: The microcontroller use bit banging to communicate with the I2C devices.

All pin configurations are defined in the *config.h* file. The I2C pins are defined as follows

Table 8.1: I2C Pinout

PIN	I2C 1	I2C 2
SDA	P15	P31
SCLK	P32	P30

8.1 SSD1306 Display

The OLED display is a 128x64 pixel monochrome display that uses the I2C protocol for communication.

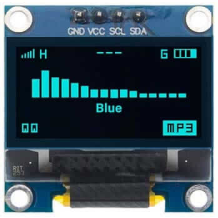

Fig. 8.1: SSD1306 Display

The following code snippet shows how to initialize the OLED display and print a message on the screen:

```
void beep(void) {
uint8_t i;
for(i=255; i; i--) {
    PIN_low(PIN_BUZZER);
    DLY_us(125);
    PIN_high(PIN_BUZZER);
    DLY_us(125);
}
}

void main(void) {
// Setup
CLK_config();                           // configure system clock
DLY_ms(5);                              // wait for clock to stabilize

OLED_init();                            // init OLED

OLED_print("*  UNITelectronics  *");
OLED_print("----------------------\n");
OLED_print("Ready\n");
beep();
while(1) {

}
}
```

Tip: Remember to includes library corresponding to the OLED display in the file.

CHAPTER

NINE

INTERRUPTS

The CH552 microcontroller supports 14 sets of interrupt signal sources. These include 6 sets of interrupts (INT0, T0, INT1, T1, UART0, and T2), which are compatible with the standard MCS51, and 8 sets of extended interrupts (SPI0, TKEY, USB, ADC, UART1, PWMX, GPIO, and WDOG). The GPIO interrupt can be chosen from 7 I/O pins.

Table 9.1: Default priority sequence of interrupt sources

Interrupt src.	Entry addr.	Inter-rupt #	Description	Default prio. seq.
INT_NO_INT0	0x0003	0	External interrupt 0	High priority
INT_NO_TMR0	0x000B	1	Timer 0 interrupt	↓
INT_NO_INT1	0x0013	2	External interrupt 1	↓
INT_NO_TMR1	0x001B	3	Timer 1 interrupt	↓
INT_NO_UART0	0x0023	4	UART0 interrupt	↓
INT_NO_TMR2	0x002B	5	Timer 2 interrupt	↓
INT_NO_SPI0	0x0033	6	SPI0 interrupt	↓
INT_NO_TKEY	0x003B	7	Touch key timer interrupt	↓
INT_NO_USB	0x0043	8	USB interrupt	↓
INT_NO_ADC	0x004B	9	ADC interrupt	↓
INT_NO_UART1	0x0053	10	UART1 interrupt	↓
INT_NO_PWMX	0x005B	11	PWM1/PWM2 interrupt	↓
INT_NO_GPIO	0x0063	12	GPIO Interrupt	↓
INT_NO_WDOG	0x006B	13	Watchdog timer interrupt	Low priority

The interrupt priority is determined by the interrupt number.

9.1 Timer 0/1 Interrupts

Timer0 and Timer1 are 16-bit timers/counters controlled by TCON and TMOD. TCON is responsible for timer/counter T0 and T1 startup control, overflow interrupt, and external interrupt control. Each timer consists of dual 8-bit registers forming a 16-bit timing unit. Timer 0's high byte counter is TH0, and its low byte counter is TL0. Similarly, Timer 1's high byte counter is TH1, and its low byte counter is TL1. Timer 1 can also serve as the baud rate generator for UART0. code example:

```
void timer0_interrupt(void) __interrupt(INT_NO_TMR0)        /* Timer0 interrupt service␣
↪routine (ISR) */
{
    PIN_toggle(PIN_BUZZER);
    TH0 = 0xFF;             /* 50ms timer value */
```

(continues on next page)

(continued from previous page)

```
    TL0 = 0x00;
}


int main(void)
{
    CLK_config();
    DLY_ms(5);
    PIN_output(PIN_BUZZER);
    EA  = 1;                /* Enable global interrupt */
    ET0 = 1;                /* Enable timer0 interrupt */

    TH0 = 0xFF;             /* 50ms timer value */
    TL0 = 0x00;
    TMOD = 0x01;            /* Timer0 mode1 */
    TR0 = 1;                /* Start timer0 */
    while(1);
}
```

9.2 External Interrupts

INT0 and INT1 are external interrupt input pins. When an external interrupt occurs, the corresponding interrupt service routine is executed. The external interrupt can be triggered by the falling edge, rising edge, or both edges of the external interrupt input signal. The trigger mode is determined by the external interrupt input pin.

