CODEN:LUTEDX/(TEIE-5271)/1-183(2010)

Industrial Electrical Engineering and Automation

Motionlogger 2

Software Implementation of Data Logging at High Frequencies

Jens Svensson

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

Motionlogger 2

– Software implementation of data logging at high frequencies

Master Thesis by:

Jens Svensson

For:

Industriprojektbyrån i Sverige AB

Tetra Pak AB

Industrial Electrical Engineering
and Automation IEA

Lund 10 August 2010

Preface

To get most out of this report interest and knowledge about Java, C and HTML programming is recommended.

The thesis work started at the summer of 2009. The work was carried out at Industriprojektbyrån i Sverige AB's office at Ideon Science Park in Lund. Parties involved in the thesis were besides Industriprojektbyrån, Tetra Pak AB and the Department of Electrical Engineering and Automation.

I would like express my thanks to Industriprojektbyrån for the support during the thesis and the opportunity to work on a different project for a month during the autumn of 2009.

The following persons deserve particular mentioning:

Daniel Biloglav, Industriprojektbyrån, for general supervision of the thesis and programming support.

Maja Arvehammar, Industriprojektbyrån for help with introduction to the Open Process Logger and support with general programming details.

Mattias Wallinius, Industriprojektbyrån, for input regarding all those complicated things.

Sven Gestegård Robertz, Department of Computer Science, for advice regarding real time programming and systems.

Bengt Ask, Tetra Pak AB for good communications and input from the customer's point of view.

Gunnar Lindstedt and Ulf Jeppsson, Department of Measurement Technology and Industrial Electrical Engineering for supervision and general advices.

Lund, 10 August 2010

Jens Svensson

Abstract

In today's industry the need for fast logging of measurements is increasing. Existing systems used by Tetra Pak AB have a maximum logging frequency which is inadequate. As a result of this Tetra Pak gave Industriprojektbyrån AB the task of developing a measuring system for this purpose. Industriprojektbyrån AB has before the start of this thesis developed the hardware set up for the system. The purpose for this thesis is to further develop the system into a product that can be delivered to Tetra Pak.

The first part of the thesis was to configure the operating system (OS). A small evaluation of possible OS's was made. The thesis deals with how to configure the OS to get adequate performance. The most important information is about how to configure a Linux kernel with real time support. A system performance test was performed to verify the deterministic behavior and logging frequency for the system.

The second part is the software design and implementation. The design of the software for the system is based on an open source project developed at Industriprojektbyrån. The general idea was to use Java Native Interface which made it possible to develop the actual logging in C. The remaining parts of the system in Java and other high level languages. The presentation part of the system is constructed as a webpage to make it possible to view data.

The thesis describes how to implement software that satisfies the specification, how to test the system and implement these tests. Of course corresponding result from the tests are presented. A section dealing with suggestions for further development is also included.

Abbreviations

A/D – Analog to Digital
CPCI – CompactPCI
HMI – Human Machine Interface
I/O – Input/Output
JNI – Java Native Interface
JRE – Java Runtime Environment
JVM – Java Virtual Machine
OPL – Open Process Logger
OS – Operating System
PLC – Programmable Logic Controller
RAM – Random Access Memory
ROM – Read Only Memory
RT – Real Time
SMI – System Management Interrupts

Table of Contents

1 Introduction

1.1 Background

In today's industry the need for fast logging of measurements from encoder sensors is increasing. Encoding sensors are primarily used for measuring position of rotating axes. Tetra Pak uses an existing system for logging measurements from incremental encoders that has a maximum logging frequency around 2 kHz which is not enough today. According to Anders Sundberg the frequency of the phenomena's that the system is designed to detect is typically up to 3-400 Hz. For future use it also could be interesting to see the output from the control loop which can go up to 2 kHz. The system is briefly described in Section 1.2. The performance for this system is not enough to fulfill the requirements from Tetra Pak. No existing commercial product which accommodates the required performance could be found.

As a result of this, Tetra Pak gave Industriprojektbyrån AB the task of developing a system with logging frequency of at least 50 kHz. Industriprojektbyrån has at the start of the thesis constructed a CPCI based system as a base for the system. This master thesis will describe the further development of the system as well as the final system.

1.2 Existing system

Tetra Pak AB already has a measuring system for logging data from encoder sensors but this system has limitations especially when it comes to logging frequency and logging time.

1.2.1 Description

The existing system works by using a gate array to store the data. The actual logging frequency for the system is only 2 kHz and the system then uses interpolation to simulate a higher logging frequency of 20 kHz. The system is not a standalone system which means that you need to connect it to a computer to start the logging. Logging up to four channels simultaneously is possible. The system can log up to twelve seconds for one channel and four seconds when logging four channels. When logging is complete the data is exported to Excel for presentation and analysis.

1.2.2 Limitations

The limitations of the existing system are fairly obvious when it comes to the logging frequency itself. When the frequency for the phenomena's comes close to the logging frequency of the system the risk for generating incorrect data becomes high. There is also a risk for the interpolation to work as a lowpass filter which filters out the target phenomenon.

Furthermore the limitation of the logging time is also a problem. Four seconds might not be a sufficient logging time. The system is also limited when it comes to presenting and analyzing data due to the fact that it relies on an external program for this. Data saving is also a limitation in the system. Data has to be saved externally.

1.3 Specification

The uses and specifications for the application are largely defined by the initial use case analysis written by Mattias Wallinius. The initial document is shown in Appendix A. The input from the customer Tetra Pak AB represented by Bengt Ask and Anders Sundberg are included.

The use case analysis is specification from the user's point of view. The use case analysis specifies all functionality and options that should be possible for the user.

The system should be able to collect data from three types of sensors: encoder, analog and digital. The data logging from the encoder sensors is the primary use for the system. The logging frequency for encoder and digital sensors should be at least 50 kHz. In the initial specification the logging frequency was set to 20 kHz but was changed to 50 kHz due to demands from Tetra Pak. The data logging from the analog sensor could be done at lower frequency. The data logging should be deterministic. The system should handle logging time from 0.5 s up to 30 s. It should be possible to log data from multiple sensors for the encoder and analog sensors.

There should be two ways to start the logging: either via a start button in HMI or via an external digital trigger. The digital trigger should be specified from HMI.

The post processing of the data should include options to derive and lowpass filter the data. It should also be possible to store the data with name, information and timestamp.

The logging data should be presented in graphical form in the HMI.

1.4 Objective

The general objective of this thesis is to develop a fully functional and tested system according to the specifications in 1.3. To achieve the objective the thesis can partitioned into four parts: system configuration, data logging, HMI and system testing.

The system configuration part involves analyzing possible OS that can be used for the system, configuring the OS and setting up the necessary software for development. The system needs to be set up for real time support.

The data logging part is the implementation of the logging algorithm is to be done. The data logging should use JNI to make it possible to handle sampled data in Java.

The HMI should be built as a web based application using Java, JavaScript and HTML. The HMI should use the Open Process Logger framework as base. The HMI should be able to present the data from the loggings. Logic for storing data should also be included.

The final part is testing where the functionality and performance of the system is tested. The performance tests are to determine the logging frequencies and determinism for the data logging.

2 Hardware description

Here follows a description of the hardware setup and the specifications of the components of the Motionlogger 2.

The system is built on a CPCI[1] system. CPCI is a standard for industrial computers where all boards are connected via a passive backplane. The system setup is show is Figure 1.

Figure 1. Industrial PC box for Motionlogger 2. To the left is the connecter plate for the encoder sensor. Next to the connector plate is the power supply. Located next the power supply is connectors for TIP logging cards. And to the right is the computer itself.

TEWS TIP116[2](encoder) is a measurement card for reading encoders. The card can count up to four channels at the same time. Each channel has a 32 bit programmable counter, a 32bit preload register, a 32 bit compare register and a 32 bit output latch. The counter can be programmed in a number of different modes. Most important ones are quadrature, up-/down and direction count. Quadrature mode is a special mode of encoding which gives both relative angular position and direction with high resolution from rotary encoders. In quadrature mode the counter can be programmed for analysis of the encoder signal in single, double or quadruple mode. The signal level of the channels can be RS422 or TTL. Before the signal is fed into the counter it is passed through a digital filter to suppress noise. The card also provides a general purpose I/O according to the RS422 standard. This can be programmed to act as a clock output with 1, 2, 4 or 8 MHz as frequencies. It can also be used as a trigger for the sampling. Another useful function is a 16 bit down-counter that acts as an interval timer.

[1] Open Modular computing specification CompactPCI, http://www.picmg.org/v2internal/compactpci.htm (2010-02-11)
[2] (2009) TEWS TIP116 datasheet http://www.tews.com/Products/Datasheets/TIP/TIP116,property=PdfFile.pdf (2010-02-11)

TEWS TIP501[3](analog) is a measurement card for analog to digital conversion. The card has sixteen single-ended, or eight differential channels with 16 bits A/D conversion. The data acquisition and conversion time is below 12 µs without channel / gain change and up to 14.5 µs with channel / gain change. The input range is 0-10V.

TEWS TIP600[4](digital) is a digital measurement card. The card has 16 digital input channels. The channels are galvanically isolated with optocouplers. Furthermore are all channels isolated from each other. Input voltage is 24V.

For the computer a Kontron CP306[5] is used. The processor is a Pentium M 1.1 GHz. The amount of RAM is 512 MB.

The setup uses two different hard drives. First is a small 3 GB flash drive that contains the OS. The other is 100 GB magnetic drive with two partitions. The idea behind two partitions is that one smaller that will be the home directory for the OS and the other will be used for database storage of the logging tasks. The sizes are 20 GB for the home partition and 80 GB for the storage part.

[3] TEWS TIP501 datasheet http://www.tews.com/Products/Datasheets/TIP/TIP501,property=PdfFile.pdf (2010-02-11)
[4] TEWS TIP600 datasheet http://www.tews.com/Products/Datasheets/TIP/TIP501,property=PdfFile.pdf (2010-02-11)
[5] Kontron CP306
http://emea.kontron.com/products/boards+and+mezzanines/3u+compactpci/x86+processor/cp306.html (2010-02-11)

3 Operating System

It turns out that configuring the operating system is a major part of the thesis. This was not intended in the original specification. There are a lot of demands on the configuration of the operating system when dealing with RT-applications. The operating system used for the system is Xubuntu with a Linux core. The reason for choosing Xubuntu is that the performance of the computer is very limited. A complete manual for how to install and configure the OS is located in Appendix B.

3.1 Demands

Since the application is a RT-applications there are a number of demands for the OS. The need for high resolution timers is obvious. With a logging frequency of 50 kHz we get a period of 20 µs. If we have higher frequency for example 160 kHz (there is a reason for this number, see chapter 4) we get a period just below 7 µs. To get a reasonable precision and acceptable jitter we need a timer resolution of at most 1 µs and a value below that would be desirable. One thing to point out is that it is desirable that the hardware supports high resolution timers. Since we want to have all our data deterministic we need access to a deterministic RT scheduler. Drivers for the cards must be available for the operating system since writing drivers is a major, non trivial and time consuming task. Since the performance of the computer is quite limited we need an operating system that is light weight. A light weight OS means that the OS itself is optimized for running on computers with low resources. No budget for buying a commercial operating system is available so an open source operating systems needs to be used.

3.2 Choosing operating system

Drivers for the cards are supported for VxWorks, Windows XP/XPE/2000, LynxOS and Linux. However, Windows does not have RT support and LynxOS and VxWorks are commercial products. That leaves a Linux based OS.

There are a lot of Linux distributions that are light weight but many of them are unstable and have limited application support. The demands for the OS make the Xubuntu distribution the obvious choice. Xubuntu is a lightweight version of Ubuntu. Some versions of Xubuntu also comes with a precompiled kernel for RT-support.

Which distribution of Xubuntu to use also needs to be decided. At the time when the version was to be decided, the latest release of Xubuntu was 9.04 Jaunty Jackalope. This distribution however has two major disadvantages concerning the kernel that comes with the distribution. The first is that the file interface has been changed in the kernels above 2.6.24[6]. and the TEWS drivers have not been updated to support kernels newer than 2.6.24. There is however a solution for this since the code sections in the driver installation is implemented in separate files which TEWS has updated to support the never kernels.
The other big issue is a bug in the KMOD (Kernel modules) that shows up when modprobe is used to build the drivers into the kernel. Modprobe is a program for building the drivers into the kernel. This bug will likely be fixed in future updates of the kernel but at the time of writing it has not been done. More about this bug can be found in section 3.6.1. In this situation there are two options. The options are either to compile an older kernel for the distribution or to change distribution. The option to use an older kernel is not a very appealing one, thus risk for compromising functionalities in the OS desktop environment is apparent and

[6] Mail conversation Gerrit Hesse, TEWS technologies 2009-10-17

configuring an compiling a kernel from scratch is a very time consuming task. So that leaves the option of changing to an older distribution of the OS.

The distribution before 9.04 is 8.10 Intrepid Ibex but this version does not have a precompiled kernel with RT support and standard kernel is newer than 2.6.24. For those reason this one is not a good choice either.

Before version 8.10 a long term support distribution was released. The distribution name is 8.04 LTS Hardy Heron. It comes at time for installation with kernel version 2.6.24. So this one is the obvious choice and that it is a LTS version is a nice bonus. It will be supported until April 2011. LTS stands for long term support.

3.3 Hard drives

The system has as stated in chapter 2 two hard drives, one small flash drive and one magnetic drive. The small flash drive contains the OS. The magnetic drive has two partitions that are intended for storage respectively home directory. Setting up the OS to use the hard drives in this way is not a trivial thing. The trickiest part is moving the home partition. Moving the home partition is necessary since the small flash does not have enough space for all the working directories in the home directory. There are a lot of things that needs to be addressed when doing this. The moving involves special copying procedures to get all links and nested folders correct. It is also necessary to update all permissions for the folders and OS. A detailed manual for how to do this is located in Appendix B.

3.4 Configure kernel

Kernel configuration for RT support is not a trivial thing and demands a lot of knowledge about the Linux kernel. It involves patching and recompiling the kernel. For this application the standard RT-kernel for Xubuntu 8.04 LTS with a few tweaks is sufficient. The specific kernel version is 2.6.24-24-rt.

3.4.1 RT-patch

The Linux kernel in itself is not deterministic but there is a way to make the Linux kernel processes deterministic. By patching the kernel with Ingo Molnar's Linux Real Time Patch this can be achieved. This patch will allow running of RT-application on the OS. For this thesis RT and deterministic should be considered the same.

3.4.2 Options

First of all it is essential to know which options to use and disable. The most important group of options to deal with is SMI's. SMI stand for System Management Interrupts. The cause of these interrupts is power management hardware on the motherboard. They are the highest priority interrupts in the system. In Linux the priority range is 1-140 but priorities over 99 is reserved for external interrupts such as SMI's. That means that there is no way to preempt the interrupts from inside the kernel. SMI's are really bad when RT is required. The SMI's can cause latencies of several 100's[7] of µs. As we have a cyclic period under 7 µs that is a really big problem. The way to deal with these interrupts is to disable them when compiling the kernel.

[7] RT_PREEMPT_HOWTO, RT Linux wiki http://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO (2010-02-28)

There are two types of SMI's: ACPI and APM. Not all of these are bad even thought they cause latencies. Especially APM options are often essential for dealing with power management on hardware level. Disabling these can lead to severe risk of burning the computer to death. The ACPI options however are mostly non safety critical interrupts. All of these options except the pm_timer should be turned off. The pm_timer is needed as a clock source for the high resolution timers.

Other options that need to be enabled are CONFIG_PREEMPT_RT and High Resolution Timer Support. The first one is the option for enabling the real time support of the kernel. The other is for activating the high resolution timer support. However this will not make any difference if the hardware does not support high resolution timers and the number of platforms that support this is limited.

3.4.3 Compiling the kernel

Now comes the part where all the knowledge and research comes to use. The kernel compilation is done according to the manual located in Appendix B. The manual is based on the Master Kernel Thread[8] in the Ubuntu forums.

First thing is to download the source code for the new kernel. There are several ways to do this but downloading the source code from the GIT repository[9] is the way to go if the desired kernel is an Ubuntu version. If vanilla kernel is needed downloading it from www.kernel.org[10] is the easiest. Vanilla means an unaltered standard release of the kernel.

Before the compilation all kernel options need to be set. It is not realistic to set all of these options individually because there are several hundreds of them. And many of them require very deep knowledge about a computers system. The way to go here is to copy the standard configuration from the current kernel and then modify the copy. During this thesis the copied configuration is from the Xubuntu Linux RT-kernel which can be installed from the Xubuntu repository.

Next, to configure the kernel use th xconfig which is a Linux graphical compilation utility. Alternatively menuconfig could be used for a terminal based version. A screen shot of the xconfig menu is shown in Figure 2.

[8] Master Kernel Thread, Ubuntu Forums http://ubuntuforums.org/showthread.php?t=311158 (2010-02-12)
[9] Ubuntu Kernel Team's Git Repositories. http://kernel.ubuntu.com/git (2010-02-15)
[10] The Linux Kernel Archives www.kernel.org (2010-04-20)

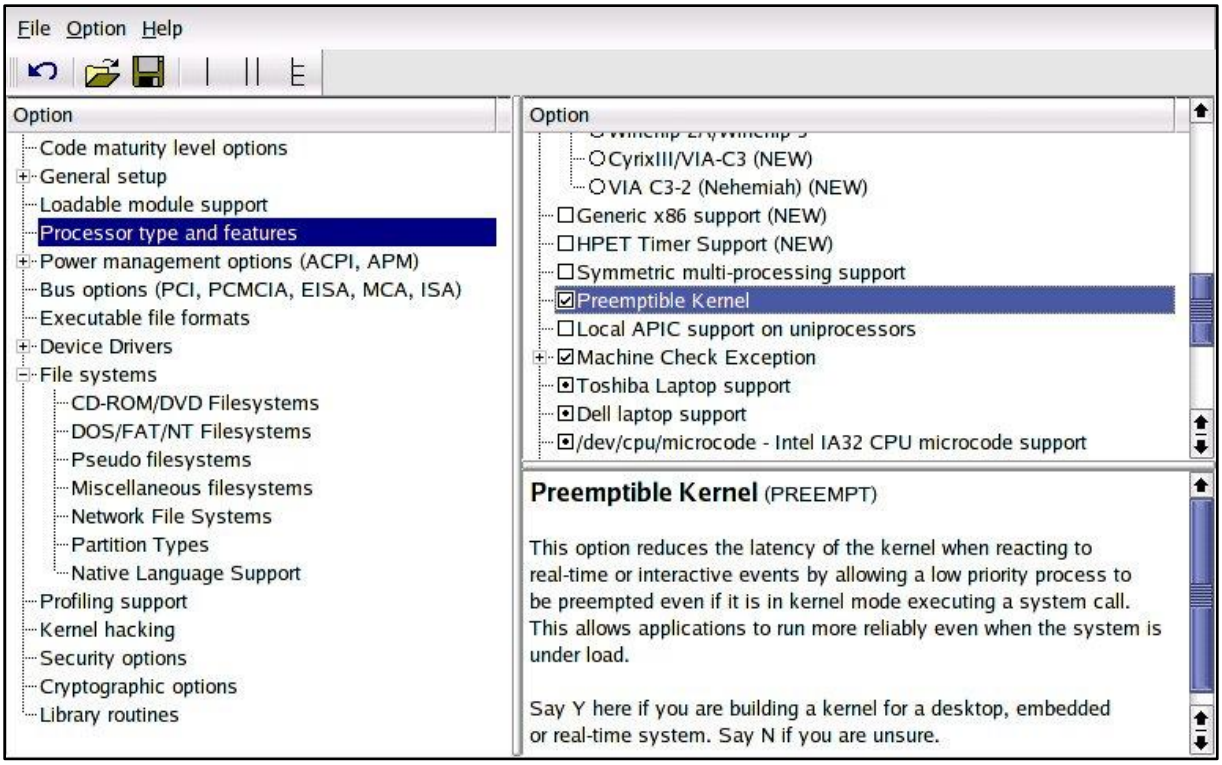

Figure 2. Screenshot of xconfig menu.

Most of the options that need to be modified is located under Power management options and make the configurations according to 3.4.2.

The compilation of the new kernel will take several hours depending on the speed of the CPU. In this case, because of the limited CPU speed, it will take about four and a half hour.

The compiler creates two files: one linux-image-<versionNumber>.deb and one linux-headers-<versionNumber>.deb. These are files that contain the new kernel and can be installed into the OS. When the kernel is installed, it will be loaded on the next boot up.

3.5 Necessary applications

To be able to develop the system there are some necessary applications and drivers that need to be installed. This section is a summary of these. The exact details for the installations are shown in Appendix B.

Eclipse

Eclipse[11] is used as the development environment for the code development. Two versions with different configurations were used.

The first one was Eclipse Classic version 3.5. Two plug-ins were used, CDT[12] and JUnit[13]. CDT is the Eclipse part for C and C++ developers and stands for C/C++ Development Tool. JUnit is a unit testing framework for Java. JUnit is already installed as standard in this version of Eclipse. This configuration is used for the development of C- parts of the application.

[11] Eclipse Development Tools http://www.eclipse.org/ (2010-02-15)
[12] Eclipse C/C++ Development Tool http://www.eclipse.org/cdt/ (2010-02-15)
[13] JUnit community site http://www.junit.org/ (2010-02-15)

The second version was Eclipse JEE 3.5.1. JEE is customized version for web development. The CDT plugin was also used in this configuration. This version is used when developing the Java part of the system.

Java runtime environment

To execute Java code in Linux we need a Java runtime environment. The installation here is very simple. Just go to Add/remove program under the system menu and search for Java runtime and then mark the one you want to install. However, there are two different choices, either OpenJDK Java Runtime or Sun Java 6 Runtime. The best choice here is the Sun Java 6 Runtime simply because later in the application Java Native Interface will be used and a quick Google search resulted in a lot of problems when using OpenJDK. However both could be installed without any problems but the JAVA_HOME variable needs to point to the chosen installation.

Apache Tomcat

Apache Tomcat[14] is an open source servlet container. Apache Tomcat provides a web server for Java applications. The exact version used for this application is v6.0.20. Before installation of Apache Tomcat Java runtime environment needs to be installed. The final application will run under Apache Tomcat.

Apache Derby

Apache Derby[15] is an open source relations database. Apache Derby is implemented entirely in Java. The database is needed for storing and easy access to old logging task. Specific version used is 10.5.3.0. Since there will be no direct speed demands for the database the most suitable place for the installation of the database is the storage partition. Apache Derby is used for storing the data from the loggings.

GCC

GCC[16] is the GNU Compiler Collection. GCC has compilers for C/C++, Objective-C among others. It also contains the standard libraries for these languages. As this application uses JNI the C compiler is needed.

3.6 TEWS drivers

The installation mostly follows the user manuals from TEWS technologies for the corresponding cards. The manuals can be downloaded from TEWS's homepage[17]. However the manuals are not entirely correct and some corrections and tweaking is necessary. One more important issue is that before installing the individual card, an IPAC carrier drive needs to be installed. For exact details and commands for the installation see the Appendix B.

3.6.1 Kernel version issue

According to the manual and the installation files the cards will work for all 2.6.* Linux kernels. This is not true. After the 2.6.24 distribution some kernel systems were updated and rebuilt. The functions in specific are the device file system interface and the modprobe. The device file system interface structure has been changed. According to a mail conversation[18] with Gerrit Hesse from TEWS Support Management TEWS have not updated the drivers to

[14] Apache Tomcat http://tomcat.apache.org/ (2010-02-15)
[15] Apache Derby http://db.apache.org/derby/ (2010-02-17)
[16] GNU Compiler Collection http://gcc.gnu.org/ (2010-02-17)
[17] TEWS TECHNOLOGIES http://www.tews.com/ (2010-04-17)
[18] Mail conversation Gerrit Hesse TEWS support Management (2009-08-17)

support the new structure. However, the installation files handling this are fairly separate and TEWS have made updates of these files supporting the new file structure in other projects. However, they are untested for the TIP116(encoder), TIP501(analog) , TIP600(digital) cards. The files are named config.h, tpmodule.h and tpmodule.c. They appear to solve the problem but when the modprobe function is used to build the drivers into the kernel, the modprobe issues a warning and the installation fails.

```
WARNING: All config files need .conf: /etc/modprobe.d/oss-compat, it will
be ignored in a future release.
```

Some investigations about this error gives that it is a bug in KMOD which modprobe uses. The bug will be fixed in future releases of the kernels but at the time of the installation it is not so, which leaves the only viable option to use an older kernel.

3.6.2 CARRIER

Since the TIP cards are connected to an IPAC carrier board the carrier drives need to be installed before any installation of the card drivers. Before anything is done the file ipac_carrier.h needs to be put in appropriate directories to make installation possible. Here the user manual is incorrect when it comes to which directories to put it in. The correct directories are /usr/include/ and /usr/src/linux-headers-2.6.24-24-rt/include.

The rest of the installation is straight forward: use the make command to build the driver, build kernel dependencies and finally build the driver into the kernel.

3.6.3 Card drivers

Installation of the TIP card drivers is similar to that of the carrier. The only major difference is that a device node needs to be created for each card on the file system. The makenode script which is used for this creates a minor device for each card and the cards can be accessed via device nodes /dev/tip116_0, /dev/tip501_0, /dev/tip600_0. Also the header files for each card need to be put in the directory /usr/include. The header files are used for application access to the card. The user manuals do not mention this step.

4 System performance tests

Before the actual implementation a series of test needs to be performed in order determine the performance for the system. The main objective for these tests is to determine overall performance in aspects of the logging frequencies and determinism. Also important for the tests is to indicate which type of logging algorithm to use in the actual implementation.
In this section all tests and test results are presented. For the exact implementation of the test files see Appendix C. The test results and conclusions are summarized in 4.2.

4.1 Performance tests

Find boards test
This test is implemented in the file testFindBoards.c. It is a simple program that finds and prints the names of the connected boards.

System performance test
Test program that is implemented in the file systemMaxPrefTest.c. Does exactly what the name suggests, reads and stores data value and time stamp as fast as possible for a fixed number of samples. Calculates time differences between the reads and prints these to a console. Also prints the number of worst time intervals in clock ticks and µs. Prints the average number of clock ticks between reads as well as the corresponding time in µs. Prints the number of intervals that are severely over the average interval time and finally the total time for the test. The purpose of this test is to determine the maximum logging frequency and investigate how deterministic the logging is. Another purpose for this program is to determine empirically how different priorities and scheduling policies affect the determinism of the logging. This test is the most important of all.

Self scheduling cyclic algorithm test
Test program that is implemented in the file PrototypeFunc.c. The program uses a self scheduling cyclic algorithm for logging of one encoder channel. It makes it possible to control and vary the logging frequency. The program prints start and stop times for each cycle as well as the data value. The purpose for the test is to investigate the possibility to use variable logging frequency.

Timer max performance test
Test program that is implemented in the file timerReadTest.c. The program does the same thing as the system max performance test but without reading and storing the data. The purpose for this test is to determine how long the timer read takes.

All counter channel max performance test
Test program that is implemented in the file testRunAllCounters.c. The program that does the same thing as the system max performance but for all four encoder channels instead of just one. The output from this program is the same as for the system max performance test. Purpose for this test is to determine the maximum logging frequency for logging on all four channels.

Analog channel max performance
Test program that is implemented in the file testRunAnalogRead.c. The program that does the same things as the system max performance test but for analog read instead of encoder. The purpose for this test is to determine max logging frequency for logging on one analog channel.

Two analog channel max performance test

Test program that is implemented in the file testRunAnalogTwoChannels.c. The program that is the same as the analog channel performance test but for logging on two analog channels instead of one. The purpose for this test is to determine the maximum logging frequency.

Digital read performance

Test program that is implemented in the file TestRundigitalCounter.c. The program that runs digital logging as fast as possible. The program prints the same information as the system performance test. The purpose of the program is to determine the maximum logging frequency for digital logging.

Running the test files

The test files have to be run from the command line in a console. The reason for this is that the files need access to change priorities and the scheduling scheme. These are restricted for normal programs and users. To be able to get around these restrictions the sudo command has to be used.

```
sudo ./motionloggerIo
```

Here it is assumed that the files are compiled and linked by eclipse into an executable that is named motionloggerIo.

4.2 System performance test results

The first and most important results of the tests are the maximum logging frequency for each different logging function.

Logging frequencies

The summary of the results from the logging frequencies follows. The logging frequencies are averages and includes timer read for each data read.

Logging type:	Frequency:	Time interval:
One encoder channel	127 kHz	7.9 µs
All encoder channels	43 kHz	23.3 µs
One analog channel	48 kHz	20.8 µs
Two analog channels	15.5 kHz	64.5 µs
Digital	160 kHz	6.3 µs
Timer Read	-	1.9 µs

As shows above the logging frequencies vary a lot between the different types of logging. The overall result satisfies the specification more than well. The only test below the specification is the four channel encoder. This is however close enough to the specified 50 kHz and was approved by Tetra Pak. The analog logging is not required to be at 50 kHz. The 1.9 µs for the timer read is very disappointing but since there are no other way to get a faster clock it has to be approved.

Determinism

The test results regarding the determinism of the logging are not as satisfying as the logging frequencies. The first problem is that during 0.75 % of the time intervals between the reads in the system performance test are above 12 µs when they are supposed to be 7.9 µs. Next problem is that the longest time interval between reads is about 40 µs and varies from logging to logging. This suggests that there are interrupts that is not preempted by the program. To analyze this further a print out (to a console) of a larger number of time intervals (>2000) were made. From that printout it appears that there probably are two types of interrupts that is not preempted and both show up at random intervals. The longer time interval does not appear

very often i.e. about 3-4 times during a 30s read while the other with shorter interval appears more frequent but also more or less randomly. To try and improve the determinism another kernel was compiled. This kernel had all but the potentially damaging SMIs disabled. Even with the more stripped kernel the behavior was the same.

Scheduling

These tests were just to determine which scheduling policy gave the best result. There are two possible scheduling rules, FIFO and Round Robin. These are the only supported in the C-libraries. The overall performance of the two are very similar. They only differ with regards to one thing. The Round Robin gives a much shorter worst case interval. The worst interval with Round Robin is around 27 µs, compared to 40 µs for FIFO. That is a significant improvement of the worst case behavior.

Priority

The testing with different priorities gave some interesting results where the most interesting was how the determinism is affected by the priority. At high RT priorities above 70 the behavior was constant. This leads to the conclusion that no process with higher priority were running in the configuration. With priorities from 51-69 determinism got worse and below 50 the behavior was almost as running without a fixed priority. This is explained by the fact that the normal kernel processes run at priority 50 and that there are probably some processes that runs with priorities in the 51-69 range.

Self scheduling periodic task

After some research and experimenting with different implementations of the cyclic task the only theoretically viable implementation of it looks as follows.

```
CurrentTime(t);
LOOP
            Logg and store values; // periodic task
            t = t + h; // is the time
            Wait until t.
END;
```

The tests were done at logging frequency of 50 kHz for one encoder channel. At first the test looked good with almost no deadlines missed. However, when the timestamps for the reads were stored and the interval between the read were printed out a major problem was discovered. Severe jitters in the time intervals were discovered. The jitter was almost 2 µs for a period time of 20 µs. The cause for this is the external clock read. Even though the external clock has a much higher resolution the effective resolution is limited by the time it takes to read the timer. Another problem with this implementation is that when a deadline is missed the task tries to catch up for the lost time. That means that in case of a missed deadline a too long period is followed by a number of shorter periods. In the 50 kHz worst observed case a 45 µs period followed by an 8 µs period.

Test conclusions

The most important conclusion to be drawn from these test results is about what method to use when logging. Because of the problem with determinism and that the periodic task did not work, the way to go is to run the logging as fast as possible and store the timestamp for each read. The reason for storing the timestamps is that it will make mathematical operations on the data easier.

5 Software concept

This chapter is a description of the overall design for the software parts of the system. The description includes which classes and parts of the system that has been created. The design is based on and uses parts of the Open Process Logger (OPL) framework. OPL is an open source project developed at Industriprojektbyrån AB. A summary of the OPL is given in Section 5.2.

5.1 Overview

The best way to show the design of the system is a block diagram, see Figure 3.

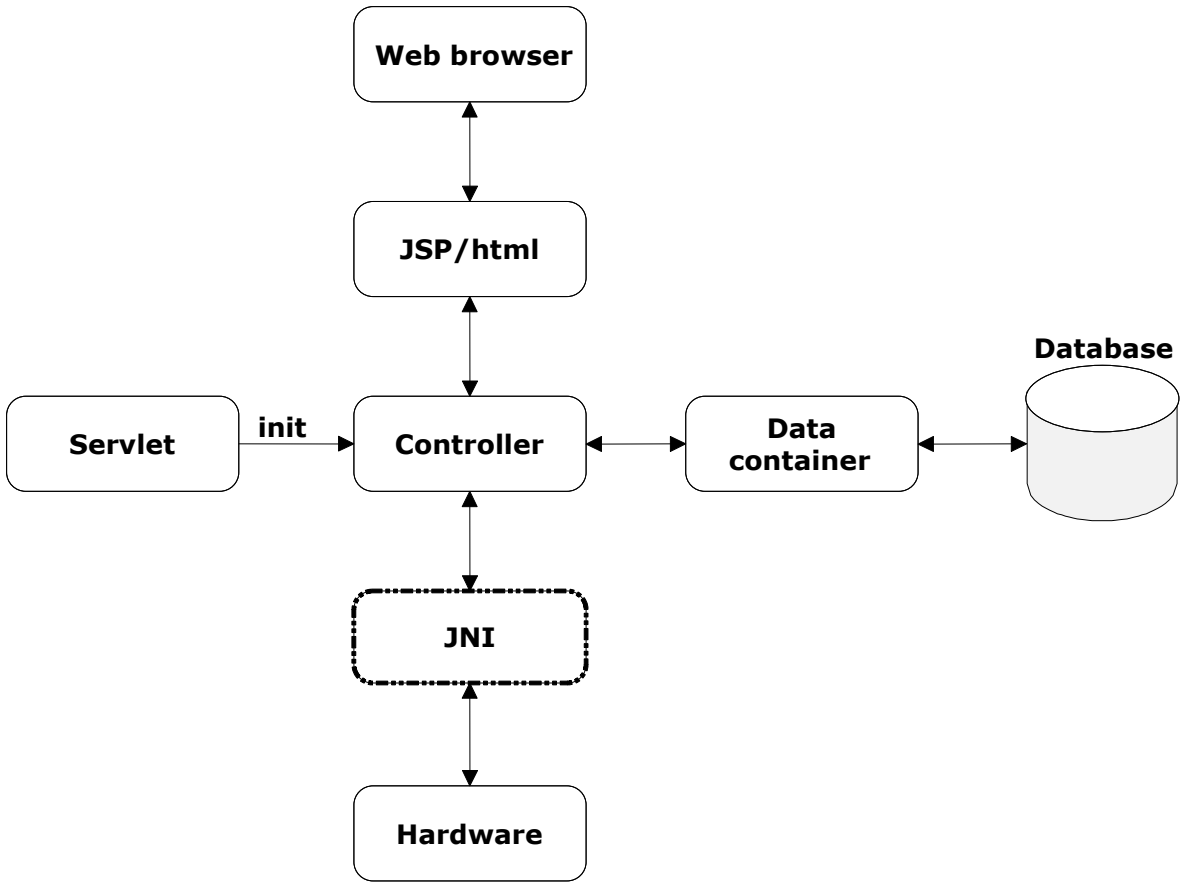

Figure 3. Block diagram for the system. Arrows represent communication paths.

First of all the system is designed as a Java servlet and will run under Apache Tomcat. The servlet will initialize the controller, which is the central communications hub. All information that needs to be passed from one part of the system to another will go through the controller. Designing the system in this way will make the code easier to read and shorten debugging time. More about the controller can be found in chapter 7.

The user interface will be constructed as a webpage using JSP and html. The webpage can naturally be accessed via a web browser. The browser in mind for the applications is Mozilla Firefox. The reason for using Firefox is that OPL is developed for this. The logging functions will be activated from the webpage and later the data is presented in graphical form. For data presentation the embedded graph viewer from OPL is used. More about the HMI can be found in chapter 9.

The communication with hardware will be done via a Java Native Interface (JNI). This is by far the largest and most complicated part of the software implementation. All logging algorithms and RT-logic are located here. Chapter 6 is devoted to the JNI part of the system.

When logging is complete the data will be kept in a special container class. Logics for post processing of the data will be done in this container. Database communication is also done from this class. Chapter 8 is devoted to the data container part.

5.2 Open Process Logger

The Open Process Logger is an open source project developed at Industriprojektbyrån AB. The Open Process Logger is a framework for collecting, storing and presenting data from PLC systems. The framework is developed as a web based application to run under Apache Tomcat. The framework can be downloaded from Google code (http://code.google.com/p/openprocesslogger/).

The framework is built in four layers. Layer one is the data acquisition layer. Layer two consists of the logic for storing the data. Layer three is the database layer. Layer four is the presentation layer.

Only layer two, three and four are used in this application.

6 Java Native Interface

This section will describe the JNI part of the application. For details about the JNI see reference [1] and the Java Native Interface Specification[19]. JNI is developed by Sun Microsystems. The C parts of the implementation dealing with the RT parts and communication with the TIP cards will be included here. Furthermore, a series of test files that test the performance of the JNI is described and important details of the test results presented.

6.1 JNI overview

Java Native Interface is a feature of the Java platform. JNI makes is possible to combine native code written in typically C and C++ with code written in Java. This makes JNI a very powerful tool. JNI is a two way interface that allows Java applications to invoke native code and vice versa.

The basic methodology when creating an application is to first write the Java class that declares the native method and then use the Java compiler to generate a header file for the native methods. Then write the native code and use the native compiler to create a native library.

6.2 Reason

The application has timing constraints and that means the RT part of the code needs to be implemented in a fast language. That in combination with the use of Java and Java scripts for major parts of the application makes it necessary to somehow call native methods for Java. JNI is a way to do just that. Using Java for the RT part is not an option since it is to slow and has RT-horrible "stop the world"- garbage collector. Another reason for using JNI is the limited RAM memory of the computer. Memory usage has to be closely monitored and controlled to get good performance and to do that a language with user implemented memory management is needed. As stated in the previous section the JNI has support for C and C++. To fulfill the demands of the system C is the obvious choice because it is faster than C++.

6.3 Java Native Interface classes

The Java class that contains the method declaration of the native methods is named TipJni.java and created as a package for easy access from other Java classes. The class has five public inner classes, one for each card type, one for an encoder channel and one for an analog channel. The analog channel is not used in the current implementation but prepared for further development. All the code will not be explained here since most of it is self explanatory but the important and essential functions are commented upon. The JNI Java class is shown in Appendix D.

A good practice when using JNI is not moving back and forward between Java and the native code more than absolutely necessary. This is because jumping back and forward tends to give complicated code which makes debugging very hard.

[19] Java Native Interface Specification http://java.sun.com/j2se/1.5.0/docs/guide/jni/spec/jniTOC.html (2010-02-22)

6.3.1 Main Class

The TipJni class has two main functions. The first is to hold the references to all the cards on the Java side. This is simply done by calling the native side and getting an array with available board names and then creating array lists for the cards.

The other function, which is the most important, is the declaration of the native methods. Declaration of the methods works in the same way as ordinary method declaration in Java but with a native modifier. The best way to show this is with an example. The example method for getting a string array with the connected boards is shown below.

```
private native String[] getBoards()
```

All native methods are also declared private to prevent from direct use. Instead the methods are accessed from public methods in the corresponding board or channel classes.

There is one more important thing that needs to be in the class, a static initializer that loads the native library that contains the native methods.

```
Static {
            System.loadLibrary("motionloggerIo")
}
```

6.3.2 TIP116 class

The class for the encoder board is named Tip116Board. This class holds all methods for calling the native methods regarding the encoder board. It also holds references to the encoder channels.

The TIP116 (encoder) class only contains calls to native methods regarding logging on all four channels. The calls for logging on a single channel are located in the encoder channel class which is more logical.

6.3.3 TIP501 class

The class for the analog board is named Tip501Board. This class holds all methods for calling the native methods regarding the analog board. It also holds references to the analog channels.

This class, however, needs more explanation since the analog card has more possible settings. The settings affect how many channels it is possible to access.

```
public static final int T501DIFF = 0x1;
```

- Final value for differential mode flag.

```
public static final int T501PIPELINE = 0x2;
```

- Final value for pipeline mode flag.

```
public static final int T501CORRECTION=0x4
```

- Final value for correction mode flag.

These three integers are just flags for different configurations. T501DIFF is a flag stating which mode the A/D converters is set in. The card will use differential inputs when this is enabled and single-ended inputs if omitted. This flag determines how many channels data can be read from. Eight channels is possible if differential mode is used and sixteen if single-ended mode is used.

T501PIPELINE indicates how to read the values. If activated then the data from the last conversion will be read and a new conversion is started. If the flag is omitted the data will be read from the new conversion.
T501CORRECTION is for use of an automatic offset and gain correction with factory settings.

These flags are used when setting the analog type variable. In the current implementation this is sent to the native side but not used there, instead the flags are set static in the C-code. They are, however, needed at the Java side to know how many analog channels to specify.

The runAllAnalog* methods only reads data from two i.e. channel 1 and 2. The reason for only reading from two channels is that reading from more than two slows down the logging frequency too much.

6.3.4 TIP600 class

The class for the encoder board is named Tip600Board. This class holds all methods for calling the native methods regarding the digital board.

6.3.5 Encoder channel class

The encoder channel class holds information and functions for calling native methods for the individual encoder channel. The class also has a system performance test function. This test functions is basically a function that test the maximum performance for the system and prints the result. This function is included in case the system performance needs to be verified later. It is unreachable from HMI in the current implementation.

6.3.6 Analog channel class

The analog channel class is not used in the current implementation. The reason for still having it is a preparation for further development. This class will grow significantly when the need for getting the actual voltage values from the analog readings occurs. Then a lot of calculations will have to be done here.

6.4 C-headers

When the native methods are declared it is time to generate the header files. The simplest way to generate these is to generate them in a terminal and then import them into the Eclipse project. It could also be done via some external tools in eclipse but to configure the external tool is quite troublesome.

The Java C header and stub file generator javah is used to for the actual generation of the files. To get javah to work the <jni.h> needs to be in the Java include path. Command for generating headers looks as follows.

```
javah -jni se.ipbyran.tews.tip.TipJni
```

This will create 6 files all with the typical C header extension: .h. The files are:

```
se_ipbyran_tews_tip_TipJni_AnalogChannel.h
se_ipbyran_tews_tip_TipJni_EncoderChannel.h
se_ipbyran_tews_tip_TipJni_TIP116Board.h
se_ipbyran_tews_tip_TipJni_TIP501Board.h
se_ipbyran_tews_tip_TipJni_TIP600Board.h
se_ipbyran_tews_tip_TipJni.h
```

All method stubs are located in se_ipbyran_tews_tip_TipJni.h. Method stubs are the generated function declaration for the native methods. The basic outline of the file looks as follows. For the complete file see Appendix E.

```
/* Header for class se_ipbyran_tews_tip_TipJni */

#ifndef _Included_se_ipbyran_tews_tip_TipJni
#define _Included_se_ipbyran_tews_tip_TipJni
#ifdef __cplusplus
extern "C" {
#endif

JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_getBoards
  (JNIEnv *, jobject);
            …
            …
            …
            …
            …
JNIEXPORT jboolean JNICALL
Java_se_ipbyran_tews_tip_TipJni_waitForAnyTransition
  (JNIEnv *, jclass, jint);

#ifdef __cplusplus
}
#endif
#endif
```

A couple of things need to be commented upon. First the “extern “C” “-line tells the Java compilers to look for external implementation of the methods inside.

