Linux, RPi and other HW for DC and Brushless/PMSM Motor Control

Pavel Pisa
pisa@cmp.felk.cvut.cz
Martin Prudek
prudemar@fel.cvut.cz

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Control Engineering

Motion control hardware developed and provided by
PiKRON s.r.o.
{ppisa,porazil}@pikron.com

Content of Presentation

Background Introduction

The motion control and precise positioning is fundamental for many of medical, laboratory and robotic instruments and controllers to which development I have contributed to and often coordinated at PiKRON company. I use GNU/Linux as my main host system for more than 10 years. We use it as the only host development platform at company. We use it sometimes in embedded applications in parallel to RTEMS and bare metal development.

GNU/Linux is extensively used as core network infrastructure at Department of Control Engineering and it is the main environment for all seminaries and lab works in subjects which I teach. We use Linux, RTEMS and other real-time operating systems for applications and infrastructure development done for worldwide car-makers and industrial partners as well.

InstallFest 2015 Folow-up

Raspberry Pi is affordable and wide spread between students. I personally consider its use for anything serious as questionable but we have dived to work with it when other company asked as to help with problems which they have when they selected Raspberry Pi as a platform for data collection at factory plants. Then I have decided to test what are limits of this toy hardware in real-time and Linux areas and how to use it for enthusiasts and students to extent their knowledge. More about possible setup with protecting root filesystem and SD card from damage and wearing in prequel

Is Raspberry Pi Usable for Industrial and Robotic Applications?

`http://installfest.cz/if15/slides/pisa_rpi.pdf`
presented on InstallFest 2015

RPi and Other Experiments

Most recent theses I lead in motion control area

- Radek Mečiar: Motor control with Raspberry Pi board and Linux, 2014 (PDF)
- Martin Meloun: FPGA Based Robotic Motion Control System, 2014 (PDF)
- Martin Prudek: Brushless motor control with Raspberry Pi board and Linux, 2015 (PDF)

Electric Motors

- main categories:
  - brushed DC – motor with mechanical commutator
  - brushless DC motor (BLDC)/Permanent magnet synchronous motor PMSM – comutator replaced by control electronics – needs position/sector sensor or position estimation
  - stepper motor – synchronous motor, position usually enforced by direct drive
  - induction or asynchronous motor – exact position is not strictly required for control, torque is function of slip of rotor behind rotating magnetic field

Raspberry Pi Overview

- affordable/cheap single board computer developed for promotion of the teaching computer science
- the low cost demand lead to significant compromises
  - version 1 is based on Broadcom BCM2835 chip based on ARM1176JZF-S (ARMv6 architecture, insufficient for Debian armhf port which demands ARMv7-A and VFPv3-D16)
  - the cheap mobile applications SoC does not include ETHERNET controller which is added as USB converter
  - no memory technology devices (MTD) or integrated storage – uses standard or micro SD-card
  - CPU performance is not great and VideoCore IV GPU is proprietary/closed
  - extension connectors mindlessly (much better on B+ and version 2)
- the Debian compatibility solved by version 2 (BCM2836 quad-core ARM Cortex-A7). Performance enhanced.

Raspberry Pi Applications

- facts
  - intended for education
  - provides decent performance for video playback
  - is really cheap
- the last point is important
  - used for many hobby projects, strong community
  - used even in commercial solutions due to low cost even that it is not intended for such use

Alternatives

Many better alternatives exists for full featured GNU/Linux systems for industrial applications.
There are listed few ones

- FreeScale i.MX53 – ARM Cortex A8, ETHERNET, CAN, USB, ...
- FreeScale i.MX6 – ARM Cortex A9
- TI AM335x Sitara ARM Cortex-A8 (Beagle Bone black) – adds quadrature encoder inputs, two real time coprocessor units (PRU)
- Ti AM437x – Cortex A9 based, 2×PRU
- Ti AM5728 – dual core Cortex A15, PowerVR GPU, 2× Cortex M4 cores, dual core C66x DSP, IVA (H.264), 2×PRU

If we speak about military and space then there even more durable systems PowerPC SPARC much broader range if Linux is not required/not an option. Some discussed at the end of presentation.

But We Focus on RPi for While

- RPi is hardware bought by many students and hobbyist
  - quite often more powerful solution is found for initial dream multimedia applications and boards are free for experiments
  - RPi can be bridge to a broad world of electronic tinkering, ideas prototyping and can open eyes people that there is much more to play with than virtual worlds and clouds

Outline

RT-Preempt Patch

Realtime is not as fast as possible - realtime is as fast as specified – Doug Niehaus, Summer 2001

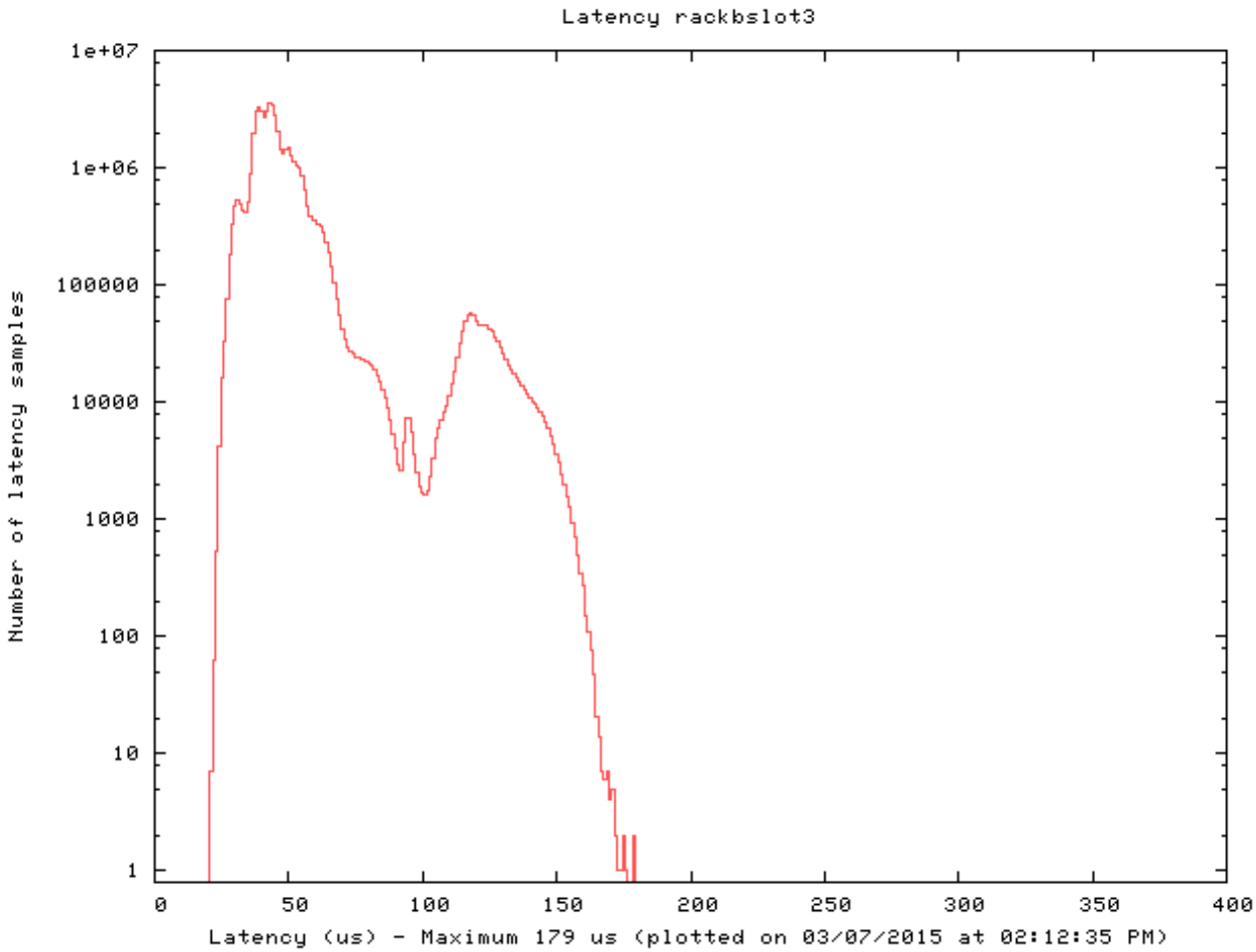
- More attempts to run RT task parallel to Linux base on same CPU (RT-Linux, RTAI) existed. But around 2001 and 2006 KURT/KUPS project tries to make whole kernel real-time. Work followed by Timesys, Thomas Gleixner, Ingo Molnar and OSADL.org now.
- The main idea behind changing Linux kernel to RTOS is to use already present support for multiple cores SMP and provide to system as many virtual CPUs as there are running threads/task.
- Realized by replacement of spin-lock synchronization by RT mutexes. redefinition of spin_lock/spin_unlock, spin_lock_irqsave/spin_unlock_irqrestore to use struct rt_mutex instead of atomic variables based lock

Actual RT State (for RPi)

- Open Source Automation Development Lab – long term testing and Quality Assurance Realtime Farm
- Latest available RT-Preempt patch is 4.1.7-rt8
  `https://www.kernel.org/pub/linux/kernel/projects/rt/`
- Maximal under about 100 µsec on powerful SMP x86 systems
- But what to expect for Raspberry Pi (BCM2708/BCM2835) Check at OSADL.org QA Farm Realtime rack-b-slot-3

