Tampere University

Patrik Parviainen

SOFTWARE MIGRATION - PORTING EMBEDDED APPLICATION INTO A NEW MICROCONTROLLER UNIT

within the STM32Fx07xx product series

Bachelor's thesis
Faculty of Information Technology and Communication Sciences
Examiners: Professor Jukka Vanhala
December 2021

ABSTRACT

Patrik Parviainen: Software migration - Porting embedded application into a new microcontroller unit
Bachelor's thesis
Tampere University
Degree Education in Computing and Electrical Engineering
December 2021

This bachelor's thesis looks at porting embedded software to an updated platform.

The upgrade takes place in a monitoring and safety device resembling a motor control unit, which has a typical operating environment of industrial hoists and cranes. The device is an existing product of Konecranes. The new hardware environment, the device, is otherwise physically identical, except that the primary microcontroller is upgraded to a newer one. This happens within the STM32 microcontroller family and the process is called migration.

The controllers and their peripherals are very similar. Their main difference is their core, the older one being Cortex M3 and the newer M4. The most essential benefit here is more RAM.

The thesis first studies microcontrollers and software migration in general. The study found that there are not standardized methods for migration, as there are basically infinite possible scenarios.

Next, the migration process was approached with general iterative development process, which was fit for migration. The process is also supported with preliminary and finishing actions, the most significant of which are defining comprehensive acceptance criteria and maintaining a consistent source tree.

The practical part includes studying the differences between the mentioned microcontrollers and updating the hardware abstraction layer, which consists of processor-specific libraries and drivers. The results are studied with hardware testing, which was done with a previously implemented test application. This ensures functionality of most peripherals of the microcontroller, but not all. Thus, additional testing is required by debugging the actual user application via the serial communication of the microcontroller. After achieving barely functional user application, the development would be continued with the aid of feedback from automated test sets.

Finally, the thesis theorizes how to achieve the goal of the migration process, which is stable operation of the actual embedded application in the new environment. A systematic, iterative and progressive process of trial and error is used.

Keywords: Microcontroller, Porting, Migration, Embedded Software, Embedded systems

TIIVISTELMÄ

Patrik Parviainen: Sulautetun ohjelmiston siirto saman mikrokontrolleriperheen sisällä
Kandidaatintyö
Tampereen yliopisto
Tieto- ja sähkötekniikan koulutus
Joulukuu 2021

Tässä kandidaatintyössä tarkastellaan sulautetun ohjelmiston siirrettävyyttä laitteiston päivityksen yhteydessä.

Päivitys toteutetaan valvonta- ja turvalaitteessa, jonka tyypillinen toimintaympäristö on teollisuusnostimien ja -nostureiden yhteydessä. Laite on Konecranes-yrityksen jo olemassa oleva tuote. Uusi laiteympäristö on fyysisellä tasolla muuten identtinen, mutta sen keskiössä oleva mikrokontrolleri päivitetään uudempaan saman STM32-mikrokontrolleriperheen sisällä. Kontrollerit ovat oheislaitteineen hyvin samanlaisia. Mikrokontrollerien oleellisin ero on päivitys Cortex-M3 ytimestä ytimeen Cortex-M4. Oleellisimpana hyötynä on 50 prosenttia suurempi keskusmuisti, ja porttauksen tarve juontuukin toimintaympäristön vaatimusten lisääntymisen myötä kasvaneesta keskusmuistin tarpeesta.

Työn varsinainen tavoite on saada ohjelmisto toimimaan päivitetyllä alustalla samalla vakaudella ja entistä paremmalla suorituskyvyllä.

Ensin tutkitaan mikrokontrollereita ja ohjelmistojen siirrettävyyttä yleisesti. Selvityksesssä todettiin, ettei tähän löydy standardoituja menetelmiä, sillä mahdollisia siirtoskenaarioita on käytännössä lukematon määrä erilaisista prossoreista, ohjelmointikielistä ja sovelluksista johtuen.

Seuraavaksi siirtoprosessia lähestyttiin yleisen tason iteratiivisella kehitysprosessilla. Prosessi sovellettiin ohjelmistojen siirtämistä varten, ja sitä tuetaan myös alustavilla- ja viimeistelytoimilla. Näistä merkittävimpiä tässä yhteydessä ovat kattavien hyväksymiskriteerien määrittäminen ja yhdenmukaisen lähdekoodirakenteen ylläpitäminen.

Käytännön osassa tarkastellaan mainittujen mikrokontrollereiden välisiä eroja ja päivitetään laitteiston abstrahointikerros, joka koostuu prosessorikohtaisista kirjastoista ja ohjaimista. Tuloksia tutkitaan laitteistotestauksella, jota toteutetaan aiemmin toteutulle laitteiston testisovelluksella. Sovelluksella voidaan varmistaa useimpien oheislaitteiden toiminta, muttei koko kokonaisuutta. Tätä varten täytyy käyttää myös muita testaustapoja, esimerkiksi debuggausta mikrokontrollerin sarjaportin kautta käyttäjäapplikaation pääohjelman alustusvaiheissa.

Aluksi pyritään kehittämään vähimmäisvaatimukset toteuttava sovellus, joka kääntyy ja jolla on mahdollista ajaa varsinaisen käyttäjäapplikaation testisarjoja. Tällä tavoin yksityiskohtiin ei jumiuduta liian aikaisin, kun koko toiminnallinen rakenne voi vielä muuttua. Tämän jälkeen samaa prosessimallia käyttäen optimoidaan ohjelmakoodia, dokumentoidaan huolellisesti ja otetaan huomioon ohjelmiston tulevaisuus ja siirrettävyys.

Lopuksi, kandidaatintyössä tarkastellaan kuinka siirtoprosessin tavoite voidaan saavuttaa. Tähän vastataan systemaattisella iteratiivisella ja progressiivisella kokeile-ja-korjaa -prosessilla.

Avainsanat: mikrokontrolleri, porttaus, siirrettävyys, sulautetut järjestelmät, sulautettu ohjelmisto

PREFACE

Työ on tehty yhteistyössä yrityksen Konecranes Oy kanssa. Ensimmäinen oman alan tietotyö tänä kesänä *Asset Intelligence Developement* -tiimissä on ollut hieno ja opettavainen kokemus.

Kandidaatin töiden yleinen tavoite toteutui, mutta käytännön arvoon jäi kehitettävää. Työn aikana kehitin valmiuksiani tieteelliseen ajatteluun ja työskentelytapoihin, sekä koen oppineeni tärkeitä asioita diplomityön tekemistä varten, tärkeimpänä keskittyminen yhteen asiaan kerrallaan. Kandidaatin työn loppusilaukset venähtivät kuitenkin koko syksyn mittaiseksi, kun yritin loppuun asti tavoitella käytännön osuuden alkuperäistä tavoitetta.

