Continuous CAN Bus Subsystem Latency Evaluation and Stress Testing on GNU/Linux-Based Systems

Embedded World Conference 2024, Session 2.3 – Connectivity Solutions

Czech Technical University in Prague
Faculty of Electrical Engineering

More information: https://canbus.pages.fel.cvut.cz/

Pavel Píša <pisa@fel.cvut.cz>
Jiří Novák <jnovak@fel.cvut.cz>
Pavel Hronek <hronepa1@fel.cvut.cz>
Matěj Vasilevski <matej.vasilevski@gmail.com>

Presentation Outline

1) CAN Latency Tester History

2) CAN BENCH Board and XCAN HW Timestamps

3) Precise Timestamps on CTU CAN FD IP Core

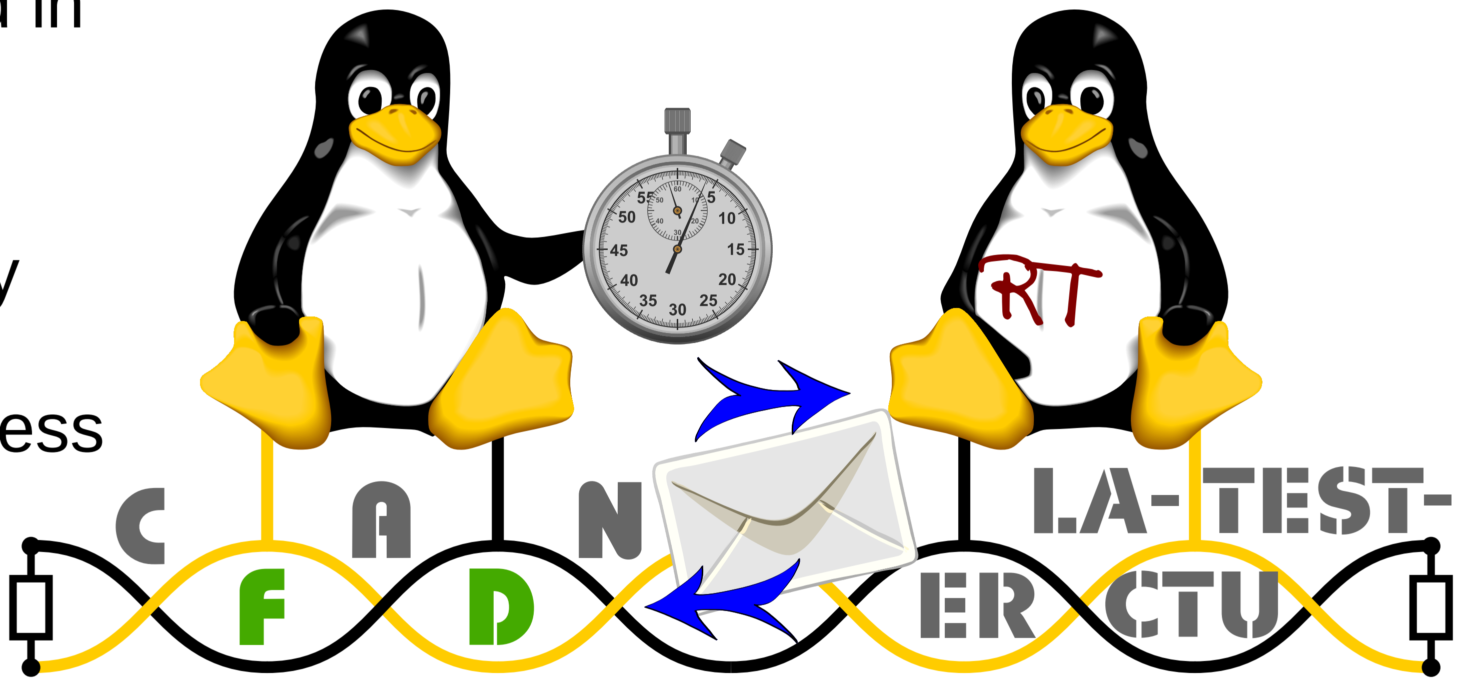
4) New Automated Testing Setup and Web Site

5) Results and Conclusion

1. CAN Latency Tester History

CAN Latency Benchmark of Linux and RTOS Kernels

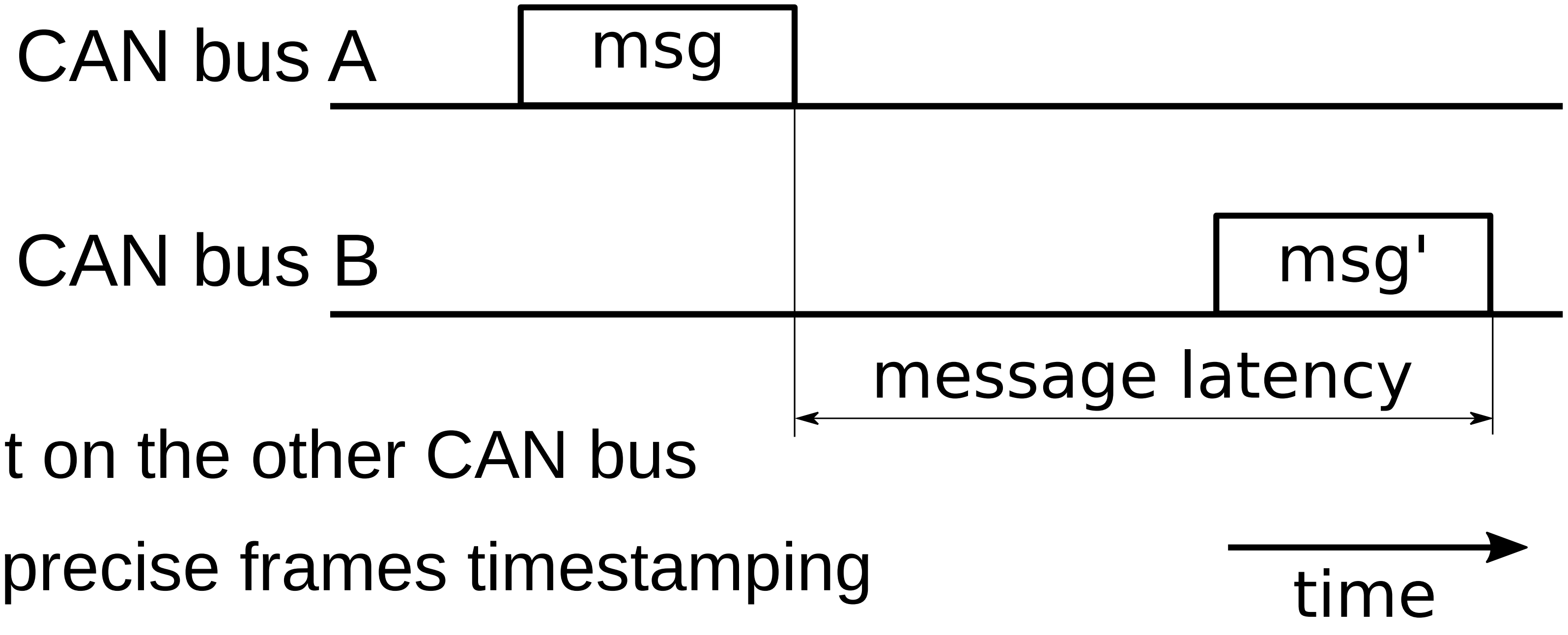
- CAN/CAN FD is often used in vehicles, industrial, and or motion control systems
- Control systems complexity grows → often full featured POSIX kernel used to process high level control, i.e. coordinate GPGPU for pedestrian detection and then act according the result



- But what is the level of knowledge of integrators about maximal latencies?
- OSADL.org QA Farm on Real-time of Mainline Linux fills gap for CPU reactions and ETHERNET communications, but what about the CAN?