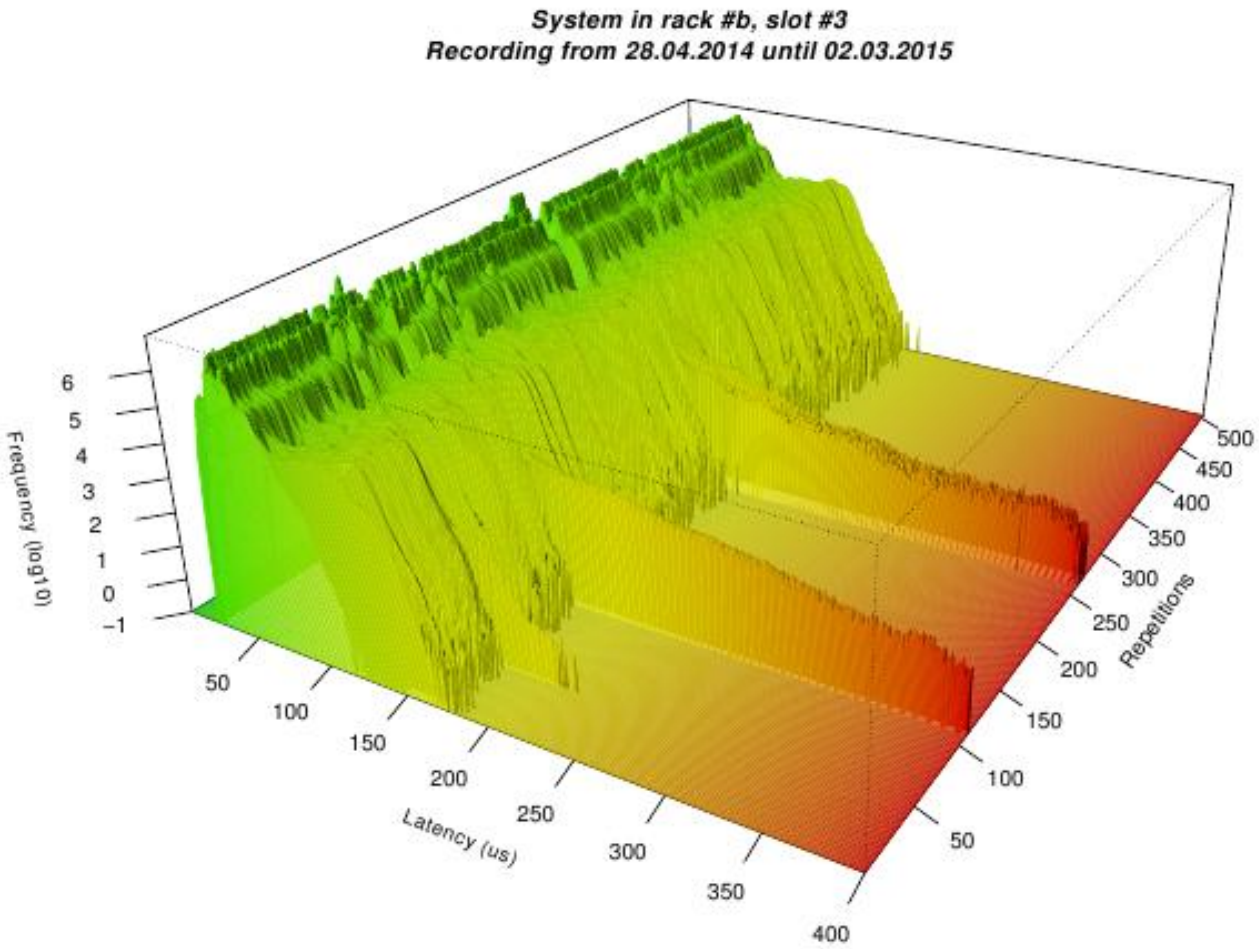
RPi 3.18.7-rt2 Latency Plot

OSADL.org – OSADL.org QA Farm Realtime – BCM2835 rack-b-slot-3
cyclictest -l50000000 -m -n -a0 -t1 -p99 -i400 -h400 -q



RPi Latency Long Term 3D

OSADL.org – OSADL.org QA Farm Realtime – BCM2835 rack-b-slot-3
Long term latency trancing for organization members available



Is It Enough?

- The standard control application is motor servo control
- Small DC and permanent magnet synchronous motors mechanical time constant is between 3 and 10 msec (electrical one can be under 1 msec, but mechanical is prevalent for control).
- System, when controlled for position, is not stable (pure integrator)⇒controller sampling period should be shorter than time constant to achieve proper control
- Maximal latencies under 200 μsec⇒RPi should be suitable for such control

RT Kernel Patches

Official RPi kernel `git://github.com/raspberrypi/linux.git`
branch rpi-3.18.y
Patches applied

```
aufs3.18.1+ kbuild patch
aufs3.18.1+ base patch
aufs3.18.1+ mmap patch
aufs3.18.1+ standalone patch
Apply Aufs 3.18-20150305 sources by J. R. Okajima
aufs3.18.1+ module build enabled in RPi default config.
Allow ARMv6/ARM1176 to be selected for ARM Versatile PB.
Apply patch-3.18.7-rt2.patch fully preemptive ker-
nel patch by Sebastian Andrzej Siewior
Provide individual CPU usage measure-
ment based on idle time by Carsten Emde (OSADL)
Save the current patchset in the kernel
Reading /proc/slabinfo may cause large latencies
Add trace latency histogram to monitor context switch time
Workaround to access sysfs cpufreq variables
ARM bcmrpi: add bcmrpi_rt_defconfig .
ovl: do not reject NFS when used as lowerdir (change is inse-
cure, could lead to crash when NFS content is changed).
```

RT Kernel Build

- Patched kernel available in repository `https://github.com/ppisa/linux-rpi` , branch rpi-3.18.y-aufs-rt-ppisa
- The ARMv6 toolchain is required to build kernel. Unfortunately, arm-linux-gnueabihf- official Debian cross compiler targets ARMv7 VFPv3-D16. It can be used to build kernel (controlled by options) but user applications are miss-compiled and link against bad GLIBC
- Custom arm-rpi-linux-gnueabihf- toolchain used for crosscompile
- Kernel build

```
make ARCH=arm CROSS_COMPILE=arm-rpi-linux-gnueabihf- \
     bcmrpi_rt_defconfig
make ARCH=arm CROSS_COMPILE=arm-rpi-linux-gnueabihf-
make ARCH=arm CROSS_COMPILE=arm-rpi-linux-gnueabihf- \
     INSTALL_MOD_PATH=$(pwd)/_modules modules_install
```

Toolchain RPi ARMv6

- Can be downloaded from DCE RTime server. There is described GCC 4.9.x sources configuration as well.
  `https://rtime.felk.cvut.cz/hw/index.php/Raspberry_Pi`
- Critical configure options to achieve compatibility with RPi ARMv6

```
--with-arch=armv6 \
--with-fpu=vfp \
--with-float=hard \
--enable-multiarch \
--disable-sjlj-exceptionshardware
```

QEMU for ARM CHROOT

- Debian packages qemu-user-static and binfmt-support
- Packages install configuration to /var/lib/binfmts
- Can be started as service

```
service binfmt-support start
```

- or single

```
echo 1 >/proc/sys/fs/binfmt_misc/status
update-binfmts --enable qemu-arm
```

- which creates/configures /proc/sys/fs/binfmt_misc/qemu-arm

```
enabled
interpreter /usr/bin/qemu-arm-static
flags: OC
offset 0
magic 7f454c4601010100000000000000000002002800
mask ffffffffffffff00fffffffffffffffffeffffff
```

NFS Root Filesystem on a Host Computer

- Debootstrap NFS root filesystem on host (usually x86)

```
mkdir -p /srv/nfs/raspbian-jessie/usr/bin
# the second stage executes ARM code
# use static QEMU user emulator
cp /usr/bin/qemu-arm-static \
  /srv/nfs/raspbian-jessie/usr/bin
wget http://archive.raspbian.org/raspbian.public.key
gpg --no-default-keyring \
  --keyring ./raspbian-archive-keyring.gpg \
  --import raspbian.public.key
debootstrap \
  --keyring=./raspbian-archive-keyring.gpg \
  --arch=armhf jessie /srv/nfs/raspbian-jessie \
  http://archive.raspbian.org/raspbian
```

- The system can be examined from chroot /srv/nfs/raspbian-jessie/

Export NFS for RPi

- Add entry to /etc/exports

```
/srv/nfs/raspbian-jessie
    192.168.1.0/24(rw,async,no_root_squash,subtree_chec
```

and run

```
service nfs-kernel-server reload (or start)
```

- or run services directly

```
modprobe nfsd
modprobe nfsv3
modprobe nfsv4
if [ ! -e /proc/fs/nfsd/exports ] ; then
  mount -t nfsd nfsd /proc/fs/nfsd
  /usr/sbin/rpc.nfsd 4
  /usr/sbin/rpc.mountd --manage-gids
fi
exportfs -
o rw,fsid=9,no_root_squash,subtree_check,no_acl,nohide \
   192.168.1.0/24:/srv/nfs/raspbian-jessie
```

Configure RPi Boot on SD card

- Install right modules of compiled kernel to NFS exported filesystem

```
make INSTALL_MOD_PATH=/srv/nfs/raspbian-jessie modules_install
```

- Copy imagetool prepared kernel (kernel-3.18.8-rt2+.img) to the first partition SD card partition
- Modify config.txt file (right kernel and DT overlays)

```
kernel=kernel-3.18.8-rt2+.img
#device_tree=bcm2835-rpi-b-plus.dtb
device_tree_overlay=overlays/enable-i2c-spi-overlay.dtb
```