code example:

```
void ext0_interrupt(void) __interrupt(INT_NO_INT0)
{
    PIN_toggle(PIN_LED);
}

int main(void)
{
    CLK_config();
    DLY_ms(5);
    PIN_output_OD(PIN_INT);
    PIN_output(PIN_LED);

    EA  = 1;     /* Enable global interrupt */
    EX0 = 1;    // Enable INT0
    IT0 = 1;    // INT0 is edge triggered

    while(1)
    {
        // Do nothing
    }

}
```

CHAPTER

TEN

PWM (PULSE WIDTH MODULATION)

The PWM module is used to generate a PWM signal on a pin. The PWM signal is generated by changing the duty cycle of the signal. The duty cycle is the ratio of the time the signal is high to the total time of the signal. The PWM module can be used to control the brightness of an LED, the speed of a motor, or the position of a servo motor.

The board contain two PWM pins, which are PIN_PWM and PIN_PWM2. The PWM module can be used to generate a PWM signal on these pins:

```
PWM 1 : P30/P15
PWM 2 : P31/P34
```

Some of the functions provided by the PWM module are:

```
#define MIN_COUNTER 10
#define MAX_COUNTER 254
#define STEP_SIZE   10

void change_pwm(int hex_value)
{
    PWM_write(PIN_PWM, hex_value);
}
void main(void)
{
    CLK_config();
    DLY_ms(5);
    PWM_set_freq(1);
    PIN_output(PIN_PWM);
    PWM_start(PIN_PWM);
    PWM_write(PIN_PWM, 0);
while (1)
{
    for (int i = MIN_COUNTER; i < MAX_COUNTER; i+=STEP_SIZE)
    {
        change_pwm(i);
        DLY_ms(20);
    }
    for (int i = MAX_COUNTER; i > MIN_COUNTER; i-=STEP_SIZE)
    {
        change_pwm(i);
        DLY_ms(20);
    }
}
```

(continues on next page)

(continued from previous page)

```
}
```

CHAPTER

ELEVEN

WS2812

This is a simple library for controlling WS2812 LEDs with an CH552 microcontroller. It is based on the WS2812 datasheet and the CH552 datasheet.

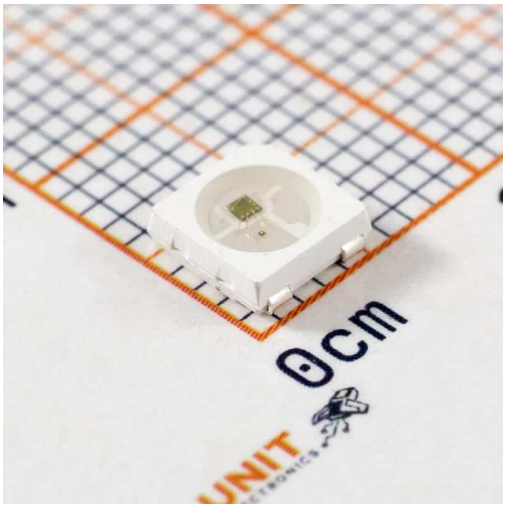

Fig. 11.1: WS2812 LED Strip

Table 11.1: Pin Mapping for WS2812

PIN	GPIO CH552
DOUT	P3.3

```
#define delay 100
#define NeoPixel 16 // Number Neopixel conect
#define level 100 // Ilumination level 0 to 255

void randomColorSequence(void) {

for(int j=0;j<NeoPixel;j++){
    uint8_t red = rand() % level;
    uint8_t green = rand() % level;
    uint8_t blue = rand() % level;
    uint8_t num = rand() % NeoPixel;
```

(continues on next page)

(continued from previous page)

```
    for(int i=0; i<num; i++){
        NEO_writeColor(0, 0, 0);
    }
    NEO_writeColor(red, green, blue);
    DLY_ms(delay);
    NEO_writeColor(0, 0, 0);
    }

    for(int l=0; l<9; l++){
        NEO_writeColor(0, 0, 0);
    }

}

void colorSequence(void) {

for(int j=0;j<=NeoPixel;j++){
        uint8_t red = rand() % level;
        uint8_t green = rand() % level;
        uint8_t blue = rand() % level;
    for(int i=0; i<j; i++){
        NEO_writeColor(red, green, blue);
    }
    DLY_ms(delay);
    for(int l=0; l<j; l++){
        NEO_writeColor(0, 0, 0);
    }
}
}

// ============================================================================
// Main Function
// ============================================================================
void main(void) {
NEO_init();                         // init NeoPixels
CLK_config();                       // configure system clock
DLY_ms(delay);                         // wait for clock to settle

// Loop
while (1) {
    randomColorSequence();
    DLY_ms(100);
    colorSequence();
    DLY_ms(100);
}
}
```

CHAPTER

TWELVE

COMMUNICATION SERIAL CDC

Cocket Nova development board is compatible with the USB CDC (Communication Device Class) protocol. This allows the Cocket Nova to be used as a virtual serial port. The CDC protocol is supported by most operating systems, including Windows, Linux, and macOS.

