Tuomas Du-Ikonen

Engine Fuel Injection Timing

A Design for an Automatic Verification System

School of Technology and Innovations
Master's thesis in Discipline
Automation and Computer Science

Vaasa 2022

UNIVERSITY OF VAASA
School of Technology and Innovations
Author: Tuomas Du-Ikonen
Title of the Thesis: Engine Fuel Injection Timing : A Design for an Automatic Verification System
Degree: Master of Science
Programme: Automation and Computer Science
Supervisor: Jarmo Alander
Year: 2022 **Pages:** 81

ABSTRACT:
This thesis describes the development of an automatic testing system for the timing of the fuel injection of a 4-stroke engine. The fuel injection timing is managed by an electronic engine control unit which has a distributed modular design. New software and hardware updates are released every few months for the engine control unit. Furthermore, fuel injection timing must be tested for each new software release, because incorrect timing could potentially lead to engine failure. Thus, automating this frequent testing procedure, which can take 2–5 days manually, is expected to save both time and money. Therefore, the object of this work is to develop a design of an automatic fuel injection timing testing system.

There are already abundant scientific studies available related to fuel injection timing and engine control unit. The majority of these studies in the literature review cover various topics about the effects of alternative injection technologies and fuels. A limited number of them comprise the subject of automatic fuel injection timing.

Design science was chosen as the research method because of its suitability for product development projects. The most important research question is what the design architecture must be like for testing injection timing. This work started with a comprehensive analysis of the different factors that could affect the design. Underlying motivation for developing an automatic testing system, stakeholders involved, alternative ways for testing implementation, and various other points of view were covered. After defining the system requirements, the setup was built to measure the timing of fuel injection pulses from the engine control unit, which utilized the National Instruments Compact RIO hardware and software programmed with LabVIEW. This program automatically generates an Excel report of the timing test.

The design of a testing system architecture that would allow measurements to be made from any of the 112 fuel injection terminals of the control unit was successfully developed. Measurements performed with Compact RIO hardware proved to be accurate and could determine the crankshaft angle with the required accuracy. The accuracy of the testing system was ±5 µs. Next, the development of communication between the testing hardware and the engine control unit's configuration software was identified as the most important issue for future development of the testing system. The proposed testing system principle is probably feasible for developing any further automatic testing systems for any electric engine control unit in which fuel injection timing needs to be verified. Moreover, Compact RIO hardware and LabVIEW software can be recommended as a tool for developing similar verification systems because they are relatively easy to use, flexible, reliable, and capable of high-speed measurements.

KEYWORDS: combustion engines, four-stroke cycle engines, fuel injection engines, product development, testing, automation, injection timing, embedded system

VAASAN YLIOPISTO
Tekniikan ja innovaatiojohtamisen yksikkö
Tekijä: Tuomas Du-Ikonen
Tutkielman nimi: Engine Fuel Injection Timing : A Design for an Automatic Verification System
Tutkinto: Diplomi-insinööri
Oppiaine: Automaatio ja tietotekniikka
Työn ohjaaja: Jarmo Alander
Valmistumisvuosi: 2022 **Sivumäärä:** 81

TIIVISTELMÄ:
Tämä diplomityö kuvaa automaattisen testausjärjestelmän kehittämistä nelitahtimoottorin polttoaineen ruiskutussignaalien ajoitukselle. Ruiskutuksen ajoitusta hallitaan sähköisellä moottorinohjausyksiköllä, millä on hajautettu modulaarinen rakenne. Uusia moottorinohjausyksikön ohjelmisto- ja laitteistoversioita julkaistaan muutaman kuukauden välein. Polttoaineensyötön oikea ajoitus täytyy testata aina, kun uusia versioita julkaistaan, koska väärä ajoitus saattaa aiheuttaa moottorihäiriön. Usein toistuvan testauksen automatisoinnin odotetaan lyhentävän siihen käytettävää aikaa ja kustannuksia merkittävästi, mikä manuaalisesti tehtynä voi kestää 2–5 päivää. Työn tavoitteena on kehittää suunnitelma automaattisesta testausjärjestelmästä polttoaineen ruiskutuksen ajoitukselle.

Polttoaineenruiskutukseen ja moottorinohjausyksiköihin liittyviä tieteellisiä julkaisuja on saatavilla runsaasti. Suurin osa kirjallisuuskatsauksessa käsitellyistä tutkimuksista kattaa eri aiheita vaihtoehtoisten ruiskutustekniikoiden ja polttoaineiden vaikutuksista polttomoottoriin. Vain muutama niistä käsittelee polttoaineensyötön automaattista testausta.

Tutkimusmenetelmäksi valittiin suunnittelutiede, koska se soveltuu hyvin tuotekehitysprojekteihin. Tärkein tutkimuskysymys on: ”Minkälainen järjestelmän arkkitehtuurin täytyy olla ruiskutuksen ajoituksen testaamista varten?” Kysymyksen tutkiminen aloitettiin analysoimalla perusteellisesti eri tekijöitä, jotka voisivat vaikuttaa toteutukseen. Mikä on se perimmäinen syy miksi automaattinen testausjärjestelmä halutaan kehittää, mukana olevat sidosryhmät, vaihtoehtoiset toteutustavat sekä useita muita näkökulmia huomioitiin. Järjestelmävaatimusten määrittelyn jälkeen rakennettiin koelaite, jolla mitattiin polttoaineensyötön pulssien ajoitusta moottorinohjausyksiköstä, mikä hyödynsi National Instruments Compact RIO laitteistoa ja ohjelmistoa mikä kehitettiin LabVIEW -kehitysympäristössä. Ohjelma luo automaattisesti Excel-raportin ajoitustesteistä.

Onnistuneesti luotiin testausjärjestelmän arkkitehtuuri, mikä mahdollistaa mittausten tekemisen mistä tahansa hajautetun moottorinohjausyksikön 112 polttoaineensyötön liittimestä. Compact RIO laitteistolla tehdyt mittaukset osoittautuivat tarkoiksi ja se pystyy määrittämään kampiakselin kulman vaaditulla tarkkuudella. Testausjärjestelmän tarkkuus oli ±5 µs. Kommunikaation kehittäminen testauslaitteiston ja moottorinohjausyksikön konfigurointi ohjelmiston välille tunnistettiin kaikkein tärkeimmäksi asiaksi testausjärjestelmän jatkokehitykselle. Ehdotettu arkkitehtuuri on todennäköisesti sopiva ratkaisu automaattisen testausjärjestelmän kehittämiseksi mille tahansa sähköiselle moottorinohjausyksikölle, jonka polttoaineen ruiskutuksen ajoitus halutaan varmentaa. Lisäksi Compact RIO laitteistoa ja LabVIEW ohjelmistoa voidaan suositella työkaluiksi vastaavien testausjärjestelmien kehittämiseen koska ne ovat kohtuullisen helppokäyttöisiä, joustavia, luotettavia ja pystyvät nopeisiin mittauksiin.

AVAINSANAT: polttomoottorit, nelitahtimoottorit, suihkutusmoottorit, tuotekehitys, testaus, automaatio, ruiskutuksen ajoitus, sulautetut järjestelmät

Contents

Figures

Tables

Abbreviations

A	Ampere
AC	Alternating Current
AO	Analog Output
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
BTDC	Before Top Dead Center
CCM	Cylinder Control Module
CCW	Counter Clockwise
CI/CD	Continuous Integration, Continuous Delivery
COM	Communication Module
CPU	Central Processing Unit
cRIO	CompactRIO
CW	Clockwise
DC	Alternating Current
DEMS	Diesel Engine Management System
DI	Digital Input
DO	Digital Output
DSP	Digital Signal Processing
DSR	Design Science Research
ECU	Engine Control Unit
EFI	Electronic Fuel Injection
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
H	Henry
HDL	Hardware Description Language
HLS	High-Level Synthesis
I/O	Input/Output
ID	Identifier
IRQ	Interrupt Request

NI	National Instruments
PC	Personal Computer
PMW	Pulse-Width Modulation
R&D	Research and Development
RAM	Random Access Memory
ROI	Return on Investment
rpm	Revolutions per Minute
SAP	System Applications and Products
SD	Standard Deviation
SE	Standard Error
SoC	System on a Chip
TDC	Top Dead Center
t_i	Injection Timing
TS	Testing System
UI	User Interface
V	Volt
VHDL	Very High-Speed Integrated Circuit Hardware Description Language
Ω	Ohm

1 Introduction

Automatic testing has become a critical part of software development as systems are getting more complicated and new releases are being introduced at an ever-accelerating pace. But these frequent additions and updates to the software can possibly cause some of its existing and already tested functions to fail. Automated testing is therefore crucial for preventing this type of software deterioration, as manual testing would be too time consuming in large and complicated software projects. The development of the electronic control unit for fuel injection in four-stroke engines is crucial for effective engine operation and improved fuel efficiency being driven by more stringent emission regulation. The benefits of automatic testing for such embedded system include time savings compared to manual testing and increased testing coverage and quality.

1.1 Internal Combustion Engine Fuel Injection

The fuel injection system is designed to deliver fuel to the cylinders of an engine, and this has a significant influence on the engine combustion process. The way fuel is delivered has beneficial effects on engine performance, emissions, and can also prevent unwanted additional noise. In order to achieve effective combustion of the engine, fuel should be injected at the proper time, that is, the injection time should be accurate. The second most important factor is that the correct amount of fuel is delivered. (Khair, M. K. & Jääskeläinen, H., 2020)

Moreover, several other factors affect the good fuel injection. These include high injection pressures of 1600–1800 bar, a small nozzle area in relation to the diameter of the cylinder, and correct injection durations to ensure proper system performance. The optimal injection duration should be less than 20 ° of the crankshaft angle to attain a short burning time, which reduces NOx emissions without losing efficiency. High pressure at the beginning of the injection reduces the delay in ignition. All elements of the

fuel injection system, such as the pump, pressure pipe, fuel valve, and nozzle, should be considered when attempting to obtain optimal performance of the fuel injection system. (Latarche, 2020, pp. 333–342)

Electronic control of fuel injection is an essential component in trying to achieve the properties mentioned above. It contributes especially to timing accuracy and duration. Thus, electronic fuel injection has mostly superseded mechanical ones in modern engines. Developments in fuel injection systems have an important role in improving engine fuel consumption and reducing harmful exhaust emissions to meet ever more stringent regulation (Mitroglou et al., 2012).

1.2 Automatic Testing

As embedded systems are getting more complex, their automatic testing has become more important. It has gone from being a high-end technology practice to being necessity as it is not possible to do a great number of tests with comprehensive coverage manually, or at least not feasibly. Testers have been for decades trying to save time by applying automated testing instead of the old time consuming manual testing, by creating test cases, comparing test results to known parameters, and creating automatic test reporting. However, it is difficult to develop well implemented and working test automation successfully. (Graham & Fewster, 2012)

The growth of information technology applications in industrial use and in everyday life, has pushed an increasing number of embedded systems constantly to a faster pace. The percentage of time used for testing, as part of the whole of software development process, has increased in the last decades. For these reasons, better solutions are being looked for, and automated testing has the most potential to solve these issues. Test automation enables increased testing coverage by increasing the number of permutations and combinations tested, while reducing the time used for the process. (Dustin, E. et al., 2009)

Regression testing is a method that is used to confirm that existing software works when new components and updates are added to it. New elements in the software project could potentially adversely affect prior features in the code even if they were working earlier, thus regressing the original code. Due to the high number of tests, regression testing should be designed to be done automatically (Dustin, E., 2002). This significantly accelerates the testing, reducing the time it takes for them to only a few days instead of several weeks. In this way, it can be ensured that the software system operates according to its specifications, resulting in increased reliability (Pensar & Storbacka, 2007).