- Specify kernel parameters in file /boot/cmdline.txt

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200
kgdboc=ttyAMA0,115200 smsc95xx.macaddr=b8:27:eb:3d:cc:29
root=/dev/nfs ro nfsroot=192.168.1.10:/srv/nfs/raspbian
ip=192.168.1.33:::::eth0 elevator=deadline rootwait
```

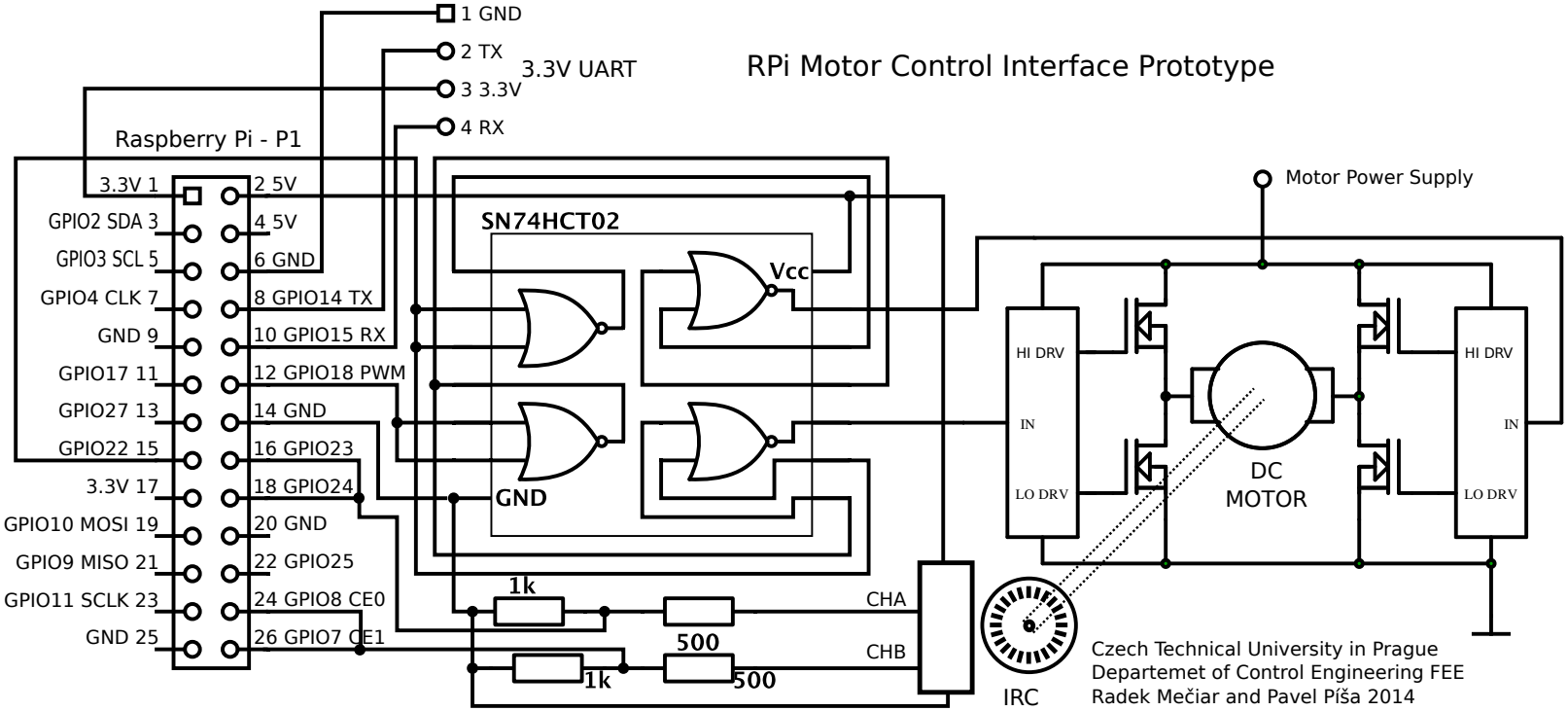
- add init=/sbin/init-overlay if overlay setup described in previous presentation is used

Outline

System Events Rates

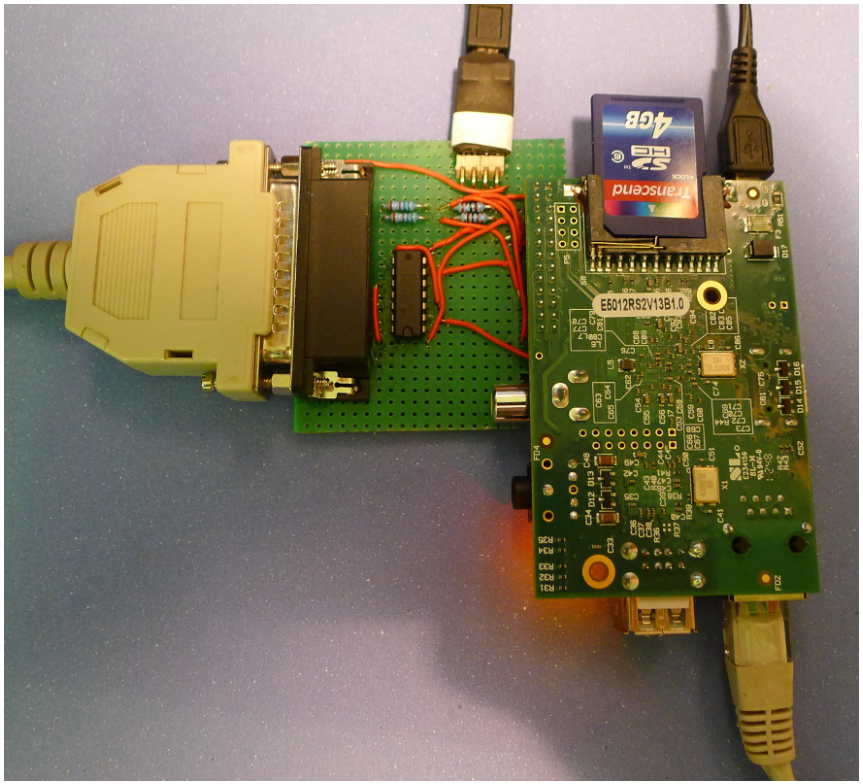
- The 1 kHz control loop sampling frequency can be achieved by RPi
- But RPi has no hardware for position (usually quadrature incremental) sensor interfacing
- Incremental encoder output changes frequency can reach MHz units
- For 500 slots wheel and 4000 RPM the required frequency is about 150 kHz
- This is too much for user-space and even proper kernel space processing on RPi but it is nice experiment for system stability under load testing
- Serious solution requires to extend RPi by incremental encoder interface implemented in hardware (FPGA)

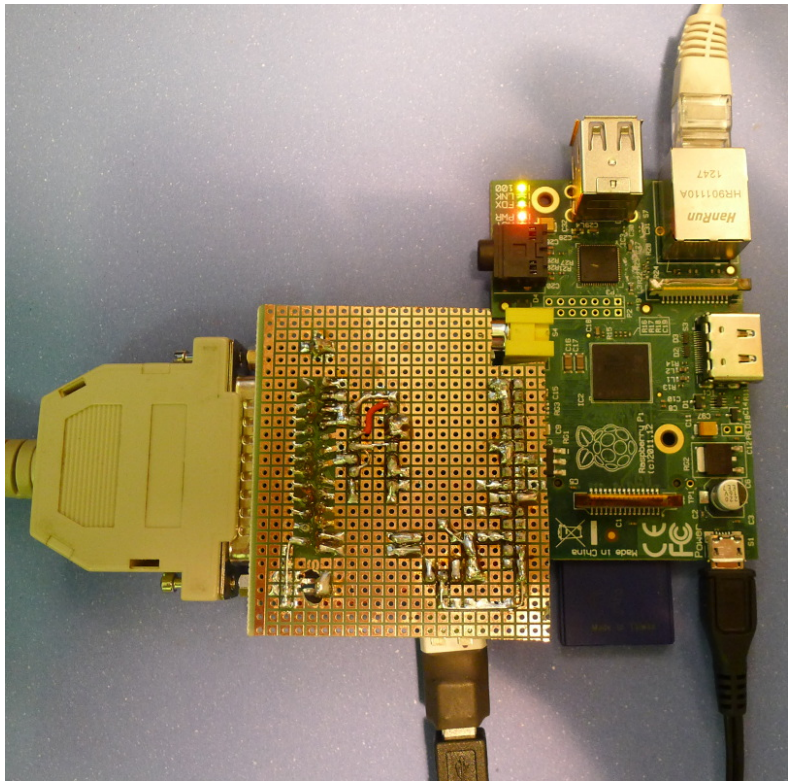
GPIO Only Based DC Motor Interfacing



- As simple as possible
- Four NOR gates (SN74HCT02)
- H-bridge (L6203)

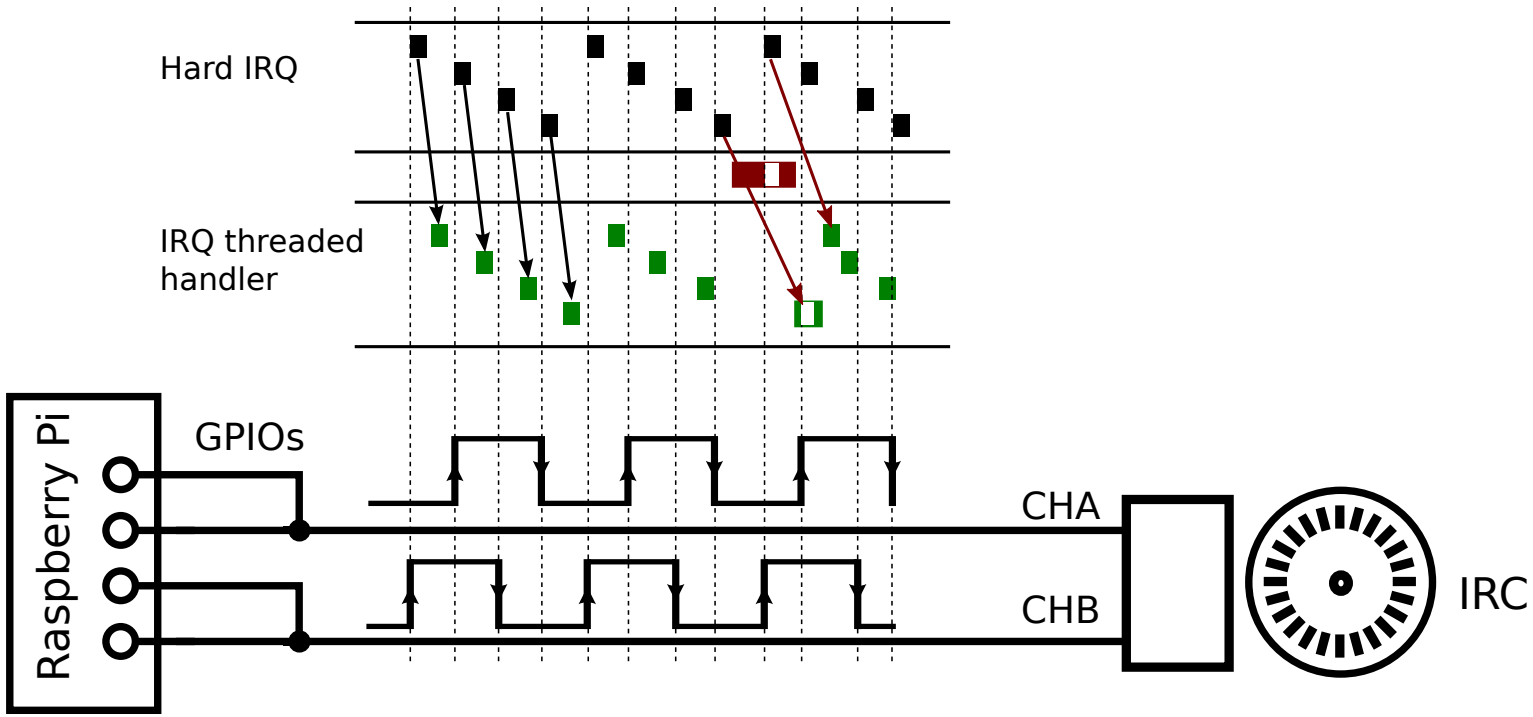
Wire-wrapped Prototype Design





- H-bridge (L6203) is located on R35PSR course DC motor kit

Software IRC Signals Processing



- Position calculation works better if derived from the order of IRQs than from the signal values read in the handler.
- FIFO run queue preserve order! (Observation from Mečiar Radek Motor control with Raspberry Pi board and Linux 2014 bachelor thesis)

IRC Kernel Driver

- Simple character device `/dev/irc0` which counts motor position
- Order of interrupts is converted to increment and accumulated in the kernel variable
- uint32_t (4-bytes) actual value is read from the device
- driver source available at GitHub repository
  `https://github.com/ppisa/rpi-rt-control`
  The core is a file kernel/modules/rpi_gpio_irc_module.c
- Test from user-space

