



DEGREE PROJECT IN ELECTRICAL ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2021

Cost-efficient method for lifetime extension of interconnected computer-based systems

WILHELM HOLMBERG

Cost-efficient method for lifetime extension of interconnected computer-based systems

WILHELM HOLMBERG

Degree Programme in Electrical Engineering
Date: June 23, 2021

Supervisors: Leandro Lopez, Niclas Andersson
Examiner: Mats Bengtsson

School of Electrical Engineering and Computer Science

Host organization: Region Stockholm

Swedish title: Kostnadseffektiv metod för livstidsförlängning av
sammanlänkade datorbaserade system

Abstract

Lifetime and obsolescence of components for computer-based systems poses issues for continued usage and maintenance of the systems. This thesis investigates possible alternatives for lifetime extension of a train identification system used in Stockholm Metro. Research of other train identification systems available on the market were made to enable a cost comparison between lifetime extension and system replacement. Methods for extending lifetime of computer-based system, where components are obsolete, were investigated. Since most system documentation was inaccessible a reverse-engineering approach was chosen. Through usage of electrical schematics acquired and open-source hardware descriptions a hardware emulator was developed, which is directly compatible with the existing hardware. The total amount of resources used indicates it is possible to extend the systems lifetime at a low cost, as compared to the cost of system replacement.

Keywords

COTS

Cost-efficiency

Emulator

FPGA

Lifetime extension

Train identification

Sammanfattning

Livslängd och åldrande av komponenter för datorbaserade system utgör problem för fortsatt användande och underhåll av systemen. Den här avhandlingen undersöker möjliga alternativ för livstidsförlängning av ett tågidentifieringssystem som används i Stockholms tunnelbana. Efterforskningar av andra tågidentifieringssystem tillgängliga på marknaden genomfördes för att möjliggöra en kostnadsjämförelse mellan livstidsförlängning och systemutbyte. Metoder för förlängning av livslängd av datorbaserade system, där komponenter är föråldrade, undersöktes. Då stora delar av systemdokumentationen inte var tillgänglig valdes baklängesutveckling som strategi. Genom användande av förvärvade elschema och öppen-källkod hårdvarubeskrivningar kunde en hårdvaruemulator utvecklas, vilken är direkt kompatibel med befintlig hårdvara. Den totala resursanvändningen indikerar att det är möjligt att förlänga systemets livslängd till en låg kostnad, jämfört med kostnaden för ett systembyte.

Nyckelord

COTS

Emulering

FPGA

Kostnadseffektivitet

Livstidsförlängning

Tågidentifiering

Acknowledgments

I would like to thank Leandro Lopez, my supervisor, for the guidance, support, and many good suggestions during my thesis project. I would like to thank my examiner, Mats Bengtsson, for the active participation and making this project possible. Finally, I want to thank Trafikförvaltningen at Region Stockholm and my supervisor there, Niclas Andersson, who has trusted me with this project.

Stockholm, June 2021

Wilhelm Holmberg

Contents

1	Introduction	1
1.1	Background	1
1.2	Problem	1
1.3	Purpose	2
1.4	Goals	3
1.5	Project procedure	3
1.6	Delimitations	3
1.7	Structure of the thesis	4
2	Background	5
2.1	Stockholm's metro system	5
2.1.1	Train identification systems	7
2.2	Emulation techniques and hardware replacement	12
2.2.1	Hardware Description Language	13
2.2.2	Software emulator	14
2.2.3	Hardware replacement strategies	15
3	Introduction to IDTS	17
3.1	Status of parts	18
3.2	Replacement of Inductive Data Transmission System (IDTS) on-board equipment	18
3.3	Suggested improvements	18
3.4	Maintaining train identification	20
3.4.1	System documentation	21
3.5	Test environment	21
3.6	IDTS components	23
3.7	Reverse-engineering of IDTS	24
3.7.1	MCU-module	24

4	Development of replacement hardware for IDTS	27
4.1	Emulator requirements	27
4.2	Software emulator	28
4.2.1	Code development and structuring	28
4.2.2	Memory content	29
4.2.3	Implementation	30
4.3	Hardware emulator	31
4.3.1	Memory content	31
4.3.2	Implementation	33
5	Testing and Verification	35
6	Discussion	39
6.1	Hardware emulation as replacement strategy	40
6.2	Reliability	46
7	Conclusions and Future work	47
7.1	Conclusions	47
7.2	Limitations	47
7.3	Future work	48
7.3.1	Further development of parts for IDTS	48
7.3.2	Further explorations in hardware replacement strategies	48
	References	49
A	Legality of reverse-engineering	53
B	Swedish - English metro dictionary	54

List of Figures

2.1	Indication of destination, from a route selector panel, on the interlocking control panel	6
2.2	Typical wayside installation	7
2.3	Train operator input window	8
2.4	Communication hierarchy of systems surrounding IDTS	9
3.1	Communication hierarchy inside a wayside computer	17
3.2	Communication hierarchy of an improved wayside computer	19
3.3	Different options for maintaining train identification	20
3.4	Illustration of black box behavior	20
3.5	Test environment setup	22
3.6	Internal connections of a master wayside computer	25
3.7	MCU-module block diagram	26
4.1	Remaining options for maintaining IDTS	28
4.2	Example content of hex-files after import to spreadsheet	30
4.3	Example content of hex-files after separation of data	30
4.4	Hardware emulator architecture	33
6.1	General approach of dealing with outdated interconnected computer systems	41

List of Tables

3.1	IDTS wayside configuration	23
4.1	Instruction execution time deviations	32
5.1	Time consumption for Microcontroller Unit (MCU)-module software emulator design	35
5.2	Time consumption for engineering new MCU-module	36
5.3	Cost of components used for the hardware emulator	37
6.1	Modules needed during expansion of Stockholm Metro	40
6.2	Steps necessary to develop replacement hardware for IDTS	42
6.3	Estimated total development costs of IDTS replacement hardware	43
6.4	Estimated hardware cost of system replacement using infsofts' solution	43
6.5	Estimated cost of wayside equipment removal and installation using infsofts' solution	45
6.6	Comparison of costs to secure train identification in Stockholm Metro	45

Listings

2.1	Software emulation of Intel 8085 MOV r1 r2 instruction . . .	15
4.1	Example content of hex-files	29

List of acronyms and abbreviations

ASIC Application Specific Integrated Circuit

COTS Commercial Off-The-Shelf

CPU Central Processing Unit

DPS Display and Public address System

FPGA Field-Programmable Gate-Array

HDL Hardware Description Language

HLS High-level synthesis

IDTS Inductive Data Transmission System

MCU Microcontroller Unit

PCB Printed Circuit Board

PROM Programmable Read-Only Memory

RAM Random Access Memory

RDTS Radio Data Transmission System

RFID Radio-Frequency Identification

ROM Read-Only Memory

RTOS Real-Time Operating System

TIS Traffic Information System

UART Universal Asynchronous Receiver-Transmitter

VM Virtual Machine

Chapter 1

Introduction

With developments of computer-based systems always occurring, and existing systems being increasingly expensive to maintain, replacement of outdated systems will almost always occur. Such replacements are usually connected to expensive installation costs, and for it to be cost-efficient it is desirable to have a long lifetime. This thesis is a case study of different strategies for lifetime extension of obsolete computer-based systems. The system used for the case study is a train identification system, named Inductive Data Transmission System (**IDTS**), which is developed for and used in the Stockholm Metro.

1.1 Background

Computer-based systems consist of hardware and software, where the hardware at some point of time may have a failure. When the system is invented, it is likely that spare parts are available, but as development continues in the hardware field, it is very likely that components previously used will become obsolete. This will finally lead down to two choices, develop new hardware, which is compatible with the existing system or initialize replacement of the system with a modern equivalent. The choice is not obvious, there are many variables to take into consideration. Such variables could be benefits of a new system compared to the current, and if the cost of system replacement is higher than the reengineering cost per lifetime increase.

1.2 Problem

Spare parts for the **IDTS** are decreasing while the demand for new parts is increasing. Each station has a wayside computer installed for identification of

trains, and each computer is connected to a central system. Stockholm Metro is currently expanding, a total of 12 new stations are under construction[1, 2, 3, 4, 5], this raises needs for parts which can be used to build up wayside computers at new stations. IDTS has been in operation since the late 1980s and components are obsolete. A replacement of the traffic management system is estimated to be installed sometime between year 2035 and 2040[6]. A new traffic management system would replace multiple of the current systems used today, among them is IDTS. Therefore, lifetime of IDTS must be increased, which raises the question; what is the most cost-efficient way to extend lifetime of an outdated interconnected computer system?

1.3 Purpose

The purpose of this case study is to determine what is the most cost-efficient way to extend lifetime of a computer system with obsolete hardware, and if the cost is motivated compared to the cost of a system replacement.

The Public transport administration at Region Stockholm will have a direct benefit of the outcome of this thesis, since their train identification system is obsolete and has almost no spare parts available. IDTS has similarities with railway signaling systems and the results from this case study could be useful for organizations still relying on the same hardware as IDTS. However, IDTS is not safety-critical and therefore safety-analysis should be carried out before applying the results for safety-critical applications such as railway signaling systems. IDTS is a soft real-time system and the learnings from this project could hopefully be applied for other real-time systems with obsolete hardware.

In a wider perspective this thesis could help other companies or organizations, which are relying on outdated computer-based systems, to make a choice how to deal with system maintenance, lifetime extension and system replacement. Complete system replacement usually requires all devices being replaced simultaneously, since systems usually are not compatible, and therefore some fully functional hardware is replaced in the process. Having a cost-efficient method for hardware replacement could therefore have positive impacts on sustainability, enabling for replacement of only faulty hardware and not requiring complete system replacement.

However, intellectual property aspects must be considered since the systems hardware and software has been developed and delivered by an external party.

1.4 Goals

The goal of this project is to find a cost-efficient method for replacement of obsolete hardware in a computer-based system, and is divided into three sub-goals:

1. Find possible replacement system available on the market which could replace **IDTS**.
2. Gain knowledge about different replacement strategies for obsolete hardware and compare them.
3. Use at least one of the replacement strategies to implement a replacement module as a proof of concept.

Testing of replacement module(s) will be carried out in a laboratory environment. Testing in a live environment can be done if the laboratory test results are satisfactory and approved by the external supervisor.

1.5 Project procedure

IDTS will be used as a reference system for this project. First, an investigation about which replacement systems are available on the market will be carried out, to set a baseline for what the cost and benefits of a replacement system are. Reengineering of the current system's wayside computer, which consists of multiple modules, will be carried out for one module. One solution will be design of a completely compatible computer-unit, which is built from Commercial Off-The-Shelf (**COTS**) hardware. The other design should be made using hardware descriptive language and programmable logic. One of the two, or a combination of both, hardware designs should be implemented to verify its functionality and proof of concept. Hardware and software costs, combined with engineering hours spent, will be used to compare all three solutions.

1.6 Delimitations

Possible replacement systems for **IDTS** available on market will be investigated, but not all of them. An exact cost of installation is outside the scope of this project. Cost estimation of installing a replacement system on rolling stock will be investigated, but replacement modules designed should only consider

wayside equipment and be directly compatible with surrounding systems. Replacement hardware shall be possible to install on one station and not require any other changes. Development costs for customization of possible replacement systems will not be covered, comparison will be between system replacement and development of replacement modules.

1.7 Structure of the thesis

Chapter 2 describes concepts and systems involved in this thesis. A deeper introduction to IDTS is present in Chapter 3, followed by a description of replacement hardware development in Chapter 4. Chapter 5 contains a summary of the results. Discussion, conclusions, and future work are presented in Chapter 6 and 7.

Chapter 2

Background

In this chapter a brief introduction to Stockholm Metro will be given, to describe the purpose of **IDTS** together with some nomenclature used by personnel and in this report. Further, different train identification systems will be presented where **IDTS** will be described in a bit more detail. Different replacement strategies and technologies will be described, together with previous work in this area.

