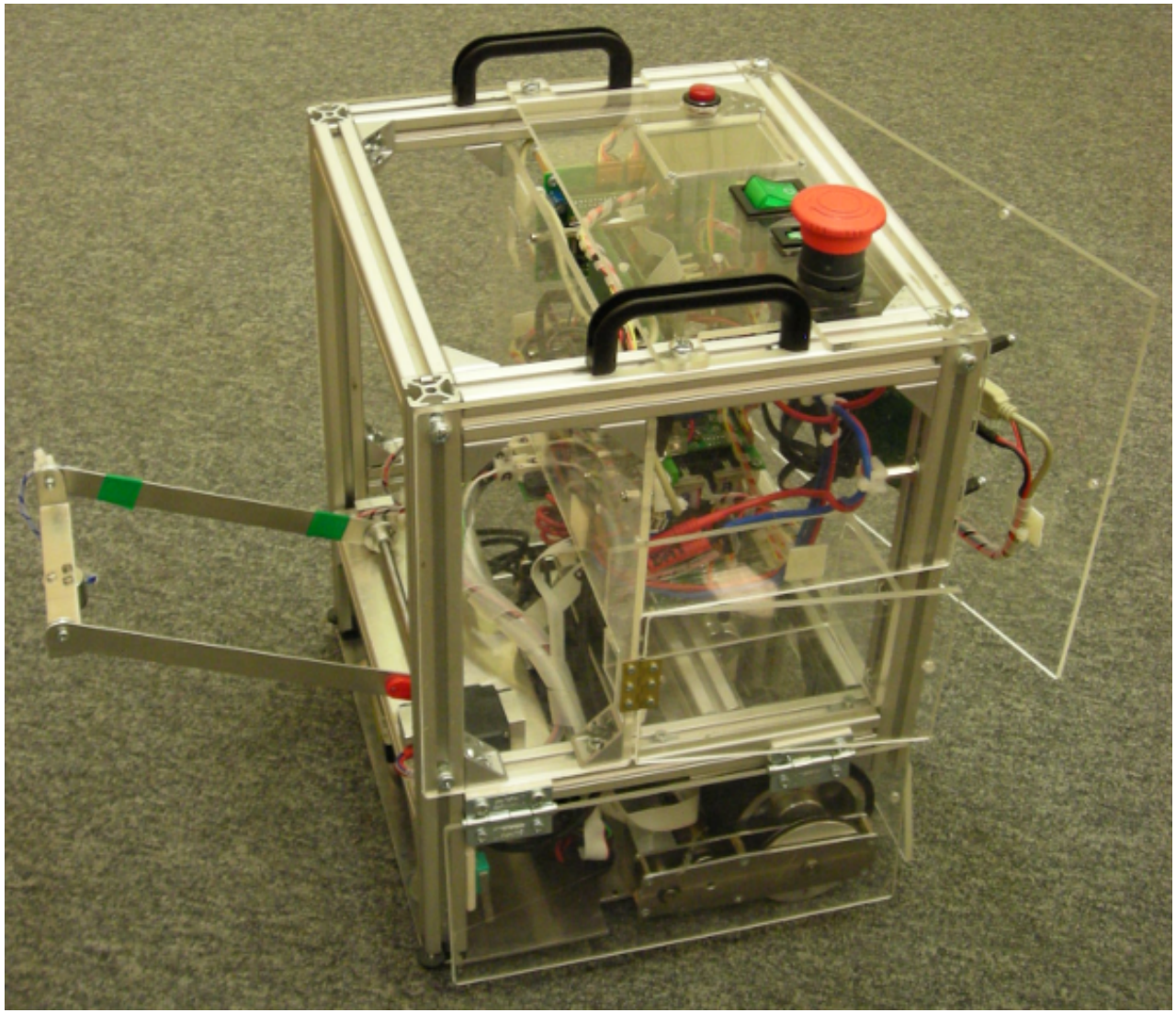
Jak na Linux v robotech a při jejich vývoji

