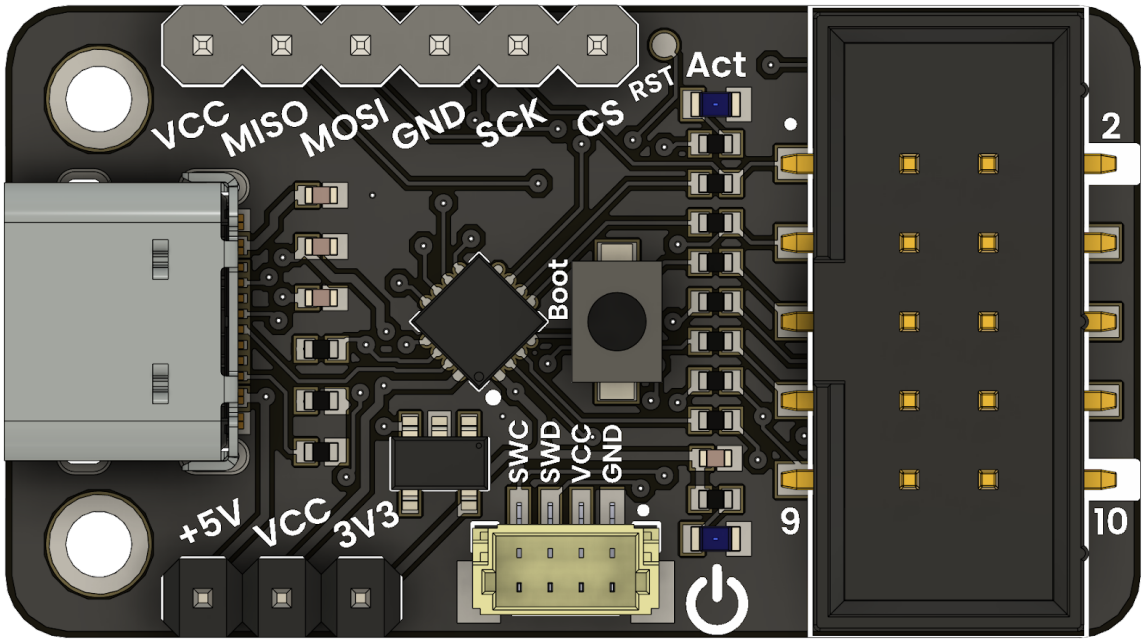


CH552 USB Multi-Protocol Programmer User Guide and Technical Reference

Release 0.0.1

Department of Research, Innovation, and Development

Oct 21, 2025

Contents

Note: This documentation is actively evolving. For the latest updates and revisions, please visit the project's GitHub repository.

Multi-Protocol Programmer

The **USB Multi-Protocol Programmer** is a compact and cost-effective device designed for embedded systems development, testing, and debugging. It supports multiple hardware architectures including **AVR**, **ARM Cortex-M (CMSIS-DAP)**, and **CPLD (MAX II)**, making it ideal for a wide range of applications such as firmware development, educational labs, and low-volume production environments.

This programmer is built around the **CH552 microcontroller**, which is based on the enhanced **8051 architecture**. It offers native USB support and a range of digital interfaces (GPIO, SPI, I2C, UART), enabling seamless communication between the host system and the target hardware.

Microcontroller Core

The programmer integrates a **CH552 microcontroller** with the following characteristics:

- 8051-based enhanced core, up to 24 MHz.
- Native USB 2.0 Full-Speed device.
- Multiple GPIO pins for signal control and mapping.
- SPI, I2C, and UART interfaces for protocol bridging.
- Low power consumption and small form factor.

Features

- **Multi-architecture support**: Compatible with AVR (ISP), ARM Cortex-M (CMSIS-DAP), and CPLD (JTAG).
- **In-System Programming (ISP)**: Flash microcontrollers without desoldering.
- **Real-time debugging**: Step-through and breakpoint debugging with OpenOCD and PyOCD.
- **JTAG boundary-scan**: For CPLD configuration and board testing.
- **Configurable GPIOs**: Adaptable for use as JTAG, SWD, or ISP lines.
- **USB 2.0 interface**: Direct connection to host PC using USB CDC or HID.
- **Toolchain compatibility**: Works with avrdude, OpenOCD, PyOCD, urJTAG, and others.
- **Cross-platform support**: Compatible with Linux and partially supported on Windows.

Advantages

- **Compact design**: Suitable for breadboards and embedded setups.
- **Versatility**: One device for multiple programming and debugging protocols.
- **Open-source firmware**: Fully customizable and community-supported.
- **Cost-effective**: Inexpensive alternative to commercial debuggers and programmers.
- **Linux-friendly**: No need for proprietary drivers on Linux systems.
- **Ideal for education**: Can be used in microcontroller courses and workshops.

Limitations

- **External power required**: Cannot supply power to high-current target boards.
- **Learning curve**: Requires knowledge of protocols like CMSIS-DAP, JTAG, or AVR ISP.
- **Firmware updates**: May require reflashing to support new features or targets.
- **Partial Windows support**: Some tools may require manual setup or driver adjustments.

Compatibility

CMSIS-DAP (ARM Cortex-M)

- Compatible with CMSIS-DAP v2.0 protocol.
- Supported by OpenOCD and PyOCD.
- Tested with:
  - STM32F0
  - RP2040 (Raspberry Pi Pico)
  - PY32 series
  - Other Cortex-M0/M3/M4 devices

AVR ISP

- Works with avrdude using USBasp-like interface.
- Supports:
  - ATmega328P
  - ATtiny85
  - ATmega2560
  - Other classic 8-bit AVR microcontrollers

CPLD JTAG

- Supports Intel (formerly Altera) MAX II series.
- Compatible with JTAG tools like urJTAG or openFPGALoader.
- JTAG signals exposed via GPIO (TDI, TDO, TCK, TMS).

Use Cases

- Firmware flashing and in-system programming.
- Debugging embedded applications with CMSIS-DAP.
- Educational labs and training environments.
- Low-cost production line programming.
- Boundary-scan tests for hardware bring-up.
- CPLD configuration and prototyping.

Resources

- Firmware: [https://github.com/wagiminator/CH552-DAPLink{}](https://github.com/wagiminator/CH552-DAPLink)
- CH552 Datasheet: Available from WCH official website.
- Tools:
  - OpenOCD, PyOCD
  - avrdude
- Community support: GitHub issues, Reddit, Hackaday, forums.

CHAPTER
ONE

TERMS, ACKNOWLEDGMENTS, AND LICENSES

1.1 Terms and Conditions

By using, modifying, or distributing the documentation, firmware, or hardware designs included in this repository, you agree to the following terms:

- All materials are provided **"as-is"**, without warranty of any kind.
- The authors and contributors shall not be held responsible for **any damages**, data loss, or legal issues arising from the use of these materials.
- Usage is intended for **educational, development, prototyping**, and other lawful purposes.
- When redistributing or reusing any part of this project, you must **retain attribution** and comply with the corresponding license terms of each component.

1.2 Acknowledgments and Contributors

This project builds upon the work of several open-source developers and projects:

1.2.1 CMSIS-DAP (DAPLink Firmware for CH552)

- **Stefan Wagner** Project: CH552-DAPLink License: Creative Commons BY-SA 3.0 Description: CMSIS-DAP firmware and hardware design
- **Ralph Doncaster** Source: nerdralph/ch554_sdcc Description: Original CMSIS-DAP firmware implementation for CH554 (SDCC)
- **Deqing Sun** Source: CH55xduino Description: CH552/CH554 Arduino-compatible toolchain

1.2.2 USB-Blaster Firmware (CH552G)

- **Vladimir Duan** Project: CH55x-USB-Blaster License: MIT Description: USB-Blaster JTAG emulation for CH55x
- **Blinkinlabs** SDK Source: ch554_sdcc Description: SDK for CH552/CH554 (SDCC)
- **Doug Brown** Blog: Fixing a Knockoff Altera USB Blaster Description: Insights into compatibility and firmware flashing

1.3 Hardware License

All hardware designs (schematics, layouts, and design files) in this repository are released under the **MIT License**, allowing unrestricted use, modification, and distribution, provided the original license and attribution are retained.