Haluan kiittää teitä, jotka olette auttaneet minua ymmärtämään sulautettua ohjelmistoympäristöä: Jouni Jocklin, Jari Pehkonen, Daniel Lorentzen, Ville Kilpinen sekä Anton Shcherbakov. Olen kiitollinen, että sain työskennellä työn parissa myös työajalla. Kiitos ohjauksesta ja kommenteista Jukka Vanhalalle yliopiston puolelta. Tahdon osoittaa kiitoksen myös Leo Tiittaselle motivoivasta lounas- ja treeniseurasta, jotka edistivät työn kulkua.

at Tampere, 14th December 2021

Patrik Parviainen

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

ADC	Analog-digital converter is a integrated peripheral in the MCU and it is used for converting analog input signals, voltages, into digital numeric values.
CPU	Central Processing Unit. The electronic circuitry that executes the program instructions.
DUT	Device under test, also known as equipment under test (EUT) and unit under test (UUT).
HW	Hardware. In embedded systems HW is used to reference the physical parts of the system, for example the PCB, MCU and connectors.
ISO	International Organization for Standardization
Legacy software	Software that has been used for a long time still serving a purpose and would need modernization.
MCU	A Microcontroller Unit. A single intergrated circuit, including the core CPU along with memory and other peripherals.
PCB	A printed circuit board. A laminate of conducting and insulating sections forming electrical functions and providing a mechanical base for components.
Software Porting	Software porting is the engineering process that transforms software so that it will operate within a new target environment.
SW	Software. In embedded systems HW is used to reference the programmable parts of the system.
TUNI	Tampere Universities
USART	Universal Synchronous and Asynchronous Receiver-Transmitter is a serial interface device that can be used for synchronous or asynchronous communication. In embedded systems it is often integrated into the MCU and is one of the key peripherals.

1. INTRODUCTION

The future can still not be foreseen, but nowadays considering portability in software development should be self-evident. The Fourth Industrial Revolution (4IR), with the fusion of the digital and the physical world, has many new technologies emerging in the market. Most of these are industrial embedded systems by definition and they are built around microcontrollers (MCUs), which are discussed in chapter 2.1. The requirements for connectivity and software performance are ever-increasing, and the MCUs are evolving with them. Those adapting to the revolution most efficiently, by preparing to use the best available technology, are the ones ahead of the business competition.

Software porting refers to the engineering process of modifying a program with the goal of it running on another system [1] [2]. Ideally, a portable software could be run in the new environment with zero re-developing costs, but in general, an application is defined as portable, if the costs of porting are less than re-developement[3] [4]. Porting generates financial savings if time can be saved in design, implementation, and testing by using existing application source code[5]. The habit of considering portability during the development process might generate little extra work, but when possible porting is considered from the beginning, the process should be considerably cheaper compared to creating a new product[6]. Portability is studied in chapter 2.3.

The MCU manufacturer, ST Microelectronics, calls the porting process *migration*, when it is about MCUs within the same series[7]. The most popular reason for migration is caused by increased software requirements, which put extra demands on memory sizes[4]. That is also the case in the scope of this thesis. The product is described in chapter 2.2 and processor selection is discussed in chapter 2.4.

This thesis studies the migration process, the target of which is to enable the embedded application to run on a new platform with better performance while maintaining stability. This goal can be verified by running a vast amount of regression tests with a prototype containing the MCU upgrade.

The process of software migration includes an innumerable amount of different scenarios, depending on the environment, the programming languages, and the application itself[8]. The general theory of migration is discussed in chapter 3. The scope of this document is still unique, even though the product concerned uses the most popular em-

bedded programming language, C, and a popular microcontroller with the most popular processor core architecture[9] [10] [11]. However, in the scope of this document, the real-world large-scale embedded application has been developed for multiple years and the hardware changes are virtually non-existent. Only the Microcontroller Unit (MCU) is upgraded within the same pin-compatible STM32 product family, from STM32F207GZ[12] to STM32F407GZ[13]. Their differences are discussed in chapter 4.1.

This thesis aims to complete the preliminary work for the migration, and provide a process template for accomplishing the migration thoroughly, by using a process template provided in chapter 3. It was discovered during the practical part of the thesis, that the embedded application in question is more complex than expected, but a case example of some work is presented in chapter 4.

2. GENERAL BACKGROUND

2.1 Microcontroller

Microcontroller(MCU) usage has a wide range from simple home instruments to industrial high-performance safety devices, and the numbers are still growing[14]. The MCUs are plenty even in consumer-level products. For example, the present-day modern automobile uses way over 50 MCUs[15].

MCUs have evolved significantly with the development of processors and system-on-chip (SoC) design. In real-world-based specific applications, MCUs are often preferred instead of microcomputer-based systems, because practical peripherals and tools for real-world interaction are integrated, for example, ADC, PWM, and Timers[14]. The processor itself is only a small part of the chip surface area. The integrated blocks are implemented as internal hardware in the MCU, presented in figure 2.1.

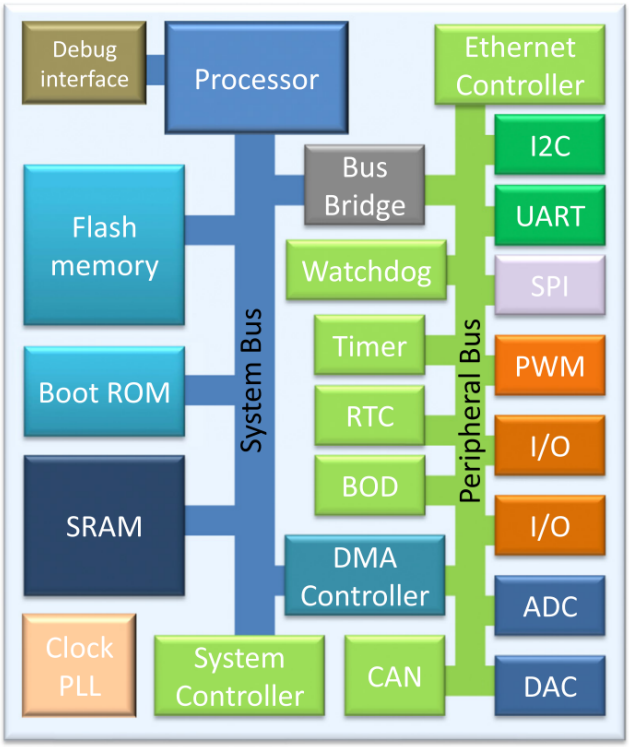

Figure 2.1. *A microcontroller contains many different blocks. [11, Chapter 1]*

The internal modularity of an MCU usually simplifies the development workload, as the developer should be able to stick to standardized high-level code, C for example. The lower level implementation is abstracted and most of the hardware-specific parts should be ready-made libraries and drivers provided by the manufacturer [16]. For example, between two microcontrollers every other peripheral represented in the figure 2.1 could be the same, but the processor. That is almost the scenario in this thesis. This way the interfaces of most peripherals could stay the same during the migration process and just a little configuration would be needed.