The next important thing is the return types in between the macros JNIEXPORT and JNICALL (the macros just ensure that function is exported from the native library correctly). These are different from that of Java and C. This is because C and Java map types differently. That makes it necessary to introduce a new set of types that can be handled by both languages.

The argument for the methods is interesting. A JNI interface pointer (JNIEnv*) is passed as an argument for each native function. The function of the pointer is to allow interaction with the JNI environment inside the native function.

Note also the names of the functions. The names of the C function are created by concatenating “Java_” prefix, the class name and the method name.

6.5 C-implementation

This is the heart of the actual logging. Here is where all communication with the physical devices is implemented. The chapter describes the general structure of the methods and then goes into details and important sections. This should give a good overview and understanding of the implementation. The final part of the chapter is about testing the function in the implementation. All native functions are implemented in a single file for convenient access to the methods and global variables. The code is located in Appendix F.

6.5.1 General structure

All the read functions basically have the same outline. The function names for all read functions start with “run” after the auto generated prefix “Java_”-prefix and the class name.

The systemPerformanceTest function is also of this type. The principle is best shown by a block flow diagram, see Figure 4.

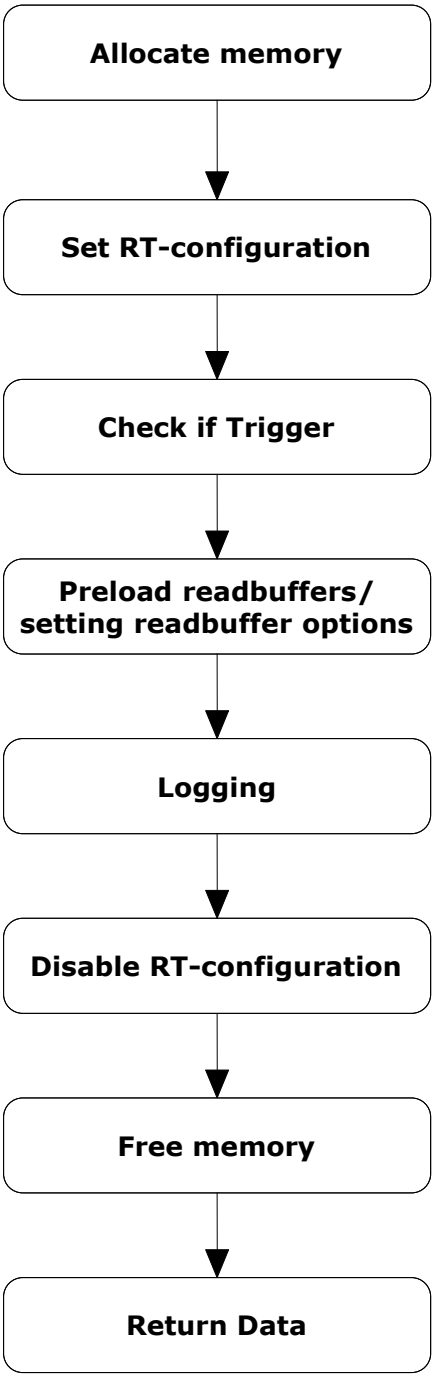

Figure 4. Block flow diagram for logging.

The trigger block is only used by those functions where trigger is specified.

There are a few functions that are different from the above structure. Most of these functions are small and just uses one TIP function regarding configuration of the channels or the timer. One however is quite different and has another outline. This is the function for getting the boards, see Figure 5.

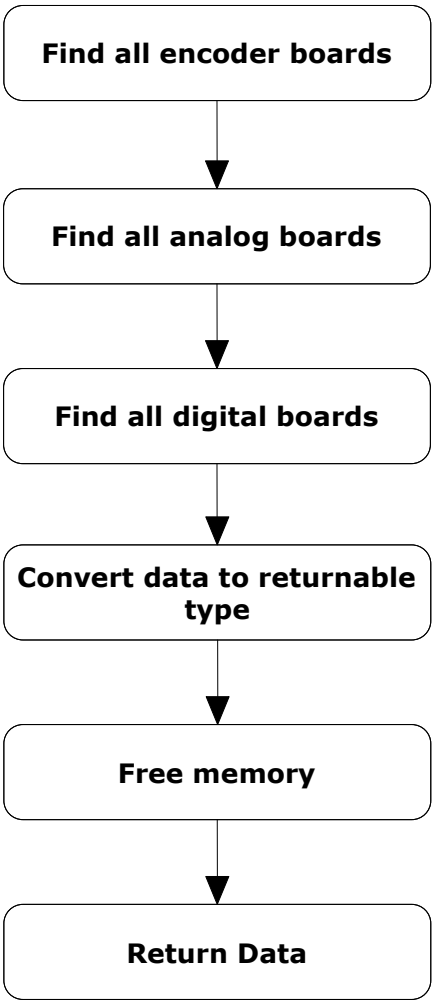

Figure 5. Block flow diagram for finding TIP cards.

6.5.3 TIP functions

In order to configure the cards and to acquire data from the connected sensor several functions from the TIP cards are required for use. There are too many configuration options in these for them to be covered in this thesis but which to use for this application is commented upon. How to use TIP functions is based on the TIP user manuals and example files that come with the drivers.

To find the connected device the following has to be done for each card. The device is accessed via the device node (/dev/tip<modelnumber>_<devicenumber>). Code for this looks as follows.

```
sprintf(Path, "/dev/tip501_%0d", i);
hDevices[i] = open(Path, O_RDWR);
currentPath = (char*)malloc(strlen(Path));
strcpy(currentPath,(char*)&Path);
paths[foundCount]=currentPath;
close(hDevices[i]);
```

In the function for getting the boards this is done for all types of devices and in loops in case multiple devices are connected. Then the information for all the cards is sent back to Java. It is also important to close the device since leaving the device open can cause problem when using the open function and connecting sensors.

Configurations

The implementation has two separate configuration functions that are not embedded in the read methods. Both these methods handles configuration of the encoder card. Specific functions are configuration of the timer and channel.

Timer

The timer works by starting counting down from a predetermined value. It decreases the value with one for each clock tick. When the value reaches zero the count starts over from the predetermined value. It also has a variable clock frequency that can be set. This is the configurations used for this application.

```
T116_TIMER_CONFIG_BUFFER    timerConfBuf;
int                         result;
int                         boardNumber;

timerConfBuf.preloadVal = 40000;
timerConfBuf.prescaler = 1;
timerConfBuf.enable = TRUE;

result = ioctl(boardNumber, T116_IOCS_CONFIG_TIMER, &timerConfBuf);
```

The struct T116_TIMER_CONFIG_BUFFER is provided by the tip116.h. Preload value for the counter is set to 40000 since this value corresponds to 5 ms with an 8 MHz clock. Reason for choosing this value is simple to make conversion from clock ticks to seconds easier later in the java part of the implementation. The prescaler value sets the clock frequency. A value of 1 is the choice for maximum clock frequency which in this case is 8 MHz. Then the configuration buffer is sent to the device via the ioctl function. The second argument in the ioctl function is the device dependent request code. The correct code for timer configuration is T116_IOCS_CONFIG_TIMER which is defined in the header file for the board. The result variable is just for error checking.

Channel

Configuration of the channel has a bit more options. The comments in the code explain most of them. Which values for the options to be used here is determined by simply testing which configuration gives the highest resolution for connected sensors. The only configuration option that needs to be commented upon is which input mode to use. The input mode to use is quadrature mode. The options are passed to the device in the same way as for the time. For additional details about the configuration see the code in Appendix F.

Trigger function

The trigger logic in the logging functions works in the following way. When a trigger is used the functions take an argument that specifies which type of trigger to look for. The trigger type is saved in an event read buffer that is provided by tip600.h. The buffer is then passed TIP600 (digital)-board which does the actual checking for the trigger. The code for the trigger looks as follows.

```
T600_EVRD_BUFFER                evBuf;
switch (trigger) {
            case 1:
                        // Waits for a high transition
                        evBuf.mode = T600_HIGH_TR;
                        break;
            case 2:
                        // Waits for a low transition
                        evBuf.mode = T600_LOW_TR;
                        break;
            case 3:
```

```
                        // Waits for any transition
                        evBuf.mode = T600_ANY_TR;
                        break;
            default:
                        // Waits for any transition
                        evBuf.mode = T600_ANY_TR;
                        break;
            }
            // some default specification for trigger: mask = position 1;
timeout 60000 ticks = 60s
evBuf.mask = (unsigned short)(1<<0);
evBuf.timeout = 60000;

result = ioctl(digitalboard, T600_IOCX_EVENT_READ, &evBuf);
```

The mask of the event read buffer specifies for which bit of the digital read to look for the trigger. In this implementation it is set to bit 1. The timeout specifies the time to wait for the trigger. The value is set in ticks. Request code to use is: T600_IOCX_EVENT_READ. One tick is 1 µs. Default value here is 60 s. The function returns an integer with value over 0 when the trigger is detected within specified time. The function returns 0 if no trigger is detected within the specified time.

Encoder logging

The encoder card does not read the sensor value directly. Instead the value that is read is that from a counter, which is connected to the sensor. The counter counts either up or down (depending on which direction the sensor rotates) from a preloaded value. This preloaded value has to be set before the actual reading of the value. It has to be set for each channel used.

```
T116_COUNTER_BUFFER         countBuf;
// Preload channel with value

countBuf.channel = channelNo;
countBuf.data = 0;
countBuf.flags = T116_IMMEDIATE_PRELOAD;

result = ioctl(encoderboard, T116_IOCS_SET_PRELOAD, &countBuf);
```

Counter buffer has some configurations. The first is data which simply sets the starting value for the counter. The flag option T116_IMMEDIATE_PRELOAD means that the preload value will be set immediate.

Now over to the actual logging code section. This is actually very similar to the preloading of the counter value. The only difference is the request code sent to ioctl. Also included here is code for storing the values.

```
// Reading and storing values in a vector
result = ioctl(encoderboard, T116_IOCG_READ_COUNTER, &countBuf);

if(result >= 0){
            // Read success. Store value
            pStore[i] = countBuf.data;
} else {
            return NULL;
}
```

When the read is completed the value can easily be accessed via dot notation.

The read function for all four channels has some small difference but works in the same way. The only difference that is worth commenting is that it uses a different type of buffer

(T116_READ_ALL_BUFFER). It is handled in the same way except that it has no channel option and the data and flags are an array of values instead of just a variable.

Analog logging

The analog read has the same structure as the encoder read but with some difference when it comes to checking if the device read was successful and configuration of the analog channel. Unlike the analog card the configuration options for the analog channel is set in the read buffer itself. The options set for the implementation are the gain which is set to 1 and the two flags, T501_DIFF and T501_PIPELINE. These flags tell the card to use differential mode and pipeline mode.

```
T501_READ_BUFFER                                rdBuf;
rdBuf.chan = channel;
rdBuf.gain = 1;
rdBuf.flags = T501_DIFF | T501_PIPELINE;
rdBuf.data = 0;
```

The actual read of the channel is the same as for the encoder except that it uses the read command instead of the ioctl and the return is the size of the read buffer.

```
NumBytes = read(analogboard, &rdBuf, sizeof(rdBuf));
if (NumBytes > 0){
          pStore[i] = rdBuf.data;
}else {
          //Analog read failed
          return NULL;
}
```

The NumBytes in case of an error has the value -1. If the reads is successful then NumBytes integer will have a value greater the zero.

Digital logging

The function for digital logging is much simpler than the others because it has no configurations that require setting. The structure of the read works in the same way as for the analog logging.

```
T600_READ_BUFFER                                readBuf;
NumBytes = read(digitalboard, &readBuf, sizeof(readBuf));
          if (NumBytes > 0){
                    pStore[i] = readBuf.value;
          }else {
                    //Read input lines failed
                    return NULL;
          }
```

There is a special thing when it comes to the trigger for the digital logging. Since all input ports are used for the read another approach is required. The approach is that one port doubles as both a trigger port and a data port. This is the simplest way to do it but it has a major disadvantage. It requires a transition on the specified port. This could be a major problem if no transition occurs naturally in the data read.

Timer read

The timer read is a function of the encoder card and uses ioctl as is standard for the encoder card. The value is stored in a buffer of type T116_TIMER_BUFFER. The time values are used in all logging functions. The reason for this is that we store the timestamp for each read from the cards.

```
T116_TIMER_BUFFER           timerBuf;
```

```
result = ioctl(encoderboard, T116_IOCG_READ_TIMER, &timerBuf);
```

6.5.4 Memory management

Since C is used for the native implementation it is necessary be careful when dealing with the memory. In this application there is also a demand for fast and easy access to memory during the read algorithms as well as constraints on conservative memory usage. Taking that into consideration the only viable option is to pre-allocate the memory before the actual read.

The simple case is when the read is from only one channel. Then a simple array is created and memory is allocated for that array. The size of the array is determined by the variable loggNo which is the total number of logging points. The total logging points is calculated for the total time of the logging.

```
long *pStore;
pStore =(long*)malloc((loggNo+1) *sizeof(*pStore));

memset(pStore,0,sizeof((loggNo+1)*sizeof(*pStore)));
```

In line with good C-programming practice the memset is used to set all bits to zero. During the functions it is also necessary to free the memory allocated. This is simply done by calling the free method.

```
free(pStore);
```

Since the application also stores timestamps for each data read an array with the timestamps also needs to be created. It is done in exactly the same way as the data array with one exception. The exception is that the type used is short integer instead of long.

The more complicated case is when the read is from multiple channels. For storing the data in this case the most convenient data structure is a multidimensional array i.e. an array of arrays. The code below is from the function for reading all encoder channels. The time array is also included in the multidimensional array. Note that long is used for type of the time array as well.

```
long **pStore;
pStore =(long**) malloc(5 * sizeof(long*));

for(i = 0; i < 5; i++)
{
            pStore[i] =(long*) malloc((loggNo+1) * sizeof(long));
            memset(pStore[i],0,sizeof((loggNo+1)*sizeof(long)));
}
```

There are other less complicated ways of allocating the array but this way makes it possible to access the individual element via indices.

To free up the allocated memory is more difficult than for the simple array but not hard. The principle is to start with freeing the memory for the inner arrays first and then the array that holds the inner arrays.

```
for(i = 0; i < 5; i++)
{
            free(pStore[i]);
}
free(pStore);
```

6.5.5 RT functions and structures

During the actual read algorithm the priority of the section needs to be set to maximum. Since these priorities are used by all reading functions they are defined globally.

```
#define RT_PRIORITY (79)
#define NON_RT_PRIO (30)
```

The important number here to consider is the Linux kernel interrupt and tasklets handler priority which is 50[20]. The RT priority should at least be higher than this and the non RT priority below this. The non-RT priority is just taken out of the blue. The RT priority of 79 is determined by empirically testing the effects of different priorities regarding the system's performance. The testing of different priorities is described in chapter 4.

Setting RT properties

When the different functions in the application enter a RT critical section the function must declare that it is an RT-task. This is done by calling the set scheduler function (sched_setschedular) with arguments for priority and scheduling policy.

```
struct sched_param param;
param.sched_priority = RT_PRIORITY;
if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
            //perror("sched_setscheduler failed");
{
```

The priority here is the globally defined 79 and the scheduling policy is round robin. The RT-priority of 79 is picked to be above 70 according to chapter 4. Round robin policy means each process gets a fixed slice to execute on. The reason for using round robin instead of FIFO (first in first out) is that the worst case delay between reads is better with round robin. More about the selection of the scheduling policy can be found in chapter 4.

Logging algorithm

The principle for the algorithm of the actual logging is simply to run a for-loop for a fixed number of data points and store the time for the read. How this works is best illustrated with an example. The example is taken from the encoder logging on one channel. The algorithm is simply running a for loop for a fixed number of times that corresponds to

```
for(i=0;i<loggNo;i++)
{
            result = ioctl(encoderboard, T116_IOCG_READ_COUNTER, &countBuf);
            pStore[i] = countBuf.data;

            result = ioctl(encoderboard, T116_IOCG_READ_TIMER, &timerBuf);
            timeCount[i] = timerBuf.value;
}
```

One important note here is that if the task gets interrupted (by a process with higher priority i.e. SMI's between the read of the encoder value and timer read then a jitter will be introduced. SMI stands for System Management Interrupts. Usually the SMI's handles hardware exceptions and power management. This jitter could potentially be a big problem when the data is analyzed.

Setting non-RT properties

When the logging algorithm leaves the RT-section all properties have to be set to non-RT. This is done in the same way as for setting the RT-properties. The only difference is the

[20]RT_PREEMPT_HOWTO, RT Linux wiki http://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO (2010-02-28)

priority which is set to the NON_RT_PRIO of 30. The priority is picked to be under 50 according to chapter 4.

6.5.6 Mapping of JNI types

Sending the data from the native code back to Java requires using a set of types introduces by JNI. The underlying reason for using this is that Java and native code map types differently. Structure of the JNI types is using the Java type with a j added at the beginning (example: jint, jboolean). The primitive types like int, boolean etc. are easy to convert. They can be treated as normal types in C with typecasting. When it comes to the non-primitive types like string and array types it is more complicated. Here the need for the JNI interface pointer (JNIEnv*) shows up.

For this thesis the interesting case is how to convert an array of arrays to the JNI type so that it is possible to return the multidimensional array. The way to do this is to treat it as an array of objects. Here is the example from the function for logging one encoder channel.

```
jclass intArrCls = (*env)->FindClass(env, "[I");
if(intArrCls == NULL){
            // Returning NULL since we have an exception here
            return NULL;
}

array = (*envy)->NewObjectArray(envy, loggNo, intArrCls, NULL);

for(i=0;i<loggNo;i++)
{
            jint tempColumn[2];
            jintArray column = (*envy)->NewIntArray(envy, 2);

            tempColumn[0] = pStore[i];
            tempColumn[1] = timeCount[i];

            (*env)->SetIntArrayRegion(envy, column, 0, 2, tempColumn);
            (*env)->SetObjectArrayElement(envy, array, i, column);
            (*env)->DeleteLocalRef(envy, column);
}
```

The first jclass call is for finding which map of the types to use. Next an array is created with length the same as the data array. After that a loop goes through the array and adds the created integer arrays as objects in the object array. Since the Java garbage collector does not extend to the native side the DeleteLocalRef call is necessary to avoid memory leaks. Now the created array can be returned as normal.

6.5.7 Compiling the library

When the native code is implemented it needs to be compiled into a shared library and put into the appropriate directory.

The compilation of the library is pretty straight forward if done in eclipse. The compiler and linker menus here are very nice where there are checkboxes for almost everything. A few important compiler options have to be enabled. First of all the –lrt flag has to be set for both the compiler and the linker. And for the linker the –shared flag also has to be enabled. If the compilation is done from the command line it looks as follows:

```
gcc –I/usr/lib/jvm/java-6-sun-1.6.0.17/include
            -I/usr/lib/jvm/java-6-sun-1.6.0.17/include/linux –O0 –g3 –Wall
            -c –fmessage-length=0 –lrt –MMD –MP –MF”motionloggerIo.d”
            -MT” motionloggerIo.d” –o” motionloggerIo.o”
```

```
            "../motionloggerIo.c"

gcc -shared -o"libmotionloggerIo.so" ./motionlogger.o -lrt
```

Most of the options passed to the compiler are just standard options generated by eclipse. There are however a few necessary options that needs commenting. The first is the –lrt option. This option is for loading the RT libraries. Next is the two include paths: –I/usr/lib/jvm/java-6-sun-1.6.0.17/include and -I/usr/lib/jvm/java-6-sun-1.6.0.17/include/linux. These are necessary for getting access to the JNI libraries.

These commands creates a shared library named libmotionloggerIo.so. This file has to be included in the LD_LIBRARY_PATH. In the case for this application it is quite simply /lib. The file is simply put in this directory. Another possibility would be to change the LD_LIBRARY_PATH to the directory where the file is located. This however is not recommended under Linux since it may affect other applications that use the LD_LIBRARY_PATH.

6.6 JNI Testing

6.6.1 JNI tests

To test and verify the JNI a series of test files was created. These files can be found in Appendix C. The test files are implemented as a JUnit test case. The test files are fundamentally simple. They just calls the native functions and then prints a sample of the returned data. A list with the available test files follows. The names are self explanatory.

```
TestJNIrunCounter.java
TestJNIrunAllCounters.java
TestJNIrunAnalogChannel.java
TestJNIrunAllAnalogChannels.java
TestJNIrunDigital.java
TestJNISystemPerformance.java
```

The last method needs some commenting. This native call works differently because it calls a native method which runs a system performance test as described in 5.1.

6.6.2 JNI test results

There are not many conclusions to draw from the JNI test except that JNI works as intended. However, one important thing was discovered during the testing. When the logging time comes close to the max logging time of 30 s the JVM runs out of memory. This out of memory exception appears because the JVM by default has a memory limit of 64 MB. To avoid this exception the memory limit has to be raised. To raise the limit options for this have to be passed to the JVM when the program is started. The options look as follows.

```
-Xms64m -Xmx128m
```

These options set the maximum memory allocation to 128 MB and the initial memory allocation to 64MB.

7 Controller

The purpose of using a controller class is to create a good structure for all the communication between the different parts of the system. For the exact implementation see Appendix G.

7.1 Singleton

The controller is implemented as singleton. A singleton means that only one instance of the class can be instantiated. The pattern for implementing a singleton is quite simple.

```
private static Controller singleton;
…

public static Controller getController(){
            if(singleton == null 0){
                        singleton = new Controller();
            return singleton;
}
```

The singleton implementation effectively prevents multiple instances and thereby multiple simultaneous calls to other classes.

7.2 Controller proxy

The HMI part of the application will access the controller from a web page using JavaScript. The JavaScript is executed at the client side (web browser). That means that the controller has to be sent over a network, which is not desirable. To solve this issue a controller proxy class is created. This proxy class implements Java Serializable Interface. The Java Serializable Interface makes it possible to turn any object into a sequence of bytes that can be fully restored.

The main functionality for the proxy class is to pass method calls to the actual controller and send the information back. The only methods that are implemented in the proxy are metadata about graphs.

7.3 Controller functions

The code for the controller is largely self explanatory. The only thing that needs further explanation is how the logging calls work. In principle all the logging calls are the same. This is an example when logging on one encoder channel.

```
public boolean loggOneEncoderChannel(float time, int channelNo){
            EncoderChannel channel = lstEncoderChannels.get(channelNo-1);
            tip116Board.configureExternalTimer();
            channel.initialize();
            long timeStamp = System.currentTimeMillis();
            boolean storeSuccess = DataContainer.getDataContainer().
            setData(channel.runCounter(time),timeStamp);

            if(storeSuccess){
                        // Logging success,
                        //Set some info about the read in dataContain
                        DataContainer.getDataContainer()
                        .setDataType("Encoder");

                        DataContainer.getDataContainer().setChannels(1);
            }
            return storeSuccess;
}
```

As seen in the code above the return of the call to native method (channel.runCounter(time)) is directly sent to the data container along with the timestamp. The timestamp is needed for the database storage. There are a couple more variables that are sent to the data container.

8 Data container

The data container is a class where the result from the loggings is located. It holds at most data from one read at a time. The data can also be loaded from or stored in this class. The data container implementation is given in appendix H.

8.1 Overview

Since the data from loggings has the potential to grow very large and a lot of operations on the data have to be done, a class for holding the data is necessary. The data container is also implemented as a singleton to further ease the strain on the limited computer memory.

The data container holds the data in a matrix of type double. This is different from the integer type matrix that the JNI returns. The data container converts this into the desired double type data matrix. The reason for having the data in double format is that the Open Process Logger framework uses double as standard.

The data container also holds metadata about the logging task. The metadata consists of information about number of channels used, data type for the logging, timestamp, name of the logging task and corresponding description.

All functionality regarding the raw data is implemented in this class. This includes filtering, derivation and sending the data to the HMI via the controller. Database functions are also located in this class.

8.2 Data functions

As the OPL works by getting the data in batches of variable size, the return functions need to be implemented to make this possible. According to specification the first and second order derivatives of the data also need to be calculated. Due to the memory limitation of the system the derivations need to be done on the fly for each requested batch. It is also not possible to use standard derivation formulas since we have discrete data points with variable time intervals between the data points. Therefore an approximation that considers the variable time intervals has to be used. Taking these demands into consideration the derivation function becomes:

$$f'(x_n) \approx \frac{f(x_{n+1}) - f(x_{n-1})}{h_{n+1,n} + h_{n,n-1}} \quad [21]$$

$$f''(x_n) \approx \frac{f(x_{n-1}) - 2 * f(x_n) + f(x_{n+1})}{h_{n-1,n} * h_{n,n+1}} \quad [22]$$

These approximations are however not perfect. The approximation will still be affected by varying time intervals but it is a lot better than assuming that the time intervals are constant. The function for getting a batch looks as shown in the code below. This is from the method when getting a batch of first derivative data.

```
public FineDataPoint[] getDataBatchFirstDerivative(int startIndex, int
                                    stopIndex, int selectedChannel){
```

[21] Resources for Financial Engineers http://www.sitmo.com/eq/274 (2010-04-23)
[22] Numerical Differentiation, Mathematics Department at Cal State Fullerton http://math.fullerton.edu/mathews/n2003/NumericalDiffMod.html (2010-04-23)

```
            if(selectedChannel > this.channels || selectedChannel <= 0)
                                                return null;
            int timeColumn = dataMatrix[0].length-1;
            int dataColumn = selectedChannel -1;
            double dData;
            FineDataPoint[] batch = new FineDataPoint[stopIndex-
                                                    startIndex+1];

            for(int i= startIndex; i<=stopIndex;i++){
                        dData = (dataMatrix[i+1][dataColumn] -
                                    dataMatrix[i-1][dataColumn])/
                                    (dataMatrix[i+1][timeColumn]-
                                    dataMatrix[i-1][timeColumn]);

                        batch[i-startIndex] = new FineDataPoint(dData,
                                    (long)(dataMatrix[i][timeColumn]/1000), 0,
                                    (dataMatrix[i][timeColumn])%1000);
            }
            return batch;
}
```

The method returns an array with FineDataPoint from start index to stop index. FineDataPoint is the extended format that the OPL framework uses for holding data with high time resolution.

8.3 Binary search

As shown above the size and endpoints of the batches are determined by indices of the matrix. This is a problem since the OPL graph creator uses time as identifier for which values to get. To fix this problem there needs to be a function that returns the index for a specific time. Since the data matrix can be very large the algorithm for this has to be fast. Adding the fact that the time column of the matrix is a sorted vector means that the obvious choice is to use a binary search algorithm[23]. The implementation looks as follows:

```
public int getIndexFromTime(double micros){
            if(dataMatrix == null || dataMatrix.length == 0 ||
             dataMatrix[dataMatrix.length-1].length == 0) return 0;
                        return binarySearch(micros, 0, dataMatrix.length-1);
            }
private int binarySearch(double micros, int first, int last){
            if (first>=last){
                        return first;
            }
            if(last-first == 1){
                        return (micros >
                                    dataMatrix[last][dataMatrix[0].length-1])
                                    ? last : first;
            }

            int middle = (first+last)/2;

            if (micros == dataMatrix[middle][dataMatrix[0].length-1]){
                        return middle;
            }
            if(micros > dataMatrix[middle][dataMatrix[0].length-1]){
                        return binarySearch(micros, middle, last);
            }
```

[23]The National Institute of Standards and Technology
http://www.itl.nist.gov/div897/sqg/dads/HTML/binarySearch.html (2010-03-21)

```
            return binarySearch(micros, first, middle);
}
```

The binary search works by inspecting the middle element of each sorted list. If the value in the middle is the sought value then the index has been found, otherwise the upper or lower part of the array is chosen for further search and the binary search method is called recursively with the chosen part of the array. The algorithm is not interested in the exact index for the specific time, only the first index before that specified micro second.

8.4 Database functions

To store the finished log tasks the database interface from Open Process Logger is used. However not ideal the functionality and structure are sufficient for the needs. By using this interface the need for designing and implementing a custom database is avoided.

There are three functions with regard to the database: pull, push and a function for getting all log tasks which are stored in the database.

The data from the logging tasks is stored by first adding a log task in the data base with metadata about the logging task. Then data itself is added to the database in connection with the log task id. To make the connection possible a separate data object has to be created. The data object is a standard form for the OPL database. The duplicate data causes further strain on the memory of the computer. The code is shown below.

```
public boolean pushToDatabase(){
            r.gc();// Garbage collector call
            // Function for setting stuff in database
            LoggingTask logTask = new LoggingTask(loggName, new
                                                HashMap<String, String>());
            logTask.addProperty("Description", description);
            logTask.addProperty("Channels", Integer.toString(channels));
            logTask.addProperty("DataType", dataType);
            logTask.addProperty("Timestamp", Long.toString(timeStamp));
            logTask.setTaskId(timeStamp);
            logTask.setStartTime(0);
            logTask.setStopTime((long)((getLastTimestamp() *
                                0.125)/1000000));
            OPLDatabase.instance().addLogTask(logTask);

            Data data = new Data(loggName, 0, logTask.getTaskId(),
                                 dataMatrix,  new Date(timeStamp));
            r.gc();
            OPLDatabase.getDataBuffer().addData(data);
            OPLDatabase.instance().flush();
            r.gc();
            return true;
}
```

One more important thing here is the garbage collector calls (r.gc) in the function. These are necessary to avoid out of memory exceptions and crashes because of memory problems. Without the garbage collector calls the maximum possible amount of data is about 25% lower.

The pull function works in the same way as the push function with the duplicated data and garbage collector calls.

In the OPL framework there are functions for getting all stored log tasks from the database. The function used returns a list with logging tasks. The instances do however not have all the necessary information that the HMI needs to present which stored tasks are available. The

way to solve this is to use a log task proxy instead. The log task proxy holds all necessary information about the task and it is serializable. The function simply uses the acquired log task list and for each log task creates a log task proxy. The log task proxies are then added to an array list and returned. The code looks as follows.

```
public ArrayList<LogTaskProxy> getAllLoggTasksProxyFromDatabase(){
            List<LoggingTaskData> loggingTasks = OPLDatabase.instance()
                                                .getFinishedLogTasks();
            LoggingTask tempLoggTask;
            LogTaskProxy tempProxy;
            ArrayList<LogTaskProxy> logTaskProxyArray = new
                                    ArrayList<LogTaskProxy>();

            for(int i = 0; i < loggingTasks.size(); i++){
                        tempLoggTask = OPLDatabase.instance().findLogTask
                                    (loggingTasks.get(i).getName());

                        tempProxy = tempLoggTask.proxify();
                        logTaskProxyArray.add(tempProxy);
            }
            return logTaskProxyArray;
}
```

This function works by getting the name for the log task via the list and then acquiring the actual log task and finally proxifies the log task.

8.5 Filter function

To fulfill the specification for the system a low pass filter also needs to be implemented. The filter is used for removing high frequency noise from the logging. The implementation and design of the filter is based on [2]. The pseudocode for the filter looks as follows.

```
y[i] = y[i-1] + α * (x[i] - y[i-1])
```

where y[] is the output array, x[] is the input array and α is the smoothing factor. The smoothing factor is calculated in the following way.

$$\alpha = \frac{h}{\tau + h}$$

Here h is the period between the samples and τ is the time constant for the filter. The time constant is linked to the cutoff frequency for the filter and is calculated in the following way.

$$\tau = \frac{1}{2\pi * f_c}$$

where f_c is the cutoff frequency. The cutoff frequency is set to 500Hz. The cutoff frequency is chosen to be above the target frequencies (3-400 Hz) for the system. The implementation of the filter looks as follows.

```
public boolean lowPassFilterData(){
            int timeColumn = dataMatrix[0].length-1;
            int channels = getChannels();
            int frequencyCutOff = 500;
            int i;
            double RC = (1/(2*Math.PI*frequencyCutOff))*8000000;
            double alfa;
            double dT;
            for(i=1;i<getDataCount()-1;i++){ // rows
                        dT = (dataMatrix[i][timeColumn]-
                                    dataMatrix[i-1][timeColumn])*0.125;
                        alfa = dT/(dT+RC);
                        for(int j=0;j<channels-1; i++){ // data columns
```

```
                                    dataMatrix[i][j] = dataMatrix[i-1][j] +
                                                alfa * (dataMatrix[i][j]
                                               -dataMatrix[i-1][j]);
                        }
                }
                r.gc();// Garbage collector call
                return true;
}
```

As shown in the code the filtering is done for every column in the data matrix. One thing that also needs commenting is that the smoothing factor is calculated for each interval. This is done because the sampling period is not constant as stated in 4.2.

9 HMI

Implementing the HMI is straightforward programming in HTML and JavaScript. All functionality, principal and general design of the HMI will be presented and explained. All details of the code will however not be explained largely due to the lack of documentation of the OPL framework. The code that is specific for the HMI is located in Appendix I.

9.1 Overview

The HMI has need for two fairly separate parts, logging view and database view. Both views use the same functions when it comes to presenting data. The graph functions are from the OPL framework. The logging view needs menus for setting options for the logging and one for the logging result. The settings menu is where the user chooses logging type, number of channels, digital trigger options. It also contains a start button for the logging. The result menu has options for post processing, viewing and storing the data. The overall design of the webpage is shown Figure 6.

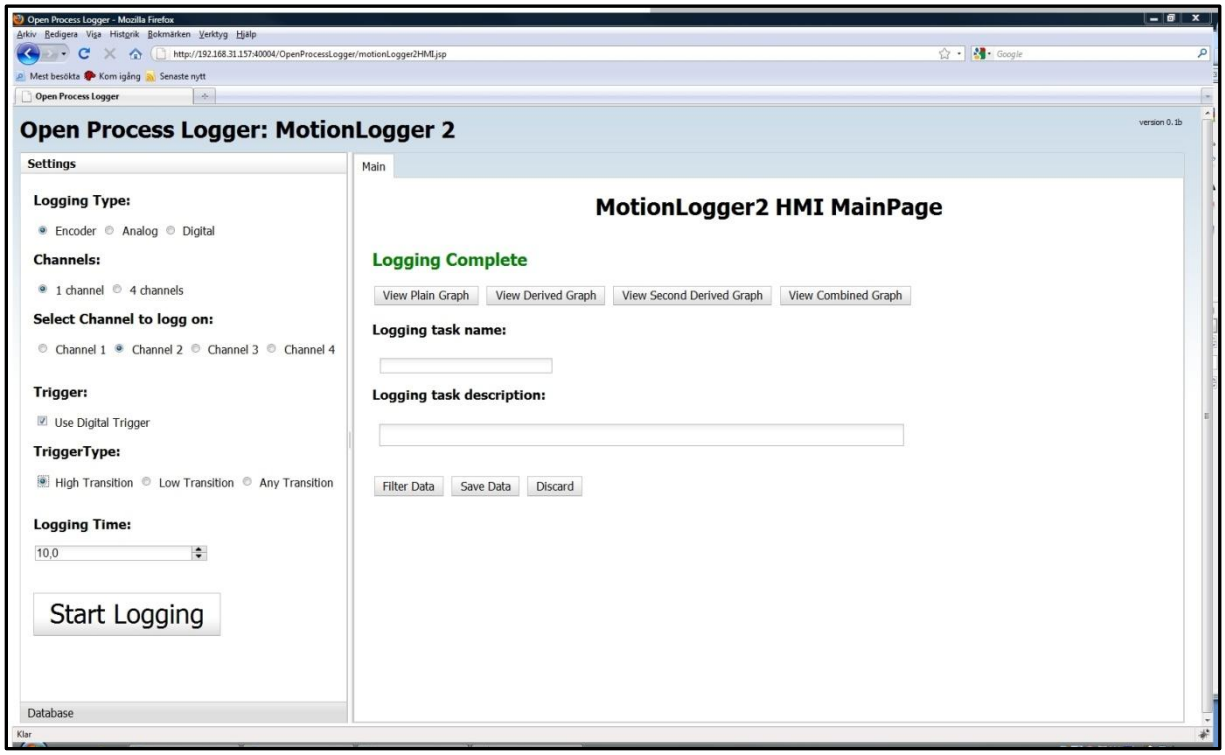

Figure 6. Screenshot of the HMI web page.

The page is organized in two panes. The left pane is for either the logging menu settings or the database menu. The right pane is for the presenting menu and post-processing options. To simplify further writing the left pane will be referred to as options pane and the right as the main pane. The graphs presenting the data will show up as tabs in the main pane.

9.2 Logging menu

The logging menu is where all options and selections for the logging are set. The view is enabled in the options pane by default when the webpage is loaded. Figure 7 shows a screenshot of the menu. For safety and easier use the menu is designed to only show the options when all prerequisites are fulfilled. Another safety feature is that the start button is disabled until all necessary prerequisites are set.

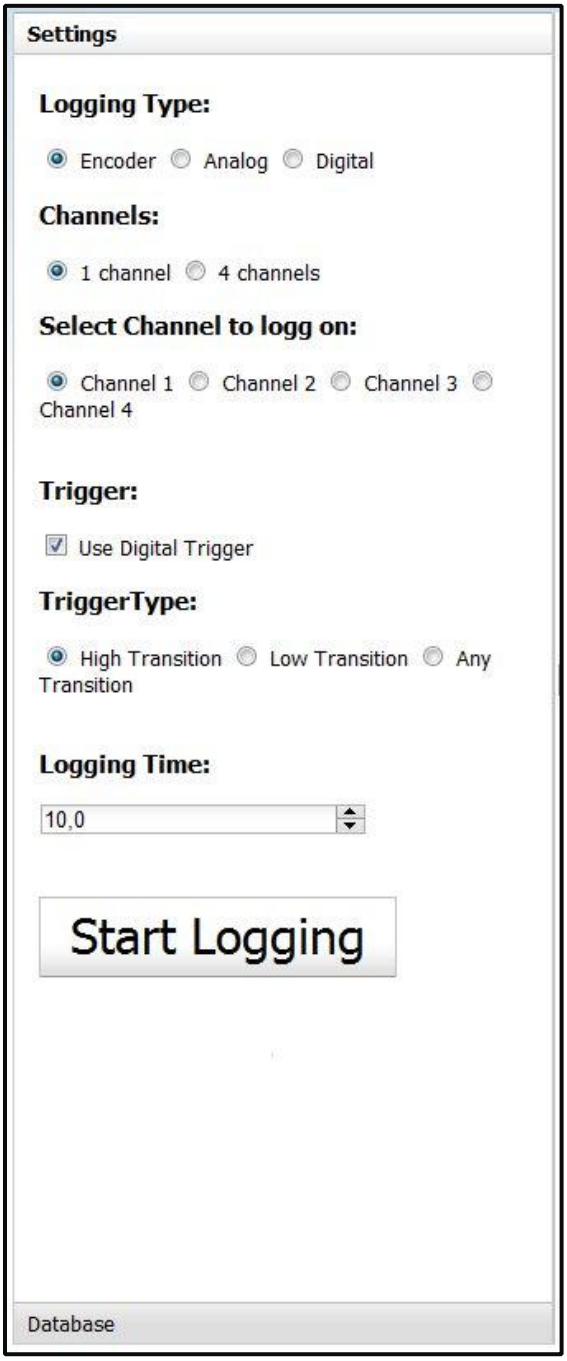

Figure 7. Screenshot of the logging menu.

9.3 Database menu

The database menu is much simpler than the logging menu. It is simply a list of the stored log tasks that exist in the database. Information is limited to the name of the log task and timestamp for the log task. The database menu is shown in Figure 8.

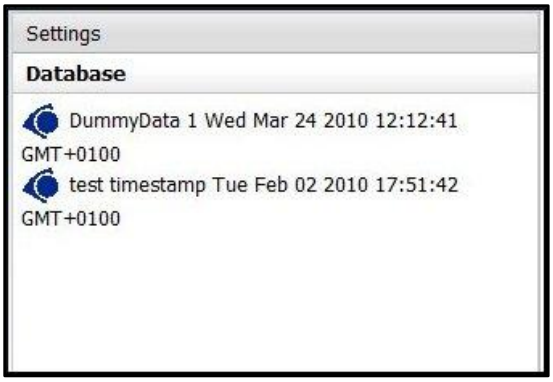

Figure 8. Screenshot of the database menu.

9.4 Main pane

The main pane is where the all the post processing options are located. How the webpage layout is shown Figures 9 and 10. Functions that are accessed via buttons are the graph viewing, filter and data saving functions. There are four different graph functions: raw data, first derivative, second derivative and a combined version that shows both the raw data and the first derivative. Figure 11 shows a screenshot of a graph.

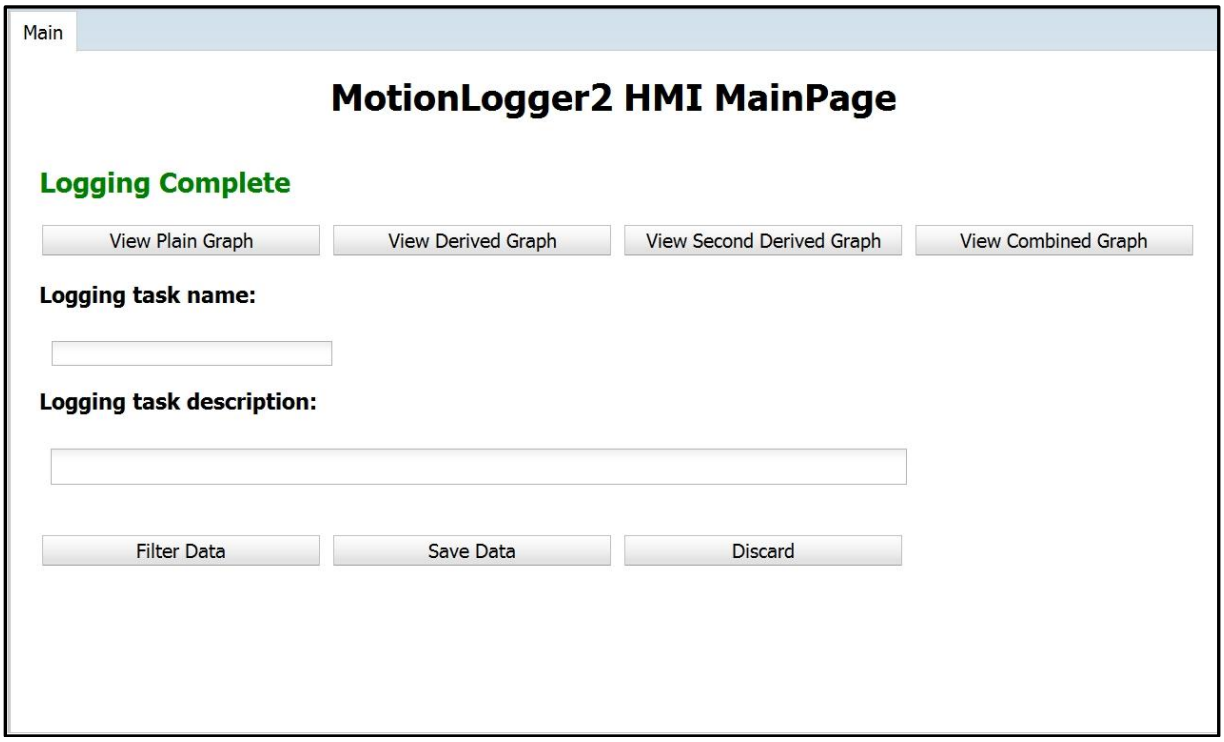

Figure 9. Screenshot of the main pane when a logging is finished.