1.4 Resources and References

Table 1.1: Source URLs

Project / Tool	Source URL
CH552 DAPLink	https://github.com/wagiminator/CH552-DAPLink
picoDAP	https://github.com/wagiminator/CH552-picoDAP
CH55xDuino	https://github.com/DeqingSun/ch55xduino
CMSIS-DAP Handbook	https://os.mbed.com/handbook/CMSIS-DAP
CH55x USB-Blaster	https://github.com/VladimirDuan/CH55x-USB-Blaster
SDCC Compiler	https://sdcc.sourceforge.net/
CH554 SDK	https://github.com/Blinkinlabs/ch554_sdcc

1.5 Licenses

1.5.1 Documentation & Visual Content

This user guide and its visual content are licensed under:

Creative Commons Attribution-ShareAlike 3.0 Unported License

1.5.2 Firmware Projects

- **CH552-DAPLink**: Creative Commons BY-SA 3.0 — © Stefan Wagner
- **CH55x-USB-Blaster**: MIT License — © Vladimir Duan
- **CH55x SDK / Tools**: MIT License — © Blinkinlabs

1.5.3 Hardware Repository

- All PCB designs and schematics are released under the **MIT License**.

Note: If you distribute this product with third-party firmware (e.g., CMSIS-DAP), you are responsible for ensuring license compliance. Only firmware developed by Unit Electronics and released under the MIT license is supported for commercial redistribution.

1.5.4 Preloaded USB-Serial Firmware

This product may include preloaded firmware based on the project by **Kongou Hikari**: "USB to Serial Converter firmware for CH552T". Original source: *[https://github.com/diodep/ch55x_dualserial/tree/master]* License: MIT

Under the terms of the MIT License, users are free to modify or replace the firmware. Unit Electronics provides this firmware for convenience only and does not offer performance guarantees.

CHAPTER
TWO

CH55X UNIT SDK DOCKER

Note: Portable SDK for CH552 firmware development using SDCC in Docker containers. Includes a cross-platform command-line tool (*spkg*) to simplify compilation on both Linux and Windows.

2.1 Project Structure

```
ch552-docker-sdk/
├── spkg/                    # Standalone␣
↪CLI build system
│   ├── spkg                 # CLI launcher␣
↪(bash script)
│   ├── Dockerfile           # SDCC-based␣
↪build environment
│   └── docker-compose.yml   # Container␣
↪configuration
├── template/                # CH552␣
↪example projects
│   └── Blink/               # Blink␣
↪example (main.c, src/, tools/, Makefile)
└── README.md
```

2.2 Main Features

- Unified command-line tool: "spkg" available on Linux and Windows (using Git Bash).
- No manual installation of SDCC or other toolchains required.
- Uses Docker containers to provide a completely isolated build environment.
- Based on a project system with a Makefile, compatible with CH552/CH55x microcontrollers.
- Includes an example project located in the `template/` directory.

2.3 Requirements

2.3.1 Common (All Platforms)

- Docker Desktop

Linux/macOS

- Git
- Python 3
- Bash shell
- Superuser privileges required to run Docker

Windows

- Docker Desktop
- Git Bash
- Docker Desktop with WSL2 or Hyper-V backend enabled
- MinGW64 (included with Git Bash) for the `make` command

Note: Running `spkg` on Linux might require `sudo` if the user is not part of the `docker` group. You can add your user with: `sudo usermod -aG docker $USER && newgrp docker`

2.4 Installation

Linux/macOS

Clone the repository:

```
git clone git@github.com:UNIT-Electronics-
↪MX/unit_ch55x_docker_sdk.git
cd ch552-docker-sdk/spkg
chmod +x spkg
```

(Optional) Install it globally:

Windows

Clone the repository:

```
git clone git@github.com:UNIT-Electronics-
↪MX/unit_ch55x_docker_sdk.git
cd ch552-docker-sdk
```

Note: Require use command ./spkg/spkg.bat to run the spkg command.

2.4.1 Building the Docker Image

Linux/macOS

Build the Docker image:

```
spkg compose
```

Windows

```
./spkg/spkg.bat compose
```

Warning: Ensure Docker is running and that your user has permission to execute it. You can verify by running `docker ps`. If no errors appear, Docker is running correctly.

2.4.2 Creating a New Project

Note: This command will create a new directory with the specified name.

Linux/macOS

To create a new project, run:

```
spkg init template/project
```

Windows

To create a new project, run:

```
./spkg/spkg.bat init template/project
```

2.4.3 Showing Help

On Linux:

```
spkg --help
```

On Windows:

```
./spkg/spkg.bat --help
```

2.4.4 Compiling a Project

On Linux:

```
spkg -p ./template/Blink
```

On Windows:

```
./spkg/spkg.bat -p ./template/Blink
```

2.4.5 Execute `make clean`, `all`, `hex`, etc.

On Linux:

```
spkg -p ./template/Blink clean
spkg -p ./template/Blink all
spkg -p ./template/Blink hex
```

On Windows:

```
spkg.bat -p ./template/Blink clean
spkg.bat -p ./template/Blink all
spkg.bat -p ./template/Blink hex
```

2.5 Output

The compiled binary will be generated at:

```
template/Blink/build/main.bin
```

You can flash it using:

- `tools/chprog.py`
- WCHISPTool

CHAPTER
THREE

DOCKER CONFIGURATION FOR NON-PRIVILEGED USERS ON LINUX

This document details the procedure for configuring Docker so that containers, including the spkg container, may be built and executed without requiring superuser privileges.

3.1 1. Install Docker Engine

There are two installation approaches. Select Option A for a simpler installation using the docker.io package, or Option B for the most current official release.

1.1 Option A – Using the docker.io Package

Update the package index and install docker.io:

```
sudo apt update
sudo apt install -y docker.io
```

1.2 Option B – Installing the Latest Official Version

The following steps remove any old installations and install the latest Docker components.

```
sudo apt remove docker docker-engine␣
↪docker.io containerd runc

sudo apt update
sudo apt install -y \
  ca-certificates \
  curl \
  gnupg

sudo install -m 0755 -d /etc/apt/keyrings

curl -fsSL https://download.docker.com/
↪linux/ubuntu/gpg | \
  sudo gpg --dearmor -o /etc/apt/keyrings/
↪docker.gpg

echo \
"deb [arch=$(dpkg --print-architecture)␣
↪signed-by=/etc/apt/keyrings/docker.gpg] \
```

(continues on next page)

(continued from previous page)

```
https://download.docker.com/linux/ubuntu
↪$(lsb_release -cs) stable" | \
sudo tee /etc/apt/sources.list.d/docker.
↪list > /dev/null

sudo apt update
sudo apt install -y docker-ce docker-ce-
↪cli containerd.io docker-compose-plugin
```

3.2 2. Verify Docker Operation

Ensure the Docker daemon is active and verify the installation:

```
sudo systemctl start docker
sudo systemctl enable docker
docker version
```

3.3 3. Configure User Access to Docker

Add your user account to the docker group to allow Docker execution without root privileges:

```
sudo usermod -aG docker $USER
```

After executing this command, re-authenticate by logging out and back in or by running:

```
newgrp docker
```

3.4 4. Validate Non-Privileged Docker Usage

Confirm that Docker commands work without using sudo:

```
docker ps
```

The command should return either an empty list or the column headers.