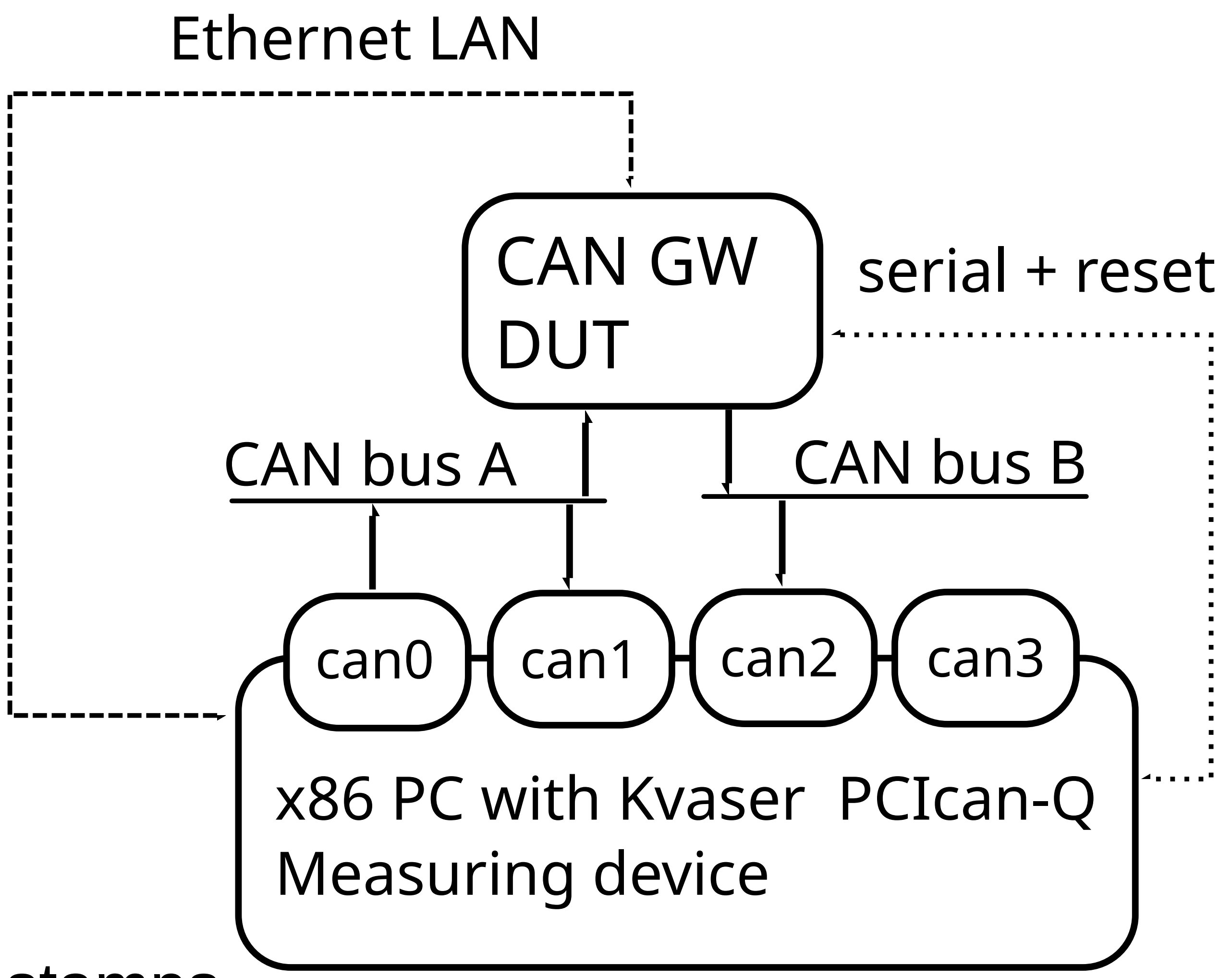
CAN Gateway Latency Testing/Benchmarking

- Pavel Pisa at CTU designed LinCAN in 2003 with CANping companion
- SocketCAN drivers benchmarking added later
- Contract to provide benchmarking of SocketCAN in kernel CAN Gateway from Volkswagen Research 2011
- Simple setup
- Generator and measurement system sends CAN frame
- Device under the test receives frame and send it on the other CAN bus
- Measuring system needs precise frames timestamping



CAN Latester Setup with x86 PC for Measurement

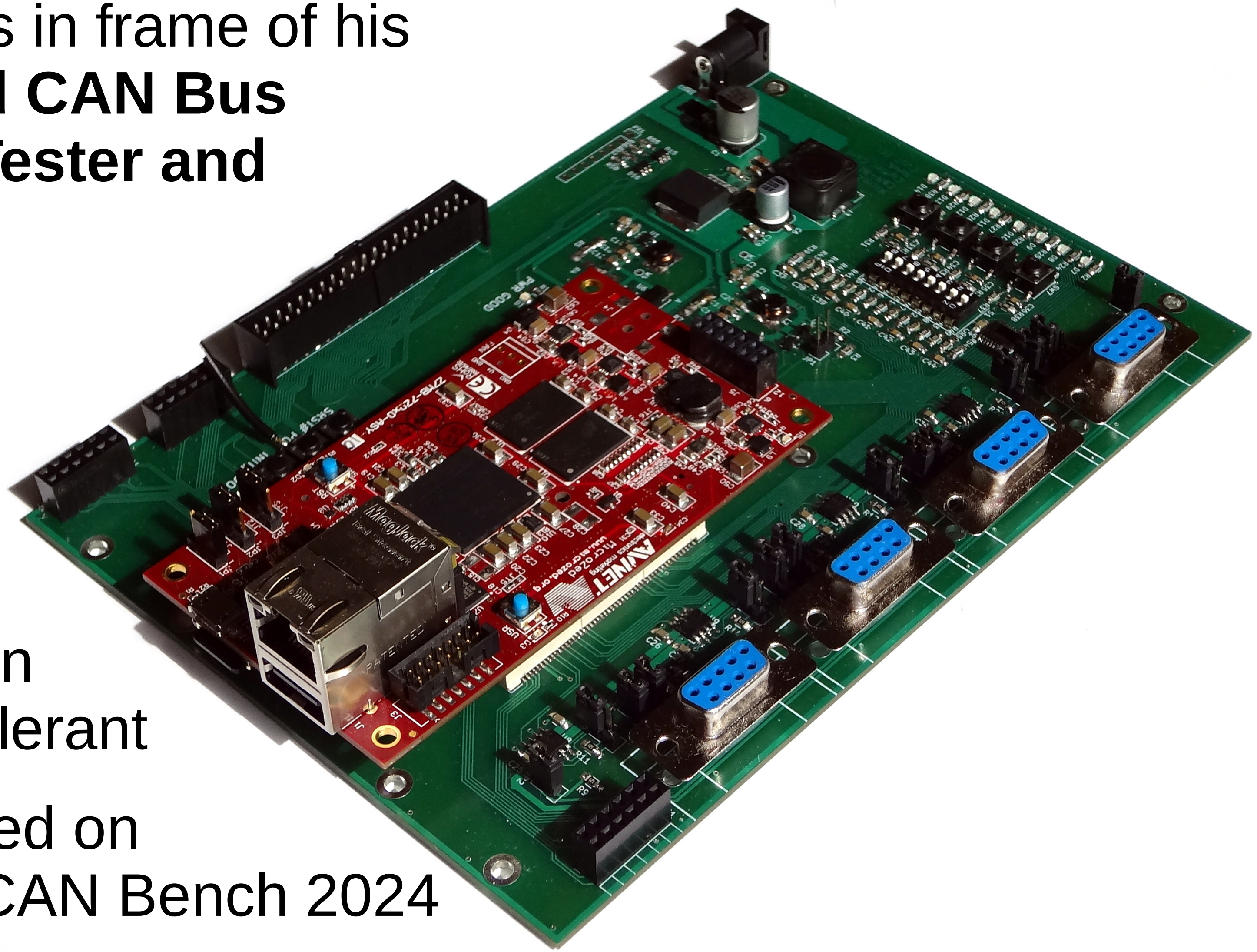
- Original CANping evolved into CANlatester
- x86 PC as measuring device preempt RT Linux kernel
- CAN frame sent by can0 received by can1 – timestamp assigned driver function run in ISR
- Devices under the test:
  x86 PC (SocketCAN, LinCAN)
  MPC5000 (SocketCAN, LinCAN, RTEMS)
- Demand for upgrade with HW timestamps



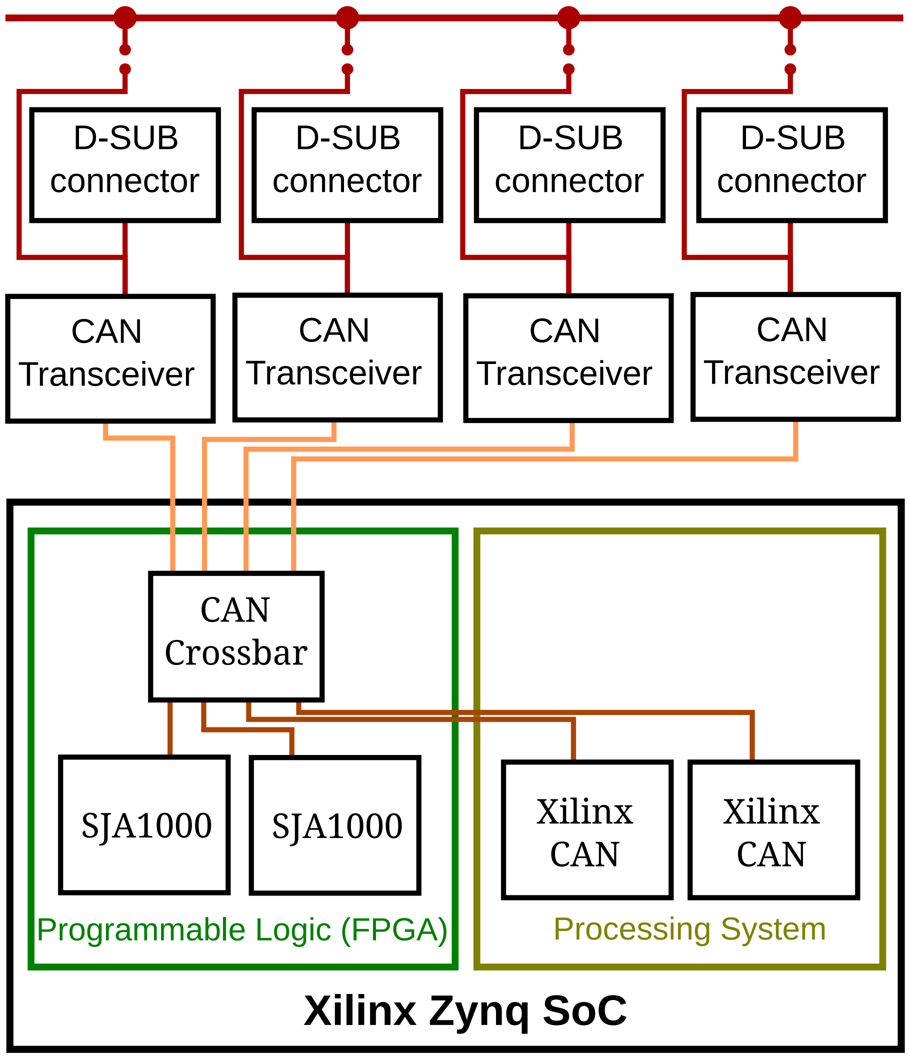
2. CAN BENCH Board and XCAN HW Timestamps

