Institutionen för datavetenskap
Department of Computer and Information Science

Final thesis

Porting embedded software for automotive applications

by

Henrik Carlgren

LIU-IDA/LITH-EX-A--10/028--SE

2010-06-24

Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings universitet
581 83 Linköping

Linköping University
Department of Computer and Information Science

Final Thesis

Embedded software porting for automotive applications

by

Henrik Carlgren

LIU-IDA/LITH-EX-A--10/028--SE

2010-06-24

Supervisor: **Andreas Bergvall**
Telematics
at Actia Nordic AB
Examiner: **Petru Eles**
Deptartment of Computer and Information Science
at Linköping University

Abstract

Developing software is both time consuming and expensive. Finding ways for minimizing development cost is therefore of great interest. One way of reducing cost in software development is reuse of existing software. Porting software is a type of reuse where an existing piece of software is adapted to run in a different environment than originally intended. Successfully porting software requires both good preparations and evaluation of results. Guidelines for this process are provided in this thesis. In this thesis an existing embedded software platform for automotive telematics was partially ported to a new type of hardware.

Keywords : PORTING, SOFTWARE, EMBEDDED

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter the background, scope and organization of this thesis is introduced.

1.1 Background

Developing software is both time consuming and expensive, finding ways of minimizing development cost is therefore of great interest. One way of reducing cost in software development is reuse of existing software. Porting software is a type of reuse where an existing piece of software is adapted to run in different circumstances than originally intended. Porting requires both good preparations and analysis of results obtained.

Actia Nordic AB owns a software platform for in-vehicle telemetric applications. The hardware platform currently used needs to be replaced by a new one that is better suited for future products. There exists a hardware platform within the company that might be a suitable replacement to the currently used hardware platform.

1.2 Purpose

The purpose of this thesis is to identify if the suggested hardware platform is a suitable replacement to the existing hardware platform. In addition, the software platform is partially ported to allow performance tests on the suggested hardware platform.

1.3 Objectives

The focus of this thesis is to solve the following problems:

- Generate technical support for a decision on whether the combination of the hardware and software platforms would be suitable for a new product.
- Make a partial port of the software platform to the hardware platform.
- Compare performance of ported software with existing in-house products.

1.4 Organization

This thesis report is organized in the following way:

Chapter 2 gives an overview of which activities are either recommended or necessary when porting embedded software. Alternative approaches for the recommended activities will be explained.

Chapter 3 contains the results of the analysis of the hardware and software that was involved in the porting.

Chapter 4 contains a summary of unplanned activities that were performed when the software platform was ported.

Chapter 5 contains the results from the performance and power consumption test of both the source platform and the target platform.

Chapter 6 contains a discussion of the results of the report and suggestions for future work.

Chapter 2

Overview

This chapter introduces four aspects of embedded software porting that were identified, studied and applied to the porting project. This chapter focuses on the theory of software porting and does not contain decisions and results from the actual porting project.

2.1 Analysis

The first step in solving any problem in a structured way requires analysis of the problem to be solved. In software porting the analysis serves two purposes, to determine if it is possible and what needs to be done to achieve desired results.

Analyzing the source and target hardware as a first step of the analysis gives insight into which parts are different and in which ways they are different. Knowing the differences between the hardware platforms is not enough to determine what needs to be ported. For that also the software architecture needs to be considered.

2.1.1 Hardware analysis

The primary objective with the hardware analysis is to determine if the target hardware can provide functionality equivalent to that of the source hardware. For embedded systems that are part of a larger system the communication interfaces are among the most important functionalities to consider. A secondary objective in the analysis can be to determine if the target hardware

provides new capabilities that the source hardware did not have.

2.1.2 Software analysis

Assuming the software to be ported has a modular design, the dependencies between modules and the modules dependencies to hardware are important to know in order to be able to predict how changes affect functionality.

Module dependencies can be expressed with a directed graph, where the directed edges denote a dependency of one module on another. A directed edge points from the module that depends on another module towards the module that it depends on.

Dependencies on third-party software require additional analysis of how the functionality provided can be ported to the new hardware platform. The preferred solution is to use a port provided by the vendor of the third-party software if such a port exists for the target hardware. If a port of the third-party software does not exist either an alternative third-party solution has to be found or a replacement has to be developed.

2.2 Effort estimation

The analysis results give information on what needs to be done to create a new product from an existing one using porting. Effort estimation is used to give information on the time consumption and cost of creating a new product. Effort estimates are important because they make it possible to take a well informed decision on whether or not to port software.

Effort estimation methods can be classified into two categories, expert effort estimation and formal effort estimation.

2.2.1 Expert effort estimation

Expert effort estimation is a class of effort estimation methods that rely on the experience of one or more experts. One of the experiences that experts use is similarity to other projects for which they know how much effort they required. Expert based effort estimation methods are preferable since they have been proven to be more accurate than formal methods [1].

2.2.2 Formal effort estimation

Formal estimation uses mathematical models typically with few inputs and an effort estimate as output. The inputs to formal estimation models are chosen such that each of them should be easier to predict on their own than predicting the complete effort directly. Formal methods are most useful when expert effort estimation is not possible because lack of expertise in the area of the project.

The simplest and most well known formal estimation method is COCOMO. The COCOMO. model uses source lines of code as an input and project type as configuration parameters. The parameters for project types are derived from historical project data using regression.

$$Effort = A * KSLOC^B \tag{2.1}$$

Equation (2.1) is the COCOMO formula, the unit of effort is man-months. For a porting projects typical values for the configuration parameters are $A = 0.93$ and $B = 1.02$ [2]. The input parameter KSLOC is the expected number of source lines of code that need to be developed measured in thousands.

2.3 Porting

Porting embedded software is a process that differs much between projects because there are so many combinations of hardware components and software architectures. The many differences between embedded software projects make it hard to create universal guidelines for porting embedded software and in reality no porting process has reached wide acceptance [3].

