Documentation CH552

Release 0.0.1

Cesar Bautista

Apr 29, 2024

CONTENTS

1 About **3**
- 1.1 Describing . . . 3
- 1.2 Requirements . . . 3
- 1.3 PinOut . . . 4

2 CH55x with SDCC and Ubuntu 23.10 **5**
- 2.1 Ubuntu 23.10 Installer Environment . . . 5
- 2.2 Install pyusb . . . 6
- 2.3 Error with pip . . . 6

3 CH55x with SDCC and Windows **7**
- 3.1 Compiler Installation . . . 7
- 3.2 Configuring MAKE . . . 7
- 3.3 Install pyusb . . . 10
- 3.4 Update driver . . . 11

4 Compile and Flash CH55x with SDCC **13**
- 4.1 Running a Program . . . 13
- 4.2 Flashing the Program . . . 13

5 General Board Control **15**
- 5.1 Recommended Operating Conditions . . . 15
- 5.2 Operating Mode . . . 15

6 Analog to Digital Converter (ADC) **19**

7 I2C (Inter-Integrated Circuit) **21**
- 7.1 Description . . . 21
- 7.2 OLED Display SD1306 . . . 21

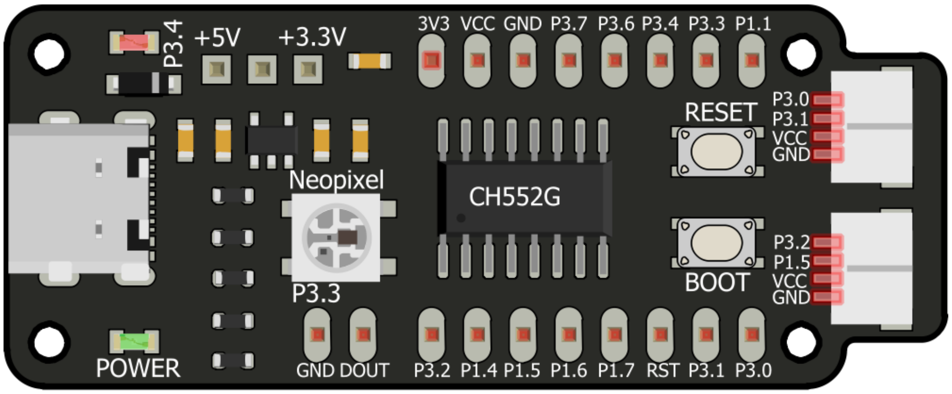
8 Interrupts **23**
- 8.1 Timer 0/1 Interrupts . . . 23
- 8.2 External Interrupts . . . 24

9 PWM (Pulse Width Modulation) **25**

10 WS2812 **27**

Cocket Nova

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.



Note: This project is under active development.

CHAPTER

ONE

ABOUT

1.1 Describing

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.

