Überblick

Praktikum Bussysteme

- Motivation
- Entwicklungsumgebung The ToolChain
- Multitasking in Mikrocontrollern
- Übung
- Fragen & Antworten
- Hausaufgabe
- Eingangstest

Quellen

- → http://hlb-labor.de
- → http://thetoolchain.com
- → http://thetoolchain.com/documentation/

Motivation

Chrysler Grand Cherokee: Software-Fehler kann ungewollt Gangschaltung verstellen

Sonntag, 12.05.2013 – 13:18 Uhr

Chicago - Chrysler ruft weltweit fast eine halbe Million Geländewagen wegen notwendiger Software-Aktualisierungen zurück. Etwa 469.000 Autos seien betroffen, teilte die Fiat-Tochter am Samstag mit. Bei einigen Fahrzeugen habe eine fehlerhafte Programmierung dazu geführt, dass die Gangschaltung ungewollt verstellt wurde.

Motivation

- Geräte an Bus zu stöpseln ist einfach
- Kommunikationssoftware is komplex
- Software-Entwicklungsaufwand oft unterschätzt
- Wer schreibt diese Software?

The ToolChain

- Sammlung von Skripten und Quelltexten
- Automatischer Download zusätzlicher Software
- Fremde Quelltexte speziell angepasst
- Compiler Programme
- Programmier- und Debugwerkzeuge
- Umfangreiche Softwarebibliothek
  - Universelle Gerätetreiber
  - Multitasking Unterstützung
  - Flexible Konfiguration
  - Ausführlich dokumentiert

Features The ToolChain

- Offene Software für offene Betriebsysteme (Linux OS)
- Automatischer Download und Installation
- FreeRTOS Multitasking Scheduler
- STM-StdPeripheralsLibrary
- Moderne Compiler Suite ARM GCC
- OpenOCD for In Circuit Programming/ Debugging
- Zusätzliche Bibliotheken (GFX, GPIO, UART, SPI, CAN, I²C, …)
- Fertige Konfigurationen für verschiedene Prototypenboards
- Flexible Konfiguration
  → Derselber Quelltext übersetzt auf verschiedenen Boards
- Viele lauffähige Code Beispiele

Multitasking

in

Eingebetteten Systemen

Multitasking in ES

- Was ist Singletasking?
- Was ist Multitasking?
- Warum Multitasking?
- Verschiedene Ansätze
- Echtzeitbetriebssysteme
- FreeRTOS
- Synchronisation
- Ein Beispielprojekt
- Debugging von multitasking Software

Was ist Singletasking?

- Immer nur ein Ziel auf einmal verfolgen
- Der effizienteste Weg um ein Problem zu lösen
- Auf jeden Algorithmus anwendbar
- Kein extra Verwaltungsaufwand
- Keine Synchronisation erforderlich
- Einfach zu programmieren
- Wiederholtes aktives Abfragen von Eingängen
- Keine Unterbrechungen

Singletasking Beispiel

```
while (1) {
    if ( Byte = receiveByte() ) { // blocks!
        Buffer[Index++] = Byte;
        if ( Index >= MessageSize) {
        M_t* M = (M_t*) Buffer;
            switch (M->Type) {
                case mt_CommandA: ...
                case mt_CommandB: …
                default: break;
            }
            Index = 0;
        }
    }
}
```

Was ist Multitasking?

- Verfolgung mehrerer Ziele
- Häufige Kontextumschaltung
- Extra Verwaltungsaufwand
- Synchronisation erforderlich
- Unterbrechungen benötigt
- Schwierig zu implementieren
- Zusätzliche Verzögerungen
- Erhöhter Speicherbedarf
- Umsetzung für viele Algorithmen schwierig

Multitasking Beispiel

```
main() {
    int Queue = ttc_queue_create(...);
    xTaskCreate(Receive, Queue, …);
    xTaskCreate(Process, Queue, …);
}

void Process(int Q) {
  while (1) {
    M = (M_t*) ttc_queue_pop_front(Q);
    if (M) {
      switch (M->Type) {
        case mt_CmdA: ...
        case mt_CmdB: ...
        default: break;
      } } } }
```

```
void Receive(int Q) {
  char Buffer[10][100];
  int Index = 0;
  while (1) {
    char* Writer = &(Buffer[Index,0]);
    int Remaining = MessageSize;
    while (Remaining > 0) {
      if ( Byte = readByte() ) { // sleeps!
        *Writer++ = Byte; Remaining--;
    }
    ttc_queue_push_back(Q, &(Buffer[Index,0]););
    Index++;
    if (Index > 99)  Index = 0;
  }
}
```

Warum dann Multitasking?

- Funktionen mehrfach startbar
- Langsame Kommunikation einfacher
- Beschleunigung durch Mehrkern CPUs
- Nur 1 zentraler Zeitgeber benötigt
- Kurze Interrupt Services
- Weniger Globale Variablen

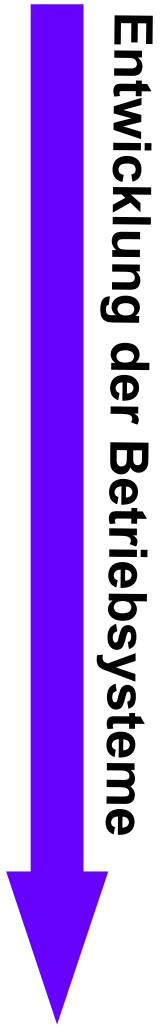
Kein Leben ohne Multitasking!

