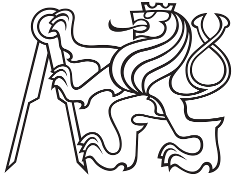
Master’s Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Enhancing Raspberry Pi Target for Simulink to Meet Real-Time Latencies

Bc. Martin Prudek
Study programme: Cybernetics and Robotics.
Specialisation: Systems and Control.

May, 2017
Supervisor: Ing. Pavel Píša, Ph.D.

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Control Engineering

DIPLOMA THESIS ASSIGNMENT

Student: **Prudek Martin**

Study programme: Cybernetics and Robotics
Specialisation: Systems and Control

Title of Diploma Thesis: **Enhancing Raspberry Pi Target for Simulink to Meet Real-Time Latencies**

Guidelines:

1. Familiarize yourself with Matlab/Similink Raspberry Pi target support and with RPP library with CAN block support
2. Benchmark current Mathworks provided Raspberry Pi target
3. Prepare Linux kernel configuration suitable for RT applications and modify Raspberry Pi Simulink template files to generate code more appropriate for RT execution.

Bibliography/Sources:

[1] Jenkin, C.- Sojka, M. : Code generation for automotive rapid prototyping platform using Matlab/Simulink, dokumentace projektu RPP, ČVUT FEL, 2014
[2] Jeřábek, M.: FPGA Based CAN Bus Channels Mutual Latency Tester and Evaluation, diplomová práce, ČVUT FEL, 2016

Diploma Thesis Supervisor: Ing. Pavel Píša, Ph.D.

Valid until the summer semester 2017/2018

L.S.

prof. Ing. Michael Šebek, DrSc. Head of Department		prof. Ing. Pavel Ripka, CSc. Dean

Prague, January 30, 2017

Acknowledgement / Declaration

I would like to express my sincere gratitude to my supervisor Ing. Pavel Píša, PhD., for his continuous support, his patience and his immense knowledge. Without his valuable advice and experience, this work would not have been possible.

I would like to thank my closest family for their support during my studies and special thanks go to Mr. Simon Rhys Jenkins for grammar correction.

I hereby formally declare that I wrote the presented thesis on my own and cited all the used information sources in compliance with the Methodical instructions about the ethical principles for writing academic theses.

In Prague, May 26, 2016.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 26. května 2016

....

Abstrakt / Abstract

Tato práce popisuje analýzu RT vlastností jednodeskového počítače Raspberry Pi v přístupu ke sběrnici CAN. Řídicí aplikace je vytvořena v prostředí založeném na podpoře nástrojů rychlého návrhu Matlab/Simulink.

Jsou prezentovány výsledky testů provedených s různou softwarovou konfigurací - různými jádry systému Linux (RT vs. non-RT), různými ERT soubory (Mathworks vs. ert-linux) a různými metodami časování v nich použitých. Zpoždění generovaných zpráv při testech na neupraveném systému nabíralo zpoždění více než 1 milisekunda na nezatíženém a více než 8 milesekund na zatíženém systému. Zpoždění se výrazně nezlepšila ani po aplikaci dostupných RT vylepšení.

Závěrem nemůžeme doporučit využití Raspberry Pi s PiCAN modulem pro žádné použití v RT prostředí a to ani po aplikaci RT vylepšení. Po aplikaci RT vylepšení je pak možné tuto sestavu provozovat bez další zátěže pro generování zpráv až do frekvence 100Hz a to pouze v non-RT prostředí.

Jako náhradu doporučujeme jiný jednodeskový počítač s GNU / Linux a integrovaným rozhraním CAN. Jako součást této práce byla testována deska SoC Zynq, u níž latence CAN při vysokém zatížení nepřekročily 120 mikrosekund.

Klíčová slova: Raspberry Pi, CAN, Linux, Real Time, Matlab

The thesis provides analysis of real-time properties of Raspberry Pi single board computer with CAN bus in control applications which are designed in a rapid prototyping environment based on Matlab/Simulink target support.

It presents tests that have been performed in various software configurations - various Linux kernels (RT vs non-RT), various ERT files (Mathworks vs. ert-linux) and various timing methods within them. During the tests, it was found that without the use of RT kernel and other patches, CAN latency can often exceed one millisecond. On a high-loaded system, the latencies are easy to move beyond 8 milliseconds. Maximum latencies are not significantly reduced by applying available enhancements.

As a result, we do not recommend the use of Raspberry Pi the PiCAN module with or without RT enhancements in RT applications at all. It can be used without any other load and after application of patches at a frequency of less than 100 Hz in a non-RT applications.

As a replacement, we recommend another single-computer with GNU/Linux and integrated CAN interface. As part of this work, a SoC Zynq board was tested for which the CAN latencies of the high-loaded system did not exceed 120 microseconds.

Keywords: Raspberry Pi, CAN, Linux, Real Time, Matlab

Contents

1 Introduction 1
2 Hardware 3
2.1 Raspberry Pi 3
2.2 CAN Bus 4
2.2.1 CAN Operation 5
2.2.2 Linux CAN Subsystem 8
2.3 CAN Controller Extension Board 8
2.4 CAN Bus Latency Tester 9
2.5 Zynq Board 10
3 Latency Minimization Enhancements 11
3.1 RT Linux 11
3.1.1 Patching the Kernel 11
3.1.2 Building the Kernel 11
3.2 Matlab Patches 12
3.2.1 Source Code Generation 13
3.2.2 Clock Nanosleep Patch Code 14
3.2.3 Memory Lock Patch Code 14
3.2.4 Application of the Patch 15
4 Benchmarks 16
4.1 Sample Time Loop Latencies 16
4.2 CAN Benchmarks 17
5 Tools 19
5.1 Automated Simulations 19
5.2 Datafile Codenames 21
5.3 Data Processing 21
5.3.1 Difference-Based Comparison 22
5.3.2 Linear Regression-Based Comparison 22
5.3.3 Moving Average-Based Comparison 23
5.3.4 Plots and Histograms 23
6 Benchmark Results 25
6.1 Oscilloscope 25
6.2 Sample Time Loop Latencies Measurements 27
6.3 CAN Bus Communication Measurements 36
7 Problems 44
7.1 Compilation Problems 44
7.1.1 Different Step Sizes 44
7.1.2 Modifing Default Make Command 45
7.1.3 Missing Libraries 46
7.2 MDL Format Problems 46
8 Conclusion 48
References 49
A Abbreviations 51

Tables / Figures

Tables

2.1. Table of Raspberry Pi models ... 4
5.1. Datafile codename symbols 1 .. 19
5.2. Datafile codename symbols 2 .. 20
5.3. Datafile codename symbols 3 .. 20
5.4. Datafile codename symbols 4 .. 21
6.1. Overview of Matlab model Sample Time Loop latencies ... 34
6.2. Overview of CAN latencies on system under no load 42
6.3. Overview of CAN latencies on a high-loaded system 42

Figures

2.1. Raspberry Pi v3 model B 3
2.2. CAN network according to 11898-2 5
2.3. CAN signalling according to 11898-2 6
2.4. CAN network according to 11898-3 6
2.5. CAN signalling according to 11898-3 7
2.6. CAN bus frame 7
2.7. PiCAN board 8
2.8. CAN-bench board 9
3.1. Matlab TLC structure 13
4.1. Matlab TLC structure 16
4.2. Matlab model with CAN blocks 17
4.3. Block diagram of CAN benchmark hardware set 17
4.4. Measurement hardware set 18
5.1. Moving average 23
6.1. Simulink model with LED 25
6.2. Oscilloscope benchmark 1 26
6.3. Oscilloscope benchmark 2 26
6.4. Oscilloscope benchmark 3 26
6.5. Oscilloscope benchmark 4 27
6.6. Histogram non-RT unpatched . 28
6.7. Time domain latency plot of loaded non-RT unpatched system 28
6.8. Time domain latencies plot of loaded non-RT unpatched system 29
6.9. Histogram non-RT patched 30
6.10. Time domain latencies plot of loaded non-RT patched 30
6.11. Histogram RT unpatched 31
6.12. Time domain latencies plot of loaded non-RT unpatched system 31
6.13. Histogram RT patched 32
6.14. Histogram RT patched 33
6.15. Time domain latencies plot of loaded RT Raspberry Pi v3 with CTU ert 33

6.16. Matlab model sample time loop cumulative latency histogram . . . 35
6.17. CAN histogram non-RT unpatched . . . 36
6.18. CAN histogram RT patched . . . 37
6.19. CAN histogram RT patched, worker priority boost . . . 38
6.20. CAN histogram RT patched . . . 38
6.21. CAN histogram RT Zynq . . . 39
6.22. Time domain latencies . . . 39
6.23. CAN histogram RT Zynq . . . 40
6.24. Time domain latencies . . . 40
6.25. CAN histogram RT Zynq . . . 41
6.26. CAN histogram RT Zynq . . . 41
6.27. Cumulative latency histogram of CAN messages . . . 43
7.1. Configuring fundamental sample time . . . 45

Chapter 1
Introduction

With the widening spread of robotics, artificial intelligence and automation across industries and everyday life, the need for a distributed control system increases every day. The nodes of such a system require a real-time communication network. Such a network can be implemented, for example, by means of a Controller Area Network communication bus (CAN bus). Application control algorithms are then often implemented in problem-oriented language environments and graphic design systems such as Matlab / Simulink. The thesis deals with the possibility of using a cheap processor system for a higher layer of such control and studies achievable real-time properties of such set-up.

The work aims to find a low-cost solution for running CAN for diagnostic and control applications with low frequency of control loop - not exceeding 100Hz. This allows us to use hardware not initially designed for control applications.

A feasible solution is Raspberry Pi with Matlab Simulink hardware support. One of the main advantages of this platform is enormous global interest and a big community of enthusiasts which attracts the interest even of professional companies seeking business and product propagation. As a result, Raspberry Pi prevails and provides better overall cost and even better reliability than much more expensive hardware with a single vendor and software maintainer.

In this system, quite demanding real-time applications have already been implemented. Among other things, the control of DC motor with IRQ up to 14 kHz [1] and PMS motor control [2]. Version 2 and version 3 models were tested and used for the purposes of this work. More about Raspberry Pi in section 2.1.

The Controller Area Network is a bus standard designed to allow nodes to communicate with each other in applications without single master node. It is a message-based protocol, designed originally for multiplex electrical wiring within cars. Nowadays it is one of the most frequently used protocols in automotive, industry and automation. More about CAN in section 2.2.

CAN hardware differs a lot in robustness, scalability, flexibility and prize. Starting from robust and costly production CAN modules to low-cost single chip expansion boards including well-know hobby boards like BeagleBone or Raspberry Pi the CAN hardware offers a solution to wide scale of problems. *PiCAN2* expansion board [1]) based on Microchip MCP2515 have been chosen for purposes of this work. More on CAN hardware used in this work in section 2.3.

Control application used in this work is based on Matlab / Simulink rapid prototyping platform. Great advantage of Raspberry Pi is the availability of Simulink hardware support [2]) as a freely downloadable add-on. With Simulink hardware support and the Matlab embedded coder, it is possible to generate C-code and run a program directly on Raspberry Pi. The RT enhancements of Matlab C-code generator are one of the

[1]) PiCAN2 `http://skpang.co.uk/catalog/pican2-canbus-board-for-raspberry-pi-2-p-1475.html`
[2]) Simulink HW support `https://www.mathworks.com/hardware-support/raspberry-pi-simulink.html`

main objectives of the thesis, they are described in detail in chapter 3. More about Simulink models tested on Raspberry Pi in chapter 4.

Specification of the task (e.g. low frequency of control loop) allows us to use universal operating system, that is easy to use and provides a possibility of future expansion. A great benefit is the availability of standard remote service and command shell access e.g. SSH or HTTP. Another advantage of a core of a full-featured operating system is readiness for full mutual separation of application address spaces, critical service operations under another user's rights, and planning priorities. GNU / Linux operating system is undoubtedly the best choice.

Mainline Linux kernel can't be used for control applications due to its insufficient support for RT application demands. Its long response time to the external interrupts is the most limiting aspect. The Linux RT capabilities can be improved by application of PREEMP-RT patch. More on Linux and its RT enhancements in section 3.1.

Simulink CAN blocks utilizing Linux kernel SocketCAN API has been[1]) developed at CTU in Prague, Department of Control Engineering. They have been used to implement CAN interoperability within this work.

Benchmarking methods have been developed to verify the effectiveness of the used enhancements. The two main aspects of the system were evaluated - Sample Time Loop Latencies of the Matlab model and latencies of CAN messages sent via Matlab CAN interface. The benchmarks are covered in chapter 4.

Tools for automation of the benchmark making and measurement data acquisition and visualization have been developed. They are described in chapter 5.

Final results and measurements along with data histograms and figures are presented in chapter 6.

[1]) CAN blocks `http://lintarget.sourceforge.net/can_bus/index.html`

Chapter 2
Hardware

This chapter describes hardware used in this work.

At first, Raspberry Pi board is discussed along with its evolution in time and consequences of the initially used obsolete architecture. Next section analyses PiCAN2 board which implements CAN interface for RPi. The last section of the chapter is dedicated to the CAN-bench board used for the destination communication node of the benchmarked CAN communication. All the CAN measurements presented in this thesis were recorded on this machine.

2.1 Raspberry Pi

Raspberry Pi is a popular, low-cost, credit card sized single-board computer, developed since 2006 by British *Raspberry Pi Foundation* [1]). It supports embedded Linux operating systems, such as Raspbian. Raspberry Pi is powered by ARM® Cortex® A processors and provides peripheral connectivity for stereo audio, digital video (1080p), USB and Ethernet. RPi is shipped in multiple models nowadays.

Backbone of the first version is SoC BCM2835 [2]), based on ARM1176JZF-S CPU running on 700 MHz, graphical coprocessor VideoCore IV and 256 MB or 512 MB RAM. None of the versions support straight connection to a HDD via SATA or other on-board MTD. Operating system and program data need to be stored on an SD card, whose slot is available.

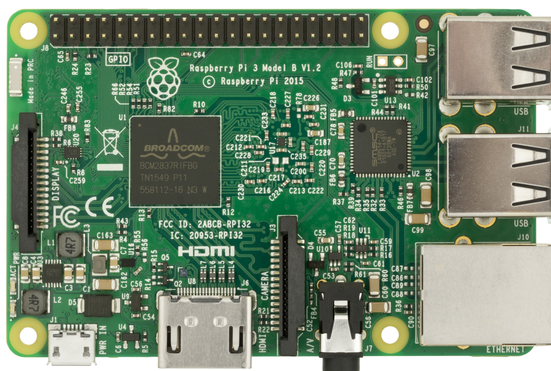


Figure 2.1. Raspberry Pi version 3 model B

ARM11 CPU makes use of obsolete ARMv6 architecture with old vector floating point unit VFPv2 [3] [3]), which is not currently supported by mainline Debian distribution for current generation of ARM-based systems. Official port for the ARM systems - Debian ARMhf enforces architecture ARMv7 at least, with VFPv3-D16 [4]). As a re-

[1]) Raspberry Pi Foundation `https://www.raspberrypi.org/`

[2]) anthill still inside

[3]) ARM11 Online Technical Reference Manual `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0301h/Cegdejjh.html`

[4]) Debian ARMhf port `https://wiki.debian.org/ArmHardFloatPort`

sult, it is necessary to recompile the packages shipped with Debian. Compiled Debian packages are provided within Raspbian distribution for RPi.

The second version of Raspberry Pi comes with BCM2836 SoC. The SoC introduces higher processor frequency (900Mhz) and four cores. Processor architecture evolved to ARMv7-A with ARM Cortex-A7 processor with VFPv4 [4] support. [1]). RAM has been expanded to 1GB.

The latest RPi v3 is equipped with BCM2837 SoC with ARMv8-A architecture. The SoC is based on 64bit ARM Cortex-A53 processor with four cores running on 1.6 GHz frequency. RAM is stick to 1GB size.

The great disadvantage of the SoCs used is the lack of integrated ethernet interface. Problem has been overcome by adding USB-Ethernet converter to the board. This solution imposes high demands on USB subsystem which may result in higher system load.

For the sake of backward compatibility all the programs that run on RPi are restricted to use older VFP instruction set.

Model	A	A+	B	B+	Bv2	Bv3
GPIO pins	26	40	26	40	40	40
RAM [MB]	256	256	256/512	512	1024	1024
USB ports	1	1	2	4	4	4
RJ45	No	No	Yes	Yes	Yes	Yes
Card slot	SDHC	MicroSD	SD	MicroSD	MicroSD	MicroSD
Power [W]	1.5	1.0	3.5	3	4	4
CPU f[MHz]	700	700	700	700	900	1600
CPU cores	1	1	1	1	4	4
CPU arch	ARMv6	ARMv6	ARMv6	ARMv6	ARMv7-A	ARMv8-A

Table 2.1. Table of Raspberry Pi models

2.2 CAN Bus

A Controller Area Network (CAN bus) is initially a vehicle oriented communication bus standard designed to allow microcontrollers and devices to communicate with each other in applications with no need of single master device. It is a message-based protocol, designed originally for multiplex electrical wiring within cars, but it is also used in many other fields of distributed control nowadays.

It's development was launched in 1983 at Robert Bosch GmbH with the first official release in 1986. The first CAN chips were manufactured by Intel and Philips and BMW 8 Series were the first production vehicle with multiplex wiring system based on CAN in 1988. [5] [2])

Several versions of the CAN specification were published during it's development. The latest is CAN 2.0 released in 1991. CAN 2.0 is divided into two parts: part A with 11-bit identifier (standard) and part B with extended 29-bit identifier. Can devices are referred as CAN 2.0A and CAN 2.0B depending on standard they use. The standards are now freely available from Bosch along with it's specifications. [6]

The International Organization for Standardization (ISO) standard for CAN consists of two parts. ISO 11898-1 describes the data link layer, and ISO 11898-2 describes the

[1]) ARM Cortex-A7 Online Technical Reference Manual `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.subset.cortexa.cortexa7/index.html`

[2]) CAN History `https://www.can-cia.org/can-knowledge/can/can-history/`

CAN physical layer. ISO 11898-3 was released later and provides description of CAN physical layer for low-speed, fault-tolerant CAN. Only the first standard is a part of original Bosch CAN 2.0 specification. Minor changes to the standards are made up to nowadays [1])

The main area of CAN application is automotive industry. Many sensors in cars use CAN bus to transmit information about speed, steering angle, air condition, rain sensors etc. These data is used by car subsystems like ABS, airbag controls, parking assist systems etc.

In recent years, the LIN bus standard has been introduced to complement CAN for non-critical subsystems such as air-conditioning and infotainment, where data transmission speed and reliability are less critical.[7]

2.2.1 CAN Operation

The CAN bus - as mentioned before - is available at two physical layers: ISO 11898-2 - "High speed CAN" and ISO 11898-3 - "low speed" or "fault tolerant CAN". Both layers share same the principle of dominant and recessive bits. The difference between them could be observed when two or more CAN devices begin to transmit at once. Each transmission starts with identifier - consisting of 11 or 29 bits - then when a dominant bit and recessive bit are transmitted at the same time only the dominant bit propagates to the bus. The device transmitting the recessive bit recognizes conflict and stops the transmission. The recessive bit equals logical 1 meanwhile the dominant value equals logical 0.