1.3 Research Problem Formulation Background

New software and hardware updates are released every few months to the *engine control unit* (ECU). Fuel injectors are wired to terminals in the ECU, which are also called *driver channels*. The correct timing for fuel injection in the four-stroke cycle is just before the piston reaches the top part of the cylinder. This timing must be accurate because the wrong timing can cause a malfunction or seriously damage the engine. Therefore, it is vital to test the correct timing with which the engine control unit sends the fuel injection pulses.

Testing pulse timing manually is time consuming and frequently needs to be done every few months. Because there are so many tests to be made, manual testing is done selectively, leaving some parts untested. Slightly different working practices or possible mistakes done by testers can also affect testing results. Automated testing is expected to speed up product development and increase engine reliability.

The object of this study is to develop an automatic system to verify that fuel injection pulses occur at the correct angles in the engine. The engine control unit has a cylinder control module that sends a fuel injection pulse based on the signal received from the position sensors of the crankshaft. The fuel injection pulse controlling fuel injection are

programmed to start and stop at certain engine angles. Before this work started, the sensor signal for the crankshaft position had already been simulated in a software and hardware setup. This could be utilized in the development of the automatic testing system.

The engine control system *input/output* (I/O) interface has been implemented in a testing system consisting of a hardware rack with all I/O and control modules in the motor with a set of software tools. The testing system can be utilized for fuel *injection timing* (t_i) verification. More accurate information on fuel injection timing can be used to experiment with different timings to test its effect on engine performance. The potential for saving costs and improved quality are the motivation for this automated testing.

1.4 Definitions of Key Expressions

This chapter defines frequently used terminology in this thesis. Some of them come from the research method used, which is design science, while others are expressions used in electronics and computer control systems. Additionally, some testing related expressions are defined. Expression is in bold, followed by its explanation.

Accuracy is an indicator of how close the observations are to the *true value*, which is an ideal case of a measurement without any error of any type. The *mean value* of an accurate measurement samples is closer to the true value than in a set of measurements that is inaccurate.

Artefact is the object of study in design science. It can have a wide variety of forms and could be, for example, a device, algorithm, process, construction method, or design logic. The effects and behavior of the artefact are studied in its environment to solve a practical problem in design science.

Precision means how close the observations are to each other. A set of observations that have high precision or measurements made with precise measurement device have low variance. It is possible for a set of measurements to be precise but not accurate, and vice versa.

Pulse in electronics and computer control systems is a short burst of electricity or electromagnetic-field energy. In this work, the word pulse is used in connection with electric fuel injector operation, which requires a certain kind of current waveform to open and inject fuel. The required current waveform changes according to the desired injection duration and fuel amount, as well as the injector type, which affects engine performance. Different current levels are obtained by varying the voltage supply using *pulse width modulation* (PWM). The nature of this electric waveform can still be seen as a short surge that occurs once per one four-stroke engine cycle.

Signal in electronics and computer control systems is an electric current or electromagnetic-field energy that carries data. In this work, the signal is used in connection with the detection of engine speed. The engine speed is detected using sensors which produce an output of a voltage waveform which changes according to the engine speed.

Testing is a more general and broader idea of the procedure, which aims to confirm that some created function meets its purpose in a reliable way. An example of testing would be to measure t_i from driver channel manually by using an oscilloscope or a smoke test of *graphical user interface* (GUI) to check that clicking the buttons in it delivers the intended result. The whole process of confirming that the fuel t_i is correct is also called testing.

Treatment is term used by Roel Wieringa in his design science framework, which refers to a proposed method, technology, or similar concept that is applied in research for the purpose of solving the problem. Using the expression *solution* is avoided because it

implies that the problem will be solved. The term *treatment* is adopted from medical science and is more objective in the sense that it comprises the idea that the proposed method may not necessarily work. (Wieringa, 2014, p. 28)

Validation is an evaluation that aims to ensure that the properties and means of some product or service will likely resolve the problem for which it is applied. The purpose of the validation process is to assess, whether the proposed solution can meet the requirements of the problem it intends to resolve, but it is insignificant by what means it is done.

Verification is a process in which it is checked whether the product under development meets its specification that is set for it. This could be, for example, checking that *user interface* (UI) matches its description, memory usage is within defined limit, and architectural design is done according to the specification.

1.5 Structure of the Thesis Work

This thesis is organized into six chapters. Chapter 2, following the introduction, contains a literature review of scientific studies and books related to automatic t_i testing. Chapter 3 describes the principles of the engine control unit with its related signals and pulses, which are necessary technologies to understand this topic. Chapter 4 introduces the research methodology. Chapter 5 contains results of the research work. Chapter 6 presents the conclusion and includes proposals for future development.

2 Related Work

This chapter covers an overview of related literature and studies on the automation of fuel t_i testing. Existing methods from a hardware technology perspective are described in the first part. The second part discusses studies that focus on various effects of different injection methods and fuels on the operation of the engine. Finally, the third part focuses on the literature that describes the guidelines and principles of developing testing automation as a process, rather than to any specific technologies that are used to implement it.

2.1 Test Methods for Fuel Injection Timing

Several studies have demonstrated that automatic fuel injection can improve the precision of measurement. Firstly, this is shown in the work by Fu and Ma who present an intelligent fuel injection control system (F. Juan & M. Xian-Min, 2009). The study uses a PC and an AT89C52 microcontroller to automatically obtain the fuel injector parameters: such as pressure, temperature, rotation speed, total oil quantity and injection oil time. Their system is implemented on an SYT240 test platform.

Similarly, measurement precision was also improved as shown in research by Du and Sterpone (Boyang Du & L. Sterpone, 2016). In this approach, a new *field-programmable gate array* (FPGA) based validation platform is used to measure fuel injection timings. The hardware chosen is Digilent Genesys Board with Xilinx Virtex-5 XC5VLX50T FPGA and a 32-bit Xilinx MicroBlaze central processing unit (CPU). This combination of two different technologies was used to obtain the angle of the crankshaft of the engine, and to generate a fuel injection pulse: They used the Enhanced Time Processor Unit developed by Freescale, and the Generic Timer Module developed by Bosch. As a result, measurement accuracy was improved by more than 20 % compared to conventional methods with a sample measurement update rate of 10 ns.

2.2 Fuel Injection Studies

Wu et al. presented a basis for the development of high-pressure common-rail injector systems (S. Wu et al., 2020). In general, the speed of the high-pressure oil pump determines internal combustion engine power, and in order to enhance its effectiveness, continual pressure in the oil pipe needs to be maintained when the pump operates at high speed. Consequently, the pressure in the oil pipe must be controlled when the cam rotates at high speed which is done by controlling the speed of the cam, adjusting fuel injectors and their t_i in the cycle. The multiple search method is applied to configure double injectors to maintain steady pressure in the high-pressure oil pipe.

The results reported by Kang et al. show that their novel high-speed fuel injection control *system on a chip* (SoC) for the *Diesel Engine Management System* (DEMS) can control fuel t_i in real time, flexibly, and with precise control (Q. Kang et al., 2017). The tests conducted in their research show that the proposed technology can operate a fuel injector well. The switching frequency can be more than 200 kHz, with the booster voltage reaching even 75 V, and the injection current can reach 16.7 A in less than 200 µs during the booster phase. As a result, engine efficiency is increased as this solution reduces both fuel consumption and emissions while delivering the same engine power.

Many studies have been done about the use of renewable fuels in internal combustion engines. Venkatraman and Devaradjane (2010) compare the performance of the diesel and *pungam methyl ester blend* (PME20) to normal diesel fuel. The objective of the study was to find the optimum control parameters for the compression ratio, the injection pressure, and the t_i. With the ideal combination of parameter values, the performance of PME20 improved significantly compared to diesel.

Most of the studies have focused on the effects of alternative injection technologies, and fuels, to internal combustion engines, but they contain only limited information related to the fuel t_i testing. At the same time, manual verification as a common approach, but which is often time consuming and has potential limitations in its accuracy.

Studies related to automatic fuel t_i testing are highly application specific, and therefore the possibility of applying the designs described in them to create testing systems for other applications is limited. This suggests that a study for the testing of fuel t_i would be a beneficial contribution to this field of research.

2.3 Principles for Developing Automation

Graham and Fewster reached the conclusion that management issues and test software architecture are two of the most important elements in developing successful test automation (Graham & Fewster, 2012, p. 1). The book, *Experiences of Test Automation: Case Studies of Software Test Automation,* describes experiences in a variety of projects. In these case studies, test automation is developed, for example, for integrated embedded systems, databases, email servers, pension and insurance client software, and web applications. Both successful and unsuccessful projects are presented, which can be used to identify common factors. Both upper-level good methodologies and, practices, as well as more detailed level scripting techniques, automated comparison, test software architecture, software tools and useful metrics are described.

A series of eight research topics from different authors about automated software testing are collected in a book by Jena et al. (Jena et al., 2020). To highlight one of them, *Test-Case Generation for Model-Based Testing of Object-Oriented Programs* used genetic algorithms, particle swarm optimization, cuckoo search, and several other algorithms to generate test cases automatically. Results were statistically compared to object orientated-based triangle classification problems. The *hybrid particle swarm algorithm,* with *gravitational search algorithm* (PSO-GSA), proved to produce the most stable and uniform test cases with a shorter computation time compared to the other metaheuristic algorithms. Although, some of the topics in the book have a limited contribution to automated testing and focus more on software development. For example, in *Object-Oriented Modelling of Multifaceted Service Delivery System Using Connected Governance,* object-orientated modelling is used to create a cloud service delivery model,

which can be utilized later in developing a complete system of secure electronic communication between citizens and different government organizations.

3 Background

This chapter covers key technologies and concepts that are necessary to understand the starting point for the development of automatic testing for the fuel t_i. The first part describes the engine control unit. In the second part, the principle of sensor signals and fuel injection pulses is explained. In the third part, an existing automatic testing framework is described.

3.1 Engine Control Unit

An *engine control unit* (ECU) is an embedded electronic control unit that handles control commands for engine actuators, as well as engine monitoring and safety. The common architecture of most ECUs includes sensors, actuators, and a microcontroller where input signals and actuator commands are processed. Figure 1 shows a simplified overview of ECU systems in general. Commonly, they have these components, but the sensor and actuator types vary by engine and fuel type. Safety functions can be isolated, and components could be redundant for increased reliability and safety.