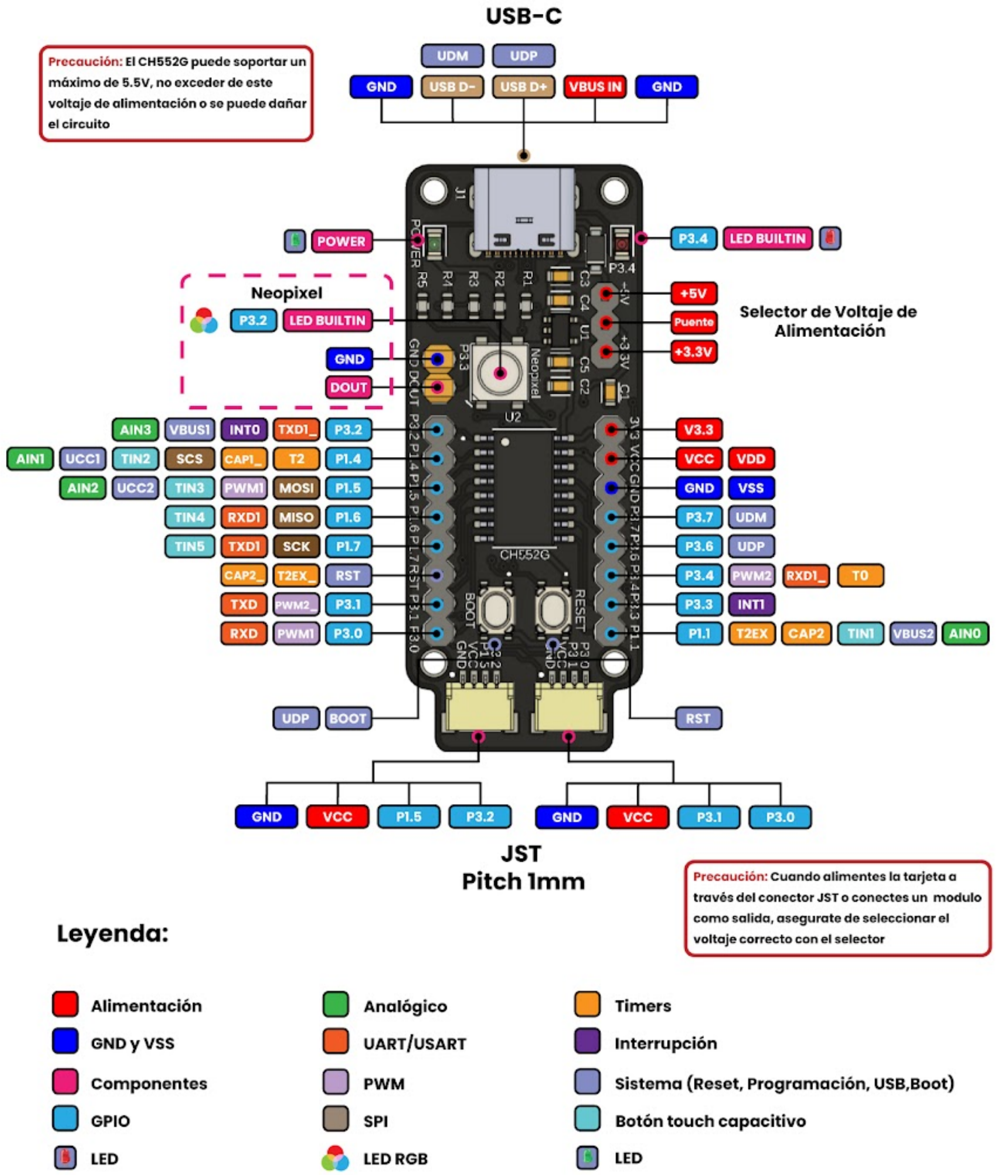
1.2 Requirements

- 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.

1.3 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:



CHAPTER

TWO

CH55X WITH SDCC AND UBUNTU 23.10

2.1 Ubuntu 23.10 Installer Environment

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

Update the operating system:

```
sudo apt update
```

Install *make* and *binutils*:

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

Clone the examples with the main code:

```
git clone https://github.com/Cesarbautista10/CH55x_SDCC_Examples.git
```

Navigate to the path:

```
cd ~/CH55x_SDCC_Examples/Software/examples/0.\ 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 flash    compile, build, and upload blink.bin to the device
make clean    remove all build files
```

Connect a device with the BOOT button pressed:

```
lsusb
```

The device will be shown with this description:

```
Descriptor
```

2.2 Install pyusb

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

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

Then verify the installation:

