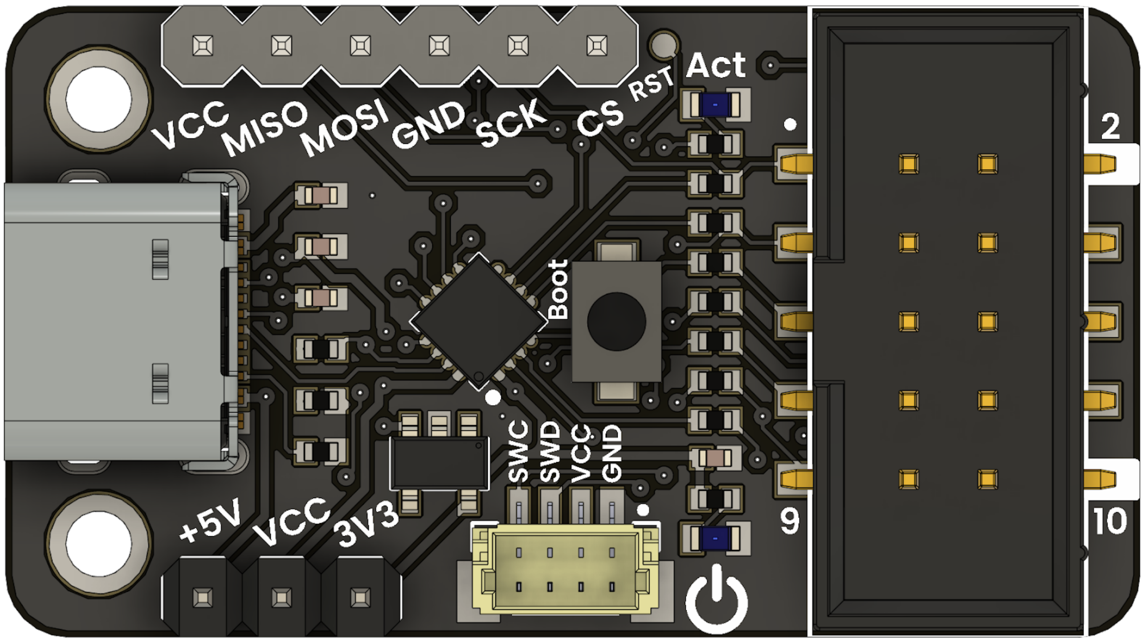


CH552 USB Multi-Protocol Programmer User Guide and Technical Reference

Release 0.0.1

Department of Research, Innovation, and Development

Jun 03, 2025

Contents

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

CH552 USB Multi-Protocol Programmer

The **CH552 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)

- Fully 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
  - urJTAG, openFPGALoader
- 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

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.

GENERAL INFORMATION

The **CH552 USB 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

Compatible with popular platforms such as **STM32**, **RP2040**, and **PY32**, this programmer integrates configurable GPIO lines, multi-protocol headers, and streamlined power delivery—making it ideal for rapid prototyping and field diagnostics.

2.1 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.

2.2 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 2.1: 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

2.3 Sections GPIO Pin Distribution

The CH552 USB 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 distribution by modifying the firmware source code to suit their application requirements.

2.3.1 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.

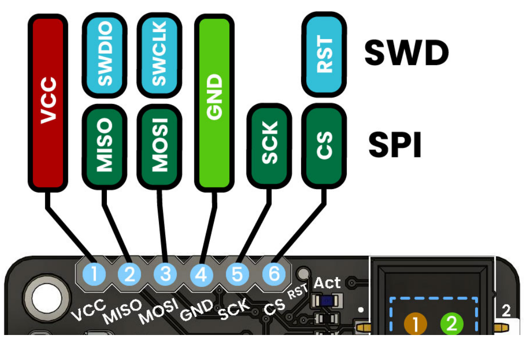


Table 2.2: Pinout

PIN	GPIO	I/O
MOSI	1.5	I/O
MISO	1.6	I/O
CS	3.0	I/O
SCK	1.7	I/O

2.4 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.

