JTAG interface documentation for the WCH CH347 chip

written by EasyDevKits – Matthias Jentsch

Version 1.3 from November 21th, 2023

Contact: info@easydevkits.com

Website: https://www.easydevkits.com

Table of contents

1 Scope of this document

I’ve written this document because I didn’t found a description about the API for the JTAG part of the WCH CH347 chip. After researching a while I found something in various source codes about the USB JTAG interface. Beside my own findings I’ve used the following documents:

- CH347 datasheet: https://www.wch-ic.com/downloads/CH347DS1_PDF.html
- ch347.c source code from WCH for OpenOCD: https://github.com/WCHSoftGroup/ch347/blob/main/OpenOCD_SourceCode_CH347/src/jtag/drivers/ch347.c
- ch347.c source code from Alexey Starikovskiy (aystarik) for Linux kernel drivers for vendor class protocols: https://github.com/aystarik/ch347_vcp/blob/main/gpio-ch347.c
- The CH347 research repository: https://github.com/nic3-14159/CH347-Research

My own JTAG implementation for the CH347 in the OpenOCD project can be found in the jtag/drivers/ch347.c file in this repository: https://github.com/EasyDevKits/openocd-easydevkits

2 Overview

These steps are needed for communicate with the CH347 chip:

- CH347T chip in mode 3.
- Connection via USB to the CH347T chip
- Open connection to the USB port
- Send init command
- Send JTAG commands
- Optional: Set GPIO pins

After the general overview all commands for initialization, JTAG and GPIO output are described in detail.

3 CH347T in mode 3

Please refer to the datasheet to get the CH347T into mode 3. In general you do this by connecting pin 10 – DTR1 via 4.7kΩ resistor to ground and connecting pin 13 – RTS1/GP7 via 4.7kΩ resistor to ground. When you power on the chip it starts in JTAG mode (mode 3). The other modes are explained in the datasheet and are not scope of this document.

3.1 Different chip versions

Currently I’ve seen these different versions of the CH347T chip. They differ in the written code at chip and differ in USB BCD code or firmware version.

- Code on chip 31101XC15 has USB BCD code 2.41 and firmware version 41
- Code on chip 31102XD02 has USB BCD code 2.41 and firmware version 41
- Code on chip 31102XD26 has USB BCD code 4.41 and firmware version 41

The BCD code 4.41 chip supports the LARGER_PACK feature. For more information about STANDARD_PACK and LARGER_PACK features see chapter 9.1.4

4 Connection via USB to the CH347T chip

The connection via USB cable to the UB+ / UB- pins and the installation of the correct drivers for Windows or Linux are not scope of this document. You can find the drivers at the WCH site: https://wch-ic.com/products/CH347.html

5 Open connection to the USB port

Via the open source library libusb https://libusb.info/ you can easily connect to the right USB port. Please use the functions **`libusb_init, libusb_get_device_list, libusb_get_device_descriptor, libusb_open`** to connect to the device. In the CH347 JTAG mode there are two interfaces:

- Interface 0 – for UART communication
- Interface 2 – for JTAG communication

You need to claim interface 2 via calling **`libusb_claim_interface`**. When communicating through the JTAG interface you communicate with these two USB endpoints:

- 0x06 – Endpoint for writing from the USB host to the device
- 0x86 – Endpoint for reading from the device to the USB host

Reading and writing is done via **`libusb_bulk_transfer`**.

6 Send init command

Before you can send JTAG commands, you need to send the init command (0xD0) together with the device speed. The device supports speed settings up to 60 MHz.

7 Send JTAG commands

After sending the init command you are ready to send JTAG byte or bit commands. There are four JTAG related commands:

- 0xD1: Sending a bit sequence
- 0xD2: Sending a bit sequence and reading data bits
- 0xD3: Sending bytes sequentially to TDI
- 0xD4: Sending bytes sequentially to TDI and reading bytes from TDO

The electrical JTAG interface consists of the TCK, TMS, TDI, TDO and optionally the TRST pins. In general JTAG is a state machine where you can go through the different states by setting TMS and pulse TCK. With command 0xD1 you can set the pins to correct states without reading any data. If you need also data back from TDO pin you need to send the 0xD2 command to the 0x06 USB endpoint and read data back from 0x86 USB endpoint. With the command 0xD3 it's possible to send whole bytes. That results in 8 bits shifted out to the TDI pin. If you need data back from the TDO pin you use the 0xD4 command and read data from the 0x86 USB endpoint. With 0xD3 and 0xD4 commands only TDI is shifted. But for the very last bit of a Shift-IR or Shift-DR you need to set TMS high. So if you need to send exactly 8 bits you need to send this with the bit commands 0xD1/0xD2 because you need to set TMS high for the last bit. But if you need to send for instance 10 bits you send 1 byte by 0xD3/0xD4 and 2 bits by 0xD1/0xD2 commands.