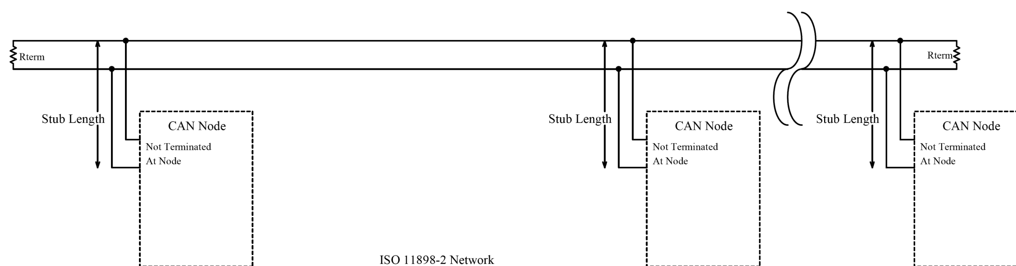


Figure 2.2. CAN network according to 11898-2

Both implementation of the layers use two wires, called CAN high and CAN low, with differential voltage for bus connection. ISO 11898-2 uses a linear bus terminated at each end with 120 Ω resistors called terminators pic 2.2. When a dominant value is signalled the CAN high wire is driven towards 5V and CAN low towards 0V - the dominant differential voltage is a nominal 2V picture 2.3

[1]) ISO 11898 `https://www.iso.org/search/x/query/11898/refine/more:standard`

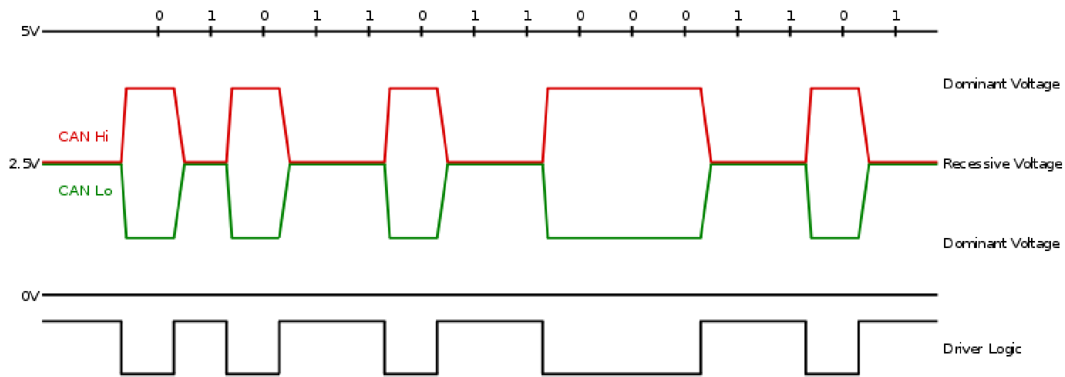


Figure 2.3. CAN signalling according to 11898-2

ISO 11898-3 uses a linear bus, star bus or multiple star buses connected by a linear bus and is terminated at each node by a fraction of the overall termination resistance. The overall termination resistance should be slightly higher or equal to 100 Ω.

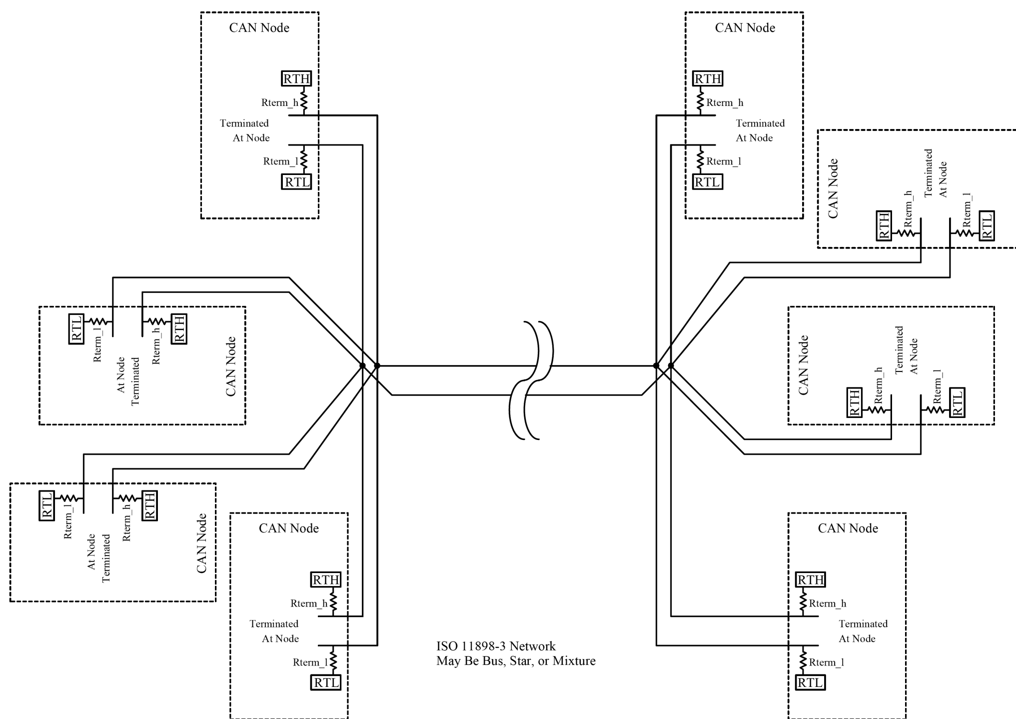


Figure 2.4. CAN network according to 11898-3

When a dominant value is signalled the CAN high wire is driven towards 5V and CAN low towards 0V. The dominant differential voltage must be greater than 2.3V and the recessive differential voltage must be lower than 0.6V The terminators passively return the CAN low wire to RTH where RTH is a minimum of 4.7V and the CAN high wire to RTL where RTL is a maximum of 0.3V. Both wires must be able to handle -27 to 40V without damage.

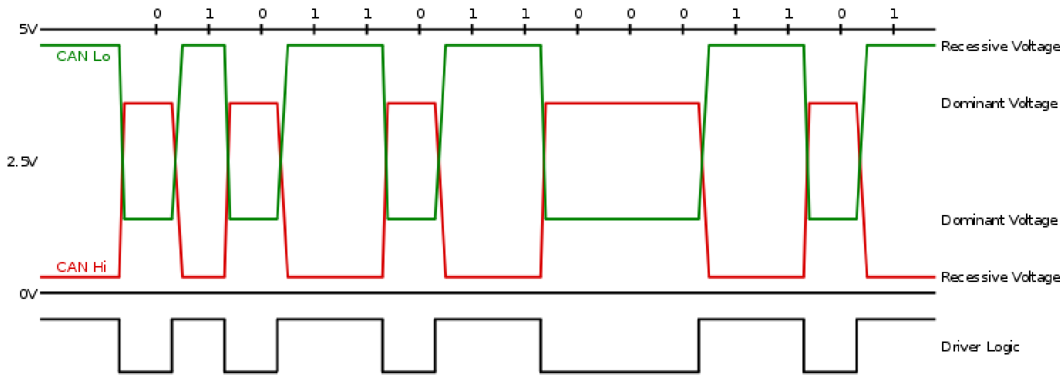


Figure 2.5. CAN signalling according to 11898-3

Terminating Bias Circuit defined in ISO11783 may be used instead of terminating resistor [8]. It provides power and ground in addition to the CAN signaling on a four-wire cable. This provides automatic electrical bias and termination at each end of each bus segment. An ISO11783 network is designed for hot plug-in and removal of bus segments and ECUs. Each node is able to send and receive messages, but not simultaneously.

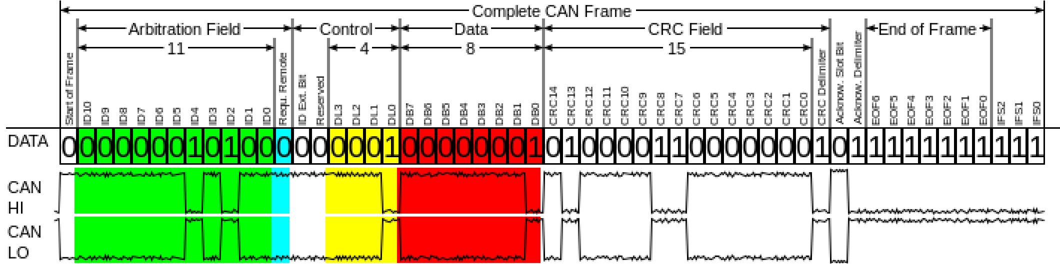


Figure 2.6. CAN bus frame

A message or a frame consists primarily of the ID (identifier of a transferred signal/data), which represents the priority of the message, and up to eight data bytes. A CRC, acknowledge slot [ACK] and other overhead are also part of the message. The example CAN bus bus frame can be seen in picture 2.6

Bit rates up to 1 Mbit/s are possible at network lengths below 40 m. Decreasing the bit rate allows longer network distances (e.g., 500 m at 125 kbit/s).

All the devices connected to a CAN network have to operate at the same bit rate. This rate however might be modified by noise, phase shifts, oscillator tolerance and oscillator drift on single devices. Precise synchronization is necessary during arbitration because the nodes in arbitration have to see both their transmitted data and the other nodes' transmitted data at the same time.

Initial synchronization is done on the first recessive to dominant transition after a period of bus idle - the start bit. Any others recessive to dominant transition during the frame trigger another resynchronization. The CAN controller expects the transition to occur at a multiple of the nominal bit time. To ensure that satisfying number of transitions are done to maintain synchronization, a bit of opposite value is inserted after five consecutive bits of the same value. The inserted bits are removed by the receiving controller.

The main advantage of the CAN bus is its deterministic behaviour. The mentioned IDs (identifiers of a transferred data) represent a priority of each message. Thank to

the arbitration phase described in the above text the messages with the higher priority suppress the low priority messages. This type of collision control is called Carrier-sense multiple access with collision avoidance (CSMA).

2.2.2 Linux CAN Subsystem

On Linux machines, the CAN subsystem, called *SocketCAN*, is a part of the networking subsystem. SocketCAN uses the Berkeley socket API, the Linux network stack and implements the CAN device drivers as network interfaces. The subsystem is divided into protocol layers and brings high level of abstraction. The CAN socket API has been designed as similar as possible to the TCP/IP protocols to allow programmers, familiar with network programming, to easily learn how to use CAN sockets. As a result the same tools are used to control the CAN link layer as to control, for example, the ethernet [9].

The advantage of this approach is that more than one application is allowed to use the CAN interface at the same time. The CAN controllers are usually implemented as as character devices.

As the networking subsystem is optimized for handling big messages to increase throughput, it has rather large overhead. Thus, the performance of handling small messages is poor and the CAN operation sub-optimal [10].

Implementation of higher protocol layers in not part of centralized CAN standard. Layer 3 implementation is done for example by CANopen - Communication protocol for embedded systems. Available drivers are CANpie – Open Source device driver for CAN, can4linux – Open Source Linux device driver for CAN.

A standalone Linux kernel modules that implements a CAN exist. One example is LinCAN module implementing driver capable of working with multiple cards. However, the module development has been stopped [11] [12].

2.3 CAN Controller Extension Board

PiCAN board has been used to provide Raspberry Pi board CAN connectivity.

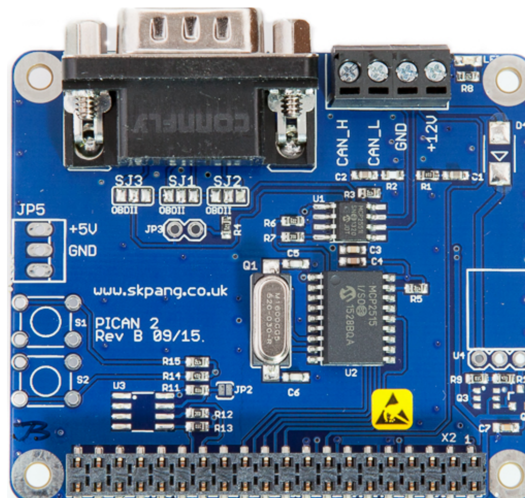


Figure 2.7. PiCAN board

The board (picture 2.7) is based on Microchip MCP2515 CAN controller. MCP2551 transceiver is used to convert controller logic output to the voltage levels defined by physical layer standard. CAN Connections are made available though DB9 or 4 way

screw terminal. This board is also available with a 5v 1A SMPS(power supply) that can power the Pi as well via the screw terminal or DB9 connector. The chip itself is connected to the group of multifunction GPIO pins available on 40 pins header on top of the RPi. The pins belongs to the group which can be configured for SPI function. Actual serial synchronous communication is configured to transfer data at frequency 10Hz.

A kernel configuration file `/boot/config.txt` has to be adjusted to request an appropriate device-tree overlay/extension. The overlay allows to recognize extension board. It is necessary to insert the following lines:

```
dtparam=spi=on
dtoverlay=mcp2515-can0,oscillator=16000000,interrupt=25
dtoverlay=spi-bcm2835-overlay
```

The CAN interface has to be configured for the intended bitrate of the bus communication and then hardware is brought up.

```
ip link set can0 up type can bitrate 500000
cansend can0 7DF#0201050000000000
```

The following command can be used to print statistics about the interface [13]:

```
ip -details -statistics link show can0
```

2.4 CAN Bus Latency Tester

Independent benchmarks of the developed enhancements are one of the major parts of the thesis. When benchmarking CAN communication, it is necessary to use very precise tool that can't be disturbed by any software failures or delays [14]. So called *CAN-bench* board developed at CTU in Prague, FEE, Department of Control Engineering is exactly what we need. This section takes a brief look on the board itself.

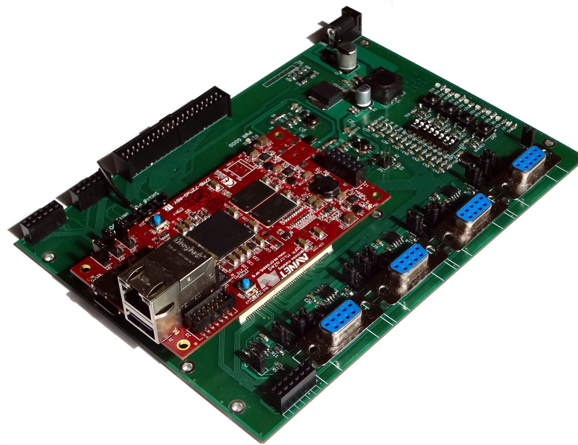


Figure 2.8. CAN-bench board

The *CAN-bench* board (image 2.8) is based on Xilinx Zynq SoC with integrated FPGA. The board runs RT-Linux and is capable of logging CAN messages with a sub-microseconds precise timestamps attached to the messages by a Xilinx CAN controller hardware [15].

2.5 Zynq Board

A part of the CAN measurements described in the following chapters were made on board based on the same SoC as the *CAN-bench*. This board will be referred to as a *Zynq board* in the further text. The board is depicted on ??. All the datafiles with measurements made on the *Zynq board* have *Zynq* in their codename. More on codenames in section 5.2.

Chapter 3
Latency Minimization Enhancements

This work targets on building real time CAN communication between machines running GNU/Linux operating system. The following chapter discuss the need for RT enhancements of various parts of the application and makes a few remarks on each of them.

The OS and the Matlab C source code generator were two major areas of interest.

3.1 RT Linux

GNU/Linux is a multi-user, multi-tasking operating system based on the same named kernel, developed by Linus Torvalds in 1991. With RT patch applied (which secures maximal latencies for responses on external interrupts) it becomes reasonable choice even for some type of control applications.

RT patch modifies the system in many different ways. It makes use of Linux support for SMP and extends the ability of running multiple processes processing system calls in parallel. Serialization of sections of code in which only one process at time may occur is maintained by spin-locks. Other parts of code with enforced mutual exclusion started to use RT-mutexes. Great effort were done in minimizing or removing such part of code which doesn't allow preemption - for example interrupt routines. (Much of their code was moved to worker threads).Implementation of priority inheritance was very important. [16] [17]

Due to the really large code base of the Linux source code it is not possible to analytically compute all the latencies and state the maximal possible latencies. Such an analytical proof would not be feasible because of usage of many diverse cache memories and SMP systems. Nevertheless the satisfying warranty of RT capabilities is provided by long-term benchmarks taken on high-loaded systems for months. The benchmarks and ongoing development of Linux RT patch is focused by OSADL laboratory [1]).

3.1.1 Patching the Kernel

The first step of patching of the Kernel is download of the patches from its website or git repository. With patch unpacked and moved to the Linux kernel source directory it's essential to run:

```
patch -p1 < patchname.patch
```

3.1.2 Building the Kernel

Building the Kernel must be done with cross-compilation tool. The toolchain is specified by path to it's executable by setting environment variable.

```
ARCH            = arm
CROSS_COMPILE   = arm-linux-gnueabihf-
```

[1]) OSADL http://www.osadl.org/

A convenient compilation method is compilation in separate directory from the directory where the kernel sources are prepared. The out-of-the-source compilation is turned on by setting another environment variable when running the *make* command.

```
make O=<path to the build directory>
```

Automation of the mentioned process is possible using *GNUMakefile*. *GNUMakefile* has higher priority then automatically generated build directory *Makefile* when calling the *make* command [1]) Patched kernel (configured to use full preemption) [18] could be built using *make* command afterwards. Proceed with these commands:

```
KERNEL=kernel7
make bcm2709_defconfig
make menuconfig             #enable full preemption model
make zImage modules dtbs -j3
```

The '-jn' switch tells the compiler how many processors we want to use. The recommended value is $n = 1.5$ x total number of processors.

After the build is finished we can move freshly compiled kernel along with modules and device tree to the target. If there is Matlab-prepared SD card inserted in a computer the following commands would do the job:

```
sudo make INSTALL_MOD_PATH=/media/${USER}/rootfs/ modules_install
cp arch/arm/boot/dts/*.dtb /media/${USER}/boot/
cp arch/arm/boot/dts/overlays/*.dtb* /media/${USER}/boot/overlays/
cp arch/arm/boot/dts/overlays/README /media/${USER}/boot/overlays/
cp arch/arm/boot/zImage /media/${USER}/boot/kernel7-4.9-rt.img
sync
```

And finally edit `/boot/config.txt`:

```
kernel=kernel7-4.9-rt.img
```

Alternatively it is possible to copy built kernel along with device tree and kernel modules to a separate folder for further use. Proceed with these commands:

```
INSTALL_PATH="</folder/with/kernel>"
mkdir -p "${INSTALL_PATH}/boot/overlays/"
sudo make INSTALL_MOD_PATH="${INSTALL_PATH}" modules_install
cp arch/arm/boot/dts/*.dtb "${INSTALL_PATH}/boot/"
cp arch/arm/boot/dts/overlays/*.dtb* "${INSTALL_PATH}/boot/overlays/"
cp arch/arm/boot/dts/overlays/README "${INSTALL_PATH}/boot/overlays/"
cp arch/arm/boot/zImage "${INSTALL_PATH}/boot/kernel7-4.9-rt.img"
tar -czvf 4.9.20-rt16-v7+.tar.gz "${INSTALL_PATH}"
```

3.2 Matlab Patches

The Target Language Compiler (TLC) is a tool originally developed for the Matlab Real-Time Workshop. It became the integral part Matlab Embedded Coder later. It enables C-code generation directly from any Simulink model.

Through customization, it can produce platform-specific code that could be run even on different operating systems [19]. Incorporation of own algorithmic changes is possible for better performance, code size, or compatibility with existing methods [20]. The main

[1]) What Name to Give Your Makefile `https://www.gnu.org/software/make/manual/html_node/Makefile-Names.html`

files of TLC, referred as *ert* or *ERT*, present the entry point of the code compilation and define all the compilation information along with the target platform specifications.

This section describes the performance analysis and enhancements of the TLC, hereinafter referred to as *Matlab patches*. Main goal is to modify (if needed) the main Matlab *ert* file (MTW *ert*) structure to meet latency requirements.

The first step in the analysis was the examination of C-code generated by the MTW ERT. Parts of code not suitable or disturbing precise timing of sampling intervals has been identified. These parts have been modified to use system-calls and mechanism known to support low latencies and correct scheduling prioritization. The modifications has been saved to two patch files. The patches replaces sections of code in C and TLC files inside Matlab home directory to reach mentioned behaviour.

