Embedded Systems Hands-On 1: Design and Implementation of Hardware/Software Systems

Task 2: Serial Wire Debugging of the Cortex-M0

Carsten Heinz

May 20, 2020

TECHNISCHE UNIVERSITÄT DARMSTADT

Embedded Systems & Applications

In this task, the debugging of the Cortex-M0 via SWD is tested to be prepared for the later development challenges. It is scheduled for two weeks.

Summary

- Understanding and implementing SWD
- Connecting the Cortex-M0 via SWD with a [GDB] server on the Cortex-A53
- Remote debugging with [Eclipse]

1 Serial Wire Debugging

In this subtask, you will implement SWD with your favorite programming language.

Summary

- Implement the basic SWD features on the Raspberry [Pi].
- At least, the ID of the debugged target processor has to be retrieved.

ARM developed SWD as a low-cost alternative for the standard JTAG debugging port. Besides the power supply connections (GND and V_{CC}), the data signals required for debugging are reduced from at least four wires (TDI, TDO, TCK, and TMS) for JTAG to two wires (SWDIO and SWCLK) for SWD without loss of functionality. With up to 4 MB/s, SWD is also relatively fast. Additional information about the SWD protocol can be found in [Core-Sight, see SW-DP].

With SWD, special registers can be read from the target processor, such as the `IDCODE` register containing the processor ID. Write a program using your favorite programming language, which is capable of reading the `IDCODE` register of the Cortex-M0 when executed on the Raspberry [Pi]. Use the GPIO pins of the Cortex-A53 determined in Task 1.2 to drive the SWD protocol. Compare the captured processor ID with the one shown in the [OpenOCD] output.

[RPi.GPIO] can be used to control the GPIO pins of the Raspberry [Pi] with Python. Similar libraries are available for other programming languages too. *Attention: SWD requires a certain reset sequence.* Without this sequence, accessing the registers will not be possible.

Minimum Expected Documentation

- The documented source code of the SWD program for reading the `IDCODE`.
- A short summary of the SWD mode of operation.

2 GDB Server

In the last task, the functionality of [OpenOCD] was already tested. In this subtask, [OpenOCD] will be used as [GDB] server for remote debugging.

Summary

- Connect `arm-none-eabi-gdb` on the host to a [GDB] server on the Raspberry [Pi].
- Apply [GDB] commands to control and observe the Cortex-M0 program execution.

By default, [OpenOCD] provides a [GDB] server via TCP, so the Cortex-M0 can be debugged on the host PC instead of the Raspberry [Pi]. Start [OpenOCD] on the Raspberry [Pi] and connect an `arm-none-eabi-gdb` on your host PC with the [GDB] server on the Raspberry [Pi] using the `target remote` [GDB] commands. As soon as the connection is established, the `monitor` commands can be used to access the [OpenOCD] functionality via the [GDB] connection. For example, `monitor reset halt` will restart the Cortex-M0 and stop the program execution immediately before the first instruction. Read out the content of the Cortex-M0 registers. Configure a breakpoint. More information about relevant [GDB] commands can be found in the [GDB-cheatsheet].

Minimum Expected Documentation

- Log of a remote Cortex-M0 debugging session

3 Debugging IDE

Although debugging via the [GDB] command line interface is feasible, an IDE significantly improves the debugging efficiency by correlating the program counter with the appropriate source code lines. In this subtask, the remote debugging of embedded systems with one of [Eclipse; GDBGUI; PWNDBG] is thus evaluated.

Summary

- Setup and use a remote debugging target in [Eclipse; GDBGUI; PWNDBG].

The process varies according to the tool used. In general however the debugger has connect to the remote GDB server provided through OpenOCD. For [Eclipse] the [GNU-ARM-Eclipse] tools are used to create a `GDB Hardware Debugging` target for remote debugging. Specify the TCP/IP address and port of the Raspberry [Pi] [GDB] server and start the debugger. You are now able to step through the Cortex-M0 program execution and to use the `monitor` commands for resetting and flashing the target processor via [OpenOCD].

Minimum Expected Documentation

- Screenshots of the setup and usage of the [Eclipse; GDBGUI; PWNDBG] remote debugging session.

Bibliography

Core-Sight ARM. *Technical Reference Manual*. 2009. URL: `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0314h/DDI0314H_coresight_components_trm.pdf` (visited on 2018-03-08).

Eclipse *C/C++ Development Tooling*. 2016. URL: `https://eclipse.org/cdt/` (visited on 2018-03-08).

GDB GNU. *The GNU Project Debugger*. URL: `https://www.gnu.org/software/gdb/` (visited on 2018-03-08).

GDB-cheatsheet Marc Haisenko. 2007. URL: `http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf` (visited on 2018-03-08).

GDBGUI *gdbgui is a modern, free, browser-based frontend to gdb*. 2019. URL: `https://gdbgui.com` (visited on 2019-04-16).

GNU-ARM-Eclipse *A family of Eclipse CDT extensions and tools for GNU ARM development*. URL: `http://gnuarmeclipse.github.io/` (visited on 2018-03-08).

OpenOCD Dominic Rath. *Open On-Chip Debugger*. URL: `http://openocd.org` (visited on 2018-03-08).

Pi Raspberry Pi Foundation. *Raspberry Pi 3 Model B*. URL: `https://www.raspberrypi.org/products/raspberry-pi-3-model-b` (visited on 2018-03-08).

PWNDBG *Exploit Development and Reverse Engineering with GDB Made Easy*. 2019. URL: `https://github.com/pwndbg/pwndbg` (visited on 2019-04-16).

RPi.GPIO Python Software Foundation. *A module to control Raspberry Pi GPIO channels*. URL: `https://pypi.python.org/pypi/RPi.GPIO` (visited on 2018-03-08).

Acronyms

GND	Ground. Electrical Reference Potential
GPIO	General Purpose IO
IDE	Integrated Development Environment
IP	Internet Protocol
JTAG	Joint Test Action Group
SWCLK	Serial Wire Clock
SWD	Serial Wire Debugging
SWDIO	Serial Wire Data Input/Output
TCK	Test Clock
TCP	Transmission Control Protocol
TDI	Test Data In
TDO	Test Data Out
TMS	Test Mode Select
V_{CC}	Voltage at the Common Collector. Electrical supply potential in BJT circuits