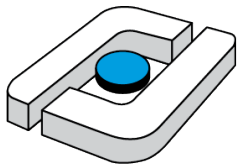
Entwicklung eines Treibers und Toolchain zur Administration eines Embedded ZPU-Systems

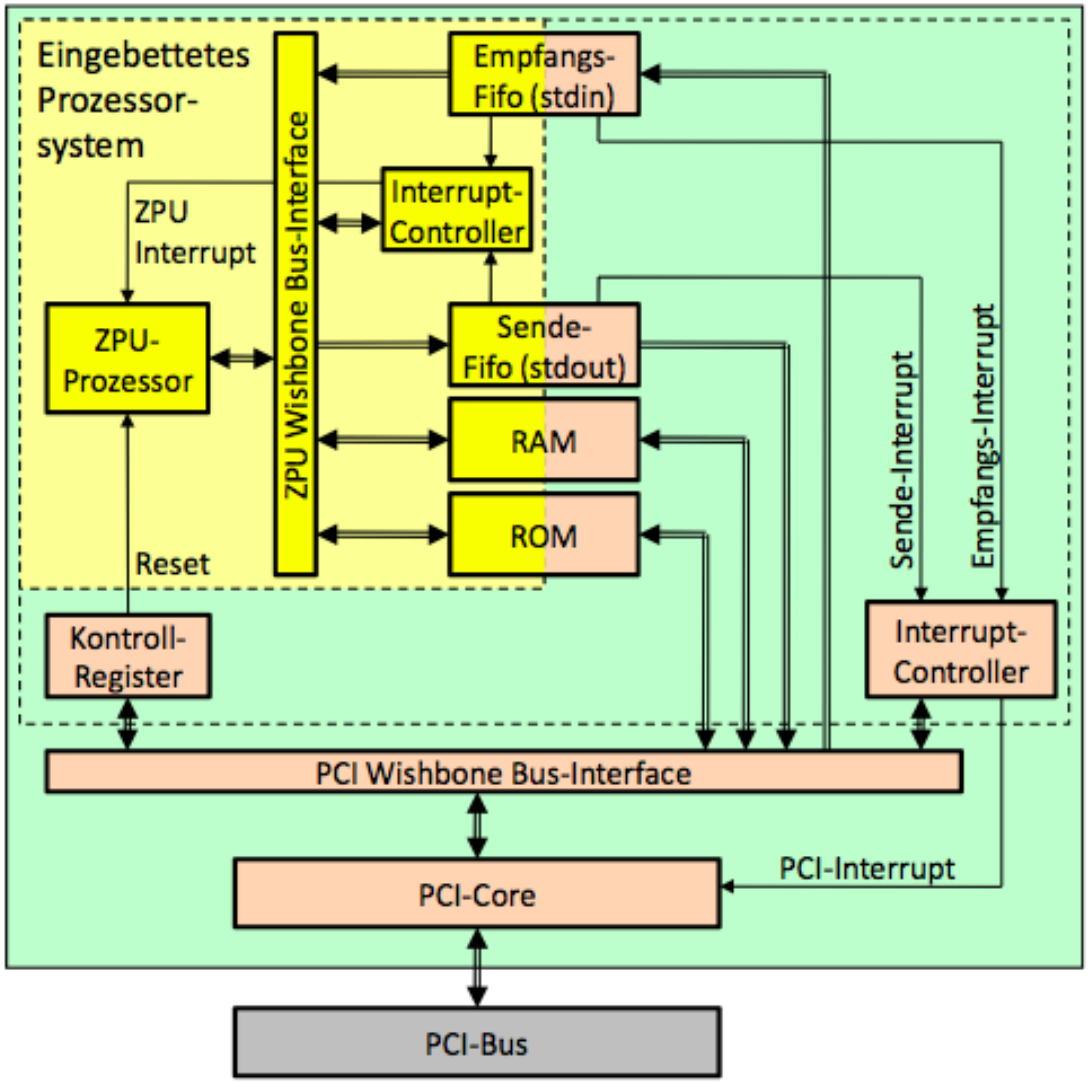


Hochschule Osnabrück
University of Applied Sciences

Oliver Erxleben
Martin Helmich

27. Februar 2013

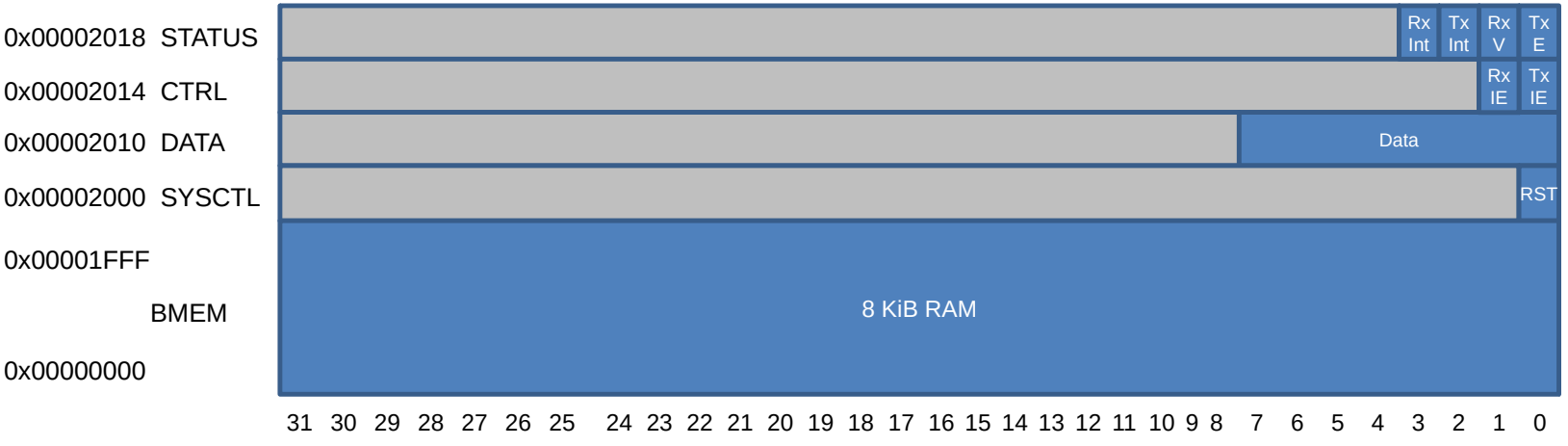
Aufbau der Hardware



Aufgabenstellung

- **Zurücksetzen** des Prozessorsystems per `ioctl`-Befehl
- Einblenden des RAMs in den Userspace-Speicher per `mmap`, um **neue Software** in das System laden zu können.
- Senden von Eingaben an den **stdin**-Buffer und Lesen von Ausgaben aus dem **stdout**-Buffer per `read`- und `write`-Methoden.