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

2.3 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
```

CHAPTER

THREE

CH55X WITH SDCC AND WINDOWS

3.1 Compiler Installation

Follow the steps below to install the necessary tools

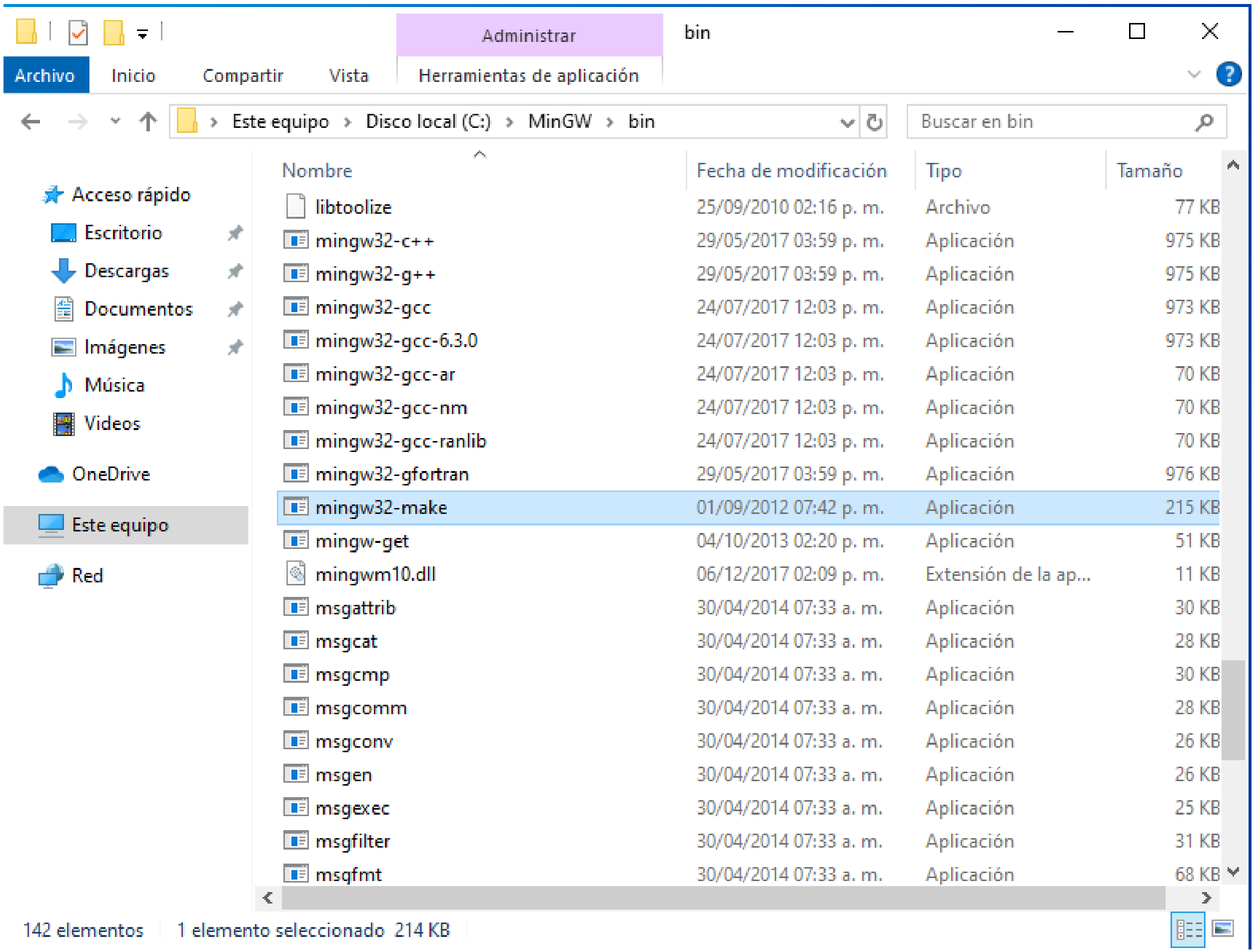
- **Installing Git for Windows**
  Download and install Git for Windows from the official Git website.
- **Installing SDCC**
  Download and install the latest version of SDCC. You can find the latest version on the SDCC downloads page. here a alternative
- **Installing MinGW**
  Install MinGW, which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

3.2 Configuring MAKE

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

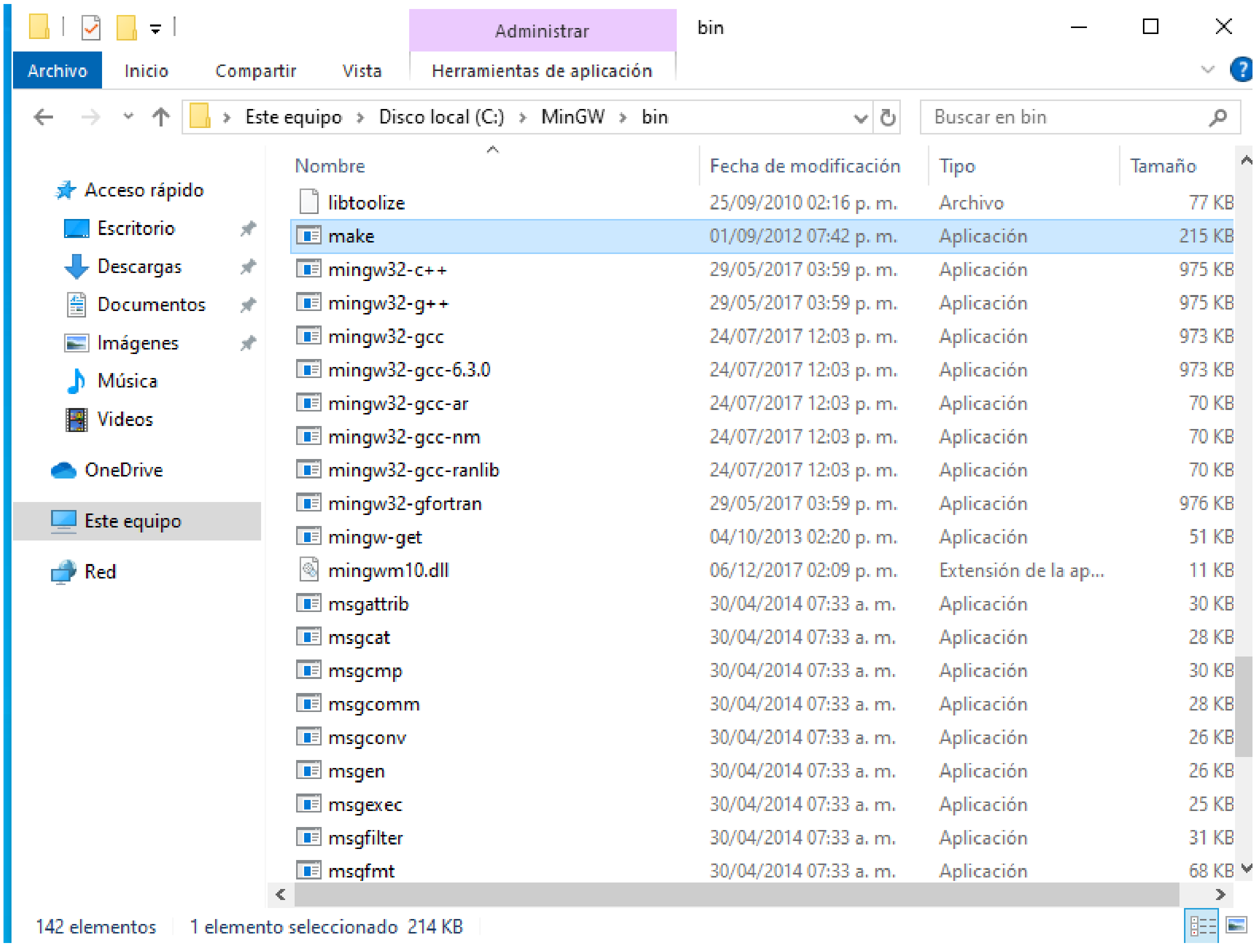
```
C:\MinGW\bin
```

Locate the file



Administrar
bin
Archivo
Inicio
Compartir
Vista
Herramientas de aplicación