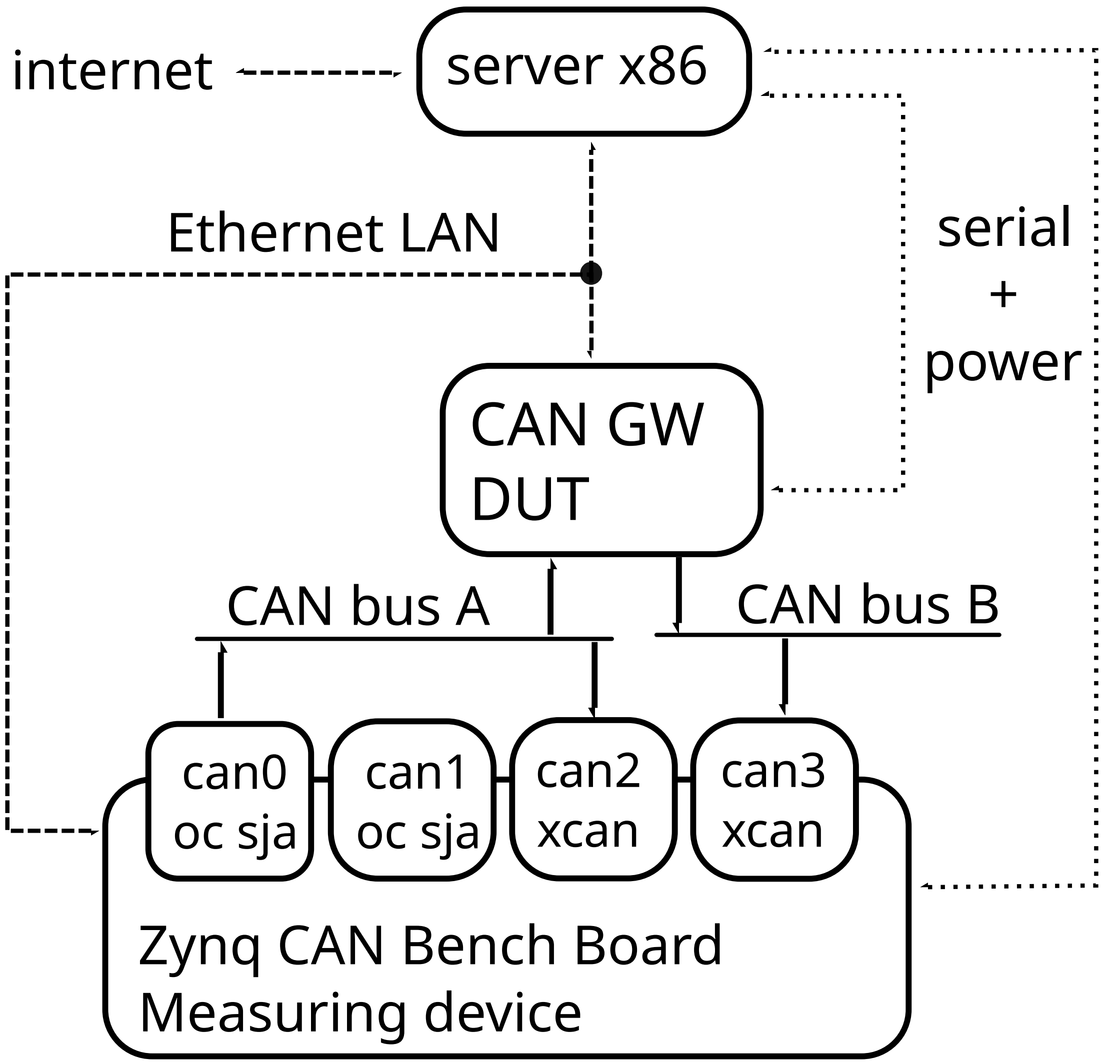
AMD/Xilinx Zynq 7000 Based CAN-BENCH Board

- Designed by Martin Jeřábek's in frame of his bachelor thesis **FPGA Based CAN Bus Channels Mutual Latency Tester and Evaluation** (2016)
- Design guidance Pavel Píša and Petr Porazil from PiKRON.com
- 4x CAN FD transceiver
- Initially 2x XCAN and 2x Open Cores SJA1000 s CTU FD tolerant
- CTU CAN FD IP core designed on MZ_APO in 2018, ported to CAN Bench 2024