Electronic fuel injection (EFI) reduces fuel consumption and emissions compared to mechanical injection, because it enables controlling the amount of fuel that is injected to the cylinder more accurately thus saving the fuel delivered. Specifically, the introduction of emission standards was an important cause for EFI to become more common. When early electronic fuel injection systems were first introduced by Lucas Bryce in the late 1970's, there was no environmental legislation for emissions. Although, EFI did enable heavy trucks to meet emission limits, which thus provided motivation for further investments and product development in EFI technology. Lucas Bryce, which was now called Delphi Bryce, stated that in addition to lowering emissions, EFI contributes to increased reliability. Moreover, it creates more accurate delivery and timing of fuel injection, which can be controlled and monitored to improve efficiency, and allows health monitoring for preventive maintenance. (Latarche, 2020, pp. 341–342)

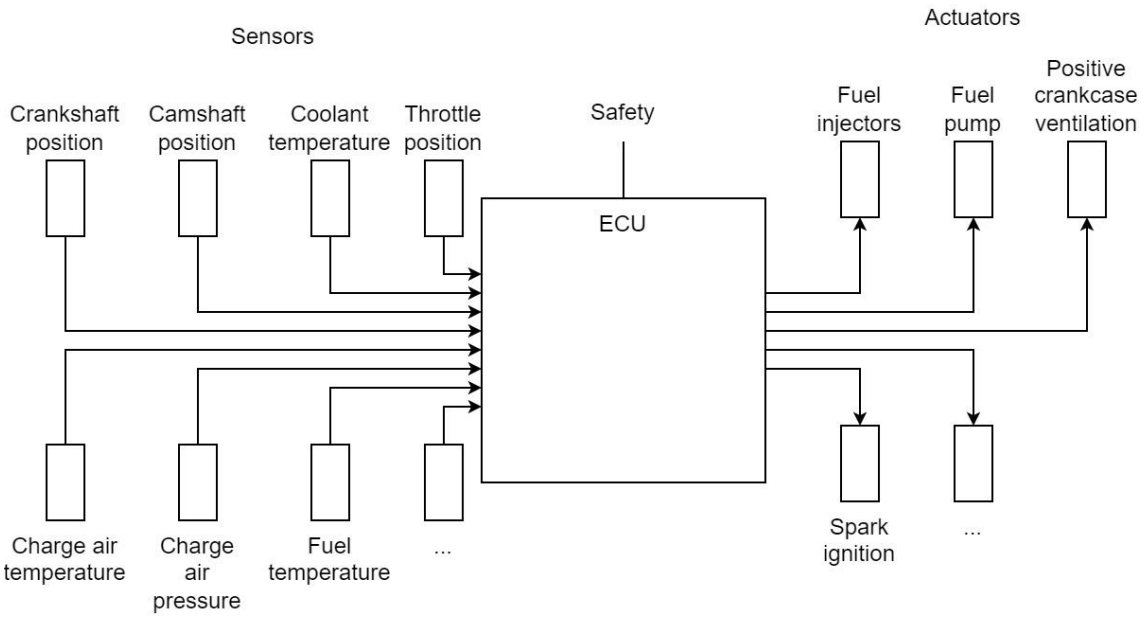

Figure 1. Simplified engine control unit overview including sensors, actuators, and safety unit

The ECU for a 4-stroke internal combustion engine, studied in this thesis is distributed to several modules that include the *cylinder control module* (CCM), the *communication module* (COM), and the *safety module*. Figure 2 illustrates the principle of distributed modular design of the ECU. Different modules are designated for different parts of the engine. Modules communicate with each other via a communication bus. This modular structure allows scaling and customization to fit various engine types. Sensors and actuators are wired to different modules rather than to a single centralized ECU. Events are time-stamped for faster troubleshooting and enable engines for enhanced efficiency. (Pensar & Storbacka, 2007)

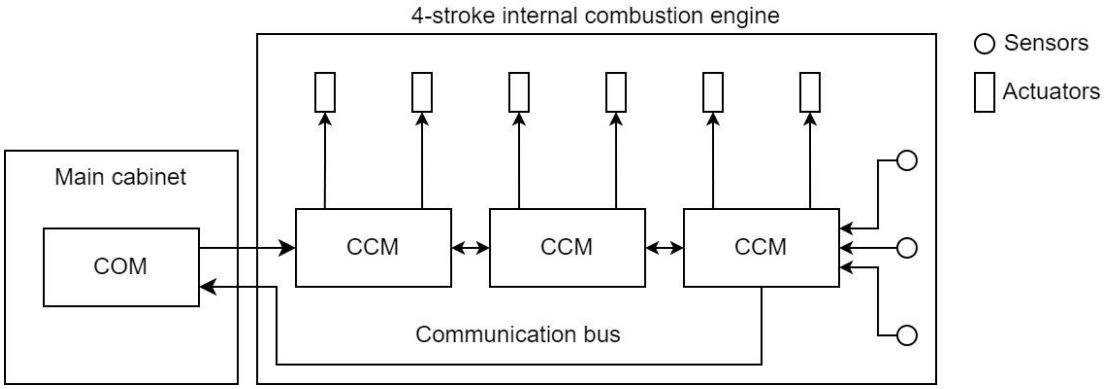

Figure 2. Distributed modular ECU design principle

3.1.1 Modules of the Engine Control Unit

Relevant engine control modules are described in this chapter from the perspective of helping to understanding this work. The modular ECU design studied in this work also includes other module types, so the following description is not exhaustive. Different modules are located in the main cabinet or distributed around the engine, wherever they are needed locally. The external engine systems and power supply are connected to the ECU through the main cabinet.

COM is located in the main cabinet and handles ECU communication between other modules, and it also provides communication to external systems. It supports multiple communication interfaces, including hardwired I/O. Engine speed and position signals are wired to COM. Engine software updates and different configurations are uploaded to the ECU via the *Ethernet* port.

The main functionalities of the CCM are the receiving of sensor signals and the sending of control commands. CCMs are on-engine modules that are positioned close to sensors and actuators in the engine. The temperature, knock, and cylinder pressure measurements are connected to the cylinder control module. CCM has 14 driver channels that can be configured to send electrical pulses to fuel injectors, gas valves, and other actuators.

3.1.2 Engine Control Unit Software

The ECU software is implemented on a modular application platform, which is itself not a software running on the modules, but a platform that is used as a base for making software that can be uploaded to the modules. The platform software architecture consists of several different layers, each of which have their own purpose. Layered architecture enables more clear software management. The goal of it is to enable the configuration of a distributed automation system.

One of the layers is an encapsulation layer that is used to hide any implementation details of accessing I/O and other hardware functions. The platform software includes implementation for communication protocols, data processing, configuration storage, I/O mappings, and other features that different systems require to control ECU modules without modifying software layer which purpose is to serve as a configurable platform for automation applications. This configurable platform for automation applications layer is accessed through an application programming interface (API) developed for it. Third important layer is a modular selection of control applications, each of which takes care of a specific control task, such as fuel t_i control, air fuel ratio control, speed and phase measurement, and ignition timing control. For example, a 20-cylinder gas engine may have more than 40 applications.

The ECU software has a complex architecture. It is important to understand that it manages engine operations through input received from sensors, such as a speed sensor, and then translates them into operations on outputs, which are actuators, such as the fuel injector. These operations include control of the fuel t_i by a table that determines the timing according to the speed and load of the engine. Depending on the type of engine, a suitable configuration of applications is selected to match the needs of that engine. This collection of applications, together with a defined collection of ECU modules and parameters, is called an engine package, which is software capable of running the actual real engine and can be uploaded to the ECU modules.

3.2 Speed Signal Measurement

Speed reference is a square wave signal generated by two twin-element *Hall sensors* that detect drilled holes in a flywheel attached to the crankshaft (Figure 3). The twin-element Hall sensor has two sensor elements inside one sensor housing. System redundancy is implemented by wiring sensors in a distributed way so that the first element of both sensors is used for speed detection information received by the ECU and the second element of both sensors is used for overspeed protection information received by the safety module. If one twin-element Hall sensor breaks, speed measurement and over speed protection will still work. The four-stroke engine cycle consists of two full 360 ° rotations (Donev & Afework, 2019). The full phase is referred to as a 720 ° cycle. To differentiate the first and second phase, one tooth of the detection plate is left empty.

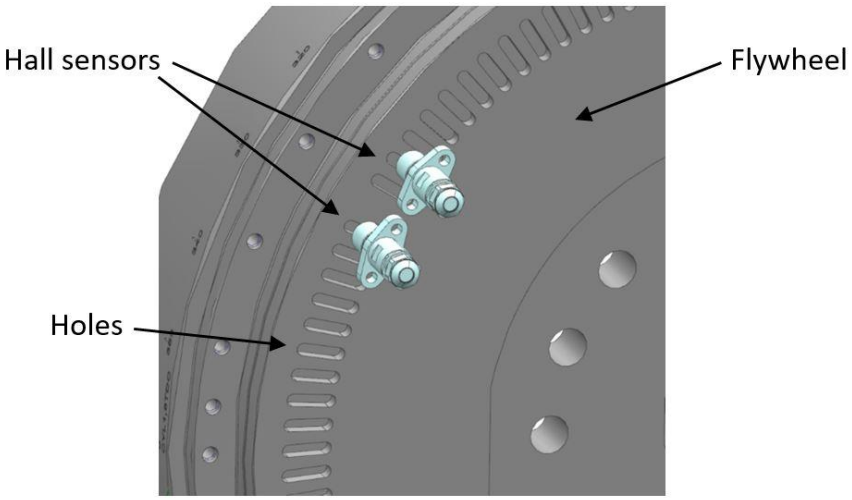

Figure 3. Installation of speed sensors

The first and second of the 360 ° phases are differentiated by two inductive phase sensors, which detect an elevated semicircle shaped disc which is installed on the end piece of the camshaft. The disc is positioned perpendicular to the *top dead center* (TDC) of the crankshaft so that the angular difference would be as high as possible to avoid measurement error, as shown in Figure 4. Two sensors are used for redundancy in the

event that one sensor fails. Phase sensor information is used only to determine the crankshaft phase when starting the engine from the missing hole position, mentioned above.

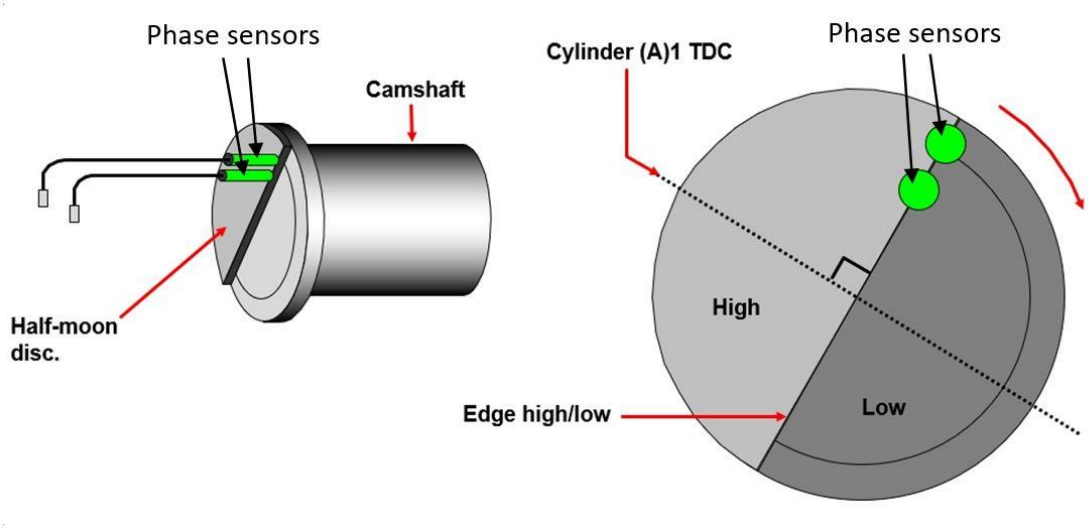

Figure 4. Phase sensor installation position principle

A precise crankshaft position in degrees is calculated from the speed and phase signals. A *0-degree* angle is specified to start from the first rising edge after the missing hole pulse. Figure 5 shows this principle; however, the graph illustration has fewer holes than a real flywheel. The standard flywheel has 120 holes.