Two common ways of approaching porting are decremental-porting and incremental-porting [4]. Decremental-porting is when staring with the whole system and removing modules one by one until all platform dependencies are removed. Then the platform dependent modules are ported one by one. The idea behind decremental-porting is that most code does not need any changes when porting and that a majority of the system therefore would be ported early. Incremental-porting is when modules are ported and added to the target platform one by one. The idea behind incremental-porting is to always have a partially ported system that is improved when more modules are ported; this method gives immediate feedback that actual progress is made.

2.4 Evaluation

The final step of a porting project is to evaluate the result of the porting. Functionality should always be evaluated after a porting, to ensure that the ported system works as intended. The functionality of the ported system can be evaluated by ensuring that it fulfills the requirement specification of the original system. Since a porting is most of the time driven by the need for something to change, it is also important to evaluate that this goal is reached. A common reason for doing a port is to improve performance. In such a case one or more performance tests should be created and tested on both the source and target platforms to get information on performance differences.

Chapter 3

Analysis

In this chapter the results of the analysis are presented.

3.1 Hardware platform comparison

The purpose of the hardware platform comparison is to identify if the target platform provides at least the same functionality as the source platform. Also of interest are potential performance improvements and new capabilities enabled by the target platform.

The software platform is currently running on a hardware platform called Telematics 3 (TEM3) which was built for a specific product at the time of construction. Since the hardware platform was developed for a specific product it was perfectly suited for that product but not necessarily for other products.

The ported software platform will be running on a hardware platform called Telematic Gateway Unit (TGU). The TGU is a more modern hardware platform than the TEM3 and is expected to be more powerful and better suited for future products.

The features considered in table 3.1 are those that are expected to require most work during the porting in case they are different.

The TGU does not have a MOST controller. This mean that the TGU cannot be used for products that require MOST connectivity. This is the only hardware difference identified that will limit the functionality of the ported software.

Feature	**TEM3**	**TGU**
Microcontroller	CP3SP33 (CR16C+)	i.MXS (ARM9)
CPU Architecture	16-bit RISC	32-bit RISC
CPU Endian	Little	Little
Max clock frequency	100MHz	200MHz
Instruction cache	4KB	16KB
Data cache	None	16KB
CPU RAM	32KB	None
RAM	1MB	8-32MB
Flash memory	4MB	8-64MB
MMU	No	Yes
CAN	2 x Built-in into MCU	2 x MCP2515
GPS	SiRF II/III	Ublox TIM-4A
GSM	Wavecom Q2686	Motorola G24
MOST	Yes	No

Table 3.1: Feature matrix for TEM3 and TGU.

3.1.1 Microcontroller and memory functionality

Both microcontrollers use little endian byte-order which means that no special care has to be taken to byte-order when porting the software platform.

The TGU has significantly more RAM and flash memory. This ensures that memory capacity will not limit the possibility to port the software to the TGU.

The MCU used in the TGU has a MMU which makes it possible to implement memory protection in the ported software, given that the software architecture supports it.

3.1.2 Microcontroller performance

Cache memories and clock frequencies are most important aspects of microcontroller performance. The TGU is more powerful in each of these aspects and it is expected to have better overall performance than TEM3.

3.2 Software platform architecture

The purpose of the software platform analysis was to understand which parts of the platform would be affected and to what extent when porting. The analysis was performed in a hierarchical way. First the platform was analyzed on a subsystem level then deeper analysis of the affected subsystems was performed.

3.2.1 Overview of software platform

The purpose of the overview of the hardware abstraction layer was to identify dependencies between the modules in the existing implementation. This information can be useful when determining the best order to port modules in.

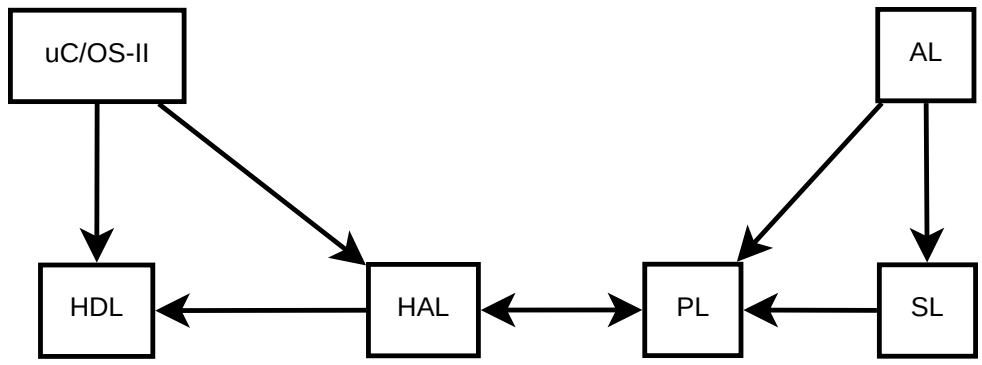

Figure 3.1: High level dependency graph for software platform.

uC/OS-II

The software platform uses a third-party real-time operating system called uC/OS-II. A suitable port of uC/OS-II for the type of MCU used in the TGU is available from the vendor of the operating system.

Hardware driver layer (HDL)

The HDL contains low-level drivers for the hardware. This subsystem does not provide a standardized interface to hardware devices. The interface to the hardware is standardized in the HAL. The purpose of the HDL is to make it easier to create drivers in the HAL.

Hardware abstraction layer (HAL)

The purpose of the HAL is to provide an interface to the hardware that does not change when the underlying hardware and HDL is changed. The porting will be isolated to this subsystem with only a few exceptions for the HDL and uC/OS-II.

Platform layer (PL)

The platform layer provides simplified interfaces to the hardware; typically low-level protocols are implemented in this layer. Modules in this layer should not need any changes when porting.

Service layer (SL)

The service layer uses functionality provided by PL to provide high-level services with specific purposes. Modules in this layer should not need any changes when porting.