3.5 5. Verify Docker-Compose Plugin Installation

Check the installation of docker-compose, whether using the legacy version or the modern plugin:

```
docker-compose version     # For the␣
↪classic (v1) version
# or
docker compose version     # For the␣
↪modern (v2) plugin
```

If docker-compose is not available, install the required package:

```
sudo apt install docker-compose
```

Or for the modern plugin:

```
sudo apt install docker-compose-plugin
```

3.6 6. Running spkg Without sudo

Once the configuration is complete, you may build images and compile projects with the spkg tool without requiring sudo privileges.

```
./spkg/spkg compose        # To build the␣
↪Docker image
./spkg/spkg -p ./my_project bin    # To␣
↪compile your project
```

3.7 Optional: Verify Docker Socket Permissions

Ensure that the Docker socket is correctly configured with the proper group ownership and permissions:

```
ls -l /var/run/docker.sock
```

The expected output should resemble:

```
srw-rw---- 1 root docker ...
```

If the permissions are not as specified, adjust them with:

```
sudo chown root:docker /var/run/docker.sock
sudo chmod 660 /var/run/docker.sock
```

CHAPTER
FOUR

GENERAL INFORMATION

The **Multi-Protocol Programmer** is a compact and versatile development tool designed for high-precision embedded system applications. It supports a broad range of protocols and device architectures, including **AVR**, **ARM (CMSIS-DAP)**, and **CPLD (MAX II)**. Its USB connectivity enables direct interfacing with standard development environments, enabling:

- In-system programming (ISP)
- Step-through debugging
- Boundary-scan testing (JTAG)
- Flash memory operations

4.1 Supported Architectures

- **AVR** — via ISP (SPI configuration)
- **ARM Cortex-M** — via CMSIS-DAP and SWD
  - **RP2040**
  - **PY32**
  - **STM32**
- **CPLD/FPGA (MAX II)** — via JTAG

All protocols are exposed via labeled headers or JST connectors, allowing fast, solderless prototyping.

This device connects to a host system via USB and allows the user to program and debug various microcontrollers and programmable logic devices.

4.2 Supported Interfaces

- **JTAG**, for full-chip debugging and boundary scan
- **SWD**, for ARM Cortex-M series
- **SPI**, for flash and peripheral programming
- **UART**, for serial bootloaders and communication
- **GPIO**, for bit-banging or peripheral testing

Table 4.1: Interface and Signal Overview

Interface	Description	Signals / Pins	Typical Use
JTAG	Standard boundary-scan and debug interface	TCK, TMS, TDI, TDO, nTRST	Full chip programming, in-circuit test, debug
SPI	High-speed serial peripheral interface	MOSI, MISO, SCK, CS	Flash memory programming, peripheral data exchange
SWD	ARM's two-wire serial debug and programming interface	SWCLK, SWDIO	Cortex-M programming and step-through debugging
JST Header	Compact connector for power and single-wire debug signals	SWC (SWCLK), SWD (SWDIO), VCC, GND	Quick-connect to target board for SWD and power

4.3 Sections GPIO Pin Distribution

The Multi-Protocol Programmer features a set of GPIO pins that can be configured for various protocols, including JTAG, SWD, and ISP. These GPIOs are mapped to specific functions in the firmware, allowing users to adapt the programmer for different applications.

The GPIO pin distribution is defined within the CH552 firmware, supporting flexible assignment for various protocols. The firmware configures the specific mapping of GPIOs to protocols, such as SPI, JTAG, or SWD, based on the loaded configuration. Users can alter the pin distri-

bution by modifying the firmware source code to suit their application requirements.

4.3.1 Protocol ISP – In-System Programming

Compatible with **AVR** microcontrollers, this protocol allows programming and debugging via the SPI interface. The programmer can be used to flash firmware directly into the target device's memory.

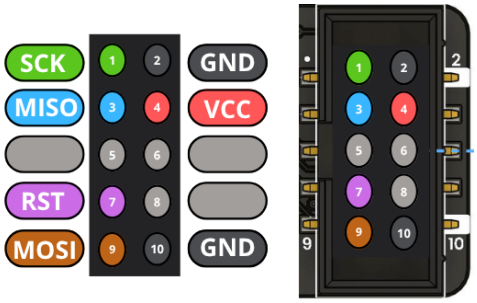


Fig. 4.1: Pinout diagram for CH552 Programmer

Table 4.2: Pinout

PIN	GPIO	I/O
MOSI	1.5	MOSI, PWM1
MISO	1.6	MISO, RXD1
CS	3.3	PWM1, TXD0
SCK	1.7	SCK, TXD1

4.4 Protocol JTAG

Compatible with **CPLD** and **FPGA** devices, this protocol allows programming and debugging via the JTAG interface. The programmer can be used to flash firmware directly into the target device's memory.

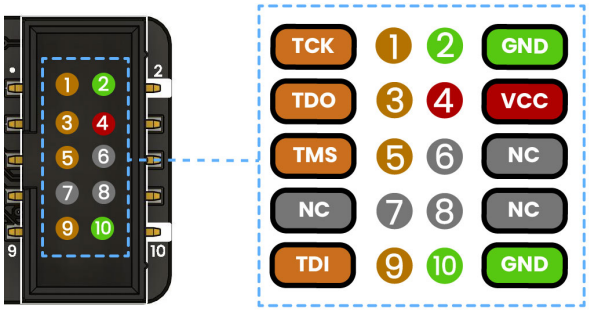


Fig. 4.2: Pinout diagram for CH552 Programmer (JTAG interface)

Table 4.3: Pinout

PIN	GPIO	I/O
TCK	1.7	SCK, TXD1
TMS	3.2	TXD1, INT0, VBUS1, AIN3
TDI	1.5	MOSI, PWM1, TIN3, UCC2, AIN2
TDO	1.6	MISO, RXD1, TIN4

Table 4.4: Pinout NC - Not Connected

PIN	GPIO	I/O
NC 6	3.4	PWM2, RXD1, T0
NC 7	3.3	INT1
NC 8	1.4	T2, CAP1, SCS, TIN2, UCC1, AIN1

4.5 Protocol SWD

Compatible with **ARM Cortex-M** microcontrollers, this protocol allows programming and debugging via the SWD interface. The programmer can be used to flash firmware directly into the target device's memory.

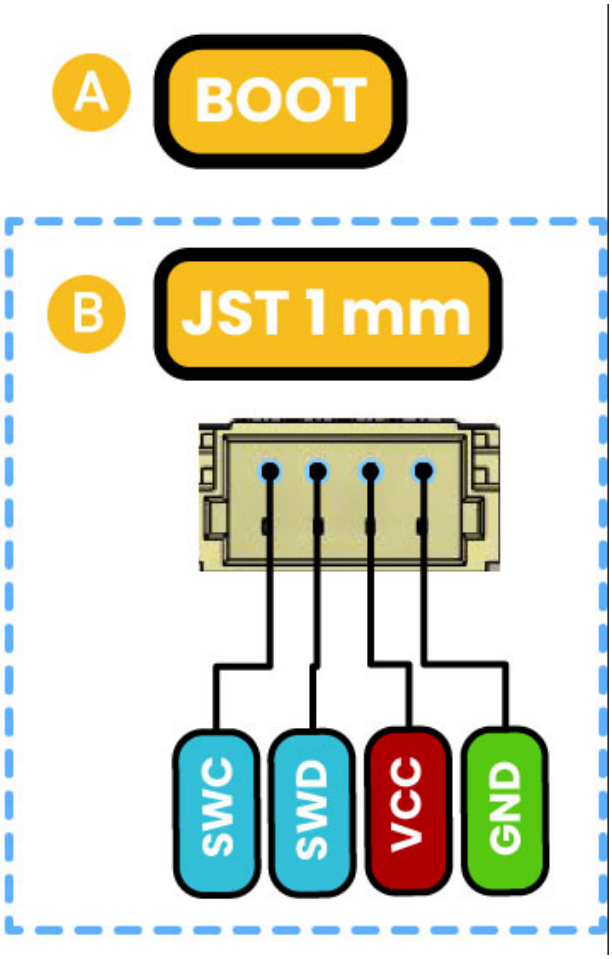


Fig. 4.3: SWD Pinout(JTAG interface)

Table 4.5: Pinout

PIN	GPIO	I/O
SWCLK	1.7	SCK, TXD1, TIN5
SWDIO	1.6	MISO, RXD1, TIN4

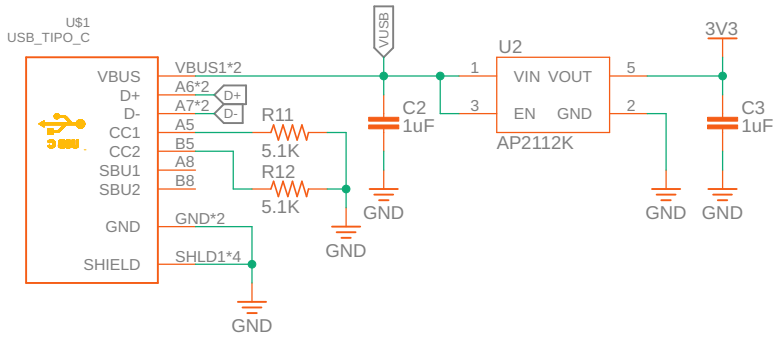
Note: GPIO numbers refer to the CH552 internal ports. Ensure correct firmware pin mapping before connecting external devices.