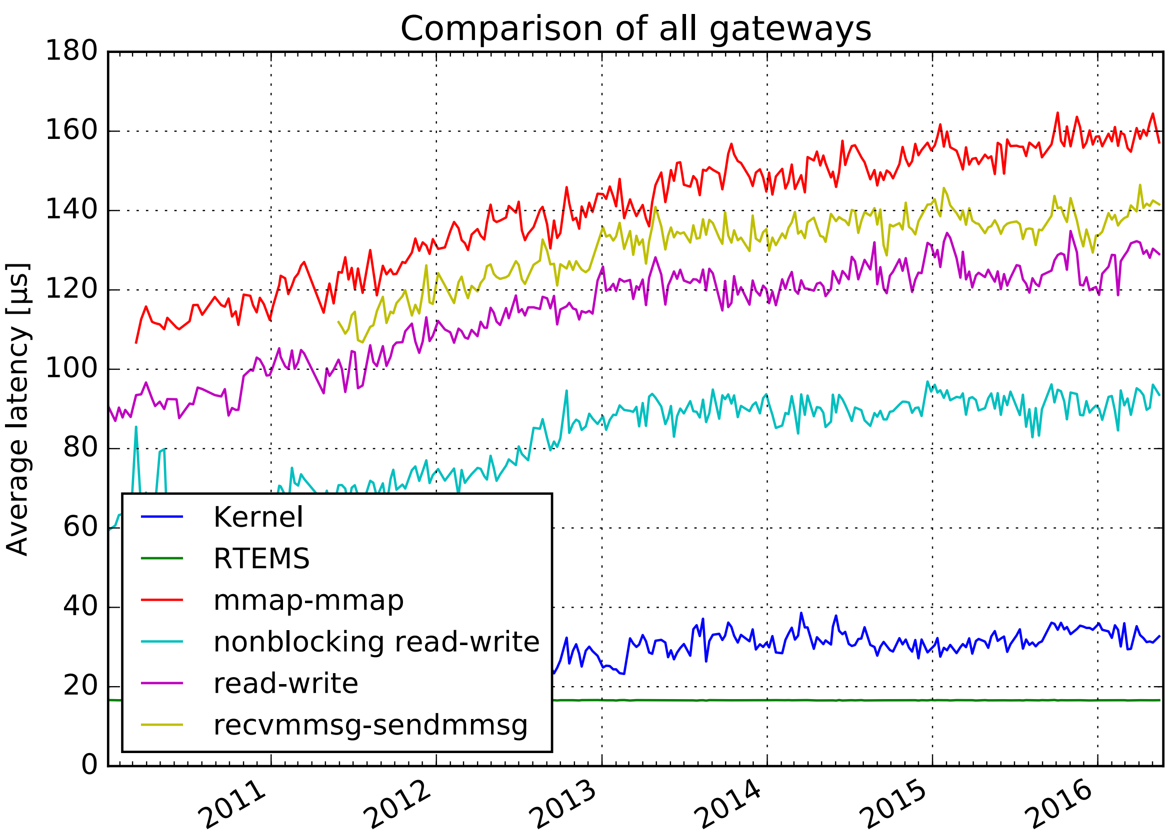


Original CAN Bench Board FPGA Design and Setup

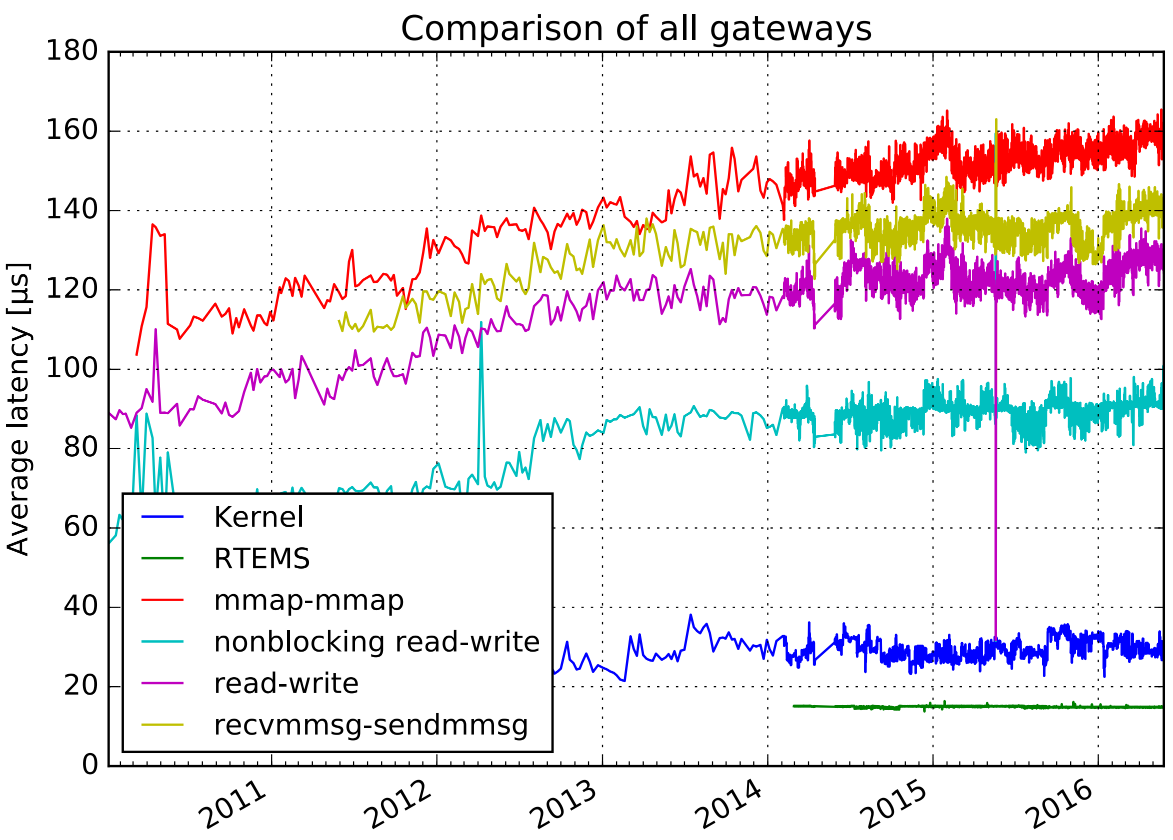




Original CAN Bench Re-run of MPC5000 Tests



Measurements from CAN Bench



Measurementsfromoriginalsystem (x86)

- All tagged Linux kernel versions (releases and release candidates) from 2.6.33-rc1 to 4.6 were tested on MPC5000 PowerPC board (DUT)

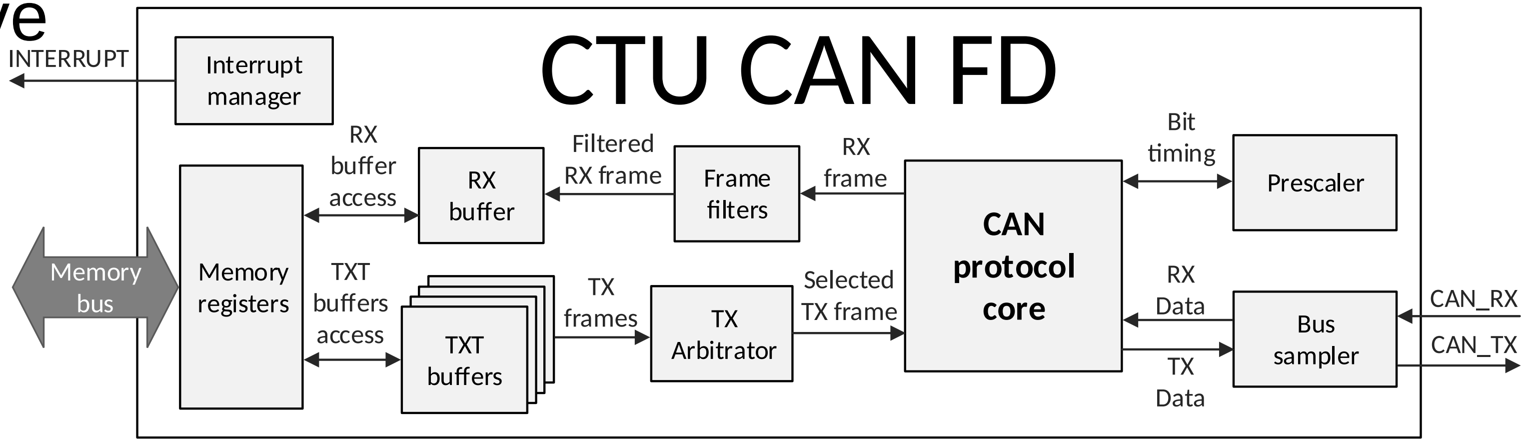
3. Precise Timestamps on CTU CAN FD IP Core

CTU CAN FD IP Core – History

- Initial design for new generation of special purpose CAN/CAN FD testers for Škoda Auto by Ondřej Ille at CTU FEE Department of Measurement under lead of Dr. doc. Jiří Novák (2015 – 2016)
- Re-architected, open-sourced and extended to general purpose can controller under Dr. Pavel Piša lead, funded by Digiteq Automotive.
- Linux kernel driver by Martin Jeřábek and Pavel Píša
- Mainlined in 2022 (Linux kernel version 5.19)
- Hardware timestamping for Linux driver implemented by Matějj Vasilevski in 2022 (has not been accepted into mainline yet), latester for CAN FD
- Ondřej Ille continues to work on ISO11898-1 2015 compliance testing framework and evaluation and full coverage of the CTU CAN FD IP core
- Michal Lenc ported CTU CAN FD driver as example and initial test target for new RTEMS CAN/CAN FD subsystem, mentored by Dr. Pavel Píša

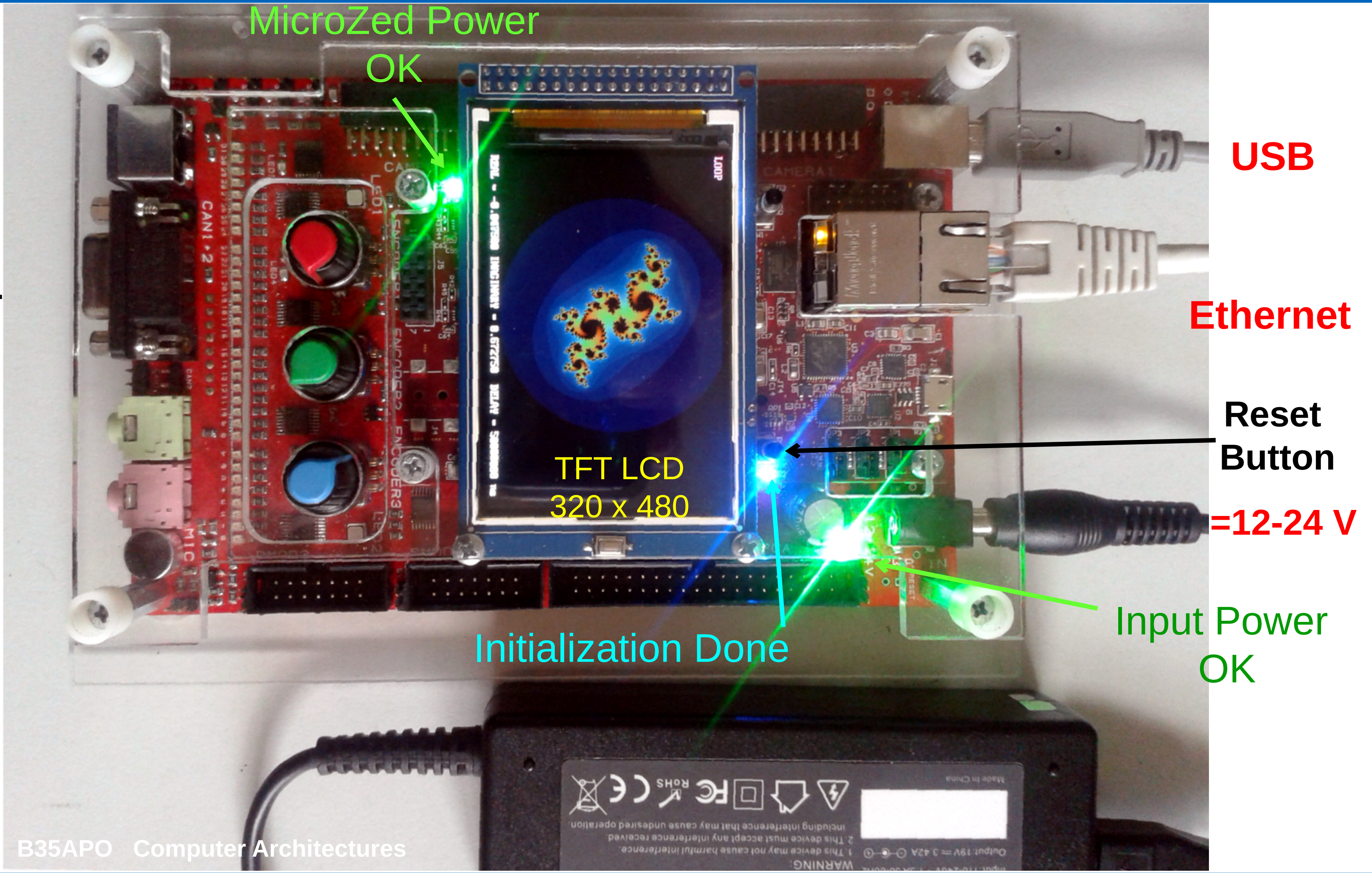
CTU CAN FD IP Core – Features

- VHDL design with no vendor-specific libraries required, yet RAM for buffers and Rx FIFO automatically inferred by Xilinx and Intel tools
- Compliant with ISO11898-1 2015
- RX buffer FIFO with 32 - 4096 words (1-204 CAN FD frames with 64 bytes of data)
- 2-8 TXT buffers (1 CAN FD frame in each TXT buffer), abort, SW Tx priority
- 32-bit APB, AHB, mem. slave
- Interrupts
- ISO and non-ISO CAN FD
- Timestamping
- Time triggered transmission
- Loopback mode, bus monitoring mode, ACK forbidden mode, self-test mode, restricted operation mode