3.2.1 Source Code Generation

The first step required to audit Simulink provided code has been to study mechanisms which control how C code sources a are generated. Debug comments were added to multiple C and TLC files in MATLAB home directory. Then the model was regenerated and changes made to the source were observed. Based on this observations files connections were tracked down and I was possible to edit the files actually responsible for source generation. The basic TLC file structure follows:

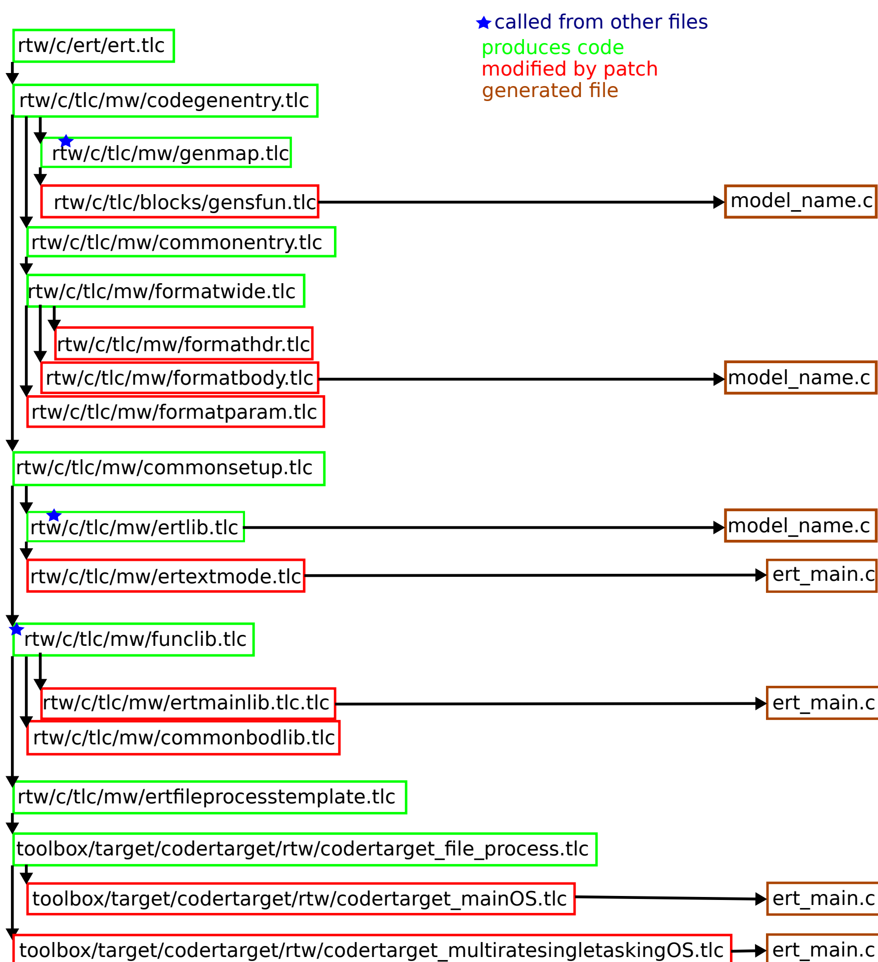


Figure 3.1. Matlab TLC structure

3.2.2 Clock Nanosleep Patch Code

The first main problem was identified in timing of model sampling loop. Snippet of the original code follows:

```
while(1) {
    waitForTimerEvent(fd);
    sem_post(&baserateTaskSem);
}
```

While *waitForTimesEvent*() function tries to read from Linux timer referenced by a file descriptor. This call could potentially increase loop latency in a non-deterministic way. Replacement code follows:

```
while(1){
    /* wait until next shot */
    clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &t, NULL);

    /* do the stuff - post the semaphore */
    sem_post(&baserateTaskSem);

    /* calculate next shot */
    t.tv_nsec += interval.tv_nsec;
    t.tv_sec += interval.tv_sec;
    if (t.tv_nsec >= NSEC_PER_SEC) {
        t.tv_nsec -= NSEC_PER_SEC;
        t.tv_sec++;
    }
}
```

`read()` call was replaced by `clock_nanosleep()`. Both version were tested and measured - the results could be found latter in this document.

3.2.3 Memory Lock Patch Code

The second main problem was identified in process memory management.

The model originally didn't force immediate load and lock of the code to main system memory (RAM). This situation could cause that any page of the program code can be swapped out to a secondary memory. Such an action would result in latency up to few milliseconds.

mlockall system call was added to the Linux initialization file to fix the situation. The *mlockall* system call ensures that no memory page is swapped out of the primary memory.

The following section of code demonstrates which part of the source have been modified. The first portion of the code has been added to the top of the file.

```
+#if defined(_POSIX_MEMLOCK)
+       #include <sys/mman.h>
+#else
+       #warning mlockall is not available (!_POSIX_MEMLOCK)
+#endif
```

The second portion of modification has been added to the `myRTOSInit` function:

```
void myRTOSInit(double baseRatePeriod, int numSubrates){
    ...
    unsigned long cpuMask = 0x1;
    unsigned int len = sizeof(cpuMask);
```

```
+   #if defined(_POSIX_MEMLOCK)
+       mlockall(MCL_CURRENT | MCL_FUTURE);
+   #endif

    UNUSED(baseRatePeriod);
    UNUSED(numSubrates);
    ...
}
```

3.2.4 Application of the Patch

This section describes the procedure of application of the patch.

`tracked_files` text-file contains list of all files modified by the patches in the 'MATLAB/R2016b/' directory. It's good idea to init a git repository in the 'MATLAB/R2016b/' before all. Then gitignore everything. Be sure to track all the files, which will be modified, before an application of each patch:

```
git add -f `cat tracked_files`
```

After application of the patch run:

```
git commit -a
```

This procedure helps to keep track of what's beeing modified and offers the ability to revert all the changes.

The patch is intended to be run from 'MATLAB/R2016b/' directory

```
patch -p1 < x-patch.patch #where 'x' stands for patch number
```

All the patch files must be run one-by-one starting with the one with the lowest number.

Chapter 4
Benchmarks

The previous chapter described modifications and patches applied to various parts of the system to improve its Real-Time qualities. This chapter introduces set-up and methods used for evaluation of achieved results. Two main aspects of the system are going to be evaluated - latencies of Matlab model sample loop and latencies on CAN messages sent via Matlab CAN interface.

4.1 Sample Time Loop Latencies

This section describes the method used to benchmark latencies of Matlab model sample loop.

New Simulink block with level 2 S-function was developed accomplish this task. It's only functionality is to precisely log exact time when sample loop activities are executed by a code generated from the Simulink model. This checks latencies caused by operating system, generated code, established set-up and system provided mechanisms.

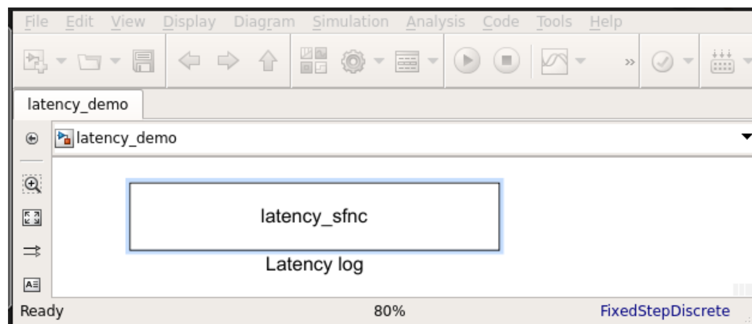


Figure 4.1. Matlab model with S-function block

The S function routine starts a new thread with lower priority. The new thread opens a file and periodically logs time stamps measured in the main routine. The main routine creates new time-stamp at the start of each cycle. Snippet of code which is executed at the beginning of each cycle follows:

```
clock_gettime(CLOCK_MONOTONIC, &now);
timespec_subtract(&times[get_time_index++],&now,&prev);
get_time_index%=TIMES_COUNT;
prev=now;
sem_post(&sem);
```

A part of lower-priority thread, which logs the data to a file, looks like this:

```
while(1){
  sem_wait(&sem);
  fprintf(f, "%u %ld\n",  (unsigned)times[saved_time_index].tv_sec,
    (long)(times[saved_time_index].tv_nsec));
  saved_time_index++;
  saved_time_index%=TIMES_COUNT;
}
```

4.2 CAN Benchmarks

Matlab model similar to the one used in the previous section was developed to benchmark CAN communication. The model makes use of CAN toolbox developed at CTU in Prague, FEE, Department of Control Engineering [1]). The model diagram is on picture 4.2.

The toolbox provides basic blocks for sending and receiving CAN messages. It was originally designed only for ERT Linux Simulink target, nevertheless it's functionality have been tested with MathWorks ERT within in the Honeywell company project and during this thesis tasks. The toolbox uses mainlined socket based CAN bus Linux kernel drivers support - SocketCAN described in section 2.2.2.

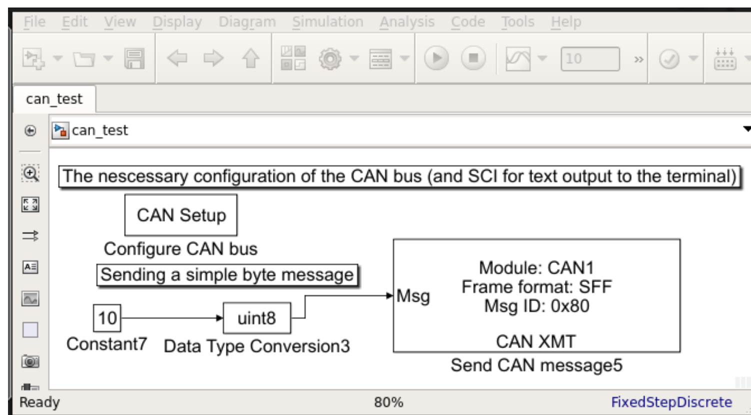


Figure 4.2. Matlab model with CAN blocks

The CAN blocks support direct encapsulation and conversion to the Simulink signals for basic data types whereas the standard Simulink defined CAN MESSAGE TYPE can be used when more complex structure of CAN message processing is needed. CAN Setup block needs to be present in a model so that CAN channel and bauderate could be set properly in embedded systems. On Linux however baudrate must be specified using standard iproute2 userspace tools. The connection between components is visualised on diagram 4.3.

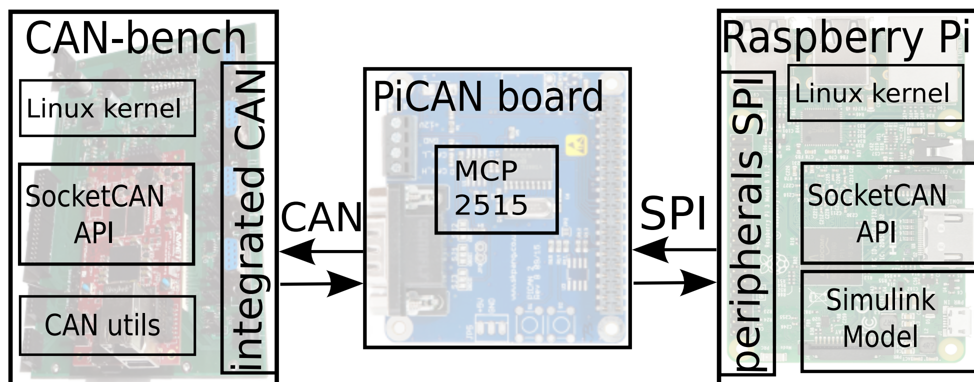


Figure 4.3. Block diagram of CAN benchmark hardware set

[1]) Simulink Blocks for CAN bus Support under Linux `http://lintarget.sourceforge.net/can_bus/index.html`

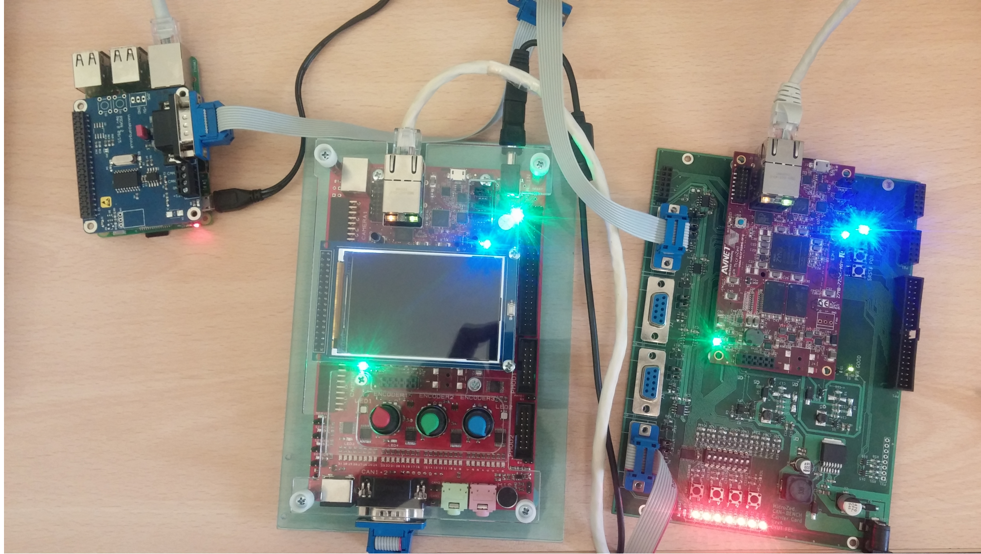


Figure 4.4. Measurement hardware set

The benchmarking is based on measuring reception times of CAN messages generated by Raspberry Pi and CAN hardware covered in sections 2.3 and 2.4. The measurements were taken on CAN-bench board developed at CTU in Prague, FEE, department of Control Engineering. The board runs RT-linux and is capable of logging CAN messages with precise timestamps attached to the messages by a Xilinx CAN controller harware. The whole hardware set used for benchmarking could be seen on picture 4.4. The results of the benchmarks are discussed in chapter 6.

Chapter 5
Tools

One of the most important problems to solve during the work was the automation of benchmark making and the processing of gathered data. As there are many parameters to be entered before any measurement begins - such as usage of Matlab patches, load, ert file, sample time or total time of Matlab simulation - it was necessary to build a tool that manages it systematically. This chapter provides an overview of such a tool along with user instructions.

5.1 Automated Simulations

This section describes a tool for the automated running of benchmarks. The main part of the tool is written in Matlab and consists of `do_simulation.m` function saved to a same-named file and a few scripts coded in Bash. `do_simulation` function is called with the parameters listed in table 5.1.

Code	Example value	Description
t	lat / can	Benchmark type
lo	No / 8W1F	Load to be run on target
p	40 / 98	Priority of the main thread
ert	MTW / CTU	Choice of ert file
si	1000	Simulation time
sa	0.01	Sample time - can message period
n	Note	Optional note
target	root(at)ip.cz	Target machine login

Table 5.1. Datafile codename symbols

The `t` argument defines the type of a benchmark - it could be either `lat` for benchmarking Sample loop latencies described in section 4.1 or `can` for benchmarking CAN communication - discussed in section 4.2. The `lo` argument starts an additional load. If it is set to value `8W1F`, then it starts eight *do-while* loops and one *find* loop that uses RAM and loads CPU cores of the board. The invocation code of them follows:

```
while true; do true; done &
while true; do find /usr -type f -exec cat '{}' ';' > /dev/null; done &
```

The `p` argument sets main thread priority and `si` and `sa` set simulation and sample times.

User public key must be added to the machine before any measurement. It serves for *ssh* and *scp* connection to the board.

A few more options must be specified when using MathWorks ert (MTW). It is possible to use Matlab patches - both for memory locking and nanosleep. Unfortunately

Code	Example value	Description
m	No / CF	Usage of Matlab patch - memory lock
cn	No / Yes	Usage of Matlab patch - clock nanosleep
target_pass	password	Password for entered login

Table 5.2. Datafile codename symbols

public key infrastructure is not sufficient at this point and user password must be specified. These options are shown in table 5.2.

Furthermore when using CAN benchmark a machine which would receive and save CAN messages must be specified. For debug purposes it is also possible to specify secondary measurement machine - this machine would generate the same output file as the primary one - with *msec* in its codename note. More on datafile codenames in the next section 5.2. Options listed in table 5.3.

Code	Example value	Description
measure	root(at)¡primary.board.cz¿	Primary measurement machine login
measure_sec	root(at)¡secondary.board.cz¿	Secondary measurement machine login

Table 5.3. Datafile codename symbols

The `do_simulation` function checks internet connectivity to the target and measurement machines at first, then gathers information about the hardware and kernel. After that, CAN interface is checked and configured. Appropriate models are loaded and their parameters set in the next step. Some of the parameters could be set via Matlab commands whereas some of them must be altered directly in model XML files. Model SLX files are unpacked into XML files in the first stage of model configuration, then the XML files are properly modified and packed again.

In the next step, bash scripts that start additional load on the target machine are moved to their destination and started. Scripts for logging are transferred to the measurement machine(s) and the logging starts. If the user wants to use any Matlab patch its code is applied in this step.

In the second stage of model configuration the Matlab function `set_param` is used and depending on the used ert `set_param` or `slbuild` is used for building the model. The built model is either moved and started automatically after the build (MathWorks ert) or moved via SCP and started using `schedtool` (CTU ert).

After the simulation is finished the load is terminated and log datafiles are transfered back to the repository folder. The Matlab patches are reverted.

The benchmarks could be started from the command line without starting Matlab IDE or Simulink explicitly. The easiest way is to run `run_benchmark.sh` from `/tools/` directory with one of the Matlab scripts prepared in `/tools/benchmark_sets/` as its argument. The script tries to start Matlab without any graphical output from command line. The `--nodisplay` mode is not available when using MTW ert - the user is asked to run Matlab in normal mode and the process continues by calling the Matlab `do_simulation` function.

5.2 Datafile Codenames

As the number of all measurements and thus all the datafiles exceeds 150 with more than 22 hours of total testing time naming is critical for data organization and establishment of a strict experiment naming convention has been necessary.

Each datafile name e.g. *codename* consists of sequences describing the benchmark simulation time, hardware of the target machine and so on. The part of the codename that could be determined before the start of a benchmark is described below in section 5.1 and consists of codes listed in tables 5.1 and 5.2 excluding the last ones *target_pass* and *target* that are not present in the name.

The rest of the codename is generated automatically depending on the settings. The sequences that are generated automatically are based on information gathered about hardware, operating system version and operating system RT modifications. The benchmark number is added as a symbol that uniquely distinguishes the measurement between each other. The automatically generated parts are shown in table 5.4.

Code	Example value	Description
hw	Zynq / RPi-v3	Target machine hardware
k	non-RT / RT	Kernel RT modification
vk	4.4 / 4.9	Kernel release version
bn	15 / 157	Benchmark number

Table 5.4. Datafile codename symbols.

Codenames come with strict format

```
t<>_hw<>_lo<>_k<>_vk<>_p<>_ert<>_m<>_cn<>_bn<>_si<>_sa<>_.rdat
```

Example codename could be

```
tlat_hwRPi-v3_lo8W1F_kRT_vk4.9_p98_ertMTW_mCF_cnNo_bn5_si10_sa0.01_.rdat
```

This codename presents a datafile of sample loop latency measurement measured on Raspberry Pi v3 loaded with 8 *do-while* loops and one *find* loop. The system was identified as Linux v4.9 with PREEMP-RT patch applied.The main model thread was started with priority 98 and MathWorks ert was used in the build process. The memory lock patch was applied meanwhile clock nanosleep patch was not. It was the 5th benchmark and lasted for 10 seconds with a sample time of 10 milliseconds.

5.3 Data Processing

With such a big acquired data set, it is hard to present the results correctly as they may vary from measurement to measurement. It is important to emphasize the main aspects of each set of measurements to make the situation clear. Various ways of data visualization and statistical processing were used to accomplish this. This section provides a guide through them.

The first stage of data processing is the preparation of absolute sample times. Both the sample loop latencies and CAN latencies datafiles are taken in count and `__abstimes` directory is filled with files containing absolute times. Absolute timing means that the sample times are relative to a very first sample and not to the sample preceding. The computation is started by calling the script `prepare_abstimes.sh` in `/tools/` directory with the directory with the datafiles

as its only argument. Running without an argument is the same as running `./prepare_abstimes.sh ../data/base_set`.