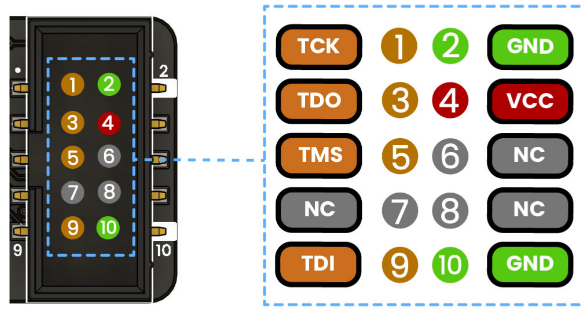


Fig. 2.1: Pinout diagram for CH552 Programmer (JTAG interface)

Table 2.3: Pinout

PIN	GPIO	I/O
TCK	1.7	I/O
TMS	3.2	I/O
TDI	1.5	I/O
TDO	1.6	I/O

Table 2.4: Pinout NC - Not Connected

PIN	GPIO	I/O
NC 6	3.4	I/O
NC 7	3.3	I/O
NC 8	1.4	I/O

2.5 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.

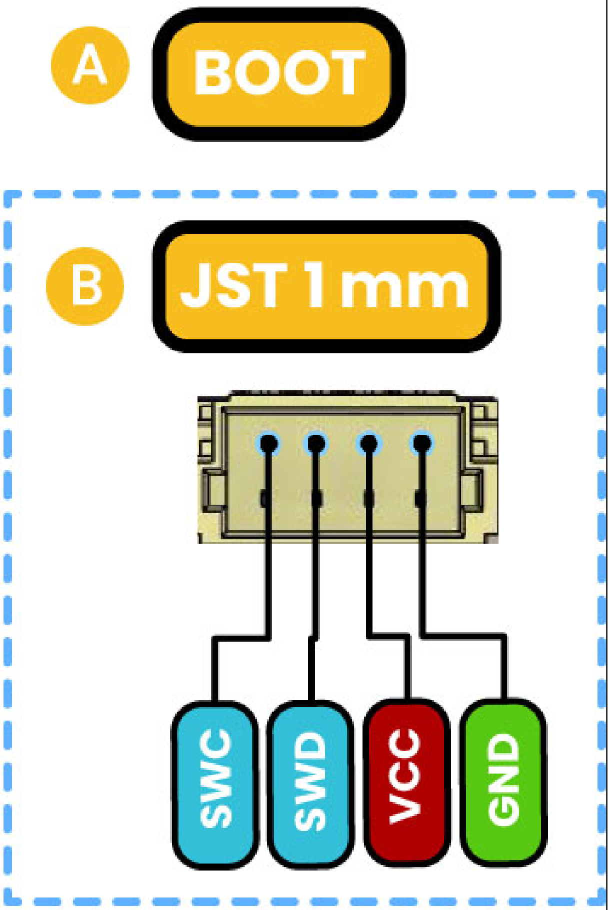


Fig. 2.2: SWD Pinout(JTAG interface)