2.1 Stockholm's metro system

Stockholm Metro was opened in 1950 between the stations Slussen and Hökarängen[7, p. 9]. The system has since expanded in varying pace, where the last expansion was finished in 1994[8]. Today the metro system is divided into three main lines, commonly referred to as the *Green*, *Red*, and *Blue* line. Stockholm Metro is owned by Region Stockholm while maintenance and operations are contracted to private companies.

Originally all three main lines used the same railway signaling system[7, p. 140], a relay based electronic entrance-exit interlocking system[7, p. 145]. The system has no capabilities of determining which train that is currently occupying the track circuit, neither is there any support for automatic train routing. The rail control center can therefore only see if a train is present, but not which train it is or where it is heading. At the station before a major junction route selector panels were installed, and the train operator had to push the button corresponding to the desired destination of the train. This signaled a lamp at the interlocking control panel, indicating to the switchman which route the train should take. Fig. 2.1 displays a section of the interlocking control panel for the Blue line. Triangles represent the color light signals along

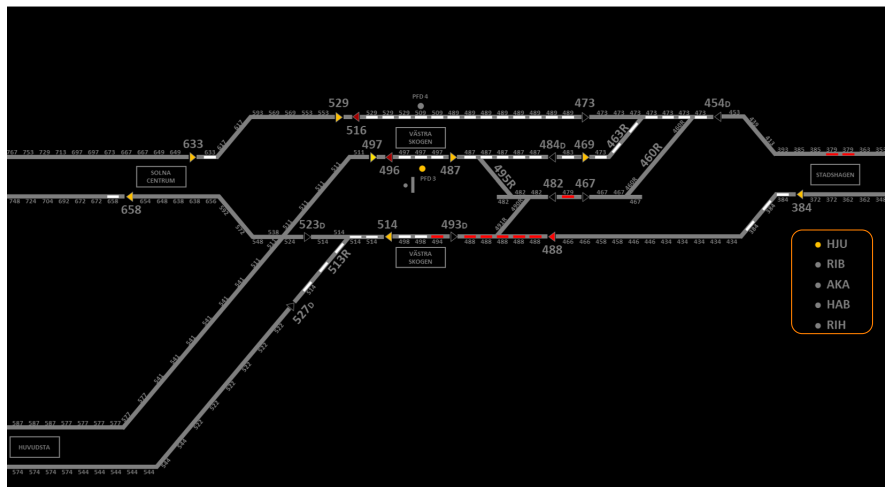


Figure 2.1: Indication of destination, from a route selector panel, on the interlocking control panel

the track. Track circuits are numbered with small font and light red when occupied by a train. In Fig. 2.1 one train is approaching Västra skogen while another train is waiting at the siding and a third train is at Stadshagen. The train movements can be monitored but there is no identification available on the panel. A route selector panel is present at Stadshagen, and the latest requested route is displayed on the interlocking panel below Stadshagen station (circled in orange in Fig. 2.1). At most minor junctions there were no route selector panels, and the switchman had to rely on the timetable and communication with the dispatcher.

A train identification system was installed in the late 1980s. The system was **IDTS** and was firstly used to identify the destination of trains entering and exiting stations, allowing electronic departure boards on each station to display the destination to passengers. **IDTS** was later connected to a network, allowing the rail control center to monitor for example destination and vehicle cycle of all trains at the line.

In the 1990s a new computer-based railway signaling system was installed on the Green line[6, p. 9]. One of its many features is the possibility of automatic train routing to the most common destinations, allowing the switchman to focus on other tasks. This was enabled through the destination information gathered from **IDTS**, and the old route selector panels were removed. Route selector panels on the other two lines remained and are still operational as redundancy in case of malfunctions in the train identification

system. There is no redundancies for the Green line, the switchman must rely on the timetable if IDTS malfunctions. Therefore, traffic can remain operational without a train identification system, but it increases workload on the two switchmen tremendously. At peak hours there are over 50 trains running on the Green line, with only two minutes apart. Furthermore, when IDTS malfunctions there is no real-time departure information available for passengers.

Rolling stock used for daily operations are subway-car types C6, C14, C15, C20 and C30. C30-stock is currently under deliverance. In total 96 C30-cars have been ordered and will replace all the C6, C14 and C15-cars[9]. The C20-stock consists of 271 cars[10], resulting in a total of 367 cars being used for daily operations within a couple of years. While the Green line only has C20-stock for daily operations all stock types are being used on the Red line and the Blue line uses C14, C15 and C20-stock.

2.1.1 Train identification systems

Inductive Data Transmission System

IDTS is the system currently used for identification of metro trains in Stockholm. The system was developed by SEL Canada in the 1980s. Inductive loops are placed at each station, and each depot, entry and exit. The inductive loops serve as antennas for wayside equipment and are connected to a wayside computer. The wayside equipment continuously polls for data from the inductive loops[11, p. 10]. The train operator inputs destination, route number, vehicle cycle and number of cars at a panel in the operator cab. Fig. 2.3 shows an example of input data to the IDTS on-board computer. This data combined with car ID, which is preset during installation of on-board equipment, are transmitted to the wayside equipment when an inductive loop is detected. Fig. 2.2 shows an example of a typical wayside installation at a station with one track in each direction and no switches between the tracks.

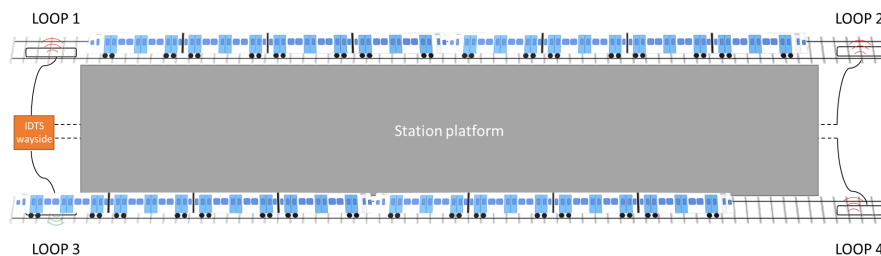


Figure 2.2: Typical wayside installation

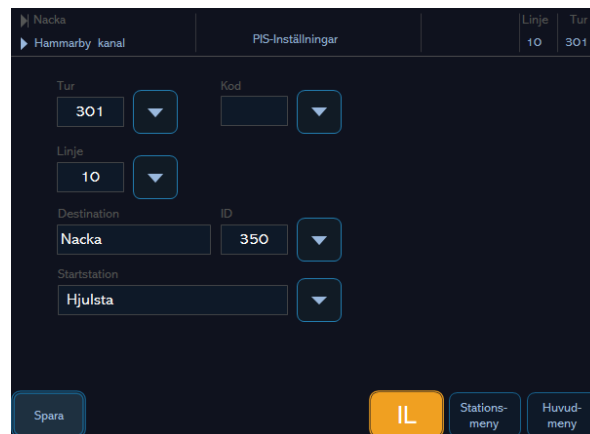


Figure 2.3: Train operator input window

The wayside computer is connected, via a serial modem, to a central system called Traffic Information System (**TIS**). **TIS** polls the wayside computers for information approximately two times per second. **TIS** forwards information about train position and destination to a real-time passenger information system called PubTrans. PubTrans processes the information and decides what information should be shown on the electronic departure boards at each station. A Display and Public address System (**DPS**)-module is placed at each station, which receives information from PubTrans and forwards it to the electronic departure boards. Fig. 2.4 contains a communication diagram. Originally **IDTS** and **TIS** managed all this information. **IDTS** wayside computers were then handling information to the electronic departure boards and had a direct connection to those. Today, this functionality of **IDTS** is no longer used, and the direct connection to the electronic departure boards is removed.

IDTS information from wayside computer to **TIS** is also used by rail control center personnel when routing trains, checking car IDs, or checking the position of a specific vehicle cycle.

Radio Data Transmission System

Radio Data Transmission System (**RDTS**) is a successor to **IDTS**, developed specially for Stockholm Metro. As the name suggests, data from the operator input panel and car ID are instead transmitted to wayside-equipment using radio. However, the information transmitted is only valuable if it can be determined which train on the line is sending it. While **IDTS** can position each train by its short transmission range, and therefore determine that the train must be on a specific track section, **RDTS** must use some other technique

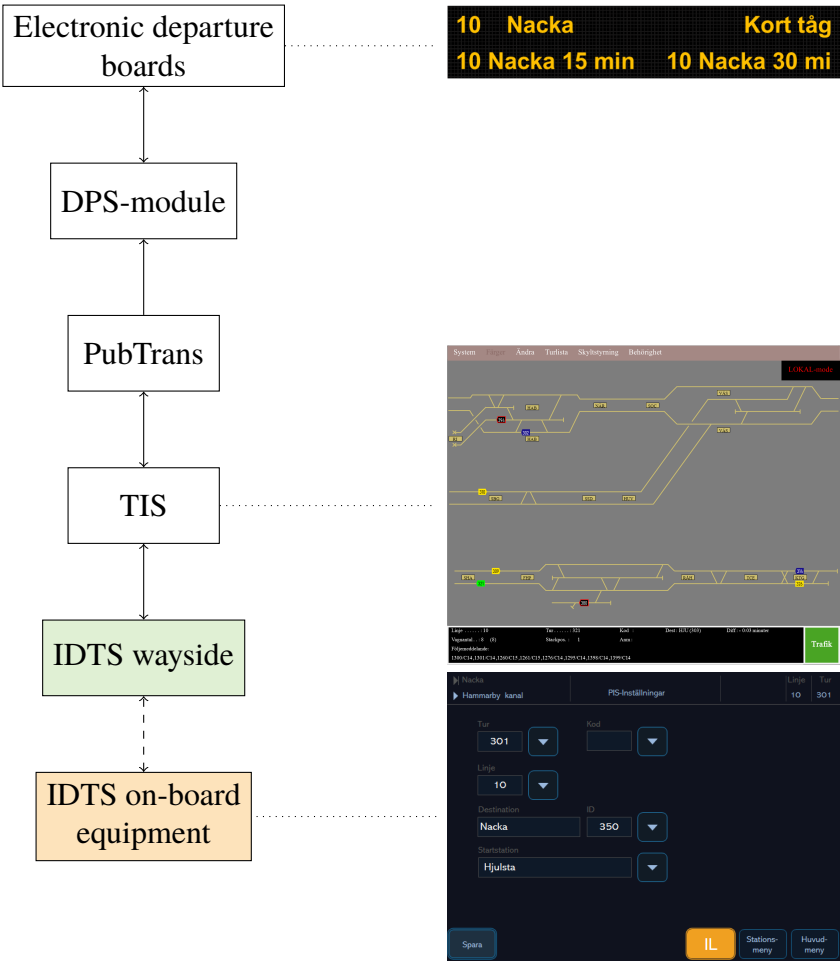


Figure 2.4: Communication hierarchy of systems surrounding IDTS

to determine the trains position. The Green line's signaling system provides a solution.

The railway signaling system on all three lines has a continuous train supervision. The track is divided into sections, track circuits. Each track circuit sends out an alternating current, to avoid disturbances from the direct current used for propulsion. A train axle will close the circuit and a current will flow through the axle. The current creates a magnetic field around each rail, which is read by antennas mounted in front of the first axle. Information can therefore be transmitted regardless of the trains exact position on the section, hence continuous.

On the Green line each track circuit transmits a telegram with information about the section, such as length and an ID of the section. Track circuit length and ID combined provides a unique position in the network. This information is therefore added to the radio telegram sent to wayside. A central computer decodes the information and identifies both the trains precise position, car ID, destination, and all other information which **IDTS** would have transmitted. Since **RDTs** do not rely on fixed data transmission points any change of information, such as destination, can be instantly transmitted to wayside.

The other two lines do not have such sophisticated signaling system. The alternating track current has a pulse coded 75 Hz carrier frequency, where the pulses depend on how far it is to the next occupied section or a stop signal[12]. The codes are decoded as *H*, *M* and *L* and indicates the maximum allowed speed. Compared to the Green line's signaling system there is no telegrams containing any data which can be used to determine the trains position. **RDTs** is therefore not an option for the other two lines without any changes to the infrastructure. However, **RDTs** could be used along the Green line, which currently has over 50 out of the total 117 wayside computers for **IDTS**. If **RDTs** is fully implemented on the Green line wayside equipment for **IDTS** could be used for the other two lines. Final testing of **RDTs** must be carried out, which requires external visitors to the rail control center. Due to the on-going Covid-19 pandemic such tests have been postponed for minimizing risks of key personnel contagion.