Table 4.6: Board Reference Table

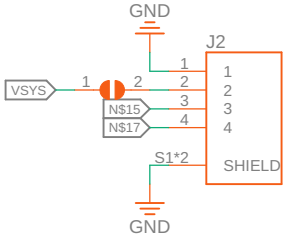
Ref.	Description
IC1	CH552 Microcontroller
U1	AP2112K 3.3V LDO Voltage Regulator
PB1	Boot Push Button
TP1	Reset Test Point
TP2	P3.1 Test Point
L1	Built-In LED
L2	Power On LED
SB1	Solder bridge to enable VCC at JTAG
SB2	Solder bridge to enable VCC at JST
J1	USB Type-C Connector
J2	Low-power I2C JST Connector
J3	JTAG Connector
JP1	Header for SWD or ICSP programming
JP2	Header to Select Operating Voltage Level

APPENDIX A: SCHEMATICS

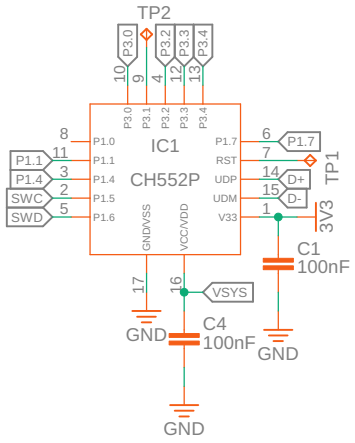
Voltage Regulator



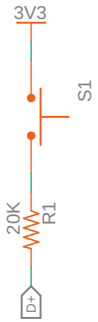
JST



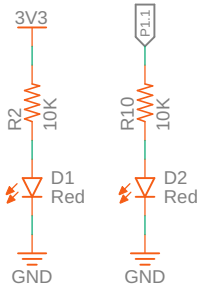
Microcontroller



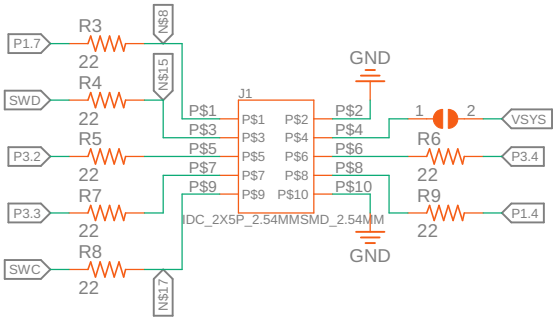
Boot



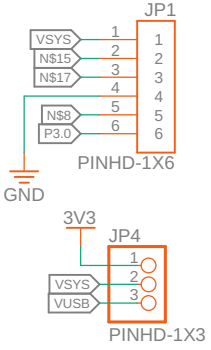
LEDS



IDC Connector



Headers



Mounting Holes

Title: CH552 USB Multi-Protocol Programmer		UNIT ELECTRONICS
SKU: UE0090	REV1.0	
Last date time: 02/05/2025 11:40 a. m.	SHEET: 1 of 1	
File:	Sheet description:	Author:Alberto Villanueva

CHAPTER
FIVE

AVR: GETTING STARTED

5.1 Introduction to AVR Microcontrollers

AVR (Advanced Virtual RISC) is a family of microcontrollers originally developed by Atmel, now part of Microchip Technology. Renowned for their simplicity, low power consumption, and ease of use, AVR microcontrollers are widely adopted in embedded systems, including Arduino boards and other DIY electronics projects.

The AVR family spans a variety of models with differing specifications—such as flash memory capacity, I/O pin count, and integrated peripherals. Common examples include the **ATmega** series (e.g., `ATmega328P`, `ATmega2560`) and the **ATtiny** series (e.g., `ATtiny85`, `ATtiny2313`).

5.2 Architecture Overview

AVR microcontrollers are based on a **modified Harvard architecture**, which separates instruction and data memory. This design allows for simultaneous access and contributes to faster instruction execution and efficient memory usage.

Developers typically write code for AVR devices using **AVR Assembly** or higher-level languages like **C/C++**, supported by environments such as **Atmel Studio** or the **Arduino IDE**.

5.3 Programming with the CH552 Multi-Protocol Programmer

Table 5.1: Pinout

PIN	GPIO	I/O
MOSI	1.5	MOSI, PWM1
MISO	1.6	MISO, RXD1
CS	3.3	PWM1, TXD0
SCK	1.7	SCK, TXD1

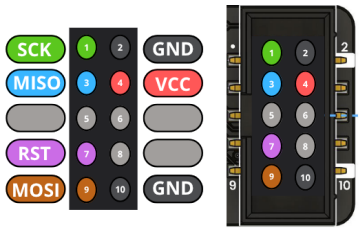


Fig. 5.1: Pinout diagram for CH552 Programmer

The **CH552 USB Multi-Protocol Programmer** provides robust support for AVR microcontrollers via the **In-System Programming (ISP)** interface. This method enables direct programming of the target microcontroller's **flash memory** and **EEPROM** without removing it from the circuit.

5.3.1 Key Advantages:

- **Non-intrusive**: Program the MCU without desoldering or removing it from the board.
- **Efficient workflow**: Ideal for development, testing, and field updates.
- **Wide compatibility**: Supports common AVR chips used in educational and commercial projects.

5.3.2 Supported Operations:

- Flash memory writing
- EEPROM access
- Fuse and lock bit configuration
- Signature verification

NOTES

CHAPTER
SIX

AVR FIRMWARE OVERVIEW

The CH552 Multi-Protocol Programmer relies on the PICO-AVR firmware for correct operation. This firmware, designed to run on CH55x microcontrollers, transforms the CH552 into a versatile USB-to-ISP bridge. It supports ISP AVR programming protocol, making it compatible with a wide range of AVR devices.

The programmer integrates seamlessly with development environments such as the Arduino IDE. It can be selected under Tools → Programmer as either USBasp or a custom driver name, allowing for easy bootloader installation and firmware updates on AVR microcontrollers like the ATmega328P.

Warning: The PICO-AVR firmware is available under the Creative Commons Attribution-ShareAlike 3.0 Unported License.

Additional resources:

- Github: wagiminator
- EasyEDA: wagiminator at EasyEDA
- License details: Creative Commons Attribution-ShareAlike 3.0 Unported License

6.1 Firmware Update Procedure

Note: The following procedure assumes that the *unit_ch55x_docker_sdk* repository is already cloned on your system. Ensure that **Docker Desktop** is running before executing the build commands, as they rely on Docker containers for compilation.

To commence the utilization of the **Multi-Protocol Programmer** in PICO ASP mode, execute the following procedures:

1. Navigate to the SDK Root Directory

```
cd unit_ch55x_docker_sdk
```

2. Compile the Firmware

On Linux

```
./spkg/spkg -p ./examples/usb/prog/avr
```

On Windows:

```
./spkg/spkg.bat -p ./examples/usb/prog/avr
```

The execution of this command will generate a .bin file within the **build** directory.

6.1.1 WCHIspStudio Interface

Launch **WCHIspStudio** to upload the firmware.

- In the Object File 1 field, select the path to the generated .bin firmware file.
- Enable the option "Automatic Download When Device Connect".
- Click the ... button to browse and confirm the correct firmware path.

Warning: Before connecting the CH552 programmer, **ensure the device is powered with +5V**. Use the onboard switch to select the appropriate voltage.

- Press the Boot button and connect your **Multi-Protocol Programmer** to the computer.
- Await the completion of the firmware update process.

Completion Notice: The firmware has been successfully updated, and the **UNIT CH552 Multi-Protocol Programmer** is now ready for use.