Application layer (AL)

The application layer is the subsystem that gives the system its functionality. The purpose of the entire platform is to make it easy to write applications. Modules in this layer should not need any changes when porting.

3.2.2 Overview of the hardware abstraction layer

The purpose of the overview of the hardware abstraction layer was to identify dependencies between the modules in the existing implementation. This information can be useful when determining the best order to port modules in.

The dependency graph in figure 3.2 shows the dependencies that exist between modules in the hardware abstraction layer for the TEM3 hardware platform. The dependency graph only visualizes dependencies between modules in the hardware abstraction layer, this mean that dependencies to higher levels are not included. A consequence of this is that modules might have no dependencies to them while they still are important for the system to operate properly.

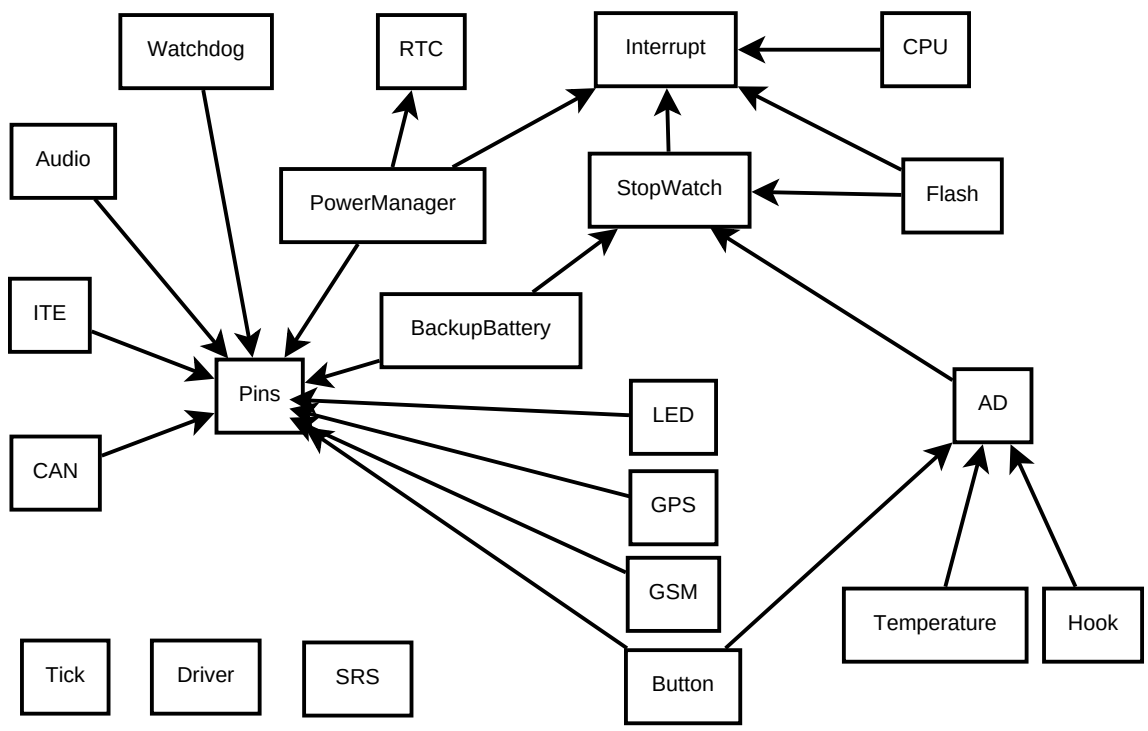

Figure 3.2: Module dependency graph for the hardware abstraction layer of the existing implementation.

Interrupt and CPU modules

The interrupt and CPU modules initialize low-level functionalities of the microcontroller. The modules cooperate to initialize and provide functionality for enabling and disabling interrupts. The functionality provided by these modules is essential to make the platform work at all.

Tick module

The tick module provides an interface to configure a timer to call a given function periodically. This module is used by the operating system to generate a tick that is used to trigger context-switches. This module is essential for the operating system to run.

Pins module

The pins module provides an interface for configuring, reading and writing pins. Pins can be configured to be input or output; all pins can also be enabled or disabled. This module is used by most other HAL modules, so this

module must be ported before all modules that depend on pins functionality.

ITE and driver modules

ITE is the Integrated Test Environment which provides debug capabilities; the driver module uses ITE to create debug printouts from drivers. These modules provide very useful debug tools and are therefore preferable to port early. The hardware dependency of this module is that it uses a serial port to send its output to and receive commands from a user.

Watchdog module

The watchdog module provides an interface for enabling, disabling and kicking a hardware watchdog. A watchdog is a mechanism that restarts the system unless it is kicked with a predetermined interval since the last kick. Under normal circumstances the operating system has a high priority task that handles the kicking of the watchdog. If this task is unable to kick the watchdog the system is considered to have crashed and the system will be restarted by the watchdog.

3.3 Effort estimation

The effort required to port the HAL can be estimate with COCOMO. It is reasonable to estimate the size of the ported HAL modules to be approximately as large as the original modules. So the input to COCOMO is the size of the HAL. The configuration parameters for COCOMO are the ones suggested in the formal estimation section in the overview chapter.

Estimating the effort required porting a single module could be estimated by assuming that the time is proportional to its size. For example if one module contains 5% of all source code then it could then be expected that the effort required is approximately 5% of the entire project.

Table 3.2 show that the total number of lines of source code is 9652. Using this as input to COCOMO gives an effort estimate of 9.4 man-months for porting the entire HAL. The accuracy of this effort estimate is evaluated in the evaluation chapter.

Module	**Lines of code**
AD	622
Audio	2899
BackupBattery	213
Button	331
CAN	85
CPU	388
Driver	157
Flash	780
GPS	286
GSM	530
Hook	406
Interrupt	270
ITE	127
LED	320
Pins	498
PowerManager	610
RTC	221
SRS	408
StopWatch	175
Temperature	195
Tick	42
Watchdog	89
Total	**9652**

Table 3.2: Number of lines of source code for HAL modules before porting.