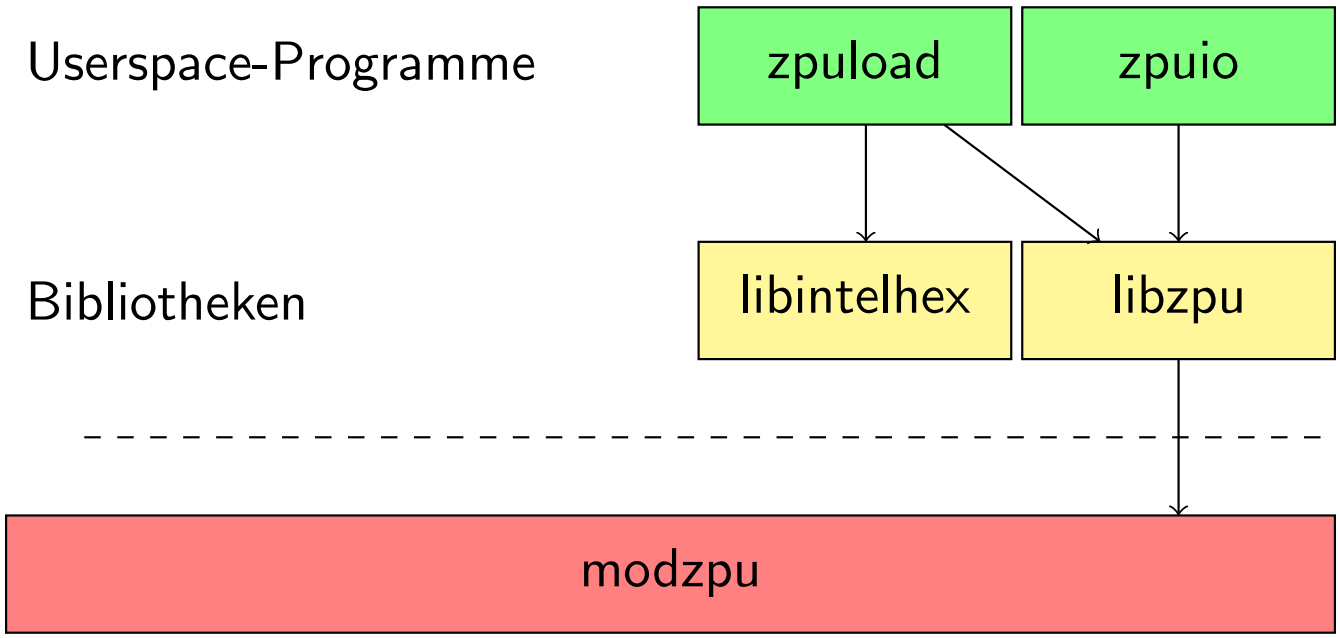
Register der Hardware



Grundideen

- Aufteilung des Projekts in **unabhängige Module**, Bibliotheken und Anwendungen.
- Parsing der Intel HEX-Dateien und Kommunikation mit dem ZPU-Treiber in jeweils **eigene Bibliotheken**, die dann aus Anwendungen heraus genutzt werden.