Michal Sojka
sojkam1@fel.cvut.cz

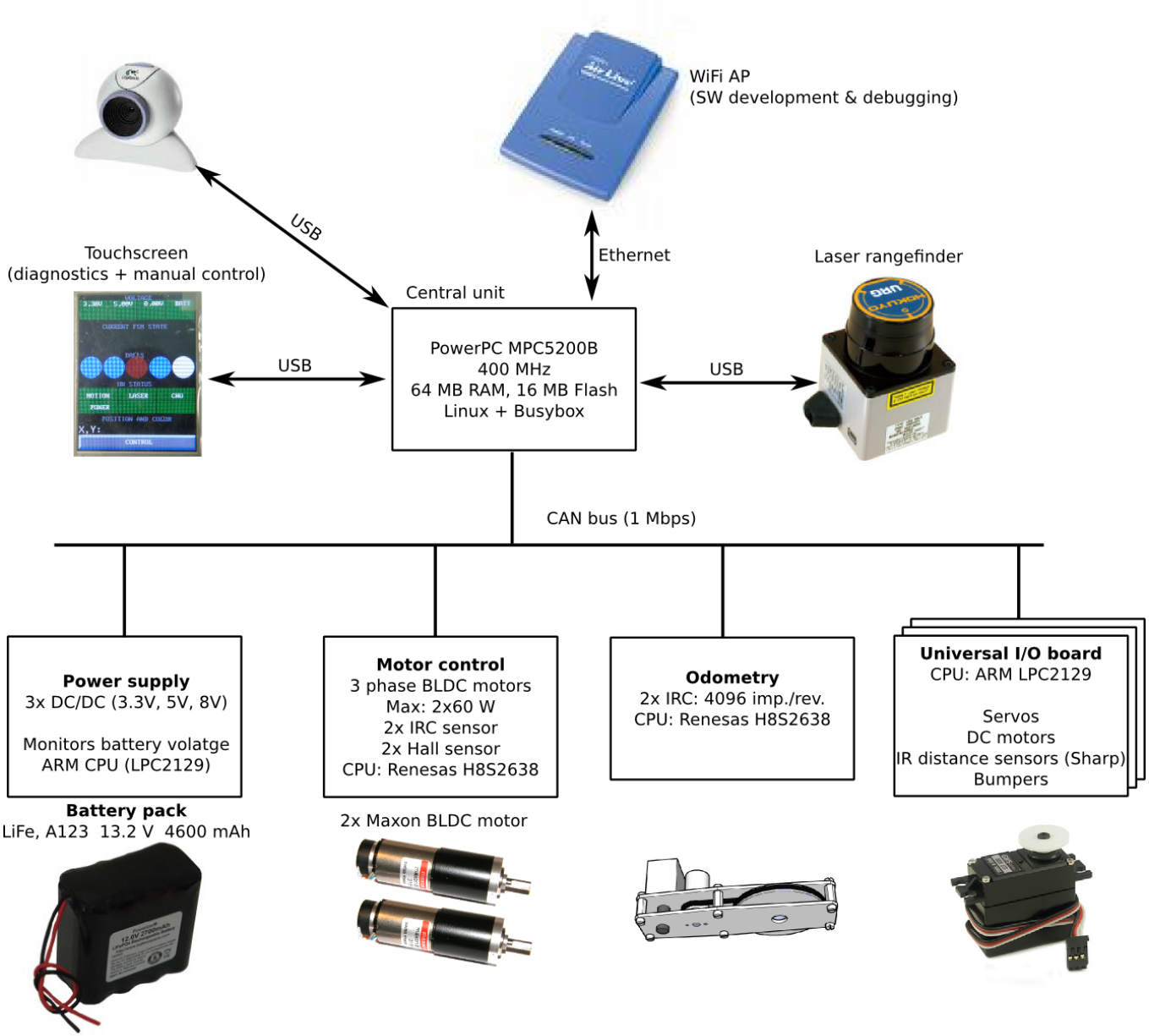
ČVUT v Praze
Fakulta elektrotechnická
Katedra řídicí techniky

