# Embedded Systems
# Kapitel 11: SW Download, Debugging

Prof. Dr. Wolfgang Mühlbauer
Fakultät für Informatik
wolfgang.muehlbauer@th-rosenheim.de

Sommersemester 2020

# Bedeutung von Embedded Software

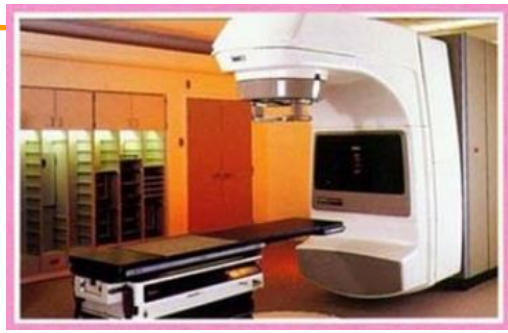
- Mikrocontroller und Software übernehmen zunehmend ***sicherheitskritische Funktionen***
  - Industrieroboter
  - Automobile
  - Schienenfahrzeuge, z.B. U-Bahn
  - Herzschrittmacher/Defibrillatoren
  - **...**

Quelle: https://www.logismarket.de/ip/kuka-deutschand-industrieroboter-kr-1000-1300-titan-pa-industrieroboter-kr-1000-1300-titan-pa-682316-FGR.jpg

- Softwarefehler können **gravierende Folgen haben! … gutes Testen**

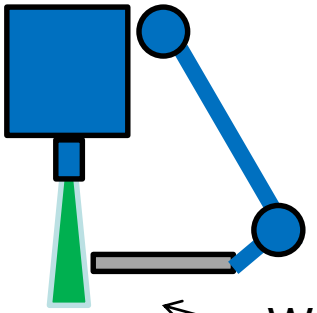
# Beispiel: Therac-25

- Linearbeschleuniger für Strahlentherapie

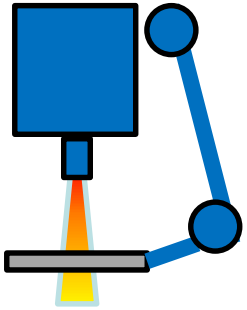


Quelle: https://hci.cs.siue.edu/NSF/Files/Semester/Week13-2/PPT-Text/images/Image3.png

**Elektronenmodus:**
Geringe Energie



**Röntgenmodus:**
Hohe Energie



Wolframplatte

- Röntgenmodus ohne Wolframplatte: **3 Tote** und etliche Verletzte!

- Ursachen: Gravierende Entwicklungsfehler
  - Inkonsistente Konfiguration nach Korrektur der Eingabedaten während der Ansteuerung des Gerätes.
  - Keine Redundanz durch Hardwareverriegelung
  - 1 Entwickler, der seinen eigenen Code testete

# Funktionale Sicherheit

- **Definition (ED 109 / DO 278)**
  - "*Software performs its intended functions under any foreseeable operating condition*"
  - System- als auch Sicherheitsanforderungen müssen definiert, umgesetzt und vollständig getestet werden!

- Funktionale Sicherheit in der Praxis
  - Erfordert klar *definierten, strukturierten, dokumentierten* Software-Entwicklungsprozess
  - Nachweis der funktionalen Sicherheit durch Zertifizierung.

- Rest der Vorlesung
  - Welche Methoden existieren für das Programmieren und Debuggen von Embedded-Software Anwendungen?
  - Wie testet man Hardware und das Zusammenspiel von Hardware und Software?

# Inhalt

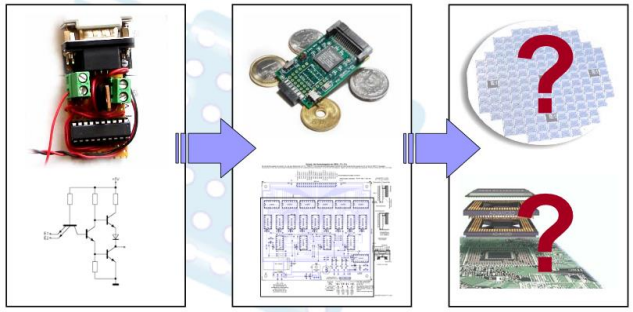
- Funktionale Sicherheit
- **JTAG: Boundary Scan**
- Download von Software auf Mikrocontroller („Flashen“)
- Debug-Methoden

# JTAG

- Joint Test Action Group"
- Standard IEEE 1491.1
- Ursprüngliche Hauptanwendung: ***Boundary Scan Test***
  - Deutsch: "Grenzpfadabtastung"
  - Standardisiertes Verfahren um digitale und analoge Bausteine in integrierten Schaltungen zu testen.
- Anwendungen für Programmierer von Mikrocontrollern
  - HW-Debugging
  - „Flashen“ ohne Bootloader