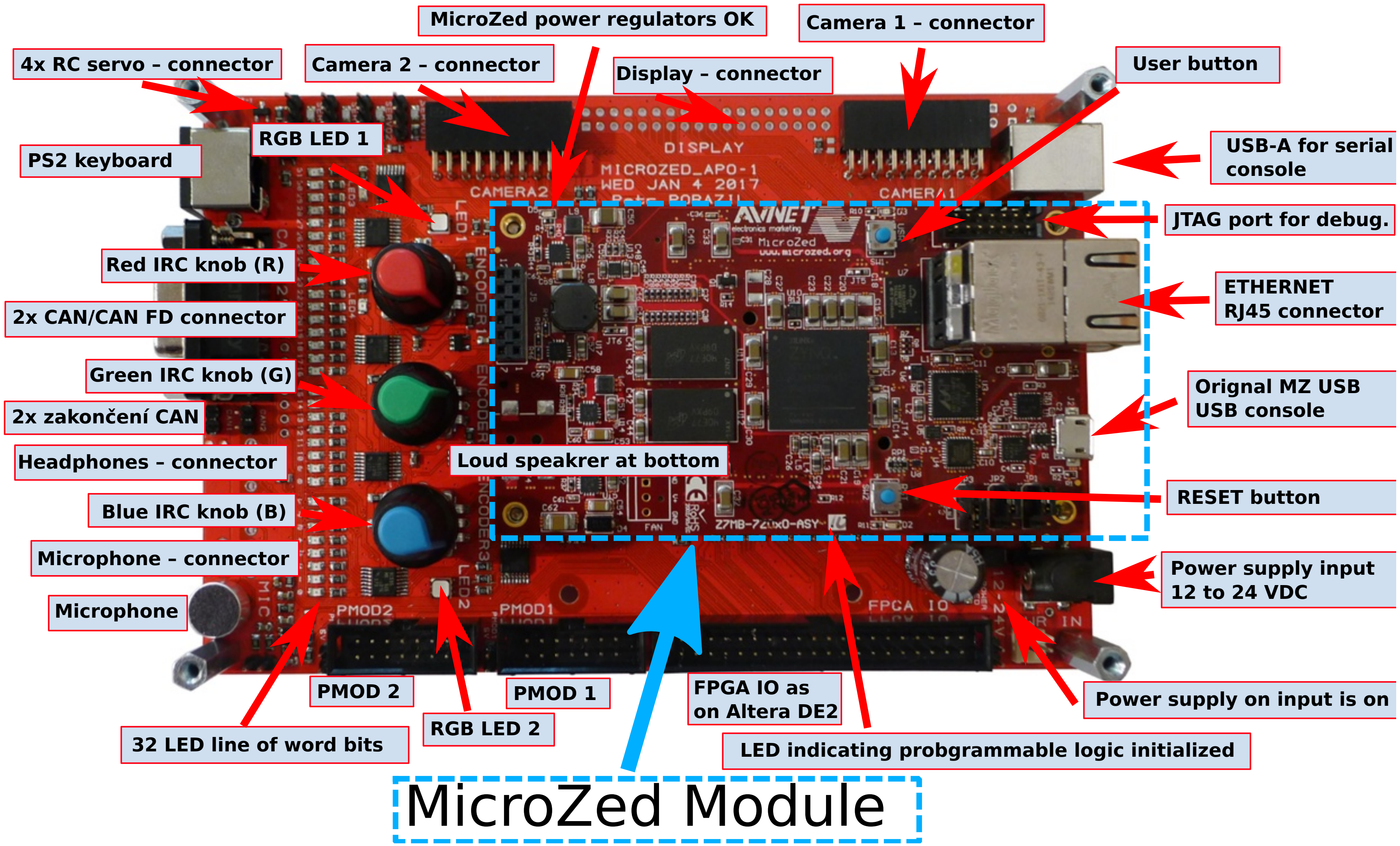


MZ_APO Xilinx Zynq 7000 Based Educational Kits

- MicroZed SBC
- Peripherals for education
- Used in B35APO course – Computer Architectures
- 2x CAN FD transceiver
- Used as main CTU CAN FD development during redesign