Chapter 4

Porting

This chapter describes some of the challenges in porting the software platform from the TEM3 hardware platform to the TGU hardware platform. Trivial parts of the porting are intentionally excluded since it does not provide any useful knowledge.

4.1 Module priorities

The first step in the porting process is to determine in which order modules should be ported. The dependencies between modules are one of the most important considerations when determining porting order. The size of modules does also affect the preferred order of porting. Small modules with high functionality are preferable to port early. The reason for preferring to port small modules early is that typically less time is spent writing code before the first tests can be performed. Testing functionality is important since it is a good indication of progress in the project.

The module that should be assigned highest priority is the interrupt module. It has many direct and indirect dependencies and used by the core of the operating system. The many dependencies require this module to be ported first.

The second most important module is the tick module which is used by the task switching mechanism of the operating system. The reason for porting this module early is that this module is very small but enables a lot of other functionality.

When interrupts, tick and pins modules are ported it is possible to port

Priority	**Module**
1	Interrupt
2	Tick
3	Pins
4	ITE
5	Watchdog
6	StopWatch
7	Flash
8	GPS

Table 4.1: Porting priority for modules.

the ITE module. Since the ITE module provides interactive debug functionality it is desirable to port this module early.

Before porting more modules that increase the functionality of the ported system it is a good idea to port the watchdog module which is a system recovery mechanism that is useful when porting.

Table 4.1 lists the priorities for the modules that was planned to be ported in the first stage of this project.

4.2 Single bank flash memory

Flash memories are typically organized into what is called banks. This concept allows different banks to be operated in parallel [5]. Multi bank flash memories allow reading one bank while simultaneously erasing another. Since flash memories need to be erased before written to erase is a common operation when the flash memory is used for storing file system data.

The software platform originally used the flash memory to store program code and file system data. Since program code is executed continuously and file system operations might occur at any time, program code and file system data should be stored in different banks to allow them to work in parallel. The flash memory used in the TGU platform has a fairly uncommon organization with only one bank. The one bank organization does not allow parallel operation which means that it would not be possible to both execute program code and use a file system on the flash memory.

The solutions used to allow file system operations was to copy all program

code to RAM at startup and from that point on only be executing program code from RAM. This solution avoids the problems that more than one operation is performed in the same bank at the same time. A drawback with this solution is that the startup time is increased because of the copying of the program code, however the executing speed will be improved since RAM has lower latency than flash memories.

4.3 Refactoring stopwatch

All HAL modules are divided into two parts, platform independent and platform specific. The platform independent part of the modules defines the abstract interface to the module. The purpose with an abstract interface is that it should be hardware independent and not need to be changed when the underlying hardware is changed.

The stopwatch module had an abstract interface that clearly was influenced by how the hardware of the TEM3 is working. This caused problems when porting because the assumptions conflicted with how the TGU works. The assumption made was that all HAL implementations could use the same platform independent data structure to store state information. The problem was that the data structure defined to be platform independent clearly was designed to work with the TEM3 and was therefore not platform independent.

The solution to his problem was to move the definition of state information to the platform specific part of the module. This allowed the two hardware platforms to have their own definitions of state.

This problem was easily identified and solved, however it shows that the software platform might need to be modified in unexpected places when porting. These kinds of problems are hard to identify before the code is being ported and is therefore hard to account for when planning.

Chapter 5

Evaluation

This chapter summarizes what has been ported and provides a performance comparison between the TEM3 and the TGU.

5.1 Functionality

The majority of porting work has been focused at implementing HAL modules. Therefore the progress in porting the entire system could be measured by considering progress in porting HAL modules.

Module	**Progress**
Audio[1]	0%
CAN [1]	0%
Flash	95%
GPS	80%
GSM[2]	10%
Interrupt	100%
ITE	100%
Pins	90%
PowerManager	10%
RTC	100%
Stopwatch	100%
Tick	100%
Watchdog	100%

Table 5.1: Porting progress for each HAL module.

It should be noted that even though most modules have been ported to a high degree still a lot of work remains since the modules with low or no progress are larger than the other modules.

All modules required by the operating system have been ported completely, which means that the operating system is fully functional.

5.2 Performance and power consumption

Performance and power consumption are among the most important characteristics when evaluating hardware platforms for an embedded system within the automotive domain.

Performance can be many things depending on what a system should be used for. This performance evaluation focuses on the performance characteristics that are important for automotive applications. The performance measurement that is the most important for the system being ported is startup time. Two other measurements that are of interest is computational per-

[1]This module has not been considered for porting in this project.

[2]The majority of the GSM functionality is not located in the HAL driver, which means that 100% progress is not the same as fully functional GSM.

formance and memory bandwidth since those measurements indicate which type of applications the system is suitable for.

It is well known that there is a relation between performance and power consumption, typically higher performance means higher power consumption. Therefore power consumption was measured and used in comparisons of how energy efficient the two hardware platforms are.

All performance and power consumption tests were executed on both TEM3 and TGU hardware so that the results can be compared to gain insight into differences and similarities between the two hardware platforms. Since the TGU is evaluated as a replacement to the TEM3, multiple tests have been run for the TGU with different configurations to know the range of performance and power consumptions obtainable.

5.2.1 Configurations

Multiple configurations of the TGU have been used for all tests. Only the default configuration of TEM3 has been used. The configurations of the TGU vary in two ways, clock frequency of the CPU and which instruction set the compiled code uses. Each configuration is given a short name that is used when referring to a particular configuration.

Configuration	**Clock frequency**	**Instruction set**
TEM3	96MHz	CR16C
TGU96T	96MHz	Thumb
TGU96A	96MHz	Arm
TGU192T	192MHz	Thumb
TGU192A	192MHz	Arm

Table 5.2: Configurations used in performance and power consumption tests.

