

E Mikrocontroller-Programmierung

E.1 Überblick

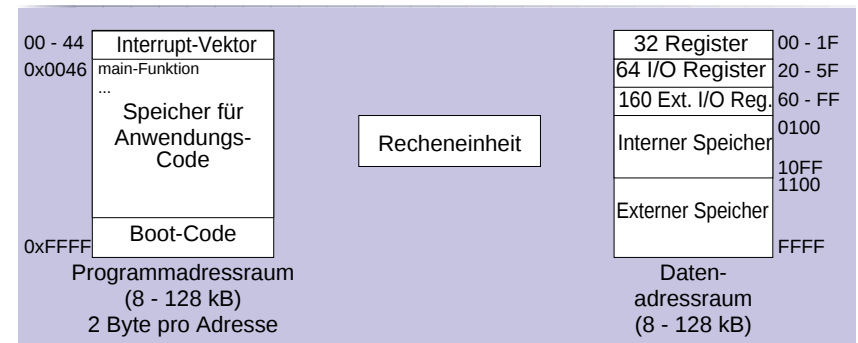
- Mikrocontroller-Umgebung
 - Prozessor am Beispiel AVR-Mikrocontroller
 - Speicher
 - Peripherie
- Programmausführung
 - Programm laden
 - starten
 - Fehler zur Laufzeit
- Interrupts
 - Grundkonzepte
 - Nebenläufigkeit
 - Umgang mit Nebenläufigkeit

1 Mikrocontroller-Umgebung

- Programm läuft "nackt" auf der Hardware
 - ➔ Compiler und Binder müssen ein vollständiges Programm erzeugen
 - keine Betriebssystemunterstützung zur Laufzeit
 - ◆ Funktionalität muss entweder vom Anwender programmiert werden oder in Form von Funktionsbibliotheken zum Programm dazugebunden werden
 - ◆ Umgang mit "lästigen Programmierdetails" (z. B. bestimmte Bits setzen) wird durch Makros erleichtert
- Es wird genau ein Programm ausgeführt
 - Programm kann zur Laufzeit "niemanden stören"
 - Fehler betreffen nur das Programm selbst
 - keine Schutzmechanismen notwendig
 - ➔ ABER: Fehler ohne direkte Auswirkung werden leichter übersehen

E.2 Beispiel: AVR-Mikrocontroller (ATmega-Serie)

1 Architektur



- Getrennter Speicher für Programm (Flash-Speicher) und Daten (SRAM)
- Register des Prozessors und Register für Ein-/Ausgabe-Schnittstellen sind in Adressbereich des Datenspeichers eingebettet

1 Architektur(2)

- Peripherie-Bausteine werden über I/O-Register angesprochen bzw. gesteuert (Befehle werden in den Bits eines I/O-Registers kodiert)
 - mehrere Timer (Zähler, deren Geschwindigkeit einstellbar ist und die bei einem bestimmten Wert einen Interrupt auslösen)
 - Ports (Gruppen von jeweils 8 Anschlüssen, die auf 0V oder V_{CC} gesetzt, bzw. deren Zustand abgefragt werden kann)
 - Output Compare Modulator (OCM) (zur Pulsweitenmodulation)
 - Serial Peripheral Interface (SPI)
 - Synchrone/Asynchrone serielle Schnittstelle (USART)
 - Analog-Comparator
 - A/D-Wandler
 - EEPROM (zur Speicherung von Konfigurationsdaten)

2 In Speicher eingebettete Register (memory mapped registers)

- Je nach Prozessor sind Zugriffe auf I/O-Register auf zwei Arten realisiert:
 - ◆ spezielle Befehle (z. B. in, out bei x86)
 - ◆ in den Adressraum eingebettet, Zugriff mittels Speicheroperationen
- Bei den meisten Mikrocontrollern sind die Register in den Speicheradressraum eingebettet
- Zugriffe auf die entsprechende Adresse werden auf das entsprechende Register umgeleitet
- ➔ sehr einfacher und komfortabler Zugriff

3 I/O-Ports

- Ein I/O-Port ist eine Gruppe von meist 8 Anschluss-Pins und dient zum Anschluss von digitalen Peripheriegeräten
- Die Pins können entweder als Eingang oder als Ausgang dienen
 - ◆ als Ausgang konfiguriert, kann man festlegen, ob sie eine logische "1" oder eine logische "0" darstellen sollen
 - Ausgang wird dann entsprechend auf V_{CC} oder GND gesetzt
 - ◆ als Eingang konfiguriert, kann man den Zustand abfragen
- Manche I/O Pins können dazu genutzt werden, einen Interrupt (IRQ = Interrupt Request) auszulösen (externe Interrupt-Quelle)
- Die meisten Pins können alternativ eine Spezialfunktion übernehmen, da sie einem integrierten Gerät als Ein- oder Ausgabe dienen können
 - ◆ z. B. dienen die Pins 0 und 1 von Port E (ATmega128) entweder als allgemeine I/O-Ports oder als Rx/D und Tx/D der seriellen Schnittstelle