```
modprobe rpi_gpio_irc_module
hexdump -e '"%d" ' /dev/irc0
```

IRC Access from C Code

```
int irc_dev_fd;

int irc_dev_init(void)
{
    irc_dev_fd = open(irc_dev_name, O_RDONLY);
    if (irc_dev_fd == -1) {
        return -1;
    }
    return 0;
}
int irc_dev_read(uint32_t *irc_val)
{
    if (read(irc_dev_fd, irc_val, sizeof(uint32_t))
            != sizeof(uint32_t)) {
        return -1;
    }
    return 0;
}
```

Boost IRC Kernel Threads Priority

- The fully preemptive kernel runs even IRQ processing in thread context, all interrupts use priority 50 by default

```
modprobe rpi_gpio_irc_module
IRC_PIDS=$(ps Hxa -o command,pid | \
  sed -n -e 's/^\[irq\/[0-9]*-irc[0-9]_ir\][ \t]*\([0-
9]*\)$/\1/p')

for P in $IRC_PIDS ; do
  schedtool -F -p 95 $P
done
```

- List threads by real-time priority

```
ps Hxa --sort rtprio -
o pid,policy,rtprio,state,tname,time,command
```

Power Output – PWM

- Only single PWM signal generator easily available on Raspberry Pi
- Direction has to be controlled by GPIO pin
- GPIO access possible through `/sys` interface

```
echo 22 >/sys/class/gpio/export
echo out >/sys/class/gpio/gpio22/direction
cat /sys/class/gpio/gpio22/value
echo 1 >/sys/class/gpio/gpio22/value
```

- But access over /sys represent significant overhead
- Direct access from users-pace to GPIO and PWM peripheral registers is used instead

Direct GPIO and PWM Registers Access

- Implemented in `int rpi_peripheral_registers_map(void)` function
- The physical address range can be accessed from user-space by `mmap()` syscall

```
mem_fd = open("/dev/mem", O_RDWR|O_SYNC)
gpio_map = mmap(NULL, BLOCK_SIZE, PROT_READ|PROT_WRITE,
                MAP_SHARED, mem_fd, GPIO_BASE);
```

- Detailed description at `http://elinux.org/RPi_Low-level_peripherals`
- Fast GPIO and PWM access functions for controller application can be found in files rpi_gpio.c and rpi_bidirpwm.c found in appl/rpi_simple_dc_servo example of `https://github.com/ppisa/rpi-rt-control` repository

Simple PID Based Speed Controller

- Implemented in file rpi_simple_dc_servo.c
- Next commands are available setpwm, readirc and runspeed.
- The real time task cannot be swapped or code paged in on demand

```
mlockall(MCL_FUTURE | MCL_CURRENT)
```

- real time priority has to be used for control task

```
pthread_attr_t attr;
struct sched_param schparam;
pthread_attr_init(&attr);
pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SC
pthread_attr_setschedpolicy(&attr, SCHED_FIFO);
schparam.sched_priority = sched_get_priority_min(SCHED_
pthread_attr_setschedparam(&attr, &schparam);
pthread_create(thread, &attr, start_routine, arg);
pthread_attr_destroy(&attr);
```

The Controller Timing

- The use CLOCK_MONOTONIC for all control related timing is critical, CLOCK_REATIME can skip forward and backward due to user or NTP adjustment

```
sample_period_nsec = 20*1000*1000;
clock_gettime(CLOCK_MONOTONIC, &sample_period_time);
do {
  sample_period_time.tv_nsec += sample_period_nsec;
  if (sample_period_time.tv_nsec > 1000*1000*1000) {
    sample_period_time.tv_nsec -= 1000*1000*1000;
    sample_period_time.tv_sec += 1;
  }
  clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME,
                  &sample_period_time, NULL);
  /* Run timed actions there */
  ...
} while(1);
```

Ensure Proper Stop When Interrupted

```
void stop_motor(void)
{
  rpi_bidirpwm_set(0);
}

void sig_handler(int sig)
{
  stop_motor();
  exit(1);
}

...
  struct sigaction sigact;
  memset(&sigact, 0, sizeof(sigact));
  sigact.sa_handler = sig_handler;
  sigaction(SIGINT, &sigact, NULL);
  sigaction(SIGTERM, &sigact, NULL);
```

The Controller Sample Time Step

Please consult application code for complete solution with overflow and anti-windup protection