Two clock frequency configurations for the TGU were used. 96MHz was chosen because it is the same frequency that the TEM3 is using. The other clock frequency was 192MHz which is the maximal clock frequency for the TGU. Instruction set was included as part of configurations because it was expected that it would influence both performance and power consumption [6].

5.2.2 Startup time

For existing products there exist requirements on startup time. Therefore it is important to know the startup time of the software on the new hardware platform to know if it meets these requirements. Startup time is measured from when power is turned on until the platform layer is initialized.

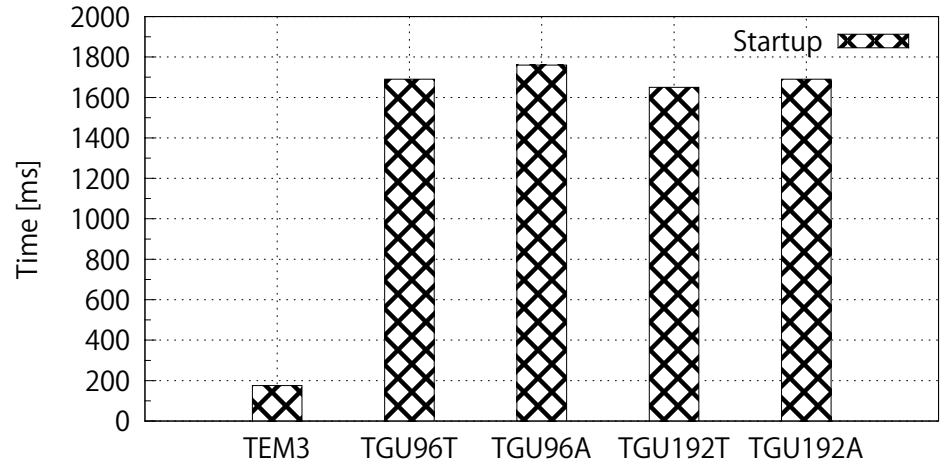

Figure 5.1: Startup time measured from power on until platform was initialized.

Surprisingly the startup of the software platform is approximately 10 times slower on the TGU than the TEM3. The reason for the slow startup time on the TGU is not determined but is expected to be related to the power-on reset circuit. The clock frequency seems to have a neglectable effect on startup time. This is explainable if the majority of the startup time is spent by the power-on reset circuit.

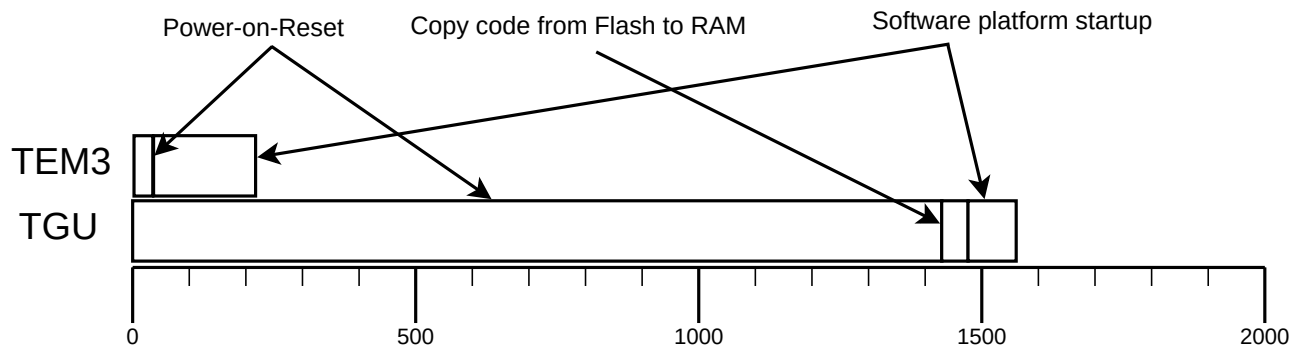

Figure 5.2: Startup sequence for the hardware platforms. The time unit of the x-axis is milliseconds.

Figure 5.2 illustrates the startup sequence of the hardware platforms. It is clear that the actual startup time of the ported software platform is faster. It is also clear that the slow startup time of the ported platform is caused by a very slow power-on reset[1].

5.2.3 Computional performance

The computational performance of the system is in this evaluation measured as the time it takes to execute a computationally intensive program that uses small amounts of memory.

Two computational performance tests were constructed in such a way that they would test different aspects of computation.

Computional test application 1 (Sorting)

The first computational test was a program that sorted numbers. The sorting algorithm used is heap-sort. The implementation is the default sort function available in the software platform. The amount of data sorted was relatively small to avoid that memory performance would influence the results too much since the test should measure computational performance. The intention with this test was to get a representative performance measurement for algorithms that use medium amounts of data.

Computional test application 2 (Math)

The second computational test was a program that performed calculation of an iterative formula, which consisted of addition, multiplication and arithmetic-shifts. The intention with this test was to get a representative performance measurement for algorithms that use very small amounts of data. The amount of data used in this algorithm is so small that it should fit within the registers of the processor.

Results of computional performance tests

The computational performance test in figure 5.3 indicates that the TGU is approximately 3 to 5 times faster than the TEM3 at computations at the

[1]This has not been verified but is the most likely explanation.

same clock frequency and 6 to 10 times faster when running at twice the clock frequency.

