Stefan, Pfeiffer
CST Institut für Informatik

Start the 

Fahrplan

- Embedded Systems / Microcontroller
- Embedded Development
- „Hallo Welt !“
- „Newlib“
- Speicher: SRam, Flash, Stack, Heap, Statisch
- Multithreading IPC

Embedded Systems / Microcontroller

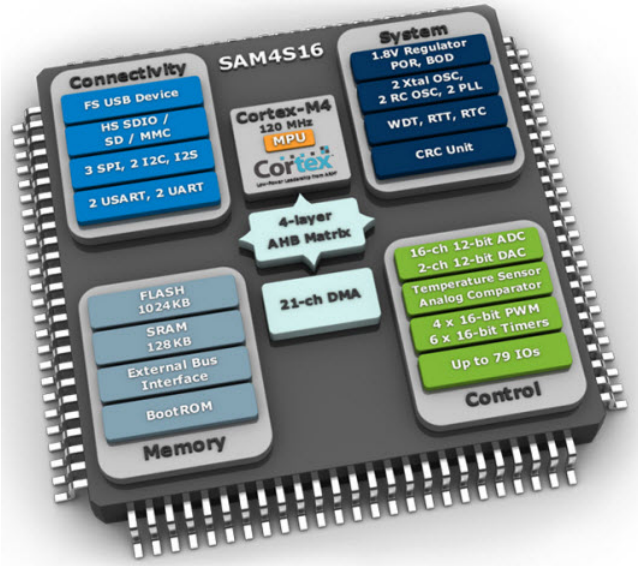
Mikroprozessor vs. Mikrocontroller

- Mikrocontroller ist ein Mikroprozessor „plus“
- z.B. GPIO, SPI, I2C, I2S, USART, USB, MCI, Ethernet, CAN, DCMI, FSMC

Unterschiede in der Benutzung

- Keinen Bildschirm, keine Tastatur, keine Massenspeicher
- D.h. keine lokale Ein und Ausgabe
- Umleitung von STDIO über UART oder USB

oder Funktechnologien oder sehr simple HID

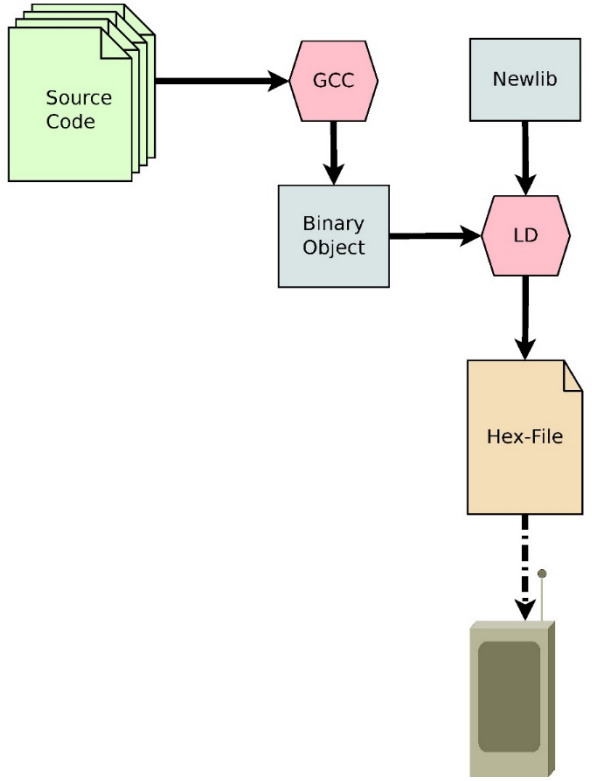


Embedded Development

- Entwickeln auf einem anderen System
  - Crosscompiler
  - Binary „Flashen“
  - Statisch linken

- Die einzige API: Das Mikrocontrollerhandbuch

Oder



RIOT

http://riot-os.org/api/

„Hallo Welt !“

RIOT Sourcecode:

git clone https://github.com/emmanuelsearch/RIOT.git

RIOT übersetzen:

cd RIOT/examples

make

RIOT starten:

make term

NEWLIB

http://sourceware.org/newlib/docs.html

- Standard ANSI C Bibliothek
- Enthält Funktionen für:
  - Ein- und Ausgabe
    `puts() printf() fprintf()`
  - Verarbeitung von Zeichenketten
    `strlen() strcat() strcmp()`
  - Mathematische Operationen
    `sqrt() pow() sin()`
  - Speicherverwaltung
    `malloc() free()`

Speicher

- Linker sammelt alle binären Objekte ein und baut ein großes Binary
- Unterscheidung zwischen verschiedenen Segmenten

```
SECTIONS
{       .text:
        {       // code und const -> flash Speicher
        }
        .data:
        {       // initialisierte Variablen -> RAM
        }
        .bss:
        {       // uninit data -> RAM
        }
}
```

Speicher

- In welchem Speicher landen Variablen in Funktionen ?

```
void main(void)
{
        unsigned int foo = 0; -> Stack -> Sram
        char* fuu = malloc(128); -> SRam
}
```

- Wieviel Speicher gibt es ?

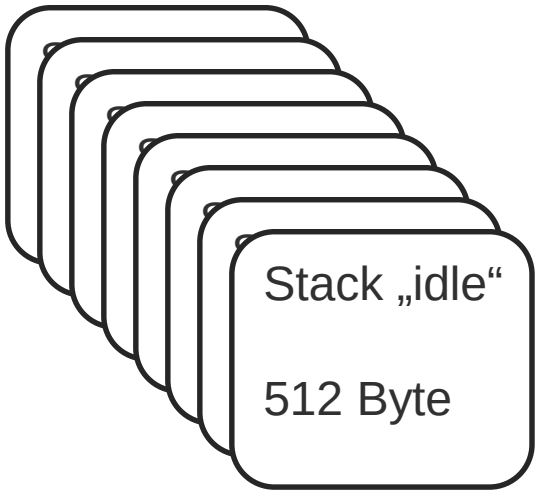
z.B. STM32f4 (arm cortex m4)

FLASH 1 MByte

.text

Sram 128 kByte

.data
.bss
.stacks
.heap



Multithreading IPC

Threads

- Prioritätsbasiert (31(idle) … 1(höchste))
- Tickless Scheduler -> Thread läuft solange:
    1. Bis er die Kontrolle abgibt -> `thread_yield()`
    2. Bis er von einem Interrupt unterbrochen wird

Threading API:

```
thread_create(…)
thread_sleep(…)
thread_wakeup(…)
thread_yield(…)
```

IPC API:

```
msg_send(…)
msg_receive(…)
msg_send_recieve(…)
msg_reply(…)
```

Links

RIOT im Internet

http://riot-os.org

RIOT on Github

https://github.com/emmanuelsearch/RIOT.git

RIOT Developers Mailingliste

devel@riot-os.org

RIOT API Dokumentation

http://riot-os.org/api

RIOT Development Image

http://download.riot-os.org/RIOT-dev.vdi.zip

Newlib Dokumentation

http://sourceware.org/newlib/docs.html