- Die erstellten Bibliotheken sind **anwendungsunabhängig** und können später auch in Drittanwendungen verwendet werden.

Architekturentwurf (1)



Architekturentwurf (2)

Kernel-Ebene:

- Das Kernel-Modul `zpu` stellt `mmap`-, `read`-, `write`- und `ioctl`-**Systemaufrufe** zur Verfügung.

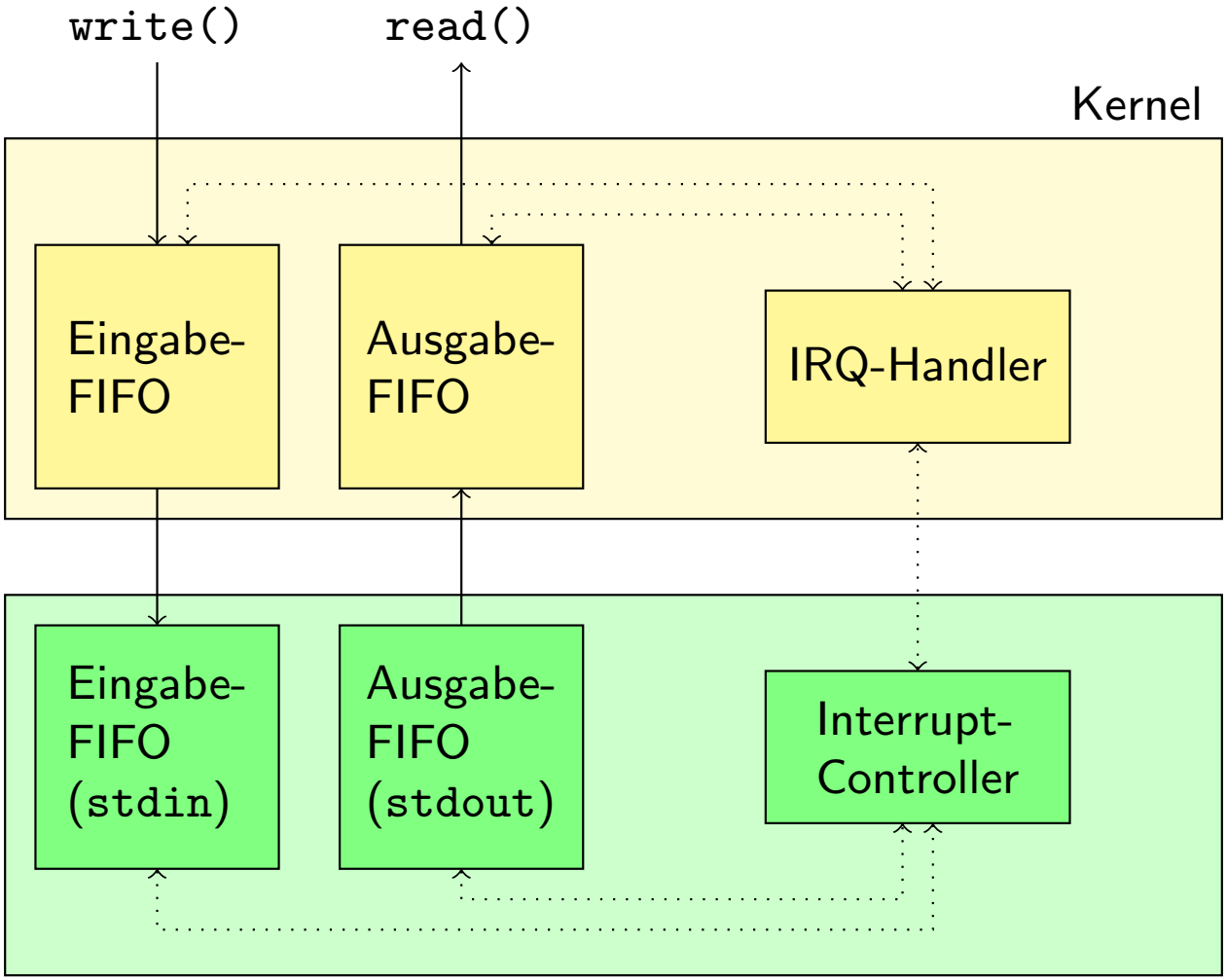
Userspace-Bibliotheken:

- Die Bibliothek `libzpu` stellt Funktionen zur Verfügung, um in Anwenderprogrammen vom Zugriff auf diese Systemaufrufe zu **abstrahieren**.
- Die Bibliothek `libintelhex` stellt Funktionen zum **Parsen von Intel HEX-Dateien** zur Verfügung.

Userspace-Programme:

- Das Programm `zpuio` kann **Eingaben** an die ZPU leiten und deren **Ausgabe** verarbeiten.
- Das Programm `zpuload` kann Intel HEX-Dateien lesen und **in die ZPU laden**.

Das zpu-Modul: Ein- und Ausgabe



Das zpu-Modul: Ein- und Ausgabe

- Die `write`-Methode schreibt Daten in den Puffer (FIFO) des Moduls.
- Anschließend gibt sie den Empfangs-Interrupt (aus ZPU-Sicht) frei und legt sich (ggf.) schlafen.
- Der Empfangs-IR-Handler schreibt so viele Daten wie möglich in den `stdin`-Puffer der ZPU und weckt eventuell schlafende Schreibprozesse auf.
- Die `read`-Methode liest Daten aus dem Puffer (FIFO) des Moduls und gibt ggf. den Sende-IR-Handler wieder frei.
- Sollte der Puffer leer sein, legt sich die `read`-Methode (ggf.) schlafen.
- Der Sende-IR-Handler (aus ZPU-Sicht) liest so viele Daten wie möglich aus dem `stdout`-Puffer der ZPU in den Puffer des Moduls und weckt eventuell schlafende Leseprozesse auf.

Ein- und Ausgabe

```
static ssize_t zpu_chr_write (struct file *filep,
  const char __user *data, size_t count, loff_t *offset) {
  fifo_t        *f = &(zpu_io_stdin);
  unsigned int  n;

  if (FIFO_FULL(f)) {
    ZPU_ENABLE_STDIN_IR();

    if (filep->f_flags & O_NONBLOCK)
      return -EAGAIN;
    else if (wait_event_interruptible(f->queue, FIFO_NOT_FULL(f))
      return -EINTR;
  }

  /* Kopieren mit copy_from_user */

  f->count = f->count + n;
  ZPU_ENABLE_STDIN_IR();

  return n;
}
```

Vorgehen

- Testgetriebene Entwicklung der Bibliotheken und Anwenderprogramme (z.B. mit CUnit[1])
- Erstellung portabler Build-Skripte mit `autoscan` und `autoconf`.[2]
- Git[3] als Versionskontrollsystem. Zentrale Verwaltung auf Github, transparente Entwicklung. Organisation der erstellten Komponenten in unabhängigen Git-Repositories.

