Embedded Systems Hands-On 1: Entwurf und Realisierung von Hardware/Software-Systemen

Jaco Hofmann M.Sc., Dipl.-Inform. Andreas Engel

Sommersemester 2016 (Stand 3. Mai 2016)

Aufgabe 2: Debugging

Als Vorbereitung auf die eigentliche Arbeit mit dem Mikrocontroller in den nächsten Wochen beschäftigt sich diese Aufgabe mit dem Debuggen über SWD.

Zusammenfassung

- SWD verstehen und implementieren
- Mit GDB Server verbinden
- Debugging mit Eclipse

2.1 Serial Wire Debug (SWD)

In dieser Teilaufgabe sollen Sie SWD in einer Programmiersprache Ihrer Wahl implementieren.

Zusammenfassung

- Implementieren Sie die Grundzüge von SWD auf dem Raspberry PI. Zumindest die Prozessor ID muss ausgelesen werden können.

Serial Wire Debug wurde von ARM als low cost alternative zum Standard JTAG Debugging Port entwickelt. Im Gegensatz zu JTAG verwendet SWD nur zwei Verbindungen (neben Ground und VCC), erlaubt aber den gleichen Zugriff auf die verschiedenen Teile des Prozessors. Zusätzlich ist SWD relativ schnell mit bis zu 4 MB/s.

Über SWD ist es möglich spezielle Register auszulesen. Eines dieser Register beinhaltet die Identifikations-ID des Prozessors. Schreiben Sie ein Programm in einer Programmiersprache Ihrer Wahl das dieses Register über die GPIO Pins des Raspberry PI auslesen kann. Verwenden Sie die selben Pins die Sie auch in Aufgabe 1 für die OpenOCD Verbindung genutzt haben. Informationen über SWD erhalten sie zum Beispiel bei ARM (SW-DP). Vergleichen Sie die von Ihnen ausgegebene ID mit der von OpenOCD ausgegebenen ID.

Zur Ansteuerung der GPIO Pins des Raspberry PI können Sie in Python RPi.GPIO verwenden. Für andere Programmiersprachen gibt es ähnliche Libraries.

Hinweis

SWD verwendet eine spezielle Reset Sequenz. Ohne diese wird kein Zugriff auf die Daten der Register gelingen.

Abgabe

- Der dokumentierte Code Ihres SWD CPUID Lese Programms
- Kurze Zusammenfassung über die Funktionsweise von SWD

2.2 GDB Server

In der letzten Aufgabe wurde die Funktion von OpenOCD bereits getestet. In dieser Aufgabe soll OpenOCD nun als GDB Server zum debuggen verwendet werden. OpenOCD verwendet standardmäßig TCP für die Bereitstellung des GDB Servers. Dies hat den Vorteil, dass das Debuggen des Cortex M0 nicht auf dem Raspberry PI erfolgen muss, sondern auf dem Host PC erfolgen kann. Starten Sie OpenOCD auf dem Raspberry PI und verbinden Sie einen `arm-none-eabi-gdb` auf Ihrem Host PC mit dem GDB Server auf dem Raspberry PI. Verwenden Sie dafür die `target remote` Befehle von GDB. Wenn die Verbindung hergestellt wurde können Sie `monitor` Befehle nutzen um Anweisung über die GDB Verbindung an OpenOCD weiterzugeben. Mit `monitor reset halt` können Sie zum Beispiel den Cortex M0 Kern neu starten und dann, bevor eine Instruktion ausgeführt wurde, anhalten. Lassen Sie sich den Inhalt der Register des Cortex M0 ausgeben. Setzen Sie einen Breakpoint. Informationen über die Befehle von GDB erhalten Sie zum Beispiel in http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf

2.3 Debugging IDE

Obwohl Sie auch auf der Kommandozeile GDB verwenden können, hilft eine IDE, die Codezeilen mit Programmcountern in Relation setzen kann, ungemein beim Debugging. Daher beschäftigt sich diese Aufgabe beispielhaft mit dem Remote Debugging von Embedded Systemen in Eclipse. Verwenden Sie unter Eclipse die GNU ARM Eclipse Tools. Erstellen Sie ein neues `GDB Hardware Debugging` Debug Target für das remote Debugging. Geben Sie die IP Adresse sowie den Port des Raspberry PI OpenOCD GDB Servers an und drücken Sie auf Debug. Sie können nun durch den Programmcode auf dem Cortex M0 steppen. Außerdem können Sie mit `monitor` Befehlen OpenOCD steuern (reset, flash etc.).