```
err =  (pos_req - actual_pos);
ctrl_i_sum += err * ctrl_i;
action = ctrl_p * err + ctrl_i_sum + ctrl_d *
  (err - ctrl_err_last);
ctrl_err_last = err;
rpi_bidirpwm_set(action >> 8);
```

More information can at pages of DCE's Real-Time systems Programming course `http://support.dce.felk.cvut.cz/psr/` and subject's Semestral Work – Motor Control pages. Other valuable information on former subject page `https://support.dce.felk.cvut.cz/pos/hlavni_uloha/`

Test Results

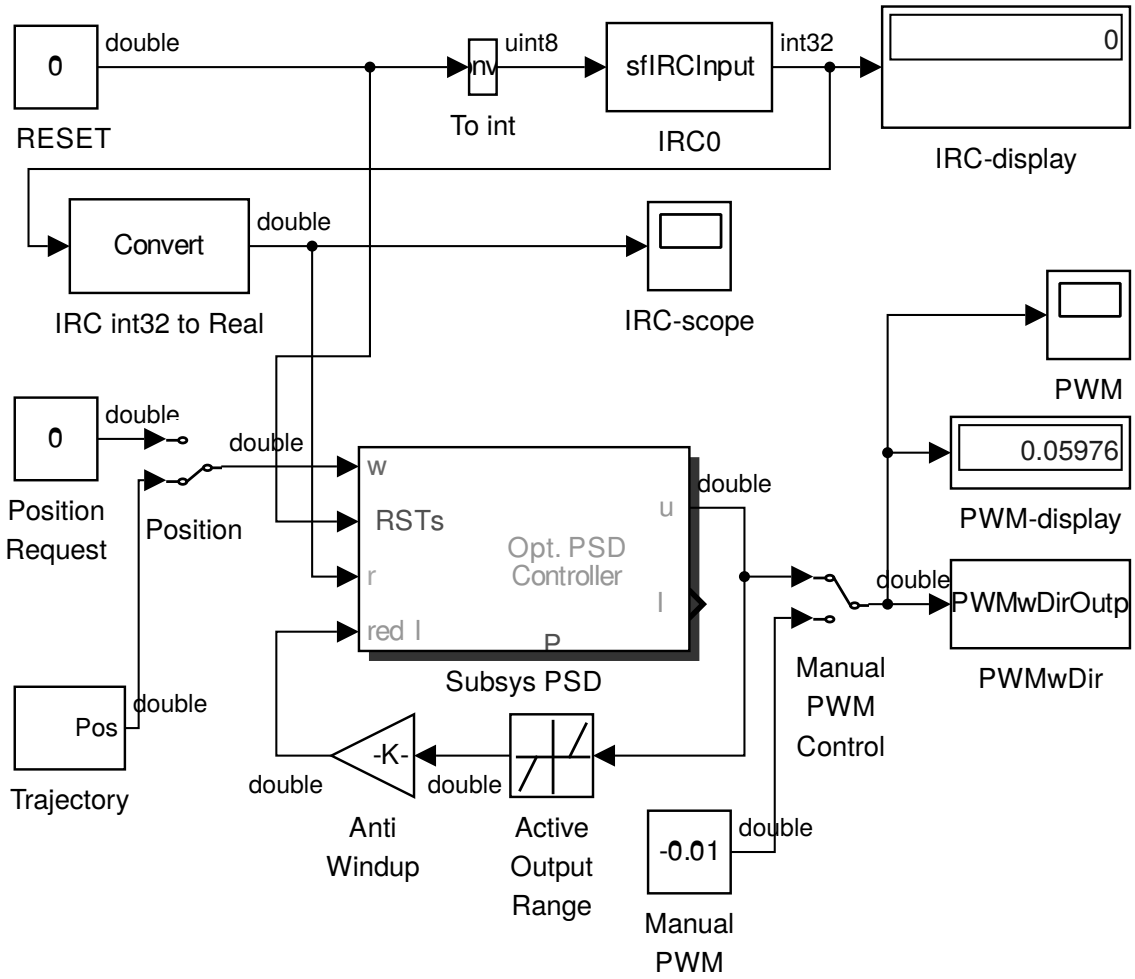
- Reliable speed control achieved for smaller speeds
- RPi capable to cope with almost full speed of PSR course DC motor with low resolution IRC sensor
- The speed limit is much lower for industry grade motor used in real PiKRON's applications
- The RPi is capable to process about 28 000 IRQ events per second but when more arrives the system and controller is overloaded/blocked and motor runs out of control
- But even in overload case system is stable when motor is externally braked/hold/slow down controller receives CPU time again and system recovers from overload
- Solution is nice for education but not safe for industrial use
- The test with FPGA based IRQ processing in in preparation but even in this case RPi is not considered by us as platform reasonable for industrial motion control applications

Outline

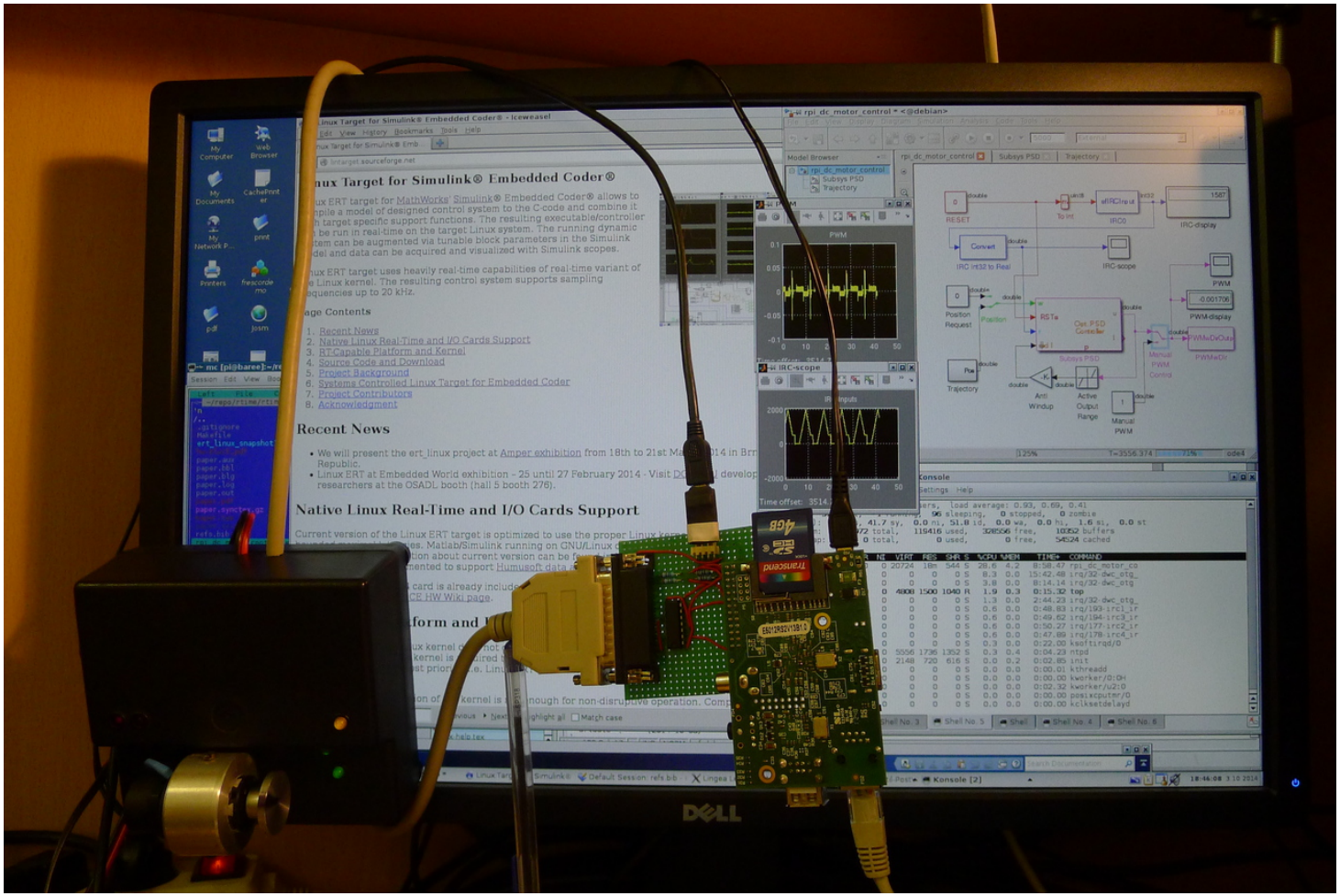