Development of **RDTs** has required an update in all C20-stock trains, including new software in one of the on-board computers developed by Bombardier, new software in the radio-panel developed by LS Elektronik and new software in the on-board signaling computer developed by Siemens. The changes are only implemented in C20-stock trains, since no other train series are used for daily operations at the Green line.

Currently the **RDTs**-project has costed approximately 23 million SEK[13].

RDTS was estimated to cost 20 million SEK[13] and was considered as a cost-efficient replacement since it only required a central wayside equipment[14].

SoftPrio

Apart from Stockholm Metro the Public transportation administration in Region Stockholm is also responsible for the local railways *Saltsjöbanan* and *Roslagsbanan*, and the tramways *Lidingöbanan*, *Nockebybanan*, *Spårväg city* and *Tvärbanan*. Installations of a system called SoftPrio is carried out at *Tvärbanan*. SoftPrio is a system for positioning of trams combined with maneuvering of switches from inside the tram operator cab. The system has been developed by and for the traffic office in Gothenburg, which provides the system software and knowledge on a license basis. SoftPrio relies on radio and Radio-Frequency Identification (RFID)-tags for vehicle positioning. Along the track passive RFID-tags are placed underground, where each tag has a unique set of information. An antenna installed under the tram reads the tag and transmits it to the wayside system, which will take the appropriate action depending on the information stored on the tag. This allows for control of switches combined with positioning of trams.

SoftPrio is not directly compatible with everything used in Stockholm Metro, but the techniques could probably be adapted. If the system could be installed as a light version, where only positioning is preserved and minor changes are made to suit the needs for the metro, it could be advantageous to have similar systems. However, if there are needs for major reengineering to fit the metro it is probably better to develop a new system based on the same idea for positioning.

Cactus TMS

Saltsjöbanan, *Roslagsbanan*, *Lidingöbanan* and *Tvärbanan* currently uses Cactus TMS (Traffic Management System)[15]. Cactus TMS can be customized for the application[16], and the information in this section covers how it is implemented in Stockholm. It should be noted that Cactus TMS is a traffic management system which includes control of switches, signals, occupied track sections etc. This description will only cover how the identification of trains are carried out.

Identification of trains in Cactus TMS are carried out by assigning each occupied track section with an ID. When the train moves along the track and occupies a new track section the ID will be carried to the new track section occupation. Using information from the timetable Cactus TMS can determine

the destination for each train. At *Saltsjöbanan* and *Roslagsbanan*, which are railways, the trains have a specific train number for each journey. When the train reaches its destination, it will get a new number. Such changes of numbers are available in the timetable and Cactus TMS can therefore determine and update the train number automatically when the train reaches its destination. The tramways, and the metro, uses a vehicle cycle number as ID and only the destination is updated at terminus. *Tvärbanan* runs both in mixed traffic and on reserved tracks. Track circuits are present at reserved tracks, while in mixed traffic no track circuits are available. Therefore, passive **RFID** tags are installed on each car. Active antennas are installed close to stops, allowing for positioning of trams along track sections without track circuits.

There is no communication between the vehicles and Cactus TMS. In case of a traffic disruption, causing a train to change destination, the train operator must change the announced destination on the train and the rail control center personnel must update in Cactus TMS to get correct announcement of destination at electronic departure boards[15].

Railway Train Tracking System by infsoft

Another technique which can be used for train positioning is Bluetooth, which *infsoft* uses for their *Railway Train Tracking System*. Bluetooth Low Energy beacons are installed in tunnels and at stations. Cars are equipped with locator nodes which receives the beacon signal and transmits data to a central monitoring system[17]. Locator nodes has a unit price of 130 €[18]. Bluetooth Low Energy beacons can be bought from various suppliers[19]. One battery powered beacon found, which is waterproof (IP67), withstands a 10-ton impact and has a 5-year battery time is listed at 25.99 £ excluding value added tax[20].

2.2 Emulation techniques and hardware replacement

Replication of a specific hardware behavior is a technique called emulation. Emulation of hardware could be done using software or hardware. Hardware emulation could be done using Hardware Description Language (**HDL**) combined with a Field-Programmable Gate-Array (**FPGA**), which allows hardware to directly act as the original system architecture. It is not necessary that the internal architecture is identical to the original system, provided that the

emulator can run the original software and produce the same results. Software emulation is instead done by writing a data structure and behavior which matches the original system.

A Virtual Machine (VM) is, depending on the definition, a special case of an emulator. Previously a VM has been defined as a duplicated, isolated, version of the original hardware on which the machine is running[21]. Today, VMs does not necessarily have any correlations with real hardware at all. A computer architecture is written in software, similar to a software emulator, allowing instructions to run on top of real hardware. For example, Java Virtual Machine is a VM which purpose is to act as a hardware independent platform for which software can be written without having to consider the underlying hardware on which it is running. The difference between emulation and virtualization is that emulation replicates a specific behavior while virtualization uses a physical device with the correct behavior[22].

Emulators has been subject for multiple legal cases, where for example console manufacturers have tried to bring a stop to emulators which mimics the behavior of their consoles. Swedish law states that *a person who has the rights to use a computer software is also allowed to observe, examine, or test the software's function in order to establish the principles behind the software's different details*[23, 26g§]. Furthermore, it is allowed to reproduce or translate computer software in order to achieve compatibility between different software's[23, 26h§]. Creating an emulator from pure observations of software behavior could therefore be considered legal in Sweden. Appendix A contains further information about legality of reverse-engineering.

2.2.1 Hardware Description Language

HDL is a way of expressing digital logic in a programmatic fashion. It can be used for design testing and verification, either by simulation using computer software or programming onto a FPGA. The two major HDLs are VHDL and Verilog[24, p. 1], both developed in the 1980s[24, p. 2][25, p. 1]. HDL enables possibilities of expressing a computer architecture similar to a software. A design can be simulated and tested in software to verify its function[24, p. 2]. The hardware description can then be used to program a FPGA or designing an Application Specific Integrated Circuit (ASIC). Usage of HDL could therefore be a viable method for replacement of obsolete hardware. A designer could write a hardware description which matches the behavior of the obsolete hardware if the existing hardware functionality is known. Since HDL is generic there is no strict hardware dependencies. The only dependencies would be the need of correct timing and propagation times, together with

appropriate inputs and outputs. Due to the weak hardware dependencies, it should not be a problem to change hardware, if the **FPGA** used in the original design becomes obsolete, to a newer version which could be programmed to have an identical hardware behavior. System hardware could then be produced or ordered on a demand basis instead of a speculative basis, since the risks of running out of spare parts are lower than in the case of strong hardware dependencies.

Usage of **FPGA** has been adopted by *Scheidt & Bachmann* to deal with issues connected to obsolete hardware components[26, p. 18].

Development of replacement hardware for an existing system design using **HDL** could be a time-demanding process for someone who is not familiar with **HDL**. High-level synthesis (**HLS**) tools translates high-level languages, such as C and C++, to **HDL**. Such tool allows a high-level software architecture being realized in hardware.

2.2.2 Software emulator

Software designed for one system can run on a non-compatible host system using software emulation. The emulator is a piece of software which provides a translation layer between the original software and the non-compatible hardware. Machine code, which is strongly dependent on hardware, can be decoded using software. For example, the Intel 8085 instruction *MOV r1, r2* moves the 8-bit data from *r2* to *r1*[27, p. 5-4]. An example of how this could be emulated in software is shown in Listing 2.1 using C-like pseudo code. An array of one-byte elements, i.e., 8 bits, represent the 8-bit registers. Another array with one-byte elements represents the 8-bit memory places. The original machine instruction can then be decoded and treated as if it was executed on an Intel 8085 platform. However, the emulation software can be compiled and run on a different type of processor which could have a completely different architecture.

One drawback of software emulation is possible variations in execution time. A possible case could be that the host system has twice as fast clock frequency compared to the original architecture. But the original architecture could have a multiplier accelerator, enabling faster multiplications than with arithmetic logic. If the host system lacks such accelerator, it might take twice as long to execute a multiplication, despite the doubled clock speed. Said variations in execution time might not be of major concern when software emulation is used to run obsolete word editing software on a PC. However, in a real-time system, where execution deadlines must be met[28, Sec. 1.2.1], such variations are of major concern.

Listing 2.1: Software emulation of Intel 8085 MOV r1 r2 instruction

```

//Register address b'110 are used for memory operations
char registers[8] = { };
char memory[65536] = { }; //Init 64KB memory
...
if (instruction == "MOV") {
    //Register H(4) and L(5) holds desired memory address
    short memoryAddress = registers[4] << 8 | registers[5];
    //HEX number 6 is equal to b'110
    if (r1 == 0x6) // MOV M, r
        memory[memoryAddress] = registers[r1];
    else if (r2 == 0x6) //MOV r, M
        registers[r2] = memory[memoryAddress];
    else //MOV r1,
        registers[r1] = registers[r2];
}

```

2.2.3 Hardware replacement strategies

Two different hardware replacement strategies are described here. Both uses COTS-hardware, FPGA can also be considered as COTS-hardware. However, FPGA is more flexible compared to other, application-specific, COTS-hardware. For example, a COTS microprocessor has a very specific behavior while a FPGA can be programmed to implement a desired behavior. Furthermore, it is likely that an obsolete FPGA can be directly replaced with a new COTS FPGA while a new COTS microprocessor probably has differences in timing or instruction set.

COTS VM approach

Investigation of hardware replacement strategies for obsolete avionics computers took place in the 1990s[29]. Usage of a VM running on top of COTS-hardware allowed for software, written for obsolete hardware, to be executed and re-used on new hardware. Hence, resources could be focused on supporting new hardware instead of rewriting software to fit the new system architecture.

A Real-Time Operating System (RTOS) runs on top of the COTS-hardware. Two VMs runs on top of the RTOS, one VM mimicking original hardware while the other VM runs new expanded software. New, faster, hardware also allows for instructions to be executed faster. Usage of two VMs allows for scheduling of extended software functions between execution of original software.

FPGA approach

Boeing investigated usage of **FPGA** as a replacement option for obsolete hardware in one of their automatic test equipment's. Development of such systems requires software certification, causing software costs higher than hardware costs in some cases. Therefore, a team of engineers' reverse-engineered an existing system and replicated the legacy systems hardware behavior[30].

The system was divided into groups and subsystems, where each circuit board was modelled using either *VHDL* or schematic capture. Each circuit board model was tested to ensure identical hardware behavior as the original board before combining the board models. The documentation available for their project had some flaws in quality regarding readability and correctness, over 200 mistakes were found and resolved[30].

Chapter 3

Introduction to IDTS

An IDTS wayside computer consists of several modules which can be divided into two different kind of groups, *LOOP* and *Central Processing Unit (CPU)*. There is one *LOOP*-group for each inductive loop and one *CPU*-group for each wayside computer. One *LOOP*-group consists of one transmitter, one receiver, one duplexer and one MCU module. The *CPU*-group consists of one memory, one Universal Asynchronous Receiver-Transmitter (UART), one modem and one CPU module. The chain of communication inside a wayside computer is shown in Fig. 3.1. More detailed information about the components of IDTS wayside computer are available in Chapter 3.6.