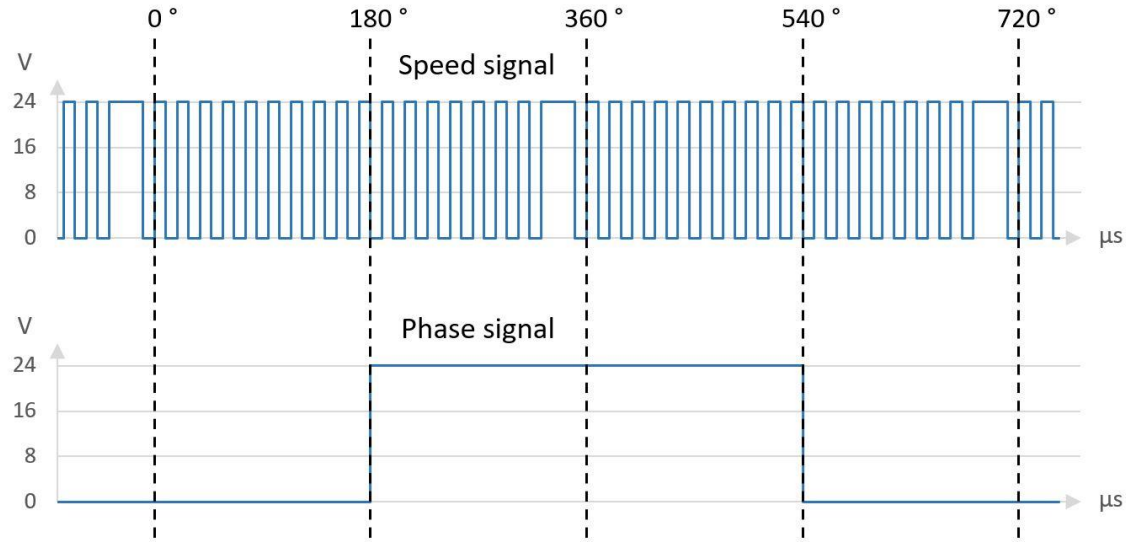

Figure 5. Two phases of a four-stroke engine cycle principle

3.3 Fuel Injection Timing

The speed signal is an input signal received by the ECU. Fuel t_i is a set time delay starting from the 0 ° crankshaft angle. In other words, when the engine is running, the t_i is the elapsed time from the zero angle of the crankshaft to the first rising edge of the fuel injection pulse voltage, as shown in Figure 6. The number of cylinders varies, and a typical engine has 6–20 cylinders. Each cylinder has individually configured timing for fuel injection, which are evenly distributed across the 720 ° cycle. Engine speed and load affect timing. The focus of this work is to verify that the actual fuel t_i matches that defined by the timing control software. As the crankshaft rotates continuously when the engine is running, the timing of the fuel injection can also be expressed by the angle of the crankshaft when the engine speed is known. Thus, the terms *fuel injection timing* and *crankshaft angle* are used interchangeably in this thesis.

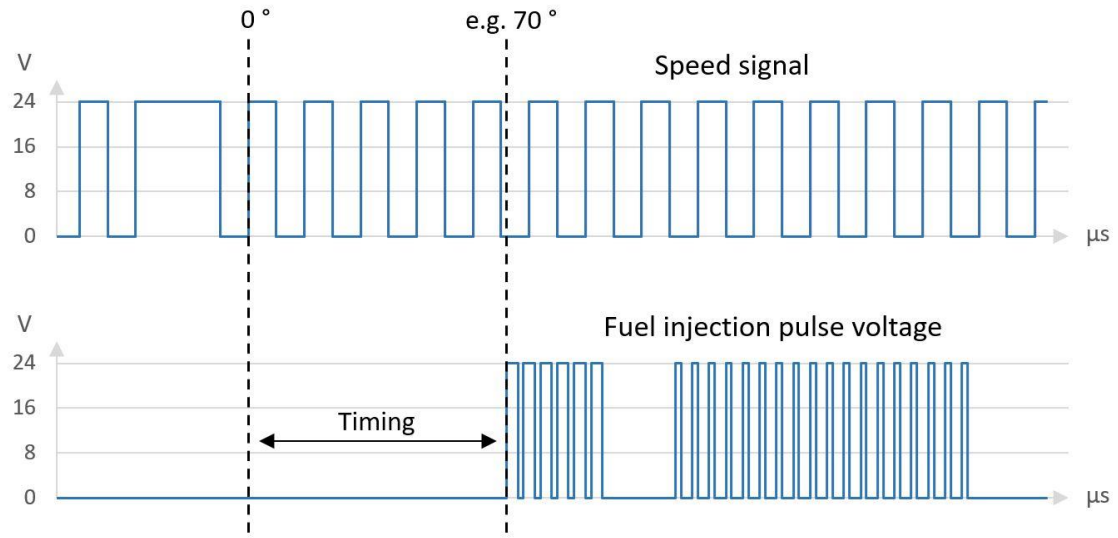

Figure 6. Fuel t_i principle

3.4 Fuel Injection Pulse Control

Timing delay is measured from crankshaft the flywheel zero angle. The fuel injection control pulse consists of two phases:

1. *Pull-in current*. The fuel injection pulse first has a higher current because the fuel injection valve requires a higher current to open.
2. *Hold-in current*. When the injection valve is opened, a lower current is sufficient to keep the injection valve open for the required time.

Different current levels are obtained using a *PWM voltage pulse*. PWM is a burst of square wave shaped voltage pulses which can be used for adjusting current level. Voltage varies between the minimum and maximum level at a fast frequency. The relationship between the minimum and maximum voltage level's duration determines the current magnitude. For example, if the voltage is in its high-level for 90 % of the time, it would result in a higher current than if it would be in high-level for only 10 % of the time. One wave cycle of the periodic waveform consists of high- and low-level phases.

The duration of the high-level phase compared to one wave cycle is called the duty cycle. The duty cycle is expressed by:

$$D = \frac{\tau}{T}, \tag{1}$$

where τ is the pulse duration in the high state and T is the duration of the cycle period (see Figure 7).

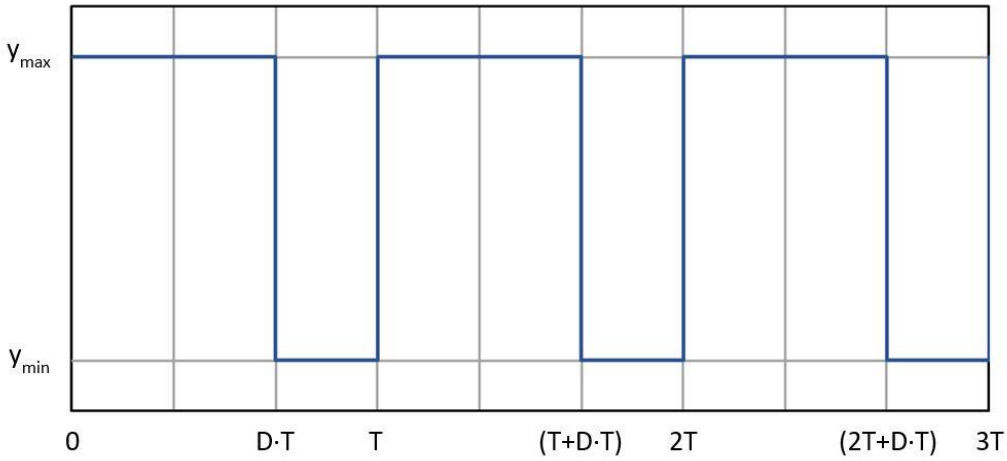

Figure 7. Definitions of y_{min}, y_{max}, D, and T in a periodic square wave

The average voltage of the PWM can be used to calculate the current. The average value of the PWM waveform is obtained by using the formula:

$$\bar{y} = \frac{1}{T}\int_{0}^{T} f(t)dt \tag{2}$$

While f(t) is a square wave pulse, its value is y_{max} for $0 < t < D \cdot T$ and y_{min} for $D \cdot T < t < T$. Therefore, Equation 2 can be expressed as follows:

$$\begin{aligned}\bar{y} &= \frac{1}{T}\left(\int_0^{DT} y_{max}\,dt + \int_{DT}^{T} y_{min}\,dt\right) \\ &= \frac{1}{T}(D \cdot T \cdot y_{max} + T(1-D)y_{min}) \\ &= D \cdot y_{max} + (1-D)y_{min}\end{aligned} \tag{3}$$

As $y_{min} = 0$, $\bar{y} = D \cdot y_{max}$. Using average voltage, current can be obtained by:

$$I = \frac{V}{R}, \tag{4}$$

where V is the voltage ($\bar{y}$), and R is the *ohmic resistance* in the circuit. Figure 8 shows the principle of typical fuel injection pulse current profile in relation to PWM, but the pulse frequency is lower for the sake of a clearer representation. The real pulse contains hundreds of PWM pulses. The current rise-up rate is faster than its decrease rate. The t_i is defined to start from the first rising edge even though the injector will open with a slight delay.

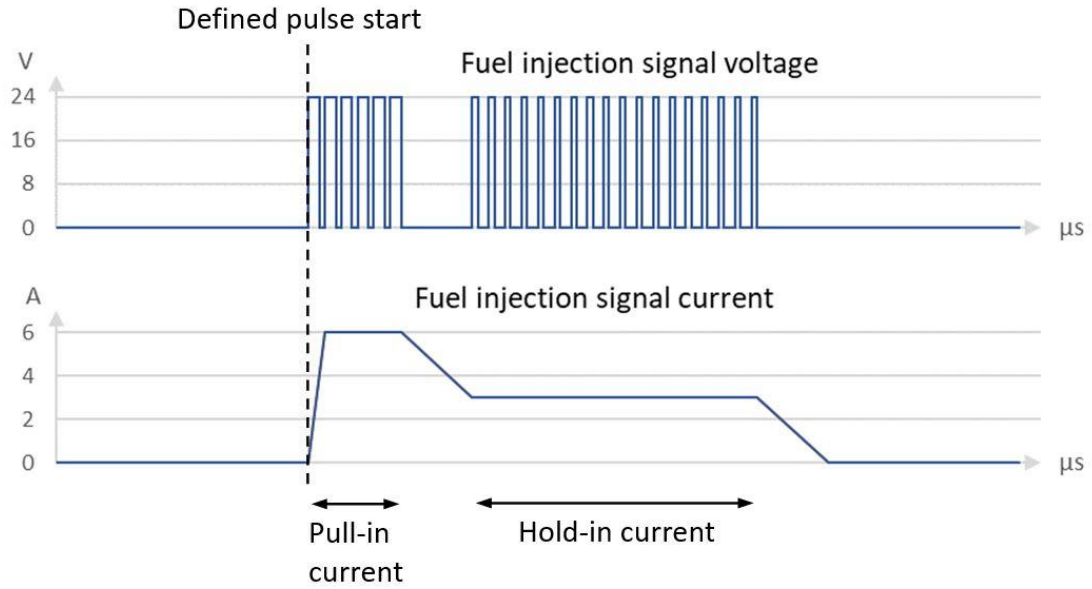

Figure 8. PWM modulated fuel injection pulse voltage and current principle

3.5 Engine Control Unit Testing System

When this work was started there was an existing system, ECU *testing system* (TS), for making various tests for the ECU. ECU TS is a set of software and hardware components that allow ECU testing without using a real engine. This enables faster and more reliable ECU testing than a manual test would. Automated test cases can be run after every release of the new program version, at night, or even after every software commit. This allows a great number of tests to be carried out, which would not be possible manually, as it would take too much time. Understanding the ECU TS architecture is important because, ideally, timing testing would be integrated into the ECU TS to achieve more unified testing practices, resulting in time savings. Moreover, ECU TS includes several useful components that can be utilized.