Note: The CDC protocol is implemented using the USB peripheral of the microcontroller. The USB peripheral is configured as a virtual serial port, which allows the microcontroller to communicate with the host computer using the USB cable.

The following code snippet shows how to configure the USB peripheral as a virtual serial port and this method only recive information from the host

```
void main(void) {
// Setup
CLK_config();                          // configure system clock
DLY_ms(5);                             // wait for clock to stabilize
CDC_init();                            // init USB CDC

// Loop
while(1) {
    if(CDC_available()) {                // something coming in?
    char c = CDC_read();               // read the character ...
    CDC_writeflush(c);                 // ... and send it back
    }
}
}
```

12.1 USB CDC Serial Configuration for CH55x Microcontrollers

12.1.1 USB Passthrough for CH55x Microcontrollers

This project implements a simple USB passthrough functionality using CH551, CH552, or CH554 microcontrollers. The microcontroller acts as a USB Communication Device Class (CDC), enabling serial communication over USB. Data received via USB is immediately sent back to the host computer. Wiring

Connect the CH55x development board to your PC via USB. It should be automatically detected as a CDC device. Compilation Instructions:

```
Chip: CH551, CH552, or CH554
Clock: 16 MHz internal
Adjust firmware parameters in src/config.h if necessary.
Ensure SDCC toolchain and Python3 with PyUSB are installed.
Press the BOOT button on the board and keep it pressed while connecting it via USB to␣
↪your PC.
Run make flash immediately afterwards to flash the firmware.
For compilation using Arduino IDE, refer to instructions in the .ino file.
```

12.1.2 USB CDC PWM Controller for CH55x Microcontrollers

This project implements a USB CDC controlled PWM functionality using CH551, CH552, or CH554 microcontrollers. The microcontroller acts as a USB Communication Device Class (CDC), allowing serial communication over USB. Data received via USB is used to set the PWM value (0-255). Wiring

Connect the CH55x development board to your PC via USB. It should be automatically detected as a CDC device. Compilation Instructions:

```
Chip: CH551, CH552, or CH554
Clock: 16 MHz internal
Adjust firmware parameters in src/config.h if necessary.
Ensure SDCC toolchain and Python3 with PyUSB are installed.
Press the BOOT button on the board and keep it pressed while connecting it via USB to␣
↪your PC.
Run make flash immediately afterwards to flash the firmware.
For compilation using Arduino IDE, refer to instructions in the .ino file.
```

12.2 Linux Configuration for USB CDC Devices

To configure permissions for USB CDC devices (/dev/ttyACM0), follow these steps:

Create a new udev rule file:

```
sudo nano /etc/udev/rules.d/99-custom-usb.rules
```

Add the following rule to the file (replace idVendor and idProduct with your device's actual IDs):

```
SUBSYSTEM=="tty", ATTRS{idVendor}=="1209", ATTRS{idProduct}=="27dd", GROUP="dialout",␣
↪MODE="0666"
```

Save the file (Ctrl + O in nano, then Enter) and exit nano (Ctrl + X).

Reload udev rules for changes to take effect:

```
sudo udevadm control --reload-rules
```

12.2.1 Example Commands for Serial Communication

Send data to USB device:

```
echo -e 'Hello World!\n' > /dev/ttyACM0
```

Read data from USB device:

```
cat /dev/ttyACM0
```

These commands allow you to interact with USB CDC devices connected to your Linux system. Adjust the device path (/dev/ttyACM0) as per your setup.