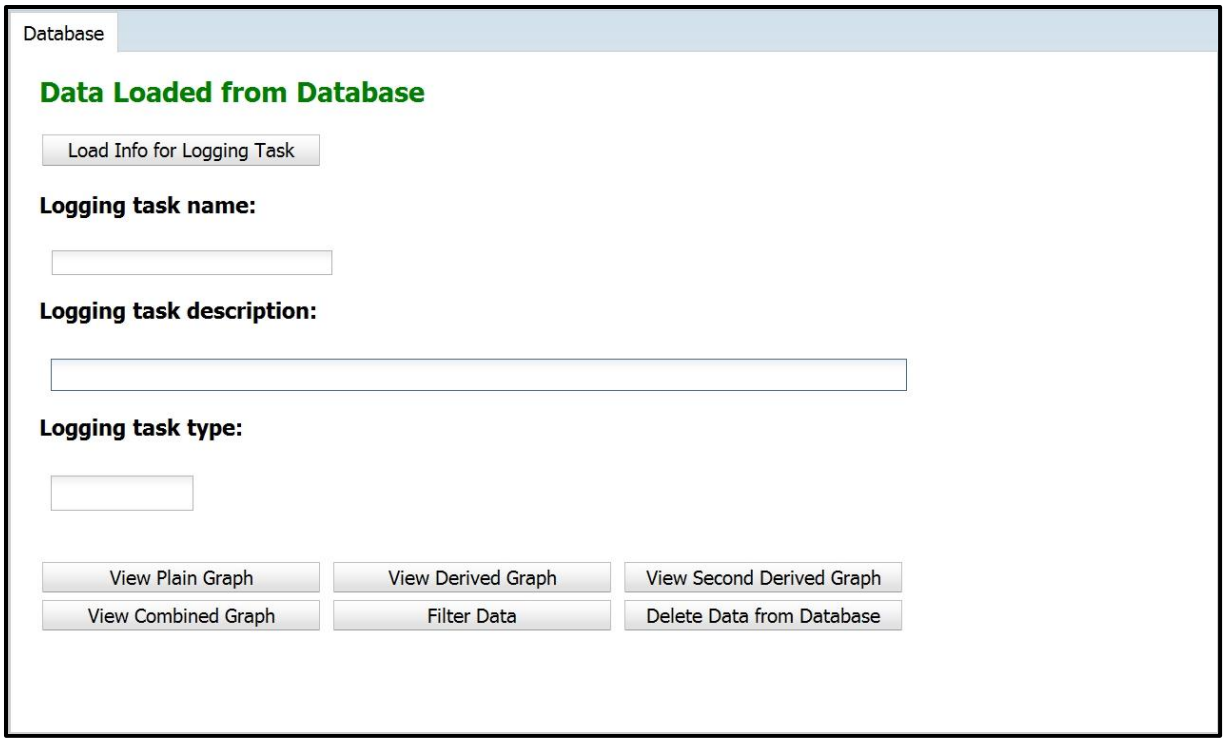

Figure 10. Screenshot of the main pane when log task data is loaded from the database.

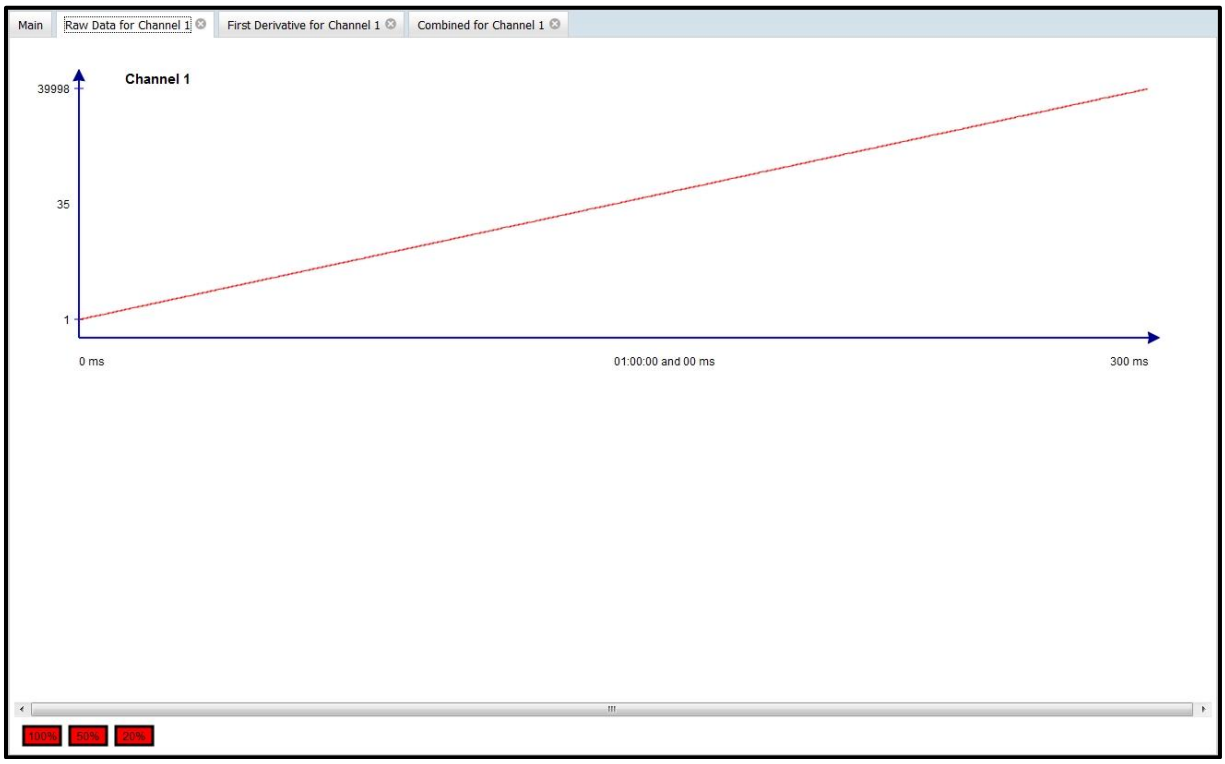

Figure 11. Screenshot of a graph. The graph shows generated test data that simulates a constant speed.

10 System testing

After the development it is time for testing the system. The system test has not been as extensive as the initial description stated due to lack of time. The testing has just been done with generated test data or data for the encoder sensor. The purpose of the test is to verify all functionality of the system and to determine the amount of time it takes for creating the graphs. The need for testing and determine the testing times came up during the later stages of the development of the system. Tetra Pak also requested an analysis of this problem.

10.1 Methodology

Methodology for the functionality test is simply to test all functions in all reasonable combinations and logging of different types of data. The digital trigger functions are not included in the test in this phase.

Methodology for the graph drawing times is to use generated data to get a worst case scenario for the graph viewer. Then let the OPL draw the graphs and measure the time it takes for different number of samples (corresponding to logging time).

10.2 Test results and conclusions

The testing verified all the functionality of the functions. However a limitation appeared during the tests. Maximum logging time that was possible without crashing the application was 20s for one encoder channel and 15s if it should be possible to save the data to the database. Since the specifications states a required logging time of 30s this is major problem. Further investigations about the problem reveals that it is caused by the limited amount of memory for the computer. Since the application runs with full privileges (and high priority) this leads to a deadlock when the program runs out of memory. However this is an acceptable decrease of system performance due to the fact that it is a physical limitation of the hardware and that the specification for the logging is set very high.

The drawing test results are best presented in table form.

Data points:	Raw data:	1st derivative:	2nd derivative:
400 000	8 min	3 min	2 min
1 500 000	45 min	7 min	7 min
5 000 000	>2 h	16 min	16 min

From this table most of the times are at an acceptable level for the 400 000 and 1500 000 data points. The problem is for the raw data graphs where time can go up to more than an hour. The reason for the long times is that the OPL redraw the graph for each data batch if the max and min values of the data changes. To minimize the number of redraws the batch size were increased from 1100 data points to 10 000 and 20 000. This led however to even longer drawing times even thought the number of redraws was considerable lower. To investigate why the overall time did not improve further experimenting with the batch sizes were carried out. The experimenting showed that the limiting factor is the network communication. Something in the Apache Tomcat or the network itself makes the transmission time for each data set longer when the batches grow in size. This phenomenon has also shown up in other OPL based applications developed by Industriprojektbyrån. Unfortunally no solution exists at the time of writing.

11 Conclusion

The purpose of the thesis was to develop a fully functional and tested measuring system according to the specifications. The first conclusion to be drawn is that from the original description a system reconfiguration part had to be added. This led to that the last part of the original specification had to be compressed into just the absolute necessary tests. Even with the compressed test part the scope probably should have been split into two separate master projects or performed by two persons.

The overall performance of the implemented system is quite impressive and in many aspects well above the specifications. Especially the logging frequencies are considerable above the specifications with the exception for the 4 encoder logging.

A conclusion that also can be draw is that there were two limiting factors for the system performance. First the low amount of RAM memory of the computer severely limited the performance of the system. The second limiting factor was the nature of the computer itself with the SMI's resulting in that the system is not fully deterministic. Both these issues should be dealt with in the further development of the system.

With some modifications a fully functional system was developed. The system is in accordance with the original specifications when it is not limited by factors that could not be changed or improved during the thesis work. The system is not ready for delivery yet due to additional requests from the customer. These requests have appeared from experience during development of the system.

12 Future work

There are three natural steps in the further development of the system. The first step is to be considered a part of the initial development. It is not included in this thesis due to lack of time and most issues has come up during the development and communication with the costumer. Steps 2 and 3 are suggestions on how this logging system can be developed further. Step 2 focuses on physical improvements of the hardware and the actual logging. Step 3 focuses on the presentation and post processing of the data. Also included in the steps is time estimation for the further development.

12.1 Step 1

Step 1 of the development is necessary for fulfilling the delivery requirements set by Tetra Pak. After step 1 the system is complete for delivery to the customer.

12.1.1 Memory management

The most important part of step 1 is to improve the memory management on the native side of the system. The system today has variable time between the push of the start button and the actual start of the logging. This delay depends on the time for memory allocation and can last up to approximately 10 s. The time varies depending on the logging time due to how the OS actually allocates the memory. This delay is very undesirable. The request from Tetra Pak is that the delay is kept at a low constant time. The specified target time is set to 100 ms.

To achieve the target time the memory management has to be rebuilt. The way to solve this is to separate the memory allocation from the actual logging. The memory allocation has to be activated by the operator as a preparation for the logging. The separation of the memory allocation is not very difficult but requires a lot of rebuilding of the Java Native Interface which is time consuming. It also requires modification of the controller and HMI. Especially the HMI has to be modified with additions of new safety functions to avoid memory leaks and multiple calls over the JNI.

12.1.2 Deletion of old data

Another request for the system is the possibility to delete old data in the database. Today the database does not incorporate any function for deleting data. The OPL database (which the system uses) only has functions for deleting the log task if there is no data connected to it. In order to make deletion possible the OPL database has to be modified. Functions for deleting the data have to be implemented as well as functions in the controller and the HMI for using the deleting functions.

12.1.3 Time estimation

Approximate time required for step 1 is roughly two weeks (80 h). Most of this time will be for the memory management. The deletion is a fairly small project which is estimated to a couple of days work at the most.

12.2 Step 2

Step 2 focuses on improving the determinism of the logging and addressing the lack of RAM memory.

12.2.1 Upgrading the computer

The only realistic way to address the low amount of memory is to change the computer. There is no simple way to add more memory due to the fact that the RAM is soldered into the board.

Another factor is that the current computer only supports up to 1 GB RAM. 1 GB should be enough but to be safe 2 GB is desired. Upgrading to a faster computer with 2 GB memory will also make it possible to run the logging from the computer itself. A faster computer will probably give better logging performance with higher logging frequencies. There is also one important demand on the new computer and that is that the global SMI is not implemented in the ROM. The reason for this is that it should be possible to disable all SMI's without the risk of burning the CPU. Furthermore it would not hurt to get a computer that supports high resolution timers even though the precision of these probably is not high enough to be really useful. It is probably necessary to make a thorough survey to find a computer with the desired characteristics. When the upgraded computer is installed a new round of system performance analysis needs to be carried out as well as modifications of the existing post processing functions.

12.2.2 New kernel

To improve the determinism of the logging a new kernel needs to be compiled. The existing kernel is an Ubuntu specific kernel with some tweaks and not ideal for this system. The way to go here is to get a Linux Vanilla kernel and patch it with the RT-patch manually. With the right configuration options this kernel will improve the determinism of the logging. The most important option is disabling the global SMI's. As stated in the previous section this requires an upgraded computer.

12.2.3 Time estimation

The time estimation for the upgrading of the computer is quite hard due to the uncertainty about how much effort has to be done to find an appropriate computer. This might take one phone call and it may take two weeks or even more. To install and test the new computer would take about a week. Estimating the time for compiling a new kernel is much easier. In total with some experimenting it will take about a week. In total the estimated time required for step 2 is 4 weeks (160 h) assuming that the computer research would take 2 weeks.

12.3 Step 3

Step 3 is about improving the HMI part and post processing of the data.

12.3.1 Graph presentation

As stated in Section 10.2 the drawing times for the graphs are extensive. The long drawing times lowers the user friendliness. There are two potential ways to address this. The simplest way is to only use sampled data, for example only use every tenth data point or so. This is only recommended as a quick fix as it may lose important information especially during logging with fast dynamics in the data.
The other alternative is to first send the raw data to the client side and then fetch the data from there. The initial time for sending all data would not be improved but after that all graph drawing would be considerably faster.

12.3.2 Export data

In the existing system that Tetra Pak uses today all post processing is done by external programs. The one most frequently used is Excel. To still be able to analyze the data in Excel an export function is required. Also recommended is to make export functions for Matlab or other data processing tools.

One warning about exporting the data to Excel is necessary. It is not certain that Excel can handle the large amount of data the system produces. Doing operations on 5 000 000 data points may cause undetermined behavior. The cause for the undetermined behavior relates to the fact that Excel only supports 1 000 000 rows.

12.3.3 Slow speed compatibility

In the system today logging data that changes slowly causes a problem due to insufficient resolution of the sensors. The problem results in that several samples have the same value. This leads to very messy graphs. The solution for this problem is adding logging methods with variable logging frequency that can be set by the operator.

12.3.4 Filtering functions

The system needs more filtering functions. Most important is noise filtration. Even in the initial test noise from the bearings could be detected. External noise from the actual measured system will also show up. Extensive research about which type of filter to use naturally has to be carried out. The guess based on the experiments and testing is that a 3rd order Butterworth bandpass filter is necessary. Another suggestion is to investigate using a Kalman filter. The Kalman filter could possibly be used either in combination with other filters or as standalone filter. Using a Kalman filter would considerably reduce the risk of filtering out useful data.

12.3.5 Fourier transform

A possible and recommended development is to implement a fast Fourier transform for the data. This would make it possible to analyze the data in the frequency domain instead of the time domain.

12.3.6 Time estimation

The time estimation for step 3 comes with a lot of uncertainty. The uncertainty largely depends on which of the upgrading alternatives are chosen. Another uncertainty factor is the filtering. The filter specifications depend largely on the characteristics of the measured system. With these uncertainties taken into consideration the estimated development time for step 3 will be 8 (320 h) weeks.

13 References

13.1 Books

[1]Shiang Liang, The Java Native Interface ISBN 0-201-32577-2 available online: http://java.sun.com/docs/books/jni/ (2010-03-19).

[2]Julius O. Smith III, Introduction to Digital Filters (September 2007 Edition) available online: https://ccrma.stanford.edu/~jos/filters/filters.html (2009-03-19)

13.2 Internet

[1] Open Modular computing specification CompactPCI, http://www.picmg.org/v2internal/compactpci.htm (2010-02-11)

[2] Tews TIP116 datasheet http://www.tews.com/Products/Datasheets/TIP/TIP116.property=PdfFile.pdf (2010-02-11)

[3] Tews TIP501 datasheet http://www.tews.com/Products/Datasheets/TIP/TIP501.property=PdfFile.pdf (2010-02-11)

[4] Tews TIP600 datasheet http://www.tews.com/Products/Datasheets/TIP/TIP501.property=PdfFile.pdf (2010-02-11)

[5] (2009) Kontron CP306 http://emea.kontron.com/products/boards+and+mezzanines/3u+compactpci/x86+processor/cp306.html (2010-02-11)

[6] Mail conversation Gerrit Hesse, TEWS technologies 2009-10-17

[7] RT_PREEMPT_HOWTO, RT Linux wiki http://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO (2010-02-28)

[8] Master Kernel Thread, Ubuntu Forums http://ubuntuforums.org/showthread.php?t=311158 (2010-02-12)

[9] Ubuntu Kernel Team's Git Repositories. http://kernel.ubuntu.com/git (2010-02-15)

[10] The Linux Kernel Archives www.kernel.org (2010-04-20)

[11] Eclipse Development Tools http://www.eclipse.org/ (2010-02-15)

[12] Eclipse C/C++ Development Tool http://www.eclipse.org/cdt/ (2010-02-15)

[13] JUnit community site http://www.junit.org/ (2010-02-15)

[14] Apache Tomcat http://tomcat.apache.org/ (2010-02-15)

[15] Apache Derby http://db.apache.org/derby/ (2010-02-17)

[16]GNU Compiler Collection http://gcc.gnu.org/ (2010-02-17)

[17] TEWS TECHNOLOGIES http://www.tews.com/ (2010-04-17)

[18] Mail conversation Gerrit Hesse, TEWS support Management (2009-08-17)

[19] Java Native Interface Specification http://java.sun.com/j2se/1.5.0/docs/guide/jni/spec/jniTOC.html (2010-02-22)

[20] RT_PREEMPT_HOWTO, RT Linux wiki http://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO (2010-02-28)

[21] Resources for Financial Engineers http://www.sitmo.com/eq/274 (2010-04-23)

[22] Numerical Differentiation, Mathematics Department at Cal State Fullerton http://math.fullerton.edu/mathews/n2003/NumericalDiffMod.html (2010-04-23)

[23] The National Institute of Standards and Technology http://www.itl.nist.gov/div897/sqg/dads/HTML/binarySearch.html (2010-03-21)

Appendix A – Initial document

Thesis Motion Logger 2

Abstract

The Motion Logger 2 is a system for logging incremental encoders connected to motion control systems to determine the behaviour of the motion itself and the motion control system behavior.

System overview

The system is based on a CPCI system with a CPCI encoder measurement card that can count 4 encoder channels at the same time. There are also a analogue input card with 8 differential analogue inputs for 0-10V and a digital input card with 16 inputs 24 VDC.

The CPU is a Intel Pentium M 1.1 Ghz processor.

The logging shall be performed in a interval of 0.5 seconds to 30 seconds and the sampling be performed with a deterministic interval higher than 20 kHz for the encoders.

The system is equipped with a Linux distribution with the Ingo Molnar real time patch that makes Linux processes deterministic.

The analogue I/O can be sampled with a lower frequency.

The digital I/O shall be able to be used as a trigger for starting the sampling fo encoders.

The data shall be stored in a database for future analysis.

The interface from the user shall be web based and the data shall be possible to analys with derivations and low pass filters. This shall be built into the graphical interface.

System modules

The system modules can be partioned into three parts.

The sampling part which shall be written in C or C++ to utilze the real time capabilities of the Linux system. This part also has a Java Native Interface to allow user interaction and to be able to store the data with Java code.

The storage part shall be written in Java and use an embedded derby database. This part of the code has the interface towards the sampling module.

The user interface and business logic for analyzing the data is written as a Java web application. For the full solution the Apache Tomcat is used as an application container.

The code is written in Java, HTML, SVG and Javascript.

User stories

The application is written around the following user stories.

Starting a test

When the user start a test he/she goes to a URL of the sampling device.

The user sets up which data I/O to log and connects it.

The user can see the data I/O in realtime to be able to diagnose the conenctions.

The user feeds in metadata about the test.

The user gives the test a unique name.

Sampling data

The user sets the interval to log during 0.5 to 30 seconds.

The user can define a digital trigger signal.

The user starts the logging.

The user gets notified when the logging is complete.

The data from the logging is displayed to the user in grahical format.

Viewing and analyzing sampled data

Once the logging is completed the user gets notified and the raw data is presented in graphical format.

The user can now define new data that will be mathematical transformed from the raw data or from other transformed data.

The user can choose to derive or to low pass filter the signals.

Storing sampled data

The user can choose to store the sampled data pressing a button in the application window.

Analyzing stored data

The user will be able to select from a pane to the left in the graphical user interface see logged tests and from a tree view see all samples stored from this test. Selecting one or more samples will render them in graphs.

The user can now define new data that will be mathematical transformed from the raw data or from other transformed data.

The user can choose to derive or to low pass filter the signals.

Exporting data

The data showed in a graph can be exported to csv format by right clicking on the graph.

Appendix B – Installation manuals

Configuring hard drives

The physical setup of the hard drives makes it necessary to mount the partitions of the larger magnetic drives. To get the list of all the partitions with size the following command is used:

```
sudo fdisk /dev/sd* -s
/dev/sda: 4013856
/dev/sda1: 3783276
/dev/sda2: 1
/dev/sda5: 224878
/dev/sdb: 97685784
/dev/sdb1: 15631213
/dev/sdb2: 1
/dev/sdb5: 1477948
/dev/sdb6: 80573944
```

From this we can determine that /dev/sdb is the magnetic disk. And the partitions we need to mount is /dev/sdb1 and /dev/sdb6. The /dev/sdb1 is the partition for the home directory and /dev/sdb6 is the storage partition.

Home partition

To mount /dev/sdb1 as the new home directory is a bit tricky with a lot of risk for things going wrong. First it is necessary to create a temporary directory for partition. When the temporary directory is created the partition is then mounted into that directory.

```
sudo mkdir /mnt/motionlogger2
sudo mount -t ext3 /dev/sdb1 /mnt/ motionlogger2
```

The –t option tells mount to look for file system type which is the standard for Linux, ext3.

Next step is to copy the files from the old home directory to the new. An ordinary copy (command: cp) will not be sufficient since the home directory contains hardlinks, softlinks, nested directories and so on. First move to the home directory and then copy the files with:

```
cd /home/motionlogger2
sudo find . -depth -print0 | sudo cpio --null --sparse -pvd /mnt/
motionlogger2/
```

Some of the options for find and cpio needs to be comment though quite a bit of tweaking was needed to get this right. First the –depth is used to get the content of the directories to be processed before the directories themselves. Next is the --null option that makes cpio consider a null character as the end of filename instead of a new line. This is important because some file names in the directory contains new line characters in the filenames.

After copying the files, the next task is to unmount the new home directory

```
sudo umount /mnt/motionlogger2
```

Now make way for the new home by moving the old home directory.

```
sudo mv /home/motionlogger2 /home/motionlogger2old
```

This also could be done by removing the directory directly (command: rm -r) but it is good practice to keep a backup in case something goes wrong.

Since we moved the home directory it does not exist so the directory needs to be created and then just mount the new home.

```
sudo mkdir /home/motionlogger2
sudo mount /dev/sdb6 /home/motionlogger2
```

To make the new location permanent the OS needs to mount the partition on every boot. To do this the following line needs to be added to /etc/fstab.

```
/dev/hda5 /home ext3 nodev,nosuid 0 2
```

Next some permissions need to be updated. This means simply that the user needs to own the home directory and all content in it. The user is named motionlogger2.

```
sudo chmod 644 /home/motionlogger2/.dmrc
sudo chown motionlogger2 /home/motionlogger2/.dmrc
sudo chmod –R 755 /home/motionlogger2/.dmrc
sudo –R motionlogger2 /home/motionlogger2/.dmrc
```

Storage partition

This is a much simpler task than the storage partition. Create the location where the partition should be mounted.

```
sudo mkdir /mnt/storage
```

Now simply add this line to /etc/fstab.

```
/dev/sdb6 /mnt/storage ext3 users,atime,auto,rw,nodev,exec,nosuid 0 0
```

Important arguments here are users and rw. These give all users read and write permission to the partition.

Configure kernel

Kernel configuration for RT support is not a trivial thing and demands a lot of knowledge about the Linux kernel. It involves patching and recompiling the kernel. For this application the standard RT-kernel for Xubuntu 8.04 LTS with a few tweaks is sufficient. The specific kernel version is 2.6.24-24-rt.

RT-patch

The Linux kernel in itself is deterministic but there is a way to make the Linux kernel processes deterministic. By patching the kernel with Ingo Molnar's Linux Real Time Patch this can be achieved.

Options

First of all it is essential to know which options to use and disable. The most important group of options to deal with is SMI's. SMI stand for system management interrupts. The cause of these interrupts is power management hardware on the motherboard. They are the highest priority interrupts in the system. In Linux the priority range is 1-140 but priorities over 99 is reserved for external interrupts such as SMI's. That means that there is no way to preempt the interrupts from inside the kernel. SMI's are really bad when RT is required. The SMI's can cause latencies of several 100's of µs. As we have a cyclic period under 7 µs that is a really big problem. The way to deal with these interrupts is to disable them when compiling the kernel.

There are two types of SMI': APCI and APM. Not all of these are bad even thought the cause latencies. Especially APM options are often essential for dealing with power management on hardware level. Disabling these can lead to severe risk of burning the computer to death. The APCI options however are mostly non safety critical interrupts. All of these options except the pm_timer should be turned off. The pm_timer is needed as a clock source for the high resolution timers.

Other options that needs to be enabled is CONFIG_PREEMPT_RT and High Resolution Timer Support. The first one is the option for enabling the real time support of the kernel. The other is for activating the high resolution timer support. However this will not make any difference if the hardware does not support high resolution timers and the number of platform that supports this is limited.

Compiling the kernel

Now comes the part where all the knowledge and research comes to use. For an online guide about how to do this is the Master Kernel Thread in the Ubuntu forums.

First of all it is necessary to install some Linux utilities that is needed for configuring the kernel. Some of the utilities may already be installed but that is not a problem since the install function just skips them in that case. The code is written in a terminal.

Code:

```
sudo apt-get install build-essential bin86 kernel-package libqt3-headers
libqt3-mt-dev wget libncurses5 libncurses5-dev
```

Next we need to change to a configuration directory. This is just a directory where all source code and the compiled kernels are stored. Also for simplifying command typing later the shell is set as root.

```
cd /usr/src/kernels
sudo -s
```

Next fetch the source code for the kernel. There are several ways to do this but downloading the source code from the GIT repository is the way to go if the desired kernel is an Ubuntu version. If vanilla kernel is needed downloading it from www.kernel.org is the easiest. Vanilla means an unaltered standard release of the kernel.

Then unpack the kernel with

```
tar -xvjf linux-2.6.32.tar.bz2
```

Now remove the link to the old kernel and make the new kernel.

```
rm –rf linux
ln –s /usr/src/linux-2.6.24-24-rt
cd usr/src/linux
```

Now it is time to configure the kernel. This is done by copying the current kernel configuration and then alter the copy. A good idea is to have the Xubuntu Linux RT-kernel installed to get their standard RT-configuration. To install the RT kernel the following command is used.

```
 sudo apt-get install linux-rt.
```

To copy the configuration files:

```
cp /boot/config-$(uname -r) .config && yes "" | make oldconfig
```

Here the uname -r command returns the current kernel version number. To get full details about the kernel the option -a could be used.

Next to configure the kernel use xconfig which is a linux graphical compilation utility. Alternatively menyconfig could be used for a terminal based version.

```
make xconfig
```

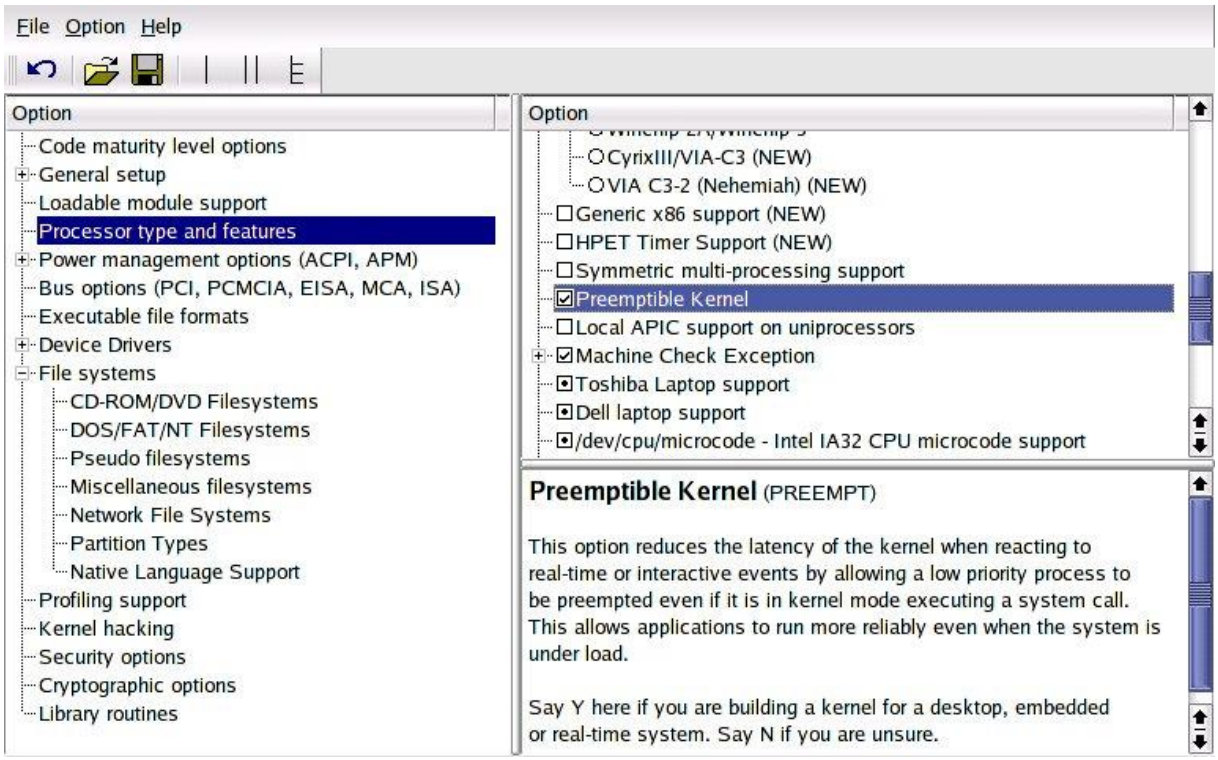

Figure A. Screenshot of xconfig menu.

Now go in under Power management options and make the configurations according to options sections of this guide.

When all configurations are made it is time to compile the kernel.

```
make-kpkg clean
```

```
INSTALL_MOD_STRIP=1 CONCURRENCY_LEVEL=3 make-kpkg --initrd --append-to-
version=-mk kernel_image kernel_headers modules_image
```

Here the append_to_version argument (in this case -mk) can be changed to anything. However the hyphen is not optional.

Now the computer will compile for a couple of hours depending on the speed of the CPU. In this case because of the limited CPU speed it will take about four and a half hour. So a nightly build is recommended.

When the kernel is compiled there should be two new files in the /usr/src. One linux-image-<versionNumber>.deb and one linux-headers-<versionNumber>.deb.

The last step is to install the new kernel in /usr/src.

```
cd ..
dpkg -i linux-image-2.6.24-24-rt.deb
dpkg -i linux-headers-2.6.24-24-rt.deb
```

Now the kernel is installed and ready to use. The new kernel will be loaded on next boot.

Necessary applications

This section describes how to install and configure the necessary applications for development and use of the system.

Eclipse

Eclipse is used as the development environment for the code development. Two version with different configurations were used.

The first one was Eclipse Classic version 3.5. Two plug-ins were used, CDT and JUnit. CDT is the Eclipse part for C and C++ developers and stands for C/C++ Development Tool. JUnit is a unit testing framework for Java. JUnit is already installed as standard in this version of Eclipse.

The second version was Eclipse JEE 3.5.1. JEE is customized version for web development. The CDT plugin were also used in this configuration.

The installation of eclipse is straight forward. Download the .tar.gz archive from the Eclipse homepage and unpack it in the desired directory. It is also possible to download and install Eclipse from the repository but it is not recommended because this is an older version of Eclipse (version 3.2) which is out of date. Unpacking command:

```
cd opt
sudo mkdir eclipse3.5.1
sudo tar xvzf eclipse-jee-galileo-SR1-linux-gtk.tar.gz
```

Installation of the plug-ins is all done within Eclipse. Under the help menu there is an install new software sub menu. This will show the repository menu.

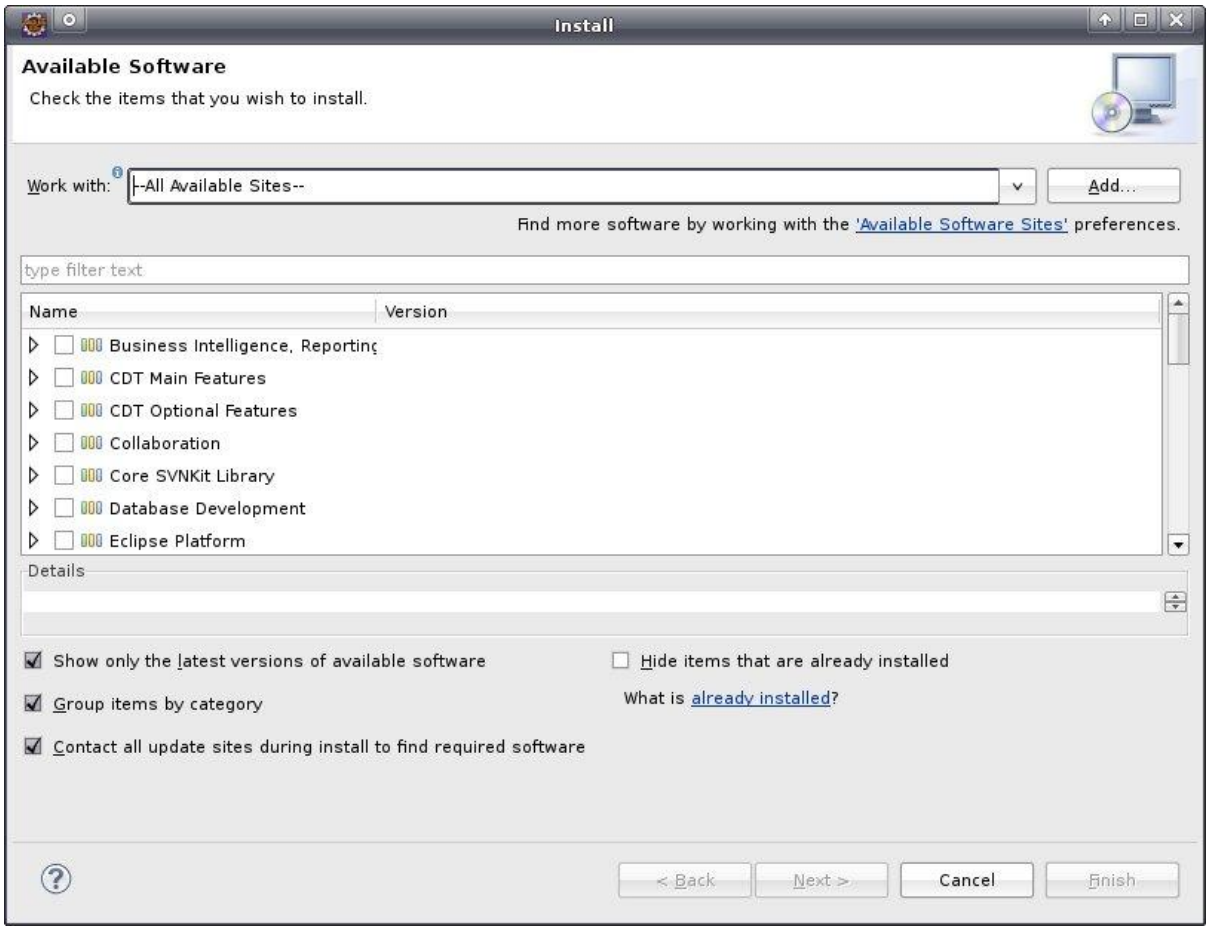

Figure B. Eclipse Install New Software menu.

The location of the CDT repository has to be added separately. At time of writing it is located at http://download.eclipse.org/tools/cdt/releases/galileo.

Apache Tomcat

Apache Tomcat is an open source servlet container. Apache Tomcat provides a web server for Java applications. The exact version used for this application is v6.0.20.

Before installation of Apache Tomcat Java runtime environment needs to be installed. To install Apache Tomcat create the target directory and move to that directory. Target directory here is /opt/ApacheTomcat.

```
sudo mkdir opt/ApacheTomcat
cd opt/ApacheTomcat
```

Then Apache Tomcat needs to be downloaded and extracted.

```
sudo wget http://apache.hoxt.com/tomcat/tomcat-6/v6.0.14/bin/apache-tomcat-6.0.14.tar.gz
sudo tar -xvzf apache-tomcat-6.0.14.tar.gz
```

Apache Tomcat also requires setting of the JAVA_HOME variable in Linux. This is done by adding the following line to /etc/bash.bashrc. There are several other files that this could be put in. The version of Sun's Java is 1.6.0.17.

```
export JAVA_HOME=/usr/lib/jvm/java-6-sun-1.6.0.17/
```

A script for auto starting the server also needs to be created. The default location for startup scripts is /etc/init.d. The script is created by the following commands. The filename is tomcat.

```
cd etc/init.d
sudo gedit tomcat
```

This will open a text editor with the script file.

The script looks as follows

```
# Tomcat auto-start
#
# description: Auto-starts tomcat
# processname: tomcat
# pidfile: /var/run/tomcat.pid

export CATALINA_OPTS="-server -Xmx128m -Xms64m"

case $1 in
start)
        sh /usr/local/tomcat/bin/startup.sh
        ;;
stop)
        sh /usr/local/tomcat/bin/shutdown.sh
        ;;
restart)
        sh /usr/local/tomcat/bin/shutdown.sh
        sh /usr/local/tomcat/bin/startup.sh
        ;;
esac
exit 0
```

In the script the CATALINA_OPTS variable is also set. The CATALINA_OPTS variable is for setting arguments to Apache Tomcat. "-server –Xmx128m –Xms64m" specifies the maximum and initial memory size for the server. Ideally the maximum memory should be larger but the amount of memory for the computer is limited. The script also needs to be made executable.

```
sudo chmod 755 /etc/init.d/tomcat
```

And finally link the script with startup folders.

```
sudo ln -s /etc/init.d/tomcat /etc/rc1.d/K99tomcat
sudo ln -s /etc/init.d/tomcat /etc/rc2.d/S99tomcat
```

This will make Tomcat start automatically when the computer starts. The script also makes it possible to start and stop Tomcat manually from a terminal. Commands for this:

```
sudo etc/init.d/tomcat start
sudo etc/init.d/tomcat stop
sudo etc/init.d/tomcat restart
```

Apache Derby

Apache Derby is an open source relations database. Apache Derby is implemented entirely in Java. The database is needed for storing and easy access to old logging task. Specific version used is 10.5.3.0.

Since there will no direct speed demands for the database the most suitable place for the installation of the database is the storage partition.

```
mkdir mnt/storage/opt/Apache
```

```
cd mnt/storage/opt/Apache
```

To tarball with installation files is downloaded from the Apache Derby homepage (http://db.apache.org/derby/) and put in the created directory. Tarball is the compressed file archive with .tar.gz as file extension. The downloaded tarball is unpacked with

```
tar –xvzf db-derby-10.5.3.0-bin.tar.gz
```

To make the installation complete some Linux environment variables also needs to be set. The following line is added to /etc/bash.bashrc.

```
export DERBY_INSTALL=/mnt/storage/opt/Apache/ db-derby-10.5.3.0-bin/bin
```

Java runtime environment

To execute Java code in Linux we need a Java runtime environment. The installation here is very simple. Just go to Add/remove program under the system menu and search for Java runtime and then mark the one you want to install. However there are two different choices, either OpenJDK Java Runtime or Sun Java 6 Runtime. The best choice here is the Sun Java 6 Runtime simply because later in the application Java native interface will be used and a quick Google search resulted in a lot of problems when using OpenJDK. However both could be installed without any problems but the JAVA_HOME variable needs to point to the chosen installation.

```
export JAVA_HOME=/usr/lib/jvm/java-6-sun-1.6.0.17/
```

GCC

GCC is the GNU Compiler Collection. GCC has compilers for C/C++, Objective-C among others. It also contains the standard libraries for these languages. As this application uses JNI the C compiler is needed. Installation is very simple since it is possible to get it from the Xubuntu repository.

```
sudo apt-get install build-essential
```

A bit counterintuitive maybe but replacing build-essential with just gcc will not do it. It will install the compilers but not the libraries.

Installing Tews drivers

The installation mostly follows the user manuals from Tews technologies for the corresponding cards. The manuals can be downloaded from TEWS's homepage. However the manuals are not entirely correct and some corrections and tweaking is necessary. One more important issue is that before installing the individual card, an IPAC carrier drive needs to be installed.

Kernel version issue

According to the manual and the installation files the cards will work for all 2.6.* Linux kernels. This is not true. After the 2.6.24 distribution some kernel systems were updated and rebuilt. The functions in specific are the device file system interface and the modprobe. The device file system interface structure has been changed. According to a mail conversation with Gerrit Hesse from Tews Support Management Tews have not updated the drivers to support the new structure. However the installation files handling this are fairly separate and Tews have made updates of these files supporting the new file structure in other projects. However there are untested for the TIP116(encoder), TIP600(digital), TIP501(analog) cards. The files are named config.h, tpmodule.h and tpmodule.c. They appear to solve the problem

but when the modprobe function is used to build the drivers into the kernel, the modprobe issues a warning and the installation fails.

```
WARNING: All config files need .conf: /etc/modprobe.d/oss-compat, it will
be ignored in a future release.
```

Some investigations about this error gives that it is a bug in KMOD which modprobe uses. The bug will be fixed in future releases of the kernels but at the time of the installation it is not so, which leaves the only viable option to use an older kernel.

CARRIER

Since the TIP cards are connected to an IPAC carrier board the carrier drives need to be installed before any installation of the card drivers. Before the installation the tarball with all the installation files need to be unpacked.

```
mkdir CARRIER-SW-82
cd CARRIER-SW-82
tar –xzvf CARRIER-SW-82-SRC.tar.gz
```

Before anything more is done the file ipac_carrier.h needs to be put in appropriate directories to make installation possible. Here the user manual is incorrect when it comes to which directories to put it in. The correct directories are /usr/include/ and /usr/src/linux-headers-2.6.24-24-rt/include.