InstallFest 2013

Ukázka robotu



HW architektura robotu

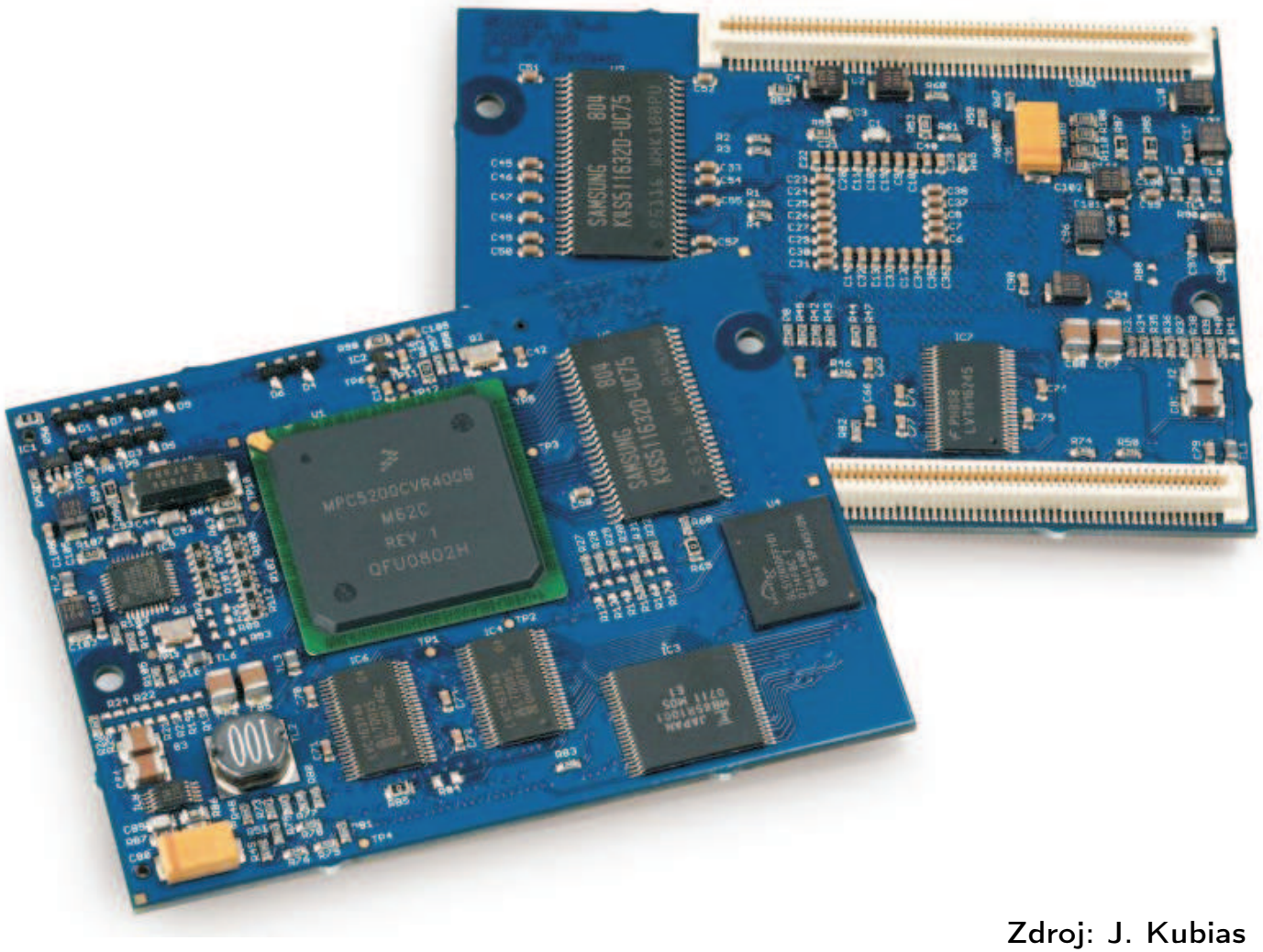


Obsah

Nebudu mluvit o ROS.

PowerPC MPC5200

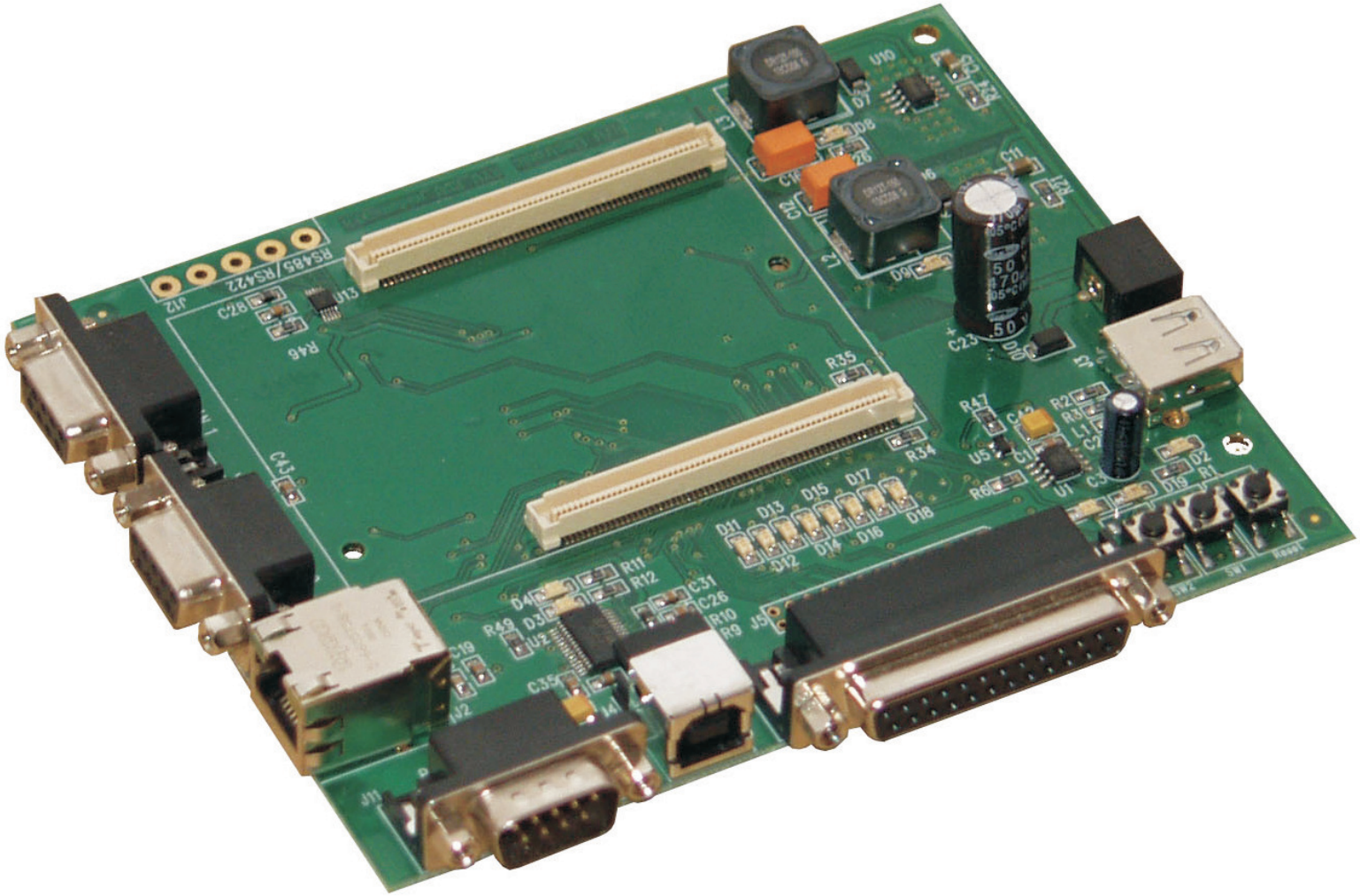
MIDAM Shark modul, 400 MHz, 128 MB RAM, 64 MB Flash