# Motivation: Testen

- Testen erschwert durch fortschreitenden Integrationsgrad
  - Schlechtere Zugänglichkeit der Komponenten
  - Mehr Funktionalität → Mehrbedarf an Tests

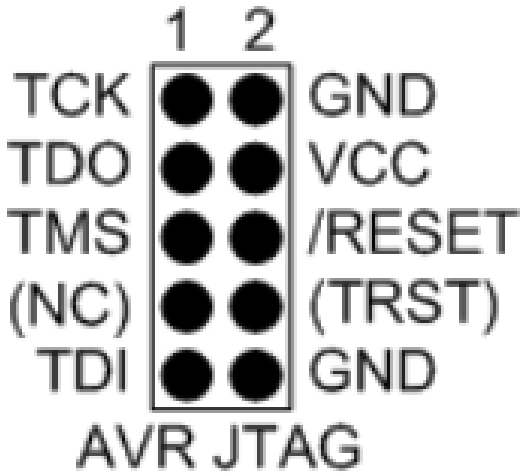



- Hohe Kosten bei unzureichenden Tests

- JTAG Boundary Scan ersetzt "***Nadelbett***"-Methode
  - Spezieller Prüfadapter mit konfigurierten, gefederten Prüfstiften wird gegen Testpunkte auf Leiterplatte gedrückt.
  - Bei JTAG dagegen Erreichbarkeit aller **virtuellen** Testpunkte über eine **einzige, einheitliche** Schnittstelle.

# JTAG: Elektrische und mechanische Schnittstelle

- 6 Leitungen
  - **TCK**: Takt
  - **TMS**: Test Mode Select → Steuerung
  - **TDI**: Test Data In → Testdateneingang
  - **TDO**: Test Data Out → Testdatenausgang
  - **VCC**: Versorgungsspannung
  - **GND**: Masse



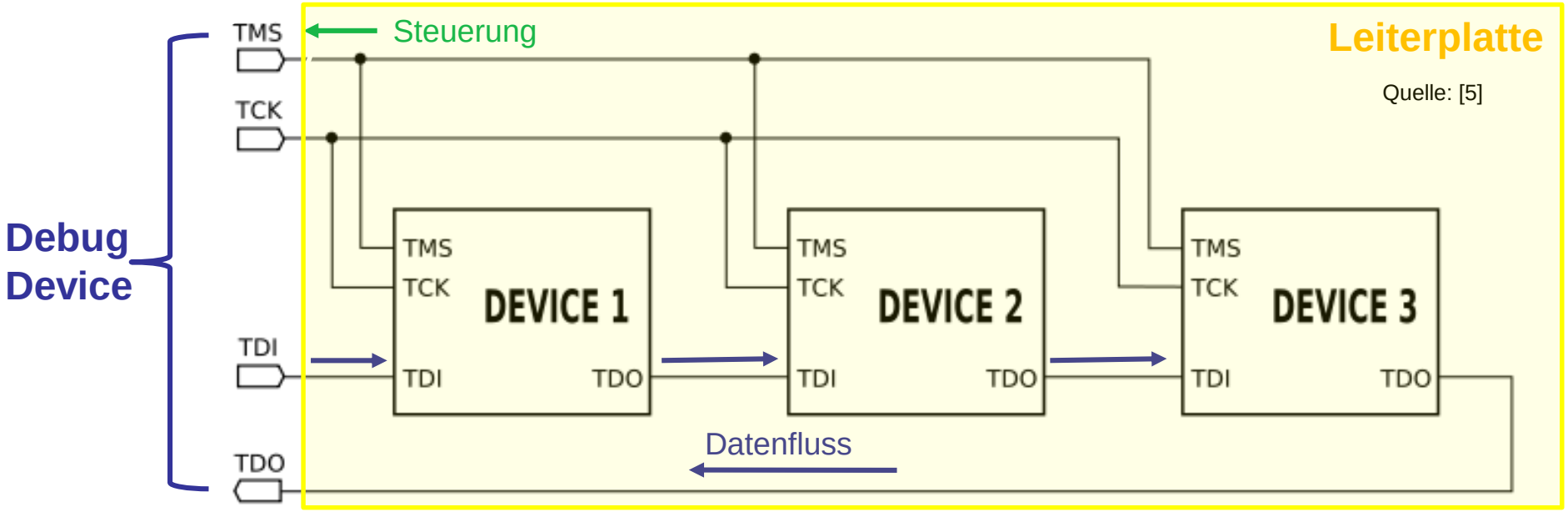

**JTAG-Stecker:**
**Pin-Belegung bei Atmel**
Quelle: [4]

- Zahlreiche unterschiedliche Steckerformen
  - Nicht genormt!