6.1.2 Resources

1. EasyEDA Design Files
2. WCH: CH552 Datasheet
3. SDCC Compiler
4. Blinkinlabs: CH55x SDK for SDCC
5. Thomas Fischl: USBasp
6. Ralph Doncaster: USBasp
7. Deqing Sun: CH55xduino

CHAPTER
SEVEN

AVR: COMPILE AND UPLOAD CODE

7.1 Toolchain Overview

To compile and upload code to an AVR microcontroller, you'll need the **AVR-GCC** toolchain. This includes essential components such as the compiler, assembler, linker, and utilities like `avr-objcopy`.

7.2 Installation

You can download and install the AVR-GCC toolchain from the official Microchip website:

Windows

Download the installer from the Microchip website and follow the installation instructions.

- AVR-GCC Compiler for Microchip Studio

Linux

You can install the AVR-GCC toolchain using your package manager. For example, on Ubuntu:

```
sudo apt-get install avr-gcc avr-binutils␣
↪avr-libc
```

macOS

You can use Homebrew to install the AVR-GCC toolchain:

```
brew tap osx-cross/avr
brew install avr-gcc
```

Ensure the toolchain is added to your system's `PATH` environment variable for global access.

7.3 Example: Compiling a Blink Program

This example demonstrates how to compile a basic blink program for two common AVR microcontrollers:

- **ATtiny88**
- **ATmega328P**

```
          +-----------------+
PC6 (RST)| 1  ATmega328 28 | PC5 (A5)
PD0 (RX) | 2            27 | PC4 (A4)
PD1 (TX) | 3            26 | PC3 (A3)
     PD2 | 4            25 | PC2 (A2)
PD3 (PWM)| 5            24 | PC1 (A1)
     PD4 | 6            23 | PC0 (A0)
    VCC  | 7            22 | GND
    GND  | 8            21 | AREF
     PB6 | 9            20 | AVCC
     PB7 |10            19 | PB5 (D13)
PD5 (D5) |11            18 | PB4 (D12)
PD6 (D6) |12            17 | PB3 (D11/PWM)
PD7 (D7) |13            16 | PB2 (D10/PWM)
PB0 (D8) |14            15 | PB1 (D9/PWM)
          +-----------------+
```

The program toggles an LED connected to **pin PB0** every second.

7.3.1 Source File

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>

int main(void) {
    DDRB |= (1 << PB0); // Set PB0 as␣
↪output
    while (1) {
        PORTB ^= (1 << PB0); // Toggle PB0
        _delay_ms(1000);
```

(continues on next page)

(continued from previous page)

```
    }
}
```

7.3.2 Compilation Commands

Use the following commands to compile and generate the HEX file:

```
# For ATtiny88
avr-gcc -mmcu=attiny88 -Os -o blink.elf␣
↪blink.c
avr-objcopy -O ihex blink.elf blink.hex

# For ATmega328P
avr-gcc -mmcu=atmega328p -DF_
↪CPU=16000000UL -Os -o blink.elf blink.c
avr-objcopy -O ihex blink.elf blink.hex
```

Explanation of flags:

- `-mmcu=` specifies the target microcontroller.
- `-Os` enables size optimization.
- `-DF_CPU` sets the clock frequency used for timing functions.

7.4 Uploading with AVRDUDE

Once the `.hex` file is generated, you can upload it to the AVR microcontroller using **AVRDUDE**.

7.5 Installation AVRDUDE Linux

You can install AVRDUDE on Linux using your package manager. For example, on Ubuntu:

```
sudo apt-get install avrdude
```

7.5.1 Upload Command

```
avrdude -p m328p -c usbasp -U␣
↪flash:w:blink.hex
```

Explanation:

- `-p m328p` specifies the target device (ATmega328P).
- `-c usbasp` sets the programmer to the Multi-Protocol Programmer.
- `-U flash:w:blink.hex` uploads the hex file to flash memory.

Replace `m328p` with the appropriate identifier for your specific AVR device (e.g., `t88` for ATtiny88). A full list of supported devices is available in the AVRDUDE user manual.

CHAPTER
EIGHT

AVR: ARDUINO IDE BOOTLOADER

8.1 Installing the Bootloader on ATMEGA328P

This guide explains how to flash the Arduino-compatible bootloader onto an **ATMEGA328** microcontroller using the **UNIT USB Multi-Protocol Programmer**. By following these steps, your ATMEGA328 can function as an **ATMEGA328P**, fully compatible with the Arduino IDE.

8.2 Required Materials

1. **Multi-Protocol Programmer**
2. **ATMEGA328P Microcontroller**

8.3 Hardware Connection

1. Use the FC cable to connect one end to the UNIT Multi-Protocol Programmer and the other to the ICSP interface of the ATMEGA328P.
2. Make sure the **MISO** pin of the programmer is aligned with **pin 1** of the ICSP header on the ATMEGA328P.

8.4 Driver Setup with Zadig

To allow USB communication between your PC and the programmer, install the required drivers using **Zadig**.

8.4.1 Step 1: Identify the COM Port

Connect the programmer to your PC and open the **Device Manager**. Under **Ports (COM & LPT)**, identify the COM port assigned to the device.

8.4.2 Step 2: Open Zadig

Launch Zadig. You should see a window like the following:

Click **Options** → **List All Devices** to display all USB interfaces:

8.4.3 Step 3: Install Drivers

Install the following two drivers:

- **picoASP Interface 0**: Install the **libusbK** driver by clicking **Replace Driver**.
- **SerialUPDI Interface 1**: Install the **USB Serial (CDC)** driver by clicking **Upgrade Driver**.

Once both drivers are installed, your UNIT Multi-Protocol Programmer is ready to flash the bootloader.

8.5 Bootloader Installation Using Arduino IDE

Open the **Arduino IDE** and follow these steps to burn the bootloader onto your ATMEGA328P.

1. **Select the Target Board**

   Navigate to **Tools** → **Board** and choose **ATMEGA328P**.

2. **Choose the Correct Port**

   Under **Tools** → **Port**, select the COM port corresponding to your programmer.

3. **Select the Programmer**

   Under **Tools** → **Programmer**, select the programmer (e.g., **USBasp** or your custom driver name).

4. **Burn the Bootloader**

   Finally, go to **Tools** → **Burn Bootloader**.

Success! Your ATMEGA328P now has a compatible Arduino bootloader installed and is ready for development.

CHAPTER
NINE

ARM CORTEX-M

ARM Cortex-M microcontrollers—such as those found in the STM32, RP2040, PY32, and similar families—are built on ARMv6-M, ARMv7-M, or ARMv8-M architectures. These cores target low-power, high-performance embedded applications.

9.1 Core Families

- Cortex-M0 / M0+ (ARMv6-M): Ultra-low power, suitable for simple tasks.
- Cortex-M3 / M4 / M7 (ARMv7-M): Higher performance; some models (like M4 and M7) support floating-point operations.
- Cortex-M23 / M33 (ARMv8-M): Introduce enhanced security features, such as TrustZone.

9.2 Additional Notes

- The RP2040 uses dual Cortex-M0+ cores (ARMv6-M).
- The PY32 series (e.g., PY32F003) typically uses a Cortex-M0+ core (ARMv6-M).
- The STM32 family spans a wide range—from Cortex-M0 to M7 and even M33 in newer models.

9.2.1 OpenOCD (Open On-Chip Debugger)

OpenOCD is an open-source software tool that facilitates debugging, in-system programming, and boundary-scan testing of embedded devices. It supports a diverse range of microcontrollers including STM32, RP2040, and PY32. Through interfaces such as JTAG or SWD, OpenOCD provides a command-line interface for effective interaction with the target devices.

9.2.2 PyOCD: A Python Interface to OpenOCD

PyOCD simplifies the use of OpenOCD by providing a Python-based high-level interface. It offers a Python API that abstracts the complexities of the OpenOCD command-line interface, making it easier to program and debug microcontrollers. PyOCD is user-friendly, flexible, and designed to be extensible for various microcontroller interactions.

Tool	Functionality
OpenOCD	Provides a command-line interface for debugging and programming microcontrollers.
PyOCD	Offers a Python API for seamless interaction with microcontrollers via OpenOCD.

9.3 ARM Cortex-M Debug Capabilities

9.3.1 Debugging Interface of ARM Cortex-M

The ARM Cortex-M architecture includes a comprehensive debug interface that allows external tools to access the microcontroller core for debugging and programming tasks. This interface includes a suite of registers and commands that empower debuggers to halt the core, access memory, and manage the execution of programs.

9.3.2 Connection Through Debug Pins

Access to the debug interface is facilitated through specific pins on the microcontroller, such as SWDIO, SWCLK, and nRESET. These are linked to a debug adapter like the ST-Link or CMSIS-DAP, which forms the physical bridge between the host computer and the target device.

9.4 CMSIS-DAP: A Standardized Debug Adapter

9.4.1 Overview of CMSIS-DAP

CMSIS-DAP (Cortex Microcontroller Software Interface Standard - Debug Access Port) serves as a standardized debug adapter for ARM Cortex-M devices. It is endorsed by OpenOCD and provides a cost-effective solution compared to proprietary debug adapters.

9.5 SWD Communication Protocols

9.5.1 Signals and Operations

The SWDIO (Serial Wire Debug Input/Output) and SWCLK (Serial Wire Clock) signals are integral for communication over the SWD interface. SWDIO is a bidirectional line used for transmitting debug commands and data, while SWCLK is a clock signal that ensures synchronized data transfer between the host and the target.

9.5.2 Advantages of SWD Over JTAG

Utilizing a two-wire connection, the SWD interface minimizes the pin count required compared to the traditional JTAG interface. This reduction is particularly beneficial for microcontrollers where pin availability is limited.

CHAPTER
TEN

USING OPENOCD

To program your microcontroller using OpenOCD, use the following command:

```
openocd -f interface/cmsis-dap.cfg -f␣
↪target/rp2040.cfg \
        -c "init" -c "reset init" \
        -c "flash write_image erase blink.
↪bin 0x10000000" \
        -c "reset run" -c "shutdown"
```

10.1 Supported Microcontrollers

This guide supports the following microcontrollers:

Micro-controller	Description
RP2040	Low-cost microcontroller with a dual-core ARM Cortex-M0+ processor.
STM32F1	Budget-friendly microcontroller with an ARM Cortex-M3 processor.
STM32F4	Cost-effective microcontroller with an ARM Cortex-M4 processor.

10.1.1 Selecting Your Microcontroller Target

Each microcontroller requires a specific configuration file. This file is a script that instructs OpenOCD on how to communicate with the microcontroller. The configuration file is tailored to the microcontroller and the debugger interface.

Microcon-troller	Configuration File
RP2040	`rp2040.cfg`
STM32F103C8	`stm32f103c8t6.cfg`
STM32F411	`stm32f411.cfg`

CHAPTER
ELEVEN

PYOCD

PyOCD is a command-line tool for interacting with ARM Cortex-M microcontrollers. It supports flashing, erasing, debugging, and low-level memory access using CMSIS-DAP, DAPLink, ST-Link, and other probes.

11.1 Basic Syntax

```
pyocd <command> [OPTIONS]
```

You can specify the target using `-t <target>` or through a configuration file using `--config <file.yaml>`.

11.2 Configuration File

Use a YAML configuration file to define global options:

```yaml
# Misc/pyocd.yaml
target_override: py32f003x6
frequency: 1000000
log_level: info
```

To load this config:

```
pyocd erase --config ./Misc/pyocd.yaml
```

11.3 Commands

Erase flash memory of the target device.

```
pyocd erase -t py32f003x6 [OPTIONS]
```

Options:

- `--chip`: Erase the entire flash memory.
- `--sector <ADDR>`: Erase a single sector by address.
- `--config <file.yaml>`: Load configuration from YAML file.

Example:

```
pyocd erase -t py32f003x6 --chip --config .
↪/Misc/pyocd.yaml
```

Flash a binary, hex, or ELF file to the target.

```
pyocd flash <file> -t py32f003x6 [OPTIONS]
```

Options:

- `--base-address <addr>`: Override the base address (for .bin files).
- `--verify`: Verify flash contents after writing.
- `--erase=chip|sector|auto`: Select erase mode.
- `--format bin|hex|elf`: Force file format.
- `--config <file.yaml>`: Load YAML config.

Example:

```
pyocd flash firmware.hex -t py32f003x6 --
↪erase=chip --verify
```

Start a GDB server for remote debugging.

```
pyocd gdbserver -t py32f003x6 [OPTIONS]
```

Options:

- `--port <number>`: GDB server port.
- `--telnet-port <number>`: Telnet monitor port.
- `--persist`: Keep the server alive after GDB disconnects.
- `--config <file.yaml>`: Load YAML config.

Reset the target microcontroller.

```
pyocd reset -t py32f003x6 [OPTIONS]
```

Options:

- `--halt`: Halt the CPU after reset.
- `--config <file.yaml>`: Load YAML config.

List connected debug probes and supported targets.

```
pyocd list
```

Read and write memory directly.

```
pyocd read32 0x08000000 4
pyocd write32 0x20000000 0x12345678
```

11.4 CMSIS-Pack Targets

To add custom target support (e.g., `py32f003x6`), use:

```
pyocd pack install py32f003x6
```

Or add a local *.pdsc* file:

```
pyocd pack add ./path/to/PY32F003.pdsc
```

11.5 Example Workflow

```
pyocd list
pyocd erase -t py32f003x6 --chip
pyocd flash build/main.hex -t py32f003x6 --
↪verify
pyocd gdbserver -t py32f003x6
pyocd reset -t py32f003x6 --halt
```

11.6 References

- https://pyocd.io/
- https://github.com/pyocd/pyocd

11.7 Command Help

```
pyocd --help
pyocd flash --help
pyocd erase --help
```

CHAPTER
TWELVE

RP2040: AN INTRODUCTION

The RP2040 is an efficient and cost-effective microcontroller developed by Raspberry Pi. It features a dual-core ARM Cortex-M0+ processor, flexible I/O capabilities, and a wide range of peripherals, making it suitable for both hobbyist and professional applications.

Note: This documentation is continuously updated and incorporates key concepts from the official Raspberry Pi Pico SDK documentation.

Tip: A Linux operating system is recommended, as the Pico-SDK is primarily developed and tested on Linux. However, it can also be configured for Windows and macOS with additional steps.

This guide provides a comprehensive introduction to working with RP2040 microcontrollers. It covers setting up the development environment, installing the required software, and writing code for RP2040-based projects.

12.1 Multi-Protocol Programmer for RP2040

The Multi-Protocol Programmer is a versatile tool for programming and debugging RP2040 microcontrollers. It enables straightforward and reliable programming using the SWD (Serial Wire Debug) interface.

12.2 Programming RP2040 with the Multi-Protocol Programmer

The Multi-Protocol Programmer supports programming of RP2040 microcontrollers via the SWD interface. This interface allows direct access to the target microcontroller's flash memory and supports debugging functionality. The programmer is compatible with RP2040f10x and RP2040f4xx series.

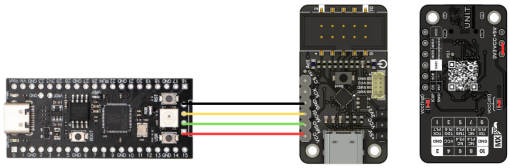


Fig. 12.1: Pinout diagram for RP2040

12.3 DualMCU RP2040 Programming with Multi-Protocol Programmer

The Multi-Protocol Programmer uses the SWD interface to program RP2040 microcontrollers. Follow these steps to program your RP2040 device:

1. **Connect the Multi-Protocol Programmer** to your computer via USB.
2. **Open Visual Studio Code** or another editor of your preference.
3. **Install the required extensions** for RP2040 development, such as "C/C++" and "CMake Tools".
4. **Create a new project** or open an existing one.
5. **Write your code** in C or C++ using the RP2040 SDK.
6. **Configure the build system** to use the RP2040 SDK and integrate the CH552 Multi-Protocol Programmer.
7. **Build the project** to generate the binary firmware file.

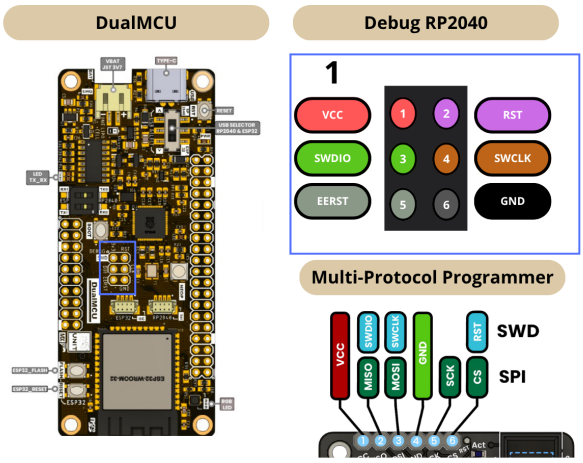


Fig. 12.2: DualMCU RP2040 Connection

CHAPTER
THIRTEEN

RP2040 FIRMWARE

13.1 Firmware update

Note: The contents of this chapter are hosted in the unit_ch55x_docker_sdk.git repository. Always remember to keep Docker Desktop open in your computer; otherwise, the commands will not work properly.

To start using your **Multi-Protocol Programmer** as a PICO DAP, follow these steps:

```
cd unit_ch55x_docker_sdk
```

Once you are at the main directory, open a terminal and enter the commands:

13.1.1 Linux

```
./spkg/spkg -p ./examples/USB/Prog/PICO-DAP
```

13.1.2 Windows

```
./spkg/spkg.bat -p ./examples/USB/Prog/
↪PICO-DAP
```

The command creates a .bin file inside the **build** folder.

13.1.3 WCHIspStudio

Then open **WCHIspStudio** to upload the firmware

- In Object File 1 make sure to enter the correct path to the directory with the firmware.
- Make sure the "Automatic Download When Device Connect" option is enabled.
- To add the path you have to click on this bottom "..." and check.

Warning: Before connecting your Multi-Protocol Programmer, make sure to power it with +5V. You can do this by setting your switch to +5V.

- Push the Boot bottom and connect your **Multi-Protocol Programmer** to your computer.
- Wait until de firmware has finished updating your device.

Done! Now you can use your UNIT CH552 Multi-Protocol Programmer!

13.2 Create a new project in Pico SDK

13.2.1 Installation

Open Visual Studio Code, in the extensions menu, search Raspeberry Pi Pico:

And install the official version from Raspberry:

In the Activity Bar, you will find the icon of your new extension in Visual Studio Code.

In the general menu, click on "New C/C++ Project"

Once the project is created, you will need to configure it. For this example, we will use a Raspberry Pico, UART, SPI and Console over UART and USB

13.2.2 Project

Inside the generated project, you will find these files.

Open the .c file, here we can change or modify the source code.

13.2.3 Examples

Here are some examples for a Raspeberry Pi Pico:

1. Hello, World! Open a serial monitor and see what's happening!

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/uart.h"

// UART defines
// By default the stdout UART is uart0, so we will use the second one
#define UART_ID uart1
#define BAUD_RATE 115200

// Use pins 4 and 5 for UART1
// Pins can be changed, see the GPIO function select table in the datasheet for information on GPIO assignments
#define UART_TX_PIN 4
#define UART_RX_PIN 5



int main()
{
    stdio_init_all();

    // Set up our UART
    uart_init(UART_ID, BAUD_RATE);
    // Set the TX and RX pins by using the function select on the GPIO
    // Set datasheet for more information on function select
    gpio_set_function(UART_TX_PIN, GPIO_FUNC_UART);
    gpio_set_function(UART_RX_PIN, GPIO_FUNC_UART);

    // Use some the various UART functions to send out data
    // In a default system, printf will also output via the default UART

    // Send out a string, with CR/LF conversions
    uart_puts(UART_ID, " Hello, UART!\n");

    // For more examples of UART use see https://github.com/raspberrypi/pico-examples/tree/master/uart

```

(continues on next page)

(continued from previous page)

```
    while (true) {
        printf("Hello, World!\n");
        sleep_ms(1000);
    }
}
```

2. Blink

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/uart.h"

// UART configuration
#define UART_ID uart1
#define BAUD_RATE 115200
#define UART_TX_PIN 4
#define UART_RX_PIN 5

// On-board LED pin (GPIO 25)
#define LED_PIN 25

int main()
{
    // Initialize standard I/O (required for printf to work)
    stdio_init_all();

    // Initialize UART1 with the specified baud rate
    uart_init(UART_ID, BAUD_RATE);

    // Configure UART TX and RX GPIO functions
    gpio_set_function(UART_TX_PIN, GPIO_FUNC_UART);
    gpio_set_function(UART_RX_PIN, GPIO_FUNC_UART);

    // Initialize GPIO 25 for LED and set it as output
    gpio_init(LED_PIN);
    gpio_set_dir(LED_PIN, GPIO_OUT);

    // Send a welcome message through UART1
    uart_puts(UART_ID, " Hello, UART!\n");

    while (true) {
        // Turn on the LED
        gpio_put(LED_PIN, 1);
        uart_puts(UART_ID, "LED ON\n");  // Send status via UART
        sleep_ms(500);                   //
```

(continues on next page)

(continued from previous page)

```
↪ Wait 500 milliseconds

        // Turn off the LED
        gpio_put(LED_PIN, 0);
        uart_puts(UART_ID, "LED OFF\n"); //
↪ Send status via UART
        sleep_ms(500);                   //
↪ Wait another 500 milliseconds
    }
}
```

13.2.4 Flashing

- Once you have your example ready to upload, just click on the Pico SDK icon on the Activity Bar.
- Select the "Flash Project (SWD)" option.

Note: To program a Raspberry Pico, use the SWD protocol. For more information, check the pinout.

Warning: The Raspberry Pi Pico operates at 3.3V. Switch to 3.3V before connecting your device.

CHAPTER
FOURTEEN

STM32: GETTING STARTED

STMicontrollers is a family of 32-bit microcontrollers designed and manufactured by STMicroelectronics. They are widely used in embedded systems and IoT applications due to their low power consumption, high performance, and rich peripheral set. STM32 microcontrollers are based on the ARM Cortex-M architecture and are available in a wide range of series, each tailored to specific applications and requirements.

14.1 Getting Started with STM32

This documentation provides a comprehensive guide to getting started with STM32 microcontrollers. It includes information on setting up the development environment, installing the necessary software, and writing code for STM32 microcontrollers.

This technical documentation is a work in progress and is an adaptation of documentation from the STM32CubeIDE and STM32CubeMX software tools. It uses the ARM-GCC compiler and visual studio code as the IDE.

CH552 Multi-Protocol Programmer is a versatile tool that supports programming and debugging of STM32 microcontrollers. It provides a simple and efficient way to program STM32 devices using the SWD (Serial Wire Debug) interface.

14.2 Programming STM32 with CH552 Multi-Protocol Programmer

Currently, the CH552 Multi-Protocol Programmer supports programming STM32 microcontrollers using the SWD interface. This allows for direct programming of the target microcontroller's flash memory and debugging capabilities STM32f10x series and STM32f4xx series.

CHAPTER
FIFTEEN

STM32 FIRMWARE

15.1 Firmware update

Note: The contents of this chapter are hosted in the unit_ch55x_docker_sdk.git repository. Always remember to keep Docker Desktop open in your computer; otherwise, the commands will not work properly.

To start using your **Multi-Protocol Programmer** as a PICO DAP, follow these steps:

```
cd unit_ch55x_docker_sdk
```

Once you are at the main directory, open a terminal and enter the commands:

15.1.1 Linux