Zdroj: J. Kubias

RYU EDU daughter board

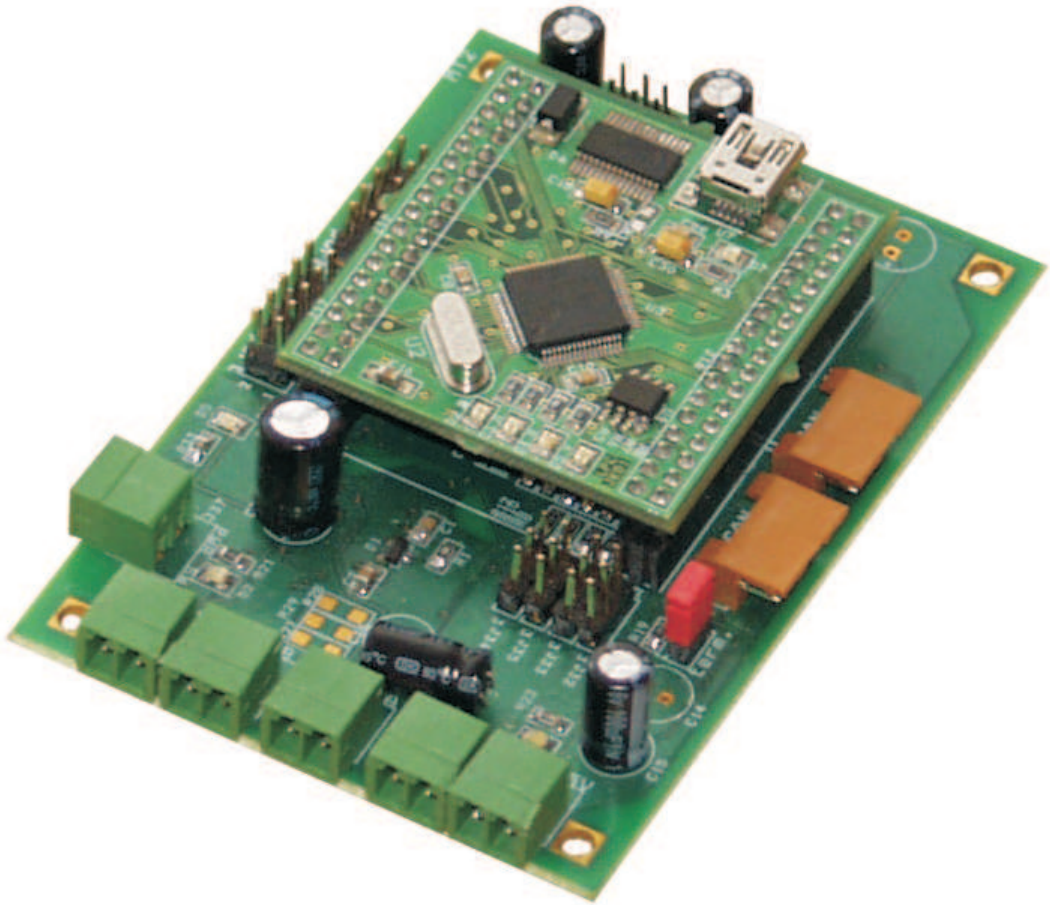
Konektory, LED diody, napájecí zdroj



Zdroj: J. Kubias

EbBoard

LPC2119 (ARM7TDMI-S), 60 MHz, 16 kB RAM, 256 kB Flash



Zdroj: J. Kubias

Vývoj SW pro mikrokontoléry

Když nemůžete nebo nechcete používat IDE.