API provides the ability to login to the ECU configuration software tool system, read and write values, *Modbus RTU* communication, controlling signal simulator, access to hardware I/O measurements, data logging, and test rack relay control. Using this interface, developers can create test cases to start the ECU configuration tool without its GUI that connects to the ECU modules. By interacting with the ECU modules and the configuration tool, test cases can determine whether the tests fail or pass according to how test cases are designed. The ECU modules are also virtualized so that they can be tested without hardware. ECU TS can be used with virtual modules, full hardware test racks, or with a selection of individual modules.

ECU TS follows the principle of *continuous integration* and *continuous delivery* (CI/CD). CI/CD is software development practice that includes automated testing as an integral part of the development process. It is an essential part of modern software development (J. Mahboob & J. Coffman, 2021). Automatic testing is especially useful in large and complicated software projects, which are under constant development. Testing takes place on a server that has test cases for that particular software project, and every software commit is tested together with the entire software. In this way, it can be ascertained that individual code commits do not break larger software projects. Re-

gression testing re-verifies that existing and working parts of software are not negatively affected by the update and thus being regressed. Large software projects can have dozens or even hundreds of developers. The advantage of the method is that new software versions can be released more often, and even the smallest improvements could create extra value for the software. Automatic testing reduces the risk that the update causes problems later.

3.5.1 Test Rack

A *test rack* is a setup of physical hardware of engine control modules arranged to a hardware rack. The crankshaft position, charge of air pressure, coolant temperature, and all other sensors could be wired to the module inputs in the test rack, and actuator commands can then be read from the module outputs to simulate engine behavior. All I/O-points that would be in a real engine are also in the test system. The advantage of the test rack is that both software and hardware modifications can be tested without running them using a real motor, which speeds up engine development and manufacturing. However, typically ECU inputs are actually simulated using the signal simulator developed for this purpose rather than real sensors used in the engine. The signal simulator has several *analog input* (AO) and *digital output* (DO) channels that can simulate engine sensors. The output wires have been bundled to a cable harness to connect the simulator to the ECU faster. The testing system has enough COMs and CCMs for any engine type configuration.

The speed signal can be simulated in a couple of different ways. First, the signal simulator has a speed testing card that can generate a speed signal. The speed testing card can be controlled using software developed for it. Another way is to use a pocket sized speed simulator that has output terminals for speed and phase signals and a knob to adjust speed. The pocket sized speed simulator is easy to use and does not require a computer to operate it. It is used by service technicians in the field, but also by product development personnel, for quick testing activities.

Additionally, the simulated crankshaft position signal has been implemented in an external system, using *National Instruments CompactRIO* (NI cRIO) hardware, which has a DO module to send the speed signal, which has been generated with LabVIEW software. The UI can be used to change the engine speed given in *crank degrees per tick* (clock cycle). This was developed for possibly future product development purposes. The benefit of NI cRIO is that the hardware and software are highly customizable. Additional I/O cards can be easily added and software can be modified and developed.

3.5.2 Configuration Software for Engine Control Unit

The configuration software tool is used to manage the ECU software platform, which was described in Chapter 3.1.2. The ECU software platform includes the handling of fuel t_i that can be managed by the ECU configuration software tool, which can be used to adjust engine parameters, testing, monitoring, and troubleshooting. It has been implemented using a plugin architecture which provides extensibility and flexibility, as the plugins can be added or removed anytime. The configuration tool is typically used in commissioning, maintenance, and product development.

The configuration software tool runs on a PC and communication between them is implemented using a standard Ethernet connection. Different engine types have different numbers of engine control unit modules, and customized driver channel configuration. The engine configuration, which is also called the *engine package*, can be uploaded to the ECU with this software. Together with a separate signal generation device and configuration software tool, the engine can be operated. Engine parameters and sensors values such as pressures and temperatures, can be read and changed in real-time even when the engine is running. Although, the engine configuration cannot be changed while the engine is running. It would require stopping the engine, uploading a new configuration to the ECU modules, and then re-starting the engine.

3.6 Testing Specification Terminology

Each test has a specification by which it is carried out and that defines its scope. The engine package to be tested is first selected. The engine type, the number of CCMs, the expected t_i of each cylinder, and the driver channel terminals are defined in the engine package. Engine type categorizes the type of fuel used, typically diesel, gas, or dual-fuel, but other types of fuel are also possible, and the number of cylinders and the configuration of the cylinders, whether it is an in-line engine or a V-engine.

Before the top dead center (BTDC) is the four-stroke timing in the engine cycle that occurs before the piston reaches its highest part in the cylinder. Fuel is injected to the cylinder before the piston reaches the *top dead center* (TDC). Optimum t_i depends on many variables, but according to Latarche, in the case of diesel engines it is approximately 10 - 20 ° BTDC (Latarche, 2020, p. 21). To make t_i more reliable under different conditions, CCMs are tested with different BTDC values.

The *revolutions per minute* (rpm) express how many times the crankshaft rotates a complete turn in a minute, and can be understood simply as an engine's speed. When the engine is in operation, the engine speed is set as rpm. The t_i is affected by the rpm; the faster the engine runs, the shorter the t_i is. Engines are divided into three classes by their operating rpm range. Low speed engines operate under the range of 300 rpm and are often used in ships and electricity generation. Medium speed engines operate in the range of 300–1200 rpm and they are used for a wide range of purposes, including marine propulsion, generation of electricity, gas compression and pumping of liquids. High speed engines operate with over 1200 rpm and are used for transport and small generators.

4 Research Methodology

This chapter describes the research methodology that was used to develop an automatic testing system for fuel t_i testing. First, the design science research (DSR) methodology is presented, and then the application of a specific framework is described in the problem context. DSR is an iterative method that works in cycles, continuously improving the solution. The nature of this method is that knowledge and understanding about the system and environment gradually build up as research progresses. However, this chapter presents all accumulated information made during this work before presenting the final version of the proposed design.

4.1 Design Science

Design science was applied as a research method in this work because it is well suited for pragmatic product development tasks. The defining character of design science is its practicality; it is an iterative process that tries to solve and understand structures and phenomena or to prove that they have created value. The design science method works in repeating cycles where a researcher tests the developed technology, method, or similar concept with the intension to solve some problem in practise. Based on the results of the test evaluation, the researcher continues to cycle the design process by improving the design and testing it again, until a good result is achieved. Originally, design science applications were used, especially in technical fields. An important part of design science is the concept of artefacts, which are the objects of the research. They can have a wide variety of forms, such as a product, a software, a working practice, or a theory, to mention a few. In the scientific community, this methodology is relatively novel (Papalambros, 2015, p. 1).

Design science is a good research method for this study because it is commonly used in pragmatic technical development tasks. It is often applied especially in computer science and engineering. Typically, natural sciences research aims to create theories

about phenomena in the universe, whilst design science endeavours to contribute to developing the world (Johannesson & Perjons, 2021, p. 1). This assignment started from the need to improve driver channel testing by automating it. Design science aims to evaluate the usefulness of a developed solution. Its results can be used in decision making as to whether further development is beneficial.

4.2 Research Framework

Different design science practices vary slightly. This study uses the research framework based on the book *Design Science Methodology for Information Systems and Software Engineering* (Wieringa, 2014). The design science method proposed by Wieringa has two main parts: the design cycle and the empirical cycle. The design cycle is practical engineering work that often aims to create a product or software, but the result could also be a diagram, method, practice, or similar concept. In this thesis, the design cycle creates a design for an automatic testing system for fuel t_i verification. As part of the study, a device and software were created to identify possible practical problems in the design and to test the accuracy of verification using signal processing methods. The empirical cycle produces information about the research. It starts with creating questions about the important issues that we want to know about the object of the study. The nature of the questions can be qualitative or quantitative.

The design cycle and the empirical cycles are run in parallel and iterated several times during the study. Both cycles interact with theories that consist of conceptual frameworks, and theoretical generalizations. In this thesis, the most important conceptual framework is the ECU TS, because the verification of the fuel t_i is most likely to be integrated with it, and it utilizes the useful features and API already existing in it. Theoretical generalizations applied in this thesis include signal processing techniques, PWM method principle, and basic electrical engineering routines like Ohms law, and mathematical unit conversions. The best practices in automation and software engineering development are also applied, even if they are not necessarily recognized as scientific

theories. The design science method by Wieringa is summarized in a diagram in his book, which is adapted to this work in Figure 9.

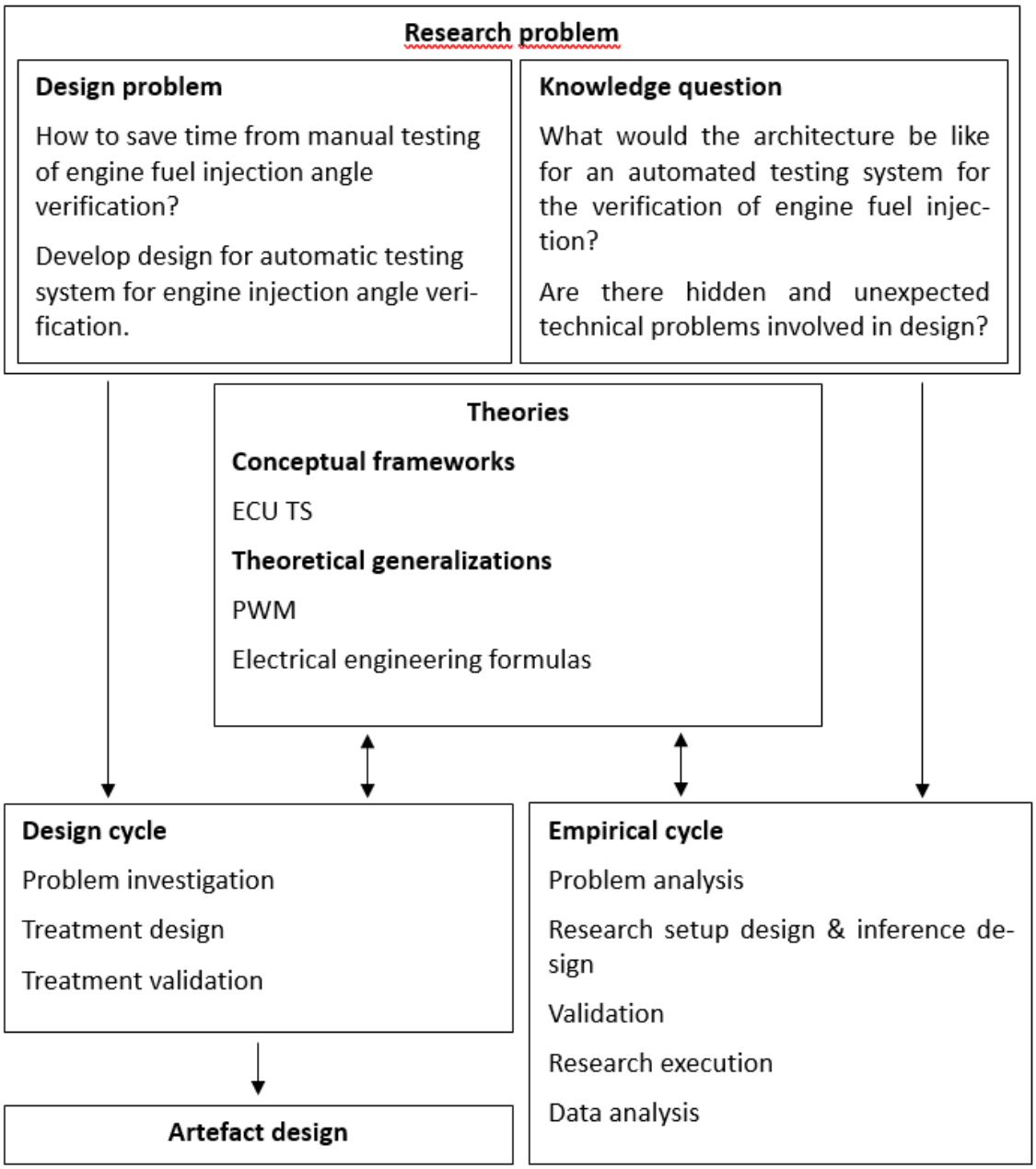

Figure 9. Design science framework for developing an automatic system for fuel t_i testing (adapted from Wieringa, 2014, p. 216)

4.2.1 Design Cycle

The design cycle aims to solve issues that arise from the design problem. The design cycle has a structure on how the problem-solving trials progresses, but it does not describe how to manage the process of engineering. The engineering process varies between different disciplines such as software engineering or mechanical engineering, and the most suitable method also depends on the problem at hand. Few examples of software engineering methodologies are the agile and waterfall model. The design science framework proposed by Wieringa introduces a design cycle that is divided into three parts (Wieringa, 2014, p. 27):

- Problem investigation,
- Treatment design, and
- Treatment validation.

