GNU/Linux, CAN and CANopen in Real-time Control Applications

Pavel Pisa
pisa@cmp.felk.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

2017-10-08 LinuxDays 2017

Content of Presentation

1. Introduction
2. CAN bus and GNU/Linux
   - Generic CAN and SocketCAN
   - QEMU SJA1000 Emulation
   - CAN on Real HW
3. Other Projects
   - Rapid Prototyping with Matlab/Simulink

Introduction CAN

The CAN (Control Area Network) is an generic term and concrete data link protocol and physical layer standardized by Robert Bosch GmbH in 1986. The main goal addresses by CAN bus solution is reduction of physical wires count in vehicles. CAN 1.0 allows to distinguish 2048 message types (signals) on the bus by attaching 11-bit identifier (ID, COBID) to up-to 8 data bytes. The extended format with 29-bit has been introduces in 1991 to address exhausted ID space in complex systems. The advantage is deterministic media arbitration and ID priority base collision avoidance/resolution. The price is relatively slow maximal speed 1 Mbit/s and network wires length limit proportional to speed (about 30 m for 1 Mbity/s). These limitations are relaxed in some level by emerging CAN FD (flexible data rate) ecosystem (max 64 data bytes, faster data transfer phase).

GNU/Linux and CAN

GNU/Linux is becoming most viable solution for many embedded systems today. Even professional routers, switches, TVs and even In-Vehicle Infotainment (IVI) systems and more and more even decision critical systems where high throughput and AI, GPU accelerated computations are required. Because CAN bus is core interconnect solution regarding control data exchange between Engine Control Units (ECU) a even for connection of telemetry systems and delivers process data to IVI systems, it is logical need that GNU/Linux based systems needs CAN bus interfacing. SocketCAN is such approach accepted to mainline Linux kernel sources tree.

Access to the CAN bus from virtual environment (i.e. QEMU) helps developers of the drivers and application. DeviceNet and CANopen higher level layers are used in control applications.

GNU/Linux and Real-Time

Linux kernel with fully preemptive support can be used for RT applications. The latencies bound/limited under 150 µfor ARM based systems and much better for x86 hardware without SMI BIOS issues (about 20 kHz).

RT-Summit 2017

The RT-Summit which will be held in Prague, Czech Republic, on October 21st, 2017 at the Czech Technical University (CTU) https://wiki.linuxfoundation.org/realtime/events/rt-summit2017/

Previous Presentations

- InstallFest 2015

Is Raspberry Pi Usable for Industrial and Robotic Applications?
`http://installfest.cz/if15/slides/pisa_rpi.pdf`

- LinuxDays 2015

Linux, RPi and other HW for DC and Brushless/PMSM Motor Control
`https://www.linuxdays.cz/2015/video/Pavel_Pisa-Rizeni_stejnosmernych_motoru.pdf`

- LinuxDays 2016

Processor Systems, GNU/Linux and Control Applications
`https://www.linuxdays.cz/2016/video/Pavel_Pisa-Procesorove_systemy_a_nejen_GNU_Linux_v_ridicich_aplikacich.pdf`

- InstallFest 2017

GNU/Linux and FPGA in Real-time Control Applications
`https://installfest.cz/if17/slides/so_t2_pisa_realtime.pdf`

Related Articles

- RTLWS 2014, OSADL, Sojka, M. – Píša, P.

Usable Simulink Embedded Coder Target for Linux
https://rtime.felk.cvut.cz/publications/public/ert_linux.pdf

- ROOT.CZ 9. 5. 2016

GNU/Linux pro řízení a rychlost jeho odezvy https://www.root.cz/clanky/gnu-linux-pro-rizeni-a-rychlost-jeho-odezvy/

- ROOT.CZ 3. 10. 2016

Linux pro řízení: minimalistické řešení řízení stejnosměrného motoru
https://www.root.cz/clanky/linux-pro-rizeni-minimalisticke-reseni-rizeni-stejnosmerneho-motoru/

Related Thesis Works

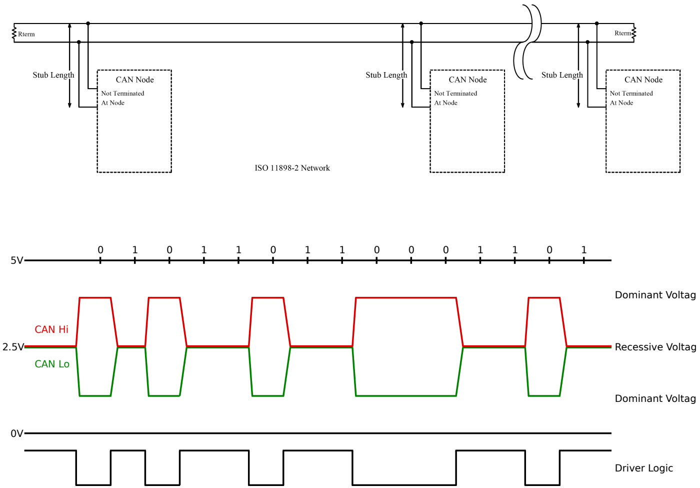
The most recent theses I have mentored in motion control, CAN communications and FPGA area