2.2 Description of the product

The commercial name of the product is *Condition Monitoring Unit MASTER PRO*, but in this thesis, the project name *ECU* is used for more concise reference.

The ECU is a programmable electronic device for the overload supervision and the protection of a hoist, and the calculation of the basic condition monitoring values of the hoist[17]. For example, the ECU can force a slowdown of the movement of the hoist or the crane at some set limits. The gathered hoist usage and condition data can be used for anticipating maintenance needs and optimizing costs. The data can also be read remotely as the data are sent via an Ethernet gateway to a cloud-based customer portal[18]. This provides value for the end customer, the crane owners, and eases the work of the service personnel.

Figure 2.2. *Condition Monitoring Unit MASTER PRO (ECU). [17]*

Physically the ECU is a cubical device, seen in figure 2.2, with a plastic casing of a volume of little over one liter, consisting of three circuit boards full of electrical components. In the

figure, the two green 18-pin terminal block connectors are used for electrically connecting the ECU with its environment. The upper one consists of *48 V AC* power signals. The lower one is for lower voltage DC measurement signals. [17]

The ECU operates with two different architecture microcontrollers because the application operation is synchronized and cross-checked between MCUs in each cycle of the main loop for reliable operation. The primary MCU is the STM32-series controller with the full user application. The secondary one uses dsPIC architecture and it double-checks most vital safety-related operations. The inter-MCU communication uses UART.

For the average user, a service technician, the use of the ECU is performed via the human-machine interface (HMI) and an ethernet cable. The HMI menu consists of an LCD screen, four buttons, and two LEDs for indicating operating status and errors. The HMI can be used for monitoring value, ensuring correct options, and changing them. For example, the user can read the hoist load in tons, set CAN speed, or read the software revision number. The ethernet link is used for general communication with ECU during production, testing, and installation. The primary functions are both updating the application and reading and writing of parameters, which are done with custom PC applications.

For developers, there are more connectors inside the protective plastic casing. There is a programming and debug connector for each of the two MCUs. The initial application programming is done via these, as the ethernet connection needs a framework and a bootloader to work. In addition, there are also 3-pin TTL serial connectors for each of the two MCUs for serial communication. The serial connection is used for reading debug messages and operating applications, for example, the HW test application, but also a version of the user application with a serial interface implemented.

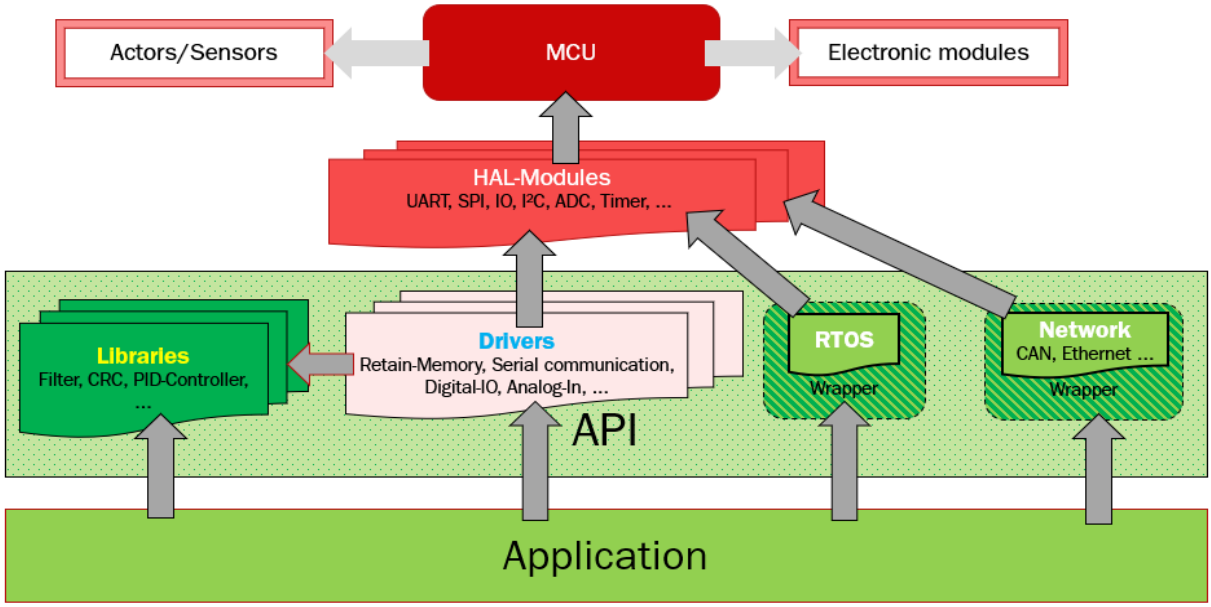

Figure 2.3. *Layered software architecture.*

In figure 2.3 an abstract model of the layered software architecture is shown. In an ideal situation, changes should be done only in processor-specific parts, mostly in drivers, libraries, and hardware abstraction layer modules (HAL).

2.3 Portability

The optimal level of software portability in the embedded application would mean the application's independence of processor architecture. [1] Practically, this would mean that the application consists only of generic C and the migration process would only consist of changing ready-made hardware-specific code files, for example, library and driver files, provided by the MCU manufacturer[8, p. 173] [19].

In smaller projects using the same series of microcontrollers, porting could be as simple as selecting the processor in the compiler, which can convert high-level source code into processor-specific code. This would require the availability of an appropriate toolset. [1] However, in large-scale embedded software projects, it might become difficult for every developer to consider portability, or it might simply be forgotten[6] [8, p.169]. Also, the dependability on specific compiler raises controversiality among developers, as the portability would break instantly when the development tool support becomes obsolete. [1] Thus, the actual application program is not instantly portable and there are many minor details in the way of achieving the target of the process. However, software quality metrics (ISO-9126) have been generally followed in the development process, facilitating the process significantly. [20]

Breaking portability can be a beneficial considered decision. There is always some level of compromise between optimization resource use and portability. Some of the possible processor-specific optimized functions are low-power modes, arithmetic units, and special peripherals. If some efficiency vectors are valued more than portability, breaking the portability is justified. [1]

Even when portability is broken, the processor-specific code should be isolated from the portable code, so the manual porting work will be convenient in the future.

2.4 Choosing MCU

Microcontrollers in embedded systems have a specific purpose and their resources are designed to be optimized for this given purpose, but *The only constant in life is change.* The varying circumstances and refinement of intended use might force a porting process, which usually isn't the desired scenario.