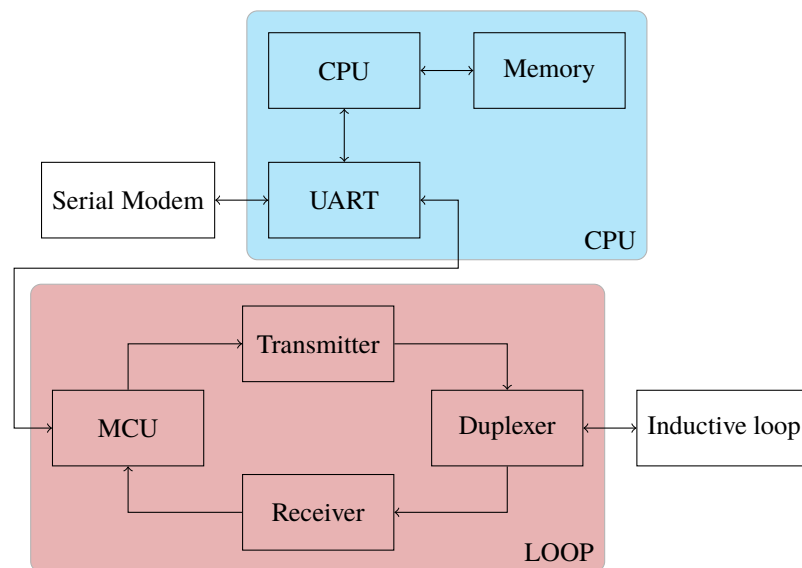


Figure 3.1: Communication hierarchy inside a wayside computer

3.1 Status of parts

Over the years, different parts of IDTS have experienced issues. Approximately 40 power supplies have been upgraded with new components[31]. Generally, there is no new spare parts available on market which fits without modifications, and one relies on repair and usage of existing components in stock. Originally information to vehicle on-board computers were fed by thumbwheel switches, these has been replaced with a digital equivalent as seen in Fig. 2.3. Otherwise, all main on-board parts are the same as when the system was delivered, new cars use equipment from cars which are no longer used for daily operations.

A currently ongoing issue is a shortage of antennas[31]. A possible solution, which is under investigation, is to use one antenna for each car instead of one for each operator cab. Such solution would reduce the number of antennas needed from 752 to 376, assuming that only C20 and C30 stock are used in daily operations.

3.2 Replacement of IDTS on-board equipment

An on-board computer unit, which is compatible with the operator input panel, would not require any software changes on other on-board equipment. Cost of replacing IDTS on-board antenna and computer is estimated to about 830 000 SEK[32] for all C20 and C30 cars. The price estimation only includes cost of labor for installation.

3.3 Suggested improvements

IDTS has the same underlying architecture as when first developed in late 1980s. Some changes have been made, such as increased memory size. However, today requirements have changed, and it could be beneficial to investigate changes of architecture when designing new spare parts.

IDTS was originally designed and used in islanded mode, there was no communication between wayside computers. When a train arrived to a station IDTS displayed the destination on the electronic departure boards. The electronic departure boards went blank when the train left. Since the electronic departure boards no longer are directly controlled by IDTS there is no purpose of allowing islanded operation. Furthermore, the CPU-group introduces a local single point of failure. The CPU-group could be removed, and a LOOP-

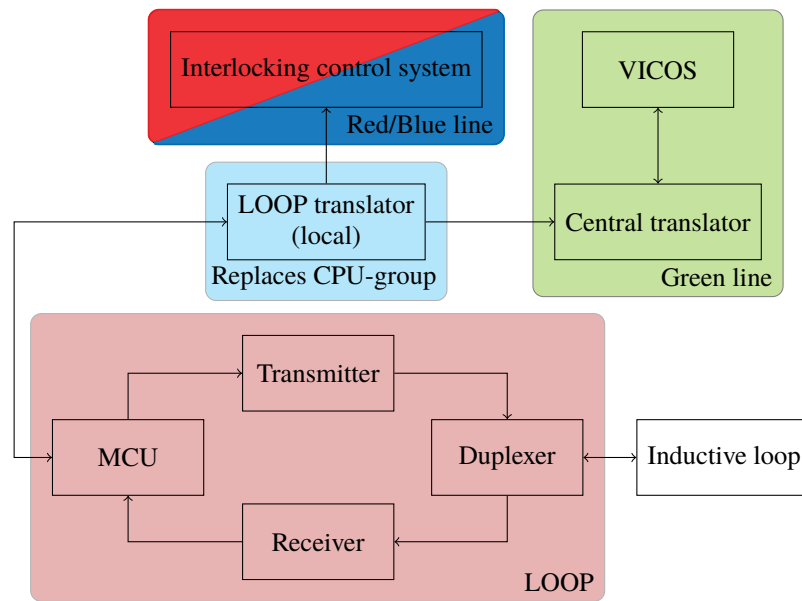


Figure 3.2: Communication hierarchy of an improved wayside computer

group translator could be introduced instead[14]. Such change would allow for a direct connection between a central system and loops, causing only minor disruptions in case of failure. A global single point of failure still exists, the connection between wayside and a central system, which is outside the scope of this project. Further benefits of such architectural change are that information can be transferred when there is a change of state, eliminating unnecessary data traffic, and the possibility of controlling everything at a central level. Information from trains could be received to a central system and interpreted there, instead of having a local interpretation which requires changes of interpretation to be carried out locally.

Investigation is carried out to install a new control system for interlocking at the Red and Blue line. Such control system could replace TIS and use the destination information from IDTS for automatic train routing[14]. A removal of the CPU-group in IDTS and replacement by a LOOP-group translator, feeding information from each loop, could be directly connected to the new control system. The Green line would require an additional translator at central level, since the signaling system has a special interpretation of TIS named VICOS. The translator would be replicating the original IDTS behavior towards VICOS. Figure 3.2 contains a block diagram of the improved architecture.

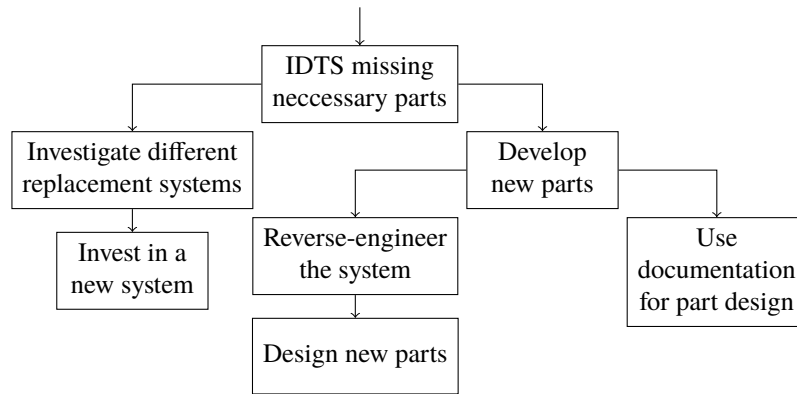


Figure 3.3: Different options for maintaining train identification

3.4 Maintaining train identification

Shortage of spare parts for IDTS poses challenges for maintaining a train identification system in Stockholm Metro. Fig. 3.3 displays different options identified for maintaining the train identification. Hence, different replacement systems were investigated. No direct replacement system was found on the market, the systems found were considered to need costly adaptations. No deeper analysis for cost of system replacement was carried out since no direct equivalent to IDTS was found.

Different hardware replacement strategies which can be applied on IDTS were studied. Through understanding of IDTS current usage, two different hardware replacement strategies seemed to require the same amount of resources: Software emulation of the existing hardware using COTS-hardware and re-using original software, or implementation of a new improved device using COTS-hardware and new software. Emulation tended to require more low-level knowledge of the system architecture and not how the system operated. Replacement with a new device would only require knowledge of the input/output-relationship, not how the operations are carried out. Fig. 3.4 shows a simplified illustration of the black box behavior of a generic IDTS-module. Emulation require mostly knowledge of what happens inside the box while replacement of both software and hardware require mostly knowledge of the function $f(x)$.

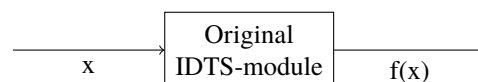


Figure 3.4: Illustration of black box behavior

3.4.1 System documentation

All system documentation was intended to be used for re-engineering of IDTS. However, most parts of the documentation available requires a written permission from the manufacturer. Meeting protocols from 1989, when IDTS was installed, indicates that SL has permissions for usage of the documentation. However, no written document verifying this was found during this project.

The manufacturer has been merged with another company and the new company have responded that they will not allow usage of the documentation for development of new hardware by others. Therefore, the manufacturer documentation was not used during this project since that could cause legal consequences. Instead, a reverse-engineering approach was chosen, where only documentation which did not require written permission were used.

From the identified options in Fig. 3.3 only one branch remained and was investigated.

3.5 Test environment

A test environment was setup which could be used for reverse-engineering and testing of new hardware. The test environment had no connection to operational systems, which provided a non-volatile test bench. Included in the test bench were;

- IDTS wayside master (see Chapter 3.6)
 - One CPU-group
 - Four LOOP-groups
 - One inductive loop
- IDTS on-board equipment
 - Vehicle on-board computer
 - Operator input panel
 - Vehicle antenna
- IDTS wayside tester
- Computer running TIS-simulator

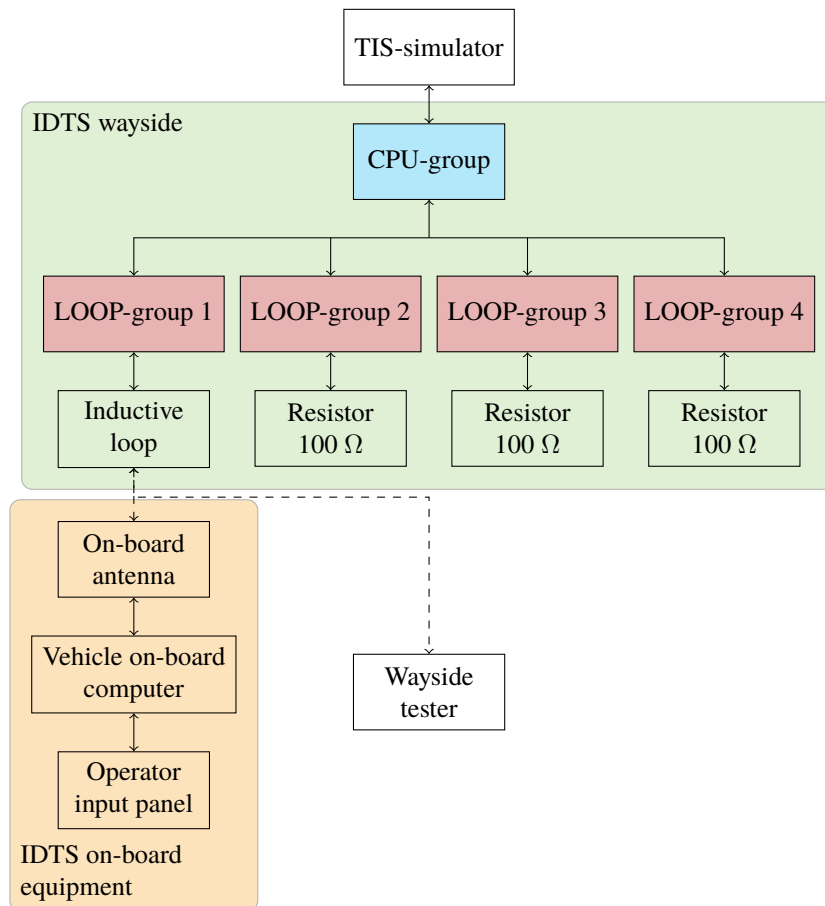


Figure 3.5: Test environment setup

Fig. 3.5 contains a graphical representation of the setup. Only one inductive loop was attached to the system, mainly because of limitations in laboratory space. LOOP-group 2, 3 and 4 had 100 Ω resistors to represent the impedance of an inductive loop, preventing malfunction in the setup due to the absence of inductive loops.

Usage of both on-board equipment and a wayside tester in the test bench allowed for observations of multiple message types.

The TIS-simulator is named *IDTSTIS*, developed by G. Blom for SL Banteknik AB. The software can be used both for communication simulation and for listening to traffic.

IDTS inductive loops are numbered from 1 to 10. The loops are assigned to different groups. Each car passing a loop is stored in the wayside memory until the same car passes another loop assigned to the same group. In some cases, it is preferred to allow updates from the train, such as changed destination, while

Table 3.1: IDTS wayside configuration

Line	2	
Station	BEH*	
Group	PDU(S)	
1	UNDEFINED	
2	UNDEFINED	
...	...	
10	UNDEFINED	
LOOP	TYPE	GROUP
1	N	1
2	N	1
3	N	2
4	N	2
5	N	UNDEFINED
...
10	N	UNDEFINED

still at a station. Therefore, some loops can be configured to only add cars to the wayside memory. An inductive loop can also be configured to only store the train's cars depending on the destination.