- Martin Meloun: FPGA Based Robotic Motion Control System, 2014 (PDF)
- Martin Prudek: Brushless motor control with Raspberry Pi board and Linux, 2015 (PDF)
- Tomáš Nepivoda: DC Motor Control Peripheral Module for Zynq Platform, 2016 (PDF)
- Martin Jeřábek: FPGA Based CAN Bus Channels Mutual Latency Tester and Evaluation, 2016 (PDF)
- Martin Prudek: Enhancing Raspberry Pi Target for Simulink to Meet Real-Time Latencies, 2017 (PDF)
- Martin Hofman: Time-Triggered Scheduler Implementation in Distributed Safety-Critical Control System (TMS570), 2017 (PDF)

Outline

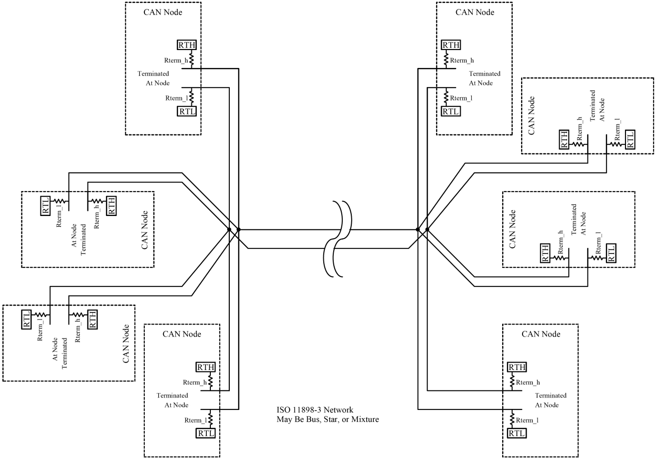
1. Introduction
2. CAN bus and GNU/Linux
   - Generic CAN and SocketCAN
   - QEMU SJA1000 Emulation
   - CAN on Real HW
3. Other Projects
   - Rapid Prototyping with Matlab/Simulink

CAN Bus Physical Layer



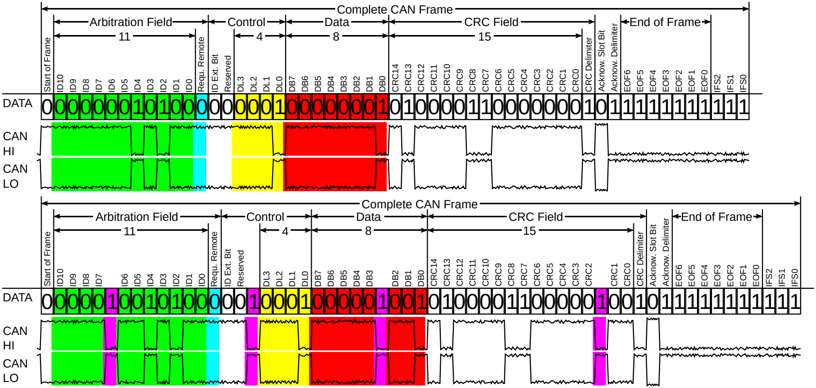
Source: https://en.wikipedia.org/wiki/CAN_bus

CAN Free Topology



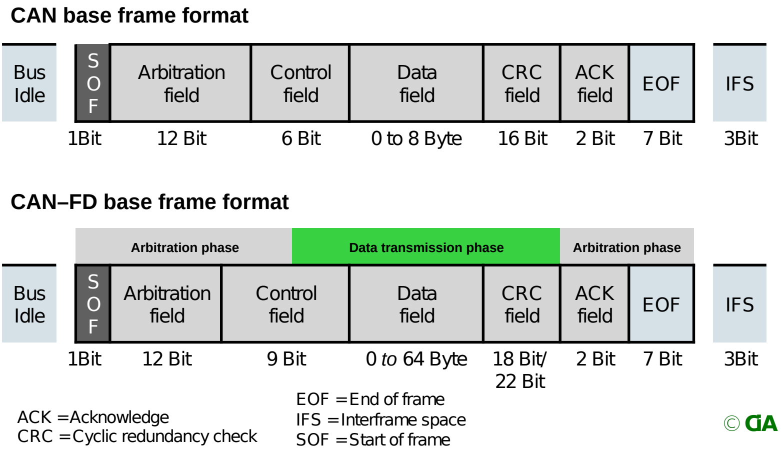
Source: https://en.wikipedia.org/wiki/CAN_bus

CAN Frame and Bit Stuffing



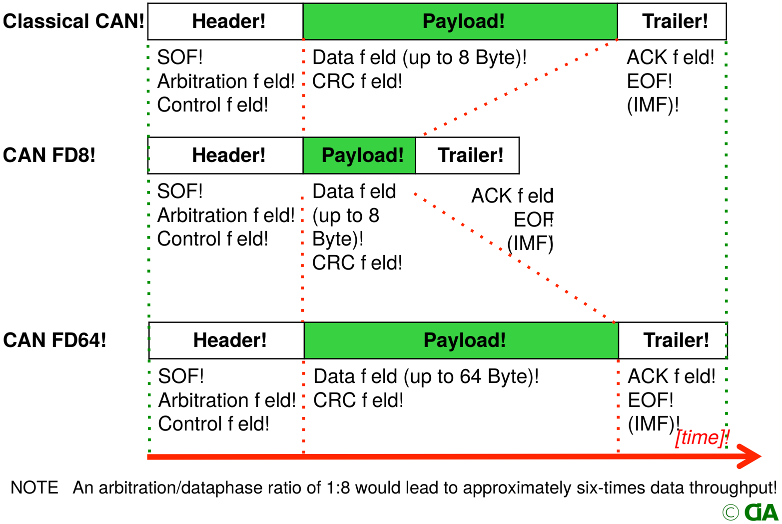
Source: https://en.wikipedia.org/wiki/CAN_bus