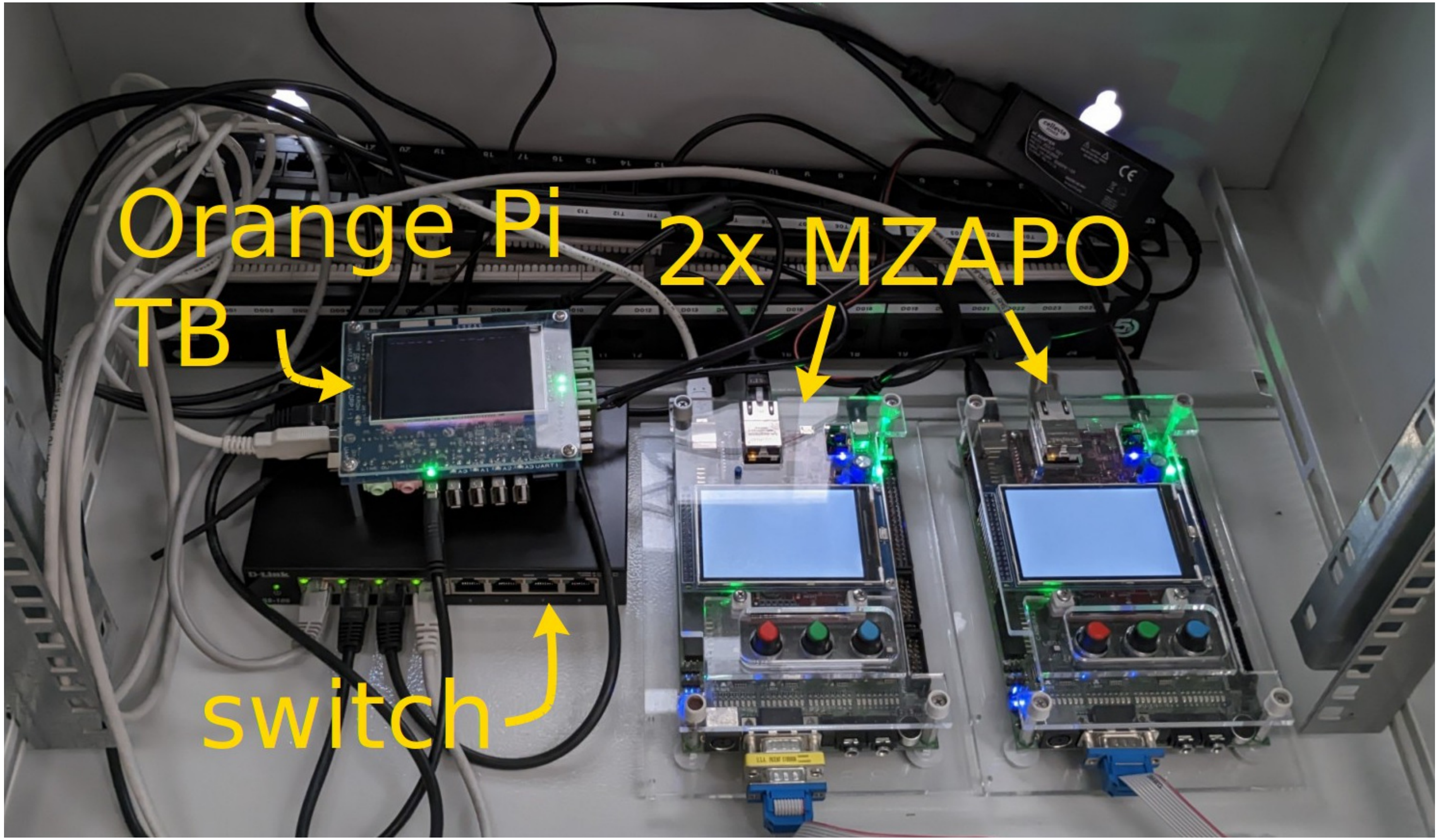
MZ_APO with Port Labels



4. New Automated Testing Setup and Web Site

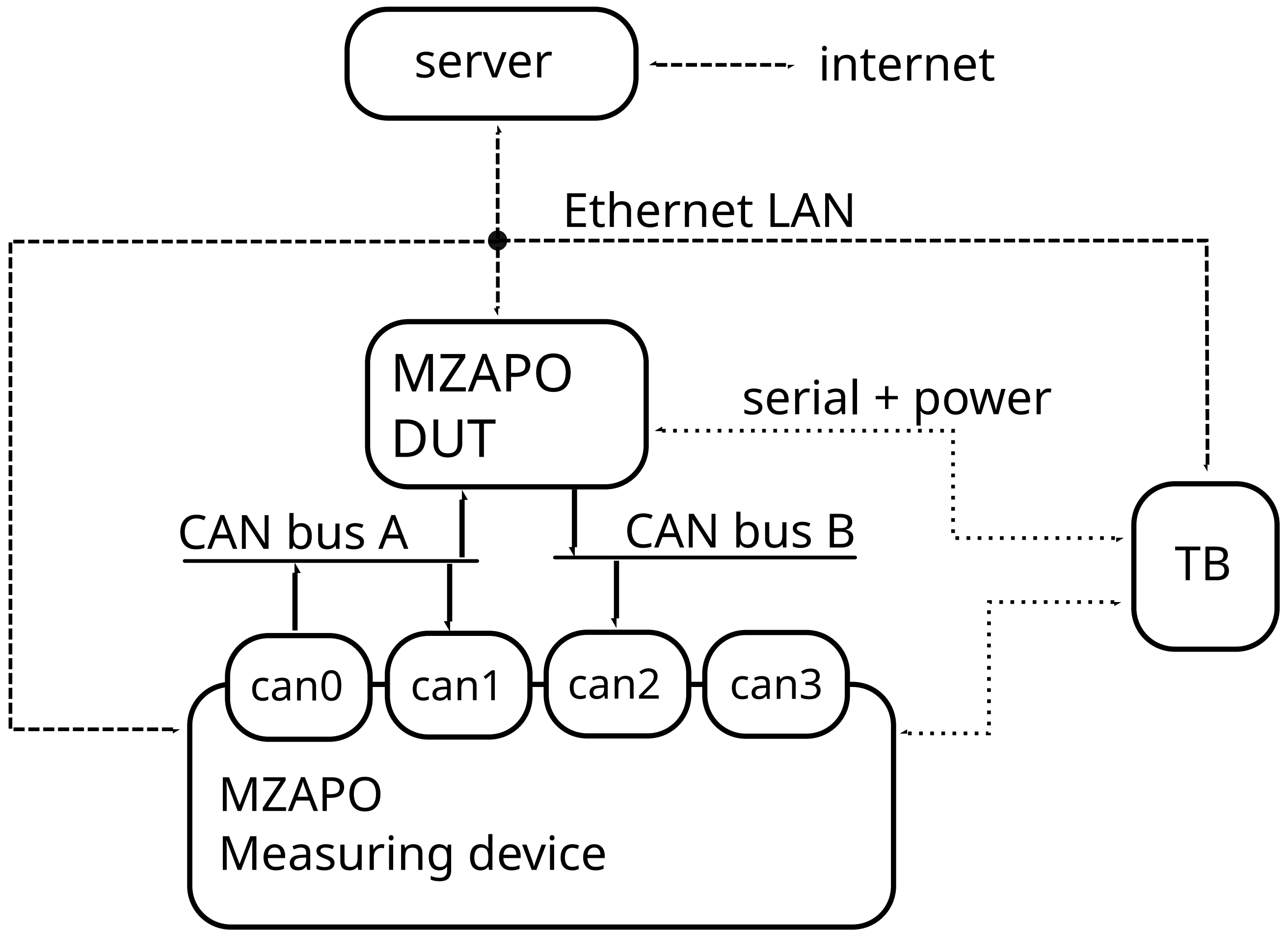
2x MZ_APO, TB Controller and Virtual Server

- Server in faculty virtualization
- Rack with 2x MZ_APO
- Orange Pi based PiKRON test-bed controller
- Monitors serial → USB of MZ_APO
- Reset by break
- Power supply control



MZ_APO Based Latency Tester – Interconnection

- Setup in frame of Pavel Hronek's bachelor thesis
- Runs from April 2023
- Tests daily mainline and PREEMP_RT Linux kernels
- TB, DUD and measuring devices U-boot loads control script and then kernel over TFTP
- Then mounts NFS exported root filesystem

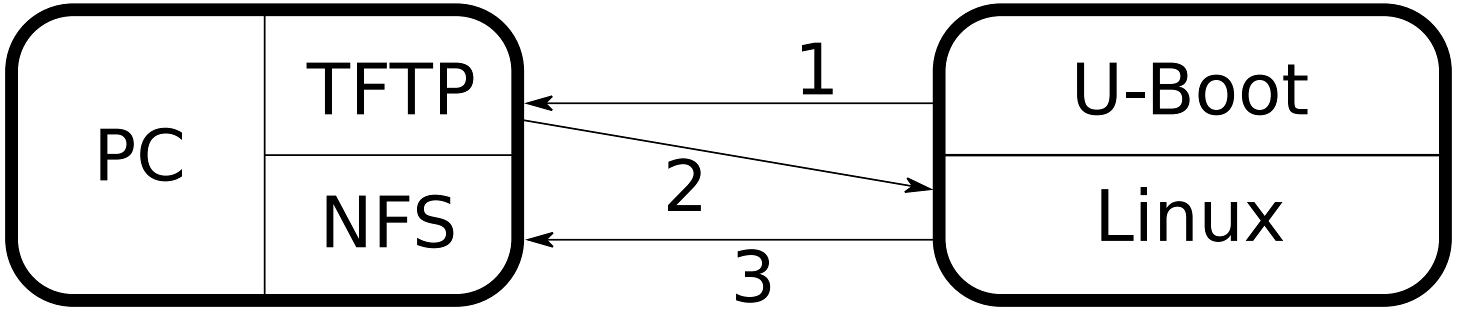


- Export is RO and init-overlay with tmpfs is used. SSH used to command

Test System Automation – Build Kernel and Boot

- Python language
- Top-level run_daily_tests.py from by the cron
- build_linux.py build a Linux kernel for each test run.
- The U-boot ITS for DUT kernel and programmable logic bitstream
- Copied to the TFTP served directory
- U-boot command sequence script prepared and loaded
- DUT and measurement distinguished from the script by Ethernet address
- Measurement device loads a stable kernel with CTU CAN FD timestamping support patch,
- TB resets the boards over serial line connections (break >2s)
- System waits till both systems respond to probe attempt to SSH port

CAN Bench Boot Sequence



- Board reset
  by power cycle
  or break over RS-232
- U-Boot commands
  bootscript_path=/zynq/autoscr.scr
  dhcp && setenv tftpserverip ${serverip} && tftpboot ${bootscript_addr} ${tftpserverip}:${bootscript_path} && source ${bootscript_addr}
- Script located at /srv/tftp/zynq/autoscr.scr on the server, its key parts
  setenv bitstream_load_address 0x04000000; setenv bitstream_load 'fpga loadb 0 ${bitstream_load_address} ${filesize}'
  setenv boot_now 'bootm ${netstart}'
  run image_tftp bitstream_unpack bitstream_load boot_now

Test System Automation – Device Configuration

- Root filesystem is based on the Debian Linux distribution
- Use init=init-overlay, tmpfs layered over base NFS read-only export, multiple devices can boot from the same export and export is protected
- Systemd unit for multi-user.target /etc/systemd/system/init-device.service
- Runs script init-device.sh
- FPGA needs to be programmed with the correct design
- The CAN crossbar switch needs to be configured on each device,
- CAN interfaces are brought up, txqueuelen increased (to prevent ENOBUF)
- The irq threads of the measuring device's receiving CAN interfaces must be given higher priority than the transmitting interface
- For DUT irq priority is controlled by configuration and only for PREEMP_RT

Test System Automation – FPGA Configuration

- Synthesized FPGA design (firmware) activated by device tree overlay file (DTBO)
- The design file is copied under /lib/firmware/ directory.
- DTBO is then loaded using the dtbocfg kernel module (modprobe dtbocfg), and then a directory under Mkdir in /sys/kernel/config/device-tree/overlays/pl
- DTBO file copied .../pl/dtbo
- activated by echo "1" > .../pl/dtbo/status
- Firmware is automatically loaded into the FPGA (DTBO firmware-name = "system.bit.bin").
- When devices boot, the run_test.py script is executed for each selected configuration
- DUT is configured as specified by the configuration

Test System Automation – Run Tests

- Setup kernel CAN GW or UGW on DUT
- Run **latester** on measurement device
- Results fetched by SCP, histogram file is converted to JSON using hist_to_json.py
- Test all specified configurations for different loads, priority, and gateway implementation
- Then process_json_dir.py all JSON files, groups tests by configuration, and merges them into one large file per group.
- Files are then served together with the website.
- Script called build_web.py needs to be executed manually only when some configuration changes.
- All scripts use the conf.json config file, which is by default located at /var/lib/latester/conf.json. Parameters, i.e. , the directories where results are stored, where the website root is located, IP addresses of the devices doing the testing, which commands to run in certain situations, and more.

Test System Automation – Main Configurations

- Options turned on or off
- For master branch kernel:
  - **flood** (send CAN messages as fast as possible)
  - **kern** (kernel gateway is used instead of user mode one)
  - **stress** (stress CPU and memory of the dut system)
  - **fd** (CAN FD messages are used)
- For PREEMP_RT additional **rt** option enables to elevate CAN IRQ thread priorities.