In the second stage of data processing, comparison of offsets between estimated and actual sample times are computed. The comparison is done in three different ways.

5.3.1 Difference-Based Comparison

The first comparison is based on the estimation of time of sample by addition of sampling period to time of of the previous sample. The great advantage of this approach is its simplicity - there is a low probability of making any computation mistake. Another benefit is that any latency peak is clearly visible and can't be misinterpreted. The disadvantage of this approach is that all the peaks are mirrored - if only one of the samples goes out of order then the next sample visually has very similar latency with opposite direction. The computation of such offsets follows:

$$o[i] = ts[i+1] - ts[i] - sa$$

Where $o[i]$ stands for i-th 'offset', $ts[i]$ stands for absolute time of i-th sample and sa for sample time. This part of data processing is invoked by calling `./prepare_doffsets.sh` from `/tools/` directory. This script creates `__offsets_diff` directory and fills it with files with computed offsets. The computation itself is done by `comp_doffsets` tool written in C.

5.3.2 Linear Regression-Based Comparison

This comparison is especially useful when evaluating CAN communication. There is no secondary channel between the target and the measurement machine so it is not possible to determine a precise time when any CAN message should arrive. As a result linear regression of the measured sample times is computed and all the offsets are computed against the resulting linear curve. We state the situation as follows:

$$ts[i] = k * pst[i] + q$$

Where $ts[i]$ stands for absolute time of i-th sample, $pts[i]$ for it's ideal predicted value $pts[i] = i * sa$ and k and q unknown constants. Let's use neat matrix notation:

$$\begin{bmatrix} ts[1] \\ ts[2] \\ . \\ . \\ . \\ ts[n] \end{bmatrix} = k \begin{bmatrix} pts[1] \\ pts[2] \\ . \\ . \\ . \\ pts[n] \end{bmatrix} + q$$

That could be rewritten into:

$$\vec{ts} = k\vec{pts} + q$$

And finally:

$$\vec{ts} = [\,\vec{pts} \quad \vec{1}\,] \begin{bmatrix} k \\ q \end{bmatrix}$$

$$\begin{bmatrix} k \\ q \end{bmatrix} = [\,\vec{pts} \quad \vec{1}\,]^{-1} \vec{ts} = A^{-1}\vec{ts}$$

The linear regression computation is based on the least-square method and carried out by `mldivide` operator in `prepare_offsets` Matlab script. The computation follows:

```
res=A\ts;
k=res(1);
q=res(2)'
```

The script could be run without starting Matlab IDE by calling

```
matlab -nodisplay -r "prepare_offsets('$1'); exit;";
```

5.3.3 Moving Average-Based Comparison

Measurements that showed strange behaviour were taken during benchmarking the Zynq based board. The strange behaviour was observed only on *Linear Regression-Based Comparison* plots. It was discovered that system clock drifted and degraded all the measurements taken with active NTP client. The client was off in all subsequent measurements.

Moving average-based comparison has been adopted to get real latencies cleaned from the drift. *Linear Regression-Based Comparison* is computed at first. Then, offsets are computed against its moving average.

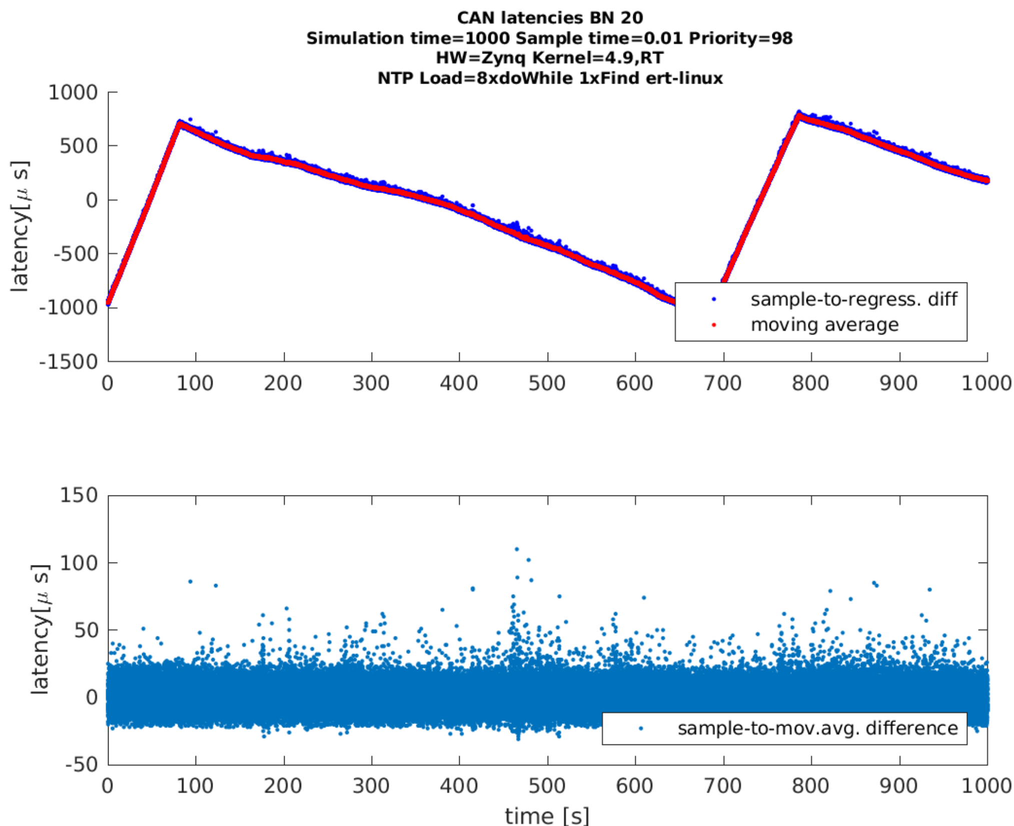


Figure 5.1. Moving average.

It can be seen that latencies computed from linear regression exceed 600 microseconds whereas real CAN latencies do not exceed 150 microseconds on picture 5.1

5.3.4 Plots and Histograms

One of the most used graphical interpretation of RT capabilities of any system is latency histograms. Two types of histograms are used in this work. Both of them are made using Gnuplot, free software distributed under its own free license [1]).

[1]) Gnuplot `http://www.gnuplot.info/`

The first type is a basic histogram, computing counts of latency sizes for a discrete number of latency offsets. The second one is not a histogram at all, but a bar plot which computes averages of given percent of worst case latencies.

All the data for histogram generation are created by calling `./prepare_hcounts.sh` and `./prepare_hpercents.sh` from `/tools/` directory with data directory as its only argument.

All the absolute times, offsets, histogram data, plots and histogram for all the data saved in `/data/*/` are generated by:

```
$ cd /tools/
$ ./prepare_all
```

Another widely used type of histogram is so called cumulative latency histogram with logarithmic scale. All the measurements could be summarized in one plot so that comparison between set-ups is possible. The first step of creating the histogram is breakdown of latencies into groups according to their size. The second step is cumulative summing up of the counts of latencies in each group. The result is a curve that says how many samples have lesser latency than the actual value on the x axis.

The script used for generating the cumulative latency histogram is a part of this work, it is named `cumul_hist.m` .The histograms are used in the bottoms of sections 6.2 and 6.3.

Chapter 6
Benchmark Results

This chapter presents all the data, plots and histograms gathered using tools and methods described in chapters 4 and 5.

The benchmarking methods are applied to diverse hardware boards including Raspberry Pi and Zynq with various parameters. This chapter provides additional comments and results of the thesis.

6.1 Oscilloscope

One of the benchmarking methods not discussed in chapter 4 is testing RT-capabilities of a PREEMPT-RT patched kernel by oscilloscope.

The initial intention was to prove that the patched kernel would secure purely deterministic behaviour of a running task. The easiest way to accomplish this task is to generate and measure precisely timed rectangular signal.

With the usage of the RT-patch we observed an improvement in the timing accuracy. Nevertheless significant violations of timing were registered. These violations were probably caused by Matlab-generated code, which doesn't lock itself in the memory. In consequence the program page could be swapped and a page fault occurs.

The measurements were obtained using following simulink model:

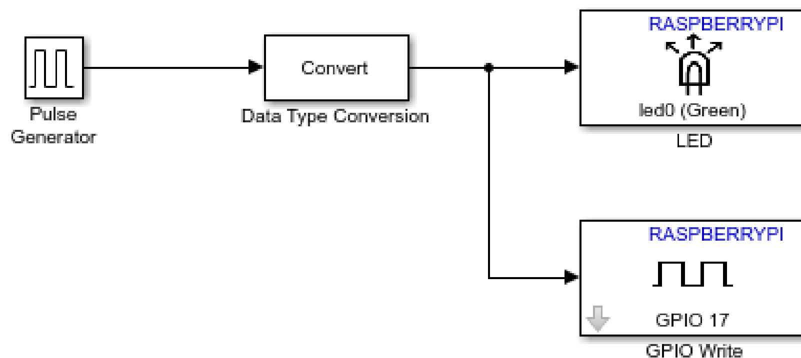


Figure 6.1. Simulink model with a LED and write to PIN

Each measurement last for 5 minutes. During the tests the processor was loaded differently.

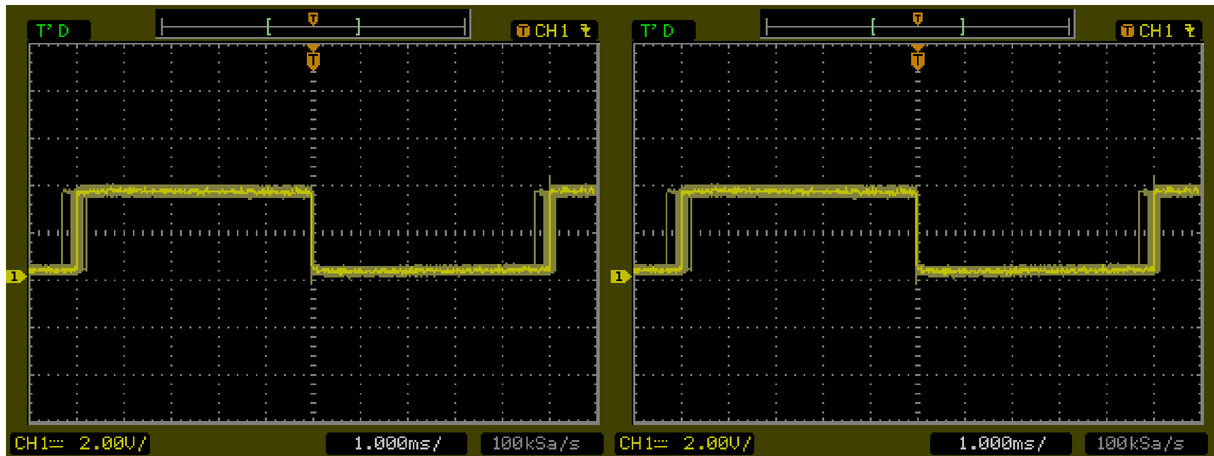


Figure 6.2. non-RT kernel (left) with no load and RT-kernel (right) with no load

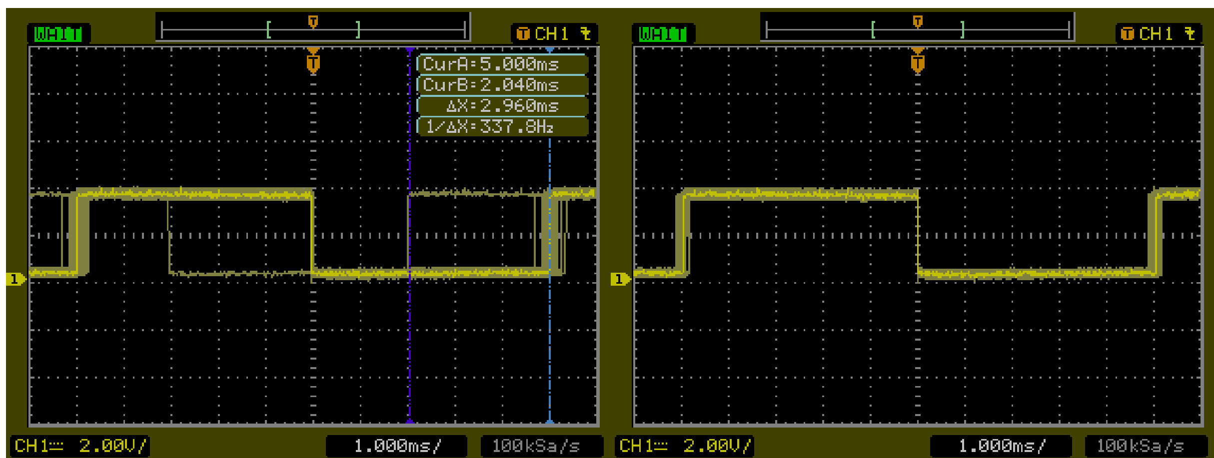


Figure 6.3. non-RT kernel (left) loaded with four do-while loops RT-kernel (right) with the same load

The first two comparisons indicate that while there are no latency violations on unloaded system, major latency peaks occur on loaded system. Thus the RT capabilities are to be evaluated mainly on loaded systems.

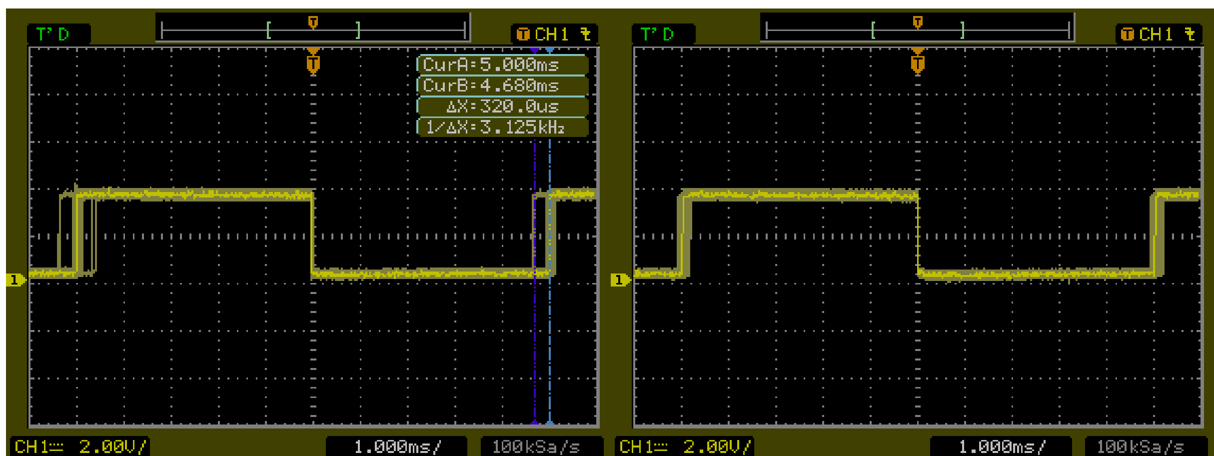


Figure 6.4. non-RT kernel (left) loaded with cyclictest loops RT-kernel (right) with the same load

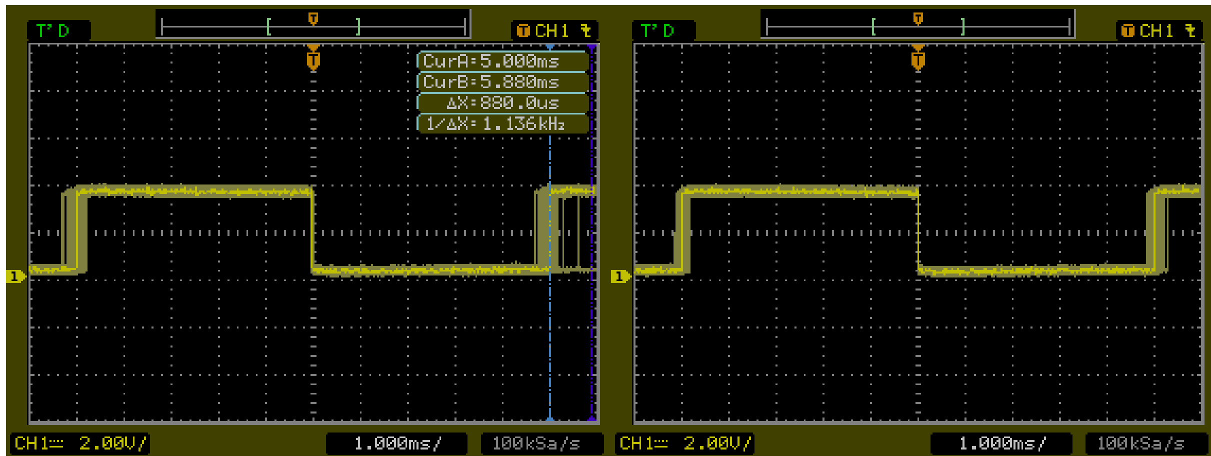


Figure 6.5. non-RT kernel (left) loaded with four do-while loops and cyclictest RT-kernel (right) with the same load

Measurement by oscilloscope was performed only without Matlab patches. Their purpose is above all to prove a timing violation caused by default code generator. In future work it would be useful to repeat the measurement even after full application of all available RT enhancements.

6.2 Sample Time Loop Latencies Measurements

The first of benchmarking methods for latency testing described in section 4.1 is the benchmarking of sample loop time latencies. It involves precise logging of the exact time when sample loop activities are executed by the Matlab model to a file. This section provides overview of the benchmark results.

During the tests, the system was loaded differently whereas the most common load involved eight *do-while* loops and one *find* loop. Most of the measurements took at least 1000 seconds with a sample time of 10 milliseconds.

The measurements are described in the same order as they were taken. We will begin with the description of benchmarking results of system with no enhancements applied. Gradually, enhancements will be added to achieve optimal results.

The first set of measurements was taken on Raspberry Pi with non-RT Linux with no Matlab patch applied.

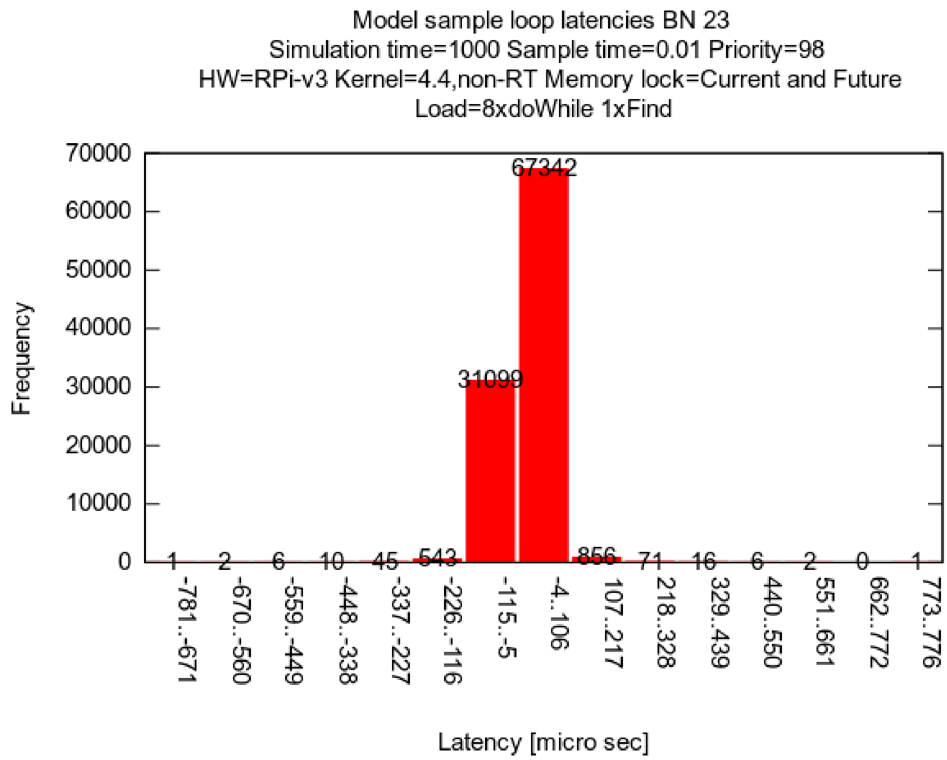


Figure 6.6. Histogram of Sample Time Loop Latencies on Raspberry Pi v3 with non-RT kernel no nanosleep patch and under high load.