CAN FD (Flexible Datarate) Frame



Source: https://www.can-cia.org/

CAN FD Faster or More Data



Source: https://www.can-cia.org/

CAN FD Faster or More Data

- Supported frame lengths 0 to 8 and 12, 16, 20, 24, 32, 48, 64 bytes
- Arbitration phase 500 kbit/s, data phase 2 Mbit/s or more
- 16, 18 or 20 bit CRC
- Deprecated/no remote request frames

Linux Kernel and CAN Drivers

- Character driver API
  - Can4Linux
  - LinCAN
  - and more
- Network subsystem
  - SocketCAN
    accepted to mainline
- Our CAN projects and drivers throughput measurements
  http://rtime.felk.cvut.cz/can/

Send CAN Frame for SocketCAN – Open

```
int s;
struct sockaddr_can addr;
struct canfd_frame frame;
struct ifreq ifr;
const char *ifname = "can0";
if((s = socket(PF_CAN, SOCK_RAW, CAN_RAW)) < 0) {
  perror("opening failed");
  returns;
}
```

Send CAN Frame for SocketCAN - Open

```
strcpy(ifr.ifr_name, ifname);
ioctl(s, SIOCGIFINDEX, &ifr);
addr.can_family = AF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;
printf("%s at index %d\n",
        ifname,
        ifr.ifr_ifindex);
if(bind(s, (struct sockaddr *)&addr,
        sizeof(addr)) < 0) {
  perror("bind failed");
  return;
}
```

Send CAN Frame for SocketCAN - Open

```
frame.can_id  = 0x123;
frame.len = 12;
/* setup data                    */
frame.data[0] = 0x12;
/* nbytes (MTU) defines classic
** frame or CAN FD frame
*/
nbytes = write(s, &frame, sizeof(struct can_frame));
printf("Wrote %d bytes\n", nbytes);
```

Source: Koppe, U., Tiderko, J., MicroControl: CAN driver API – migration from Classical CAN to CAN FD
https://www.can-cia.org/fileadmin/resources/documents/proceedings/2017_koppe.pdf

Outline

1. Introduction

2. CAN bus and GNU/Linux
   - Generic CAN and SocketCAN
   - QEMU SJA1000 Emulation
   - CAN on Real HW

3. Other Projects
   - Rapid Prototyping with Matlab/Simulink

CAN Controller in QEMU Motivation

- The RTEMS community interrest to have extendable CAN subsystem
- GSoC slot to implement/port CAN subsystem granted by Google
- LinCAN driver initially considered
- But how core maintainers test results without the same HW
- How to ensure automated testing then
- New priority, provide testbench the first

Which CAN Controller to Start with?

- RTEMS supports broad range of systems and CPU architectures
- QEMU and Skyeye are mostly used for automated testing of the system – none of them supports industrial and automotive interfaces like CAN
- System specific tools are used too – e.g. TSIM for Aeroflex GR712RC SPARC with CAN controller emulation included but covers single target only
- The CAN infrastructure should be tested against all/more supported architectures during development
- SJA1000 CAN controller selected – well know, still often used, not directly tied to single CPU architecture
- Controller should be "placed" onto PCI/PCIe card to be plugable to more systems (x86, PowerPC, ARM and SPARC)

Actual Project Status

- Student Jin Yang finished the GSoC 2013 project (mentor Pavel Pisa)
- The basic PCI memory-mapped SJA1000 prototype implemented during GSoC
- Supported connection to Linux host system PF_CAN (SocketCAN)
- Then code has been cleaned at CTU
- Added emulation of existing HW card
  Kvaser PCI selected because we are familiar with it from LinCAN and other projects
- We keep the implementation up-to-date with QEMU stable releases
- Used only for Linux till now

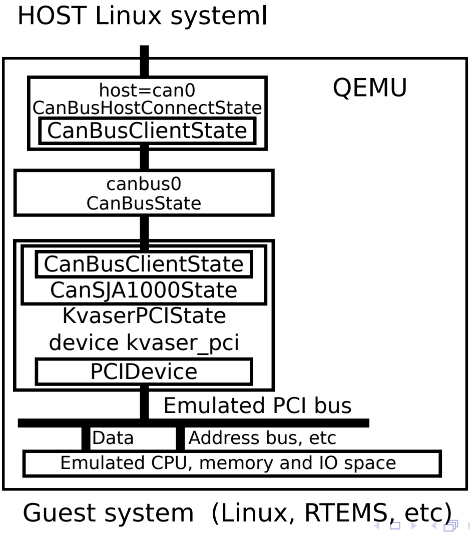
Why Broader Audience Can Be Interrested

- Enables automated testing of drivers and systems using CAN
- Enables tests of CAN applications in multi node environment
- Enables unmodified application, systems and drivers testing with virtual hardware
- If more controllers models implemented
  - Can help with development of drivers for not yet available HW when specification exists
  - There is significant milestone on CAN world horizon - CAN FD and CANopen FD – hardware is rare still but preparation for this major change has to start now