Many factors might force a product to be ported. Real-time performance requirements might increase, the application could demand more memory, or the industry could re-

quire new peripherals. [11, Ch.1] The industrial internet of things (IIoT) might introduce a need for new kinds of peripherals. Programming languages and even architecture might become outdated. There could even be a worldwide pandemic affecting microchip availability and thus prices. An MCU might also become too expensive if the mass production volumes go down. Production volumes also define the amount of general support. The porting process can be avoided for longer if the MCU selection is considered thoroughly and immediately after the requirements specifications.[3]

Having only the necessary capabilities, the MCUs are a cost-effective option in terms of production cost, years of operation, and power consumption. The most typical factors affecting general processor selection in a new product are cost, performance, physical size, memory type and sizes, peripheral set, and power consumption.[3]

As for existing products, considering processor selection thoroughly might be even more significant, as it defines the required amount of re-development. Multiple factors may generate this extra workload in the porting process. Some of the most significant factors for a successful MCU selection in an existing product consist of, but are not limited to the following:

- The chips are pin-compatible.
- The same peripheral functions are supported within the same interfaces and pins.
- Same instruction set is used.
- The source code can be the same language and remain unchanged.
- There are up-to-date guides by the manufacturer.
- The future supportability is ensured, due to popularity.

Taking all of these into account might provide significant savings. [11, Ch. 1] In some cases all of the mentioned can not be fulfilled, but the chosen MCU, STM32F4, seems to be fit. The chips are pin-compatible, the core uses the same instruction set and peripheral functions are mainly similar. There is a 50 percent increase in random access memory, from *128 KB* to *196 KB*. Due to increased bandwidth, some frequencies are also a bit higher, for example in the core, timers, SPIs, UARTs, and ADCs.

Of all the processor architectures, ARM is nowadays the most popular. Specifically the Cortex-M3(F2) and Cortex-M4(F4) series have made a breakthrough in both mobile and industrial devices. [11]

2.5 Acceptance criteria and testing process

Acceptance criteria are unambiguous, preferably measurable, criteria regarding software completion. When acceptance criteria are met, the task can be considered done. The criteria do not necessarily describe all measurable characteristics but emphasize key issues

that are essential for acceptance. [21, p.48]

In principle, all the software developing queue entries should be unambiguous and linked with acceptance criteria, especially in test-driven development. [21] However, the porting process is usually difficult to determine in advance and to split into smaller tasks. In this case, the general level requirements for the porting process are clear. The main application is required to be operated in an upgraded environment with the same stability. This goal of the porting process can be verified by running comprehensive automated and manual tests on the upgraded ECU unit with the latest release of actual user application running.

When the acceptance criteria are clear, the development processes, especially with embedded systems, should include continuous testing for verifying the results. The testing should be executed according to a plan. [8] [21] The three-layered V-model presented in figure 2.4 describes the relations between requirements, implementation and testing in a software development process.

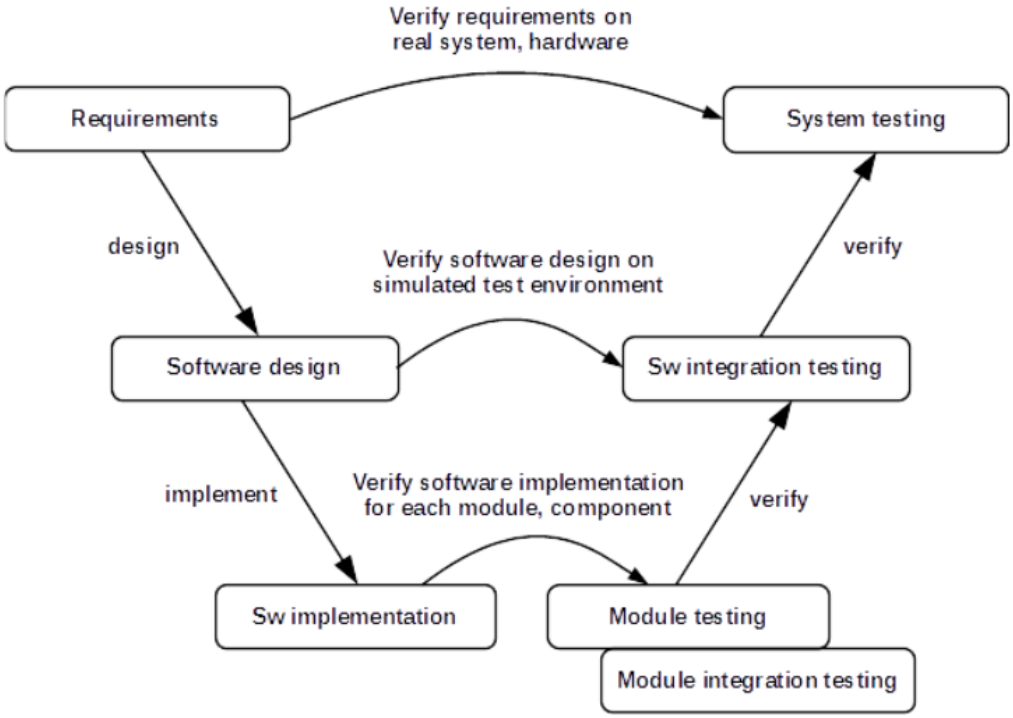

Figure 2.4. *The different levels of testing in the V-model of software development.*

The V-model is meant for software development, and it can be used for the migration process to describe the depth of testing. Even though the embedded application cannot completely be tested elsewhere than with the actual software in a real environment, it is recommended to first test the hardware functionality a simple approach, as seen in the uppermost arrow of the figure 2.4. [8, p. 154] A hardware test application has been made for this purpose.

An example of acceptance criteria are the requirements for the hardware test application, which specify the following functionality[22] for the ECU:

- Ability to set the state of MCU pins
- Ability to read the state of MCU pins
- Ability to send messages through the onboard SPI or I2C buses.
- Ability to write to and read from the external memories
- Ability to read results of the A/D conversions
- Ability to set PWM duty for analog output
- Ability to read states of the onboard buttons
- Ability to test the Ethernet switch
- Ability to test the RTC (test frequency for frequency measurement)
- Ability to send CAN messages
- Ability to write to the LCD display

These requirements serve as acceptance criteria for the first stage of the main porting process. Meeting these requirements is necessary for executing even the most basic level of the actual user application.

The case example in chapter 4 includes verifying the specified hardware requirements on a real hardware system. The testing procedure consists of preparation, actual measurements, recording of results, and comparing results with F4 to expected results of F2.

3. MIGRATION PROCESS

Thorough preparation is crucial for successful software migration. Instead of directly jumping into implementing the migration, studying a systematic approach is preferred.