- Jedes Eingebettete System braucht MT
- MT oft durch Interrupt implementiert
  - Komplexe Interrupt Service Routinen
  - Datenaustausch durch globale Variablen
  - Schwierig zu debuggen
- Typischer Ansatz: Super-Loop
  - Ruft Zustandsautomaten periodisch auf
  - Ähnlich einem Task Scheduler

=> Warum nicht gleich richtiges Multitasking?

Verschiedene Ansätze

- Multiprogramming
  - Historischer Ansatz für Kommunikationsschnittstellen
  - Realisiert durch Terminal Stay Ready (TSR) unter MSDOS
- Kooperatives Multitasking
  - Zentraler Scheduler verwaltet Prozesse
  - Jeder Process gibt CPU an andere Prozesse weiter
  - Nachteil: Einzelner Prozess kann gesamtes System blockieren
- Preemptives Multitasking
  - Scheduler unterbricht aktuellen Prozess periodisch
  - Benötigt zentralen Zeitgeber
- Preemptible Multitasking
  - Prozesse mit hoher Priorität können andere Prozesse unterbrechen (OS/2, Linux, FreeRTOS)
  - Erlaubt kürzere Antwortzeiten



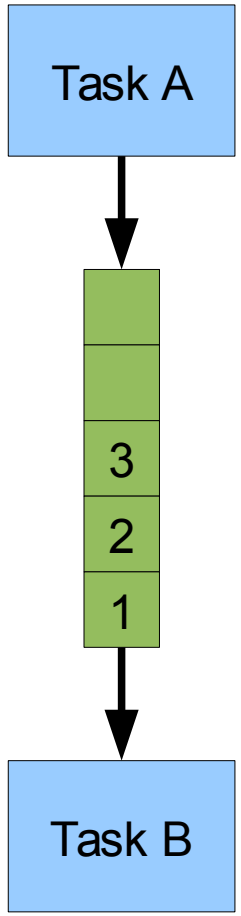
FreeRTOS

- Multitasking Scheduler
  - Premptibles Multitasking
  - Tasks hoher Priorität unterbrechen andere
- Inter Task Kommunikation
  - Semaphoren
  - Queues
  - Mutexes
- Speziell entwickelt für Eingebettete Systeme
- Open Source
- FreeWare mit Kommerziellem Support
- Auf verschiedene µC Architekturen portiert

  → http://www.freertos.org/

Queues

- Grundlage von Task Kommunikation
- Nachricht verschicken
  - Task → Task
  - Interrupt Service Routine → Task
- Call by Value
- Aus leere Queue lesen
  - Funktion wartet bis Queue gefüllt
  - Task schläft ohne CPU-Zeit



ttc_queue

- Drei Arten von Queues in The ToolChain
  - Generische Queues
    ttc_queue_*()
  - Byte Queues
    ttc_queue_byte_*()
  - Pointer Queues
    ttc_queue_pointer_*()

Generische Queues

- Speichert Elemente beliebiger Größe
- Elementgröße bei Erstellung festgelegt
- Grundfunktionen (→ ttc_queue.c/ .h)
  - ttc_queue_create()
  - ttc_queue_push_back()
  - ttc_queue_pop_front()

Byte Queues

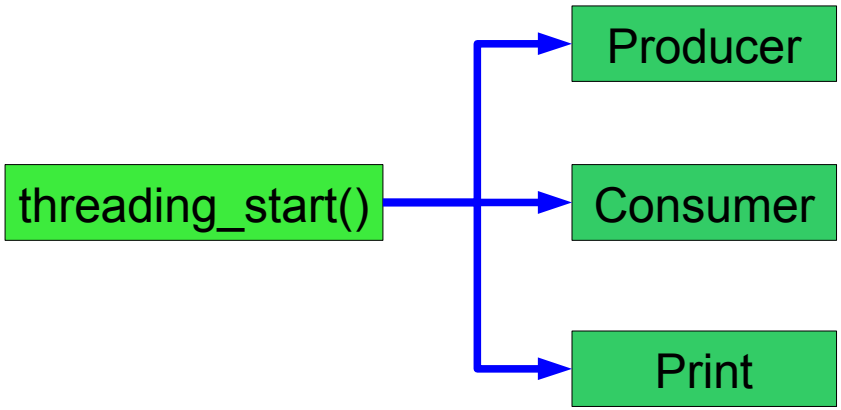
- Transportiert einzelne Bytes
- Vorteile gegenüber Generischen Queues
  - Weniger Speicherbedarf
  - Schnellere PUSH und POP Operationen
- Grundfunktionen (→ ttc_queue.c/ .h)
  - ttc_queue_byte_create()
  - ttc_queue_byte_push_back()
  - ttc_queue_byte_pop_front()