# Boundary Scan

- **Ziel:**
  - Testen benötigt nur 4 Verbindungen
  - Dennoch: Abfragen und Schreiben **aller** Testpunkte (meist Pins) **aller** ICs einer Leiterplatte.

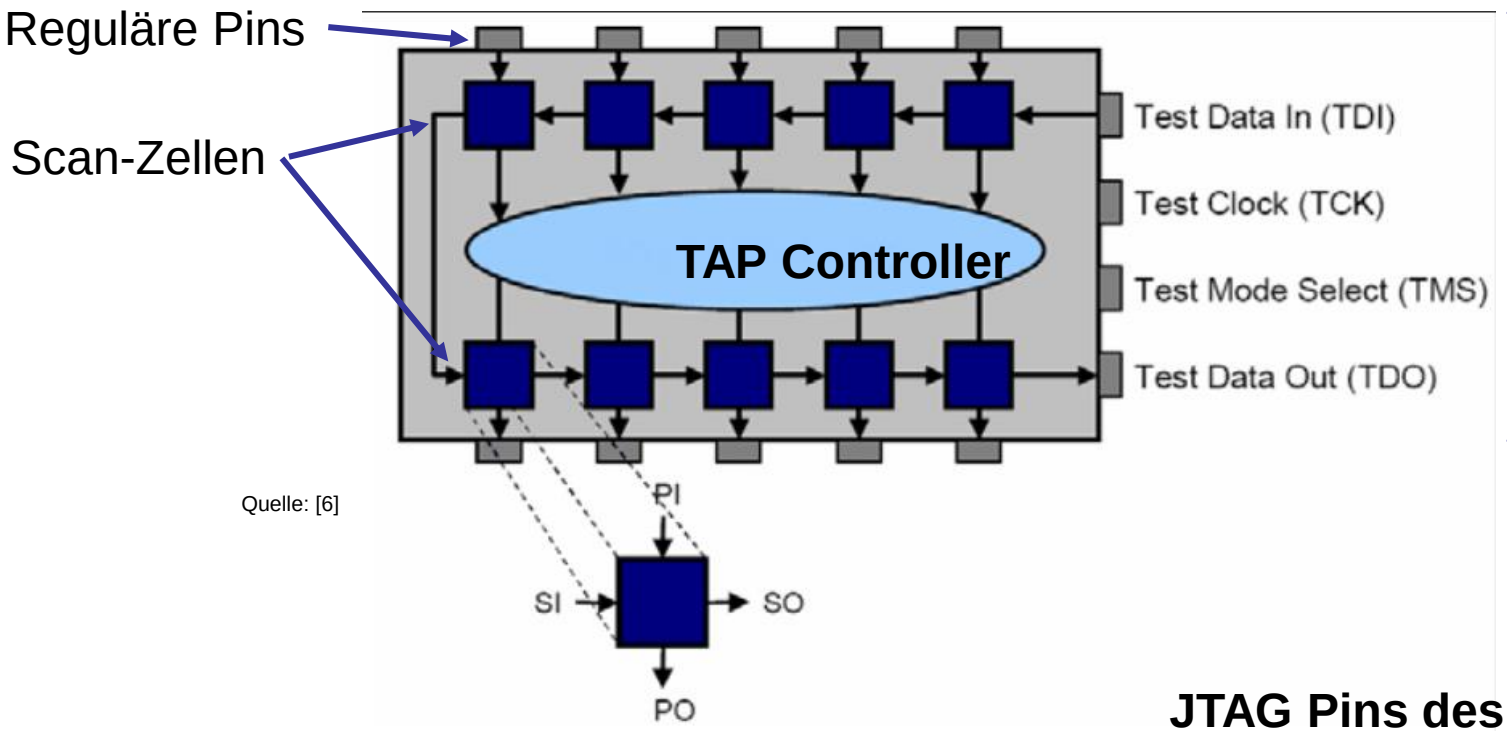
- **Ansatz: JTAG Chain**
  - Verbinde alle ICs („Devices“) in einer Kette.
  - „Kreisförmiges“ Weiterschieben der Daten pro Takteinheit mittels **Schieberegister.**
  - Mit der Zeit bekommt man Zugriff auf den gewünschten Messwert.



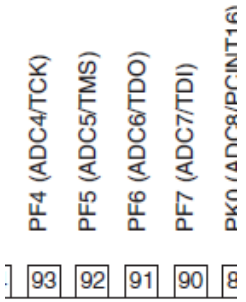
# Boundary-Scan

- Blick in ein Device /IC, siehe unten.
- Scan-Zellen (=Messeinheit) sind integriert in Device
  - Greifen z.B. externe, physikalische Pins ab.
  - Sind untereinander über Schieberegister gekoppelt.
  - Sind "abschaltbar“.