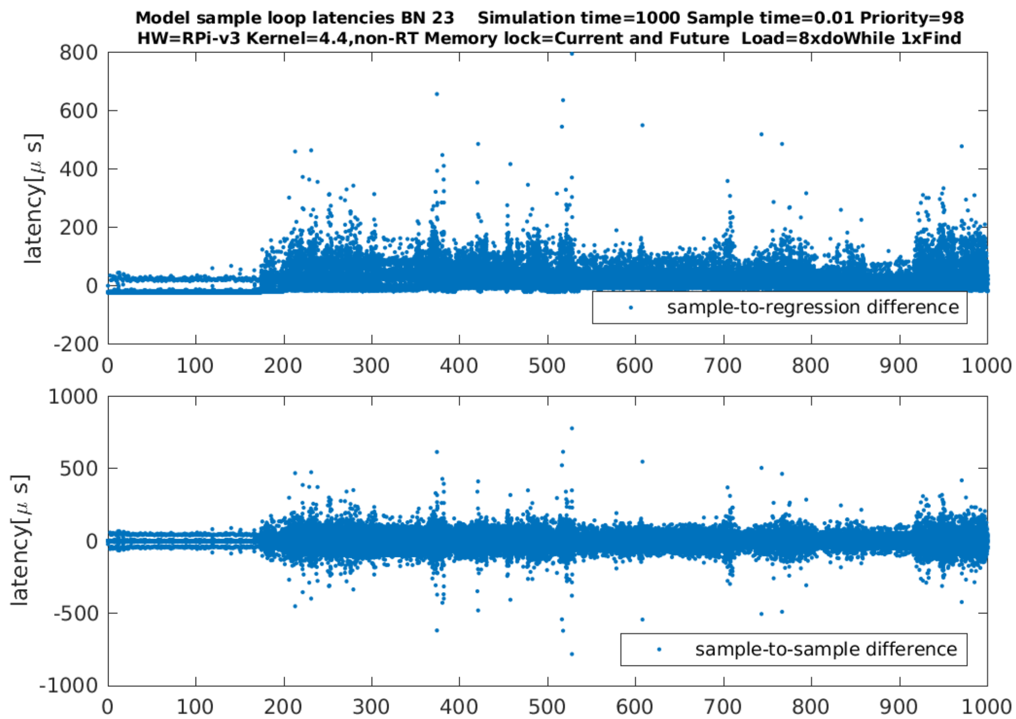


Figure 6.7. Time domain latencies plot of Sample Time Loop Latencies on Raspberry Pi v3 with non-RT kernel no nanosleep patch and under high load.

From 6.6 it can be seen that for a loaded system the latencies easily exceed the value of 500 microseconds many times. This is an expected outcome as the mainline Linux

kernel only meets soft real-time requirements. It provides only basic POSIX operations for userspace time handling but has no guarantees for hard timing deadlines.

Another representation of the same measurement on fig.6.7 demonstrates that latencies greater than 400 microseconds are not a rare phenomenon clustered into a narrow time period but repeats almost regularly and violate the RT behaviour.

The figure 6.7 show, among other things, how to load triggered about 180 seconds influences the whole system. Not only the average latencies were increased but the maximal latencies started to exceed hundreds of microseconds.

Increasing sample time from 10 milliseconds (fig 6.7) to 100 milliseconds (fig. 6.8) lowers the amount of samples by 10 to 10000. This is followed by a lowered percentage of latency peaks so that there can't be seen any peak higher than 200 microseconds on the figure 6.8. The great benefit of this measurement is the highlighted visibility of latencies noise around 75 and between 100 and 150 microseconds. This noise is going to be eliminated in further measurements by using CTU ert.

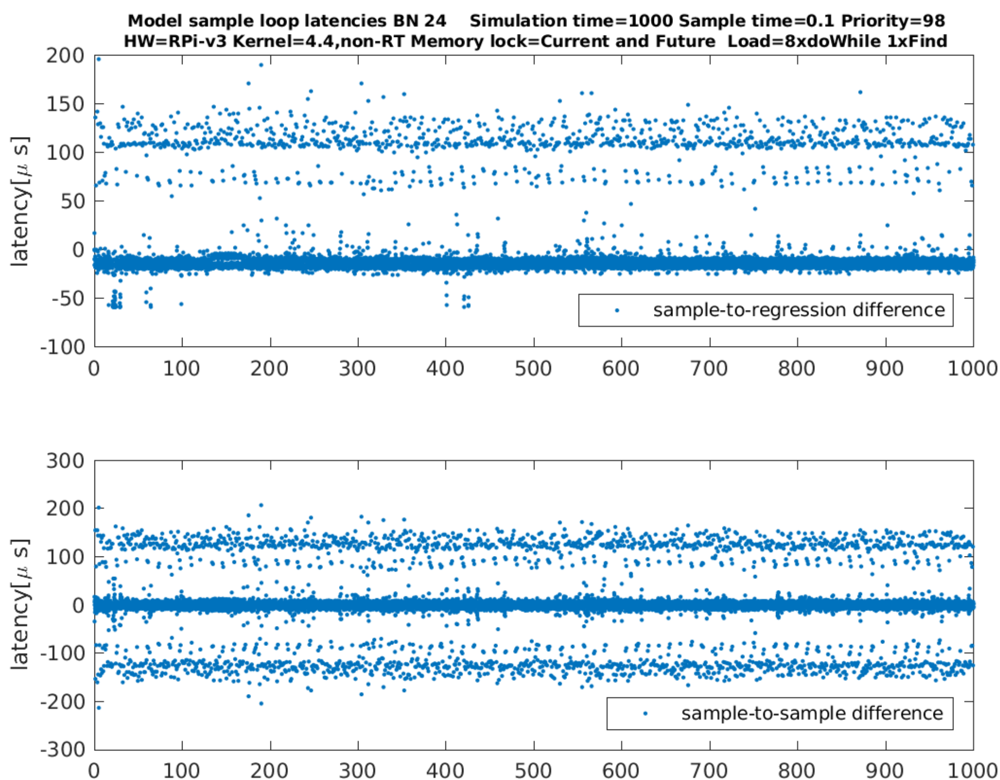


Figure 6.8. Time domain latencies plot of Sample Time Loop Latencies on Raspberry Pi v3 with non-RT kernel no nanosleep patch and under high load.

Since this is just a confirmation that a non-RT kernel is not capable of securing RT-behaviour, the work continues with patching the kernel with PREEMP-RT patch as discussed in section 3.1. Before that, we are going to use the Matlab patches without PREEMP-RT patch to demonstrate that PREEMP-RT patch is really necessary.

The result of measurement of a non-RT system with both Matlab patches applied are displayed on 6.9 and fig 6.10.

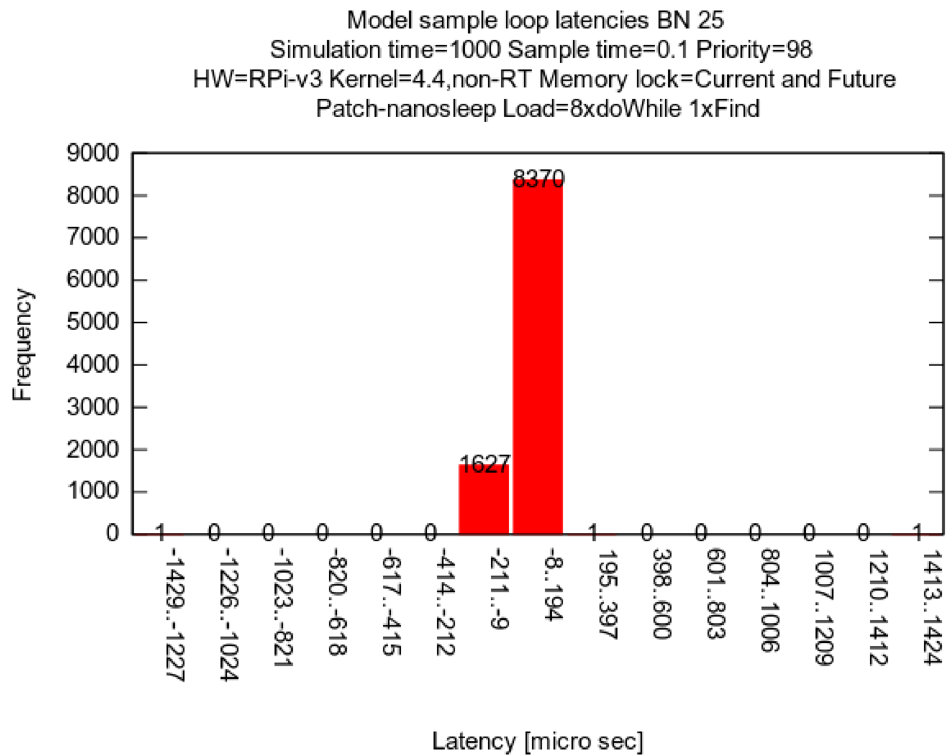


Figure 6.9. Histogram of Sample Time Loop Latencies on Raspberry Pi v3 with non-RT kernel and the Matlab patch applied, under high load.

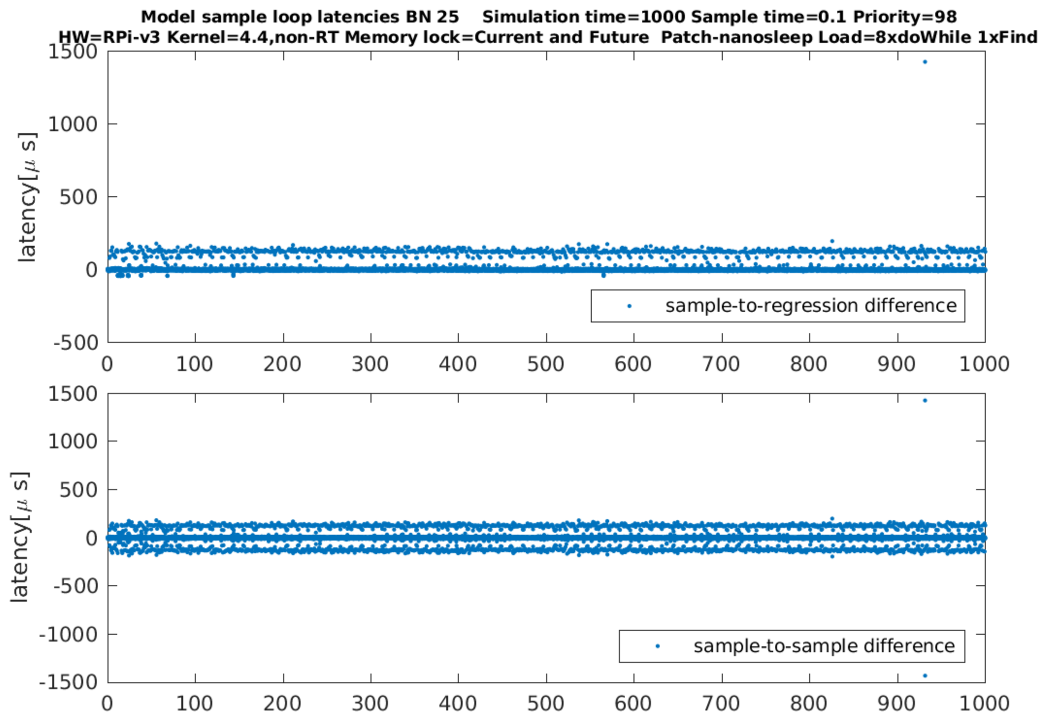


Figure 6.10. Time domain latencies plot of Sample Time Loop Latencies on Raspberry Pi v3 with non-RT kernel and the Matlab patch applied, under high load.

The majority of latencies seem to be lowered and improved, nevertheless one timing glitch overcomes the limit of one millisecond and hits the value of 1400 microseconds.

The maximal latencies are still too high, however this measurement indicates that the patched Matlab C-code generator generates source code that is less demanding on the system sources.

The next step is to prove that even a RT-capable Linux kernel alone can't secure proper timing. We are going to use original MTW ert without any patches again. The results are depicted on hist.6.11 and fig. 6.12. These measurements prove that although the most of the latencies are lowered, reaching 1000 milliseconds is not rare at all.

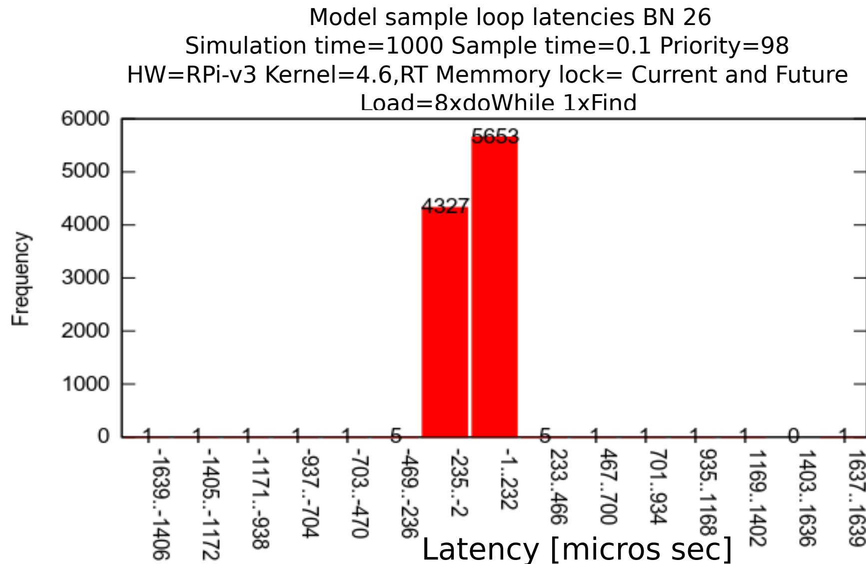


Figure 6.11. Histogram of Sample Time Loop Latencies on Raspberry Pi v3 with RT kernel and without Matlab nanosleep patch applied, under high load.

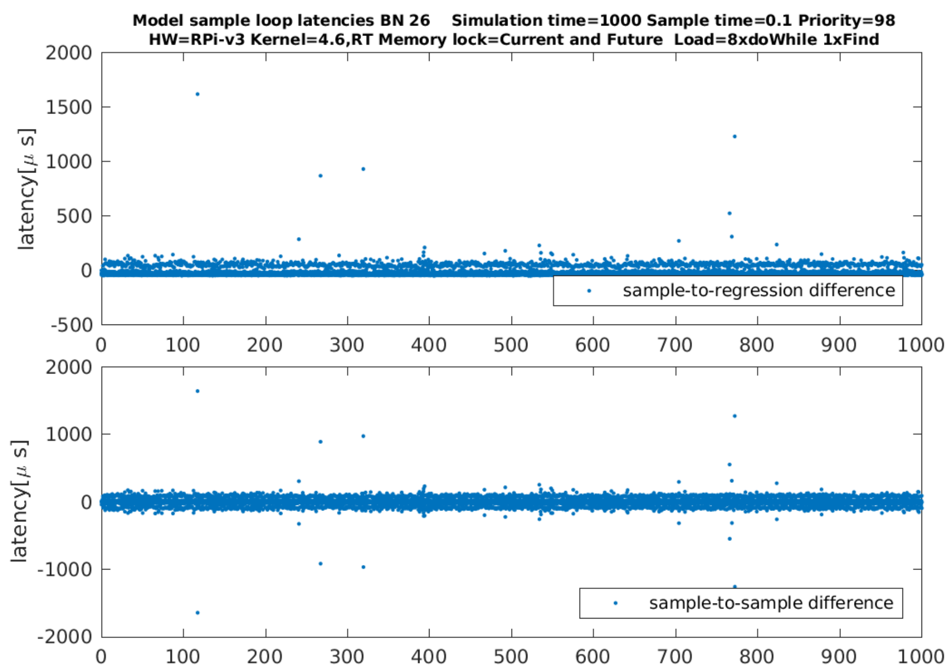


Figure 6.12. Time domain latencies plot of Sample Time Loop Latencies on Raspberry Pi v3 with RT kernel and without Matlab nanosleep patch applied, under high load.

The use of an RT-capable kernel brings much higher latencies. This fact could be explained by the incorrect implementation of the model code in C. Meanwhile the non-

RT kernel focuses on the highest throughput, the RT kernel tries to secure maximal latency cap. In the case of incorrect implementation the use of an RT kernel results in nothing but slowing down the whole system.

The next improvement would be therefore correction of the model source - patching Matlab C-code generator. The results are on pic.6.13.

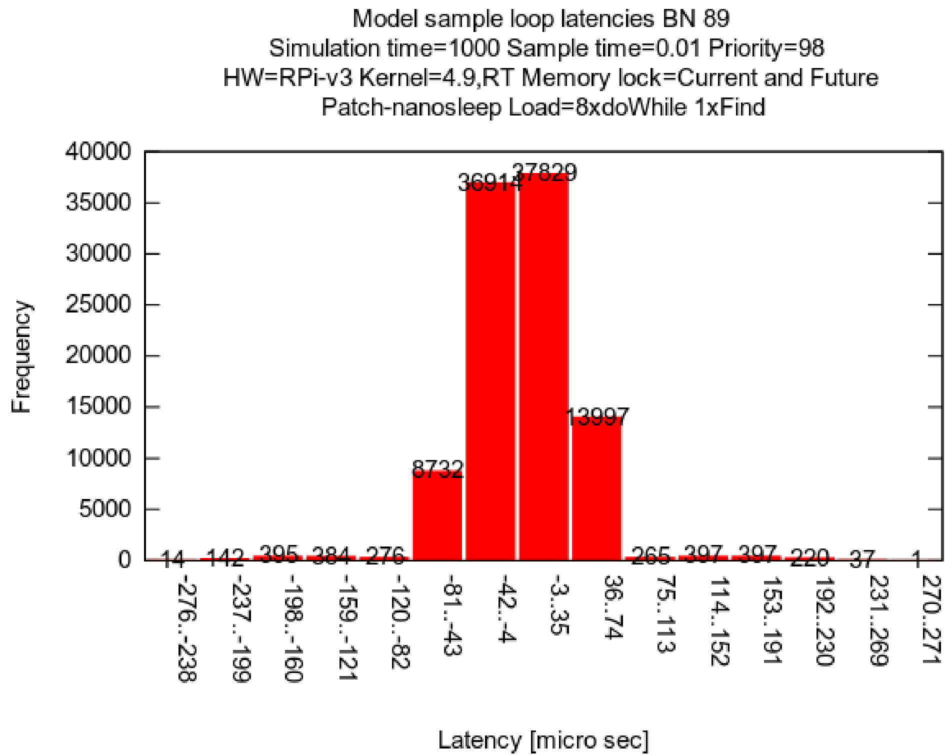


Figure 6.13. Histogram RT of Sample Time Loop Latencies on Raspberry Pi v3 with RT kernel and the Matlab patches applied, under high load.

This result satisfies our demands. To make this work more complex we've made the same tests using ert developed at CTU in Prague, FEE, Department of Control Engineering [1]). Tests made with this „CTU“ ert are marked with *ert-linux* in its codename note.

The worst of the all measurements made with CTU ert can be seen on histogram 6.14. Maximal latencies for CTU ert do not cross 160 microseconds - with PREEMPT-RT patched Kernel. This makes CTU ert the ideal candidate for further use in RT applications.

The use of CTU ert comes with yet another improvement. The noise of latencies around 75 and between 100 and 150 microseconds mentioned above in this section has been eliminated. The descriptive record could be seen on figure 6.15.

One interesting phenomenon that haven't been explained yet is the periodical sequence of narrow clusters of higher latencies visible on figure 6.15. They are present in all measurements taken on Raspberry Pi with CTU ert.