QEMU Architecture and Host CAN bus

- QEMU runs as user-space program on the host
- Hardware components represented by QEMU Object Model (QOM)
  based on GLib Objects (GTK+/GNOME origin)
- Device objects (QDev – structure `DeviceState`)
- Connected to buses (structure `BusState`).
- Object `PCIDevice` inherits from `QDev`
- If host = Linux
  CAN protocol/address family PF_CAN/AF_CAN
  (SocketCAN) allows access real (can0) or software only host virtual CAN bus (vcan0)

QEMU Emulated CAN Controller Device Architecture



QEMU CAN Device Representation

- Seen as PCI devices by the guest operating system
- Controllers groups (interconnection) represents virtual can buses
  group specified by parameter `canbus`
- Connection to host SocketCAN bus can be specified by `host` argument once per group
- Guest access CAN controller as set of registers
  - mapped into computer systems memory address space
  - represented as I/O ports
  - hidden behind index and data registers
- The SJA1000 single BAR memory space PCI device implemented the first (tested by LinCAN)
- Then complete Kvaser PCI CAN card with AMCC S5920 PCI bridge and I/O mapped SJA1000 implemented (mainline `kvaser_pci` driver compatible)

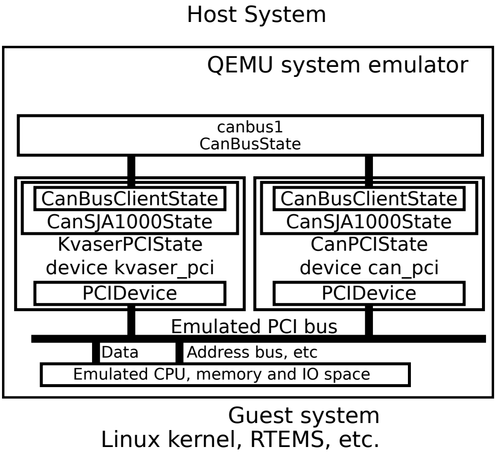
Setup of CAN Instance in QEMU

```
qemu-system-x86_64 -device kvaser_pci,canbus=canbus0,host=can0
```

- **-device** specify non platform implicit device (for CAN `pci_can` or `kvaser_pci`)
- **canbus=** which QEMU virtual CAN bus connect to (default `canbus0`)
- **host=** which host system CAN bus to connect to (usually `can0` or `vcan0` for virtual only one)
- **model=** for `pci_can` can allow choose chip model, `SJA1000` only for now

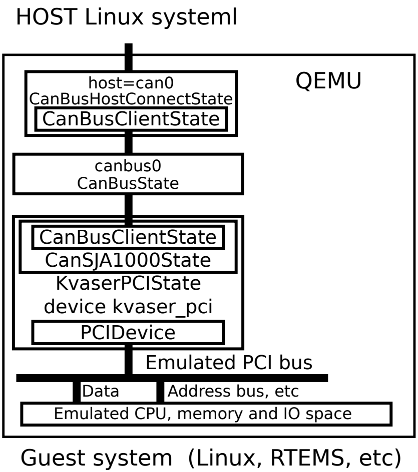
Two Interconnected CAN Controllers in QEMU

```
qemu -device kvaser_pci,canbus=canbus0 \
       -device can_pci,canbus=canbus0
```

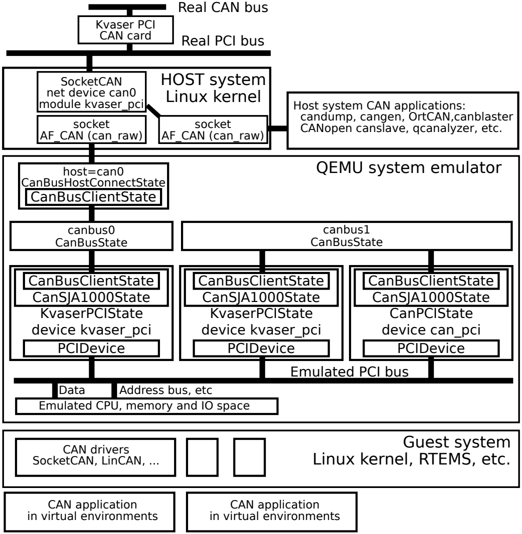


QEMU CAN Controller Connected to the Host

```
qemu -device kvaser_pci,canbus=canbus0,host=can0
```



