Orbtrace

Orbcode

Jun 26, 2023

CONTENTS

Orbtrace is a lightweight, cost effective, USB2-HS Debug and Trace interface for ARM CORTEX-M processors. It allows you to debug your code on target using CMSIS-DAP and, optionally, to extract Debug and Trace data from it in real time, according to the Arm Debug Interface Architecture Specification ADIv5.

CHAPTER

ONE

PURPOSE

Orbtrace performs two essential functions, with a number of other supporting bits and pieces around the edges. Specifically it;

1. Presents a *Debug Interface* to which CMSIS-DAP v1 or v2 compliant interface software can attach, to control and debug a target device over either a SWD or JTAG interface.
2. Collects data from the *Parallel Trace Port* of the target device over a 1, 2 or 4 bit interface and presents it to software on the host PC. These data can be used for reporting on the progress of the software running on the target device, and optionally to reconstruct it's recent actions.

In addition to the above, Orbtrace can also, depending on the hardware it's running on, provide power to the target and measure its current consumption during program execution. It can also provide secondary serial links. All of these data streams are carried over a single USB2-HS communnication link between Orbtrace and the host PC.

The normal connection between Orbtrace and the target is the 2x10-way 0.05" connector you will see on the target PCB. When doing debug only (i.e. no tracing functions) then the smaller 2x5-way 0.05" connector is typically used instead. Details on these connectors are available from the Keil Website.

In respect of (1) above, typically software that would communicate with Orbtrace would be BlackMagic Probe or PyOCD. These would then connect to a debugger such as gdb.

In respect of (2) above, exploiting the trace flow from Orbtrace, the reader is directed to the Orbuculum suite for detailed information.

CHAPTER

TWO

USB TRACE INTERFACE

The trace interface is identified by `bInterfaceClass = 0xff` and `bInterfaceSubclass = 0x54 ('T')`. It may have multiple alternate settings with different `bInterfaceProtocol` values to support different trace protocols. Host software negotiates protocol by reading the list of supported alternate settings and selecting the preferred one.

2.1 Control Requests

Control requests are vendor-specific interface-directed, i.e. with `bmRequestType = 0x41 or 0xc1` and the lower half of `wIndex` containing `bInterfaceNumber`.

2.1.1 Set Input Format

bmRequestType	bRequest	wValue	wIndex	wLength
0x41	0x01	Type	bInterfaceNumber	0

Type	Description
0x00	Disabled
0x01	1-bit synchronous
0x02	2-bit synchronous
0x03	4-bit synchronous
0x10	Manchester asynchronous (ITM)
0x11	Manchester asynchronous (TPIU)
0x12	NRZ asynchronous (ITM)
0x13	NRZ asynchronous (TPIU)

2.1.2 Set Async Baudrate

bmRequestType	bRequest	wValue	wIndex	wLength
0x41	0x02	0x00	bInterfaceNumber	4

Payload is baudrate as a 32-bit little endian integer.

2.2 Protocols

2.2.1 Undefined

bInterfaceClass	bInterfaceSubclass	bInterfaceProtocol
0xff	0x54	0x00

Trace interfaces with undefined protocol should be used by devices not aware of the format of the data stream (e.g. when capturing raw SWO). In this case, the user is expected to manually configure the host software for the correct format.

2.2.2 TPIU

bInterfaceClass	bInterfaceSubclass	bInterfaceProtocol
0xff	0x54	0x01

This protocol uses one endpoint that will send one or more 16-byte TPIU frames per transfer. TPIU frames are aligned to USB transfer boundaries.

2.2.3 ITM

bInterfaceClass	bInterfaceSubclass	bInterfaceProtocol
0xff	0x54	TBD

TBD

2.2.4 ETM

bInterfaceClass	bInterfaceSubclass	bInterfaceProtocol
0xff	0x54	TBD

TBD

2.2.5 ITM + ETM

bInterfaceClass	bInterfaceSubclass	bInterfaceProtocol
0xff	0x54	TBD

This protocol provides both an *ITM* and an *ETM* endpoint. Refer to the respective sections for details.

CHAPTER

THREE

USB POWER INTERFACE

The power interface is identified by `bInterfaceClass = 0xff` and `bInterfaceSubclass = 0x50 ('P')`.

3.1 Control Requests

Control requests are vendor-specific interface-directed, i.e. with `bmRequestType = 0x41 or 0xc1` and the lower half of `wIndex` containing `bInterfaceNumber`.

3.1.1 Set enable

bmRequestType	bRequest	wValue	wIndex	wLength
0x41	0x01	Enable	Channel << 8 | bInterfaceNumber	0

Channel	Description
0x00	VTREF
0x01	VTPWR
0xFF	All channels

3.1.2 Set voltage

bmRequestType	bRequest	wValue	wIndex	wLength
0x41	0x02	Voltage	Channel << 8 | bInterfaceNumber	0

Channel	Description
0x00	VTREF
0x01	VTPWR

Voltage is expressed in millivolts.

3.1.3 Get status

bmRequestType	bRequest	wValue	wIndex	wLength
0xc1	TBD	TBD	bInterfaceNumber	TBD

CHAPTER

FOUR

USB VERSION INTERFACE

The version interface is identified by `bInterfaceClass = 0xff` and `bInterfaceSubclass = 0x56 ('V')`.

The interface string of the version interface contains the version of the current gateware build, as per `git describe --always --long --dirty`.

Example: `Version: v1.0.0-0-g3ad3fa4`

4.1 Control Requests

The version interface currently has no defined control requests.

CHAPTER

FIVE

USB CONTROL PROXY INTERFACE

The control proxy interface is identified by `bInterfaceClass = 0xff` and `bInterfaceSubclass = 0x58 ('X')`.

Certain operating systems (e.g. Windows) disallows issuing control requests to an interface that's already claimed for bulk transfer by another process. To allow e.g. configuring the *USB Trace Interface* while it's already opened for capture, this interface is provided as a workaround.

5.1 Control Requests

Control requests are vendor-specific interface-directed, i.e. with `bmRequestType = 0x41 or 0xc1` and the lower half of `wIndex` containing the `bInterfaceNumber` of this interface.

`bRequest` has a range for each supported target interface with an associated offset. When a request is handled, the offset is subtracted from `bRequest` and the request is forwarded to the target interface's handler.

bRequest range	Offset	Target interface
0x01 - 0x0f	0x00	*USB Trace Interface*

CHAPTER

SIX

DOCUMENTATION

Orbtrace must be properly documented! The documentation is maintained at Read The Docs and is auto-built from the committed github main repository.

6.1 Editing

Edit the contents of `docs/source/*.rst` to update the documentation. If you have the Sphinx Documentation Generator installed locally you can get a live preview of the current code by running something like;

```
`sphinx-autobuild --port 1232 ~/Develop/orbtrace/docs/source/ /tmp/sp`
```

. . . and then pointing your browser at `localhost:1232`.

6.2 Style

Documentation is not a tutorial, it's there to tell users what to do, not nessesarily to teach them how to do it. Keep it brief but content rich, and link to other sources whenever possible so folks aren't left flapping in the wind.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search