Embedded Real Time and the DCE Department

- CTU FEE Department of Control Engineering has been and is involved in Matlab/Simulink real-time support from its beginning (origin of real-time toolbox can be trace to our department)
- We have long term experience with fully preemptive kernel and hardware interfacing
- Embedded Real-time Target has been adapted/partially rewritten by Michal Sojka to be usable for real applications (MathWork included embedded solutions are often Windows only and use POSIX timers and signals which have uncontrolled latencies during delivery)
- The blocks for SocketCAN, Humusoft data acquisition PCI cards and minimal set of RPi periferals has been implemented
- COMEDI blockset has been updated and tested with our Linux ERT version as well

RPi DC Motor Control Simulink Diagram



RPi DC Motor Control Simulink Prototype



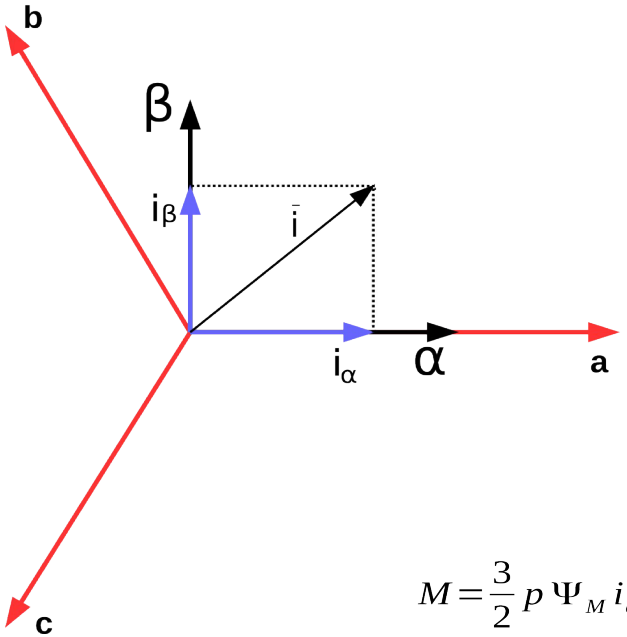
RPi DC Motor Control Simulink Remarks and Pointers

- Incremental encoder input implemented as S-function sfIRCInput.c which opens `/dev/irc0` and reads actual position from kernel driver
- Bidirectional PWM output is implemented in S-function sfPWMwDirOutput.c and uses same registers direct access approach as described in the previous section
- The whole setup is documented on respective Lintarget/Linux ERT project page `http://lintarget.sourceforge.net/rpi-motor-control/index.html`