Complex QEMU CAN Busses Setup



CAN or ARM QEMU Targets

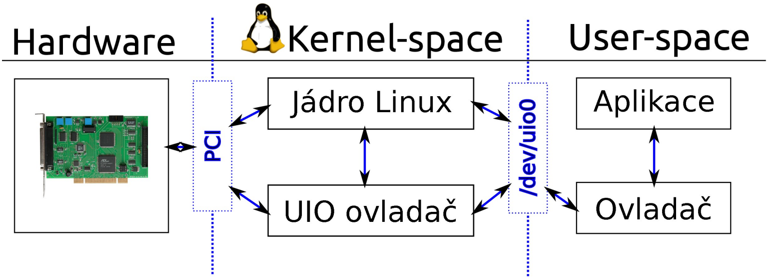
```
qemu-system-arm -cpu arm1176 \
  -m 256 -M versatilepb
```

- Cortex (`realview-pbx-a9` or `vexpress-a15`) for Debian armhf
- `xilinx-zynq-a9` interresting but without PCI in QEMU
- `virt` device tree specified machine hardware for QEMU
- BeagleBone and other if their controller model implemented in setup infrastructure

CANopen and Industrial I/O Devices

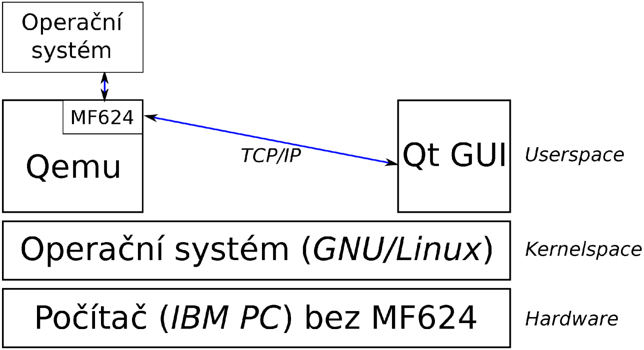
- Complete node emulation and SW stack testing
- CAN is the communication but there is other end – I/O terminals
- Example Humusoft MF624 - PCI multifunction I/O card
  - Supported by mainline UIO and Comedi
  - QEMU hardware model exists
- Experimental CANopen stack exists in OrtCAN project
- The CANslave program dictionary defined by EDS
- Connection to the hardware possible by shared libraries
- CommediHW.so writtent to demonstrate the complete setup

MF624/MF634 Drivers Support



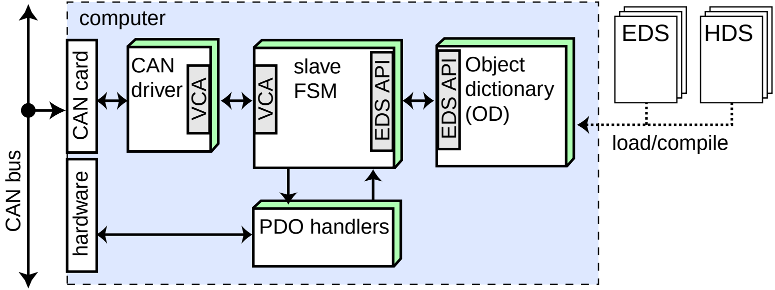
Rostislav Lisový: Prostředí pro výuku vývoje PCI ovladačů do operačního systému GNU/Linux, 2011, Departmet of Control Engineering, FEE, CTU https://support.dce.felk.cvut.cz/mediawiki/images/2/20/Dp_2011_lisovy_rostislav.pdf

MF624/MF634 as QEMU Virtual Hw



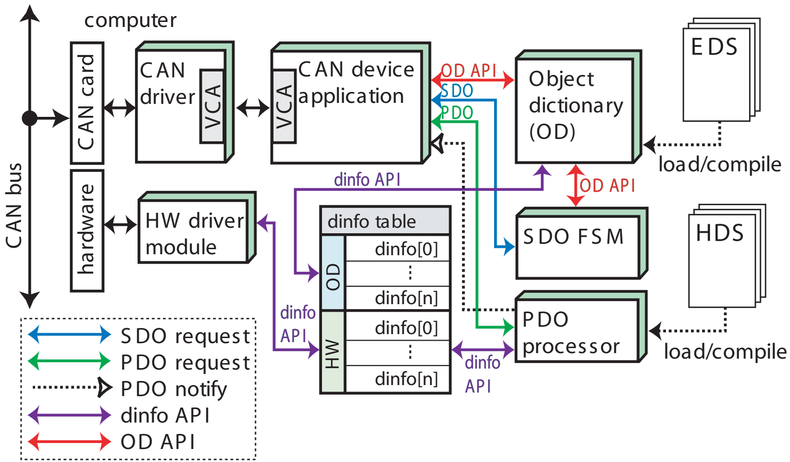
Rostislav Lisový: Prostředí pro výuku vývoje PCI ovladačů do operačního systému GNU/Linux, 2011, Departmet of Control Engineering, FEE, CTU `https://support.dce.felk.cvut.cz/mediawiki/images/2/20/Dp_2011_lisovy_rostislav.pdf`

CANopen Device Build from EDS

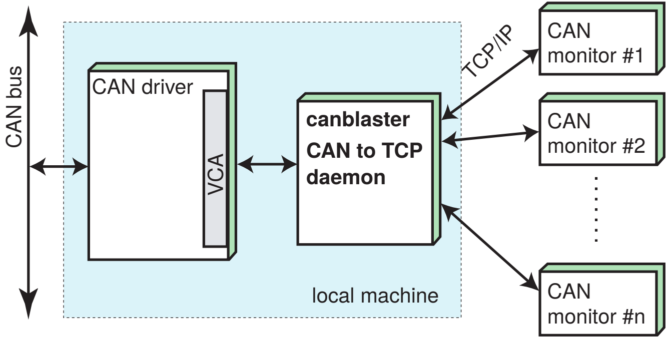


Frantisek Vacek: OCERA Project 2003, Departmet of Control Engineering, FEE, CTU, Separated as OrtCAN project `http://ortcan.sourceforge.net/`