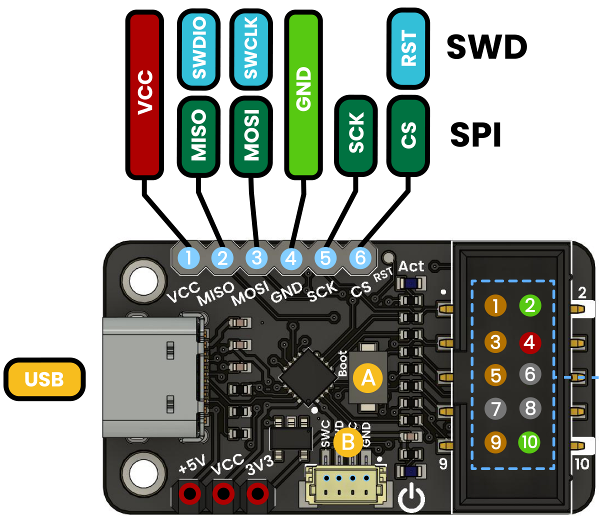
Table 2.5: Pinout

PIN	GPIO	I/O
SWCLK	1.7	I/O
SWDIO	1.6	I/O

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

Multi-Protocol Programmer

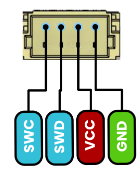
Top view



TCK	1	2	GND
TDO	3	4	VCC
TMS	5	6	NC
NC	7	8	NC
TDI	9	10	GND

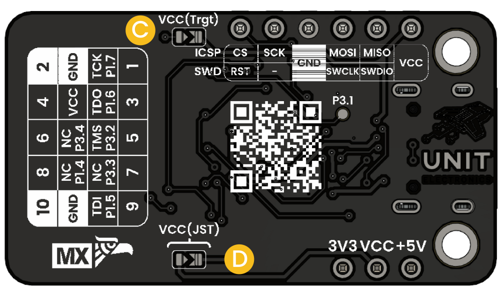
A BOOT

B JST 1 mm



VCC depends on the supply voltage of the microconroller (3V3 or 5V) to be programmed