4 Programme laden

- generell bei Mikrocontrollern mehrere Möglichkeiten
- ▲ Programm ist schon da (ROM)
- ▲ Bootloader-Programm ist da, liest Anwendung über serielle Schnittstelle ein und speichert sie im Programmspeicher ab
- ▲ spezielle Hardware-Schnittstelle
 - "jemand anderes" kann auf Speicher zugreifen
 - Beispiel: JTAG
 - spezielle Hardware-Komponente im AVR-Chip, die Zugriff auf die Speicher hat und mit der man über spezielle Pins kommunizieren kann

5 Programm starten

- Reset bewirkt Ausführung des Befehls an Adresse 0x0000
 - dort steht ein Sprungbefehl auf die Speicheradresse einer start-Funktion, die nach einer Initialisierungsphase die main-Funktion aufruft
 - alternativ: Sprungbefehl auf Adresse des Bootloader-Programms, Bootloader lädt Anwendung, initialisiert die Umgebung und springt dann auf main-Adresse

6 Fehler zur Laufzeit

- Zugriff auf ungültige Adresse
 - ◆ es passiert nichts:
 - Schreiben geht in's Leere
 - Lesen ergibt zufälliges Ergebnis
- ungültige Operation auf nur-lesbare / nur-schreibbare Register/Speicher
 - hat keine Auswirkung

1 Motivation

- An einer Peripherie-Schnittstelle tritt ein Ereignis auf
 - Spannung wird angelegt
 - Zähler ist abgelaufen
 - Gerät hat Aufgabe erledigt (z. B. serielle Schnittstelle hat Byte übertragen, A/D-Wandler hat neuen Wert vorliegen)
 - Gerät hat Daten für die Anwendung bereit stehen (z. B. serielle Schnittstelle hat Byte empfangen)
- ? wie bekommt das Programm das mit?
 - Zustand der Schnittstelle regelmäßig überprüfen (= **Polling**)
 - Schnittstelle meldet sich von sich aus beim Prozessor und unterbricht den Programmablauf (= **Interrupt**)

1 Motivation (2)

- Polling vs. Interrupts: Vor und Nachteile
 - ◆ Polling
 - + Pollen erfolgt **synchron** zum Programmablauf, Programm ist in dem Moment auf das Ereignis vorbereitet
 - Interrupts unterbrechen den Programmablauf irgendwo (**asynchron**), sie könnten in dem Augenblick stören
 - ↳ durch die Interrupt-Bearbeitung entsteht **Nebenläufigkeit**
 - Pollen erfolgt explizit im Programm und meistens umsonst — Rechenzeit wird verschwendet

1 Motivation (3)

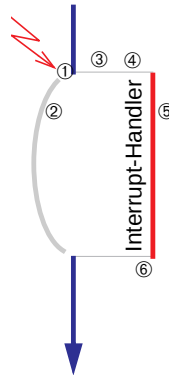
- Polling vs. Interrupts: Vor und Nachteile
 - ◆ Interrupts
 - + Interrupts melden sich nur, wenn tatsächlich etwas zu erledigen ist
 - + Interrupt-Bearbeitung ist in einer Funktion kompakt zusammengefasst
 - Polling-Funktionalität ist in den normalen Programmablauf eingestreut — und hat mit der "eigentlichen" Funktionalität dort meist nichts zu tun

2 Implementierung

- typischerweise mehrere Interrupt-Quellen
- Interrupt-Vektor
 - ◆ Speicherbereich (Tabelle), der für jeden Interrupt Informationen zur Bearbeitung enthält
 - Maschinenbefehl (typischerweise ein Sprungbefehl auf eine Adresse, an der eine Bearbeitungsfunktion steht) oder
 - Adresse einer Funktion (**Interrupt-Handler**)
 - ◆ feste Position im Speicher — ist im Prozessorhandbuch nachzulesen
- Maskieren von Interrupts
 - Bit im Prozessor-Statusregister schaltet den Empfang aller Interrupts ab
 - zwischenzeitlich eintreffende Interrupts werden gepuffert (nur einer!)
 - die Erzeugung einzelner Interrupts kann am jeweiligen Gerät unterbunden werden

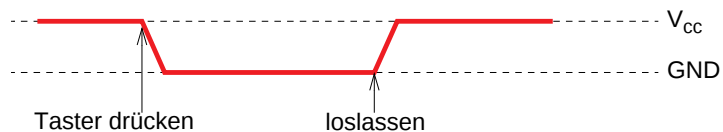
3 Ablauf

- ① Gerät löst Interrupt aus, Ablauf des Anwendungsprogramms wird unmittelbar unterbrochen
- ② weitere Interrupts werden deaktiviert
- ③ aktuelle Position im Programm und Registerinhalte werden im Datenspeicher gesichert
- ④ Eintrag im Interrupt-Vektor ermitteln
- ⑤ Befehl wird ausgeführt bzw. Funktion aufrufen
- ⑥ am Ende der Bearbeitungsfunktion bewirkt ein Befehl "Return from Interrupt" die Fortsetzung des Anwendungsprogramms und die Reaktivierung der Interrupts



4 Pegel- und Flanken-gesteuerte Interrupts

■ Beispiel: Signal eines Tasters



■ Flanken-gesteuert

- Interrupt wird durch die Flanke (= Wechsel des Pegels) ausgelöst
- welche Flanke einen Interrupt auslöst kann bei manchen Prozessoren konfiguriert werden

■ Pegel-gesteuert

- solange ein bestimmter Pegel anliegt (hier Pegel = GND) wird immer wieder ein Interrupt ausgelöst