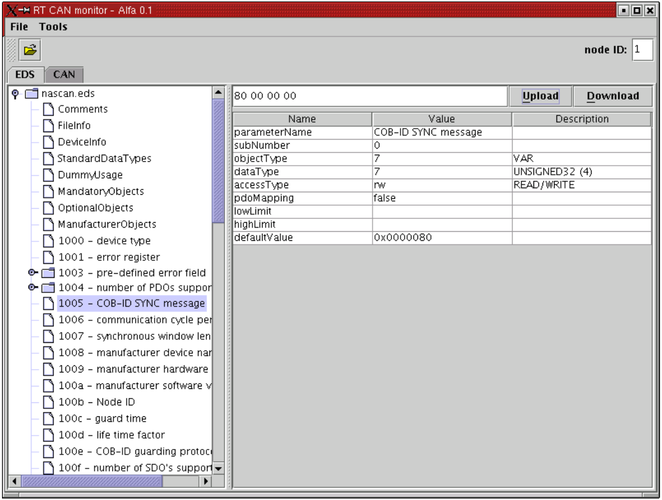
CANopen Device – Objects Dictionary



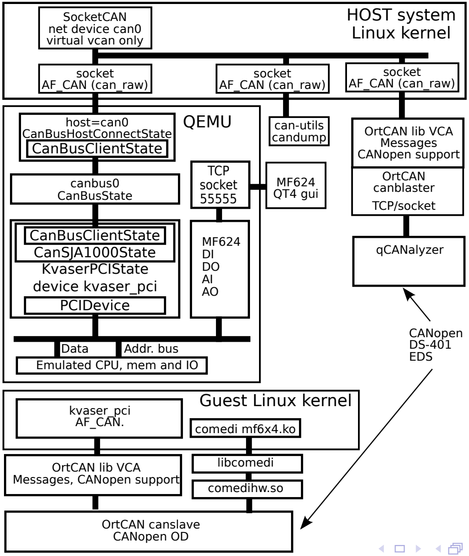
CANblaster – CAN/CANopen TCP Gateway



CANopen Java Based Monitor



QEMU, CAN, CANopen, Comedi, MF624 Example



Pointers to Other Related Projects

- CANopen and monitoring code
  `http://ortcan.sourceforge.net/`
- Virtual Humusoft MF624 data acquisition card
  P. Pisa, R. Lisovy, "COMEDI and UIO drivers for PCI Multifunction Data Acquisition and Generic I/O Cards and Their QEMU Virtual Hardware Equivalents", in *13th Real-Time Linux Workshop*, OSADL 2011

QEMU CAN Possible Enhancements and Questions

- Model SJA100 FIFO to hold more incoming messages
- Consider messages rate slowdown as on real CAN bus
- Some mechanism prevent to some limit lost of messages when guest application is slow
- Convert CAN bus model from plain C to QOM (Controllers are QOM/Qdev already)
- More CAN controllers model emulation (BOSCH/Ti C_CAN, Freescale FlexCAN, etc.)
- CAN FD (Flexible Datarate) controller emulation ???

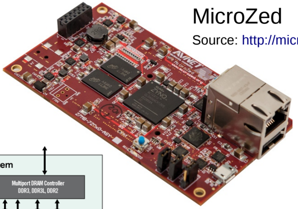
CAN QEMU Support Summary

- Code works for basic cases
- is maintained through more QEMU mainline releases
- is available – actual branches can-pci and merged-2.10
  `https://github.com/CTU-IIG/qemu`

Outline

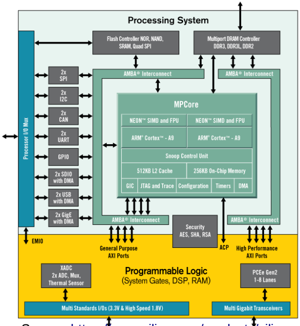
1. Introduction
2. CAN bus and GNU/Linux
   - Generic CAN and SocketCAN
   - QEMU SJA1000 Emulation
   - CAN on Real HW
3. Other Projects
   - Rapid Prototyping with Matlab/Simulink