The rest of the installation is straight forward. Commands for installation looks as follows.

```
sudo –s
cd ipac_carrier/class
make install
depmod –q
modprobe carrier_class
modprobe carries_tews_pci
```

TIP116

Installation of the TIP116 driver is similar to that of the carrier. The only major difference is that a device node needs to be created on the file system. The makenode script creates a minor device for the card and the cards can be accessed via device node /dev/tip116_0. Also the tip116.h file needs to be put in the directory /usr/include. This header file is used for application access to the card. The user manual does not mention this step. Commands for installation look as follows.

```
mkdir TIP116-SW-82
cd TIP116-SW-82
tar –xzvf TIP116-SW-82-SRC.tar.gz
sudo –s
cd tip116/class
make install
depmod –q
modprobe tip116drv
sh makenode
```

When the make install command is executed the compiler will issue some warnings about undefined ipac_* symbols. These are normal and can be ignored.

TIP501

Installation of the TIP501 driver is identical to that of the TIP116 except for the filenames and directories. The makenode script creates a minor device for the card and the cards can be

accessed via device node /dev/tip501_0. Also the tip501.h file needs to be put in the directory /usr/include. Commands for installation look as follows.

```
mkdir TIP501-SW-82
cd TIP501-SW-82
tar –xzvf TIP116-SW-82-SRC.tar.gz
sudo –s
cd tip501
make install
depmod –q
modprobe tip116drv
sh makenode
```

When the make install command is executed the compiler will issue some warnings about undefined ipac_* symbols. These are normal and can be ignored.

TIP600

Installation of the TIP600 driver is identical to that of the TIP116 and TIP501 except for the filenames and directories. The makenode script creates a minor device for the card and the cards can be accessed via device node /dev/tip600_0. As for installation of TIP116 the need for putting the tip600.h file in /usr/include is not mentioned in the user manual. Commands for installation look as follows.

```
mkdir TIP600-SW-82
cd TIP600-SW-82
tar –xzvf TIP600-SW-82-SRC.tar.gz
sudo –s
cd tip600
make install
depmod –q
modprobe tip116drv
sh makenode
```

When the make install command is executed the compiler will issue some warning about undefined ipac_* symbols. These are normal and can be ignored.

Startup script

For some reason when the computer restart the device nodes is not in the /dev directory. The reason for this is unknown at the time of writing. But likely it is some configuration issue in the kernel that causes it. However there is a quick fix for it. The fix is to create a script for creating the nodes at startup.

To do this a .sh file tewsmakenode.sh is created and put in /etc/init.d. /etc/init.d is Linux's directory for scripts that run at startup. The following code is put in the tewsmakenode.sh.

```
#!/bin/bash
cd /home/motionlogger2/TIP116-SW-82/tip116
sh makenode
cd /home/motionlogger2/TIP501-SW-82/tip116
sh makenode
cd /home/motionlogger2/TIP600-SW-82/tip116
sh makenode
```

To make the file executable a simple Linux command is issued.

```
sudo chmod +x tewsmakenode.sh
```

Appendix C – Test files

systeMaxPerformanceTest.c

```
#include <stdio.h>
#include <sys/mman.h> // Needed for mlockall()
#include <unistd.h> // needed for sysconf(int name);
#include <malloc.h>
#include <sys/time.h> // needed for getrusage
#include <sys/resource.h> // needed for getrusage
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <sys/ioctl.h>
#include <sys/times.h>
#include <sys/timeb.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <time.h>
#include <sched.h>

#include <tip501.h>
#include <tip600.h>
#include <tip116.h>

#define DEVICE_NAME     "TIP116"
#define MAX_DEVICES     32
#

#define TIP116EXA_VERSION  "1.0.0"

int hDevices[MAX_DEVICES];

#ifndef FALSE
#define FALSE 0
#endif
#ifndef TRUE
#define TRUE !FALSE
#endif

#define MY_PRIORITY (99)

#define MAX_LEN 100

#define SOMESIZE (152*1024*1024)

static void PrintErrorMessage(void);

int main()
{
                // Allocate some memory + variables for loading cards
                int i, page_size;
                char* buffer;
                struct rusage usage;
                int found;
                int hDevices[32];
                char Path[100];
                int foundCount=0;
                char* paths[32];
```

```
        char* currentPath;
found =0;

// TIP116 variables
int             value;
int             result;
        int             hCurrent = 0;
        int             DeviceIndex = 0;
        //int           selnum = 1;
        int             channel = 2;
        T116_COUNTER_BUFFER     countBuf;
        T116_READ_ALL_BUFFER    allBuf;
        T116_COUNTER_RESET_BUFFER   resetBuf;
        T116_COUNTER_CONFIG_BUFFER  countConfBuf;
        T116_INPUT_STATE_BUFFER     inputBuf;
        T116_TIMER_BUFFER       timerBuf;
        T116_TIMER_BUFFER                                       currentTimeBuf;
        T116_TIMER_CONFIG_BUFFER    timerConfBuf;
        T116_GPIO_BUFFER        gpioBuf;
        T116_CONFIG_GPIO_BUFFER     gpioConfBuf;
        T116_EVENT_BUFFER       evBuf;
        int             timeoutVal = 0;
        long            preloadValue = 5000;
        long            compareValue = 0;

memset((void*)&Path,0,100);
memset((void*)&paths,0,32);

        // Now lock all current and future pages from preventing of being paged
        if (mlockall(MCL_CURRENT | MCL_FUTURE ))
        {
                perror("mlockall failed:");
        }
        //printf("mlockall fails since we are not running as root\n");

        // Turn off malloc trimming.
        mallopt (M_TRIM_THRESHOLD, -1);


        // Turn off mmap usage.
        mallopt (M_MMAP_MAX, 0);


        page_size = sysconf(_SC_PAGESIZE);
        buffer = malloc(SOMESIZE);


        // Touch each page in this piece of memory to get it mapped into RAM
        for (i=0; i < SOMESIZE; i+=page_size)
        {
                // Each write to this buffer will generate a pagefault.
                // Once the pagefault is handled a page will be locked in memory and never
                // given back to the system.
                buffer[i] = 0;
                // print the number of major and minor pagefaults this application has triggered
                getrusage(RUSAGE_SELF, &usage);

        }
        free(buffer);
        printf("Major-pagefaults:%d, Minor Pagefaults:%d\n", usage.ru_majflt, usage.ru_minflt);
```

```
		struct sched_param param;
		/* Declare ourself as a real time task */
		param.sched_priority = MY_PRIORITY;
		if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
				perror("sched_setscheduler failed");
				exit(-1);
		}
		printf("RT-priority: %d\n", MY_PRIORITY);


		/*
		 * Clock_resolutions test
		 */
		struct timespec	clock_resolution;
		int stat;
		stat = clock_getres(CLOCK_REALTIME, &clock_resolution);
		printf("Clock resolution is %d seconds, %ld nanoseconds\n",
					clock_resolution.tv_sec, clock_resolution.tv_nsec);

		// finding the devices!!!
  found = FALSE;
		for (i=0; i<MAX_DEVICES; i++)
		{

				sprintf(Path, "/dev/tip116_%0d", i);
				hDevices[i] = open(Path, O_RDWR);

				if (hDevices[i] >= 0)
				{
					found = TRUE;
					printf("    %s Device found --> %s\n\n", DEVICE_NAME,
Path);
				}
		}
		if (!found)
		{
				printf("\n\nNo %s Devices found. Exit Example Application...\n",
DEVICE_NAME);
				exit(1);
		}

		// Opening TIP116 card
		for (i=0; i<MAX_DEVICES; i++)
		{
				if (hDevices[i] >= 0)
				{
					DeviceIndex = i;
					hCurrent = hDevices[DeviceIndex];
					printf("\n\nOpen %s_%d for I/O...\n\n", DEVICE_NAME, i);
					break;
				}
		}

		/*
		 *  Implemetation of external timer
		 */
		timerConfBuf.preloadVal = 40000;
		timerConfBuf.prescaler = 1;
```

```
timerConfBuf.enable = TRUE;

result = ioctl(hCurrent, T116_IOCS_CONFIG_TIMER, &timerConfBuf);
if(result >= 0){
        printf(" Timer config OK at 8MHz");
}else{
        printf("Timer config failed  -->  Error = %d\n", errno);
        PrintErrorMessage();
}

/*
 **  Setting TIP116 values in a buffer
 */
// setting  Quadrature mode
countConfBuf.inputMode = T116_QUADCOUNT;
// No specialmode
countConfBuf.specMode = T116_SPECMODENO;
// No z-control
countConfBuf.ZControl = T116_SPECMODENO;
// Quadcontrol = 4x
countConfBuf.QuadControl = T116_4X;
//XYZpolarity = low
countConfBuf.XYZPolarity = 0;
countConfBuf.XYZPolarity |= T116_XLOWACTIV;
countConfBuf.XYZPolarity |= T116_YLOWACTIV;
countConfBuf.XYZPolarity |= T116_ZLOWACTIV;
// Set channel number
countConfBuf.channel = channel;

/*
 * Call to Card with buffer address
 */
result = ioctl(hCurrent, T116_IOCS_CONFIG_COUNTER, &countConfBuf);
if (result >= 0)
{
        printf("OK\n\n");
}
else
{
        printf("failed  -->  Error = %d\n", errno);
        PrintErrorMessage();
}
/*
 ** Preload counter with value
 */

printf("\nSet counter preload value: 5000 ... ");
countBuf.channel = channel;
countBuf.data = 5000;
countBuf.flags = T116_IMMEDIATE_PRELOAD;

result = ioctl(hCurrent, T116_IOCS_SET_PRELOAD, &countBuf);
if (result >= 0)
{
        printf("OK\n\n");
}
else
{
printf("failed  -->  Error = %d\n", errno);
PrintErrorMessage();
```

```
}

long *pStore;
pStore =(long*)malloc((4000000+1) *sizeof(*pStore));
if( pStore == NULL){
                printf("not enough memory\n");
                return 1;
}

unsigned short *timeCount;
timeCount =(unsigned short*)malloc((4000000+1)* sizeof(*timeCount));
if( timeCount == NULL){
                printf("not enough memory\n");
                return 1;
}
/*
 ** Reading and storing values in a vector
 */
for(i=0;i<4000000;i++)
{
                result = ioctl(hCurrent, T116_IOCG_READ_COUNTER, &countBuf);
                if(result >= 0){
                                pStore[i] = countBuf.data;
                }
                else
                {
                                printf("ReadFunction failed  -->  Error = %d\n", errno);
                                PrintErrorMessage();
                                break;
                }
                result = ioctl(hCurrent, T116_IOCG_READ_TIMER, &timerBuf);
                if(result >= 0){
                                timeCount[i] = timerBuf.value;
                }
                else
                {
                                printf("TimerCount failed  -->  Error = %d\n", errno);
                                PrintErrorMessage();
                                break;
                }
}

int badcount = 0;
int shifts = 0;
int worst = 0;
long total = 0;
float average = 0;

for(i=0; i<3999999;i++){
                if ((timeCount[i]-timeCount[i+1]) >100){
                                badcount++;
                                if((timeCount[i]-timeCount[i+1])> worst){
                                                worst = timeCount[i]-timeCount[i+1];
                                }
                }
                if ((timeCount[i]-timeCount[i+1]) >0){
                                total = total + (timeCount[i]-timeCount[i+1]);
                }else{
                                total = total + (timeCount[i]+ (40000-timeCount[i+1]));
                }
```

```
                        if((timeCount[i]-timeCount[i+1]) <0){
                                shifts++;
                        }
                }
                average = (total/4000000);
                for(i=0; i<600;i++)
                {
                        printf("TimeInterval number %d  from timeCount: %d\n", i, timeCount[i]-
timeCount[i+1]);
                }
                printf("Number of interval over 100 clockticks: %d\n", badcount);
                printf("Number of clock transitions from 0 to 50000: %d\n", shifts);
                printf("Worst interval i clocktics: %d\n", worst);
                printf("Worst interval i micros: %f\n", (worst*0,125));
                printf("Average number clock ticks between reads %f\n", average);
                printf("Average time in micros between reads %f\n", (average*0,125));

                free(pStore);
                free(timeCount);
                //Closing TIP116 card
                for (i=0; i<MAX_DEVICES; i++)
                {
                        if (hDevices[i] >= 0)
    {
                                close(hDevices[i]);
    }
                }

                printf("TIP116 Device closed\n\n");

                printf("program finished");

                return 0;
}

static void PrintErrorMessage(void)

{
   printf("%s\n\n", strerror (errno));
}
```

testRunAllCounters.c

```
#include <stdio.h>
#include <sys/mman.h> // Needed for mlockall()
#include <unistd.h> // needed for sysconf(int name);
#include <malloc.h>
#include <sys/time.h> // needed for getrusage
#include <sys/resource.h> // needed for getrusage
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <sys/ioctl.h>
#include <sys/times.h>
#include <sys/timeb.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <time.h>
#include <sched.h>
```

```
#include <tip501.h>
#include <tip600.h>
#include <tip116.h>

#define DEVICE_NAME     "TIP116"
#define MAX_DEVICES     32
#

#define TIP116EXA_VERSION  "1.0.0"

int hDevices[MAX_DEVICES];

#ifndef FALSE
#define FALSE 0
#endif
#ifndef TRUE
#define TRUE !FALSE
#endif

#define MY_PRIORITY (79)

#define MAX_LEN 100

#define SOMESIZE (56*1024*1024)

static void PrintErrorMessage(void);

int main()
{
                // Allocate some memory + variables for loading cards
                int i, page_size;
                char* buffer;
                struct rusage usage;
                int found;
                int hDevices[32];
                char Path[100];
                int loggNo = 1500000;
                char* paths[32];
  found =0;

  // TIP116 variables
  int               result;
                int             hCurrent = 0;
                int             DeviceIndex = 0;
                int             channel = 2;
                T116_COUNTER_BUFFER       countBuf;
                T116_READ_ALL_BUFFER      allBuf;
                T116_COUNTER_CONFIG_BUFFER  countConfBuf;
                T116_TIMER_BUFFER         timerBuf;
                T116_TIMER_CONFIG_BUFFER    timerConfBuf;


  memset((void*)&Path,0,100);
  memset((void*)&paths,0,32);

                // Now lock all current and future pages from preventing of being paged
                if (mlockall(MCL_CURRENT | MCL_FUTURE ))
                {
                                perror("mlockall failed:");
```

```
		}
		//printf("mlockall fails since we are not running as root\n");

		page_size = sysconf(_SC_PAGESIZE);
		buffer = malloc(SOMESIZE);

		// Touch each page in this piece of memory to get it mapped into RAM
		for (i=0; i < SOMESIZE; i+=page_size)
		{
			// Each write to this buffer will generate a pagefault.
			// Once the pagefault is handled a page will be locked in memory and never
			// given back to the system.
			buffer[i] = 0;
			// print the number of major and minor pagefaults this application has triggered
			getrusage(RUSAGE_SELF, &usage);

		}
		printf("Major-pagefaults:%d, Minor Pagefaults:%d\n", usage.ru_majflt, usage.ru_minflt);


		struct sched_param param;
		/* Declare ourself as a real time task */
		param.sched_priority = MY_PRIORITY;
		if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
			perror("sched_setscheduler failed");
			exit(-1);
		}
		printf("RT-priority: %d\n", MY_PRIORITY);


		/*
		 * Clock_resolutions test
		 */
		struct timespec	clock_resolution;
		int stat;
		stat = clock_getres(CLOCK_REALTIME, &clock_resolution);
		printf("Clock resolution is %d seconds, %ld nanoseconds\n",
					clock_resolution.tv_sec, clock_resolution.tv_nsec);

		// finding the devices!!!
  found = FALSE;
		for (i=0; i<MAX_DEVICES; i++)
		{

			sprintf(Path, "/dev/tip116_%0d", i);
			hDevices[i] = open(Path, O_RDWR);

			if (hDevices[i] >= 0)
			{
				found = TRUE;
				printf("    %s Device found --> %s\n\n", DEVICE_NAME,
Path);
			}
		}
		if (!found)
		{
			printf("\n\nNo %s Devices found. Exit Example Application...\n",
DEVICE_NAME);
			exit(1);
		}
```

```
// Opening TIP116 card
for (i=0; i<MAX_DEVICES; i++)
{
		if (hDevices[i] >= 0)
		{
				DeviceIndex = i;
				hCurrent = hDevices[DeviceIndex];
				printf("\n\nOpen %s_%d for I/O...\n\n", DEVICE_NAME, i);
				break;
		}
}

/*
 *  Implemetation of external timer
 */
timerConfBuf.preloadVal = 40000;
timerConfBuf.prescaler = 1;
timerConfBuf.enable = TRUE;

result = ioctl(hCurrent, T116_IOCS_CONFIG_TIMER, &timerConfBuf);
if(result >= 0){
		printf(" Timer config OK at 8MHz");
}else{
		printf("Timer config failed  -->  Error = %d\n", errno);
		PrintErrorMessage();
}

/*
 **  Setting TIP116 values in a buffer
 */
// setting  Quadrature mode
countConfBuf.inputMode = T116_QUADCOUNT;
// No specialmode
countConfBuf.specMode = T116_SPECMODENO;
// No z-control
countConfBuf.ZControl = T116_SPECMODENO;
// Quadcontrol = 4x
countConfBuf.QuadControl = T116_4X;
//XYZpolarity = low
countConfBuf.XYZPolarity = 0;
countConfBuf.XYZPolarity |= T116_XLOWACTIV;
countConfBuf.XYZPolarity |= T116_YLOWACTIV;
countConfBuf.XYZPolarity |= T116_ZLOWACTIV;
// Set channel number
countConfBuf.channel = channel;

/*
 * Call to Card with buffer address
 */
for(i=1; i<=4; i++)
{
		countConfBuf.channel = i;
		result = ioctl(hCurrent, T116_IOCS_CONFIG_COUNTER, &countConfBuf);
		if (result >= 0)
		{
				printf("OK\n\n");
		}
		else
		{
```

```
                        printf("failed  -->  Error = %d\n", errno);
                        PrintErrorMessage();
                }
                printf("\nSet counter preload value: 5000 ... ");
                countBuf.channel = i;
                countBuf.data = 5000+i;
                countBuf.flags = T116_IMMEDIATE_PRELOAD;

                result = ioctl(hCurrent, T116_IOCS_SET_PRELOAD, &countBuf);
                if (result >= 0)
                {
                        printf("OK\n\n");
                }
                else
                {
                        printf("failed  -->  Error = %d\n", errno);
                        PrintErrorMessage();
                }
}

long **pStore;
pStore =(long**) malloc(4 * sizeof(long*));
if(pStore == NULL)
{
                printf("not enough memory\n");
                return 1;
}
for(i = 0; i < 4; i++)
{
                pStore[i] =(long*) malloc((loggNo+1) * sizeof(long));
                if(pStore[i] == NULL)
                {
                        printf("not enough memory\n");
                        return 1;
                }
}



unsigned short *timeCount;
timeCount =(unsigned short*)malloc((loggNo+1)* sizeof(*timeCount));
if( timeCount == NULL){
                printf("not enough memory\n");
                return 1;
}
/*
 ** Reading and storing values in a vector
 */
for(i=0;i<loggNo;i++)
{
                result = ioctl(hCurrent, T116_IOCG_READ_ALL_COUNTER, &allBuf);
                if(result >= 0){
                        pStore[0][i] = allBuf.data[0];
                        pStore[1][i] = allBuf.data[1];
                        pStore[2][i] = allBuf.data[2];
                        pStore[3][i] = allBuf.data[3];
                }
                else
                {
                        printf("ReadFunction failed  -->  Error = %d\n", errno);
```

```
                                        PrintErrorMessage();
                                        break;
                                }
                                result = ioctl(hCurrent, T116_IOCG_READ_TIMER, &timerBuf);
                                if(result >= 0){
                                        timeCount[i] = timerBuf.value;
                                }
                                else
                                {
                                        printf("TimerCount failed  -->  Error = %d\n", errno);
                                        PrintErrorMessage();
                                        break;
                                }
                }

                int badcount = 0;
                int shifts = 0;
                int worst = 0;
                long total = 0;
                float average = 0;

                for(i=0; i<loggNo;i++){
                                if ((timeCount[i]-timeCount[i+1]) >215){
                                        badcount++;
                                        if((timeCount[i]-timeCount[i+1])> worst){
                                                worst = timeCount[i]-timeCount[i+1];
                                        }
                                }
                                if ((timeCount[i]-timeCount[i+1]) >0){
                                        total = total + (timeCount[i]-timeCount[i+1]);
                                }else{
                                        total = total + (timeCount[i]+ (40000-timeCount[i+1]));
                                }
                                if((timeCount[i]-timeCount[i+1]) <0){
                                        shifts++;
                                }
                }
                average = (total/loggNo);
                for(i=0; i<100;i++)
                {
                                printf("TimeInterval number %d  from timeCount: %d\n", i, timeCount[i]-
timeCount[i+1]);
                                printf("Value from loggmatrice: channel 1: %ld\n", pStore[0][i]);
                }
                printf("\n\nNumber of logging values: %d\n",loggNo);
                printf("Number of clock transitions from 0 to 40000: %d\n", shifts);
                printf("Time take for test in seconds : %f\n",((float) (shifts*0.005)));
                printf("Number of interval over 215 clockticks: %d\n", badcount);
                printf("Worst interval i clocktics: %d\n", worst);
                printf("Worst interval i micros: %f\n",(float) (worst*0.125));
                printf("Average number clock ticks between reads %f\n", average);
                printf("Average time in micros between reads %f\n",(float) (average*0.125));
                printf("Logging frequency %f\n\n",(float) 1/(average*0.000000125));

                free(pStore);
                free(timeCount);
                free(buffer);
                //Closing TIP116 card
                for (i=0; i<MAX_DEVICES; i++)
                {
```

```
                                    if (hDevices[i] >= 0)
    {
                                                close(hDevices[i]);
    }
                }

                printf("TIP116 Device closed\n\n");

                printf("program finished");

                return 0;
}

static void PrintErrorMessage(void)

{
   printf("%s\n\n", strerror (errno));
}
```

testRunAnalogRead.c

```
#include <stdio.h>
#include <sys/mman.h> // Needed for mlockall()
#include <unistd.h> // needed for sysconf(int name);
#include <malloc.h>
#include <sys/time.h> // needed for getrusage
#include <sys/resource.h> // needed for getrusage
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <sys/ioctl.h>
#include <sys/times.h>
#include <sys/timeb.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <time.h>
#include <sched.h>

#include <tip501.h>
#include <tip600.h>
#include <tip116.h>

#define DEVICE_NAME     "TIP116"
#define MAX_DEVICES     32
#

#define TIP116EXA_VERSION  "1.0.0"

int hDevices[MAX_DEVICES];

#ifndef FALSE
#define FALSE 0
#endif
#ifndef TRUE
#define TRUE !FALSE
#endif

#define MY_PRIORITY (79)
```

```
#define MAX_LEN 100

#define SOMESIZE (56*1024*1024)

static void PrintErrorMessage(void);

int main()
{
		// Allocate some memory + variables for loading cards
		int i, page_size;
		char* buffer;
		struct rusage usage;
		int found;
		int hDevices[32];
		char Path[100];
		int loggNo = 1500000;
		char* paths[32];
  found =0;

  // TIP116 variables
  int		result;
		int		hCurrent = 0;
		int		DeviceIndex = 0;
		int		channel = 2;
		T116_COUNTER_BUFFER	countBuf;
		T116_READ_ALL_BUFFER	allBuf;
		T116_COUNTER_CONFIG_BUFFER	countConfBuf;
		T116_TIMER_BUFFER	timerBuf;
		T116_TIMER_CONFIG_BUFFER	timerConfBuf;


  memset((void*)&Path,0,100);
  memset((void*)&paths,0,32);

		// Now lock all current and future pages from preventing of being paged
		if (mlockall(MCL_CURRENT | MCL_FUTURE ))
		{
			perror("mlockall failed:");
		}
		//printf("mlockall fails since we are not running as root\n");

		page_size = sysconf(_SC_PAGESIZE);
		buffer = malloc(SOMESIZE);

		// Touch each page in this piece of memory to get it mapped into RAM
		for (i=0; i < SOMESIZE; i+=page_size)
		{
			// Each write to this buffer will generate a pagefault.
			// Once the pagefault is handled a page will be locked in memory and never
			// given back to the system.
			buffer[i] = 0;
			// print the number of major and minor pagefaults this application has triggered
			getrusage(RUSAGE_SELF, &usage);

		}
		printf("Major-pagefaults:%d, Minor Pagefaults:%d\n", usage.ru_majflt, usage.ru_minflt);


		struct sched_param param;
```

```
		/* Declare ourself as a real time task */
		param.sched_priority = MY_PRIORITY;
		if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
			perror("sched_setscheduler failed");
			exit(-1);
		}
		printf("RT-priority: %d\n", MY_PRIORITY);


		/*
		 * Clock_resolutions test
		 */
		struct timespec	clock_resolution;
		int stat;
		stat = clock_getres(CLOCK_REALTIME, &clock_resolution);
		printf("Clock resolution is %d seconds, %ld nanoseconds\n",
					clock_resolution.tv_sec, clock_resolution.tv_nsec);

		// finding the devices!!!
  found = FALSE;
		for (i=0; i<MAX_DEVICES; i++)
		{

			sprintf(Path, "/dev/tip116_%0d", i);
			hDevices[i] = open(Path, O_RDWR);

			if (hDevices[i] >= 0)
			{
				found = TRUE;
				printf("    %s Device found --> %s\n\n", DEVICE_NAME,
Path);
			}
		}
		if (!found)
		{
			printf("\n\nNo %s Devices found. Exit Example Application...\n",
DEVICE_NAME);
			exit(1);
		}

		// Opening TIP116 card
		for (i=0; i<MAX_DEVICES; i++)
		{
			if (hDevices[i] >= 0)
			{
				DeviceIndex = i;
				hCurrent = hDevices[DeviceIndex];
				printf("\n\nOpen %s_%d for I/O...\n\n", DEVICE_NAME, i);
				break;
			}
		}

		/*
		 *  Implemetation of external timer
		 */
		timerConfBuf.preloadVal = 40000;
		timerConfBuf.prescaler = 1;
		timerConfBuf.enable = TRUE;

		result = ioctl(hCurrent, T116_IOCS_CONFIG_TIMER, &timerConfBuf);
```

```
if(result >= 0){
		printf(" Timer config OK at 8MHz");
}else{
		printf("Timer config failed  -->  Error = %d\n", errno);
		PrintErrorMessage();
}

/*
 **  Setting TIP116 values in a buffer
 */
// setting  Quadrature mode
countConfBuf.inputMode = T116_QUADCOUNT;
// No specialmode
countConfBuf.specMode = T116_SPECMODENO;
// No z-control
countConfBuf.ZControl = T116_SPECMODENO;
// Quadcontrol = 4x
countConfBuf.QuadControl = T116_4X;
//XYZpolarity = low
countConfBuf.XYZPolarity = 0;
countConfBuf.XYZPolarity |= T116_XLOWACTIV;
countConfBuf.XYZPolarity |= T116_YLOWACTIV;
countConfBuf.XYZPolarity |= T116_ZLOWACTIV;
// Set channel number
countConfBuf.channel = channel;

/*
 * Call to Card with buffer address
 */
for(i=1; i<=4; i++)
{
		countConfBuf.channel = i;
		result = ioctl(hCurrent, T116_IOCS_CONFIG_COUNTER, &countConfBuf);
		if (result >= 0)
		{
			printf("OK\n\n");
		}
		else
		{
			printf("failed  -->  Error = %d\n", errno);
			PrintErrorMessage();
		}
		printf("\nSet counter preload value: 5000 ... ");
		countBuf.channel = i;
		countBuf.data = 5000+i;
		countBuf.flags = T116_IMMEDIATE_PRELOAD;

		result = ioctl(hCurrent, T116_IOCS_SET_PRELOAD, &countBuf);
		if (result >= 0)
		{
			printf("OK\n\n");
		}
		else
		{
			printf("failed  -->  Error = %d\n", errno);
			PrintErrorMessage();
		}
}

long **pStore;
```

```
pStore =(long**) malloc(4 * sizeof(long*));
if(pStore == NULL)
{
		printf("not enough memory\n");
		return 1;
}
for(i = 0; i < 4; i++)
{
		pStore[i] =(long*) malloc((loggNo+1) * sizeof(long));
		if(pStore[i] == NULL)
		{
			printf("not enough memory\n");
			return 1;
		}
}



unsigned short *timeCount;
timeCount =(unsigned short*)malloc((loggNo+1)* sizeof(*timeCount));
if( timeCount == NULL){
		printf("not enough memory\n");
		return 1;
}
/*
 ** Reading and storing values in a vector
 */
for(i=0;i<loggNo;i++)
{
		result = ioctl(hCurrent, T116_IOCG_READ_ALL_COUNTER, &allBuf);
		if(result >= 0){
			pStore[0][i] = allBuf.data[0];
			pStore[1][i] = allBuf.data[1];
			pStore[2][i] = allBuf.data[2];
			pStore[3][i] = allBuf.data[3];
		}
		else
		{
			printf("ReadFunction failed  -->  Error = %d\n", errno);
			PrintErrorMessage();
			break;
		}
		result = ioctl(hCurrent, T116_IOCG_READ_TIMER, &timerBuf);
		if(result >= 0){
			timeCount[i] = timerBuf.value;
		}
		else
		{
			printf("TimerCount failed  -->  Error = %d\n", errno);
			PrintErrorMessage();
			break;
		}
}

int badcount = 0;
int shifts = 0;
int worst = 0;
long total = 0;
float average = 0;
```

```
		for(i=0; i<loggNo;i++){
			if ((timeCount[i]-timeCount[i+1]) >215){
				badcount++;
				if((timeCount[i]-timeCount[i+1])> worst){
					worst = timeCount[i]-timeCount[i+1];
				}
			}
			if ((timeCount[i]-timeCount[i+1]) >0){
				total = total + (timeCount[i]-timeCount[i+1]);
			}else{
				total = total + (timeCount[i]+ (40000-timeCount[i+1]));
			}
			if((timeCount[i]-timeCount[i+1]) <0){
				shifts++;
			}
		}
		average = (total/loggNo);
		for(i=0; i<100;i++)
		{
			printf("TimeInterval number %d  from timeCount: %d\n", i, timeCount[i]-
timeCount[i+1]);
			printf("Value from loggmatrice: channel 1: %ld\n", pStore[0][i]);
		}
		printf("\n\nNumber of logging values: %d\n",loggNo);
		printf("Number of clock transitions from 0 to 40000: %d\n", shifts);
		printf("Time take for test in seconds : %f\n",((float) (shifts*0.005)));
		printf("Number of interval over 215 clockticks: %d\n", badcount);
		printf("Worst interval i clocktics: %d\n", worst);
		printf("Worst interval i micros: %f\n",(float) (worst*0.125));
		printf("Average number clock ticks between reads %f\n", average);
		printf("Average time in micros between reads %f\n",(float) (average*0.125));
		printf("Logging frequency %f\n\n",(float) 1/(average*0.000000125));

		free(pStore);
		free(timeCount);
		free(buffer);
		//Closing TIP116 card
		for (i=0; i<MAX_DEVICES; i++)
		{
			if (hDevices[i] >= 0)
    {
				close(hDevices[i]);

    }
		}

		printf("TIP116 Device closed\n\n");

		printf("program finished");

		return 0;
}

static void PrintErrorMessage(void)

{
   printf("%s\n\n", strerror (errno));
}
```

testRunTwoAnalog.c

```
#include <stdio.h>
#include <sys/mman.h> // Needed for mlockall()
#include <unistd.h> // needed for sysconf(int name);
#include <malloc.h>
#include <sys/time.h> // needed for getrusage
#include <sys/resource.h> // needed for getrusage
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <sys/ioctl.h>
#include <sys/times.h>
#include <sys/timeb.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <time.h>
#include <sched.h>

#include <tip501.h>
#include <tip600.h>
#include <tip116.h>

#define DEVICE_NAME     "TIP116"
#define DEVICE_NAME501     "TIP501"
#define MAX_DEVICES     32
#

#define TIP116EXA_VERSION  "1.0.0"
#define TIP501EXA_VERSION  "1.2.0"

int hDevices[MAX_DEVICES];
int hTIP501Devices[MAX_DEVICES];

#ifndef FALSE
#define FALSE 0
#endif
#ifndef TRUE
#define TRUE !FALSE
#endif

#define MY_PRIORITY (79)

#define MAX_LEN 100

#define SOMESIZE (56*1024*1024)

static void PrintErrorMessage(void);

int main()
{
                    // Allocate some memory + variables for loading cards
                    int i, page_size;
                    char* buffer;
                    struct rusage usage;
                    int found;
                    int hDevices[32];
                    char Path[100];
                    char* paths[32];
   found =0;
```

```
// TIP variables
int                 result;
int
                loggNo = 400000;
                int                 hCurrent = 0;
                int                             NumBytes;
                int
                tip501current;
                int                 DeviceIndex = 0;
                T116_TIMER_BUFFER       timerBuf;
                T116_TIMER_CONFIG_BUFFER    timerConfBuf;
                T501_READ_BUFFER                                        rdBuf;
                T501_READ_BUFFER                                        rdBuf2;
                T501_INFO_BUFFER                                        infoBuf;

memset((void*)&Path,0,100);
memset((void*)&paths,0,32);

                // Now lock all current and future pages from preventing of being paged
                if (mlockall(MCL_CURRENT | MCL_FUTURE ))
                {
                                perror("mlockall failed:");
                }
                //printf("mlockall fails since we are not running as root\n");

                page_size = sysconf(_SC_PAGESIZE);
                buffer = malloc(SOMESIZE);

                // Touch each page in this piece of memory to get it mapped into RAM
                for (i=0; i < SOMESIZE; i+=page_size)
                {
                                // Each write to this buffer will generate a pagefault.
                                // Once the pagefault is handled a page will be locked in memory and never
                                // given back to the system.
                                buffer[i] = 0;
                                // print the number of major and minor pagefaults this application has triggered
                                getrusage(RUSAGE_SELF, &usage);

                }
                printf("Major-pagefaults:%ld, Minor Pagefaults:%ld\n", usage.ru_majflt, usage.ru_minflt);

                struct sched_param param;
                /* Declare ourself as a real time task */
                param.sched_priority = MY_PRIORITY;
                if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                                perror("sched_setscheduler failed");
                                exit(-1);
                }
                printf("RT-priority: %d\n", MY_PRIORITY);


                // finding TIP116 device
found = FALSE;
                for (i=0; i<MAX_DEVICES; i++)
                {
                                sprintf(Path, "/dev/tip116_%0d", i);
                                hDevices[i] = open(Path, O_RDWR);

                                if (hDevices[i] >= 0)
                                {
```

```
                                        found = TRUE;
                                        printf("    %s Device found --> %s\n\n", DEVICE_NAME,
Path);
                        }
                }
                if (!found)
                {
                        printf("\n\nNo %s Devices found. Exit Example Application...\n",
DEVICE_NAME);
                        exit(1);
                }
                // Choose first found TIP116 to hCurrent;
                for (i=0; i<MAX_DEVICES; i++)
                {
                        if (hDevices[i] >= 0)
                        {
                                DeviceIndex = i;
                                hCurrent = hDevices[DeviceIndex];
                                printf("\n\nOpen %s_%d for I/O...\n\n", DEVICE_NAME, i);
                                break;
                        }
                }
                memset((void*) &hDevices,0,(32*sizeof(int)));
                memset((void*)&Path,0,100);
                memset((void*)&paths,0,32);

                found = FALSE;
                for (i = 0; i < MAX_DEVICES; i++)
                {
                        sprintf(Path, "/dev/tip501_%0d", i);
                        hTIP501Devices[i] = open(Path, O_RDWR);

                        if (hTIP501Devices[i] >= 0)
                        {
                                found = TRUE;
                                printf("    %s Device found --> %s\n\n", DEVICE_NAME501,
Path);
                        }
                }

                if (!found)
                {
                        printf("\n\nNo %s Devices found. Exit Example Application...\n\n",
DEVICE_NAME501);
                        exit(1);
                }
                 /*
                  **  Select the first TIP600 device we found as current I/O device
                  */
                for (i=0; i<MAX_DEVICES; i++)
                {
                        if (hTIP501Devices[i] >= 0)
                        {
                                DeviceIndex = i;
                                tip501current = hTIP501Devices[DeviceIndex];
                                printf("\n\nOpen %s_%d for I/O...\n\n", DEVICE_NAME501,
i);
                                break;
                        }
                }
```

```
                /*
                 *  Implemetation of external timer
                 */
                timerConfBuf.preloadVal = 40000;
                timerConfBuf.prescaler = 1;
                timerConfBuf.enable = TRUE;

                result = ioctl(hCurrent, T116_IOCS_CONFIG_TIMER, &timerConfBuf);
                if(result >= 0){
                                printf("Timer config OK at 8MHz\n\n");
                }else{
                                printf("Timer config failed  -->  Error = %d\n\n", errno);
                                PrintErrorMessage();
                }

                //information about the variant for TIP501
result = ioctl(tip501current, T501_IOCG_INFO, &infoBuf);
if (result < 0) {
   /* This error should never occur if open was successful, but who knows... */
   printf("\nRead device information failed -->  Error = %d\n", errno);
   PrintErrorMessage();
   exit(1);
}


                int *pStore;
                pStore =(int*)malloc((loggNo+1) * sizeof(*pStore));
                if(pStore == NULL)
                {
                                printf("not enough memory\n");
                                return 1;
                }
                memset(pStore,0,sizeof((loggNo+1)*sizeof(*pStore)));

                int *pStore2;
                pStore2 =(int*)malloc((loggNo+1) * sizeof(*pStore2));
                if(pStore2 == NULL)
                {
                                printf("not enough memory\n");
                                return 1;
                }
                memset(pStore2,0,sizeof((loggNo+1)*sizeof(*pStore2)));

                unsigned short *timeCount;
                timeCount =(unsigned short*)malloc((loggNo+1)* sizeof(*timeCount));
                if( timeCount == NULL){
                                printf("not enough memory\n");
                                return 1;
                }
                memset(timeCount,0,sizeof((loggNo+1)*sizeof(unsigned short)));

                rdBuf.chan = 1;
                rdBuf.gain = 1;
                rdBuf.flags = 0;
                rdBuf.data = 2;

                rdBuf2.chan = 2;
                rdBuf2.gain = 1;
                rdBuf2.flags = 0;
```

```
rdBuf2.data = 2;

/*
 ** Reading and storing values in a vector
 */
for(i=0;i<loggNo;i++)
{
		NumBytes = read(tip501current, &rdBuf, sizeof(rdBuf));
		if (NumBytes > 0){
			pStore[i] = rdBuf.data;
		}else {
			printf("\nRead input lines failed -->  Error = %d\n", errno );
			PrintErrorMessage();
		}
		NumBytes = read(tip501current, &rdBuf2, sizeof(rdBuf2));
		if (NumBytes > 0){
			pStore[i] = rdBuf2.data;
		}else {
			printf("\nRead input lines failed -->  Error = %d\n", errno );
			PrintErrorMessage();
		}

		result = ioctl(hCurrent, T116_IOCG_READ_TIMER, &timerBuf);
		if(result >= 0){
			timeCount[i] = timerBuf.value;
		}else{
			printf("TimerCount failed  -->  Error = %d\n", errno);
			PrintErrorMessage();
			break;
		}
}

int badcount = 0;
int shifts = 0;
int worst = 0;
long total = 0;
float average = 0;

for(i=0; i<loggNo-1;i++){
		if ((timeCount[i]-timeCount[i+1]) >600){
			badcount++;
			if((timeCount[i]-timeCount[i+1])> worst){
				worst = timeCount[i]-timeCount[i+1];
			}
		}
		if ((timeCount[i]-timeCount[i+1]) >0){
			total = total + (timeCount[i]-timeCount[i+1]);
		}else{
			total = total + (timeCount[i]+ (40000-timeCount[i+1]));
		}
		if((timeCount[i]-timeCount[i+1]) <0){
			shifts++;
		}
}
average = (total/loggNo);
for(i=0; i<10;i++)
{
		printf("TimeInterval number %d  from timeCount: %d\n", i, timeCount[i]-
timeCount[i+1]);
		printf("Value from loggVector: %d\n", pStore[i]);
```

```
                        printf("Value from loggVector: %d\n", pStore2[i]);
                }
                printf("\nRunAnalogTwoChannels\n");
                printf("\nNumber of logging values: %d\n",loggNo);
                printf("Number of clock transitions from 0 to 40000: %d\n", shifts);
                printf("Time take for test in seconds : %f\n",((float) (shifts*0.005)));
                printf("Number of interval over 600 clockticks: %d\n", badcount);
                printf("Worst interval i clocktics: %d\n", worst);
                printf("Worst interval i micros: %f\n",(float) (worst*0.125));
                printf("Average number clock ticks between reads %f\n", average);
                printf("Average time in micros between reads %f\n",(float) (average*0.125));
                printf("Logging frequency %f\n\n",(float) 1/(average*0.000000125));

                free(pStore);
                free(pStore2);
                free(timeCount);
                free(buffer);
                //Closing TIP116 card
                for (i=0; i<MAX_DEVICES; i++)
                {
                        if (hDevices[i] >= 0)
    {
                                close(hDevices[i]);
    }
                }
                //Closing TIP501 card
                for (i=0; i<MAX_DEVICES; i++)
                {
                        if (hTIP501Devices[i] >= 0)
                        {
                                close(hTIP501Devices[i]);
                        }
                }

                printf("program finished");

                return 0;
}

static void PrintErrorMessage(void)

{
   printf("%s\n\n", strerror (errno));
}
```

timerReadTest.c

```
#include <stdio.h>
#include <sys/mman.h> // Needed for mlockall()
#include <unistd.h> // needed for sysconf(int name);
#include <malloc.h>
#include <sys/time.h> // needed for getrusage
#include <sys/resource.h> // needed for getrusage
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <sys/ioctl.h>
#include <sys/times.h>
#include <sys/timeb.h>
```

```
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <time.h>
#include <sched.h>

#include <tip501.h>
#include <tip600.h>
#include <tip116.h>

#define DEVICE_NAME     "TIP116"
#define MAX_DEVICES     32
#

#define TIP116EXA_VERSION  "1.0.0"

int hDevices[MAX_DEVICES];

#ifndef FALSE
#define FALSE 0
#endif
#ifndef TRUE
#define TRUE !FALSE
#endif

#define MY_PRIORITY (99)

#define MAX_LEN 100

#define SOMESIZE (10*1024*1024)

static void PrintErrorMessage(void);

int main()
{
                // Allocate some memory + variables for loading cards
                int i, page_size;
                char* buffer;
                struct rusage usage;
                int found;
                int hDevices[32];
                char Path[100];
                char* paths[32];
   found =0;
                int shifts = 0;
                int worst = 0;
                int average = 0;
                long total =0;

   // TIP116 variables
   int              result;
                int             hCurrent = 0;
                int             DeviceIndex = 0;
                T116_TIMER_BUFFER       timerBuf;
                T116_TIMER_CONFIG_BUFFER    timerConfBuf;

   memset((void*)&Path,0,100);
   memset((void*)&paths,0,32);

                // Now lock all current and future pages from preventing of being paged
```

```
		if (mlockall(MCL_CURRENT | MCL_FUTURE ))
		{
				perror("mlockall failed:");
		}
		//printf("mlockall fails since we are not running as root\n");

		page_size = sysconf(_SC_PAGESIZE);
		buffer = malloc(SOMESIZE);

		// Touch each page in this piece of memory to get it mapped into RAM
		for (i=0; i < SOMESIZE; i+=page_size)
		{
				// Each write to this buffer will generate a pagefault.
				// Once the pagefault is handled a page will be locked in memory and never
				// given back to the system.
				buffer[i] = 0;
				// print the number of major and minor pagefaults this application has triggered
				getrusage(RUSAGE_SELF, &usage);
		}
		//printf("Major-pagefaults:%ld, Minor Pagefaults:%ld\n", usage.ru_majflt, usage.ru_minflt);
		// buffer is never released, or swapped, so using it from now will never lead to any pagefault

		printf("*** Test Program for determining readTime for TIP116 external timer ***\n\n");

		struct sched_param param;
		/* Declare ourself as a real time task */
		param.sched_priority = MY_PRIORITY;
		if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
				perror("sched_setscheduler failed");
				exit(-1);
		}
		printf("Priority for program: %d.\n\n", MY_PRIORITY);


		/*
		 * Clock_resolutions test
		 */
		struct timespec	clock_resolution;
		int stat;
		stat = clock_getres(CLOCK_REALTIME, &clock_resolution);
		printf("Internal clock resolution is %d seconds, %ld nanoseconds\n\n",
					clock_resolution.tv_sec, clock_resolution.tv_nsec);

		// finding the devices!!!
  found = FALSE;
		for (i=0; i<MAX_DEVICES; i++)
		{
				sprintf(Path, "/dev/tip116_%0d", i);
				hDevices[i] = open(Path, O_RDWR);

				if (hDevices[i] >= 0)
				{
						found = TRUE;
						printf("    %s Device found --> %s\n\n", DEVICE_NAME,
Path);
				}
		}
		if (!found)
		{
```

```
                        printf("        No %s Devices found. Exit Example Application...\n\n",
DEVICE_NAME);
                        exit(1);
        }
        // Opening TIP116 card
        for (i=0; i<MAX_DEVICES; i++)
        {
                        if (hDevices[i] >= 0)
                        {
                                        DeviceIndex = i;
                                        hCurrent = hDevices[DeviceIndex];
                                        printf("        Open %s_%d for I/O...\n\n", DEVICE_NAME,
i);
                                        break;
                        }
        }

        /*
         *  Implemetation of external timer
         */
        timerConfBuf.preloadVal = 40000;
        timerConfBuf.prescaler = 1;
        timerConfBuf.enable = TRUE;

        result = ioctl(hCurrent, T116_IOCS_CONFIG_TIMER, &timerConfBuf);
        if(result >= 0){
                        printf(" External timer set to 8MHz\n\n");
        }else{
                        printf(" Timer config failed  -->  Error = %d\n\n", errno);
                        PrintErrorMessage();
        }

        /*
         ** Reading and storing values in a vector
         */
        unsigned short *timeCount;
        timeCount =(unsigned short*)malloc((1000000+1)* sizeof(*timeCount));
        if( timeCount == NULL){
                        printf("Not enough memory for to allocate timevector\n\n");
                        return 1;
        }

        for(i=0;i<1000000;i++)
        {
                        result = ioctl(hCurrent, T116_IOCG_READ_TIMER, &timerBuf);
                        if(result >= 0){
                                        timeCount[i] = timerBuf.value;
                        } else {
                                        printf("TimerCount failed  -->  Error = %d\n", errno);
                                        PrintErrorMessage();
                                        break;
                        }
        }

        /* Printing function */
        printf(" Prints first 20 time reads\n\n");
        for(i=0; i<20;i++){
                        printf("Time between reads %d from timeCount: %d\n", i, (timeCount[i]));
        }
        printf("\n Prints first 20 time intervals\n\n");
```

```
		for(i=0; i<20;i++){
			printf("Time between reads %d from timeCount: %d\n", i, (timeCount[i]-
timeCount[i+1]));
		}

		/* Calculationg function for all imortant values */
		for(i=0; i<999999;i++){
			if ((timeCount[i]-timeCount[i+1]) >0){
				total = total + (timeCount[i]-timeCount[i+1]);
			}else{
				total = total + (timeCount[i]+ (40000-timeCount[i+1]));
			}
			if((timeCount[i]-timeCount[i+1])> worst){
				worst = timeCount[i]-timeCount[i+1];
			}
			if((timeCount[i]-timeCount[i+1]) <0){
				shifts++;
			}
		}
		average = (total/999999);

		printf("\nTimerReadTest\n");
		printf("\nAverage time between reads in clockticks: %d\n", average);
		printf("Average time between reads in micros: %f\n",(float) (average*0.125));
		printf("Shiftcount: %d\n", shifts);
		printf("Time take for test in seconds : %f\n",((float) (shifts*0.005)));
		printf("Worst latency in clockticks: %d\n", worst);
		printf("Worst latency in micros: %f\n",(float) (worst*0.125));

		free(timeCount);
		//Closing TIP116 card
		for (i=0; i<MAX_DEVICES; i++)
		{
			if (hDevices[i] >= 0)
    {
				close(hDevices[i]);
    }
		}


		printf("\nprogram finished");

		return 0;
}
void stack_prefault(void) {

		//unsigned char dummy[MAX_SAFE_STACK];

		//memset(&dummy, 0, MAX_SAFE_STACK);
		return;
}


static void PrintErrorMessage(void)

{
   printf("%s\n\n", strerror (errno));
}
```