[1]) Lintarget `http://lintarget.sourceforge.net/`

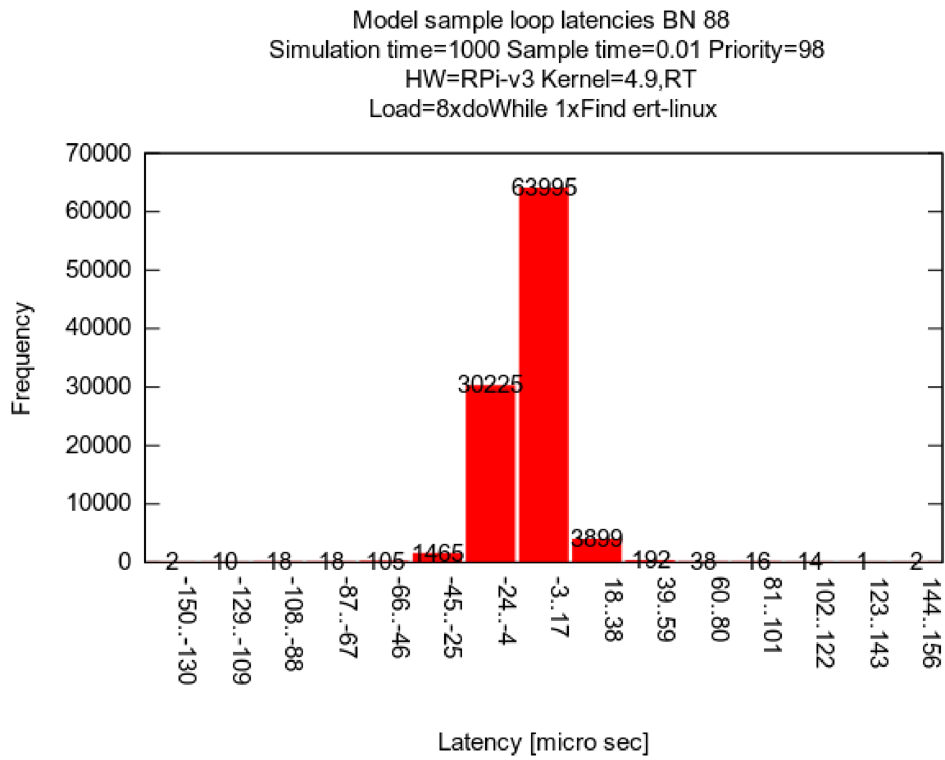


Figure 6.14. Histogram of Sample Time Loop Latencies on Raspberry Pi v3 with RT kernel and CTU ert, under high load.

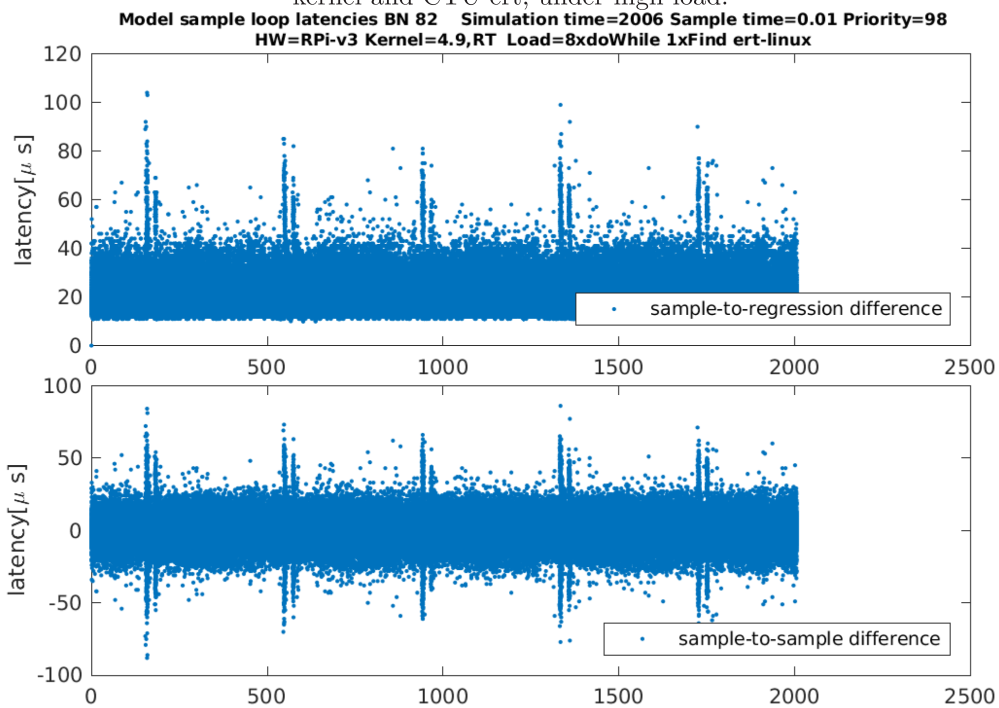


Figure 6.15. Time domain latencies plot of Sample Time Loop Latencies on Raspberry Pi v3 with RT kernel and with CTU ert, under high load.

All the results from this section are summarized in table 6.1 with many tests run multiple times.

HW	load	kernel	Patch	avg.[us]	max lat.[us]	note
RPi-v3	8W1F	4.4,non-RT	No	22	776	bn23
RPi-v3	8W1F	4.4,non-RT	No	31	207	bn24
RPi-v3	8W1F	4.4,non-RT	Yes	30	1424	bn25
RPi-v3	8W1F	4.6,RT	No	30	1639	bn26
RPi-v3	8W1F	4.6,RT	Yes	22	168	bn27
RPi-v3	8W1F	4.9,RT		6	77	ert-linux,bn124
RPi-v3	8W1F	4.9,RT		6	84	ert-linux,bn127
RPi-v3	8W1F	4.9,RT		6	85	ert-linux,bn82
RPi-v3	8W1F	4.9,RT		8	105	ert-linux,bn84
RPi-v3	8W1F	4.9,RT		8	112	ert-linux,bn86
RPi-v3	8W1F	4.9,RT		7	156	ert-linux,bn88
RPi-v3	8W1F	4.9,RT		6	71	ert-linux,bn90
RPi-v3	8W1F	4.9,RT		6	77	ert-linux,bn93
RPi-v3	8W1F	4.9,RT	No	23	228	bn126
RPi-v3	8W1F	4.9,RT	No	22	209	bn34
RPi-v3	8W1F	4.9,RT	No	27	237	bn91
RPi-v3	8W1F	4.9,RT	Yes	18	211	bn125
RPi-v3	8W1F	4.9,RT	Yes	21	235	bn35
RPi-v3	8W1F	4.9,RT	Yes	22	212	bn37
RPi-v3	8W1F	4.9,RT	Yes	28	296	bn87
RPi-v3	8W1F	4.9,RT	Yes	28	271	bn89
RPi-v3	8W1F	4.9,RT	Yes	17	191	bn92
RPi-v3	8W1F	4.9,RT	Yes	21	211	bn94
RPi-v3	No	4.9,RT	No	5	221	bn41
RPi-v3	No	4.6,RT	Yes	21	140	bn28
Zynq	8W1F	4.9,RT		9	100	ert-linux,NTP,bn29
Zynq	8W1F	4.9,RT		10	100	ert-linux,NTP,bn30

Table 6.1. Overview of Matlab model Sample Time Loop latencies

Table 6.1 brings additional information about latencies measured with CTU ert on Zynq board. This board is based on an ARM processor thus could be seen as a reference to results measured on Raspberry Pi.

All the measurements that lasted longer than 1000 seconds have been summarized in one cumulative latency histogram so that comparison between set-ups is possible. The result is depicted on figure 6.16.

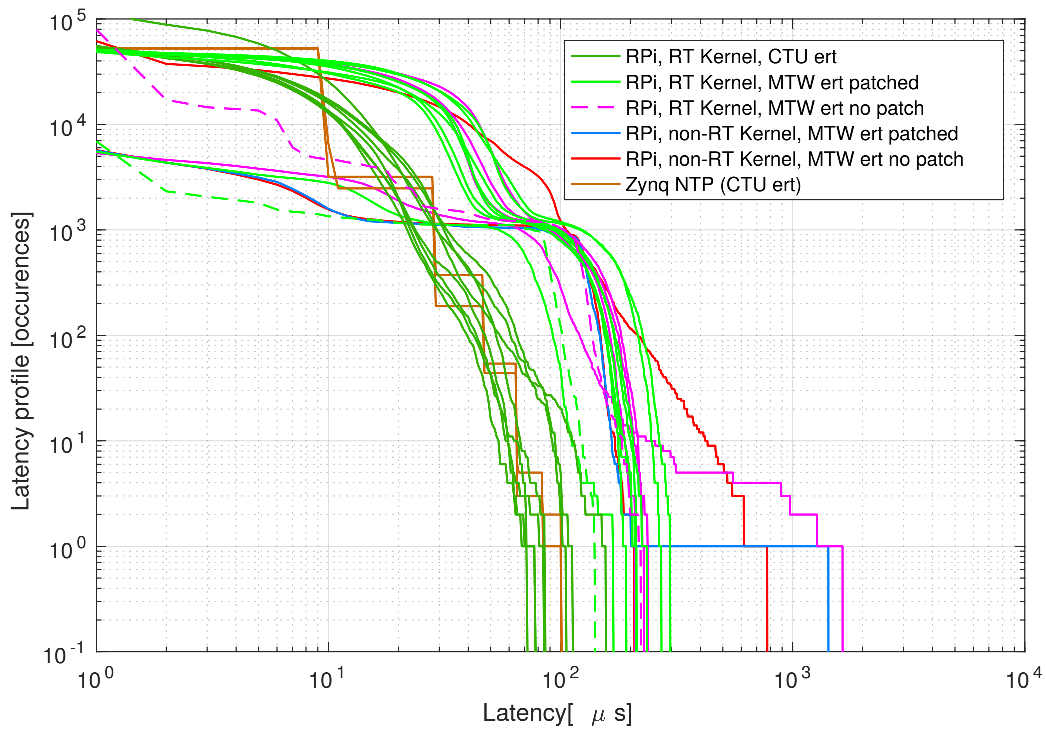


Figure 6.16. Matlab model sample time loop cumulative latency histogram.

It can be seen that measurements that were taken with non-RT kernel or without use of Matlab patches shows much bigger latencies than measurements with all the enhancements applied. This is a confirmation of the observation made above in this section.

6.3 CAN Bus Communication Measurements

The final stage of benchmarking and the second benchmarking method described in section 4.2 involves benchmarking CAN bus communication. This section provides an overview of the benchmark results and adds additional commentaries.

Main purpose of the measurements is to verify that the RT enhancements are as efficient in improving CAN bus communication as they were in improving the Matlab model sample time loop latencies. If the enhancements are not as successful as expected, this section will cover the most probable reasons for its failure.

The following measurements were accomplished for Raspberry Pi version 2 and version 3. Additional measurements were made on the Zynq board.

The measurements were performed on the CAN-bench with the help of *can-utils*. The command used to obtain measurements follows:

```
candump -t a -H can0
```

The very first measurement was made with a non-RT kernel and without the Matlab patches. The result can be seen on histogram 6.17.

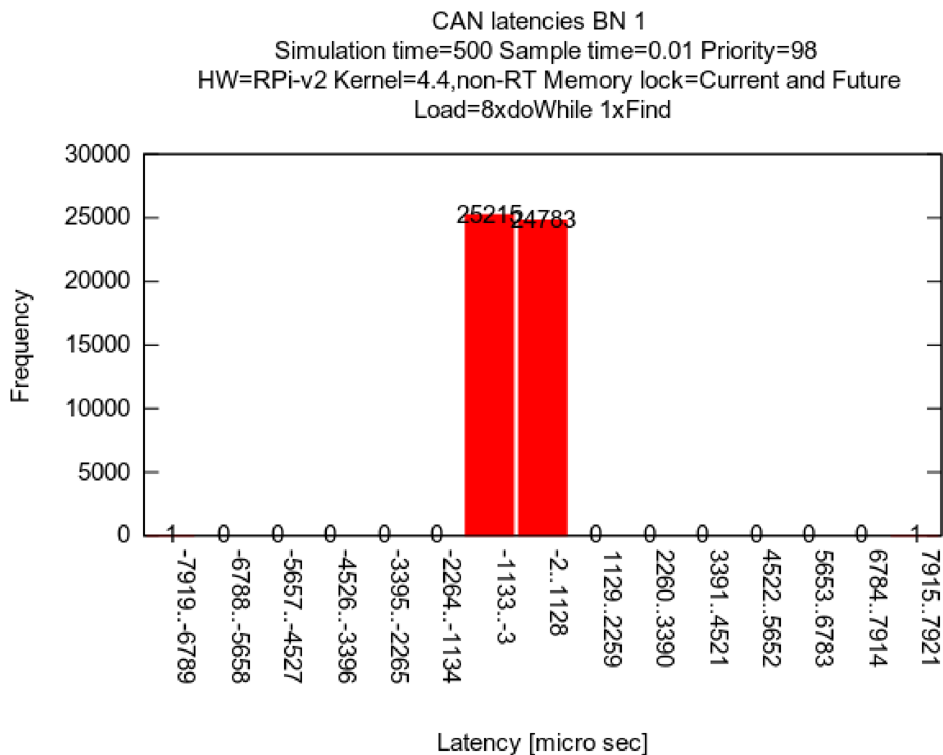


Figure 6.17. Histogram of CAN message Latencies on Raspberry Pi v2 with non-RT kernel and without Matlab nanosleep patch, under high load.

One latency exceeding 7 millisecond can be seen in histogram 6.17. This observation confirms the assumption of poor results and the need for patches.

Using the same approach as in section 6.2, both Matlab patches were applied and RT-kernel was used. The results of the first measurements of fully patched and enhanced system are on histogram 6.18.

It was revealed that the latencies were even worse with the application of the patches and with use of RT kernel. In an attempt to improve this, we tried to boost SPI and

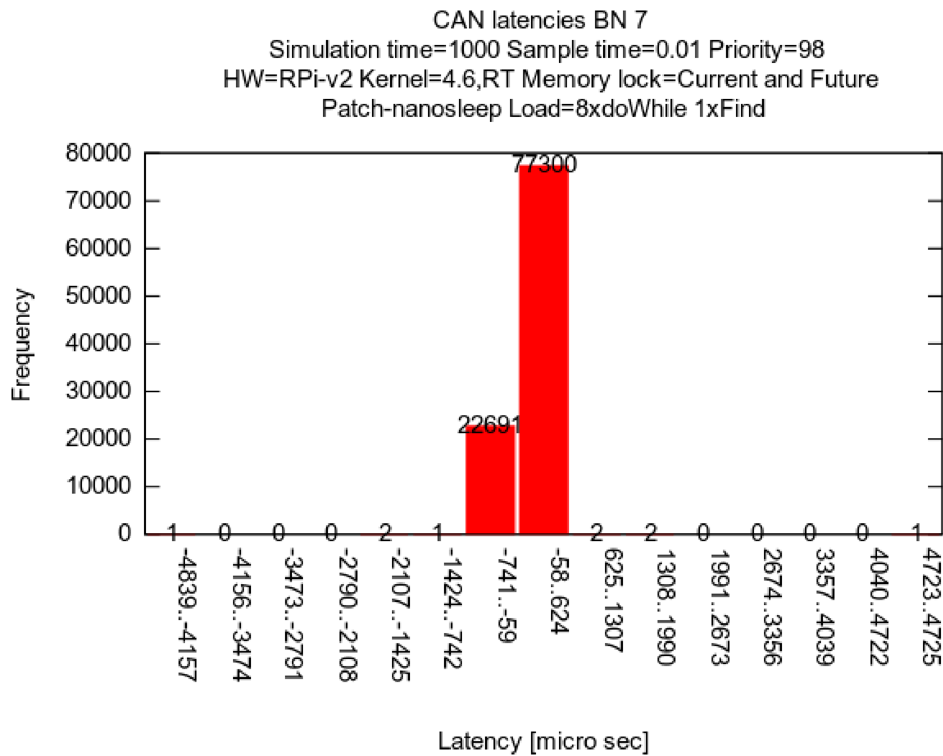


Figure 6.18. Histogram of CAN message Latencies on Raspberry Pi v2 with non-RT kernel and the Matlab patches applied , under high load.

MCP1155 interrupt worker thread priorities. The corresponding thread IDs could be found the following way:

```
$ cat /proc/interrupts
84:    0    0    0    0  ARMCTRL-level  86 Edge   3f204000.spi
191:   0    0    0    0  ARMCTRL-level  86 Edge   mcp251x
```

The id's of the interrupts must be saved. Then, the running processes are scanned and the ones responsible for interrupt handling are found using the saved id's. The example output follows:

```
$ ps xaw
   639 ?        S      0:00 [irq/191-mcp251x]
   213 ?        S      0:00 [irq/84-3f204000]
```

The priority boot can be done by *schedtool* , which modifies the scheduling policy and priority. The policy was set to `SCHED_FIFO` and the priorities have been increased to 95. The following commands do the job:

```
$ schedtool -F -p95 639
$ schedtool -F -p95 213
```

The improvement of results with boosted priority are depicted on hist 6.19.

The priority boost improved the vast majority of latencies but the worst case latency persists. After all the patches and enhancements the CAN RT behaviour stays unpredictable with high latencies. Another typical example can be seen on histogram 6.20.

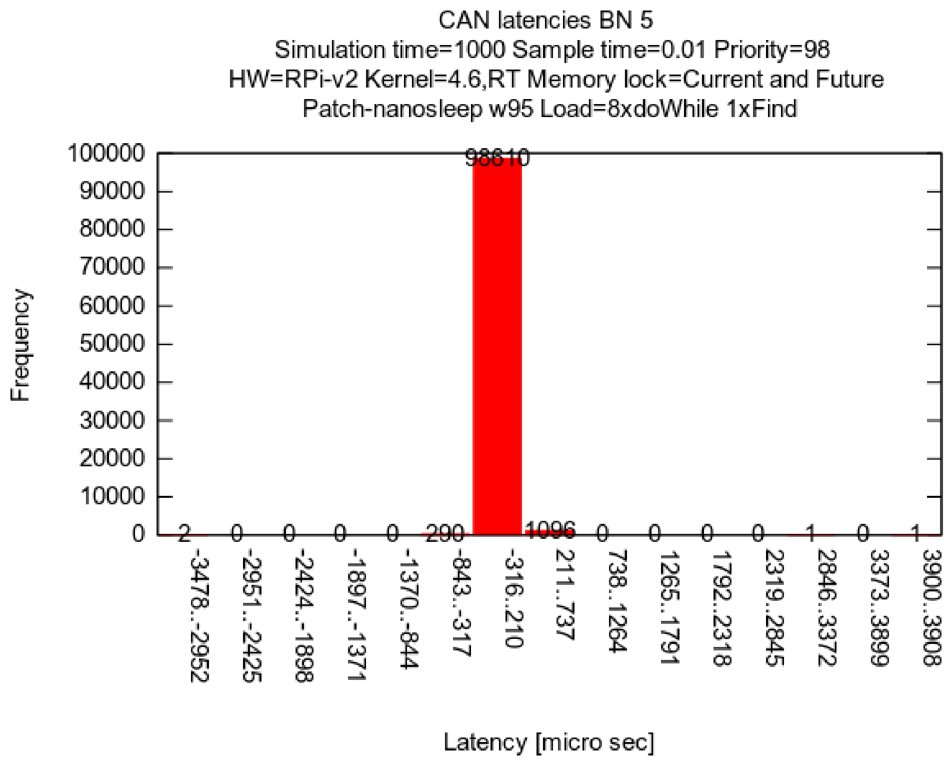


Figure 6.19. Histogram of CAN message Latencies on Raspberry Pi v2 with RT kernel and the Matlab patches applied, under high load and a worker thread priority boost.