These steps are repeated several times in design science research, with the purpose of each time improving the proposed treatment in a consistent and systematic way. Hence, the design is gradually improved in an iterative cycle. Iteration rounds are repeated until all features of the system requirement specification are met. The result of the design cycle after iterations is the artefact design that can be implemented. What implementation means depends on the artefact, design goal, and engineering discipline, but in automation technology, it could typically mean programming software and building a hardware setup.

To evaluate the viability of the proposed treatment, it must be validated. The chosen validation method was Expert opinion. This means that the proposed design is shown to testers, developers, and other qualified people who have expertise in the area of the problem to be solved, and then feedback will be collected. In the validation meeting, there are two things that must be done:

- Confirming that effects satisfy requirements, and
- Agree on that the treatment design is most likely to produce the desired effects.

The design cycle of this research was carried out in three iteration rounds. Feedback from experts was collected at the validation meetings. The experts chosen for the validation were developers working in the research and development (R&D) department, and some of them had experience with manual testing. The treatment design was presented to the experts at each meeting, which was then followed by discussion, improvement ideas, and additional questions. On the basis of this feedback, the design process returns back to the problem investigation. In total, three different versions of the documented design proposals were made on the basis of these meetings. After the third iteration round, it was predicted that the design would probably lead to a result that would meet the system requirements, and then the next stage of the building of the testing system began.

4.2.2 Empirical Cycle

The purpose of the empirical cycle is to produce information on the object of the research, and to answer the questions that arise from the topic. Despite the organized structure of the design science framework presented in Figure 9, the empirical cycle does not have a strict sequence of how it should be conducted. However, the framework provides questions and several suggested ways to conduct research that help to produce knowledge about the object of the study. The empirical cycle is carried out concurrently with the design cycle throughout the research rather than completing them one after another sequentially.

Knowledge questions are the central part of the empirical cycle. Since the research context is utility driven in this work, the knowledge goal is to find useful information for practical problems. Knowledge questions are presented in the following chapter after presenting the design problems. The goal is that the answers to the knowledge questions could be generalized to other similar engineering problems. Generalizations are fallible, so it is possible that the treatment application does not produce desired results when applied to other problems. However, the benefit of generalization is that,

when they are applied successfully, it leads to time savings when a good practice can be copied elsewhere. All knowledge questions in this work are qualitative by nature. The case study was chosen as the research setup for this work.

4.3 Research Problem

The research problem includes two parts in the DSR approach proposed by Wieringa: a design problem, and a knowledge question. The design problem is a practical engineering part that aims to create value by usually creating software, device, a design, or similar, which is called an artefact in DSR. For a design problem, there are possibly several ways of implementation that could bring the desired outcome. The stakeholders evaluate whether the outcome delivers the intended effect. The knowledge question is part of the research problem that creates knowledge about matters of interest. The DSR can even have several knowledge questions. But ideally, for the knowledge question, there is only one answer, but in practice, it can be subject to fallibilism. There could be uncertainty in some measure in the answer, the answer could be partial, or it could be correct in most occurrences but not in all, for example. Answers to knowledge questions are evaluated by truth, not by stakeholder goals. (Wieringa, 2014, pp. 4 - 6)

4.3.1 Design Problem

New software and hardware updates are introduced to the ECU several times a year. In order to verify that new updates do not cause any problems, they have to be tested. Untested driver channels could cause serious problems in engine operation and safety. At the moment, testing is done manually using an oscilloscope. The problem is that this is very time consuming, taking perhaps 2–5 days and is considered a monotonous and dull task. Developer engineers in the R&D department have reported that they prefer to be doing more productive work.

The purpose of testing is to confirm that the embedded system of the ECU, which is a combination of software, electronics, and I/O peripherals, works as intended and enables reliable and safe engine operation. Finding issues as early as possible is therefore important, so that fixing and solving problems in the engine production saves costs the earlier they are identified. Automatic testing is expected to enable regression testing to confirm that all system operations remain functional, accelerates the development cycle, and brings an ability to release faster on demand. Many other parts of software development testing, other than fuel t_i, have already been implemented in the ECU TS, and ideally timing testing could be integrated into the existing testing architecture for more unified testing practices.

While considering the broad view of the purpose of automatic testing, the final objective of this work is to be able to build the best four-stroke engines as possible. From that perspective, firstly, the advantage of automatic testing is the increased reliability of the motor, as a result of the increased test coverage, more possible problems are identified and fixed. Secondly, the lead time of manufacturing is decreased, as the ECU testing is done faster. Thirdly, the benefit for research and development is that more accurate information on the timing of the fuel injection pulse can be used to experiment with different timings, possibly leading to improved energy efficiency, as well as other benefits, such as increased power, and reduced exhaust fumes, and emissions.

From the viewpoint of testing activities, automation reduces costs, as it is expected to save time compared to manual testing. The quality of testing is also improved because the automatic testing system will trigger to the fuel injection pulse more consistently compared to a manual tester, which is affected by random measurement errors, e.g. due to manual oscilloscope adjustments. During initial discussions at the start of the work, developers noted that doing manual testing is viewed as an uninteresting assignment and they would prefer to use the time doing something more useful. Automating testing can possibly increase the attraction of work. Also, to some degree, vary-

ing testing practices and manual testing activities, which may sometimes involve less experienced developers, which can cause test results to be inconsistent.

Wieringa introduces a template for a design problem or a research question that can be condensed into a single sentence (Wieringa, 2014, p. 16). The following *italicized* phrase below, applies that template to this work. The purpose of the template is to assist in designing the artefact that would solve the design problem by defining its goal. Defining the goal and requirements of the stakeholder can also help to identify possible missing pieces of information that are needed to create the artefact that would create value in its context. The term *'injection angle'* in the sentence refers to the crankshaft angle when fuel injection occurs in a four-stroke engine cylinder.

Improve the engine control unit by designing an automatic testing system for the verification of the engine fuel injection timing that satisfies the requirement of measuring the injection angle of an engine by using one of the latest engine control unit configurations in 720 ° four-stroke cycle with the accuracy of 0.1 ° and produces a report that shows a comparison of the actual injection angle to the expected angle in order to save work time for the developers of the research and development department and improve the reliability of the engine.

The proposed solution to this problem is to develop a design for an automatic system to verify that fuel injection pulses occur at the correct engine angles. The system here means a combination of hardware and software, which can send a simulated engine speed signal to the ECU as an output signal and measure the timing of the fuel injection pulse in relation to the angle of the crankshaft as an input. The accuracy of the driver channel tester will be checked by comparing the results with the manual oscilloscope measurement. This kind of system is capable of fulfilling the requirements defined in the template. The sentence in the template could be formulated as a design problem: “How to save time from manual testing of engine fuel injection angle verification?”.

4.3.2 Knowledge Questions

Wieringa explains that motivation for knowledge questions could be curiosity, but the motivation can also be utility driven. Knowledge questions do not necessarily aim to solve some practical problem nor are they evaluated by stakeholder interest. Wieringa classifies knowledge questions into two categories: *analytical and empirical*. Answers to analytical questions describe conceptual frameworks of the world which can be created by conceptual analysis like mathematics or logic, while answers to empirical questions require collecting data and analysing it. Empirical questions are further classified as *descriptive* and *explanatory*. The goal of descriptive questions is to describe phenomena and explain them, while explanatory questions aim to answer why something happened. One way of creating knowledge questions is to first find out design problems and then to form knowledge questions based on those problems. (Wieringa, 2014, pp. 6–18)

This guideline was used to formulate knowledge questions. As the design problem is now defined, two knowledge questions arise from it:

1. What would the architecture be like for an automated testing system for the verification of engine fuel injection?
2. Are there any hidden or unexpected technical problems involved in the design?

Motivation for the first knowledge is utility driven and the question aligns with stakeholder goals. The answer to the first knowledge question could be used to design an automatic testing system. The result of the first knowledge question will be a diagram. It will be formed as an outcome of iterative cycles in the design phase.

The second knowledge question is also utility driven. This question aims to identify the detailed practical problems involved with the implementation of the design. If the automatic testing system proves to be too expensive or time consuming to build, and the drawbacks are greater than the benefits, the design should be abandoned. Any unex-

pected problems should be remarked if they should arise during the development process.

4.4 Problem Investigation

4.4.1 Issues Affecting the Design

To create a treatment in an analytical and critical way, several issues were considered. Different factors need to be looked at, because they can possibly have an effect on how the artefact is designed. The motivation for developing an automatic testing system is that it is expected to save time for developers. Furthermore, it increases test coverage because it enables more frequent and easier testing. The research and development department benefits as developers have more time to do more productive tasks.

The stakeholders involved with the testing system are ECU developers who are occasionally assigned the task of testing the correct timing of fuel injection pulses. This is a very time-consuming task, and they wish that this testing could be automatized in order to save time in favor for more interesting and productive tasks. Manual testing starts by assigning the specification for the testing task in a meeting. The ECU configuration to be tested is defined at that point, as well as the scope of the test parameters, including the rpm and BTDC values. Depending on the type of engine, the configuration typically has 6–20 fuel injection channels in it. Driver channels can also be configured for other types of pulses than fuel injection, for example, spark ignition or intake and exhaust air control valves. Sometimes, testing of these pulse types might also be required, which significantly increases the number of measurements to be made. For instance, a 20-cylinder gas engine has in total 44 driver channels configured.

According to the recommendation of the test guidelines, CCMs are tested with three different rpm and BTDC values in order to ensure that the timing works correctly in all

conditions. This multiplies the number of individual measurements by nine. If every driver channel from the 20-cylinder gas engine is tested with three different rpm and BTDC values, it means in total 396 individual measurements. Fuel injection or any other type of pulse timing is then measured manually using an oscilloscope and positioning two cursors on display to measure timing the difference of crankshaft zero angle to the defined pulse measurement point. Changing the BTDC values requires stopping the engine and reuploading the ECU software to the modules. Each upload takes a couple of minutes, which naturally slows down the testing process. The great number of measurement points combined with the manual use of an oscilloscope and the required reuploads makes manual testing costly in terms of time.

