QEMU CAN Controller Emulation with Connection to a Host System

Pavel Píša, Jin Yang, Michal Sojka

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

17th Real-Time Linux Workshop
October 22, 2015
Graz, Austria

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 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

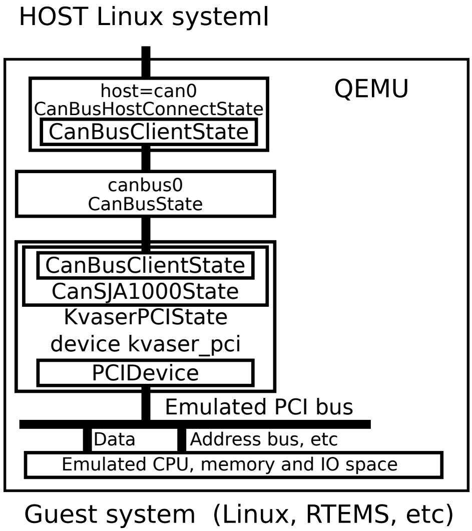
Why Broader Audience Can Be Interrested

- 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)

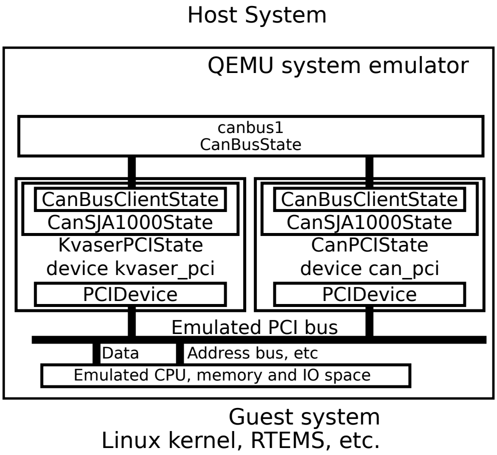
Setup of CAN Instance in QEMU

```
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

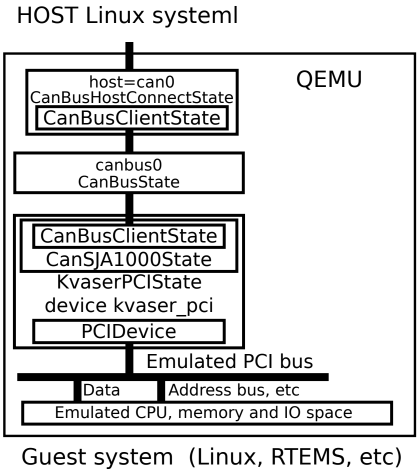
Two Interconnected CAN Controllers in QEMU

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

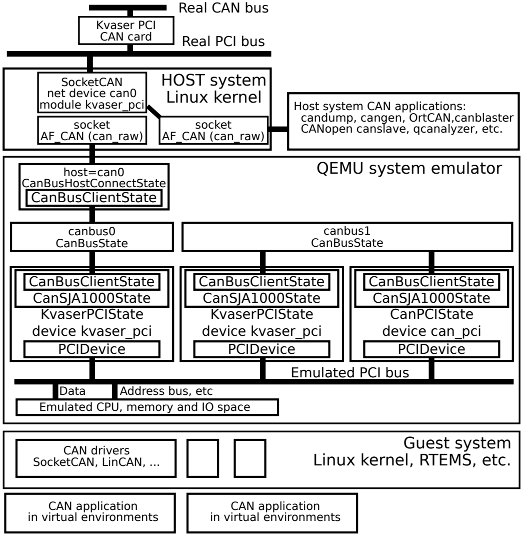


QEMU CAN Controller Connected to the Host

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



Complex QEMU CAN Busses Setup



CAN or ARM QEMU Targets

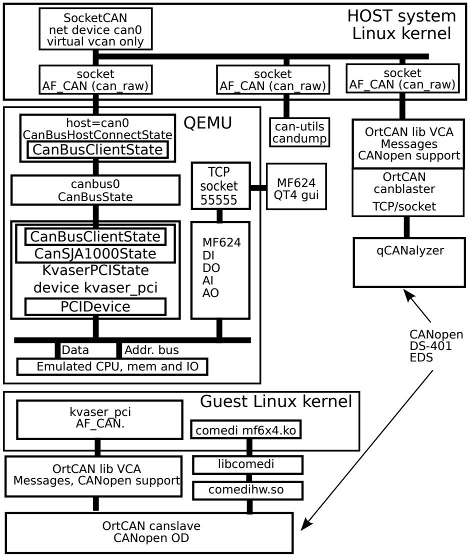
```
qemu-system-arm -cpu arm1176 \
  -m 256 -M versatilepb
```

- 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 data acquisition 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

Complete 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 ???

Concussion

- Code works for basic cases
- is maintained through more QEMU mainline releases
- is available – actual branches can-pci and merged-2.4
  `https://github.com/CTU-IIG/qemu`

Thanks for attention

Place for your questions and feedback