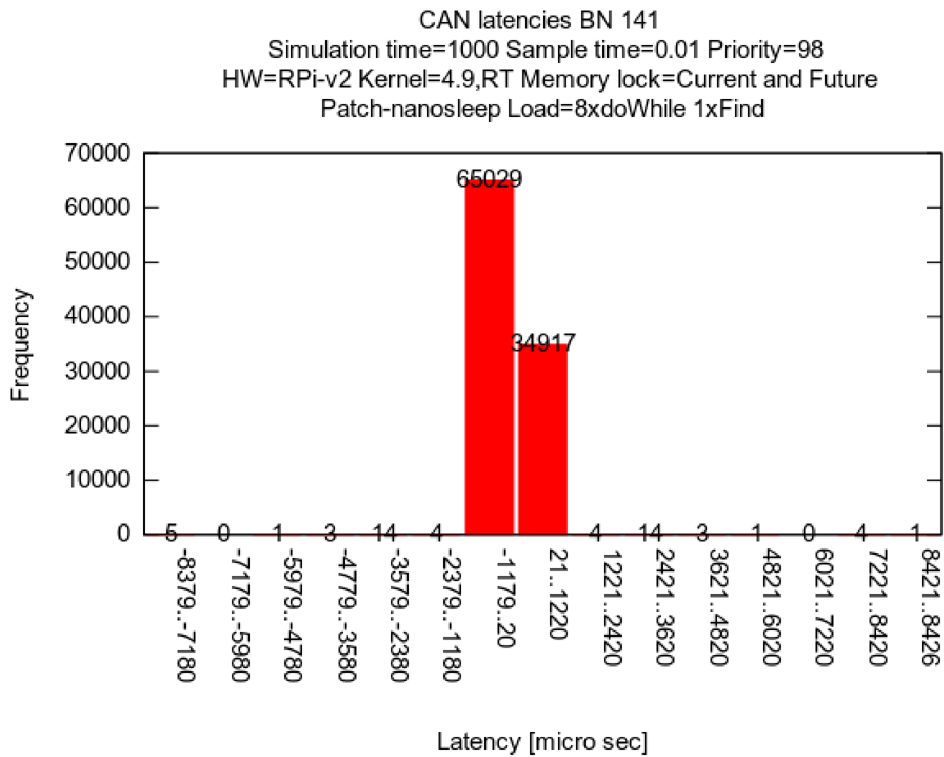


Figure 6.20. Histogram of CAN message Latencies on Raspberry Pi v2 with RT kernel and the Matlab patches applied, under high load.

The same test were run on Raspberry Pi with CTU ert, nevertheless the performance stayed poor.

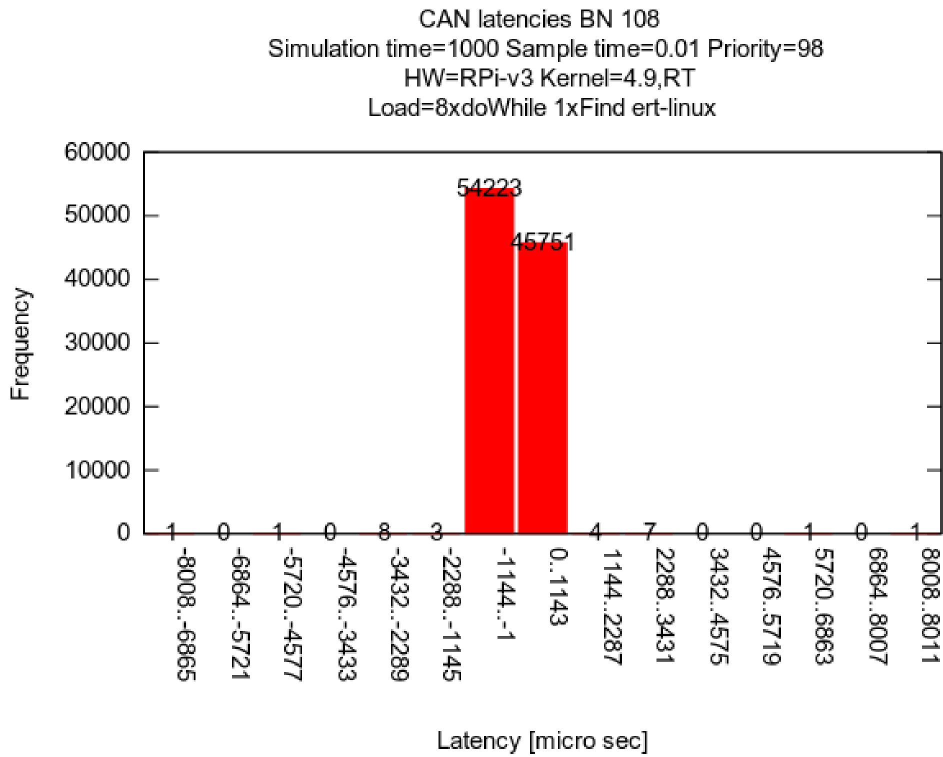


Figure 6.21. Histogram of CAN message Latencies on the Zynq board with non-RT kernel and the Matlab patches applied, under high load.

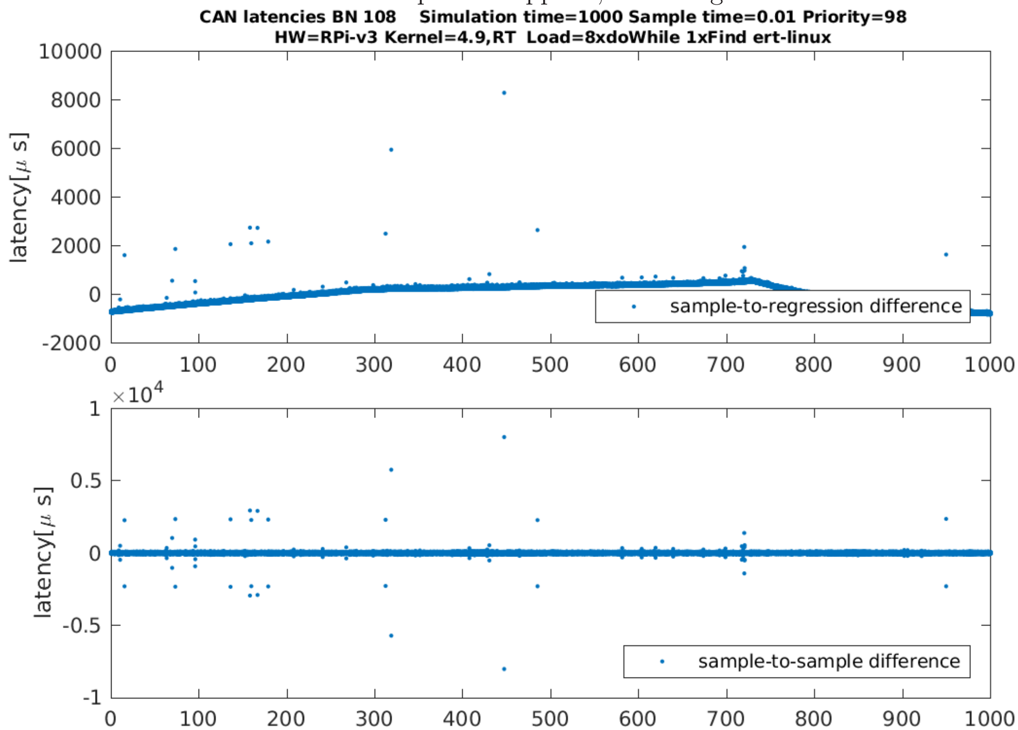


Figure 6.22. Time domain latencies of CAN message Latencies on the Zynq board with RT kernel, under high load. The NTP client is turned on.

Similar testing was done on a Zynq board. The results are in hist.6.23 These results are especially interesting in the time-domain on figure 6.24

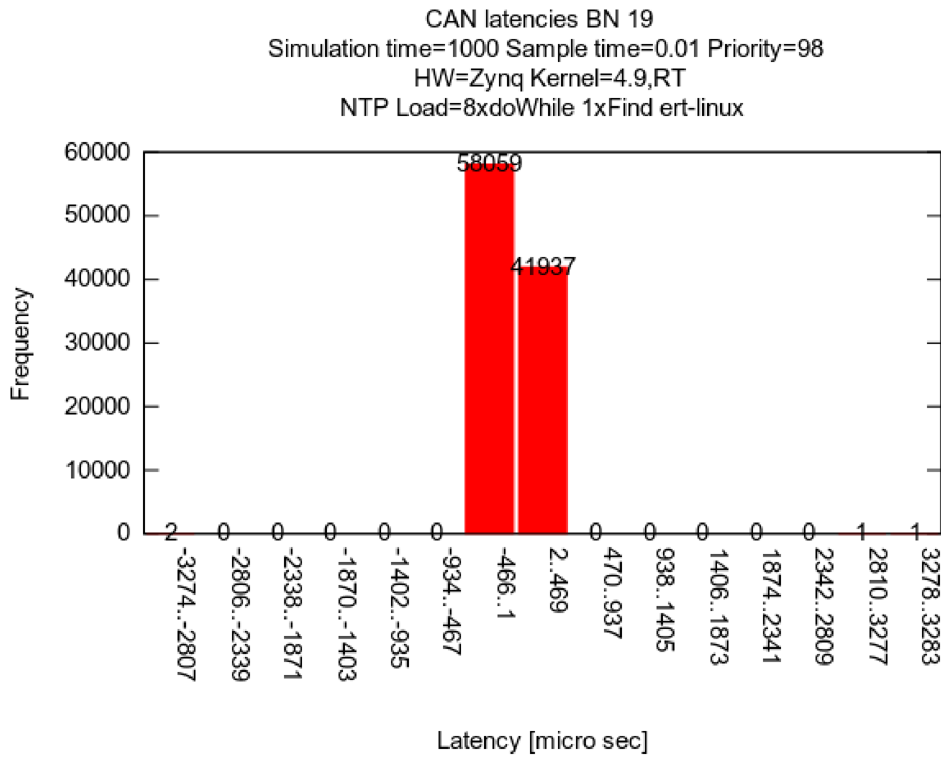


Figure 6.23. Histogram of CAN message Latencies on the Zynq board with non-RT kernel and the Matlab patches applied, under high load.

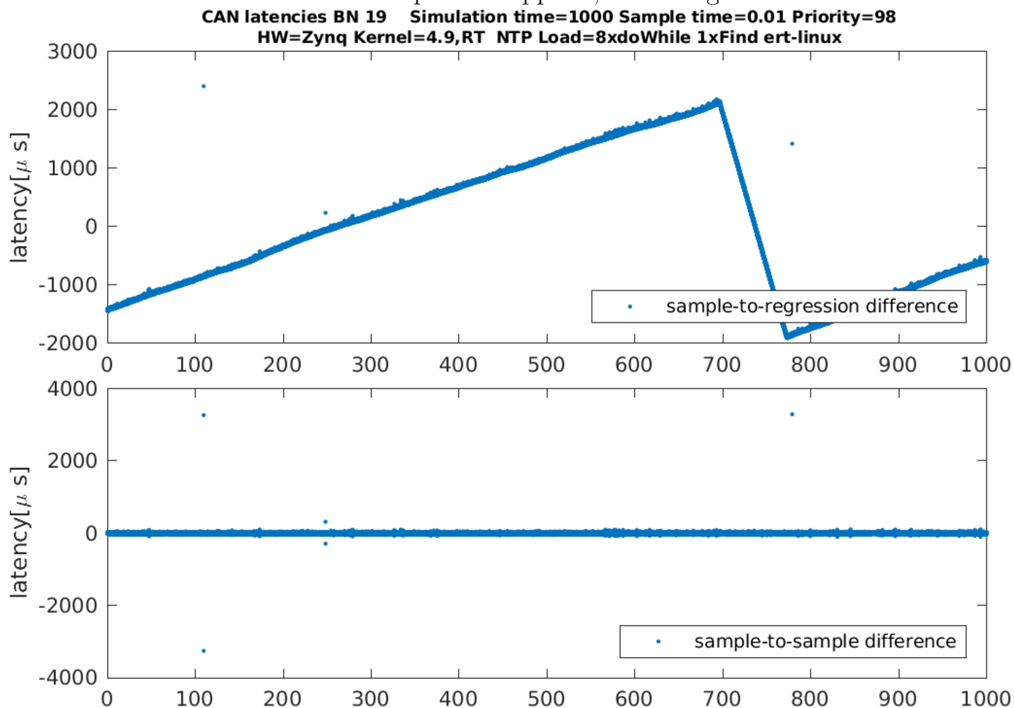


Figure 6.24. Time domain latencies of CAN message Latencies on the Zynq board with RT kernel, under high load. The NTP client is turned on.

The very steep evolution of latencies on figure 6.24 indicates problems with time synchronization. After a little examination it was discovered that a NTP client is

running on Zynq machine. The client was turned off and a new set of measurements was made. Typical example on histogram 6.25.

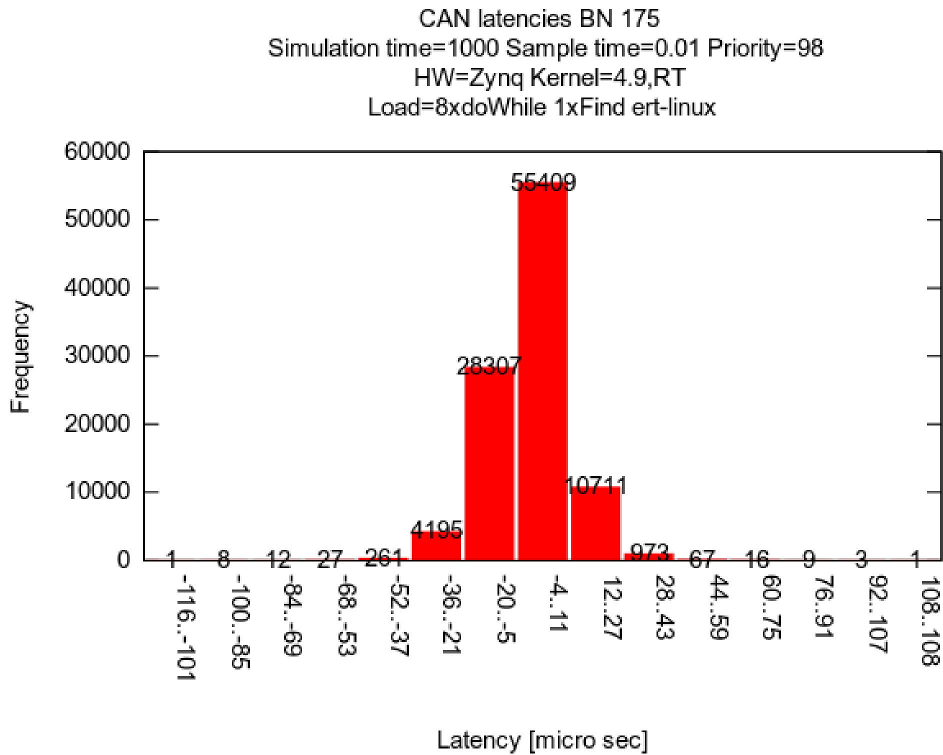


Figure 6.25. Histogram of CAN message Latencies on the Zync board with RT kernel, under high load. The NTP client is turned off.

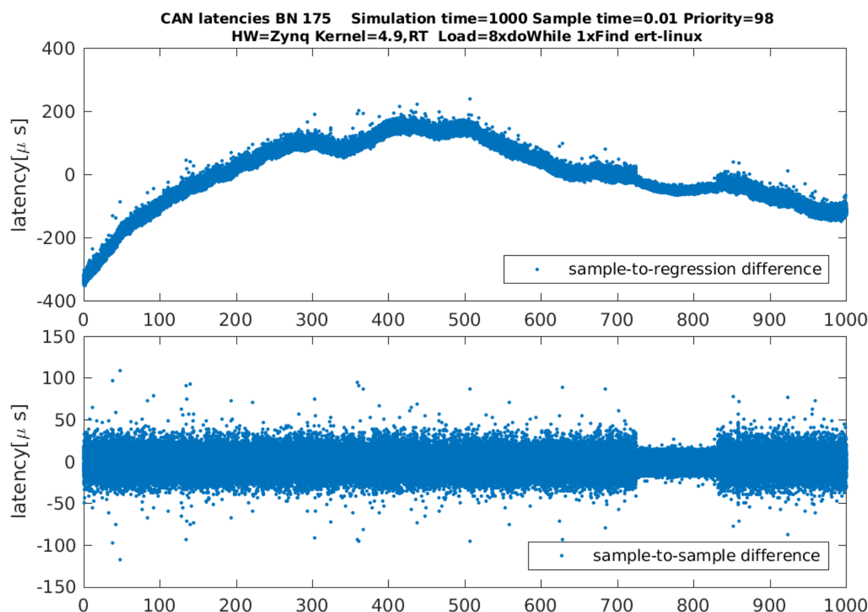


Figure 6.26. Time domain latencies on the Zync board with RT kernel, under high load. The NTP client is turned off.

All the measurements done with running NTP client are marked with *NTP* at it's codename note. *Moving Average-Based Comparison* described in section 5.3.3 was created to handle these disturbances and show real latencies.

A quick overview of the results of the measurements are in the tables 6.2 and 6.3

HW	kern	nslp	avg	max	note
RPi-v2	4.4,non-RT	No	9	1075	bn9
RPi-v2	4.6,RT	Yes	12	896	w95,bn10
RPi-v2	4.9,RT		12	9608	ert-linux,bn137
RPi-v2	4.9,RT		7	3753 0	ert-linux,bn143
RPi-v2	4.9,RT		10	6117	ert-linux,w90,bn145
RPi-v2	4.9,RT	Yes	16	9695	bn139
RPi-v3	4.9,RT		8	9322	ert-linux,bn106
Zynq	4.9,RT		7	131	ert-linux,NTP,bn22
Zynq	4.9,RT		7	88	ert-linux,NTP,bn165
Zynq	4.9,RT		7	116	ert-linux,bn168
Zynq	4.9,RT		7	75	ert-linux,bn170
Zynq	4.9,RT		7	194	ert-linux,bn172
Zynq	4.9,RT		7	85	ert-linux,bn174

Table 6.2. Overview of CAN latencies on system under no load

HW	kern	nslp	avg	max	note
RPi-v2	4.4,non-RT	No	57	7921	bn1
RPi-v2	4.6,RT		42	1725	ert-linux,w95,bn2
RPi-v2	4.6,RT	No	56	17085	bn4
RPi-v2	4.6,RT	Yes	61	3908	w95,bn5
RPi-v2	4.6,RT	Yes	63	21323	w95,bn6
RPi-v2	4.6,RT	Yes	56	4725	bn7
RPi-v2	4.9,RT	Yes	52	8426	bn141
RPi-v3	4.4,non-RT	No	35	5271	bn11
RPi-v3	4.9,RT		29	8011	ert-linux,bn108
Zynq	4.9,RT		10	3283	ert-linux,NTP,bn16
Zynq	4.9,RT		9	102	ert-linux,NTP,bn17
Zynq	4.9,RT		9	3285	ert-linux,NTP,bn18
Zynq	4.9,RT		10	3283	ert-linux,NTP,bn19
Zynq	4.9,RT		10	120	ert-linux,NTP,bn20
Zynq	4.9,RT		9	96	ert-linux,NTP,bn21
Zynq	4.9,RT		11	301	ert-linux,NTP,bn166
Zynq	4.9,RT		7	111	ert-linux,bn169
Zynq	4.9,RT		8	95	ert-linux,bn171
Zynq	4.9,RT		8	90	ert-linux,bn173
Zynq	4.9,RT		9	108	ert-linux,bn175
Zynq	4.9,RT		8	119	ert-linux,bn176
Zynq	4.9,RT		9	109	ert-linux,bn177

Table 6.3. Overview of CAN latencies on a high-loaded system.

The cumulative latency histogram for CAN messages in on figure 6.27. Individual set-ups are distinguished by color. Measurements taken on not loaded systems are marked by dashed lines meanwhile the measurements taken on high-loaded systems are marked with solid lines.