Este equipo › Disco local (C:) › MinGW › bin
Buscar en bin
Acceso rápido
Escritorio
Descargas
Documentos
Imágenes
Música
Videos
OneDrive
Este equipo
Red
Nombre
Fecha de modificación
Tipo
Tamaño
libtoolize 25/09/2010 02:16 p. m. Archivo 77 KB
mingw32-c++ 29/05/2017 03:59 p. m. Aplicación 975 KB
mingw32-g++ 29/05/2017 03:59 p. m. Aplicación 975 KB
mingw32-gcc 24/07/2017 12:03 p. m. Aplicación 973 KB
mingw32-gcc-6.3.0 24/07/2017 12:03 p. m. Aplicación 973 KB
mingw32-gcc-ar 24/07/2017 12:03 p. m. Aplicación 70 KB
mingw32-gcc-nm 24/07/2017 12:03 p. m. Aplicación 70 KB
mingw32-gcc-ranlib 24/07/2017 12:03 p. m. Aplicación 70 KB
mingw32-gfortran 29/05/2017 03:59 p. m. Aplicación 976 KB
mingw32-make 01/09/2012 07:42 p. m. Aplicación 215 KB
mingw-get 04/10/2013 02:20 p. m. Aplicación 51 KB
mingwm10.dll 06/12/2017 02:09 p. m. Extensión de la ap... 11 KB