**Aufbau eines Device/ICs mit JTAG Unterstützung**



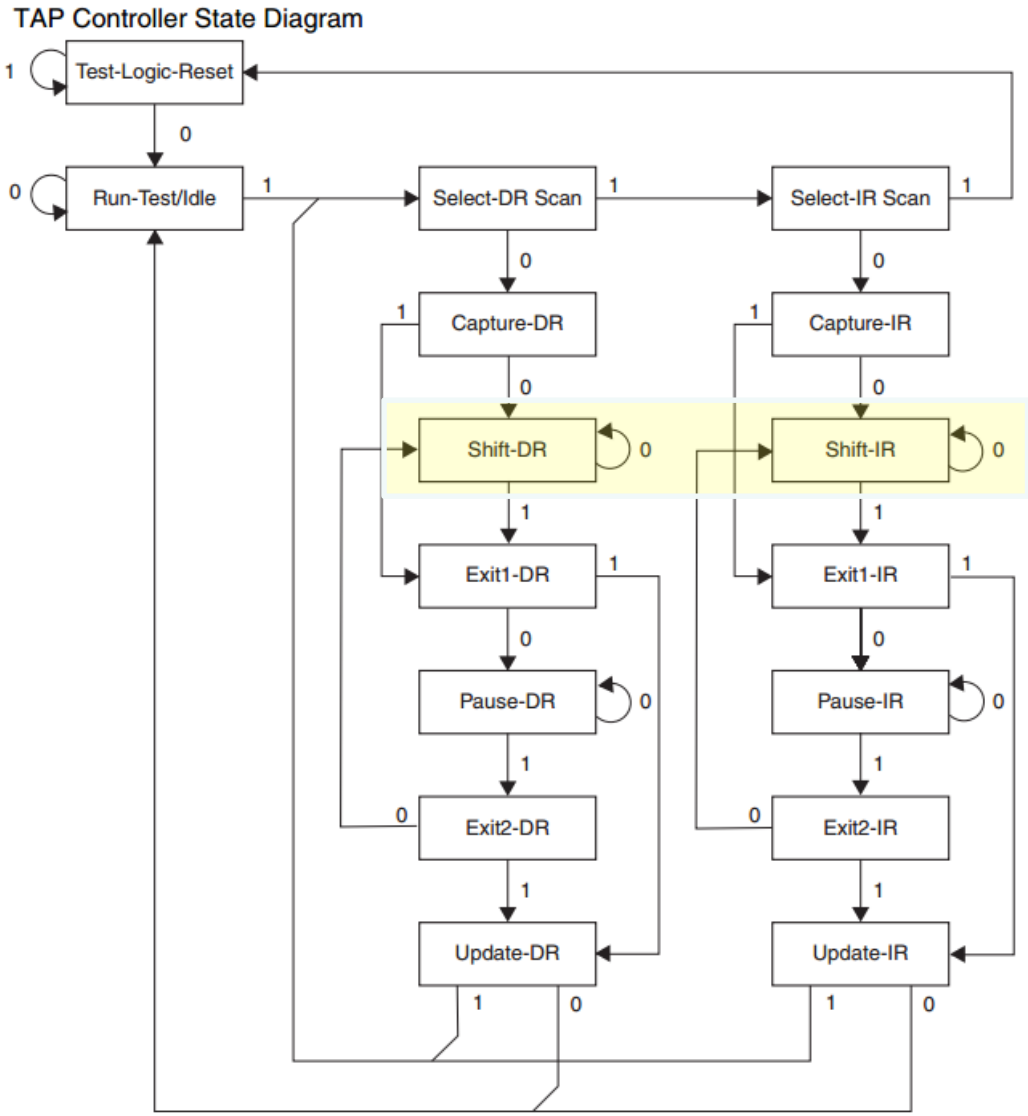

Quelle: [6]




**JTAG Pins des ATmega**

# JTAG-kompatible ICs bestehen aus

- **TAP Controller** (s. rechts)
  - Zustandsautomat, der Zustandslogik steuert.
  - Gesteuert durch **TMS** Eingang.
- In **Zustand Shift-IR**
  - Empfangene Bits an TDI werden als Instruktion ausgeführt.
  - Beispiel: Teile über TDI/TDO mit, dass Pin 3 des Ports E gelesen werden soll und in JTAG Chain eingebaut werden soll.
- In **Zustand Shift-DR**
  - Bits in TDI/TDO Chain werden als zu schreibende/lesende Daten interpretiert.
  - Beispiel: Nun legt der JTAG Controller den Zustand von Pin3 des Ports E in JTAG Chain.