Because of the high number of measurements to be made, manual testing is done by taking samples only. For example, one test report made manually had all cylinders tested from each of the three rpm and BTDC value ranges once, thus completing 3 out of 9 of the possible combinations. This reduces the coverage and quality of the testing, because it is possible that some erroneous timings will be left undetected. The benefit of automatic testing is that all measurement points can be easily tested.

When designing the testing system, it should be considered what its intended purpose is and how to design it in a way that it would perform as well as possible in that context. Also, criterion to evaluate the usefulness of the artefact compared to manual testing must be defined so that comparison can be done objectively. These non-functional properties are sometimes called *quality properties* (Wieringa, 2014, p. 54). The testing system compares the set crankshaft angle with the crankshaft angle measured from the fuel injection pulse and checks if the timing is within acceptable limits. The performance of the automatic testing compared to the manual testing is evaluated in three ways:

- The duration of completing tests,
- The quantity of tests, and
- The accuracy and consistency.

Weaknesses of manual testing includes it being slow and prone to possible mistakes and human errors. While on the other hand, its strengths include that testers have the opportunity to develop insight to better understand testing and test results. They could be able to tell faster which problem is only a glitch or interference and which one is a real problem. They could understand the purpose of testing and then focus the tests on the relevant parts. On the other hand, automated testing is faster, more consistent, offers better coverage, and enables automatic reporting.

The effect of the artefact outside of its intended context is also considered, because its value could be higher if it could be utilized elsewhere. Also, possible risks that it could pose have to be taken into account too. Automated testing is probably more or less useless outside of the R&D department. The distributed modular design and specific properties of the signals and pulses of its I/O make it a highly specific artifact useful only for this particular ECU design studied in this work, although some upper-level design principles could possibly be applied to other ECU types. Engine control modules are, of course, an important part of the engine, and that without them, could not even run. Indirectly testing should improve engine reliability overall, and therefore it affects anyone who is using the engine. It is unlikely that the testing system would have any negative effects outside of its context, since probably it cannot be misused in any harmful way.

The change in the size of the context in which the artefact is used needs to be taken into account, because it could have an effect on the way it is designed. The testing system context is the R&D developers, and the more users it is expected to have, the more efforts are needed in the development of its usability and documentation. R&D department size could possibly grow or be reduced. Developers of R&D have high technical skills levels, and they are able to use complex systems with less instructions and guidance, but ease of use is beneficial since it lowers the learning curve and speeds up the use of a testing system. If an automated testing system is well developed and easy to use, the size of the R&D department should not matter much. In other words, de-

velopers can be expected to benefit from the testing system, regardless of the size of the organization.

Artefacts interact with their environment when they are applied for their intended use. Technical functional requirements have to be defined because they set constrains on how the details of the artefact are designed so that it would fulfil its purpose in its context. For example, the speed reference, sent by the testing system, has specific properties so that the engine control unit can read it correctly. It has a square wave signal with a specific frequency and voltage, and with the missing hole signal showing the change of the 360-degree period. It also assumes that the information about the expected timing of the fuel injection pulse is available. The system also expects that the fuel injection pulse has a certain voltage profile.

The testing system has the following requirements. The nominal speed of the engine is 750 rpm, but the engine control units are rated up to run up to the maximum speed of 1500 rpm. Therefore, 1500 rpm is used to calculate the required update rate. The angle of the crankshaft must be measured with an accuracy of 0.1 °. The required pulse measurement update rate is obtained using the formula:

$$\mathrm{T} = \left(60 \cdot \omega_{\mathrm{rpm}}\right)^{-1}, \tag{5}$$

where T is the update rate in seconds and ω_{rpm} is the rotational speed in rpm. This would give an update rate of 11 µs at a speed of 1500 rpm. The update rate at 750 rpm would be 22 µs. Nonfunctional requirements and properties include:

- Measurement has to be done using one of the latest ECU configurations,
- The accuracy of the crankshaft angle must be 0.1 degrees, which requires taking samples every 11 µs at 1500 rpm, and
- Automatic test report generation.

Ideally, the treatment design should cover all these requirements. Whether the requirements are met or not is evaluated in treatment validation. In the validation, the proposed artefact is investigated and compared with the nonfunctional requirements and properties. On the basis of this, a *design theory* is developed. Design theory means an event where the artifact interacts with its context. Using design theory, it is predicted whether the artefact would meet the requirements proposed for it if it were implemented. (Wieringa, 2014, pp. 31–44)

Now that the design problem and the requirements of the system have been defined, it is possible to formulate the treatment. Because of the constraints set by nonfunctional requirements, neither of the two following ideas seem viable, the first being a non-technical solution like improving the manual testing working methods, or by accepting the proposal to not verify the t_i. Thus, the proposed treatment is to design a hardware system. The timing of the fuel injection pulse with respect to the angle of the crankshaft can be measured with the system setup presented in Figure 10. The crankshaft position signal simulation generates a speed reference with the specific properties mentioned in Chapter 3.2. The speed signal is received by the COM, which further communicates with the CCM, which generates a timing for the fuel injection pulse. This pulse is then received by measurement verification, and then timing can be determined as described in Chapter 3.3 by the pulse verification system.

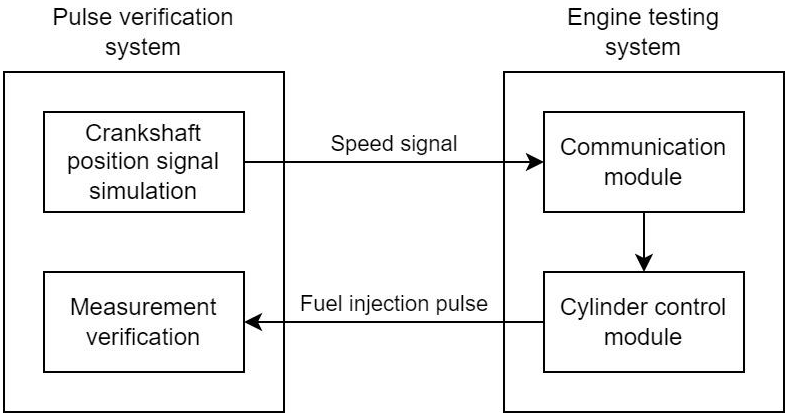

Figure 10. Pulse verification system principle

4.4.2 Hardware and Software Selection

There are several different hardware options that can achieve the functionality described above. The initially proposed platform alternatives on which to create the pulse verification system were a National Instruments CompactRIO controller, a LeCroy automated oscilloscope measurement, including other available products from the same supplier, or a PC based I/O card. Several NI cRIO hardware units already existed in the R&D department, and there were personnel experienced in using them. These different options were then compared.

CompactRIO was considered the best option, because, from earlier test systems, there was already an existing software for simulating the crankshaft position. Also, an associated LabVIEW software can be programmed to meet the needs of this application. LabVIEW is an established software for scientific research involving ECU systems. Some examples of studies using LabVIEW are:

- *The design and statistical validation of spark ignition engine electronic control unit for hardware-in-the-loop testing* (V. Vasquez Lopez et al., 2017)
- *Time-division multiplexing based system-level FPGA routing for logic verification* (J. Zhou et al., 2010)

- *Development of engine control technique for flex-fuel motorcycle* (A. Keawtubtimthong et al., 2010).

LabVIEW is a graphical programming environment, unlike most programming languages that are text-based, such as *C*, *Java*, and *Python*. When comparing graphical user interface to structured text base, it has similar programming elements in it from selection structures to loops and functions, but it requires some time to get used to it if a developer only has earlier experience just from text-based programming languages. LabVIEW programming is based on the idea that GUI and graphical source code are developed at the same time. The front panel or dashboard, which is the GUI part, and the block diagram, which is the source code part, are open in separate windows at the same time. In LabVIEW, *controls* are equivalent to input variables, and *indicators* are equivalent to output variables. When controls are placed in the front panel, they also appear in the block diagram, where the programming logic of how the input variables are translated to the output variables can be developed by using programming operators and functions.

The advantage of graphical programming, especially to beginners and experienced users in simple cases, is that the code can be faster and more intuitive to read and understand than text-based programs. On the other hand, larger and complicated code can be more difficult to understand and manage, because of the concurrent computing principle in which several computations are happening simultaneously, both in programming logic sense and in the CPU computing level, which can affect the states of other pieces of code. This overlap can possibly deteriorate code readability and cause difficulties in debugging. In complex cases, text-based programming code may be easier to understand because a program is implemented one row of a code at a time. The differences of comprehending graphical or text-based programming code are a matter of habit and varies from case to case.

The products of National Instruments and LabView software are suitable for laboratory tests that allow the development of an application in a short development cycle. It has a good interface for developing GUI, and as a globally popular and well documented product, and solutions to problems could be found easily (J. Zhou et al., 2010). The I/O modules available for CompactRIO have a sufficiently higher sampling rates for this use case. The product portfolio has controllers and data acquisition hardware with exchangeable I/O card slots, which enables later modifications and expansions to be done more easily. NI products typically support various communication interfaces, which in turn support compatibility with other devices.

The cRIO-9104 chassis includes an FPGA which excels in *digital signal processing* (DSP) applications with time critical requirements. The chassis can be equipped with a controller that includes the CPU. CPU is more suitable for non-time critical computations, which require handling floating-point arithmetic, and it has reliable and predictable behavior. Software architecture should be designed in a way that the most suitable target, either CPU or FPGA, is used for the task which it is best suited for.

FPGAs are integrated circuits similar to *application-specific integrated circuits* (ASIC), but FPGAs allow their circuits and logic connections to be changed, thus making them more flexible in their usage. FPGAs consist of a matrix of logic blocks which can be connected to a circuit to create the desired output. Whereas in contrast, the circuit and logic of CPUs are fixed and software is used to create the desired output. In FPGA, the output is created by wiring logic blocks in a process called *synthetization*. When the use of FPGAs started to increase, the first FPGA configurations were created by using schematic capture software. The next step of configuration tool development was *hardware description languages* (HDL), such as the *very high-speed integrated circuit hardware description language* (VHDL) and Verilog. In this thesis, a *high-level synthesis* (HLS) tool NI LabVIEW FPGA is used to create FPGA configurations. The HLS tools are the next level up from the HDL tools in FPGA configuration tool development. FGPAs have many applications, one of which is that they are used to prototype and verify the

design of ASICs. There are several benefits with using FPGAs, including programmability, low latency, parallelism, and high throughput. (T. Stratoudakis, 2021, pp. 16–65)

PC applications based on a general purpose operating system could start running some background applications, which would allow IRQ to delay pulses coming from the cylinder control modules, and thus not providing accurate enough measurement for the timestamp. With a cRIO controller combined with an FPGA chassis, CPU system processes and pulse input measurements are running separately. The pulse measurement is made accurately in a time window of microseconds in each loop. The pulse input is wired directly into the FPGA to eliminate the latency caused by the communication bus. (National Instruments, 2014)

Based on the above remarks, a suitable hardware setup was selected. It was based on an existing speed simulator setup available in the R&D department, with some modifications. For this purpose, a new DI input module was selected that has a sufficient sampling rate. The DO module that sends speed and phase signals from the original setup was kept, and the extra modules used in previous tests were removed. The chosen hardware is listed below:

- NI cRIO-9014 controller
- NI cRIO-9104 chassis
    - NI 9474 DO module
    - NI 9425 DI module

The NI 9425 DI sampling frequency of 7 µs meets the minimum requirement of 11 µs. Setup can be fitted on a tabletop in the laboratory, and the necessary I/O can be wired to the testing rack or to selected individual ECU modules. The physical size of the hardware is suitable to be integrated as part of testing rack later.