After examining different factors regarding migration, the actual process can be defined. The migration process can be divided into four stages according to the objectives as follows:

1. The preliminary phase: describing the migration process and the acceptance criteria, which verify a completed migration.
2. The main phase of porting: producing to some extent functional user application and considering structural changes.
3. Refining: completely fulfilled acceptance criteria via optimization and a single source tree for both platforms.
4. Finishing: documentation, updating guidelines, and committing to revision control.

There could also be more additional phases in between the preliminary and finishing phases. In this thesis, two are used. The main and refining phases consist of the general quadruple principle of analyzing, designing, implementing, and test. The porting process is visualized with a flow chart presented in the figure 3.1 on the next page.

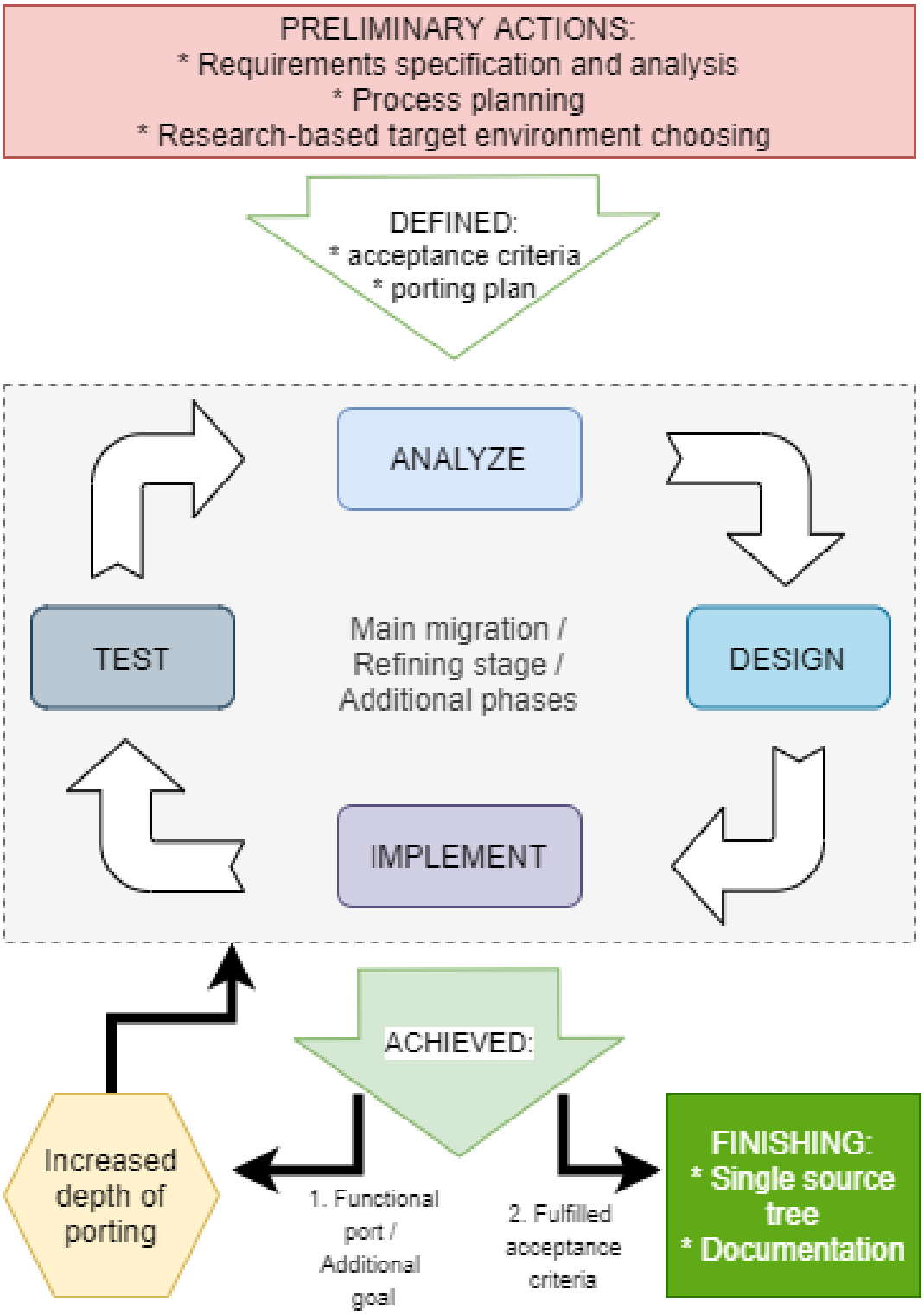

Figure 3.1. *Migration process flow chart*

After outlining the process, each step is studied and explained in the next subchapters.

3.1 Preliminary phase

The preliminary phase is meant to aid with the consistency of the migration process. The phase consists of requirements analysis and planning the process accordingly.

Some assessment for costs and benefits should be done to ensure the viability of the migration. At this point, it is still relatively easy to cancel the migration process, before there are too many sunk costs. The most significant factors are the code portability, the magnitude of architectural changes, and the team experience. [1]

If the porting seems the most viable option, the process can continue. The requirements should be created and documented, and the target environment should be designed accordingly. The requirements should be molded into comprehensive acceptance criteria, which define the accomplishment of the migration process. The criteria can help with the scope of the project. The environment design consists of hardware changes, starting with the choice of the MCU and then adapting the rest of the system with it.

3.2 The main migration

The main phase of the migration process can be split into multiple sections regarding the milestones of the final goal. Each of the intermediate stages can be approached with the same general principle iteration of analyzing, designing, implementing, and testing. Each stage should apply this principle in increasing levels of depth, quality, and practicality.

A good example of the first objective of the process is implementing a barely functional port for the target environment, as the developed automated test sets for the actual user application can't be run before the application is to some extent operable.

In this phase, the analysis consists of reviewing the source code, its functional structure, and portability. Similar recurring parts, that are necessary to change, should be recorded.

Next, during design, reproducible solutions should be considered. The processor-specific parts should be separated from the portable code, and thus there might be needs for higher levels of abstraction. The functional structure may also require re-planning if portability and performance are conflicting.

Then obviously the planned changes are implemented. This consists of updating the hardware abstraction layer, which consists of libraries, drivers and allowing for selecting the hardware-specific configurations. Here, compliance with the software quality metrics will facilitate future work. [20]

The last phase of testing should provide feedback for the next iteration and verify working parts. This can be done with a test application or debug compiling. Not everything can be tested with just another, so both are needed.

A possible test application should be built so that it is initialized the same way that the user application is. This means using the same peripheral settings and configuration files.

The other way is to use a debugger or debug-compiling, in which debug messages are printed into the serial communication port and read there with a PC. This way the tracking of error sources might be easier, especially when the initialization is reviewed.