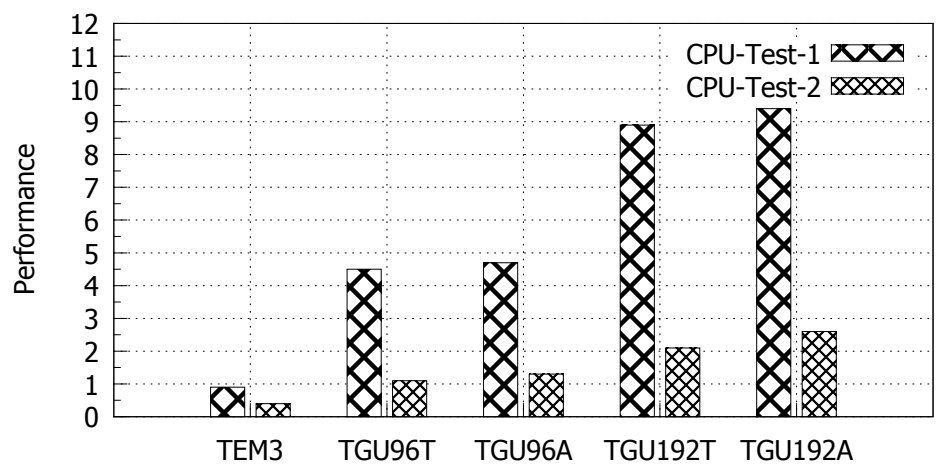

Figure 5.3: Computional performance of platforms. The unit of performance is million iterations per second.

5.2.4 Memory performance

Knowing the memory bandwidth of a system is interesting for at least two reasons, the first being that overall system performance is influenced by the memory bandwidth. The second reason is that it is important to know the memory bandwidth when determining which type of applications the system can be used for, based on memory usage.

Memory test application 1

The first memory test was a program that exchanged data between randomly chosen locations within a memory block. The program was run with memory blocks of different sizes to understand how the performance changed with increasing size of the memory block. The intention with this test was to show the effects of having a data cache and how performance depends on the amount of memory used.

Memory test application 2

The second memory test was a program that copied data repeatedly between two predetermined locations in memory. An alternative way of describing

it is that data was streamed between the two locations. The test program was run with two different configurations, one were small amount of data was copied and another were large amounts of data was copied. The two different configurations were aimed at testing two common memory usage situations. Repeatedly copying small amounts of data between the same locations is a very common situation in operating systems and drivers. Therefore, the memory performance for small amounts of data could be seen as an approximation of how memory performance relates to operating system performance.

Results of memory performance tests

In figure 5.4 memory performance characteristics can be observed for increasing use of memory. The memory performance of TEM3 is very low and does not depend on the size of the data that is manipulated; the reason for this is that the TEM3 does not have a data cache. In contrast to the TEM3 the TGU does have a data cache; the size of the data cache is 16KB. A drop in memory performance for the TGU occurs approximately when memory blocks larger than the data cache are used.

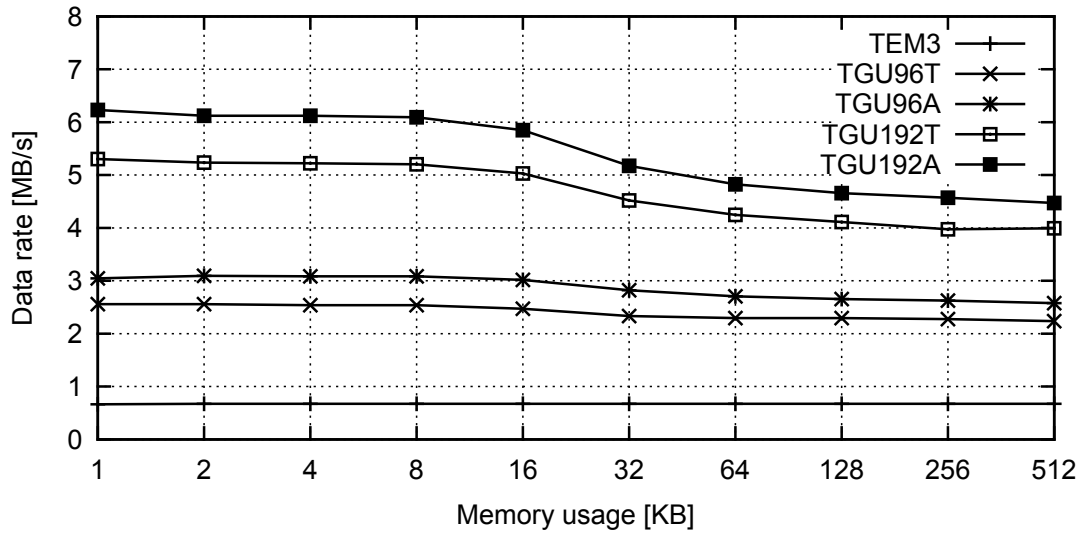

Figure 5.4: Memory performance for test 1.

Figure 5.5 illustrates the effects of the existence of a data cache. When small blocks are copied the data cache usage is expected to be very high but when large blocks are used it is expected to be used very little. The performance difference shows the importance of the data cache; for the 192MHz

configurations of the TGU the data cache improve the memory performance with approximately 45%, which is significant.

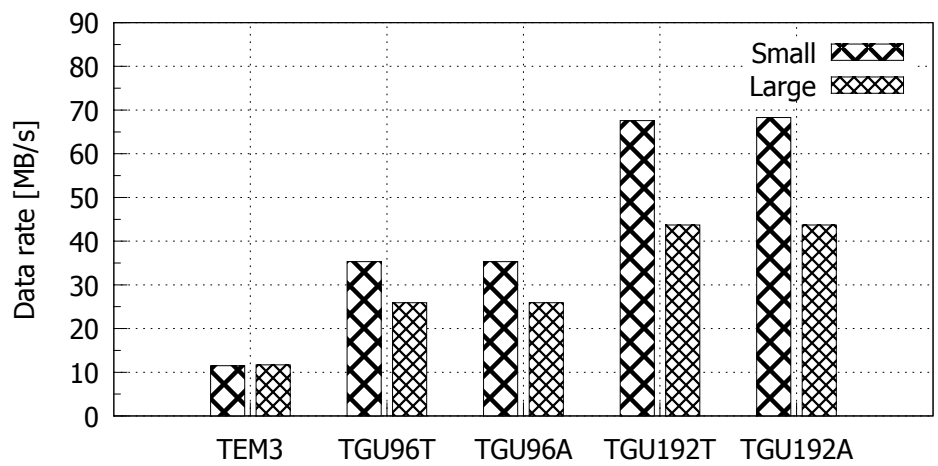

Figure 5.5: Memory performance for test 2.