JNI test file: system performance test

```
package se.ipbyran.tews.tip.test;

import se.ipbyran.tews.tip.TipJni;
import se.ipbyran.tews.tip.TipJni.Tip116Board;
import se.ipbyran.tews.tip.TipJni.Tip501Board;
import se.ipbyran.tews.tip.TipJni.Tip600Board;
import se.ipbyran.tews.tip.TipJni.EncoderChannel;
import se.ipbyran.tews.tip.TipJni.AnalogChannel;
import java.util.*;


public class TestJNISystemPerformance {

		public TipJni jni;
		public Tip116Board[] tip116Boards;
		public Tip501Board[] tip501Boards;
		public Tip600Board[] tip600Boards;
		public ArrayList<EncoderChannel> lstEncoderChannels;
		public AnalogChannel[] lstAnalogChannels;

		public static void main(String[] args) {
			final TestJNISystemPerformance main = new TestJNISystemPerformance();
			main.testSystemPerformance();
		}

		public void testTipJni() {
			// Constructor
			jni = new TipJni();

		}

		public void testInitialize() {

			testTipJni();
			jni.initialize();
		}

		public void testGetBoards(){

			testInitialize();
			tip116Boards = jni.getAvailableTip116Boards();
			tip501Boards = jni.getAvailableTip501Boards();
			tip600Boards = jni.getAvailableTip600Boards();
		}
		public void testSystemPerformance(){
			testGetBoards();
			Tip116Board board116 = tip116Boards[0];
			board116.initialize();
			System.out.println("Boardnumber for opened board(tip116): "
+board116.getBoard());
			lstEncoderChannels = board116.getEncoderChannels();
			System.out.println("Number of encoder channels: "
+lstEncoderChannels.size());
			for(EncoderChannel current:lstEncoderChannels){
				System.out.println("EncoderChannel number: "
+current.getChannelNo());
			}
			board116.configureExternalTimer();
```

```
                    lstEncoderChannels.get(2).runSystemTest();
                    board116.close();
            }
}
```

JNI test file: run enconder

```
package se.ipbyran.tews.tip.test;

import se.ipbyran.tews.tip.TipJni;
import se.ipbyran.tews.tip.TipJni.Tip116Board;
import se.ipbyran.tews.tip.TipJni.Tip501Board;
import se.ipbyran.tews.tip.TipJni.Tip600Board;
import se.ipbyran.tews.tip.TipJni.EncoderChannel;
import se.ipbyran.tews.tip.TipJni.AnalogChannel;
import java.util.*;


public class TestJNIrunCounter {

            public TipJni jni;
            public Tip116Board[] tip116Boards;
            public Tip501Board[] tip501Boards;
            public Tip600Board[] tip600Boards;
            public ArrayList<EncoderChannel> lstEncoderChannels;
            public AnalogChannel[] lstAnalogChannels;

            public static void main(String[] args) {
                    // TODO Auto-generated method stub
                    final TestJNIrunCounter main = new TestJNIrunCounter();
                    main.testRunCounter();
            }

            public void testTipJni() {
                    // Constructor

                    // Prints java.library.path
                    // System.out.println(System.getProperty("java.library.path"));
                    jni = new TipJni();

            }

            public void testInitialize() {

                    testTipJni();
                    jni.initialize();
                    //fail("Not yet implemented");
            }

            public void testGetBoards(){
                    // Function for getting boardinfo

                    testInitialize();
                    tip116Boards = jni.getAvailableTip116Boards();
                    tip501Boards = jni.getAvailableTip501Boards();
                    tip600Boards = jni.getAvailableTip600Boards();
            }

            public void testRunCounter(){
                    testGetBoards();
                    Tip116Board board116 = tip116Boards[0];
```

```
                        board116.initialize();
                        System.out.println("Boardnumber for opened board(tip116): "
+board116.getBoard());
                        lstEncoderChannels = board116.getEncoderChannels();
                        System.out.println("Number of encoder channels: "
+lstEncoderChannels.size());
                        for(EncoderChannel current:lstEncoderChannels){
                                System.out.println("EncoderChannel number: "
+current.getChannelNo());
                        }
                        board116.configureExternalTimer();

                        int[][] result = lstEncoderChannels.get(2).runCounter(1);
                        if (result != null){
                                for(int i = 0; i<100; i++){
                                        System.out.println("From Java-side:
Resultvector index: "+i+" with value: "+result[i][0]);
                                        System.out.println("From Java-side:
Resultvector index: "+i+" with time: "+result[i][1]);
                                }
                        }else{
                                System.out.println("From Java side: Resultvector is empty");
                        }
                        board116.close();
                }
}
```

JNI test file: run all enconder channels

```
package se.ipbyran.tews.tip.test;

import java.util.ArrayList;

import se.ipbyran.tews.tip.TipJni;
import se.ipbyran.tews.tip.TipJni.AnalogChannel;
import se.ipbyran.tews.tip.TipJni.EncoderChannel;
import se.ipbyran.tews.tip.TipJni.Tip116Board;
import se.ipbyran.tews.tip.TipJni.Tip501Board;
import se.ipbyran.tews.tip.TipJni.Tip600Board;

public class TestJNIrunAllCounters {

        public TipJni jni;
        public Tip116Board[] tip116Boards;
        public Tip501Board[] tip501Boards;
        public Tip600Board[] tip600Boards;
        public ArrayList<EncoderChannel> lstEncoderChannels;
        public AnalogChannel[] lstAnalogChannels;

        public static void main(String[] args) {
                final TestJNIrunAllCounters main = new TestJNIrunAllCounters();
                main.testRunAllCounter();

        }
        public void testTipJni() {
                // Constructor
```

```
                        // Prints java.library.path
                        // System.out.println(System.getProperty("java.library.path"));
                        jni = new TipJni();

                }

                public void testInitialize() {

                        testTipJni();
                        jni.initialize();
                        //fail("Not yet implemented");
                }

                public void testGetBoards(){
                        // Function for getting boardinfo

                        testInitialize();
                        tip116Boards = jni.getAvailableTip116Boards();
                        tip501Boards = jni.getAvailableTip501Boards();
                        tip600Boards = jni.getAvailableTip600Boards();
                }

                public void testRunAllCounter(){
                        testGetBoards();
                        Tip116Board board116 = tip116Boards[0];
                        board116.initialize();
                        System.out.println("Boardnumber for opened board(tip116): "
+board116.getBoard());
                        lstEncoderChannels = board116.getEncoderChannels();
                        System.out.println("Number of encoder channels: "
+lstEncoderChannels.size());
                        for(EncoderChannel current:lstEncoderChannels){
                                System.out.println("EncoderChannel number: "
+current.getChannelNo());
                        }
                        board116.configureExternalTimer();

                        int[][] result = board116.runAllCounters(1);

                        if (result != null){
                                for(int i = 0; i<10; i++){
                                        System.out.println("From Java-side:
Resultvector index: "+i+" column1 with value: "+result[i][0]);
                                        System.out.println("From Java-side:
Resultvector index: "+i+" column2 with value: "+result[i][1]);
                                        System.out.println("From Java-side:
Resultvector index: "+i+" column3 with value: "+result[i][2]);
                                        System.out.println("From Java-side:
Resultvector index: "+i+" column4 with value: "+result[i][3]);
                                        System.out.println("From Java-side:
Resultvector index: "+i+" column5 with time: "+result[i][4]);
                                }
                        }else{
                                System.out.println("From Java side: Resultvector is empty");
                        }
                        board116.close();
                }
}
```

JNI test file: run Analog Channel

```
package se.ipbyran.tews.tip.test;

import se.ipbyran.tews.tip.TipJni;
import se.ipbyran.tews.tip.TipJni.Tip116Board;
import se.ipbyran.tews.tip.TipJni.Tip501Board;
import se.ipbyran.tews.tip.TipJni.Tip600Board;
import se.ipbyran.tews.tip.TipJni.EncoderChannel;
import se.ipbyran.tews.tip.TipJni.AnalogChannel;
import java.util.*;


public class TestJNIrunAnalogChannel {

		public TipJni jni;
		public Tip116Board[] tip116Boards;
		public Tip501Board[] tip501Boards;
		public Tip600Board[] tip600Boards;
		public ArrayList<EncoderChannel> lstEncoderChannels;
		public AnalogChannel[] lstAnalogChannels;

		public static void main(String[] args) {
				// TODO Auto-generated method stub
				final TestJNIrunAnalogChannel main = new TestJNIrunAnalogChannel();
				main.testRunCounter();
		}

		public void testTipJni() {
				// Constructor

				// Prints java.library.path
				// System.out.println(System.getProperty("java.library.path"));
				jni = new TipJni();

		}

		public void testInitialize() {

				testTipJni();
				jni.initialize();
				//fail("Not yet implemented");
		}

		public void testGetBoards(){
				// Function for getting boardinfo

				testInitialize();
				tip116Boards = jni.getAvailableTip116Boards();
				tip501Boards = jni.getAvailableTip501Boards();
				tip600Boards = jni.getAvailableTip600Boards();
		}

		public void testRunCounter(){
				testGetBoards();
				Tip116Board board116 = tip116Boards[0];
				Tip501Board board501 = tip501Boards[0];
				board116.initialize();
				board501.initialize();
```

```
                        System.out.println("Boardnumber for opened board(tip116): "
+board116.getBoard());
                        System.out.println("Boardnumber for opened board(tip501): "
+board501.getBoard());
                        lstEncoderChannels = board116.getEncoderChannels();
                        System.out.println("Number of encoder channels: "
+lstEncoderChannels.size());
                        for(EncoderChannel current:lstEncoderChannels){
                                System.out.println("EncoderChannel number: "
+current.getChannelNo());
                        }
                        board116.configureExternalTimer();

                        int[][] result = board501.runAnalog(1, board116, 1);
                        if (result != null){
                                for(int i = 0; i<100; i++){
                                        System.out.println("From Java-side:
Resultvector index: "+i+" with value: "+result[i][0]);
                                        System.out.println("From Java-side:
Resultvector index: "+i+" with time: "+result[i][1]);
                                }
                        }else{
                                System.out.println("From Java side: Resultvector is empty");
                        }
                        board116.close();
                        board501.close();
                }
}
```

JNI test file: run All Analog Channel

```
package se.ipbyran.tews.tip.test;

import se.ipbyran.tews.tip.TipJni;
import se.ipbyran.tews.tip.TipJni.Tip116Board;
import se.ipbyran.tews.tip.TipJni.Tip501Board;
import se.ipbyran.tews.tip.TipJni.Tip600Board;
import se.ipbyran.tews.tip.TipJni.EncoderChannel;
import se.ipbyran.tews.tip.TipJni.AnalogChannel;
import java.util.*;


public class TestJNIrunAnalogChannel {

        public TipJni jni;
        public Tip116Board[] tip116Boards;
        public Tip501Board[] tip501Boards;
        public Tip600Board[] tip600Boards;
        public ArrayList<EncoderChannel> lstEncoderChannels;
        public AnalogChannel[] lstAnalogChannels;

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                final TestJNIrunAnalogChannel main = new TestJNIrunAnalogChannel();
                main.testRunCounter();
        }

        public void testTipJni() {
                // Constructor
```

```
                        // Prints java.library.path
                        // System.out.println(System.getProperty("java.library.path"));
                        jni = new TipJni();

                }

                public void testInitialize() {

                        testTipJni();
                        jni.initialize();
                        //fail("Not yet implemented");
                }

                public void testGetBoards(){
                        // Function for getting boardinfo

                        testInitialize();
                        tip116Boards = jni.getAvailableTip116Boards();
                        tip501Boards = jni.getAvailableTip501Boards();
                        tip600Boards = jni.getAvailableTip600Boards();
                }

                public void testRunCounter(){
                        testGetBoards();
                        Tip116Board board116 = tip116Boards[0];
                        Tip501Board board501 = tip501Boards[0];
                        board116.initialize();
                        board501.initialize();
                        System.out.println("Boardnumber for opened board(tip116): "
+board116.getBoard());
                        System.out.println("Boardnumber for opened board(tip501): "
+board501.getBoard());
                        lstEncoderChannels = board116.getEncoderChannels();
                        System.out.println("Number of encoder channels: "
+lstEncoderChannels.size());
                        for(EncoderChannel current:lstEncoderChannels){
                                System.out.println("EncoderChannel number: "
+current.getChannelNo());
                        }
                        board116.configureExternalTimer();

                        int[][] result = board501.runAnalog(1, board116, 1);
                        if (result != null){
                                for(int i = 0; i<100; i++){
                                        System.out.println("From Java-side:
Resultvector index: "+i+" with value: "+result[i][0]);
                                        System.out.println("From Java-side:
Resultvector index: "+i+" with time: "+result[i][1]);
                                }
                        }else{
                                System.out.println("From Java side: Resultvector is empty");
                        }
                        board116.close();
                        board501.close();
                }
}
```

JNI test file: run Digital

```
package se.ipbyran.tews.tip.test;

import se.ipbyran.tews.tip.TipJni;
import se.ipbyran.tews.tip.TipJni.Tip116Board;
import se.ipbyran.tews.tip.TipJni.Tip501Board;
import se.ipbyran.tews.tip.TipJni.Tip600Board;
import se.ipbyran.tews.tip.TipJni.EncoderChannel;
import se.ipbyran.tews.tip.TipJni.AnalogChannel;
import java.util.*;


public class TestJNIrunDigital {

		public TipJni jni;
		public Tip116Board[] tip116Boards;
		public Tip501Board[] tip501Boards;
		public Tip600Board[] tip600Boards;
		public ArrayList<EncoderChannel> lstEncoderChannels;
		public AnalogChannel[] lstAnalogChannels;

		public static void main(String[] args) {
			// TODO Auto-generated method stub
			final TestJNIrunDigital main = new TestJNIrunDigital();
			main.testRunCounter();
		}

		public void testTipJni() {
			// System.out.println(System.getProperty("java.library.path"));
			jni = new TipJni();

		}

		public void testInitialize() {

			testTipJni();
			jni.initialize();
			//fail("Not yet implemented");
		}

		public void testGetBoards(){
			// Function for getting boardinfo

			testInitialize();
			tip116Boards = jni.getAvailableTip116Boards();
			tip501Boards = jni.getAvailableTip501Boards();
			tip600Boards = jni.getAvailableTip600Boards();
		}

		public void testRunCounter(){
			testGetBoards();
			Tip116Board board116 = tip116Boards[0];
			Tip600Board board600 = tip600Boards[0];
			board116.initialize();
			board600.initialize();
			System.out.println("Boardnumber for opened board(tip116): "
+board116.getBoard());
			System.out.println("Boardnumber for opened board(tip600): "
+board600.getBoard());
			lstEncoderChannels = board116.getEncoderChannels();
```

```
                                        System.out.println("Number of encoder channels: "
+lstEncoderChannels.size());
                                        for(EncoderChannel current:lstEncoderChannels){
                                                System.out.println("EncoderChannel number: "
+current.getChannelNo());
                                        }
                                        board116.configureExternalTimer();
                                        System.out.println("Runtime get Total memory: "
+Runtime.getRuntime().maxMemory());

                                        int[][] result = board600.runDigitalLogging(board116, 30);
                                        if (result != null){
                                                for(int i = 0; i<100; i++){
                                                        System.out.println("From Java-side:
Resultvector index: "+i+" with value: "+result[i][0]);
                                                        System.out.println("From Java-side:
Resultvector index: "+i+" with time: "+result[i][1]);
                                                }
                                        }else{
                                                System.out.println("From Java side: Resultvector is empty");
                                        }
                                        board116.close();
                                        board600.close();
                        }
}
```

Appendix D – JNI main class

```
package se.ipbyran.tews.tip;

import java.util.ArrayList;

public class TipJni {

		private ArrayList<Tip501Board> lstAnalogBoards = new ArrayList<Tip501Board>();
		private ArrayList<Tip600Board> lstDigitalBoards = new ArrayList<Tip600Board>();
		private ArrayList<Tip116Board> lstEncoderBoards = new ArrayList<Tip116Board>();

		public TipJni(){

		}

		public void initialize(){
			String[] boardNames = getBoards();
			for(String boardName:boardNames){
				if(boardName.contains("tip116")){
					lstEncoderBoards.add(new
Tip116Board(boardName));
				}else if(boardName.contains("tip501")){
					lstAnalogBoards.add(new
Tip501Board(boardName));
				}else if(boardName.contains("tip600")){
					lstDigitalBoards.add(new
Tip600Board(boardName));
				}
			}
		}

		public Tip116Board[] getAvailableTip116Boards(){

			int size = lstEncoderBoards.size();
			Tip116Board[] encoderBoards = new Tip116Board[size];

			for(int i = 0; i<lstEncoderBoards.size();i++){
				encoderBoards[i]=lstEncoderBoards.get(i);
			}
			return encoderBoards;
		}
		public Tip501Board[] getAvailableTip501Boards(){
			int size = lstAnalogBoards.size();
			Tip501Board[] analogBoards = new Tip501Board[size];

			for(int i = 0; i<lstAnalogBoards.size();i++){
				analogBoards[i]=lstAnalogBoards.get(i);
			}
			return analogBoards;
		}
		public Tip600Board[] getAvailableTip600Boards(){
			int size = lstDigitalBoards.size();
			Tip600Board[] digitalBoards = new Tip600Board[size];

			for(int i = 0; i<lstDigitalBoards.size();i++){
				digitalBoards[i]=lstDigitalBoards.get(i);
			}
			return digitalBoards;
		}
```

```
		public int numberOfTip116Boards(){
			return lstEncoderBoards.size();
		}
		public int numberOfTip501Boards(){
			return lstAnalogBoards.size();
		}
		public int numberOfTip600Boards(){
			return lstDigitalBoards.size();
		}

		public class Tip116Board{

			private int board = 0;
			private String name;
			private ArrayList<EncoderChannel> lstChannels = new
ArrayList<EncoderChannel>();

			public Tip116Board(String name){
				this.name=name;
			}

			public String getBoardStringName(){
				return this.name;
			}

			public int getBoard() {
				return this.board;
			}

			public void setBoard(int board) {
				this.board = board;
			}

			public void initialize(){
				open();
			}

			public void open(){
				this.board = openBoard(name);
				if(board!=0){
					for(int x=1;x<5;x++){
						lstChannels.add(new
EncoderChannel(this,x));
					}
					for(EncoderChannel ch:lstChannels){
						ch.initialize();
					}
				}
			}

			public void close(){
				closeBoard(board);
			}

			public ArrayList<EncoderChannel> getEncoderChannels(){
				return lstChannels;
			}

			public int[][] runAllCounters(float time){
				return runAllCounterChannels(this.board, time);
```

```
                }

                public int[][] runAllCountersWithTrigg(Tip600Board tip600Board,int
triggerType, float time){
                        return runAllCountersWithTrigger(this.board,
tip600Board.getBoard(), triggerType, time);
                }

                public void configureExternalTimer(){
                        configureTimer(this.board);
                }
        }

        public class EncoderChannel{

                private Tip116Board board;
                private int channelNo = 0; //Invalid channel default.

                public EncoderChannel(Tip116Board board, int channel){
                        channelNo=channel;
                        this.board = board;
                }

                public int getChannelNo() {
                        return channelNo;
                }

                public void setChannelNo(int channelNo) {
                        this.channelNo = channelNo;
                }

                public void initialize(){
                        setCfg();
                }

                // Sets all Channelsetting to default
                private void setCfg(){
                        configureChannel(board.getBoard(), this.channelNo);
                }

                //runs Counter and loggs value for a time seconds
                public int[][] runCounter(float time){
                        return runCounterChannel(board.getBoard(),
this.getChannelNo(), time);
                }

                public int[][] runCounterWithTrigger(Tip600Board tip600board, int
triggerType, float time ){
                        return runCounterChannelWithTrigger(board.getBoard(),
this.getChannelNo(), tip600board.getBoard(), triggerType, time);
                }

                public void runSystemTest(){
                        systemPerformanceTest(board.getBoard(),
this.getChannelNo());
                }
        }

        public class Tip501Board {
```

```
		//necessary since analogType specifies the number of analog channels
		public static final int T501DIFF = 0x1;
		public static final int T501PIPELINE=0x2;
		public static final int T501CORRECTION=0x4;
		private int analogType=T501DIFF|T501PIPELINE; //Use pipeline mode and
differential mode default
		private int board;
		private String name;
		private AnalogChannel[] channels = new AnalogChannel[8];

		public Tip501Board(String name){
			this.name = name;
		}

		public String getBoardStringName(){
			return name;
		}

		public int getBoard() {
			return board;
		}

		public void setBoard(int board) {
			this.board = board;
		}

		public void initialize(){
			open();
		}

		public void open(){
			board = openBoard(name);
			for(int x=0;x<channels.length;x++){
				channels[x] = new AnalogChannel(x);
			}
		}
		//necessary since analogType specifies the number of channels
		public int getAnalogType() {
			return analogType;
		}
		//necessary since analogType specifies the number of channels
		public void setAnalogType(int analogType) {
			if(this.analogType==analogType)return;
			this.analogType = analogType;
			if((analogType&T501DIFF)==T501DIFF){
				channels = new AnalogChannel[8];
			}else{
				channels = new AnalogChannel[16];
			}
			for(int x=0;x<channels.length;x++){
				channels[x]=new AnalogChannel(x);
			}
		}

		public void close(){
			if(board == 0){
				return;
			}else{
				closeBoard(this.board);
			}
```

```
                }

                public AnalogChannel[] getAnalogChannels(){
                        return channels;
                }

                public int[][] runAnalog(int channel, Tip116Board tip116board, float time){
                        return runAnalogChannel(this.board, channel,
tip116board.getBoard(), this.analogType, time);
                }

                public int[][] runAnalogWithTrigger(int channel, Tip116Board tip116board,
Tip600Board tip600board, int triggerType, float time){
                        return runAnalogChannelWithTrigger(this.board, channel,
tip116board.getBoard(), tip600board.getBoard(),
                                                                triggerType, this.analogType,
time);
                }

                public int[][] runAllAnalog(Tip116Board tip116Board, float time){
                        return runAllAnalogChannels(this.board,
tip116Board.getBoard(), this.analogType, time);
                }

                public int[][] runAllAnalogWithTrigger(Tip116Board tip116board,
Tip600Board tip600board, int triggerType, float time ){
                        return runAllAnalogChannelsWithTrigger(this.board,
tip116board.getBoard(), tip600board.getBoard(),
                                                                triggerType, this.analogType,
time);
                }
        }

        public class AnalogChannel{

                private int channelNo = 0;

                public AnalogChannel(int channelNo){
                        this.channelNo = channelNo;
                }
                public int getAnalogChannel(){
                        return this.channelNo;
                }
        }

        public class Tip600Board{

                private String name;
                private int board;

                public Tip600Board(String name){
                        this.name = name;
                }

                public String getBoardStringName(){
                        return name;
                }

                public void setBoard(int board){
                        this.board = board;
```

```
                }

                public int getBoard(){
                        return this.board;
                }

                public void initialize(){
                        open();
                }

                public void open(){
                        board=openBoard(this.name);
                }

                public void close(){
                        if(board == 0){
                                return;
                        }else{
                        closeBoard(this.board);
                        }
                }

                public int[][] runDigitalLogging(Tip116Board tip116Board, float time){
                        return runDigitalInput(this.board, (tip116Board.getBoard()),
time);
                }

                public int[][] runDigitalLoggingWithTrigger(Tip116Board tip116Board, int
triggerType, float time){
                        return runDigitalInputWithTrigger(this.board,
(tip116Board.getBoard()), triggerType, time);
                }

                public boolean waitForAnyTrigg(){
                        return waitForAnyTransition(this.board);
                }

                public boolean waitForHighTrigg(){
                        return waitForHighTransition(board);
                }
        }

        /*
         * Generic Tip functions
         */
        private native String[] getBoards();
        private native int openBoard(String board);
        private native boolean closeBoard(int board);

        /*
         * Tip116 functions
         */
        private static native int[][] runCounterChannel(int board, int channel, float time);
        private static native int[][] runCounterChannelWithTrigger(int encoderBoard, int channel, int
digitalBoard, int triggerType, float time);
        private static native int[][] runAllCounterChannels(int board, float time);
        private static native int[][] runAllCountersWithTrigger(int encoderBoard, int digitalBoard, int
triggertype, float time);
        private static native boolean configureTimer(int board);
        private static native boolean configureChannel(int board, int channelNo);
```

```
		private static native boolean systemPerformanceTest(int board, int channelNo);

		/*
		 * Tip501 functions
		 */
		private static native int[][] runAnalogChannel(int analogBoard, int channel, int encoderBoard,
int conversionType, float time);
		private static native int[][] runAnalogChannelWithTrigger(int analogBoard, int channel, int
encoderBoard, int digitalBoard, int triggerType, int conversionType, float time);
		private static native int[][] runAllAnalogChannels(int analogBoard, int encoderBoard, int
conversionType, float time);
		private static native int[][] runAllAnalogChannelsWithTrigger(int analogBoard, int
encoderBoard, int digitalBoard, int triggerType, int conversionType, float time);
		/*
		 * Tip600 functions
		 */
		private static native int[][] runDigitalInput(int digitalBoardNo, int encoderBoard, float time);
		private static native int[][] runDigitalInputWithTrigger(int digitalBoard, int encoderBoard, int
triggerType, float time);
		private static native boolean waitForHighTransition(int board);
		private static native boolean waitForAnyTransition(int board);

		static{
			System.loadLibrary( "motionloggerIo" );
		}
}
```

Appendix E – C-headers

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class se_ipbyran_tews_tip_TipJni */

#ifndef _Included_se_ipbyran_tews_tip_TipJni
#define _Included_se_ipbyran_tews_tip_TipJni
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     se_ipbyran_tews_tip_TipJni
 * Method:    getBoards
 * Signature: ()[Ljava/lang/String;
 */
JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_getBoards
  (JNIEnv *, jobject);

/*
 * Class:     se_ipbyran_tews_tip_TipJni
 * Method:    openBoard
 * Signature: (Ljava/lang/String;)I
 */
JNIEXPORT jint JNICALL Java_se_ipbyran_tews_tip_TipJni_openBoard
  (JNIEnv *, jobject, jstring);

/*
 * Class:     se_ipbyran_tews_tip_TipJni
 * Method:    closeBoard
 * Signature: (I)Z
 */
JNIEXPORT jboolean JNICALL Java_se_ipbyran_tews_tip_TipJni_closeBoard
  (JNIEnv *, jobject, jint);

/*
 * Class:     se_ipbyran_tews_tip_TipJni
 * Method:    runCounterChannel
 * Signature: (IIF)[[I
 */
JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_runCounterChannel
  (JNIEnv *, jclass, jint, jint, jfloat);

/*
 * Class:     se_ipbyran_tews_tip_TipJni
 * Method:    runCounterChannelWithTrigger
 * Signature: (IIIIF)[[I
 */
JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_runCounterChannelWithTrigger
  (JNIEnv *, jclass, jint, jint, jint, jint, jfloat);

/*
 * Class:     se_ipbyran_tews_tip_TipJni
 * Method:    runAllCounterChannels
 * Signature: (IF)[[I
 */
JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_runAllCounterChannels
  (JNIEnv *, jclass, jint, jfloat);

/*
 * Class:     se_ipbyran_tews_tip_TipJni
```

```
 * Method:    runAllCountersWithTrigger
 * Signature: (IIIF)[[I
 */
JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_runAllCountersWithTrigger
  (JNIEnv *, jclass, jint, jint, jint, jfloat);

/*
 * Class:     se_ipbyran_tews_tip_TipJni
 * Method:    configureTimer
 * Signature: (I)Z
 */
JNIEXPORT jboolean JNICALL Java_se_ipbyran_tews_tip_TipJni_configureTimer
  (JNIEnv *, jclass, jint);

/*
 * Class:     se_ipbyran_tews_tip_TipJni
 * Method:    configureChannel
 * Signature: (II)Z
 */
JNIEXPORT jboolean JNICALL Java_se_ipbyran_tews_tip_TipJni_configureChannel
  (JNIEnv *, jclass, jint, jint);

/*
 * Class:     se_ipbyran_tews_tip_TipJni
 * Method:    systemPerformanceTest
 * Signature: (II)Z
 */
JNIEXPORT jboolean JNICALL Java_se_ipbyran_tews_tip_TipJni_systemPerformanceTest
  (JNIEnv *, jclass, jint, jint);

/*
 * Class:     se_ipbyran_tews_tip_TipJni
 * Method:    runAnalogChannel
 * Signature: (IIIIF)[[I
 */
JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_runAnalogChannel
  (JNIEnv *, jclass, jint, jint, jint, jint, jfloat);

/*
 * Class:     se_ipbyran_tews_tip_TipJni
 * Method:    runAnalogChannelWithTrigger
 * Signature: (IIIIIIF)[[I
 */
JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_runAnalogChannelWithTrigger
  (JNIEnv *, jclass, jint, jint, jint, jint, jint, jint, jfloat);

/*
 * Class:     se_ipbyran_tews_tip_TipJni
 * Method:    runAllAnalogChannels
 * Signature: (IIIF)[[I
 */
JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_runAllAnalogChannels
  (JNIEnv *, jclass, jint, jint, jint, jfloat);

/*
 * Class:     se_ipbyran_tews_tip_TipJni
 * Method:    runAllAnalogChannelsWithTrigger
 * Signature: (IIIIIF)[[I
 */
```

```
JNIEXPORT jobjectArray JNICALL
Java_se_ipbyran_tews_tip_TipJni_runAllAnalogChannelsWithTrigger
  (JNIEnv *, jclass, jint, jint, jint, jint, jint, jfloat);

/*
 * Class:     se_ipbyran_tews_tip_TipJni
 * Method:    runDigitalInput
 * Signature: (IIF)[[I
 */
JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_runDigitalInput
  (JNIEnv *, jclass, jint, jint, jfloat);

/*
 * Class:     se_ipbyran_tews_tip_TipJni
 * Method:    runDigitalInputWithTrigger
 * Signature: (IIIF)[[I
 */
JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_runDigitalInputWithTrigger
  (JNIEnv *, jclass, jint, jint, jint, jfloat);

/*
 * Class:     se_ipbyran_tews_tip_TipJni
 * Method:    waitForHighTransition
 * Signature: (I)Z
 */
JNIEXPORT jboolean JNICALL Java_se_ipbyran_tews_tip_TipJni_waitForHighTransition
  (JNIEnv *, jclass, jint);

/*
 * Class:     se_ipbyran_tews_tip_TipJni
 * Method:    waitForAnyTransition
 * Signature: (I)Z
 */
JNIEXPORT jboolean JNICALL Java_se_ipbyran_tews_tip_TipJni_waitForAnyTransition
  (JNIEnv *, jclass, jint);

#ifdef __cplusplus
}
#endif
#endif
```

Appendix F – C-implementation

```
#include <jni.h>
#include "se_ipbyran_tews_tip_TipJni.h"
#include "se_ipbyran_tews_tip_TipJni_Tip116Board.h"
#include "se_ipbyran_tews_tip_TipJni_Tip501Board.h"
#include "se_ipbyran_tews_tip_TipJni_Tip600Board.h"
#include "se_ipbyran_tews_tip_TipJni_AnalogChannel.h"
#include "se_ipbyran_tews_tip_TipJni_EncoderChannel.h"
#include <stdio.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <string.h>
#include <malloc.h>
#include <tip116.h>
#include <tip501.h>
#include <tip600.h>
#include <sys/mman.h> // Needed for mlockall()
#include <unistd.h> // needed for sysconf(int name)
#include <sys/time.h> // needed for getrusage
#include <sys/resource.h> // needed for getrusage
#include <limits.h>
#include <errno.h>
#include <sched.h>

#define RT_PRIORITY (79)
#define NON_RT_PRIO (30)
#define SOMESIZE (32*1024*1024)

#ifndef FALSE
#define FALSE 0
#endif
#ifndef TRUE
#define TRUE !FALSE
#endif

static void PrintErrorMessage(void);

//Method: getBoards
JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_getBoards
                (JNIEnv *env, jobject boards)
{
                jobjectArray ar;
                int found;
                int i;
                int hDevices[32];
                char Path[100];
                int foundCount=0;
                char* paths[32];
                char* currentPath;

                found =0;
                memset((void*)&Path,0,100);
                memset((void*)&paths,0,32);

                //Find all the Encoder cards
                for (i=0; i<8; i++)
                {
```

```
                        sprintf(Path, "/dev/tip116_%0d", i);

                        hDevices[i] = open(Path, O_RDWR);

                        if (hDevices[i] >= 0)
                        {
                                currentPath = (char*)malloc(strlen(Path));
                                strcpy(currentPath,(char*)&Path);
                                paths[foundCount]=currentPath;
                                found = 1;
                                foundCount++;
                                close(hDevices[i]);
                        }
                }
                //Find all the Analogue I/O cards of type 501
                for (i=0; i<4; i++)
                {
                        sprintf(Path, "/dev/tip501_%0d", i);

                        hDevices[i] = open(Path, O_RDWR);

                        if (hDevices[i] >= 0)
                        {
                                currentPath = (char*)malloc(strlen(Path));
                                strcpy(currentPath,(char*)&Path);
                                paths[foundCount]=currentPath;
                                found = 1;
                                foundCount++;
                                close(hDevices[i]);
                        }
                }

                //FInd all the Digital I/O cards of type 600
                for (i=0; i<8; i++)
                {

                        sprintf(Path, "/dev/tip600_%0d", i);

                        hDevices[i] = open(Path, O_RDWR);

                        if (hDevices[i] >= 0)
                        {
                                currentPath = (char*)malloc(strlen(Path));
                                strcpy(currentPath,(char*)&Path);
                                paths[foundCount]=currentPath;
                                found = 1;
                                foundCount++;
                                close(hDevices[i]);
                        }
                }

                ar = (jobjectArray)(*env)->NewObjectArray(env,foundCount,(*env)-
>FindClass(env,"java/lang/String"),(*env)->NewStringUTF(env,""));
                for(i=0;i<foundCount;i++){
                        if(paths[i]!=0){
                                (*env)->SetObjectArrayElement(env,ar, i,(*env)-
>NewStringUTF(env,paths[i]));
                                                free((void*)paths[i]);
                        }
                }
```

```
		return ar;
}


//Method: openBoard
JNIEXPORT jint JNICALL Java_se_ipbyran_tews_tip_TipJni_openBoard
		(JNIEnv *env, jobject object, jstring boardName)
{
		int device=0;
		device = open((*env)->GetStringUTFChars(env,boardName,0),O_RDWR);
		return(jint)device;
}

//Method: closeBoard
JNIEXPORT jboolean JNICALL Java_se_ipbyran_tews_tip_TipJni_closeBoard
		(JNIEnv *env, jobject obj, jint boardNo)
{
		int board =(int) boardNo;
		if(board!=0){
			//printf("From C side: Board %d is closing .....\n", boardNo);
			close(board);
		}
		return (jboolean)1;
}

//Method: runCounterChannel
JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_runCounterChannel
		(JNIEnv *env, jclass cls, jint boardNo, jint channel, jfloat time)
{
		jintArray                                                                array;
		int
		i, page_size;
		int
		result;
		int
			encoderboard;
		int
			loggNo;
		int
			channelNo;
		char*
		buffer;
		struct
		rusage usage;
		T116_COUNTER_BUFFER	countBuf;
		T116_TIMER_BUFFER	timerBuf;
		encoderboard = (int) boardNo;
		channelNo = (int) channel;


		if(time <= 0.5 ){
			loggNo = (int) (0.5*127000);
		}else if(time >=30){
			loggNo = (int) (30*127000);
		}else{
			loggNo = (int) (time*127000);
		}

		if (mlockall(MCL_CURRENT | MCL_FUTURE )){
```

```
                    //perror("mlockall failed:");
                    return NULL;
}
page_size = sysconf(_SC_PAGESIZE);
buffer = malloc(SOMESIZE);
// Touch each page in this piece of memory to get it mapped into RAM
for (i=0; i < SOMESIZE; i+=page_size){
                    buffer[i] = 0;
                    getrusage(RUSAGE_SELF, &usage);
}
// buffer is never released, or swapped, so using it from now will never lead to any pagefault
long *pStore;
pStore =(long*)malloc((loggNo+1) *sizeof(*pStore));
if( pStore == NULL){
                    return NULL;
}
memset(pStore,0,sizeof((loggNo+1)*sizeof(*pStore)));

unsigned short *timeCount;
timeCount =(unsigned short*)malloc((loggNo+1)*sizeof(*timeCount));
if(timeCount == NULL){
                    return NULL;
}
memset(timeCount,0,sizeof((loggNo+1)*sizeof(unsigned short)));

/* Declare ourself as a real time task */
struct sched_param param;
param.sched_priority = RT_PRIORITY;
if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                    //perror("sched_setscheduler failed");
                    free(timeCount);
                    free(pStore);
                    free(buffer);
                    return NULL;
}

countBuf.channel = channelNo;
countBuf.data = 0;
countBuf.flags = T116_IMMEDIATE_PRELOAD;

result = ioctl(encoderboard, T116_IOCS_SET_PRELOAD, &countBuf);
if (result >= 0){
                    //OK
}else{
                    //PrintErrorMessage();
}

//Reading and storing values in a vector
for(i=0;i<loggNo;i++)
{
                    result = ioctl(encoderboard, T116_IOCG_READ_COUNTER, &countBuf);
                    if(result >= 0){
                                        pStore[i] = countBuf.data;
                    } else {
                                        //printf("ReadFunction failed  -->  Error = %d\n", errno);
                                        //PrintErrorMessage();
                                        return NULL;
                    }
                    result = ioctl(encoderboard, T116_IOCG_READ_TIMER, &timerBuf);
                    if(result >= 0){
```