3.3 Refining

Some refining is often needed when a functional port of the user application is ready, for example, the application boots and sends a start-up confirmation via serial port or human-machine interface shows correct menu display. This refining consists of optimizing performance vectors, improvements of portability and documentation. The previously used cycle can be used at a different level.

The loop can be started at any point, so let's start with the main difference, testing. When there is some kind of functional port, using the primary automated test set for testing can be started. Usually, these kinds of tests provide exact feedback, and error sourcing becomes less burdensome. Fixing should be done in the order in which the most fundamental ones are handled first.

The tests should provide targets for source code analysis. In addition, nonportable solutions made in the previous stage should be analyzed. This may be way easier by using version control system features. Further analysis might be required if the application has some special or high-intensity features.

In the refining stage, design should focus on long-term quality. Conditional compiling, structural decisions and hardware-specific parts should leave room for future modifications. Possible needs for performance improvements may require a special design with the aid of datasheets and manuals. At this point, the implementations should be done precisely considering long-term software quality. There is usually less trial-and-error the further migration process goes. [20]

3.4 Finalizing

Finalizing the project happens by fulfilling acceptance criteria. In addition to this, the team should review the process.

Committing to the revision control should be scheduled with the developing team to avoid generating possible unnecessary troubleshooting. It can be agreed that after the commit, the team members run their tests in different environments, after which the results are given for the persons performing the port. [23]

Especially in the embedded systems, when everything seems to work accordingly in laboratory circumstances, an unsuspecting attitude might become expensive. The migration process should be finished with comprehensive testing of the migrated application in a controlled real user environment with all possible external interfaces, peripheral devices, and error sources. [1]

4. CASE

In this chapter, the general differences between the processors are studied, the peripheral requirements are specified, and the features are tested with the hardware test application.

4.1 Processor comparison

The differences of features in both processors are presented in table 4.1. The key differences are the core and SRAM size. The F4 has a bit higher bandwidths overall, but the peripherals are almost identical. The full table including all the similarities is included in appendix A.

Table 4.1. *STM32 series feature comparison. [7]*

Peripheral		STM32 series	
		F2	F4
Core		Cortex-M3 r2p0 w/ MPU	Cortex-M4 r0p1 w/ FPU and MPU
FLASH memory		1024 KB	1024 KB
SRAM		128 KB	128 + 64 KB
Max CPU Frequency		120 MHz	168 MHz
RTC		Hardware calendar	Hardware calendar + subseconds
COMs	SPI(I2S)	3(2)	
	Max freq.	up to 15 or 30 Mbits/s	up to 37.5 Mbits/s
	I2C	3	2
	UART	2	2

	Max freq.	3.75 or 7.5 Mbits/s	10.5 Mbits/s
GPIOs		140	up to 140
12-bit ADC		3	3
Max sampling freq.		2 MSPS	2.4 MSPS

These kinds of peripheral differences can be examined with datasheet comparison, using a hardware test application, and by debug messages.

4.2 Hardware Test Application

A hardware test application can be used for verifying the requirements mentioned in the section 2.5. [22] For this work, two ECUs in a test box were used. The ECUs are identical, except for the primary microcontroller. The other one has the F2 MCU and the other has the F4 MCU.

In preparation for the test procedures, the PCBs were visually checked to be faultless and power supplies were checked. The test box inputs were checked to be secure. The correct operating voltages of *3.3 V, 5 V, 10 V* and *12 V* were measured on the MCU board in the corresponding connectors, according to the electrical schematic. [22] The tests were performed in a suitable laboratory with all the necessary equipment, according to the safety protocol SFS6002. [24] [25]

Figure 4.1. *Two ECUs in a test box. HW test application OK screens can be seen.*

The tests are done according to *ECU Prototype Production Test Specification*. [22] The most simple tests are listed here in brief. HW test applications were successfully flashed into both of the devices via programming connectors. The applications are run in the ECU devices but operated via serial bus on PuTTY SSH client PC application. The serial bus is connected to PC USB-port via TTL-232R-3V3-connector. The connector works as expected in both devices.

After flashing the ECUs with the HW test application, the LCD screens show an OK message, as seen in figure 4.1. In addition, the green and red status LEDs were successfully tested using *pinset* and *pinreset* commands on the F4 MCU and MCU2. An interactive *Buttontest* command was used to verify that buttons in the human-machine interface are working correctly.

In the subsections of this chapter, the testing results of peripherals are documented for both F2 and F4. The goal of this procedure is to verify the correct behavior on the hardware level.

4.2.1 LCD

The LCD panel is provided by an external manufacturer, but the 6800 interface and 18-pin connection to the MCU board remains the same. The OK-screen shown in picture 4.1 verifies that the basic function of LCD is operating correctly. The backlight of the LCD was powered off and on successfully using *pinset* command on the dedicated pin.

Additionally, the editing of the content shown in the LCD screen was tested using commands *lcdqr, lcdp, and lcdclear.* These print a QR code to the screen, print a parameter text, and clear the screen, accordingly.

Figure 4.2. *Test print is seen on the LCD using HW test application.*

Test prints successfully appeared on the screen in both devices and took identical places.

4.2.2 Digital inputs

The ECU has *48 V AC* inputs, as mentioned in the chapter 2.2. Eight of these serve as digital inputs for hoist control and multifunction (MF) programming.

As seen in figure 4.3, signals named *HIN, LIN, and FIN* stand for requests for hoisting, lowering, and fast speed, respectively. MFI means programmable multifunction input. In

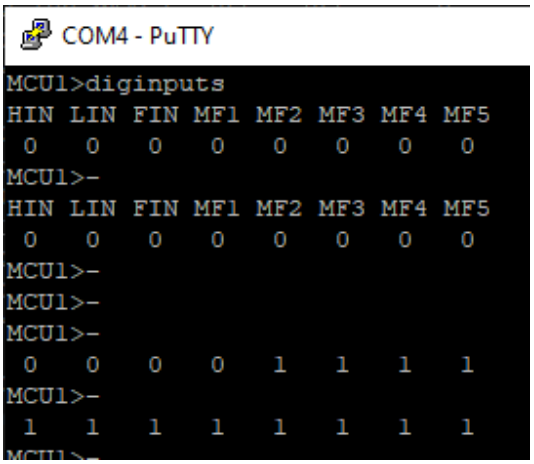

Figure 4.3. *PuTTY terminal. The commands were run on approximately 1 second interval. A delay can be seen.*

this test, the eight digital inputs were connected to *48 V AC* simultaneously right after the second command. In the HW test application terminal, a '-' is used for repeating the last command. It can be observed that the HW test application reacts with a delay, first the last four are activated and then the remaining four. This might be due to Not Connected (NC) status of the pin. This could be solved using a switch to switch between ground and operating voltages, but the overall behavior seems correct, so no further investigation is needed.