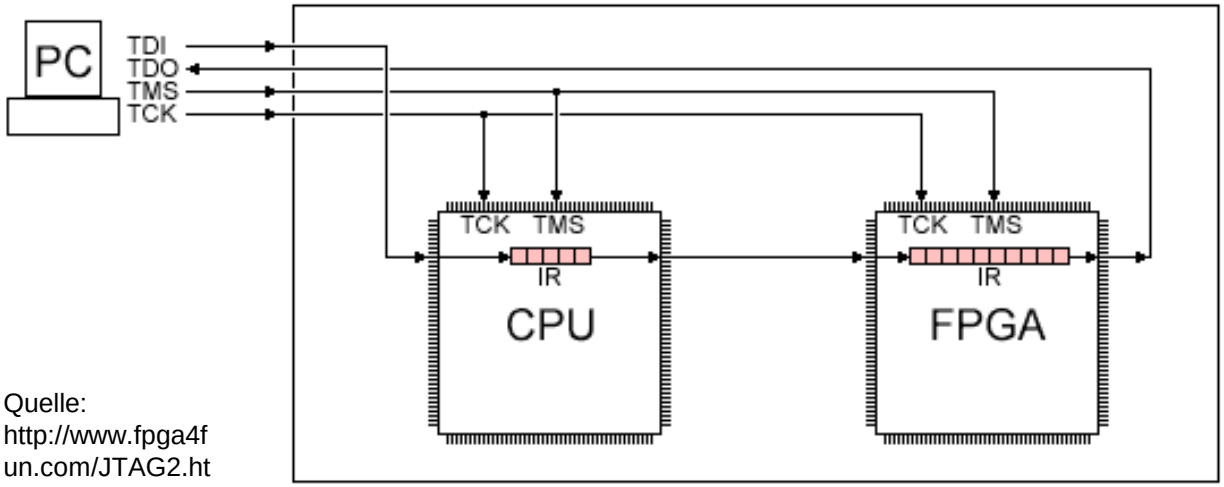



**JTAG Controller Atmega2560**

Quelle: ATMega Handbuch

# Ausführen einer typischen Aktion

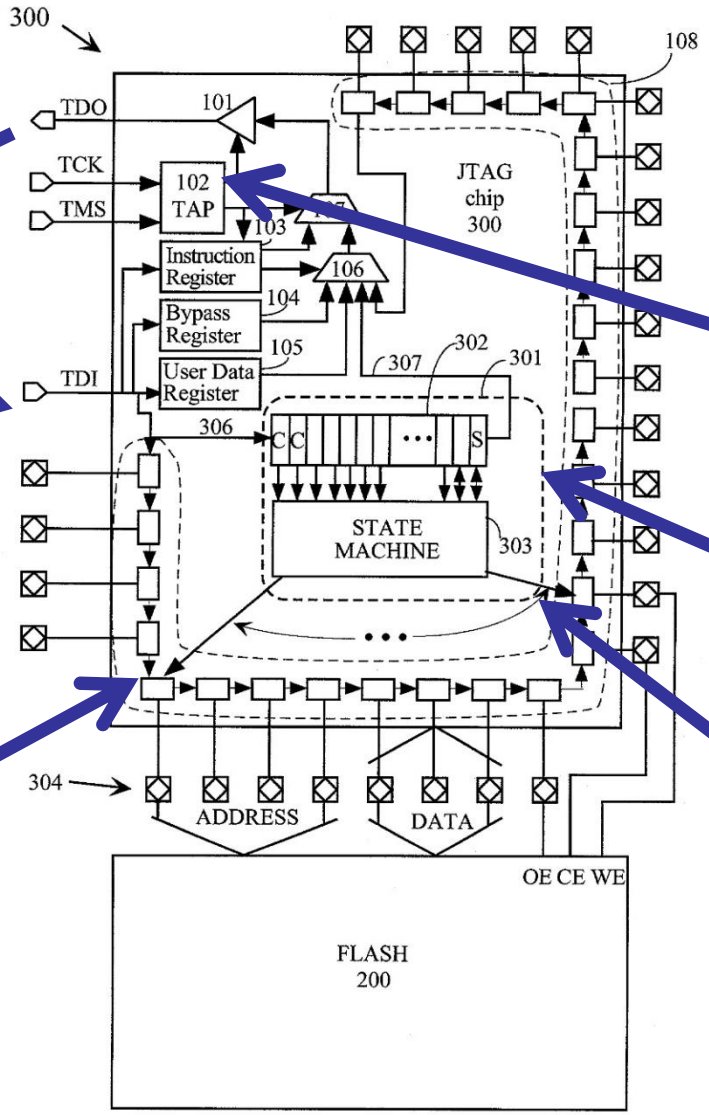
- Gehe in Zustand **Shift-IR**
- Schreibe Wert in IR-Register
  - Wert gibt an, was gemacht werden soll.
  - Semantik der Werte meist herstellerabhängig (BFEL-Files).
  - Wenige Werte standardisiert.
  - BYPASS-Kommando: Leitet TDI Eingang direkt an TDO weiter.
- Gehe dann in Zustand **Shift-DR**
  - Nun können Inhalt der ausgewählten Datenregister gelesen oder manipuliert werden.




