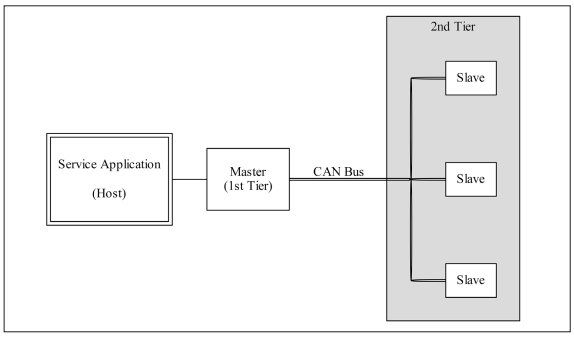
Felix
Hofer

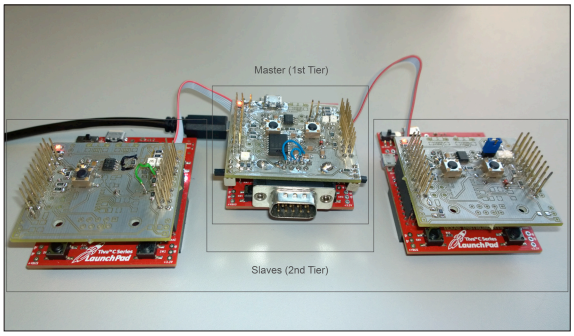
Diplomand	Felix Hofer
Examinator	Prof. Erwin Brändle
Experte	Theo Scheidegger, swens GmbH, Schänis, GL
Themengebiet	Embedded Systems

Universal Bootloader Environment

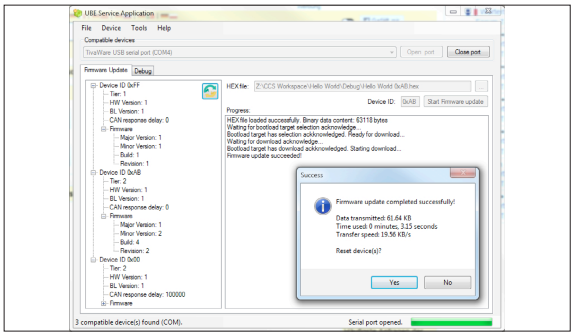
Konzeption und Implementation einer universellen Bootloader-Lösung für die ARM Cortex-M4-Familie



Beispiel-Systemaufbau (Prinzip-Darstellung)



Realer Hardwareaufbau



PC-seitige Serviceapplikation

Ausgangslage: Während der Entwicklung von Software für Embedded Devices wird diese meist über ein JTAG-Interface in den Programmspeicher des Zielsystems geladen. Die Debug-Schnittstelle eröffnet zugleich vielfältige Debug-Möglichkeiten, die im Produktivbetrieb jedoch meistens nicht mehr notwendig sind. Der Zugang zum Debug-Interface ist dann nicht mehr gewährleistet. Um trotzdem Firmware-Updates vornehmen zu können, ist man deshalb auf einen sogenannten Bootloader angewiesen, der diese Funktionalität über eine alternative Schnittstelle zur Verfügung stellt.

Ziel der Arbeit: Es soll eine universelle Bootloader-Lösung für die ARM Cortex-M4-Familie auf TI Tiva C-Launchpads konzipiert, mit der Programmiersprache C implementiert und umfassend getestet werden. Mit der entwickelten Bootloader-Lösung sollen Firmware-Updates zuverlässig über mindestens zwei Ebenen (Tiers) möglich sein. Dazu muss auch eine PC-seitige Serviceapplikation entwickelt werden, die in einer Windows-Umgebung mit .NET Framework lauffähig sein soll und die Kommunikation mit dem 1st Tier (Master) über eine COM-Schnittstelle übernimmt. Weiter muss für die Launchpads eine Zusatzhardware zur Energieversorgung der Slaves (2nd Tier) und zur Vernetzung von 1st und 2nd Tier über einen CAN-Bus entwickelt werden. Das Ausgangsformat für ein Firmware-Update über die entwickelte Bootloader-Lösung soll das Intel-Hex-Format sein, wie es die Entwicklungsumgebung Code-Composer-Studio erzeugt. Über eine eindeutige Kennzeichnung in der Hex-Datei und im Speicher der Embedded Devices soll die Kompatibilität der Firmware zur Hardware überprüft und so unzulässige Konstellationen verhindert werden. Weiter sollen Übertragungs- und andere Fehler erkannt und dementsprechende Massnahmen eingeleitet werden.

Ergebnis: Trotz massiven Schwierigkeiten aufgrund einer nicht (richtig) funktionierenden CAN-Arbitrierung bzw. Kollisions-Detektion und entsprechendem Handling seitens CAN-Controller/-Libraries konnte dank einer manuellen Implementation schlussendlich dennoch ein Gesamtsystem entwickelt werden, welches die gegebenen Anforderungen erfüllt und in der Lage ist, auch über CAN zuverlässig Firmware-Updates durchzuführen.