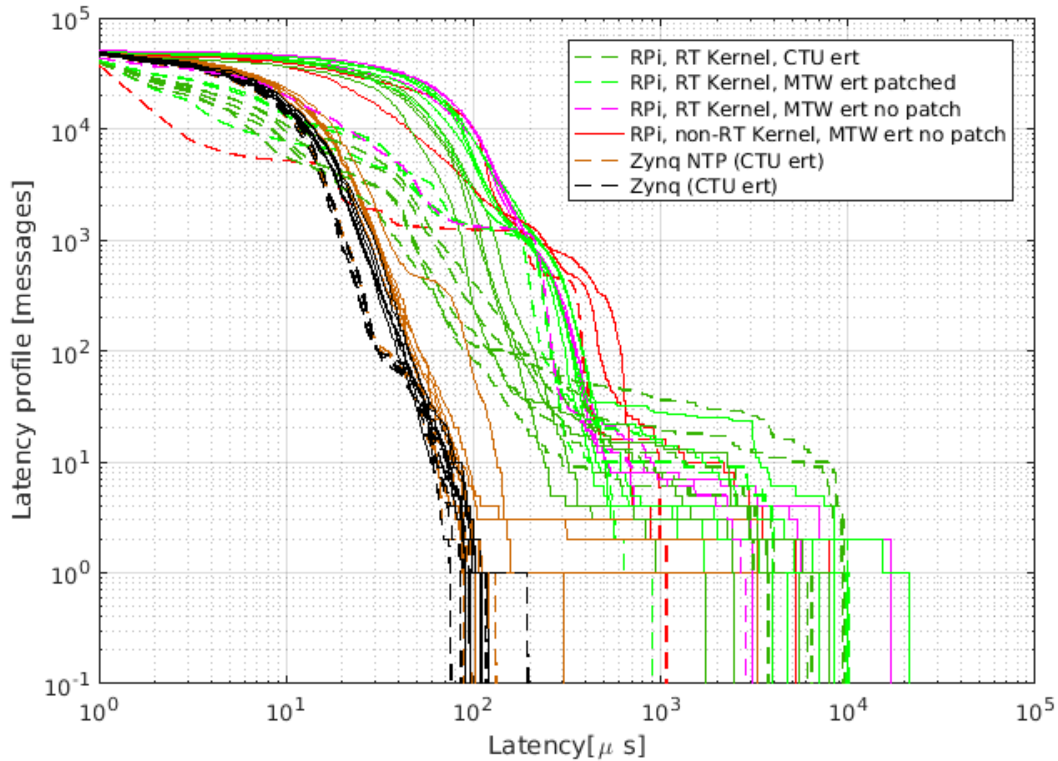


Figure 6.27. Cumulative latency histogram of CAN messages.

It can be seen that measurements with the same set-up and taken on the same hardware tend to associate in shapes reminiscent of a disintegrating rope. The black-coloured Zynq measurements are the most left, which means they have the lowest both average and maximal latencies. It is also obvious that all the pink and light and dark green measurements made on Raspberry do not differentiate each other a lot. This means that no enhancements nor the use of CTU ert could secure meeting requirements of RT application.

The explanation of the poor results of CAN communication on Raspberry Pi could be combination of several factors. The weakest part of the whole system is the MCP2515 CAN controller used by the *PiCAN* board. It uses the SPI interface for communication with Raspberry Pi. Unfortunately the SPI is used for reading and writing to controllers registers instead of handling a CAN message as a whole. As a result, the SPI latencies generated on each SPI interrupt are counted together.

Due to SPI implementation in the Linux kernel (even the PREEMP-RT patched), a single SPI latency could reach around one millisecond.

Another drawback might be the use of the SocketCAN API, which is far from ideal. On the other hand, all benchmarks made on the Zynq board, which also uses Socket-CAN, show good results. [21] [9].

Chapter 7
Problems

During the development process, many technical problems occurred. They were not in the focus of interest but took a time to resolve. This chapter provides a descriptions and solutions those which could affect later users or are particularly interesting ones.

7.1 Compilation Problems

This section describes problems that could be solved by modifying the model itself by altering s-function source files or by modifying some Matlab TLC files. Compilation problem occur at the compilation time.

7.1.1 Different Step Sizes

The first problem occurred when the `X` in

```
ssSetSampleTime(S, 0, X);
```

was set to 0.1 although it is the `time_t` data type and it is not explicitly said to be integer [1]) . The error message was the following:

```
Invalid setting for fixed-step size (1.0) in model $write_fnc$. All
sample times in your model must be an integer multiple of the fixed-step
 size.
Component:Simulink | Category:Modelerror
The sample time period (0.1) of $write_fnc/S-Function1$ is not an
integer multiple of the fixed step size (1.0) specified for model.
Defined in: ssSetSampleTime
```

Setting `X=INHERITED_SAMPLE_TIME` works properly. Setting `X` an integer value works properly as well and sets the fixed sample time to the required value.

[1]) Matlab Documentation ssSetSampleTime `https://www.mathworks.com/help/simulink/sfg/sssetsampletime.html`

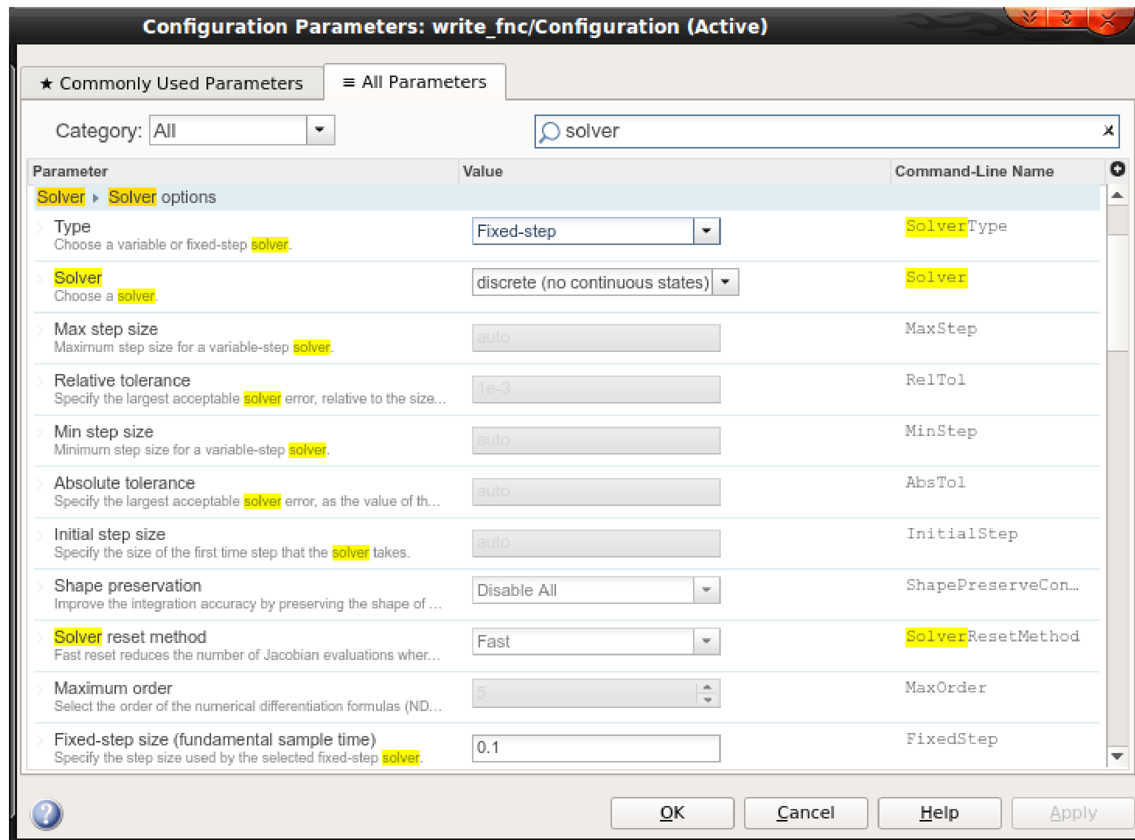


Figure 7.1. Configuring fundamental sample time

7.1.2 Modifing Default Make Command

When using MathWorks default `ert` with support fo Raspberry Pi it is impossible to change the make command (MakeCommand property). Its box in the model configuration pane is disabled. Modifying it's value in model XML results in the following compilation error:

```
Error encountered while executing PostCodeGenCommand for model
 'latency_demo': The following model parameters are not compatible
  with the selected hardware board:
   MakeCommand is set to make_rtw -DPATCH_NANOSLEEP, but it is
    expected to be set to make_rtw
To resolve this conflict, set incompatible model parameters to their
 default values. Otherwise, set 'System target file' to a different
  value.
Caused by:
The following model parameters are not compatible with the selected
 hardware board:
   MakeCommand is set to make_rtw -DPATCH_NANOSLEEP, but it is expected
    to be set to make_rtw
To resolve this conflict, set incompatible model parameters to their
 default values. Otherwise, set 'System target file' to a different
 value.
```

7.1.3 Missing Libraries

Another error occurred when using Matlab CAN blocks with MTW ert on Raspberry Pi. The error message follows:

```
Error executing command "touch -c /home/pi/matPiBuild/
can_test_ert_rtw/*.*;make  -f can_test.mk all
-C /home/pi/matPiBuild/can_test_ert_rtw". Details:

STDERR: In file included from can_test.c:19:0:
can_test.h:41:82: fatal error: /usr/local/MATLAB/R2016b/
toolbox/shared/can/src/scanutil/can_message.h:
No such file or directory
 #include "/usr/local/MATLAB/R2016b/toolbox/shared/can/
 src/scanutil/can_message.h"

compilation terminated.
make: *** [can_test.c.o] Error 1
```

This error has been solved by adding complete library structure with the file to the appropriet directory on Raspberry Pi.

```
raspberrypi-matlab:~ $ sudo mkdir -p /usr/local/MATLAB/R2016b/\
toolbox/shared/can/src/scanutil
home:~$ scp /usr/local/MATLAB/R2016b/\
toolbox/shared/can/src/scanutil/can_message.h pi@172.22.4.2:
raspberrypi-matlab:~ $ sudo mv can_message.h /usr/local/MATLAB/R2016b/\
toolbox/shared/can/src/scanutil/
```

7.2 MDL Format Problems

The MDL model format was used to store Matlab model at the beginning of the work. The format is obsolete according to MathWorks, nevertheless it was used for it's simplicity and easy versioning. The problem occurred when converting the old MDL models to XSL. The following error message was displayed when trying to run the converted model on Raspberry Pi in external mode:

```
Error occurred while executing External Mode MEX-file 'ext_comm':
Failed to connect to the target. Possible reasons for the failure:
a) The target is not switched on.
b) The target is not connected to your host machine.
c) The application for the model is not running on the target. You might
have clicked the Stop button. If the Run button is not dimmed, click it.
Otherwise, click the Build button, which downloads and runs your
application on the target.
Caused by:
Verbosity argument must be a real, scalar, integer value in the range:
 [0-1].
```

This error is probably caused by the lack of support for translation from MDL to XSL. The following code snippet demonstrates the difference between a properly generated XLS file and a file re-generated from an MDL model file. The logical type is translated to type double which results in the execution error listed below.

```
<Array PropName="CoderTargetData" Type="Struct" Dimension="1*1">
  <MATStruct>
  <Field Name="UseCoderTarget" Class="double">1.0</Field>
     <Field Name="TargetHardware" Class="char">Raspberry Pi</Field>
     <Array PropName="ConnectionInfo" Type="Struct" Dimension="1*1">
       <MATStruct>
         <Array PropName="TCPIP" Type="Struct" Dimension="1*1">
           <MATStruct>
             <Field Name="IPAddress" Class="char">
               codertarget.raspi.getDeviceAddress</Field>
             <Field Name="Port" Class="char">17725</Field>
             <Field Name="Verbose" Class="double">0.0</Field>
             <Field Name="RunInBackground" Class="double">0.0</Field>
           </MATStruct>
         </Array>
       </MATStruct>
     </Array>
```

```
<Array PropName="CoderTargetData" Type="Struct" Dimension="1*1">
  <MATStruct>
  <Field Name="UseCoderTarget" Class="logical">1</Field>
     <Field Name="TargetHardware" Class="char">Raspberry Pi</Field>
     <Array PropName="ConnectionInfo" Type="Struct" Dimension="1*1">
       <MATStruct>
         <Array PropName="TCPIP" Type="Struct" Dimension="1*1">
           <MATStruct>
             <Field Name="IPAddress" Class="char">
               codertarget.raspi.getDeviceAddress</Field>
             <Field Name="Port" Class="char">17725</Field>
             <Field Name="Verbose" Class="logical">0</Field>
             <Field Name="RunInBackground" Class="logical">0</Field>
           </MATStruct>
         </Array>
       </MATStruct>
     </Array>
```

Chapter 8
Conclusion

It has been verified that application of real-time enhancements (PREEMP-RT patch) on the Linux kernel improves it's real-time behaviour. The main impact has been observed in the shortening of latencies of Matlab model Sample Time Loop.

The latencies exceed 700 microseconds on high-loaded system without any Matlab or Linux patch applied. The latencies goes even worse with the application of PREEMP-RT patch and reaches 1639 microseconds in the worst case. Similar latencies exceeding 1420 microseconds were observed with a non-RT Linux kernel but Matlab patches applied.

Improved latencies occurred after application both Matlab and PREEMPT-RT patches. The latencies did not exceed 300 microseconds in benchmark that taken more than 2 hours with 100 samples per second.

The two ert files were used during the benchmarks. The first of them, MathWorks (MTW) ert has been modified by the Matlab patch. The second one, developed at CTU was used without any further modification. It was revealed that even after the application of Matlab patch the RT capabilities of CTU ert are better than that of MTW ert. Average latencies of MTW ert (with all the patches applied) are around 20-28 microseconds meanwhile average latencies of CTU ert were around 7 microseconds and the maximum latencies didn't exceed 160 microseconds.

The second main part of the thesis deals with benchmarking CAN bus based on PiCAN extension board and Simulink CAN blocka and SocketCAN Linux API.

The latencies of CAN messages transmitted on Raspberry Pi exceed 7 milliseconds on a high-loaded system without any Matlab or Linux patch applied. The same patches as in the previous part were applied but with no significant impact on the latency times. The latencies often exceed 8 milliseconds on a system with both Matlab and PREEMP-RT patch applied. The similar latencies were measure on system under no load, nevertheless there were no latencies exceeding 10 milliseconds at all.

No improvement was observed even with use of CTU ert. The latencies exceed 8 milliseconds.

As a result, we do not recommend the use of Raspbery Pi with the PiCAN module without RT enhancements in RT applications at all. It can be used without any other load and after application of patches at a frequency of less than 100 Hz in non-RT applications.

As a replacement, we recommend another single-board computer with GNU/Linux and integrated CAN interface. As part of this work, a SoC Zynq board was tested for which the CAN latencies of the high-loaded system did not exceed 120 microseconds. The board was tested with CTU ert and PREEMP-RT patched Linux kernel.

References

[1] Radek Mečiar. *Řízení motorů s deskou Raspberry Pi a Linuxem*. 2014.
http://support.dce.felk.cvut.cz/mediawiki/images/1/10/Bp_2014_meciar_radek.pdf.

[2] Prudek Martin. *Řízení bezkartáčových motorů s deskou Raspberry Pi a Linuxem*. 2015.
https://support.dce.felk.cvut.cz/mediawiki/images/d/da/Bp_2015_prudek_martin.pdf.

[3] ARM Holdings plc. *ARM1176JZF-S Technical Reference Manual*. 2009.
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0301h/DDI0301H_arm1176jzfs_r0p7_trm.pdf.

[4] ARM Holdings plc. *Cortex-A7 Floating-Point Unit*. 2012.
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0463d/DDI0463D_cortex_a7_fpu_r0p3_trm.pdf.

[5] J.M. Flores-Arias M. Ortiz-Lopez F.J. Quiles-Latorre V. Pallares A.Chen. *Complete Hardware a Software Bench for the CAN Bus*. 2016.
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7430584&isnumber=7430494.

[6] Robert Bosch GmbH. *CAN Specification*. 1991.
http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can2spec.pdf.

[7] Qiangsheng Ye. *Research and application of CAN and LIN bus in automobile Network System*. 2010.
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5579409&isnumber=5579162.

[8] Peter Fellmeth. *ISO11783 a Standardized Tractor – Implement Interface*. 2003.
https://www.can-cia.org/fileadmin/resources/documents/proceedings/2003_fellmeth.pdf.

[9] D. Saranyaraj, R. Nandhakishore, and P. Venkatesh. *Benchmarking and analysis of can transmision on real-time environment.* In: *2015 2nd International Conference on Electronics and Communication Systems (ICECS)*. 2015. 399-404.
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7124934&isnumber=7124722.

[10] *Readme file for the Controller Area Network Protocol Family (aka SocketCAN)*. 2017.
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/networking/can.txt.

[11] Ing. Pavel Píša Ph.D. *Linux/RT-Linux CAN Driver*. 2005.
http://cmp.felk.cvut.cz/~pisa/can/doc/lincandoc-0.3.pdf.

[12] M. Sojka P. Píša M. Petera O. Špinka Z. Hanzálek. *A comparison of Linux CAN drivers and their applications*. 2010.
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5551367&isnumber=5551360.

[13] SK Pang Electronics Ltd. *PiCAN 2 USER GUIDE V1.2*. 2016.
http://skpang.co.uk/catalog/images/raspberrypi/pi_2/PICAN2UG12.pdf.

[14] M. Sojka, P. Píša, O. Špinka, and Z. Hanzálek. *Measurement automation and result processing in timing analysis of a Linux-based CAN-to-CAN gateway.* In: *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems*. 2011. 963-968.
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6072917&isnumber=6072809.

[15] Martin Jeřábek. *FPGA Based CAN Bus Channels Mutual Latency Tester and Evaluation*. 2016.
https://rtime.felk.cvut.cz/can/F3-BP-2016-Jerabek-Martin-Jerabek-thesis-2016.pdf.

[16] Ing. Pavel Píša Ph.D. *GNU/Linux, rychlost odezvy a výuka řízení*. 2015.

[17] Paul E. McKenney. *A realtime preemption overview*. 2005.
http://lwn.net/Articles/146861/.

[18] Luotao Fu Robert Schwebel. *RT PREEMPT HOWTO*. 2017.
https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO.

[19] Carlos Jenkins Michal Sojka. *Code generation for automotive rapid prototyping platform using Matlab/Simulink*. 2014.
https://rtime.felk.cvut.cz/rpp-tms570/rpp_simulink.pdf.

[20] The Math Woks Inc. *Target Language Compiler*. 2000.
http://radio.feld.cvut.cz/matlab/pdf_doc/rtw/targetlanguagecompiler.pdf.

[21] M. Sojka, P. Píša, and Z. Hanzálek. *Performance evaluation of Linux CAN-related system calls.* In: *2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014)*. 2014. 1-8.
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6837608&isnumber=6837570.

Appendix A
Abbreviations

AC	Alternating current
API	Application Programming Interface
CAN	Controller Area Network
CPU	Central processing unit
CSMA	Carrier-Sense Multiple Access
CTU ert	ert-linux file, developped at CTU in Prague
DC	Direct current
FPGA	Field Programable Gate Array
FPU	Floating-Point Unit
HTTP	Hypertext Transfer Protocol
HW	Hardware
IRC	Incremental Rotary Encoder
MCU	Microcontroller
MTD	Memory Technology Device
MTW ert	Default ert file provided by MathWorks
OS	Opperating System
PMSM	Permanent Magnet Synchronous Motor
RAM	Random Access Memory
RT	Real Time
SATA	Serial ATA
SMP	Symmetric Multiprocessing
SoC	System on a chip
SSH	Secure Shell
TLC	Target Language Compiler
USB	Universal Serial Bus
VFP	Vector Floating-Point