```
                                        timeCount[i] = timerBuf.value;
                        } else {
                                        //printf("TimerCount failed  -->  Error = %d\n", errno);
                                        //PrintErrorMessage();
                                        return NULL;
                        }
        }
        // Set prio to NON_RT since we are out from RT-section
        param.sched_priority = NON_RT_PRIO;
        if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                        //perror("sched_setscheduler failed");
                        return NULL;
        }
        jclass intArrCls = (*env)->FindClass(env, "[I");
        if(intArrCls == NULL){
                        // Returning NULL since we have an exception here
                        return NULL;
        }

        array = (*env)->NewObjectArray(env, loggNo, intArrCls, NULL);
        if (array == NULL){
                        // Out of memory in heap
                        return NULL;
        }
        for(i=0;i<loggNo;i++)
        {
                        jint tempColumn[2];
                        jintArray column = (*env)->NewIntArray(env, 2);
                        if (column == NULL){
                                        // out of memory
                                        return NULL;
                        }
                        tempColumn[0] = pStore[i];
                        tempColumn[1] = timeCount[i];
                        (*env)->SetIntArrayRegion(env, column, 0, 2, tempColumn);
                        (*env)->SetObjectArrayElement(env, array, i, column);
                        (*env)->DeleteLocalRef(env, column);
        }

        free(timeCount);
        free(pStore);
        free(buffer);

        return array;
}
JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_runCounterChannelWithTrigger
(JNIEnv *env, jclass cls, jint encoderBoard, jint channel, jint digitalBoard, jint triggerType, jfloat time)
{
        jintArray                                                                       array;
        int
        i, page_size;
        int
        result;
        int
                        encoderboard;
        int
                        digitalboard;
        int
                        loggNo;
```

```
int
                channelNo;
char*
buffer;
struct
rusage usage;
T116_COUNTER_BUFFER     countBuf;
T116_TIMER_BUFFER       timerBuf;
T600_EVRD_BUFFER                                        evBuf;
encoderboard = (int)                            encoderBoard;
digitalboard = (int)                            digitalBoard;
channelNo = (int)                                       channel;

if(time <= 0.5 ){
                loggNo = (int) (0.5*127000);
}else if(time >=30){
                loggNo = (int) (30*127000);
}else{
                loggNo = (int) (time*127000);
}

if (mlockall(MCL_CURRENT | MCL_FUTURE )){
                //perror("mlockall failed:");
                return NULL;
}

page_size = sysconf(_SC_PAGESIZE);
buffer = malloc(SOMESIZE);
// Touch each page in this piece of memory to get it mapped into RAM
for (i=0; i < SOMESIZE; i+=page_size)
{
                buffer[i] = 0;
                getrusage(RUSAGE_SELF, &usage);
}
// buffer is never released, or swapped, so using it from now will never lead to any pagefault

long *pStore;
pStore =(long*)malloc((loggNo+1) *sizeof(*pStore));
if( pStore == NULL){
                //printf("not enough memory\n");
                return NULL;
}
memset(pStore,0,sizeof((loggNo+1)*sizeof(*pStore)));

unsigned short *timeCount;
timeCount =(unsigned short*)malloc((loggNo+1)*sizeof(*timeCount));
if(timeCount == NULL){
                return NULL;
}
memset(timeCount,0,sizeof((loggNo+1)*sizeof(unsigned short)));

// Trigger logic
int trigger = (int) triggerType;
switch (trigger) {
case 1:
                // Waits for a high transition
                evBuf.mode = T600_HIGH_TR;
                break;
case 2:
                // Waits for a low transition
```

```
                    evBuf.mode = T600_LOW_TR;
                    break;
case 3:
                    // Waits for any transition
                    evBuf.mode = T600_ANY_TR;
                    break;
default:
                    // Waits for any transition
                    evBuf.mode = T600_ANY_TR;
                    break;
}
// some default specification for trigger: mask = position 1; timeout 60000 ticks = 60 sek
evBuf.mask = (unsigned short)(1<<0);
evBuf.timeout = 60000;

result = ioctl(digitalboard, T600_IOCX_EVENT_READ, &evBuf);
if (result >= 0){
                    // OK Trigger signal
}else{
                    //PrintErrorMessage();
                    // return null since read failed or timed out
                    free(timeCount);
                    free(pStore);
                    free(buffer);
                    return NULL;
}
/* Declare ourself as a real time task */
struct sched_param param;
param.sched_priority = RT_PRIORITY;
if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                    //perror("sched_setscheduler failed");
                    free(timeCount);
                    free(pStore);
                    free(buffer);
                    return NULL;
}

// Preload channel with value
                    countBuf.channel = channelNo;
                    countBuf.data = 0;
                    countBuf.flags = T116_IMMEDIATE_PRELOAD;

                    result = ioctl(encoderboard, T116_IOCS_SET_PRELOAD, &countBuf);
                    if (result >= 0){
                                        //Preload OK
                    }else{
                                        //PrintErrorMessage();
                                        param.sched_priority = NON_RT_PRIO;
                                        if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                                                            //perror("sched_setscheduler failed");
                                                            return NULL;
                                        }
                                        return NULL;
                    }

// Reading and storing values in a vector
for(i=0;i<loggNo;i++)
{
                    result = ioctl(encoderboard, T116_IOCG_READ_COUNTER, &countBuf);
                    if(result >= 0){
```

```
                        pStore[i] = countBuf.data;
                } else {
                        //ReadFunction failed
                        //PrintErrorMessage();
                        return NULL;
                }
                result = ioctl(encoderboard, T116_IOCG_READ_TIMER, &timerBuf);
                if(result >= 0){
                        timeCount[i] = timerBuf.value;
                } else {
                        // TimerRead failed
                        //PrintErrorMessage();
                        return NULL;
                }

        }
                // Set prio to NON_RT since we are out from RT-section
                param.sched_priority = NON_RT_PRIO;
                if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                        //perror("sched_setscheduler failed");
                        return NULL;
                }
                jclass intArrCls = (*env)->FindClass(env, "[I");
                if(intArrCls == NULL){
                        // Returning NULL since we have an exception here
                        return NULL;
                }

                array = (*env)->NewObjectArray(env, loggNo, intArrCls, NULL);
                if (array == NULL){
                        //out of memory
                        return NULL;
                }
                for(i=0;i<loggNo;i++)
                {
                        jint tempColumn[2];
                        jintArray column = (*env)->NewIntArray(env, 2);
                        if (column == NULL){
                                // out of memory
                                return NULL;
                        }
                        tempColumn[0] = pStore[i];
                        tempColumn[1] = timeCount[i];
                        (*env)->SetIntArrayRegion(env, column, 0, 2, tempColumn);
                        (*env)->SetObjectArrayElement(env, array, i, column);
                        (*env)->DeleteLocalRef(env, column);
                }

                free(timeCount);
                free(pStore);
                free(buffer);

                return array;
        }
```

```
//Method:    runAllCounterChannels
JNIEXPORT jobjectArray JNICALL
Java_se_ipbyran_tews_tip_TipJni_runAllCounterChannels(JNIEnv *env, jclass cls, jint boardNo, jfloat
time)
```

```
{
        jintArray                                                       array;
        int
        i, page_size;
        int
        result;
        int
                encoderBoard;
        int
                loggNo;
        char*
        buffer;
        struct
        rusage usage;
        T116_COUNTER_BUFFER     countBuf;
        T116_READ_ALL_BUFFER                allBuf;
        T116_TIMER_BUFFER                                       timerBuf;
        encoderBoard = (int) boardNo;

        if(time <= 0.5 ){
                loggNo = (int) (0.5*43000);
        }else if(time >=30){
                loggNo = (int) (30*43000);
        }else{
                loggNo = (int) (time*43000);
        }

        if (mlockall(MCL_CURRENT | MCL_FUTURE ))
        {
                //perror("mlockall failed:");
        }

        page_size = sysconf(_SC_PAGESIZE);
        buffer = malloc(SOMESIZE);
        if (buffer == NULL){
                return NULL;
        }
        // Touch each page in this piece of memory to get it mapped into RAM
        for (i=0; i < SOMESIZE; i+=page_size)
        {
                buffer[i] = 0;
                getrusage(RUSAGE_SELF, &usage);
        }
        // buffer is never released, or swapped, so using it from now will never lead to any pagefault

        long **pStore;
        pStore =(long**) malloc(5 * sizeof(long*));
        if(pStore == NULL)
        {
                //not enough memory
                return NULL;
        }
        for(i = 0; i < 5; i++)
        {
                pStore[i] =(long*) malloc((loggNo+1) * sizeof(long));
                memset(pStore[i],0,sizeof((loggNo+1)*sizeof(long)));
                if(pStore[i] == NULL)
                {
                        //not enough memory
                        return NULL;
```

```
                }
}
/* Declare ourself as a real time task */
struct sched_param param;
param.sched_priority = RT_PRIORITY;
if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                //perror("sched_setscheduler failed");
                for(i = 0; i < 5; i++)
                {
                                free(pStore[i]);
                }
                free(pStore);
                free(buffer);
                return NULL;
}
for(i=1; i<=4; i++)
{
                // Set counter preload value
                countBuf.channel = i;
                countBuf.data = 0;
                countBuf.flags = T116_IMMEDIATE_PRELOAD;

                result = ioctl(encoderBoard, T116_IOCS_SET_PRELOAD, &countBuf);
                if (result >= 0){
                                //Preload OK
                }else{
                                //PrintErrorMessage();
                                return NULL;
                }
}

// Reading and storing values( + timestamps) in a vector
for(i=0;i<loggNo;i++)
{
                result = ioctl(encoderBoard, T116_IOCG_READ_ALL_COUNTER, &allBuf);
                if(result >= 0){
                                pStore[0][i] = allBuf.data[0];
                                pStore[1][i] = allBuf.data[1];
                                pStore[2][i] = allBuf.data[2];
                                pStore[3][i] = allBuf.data[3];
                }else{
                                //ReadFunction failed
                                //PrintErrorMessage();
                                return NULL;
                }
                result = ioctl(encoderBoard, T116_IOCG_READ_TIMER, &timerBuf);
                if(result >= 0){
                                pStore[4][i] = (long) timerBuf.value;
                } else {
                                //TimerRead failed
                                //PrintErrorMessage();
                                return NULL;
                }
}

// Set timer as non RT priority
param.sched_priority = NON_RT_PRIO;
if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                //perror("sched_setscheduler failed");
                return NULL;
```

```
		}

		//Section for returning Matrix ( array of arrays)
		jclass intArrCls = (*env)->FindClass(env, "[I");
		if(intArrCls == NULL){
			// Returning NULL since we have an exception here
			return NULL;
		}

		array = (*env)->NewObjectArray(env, loggNo, intArrCls, NULL);
		if (array == NULL){
			//not enough memory
			return NULL;
		}
		for(i=0;i<loggNo;i++)
		{
			jint tempColumn[5];
			int j;
			jintArray column = (*env)->NewIntArray(env, 5);
			if (column == NULL){
				//not enough memory
				return NULL;
			}
			for(j=0;j<5;j++)
			{
				tempColumn[j] = (jint) pStore[j][i];
			}
			(*env)->SetIntArrayRegion(env, column, 0, 5, tempColumn);
			(*env)->SetObjectArrayElement(env, array, i, column);
			(*env)->DeleteLocalRef(env, column);
		}

		for(i = 0; i < 5; i++)
		{
			free(pStore[i]);
		}
		free(pStore);
		free(buffer);

		return array;
}
//Method:	runAllCountersWithTrigger
JNIEXPORT jobjectArray JNICALL
Java_se_ipbyran_tews_tip_TipJni_runAllCountersWithTrigger
(JNIEnv *env, jclass cls, jint encoderBoard, jint digitalBoard, jint triggerType, jfloat time)
{
		jintArray
array;
		int
		i, page_size;
		int
		result;
		int
			encoderboard;
		int
			digitalboard;
		int
			loggNo;
		char*
		buffer;
```

```
struct
rusage usage;
T116_COUNTER_BUFFER        countBuf;
T116_READ_ALL_BUFFER                    allBuf;
T116_TIMER_BUFFER                                   timerBuf;
T600_EVRD_BUFFER                        evBuf;
encoderboard = (int)                    encoderBoard;
digitalboard = (int)                    digitalBoard;

if(time <= 0.5 ){
                loggNo = (int) (0.5*43000);
}else if(time >=30){
                loggNo = (int) (30*43000);
}else{
                loggNo = (int) (time*43000);
}


if (mlockall(MCL_CURRENT | MCL_FUTURE )){
                //perror("mlockall failed:");
                return NULL;
}

page_size = sysconf(_SC_PAGESIZE);
buffer = malloc(SOMESIZE);
if (buffer == NULL){
                return NULL;
}
// Touch each page in this piece of memory to get it mapped into RAM
for (i=0; i < SOMESIZE; i+=page_size)
{
                buffer[i] = 0;
                getrusage(RUSAGE_SELF, &usage);
}
// buffer is never released, or swapped, so using it from now will never lead to any pagefault

long **pStore;
pStore =(long**) malloc(5 * sizeof(long*));
if(pStore == NULL)
{
                //not enough memory
                return NULL;
}
for(i = 0; i < 4; i++)
{
                pStore[i] =(long*) malloc((loggNo+1) * sizeof(long));
                memset(pStore[i],0,sizeof((loggNo+1)*sizeof(long)));
                if(pStore[i] == NULL)
                {
                                //not enough memory
                                return NULL;
                }
}
// Trigger logic
int trigger = (int) triggerType;
switch (trigger) {
                case 1:
                                // Waits for a high transition
                                evBuf.mode = T600_HIGH_TR;
                                break;
```

```
                                case 2:
                                        // Waits for a low transition
                                        evBuf.mode = T600_LOW_TR;
                                        break;
                                case 3:
                                        // Waits for any transition
                                        evBuf.mode = T600_ANY_TR;
                                        break;
                                default:
                                        // Waits for any transition
                                        evBuf.mode = T600_ANY_TR;
                                        break;
                }
                // some default specification for trigger: mask = position 1; timeout 60000 ticks = 60 s
                evBuf.mask = (unsigned short)(1<<0);
                evBuf.timeout = 60000;

result = ioctl(digitalboard, T600_IOCX_EVENT_READ, &evBuf);
if (result >= 0){
   // Trigger signal OK
} else {
   //Read input lines failed or timed out
   //PrintErrorMessage();
                /*for(i = 0; i < 5; i++)
                {
                        free(pStore[i]);
                }
                free(pStore);
                free(buffer);
                */
   return NULL;
}
/* Declare ourself as a real time task */
struct sched_param param;
param.sched_priority = RT_PRIORITY;
if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                //perror("sched_setscheduler failed");
                for(i = 0; i < 5; i++)
                {
                        free(pStore[i]);
                }
                free(pStore);
                free(buffer);
                return NULL;
}

for(i=1; i<=4; i++)
{
                //Set counter preload value
                countBuf.channel = i;
                countBuf.data = 0;
                countBuf.flags = T116_IMMEDIATE_PRELOAD;

                result = ioctl(encoderboard, T116_IOCS_SET_PRELOAD, &countBuf);
                if (result >= 0){
                        // Preload OK
                }else{
                        //PrintErrorMessage();
                        return NULL;
                }
```

```
}

// Reading and storing values( + timestamps) in a vector. RT critical section.
		for(i=0;i<loggNo;i++)
		{
				result = ioctl(encoderboard, T116_IOCG_READ_ALL_COUNTER, &allBuf);
				if(result >= 0){
						pStore[0][i] = allBuf.data[0];
						pStore[1][i] = allBuf.data[1];
						pStore[2][i] = allBuf.data[2];
						pStore[3][i] = allBuf.data[3];
				}else{
						//ReadFunction failed
						//PrintErrorMessage();
						return NULL;
				}
				result = ioctl(encoderboard, T116_IOCG_READ_TIMER, &timerBuf);
				if(result >= 0){
						pStore[4][i] = (long) timerBuf.value;
				} else {
						//TimerRead failed
						//PrintErrorMessage();
						return NULL;
				}
		}
		// Set timer as non RT priority
		param.sched_priority = NON_RT_PRIO;
		if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
				//perror("sched_setscheduler failed");
				return NULL;
		}


		//Section for returning Matrix ( array of arrays)
		jclass intArrCls = (*env)->FindClass(env, "[I");
		if(intArrCls == NULL){
				// Returning NULL since we have an exception here
				return NULL;
		}

		array = (*env)->NewObjectArray(env, loggNo, intArrCls, NULL);
		if (array == NULL){
				//not enough memory
				return NULL;
		}
		for(i=0;i<loggNo;i++)
		{
				jint tempColumn[5];
				int j;
				jintArray column = (*env)->NewIntArray(env, 5);
				if (column == NULL){
						//not enough memory
						return NULL;
				}
				for(j=0;j<5;j++)
				{
						tempColumn[j] = (jint) pStore[j][i];
				}
				(*env)->SetIntArrayRegion(env, column, 0, 5, tempColumn);
				(*env)->SetObjectArrayElement(env, array, i, column);
```

```
                        (*env)->DeleteLocalRef(env, column);
                }

                for(i = 0; i < 5; i++)
                {
                        free(pStore[i]);
                }
                free(pStore);
                free(buffer);

                return array;
}

//Method:    configureTimer
JNIEXPORT jboolean JNICALL
Java_se_ipbyran_tews_tip_TipJni_configureTimer (JNIEnv *env, jclass cls, jint boardNo)
{
                T116_TIMER_CONFIG_BUFFER    timerConfBuf;
                int
                result;
                int
                        boardNumber;

                timerConfBuf.preloadVal = 40000;
                timerConfBuf.prescaler = 1;
                timerConfBuf.enable = TRUE;
                boardNumber = boardNo;
                result = ioctl(boardNumber, T116_IOCS_CONFIG_TIMER, &timerConfBuf);
                if(result >= 0){
                        return (jboolean)1;
                        //printf("From C side: Timer config OK at 8MHz\n");
                }else{
                        //printf("Timer config failed  -->  Error = %d\n", errno);
                        //PrintErrorMessage();
                        return (jboolean)0;
                }

}
//Method:    configureChannel
JNIEXPORT jboolean JNICALL
Java_se_ipbyran_tews_tip_TipJni_configureChannel(JNIEnv *env, jclass cls, jint board, jint channelNo)
{
                T116_COUNTER_CONFIG_BUFFER              countConfBuf;
                int
                                        channelNumber;
                int                     result;
                int
                                        boardNo;
                boardNo =(int) board;
                channelNumber =(int) channelNo;

                // Setting default TIP116 values in a buffer
                // setting  Quadrature mode
                countConfBuf.inputMode = T116_QUADCOUNT;
                // No specialmode
                countConfBuf.specMode = T116_SPECMODENO;
                // No z-control
                countConfBuf.ZControl = T116_SPECMODENO;
                // Quadcontrol = 4x
                countConfBuf.QuadControl = T116_4X;
```

```
		//XYZpolarity = low
		countConfBuf.XYZPolarity = 0;
		countConfBuf.XYZPolarity |= T116_XLOWACTIV;
		countConfBuf.XYZPolarity |= T116_YLOWACTIV;
		countConfBuf.XYZPolarity |= T116_ZLOWACTIV;
		// Set channel number
		countConfBuf.channel = channelNumber;


		// Call to Card with buffer address
		result = ioctl(boardNo, T116_IOCS_CONFIG_COUNTER, &countConfBuf);
		if (result >= 0){
			//Configure EncoderChannel OK
			return (jboolean)1;
		}else{
			// Configure EncoderChannel failed
			//PrintErrorMessage();
			return (jboolean)0;
		}
}
//Method:    systemPerformanceTest
JNIEXPORT jboolean JNICALL
Java_se_ipbyran_tews_tip_TipJni_systemPerformanceTest(JNIEnv *env, jclass cls, jint boardNo, jint
channelNo)
{
		int
		i, page_size, result;
		int
		boardNumber = (int) boardNo;
		char*
		buffer;
		int
			loggNo;
		struct rusage                                   usage;
		T116_TIMER_BUFFER                               timerBuf;
		T116_COUNTER_BUFFER                 countBuf;

		loggNo = 4000000;
		// Now lock all current and future pages from preventing of being paged
		if (mlockall(MCL_CURRENT | MCL_FUTURE )){
			//perror("mlockall failed:");
		}

		page_size = sysconf(_SC_PAGESIZE);
		buffer = malloc(SOMESIZE);
		// Touch each page in this piece of memory to get it mapped into RAM
		for (i=0; i < SOMESIZE; i+=page_size)
		{
			buffer[i] = 0;
			getrusage(RUSAGE_SELF, &usage);
		}
		printf("From C side: Major-pagefaults:%ld, Minor Pagefaults:%ld\n", usage.ru_majflt,
usage.ru_minflt);

			/* Declare ourself as a real time task */
		struct sched_param param;
		param.sched_priority = RT_PRIORITY;
		if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
			perror("sched_setscheduler failed");
			return (jboolean)0;
```

```
		}
		printf("Priority for program: %d.\n\n", RT_PRIORITY);

		printf("Set counter preload value: 5000 ...\n\n ");
		countBuf.channel = channelNo;
		countBuf.data = 5000;
		countBuf.flags = T116_IMMEDIATE_PRELOAD;

		result = ioctl(boardNo, T116_IOCS_SET_PRELOAD, &countBuf);
		if (result >= 0){
				printf(" Preload: OK\n\n");
		}else{
				printf(" Preload failed  -->  Error = %d\n", errno);
				PrintErrorMessage();
				return (jboolean)0;
		}

		/* Allocate memory for the storage Vectors */
long *pStore;
pStore =(long*)malloc((loggNo+1) *sizeof(*pStore));
if( pStore == NULL){
		//not enough memory
		return (jboolean)0;
}

unsigned short *timeCount;
timeCount =(unsigned short*)malloc((loggNo+1)* sizeof(*timeCount));
if( timeCount == NULL){
		//not enough memory
		return (jboolean)0;
}

//Reading and storing values in a vector
		for(i=0;i<loggNo;i++)
		{
				result = ioctl(boardNumber, T116_IOCG_READ_COUNTER, &countBuf);
				if(result >= 0){
						pStore[i] = countBuf.data;
				}else{
						printf("ReadFunction failed  -->  Error = %d\n", errno);
						//PrintErrorMessage();
						return (jboolean)0;
				}
				result = ioctl(boardNumber, T116_IOCG_READ_TIMER, &timerBuf);
				if(result >= 0){
						timeCount[i] = timerBuf.value;
				}else{
						printf("TimerRead failed  -->  Error = %d\n", errno);
						//PrintErrorMessage();
						return (jboolean)0;
				}
		}

		int badcount = 0;
		int shifts = 0;
		int worst = 0;
		long total = 0;
		float average = 0;

		for(i=0; i<loggNo;i++){
```

```
                        if ((timeCount[i]-timeCount[i+1]) >100){
                                badcount++;
                                if((timeCount[i]-timeCount[i+1])> worst){
                                        worst = timeCount[i]-timeCount[i+1];
                                }
                        }
                        if ((timeCount[i]-timeCount[i+1]) >0){
                                total = total + (timeCount[i]-timeCount[i+1]);
                        }else{
                                total = total + (timeCount[i]+ (40000-timeCount[i+1]));
                        }
                        if((timeCount[i]-timeCount[i+1]) <0){
                                shifts++;
                        }
                }
                average = (total/loggNo);

                for(i=0; i<50;i++)
                {
                        printf("TimeInterval number %d  from timeCount: %d\n", i, timeCount[i]-
timeCount[i+1]);
                        printf("Logging values number %d  from timeCount: %ld\n",i, pStore[i]);
                }
                printf("\nSystem preformance test One Channel\n");
                printf("\nNumber of logging values: %d\n",loggNo);
                printf("Number of interval over 100 clockticks: %d\n", badcount);
                printf("Number of clock transitions from 0 to 40000: %d\n", shifts);
                printf("Time take for test in seconds : %f\n",((float) (shifts*0.005)));
                printf("Worst interval i clocktics: %d\n", worst);
                printf("Worst interval i micros: %f\n",(float) (worst*0.125));
                printf("Average number clock ticks between reads %f\n", average);
                printf("Average time in micros between reads %f\n",(float) (average*0.125));
                printf("Logging frequency %f\n\n",(float) 1/(average*0.000000125));

                free(pStore);
                free(timeCount);
                free(buffer);

                return (jboolean)1;
}

//Method:    runAnalogChannel
JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_runAnalogChannel
                (JNIEnv *env, jclass cls, jint analogBoard, jint channel, jint encoderBoard, jint conversionType,
jfloat time)
{
                jintArray                                                                       array;
                int
                i, page_size;
                int
                result;
                int
                        encoderboard;
                int
                analogboard;
                int
                        loggNo;
                int                             NumBytes = 0;
                char*
                buffer;
```

```
struct
rusage usage;
T501_READ_BUFFER                                        rdBuf;
T116_TIMER_BUFFER       timerBuf;
T501_INFO_BUFFER                                        infoBuf;
encoderboard = (int)                            encoderBoard;
analogboard = (int)                                     analogBoard;

if(time <= 0.5 ){
                loggNo = (int) (0.5*48000);
}else if(time >=30){
                loggNo = (int) (30*48000);
}else{
                loggNo = (int) (time*48000);
}


if (mlockall(MCL_CURRENT | MCL_FUTURE )){
                //perror("mlockall failed:");
                return NULL;
}

page_size = sysconf(_SC_PAGESIZE);
buffer = malloc(SOMESIZE);
// Touch each page in this piece of memory to get it mapped into RAM
for (i=0; i < SOMESIZE; i+=page_size)
{
                buffer[i] = 0;
                getrusage(RUSAGE_SELF, &usage);
}
// buffer is never released, or swapped, so using it from now will never lead to any pagefault

//information about the variant for TIP501
result = ioctl(analogboard, T501_IOCG_INFO, &infoBuf);
if (result < 0) {
                /* This error should never occur if open was successful, but who knows... */
                //Read device information failed
                //PrintErrorMessage();
                return NULL;
}else{
                // OK
}

int *pStore;
pStore =(int*)malloc((loggNo+1) * sizeof(*pStore));
if(pStore == NULL){
                //not enough memory
                return NULL;
}
memset(pStore,0,sizeof((loggNo+1)*sizeof(*pStore)));

unsigned short *timeCount;
timeCount =(unsigned short*)malloc((loggNo+1)* sizeof(*timeCount));
if( timeCount == NULL){
                //not enough memory
                return NULL;
}
memset(timeCount,0,sizeof((loggNo+1)*sizeof(unsigned short)));

rdBuf.chan = 1;
```

```
rdBuf.gain = 1;
rdBuf.flags = T501_DIFF | T501_PIPELINE;
rdBuf.data = 0;

/* Declare ourself as a real time task */
struct sched_param param;
param.sched_priority = RT_PRIORITY;
if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                //perror("sched_setscheduler failed");

                free(timeCount);
                free(pStore);
                free(buffer);
                return NULL;
}
//Reading and storing values in a vector
for(i=0;i<loggNo;i++)
{
                NumBytes = read(analogboard, &rdBuf, sizeof(rdBuf));
                if (NumBytes > 0){
                                pStore[i] = rdBuf.data;
                }else {
                                //Analog read failed
                                //PrintErrorMessage();
                                return NULL;
                }
                result = ioctl(encoderboard, T116_IOCG_READ_TIMER, &timerBuf);
                if(result >= 0){
                                timeCount[i] = timerBuf.value;
                }else{
                                //TimerCount failed
                                //PrintErrorMessage();
                                return NULL;
                }
}
// Set prio to NON_RT since we are out from RT-section
param.sched_priority = NON_RT_PRIO;
if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                //perror("sched_setscheduler failed");
                return NULL;
}
jclass intArrCls = (*env)->FindClass(env, "[I");
if(intArrCls == NULL){
                // Returning NULL since we have an exception here
                return NULL;
}

array = (*env)->NewObjectArray(env, loggNo, intArrCls, NULL);
if (array == NULL){
                //not enough memory
                return NULL;
}
for(i=0;i<loggNo;i++)
{
                jint tempColumn[2];
                jintArray column = (*env)->NewIntArray(env, 2);
                if (column == NULL){
                                //not enough memory
                                return NULL;
                }
```

```
                        tempColumn[0] = pStore[i];
                        tempColumn[1] = timeCount[i];
                        (*env)->SetIntArrayRegion(env, column, 0, 2, tempColumn);
                        (*env)->SetObjectArrayElement(env, array, i, column);
                        (*env)->DeleteLocalRef(env, column);
                }

                free(timeCount);
                free(pStore);
                free(buffer);

                return array;
}
//Method:       runAnalogChannelWithCounter
JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_runAnalogChannelWithTrigger
                (JNIEnv *env, jclass cls, jint analogBoard, jint channel, jint encoderBoard, jint digitalBoard, jint
triggerType, jint conversionType, jfloat time)
{
                jintArray                                                                       array;
                int
                i, page_size;
                char*
                buffer;
                struct
                rusage usage;
                int
                result;
                int
                        encoderboard;
                int
                analogboard;
                int
                digitalboard;
                int
                        loggNo;
                int                     NumBytes = 0;
                T501_READ_BUFFER                                        rdBuf;
                T501_INFO_BUFFER                                        infoBuf;
                T116_TIMER_BUFFER       timerBuf;
                T600_EVRD_BUFFER                                        evBuf;
                encoderboard = (int)                            encoderBoard;
                analogboard = (int)                                     analogBoard;
                digitalboard = (int)                            digitalBoard;

                if(time <= 0.5 ){
                        loggNo = (int) (0.5*48000);
                }else if(time >=30){
                        loggNo = (int) (30*48000);
                }else{
                        loggNo = (int) (time*48000);
                }

                if (mlockall(MCL_CURRENT | MCL_FUTURE )){
                        //perror("mlockall failed:");
                        return NULL;
                }

                page_size = sysconf(_SC_PAGESIZE);
                buffer = malloc(SOMESIZE);
                // Touch each page in this piece of memory to get it mapped into RAM
```

```
for (i=0; i < SOMESIZE; i+=page_size)
{
                    buffer[i] = 0;
                    getrusage(RUSAGE_SELF, &usage);
}
// buffer is never released, or swapped, so using it from now will never lead to any pagefault

//information about the variant for TIP501
result = ioctl(analogboard, T501_IOCG_INFO, &infoBuf);
if (result < 0) {
                    /* This error should never occur if open was successful, but who knows... */
                    //Read device information failed
                    //PrintErrorMessage();
                    return NULL;
}else{
                    // OK
}

int *pStore;
pStore =(int*)malloc((loggNo+1) * sizeof(*pStore));
if(pStore == NULL)
{
                    //not enough memory
                    return NULL;
}
memset(pStore,0,sizeof((loggNo+1)*sizeof(*pStore)));

unsigned short *timeCount;
timeCount =(unsigned short*)malloc((loggNo+1)* sizeof(*timeCount));
if( timeCount == NULL){
                    //printf("not enough memory\n");
                    return NULL;
}
memset(timeCount,0,sizeof((loggNo+1)*sizeof(unsigned short)));

rdBuf.chan = 1;
rdBuf.gain = 1;
rdBuf.flags = T501_DIFF | T501_PIPELINE;
rdBuf.data = 0;
// Trigger logic
int trigger = (int)triggerType;
switch (trigger) {
case 1:
                    // Waits for a high transition
                    evBuf.mode = T600_HIGH_TR;
                    break;
case 2:
                    // Waits for a low transition
                    evBuf.mode = T600_LOW_TR;
                    break;
case 3:
                    // Waits for any transition
                    evBuf.mode = T600_ANY_TR;
                    break;
default:
                    // Waits for any transition
                    evBuf.mode = T600_ANY_TR;
                    break;
}
// some default specification for trigger: mask = position 1; timeout 60000 ticks = 60s
```

```
evBuf.mask = (unsigned short)(1<<0);
evBuf.timeout = 60000;

result = ioctl(digitalboard, T600_IOCX_EVENT_READ, &evBuf);
if (result >= 0){
                    // Trigger OK
} else {
                    // read failed or timed out
                    //PrintErrorMessage();
                    free(timeCount);
                    free(pStore);
                    free(buffer);
                    return NULL;
}
/* Declare ourself as a real time task */
struct sched_param param;
param.sched_priority = RT_PRIORITY;
if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                    //perror("sched_setscheduler failed");
                    free(timeCount);
                    free(pStore);
                    free(buffer);
                    return NULL;
}
//Reading and storing values in a vector
for(i=0;i<loggNo;i++)
{
                    NumBytes = read(analogboard, &rdBuf, sizeof(rdBuf));

                    if (NumBytes > 0){
                                        pStore[i] = rdBuf.data;
                    }else {
                                        //nAnalog read failed
                                        //PrintErrorMessage();
                    }

                    result = ioctl(encoderboard, T116_IOCG_READ_TIMER, &timerBuf);
                    if(result >= 0){
                                        timeCount[i] = timerBuf.value;
                    }else{
                                        //TimerRead failed
                                        //PrintErrorMessage();
                                        return NULL;
                    }
}
// Set prio to NON_RT since we are out from RT-section
param.sched_priority = NON_RT_PRIO;
if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                    //perror("sched_setscheduler failed");
                    return NULL;
}
jclass intArrCls = (*env)->FindClass(env, "[I");
if(intArrCls == NULL){
                    // Returning NULL since we have an exception here
                    return NULL;
}

array = (*env)->NewObjectArray(env, loggNo, intArrCls, NULL);
if (array == NULL){
                    // not enought memory
```

```
                        return NULL;
                }
                for(i=0;i<loggNo;i++)
                {
                        jint tempColumn[2];
                        jintArray column = (*env)->NewIntArray(env, 2);
                        if (column == NULL){
                                // out of memory
                                return NULL;
                        }
                        tempColumn[0] = pStore[i];
                        tempColumn[1] = timeCount[i];
                        (*env)->SetIntArrayRegion(env, column, 0, 2, tempColumn);
                        (*env)->SetObjectArrayElement(env, array, i, column);
                        (*env)->DeleteLocalRef(env, column);
                }

                free(timeCount);
                free(pStore);
                free(buffer);

                return array;
}

//Method:    runAllAnalogChannels
JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_runAllAnalogChannels
                (JNIEnv *env, jclass cls, jint analogBoard, jint encoderBoard, jint conversionType, jfloat time)
{
                jintArray                                                                       array;
                int
                i, page_size;
                char*
                buffer;
                struct
                rusage usage;
                int
                result;
                int
                        encoderboard;
                int
                analogboard;
                int
                        loggNo;
                int                             NumBytes = 0;
                T501_READ_BUFFER                                        rdBuf;
                T501_READ_BUFFER                                        rdBuf2;
                T501_INFO_BUFFER                                        infoBuf;
                T116_TIMER_BUFFER       timerBuf;
                encoderboard = (int)                            encoderBoard;
                analogboard = (int)                                     analogBoard;

                if(time <= 0.5 ){
                        loggNo = (int) (0.5*15500);
                }else if(time >=30){
                        loggNo = (int) (30*15500);
                }else{
                        loggNo = (int) (time*15500);
                }
                //printf("From C-side: Number och values to logg: %d\n", loggNo);
```

```
if (mlockall(MCL_CURRENT | MCL_FUTURE )){
                    //perror("mlockall failed:");
}

page_size = sysconf(_SC_PAGESIZE);
buffer = malloc(SOMESIZE);
// Touch each page in this piece of memory to get it mapped into RAM
for (i=0; i < SOMESIZE; i+=page_size)
{
                    buffer[i] = 0;
                    getrusage(RUSAGE_SELF, &usage);
}
// buffer is never released, or swapped, so using it from now will never lead to any pagefault

//information about the variant for TIP501
result = ioctl(analogboard, T501_IOCG_INFO, &infoBuf);
if (result < 0) {
                    /* This error should never occur if open was successful, but who knows... */
                    //Read device information failed
                    //PrintErrorMessage();
                    return NULL;
}else{
                    // OK
}

int *pStore;
pStore =(int*)malloc((loggNo+1) * sizeof(*pStore));
if(pStore == NULL){
                    //not enough memory
                    return NULL;
}
memset(pStore,0,sizeof((loggNo+1)*sizeof(*pStore)));

int *pStore2;
pStore2 =(int*)malloc((loggNo+1) * sizeof(*pStore2));
if(pStore2 == NULL){
                    //not enough memory
                    return NULL;
}
memset(pStore2,0,sizeof((loggNo+1)*sizeof(*pStore2)));

unsigned short *timeCount;
timeCount =(unsigned short*)malloc((loggNo+1)* sizeof(*timeCount));
if( timeCount == NULL){
                    //not enough memory
                    return NULL;
}
memset(timeCount,0,sizeof((loggNo+1)*sizeof(unsigned short)));

rdBuf.chan = 1;
rdBuf.gain = 1;
rdBuf.flags = T501_DIFF | T501_PIPELINE;
rdBuf.data = 2;

rdBuf.chan = 2;
rdBuf.gain = 1;
rdBuf.flags = T501_DIFF | T501_PIPELINE;
rdBuf.data = 2;
```

```
/* Declare ourself as a real time task */
struct sched_param param;
param.sched_priority = RT_PRIORITY;
if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
		//perror("sched_setscheduler failed");
		free(timeCount);
		free(pStore);
		free(pStore2);
		free(buffer);
		return NULL;
}
//Reading and storing values in a vector
for(i=0;i<loggNo;i++)
{
		NumBytes = read(analogboard, &rdBuf, sizeof(rdBuf));
		if (NumBytes > 0){
			pStore[i] = rdBuf.data;
		}else {
			//Analog read failed
			//PrintErrorMessage();
			return NULL;
		}
		NumBytes = read(analogboard, &rdBuf, sizeof(rdBuf));
		if (NumBytes > 0){
			pStore2[i] = rdBuf2.data;
		}else {
			//Analog read failed
			//PrintErrorMessage();
			return NULL;
		}

		result = ioctl(encoderboard, T116_IOCG_READ_TIMER, &timerBuf);
		if(result >= 0){
			timeCount[i] = timerBuf.value;
		}else{
			//TimerRead failed
			//PrintErrorMessage();
			return NULL;
		}
}
// Set prio to NON_RT since we are out from RT-section
param.sched_priority = NON_RT_PRIO;
if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
		//perror("sched_setscheduler failed");
		return NULL;
}
jclass intArrCls = (*env)->FindClass(env, "[I");
if(intArrCls == NULL){
		//not enough memory
		return NULL;
}

array = (*env)->NewObjectArray(env, loggNo, intArrCls, NULL);
if (array == NULL){
		//not enough memory
		return NULL;
}
for(i=0;i<loggNo;i++)
{
		jint tempColumn[3];
```

```
                        jintArray column = (*env)->NewIntArray(env, 3);
                        if (column == NULL){
                                // out of memory
                                return NULL;
                        }
                        tempColumn[0] = pStore[i];
                        tempColumn[1] = pStore[i];
                        tempColumn[2] = timeCount[i];
                        (*env)->SetIntArrayRegion(env, column, 0, 3, tempColumn);
                        (*env)->SetObjectArrayElement(env, array, i, column);
                        (*env)->DeleteLocalRef(env, column);
                }

                free(timeCount);
                free(pStore);
                free(pStore2);
                free(buffer);

                return array;
}
//Method        runAllAnalogChannelsWithTrigger
JNIEXPORT jobjectArray JNICALL
Java_se_ipbyran_tews_tip_TipJni_runAllAnalogChannelsWithTrigger
                (JNIEnv *env, jclass cls, jint analogBoard, jint encoderBoard, jint digitalBoard, jint triggerType,
jint conversionType, jfloat time)
{
                jintArray                                                       array;
                int
                i, page_size;
                char*
                buffer;
                struct
                rusage usage;
                int
                result;
                int
                        encoderboard;
                int
                analogboard;
                int
                digitalboard;
                int
                        loggNo;
                int                     NumBytes = 0;
                T501_READ_BUFFER                                        rdBuf;
                T501_READ_BUFFER                                        rdBuf2;
                T501_INFO_BUFFER                                        infoBuf;
                T116_TIMER_BUFFER       timerBuf;
                T600_EVRD_BUFFER                                        evBuf;
                encoderboard = (int)                            encoderBoard;
                analogboard = (int)                                     analogBoard;
                digitalboard = (int)                            digitalBoard;

                if(time <= 0.5 ){
                        loggNo = (int) (0.5*15500);
                }else if(time >=30){
                        loggNo = (int) (30*15500);
                }else{
                        loggNo = (int) (time*15500);
                }
```

```
if (mlockall(MCL_CURRENT | MCL_FUTURE )){
                    //perror("mlockall failed:");
}

page_size = sysconf(_SC_PAGESIZE);
buffer = malloc(SOMESIZE);
// Touch each page in this piece of memory to get it mapped into RAM
for (i=0; i < SOMESIZE; i+=page_size)
{
                    buffer[i] = 0;
                    getrusage(RUSAGE_SELF, &usage);
}
// buffer is never released, or swapped, so using it from now will never lead to any pagefault


//information about the variant for TIP501
result = ioctl(analogboard, T501_IOCG_INFO, &infoBuf);
if (result < 0) {
                    /* This error should never occur if open was successful, but who knows... */
                    //Read device information failed
                    //PrintErrorMessage();
                    return NULL;
}else{
                    // OK
}

int *pStore;
pStore =(int*)malloc((loggNo+1) * sizeof(*pStore));
if(pStore == NULL)
{
                    //not enough memory
                    return NULL;
}
memset(pStore,0,sizeof((loggNo+1)*sizeof(*pStore)));

int *pStore2;
pStore2 =(int*)malloc((loggNo+1) * sizeof(*pStore2));
if(pStore == NULL)
{
                    //not enough memory
                    return NULL;
}
memset(pStore2,0,sizeof((loggNo+1)*sizeof(*pStore2)));

unsigned short *timeCount;
timeCount =(unsigned short*)malloc((loggNo+1)* sizeof(*timeCount));
if( timeCount == NULL){
                    //not enough memory
                    return NULL;
}
memset(timeCount,0,sizeof((loggNo+1)*sizeof(unsigned short)));

rdBuf.chan = 1;
rdBuf.gain = 1;
rdBuf.flags = T501_DIFF | T501_PIPELINE;
rdBuf.data = 2;

rdBuf2.chan = 2;
rdBuf2.gain = 1;
```

```
rdBuf2.flags = T501_DIFF | T501_PIPELINE;
rdBuf2.data = 2;

// Trigger logic
int trigger = (int)triggerType;
switch (trigger) {
case 1:
                    // Waits for a high transition
                    evBuf.mode = T600_HIGH_TR;
                    break;
case 2:
                    // Waits for a low transition
                    evBuf.mode = T600_LOW_TR;
                    break;
case 3:
                    // Waits for any transition
                    evBuf.mode = T600_ANY_TR;
                    break;
default:
                    // Waits for any transition
                    evBuf.mode = T600_ANY_TR;
                    break;
}
// some default specification for trigger: mask = position 1; timeout 60000 ticks = 60 s
evBuf.mask = (unsigned short)(1<<0);
evBuf.timeout = 60000;

result = ioctl(digitalboard, T600_IOCX_EVENT_READ, &evBuf);
if (result >= 0){
                    //Trigger OK
} else {
                    //PrintErrorMessage();
                    // return null since read failed or timed out
                    return NULL;
}
/* Declare ourself as a real time task */
struct sched_param param;
param.sched_priority = RT_PRIORITY;
if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                    //perror("sched_setscheduler failed");
                    free(timeCount);
                    free(pStore);
                    free(pStore2);
                    free(buffer);
                    return NULL;
}