8 Optional: Set GPIO pins

With the command 0xCC it's possible to set GPIO pins high or low. The 0xCC command is always a read/write command. So after writing it to the 0x06 endpoint you need to read the data back from the 0x86 endpoint. In JTAG mode of the CH347T only the following GPIO's can be used:

- GPIO3 (Pin11 / SCL)
- GPIO4 (Pin15 / ACT)
- GPIO5 (Pin9 / TRST)
- GPIO6 (Pin2 / CTS1)

9 Commands in detail

9.1 Command structure and write read behavior

Each command consists of a 3 byte header and after the header follow the data bytes.

9.1.1 Header bytes

- Byte 0: Command type
- Byte 1: Data length – low byte
- Byte 2: Data length – high byte

9.1.2 Data bytes

After the 3 header bytes the exact amount of bytes which is set in the header byte 1 and 2 needs to follow.

9.1.3 Command concatenation

It’s possible to concatenate the read and read/write JTAG commands (command types 0xD1, 0xD2, 0xD3, 0xD4) together and send more than one command in one transfer. But you can’t put not more than 4096 TDO bit reads into one package. And in STANDARD_PACK mode you can’t send only one USB package with max 507 data bytes. After this one USB package you need to read the TDO data back.

9.1.4 Write and read behavior

If a command is only a write to the device action, the header and data bytes should be send via **`libusb_bulk_transfer`** to the USB write endpoint 0x06. But if it’s a write and read action, then the you additionally should read data via **`libusb_bulk_transfer`** from the USB read endpoint 0x86. How many bytes depends on the command. Older chip versions support the STANDARD_PACK mode. Newer chip versions support LARGER_PACK mode, but you can use the chip also in STANDARD_PACK mode. If the device supports LARGER_PACK mode is determined by the last byte that’s read from the JTAG init command (command type 0xD0).

9.1.5 STANDARD_PACK mode

If the last byte that’s read from the JTAG init command (command type 0xD0) is 0 the chip supports STANDARD_PACK mode. That means you can send one USB package with maximum 507 data bytes and should after that package read data back from USB read endpoint. But be aware of the 4096 TDO bits limit. You can’t send concatenated commands that result in more than 4096 TDO bits to be read.

9.1.6 LARGER_PACK mode

If the last byte that’s read from the JTAG init command (command type 0xD0) is not 0 the chip supports LARGER_PACK mode. Even if the chip supports this mode you can use also the STANDARD_PACK mode and send only one USB package with maximum 507 data bytes. In the

LARGER_PACK mode you can also send packages via **`libusb_bulk_transfer`** that are larger than 507 data bytes. The limit is a total package length of 51200 bytes, but no more than 4096 TDO bits read. That means, you can send one large USB package and read the data back from USB read endpoint with one large package.

9.2 0xD0: Init command

Header bytes: Always 0xD0, 0x06; 0x00

6 data bytes:

- Byte 0: Always 0
- Byte 1: Speed index
- Bytes 2 to 5: Always 0

This command can not be concatenated with other commands into one USB package.

9.2.1 Speed index

You need to issue the init command twice. First with speed index 9. After sending the command you read always 4 bytes back. If in the read data the first data byte (first byte after header) is 0 then the device has the STANDARD_PACK feature. Otherwise it's a LARGER_PACK device.

For example the USB BCD code 4.41 chip supports this LARGER_PACK feature.

ATTENTION: When you have a LARGER_PACK device you have to deliver other values for the speed index than for a STANDARD_PACK device!

For the STANDARD_PACK device you send the command again with one of this speed index:

- 0 = JTAG speed 1.875 MHz
- 1 = JTAG speed 3.75 MHz
- 2 = JTAG speed 7.5 MHz
- 3 = JTAG speed 15 MHz
- 4 = JTAG speed 30 MHz
- 5 = JTAG speed 60 MHz

For a LARGER_PACK device the speed indexes are:

- 0 = JTAG speed 468.75 kHz
- 1 = JTAG speed 937.5 kHz
- 2 = JTAG speed 1.875 MHz
- 3 = JTAG speed 3.75 MHz
- 4 = JTAG speed 7.5 MHz
- 5 = JTAG speed 15 MHz

- 6 = JTAG speed 30 MHz
- 7 = JTAG speed 60 MHz

9.2.2 Example

Write: **0xD0, 0x06, 0x00, 0x00, 0x09, 0x00, 0x00, 0x00, 0x00** – Test with speed index 9

Read: **0xD0, 0x01, 0x00, 0x00** – It's a STANDARD_PACK device

Write: **0xD0, 0x06, 0x00, 0x00, 0x05, 0x00, 0x00, 0x00, 0x00** – Speed index 5 = 15 MHz

Read: **0xD0, 0x01, 0x00, 0x00** – It's a STANDARD_PACK device

9.3 0xD1 / 0xD2: Bit operation without and with read

Header bytes:

- Byte 0: 0xD1 for write only operation; 0xD2 for write and read operation
- Byte 1 and 2 data length; low byte first

Each data byte represents the for JTAG pins TCK, TMS, TDI and TRST

- Bit 0: 0= TCK low; 1 = TCK high
- Bit 1: 0= TMS low; 1 = TMS high
- Bit 2: Always 0
- Bit 3: Always 0
- Bit 4: 0= TDI low; 1 = TDI high
- Bit 5: 0= TRST low; 1 = TRST high
- Bit 6: Always 0
- Bit 7: Always 0

When you expect to read the bit data from TDO you should read 3 + expected bit count from the reading USB endpoint. For instance if you shift out 2 bits to TDI you expect 2 TDO bits and therefore you read 5 (3 + 2) data bytes. In the read data bytes (after the 3 header bytes) each 0x00 represents a TDO low. Each 0x01 is a TDO high bit.