Quelle: http://www.fpga4fun.com/JTAG2.html

**Beispiel Schreiben der IR-Register:** Zunächst müssen beide Controller in den Zustand SHIFT-IR versetzt werden (TMS: 01100). Dann sendet man nacheinander die 10 Bits für das IR des FPGAs über TDI, dann die 5 Bits für das IR der CPU über TDI.

# Detailansicht (Exkurs)




Pro Takt 1 Bit rein- und raus-schieben

(langsam!)

Verlinkung der Boundary Scan Zellen

Beispielinstruktionen:
BYPASS (TDO = TDI)
IDCODE (chip ID)
…

TAP Controller: Shift-DR oder Shift-IR

Zwischenspeichern von Bits, die die JTAG Instruktion bilden

Führe JTAG Instruktion aus

# JTAG: Zusammenfassung

- Boundary Scan: Ursprünglich Hauptanwendung von JTAG.
  - Alle (!) DR-Register werden in die JTAG Chain eingebunden.
  - Viele Messdaten können über nur 6 Leitungen ausgelesen werden.

- Weitere Anwendungen von JTAG
  - In-System Programming → nächster Abschnitt!
  - Debugging → übernächster Abschnitt!

# Inhalt

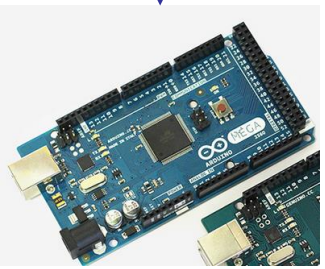
- Funktionale Sicherheit
- JTAG: Boundary Scan
- **Download von Software auf Mikrocontroller („Flashen“)**
- Debug-Methoden

# Software Download: WAS?

- Begriff: *Software Download*
  - Wie kommt die SW in den Mikrocontroller ("Flashing")?

- **WAS** muss auf den Mikrocontroller geladen werden?
  - Verschiedene **Segmente**, z.B. beim ATmega2560
    - Programmcode (Flash)
    - Statische Daten (kein eigener Speicher, in Programmcode integriert)
    - EEPROM Daten
  - Falls kein gemeinsamer Adressraum: Separate Dateien und separates Laden der Segmente
    - ATmega2560: Flash und EEPROM müssen separat geladen werden.

Segmente, Datei(en)



# Software Download: Hex-Datei

- Intel **Hex-Format**
  - Standardformat für SW Download
  - Menschenlesbar, ASCII Format.

- **Idee:** Nicht nur Sequenz der Opcodes, sondern auch
  - Checksumme → Absicherung der Datenintegrität!
  - Information über Programmgröße
  - Information wo im Zielspeicher die Opcodes gespeichert werden sollen.

| | Field | #chars | Description |
|---|---|---|---|
| 1 | Mark | 1 | a simple colon, ':' |
| 2 | Length | 2 | number of bytes in the data field |
| 3 | Offset | 4 | the address (2 byte) at which data should be programmed |
| 4 | Type | 2 | record type (00, 01, or 02) |
| 5 | Data | 0-2$k$ | 0 to $k$ bytes; this contains the opcodes |
| 6 | Checksum | 2 | sum of bytes in fields 2-5 plus checksum are zero |