5. Results and Conclusion

CAN Latester Results to Web

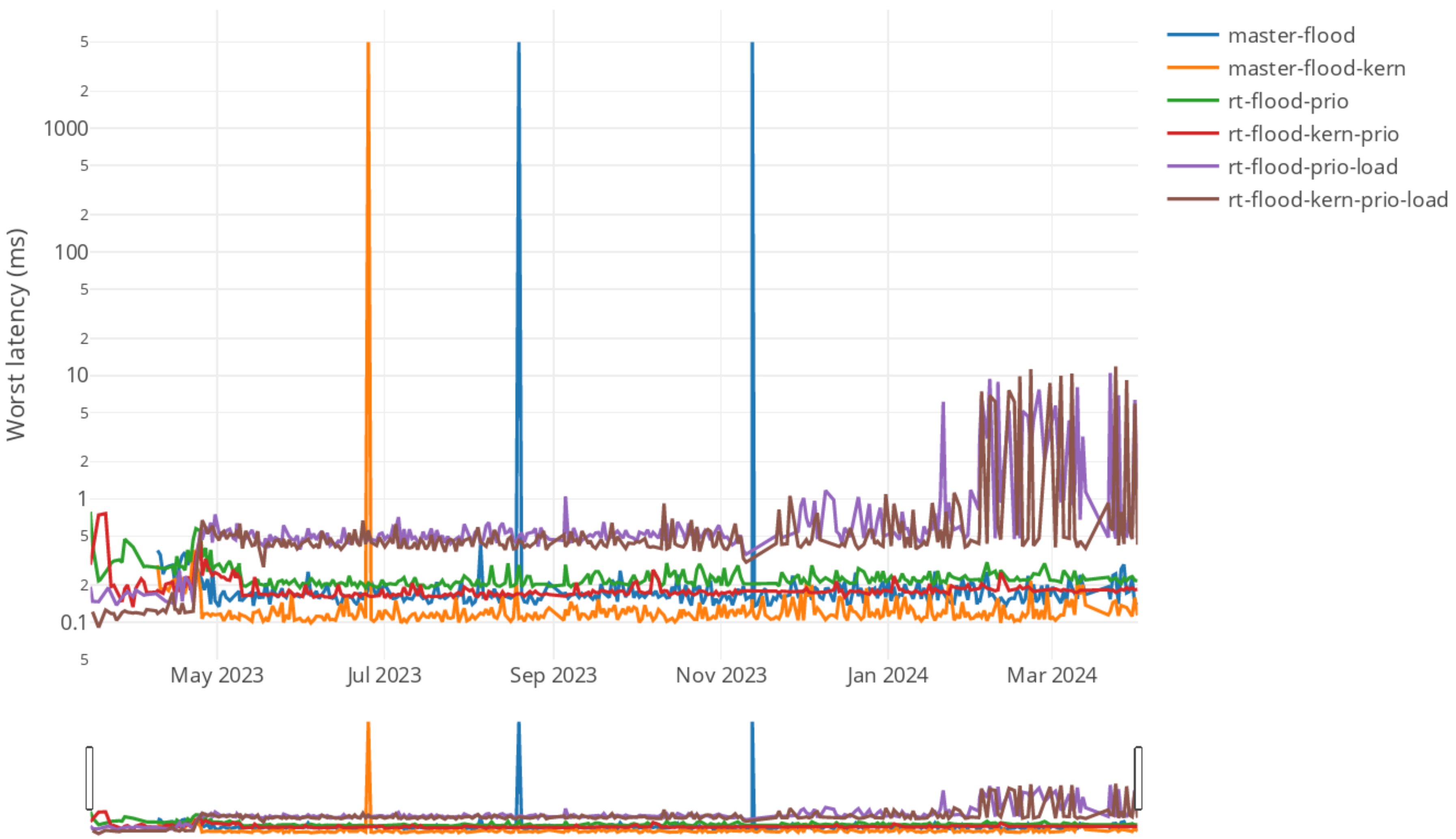
- Static pages (Apache or NGINX) on the server side
- Dynamic elements are implemented on the client side in JavaScript
- Plots are drawn using a JavaScript library called **plotly**
- Continuous operation at the start of April 2023.
- The gateway process's priority and latencies' measurement under the load (stress --cpu 2 --vm 2 and ping flood on the DUT) has been adjusted
- 25 April 2023 daily under the same conditions
- Typical latencies for the unloaded mainline Linux kernel are in a range of 0.1 ms
- The maximal latencies measured on loaded mainline kernel are usually around 1 ms for in kernel CAN gateway, and around 3 ms for UGW (SCHED_RR 80)

PREEMP_RT Kernel Experience Vary for Versions

- Linux kernel on many ARM-based systems, our expectation is that about 0.2 ms (4 kHz with safe margin)
- The Linux kernel networking stack is known to be problematic from a real-time point of view
- PREEMP_RT Linux kernels can keep usually the maximal CAN gateway latency around 0.2 ms. There are some glitches up to 0.45
- Our system catch ARM 32-bit family failures in 6.3 development cycle, quickly resolved by the PREEMP_RT development team (ARM: vfp: Fixes for PREEMP_RT patch by Ard Biesheuvel and Sebastian Andrzej Siewior)
- The 6.8.x series problematic, starting from 6.8.0-rc1-rt1