- used ert_linux target and CC=arm-rpi-linux-gnueabihf-gcc set for make_rtw. scp and ssh are used to copy and run binary on target. Simulink external mode (parameters on-line tune and signals scope windows) is available.
- The generated code performance is the same as for hand written case – limitation is IRC events processing in the kernel

Outline

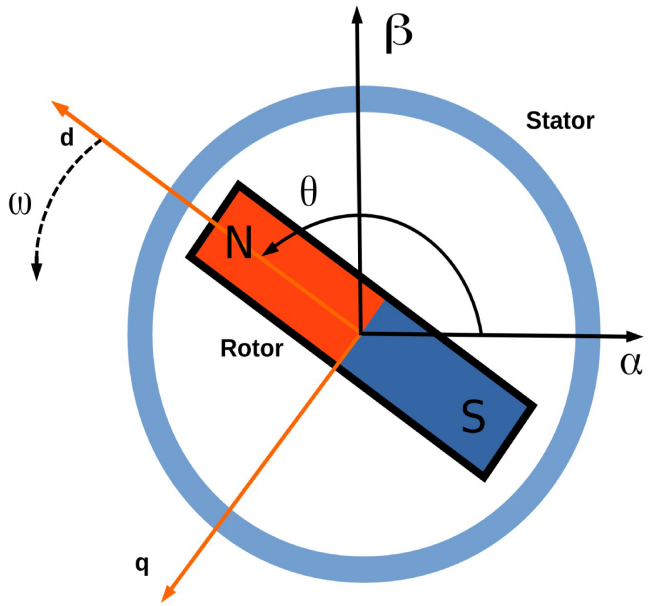
Three Phase Circuit Transformations

Clarke transformation

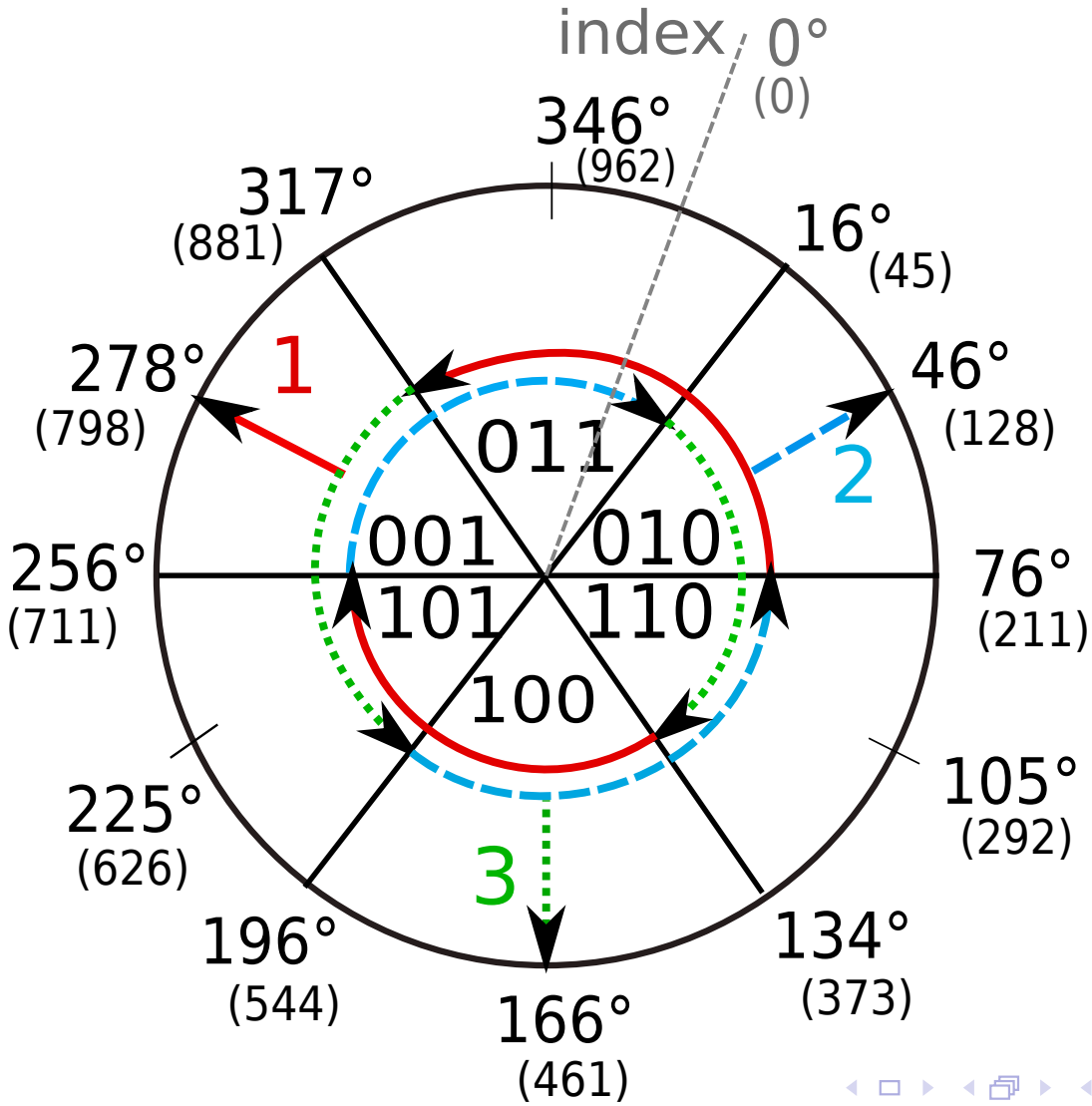


$$M = \frac{3}{2} p \Psi_M i_q$$

Park's transformation

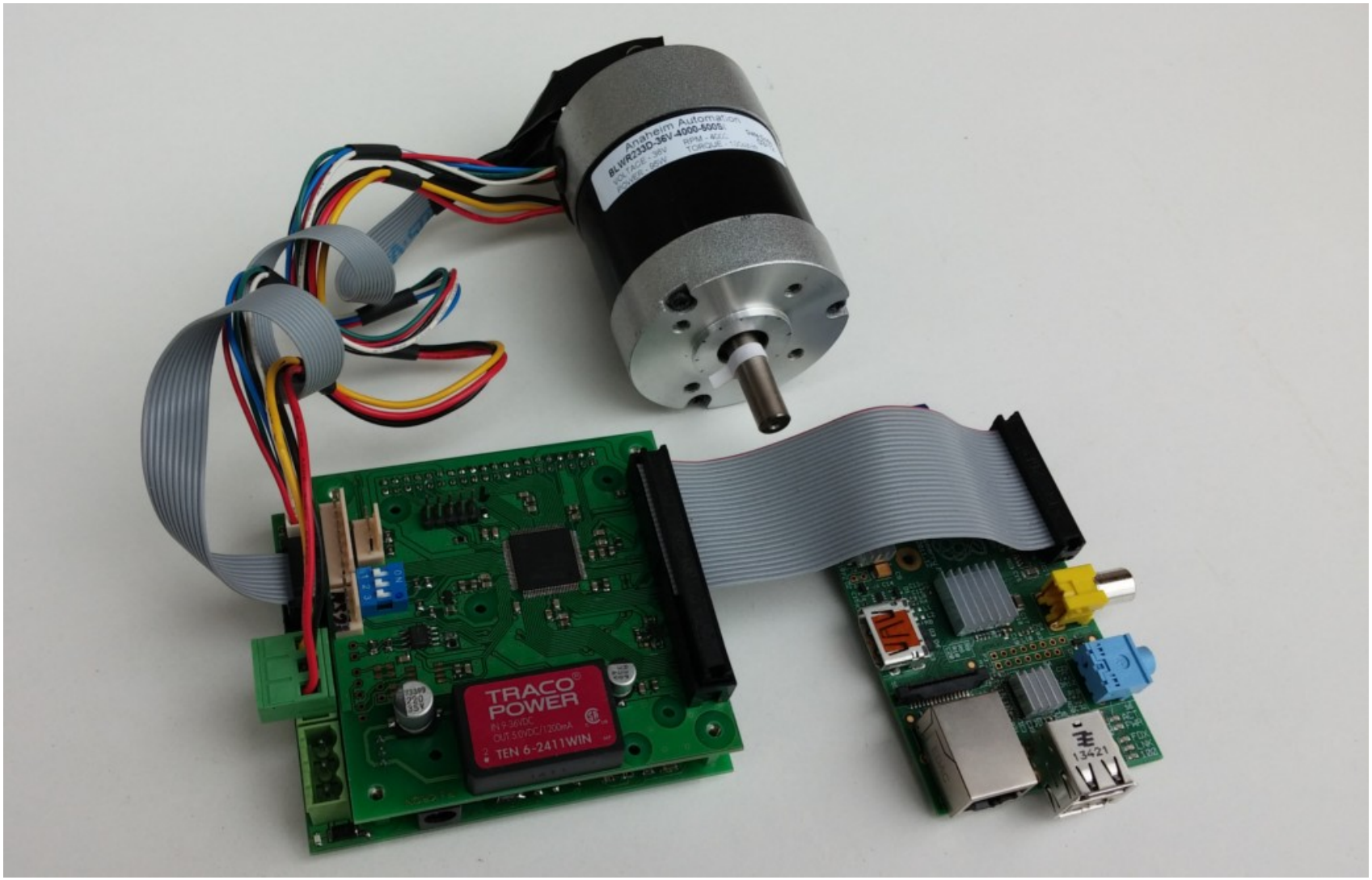


Hall Sensors to Sectored Position



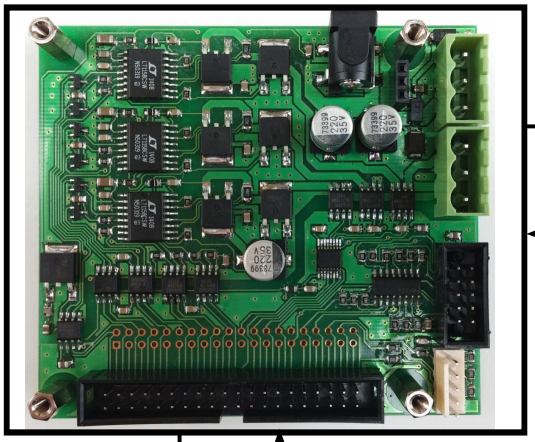
Outline

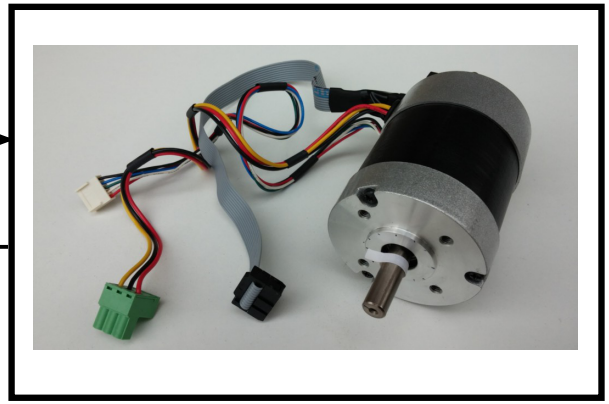
Experimental Setup for Thesis



Motor, Boards and Interconnection

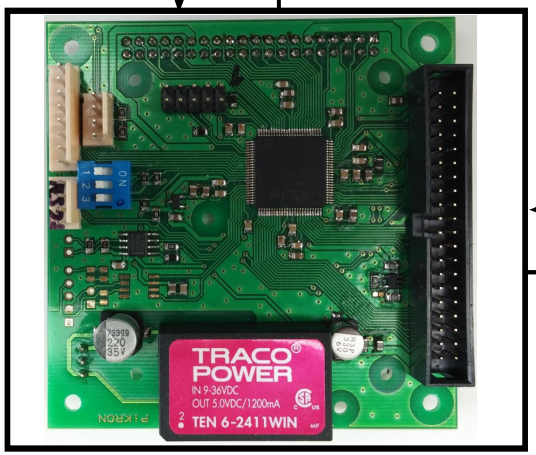
Motor driver

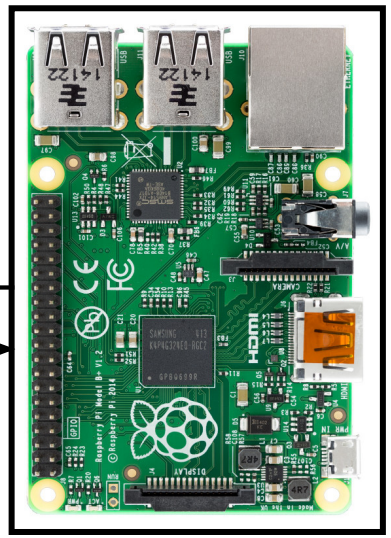




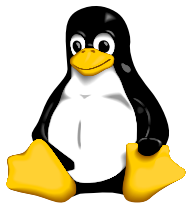
Motor

Expansion unit with FPGA

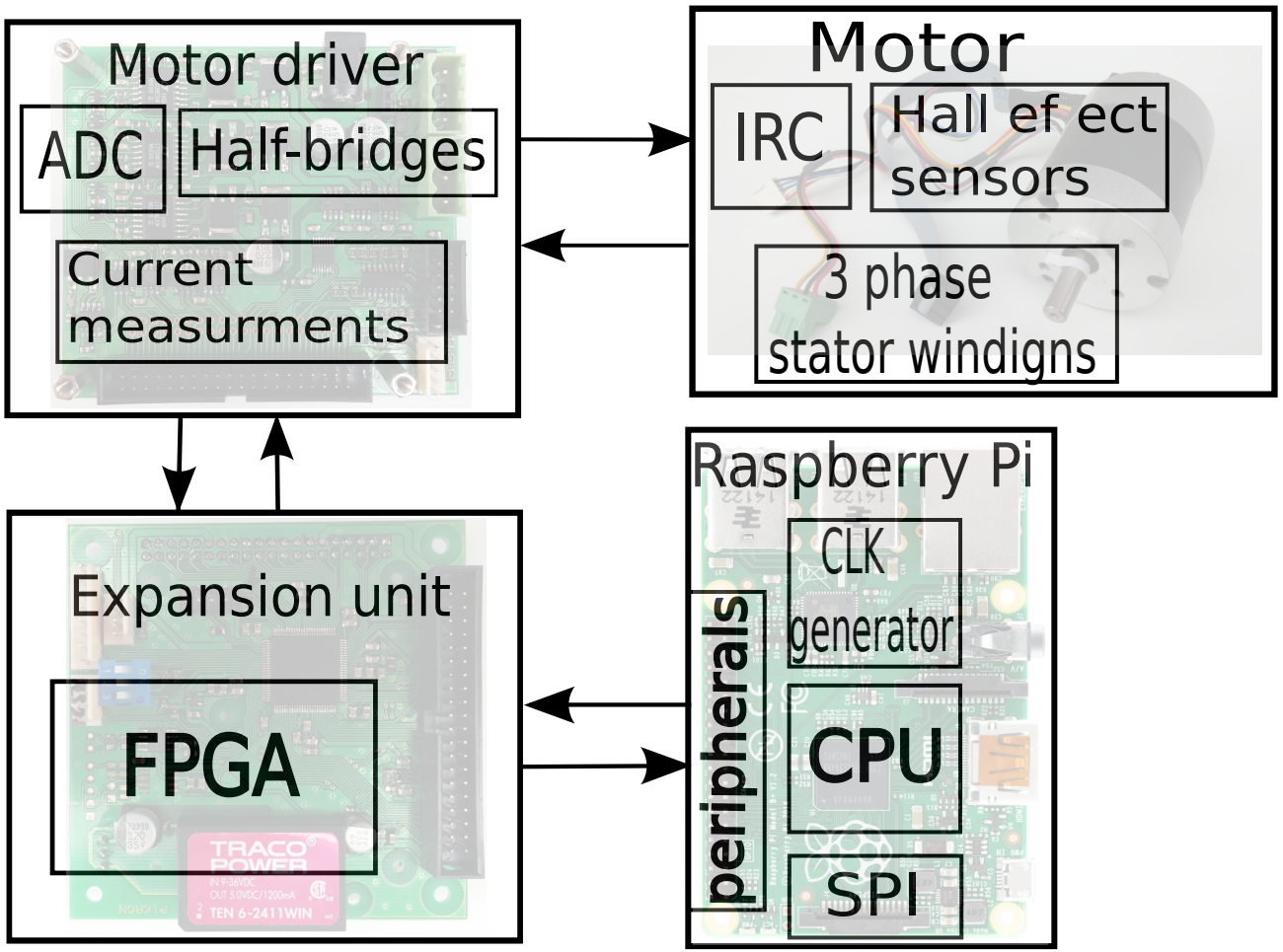




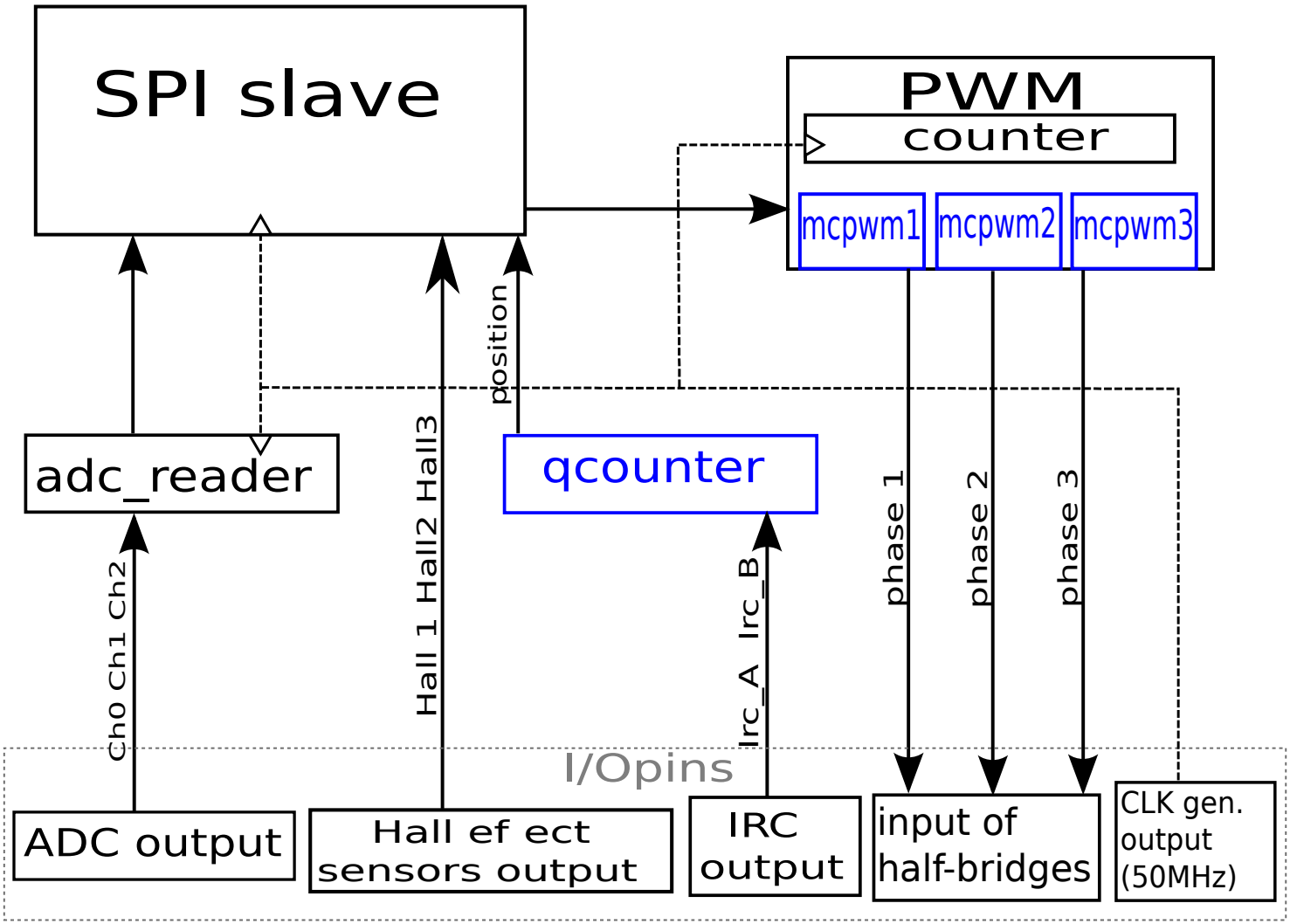
Raspberry Pi
+ Linux 3.18-RT



Function Placement to the Borads



FPGA Functional Blocks

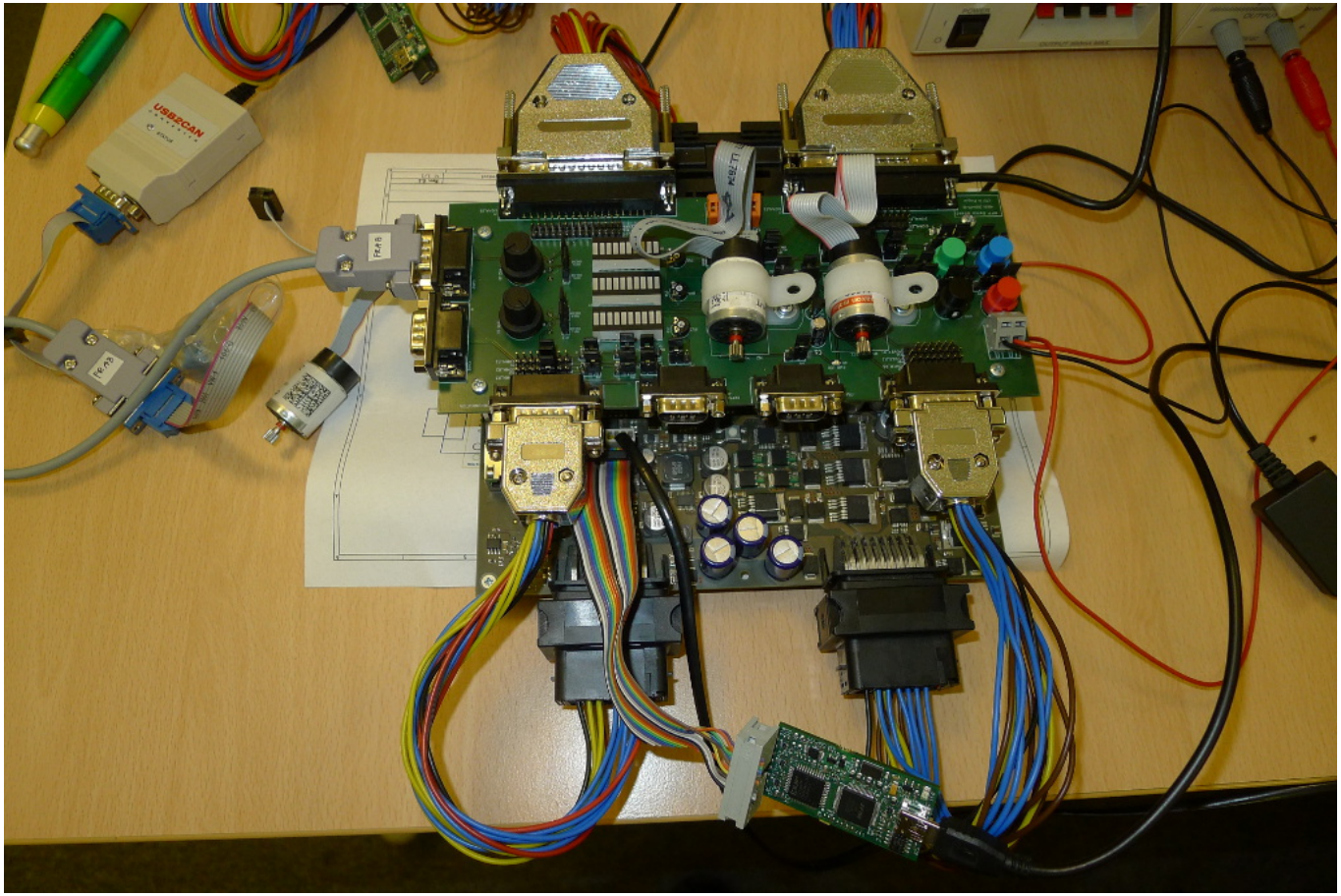


Outline

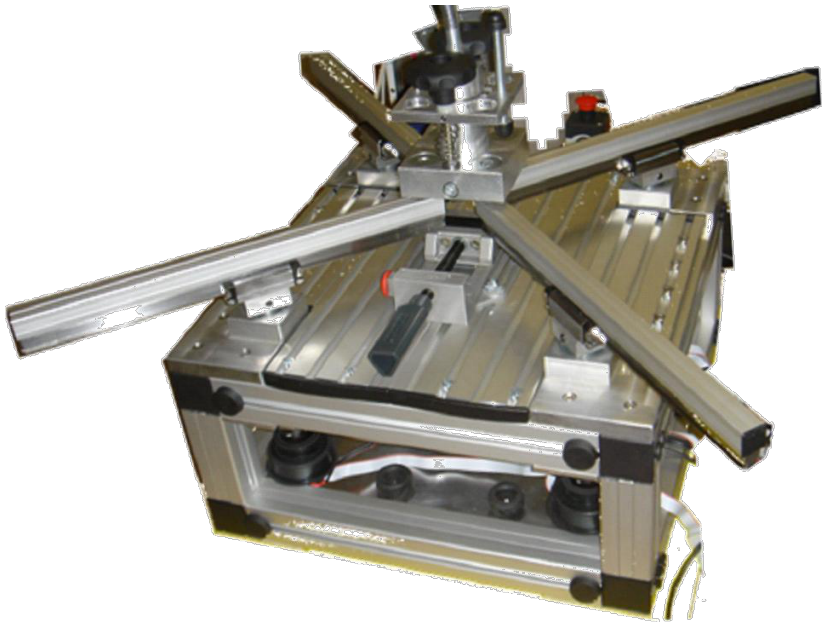
SocketCAN Simulink Blockset

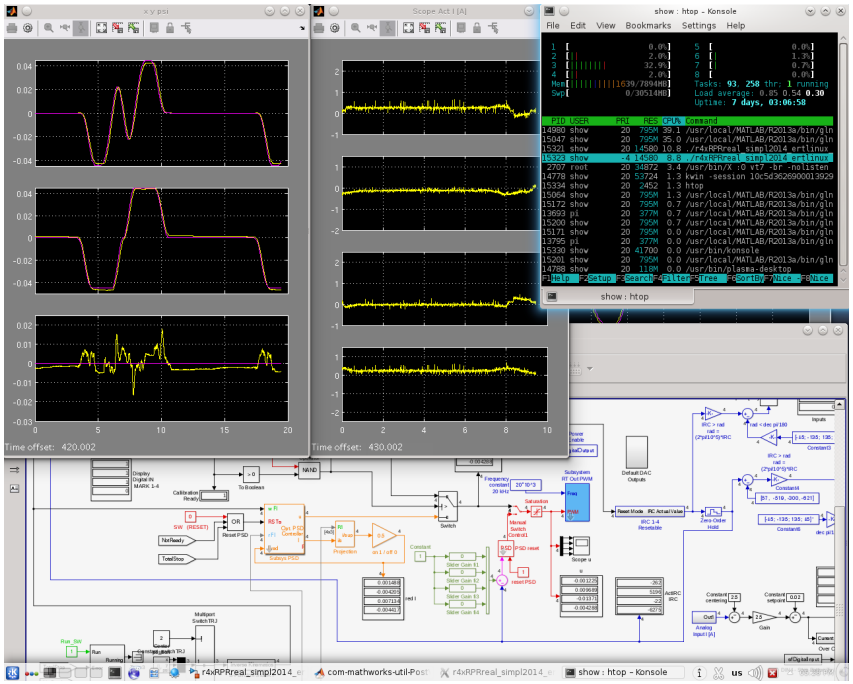
- The blockset is quick proof port of the CAN Autosar API based blocks developed at DCE initially for own automotive grade ARM Cortex-R4 based embedded platform
- The code is generated under designed control of TLC (Target Language Compiler) blocks description which allows to optimize blocks code for used data-types and interconnection
- For more information about embedded systems rapid prototyping support developed in our group look at `http://rtime.felk.cvut.cz/rpp-tms570/`
- Notices about more Linux and embedded hardware used, tested and even some designed look at `https://rtime.felk.cvut.cz/hw/`

Cortex-R4 Automotive Platform and Test Board



x86 Linux ERT and Parallel Kinematic Robot Control





- 4 DC motors, 4 incremental encoders, other I/Os
- Presented at Embedded world 2014
- Sampling period 1 ms but complex computations
- More reliable that previously used Windows target

Conclusion

- More ready to be uses open-source building blocks for control applications have been presented and are available online
- We are looking for students who has interest in real-time, operating systems and control/embedded hardware
- We cooperate with more industrial partners on many projects and students can gain experience and valuable knowledge during their work on the project in frame of thesis
- We offer control related courses Real-Time systems programming and participate on generic computer architectures courses at CTU FEE

Thanks for attention and questions