12.3 Windows Configuration for USB CDC Devices

To configure permissions for USB CDC devices in Windows, follow these steps:

1. Identify the device's COM port number in Device Manager.

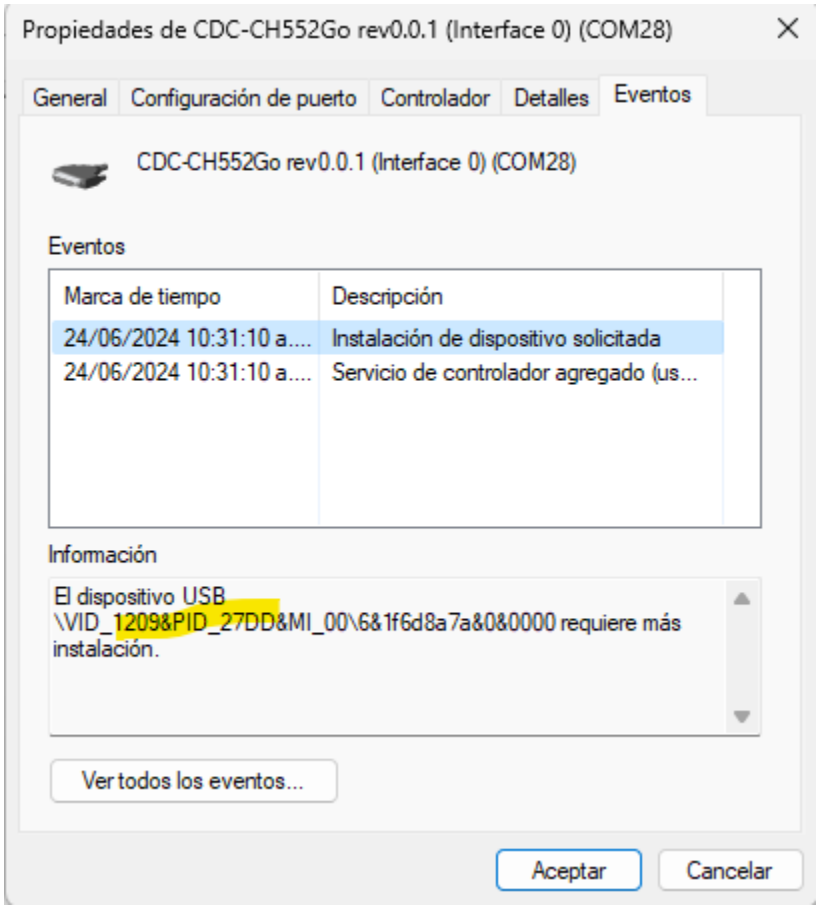

Fig. 12.1: CDC Serial Device Manager

2. Right-click on the device and select Properties.
3. Open Zadig (https://zadig.akeo.ie/).

4. Go to Options > List All Devices.
5. Select the device from the drop-down list.

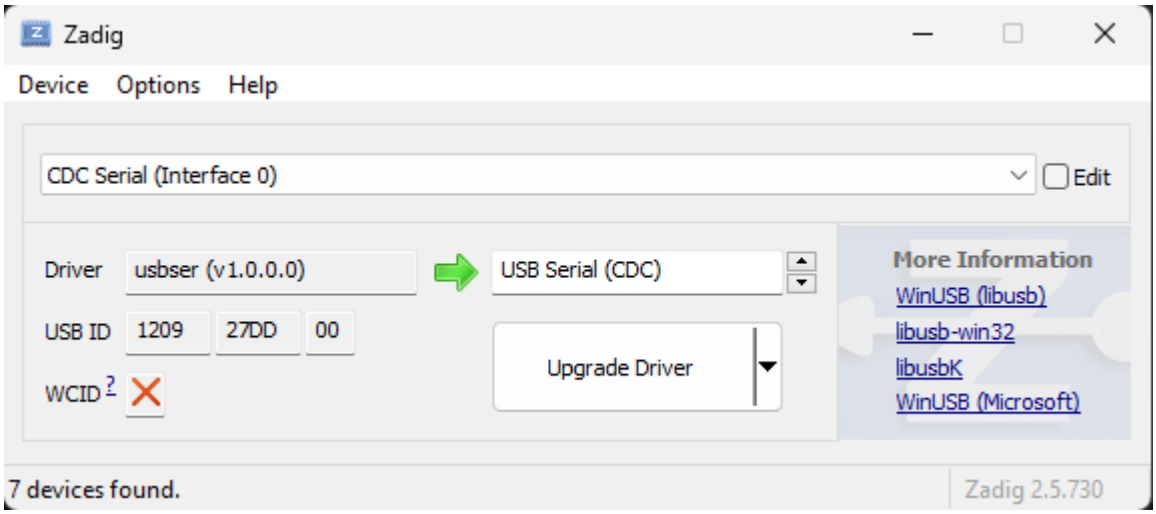

Fig. 12.2: Zadig CDC Device Selection