Configuration of the IDTS wayside is shown in Table 3.1. Fig. 2.2 represents the same setup. A train passing over loop 1 must pass over loop 1 or 2 to be removed from the wayside memory.

Configuration of electronic departure boards are made when an IDTS wayside is configured. These setup fields can be left empty since the electronic departure boards are no longer directly controlled by IDTS.

Most parts included in the test bench were over 30-year-old and gathered from storage. A considerable amount of time was spent preparing the test environment, including debugging, support, and replacement of faulty parts.

3.6 IDTS components

IDTS wayside units consists of one master unit and up to two slave units. A master unit consists of one CPU-group and up to four LOOP-groups. Slave units consists of up to four LOOP-groups.

Each module contained in one group is connected to a motherboard, Fig. 3.6 shows a more detailed connection layout of a IDTS wayside unit. LOOP

* Abbreviation of Bergshamra

motherboards are connected in series inside each wayside unit, and in series with the CPU motherboard in a master unit. Slave units connect through the main motherboard, mounted in the back of each wayside unit.

3.7 Reverse-engineering of IDTS

Schematics for different modules were deduced by adding all visible components into a KiCad schematic. All modules had only two-layer Printed Circuit Board (PCB)s, making it relatively easy to deduce all connections. Most modules were covered in protective coating, which made it difficult to test all possible connections using only a multimeter. Instead, traces were followed visually combined with using a multimeter to verify the connections using a sharp probe which pierced through the protective coating. All connections found were documented in the KiCad schematic.

The original aim was mainly to develop replacement parts for the CPU-group. However, the CPU-group is using an Intel 8085 processor and contains more circuitry than the LOOP-group modules. Due to the more potential sources of errors arising from having more circuitry it was determined that replacement of the CPU-group could be too complex given the time frame for this project. Instead, the LOOP-group were chosen for development of replacement parts, where mainly the MCU-module was considered which uses the less complex Intel 8032 microcontroller unit.

3.7.1 MCU-module

MCU-modules serves as a control device between the CPU and the inductive loop. Poll telegrams are transmitted from the MCU, through the TX and DPX-modules, to the inductive loop. Telegrams are received, through the DPX and RX-module, and stored in the MCU. Poll telegrams are sent from the CPU to each MCU, which in turns responds with status and possible stored train telegrams.

An Intel 8032 MCU with a separate 32 KB Read-Only Memory (ROM) and an Intel 8255 programmable peripheral interface is mounted to the MCU-module. Additionally, there are several components used for addressing, providing of test signals and signal level handling. Fig. 3.7 contains a block diagram of the major MCU-module components.

With the help of a complete wiring diagram of the LOOP-motherboard it was determined that many connections to the MCU-module 64-pin connector did not connect to anything outside the module. More time would have

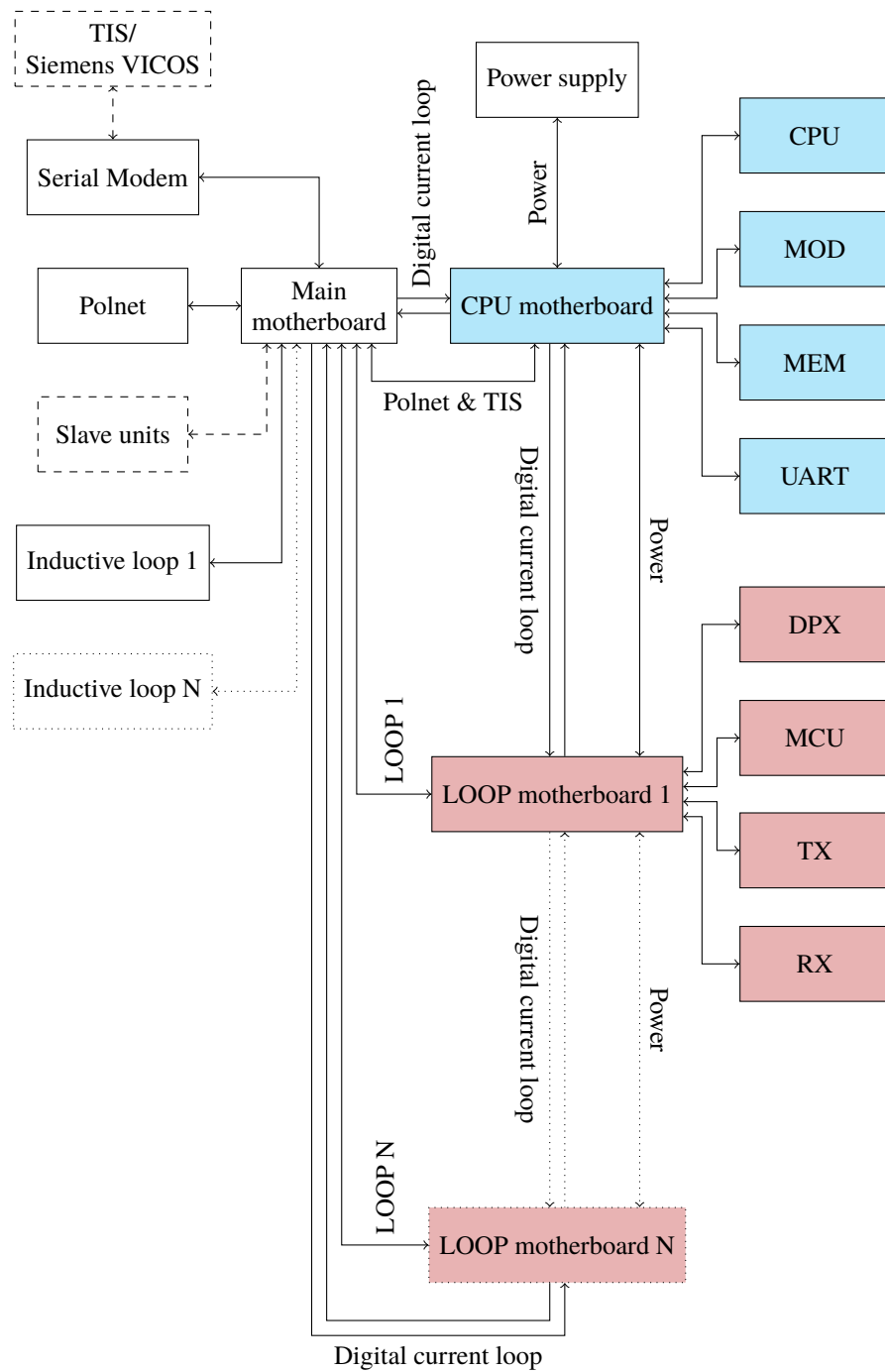


Figure 3.6: Internal connections of a master wayside computer

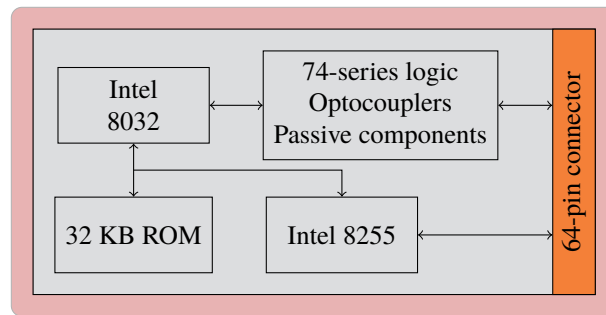


Figure 3.7: MCU-module block diagram

been necessary for reverse-engineering the **MCU**-module if there had been no wiring diagram of the LOOP-motherboard.

Some of the telegrams between the CPU and **MCU** were successfully read and partly decoded. However, the tail of each telegram contained a checksum which was not successfully decoded. Furthermore, it was determined that the observed telegrams could be just a subset of all possible telegram types. Extraction of all possible telegram types and decoding the checksum algorithm were assumed to be less efficient than construction of an emulator.

Chapter 4

Development of replacement hardware for IDTS

Since no compatible replacement hardware for IDTS was found, and the system documentation was not used, emulation remained the only feasible option for maintaining IDTS. Fig. 4.1 illustrates the investigated and remaining options.

The emulator aimed to replace the MCU-module, requiring interfacing to all surrounding modules. A hardware-based emulator, written in *System Verilog* or *VHDL* syntax, was considered too complex as a first-time emulator attempt given the developers experience with HDL. Instead, a software-based emulator was developed, written in C++. C++ was chosen based on the developers' familiarity with high level languages combined with the possibilities of easily converting the source code into C-standard with a small memory footprint. Furthermore, HLS tools often uses C or C++ source code as input. Once a fully operational software-based emulator had been developed it would be possible to adapt the software-based emulator for HLS for realization of a hardware-based emulator.

4.1 Emulator requirements

The original MCU-module uses an Intel 8032, a ROM-less microprocessor in Intel MCS-51 family, to run its software. It is equipped with a 9.2160 MHz crystal oscillator. One instruction cycle takes 12 clock cycles to execute. Hence, one instruction cycle is executed in $\frac{12}{9.2160 \cdot 10^6} \approx 1.3021 \mu\text{s}$. Instructions executed by an emulator must be executed at the same rate or faster. A 8255 programmable peripheral interface is connected to the

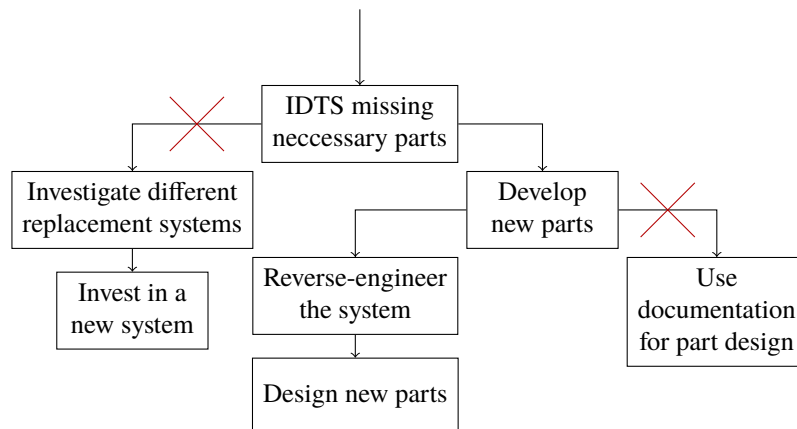


Figure 4.1: Remaining options for maintaining IDTS

8032 microprocessor. The interface has three ports and can be programmed to a variety of configurations. Data available on the ports are in some configurations latched, with corresponding control signals. The data is expected to be latched within 300 ns from the control signal arrives. The emulator must therefore read and store external pin values within less than 300 ns. On the **MCU**-module is also a 32 KB **ROM** used as program memory which should be implemented in the emulator. Remaining 74-series logic, optocouplers and passive components can be bought and used on an emulator **PCB**.

4.2 Software emulator

4.2.1 Code development and structuring

Development was carried out on a PC running 64-bit Microsoft Windows 10 Enterprise*, using Cygwin with g++[†] for compilation and initial testing.

A memory initialization function was developed, enabling import of **ROM** content from a text file to a byte array. A machine code translator was developed through usage of the MCS-51 instruction set reference. Development was carried out in iterations, where one instruction type was developed and partly tested at a time. Said approach allowed for rapid identification of possible coding errors. Code efficiency was considered, however, not highly prioritized during the first development phase since the main goal

* Version 10.0.17763 version 17763

[†] g++ (GCC) 10.2.0

was functionality. Therefore, the machine code translator was separated into two parts, one opcode parser and one opcode executor. The parser reliably determined operation type (*ADD*, *RR*, *XCH* etc.) based on machine code and returned operation type as a string. The string was passed to the executor which decided correct actions based on operation type, current machine code and possible following machine codes.