The overall memory performance of the TGU is approximately 3 times higher than the TEM3 at the same clock frequency and approximately 5 times higher at twice the clock frequency.

5.2.5 Power consumption

The hardware platforms tested are intended for use in vehicles. There are limits to how much power can be used without running the risk of draining the batteries. This makes power consumption an important factor when evaluating the suitability of hardware for vehicles. Power consumption will be measured during idle and load conditions.

Closely related to power consumption is energy efficiency which can be expressed as the amount of computation per unit of energy consumed. All performance tests will be used to determine the energy efficiency of the TEM3 and the various configurations of the TGU.

In figure 5.6 the power consumption for the hardware platforms is shown. It is clear that the TGU consumes significantly more power than the TEM3 during both idle and load conditions. The measurements show that the power consumption for the TGU during idle conditions does not depend on the clock frequency or the instruction set used.

Figure 5.7 shows that the TGU is at least as power efficient as the TEM3 for CPU intensive applications when running at the same clock frequency

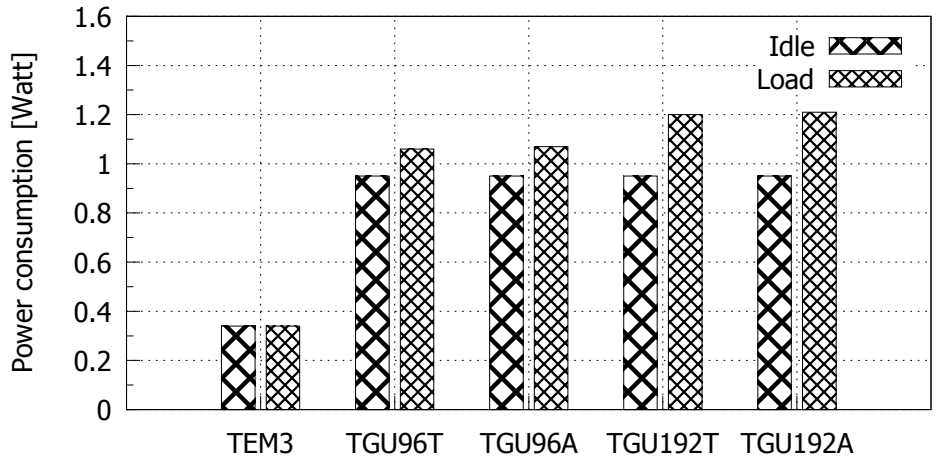

Figure 5.6: Power consumption for idle and load conditions.

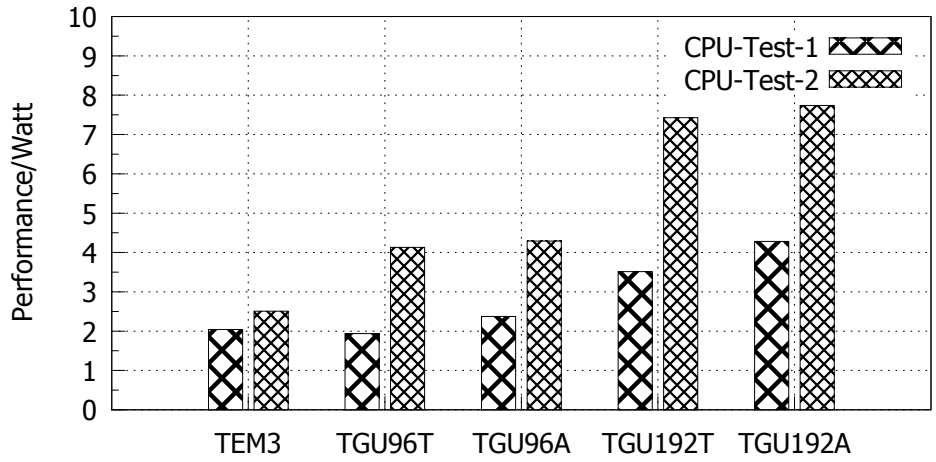

Figure 5.7: Power efficeny on high CPU load.

and significantly more power efficient when running at twice as high clock frequency.

Figure 5.8 shows that the TGU is not as power efficient as the TEM3 for memory intensive applications when running at the same clock frequency but is more power efficient when operating at twice the clock frequency.

Overall the most power efficient hardware configuration is TGU running at 192MHz. That configuration also has the highest computation and memory performance.

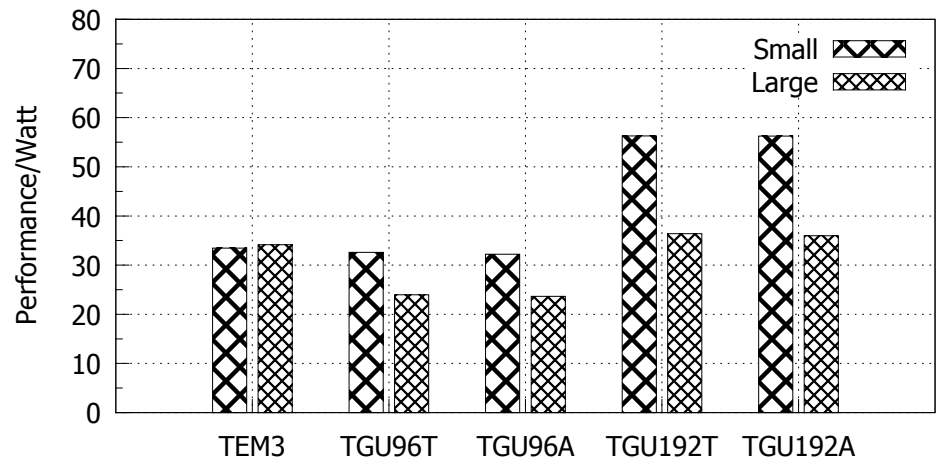

Figure 5.8: Power efficeny on high memory load.