[1] `http://cunit.sourceforge.net/`
[2] `http://www.gnu.org/software/autoconf/`
[3] `http://git-scm.com/`

Das Intel HEX-Format

```
:100000000000000000000000000000000000000000F0
:0400010000000000EC
:100400000B0B0B98B02D0B0B0B888004000000029
:10041000000000000000000000000000000000DC
//  ...
:1010E200000008D9000008D9000008D9000008D97A
:1010F200000008D9000008D9000008D9000008D96A
:0C1102000000008D9000008D9000008D93E
:040000030000400F5
:00000001FF
```

Die `libintelhex`-Bibliotek: API

Die `libintelhex`-Bibliothek stellt Methoden zum Einlesen von Intel HEX-Dateien zur Verfügung.

`struct *ihex_recordset ihex_rs_from_file(char* filename)`
: Liest Binärdaten aus einer Dateieingabe ein.

`struct *ihex_recordset ihex_rs_from_str(char* input)`
: Liest Binärdaten aus einer Zeichenkette ein.

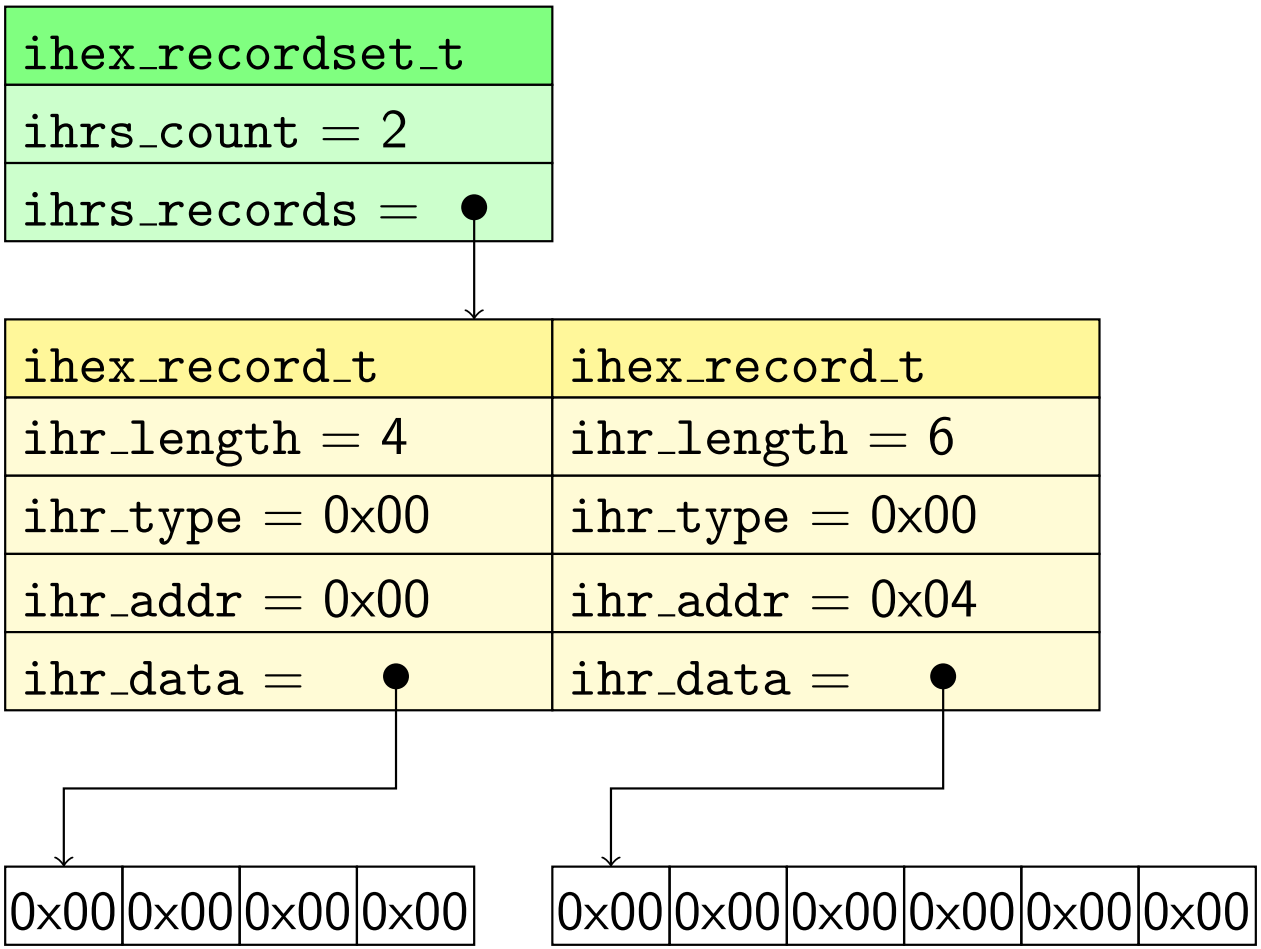
`int ihex_mem_copy(struct *ihex_records rec, void* dst, uint_t l, ihex_width_t w, ihex_byteorder_t o)`
: Kopiert Daten an eine beliebige (durch `dst`) angegebene Position. `l` ist die Größe des Ziel-Speicherbereichs. `w` gibt die Art des Speicherzugriffs an (bei der ZPU 32 Bit) und `o` die Byte-Reihenfolge.