Pointer Queues

- Jedes Element speichert einen Zeiger
- Vorteile gegenüber Generischen Queues
  - Weniger Speicherbedarf
  - Schnellere PUSH und POP Operationen
- Grundfunktionen (→ ttc_queue.c/ .h)
  - ttc_queue_pointer_create()
  - ttc_queue_pointer_push_back()
  - ttc_queue_pointer_pop_front()

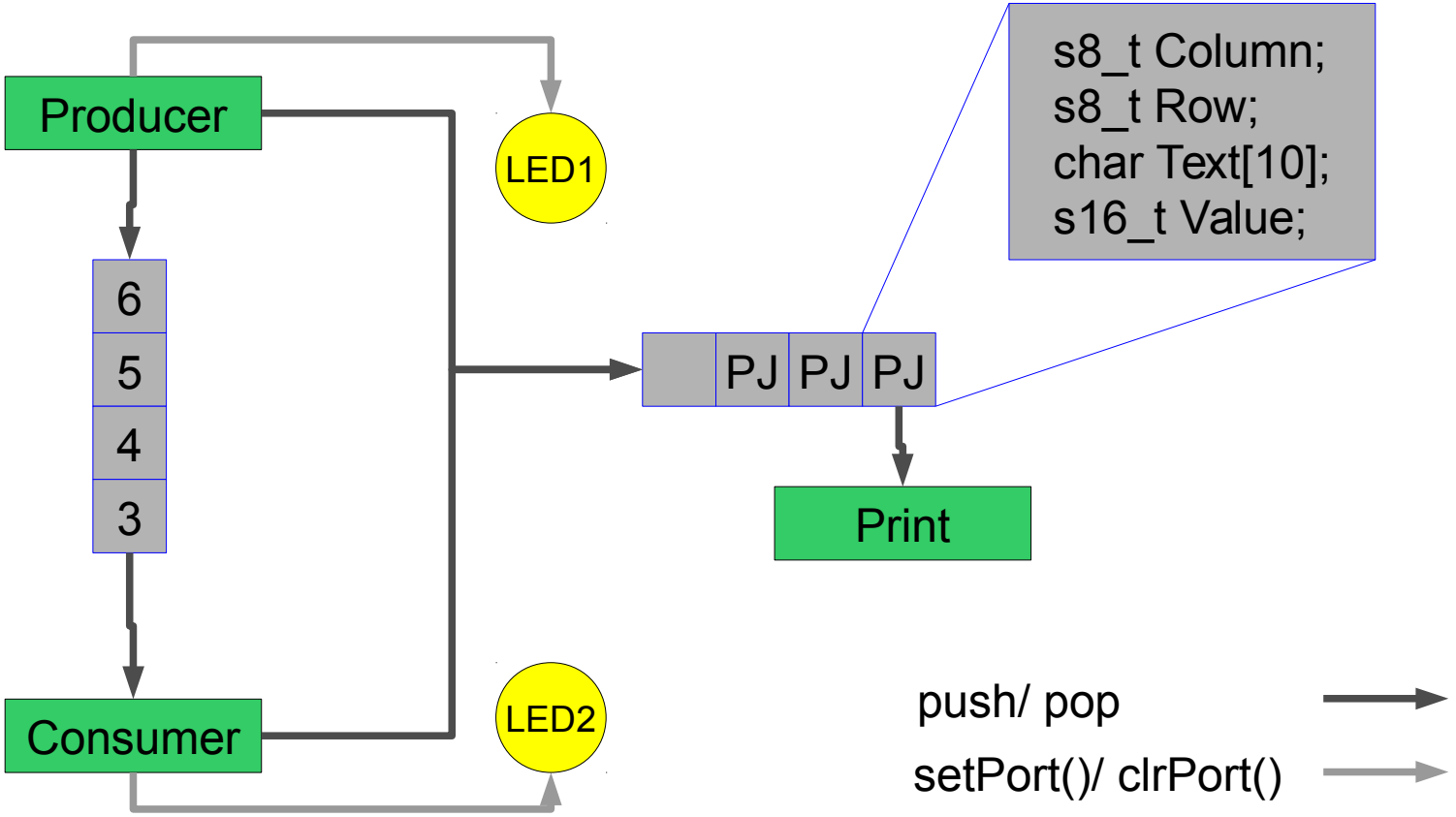
Multithreading mit Queues

- Tasks starten



Multithreading mit Queues

- Kommunikation durch Queues



Hausaufgaben

- Quelle: http://thetoolchain.com/documentation/

1) TheToolChain_Schnelleinstieg.pdf
   - Handschriftliche Ausarbeitung
   - Zum ersten Praktikum mitbringen

   → **Voraussetzung zur Teilnahme am Praktikum!**

2) C_fuer_Microcontroller.pdf
   - Wiederholung bereits behandelter Inhalte
   - Durchspielen mittels C-Compiler empfohlen (jedes Betriebsystem)
   - Überprüfung der Inhalte durch einen Eingangstest im 1. Praktikum

   → Bei Nichtbestehen 1 Wiederholungstermin

   → **Bestehen des Eingangstests ist Teilnahmevoraussetzung!**