GCC cross-compiler pro ARM

```
arm-elf-gcc -mcpu=arm7tdmi -Wl,-T,lpc21xx-flash.ld -g -O2 -Wall \
        -I ../include -o main main.c
```

Loader

`http://sourceforge.net/projects/lpc21isp/`

Makesystem

- Automatizuje činnosti (`make; make load`)
- Možnost integrace s IDE (kdevelop, eclipse)
- Jeden makesystem pro všechny HW platformy v projektu

Blikání LEDkou

```
#include <lpc21xx.h>

int main()
{
  volatile int tmp;
  while (1) {
    IOSET0 = 1<<3; // pin cislo 3
    for (tmp = 0; tmp < 1000000; tmp++);
    IOCLR0 = 1<<3;
    for (tmp = 0; tmp < 1000000; tmp++);
  }
}
```

Jádro Linuxu

Klasický postup

```
git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linu
cd linux
make menuconfig ARCH=powerpc
make ARCH=powerpc CROSS_COMPILE=powerpc-linux-gnu-
```

Jádro Linuxu

Klasický postup

```
git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linu
cd linux
make menuconfig ARCH=powerpc
make ARCH=powerpc CROSS_COMPILE=powerpc-linux-gnu-
```

Lepší varianta

```
make menuconfig O=../build
cd ../build
cp Makefile GNUmakefile
vi GNUmakefile # pridat ARCH, CROSS_COMPILE, ...
make
```

- Snadné verzování konfigurace
- Automatická instalace na TFTP server atd.

Root filesystem

Mnoho variant

- Busybox
- Buildroot
- OpenEmbedded
- Yocto
- Emdebian
- ...

Root filesystem

Mnoho variant

- Busybox
- Buildroot
- OpenEmbedded
- Yocto
- Emdebian
- ...

Buildroot

- `http://buildroot.uclibc.org/`
- Rychlé, jednoduché, funguje.
- „Naprogramováno“ v *GNU Make*

Root filesystem

Mnoho variant

- Busybox
- Buildroot
- OpenEmbedded
- Yocto
- Emdebian
- ...

Buildroot

- `http://buildroot.uclibc.org/`
- Rychlé, jednoduché, funguje.
- „Naprogramováno“ v *GNU Make*

Ukázka...

Verzování

- Je nepraktické mít ve verzovacím systému binárky celého root filesystém nebo jádra
- Použití git submodulů a „out-of-tree build“:

```
$ git ls-files
buildroot/build/.config
buildroot/build/Makefile
buildroot/build/overlay/bin/motor-reg.sh
buildroot/build/overlay/etc/init.d/S50can
buildroot/build/overlay/root/.ssh/authorized_keys
buildroot/repo     # submodule with buildroot sources
linux/3.4-rt/build/.config
linux/3.4-rt/build/.gitignore
linux/3.4-rt/build/GNUmakefile
linux/3.4-rt/repo # submodule with Linux sources
```

- Vytvoření build adresáře:

```
cd buildroot/repo; make O=../build menuconfig
```

- Každý může zreprodukovat vytvoření root filestému

```
cd buildroot/build; make
```

Bootování

- Při vývoji/testování je výhodné bootovat ze sítě:
  - Podporováno většinou bootloaderů (U-Boot, ...)
  - TFTP server pro jádro
  - NFS pro root filesystém
- S buildrootem:

```
tar -C /srv/nfs/robot -xf images/rootfs.tar
exportfs -o rw,no_root_squash 10.1.1.1:/srv/nfs/robot
```

- Příkazová řádka Linuxu (nastavená v bootloaderu):

```
root=/dev/nfs nfsroot=10.1.1.2:/srv/nfs/robot rw ip=dhcp
```

Real-Time Preemption Patches (rt-preempt)

Hodí se standardní jádro pro řízení robotů?

- Jak rychle zareaguje uživatelská aplikace na vnější podnět?

```
robot$ cyclictest -p 80 -n -t 1
host$ ping -f -s 64000 robot
```

Real-Time Preemption Patches (rt-preempt)

Hodí se standardní jádro pro řízení robotů?

- Jak rychle zareaguje uživatelská aplikace na vnější podnět?

```
robot$ cyclictest -p 80 -n -t 1
host$ ping -f -s 64000 robot
```

- 3000 μs $\Rightarrow$ 333 Hz (když budeme mít štěstí)

Real-Time Preemption Patches (rt-preempt)

Hodí se standardní jádro pro řízení robotů?

- Jak rychle zareaguje uživatelská aplikace na vnější podnět?

```
robot$ cyclictest -p 80 -n -t 1
host$ ping -f -s 64000 robot
```

- 3000 μs $\Rightarrow$ 333 Hz (když budeme mít štěstí)
- rt-preempt: 150 μs $\Rightarrow$ 6,6 kHz (závisí mimo jiné na HW)

Real-Time Preemption Patches (rt-preempt)

Hodí se standardní jádro pro řízení robotů?