Die libintelhex-Bibliotek: Datenstrukturen

```
typedef struct ihex_record {
    unsigned int ihr_length;
    ihex_rtype_t ihr_type;      // enum {...}
    ihex_addr_t  ihr_address;   // uint16
    ihex_rdata_t ihr_data;      // uint8*
    ihex_rchks_t ihr_checksum; // uint8
} ihex_record_t;

typedef struct ihex_recordset {
    unsigned int    ihrs_count;
    ihex_record_t* ihrs_record;
} ihex_records_t;
```

Die libintelhex-Bibliothek: Datenstrukturen



Anwendungsbeispiel

```
 1  #include <stdlib.h>
 2  #include <cintelhex.h>
 3
 4  int main()
 5  {
 6    ihex_recordset_t* r = ihex_rs_from_file("in.hex");
 7    void*             d = malloc(8192);
 8
 9    if (r != NULL)
10    {
11      return ihex_mem_copy(r, d, 8192, IHEX_WIDTH_32BIT,
            IHEX_ORDER_BIGENDIAN);
12    }
13    return ihex_errno();
14  }
```

Die **cIntelHEX**-Bibliothek ist **Open Source** (GNU LGPL):

- `https://github.com/martin-helmich/libcintelhex`

Die libzpu-Bibliothek

Die libzpu-Bibliothek soll im Userspace stark abstrahierte Methoden zur Steuerung der ZPU zur Verfügung stellen. Unterstützte Funktionen sind das **Laden neuer Programme** und **Ein- und Ausgabebehandlung**. Die Methoden werden in `zpu.h` definiert.

`int zpu_from_hexfile(char* filename)`
: Lädt ein ZPU-Programm aus einer Intel HEX-Datei in die ZPU. Liefert im Erfolgsfall `0` zurück. Stoppt die ZPU implizit und startet sie anschließend wieder.

`int zpu_stop()`
: Versetzt die ZPU in den Reset-Modus.

`int zpu_start()`
: Startet die ZPU wieder.

Userspace-Programme

```
1 > zpuload input.hex
2 Loading program "input.hex"...
3 Loaded program.
```

```
1 > zpuio
2 output (11) > Hallo Welt!
3 input > Hallo auch.
4 output (11) > HAllo Auch.
5 intput >
```

Ausblick

- Gerätedateien automatisch bei Registrierung des Geräts erstellen (geht mit **udev**).
- Beim Booten automatisch eine Firmware aus dem Dateisystem in das Gerät laden (z.B. über eine **udev-Regel**, die zpuload aufruft).

Fragen?