The memory structure was created from one-byte arrays since the Intel MCS-51 family has register and memory spaces with one byte each. Intel 8032 has 256 bytes of Random Access Memory (*RAM*) where the lower half is both direct and indirect addressable. The upper half is only indirect addressable. Direct addressing of high order addresses is used for access to the special function registers. Solving of potential addressing issues were carried out by allocating 256 bytes for both *RAM* and the special function registers, causing 128 bytes being allocated without being accessible during emulation. 64 KB was allocated for the *ROM*. A two-byte variable was allocated for the 16-bit performance counter.

A 8255 object was defined with helper methods such as *reset* and *read*. Public methods such as *recieveStrobedDataA* were defined to be attached to interrupts, ensuring data to be read and stored when operating in strobed mode.

4.2.2 Memory content

Intel .hex-files, containing raw memory content, were available for both CPU and *MCU*. The content would otherwise have needed to be extracted from the Programmable Read-Only Memory (*PROM*).

The files were opened in a text editor, Listing 4.1 contains an example of the file content. For easier analysis and filtering of data all lines were copied to a spreadsheet in Microsoft Excel, with each line on a separate row and in one column, shown in Fig. 4.2. Microsoft Excel's *text-to-column* function was used, separating the text into multiple columns. Finally, labels were added, and the memory content was analyzed. Fig. 4.3 displays the final spreadsheet result. The relevant content was copied to a .csv-file, formatted to be read by a IDTS emulator.

Listing 4.1: Example content of hex-files

```
: 020000040000FA
: 100000006D8ABF4A3702DC579545649654B9DD3195
: 1000100011B137162013FA9D0F5B797505B15F2C6E
: 10002000B57454126A9934BFFD40A6178DA8A588EF
: 100030006D22B6E8BD0F122BE19AFF85CEA44C7855
```

:020000040000FA
:100000006D8ABF4A3702DC579545649654B9DD3195
:1000100011B137162013FA9D0F5B797505B15F2C6E
:10002000B57454126A9934BFFD40A6178DA8A588EF
:100030006D22B6E8BD0F122BE19AFF85CEA44C7855

Figure 4.2: Example content of hex-files after import to spreadsheet

Start code	Byte count	Address	Record type	Data	Checksum
:	02	0000	04	0000	FA
:	10	00	0000	6D 8A BF 4A 37 02 DC 57 95 45 64 96 54 B9 DD 31 95	
:	10	00	1000	11 B1 37 16 20 13 FA 9D 0F 5B 79 75 05 B1 5F 2C 6E	
:	10	00	2000	B5 74 54 12 6A 99 34 BF FD 40 A6 17 8D A8 A5 88 EF	
:	10	00	3000	6D 22 B6 E8 BD 0F 12 2B E1 9A FF 85 CE A4 4C 78 55	

Figure 4.3: Example content of hex-files after separation of data

4.2.3 Implementation

Implementation of the [MCU](#)-emulator was carried out using an ARM Cortex-M4 microcontroller, a STM32F411CEU6 clocked at 100 MHz. The microcontroller has 512 KB of program memory and an additional 128 KB for variables. Relatively low pricing of the development board, the possibility of programming it using C++ code and high clock speed were the main motivations for choosing said microcontroller. A [FPGA](#) based solution was considered and ruled out partly due to higher pricing of development boards. It was also considered to introduce more sources of errors and require more work when transitioning from high-level programming to [HDL](#). The microcontroller was estimated to provide a fully working implementation faster. With its clock speed of 100 MHz, it needed to carry out a 8032 instruction cycle within $\frac{12 \cdot 100}{9.2160} \approx 130$ clock cycles. According to the Cortex-M4 technical reference manual most instructions takes one cycle to execute. The most demanding instruction, not including some load instructions or barrier or wait instructions, is division which takes up to 12 cycles.

Further development and programming were carried out using Arduino IDE with STM32duino library, allowing for usage of STM32-controllers with the Arduino IDE.

Only minor code changes were necessary before compilation and uploading to the microcontroller.

Development of hardware dependencies were carried out once the code

had been properly compiled for the implementation architecture. Functions of the 8032 architecture which were close to hardware, for example baud rate generation using hardware timer, turned out to be more difficult than anticipated.

Each call for a 8032 instruction execution was designed to return the amount of 8032 instruction cycles required for that operation, 1, 2, 3 or 4 instruction cycles. A hardware timer on the Cortex-M4 controller were clocked at 96 MHz and used to ensure the timing requirements were met. Eq. (4.1) shows the timing requirement measured in ticks, where n is the number of 8032 instruction cycles required. Hence, the maximum number of ticks for a single cycle instruction were 125 ticks.

$$T \leq \frac{f_{8032}}{12f_{Cortex-M4}}n = \frac{9.2160}{12 \cdot 96}n = 125n \quad (4.1)$$

4.3 Hardware emulator

A hardware emulator was designed using an Anlogic EG4S20 based FPGA board. An HLS tool was meant to re-use the code written for the software emulator. However, the tool did not manage to synthesize the C++-code without any changes. Therefore, open-source hardware descriptions were used for the 8032 and 8255, gathered from different sources.

The 8032 hardware description, available under the name T51, contained all features available in an Intel 8032. However, there were some differences in the timing. Firstly, one instruction cycle for an Intel 8032 requires 12 clock cycles while the T51 requires only one clock cycle per instruction cycle. Secondly, there were deviations in some instruction execution time. Table 4.1 displays the identified deviations between the original Intel 8032 and the T51 emulated implementation.

4.3.1 Memory content

Included with the 8032 hardware description was a C++-program, which took an Intel .hex-file as input and returned a full VHDL description of a ROM containing the specified memory content.

Table 4.1: Instruction execution time deviations

HEX-code(s)	Instruction mnemonic	Instruction cycles	
		Intel 8032	T51
11, 31, 51, 71, 91, B1, D1, F1	ACALL	2	3
24, 25	ADD	1	2
34, 35	ADDC	1	2
01, 21, 41, 61, 81, A1, C1, E1	AJMP	2	3
53	ANL	2	3
54, 55, 82, B0	ANL	1	2
B4, B5, B6, B7, B8, B9, BA, BB, BC, BD, BE, BF	CJNE	2	3
C2	CLR	1	2
B2	CPL	1	2
15	DEC	1	2
84	DIV	4	12
D5	DJNZ	2	3
05	INC	1	2
A3	INC	2	1
20	JB	2	3
10	JBC	2	3
30	JNB	2	3
12	LCALL	2	3
02	LJMP	2	3
74, 76, 77, 78, 79, 7A, 7B, 7C, 7D, 7E, 7F, A2, E5, F5	MOV	1	2
75, 85, 90	MOV	2	3
83	MOVC	1	2
F0, F2, F3	MOVX	2	1
A4	MUL	4	1
42, 44, 45, A0	ORL	1	2
43	ORL	2	3
D0	POP	2	3
C0	PUSH	2	3
22	RET	2	3
32	RETI	2	3
D2	SETB	1	2
80	SJMP	2	3
94, 95	SUBB	1	2
C5	XCH	1	2
62, 64, 65	XRL	1	2
63	XRL	2	3

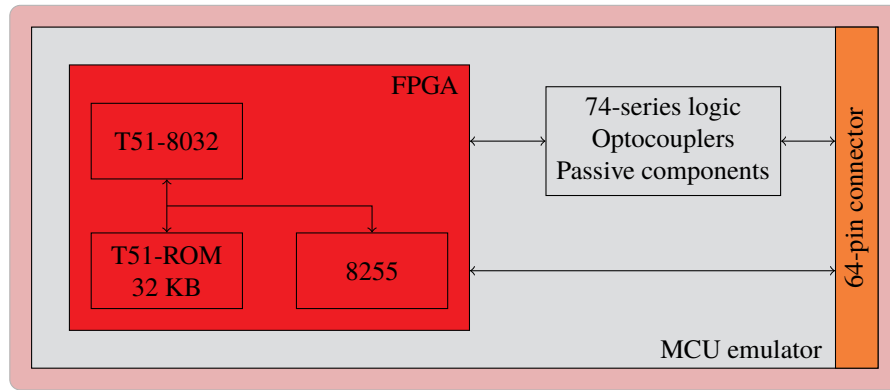


Figure 4.4: Hardware emulator architecture

4.3.2 Implementation

A **MCU**-entity was defined in order to connect the different components and to provide desired input- and output signals to physical components. Fig. 4.4 illustrates the emulator architecture.

Verification of the **MCU**-entity was carried out using ModelSim. A simulation was setup where the internal signals were observed and compared against the MCS-51 instruction set reference.

Intel 8032 hardware timers are incremented every machine cycle. The serial interface baud rate can be set to depend on the overflow-rate of a hardware timer. Hence, it was considered necessary keeping the same machine cycle frequency to ensure proper serial communication and correct notion of time. The original machine cycle frequency was $\frac{9.216 \cdot 10^6}{12} = 0.768$ MHz. The crystal oscillator with the closest frequency multiple was at 18.432 MHz. To solve for this the **MCU**-entity was extended with a frequency divider at its clock-input, dividing the clock frequency by 24.

Chapter 5

Testing and Verification

A timing issue was discovered during development of timing synchronization of the software emulator. The original hardware would in worst-case perform one instruction in 500 ticks. The best-case timing of one software emulated instruction took over 2000 ticks. A worst case was observed at 4 000 000 000 ticks. Timing of individual functions were carried out, determining that the majority of ticks were consumed on op-code decoding and op-code execution. However, also smaller helper functions, for instance updating of the PSW-register parity bit, consumed over 20 ticks each. Closer inspections of the assembly code confirmed the measurements. Further development of the software emulator was stopped since the relation between requirement and actual result differed by at least a factor 16. When development was stopped some features were yet to be implemented. Interrupt behavior was designed without support for raising of interrupts. Moreover, timers were not implemented, and the serial connection only supported 8 data bits. Table 5.1 displays the time spent developing the software emulator.

Table 5.1: Time consumption for MCU-module software emulator design

Step	Hours
8032 software emulator coding	30
Software emulator coding (not including 8032)	60
	90

The hardware emulated replacement module was installed in the laboratory wayside computer, replacing one of the MCU-modules. A USB-to-serial interface was connected between the MCU-module and a computer, allowing for monitoring of data exchange between MCU-modules and the CPU. The replacement module was observed to properly send status responses to the

CPU. It was also verified that the loop-ID was properly set by replacing different MCU-modules in the setup and observing the new status response. However, the communication with the inductive loop was malfunctioning. Status LEDs on the receiver- and transmitter-modules should be flashing in a certain pattern specified in the service handbook. The receiver-modules' LEDs were observed to have correct pattern, while the transmitter-modules' LEDs were having a static pattern. An original MCU-module and the replacement module were brought to an electronics laboratory which had a digital oscilloscope, allowing for comparison of signals. Multiple bugs were detected, as for example a faulty placed junction in the new schematic which connected a pin adjacent to the desired pin. Another bug detected was a deviation in pulse width of a signal generated from a monostable multivibrator. The signals' pulse width is set by a resistor-capacitor network. The pulse width deviation was considered to be caused by air-wires and short spacing between some PCB-traces, affecting the circuits' capacitance. Both modules produced the same output on the 64-pin connector in the electronics laboratory, despite the pulse width deviation. Nonetheless, when the replacement module was installed to the laboratory wayside it was still not properly communicating with the inductive loop. One possible source of error was identified to be the necessary level conversion between the original 5 V logic one and the FPGAs' corresponding 3.3 V. Another identified source of error was the capacitance and interference caused by the air-wires and shortly distanced traces. It was concluded that the most efficient solution would be to correct all identified bugs and manufacture a new PCB.

Table 5.2 shows the time consumption during all steps of engineering a new, directly compatible, MCU-module for IDTS. Pricing, not including value added tax or shipping costs, of all components used in the hardware emulator are shown in Table 5.3.