- Jak rychle zareaguje uživatelská aplikace na vnější podnět?

```
robot$ cyclictest -p 80 -n -t 1
host$ ping -f -s 64000 robot
```

- 3000 μs $\Rightarrow$ 333 Hz (když budeme mít štěstí)
- rt-preempt: 150 μs $\Rightarrow$ 6,6 kHz (závisí mimo jiné na HW)
- OSADL QA Farm – dlouhodobé měření latence

Digitální vstupy/výstupy (GPIO)

- Ovládání HW připojeného k desce.
- Přístup ke GPIO bez problému z jádra.
- Lze „vyexportovat“ do user-space přes `sysfs`.
- Jaké číslo má daný GPIO v Linuxu?
  `dmesg|grep gpio`
- Práce s GPIO v shellu:

```
echo 239 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio239/direction
echo 1   > /sys/class/gpio/gpio239/value
```

linuxblink.c

```c
#include <unistd.h>
#include <fcntl.h>

int main()
{
    int led;
    led = open("/sys/class/gpio/gpio239/value", O_RDWR);
    while (1) {
        write(led, "1", 1);
        usleep(300000);
        write(led, "0", 1);
        usleep(300000);
    }
}
```

CAN bus

Propojení Linuxové desky s mikrokontroléry

- 3 vodiče: CAN-L, CAN-H, GND (+ napájení?)
- Odolnost proti rušení
- Více zařízení
- Deterministická MAC (pro automobilový průmysl)
- Většina mikrokontrolérů má variantu s CANem
- Max. 1 Mbit/s, 8 bytů/frame

CAN bus

Propojení Linuxové desky s mikrokontroléry

- 3 vodiče: CAN-L, CAN-H, GND (+ napájení?)
- Odolnost proti rušení
- Více zařízení
- Deterministická MAC (pro automobilový průmysl)
- Většina mikrokontrolérů má variantu s CANem
- Max. 1 Mbit/s, 8 bytů/frame

V Linuxu

- CAN je podporován v mainline (dříve projekt SocketCAN)
- Velmi podobné práci s TCP/UDP (sockety)
- `can-utils` (shell)

can-utils

- `cansend` – posílá CANový rámec
- `candump` – zobrazuje provoz na sběrnici
- `canplayer`
- `cangen`
- ...

```
cansend can0 001#0f       # LED
cansend can0 034#019000   # zvedak
candump can0,15:7ff       # odometrie
```

Vzdálené blikání LEDkou

Aplikace pro mikrokontrolér

```
#include <lpc21xx.h>
#include <deb_led.h>
#include <periph/can.h>
#include <system_def.h>

void can_rx(can_msg_t *msg) {
    deb_led_change(LEDB);
    if (msg->id == 0x01)
        deb_led_set(msg->data[0] & 0x0f);
}

int main (void)
{
    can_init(1000000 /*bits/s*/, 0, can_rx);
    while (1)
        // Low power mode, woken up by IRQ e.g. CAN_RX_IRQ
        PCON = PCON_IDL;
}
```

Vzdálené blikání LEDkou

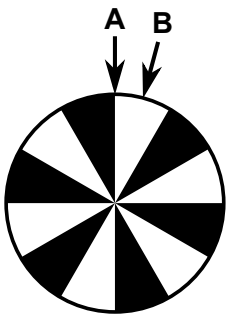
Aplikace pro Linux: canblink.c

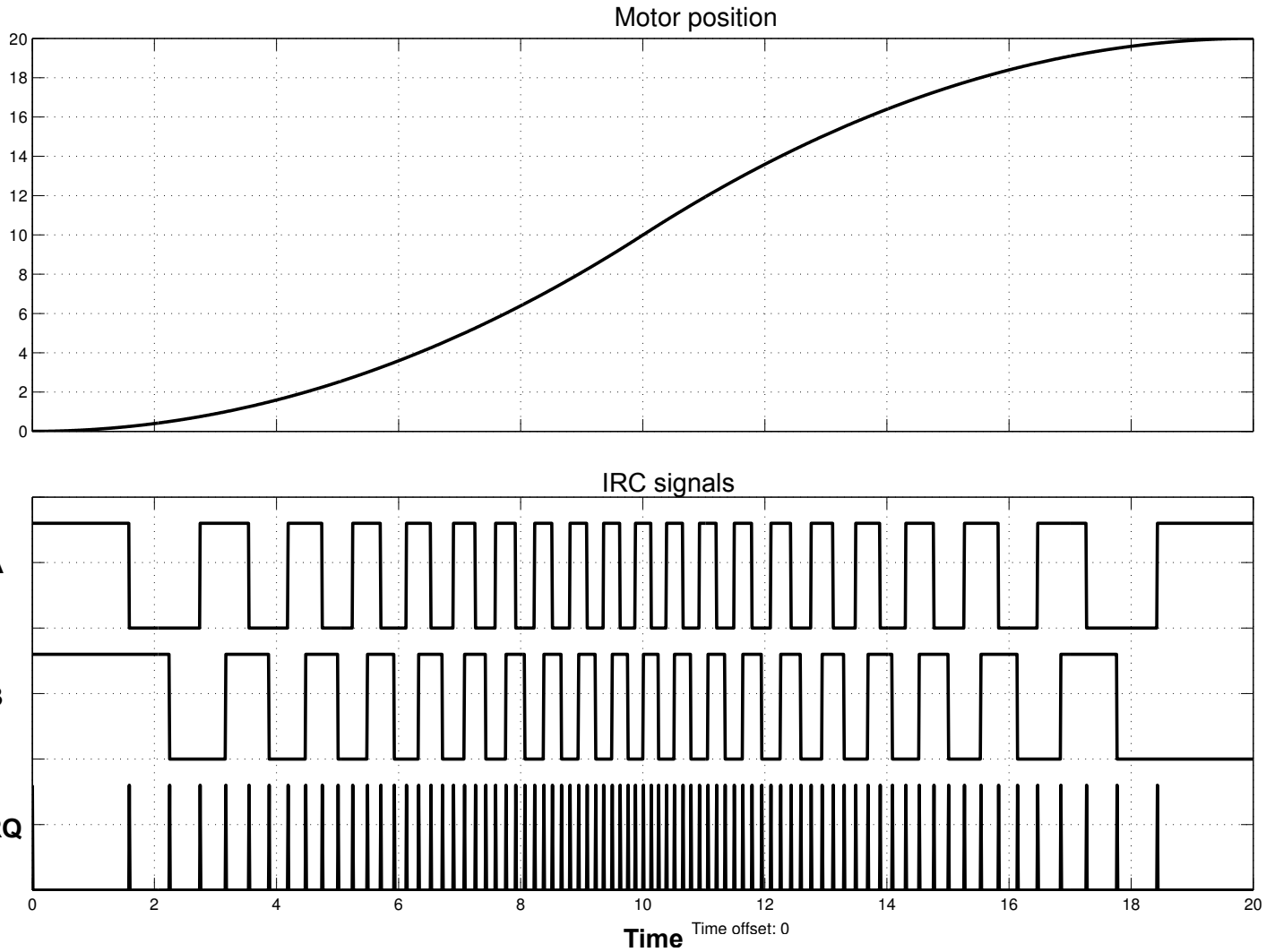
```
#include <linux/can.h>
#include <sys/socket.h>
#include <net/if.h>
#include <string.h>
#include <sys/ioctl.h>
#include <unistd.h>

int main()
{
    int led;
    struct sockaddr_can addr;
    struct ifreq ifr;
    led = socket(PF_CAN, SOCK_RAW, CAN_RAW);
    strcpy(ifr.ifr_name, "can0" );
    ioctl(led, SIOCGIFINDEX, &ifr);
    addr.can_family = AF_CAN;
    addr.can_ifindex = ifr.ifr_ifindex;
    bind(led, (struct sockaddr *)&addr,
         sizeof(addr));
    // ...
    struct can_frame f =
        { .can_id = 1, .can_dlc = 1 };

    while (1) {
        write(led, &f, sizeof(f));
        f.data[0]++;
        usleep(100000);
    }
}
```