//Reading and storing values in a vector
for(i=0;i<loggNo;i++)
{
                    NumBytes = read(analogboard, &rdBuf, sizeof(rdBuf));
                    if (NumBytes > 0){
                                        pStore[i] = rdBuf.data;
                    }else {
                                        //Analog read failed
                                        //PrintErrorMessage();
                                        return NULL;
                    }
                    NumBytes = read(analogboard, &rdBuf, sizeof(rdBuf));
                    if (NumBytes > 0){
```

```
                                        pStore2[i] = rdBuf2.data;
                                }else {
                                        // Analog read failed
                                        //PrintErrorMessage();
                                        return NULL;
                                }

                                result = ioctl(encoderboard, T116_IOCG_READ_TIMER, &timerBuf);
                                if(result >= 0){
                                        timeCount[i] = timerBuf.value;
                                }else{
                                        //TimerRead failed
                                        //PrintErrorMessage();
                                        return NULL;
                                }
                        }
                        // Set prio to NON_RT since we are out from RT-section
                        param.sched_priority = NON_RT_PRIO;
                        if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                                //perror("sched_setscheduler failed");
                                return NULL;
                        }
                        jclass intArrCls = (*env)->FindClass(env, "[I");
                        if(intArrCls == NULL){
                                // Returning NULL since we have an exception here
                                return NULL;
                        }

                        array = (*env)->NewObjectArray(env, loggNo, intArrCls, NULL);
                        if (array == NULL){
                                //not enough memory
                                return NULL;
                        }
                        for(i=0;i<loggNo;i++)
                        {
                                jint tempColumn[3];
                                jintArray column = (*env)->NewIntArray(env, 3);
                                if (column == NULL){
                                        //not enough memory
                                        return NULL;
                                }
                                tempColumn[0] = pStore[i];
                                tempColumn[1] = pStore[i];
                                tempColumn[2] = timeCount[i];
                                (*env)->SetIntArrayRegion(env, column, 0, 3, tempColumn);
                                (*env)->SetObjectArrayElement(env, array, i, column);
                                (*env)->DeleteLocalRef(env, column);
                        }

                        free(timeCount);
                        free(pStore);
                        free(pStore2);
                        free(buffer);

                        return array;
}

//Method:    runDigitalInput
JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_runDigitalInput
                        (JNIEnv *env, jclass cls, jint digitalBoard, jint encoderBoard, jfloat time)
```

```
{
        jintArray                                                       array;
        int
        i, page_size;
        char*
        buffer;
        struct
        rusage usage;
        int
        result;
        int
                encoderboard;
        int
        digitalboard;
        int
                loggNo;
        int                      NumBytes = 0;
        T116_TIMER_BUFFER       timerBuf;
        T600_READ_BUFFER                                                readBuf;
        encoderboard = (int)                            encoderBoard;
        digitalboard = (int)                            digitalBoard;

        if(time <= 0.5 ){
                loggNo = (int) (0.5*160000);
        }else if(time >=30){
                loggNo = (int) (30*160000);
        }else{
                loggNo = (int) (time*160000);
        }

        if (mlockall(MCL_CURRENT | MCL_FUTURE )){
                //perror("mlockall failed:");
        }

        page_size = sysconf(_SC_PAGESIZE);
        buffer = malloc(SOMESIZE);
        // Touch each page in this piece of memory to get it mapped into RAM
        for (i=0; i < SOMESIZE; i+=page_size)
        {
                buffer[i] = 0;
                getrusage(RUSAGE_SELF, &usage);
        }
        // buffer is never released, or swapped, so using it from now will never lead to any pagefault

        unsigned short *pStore;
        pStore =(unsigned short*)malloc((loggNo+1) * sizeof(*pStore));
        if(pStore == NULL)
        {
                //not enough memory
                return NULL;
        }
        memset(pStore,0,sizeof((loggNo+1)*sizeof(*pStore)));

        unsigned short *timeCount;
        timeCount =(unsigned short*)malloc((loggNo+1)* sizeof(*timeCount));
        if( timeCount == NULL){
                //not enough memory
                return NULL;
        }
        memset(timeCount,0,sizeof((loggNo+1)*sizeof(unsigned short)));
```

```
/* Declare ourself as a real time task */
struct sched_param param;
param.sched_priority = RT_PRIORITY;
if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                    //perror("sched_setscheduler failed");
                    free(timeCount);
                    free(pStore);
                    free(buffer);
                    return NULL;
}

//Reading and storing values in a vector
for(i=0;i<loggNo;i++)
{
                    NumBytes = read(digitalboard, &readBuf, sizeof(readBuf));
                    if (NumBytes > 0){
                                        pStore[i] = readBuf.value;
                    }else {
                                        //Read input lines failed
                                        //PrintErrorMessage();
                                        return NULL;
                    }

                    result = ioctl(encoderboard, T116_IOCG_READ_TIMER, &timerBuf);
                    if(result >= 0){
                                        timeCount[i] = timerBuf.value;
                    }else{
                                        //TimerRead failed
                                        //PrintErrorMessage();
                                        return NULL;
                    }
}

// Set prio to NON_RT since we are out from RT-section
param.sched_priority = NON_RT_PRIO;
if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                    //perror("sched_setscheduler failed");
                    return NULL;
}

jclass intArrCls = (*env)->FindClass(env, "[I");
if(intArrCls == NULL){
                    //not enough memory
                    return NULL;
}

array = (*env)->NewObjectArray(env, loggNo, intArrCls, NULL);
if (array == NULL){
                    //not enough memory
                    return NULL;
}
for(i=0;i<loggNo;i++)
{
                    jint tempColumn[2];
                    jintArray column = (*env)->NewIntArray(env, 2);
                    if (column == NULL){
                                        //not enough memory
                                        return NULL;
                    }
```

```
                        tempColumn[0] = pStore[i];
                        tempColumn[1] = timeCount[i];
                        (*env)->SetIntArrayRegion(env, column, 0, 2, tempColumn);
                        (*env)->SetObjectArrayElement(env, array, i, column);
                        (*env)->DeleteLocalRef(env, column);
                }

                free(timeCount);
                free(pStore);
                free(buffer);

                return array;
}

//Method:    runDigitalInput
JNIEXPORT jobjectArray JNICALL Java_se_ipbyran_tews_tip_TipJni_runDigitalInputWithTrigger
                (JNIEnv *env, jclass cls, jint encoderBoard, jint digitalBoard, jint triggerType, jfloat time)
{
                jintArray                                                                       array;
                int
                i, page_size;
                char*
                buffer;
                struct
                rusage usage;
                int
                result;
                int
                        encoderboard;
                int
                digitalboard;
                int
                        loggNo;
                int                      NumBytes = 0;
                T116_TIMER_BUFFER        timerBuf;
                T600_EVRD_BUFFER                                        evBuf;
                T600_READ_BUFFER                                        readBuf;
                encoderboard = (int)                    encoderBoard;
                digitalboard = (int)                    digitalBoard;

                if(time <= 0.5 ){
                        loggNo = (int) (0.5*160000);
                }else if(time >=30){
                        loggNo = (int) (30*160000);
                }else{
                        loggNo = (int) (time*160000);
                }

                if (mlockall(MCL_CURRENT | MCL_FUTURE )){
                        //perror("mlockall failed:");
                }

                page_size = sysconf(_SC_PAGESIZE);
                buffer = malloc(SOMESIZE);
                // Touch each page in this piece of memory to get it mapped into RAM
                for (i=0; i < SOMESIZE; i+=page_size)
                {
                        buffer[i] = 0;
                        getrusage(RUSAGE_SELF, &usage);
                }
```

```
// buffer is never released, or swapped, so using it from now will never lead to any pagefault


int *pStore;
pStore =(int*)malloc((loggNo+1) * sizeof(*pStore));
if(pStore == NULL)
{
                //not enough memory
                return NULL;
}
memset(pStore,0,sizeof((loggNo+1)*sizeof(*pStore)));

unsigned short *timeCount;
timeCount =(unsigned short*)malloc((loggNo+1)* sizeof(*timeCount));
if( timeCount == NULL){
                //not enough memory
                return NULL;
}
memset(timeCount,0,sizeof((loggNo+1)*sizeof(unsigned short)));

// Trigger logic
int trigger = (int)triggerType;
switch (trigger) {
case 1:
                // Waits for a high transition
                evBuf.mode = T600_HIGH_TR;
                break;
case 2:
                // Waits for a low transition
                evBuf.mode = T600_LOW_TR;
                break;
case 3:
                // Waits for any transition
                evBuf.mode = T600_ANY_TR;
                break;
default:
                // Waits for any transition
                evBuf.mode = T600_ANY_TR;
                break;
}
// some default specification for trigger: mask = position 1; timeout 60000 ticks = 60 s
evBuf.mask = (unsigned short)(1<<0);
evBuf.timeout = 60000;

result = ioctl(digitalboard, T600_IOCX_EVENT_READ, &evBuf);
if (result >= 0){
                // Trigger OK
} else {
                //PrintErrorMessage();
                // return null since read failed or timed out
                return NULL;
}
/* Declare ourself as a real time task */
struct sched_param param;
param.sched_priority = RT_PRIORITY;
if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                //perror("sched_setscheduler failed");
                free(timeCount);
                free(pStore);
                free(buffer);
```

```
                return NULL;
}

//Reading and storing values in a vector
for(i=0;i<loggNo;i++)
{
                NumBytes = read(digitalboard, &readBuf, sizeof(readBuf));

                if (NumBytes > 0){
                                pStore[i] = readBuf.value;
                }else{
                                //Read input lines failed
                                //PrintErrorMessage();
                                return NULL;
                }
                result = ioctl(encoderboard, T116_IOCG_READ_TIMER, &timerBuf);
                if(result >= 0){
                                timeCount[i] = timerBuf.value;
                }else{
                                //TimerCount failed
                                //PrintErrorMessage();
                                return NULL;
                }
}
// Set prio to NON_RT since we are out from RT-section
param.sched_priority = NON_RT_PRIO;
if(sched_setscheduler(0, SCHED_RR, &param) == -1) {
                //perror("sched_setscheduler failed");
                return NULL;
}
jclass intArrCls = (*env)->FindClass(env, "[I");
if(intArrCls == NULL){
                //not enough memory
                return NULL;
}

array = (*env)->NewObjectArray(env, loggNo, intArrCls, NULL);
if (array == NULL){
                //not enough memory
                return NULL;
}
for(i=0;i<loggNo;i++)
{
                jint tempColumn[2];
                jintArray column = (*env)->NewIntArray(env, 2);
                if (column == NULL){
                                //not enough memory
                                return NULL;
                }
                tempColumn[0] = pStore[i];
                tempColumn[1] = timeCount[i];
                (*env)->SetIntArrayRegion(env, column, 0, 2, tempColumn);
                (*env)->SetObjectArrayElement(env, array, i, column);
                (*env)->DeleteLocalRef(env, column);
}

free(timeCount);
free(pStore);
free(buffer);
```

```
		return array;
}

// Method:	waitForHighTransition
JNIEXPORT jboolean JNICALL Java_se_ipbyran_tews_tip_TipJni_waitForHighTransition
		(JNIEnv *env, jclass cls, jint digitalBoard)
{
		int
		digitalboard;
		int
			result;
		T600_EVRD_BUFFER 		evBuf;
		digitalboard = (int) 		digitalBoard;

		evBuf.mode = T600_HIGH_TR;
		evBuf.mask = (unsigned short)(1<<0);
		evBuf.timeout = 20000;

		// Send read request to the device driver
		result = ioctl(digitalboard, T600_IOCX_EVENT_READ, &evBuf);
		if (result >= 0){
			// Trigger OK
			return (jboolean)1;
		}else{
			// No trigger signal or failed read
			//PrintErrorMessage();
			return (jboolean)0;
		}
}

//Method:	waitForAnyTransition
JNIEXPORT jboolean JNICALL
Java_se_ipbyran_tews_tip_TipJni_waitForAnyTransition(JNIEnv *env, jclass cls, jint digitalBoard)
{
		int
		digitalboard;
		int
			result;
		T600_EVRD_BUFFER 		evBuf;
		digitalboard = (int) 		digitalBoard;

		evBuf.mode = T600_HIGH_TR;
		evBuf.mask = (unsigned short)(1<<0);
		evBuf.timeout = 20000;

		//Send read request to the device driver
		result = ioctl(digitalboard, T600_IOCX_EVENT_READ, &evBuf);
		if (result >= 0)
		{
			//Trigger OK
			return (jboolean)1;
		}else{
			// No trigger signal or failed read
			//PrintErrorMessage();
			return (jboolean)0;
		}
}

static void PrintErrorMessage(void)
{
```

```
    printf("%s\n\n", strerror (errno));
}
```

Appendix G – Controller

Controller

```
package se.ipbyran.tews.tip.controller;

import java.util.ArrayList;
import java.util.List;

import se.ipbyran.tews.tip.*;
import se.ipbyran.tews.tip.TipJni.AnalogChannel;
import se.ipbyran.tews.tip.TipJni.EncoderChannel;
import se.ipbyran.tews.tip.TipJni.Tip116Board;
import se.ipbyran.tews.tip.TipJni.Tip501Board;
import se.ipbyran.tews.tip.TipJni.Tip600Board;
import se.openprocesslogger.proxy.LogTaskProxy;
import se.openprocesslogger.proxy.LoggingTaskData;
import se.openprocesslogger.svg.data.FineDataPoint;

public class Controller {

		private static Controller singleton;
		private TipJni jni;
		private Tip116Board tip116Board;
		private Tip501Board tip501Board;
		private Tip600Board tip600Board;
		private ArrayList<EncoderChannel> lstEncoderChannels;
		@SuppressWarnings("unused")
		private AnalogChannel[] lstAnalogChannels;

		private Controller(){

				jni = new TipJni();
		jni.initialize();
		//System.out.println(jni.getAvailableTip116Boards().length);
		tip116Board = jni.getAvailableTip116Boards()[0];
				tip501Board = jni.getAvailableTip501Boards()[0];
				tip600Board = jni.getAvailableTip600Boards()[0];
				tip116Board.initialize();
				tip501Board.initialize();
				tip600Board.initialize();
				lstEncoderChannels = tip116Board.getEncoderChannels();
				lstAnalogChannels = tip501Board.getAnalogChannels();
		}

		public static Controller getController(){
				if(singleton == null){
						singleton = new Controller();
				}
				return singleton;
		}

		public void configureTimer(){
				tip116Board.configureExternalTimer();
		}

		public boolean loggOneEncoderChannel(float time, int channelNo){
				EncoderChannel channel = lstEncoderChannels.get(channelNo-1);
				tip116Board.configureExternalTimer();
```

```
                channel.initialize();
                long timeStamp = System.currentTimeMillis();
                boolean storeSuccess =
DataContainer.getDataContainer().setData(channel.runCounter(time),timeStamp);
                if(storeSuccess){
                        // Logging success, Set some info about the read in data
container
                        DataContainer.getDataContainer().setDataType("Encoder");
                        DataContainer.getDataContainer().setChannels(1);
                }
                return storeSuccess;
        }

        public boolean loggOneEncoderChannelWithTrigger(float time, int channelNo, int
triggerType){
                EncoderChannel channel = lstEncoderChannels.get(channelNo-1);
                tip116Board.configureExternalTimer();
                channel.initialize();
                long timeStamp = System.currentTimeMillis();
                boolean storeSuccess =
DataContainer.getDataContainer().setData(channel.runCounterWithTrigger(tip600Board, triggerType,
time),timeStamp);
                if(storeSuccess){
                        // Logging success, Set some info about the read in data
container
                        DataContainer.getDataContainer().setDataType("Encoder");
                        DataContainer.getDataContainer().setChannels(1);
                }
                return storeSuccess;
        }
        public boolean loggAllEncoderChannels(float time){
                tip116Board.configureExternalTimer();
                long timeStamp = System.currentTimeMillis();
                boolean storeSuccess =
DataContainer.getDataContainer().setData(tip116Board.runAllCounters(time),timeStamp);
                if(storeSuccess){
                        // Logging success, Set some info about the read in data
container
                        DataContainer.getDataContainer().setDataType("Encoder");
                        DataContainer.getDataContainer().setChannels(4);
                }
                return storeSuccess;
        }
        public boolean loggAllEncoderChannelsWithTrigger(float time, int triggerType){
                tip116Board.configureExternalTimer();
                long timeStamp = System.currentTimeMillis();
                boolean storeSuccess =
DataContainer.getDataContainer().setData(tip116Board.runAllCountersWithTrigg(tip600Board, triggerType,
time),timeStamp);
                if(storeSuccess){
                        // Logging success, Set some info about the read in data
container
                        DataContainer.getDataContainer().setDataType("Encoder");
                        DataContainer.getDataContainer().setChannels(4);
                }
                return storeSuccess;
        }
        public boolean loggOneAnalogChannels(float time, int activeAnalogChannel){
                tip116Board.configureExternalTimer();
                long timeStamp = System.currentTimeMillis();
```

```
                        boolean storeSuccess =
DataContainer.getDataContainer().setData(tip501Board.runAnalog(activeAnalogChannel, tip116Board,
time),timeStamp);
                        if(storeSuccess){
                                // Logging success, Set some info about the read in data
container
                                DataContainer.getDataContainer().setDataType("Analog");
                                DataContainer.getDataContainer().setChannels(1);
                        }
                        return storeSuccess;
                }

                public boolean loggOneAnalogChannelsWithTrigger(float time,int activeAnalogChannel,int
triggerType){
                        tip116Board.configureExternalTimer();
                        long timeStamp = System.currentTimeMillis();
                        boolean storeSuccess =
DataContainer.getDataContainer().setData(tip501Board.runAnalogWithTrigger(activeAnalogChannel,
tip116Board, tip600Board, triggerType, time),timeStamp);
                        if(storeSuccess){
                                // Logging success, Set some info about the read in data
container
                                DataContainer.getDataContainer().setDataType("Analog");
                                DataContainer.getDataContainer().setChannels(1);
                        }
                        return storeSuccess;
                }
                public boolean loggTwoAnalogChannels(float time){
                        tip116Board.configureExternalTimer();
                        long timeStamp = System.currentTimeMillis();
                        boolean storeSuccess =
DataContainer.getDataContainer().setData(tip501Board.runAllAnalog(tip116Board, time),timeStamp);
                        if(storeSuccess){
                                DataContainer.getDataContainer().setDataType("Analog");
                                DataContainer.getDataContainer().setChannels(2);
                        }
                        return storeSuccess;
                }
                public boolean loggTwoAnalogChannelsWithTrigger(float time, int triggerType){
                        tip116Board.configureExternalTimer();
                        long timeStamp = System.currentTimeMillis();
                        boolean storeSuccess =
DataContainer.getDataContainer().setData(tip501Board.runAllAnalogWithTrigger(tip116Board, tip600Board,
triggerType, time),timeStamp);
                        if(storeSuccess){
                                DataContainer.getDataContainer().setDataType("Analog");
                                DataContainer.getDataContainer().setChannels(2);

                        }
                        return storeSuccess;
                }
                public boolean loggDigital(float time){
                        tip116Board.configureExternalTimer();
                        long timeStamp = System.currentTimeMillis();
                        boolean storeSuccess =
DataContainer.getDataContainer().setData(tip600Board.runDigitalLogging(tip116Board, time), timeStamp);
                        if(storeSuccess){
                                DataContainer.getDataContainer().setDataType("Digital");
                                DataContainer.getDataContainer().setChannels(1);
```

```
                        }
                        return storeSuccess;
                }
                public boolean loggDigitalWithTrigger(float time, int triggerType){
                        tip116Board.configureExternalTimer();
                        long timeStamp = System.currentTimeMillis();
                        boolean storeSuccess =
DataContainer.getDataContainer().setData(tip600Board.runDigitalLoggingWithTrigger(tip116Board,
triggerType, time), timeStamp);
                        if(storeSuccess){
                                DataContainer.getDataContainer().setDataType("Digital");
                                DataContainer.getDataContainer().setChannels(1);
                        }
                        return true;
                }

                public void closeBoars(){
                        tip116Board.close();
                        tip501Board.close();
                        tip600Board.close();
                }

                /*
                 * Section for dummy functions used by graphviewer
                 */
                public boolean generateTestMatrix(){
                        int count = 40000;
                        int timestamp = 39998;
                        int[][] testMatrixArray = new int[count][2];
                        for(int i = 0; i<count; i++){
                                testMatrixArray[i][0] = i;
                                testMatrixArray[i][1] = timestamp;
                                timestamp -= 60;
                                if (timestamp < 0)
                                        timestamp += 40000;
                        }
                        DataContainer.getDataContainer().setChannels(1);
                        long startTime = System.currentTimeMillis();
                        DataContainer.getDataContainer().setTimeStamp(startTime);
                        DataContainer.getDataContainer().setDataType("Test");
                        return DataContainer.getDataContainer().setData(testMatrixArray, startTime);
                }
                /*
                 * DataContainerCalls
                 */
                public void setNameToDataContainer(String name){
                        DataContainer.getDataContainer().setName(name);
                }
                public String getNameFromDataContainer(){
                        return DataContainer.getDataContainer().getName();
                }
                public String getDescriptionFromDataContainer() {
                        return DataContainer.getDataContainer().getDescription();
                }
                public void setDescriptionToDataContainer(String description) {
                        DataContainer.getDataContainer().setDescription(description);
                }
                public String getTypeFromDataContainer(){
                        return DataContainer.getDataContainer().getType();
                }
```

```
		public void setTypeToDataContainer(String DataType){
			DataContainer.getDataContainer().setDataType(DataType);
		}
		public FineDataPoint[] getData(int startIndex, int stopIndex, int channel){
			return DataContainer.getDataContainer().getDataBatch(startIndex, stopIndex,
channel);
		}
		public FineDataPoint[] getDataBatchFirstDerivative(int startIndex, int stopIndex, int channel){
			return
DataContainer.getDataContainer().getDataBatchFirstDerivative(startIndex, stopIndex, channel);
		}
		public FineDataPoint[] getDataBatchSecondDerivative(int startIndex, int stopIndex, int
channel){
			return
DataContainer.getDataContainer().getDataBatchSecondDerivative(startIndex, stopIndex, channel);
		}
		public int getDataCount(){
			return DataContainer.getDataContainer().getDataCount();
		}
		public boolean filterData(){
			return DataContainer.getDataContainer().lowPassFilterData();
		}
		public boolean pushToDatabase(){
			return DataContainer.getDataContainer().pushToDatabase();
		}
		public boolean pullFromDatabase(String LoggName){
			return DataContainer.getDataContainer().pullFromDatabase(LoggName);
		}
		/* check last two for removal of one of them */
		public List<LoggingTaskData> getAllLoggTasksFromDatabase(){
			return DataContainer.getDataContainer().getAllLoggTasksFromDatabase();
		}
		public ArrayList<LogTaskProxy> getAllLoggTasksProxyFromDatabase(){
			return
DataContainer.getDataContainer().getAllLoggTasksProxyFromDatabase();
		}
}
```

Controller proxy

```
package se.ipbyran.tews.tip.controller;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

import se.openprocesslogger.proxy.LogTaskProxy;
import se.openprocesslogger.proxy.LoggingTaskData;
import se.openprocesslogger.proxy.graphviewer.GraphMetadata;
import se.openprocesslogger.svg.data.FineDataPoint;

public class ControllerProxy implements Serializable {

		/**
		 *
		 */
		private static final long serialVersionUID = -5541414450275874900L;

		public ControllerProxy() {}
```

```
		public boolean loggOneEncoderChannel(float time, int channelNo){
			return Controller.getController().loggOneEncoderChannel(time, channelNo);
		}

		public boolean loggOneEncoderChannelWithTrigger(float time, int channelNo, int
triggerType){
			return Controller.getController().loggOneEncoderChannelWithTrigger(time,
channelNo, triggerType);
		}

		public boolean loggAllEncoderChannels(float time){
			return Controller.getController().loggAllEncoderChannels(time);
		}

		public boolean loggAllEncoderChannelsWithTrigger(float time, int triggerType){
			return Controller.getController().loggAllEncoderChannelsWithTrigger(time,
triggerType);
		}

		public boolean loggOneAnalogChannels(float time, int activeChannel){
			return loggOneAnalogChannels(time, activeChannel);
		}

		public boolean loggOneAnalogChannelsWithTrigger(float time,int activeAnalogChannel,int
triggerType){
			return Controller.getController().loggOneAnalogChannelsWithTrigger(time,
activeAnalogChannel, triggerType);
		}

		public boolean loggTwoAnalogChannels(float time){
			return Controller.getController().loggTwoAnalogChannels(time);
		}

		public boolean loggTwoAnalogChannelsWithTrigger(float time, int triggerType){
			return Controller.getController().loggTwoAnalogChannelsWithTrigger(time,
triggerType);
		}

		public boolean loggDigital(float time){
			return Controller.getController().loggDigital(time);
		}

		public boolean loggDigitalWithTrigger(float time, int triggerType){
			return Controller.getController().loggDigitalWithTrigger(time, triggerType);
		}

		public void configureTimer(){
			Controller.getController().configureTimer();
		}

		public void closeBoards(){
			Controller.getController().closeBoars();
		}
		public GraphMetadata CreateGraphMetaData(int channel, int derivative){
			GraphMetadata metadata = new GraphMetadata();
			DataContainer dataContainer = DataContainer.getDataContainer();
			metadata.setFrom(0);
			double lastTs = dataContainer.getLastTimestamp();
			double lastTsMillis = lastTs/1000;
			metadata.setTo((long) Math.ceil(lastTsMillis));
```

```
                                metadata.setChannel(channel);
                                metadata.setDerivative(derivative);
                                //System.out.println("From Controller Proxy: datachannel: "+channel);
                                //System.out.println("From Controller Proxy: derivative: "+derivative);
                                //System.out.println("From Controller Proxy: To time "+lastTs+" micros,
"+lastTsMillis +" millis, "+metadata.getTo()+" channel: "+metadata.getChannel());
                                return metadata;
                }
                public GraphMetadata CreateCombinedGraphMetaData(int channel){
                                GraphMetadata metadata = new GraphMetadata();
                                DataContainer dataContainer = DataContainer.getDataContainer();
                                metadata.setFrom(0);
                                double lastTs = dataContainer.getLastTimestamp();
                                double lastTsMillis = lastTs/1000;
                                metadata.setTo((long) Math.ceil(lastTsMillis));
                                metadata.setChannel(channel);
                                metadata.setDerivative(1);
                                metadata.setCombined(true);
                                return metadata;

                }
                /*
                 *  DataContainer Calls
                 */
                public void setNameToDataContainer(String name){
                                Controller.getController().setNameToDataContainer(name);
                }
                public String getNameFromDataContainer(){
                                return Controller.getController().getNameFromDataContainer();
                }
                public String getDescriptionFromDataContainer() {
                                return Controller.getController().getDescriptionFromDataContainer();
                }
                public void setDescriptionToDataContainer(String description) {
                                Controller.getController().setDescriptionToDataContainer(description);
                }
                public int channelCountFromDataContainer(){
                                return DataContainer.getDataContainer().getChannels();
                }
                public void discardDataFromDataContainer(){
                                DataContainer.getDataContainer().clearDataContainer();
                }
                public FineDataPoint[] getDataBatchFirstDerivative(int startIndex, int stopIndex, int channel){
                                return Controller.getController().getDataBatchFirstDerivative(startIndex,
stopIndex, channel);
                }
                public FineDataPoint[] getDataBatchSecondDerivative(int startIndex, int stopIndex, int
channel){
                                return Controller.getController().getDataBatchSecondDerivative(startIndex,
stopIndex, channel);
                }
                public boolean filterData(){
                                return Controller.getController().filterData();
                }
                public boolean pushToDatabase(){
                                return Controller.getController().pushToDatabase();
                }
                public boolean pullFromDatabase(String LoggName){
                                return Controller.getController().pullFromDatabase(LoggName);
                }
```

```
		public List<LoggingTaskData> getAllLoggTasksFromDatabase(){
			return Controller.getController().getAllLoggTasksFromDatabase();
		}
		public ArrayList<LogTaskProxy> getAllLoggTasksProxyFromDatabase(){
			return Controller.getController().getAllLoggTasksProxyFromDatabase();
		}
		public String getTypeFromDataContainer(){
			return Controller.getController().getTypeFromDataContainer();
		}
		public void setTypeToDataContainer(String DataType){
			Controller.getController().setTypeToDataContainer(DataType);
		}
		/*
		 * Dummy for testing graphs
		 */
		public boolean generateTestMatrix(){
			return Controller.getController().generateTestMatrix();
		}
		public FineDataPoint[] getData(int startIndex, int stopIndex, int channel){
			return Controller.getController().getData(startIndex, stopIndex, channel);
		}
		public int getDataCount(){
			return Controller.getController().getDataCount();
		}
}
```

Graph Creator

```
package se.ipbyran.tews.tip.controller;

import java.io.Serializable;

import se.openprocesslogger.proxy.graphviewer.Batch;
import se.openprocesslogger.proxy.graphviewer.BatchInfo;
import se.openprocesslogger.proxy.graphviewer.GraphMetadata;
import se.openprocesslogger.proxy.graphviewer.GraphMetadataResponse;
import se.openprocesslogger.proxy.graphviewer.IGraphInfoCreator;
import se.openprocesslogger.proxy.graphviewer.IGraphViewerProxy;
import se.openprocesslogger.proxy.graphviewer.SingleChartInfo;
import se.openprocesslogger.proxy.graphviewer.SingleData;
import se.openprocesslogger.svg.style.Style;

public class GraphCreator implements IGraphViewerProxy, Serializable {

		/**
		 *
		 */
		private static final long serialVersionUID = 267261681857025234OL;

		@Override
		public Batch getBatch(BatchInfo batchInfo) {
			DataContainer dataContainer = DataContainer.getDataContainer();
			int from = dataContainer.getIndexFromTime(batchInfo.getFrom()*1000);
			int to = dataContainer.getIndexFromTime(batchInfo.getTo()*1000);
			if(from == 0){
				from++;
			}
			if (to-from > batchInfo.getBatchSize()){
				to = from + batchInfo.getBatchSize() -1;
			}else{
```

```
                                        to--;
                                }
                                Batch batch = new Batch();
                                //System.out.println("From GraphCreator get batch: channelInfo:
channelToView: "+batchInfo.getChannel());
                                //System.out.println("From GraphCreator get batch: derivative:
"+batchInfo.getDerivative());
                                if(batchInfo.getDerivative() == 0){
                                        batch.setData(Controller.getController().getData(from, to,
batchInfo.getChannel()));
                                }else if(batchInfo.getDerivative() == 1){

                batch.setData(Controller.getController().getDataBatchFirstDerivative(from, to,
batchInfo.getChannel()));
                                }else if(batchInfo.getDerivative()==2){

                batch.setData(Controller.getController().getDataBatchSecondDerivative(from, to,
batchInfo.getChannel()));
                                }

                                //System.out.println("From GraphCreator get batch: batch created: "+from +" --
> "+to +" count: "+batch.getData().length);
                                return batch;
                }

                @Override
                public Batch getEventBatch(BatchInfo batchInfo) {
                                DataContainer dataContainer = DataContainer.getDataContainer();
                                int from = dataContainer.getIndexFromTime(batchInfo.getFrom()*1000);
                                int to = dataContainer.getIndexFromTime(batchInfo.getTo()*1000);
                                Batch batch = new Batch();
                                batch.setData(Controller.getController().getData(from, to,
batchInfo.getChannel()));
                                return batch;
                }

                @Override
                public GraphMetadataResponse getGraphViewerShell(GraphMetadata graphInfo) {
                                GraphMetadataResponse response = new GraphMetadataResponse();
                                DataContainer dataContainer = DataContainer.getDataContainer();
                                SingleChartInfo[] charts;
                                if(graphInfo.isCombined()){
                                        charts = new SingleChartInfo[2];
                                        for(int i=0;i<charts.length;i++){
                                                charts[i] = new SingleChartInfo();
                                                charts[i].setChartName("Channel
"+graphInfo.getChannel());

                charts[i].setDataCount(dataContainer.getDataCount());

                charts[i].setTagType("se.openprocesslogger.svg.taglib.AnalogChartShell");
                                                charts[i].setVarName("Channel
"+graphInfo.getChannel());
                                                charts[i].setChannel(graphInfo.getChannel());
                                                charts[i].setDerivative(i);
                                        }
                                }else{
                                        charts = new SingleChartInfo[1];
                                //SingleChartInfo[] charts = new SingleChartInfo[dataContainer.getChannels()];
                                /*for(int i=0;i<charts.length;i++){
```

```
                                        charts[i] = new SingleChartInfo();
                                        charts[i].setChartName("Channel "+(i+1));
                                        charts[i].setDataCount(dataContainer.getDataCount());

                charts[i].setTagType("se.openprocesslogger.svg.taglib.AnalogChartShell");
                                        charts[i].setVarName("Channel "+(i+1));
                                        charts[i].setChannel((i+1));
                                        charts[i].setDerivative(graphInfo.getDerivative());
                                        //System.out.println("From GraphCreator: Channel
"+charts[i].getChannel() +" count " +dataContainer.getDataCount());
                                        //System.out.println("From GraphCreator: channel in graphInfo:
"+graphInfo.getChannel());
                                        //System.out.println("From GraphCreator: derivative:
"+graphInfo.getDerivative());
                                }*/
                                        charts[0] = new SingleChartInfo();
                                        charts[0].setChartName("Channel "+graphInfo.getChannel());
                                        charts[0].setDataCount(dataContainer.getDataCount());

                charts[0].setTagType("se.openprocesslogger.svg.taglib.AnalogChartShell");
                                        charts[0].setVarName("Channel "+graphInfo.getChannel());
                                        charts[0].setChannel(graphInfo.getChannel());
                                        charts[0].setDerivative(graphInfo.getDerivative());
                                }
                                response.setCharts(charts);
                                response.setFromMillis(0);
                                response.setToMillis((long)
Math.ceil(dataContainer.getLastTimestamp()/1000));
                                response.setStyle(Style.defaultStyle);
                                response.setChannel(graphInfo.getChannel());
                                response.setDerivative(graphInfo.getDerivative());
                                return response;
                }

                @Override
                public GraphMetadataResponse getGraphViewerShellSingleData(SingleData data,
                                        IGraphInfoCreator parser) {
                                return null;
                }

}
```

Appendix H – Data Container

```
package se.ipbyran.tews.tip.controller;

import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;

import se.openprocesslogger.Data;
import se.openprocesslogger.db.OPLDatabase;
import se.openprocesslogger.log.LoggingTask;
import se.openprocesslogger.proxy.LogTaskProxy;
import se.openprocesslogger.proxy.LoggingTaskData;
import se.openprocesslogger.svg.data.FineDataPoint;


public class DataContainer {

		private static DataContainer singleton;
		private double[][] dataMatrix = null;
		private String dataType = "";
		private int channels = 0;
		private String loggName = null;
		private String description = null;
		private long timeStamp = 0;
		Runtime r = Runtime.getRuntime();

		private DataContainer(){

		}
		public static DataContainer getDataContainer(){
			if(singleton == null){
				singleton = new DataContainer();
			}
			return singleton;
		}

		public int getDataCount(){
			if (dataMatrix !=null){
				return dataMatrix.length;
			}else{
				return 0;
			}
		}

		public FineDataPoint[] getDataBatch(int startIndex,int stopIndex, int selectedChannel){
			//implemented for test purposes
			if(selectedChannel > this.channels || selectedChannel <= 0) return null;
			int timeColumn = dataMatrix[0].length-1;
			int dataColumn = selectedChannel -1;
			FineDataPoint[] batch = new FineDataPoint[stopIndex-startIndex+1];
			for(int i= startIndex; i<=stopIndex;i++){
				batch[i-startIndex] = new
FineDataPoint(dataMatrix[i][dataColumn], (long)(dataMatrix[i][timeColumn]/1000), 0,
(dataMatrix[i][timeColumn])%1000);
			}
			return batch;
		}
		public FineDataPoint[] getDataBatchFirstDerivative(int startIndex, int stopIndex, int
selectedChannel){
```

```
                        if(selectedChannel > this.channels || selectedChannel <= 0) return null;
                        int timeColumn = dataMatrix[0].length-1;
                        int dataColumn = selectedChannel -1;
                        double dData;
                        FineDataPoint[] batch = new FineDataPoint[stopIndex-startIndex+1];
                        for(int i= startIndex; i<=stopIndex;i++){
                                dData = (dataMatrix[i+1][dataColumn] - dataMatrix[i-
1][dataColumn])/
                                                (dataMatrix[i+1][timeColumn]
-dataMatrix[i-1][timeColumn]);

                                batch[i-startIndex] = new FineDataPoint(dData,
(long)(dataMatrix[i][timeColumn]/1000), 0, (dataMatrix[i][timeColumn])%1000);
                        }
                        return batch;
        }
        public FineDataPoint[] getDataBatchSecondDerivative(int startIndex, int stopIndex, int
selectedChannel){
                        if(selectedChannel > this.channels || selectedChannel <= 0) return null;
                        int timeColumn = dataMatrix[0].length-1;
                        int dataColumn = selectedChannel -1;
                        double dData;
                        FineDataPoint[] batch = new FineDataPoint[stopIndex-startIndex+1];
                        for(int i= startIndex; i<=stopIndex;i++){
                                dData = (dataMatrix[i+1][dataColumn] - 2 *
dataMatrix[i][dataColumn] + dataMatrix[i-1][dataColumn])/

        ((dataMatrix[i+1][timeColumn]-dataMatrix[i][timeColumn]) * (dataMatrix[i][timeColumn]-
dataMatrix[i-1][timeColumn]));

                                batch[i-startIndex] = new FineDataPoint(dData,
(long)(dataMatrix[i][timeColumn]/1000),0,(dataMatrix[i][timeColumn])%1000);
                        }
                        return batch;
        }

        public boolean setData(int[][] rawDataMatrix, long TimeStamp){
                        if(rawDataMatrix != null){
                                r.gc();
                                setChannels(rawDataMatrix[0].length-1);
                                dataMatrix= new
double[rawDataMatrix.length][rawDataMatrix[0].length];
                                //Nested loops to copy all data values
                                for(int i=0; i<rawDataMatrix[0].length-1; i++ ){
                                        for(int j=0; j<rawDataMatrix.length; j++){
                                                dataMatrix[j][i] =
rawDataMatrix[j][i];
                                        }
                                }
                                //Loop to convert clockticks to micros
                                int n=1;
                                int timeColumn = rawDataMatrix[0].length-1;
                                for(int k=0; k< rawDataMatrix.length; k++){
                                        dataMatrix[k][timeColumn]= (40000*n -
rawDataMatrix[k][timeColumn])*0.125;
                                        if(k+1 < rawDataMatrix.length){

        if(rawDataMatrix[k+1][timeColumn] > rawDataMatrix[k][timeColumn]){
                                                                n++;
                                                }
```

```
                                                }
                                        }
                                        this.timeStamp = TimeStamp;
                                        r.gc();
                                        return true;
                                }else{
                                        // If here dataMatrix is null
                                        return false;
                                }
                }
                public boolean lowPassFilterData(){
                                System.out.println("From DataContainer: lowPassFilterData()");
                                int timeColumn = dataMatrix[0].length-1;
                                int channels = getChannels();
                                int frequencyCutOff = 500;
                                int i;
                                double RC = (1/(2*Math.PI*frequencyCutOff))*8000000; // in ticks
                                double alfa;
                                double dT;
                                for(i=1;i<getDataCount()-1;i++){ // rows
                                        dT = (dataMatrix[i][timeColumn]-dataMatrix[i-
1][timeColumn])*0.125; // in micros
                                        alfa = dT/(dT+RC);
                                        for(int j=0;j<channels-1; i++){ // data columns
                                                dataMatrix[i][j] = dataMatrix[i-1][j] + alfa *
(dataMatrix[i][j]-dataMatrix[i-1][j]);
                                        }
                                }
                                r.gc();// Garbage collector call
                                return true;
                }

                public void setDataType(String DataType){
                                this.dataType = DataType;
                }
                public String getType(){
                                return dataType;
                }
                public void setChannels(int Channels){
                                this.channels = Channels;
                }
                public int getChannels(){
                                return channels;
                }
                public void setName(String name){
                                this.loggName = name;
                }
                public String getName(){
                                return loggName;
                }
                public String getDescription() {
                                return description;
                }
                public void setDescription(String description) {
                                this.description = description;
                }
                public long getTimeStamp() {
                                return timeStamp;
                }
                public void setTimeStamp(long timeStamp) {
```

```
                        this.timeStamp = timeStamp;
        }
        public void clearDataContainer(){
                        dataMatrix = null;
                        dataType = "";
                        channels = 0;
                        loggName = "";
                        description = "";
                        timeStamp = 0;
                        r.gc();// Garbage collector call
        }
        public double getLastTimestamp(){
                        if(dataMatrix == null || dataMatrix.length == 0 || dataMatrix[dataMatrix.length-
1].length == 0) return 0;
                        //System.out.println("Last ts "+dataMatrix[dataMatrix.length-
1][dataMatrix[dataMatrix.length-1].length -1]);
                        return dataMatrix[dataMatrix.length-1][dataMatrix[dataMatrix.length-1].length
-1];
        }
        public int getIndexFromTime(double micros){
                        if(dataMatrix == null || dataMatrix.length == 0 || dataMatrix[dataMatrix.length-
1].length == 0) return 0;
                        return binarySearch(micros, 0, dataMatrix.length-1);
        }

        private int binarySearch(double micros, int first, int last){
                        if (first>=last)
                                        return first;
                        if (last-first == 1)
                                        return (micros > dataMatrix[last][dataMatrix[0].length-1]) ?
last : first;

                        int middle = (first+last)/2;

                        if (micros == dataMatrix[middle][dataMatrix[0].length-1])
                                        return middle;
                        if (micros > dataMatrix[middle][dataMatrix[0].length-1])
                                        return binarySearch(micros, middle, last);

                        return binarySearch(micros, first, middle);
        }
        /*
         * DatabaseFunctions
         */
        public boolean pushToDatabase(){
                        r.gc();// Garbage collector call
                        // Function for setting stuff in database
                        LoggingTask logTask = new LoggingTask(loggName, new HashMap<String,
String>());
                        logTask.addProperty("Description", description);
                        logTask.addProperty("Channels", Integer.toString(channels));
                        logTask.addProperty("DataType", dataType);
                        logTask.addProperty("Timestamp", Long.toString(timeStamp));
                        logTask.setTaskId(timeStamp);
                        logTask.setStartTime(0);
                        logTask.setStopTime((long) ((getLastTimestamp() * 0.125)/1000000));
                        OPLDatabase.instance().addLogTask(logTask);

                        Data data = new Data(loggName, 0, logTask.getTaskId(), dataMatrix,  new
Date(timeStamp));
```