This command can be concatenated with other commands into one USB package.

9.3.1 Example

Write: **0xD1, 0x05, 0x00, 0x02, 0x03, 0x00, 0x01, 0x00** – Two clock pulses; TMS from high to low

Write: **0xD2, 0x06, 0x00, 0x10, 0x11, 0x10, 0x12, 0x13, 0x12** – Two clock pulses; TMS from low to high to signal the last bit

Read: **0xD2, 0x02, 0x00, 0x01, 0x00** – TDI first bit high; second bit low

9.4 0xD3 / 0xD4: Byte operation without and with read

Header bytes:

- Byte 0: 0xD3 for write only operation; 0xD4 for write and read operation
- Byte 1 and 2 data length; low byte first

Each data byte represents 8 TDI bits which will be shifted out. If you expect data at TDO pin you issue the 0xD4 command and read exactly the same amount of bytes back.

There is at least on report that this functions are not working correctly if the device version is below 2.41. https://github.com/WCHSoftGroup/ch347/issues/13#issuecomment-1746154265 To check the version you can get the device descriptor via **`libusb_get_device_descriptor`** call and read out the **`bcdDevice`**. The high byte of this 16-bit value is the major version; the low byte the minor version. E.g. 0x0241 is the version 2.41.

This command can be concatenated with other commands into one USB package.

9.4.1 Example

Write: **`0xD3, 0x02, 0x00, 0x55, 0xAA`** – Two bytes (16 bits) shifted out to TDI

Write: **`0xD4, 0x02, 0x00, 0xFF, 0xFF`** – Two bytes (16 bits) shifted out to TDI

Read: **`0xD4, 0x02, 0x00, 0x12, 0x34`** – Two bytes (16 bits) are read from TDO

9.5 0xCC: Set GPIO pins

Header bytes: Always **`0xCC, 0x08, 0x00`**

There are 8 data bytes. One for each GPIO. But for the CH347T chip in JTAG mode only GPIO 3, 4, 5 and 6 can be used. Maybe there can be more pins used for the CH347F version. For each GPIO that should be set to high or low the corresponding byte should be set. The other bytes should be zero. The bits in each data byte have the following meanings:

- Bit 0-2: not used
- Bit 3: 0 = output low; 1 = output high
- Bit 4-5: 00 = GPIO direction is input; 11 = GPIO direction is output
- Bit 6-7: 00 = don’t change the GPIO; 11 = Set GPIO direction and output to high or low

This results in these meaningful byte values for each GPIO:

- 0x00: Leave GPIO unchanged
- 0xC0: Set to GPIO direction to input and GPIO pin to low
- 0xC8: Set to GPIO direction to input and GPIO pin to high
- 0xF0: Set to GPIO direction to output and GPIO pin to low
- 0xF8: Set to GPIO direction to output and GPIO pin to high

After sending the 11 bytes you should read 11 bytes. In the read data the input status can be found. The bits in each data byte have the following meanings:

- Bit 0-5: not used
- Bit 6: 0 = GPIO input is low; 1 = GPIO input is high
- Bit 7: 0 = GPIO is an input pin; 1 = GPIO is an output pin

This results in these meaningful byte values for each GPIO:

- 0x00: Input pin read as low
- 0x40: Input pin read as high
- 0x80: Output pin read as low
- 0xC0: Output pin read as high

This command can not be concatenated with other commands into one USB package.

9.5.1 Example

Write: **`0xCC, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF8, 0x00, 0x00, 0x00`** – Set GPIO4 to output high; leave all others untouched

Read: **`0xCC, 0x08, 0x00, 0x00, 0x40, 0x00, 0x40, 0xC0, 0x00, 0x40, 0x40`** – GPIO4 is output pin set to high; GPIO0, 2 and 5 are input pins read as low; GPIO1, 3, 6 and 7 are input pins read as high

9.6 SWD commands

A CH347 chip in JTAG mode (mode 3) support also serial wire debugging (SWD). Serial Wire Debug (SWD) is a 2-pin (SWDIO/SWCLK) electrical alternative JTAG interface that has the same JTAG protocol on top. I’ve not digged into the SWD commands, but here are the known command types:

- 0xE5: Initialization command that sets the clock speed
- 0xE8: All 0xA0, 0xA1 and 0xA2 commands are send as data bytes in this command
- 0xA0: Write register
- 0xA1: Write switch sequence / special sequence
- 0xA2: Read register