Řízení motorku Linuxem





Jak na to?

- Obsluha IRQ, čtení GPIO vstupů (A, B), generování PWM signálu.
- Implementace v user-space je možná (například pomocí UIO), ale byla by pomalá.
- Motorek generuje až 10000 IRQ za sekundu.
- Řešení: vlastní driver.

Registrace a inicializace ovladače

Zjednodušeno, některé detaily vynechány – viz odkazy

```
struct motorek {
        struct mpc52xx_gpt *pwmf, *pwmb, *irca, *ircb;
        atomic_t pos;
        int irq;
};
static int motorek_probe(struct platform_device* dev)
{
        m = kzalloc(sizeof(*m), GFP_KERNEL);
        m->irq = irq_of_parse_and_map(dn, 0); // m->pwnf = ...;
        err = request_irq(m->irq, motorek_irq, 0, "motorek", m);
        platform_set_drvdata(dev, m);
}
static struct platform_driver motorek_driver = {
        .probe  = motorek_probe,
        .driver = { .name = "motorek" }
};
static int motorek_init_module(void)
{
        return platform_driver_register(&motorek_driver);
}
```

Čtení pozice motorku

```
static irqreturn_t motorek_irq(int irq, void *dev_id)
{
        struct motorek *m = dev_id;
        int irc_state = ((in_be32(&m->irca) & MPC52xx_GPT_STATUS_PIN) ? 1 : 0) |
                        ((in_be32(&m->ircb) & MPC52xx_GPT_STATUS_PIN) ? 2 : 0);
        m->pos += table[irc_state | m->last_irc_state << 2];
        return IRQ_HANDLED;
}
static ssize_t show_position(struct device *dev,
                             struct device_attribute *attr, char *buf)
{
        struct platform_device  *pdev = to_platform_device(dev);
        struct motorek          *m    = platform_get_drvdata(pdev);
        return snprintf(buf, PAGE_SIZE, "%d\n", atomic_read(&m->pos));
}
DEVICE_ATTR(position, 0444, show_position, NULL);

static int __devinit motorek_probe(struct platform_device* dev)
{       // ...
        device_create_file(&dev->dev, &dev_attr_position);
}
```

Nastavení napětí na motorku

```
static void motorek_action(struct motorek *m, unsigned val)
{
    u16 width = val*PWM_PERIOD/1000;
    out_be32(m->pwmf, (width<<16) | MPC52xx_GPT_PWM_OP);
}

static ssize_t set_action(struct device *dev, struct device_attribute *attr,
                          const char *buf, size_t count) {
    struct platform_device  *pdev = to_platform_device(dev);
    struct motorek          *m = platform_get_drvdata(pdev);

    unsigned a;
    sscanf(buf, "%u", &a);
    motorek_action(m, a);
    return strnlen(buf, PAGE_SIZE);
}

DEVICE_ATTR(action, 0644, NULL, set_action);

static int __devinit motorek_probe(struct platform_device* dev)
{       // ...
        device_create_file(&dev->dev, &dev_attr_action);
}
```