MZ_APO – Kit for Education

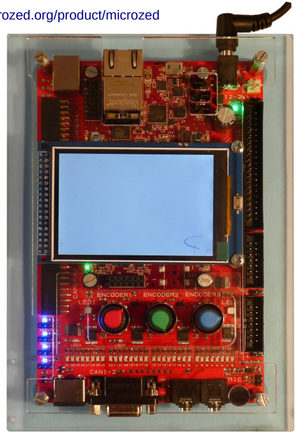


MicroZed

Source: http://microzed.org/product/microzed



Source: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html



Source: https://cw.fel.cvut.cz/wiki/courses/b35apo/start

The Core – MicroZed and System

- The core chip: Zynq-7000 All Programmable SoC
- Family member: Z-7010, device XC7Z010
- CPU: Dual ARM® Cortex™-A9 MPCore™ @ 866 MHz (NEON™ & Single / Double Precision Floating Point) 2× L1 32+32 kB, L2 512 KB
- FPGA: 28K Logic Cells (~430K ASIC logic gates, 35 kbit)
- Computational capability of FPGA DSP blocks: 100 GMACs
- Memory for FPGA design: 240 KB
- Memory on MicroZed board: 1GB
- Operating system: GNU/Linux
  - GNU LIBC (libc6) 2.19-18+deb8u7
  - Kernel: Linux 4.9.9-rt6-00002-ge6c7d1c
  - Distribution: Debian Jessie

MZ_APO Hardware Education Kit

- The kit is designed to be universal for multiple courses
  - Computer architectures courses (B3B35APO, B4B35APO)
  - Advanced Computer Architectures (B4M35PAP, BE4M35PAP)
  - Real -Time Systems Programming (B3M35PSR)
- Interfaces accessible directly on MicroZed module (single board computer SBC)
  - 1G ETHERNET
  - USB Host, connector A
  - serial port UART1 connected to USB to serial converter soldered on the module, USB micro-B
  - micro SD card slot
  - on-board Flash,
  - one user controlled LED
  - reset switch and user switch

MZ_APO Board Peripherals

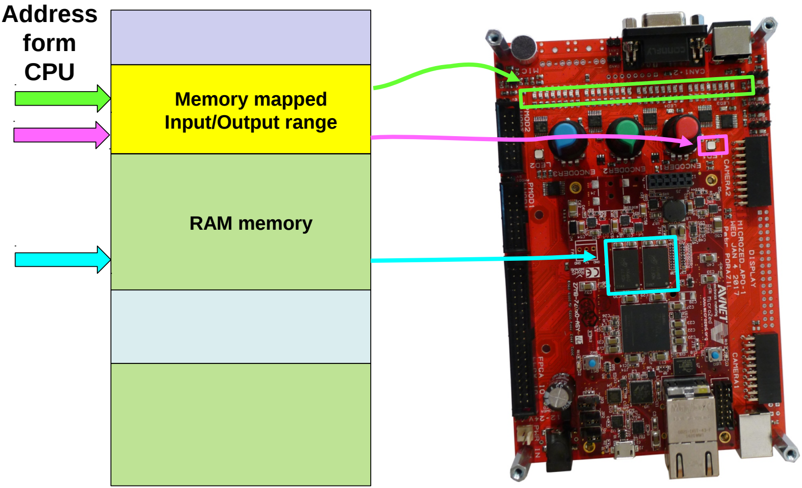
- Small 16-bit parallel bus connected LCD display ($480\times$ 320, RGB 565)
- 32 LEDs for direct visualization of 32-bit word (SPI connected), presented as single 32-bit register in physical memory address space
- $2\times$ RGB LED (SPI connected, 8-bit PWM), $2\times$32-bit register
- $3\times$ incremental encoder rotary knob (RGB 888, SPI connected) , presented as 3 three 8-bit fields in 32-bit word
- $4\times$ RC model servo interface (5 VDC power and pulse output, resolution 10 ns)
- $1\times$ 40 pin FPGA IO connector, 36 FPGA 3.3 VDC signals, jumper enables +5 VDC power, signals match Altera DE2 kits which we use in other courses
- $2\times$ PMOD connectors extended by optional +5 VDC power, each provides 8 FPGA signals shared with FPGA IO connector

MZ_APO Board Peripherals (cont.)

- 2× parallel camera interface, one 10-bit and one 8-bit
- 2× CAN bus transceivers, 5 Mbit/s capable, connected to Xilinx CAN peripheral, but FPGA CAN-FD possible in the future
- audio output by simple PWM modulator, on-board speaker and JACK available
- audio input to Xilinx integrated ADC, on-board microphone and JACK
- standard notebook power supply JACK 12 to 24 VDC
- UART0 serial port, galvanic isolated connection to FTDI serial to USB chip, robust USB A connector
- break signal longer than 1s resets the board

Full list at `https://cw.fel.cvut.cz/wiki/courses/b35apo/documentation/mz_apo/start`

Peripherals Mapped into Physical Address Space



Source: ČVUT FEL Computer Architectures Course https://cw.fel.cvut.cz/wiki/courses/b35apo/start

Knobs and LEDs Registers

Peripheral registers block at SPILED_REG_BASE_PHYS 0x43c40000

Register	Offset	Description
..._LED_LINE_o	0x04	32-bit LEDs line
..._LED_RGB1_o	0x10	The first RGB888 LED
..._LED_RGB2_o	0x14	The second RGB888 LED
..._LED_KBDWR_DIRECT_o	0x18	LED direct and key scan
..._KBDRD_KNOBS_DIRECT_o	0x20	raw key and knobs
..._KNOBS_8BIT_o	0x24	knobs processed as RGB888