The test behaved in the same way with both devices.

4.2.3 Relays

The Ecu has *48 V AC* relays on the same terminal connector. (See figure 2.2) The relays direct the power from supply connectors to relay outputs. The relay outputs are used for sending according to requests or control voltages to hoist-related contactors and control devices.

A *feedback* command shows the current state of different relays, using feedback detection. According to the test specification, all seven shown signals should've been at a high state after command *relayson*, but only two of them were. This was verified by using a multimeter on the corresponding pins, so the feedback function seems to correlate with the real world. The situation was the same with both devices. This might be due to some relay-related software change.

4.2.4 Analog inputs

DC measurement signals in ECU use a *0 to 10 V* voltage input range. There are two analog inputs, which use analog-digital converter (ADC). The 12-bit ADC has a maximum

Table 4.2. *Analog input values*

		DUT A.		DUT B.	
U_in	ADC value	MCU1 (F2)	MCU2	MCU1 (F4)	MCU2
1 V	AIN11	415	102	414	102
	AIN21	412	102	413	102
9 V	AIN11	3699	925	3727	931
	AIN21	3699	924	3728	931

value of

$$2^{12} = 4096, \tag{4.1}$$

and 10-bit ADC has a maximum value of

$$2^{10} = 1024. \tag{4.2}$$

These values represent the *10 V* digital equivalent. In the measurements, external *1 and 9 V* voltages were used for the two analog input pins *AIN11* and *AIN21.*

The analog values can be read using the command *adcread*, which prints relevant ADC channel values into the serial communication console. The two analog input pin values with the measurement voltages are read by both MCU1 and MCU2. The results are documented in the table 4.2.

Each measure was repeated three times to check variance. The ADC results were stable. A more significant error source is the DC voltage source and the voltage meter[26]. The ones in use had only one decimal digit so the ADC accuracy in all MCUs seems correct, according to table 4.2.

The results between F2 and F4 vary just

$$\frac{3727 - 3699}{4096} * 100\% = 0.7\%,$$

which is an acceptable accuracy at this stage of testing, so no further measurements are done.

In addition to analog inputs, the same *adcread* command also displays internal voltages and temperature sensor values.

4.3 Case example - USART

There are multiple peripherals, which will most probably need some reconfiguration when porting into a new system. The configurations may be as simple as recalculating or setting a new operating frequency.

In this thesis, USART was chosen as the peripheral to be reviewed. When first trying to boot the user application, the program was discovered to be stuck into a reboot loop and not proceeding to complete the initialization. The specific location of the error was yet unknown. The most probable cause was the watchdog timer, which resets the controller if the program gets stuck.

The communication peripheral, USART, depends on baud rate and voltage levels, which can, fortunately, remain the same when accomplishing the porting inside such similar MCUs. The parameters are defined in the source code's configuration files, which worked for both controllers in this case.

The investigation of the location of the error was begun by first initializing the serial communication and setting debug prints in the *main.c* source file. There are multiple levels of serial communication, abstracted away from hardware. First, a single character was printed, then 6 digits, and finally a whole string. When debug prints for strings worked, the debug prints were inserted both before and after the source code's initialization phases. The outcome can be seen in the picture 4.4

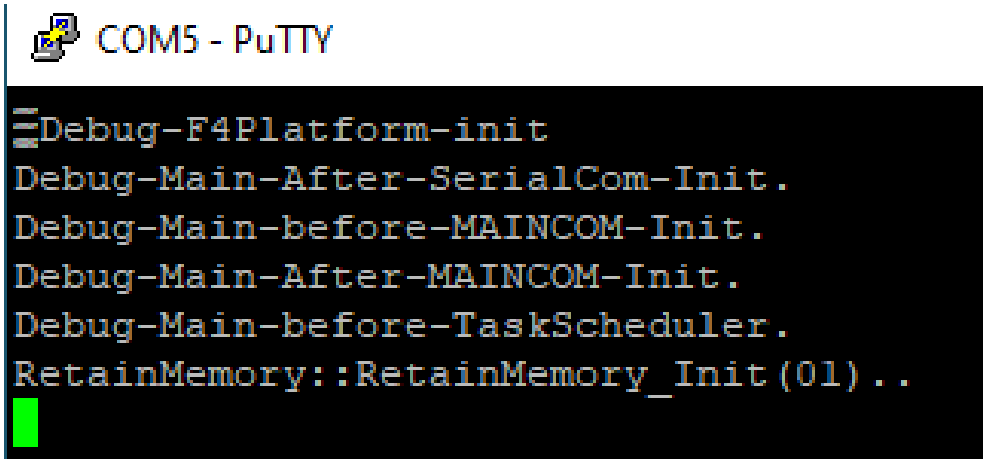

Figure 4.4. *Debug prints via serial communication path read by PC help sourcing the errors.*

During this, it was discovered that the memory initialization gets stuck and a candidate for the error location was found. The error could also originate from the task scheduler. This would require further analysis. Testing the memory could be isolated by disabling most of the other features, for example, the watchdog timer. The hardware test application could be used again in a more precise way to confirm that the reading and writing the EEPROM works correctly. However, this work is out of the scope of this thesis.

5. CONCLUSION

The main catch of this thesis is that foresight and a systematic approach are the keys to success in unpredictable migration processes, which nonetheless usually consist of trial and error.

The inevitable change of industrial environments is generating more demanding software requirements for the control units. Eventually, the hardware (e.g. processor core) might become a limiting factor and needs an update. Then within the updated hardware environment, the product owner is left with two options regarding software: redesign or reuse.

Reusing code might be significantly more cost-effective than a redesign if portability has been considered during the software development process. It should have been, as portability is one of the six factors listed in *Software Quality ISO Standard 9126*.

There are an indefinite amount of different migration scenarios due to the number of different processor pairings, source code, and environments. Therefore there is no standard method for porting an application into a new microcontroller unit. This emphasizes the importance of thorough planning of the processor selection and the migration process, such as the one presented in figure 3.1.

REFERENCES

[1] Cravotta, R. Fear and loathing of porting embedded software. *EDN* 47.17 (2002). ISSN: 00127515.

[2] Techopedia. *Porting definition*. 2011. URL: https://www.techopedia.com/definition/8925/porting (visited on 2021).

[3] Mooney, J. Bringing portability to the software process. *Dept. of Statistics and Comp. Sci., West Virginia Univ., Morgantown WV* (1997), pp. 559–570. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5961733.

[4] Garen, K. Software portability: weighing options, making choices. eng. *The CPA journal (1975)* 77.11 (2007), p. 10. ISSN: 0732-8435.