```
./spkg/spkg -p ./examples/USB/Prog/PICO-DAP
```

15.1.2 Windows

```
./spkg/spkg.bat -p ./examples/USB/Prog/
↪PICO-DAP
```

The command creates a .bin file inside the **build** folder.

15.1.3 WCHIspStudio

Then open **WCHIspStudio** to upload the firmware

- In Object File 1 make sure to enter the correct path to the directory with the firmware.
- Make sure the "Automatic Download When Device Connect" option is enabled.
- To add the path you have to click on this bottom "..." and check.

Warning: Before connecting your Multi-Protocol Programmer, make sure to power it with +5V. You can do this by setting your switch to +5V.

- Push the Boot bottom and connect your **Multi-Protocol Programmer** to your computer.
- Wait until de firmware has finished updating your device.

Done! Now you can use your UNIT CH552 Multi-Protocol Programmer!

15.2 Create a new project in PlatformIO

15.2.1 Installation

Open Visual Studio Code, in the extensions menu, search PlatformIO:

And install the official version from PlatformIO:

In the Activity Bar, you will find the icon of your new extension in Visual Studio Code.

In the general menu, click on "Create New Project"

In the Quick Access menu, we can open an existing file for our STM32F4xx or create a new one:

Next, in the Project Wizard:

- We will name our file.
- Select the specific model of our STM32F4xx board.
- Choose the framework (for this example, we will use CMSIS)
- Specify the location where we want to save the project.

15.2.2 Project

Inside the generated project, you will find this project structure.

If you need to change your COM port, follow these steps:

Inside the .ini file, make sure you have this configuration:

```
[env:blackpill_f411ce]
platform = ststm32
board = blackpill_f411ce
framework = cmsis
upload_protocol = cmsis-dap
debug_tool = cmsis-dap
```

Note: If you are using a different board, change the "[env: ...]" and "board = ..." to match your board model.

Here is a list of common STM32 microcontrollers and their corresponding **board** identifiers used in PlatformIO's **platformio.ini** file.

Table 15.1: STM32 Microcontrollers

Microcontroller / Board	PlatformIO board name
STM32F103C8 (Blue Pill)	`bluepill_f103c8`
STM32F401RE (Nucleo)	`nucleo_f401re`
STM32F446RE (Nucleo)	`nucleo_f446re`
STM32F103RB	`genericSTM32F103RB`
STM32F407VG (Discovery)	`disco_f407vg`
STM32F411CEU6 (Blackpill)	`blackpill_f411ce`

15.2.3 Example

The following example is a Blink using GPIOC (PC13) as an output.

```
#include "stm32f4xx.h"

// Simple software delay loop
void delay(volatile uint32_t t) {
    while (t--);
}

int main(void) {
    // 1. Enable the clock for GPIOC␣
↪(required before accessing GPIOC␣
↪registers)
```

(continues on next page)

(continued from previous page)

```
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOCEN;

    // 2. Configure pin PC13 as general-
↪purpose output (MODER13 = 0b01)
    GPIOC->MODER &= ~(0x3 << (13 * 2)); //␣
↪Clear MODER13 bits
    GPIOC->MODER |=  (0x1 << (13 * 2)); //␣
↪Set MODER13 to output mode

    // 3. Main loop
    while (1) {
        GPIOC->ODR ^= (1 << 13); // Toggle␣
↪PC13 (invert LED state)
        delay(500000);           // Simple␣
↪delay (not accurate, for testing␣
↪purposes)
    }
}
```

15.2.4 Flashing

- Once you have your example ready to upload, just click on the PlatformIO icon on the Activity Bar.
- Select the "Upload" option.

Note: To program a STM32F4xx, use the SWD protocol. For more information, check the pinout.

Warning: The STM32F4xx operates at 3.3V. Switch to 3.3V before connecting your device.

CHAPTER
SIXTEEN

CPLD/FPGA

This firmware is not an official programmer from Altera or Intel. Instead, it is a JTAG programmer for CPLD/FPGA devices based on the WCH CH552 USB microcontroller. It supports the Intel (formerly Altera) MAX II series and is compatible with Quartus Prime Lite.

Warning: This firmware is a third-party implementation and **is not affiliated with or endorsed by Altera or Intel**. It is intended for educational, development, prototyping, and other lawful purposes only.

Ensure that you comply with all applicable laws and regulations when using this firmware. **Use is at your own risk.** The user assumes all responsibility for any consequences, including potential legal implications. The author and distributor of this firmware **are not liable** for any damage, misuse, or legal issues arising from its use.

Proceed with caution and discretion.

For programming Altera FPGA and CPLD devices, the Quartus software is used. A free version, Quartus Lite, is available. Quartus is the suite from Altera (now Intel) for programming, configuring, and using its products, and it fully supports the USB Blaster programmer and all its features.

16.1 Quick installation

Note: To download the Intel Quartus Prime software, visit Quartus Prime Installer.

1. **Open the file .exe**
    a. In the 24.1 version, add your devices
    b. Check the option "Agree to Intel License Agreement"
    c. Click on "Download and Install"
    d. Wait until the installation has completed succesfully

16.2 Create a new project in Quartus

Once installed the software and the driver for your CH552, we can create a new project. It's necessary that you've conected your device to your computer.

In the upper left, you will find the File > New Project Wizard tab.

a. You will see the New Project Wizard window
a. Select the option "Next".
b. Choose an appropriate name to start the project.
c. Select the folder where you will save your project.
d. In the project type, select the option empty project as shown above:
e. Afterwards, click Next.
f. For this example, it is not necessary to add additional files.
g. Afterwards, click Next.
h. This is the where you can select your device, package, pin count, core speed grade.
i. EDA Tools Settings.

Note: These options do not define the device family (e.g., MAX II, Cyclone IV, etc.); rather, they specify how Quartus will connect to third-party tools (ModelSim, Synopsys, etc.). If you do not have any external EDA tools, you may leave all options set to <None>, and Quartus will use its internal functionality to compile and generate programming files.

j. In this window, you will be able to see a summary of your settings.

k. Afterwards, click Finish.

CHAPTER
SEVENTEEN

CPLD FIRMWARE

17.1 Firmware update

Warning: This firmware is a third-party implementation and **is not affiliated with or endorsed by Altera or Intel**. It is intended for educational, development, prototyping, and other lawful purposes only.

Ensure that you comply with all applicable laws and regulations when using this firmware. **Use is at your own risk.** The user assumes all responsibility for any consequences, including potential legal implications. The author and distributor of this firmware **are not liable** for any damage, misuse, or legal issues arising from its use.

Proceed with caution and discretion.

Note: The contents of this chapter are hosted in the unit_ch55x_docker_sdk.git repository. Always remember to keep Docker Desktop open in your computer; otherwise, the commands will not work properly.

To start using your **Multi-Protocol Programmer** as a CPLD/FPGA programmer, follow these steps:

```
cd examples/USB/Prog/CPLD/src
```

In that directory, open **descriptor.h** inside that file, change lines 13 and 14 of the code as shown below:

Only change the **zeros** for **ones**

Once you've changed this data, run the following command in your terminal:

17.1.1 Linux

```
./spkg/spkg -p ./examples/USB/Prog/CPLD
```

17.1.2 Windows

```
./spkg/spkg.bat -p ./examples/USB/Prog/CPLD
```

The command creates a .bin file inside the **build** folder. Then open **WCHIspStudio** to upload the firmware

- In Object File 1 make sure to enter the correct path to the directory with the firmware.
- Make sure the "Automatic Download When Device Connect" option is enabled.
- To add the path you have to click on this bottom "…" and check.

Note: Before connecting your Multi-Protocol Programmer, make sure to power it with +5V. You can do this by setting your switch to +5V.

- Push the Boot bottom and connect your **Multi-Protocol Programmer** to your computer.
- Wait until de firmware has finished updating your device.

Done! Now you can use your UNIT CH552 Multi-Protocol Programmer!

Note: To program an FPGA and a CPLD, use the JTAG Protocol. For more information, check the pinout.

CHAPTER
EIGHTEEN

HOW TO GENERATE AN ERROR REPORT

This guide explains how to generate an error report using GitHub repositories.

18.1 Steps to Create an Error Report

1. **Access the GitHub Repository**
   Navigate to the GitHub repository where the project is hosted.
2. **Open the Issues Tab**
   Click on the "Issues" tab located in the repository menu.
3. **Create a New Issue**
   - Click the "New Issue" button.
   - Provide a clear and concise title for the issue.
   - Add a detailed description, including relevant information such as:
     - Steps to reproduce the error.
     - Expected and actual results.
     - Any related logs, screenshots, or files.
4. **Submit the Issue**
   Once the form is complete, click the "Submit" button.

18.2 Review and Follow-Up

The development team or maintainers will review the issue and take appropriate action to address it.