msgattrib 30/04/2014 07:33 a. m. Aplicación 30 KB
msgcat 30/04/2014 07:33 a. m. Aplicación 28 KB
msgcmp 30/04/2014 07:33 a. m. Aplicación 30 KB
msgcomm 30/04/2014 07:33 a. m. Aplicación 28 KB
msgconv 30/04/2014 07:33 a. m. Aplicación 26 KB
msgen 30/04/2014 07:33 a. m. Aplicación 26 KB
msgexec 30/04/2014 07:33 a. m. Aplicación 25 KB
msgfilter 30/04/2014 07:33 a. m. Aplicación 31 KB
msgfmt 30/04/2014 07:33 a. m. Aplicación 68 KB
142 elementos
1 elemento seleccionado 214 KB

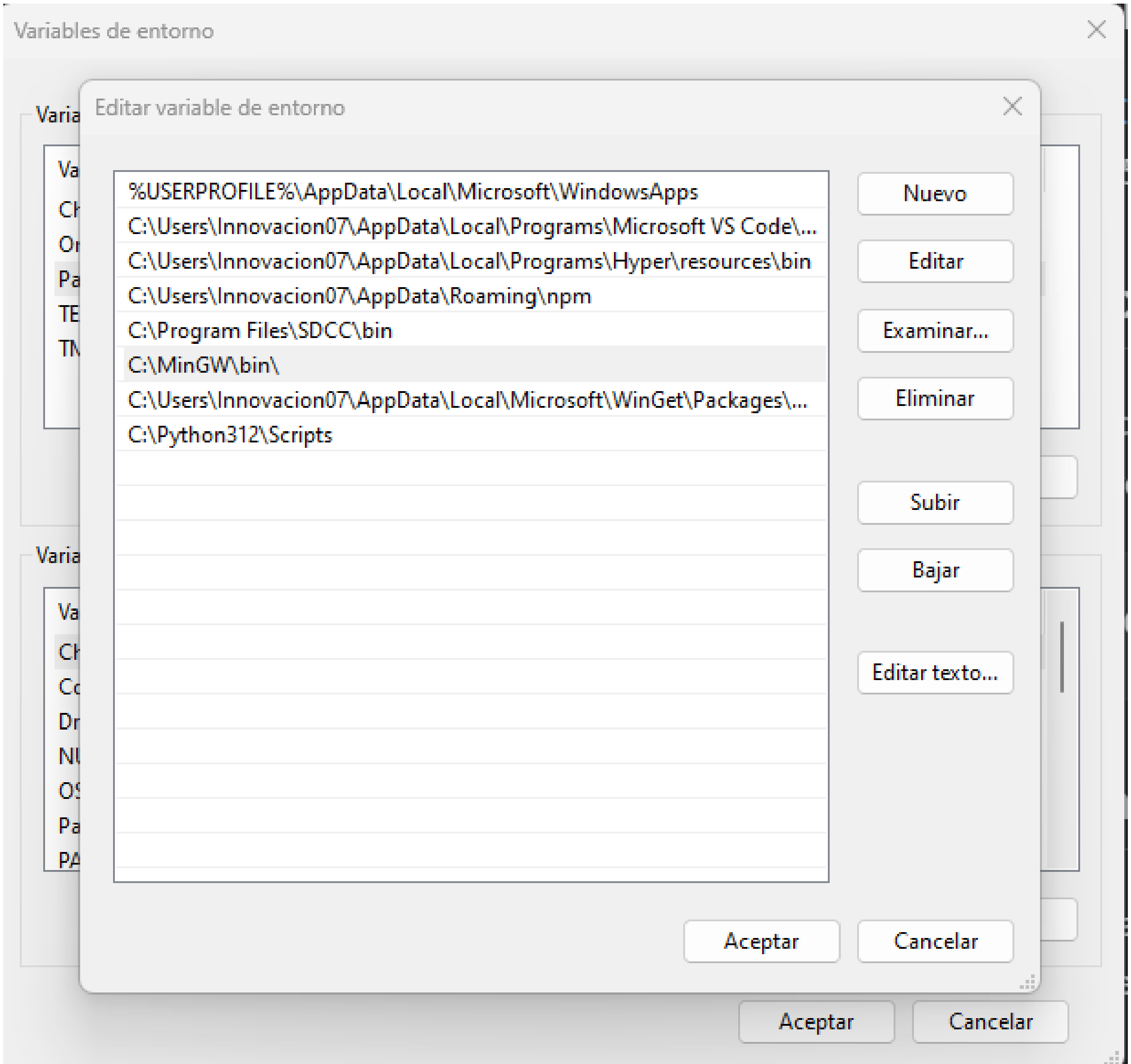
Rename it:



Warning: Sometimes, it is necessary to create a copy of the file and rename it to make.exe.

path:

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



3.3 Install pyusb

Verify the installation with *python –version*.

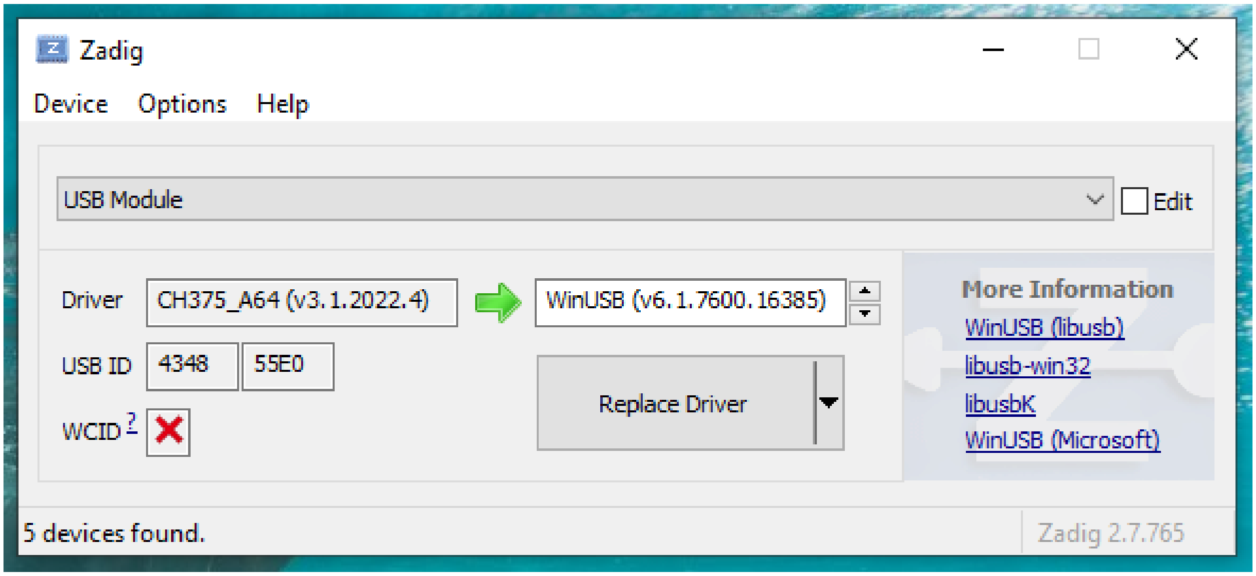
If not installed, download and install: - Python

Use pip for install py usb:

```
pip install pyusb
```

3.4 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.



CHAPTER

FOUR

COMPILE AND FLASH CH55X WITH SDCC

4.1 Running a Program

To run the program, use a bash terminal.

Clone the examples with the main code:

```
git clone https://github.com/Cesarbautista10/ESE_CH552_Examples_C.git
```

Navigate to the path:

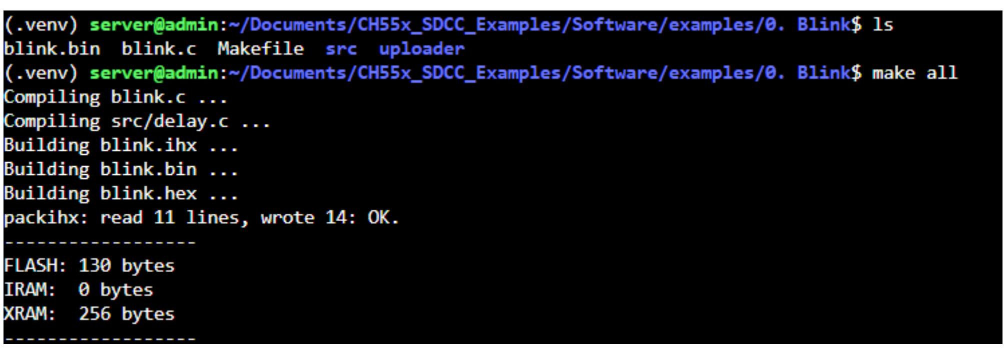
```
cd ~/CH55x_SDCC_Examples/Software/examples/Blink/
```

Connect a device with the BOOT button pressed.

Execute the command:

```
make all
```

This will compile the project and generate files with the following extensions:

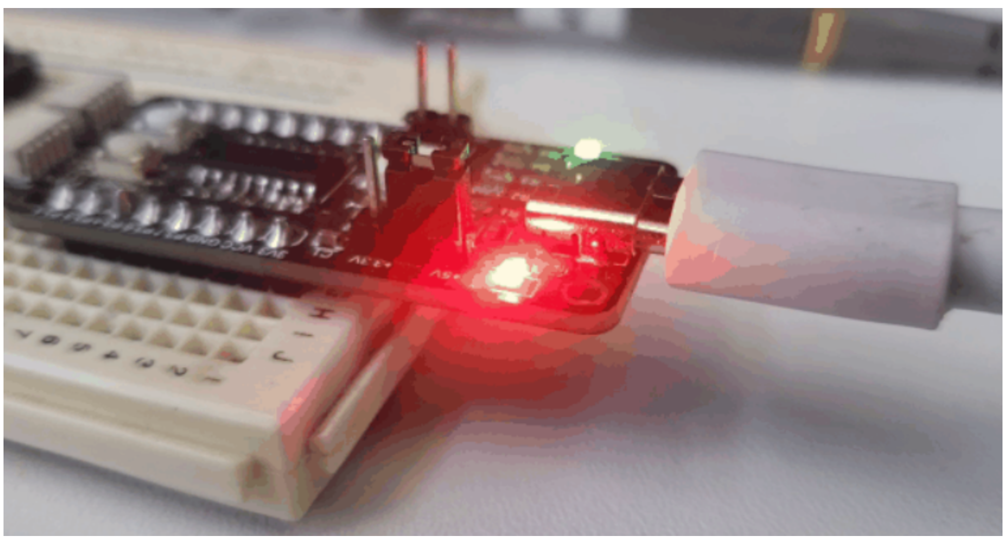


4.2 Flashing the Program

Connect a device and press the BOOT button, then write the command:

```
make flash
```

If the project is successful, the code will generate a blinking effect as shown below:



CHAPTER

FIVE

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.

5.1 Recommended Operating Conditions

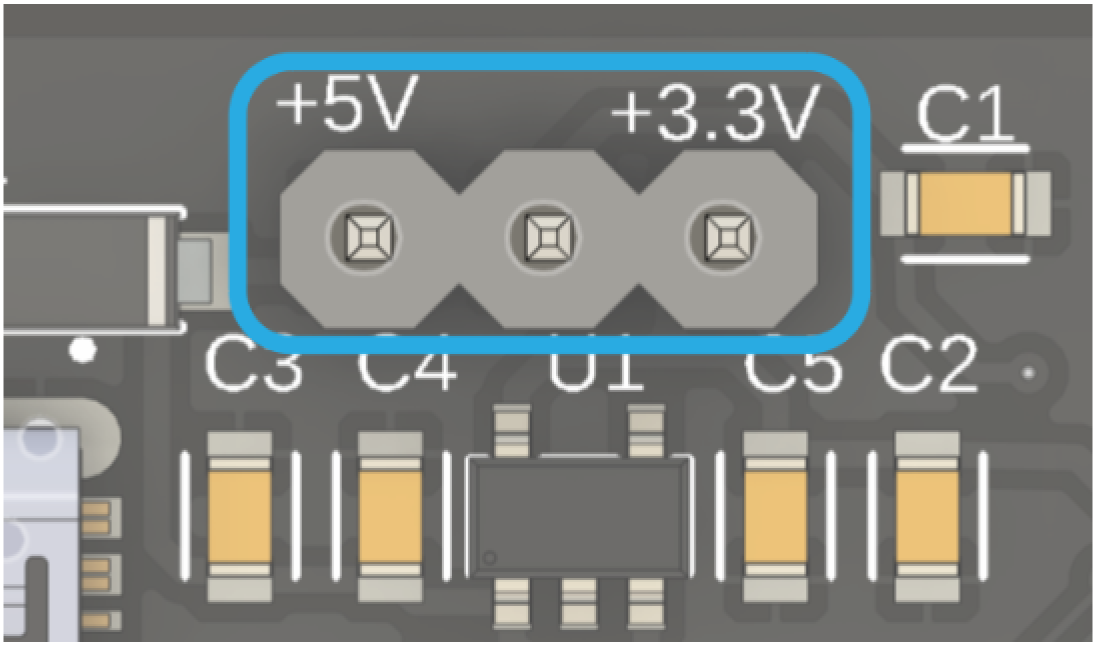
Table 1: Recommended Operating Conditions

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

5.2 Operating Mode

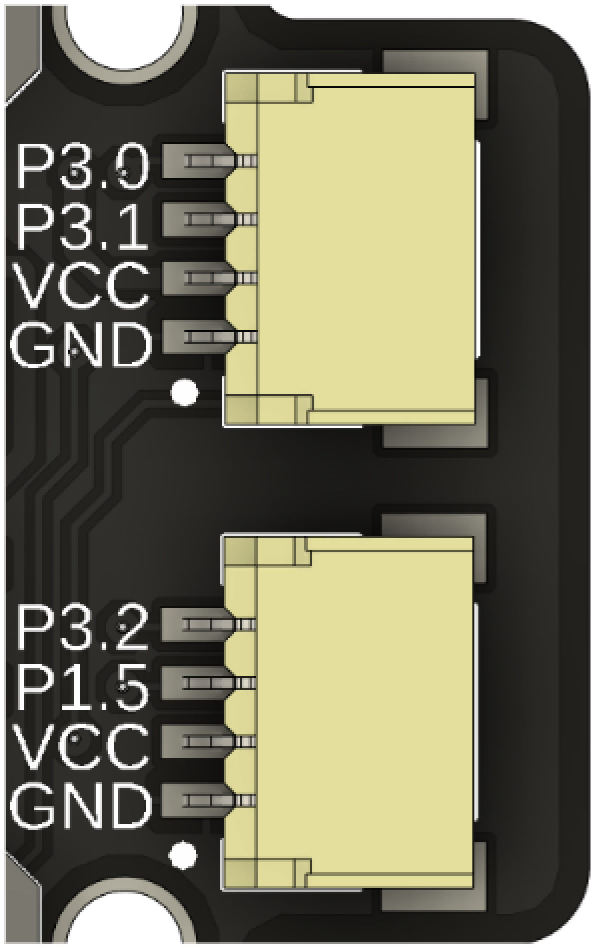
5.2.1 Voltage Selector

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.



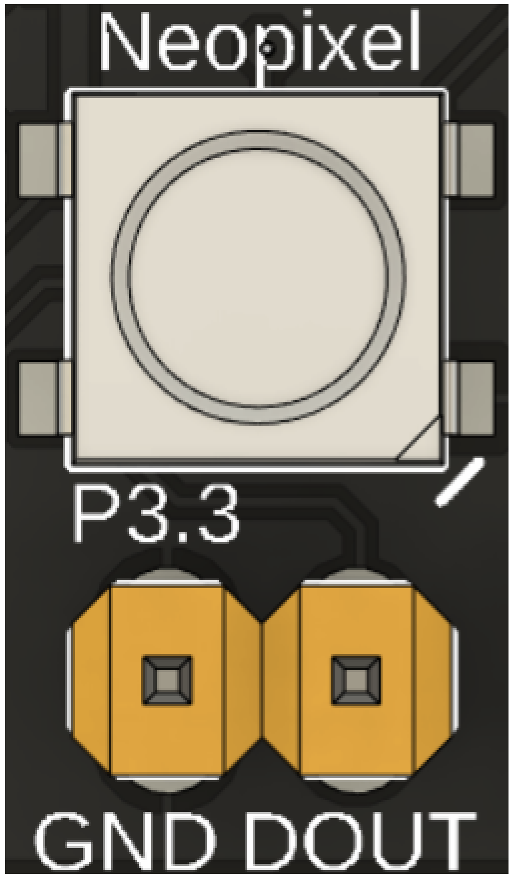
5.2.2 JST Connectors

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.



5.2.3 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.


Neopixel
P3.3
GND DOUT

CHAPTER

SIX

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:

```
>> ADC0: P1.1
>> ADC1: P1.4
>> ADC2: P1.5
>> ADC3: 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 SEVEN

I2C (INTER-INTEGRATED CIRCUIT)

7.1 Description

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. The master device generates the clock signal and initiates the data transfer. The slave devices are addressed by the master device, and they respond to the master device based on the address they are assigned. The I2C protocol supports multiple devices on the same bus, and each device has a unique address. This allows multiple devices to communicate with each other using the same bus.

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:

```
#define I2C_SCL_PIN  P32
#define I2C_SDA_PIN  P15
```

7.2 OLED Display SD1306

The OLED display is a 128x64 pixel monochrome display that uses the I2C protocol for communication. The display has a built-in controller that handles the display refresh and data transfer. The display is controlled by the microcontroller using the I2C protocol. The display has a resolution of 128x64 pixels, which allows it to display text, graphics, and images. The display is monochrome, which means that it can only display one color (white) on a black background. The display has a built-in controller that handles the display refresh and data transfer. The display is controlled by the microcontroller using the I2C protocol. The display has a resolution of 128x64 pixels, which allows it to display text, graphics, and images. The display is monochrome, which means that it can only display one color (white) on a black background:

```
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);
}
}
```

(continues on next page)

(continued from previous page)

```
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) {

}
}
```

CHAPTER

EIGHT

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 1: Default priority sequence of interrupt sources

Interrupt src.	Entry addr.	Interrupt #	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.

8.1 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);
}
```

8.2 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

NINE

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 TEN

WS2812

This is a simple library for controlling WS2812 LEDs with an CH552 microcontroller. It is based on the [WS2812 datasheet](https://cdn-shop.adafruit.com/datasheets/WS2812.pdf) and the [CH552 datasheet](https://www.waveshare.com/w/upload/1/1d/CH552T-DS-EN-1.0.pdf).

The library is written in C and can be used with the [SDCC](http://sdcc.sourceforge.net/) compiler.

Code:

```
#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;

    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++){
```

(continues on next page)

(continued from previous page)

```
        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);
}
}
```