[5] Palkonen, J. Teksti-TV-sovelluksen muokkaaminen MIDP:stä Androidille. Master of Science Thesis. Tampere University of Technology, 2011.

[6] Silakov, D. V. and Khoroshilov, A. V. Ensuring portability of software. *Programming and Computer Software* 37.1 (2011), pp. 41–47. ISSN: 03617688. DOI: 10.1134/S0361768811010051.

[7] ST Microelectronics. AN3364 Migration and compatibility guidelines for STM32 microcontroller applications. May (2012). Doc ID 018608 Rev 3, pp. 1–17.

[8] Mikkonen, T. and Järvinen, H.-M. *Sulautettu ohjelmointi*. Tampere University of Technology, 2010, pp. 1–240.

[9] Stephen, C. *The 2016 Top Programming Languages*. 2016. URL: https://spectrum.ieee.org/the-2016-top-programming-languages.

[10] ST Microelectronics. PM0214, STM32 Cortex®-M4 MCUs and MPUs programming manual. March (2020).

[11] ARM Ltd. and Yiu, J. *The Definitive Guide to ARM® CORTEX®-M3 and CORTEX®-M4 Processors*. Third Edit. Cambridge, UK, 2014. ISBN: 978-0-12-408082-9. DOI: 10.1016/c2012-0-01372-5.

[12] *STM32F207xx - Arm®-based 32-bit MCU*. DS6329. Datasheet, Rev 18. ST Microelectronics. 2020. URL: http://www.st.com.

[13] *STM32F407xx - Arm® Cortex®-M4 32b MCU+FPU*. DS8626. Datasheet, Rev 9. ST Microelectronics. 2020. URL: http://www.st.com.

[14] Güven, Y., Coşgun, E., Kocaoğlu, S., Gezici, H. and Yilmazlar, E. Understanding the Concept of Microcontroller Based Systems To Choose The Best Hardware For Applications. *Research Inventy: International Journal of Engineering And Science* 7.December (2017), p. 38. ISSN: (e) 2278-4721. URL: www.researchinventy.com.

[15] Fleming, B. Microcontroller units in automobiles. *IEEE Vehicular Technology Magazine* 6.3 (2011), pp. 4–8. ISSN: 15566072. DOI: 10.1109/MVT.2011.941888.

[16] White, E. *Making Embedded Systems*. eng. Sebastopol: O'Reilly Media, Incorporated, 2011. ISBN: 1449302149.

[17] Konecranes Global Corporation. *Service instructions, Condition Monitoring Unit MASTER PRO*. 2021.

[18] Konecranes Global Corporation. *Service Instructions, GATEWAY*. 2020.

[19] *Reference manual*. RM0090 Rev 19. ST Microelectronics. 2013. URL: http://www.st.com.

[20] Kanellopoulos, Y., Antonellis, P., Antoniou, D., Makris, C., Theodoridis, E., Tjortjis, C. and Tsirakis, N. Code Quality Evaluation Methodology Using The ISO/IEC 9126 Standard. eng. *International Journal of Software Engineering & Applications* 1.3 (2010), pp. 17–36. ISSN: 0976-2221. DOI: 10.5121/ijsea.2010.1302.

[21] Lehtonen, T., Tuomivaara, S., Rantala, V., Känsälä, M., Mäkilä, T., Jokela, T., Könnölä, K., Kaisti, M., Suomi, S., Isomäki, M. and Ylitolva, M. *Sulautettujen järjestelmien ketterä käsikirja*. Turku: Turun yliopisto, Työterveyslaitos, TEKES, 2014. ISBN: 978-951-29-5838-2. URL: http://embedded.utu.fi/kasikirja.

[22] Konecranes, P. L. C. ECU prototype 1: Production test specification. (2019).

[23] Salman, S. M., Papadopoulos, A. V., Mubeen, S. and Nolte, T. A systematic methodology to migrate complex real-time software systems to multi-core platforms. *Journal of Systems Architecture* 117.February (2021), p. 102087. ISSN: 13837621. DOI: 10.1016/j.sysarc.2021.102087. URL: https://doi.org/10.1016/j.sysarc.2021.102087.

[24] Mäkinen, P. A. *SFS 6002 käytännössä*. fin. Espoo, 2010.

[25] Tiiro, J. Uusi sähkötyöturvallisuusstandardi SFS 6002. *Metropolia Ammattikorkeakoulu* (2015).

[26] Kularatna Nihal. Waveform Parameters, Multimeters and Pulse Techniques. eng. *Digital and Analogue Instrumentation - Testing and Measurement*. Institution of Engineering and Technology, 2003, pp. 1–1. ISBN: 0852969996.

APPENDIX A: STM32 SERIES PERIPHERAL COMPARISON SHEET

Table A.1. *STM32 series feature comparison. [7]*

Peripheral		STM32 series	
		F2	F4
Core		Cortex-M3 r2p0 w/ MPU	Cortex-M4 r0p1 w/ FPU and MPU
FLASH memory		1024 KB	1024 KB
SRAM		128 KB	128 + 64 KB
FSMC		Yes	Identical
Max CPU Frequency		120 MHz	168 MHz
Operating Voltage		1.8 to 3.6 V	Identical
Bootloader		USART1/3 CAN2 DFU	Identical
Timers	Advanced 4 channel (3 cmpl. out)	2	Identical
	General purpose	up to 10	Identical
	4 channels	up to 4	Identical
	2 channels	up to 2	Identical

	1 channel	up to 4	Identical
RTC		Hardware calendar	Hardware calendar + subseconds
COMs	SPI(I2S)	3(2)	
	TI mode	Yes	Identical
	Max freq.	up to 15 or 30 Mbits/s	up to 37.5 Mbits/s
	Audio freq.	8 kHz up to 192 kHz	Identical
	I2C	3	2
	Max freq.	400 kHz	Identical
	USART	4	Identical
	UART	2	2
	Max freq.	3.75 or 7.5 Mbits/s	10.5 Mbits/s
	USB	OTG FS OTG HS	Identical
	CAN	2	Identical
	SDIO	1	Identical
	CEC	No	Identical
	Ethernet MAC	Yes w/ HW IEEE1588 v2	Identical
GPIOs		140	up to 140
12-bit ADC		3	3
Max sampling freq.		2 MSPS	2.4 MSPS
Number of channels		16/24	up to 24
12-bit DAC		2	Identical
Max sampling freq.		1 MS/s	Identical
Number of channels		2	Identical
Comparators		No	Identical
LCD (COM x SEG)		No	Identical
DCMI		Yes	Identical

Crypto / hash processor	Yes DES, 3DES, AES 256-bit, Shaa-1, MD5, HMAC	Identical
RNG	Yes	Identical
Package	LQFP144	Identical