PMSM/BLDC Peripheral registers

Peripheral registers block at Z3PMDRV1_REG_BASE_PHYS 0x43c20000

Register	Offset	Description
Z3PMDRV1_REG_IRC_POS_o	0x08	32-bit position
Z3PMDRV1_REG_IRC_IDX_POS_o	0x0C	position of index
Z3PMDRV1_REG_PWM1_o	0x10	14-bit PWM
Z3PMDRV1_REG_PWM2_o	0x14	bit 30 enable
Z3PMDRV1_REG_PWM3_o	0x18	bit 31 shutdown
Z3PMDRV1_REG_ADC_SQN_STAT_o	0x20	HAL and status bits
Z3PMDRV1_REG_ADC1_o	0x24	24 - bit ADC
Z3PMDRV1_REG_ADC2_o	0x28	cumulative
Z3PMDRV1_REG_ADC3_o	0x2C	sums

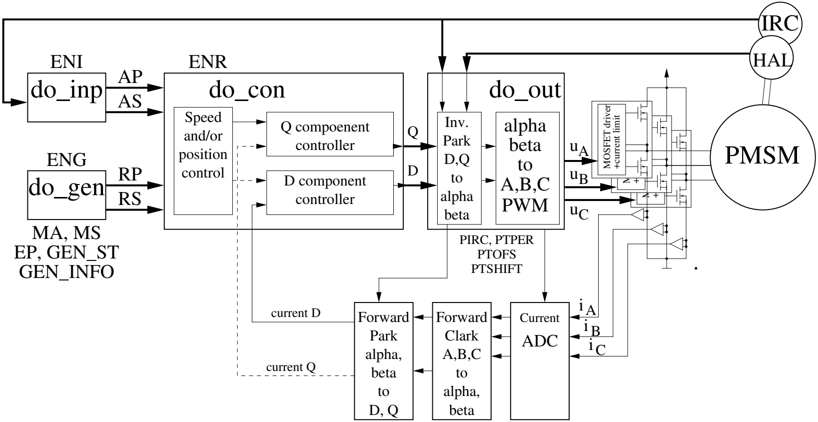
HAL and Status Register Bits

Symbol	Mask	Description
Z3PMDRV1_REG_ADSQST_SQN_m	0x000001FF	ADC cycles
Z3PMDRV1_REG_ADSQST_HAL1_m	0x00010000	HAL1
Z3PMDRV1_REG_ADSQST_HAL2_m	0x00020000	HAL2
Z3PMDRV1_REG_ADSQST_HAL3_m	0x00040000	HAL3
Z3PMDRV1_REG_ADSQST_ST1_m	0x00100000	Overload/ error signal phase driver
Z3PMDRV1_REG_ADSQST_ST2_m	0x00200000	
Z3PMDRV1_REG_ADSQST_ST3_m	0x00400000	
Z3PMDRV1_REG_ADSQST_PWST_m	0x01000000	power present

Actual Code Sources

- VHDL sources of SPI connected PMSM peripheral for Raspberry Pi
  `https://rtime.felk.cvut.cz/gitweb/fpga/rpi-motor-control.git/tree/HEAD:/pmsm-control`
- Repository with experimental sources with C based motion control code port, only for testing, locking and synchronization not complete for SMP systems
  `https://rtime.felk.cvut.cz/gitweb/fpga/rpi-motor-control-pxmc.git/blob/HEAD:/src/app/rpi-pmsm-test1/zynq_3pmdrv1_mc.c`
- Matlab/Simulink Lintaget pages
  `http://lintarget.sourceforge.net/`
- Matlab/Simulink model for Raspberry Pi and Xilinx Zynq
  `https://github.com/ppisa/rpi-rt-control/tree/master/simulink`

PXMC – DC, Stepper, PMSM Motion Control Library



PiKRON s.r.o. `http://www.pikron.com/`

project page `http://www.pxmc.org/`

Outline

1. Introduction
2. CAN bus and GNU/Linux
   - Generic CAN and SocketCAN
   - QEMU SJA1000 Emulation
   - CAN on Real HW
3. Other Projects
   - Rapid Prototyping with Matlab/Simulink

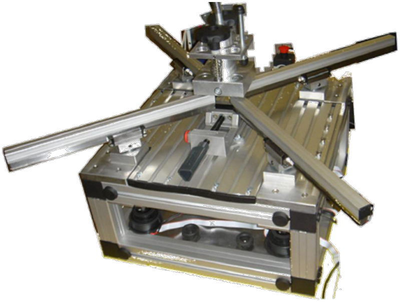
Embedded Real Time and the DCE Department

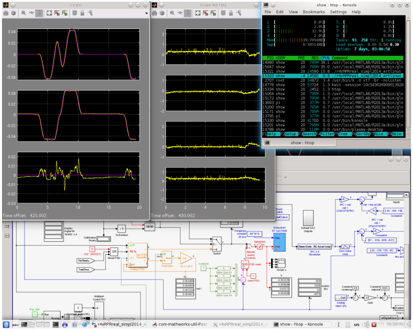
- CTU FEE Department of Control Engineering has been and is involved in Matlab/Simulink real-time support from start (origin of real-time toolbox can be trace to our department)
- We have long term experience with fully preemptive kernel and hardware interfacing
- Embedded Real-time Target has been adapted/partially 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 peripherals has been implemented
- COMEDI blockset has been updated and tested with our Linux ERT version as well

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/`

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