Table 5.2: Time consumption for engineering new MCU-module

Step	Hours
Deducing LOOP-motherboard schematics	6
Deducing MCU-module schematics	17
Hardware emulator coding	16
Hardware emulator circuit design	30
Prototype manufacturing	6
	75

Table 5.3: Cost of components used for the hardware emulator

Component	Price [SEK]
FPGA	196
64-pin connector	92
74-series logic, optocouplers, passive components	98
	386

Chapter 6

Discussion

While carrying out this project a major obstacle was not being allowed to use key documentation available. Without the documentation only one replacement method appeared to be efficient, emulating the obsolete components and run original software.

Another learning was how difficult it was to reverse-engineer the behavior of a system. Reverse-engineering one piece of a system's behavior, for example figuring out message content, can be relatively simple. However, determining the complete system behavior, guaranteeing all possible states are covered is a demanding task without seeing all system components. A major system component is the software, which may not be disassembled for the purpose of replicating the system behavior. The software may however be disassembled if the purpose is to achieve compatibility between different systems. This project aimed to develop a **IDTS** compatible product, which could motivate that disassembling of source code could be allowed for achieving compatibility. On the other hand, to make sure no legal boundaries are crossed it is suggested to not disassemble the source code which is to be replaced. Such action would possibly be considered as copyright infringement. Instead, source code for the remaining systems should be disassembled. For example, designing of an **MCU**-module replacement might have been made by disassembling the **CPU**-modules' and the vehicle on-board computers' source code. However, the **CPU**-module has a program memory of 128 KB. Translation of 128 KB of machine code into assembly code might be done using a disassembler. The assembly code would then have to be analyzed to find all communication related commands, which also would require knowledge of the architecture to understand how access to peripherals are made. Thus, making it likely to take longer time than replacing the **MCU**-module with an emulator.

All legal and practical aspects from this project lead up to a suggested generic approach to dealing with outdated interconnected computer systems, shown in Fig. 6.1.

6.1 Hardware emulation as replacement strategy

The results in Table 5.2 suggests that a replacement MCU-module can be developed to a relatively low cost. However, the hardware emulator relies on open-source hardware descriptions which were available on the Internet. This is not the case for all integrated circuits. Hence, one should add the cost of hardware description development. The total time necessary to develop an MCU-module replacement is therefore 165 hours, assuming that coding an emulator in C++ is equal to coding in VHDL.

Stockholm Metro is to be extended with a total of 12 new stations until 2030, according to current plans. All stations will require at least four LOOP-groups and one CPU-group, Table 6.1 contains a summary of modules needed. With an assumed engineering cost of 1500 SEK per hour[33], the cost of a replacement MCU-module design would be approximately 247 500 SEK. Based on the hardware cost reported in Table 5.3 the total cost of hardware is estimated to 500 SEK for each module, resulting in a unit-cost of approximately 5700 SEK.

One could assume that the major cost of replacement hardware design is related to the CPU-, UART- and MCU-hardware since these modules contains obsolete integrated circuits. The MEM-module contains PROMs and

Table 6.1: Modules needed during expansion of Stockholm Metro

Module name	Amount needed
CPU	12
Memory	12
MOD	12
UART	12
Amplifier	0
Duplexer	48
MCU	48
Transmitter	48
Receiver	48

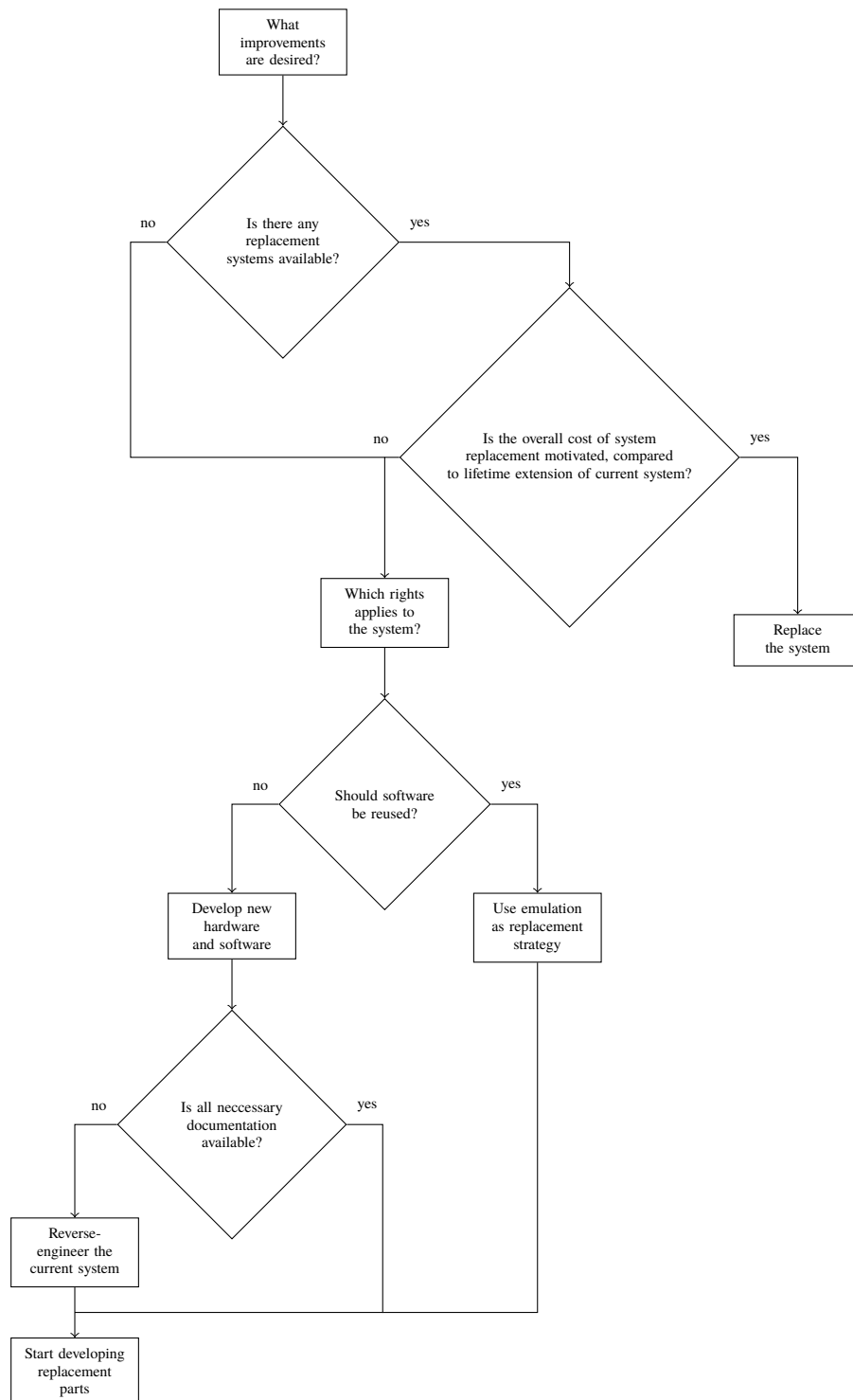


Figure 6.1: General approach of dealing with outdated interconnected computer systems

addressing logic, which are generic hardware available as open-source and can alternatively be designed following tutorials online. Other modules contain passive components and generic integrated circuits. Hence, most engineering time would be needed to deduce schematics and designing a **PCB**.

Table 6.2: Steps necessary to develop replacement hardware for IDTS

Module	Required steps			
	Deduce schematics	Develop emulator	Develop new PCB	Prototyping
CPU	X	X	X	X
Memory	X		X	X
MOD	X		X	X
UART	X	X	X	X
Amplifier	X		X	X
Duplexer	X		X	X
MCU	X	X	X	X
Transmitter	X		X	X
Receiver	X		X	X

IDTS vehicle on-board computer consists of modules similar to the wayside LOOP-group, namely:

- Duplexer
- **MCU**
- Transmitter
- Receiver
- I/O expander

Despite the naming only the receiver is identical for wayside and on-board equipment. Other modules have the same **PCBs** but a few differences regarding components used. Therefore, it is likely possible to re-use most parts of the work for wayside equipment. However, the cost estimation below includes a full development cost to not underestimate the price.

Table 6.2 displays the anticipated required steps necessary for a full development of new **IDTS** parts. It is likely that the development cost for the CPU-module is higher than for the **MCU**-module, given that the CPU-module is more complex. However, it is also likely that the development cost for the

Table 6.3: Estimated total development costs of IDTS replacement hardware

Module name	Development cost
CPU	247 500
Memory	88 500
MOD	88 500
UART	247 500
Amplifier	88 500
Duplexer	88 500
MCU	247 500
Transmitter	88 500
Receiver	88 500
On-board duplexer	88 500
On-board MCU	247 500
On-board transmitter	88 500
On-board I/O expander	88 500
	1 786 500

UART-module is lower as compared to the MCU-module, since the UART-module contains less complex components than the MCU-module. Hence, one could assume that the average development cost of a CPU-module and an UART-module are approximately the same as the MCU-modules development cost.

By combining the data in Table 6.1 with the time spent on different steps developing a MCU replacement one could estimate the cost of compatible hardware development for each module using Eq. (6.1). The results are shown in Table 6.3.

$$C_{module} = C_{engineer} (T_{schematics} + T_{emulator} + T_{PCB} + T_{prototyping}) \quad (6.1)$$

Table 6.4: Estimated hardware cost of system replacement using infsofts' solution

	Amount needed	Unit price	Total cost
infsoft locator node	367	1300	477 100
Beacon	498	300	149 400
			626 500

Comparing the estimated total cost of developing compatible spare parts with installation of new equipment indicates that hardware emulation could be

a cost-efficient method for lifetime extension. Table 6.4 shows an estimated hardware cost using the train tracking solution by infsoft. It would require one locator node per metro car, in total 367. One beacon should be placed where there currently is an inductive loop. In total there are 498 inductive loops in Stockholm Metro. The exchange rates used are 1 € for 10 SEK and 1 £ per 11.7 SEK. Further costs related to such system replacement are:

- Installation of new wayside equipment
- Installation of new on-board equipment
- Removal of IDTS-equipment
- Adaptation to allow for train operator input, as shown in Fig. 2.3
- Adaptations to forward information to TIS and Siemens VICOS
- Designing new equipment installation
 - Wiring diagrams
 - Physical placement
- Project planning and system testing

Installation of new wayside equipment and removal of inductive loops requires blocking of the track section to ensure safety of the personnel working. A standard cost of such works is approximately 24 000 SEK per night[33]. Removal of remaining IDTS wayside equipment can be carried out daytime at an approximated cost of 12 000 SEK per day[33]. Table 6.5 contains a summary of the estimated cost for wayside related works for a train identification system change.

Replacement of IDTS on-board equipment has been estimated to 830 000 SEK under the assumption that there is no need for re-wiring. The estimation does not consider the need of having two different systems running in parallel during installation works. Hence, the estimate should be nearly doubled since cars should be stripped of the IDTS on-board equipment once the new system has been put into service. Combining costs for wayside and on-board labor with the cost estimated in Table 6.4 a system replacement would cost at least 7 084 500 SEK. Considering the RDTS project, which currently has an accumulated cost of approximately 23 million SEK, 7.1 million SEK is a relatively low estimate and does not include all costs which are related to such system replacement as for example adaptations for communication with TIS.

Table 6.5: Estimated cost of wayside equipment removal and installation using infsofts' solution

	Days/Nights needed	Unit price	Total cost
Removal of inductive loops	117	24 000	2 808 000
Removal of remaining IDTS wayside equipment	117	12 000	1 404 000
Installation of new wayside equipment	59	24 000	1 416 000
			5 628 000

Table 6.6: Comparison of costs to secure train identification in Stockholm Metro

System		Cost [SEK]
RDTs		23 million
infsoft railway train tracking		7.1 million
Hardware replacement using hardware emulation	Development costs (starting cost)	1.8 million
	Hardware for 12 new stations	0.1 million
	New hardware at all current stations and all cars	2.1 million

Table 6.6 contains a summary of the costs for infsofts' railway train tracking solution and RDTs. It also includes the cost of designing new components for IDTS and the estimated cost of hardware in different scenarios, calculated using Eq. (6.2) with $C_{modules} = 500$. The cost of replacing and installing new compatible hardware has been discarded since it can be carried out during regular maintenance. All costs are rounded to the nearest 100 000 SEK.

$$C_{replacement-hardware} = C_{modules} (4N_{loops} + 4N_{waysideMasters} + 5N_{cars}) \quad (6.2)$$

The solution provided by infsoft should not be considered expensive, the main driver of cost is installation and not the technical solution. Since IDTS is already installed, and fulfills all requirements, there is no motivation to changing the system except shortage of parts.