```
                        if(data.getValue() == null){
                                System.out.println("From DataContainer: pushToDatabase:
Data.getValue is null");
                        }
                        r.gc();
                        OPLDatabase.getDataBuffer().addData(data);
                        OPLDatabase.instance().flush();
                        r.gc();
                        return true;
                }
                public boolean pullFromDatabase(String LoggName){
                        // Function for getting data from database
                        LoggingTask logTask = OPLDatabase.instance().findLogTask(LoggName);
                        if (logTask != null){
                                //System.out.println("From DataContainer: pullFromDatabase():
logtask != null");
                                this.loggName = logTask.getName();
                                this.description = logTask.getProperty("Description");
                                this.channels = new Integer(logTask.getProperty("Channels"));
                                this.dataType = logTask.getProperty("DataType");
                                this.timeStamp = new
Long(logTask.getProperty("Timestamp"));
                                //this.timeStamp = logTask.getStartTime();
                        } else {
                                //System.out.println("From DataContainer: logTask doesn't
exist in Database");
                                return false;
                        }
                        Data temp = OPLDatabase.instance().getSampleData(logTask.getName(),
logTask.getTaskId());
                        if (temp != null){
                                this.dataMatrix = (double[][])
OPLDatabase.instance().getSampleData(logTask.getName(), logTask.getTaskId()).getValue();
                                if(this.dataMatrix == null){
                                        //System.out.println("From DataContainer:
pullFromDatabase: value is null");
                                }
                                r.gc(); // Garbage collector call
                                return true;
                        }else {
                                //System.out.println("From DataContainer: pullFromDataBase:
if this we have a big problem");
                                return false;
                        }
                }
                public List<LoggingTaskData> getAllLoggTasksFromDatabase(){
                        return OPLDatabase.instance().getFinishedLogTasks();
                }
                public ArrayList<LogTaskProxy> getAllLoggTasksProxyFromDatabase(){
                        List<LoggingTaskData> loggingTasks =
OPLDatabase.instance().getFinishedLogTasks();
                        LoggingTask tempLoggTask;
                        LogTaskProxy tempProxy;
                        ArrayList<LogTaskProxy> logTaskProxyArray = new
ArrayList<LogTaskProxy>();
                        for(int i = 0; i < loggingTasks.size(); i++){
                                tempLoggTask =
OPLDatabase.instance().findLogTask(loggingTasks.get(i).getName());
                                tempProxy = tempLoggTask.proxify();
                                logTaskProxyArray.add(tempProxy);
```

```
                }
                return logTaskProxyArray;
            }
        }
```

Appendix I – HMI

Motionlogger.js

```
function init(){
		jsonrpc = new JSONRpcClient('/OpenProcessLogger/JSON-RPC');
}

function closeBoardsFromController(){
		jsonrpc.controller.closeBoards();
}

function configureTimerFromController(){
		jsonrpc.controller.configureTimer();
}

function showEncoderChannelMeny(){
		dijit.byId('channelMeny').setHref("MotionLogger2/jsp/ChannelMenyEncoder.html");
}
function showAnalogChannelMeny(){
		dijit.byId('channelMeny').setHref("MotionLogger2/jsp/ChannelMenyAnalog.html");
}
function showDigitalChannelMeny(){
		//solution used since only page should be blank
		var channelMenyHolder = document.getElementById("channelMeny");
		var strContent = "";
		channelMenyHolder.innerHTML = strContent;
		hideSelectChannelMeny();
}
function showSelectChannelMeny(){
		dijit.byId('selectChannelMeny').setHref("MotionLogger2/jsp/SelectChannelMeny.html");
}
function hideSelectChannelMeny(){
		var selectChannelMenyHolder = document.getElementById("selectChannelMeny");
		var strContent = "";
		selectChannelMenyHolder.innerHTML = strContent;
}

function showSelectAnalogChannelMeny(){
		dijit.byId('selectChannelMeny').setHref("MotionLogger2/jsp/SelectAnalogChannelMeny.html");
}
function hideSelectAnalogChannelMeny(){
		var selectChannelMenyHolder = document.getElementById("selectChannelMeny");
		var strContent = "";
		selectChannelMenyHolder.innerHTML = strContent;
}

function showDefineDigitalTrigger(useTrigger){
		if(useTrigger == true){

		dijit.byId('triggerDefintionMeny').setHref("MotionLogger2/jsp/TriggerMeny.html");
		} else {
				var triggerMenyHolder = document.getElementById("triggerDefintionMeny");
				var strContent = "";
				triggerMenyHolder.innerHTML = strContent;
		}
}
```

motionlogger2Database.js

```
var logging_logTasks;


function motionlogger2_loadDatabaseMeny(){
		dijit.byId('tabContainer').selectChild('tabSettings');
		dijit.byId('tabSettings').setHref("MotionLogger2/jsp/MotionLogger2DatabasePane.html");
		//dijit.byId('DatabasePane').setHref("MotionLogger2/jsp/MotionLogger2DatabaseMeny.html");
		//dijit.byId('tabSettings').onLoad = buildLogTaskView;
		//logging_fillLogTaskPane(); // logging.js
		getLogTasks();

}

function getLogTasks(){
		//var list = jsonrpc.controller.getAllLoggTasksFromDatabase();
		//jsonrpc.controller.getAllLoggTasksFromDatabase(getLogTasksCB);
		jsonrpc.controller.getAllLoggTasksProxyFromDatabase(getLogTasksCB);
}

function getLogTasksCB(logtasksData, exception){
		logging_logTasks = logtasksData.list;

		var objContainer = document.getElementById("TaskContainer");
		var objTaskTable = gui_createItemTable("DatabasePaneTable", objContainer);

		var templateName = "Timestamp";

		for(i = 0; i < logging_logTasks.length; i++)
		{
			var taskName = logging_logTasks[i].name;
			var started = logging_logTasks[i].started;
			var timestamp = logging_logTasks[i].properties.map.Timestamp;
			var time = parseInt(timestamp);
			var date = new Date(time);
			var tRow = document.createElement("div");
			tRow.setAttribute("id", "taskpanelist" + i);
			tRow.setAttribute("onclick",
"motionlogger2_loadTaskToDataContainerInfo("+i+")");
			tRow.appendChild(getImage(started)); // getImage is in openprocesslogger.js

			tRow.innerHTML += taskName + " <span id=\"taskmsg" + i + "\"></span>"
+date;
			objTaskTable.appendChild(tRow);

			if(logging_chosenTaskProxy && (taskName ==
logging_chosenTaskProxy.name))
			{
				chosenIndex = i;
				logging_chosenTemplateIndex = -1;
			}
		}
		objContainer.appendChild(objTaskTable);
}
function motionlogger2_loadTaskToDataContainerInfo(index){
		dijit.byId('tabSettings').setHref("MotionLogger2/jsp/MotionLogger2DatabasePane.html");
		discardData();
		opl_debug("logging_loadActiveTaskInfo - index: " + index);
		if(index >= 0)
```

```
		{
			gui_setSelectedItem("taskpanelist", index, "logTaskIndex");
			var task = logging_logTasks[index];
			if(task != undefined)
			{
				var loaded = jsonrpc.controller.pullFromDatabase(task.name);
				if(loaded == true){
					motionlogger2_loadDataMeny();
				}else{
					alert("Data not loaded");
				}
			}
		}
}
function motionlogger2_loadDataMeny(){
		if(jsonrpc.controller.channelCountFromDataContainer() == 2){

		dijit.byId('tabSettings').setHref("MotionLogger2/jsp/MotionLogger2DatabaseMeny2Channels.ht
ml");
		}else if(jsonrpc.controller.channelCountFromDataContainer() == 4){

		dijit.byId('tabSettings').setHref("MotionLogger2/jsp/MotionLogger2DatabaseMeny4Channels.ht
ml");
		}else{

		dijit.byId('tabSettings').setHref("MotionLogger2/jsp/MotionLogger2DatabaseMeny.html");
		}
		//var x =dijit.byId('logTaskNameDB');
		//alert(x);
}
function motionLogger2_loadInfo(){
		var x = dijit.byId('logTaskNameDB');
		x.attr("value", jsonrpc.controller.getNameFromDataContainer());
		x.setDisabled(true);
		var y = dijit.byId('loggingTaskInfoDB');
		y.attr("value", jsonrpc.controller.getDescriptionFromDataContainer());
		y.setDisabled(true);
		var z = dijit.byId('loggingTaskTypeDB');
		z.attr("value", jsonrpc.controller.getTypeFromDataContainer());
		z.setDisabled(true);
}
```

motionlogger2Graph.js

```
var input_graphviwer_graphInfo2;

function motionLogger2_createGraph_Plain(){
		var channels = jsonrpc.controller.channelCountFromDataContainer();
		var graphChannel = getSelectedChannel(channels);
		if (graphChannel == 0) {
			alert("Please Specify channel to View");
			return;
		}
		input_graphviwer_graphInfo2 = new Object();
		input_graphviwer_graphInfo2.graphInfo =
jsonrpc.controller.CreateGraphMetaData(graphChannel, 0);
		taskanalyze_loadNewTab("graphviewer/jsp/motionlogger2timeAnalyze_1.jsp", "Raw Data for
Channel "+graphChannel);
}
```

```
function motionLogger2_createGraph_Derived(){
		var channels = jsonrpc.controller.channelCountFromDataContainer();
		var graphChannel = getSelectedChannel(channels);
		if (graphChannel == 0) {
			alert("Please Specify channel to View");
			return;
		}
		input_graphviwer_graphInfo2 = new Object();
		input_graphviwer_graphInfo2.graphInfo =
jsonrpc.controller.CreateGraphMetaData(graphChannel, 1);
		taskanalyze_loadNewTab("graphviewer/jsp/motionlogger2timeAnalyze_1.jsp", "First Derivative
for Channel " +graphChannel);
}

function motionLogger2_createGraph_SecondDerivative(){
		var channels = jsonrpc.controller.channelCountFromDataContainer();
		var graphChannel = getSelectedChannel(channels);
		if (graphChannel == 0) {
			alert("Please Specify channel to View");
			return;
		}
		input_graphviwer_graphInfo2 = new Object();
		input_graphviwer_graphInfo2.graphInfo =
jsonrpc.controller.CreateGraphMetaData(graphChannel, 2);
		taskanalyze_loadNewTab("graphviewer/jsp/motionlogger2timeAnalyze_1.jsp", "Second
Derivative for Channel " +graphChannel);
}
function motionLogger2_createGraph_Combined(){
		var channels = jsonrpc.controller.channelCountFromDataContainer();
		var graphChannel = getSelectedChannel(channels);
		if (graphChannel == 0) {
			alert("Please Specify channel to View");
			return;
		}
		input_graphviwer_graphInfo2 = new Object();
		input_graphviwer_graphInfo2.graphInfo =
jsonrpc.controller.CreateCombinedGraphMetaData(graphChannel);
		taskanalyze_loadNewTab("graphviewer/jsp/motionlogger2timeAnalyze_1.jsp", "Combined for
Channel " +graphChannel);
}

function getSelectedChannel(channels){
		if( channels == 2){
			var activeChannel1 =
document.getElementById("activeGraphTwoChannels1");
			var activeChannel2 =
document.getElementById("activeGraphTwoChannels2");
			if(activeChannel1.checked){
				return 1;
			}else if(activeChannel2.checked){
				return 2;
			}else {
				// if no Selected channel
				return 0;
			}
		} else if (channels == 4){
			var activeChannel1 = document.getElementById("activeGraphFourChannel1");
			var activeChannel2 = document.getElementById("activeGraphFourChannel2");
			var activeChannel3 = document.getElementById("activeGraphFourChannel3");
			var activeChannel4 = document.getElementById("activeGraphFourChannel4");
```

```
                        if(activeChannel1.checked){
                                return 1;
                        }else if(activeChannel2.checked){
                                return 2;
                        }else if(activeChannel3.checked){
                                return 3;
                        }else if(activeChannel4.checked){
                                return 4;
                        }else{
                                // if no Selected channel
                                return 0;
                        }
                } else {
                        return 1;
                }
}
function motionlogger2_Filter(){
                if(confirm("Low-pass filter data?")){
                        var filtered = jsonrpc.controller.filterData();
                }
                if(filtered == true){
                        alert("Filtration sucessfully");
                }else{
                        alert("Filtration failed");
                }
}

function motionlogger2_discard(){
                var discard = confirm("Discard Data?");
                if(discard){
                        discardData();
                }
}
```

Motionlogger2Logging.js

```
function startLogging(){
                //Maybe a check if you want to discard current data
                discardData();

                var type=getType();
                switch(type)
                {
                case 1:
                        // Encoder
                        startEncoder();
                        break;
                case 2:
                        // Analog
                        startAnalog();
                        break;
                case 3:
                        // Digital
                        startDigital();
                        break;
                default:
                        // No type selected
                        alert("No type selected");
                }
}
```

```
function startEncoder(){
		var channels = getNumberOfEncoderChannels();
		switch(channels)
		{
		case 1:
				//alert("Debug: startEncoder: case 1 ");
				// One channel, Call to new function that
				startOneEncoderChannels();
				break;
		case 4:
				//alert("Debug: startEncoder: case 4 ");
				// Log on All channels
				startAllEncoderChannels();
				break;
		default:
				alert("No number of channels selected");
		}
}

function startAnalog(){
		var channels = getNumberOfAnalogChannels();
		switch(channels)
		{
		case 1:
				// One channel, Call to new function that
				startOneAnalogChannels();
				break;
		case 2:
				// Log on All channels
				startTwoAnalogChannels();
				break;
		default:
				alert("No number of channels selected");
		}
}

function startDigital(){
		var time = getLoggTime();
		var trigger = getTrigger();
		if(trigger == null){
				// No trigger type selected = don't start logging
				alert("No trigger type selected");
				return;
		}
		if(trigger != 0){
				// Logg with Trigger
				loggingSuccess(jsonrpc.controller.loggDigitalWithTrigger(time, trigger));
		}else{
				// Logg without Trigger
				loggingSuccess(jsonrpc.controller.loggDigital(time));
		}
}


function startOneEncoderChannels(){
		//alert("Debug: startEncoder: startOneEncoderChannels: row 1 ");
		var activeChannel = getActiveEncoderChannel();
		if (activeChannel == null){
				alert("No channel to logg on is specified");
				return;
```

```
		}
		var time = getLoggTime();
		var trigger = getTrigger();
		if(trigger == null){
				// No trigger type selected = don't start logging
				alert("No trigger type selected");
				return;
		}else if(trigger != 0){
				if (activeChannel == null){
						// No channel selected
						alert("No channel to logg on selected");
				}else{
						//Call to controller to logg on one channel with trigger

		loggingSuccess(jsonrpc.controller.loggOneEncoderChannelWithTrigger(time, activeChannel,
trigger));
				}
		}else{
				// No trigger
				if(activeChannel == null){
						// No channel selected
						alert("No channel to logg on selected");
				}else{

		loggingSuccess(jsonrpc.controller.loggOneEncoderChannel(time, activeChannel));
				}
		}
}

function startAllEncoderChannels(){
		//alert("Debug: startEncoder: startAllEncoderChannels(): row 1 ");
		var time = getLoggTime();
		var trigger = getTrigger();
		if(trigger == null){
				// No trigger type selected = don't start logging
				alert("No trigger type selected");
				return;
		}
		if(trigger != 0){
				// Logg with Trigger
				loggingSuccess(jsonrpc.controller.loggAllEncoderChannelsWithTrigger(time,
trigger));
		}else{
				// Logg without Trigger
				//alert("Debug: startEncoder: startAllEncoderChannels(): beforeCallToJava");
				loggingSuccess(jsonrpc.controller.loggAllEncoderChannels(time));
		}
}

function startOneAnalogChannels(){
		var activeAnalogChannel = getActiveAnalogChannel();
		if (activeAnalogChannel == null){
				alert("No channel to logg on is specified");
				return;
		}
		var time = getLoggTime();
		var trigger = getTrigger();
		if(trigger == null){
				// No trigger type selected = don't start logging
				alert("No trigger type specified");
```

```
                        return;
                }else if(trigger != 0){
                        if (activeAnalogChannel == null){
                                // No channel selected
                                alert("No channel to logg on specified");
                        }else{
                                // Logg with Trigger

                loggingSuccess(jsonrpc.controller.loggOneAnalogChannelsWithTrigger(time,
activeAnalogChannel, trigger));
                        }
                }else{
                        if (activeAnalogChannel == null){
                                // No channel selected
                                alert("No channel to logg on specified");
                        }else{
                        // Logg without Trigger
                        loggingSuccess(jsonrpc.controller.loggOneAnalogChannels(time,
activeAnalogChannel));
                        }
                }
}
function startTwoAnalogChannels(){
                var time = getLoggTime();
                var trigger = getTrigger();
                if(trigger == null){
                        // No trigger type selected = don't start logging
                        alert("No trigger type selected");
                        return;
                }else if(trigger != 0){
                        // Logg with Trigger
                        loggingSuccess(jsonrpc.controller.loggTwoAnalogChannelsWithTrigger(time,
trigger));
                }else{
                        // Logg without Trigger
                        loggingSuccess(jsonrpc.controller.loggTwoAnalogChannels(time));
                }
}

function getType(){
                var encoder = document.getElementById("typeEncoder");
                var analog = document.getElementById("typeAnalog");
                var digital = document.getElementById("typeDigital");
                if(encoder.checked){
                        return 1;
                }else if(analog.checked){
                        return 2;
                }else if(digital.checked){
                        return 3;
                }else{
                        return null;
                }
}

function getNumberOfEncoderChannels(){
                var oneChannels = document.getElementById("encoderChannelsOne");
                var allChannels = document.getElementById("encoderChannelsAll");
                if(oneChannels != null && allChannels != null){
                        if(oneChannels.checked){
                                return 1;
```

```
                        }else if(allChannels.checked){
                                return 4;
                        }else{
                                return 0;
                        }
                }else{
                        return 0;
                }
}

function getNumberOfAnalogChannels(){
                var oneAChannels = document.getElementById("analogChannels1");
                var twoAChannels = document.getElementById("analogChannels2");
                if(oneAChannels != null && twoAChannels != null){
                        if(oneAChannels.checked){
                                return 1;
                        }else if(twoAChannels.checked){
                                return 2;
                        }else {
                                return 0;
                        }
                }else{
                        return 0;
                }
}

function getActiveEncoderChannel(){
                var activeChannel1 = document.getElementById("activeChannel1");
                var activeChannel2 = document.getElementById("activeChannel2");
                var activeChannel3 = document.getElementById("activeChannel3");
                var activeChannel4 = document.getElementById("activeChannel4");
                if(activeChannel1 != null && activeChannel2 != null && activeChannel3 != null &&
activeChannel4 != null ){
                        if(activeChannel1.checked){
                                return 1;
                        }else if(activeChannel2.checked){
                                return 2;
                        }else if(activeChannel3.checked){
                                return 3;
                        }else if(activeChannel4.checked){
                                return 4;
                        }else{
                                return null;
                        }
                }else{
                        return null;

                }
}

function getActiveAnalogChannel(){
                var activeAnalogChannel1 = document.getElementById("activeAnalogChannel1");
                var activeAnalogChannel2 = document.getElementById("activeAnalogChannel2");
                var activeAnalogChannel3 = document.getElementById("activeAnalogChannel3");
                var activeAnalogChannel4 = document.getElementById("activeAnalogChannel4");
                var activeAnalogChannel5 = document.getElementById("activeAnalogChannel5");
                var activeAnalogChannel6 = document.getElementById("activeAnalogChannel6");
                var activeAnalogChannel7 = document.getElementById("activeAnalogChannel7");
                var activeAnalogChannel8 = document.getElementById("activeAnalogChannel8");
```

```
		if(activeAnalogChannel1 != null && activeAnalogChannel2 != null && activeAnalogChannel3
!= null &&
							activeAnalogChannel4 != null && activeAnalogChannel5 !=
null && activeAnalogChannel6 != null &&
							activeAnalogChannel7 != null && activeAnalogChannel8 !=
null){
					if(activeAnalogChannel1.checked){
							return 1;
					}else if(activeAnalogChannel2.checked){
							return 2;
					}else if(activeAnalogChannel3.checked){
							return 3;
					}else if(activeAnalogChannel4.checked){
							return 4;
					}else if(activeAnalogChannel5.checked){
							return 5;
					}else if(activeAnalogChannel6.checked){
							return 6;
					}else if(activeAnalogChannel7.checked){
							return 7;
					}else if(activeAnalogChannel8.checked){
							return 8;
					}else{
							return null;
					}
		}else{
					return null;
		}
}

function getLoggTime(){
		var timeHolder = dijit.byId("loggingTime");
		return timeHolder.value;
}

function getTrigger(){
		// should return 0 if no trigger, 1 if low, 2 if high, 3 if any or null if no type is selected
		var trigger = document.getElementById("useTrigger");
		if(trigger.checked){
				var high = document.getElementById("triggerHigh");
				var low = document.getElementById("triggerLow");
				var		any = document.getElementById("triggerAny");
				if(high != null && low != null && any != null){
						if(high.checked){
								// High transition selected
								return 1;
						}else if(low.checked){
								// Low transition selected
								return 2;
						}else if(any.checked){
								// Any transition selected
								return 3;
						}else{
								// no trigger selected
								return null;
						}
				}else{
						return null;
				}
		}else{
```

```
                        return 0;
                }
}

function enableStartButtonCB(){
                var type = getType();
                if(type != null){
                        // Type should be selected
                        if(type == 3){
                                //Digital selected == no secondary meny here needed
                                checkTrigger();
                        }else if(type == 1){
                                //Encoder selected = check channels
                                var EncoderChannel = getNumberOfEncoderChannels();
                                if(EncoderChannel == 1){
                                        // one Channel Selected
                                        if(getActiveEncoderChannel() != null){
                                                checkTrigger();
                                        }else{
                                                disableStartButton();
                                        }
                                }else if(EncoderChannel == 4){
                                        // Four selected
                                        checkTrigger();
                                }else{
                                        disableStartButton();
                                }
                        }else if(type = 2){
                                var AnalogChannels = getNumberOfAnalogChannels();
                                if(AnalogChannels == 1){
                                        // one Channel Selected
                                        if(getActiveAnalogChannel() != null){
                                                checkTrigger();
                                        }else{
                                                disableStartButton();
                                        }
                                }else if(AnalogChannels == 2){
                                        // Two selected
                                        checkTrigger();
                                }else {
                                        disableStartButton();
                                }
                        }else{
                                disableStartButton();
                        }
                }else{
                        disableStartButton();
                }
}

function checkTrigger(){
                if(getTrigger() != null){
                        dijit.byId("startButton").setAttribute('disabled', false);
                }else{
                        disableStartButton();
                }
}

function disableStartButton(){
                dijit.byId("startButton").setAttribute('disabled', true);
```

```
}

function loggingSuccess(complete){
		if(complete == true){
			if(jsonrpc.controller.channelCountFromDataContainer() == 2){

		dijit.byId('resultHolder').setHref("MotionLogger2/jsp/ResultSuccess2Channels.html");
			}else if(jsonrpc.controller.channelCountFromDataContainer() == 4){

		dijit.byId('resultHolder').setHref("MotionLogger2/jsp/ResultSuccess4Channels.html");
			}else{

		dijit.byId('resultHolder').setHref("MotionLogger2/jsp/ResultSuccess.html");
			}
		}else{
			dijit.byId('resultHolder').setHref("MotionLogger2/jsp/ResultFail.html");
		}
}

function discardData(){
		jsonrpc.controller.discardDataFromDataContainer();
		var menyHolder = dijit.byId('resultHolder');
		if(menyHolder != null){
			menyHolder.setHref("MotionLogger2/jsp/blank.html");
		}
		/*var menyHolder = document.getElementById("resultHolder");
		if (menyHolder != null){
			var strContent = "";
			menyHolder.innerHTML = strContent;
		}*/
}
function motionlogger2_Save(){
		var loggName = dijit.byId("logTaskName").value;
		var loggDescription = dijit.byId("logTaskName").value;
		if (loggName == "" ){
			alert("Please type a logging task name");
		} else if(loggDescription==""){
			alert("Please type description for the logging task");
		}else{
			jsonrpc.controller.setNameToDataContainer(loggName);
			jsonrpc.controller.setDescriptionToDataContainer(loggDescription);

			var saved = jsonrpc.controller.pushToDatabase();
			if (saved == true){
				alert("Data saved");
			}else {
				alert("Save data failed");
			}
		}
}
/*
 * Dummy/Debugging stuff
 */

function generateDummyData(){
		discardData();
		var bool = jsonrpc.controller.generateTestMatrix();
		loggingSuccess(bool);
}
```

Motionlogger2Settings.js

```
function motionlogger2_loadMain(){
		dijit.byId('tabContainer').selectChild('tabSettings');
dijit.byId('tabSettings').setHref("MotionLogger2/jsp/MotionLogger2Main.html");
dijit.byId('MainPane').setHref("MotionLogger2/jsp/MotionLogger2Settings.html");
		//dijit.byId('tabSettings').onLoad = buildLogTaskView;
		//logging_fillLogTaskPane(); // logging.js
}
```

Motionlogger2 HTML documents

ChannelMenyAnalog.html

```
<h3>Channels: </h3>
<input id="analogChannels1" type="radio" name="channels" value="1"
onclick="showSelectAnalogChannelMeny(); enableStartButtonCB();"> 1
<input id="analogChannels2" type="radio" name="channels" value="2"
onclick="hideSelectAnalogChannelMeny(); enableStartButtonCB();"> 2 <br>
```

ChannelMenyEncoder.html

```
<h3>Channels: </h3>
<input id="encoderChannelsOne" type="radio" name="channels" value="1"
onclick="showSelectChannelMeny(); enableStartButtonCB();"> 1 channel
<input id="encoderChannelsAll"type="radio" name="channels" value="4" onclick="hideSelectChannelMeny();
enableStartButtonCB();"> 4 channels <br>
```

Motionlogger2DatabaseMeny.html

```
<div style="margin-left: 20px;">
		<h2 style="color:green" >Data Loaded from Database</h2>
		<button id="load info" dojoType="dijit.form.Button" value="Load info"
onclick="motionLogger2_loadInfo();">Load Info on Logging Task</button>
		<div id="trEquipmentID">
			<h3>Logging task name:</h3>
			<div class="formSubArea">
				<input dojoType="dijit.form.TextBox" promptMessage="Enter
the name of the logging task" id="logTaskNameDB" maxlength="40">
			</div>
		</div>

		<div id="trLogInfo">
			<h3>Logging task description:</h3>
			<div class="formSubArea">
				<textarea dojoType="dijit.form.Textarea"
id="loggingTaskInfoDB" style="width:600px;"></textarea>
			</div>
		</div>

		<div id="trLogInfo">
			<h3>Logging task type:</h3>
			<div class="formSubArea">
				<textarea dojoType="dijit.form.Textarea"
id="loggingTaskTypeDB" style="width:100px;"></textarea>
			</div>
		</div>
<br><br>
		<div id="graphChannelMenyDB" ></div>
```

```
		<button id="filterData" dojoType="dijit.form.Button" value="Filter Data"
onclick="motionlogger2_Filter();">Filter Data</button>
		<button id="viewGraph" dojoType="dijit.form.Button" value="View Plain Graph"
onclick="motionLogger2_createGraph_Plain();">View Plain Graph</button>
		<button id="viewFirstDerivedGraph" dojoType="dijit.form.Button" value="View Derived
Graph" onclick="motionLogger2_createGraph_Derived();">View Derived Graph</button>
		<button id="viewSecondDerivedGraph" dojoType="dijit.form.Button" value="View two Derived
Graph" onclick="motionLogger2_createGraph_SecondDerivative();">View Second Derived Graph</button>
		<button id="viewCombined" dojoType="dijit.form.Button" value="View Combined Graph"
onclick="motionLogger2_createGraph_Combined();">View Combined Graph</button>
</div>
```

Motionlogger2DatabaseMeny2Channels.html

```
<div style="margin-left: 20px;">
		<h2 style="color:green" >Data Loaded from Database</h2>
		<button id="load info" dojoType="dijit.form.Button" value="Load info"
onclick="motionLogger2_loadInfo();">Load Info on Logging Task</button>
		<div id="trEquipmentID">
			<h3>Logging task name:</h3>
			<div class="formSubArea">
				<input dojoType="dijit.form.TextBox" promptMessage="Enter
the name of the logging task" id="logTaskNameDB" maxlength="40">
			</div>
		</div>

		<div id="trLogInfo">
			<h3>Logging task description:</h3>
			<div class="formSubArea">
				<textarea dojoType="dijit.form.Textarea"
id="loggingTaskInfoDB" style="width:600px;"></textarea>
			</div>
		</div>
		<div id="trLogInfo">
			<h3>Logging task type:</h3>
			<div class="formSubArea">
				<textarea dojoType="dijit.form.Textarea"
id="loggingTaskTypeDB" style="width:100px;"></textarea>
			</div>
		</div>
		<br><br>
		<h3>Select Channel to View </h3>
			<input id="activeGraphTwoChannels1" type="radio"
name="activeGraphChannelTwo" value="1" "> Channel 1
			<input id="activeGraphTwoChannels2" type="radio"
name="activeGraphChannelTwo" value="2" "> Channel 2 <br><br>
			<button id="filterData" dojoType="dijit.form.Button" value="Filter Data"
onclick="motionlogger2_Filter();">Filter Data</button>
			<button id="viewGraph" dojoType="dijit.form.Button" value="View Plain
Graph" onclick="motionLogger2_createGraph_Plain();">View Plain Graph</button>
			<button id="viewFirstDerivedGraph" dojoType="dijit.form.Button"
value="View Derived Graph" onclick="motionLogger2_createGraph_Derived();">View Derived
Graph</button>
			<button id="viewSecondDerivedGraph" dojoType="dijit.form.Button"
value="View two Derived Graph" onclick="motionLogger2_createGraph_SecondDerivative();">View Second
Derived Graph</button>
			<button id="viewCombined" dojoType="dijit.form.Button" value="View
Combined Graph" onclick="motionLogger2_createGraph_Combined();">View Combined Graph</button>
</div>
```

Motionlogger2DatabaseMeny4Channels.html

```
<div style="margin-left: 20px;">
		<h2 style="color:green" >Data Loaded from Database</h2>
		<button id="load info" dojoType="dijit.form.Button" value="Load info"
onclick="motionLogger2_loadInfo();">Load Info on Logging Task</button>
		<div id="trEquipmentID">
			<h3>Logging task name:</h3>
			<div class="formSubArea">
				<input dojoType="dijit.form.TextBox" promptMessage="Enter
the name of the logging task" id="logTaskNameDB" maxlength="40">
			</div>
		</div>

		<div id="trLogInfo">
			<h3>Logging task description:</h3>
			<div class="formSubArea">
				<textarea dojoType="dijit.form.Textarea"
id="loggingTaskInfoDB" style="width:600px;"></textarea>
			</div>
		</div>
		<div id="trLogInfo">
			<h3>Logging task type:</h3>
			<div class="formSubArea">
				<textarea dojoType="dijit.form.Textarea"
id="loggingTaskTypeDB" style="width:100px;"></textarea>
			</div>
		</div>
<br><br>
<h3>Select Channel to View: </h3>
		<input id="activeGraphFourChannel1" type="radio" name="activeGraphChannelFour"
value="1"> Channel 1
		<input id="activeGraphFourChannel2" type="radio" name="activeGraphChannelFour"
value="2"> Channel 2
		<input id="activeGraphFourChannel3" type="radio" name="activeGraphChannelFour"
value="3"> Channel 3
		<input id="activeGraphFourChannel4" type="radio" name="activeGraphChannelFour"
value="4"> Channel 4 <br><br>
		<button id="filterData" dojoType="dijit.form.Button" value="Filter Data"
onclick="motionlogger2_Filter();">Filter Data</button>
		<button id="viewGraph" dojoType="dijit.form.Button" value="View Plain Graph"
onclick="motionLogger2_createGraph_Plain();">View Plain Graph</button>
		<button id="viewFirstDerivedGraph" dojoType="dijit.form.Button" value="View Derived
Graph" onclick="motionLogger2_createGraph_Derived();">View Derived Graph</button>
		<button id="viewSecondDerivedGraph" dojoType="dijit.form.Button" value="View two Derived
Graph" onclick="motionLogger2_createGraph_SecondDerivative();">View Second Derived Graph</button>
		<button id="viewCombined" dojoType="dijit.form.Button" value="View Combined Graph"
onclick="motionLogger2_createGraph_Combined();">View Combined Graph</button>
</div>
```

Motionlogger2Main.html

```
<h2 style="text-align:center"><big>MotionLogger2 HMI MainPage</big></h2>

<div style="margin-left: 20px;">

		<div id="resultHolder" dojoType="dijit.layout.ContentPane"></div>
</div>
```

Motionlogger2Settings.html

```
<div style="margin-left: 10px;">
		<h3>Logging Type:</h3>
		<input id="typeEncoder" type="radio" name="loggingType" value="Encoder"
onclick="showEncoderChannelMeny(); hideSelectChannelMeny(); enableStartButtonCB();"> Encoder
		<input id="typeAnalog" type="radio" name="loggingType" value="Analog"
onclick="showAnalogChannelMeny(); hideSelectChannelMeny(); enableStartButtonCB();"> Analog
		<input id="typeDigital" type="radio" name="loggingType" value="Digital"
onclick="showDigitalChannelMeny(); hideSelectChannelMeny(); enableStartButtonCB();"> Digital
		<br>
		<div id="channelMeny" dojoType="dijit.layout.AccordionPane"></div>
		<div id="selectChannelMeny" dojoType="dijit.layout.AccordionPane"></div>
		<br>
		<h3>Trigger: </h3>
		<input id="useTrigger" type="checkbox" name="useTrigger"
onclick="showDefineDigitalTrigger(this.checked); enableStartButtonCB();"> Use Digital Trigger
		<br>
		<div id="triggerDefintionMeny" dojoType="dijit.layout.AccordionPane"></div>
		<!--  <label for="loggingTime" > Logging Time: </label>-->
		<br>
		<h3>Logging Time:</h3>
		<input dojoType="dijit.form.NumberSpinner" value="10.0" smallDelta="0.5" largeDelta="1.0"
maxLength="20"
				constraints="{min:0.5,max:30.0,places:1,round:true}"
name="loggingTime" id="loggingTime"></input>
		<br><br><br>
		<button id="startButton" dojoType="dijit.form.Button" value="Start Logging"
onclick="startLogging();"
			style="font-size: 30px; margin-left: 0px;" disabled="true"> Start Logging
</button>

		<br><br>
		<button id="testButton" dojoType="dijit.form.Button" value="generateDummyData"
onclick="generateDummyData();"
			style="margin-left: 0px;">GenerateDummyData</button>
</div>
```

ResultFail.html

```
<h2 style="color:red" >Logging Failed</h2>
```

ResultSucess.html

```
<h2 style="color:green" >Logging Complete</h2>
		<div id="graphChannelMeny" dojoType="dijit.layout.ContentPane" ></div>
		<button id="viewGraph" dojoType="dijit.form.Button" value="View Plain Graph"
onclick="motionLogger2_createGraph_Plain();" >View Plain Graph</button>
		<button id="viewFirstDerivedGraph" dojoType="dijit.form.Button" value="View Derived
Graph" onclick="motionLogger2_createGraph_Derived();">View Derived Graph</button>
		<button id="viewSecondDerivedGraph" dojoType="dijit.form.Button" value="View two Derived
Graph" onclick="motionLogger2_createGraph_SecondDerivative();">View Second Derived Graph</button>
		<button id="viewCombined" dojoType="dijit.form.Button" value="View Combined Graph"
onclick="motionLogger2_createGraph_Combined();">View Combined Graph</button>

		<div id="trEquipmentID">
			<h3>Logging task name:</h3>
			<div class="formSubArea">
				<input dojoType="dijit.form.TextBox" promptMessage="Enter
the name of the logging task" id="logTaskName" maxlength="40">
```

```
                        </div>
                </div>

                <div id="trLogInfo">
                        <h3>Logging task description:</h3>
                        <div class="formSubArea">
                                <textarea dojoType="dijit.form.Textarea" id="loggingTaskInfo"
style="width:600px;"></textarea>
                        </div>
                </div>
<br><br>
<button id="filterData" dojoType="dijit.form.Button" value="Filter Data"
onclick="motionlogger2_Filter();">Filter Data</button>
<button id="viewSave" dojoType="dijit.form.Button" value="Save Data"
onclick="motionlogger2_Save();">Save Data</button>
<button id="DiscardData" dojoType="dijit.form.Button" value="Discard Data"
onclick="motionlogger2_discard();">Discard</button>
```

ResultSucess2Channels.html

```
<h2 style="color:green" >Logging Complete</h2>
                <h3>Select Channel to View </h3>
                <input id="activeGraphTwoChannels1" type="radio" name="activeGraphChannelTwo"
value="1" "> Channel 1
                <input id="activeGraphTwoChannels2" type="radio" name="activeGraphChannelTwo"
value="2" "> Channel 2 <br><br>
                <button id="viewGraph" dojoType="dijit.form.Button" value="View Plain Graph"
onclick="motionLogger2_createGraph_Plain();" >View Plain Graph</button>
                <button id="viewFirstDerivedGraph" dojoType="dijit.form.Button" value="View Derived
Graph" onclick="motionLogger2_createGraph_Derived();">View Derived Graph</button>
                <button id="viewSecondDerivedGraph" dojoType="dijit.form.Button" value="View two Derived
Graph" onclick="motionLogger2_createGraph_SecondDerivative();">View Second Derived Graph</button>
                <button id="viewCombined" dojoType="dijit.form.Button" value="View Combined Graph"
onclick="motionLogger2_createGraph_Combined();">View Combined Graph</button>

                <div id="trEquipmentID">
                        <h3>Logging task name:</h3>
                        <div class="formSubArea">
                                <input dojoType="dijit.form.TextBox" promptMessage="Enter
the name of the logging task" id="logTaskName" maxlength="40">
                        </div>
                </div>

                <div id="trLogInfo">
                        <h3>Logging task description:</h3>
                        <div class="formSubArea">
                                <textarea dojoType="dijit.form.Textarea" id="loggingTaskInfo"
style="width:600px;"></textarea>
                        </div>
                </div>
<br><br>
<button id="filterData" dojoType="dijit.form.Button" value="Filter Data"
onclick="motionlogger2_Filter();">Filter Data</button>
<button id="viewSave" dojoType="dijit.form.Button" value="Save Data"
onclick="motionlogger2_Save();">Save Data</button>
<button id="DiscardData" dojoType="dijit.form.Button" value="Discard Data"
onclick="motionlogger2_discard();">Discard</button>
```

ResultSucess4Channels.html

```
<h2 style="color:green" >Logging Complete</h2>
		<h3>Select Channel to View: </h3>
		<input id="activeGraphFourChannel1" type="radio" name="activeGraphChannelFour"
value="1"> Channel 1
		<input id="activeGraphFourChannel2" type="radio" name="activeGraphChannelFour"
value="2"> Channel 2
		<input id="activeGraphFourChannel3" type="radio" name="activeGraphChannelFour"
value="3"> Channel 3
		<input id="activeGraphFourChannel4" type="radio" name="activeGraphChannelFour"
value="4"> Channel 4 <br><br>
		<button id="viewGraph" dojoType="dijit.form.Button" value="View Plain Graph"
onclick="motionLogger2_createGraph_Plain();" >View Plain Graph</button>
		<button id="viewFirstDerivedGraph" dojoType="dijit.form.Button" value="View Derived
Graph" onclick="motionLogger2_createGraph_Derived();">View Derived Graph</button>
		<button id="viewSecondDerivedGraph" dojoType="dijit.form.Button" value="View two Derived
Graph" onclick="motionLogger2_createGraph_SecondDerivative();">View Second Derived Graph</button>
		<button id="viewCombined" dojoType="dijit.form.Button" value="View Combined Graph"
onclick="motionLogger2_createGraph_Combined();">View Combined Graph</button>

		<div id="trEquipmentID">
			<h3>Logging task name:</h3>
			<div class="formSubArea">
				<input dojoType="dijit.form.TextBox" promptMessage="Enter
the name of the logging task" id="logTaskName" maxlength="40">
			</div>
		</div>

		<div id="trLogInfo">
			<h3>Logging task description:</h3>
			<div class="formSubArea">
				<textarea dojoType="dijit.form.Textarea" id="loggingTaskInfo"
style="width:600px;"></textarea>
			</div>
		</div>
<br><br>
<button id="filterData" dojoType="dijit.form.Button" value="Filter Data"
onclick="motionlogger2_Filter();">Filter Data</button>
<button id="viewSave" dojoType="dijit.form.Button" value="Save Data"
onclick="motionlogger2_Save();">Save Data</button>
<button id="DiscardData" dojoType="dijit.form.Button" value="Discard Data"
onclick="motionlogger2_discard();">Discard</button>
```

SelectAnalogChannelMeny.html

```
<h3>Select Channel to logg on: </h3>
<input id="activeAnalogChannel1" type="radio" name="activeAnalogChannel" value="1"
onclick="enableStartButtonCB();"> Channel 1
<input id="activeAnalogChannel2" type="radio" name="activeAnalogChannel" value="2"
onclick="enableStartButtonCB();"> Channel 2
<input id="activeAnalogChannel3" type="radio" name="activeAnalogChannel" value="3"
onclick="enableStartButtonCB();"> Channel 3
<input id="activeAnalogChannel4" type="radio" name="activeAnalogChannel" value="4"
onclick="enableStartButtonCB();"> Channel 4 <br>
<input id="activeAnalogChannel5" type="radio" name="activeAnalogChannel" value="5"
onclick="enableStartButtonCB();"> Channel 5
<input id="activeAnalogChannel6" type="radio" name="activeAnalogChannel" value="6"
onclick="enableStartButtonCB();"> Channel 6
```

```
<input id="activeAnalogChannel7" type="radio" name="activeAnalogChannel" value="7"
onclick="enableStartButtonCB();"> Channel 7
<input id="activeAnalogChannel8" type="radio" name="activeAnalogChannel" value="8"
onclick="enableStartButtonCB();"> Channel 8
<br>
```

SelectChannelMeny.html

```
<h3>Select Channel to logg on: </h3>
<input id="activeChannel1" type="radio" name="activeChannel" value="1"
onclick="enableStartButtonCB();"> Channel 1
<input id="activeChannel2" type="radio" name="activeChannel" value="2"
onclick="enableStartButtonCB();"> Channel 2
<input id="activeChannel3" type="radio" name="activeChannel" value="3"
onclick="enableStartButtonCB();"> Channel 3
<input id="activeChannel4" type="radio" name="activeChannel" value="4"
onclick="enableStartButtonCB();"> Channel 4
<br>
```

TriggerMeny.html

```
<h3>TriggerType: </h3>
<input id="triggerHigh" type="radio" name="trigger" value="high" onclick="enableStartButtonCB();"> High
Transition
<input id="triggerLow" type="radio" name="trigger" value="low" onclick="enableStartButtonCB();"> Low
Transition
<input id="triggerAny" type="radio" name="trigger" value="any" onclick="enableStartButtonCB();"> Any
Transition
<br>
```