'00' entspricht Datenrecord

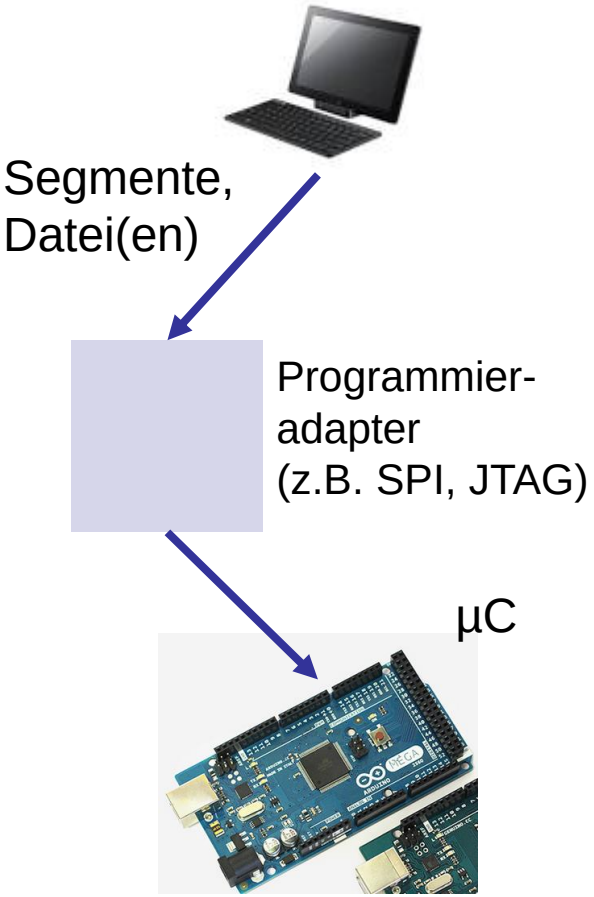
Quelle: [1]

# In-System Programming

- Mikrocontroller lassen sich direkt im *Einsatzsystem* programmieren.
  - Programmierung auch nach Einbau in Leiterplatte.

- ***Programmierung mit seriellen Schnittstellen***
  - Typisch: SPI, JTAG
  - Benötigt: **Zusatzhardware**, die zuvor erstelltes Programm/Daten direkt in internen, nichtflüchtigen Speicher (EEPROM, Flash) des µC schreibt.
  - µC erkennt Programmierung durch spezielle Signalfolgen, Timing, ungewöhnliche Spannungspegel.
    - Beispiel für SPI: Handbuch, Seite 338

- ***Programmierung mit Bootloader***
  - Typischerweise: USB
  - Benötigt keine Zusatzhardware, dafür Bootloader.
  - **Bootloader** == Programm zum Programmladen + USB Kommunikation
  - Viele µC bieten spezielle Unterstützung
    - Bootloader-Support: Handbuch, Seite 310



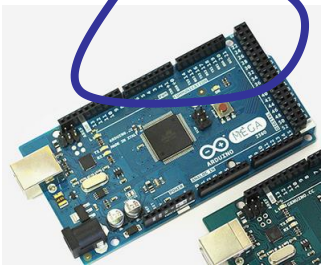

**Programmierung mit Zusatz-HW**

# Bootloader beim Arduino Mega

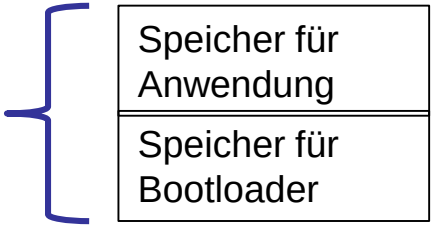
- **Funktionsweise**
  - Bootloader lauscht nach Reset, ob ein neues Programm über USB hochgeladen werden soll.
  - Falls nein: Wird das bereits vorhandene Programm gestartet.
  - https://www.arduino.cc/en/tutorial/arduinoISP
- **Vorteile**
  - USB-Schnittstelle (USB-zu Seriell Wandlung) in jedem PC vorhanden.
  - Bootloader kann geändert werden, was allerdings eine serielle Programmierung erfordert.
- **Nachteile**
  - Benötigt Speicherplatz.
  - Benötigt Unterstützung durch Mikrocontroller. Sonst leichtes versehentliches Überschreiben des Bootloaders.

Segmente, Datei(en)

Bootloader



uC unterstützt separaten Speicherbereich für Bootloader



# ATmega2560: Fuses, Signatures, Calibration

- **Fuse Bits**
  - Konfiguration zentraler Eigenschaften von Mikrocontrollern
    - Nicht möglich über Bootloader.
  - ATmega2560: Fuse OCDEN (On-Chip Debug) standardmäßig deaktiviert.

- **Signature Bytes**
  - Eindeutige ID für jeden Mikrocontrollertyp.
  - Wird z.B. überprüft bei In-System Programming (ISP)

- **Calibration Byte**
  - Kalibrierung des internen Oszillators.

- ISP, Fuses, etc. bei Atmega in AVR Studio:
  - http://www.atmel.com/webdoc/atmelstudio/atmelstudio.AVRStudio.ProgrammingDialog.html

# Inhalt

- Funktionale Sicherheit
- JTAG: Boundary Scan
- Download von Software auf Mikrocontroller
- **Debug-Methoden („Flashen“)**

# Debugging

- **Anforderungen** an Debugger
  - *Breakpoints*: Anhalten der Ausführung an gekennzeichneten Stellen.
  - Codeausführung in *Einzelschritten*
  - Bei angehaltener Ausführung: *Auswerten und Ändern des Zustands*
    - Wert von Variablen
    - Stack der Funktionsaufrufe
    - Aufrufparameter

- HW vs. SW Breakpoints
  - **SW Breakpoint**: Opcode am Ort des Breakpoints wird vorübergehend mit einer speziellen "Halte"-Instruktion ersetzt.
  - **HW Breakpoint**: Spezielles HW-Modul überwacht Adressbus und wartet auf Holen einer Instruktion von einer bestimmten Adresse.