5.3 Effort estimation

In the analysis chapter an effort estimate for porting the HAL was performed. The estimated effort for porting the HAL was 9.4 man-months. The accuracy of this estimate will be evaluated by comparing the estimated effort with the actual effort for the modules that have been ported. If the estimate and the actual effort deviate too much it is an indication that the configuration parameters for the COCOMO models were not suited for this project. If the configuration parameters are not applicable, new ones should be estimated to allow better effort estimates for the remainder of the project. During this porting project 2754 lines of code were developed which is 29% of the expected number of lines needed to port the entire HAL. Approximately 3 man-months was spent developing code which is 32% of the estimated effort. So approximately one third of the porting was completed in one third of the estimated time. This indicates that the effort estimation is reasonably accurate for this project.

Chapter 6

Discussion

In this chapter the result of the porting is discussed. Part of the discussion is reflections about limitations and possible future work.

6.1 Limitations

The porting performed in this thesis is only partial, significant parts of the software platform are not operational because of this. The time limitations forced the lower levels of the platform to be ported to get the core functionality of the operating system running. Some of the core functionalities of a telematics platform were excluded from the porting because they were expected to be too time consuming to port. Most notably the CAN and GSM modules were not ported because of their complexity.

6.2 Conclusions

The ported software platform is fully operational for general use, which means that it can run applications in the same way as the original software platform. With the operating system working, hardware independent modules in the platform and platform layers are working as expected. This shows that the operating system and drivers for simple devices can be ported to a new platform with relatively little effort.

The TGU is computationally a more powerful hardware platform than the TEM3. If computational performance is expected to be a limiting factor

in the future the TGU might be a suitable replacement to the TEM3 if the loss of functionality[1] is acceptable.

6.3 Future work

Since the porting was only partial, continuing work with porting other modules is a natural step for continuing in the same direction. It would be preferable to continue with porting of the CAN and GPS modules to gain confidence in that the hardware platform is suited for the type of telematic applications that the software platform implements.

The MCU used in the TGU has an MMU, which currently is only used for rearranging addresses so that the address of the interrupt vector points to RAM. The MMU could in theory be used to implement memory protection that only allows access to a memory area to the application that allocated it. Utilizing the MMU for full application space protection requires an operating system architecture that is prepared for this. Unfortunately the existing operating system is not prepared for using a MMU. To utilize the MMU for memory protection research has to be done on how the architecture should be modified to allow the use of memory protection.

[1] The TGU does not have a MOST controller.

Glossary

CAN Controller Area Network is a communication bus that is widely used in automotive systems. , 8, 20, 29,30

COCOMO Constructive Cost Model. , 5

GPS Global Positioning System. , 8, 20, 30

GSM Global System for Mobile communications. , 8, 20, 29

HAL Hardware Abstraction Layer. , 9,10, 17, 19,20

HDL Hardware Driver Layer. , 9

ITE Integrated Test Environment is a interactive debug applications that allow the developer to interact with system modules via a command line interface. , 20

KSLOC Kilo Source Lines Of Code. , 5

MCU Microcontroller Unit. , 8

MMU Memory Management Unit. , 8

MOST Media Oriented Systems Transport. , 7,8, 30

PL Platform Layer. , 10

RISC Reduced Instruction Set Computer. , 8

RTC Real-Time Clock. , 20

TEM3 Telematics 3. , 7,8, 10, 15, 17, 19, 21–23, 25–27, 29,30

TGU Telematic Gateway Unit. , 7,8, 15–17, 19, 21–23, 25–27, 29,30

Bibliography

[1] K. Moløkken and M. Jørgensen. A review of surveys on software effort estimation. In *IEEE International Symposium on Empirical Software Engineering, Rome, Italy*. Citeseer, 2003.

[2] R. Dillibabu and K. Krishnaiah. Cost estimation of a software product using COCOMO II. 2000 model-a case study. *International Journal of Project Management*, 23(4):297–307, 2005.

[3] R. Cravotta. Fear and loathing of porting embedded software. *EDN*, 47(17):55–62, 2002.

[4] Mindfire Solutions. Porting: A Development Primer. 2001.

[5] Spansion. *Implementing Simultaneous Operations in a Simultaneous Read/Write Enabled MCP*, 2004.

[6] A. Krishnaswamy and R. Gupta. Profile guided selection of ARM and thumb instructions. *ACM SIGPLAN Notices*, 37(7):56–64, 2002.

LINKÖPINGS UNIVERSITET

Avdelning, Institution
Division, Department

ESLAB,
Dept. of Computer and Information Science
581 83 LINKÖPING

Datum
Date

2010-06-24

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English

☐ _____

Rapporttyp
Report category

☐ Licentiatavhandling
☒ Examensarbete
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport

☐ _____

ISBN
—

ISRN
LITH-IDA-EX--10/028--SE

Serietitel och serienummer **ISSN**
Title of series, numbering —

URL för elektronisk version
http://www.ep.liu.se/exjobb/ida/2010/dd-d/028/

Titel Porting av inbyggda mjukvarusystem för fordonssystem

Title Embedded software porting for automotive applications

Författare Henrik Carlgren
Author

Sammanfattning
Abstract

Developing software is both time consuming and expensive. Finding ways for minimizing development cost is therefore of great interest. One way of reducing cost in software development is reuse of existing software. Porting software is a type of reuse where an existing piece of software is adapted to run in a different environment than originally intended. Successfully porting software requires both good preparations and evaluation of results. Guidelines for this process are provided in this thesis. In this thesis an existing embedded software platform for automotive telematics was partially ported to a new type of hardware.

Nyckelord
Keywords PORTING, SOFTWARE, EMBEDDED

Copyright

Svenska

English