CAN Latency Tester Overview Page

Time series of maximum latency measured in selected configurations



CAN Latency – Detail for One Test Configuration

Overview Inspect Compare

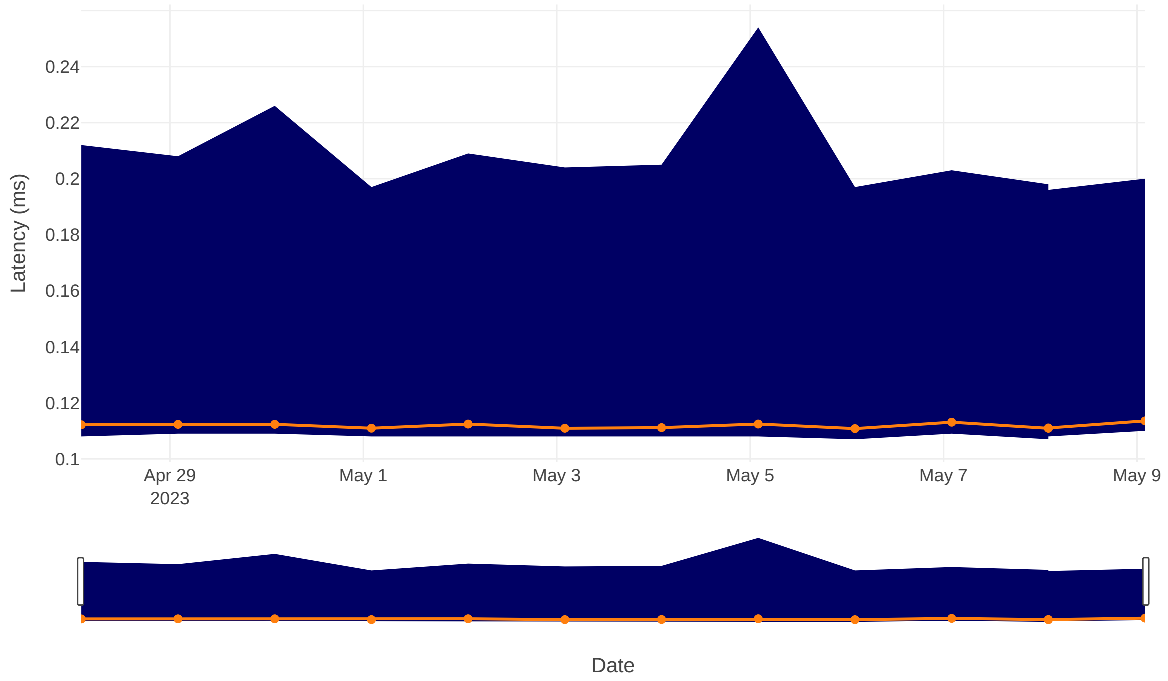
master · RT

Under load · RT priority set · Kernel GW Flood · CAN FD

Gateway latency

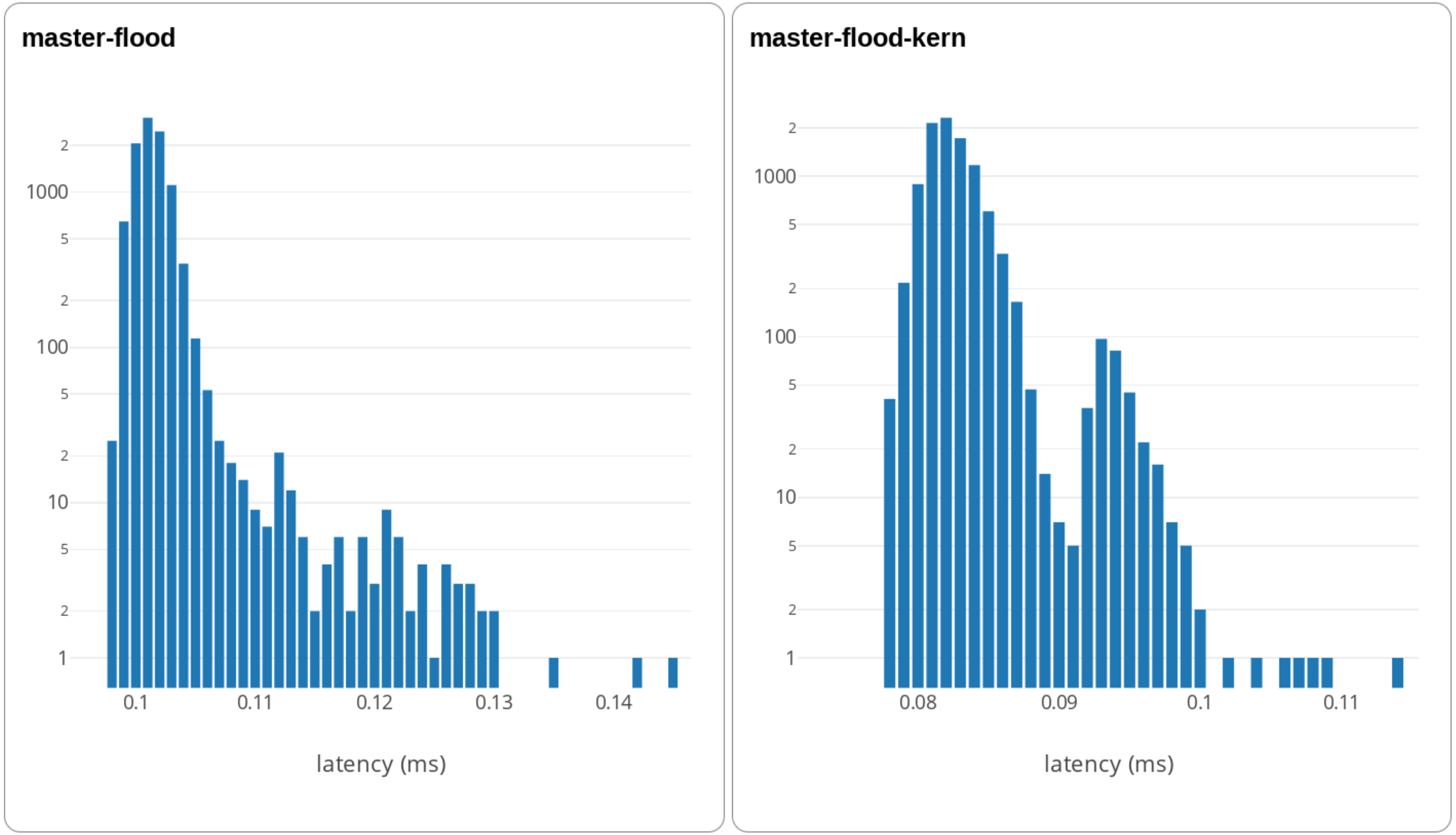
Line plot Heatmap view Surface view

Click into graph to show individual histogram below

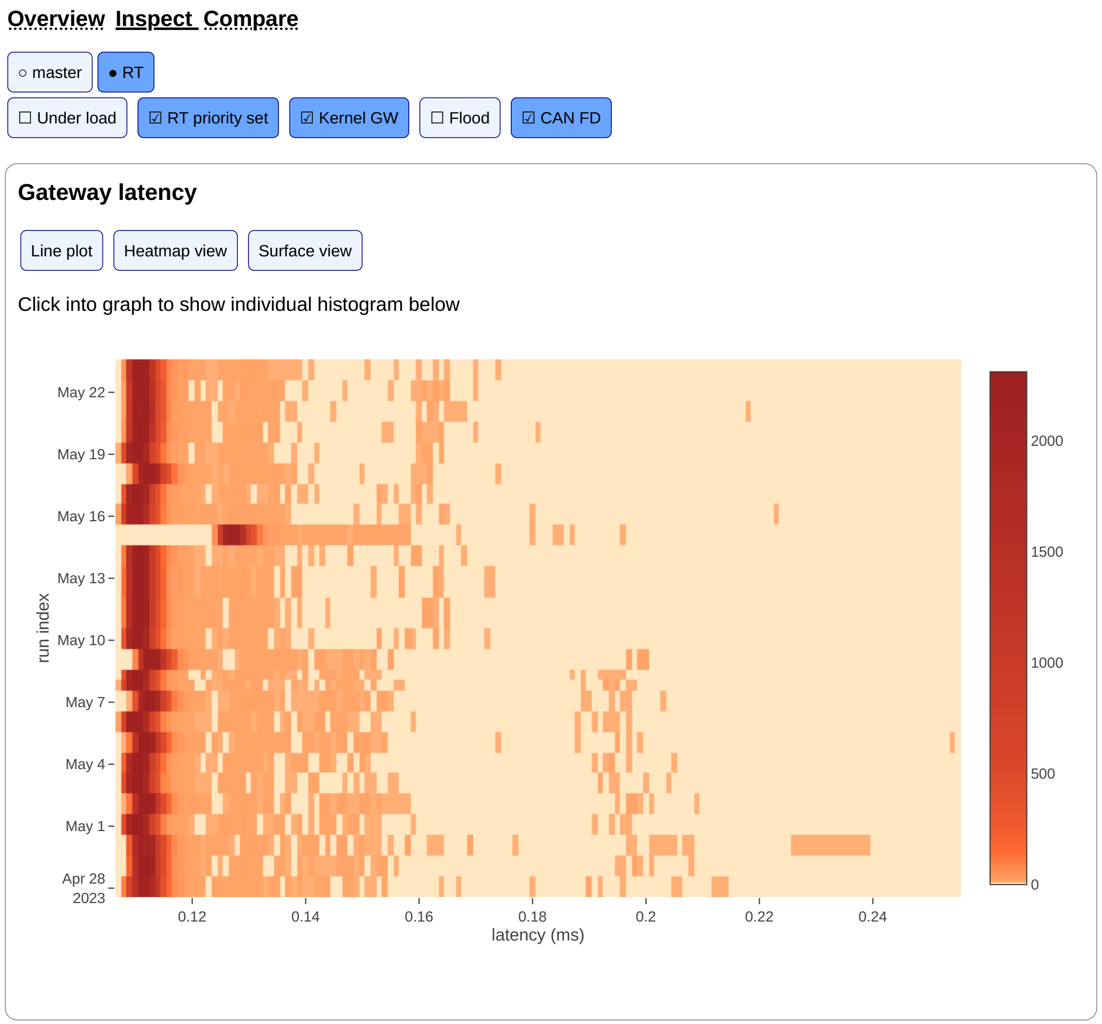


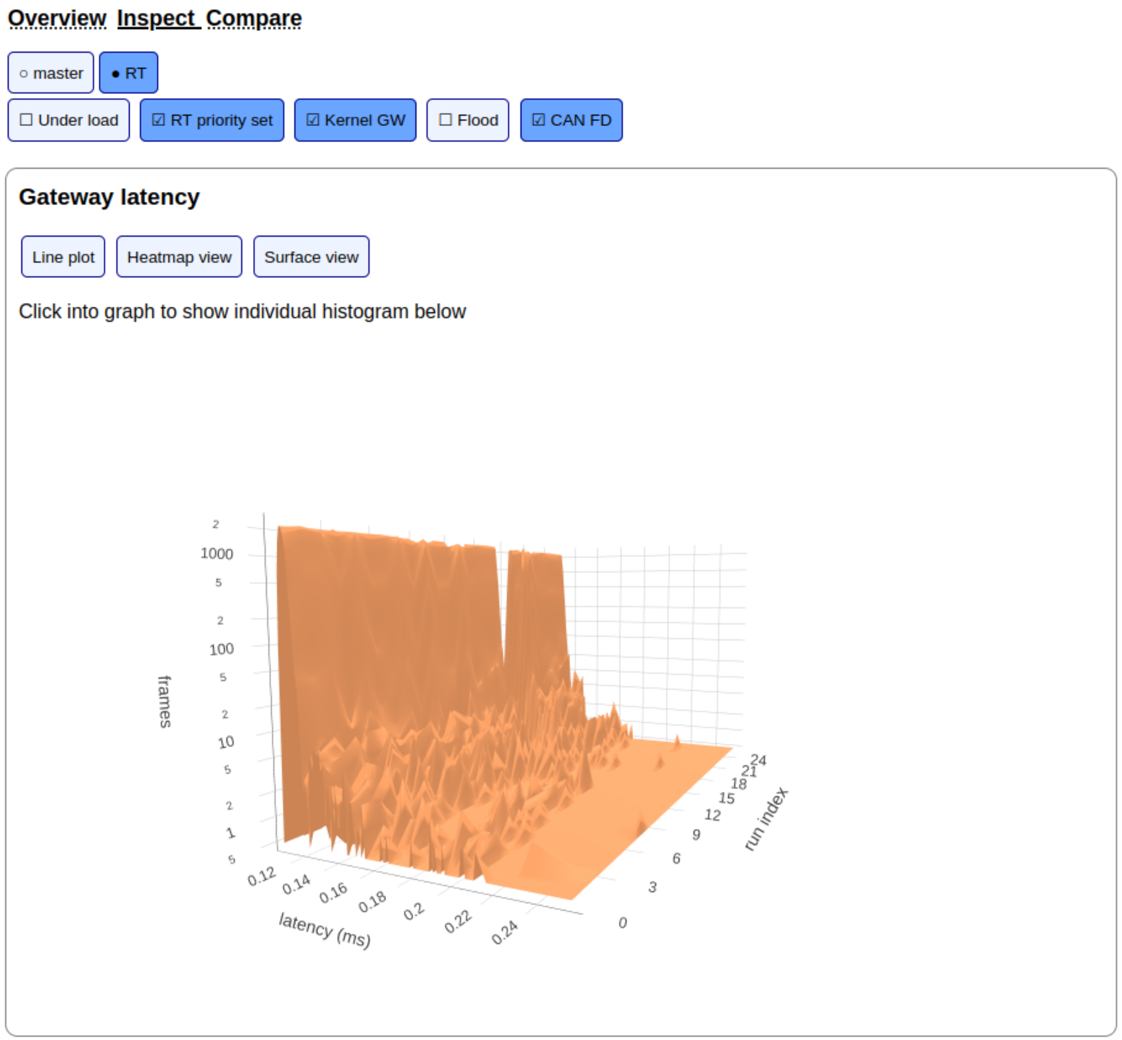
CAN Latency Tester Histograms for Individual Tests

Latest histogram of each series



CAN Latency over Time – Heatmap





Results and Much More CAN Related Projects Online

Visit us at Embedded World 2024
Hall 4, OSADL.org
booth 4-168

https://canbus.pages.fel.cvut.cz/

More CTU CAN/CAN FD Related Projects

- CAN/CAN FD emulation subsystem for QEMU
  - https://www.qemu.org/docs/master/system/devices/can.html
- CAN/CAN FD stack for RTEMS
  - https://gitlab.fel.cvut.cz:otrees/rtems/rtems-canfd
- CTU CAN FD IP Core
  - https://gitlab.fel.cvut.cz/canbus/CAN_FD_IP_Core
- CANopen capable to mimic other devices based on EDS
  - https://ortcan.sourceforge.net/
- CAN drivers for iMXRT and ESP32C3 (ESP32)
  - https://github.com/apache/nuttx