Regulátor motorku v shellu :-)

```
#!/bin/sh
ref=${1:-0}
while true; do
        p=$(cat /sys/devices/motorek.1/position);
        e=$((ref-p));
        echo $((e/2)) > /sys/devices/motorek.1/action
        echo $p;
        usleep 10000;
done
```

Odkazy

- Chcete se zapojit? → `http://flamingos.felk.cvut.cz/`
- HW wiki: `https://rtime.felk.cvut.cz/hw/`
- `https://rtime.felk.cvut.cz/gitweb/shark/motorek-5200.git/blob/HEAD:/motorek.c`
- `https://rt.wiki.kernel.org/`
- `https://osadl.org/`

Demo

Demo

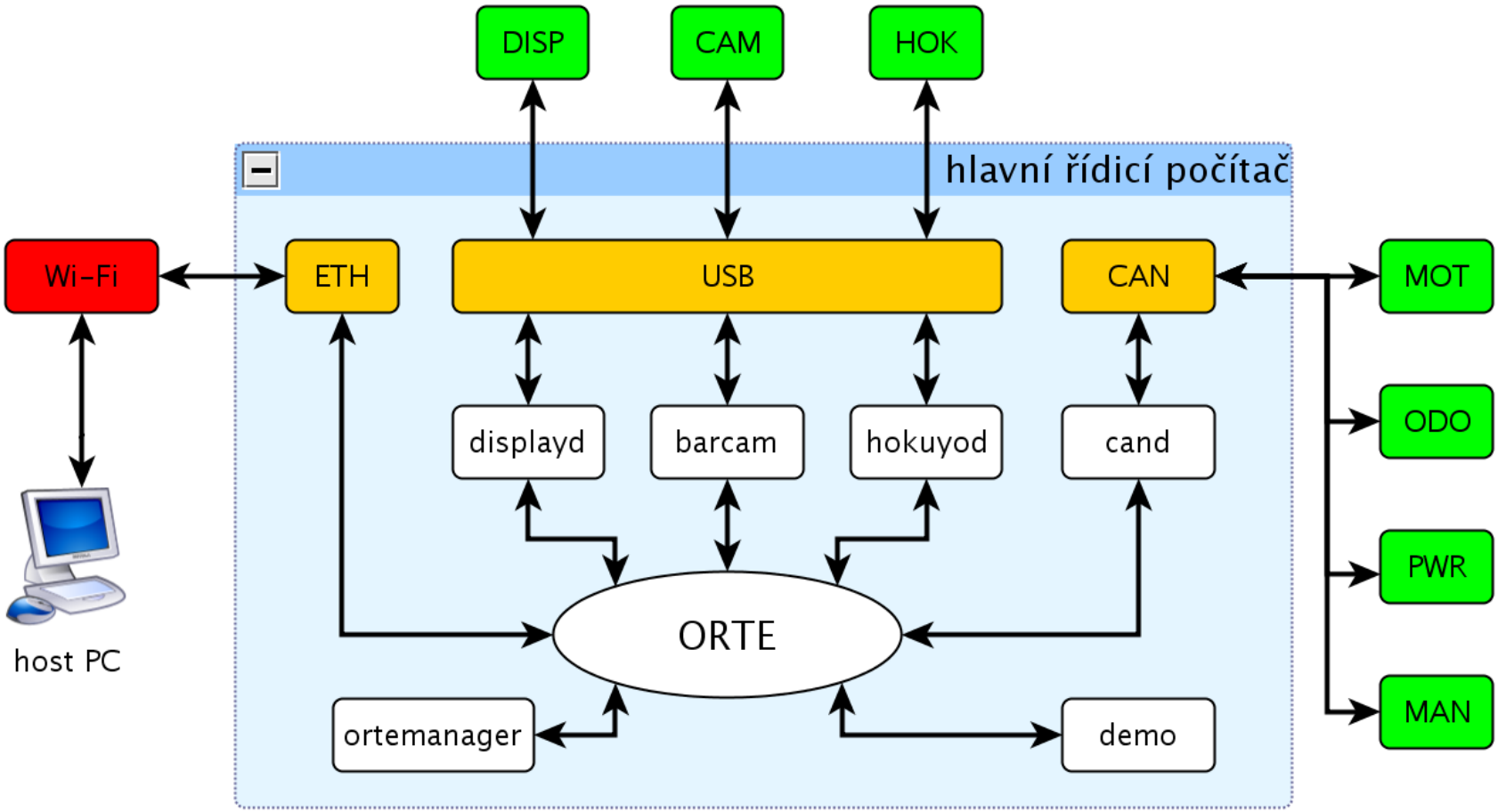
Děkuji za pozornost!

Demo

Děkuji za pozornost!

Otázky?

Komunikace v robotu



Zdroj: Michal Vokáč