Bottom view



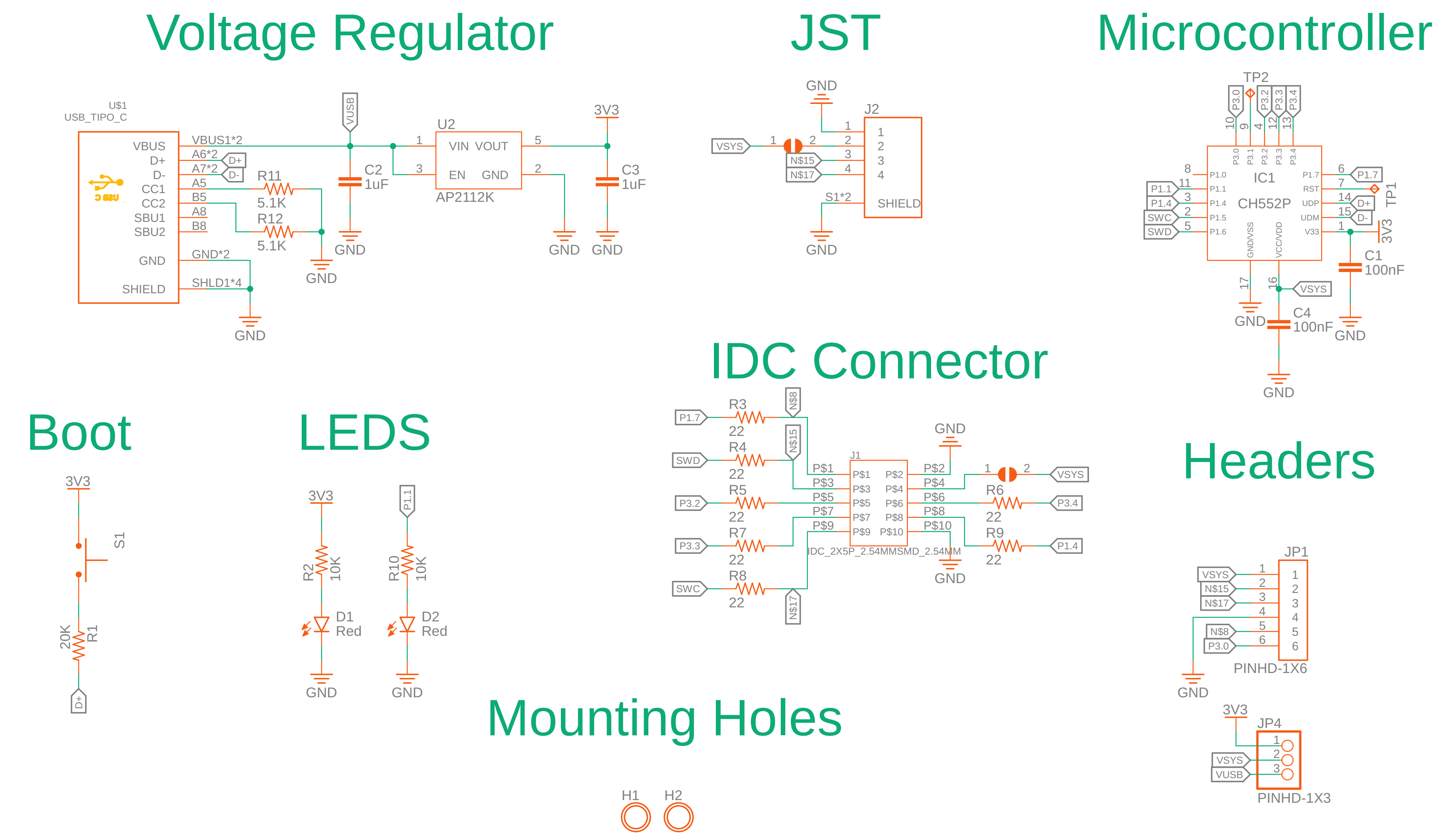
C TRGT

D JST

Description:

- Supply voltage
- GND
- Components
- No conection
- SPI
- JTAG
- SWD

APPENDIX A: SCHEMATICS



Title: CH552 USB Multi-Protocol Programmer

SKU: UE0090 | REV1.0

Last date time: 02/05/2025 11:40 a. m. | SHEET: 1 of 1

File: | Sheet description:

UNIT ELECTRONICS

Author:Alberto Villanueva

CHAPTER THREE

AVR GUIDE

3.1 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`).

3.2 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**.

3.3 Programming with the CH552 Multi-Protocol 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.

3.3.1 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.

3.3.2 Supported Operations:

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

3.4 Getting Started

To program AVR microcontrollers using the CH552 programmer:

1. Connect the ISP header to the target board (MISO, MOSI, SCK, RESET, VCC, GND).
2. Use compatible software such as `avrdude` or the Arduino IDE with custom programmer settings.
3. Ensure proper voltage levels (typically 5V or 3.3V depending on the target device).
4. Run the programming command to flash the firmware or configure the device.

```
avrdude -p m328p -c ch552 -U flash:w:firmware.hex
```

NOTES

CHAPTER

FOUR

AVR: COMPILE AND UPLOAD CODE

4.1 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`.

4.2 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.

4.3 Example: Compiling a Blink Program

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

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

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

4.3.1 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);
    }
}
```

4.3.2 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.

4.4 Uploading with AVRDUDE

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

4.4.1 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 CH552 USB 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

FIVE

AVR: ARDUINO IDE BOOTLOADER

5.1 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.

5.2 Required Materials

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

5.3 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.

5.4 Driver Setup with Zadig

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

5.4.1 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.

5.4.2 Step 2: Open Zadig

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

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

5.4.3 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.

5.5 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

SIX

STM32 MICROCONTROLLERS

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.

6.1 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.

CHAPTER

SEVEN

OPENOCD AND PYOCD OVERVIEW

7.1 Introduction

7.1.1 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.

7.1.2 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.

7.2 ARM Cortex-M Debug Capabilities

7.2.1 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.

7.2.2 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.

7.3 CMSIS-DAP: A Standardized Debug Adapter

7.3.1 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.

7.4 SWD Communication Protocols

7.4.1 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.

7.4.2 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
EIGHT

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"
```

8.1 Supported Microcontrollers

This guide supports the following microcontrollers:

Microcontroller	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.

8.1.1 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.

Microcontroller	Configuration File
RP2040	`rp2040.cfg`
STM32F103C8	`stm32f103c8t6.cfg`
STM32F411	`stm32f411.cfg`

CHAPTER

NINE

PYOCD

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

CHAPTER

TEN

HOW TO GENERATE AN ERROR REPORT

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

10.1 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.

10.2 Review and Follow-Up

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