# Debugging bei Mikrocontrollern: Hilfsmethoden

- **LEDs**, z.B.
  - *Überwachen der Programmausführung*:
    - Verschiedene LEDs bei verschiedenen Codestellen an- und ausschalten.
  - *Anzeige des Inhalts von Registern oder Variablen*
    - evtl. mehrere LEDs notwendig

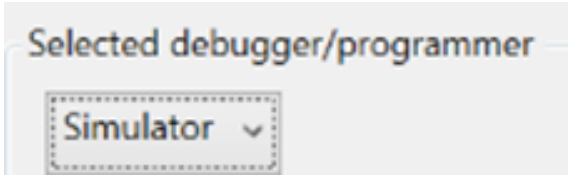
- **Taster und Schalter**
  - *Emuliere Ausführung im Einzelschrittverfahren*.
    - Baue nach jeder Instruktion eine Endlosschleife ein, die erst nach Drücken des Tasters terminiert.
  - *Break-Mechanismus*:
    - Taster löst bei Betätigung Interrupt aus; dann z.B. Zustand ausgeben
  - Verwende Schalter um digitales Eingangssignal zu erzeugen.

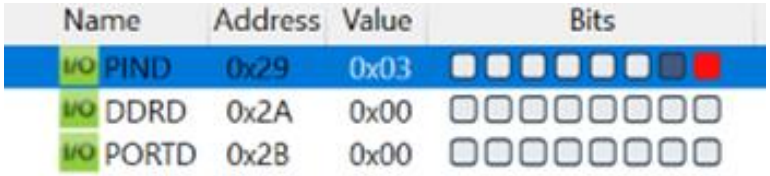
- **UART**
  - Ausgabe von menschenlesbarer Information
  - Ggfs. Berücksichtigung von Benutzereingaben
  - Problem: Extra Code innerhalb der Anwendung.

# Debugging bei Mikrocontrollern: Simulation

- ***Simulation*** des Target Controllers ***auf dem Host System***
- Wird in der Literatur oft auch als ***Instruction Set Simulator*** bezeichnet.
- Teilweise Manipulation der I/O Ports in Simulation während der Laufzeit
- Fehler an der Schnittstelle zur HW zu entdecken nicht ganz einfach!
- ATmega2560: "Simulation" innerhalb von Atmel Studio, siehe Übung und s. rechts.

**Simulation in Atmel Studio**



# Debugging bei Mikrocontrollern: HW-Debugging

- ***Debugging direkt auf der Ziel-Hardware***
  - HW-Breakpoints auf Mikrocontroller.
- Standard-Schnittstelle: **JTAG**
  - Kommunikation von PC zu Mikrocontroller über Zusatz-HW / JTAG-Adapter.
  - Kontrolle des Debuggings dann durch PC.
  - *DebugWire*: Bei kleineren AVRs
- Häufig werden nur die Register in JTAG Chain eingebunden, die für Debugging wichtig sind.
- Leider kein Hersteller-übergreifender Standard.
- ATmega2560:
  - On-Chip Debug System über JTAG
  - Fuse OCDEN


Atmel-ICE
JTAG Debugger

# Quellenverzeichnis


[1] G. Gridling und B. Weiss. *Introduction to Microcontrollers*, Version 1.4, 26. Februar 2007, verfügbar online: https://ti.tuwien.ac.at/ecs/teaching/courses/mclu/theory-material/Microcontroller.pdf (abgerufen am 08.03.2018)

[2] http://www.fpga4fun.com/JTAG1.html (abgerufen am 13.06.2019)

[3] https://hci.cs.siue.edu/NSF/Files/Semester/Week13-2/PPT-Text/images/Image3.png (abgerufen am 02.06.2017)

[4] http://www.atmel.com/webdoc/atmelice/atmelice.using_ocd_physical_jtag.html (abgerufen am 02.06.2017)

[5] Von Vindicator - Eigenes Werk., CC BY 2.5, https://commons.wikimedia.org/w/index.php?curid=838166

[6] https://tu-dresden.de/die_tu_dresden/fakultaeten/fakultaet_informatik/tei/vlsi/lehre/vortr_pro_haupt/hs_ws_0708/jtag-schnittstelle.pdf (abgerufen am 02.06.2017)