6.2 Reliability

There have been no indications of malfunction of the open-source hardware descriptions used. Nonetheless, there could be hidden bugs which are not covered by any guarantee. In the case of [IDTS](#) such issues are relatively small. In general, this could be a more significant issue, if the hardware description for example is used to replace obsolete parts in a safety-critical system such as a railway signaling system. Development of a random test bench, which verifies the correct behavior of the hardware descriptions, could help avoid issues with bugs not detected during operational tests.

Chapter 7

Conclusions and Future work

7.1 Conclusions

Hardware replacement methods which are not re-using original software, such as a [COTS MCU](#) running new and compatible software, have shown indications of being an unfavorable choice when system documentation is not available. Emulation, which re-uses original software, has shown to be both a viable replacement method and cost-efficient. Replacement systems for [IDTS](#) were identified during the project. However, the cost of installation and customization suggests it is more cost-efficient to develop new hardware for [IDTS](#). Installation of a replacement system for [IDTS](#) has been estimated to cost over 7 million SEK, while development and cost of replacement hardware were estimated to cost 4 million SEK. Thus, using hardware emulation can be a cost-efficient method to extend lifetime of outdated interconnected computer systems such as [IDTS](#). In the case of [IDTS](#) the main benefit of developing compatible parts were the possibility of continuing using existing hardware until it fails, thus preventing expensive premature change of equipment and organizing a major project around it.

Since the hardware emulator was not fully operational during laboratory testing it was never tested in a live environment. Nevertheless, it was partly operational and clearly indicated the viability of hardware emulation as a replacement strategy.

7.2 Limitations

Results and conclusions drawn from this case study could hopefully help others in their choice of obsolete hardware replacement. However, the

conclusions are based on multiple assumptions which can vary in different cases, thus changing the cost-efficiency of hardware emulation as a replacement strategy.

No consideration has been taken to cost of maintenance when comparing system replacement with development of new spare parts.

Other factors to consider is the legal aspects as for example patent infringement. During this case study all components were well over 20 years old and could therefore not have any valid patents in Sweden.

7.3 Future work

7.3.1 Further development of parts for IDTS

To fully extend lifetime of IDTS all remaining modules should be reverse-engineered and replaced with new hardware. Since the CPU-group mainly is to be emulated it is suggested to put all four modules into one hardware description and run it on a FPGA. Hence, reducing four PCBs down to one. Modules in the LOOP-group might be implemented on one PCB, but the extra engineering cost needed to carry out such operations could be higher than the savings in material. Additionally, in some wayside configurations the duplexer module is replaced with an amplifier module. Combining the LOOP-group modules would require different variations of the replacement-device. Furthermore, the receiver module is shielded, possibly due to potential interferences to or by the wireless transmission.

7.3.2 Further explorations in hardware replacement strategies

Reverse-engineering and development of a hardware emulator shows indications of being an efficient lifetime extension strategy for IDTS. The components involved were 8-bit architectures with 256 operation codes. It would be interesting to investigate the extra effort needed to make an emulator for a 16- or 32-bit architecture, which most likely has 2^{16} or 2^{32} operation codes.

It was intended to use an HLS tool if a software emulator were not sufficient. It later turned out to be issues with compiling the C++ code written, and no further investigations were carried out. Future work could try comparing efficiency of developing a hardware emulator from C or C++, and use an HLS tool to create a hardware description, with the alternative of writing a hardware description directly.

References

- [1] Region Stockholm. De nya stationerna. [Online]. Available: <https://nyatunnelbanan.se/sv/barkarby-stationer> Accessed: 2021-02-17
- [2] ——. De nya stationerna. [Online]. Available: <https://nyatunnelbanan.se/sv/stationer-nacka> Accessed: 2021-02-17
- [3] ——. De nya stationerna. [Online]. Available: <https://nyatunnelbanan.se/sv/de-nya-stationer> Accessed: 2021-02-17
- [4] ——. De nya stationerna. [Online]. Available: <https://nyatunnelbanan.se/sv/arenastaden-stationer> Accessed: 2021-02-17
- [5] ——. De nya stationerna. [Online]. Available: <https://nyatunnelbanan.se/sv/de-nya-stationer-s-o> Accessed: 2021-02-17
- [6] Trafikförvaltningen, Region Stockholm. Livstidsförlängning trafikstyrningssystem gröna linjen. [Online]. Available: <https://www.sll.se/globalassets/5.-politik/politiska-organ/trafiknamnden/2020/8-22-sep.-2020/11.-delning-av-investeringsobjekt-anpassning-trafikstyrningssystem-befintlig-tbana-samt-livstidsforlangn-gr.-linje.pdf> Accessed: 2021-02-17
- [7] *STOCKHOLMS TUNNELBANOR 1975 : teknisk beskrivning*. Stockholm: Stockholms läns landsting, 1975. ISBN 991-355974-x
- [8] P. Heick. Ur arkivet: Här inviger kungen Skarpnäcks t-bana. [Online]. Available: <https://www.svt.se/nyheter/lokalt/stockholm/har-inviger-kungen-skarpnacks-t-bana> Accessed: 2021-02-23
- [9] Region Stockholm. Nya tunnelbanetåg på röda linjen. [Online]. Available: <https://www.sll.se/c30> Accessed: 2021-02-23

- [10] ——. Moderniserade tunnelbanevagnar. [Online]. Available: <https://www.sll.se/verksamhet/kollektivtrafik/aktuella-projekt/c20/> Accessed: 2021-02-23
- [11] TPUT Hy, *Tågidentifieringssystemet Vagnsutrustning*.
- [12] Aktiebolaget Stockholms Spårvägar. Förslag till automatiskt signalsystem för tunnelbanan. [Online]. Available: https://www.ekeving.se/rt/tub/si/HS/19470510_Boberg.pdf Accessed: 2021-02-18
- [13] M. Ohlsson/Region Stockholm, “Kostnader för RDTS,” Mar 2021.
- [14] A. Ahrsjö/Region Stockholm, “Förbättringar av IDTS,” Mar 2021.
- [15] A. Ulmestig/AB HgE Konsult, “Introduktion till Cactus trafikstyrningssystem,” Mar 2021.
- [16] Cactus. Cactus TMS. [Online]. Available: <https://www.cactus.se/en/cactus-tms> Accessed: 2021-03-02
- [17] infsoft GmbH. Railway train tracking system. [Online]. Available: <https://www.infsoft.com/use-cases/railway-train-tracking-system> Accessed: 2021-02-22
- [18] ——. infsoft Locator Node 1400. [Online]. Available: <https://www.infsoft.com/technology/receiver-hardware/infsoft-locator-node-1400> Accessed: 2021-04-11
- [19] T. Winkler/infsoft GmbH, “E-mail about Railway Train Tracking System,” Mar 2021.
- [20] BeaconZone Ltd. Road Stud. [Online]. Available: <https://www.beaconzone.co.uk/waterproof/RoadStud> Accessed: 2021-05-26
- [21] G. J. Popek and R. P. Goldberg, “Formal requirements for virtualizable third generation architectures,” *Commun. ACM*, vol. 17, no. 7, p. 412–421, Jul. 1974. doi: 10.1145/361011.361073. [Online]. Available: <https://doi-org.focus.lib.kth.se/10.1145/361011.361073>
- [22] VMware. CPU Virtualization Basics. [Online]. Available: <https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.resmgmt.doc/GUID-DFFA3A31-9EDD-4FD6-B65C-86E18644373E.html> Accessed: 2021-04-16

- [23] Sveriges riksdag. Lag (1960:729) om upphovsrätt till litterära och konstnärliga verk. [Online]. Available: https://www.riksdagen.se/sv/dokument-lagar/dokument/svensk-forfattningssamling/lag-1960729-om-upphovsratt-till-litterara-och_sfs-1960-729 Accessed: 2021-02-17
- [24] B. J. LaMeres, *Quick Start Guide to VHDL*, 1st ed., 2019. ISBN 3-030-04516-1
- [25] O. Gazi, *A Tutorial Introduction to VHDL Programming*, 1st ed., 2019. ISBN 981-13-2309-7
- [26] Scheidt & Bachmann GmbH. Zsb 2000. [Online]. Available: https://www.scheidt-bachmann.se/fileadmin/downloads/en/signalling-systems/brochures/ZSB/002_Prospekt_ZSB2000_2018_0730_en.pdf?utm_source=website&utm_campaign=download_tracking&utm_medium=link&utm_content=002_Prospekt_ZSB2000_2018_0730_en.pdf Accessed: 2021-02-16
- [27] Intel. The MCS-80/85 Family User's Manual. [Online]. Available: http://www.bitsavers.org/components/intel/MCS80/MCS80_85_Users_Manual_Jan83.pdf Accessed: 2021-02-17
- [28] J. Cooling, *The the Complete Edition - Software Engineering for Real-Time Systems: A Software Engineering Perspective Toward Designing Real-Time Systems*. Birmingham: Packt Publishing, Limited, 2019. ISBN 1839216581
- [29] J. Luke, J. W. Bittorie, W. J. Cannon, and D. G. Haldeman, "Replacement strategy for aging avionics computers," *IEEE Aerospace and Electronic Systems Magazine*, vol. 14, no. 3, pp. 7–11, 1999. doi: 10.1109/62.750422
- [30] C. Thompson, "Upgrading obsolete integrated circuits using field programmable gate arrays (fpga)," in *2014 IEEE AUTOTEST*, 2014. doi: 10.1109/AUTEST.2014.6935173 pp. 365–371.
- [31] R. Rahlen/MTR Tech, "Upparbetning av IDTS, introduktion," Feb 2021.
- [32] G. Spång/Region Stockholm, "Kostnadsestimering utbyte av IDTS vagnsutrustning," Mar 2021.

- [33] N. Andersson/Region Stockholm, “Kostnadsestimering utbyte av IDTS waysideutrustning,” June 2021.
- [34] P. Viktorsson, “Patenträttsliga aspekter på reverse engineering,” 2008. [Online]. Available: <http://lup.lub.lu.se/student-papers/record/1335238/file/1646546.pdf> Accessed: 2021-02-23
- [35] Sveriges riksdag. Patentlag (1967:837). [Online]. Available: https://www.riksdagen.se/sv/dokument-lagar/dokument/svensk-forfattningssamling/patentlag-1967837_sfs-1967-837 Accessed: 2021-02-23

Appendix A

Legality of reverse-engineering

Previous research concluded that reverse-engineering of a patented product is legal in Sweden, to a limited extent[34, p. 8]. Reverse-engineering as a professional can be patent infringement, while reverse-engineering for personal use has no limitations. The researcher concluded that reverse-engineering and duplication of a patented product is allowed for research and development, but the product must be destroyed afterwards and may not be transferred to a third party or be used for other projects[34, p. 9]. Products sold before filing of a patent application are not covered by the Swedish patent law[34, p. 9]. A Swedish patent can be valid up to 20 years after patent application[35, 40§]. Hence, since **IDTS** was developed over 30 years ago there should be no valid Swedish patents for **IDTS**.

Appendix B

Swedish - English metro dictionary

Listed below are the translations used, from Swedish to English, in this report.

Swedish	English
Egen banvall	Reserved track
Huvudljussignal	Color light signal
Hyttsignalsystem	Cab signalling
In-ut ställverk	Entrance-exit interlocking
Indikeringslåda	Route selector panel
Spårvagn	Tram
Spårväg	Tramway
Ställverk (signal)	Interlocking
Ställverksoperatör	Switchman
Ställverkstavla	Interlocking control panel
Trafikledare	Dispatcher
Trafikledningscentral	Rail control center
Trafikstyrningssystem	Traffic management system
Tunneltågförare	Train operator

TRITA-EECS-EX-2021:521