The proposed implementation is a realistic way to reach the problem goals. The design cycle includes a budget requirement set by the stakeholder, who acts as a sponsor of

the work. The cost of setup is within the budget of the R&D department, and the building and programming can be done in a few work months. The treatment is not limited by the accessibility of technology or the nonfunctional requirements and properties set to it.

4.4.3 Automating Measurements from Several Driver Channels

Timing measurement must be done using a large number of driver channels. The test rack has 8 cylinder control modules with 14 driver channels each, which together adds up to 112 driver channels. Figure 11 visualizes the number of all the possible measurement points. The big boxes represent physical CCMs, while the smaller boxes are driver channel terminals, which are wired to the fuel injectors in the engine.

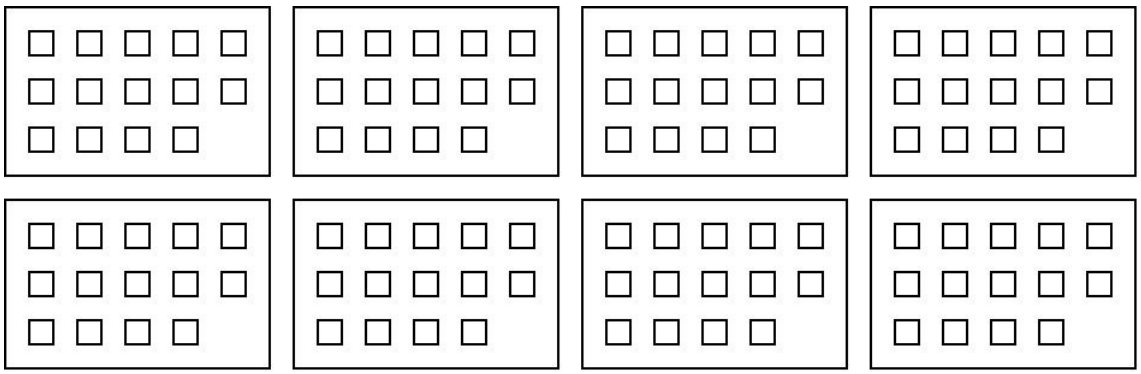

Figure 11. Visualization of the total number of driver channels

The number of configured driver channels varies according to engine type. E.g. a 6-cylinder inline diesel engine has 6 driver channels configured for fuel injection (Wärtsilä, 2022) and a 16-cylinder dual-fuel V engine has 16 driver channels configured for fuel injection (Wärtsilä, 2019). The driver channels are distributed to different modules. Figure 12 shows a simplified example of the driver channel assignment, where the number of modules and terminals has been reduced for the sake of a clearer representation. Four big boxes in each configuration represent CCMs, and the small boxes inside

them represent driver channels. The black box is an assigned channel, and the white box is an unassigned channel.

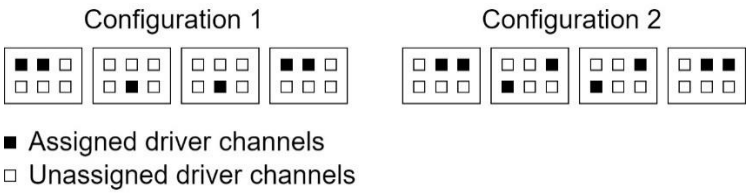

Figure 12. Example of driver channel assignment

While the number of CCMs, assigned channels, and their location changes by configuration and engine type, a full 8 CCM setup in a test rack is enough to cover all the possible combinations. In order to test driver channels manually, a tester starts the engine in a simulated state where the ECU receives speed signal, throttle position, and all sensor data, and the schedules fuel injection pulses as if a physical motor would be running. Then the tester starts to measure the assigned driver channels using an oscilloscope and records the injection pulse timing. Different rpm values can be tested by changing the rpm setting while the ECU is running, but testing different BTDC values requires stopping the engine, uploading a new ECU configuration, and restarting the ECU.

Driver channels are optionally used for other purposes than for fuel injection. The process of pulse timing measurement should also consider the pulse type. The scope of this thesis was limited to testing of fuel injection pulses, but ideal testing automation should also have the option to expand testing for other pulse types later. The pulse types of the other driver channels are:

- Ignition channel,
- Reset channel,
- Intake control valve,
- Exhaust control valve, and
- Multifunctional injection

The mechanism for measuring different types of pulses varies. For example, the ignition channel controls a spark plug that ignites a fuel-air mixture in the cylinder, which transforms into the kinetic energy of the crankshaft (Eteläpää, 2021, p. 23). The ignition channel is wired to the primary winding of the ignition coil. A spark is generated after the falling edge of the ignition channel pulse (Knol et al., 2022 pp. 8–10). When the ignition channel is tested, the timing of the falling edge is measured, whereas the rising edge is measured in the case of the fuel injection test.

When all pulse types of the driver channels are included, the number of channels to be tested increases significantly. For example, a 20-cylinder gas engine uses 44 driver channels. Therefore, all 112 driver channels could potentially be the subject of testing. Test automation should have the means to test any driver channel, since the number of possible measuring points cannot be restricted.

To measure the timing of the driver channels, it would require wiring them to the *digital inputs* (DI). Two options for this are shown in Figure 13. The first option on the top is to use a single DI, with 112 relays to switch measurement points. This multiplexing principle would save DIs and loads. There needs to be a load that simulates the injector to create a fuel injection pulse. The downside of this approach is that it slows down the measurement. The engine must rotate 112 times at minimum, while the relays open and close after each rotation, to complete the measurement in all channels. In practice, testing is done by letting the engine run several times and observing that the fuel injection pulse is behaving consistently. Another option shown on the bottom is to use 112 DIs. The downside of this approach is the large number of DI points. The benefit is that the testing can be done faster. This setup can also be used to measure missing pulses and extra pulses.

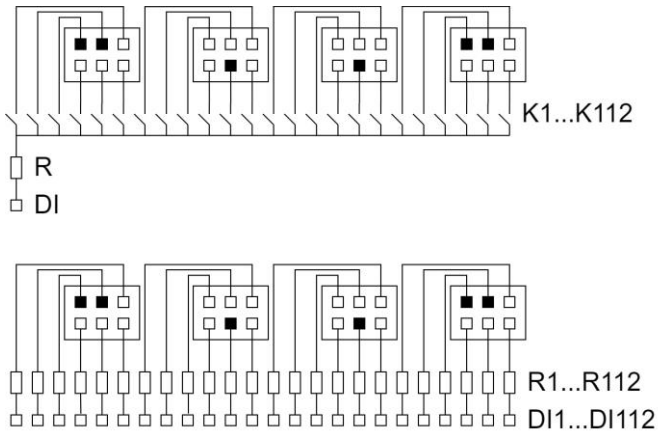

Figure 13. DI options for measuring pulses. *K* is a symbol of a relay contact, *R* represents a load, and *DI* represents the digital input terminal in the figure.

Another option is to measure all channels from one module and swap *identifiers* (IDs). It is possible to change the IDs of cylinder control modules using the ECU software configuration tool. By swapping IDs, the driver channels can be tested from another module even if the measurement is made physically by the same module. This is a compromise in terms of the number of DI points. However, this requires stopping the motor, uploading a new configuration to the modules, and starting the motor again. The testing system needs to communicate with the ECU TS which provides API for these kinds of actions. This swap method is illustrated in Figure 14. Again, the number of modules and I/O points is reduced for the sake of a clearer presentation.

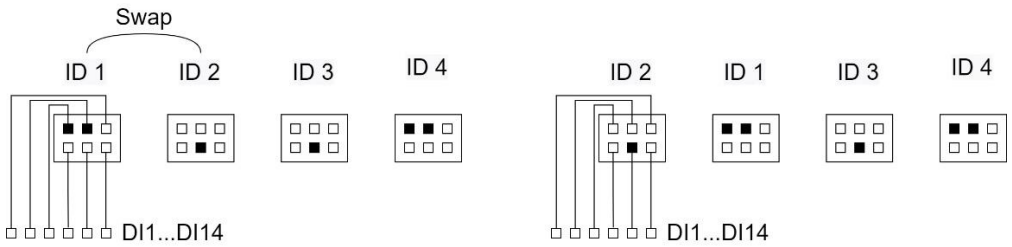

Figure 14. Driver channel configuration before and after ID swap

4.4.4 Creation of the Validated Design Proposal

The ID swap option was chosen as the approach because it has the least relays, DI-card modules, and wiring required. But still, all the 112 channels could be measured. This is possible by integrating cRIO into the ECU TS test rack, as shown in Figure 15. The approach is a good trade-off between testing coverage and the amount of wiring required.

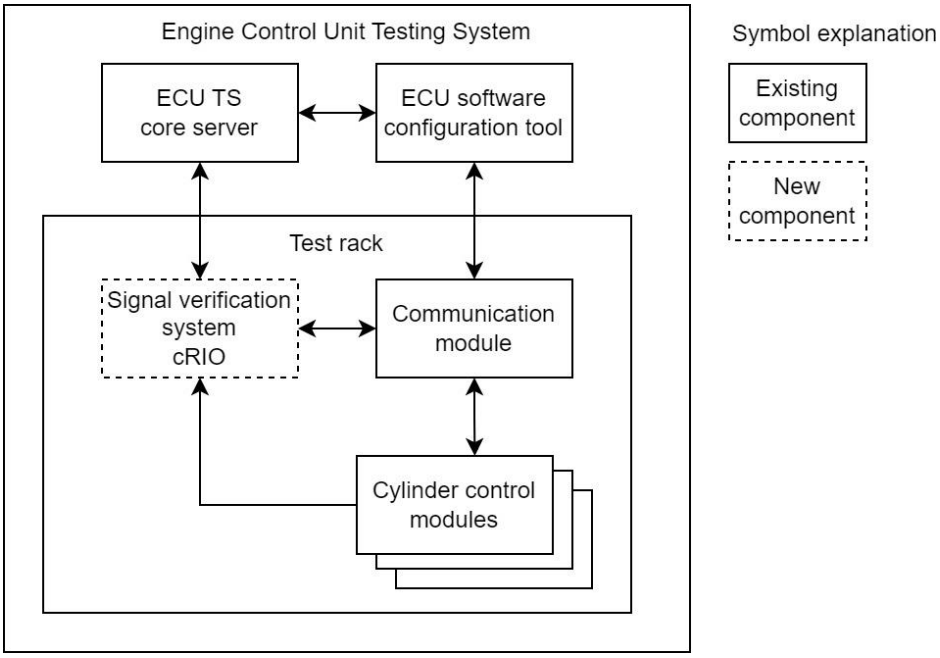

Figure 15. Integrating pulse verification system to the ECU TS architecture

ECU TS has an API that allows external systems to access and use its functions. cRIO could acquire all software configurations from the ECU TS by using that particular API, but the program implementation still needs to be done for this purpose. A user interface would need to be developed that would start the test, which could be then run whenever a new version has been developed or even every nighttime period. The advantage of this kind of system would be good testing coverage, and it would be easy to use.

ID swap can also be done manually using the ECU software configuration tool. This approach was chosen to reduce the scope of the work. The system setup at this stage is

presented in Figure 16. The ECU software configuration tool is used to upload engine configuration to modules. Simulated speed signal generation and measurement of driver channels is implemented with LabVIEW. Start, stop, and reset are wired to the communication module to control the engine. An alternative option to use two computers would be to run the ECU software configuration tool, on a computer with two network interface cards.

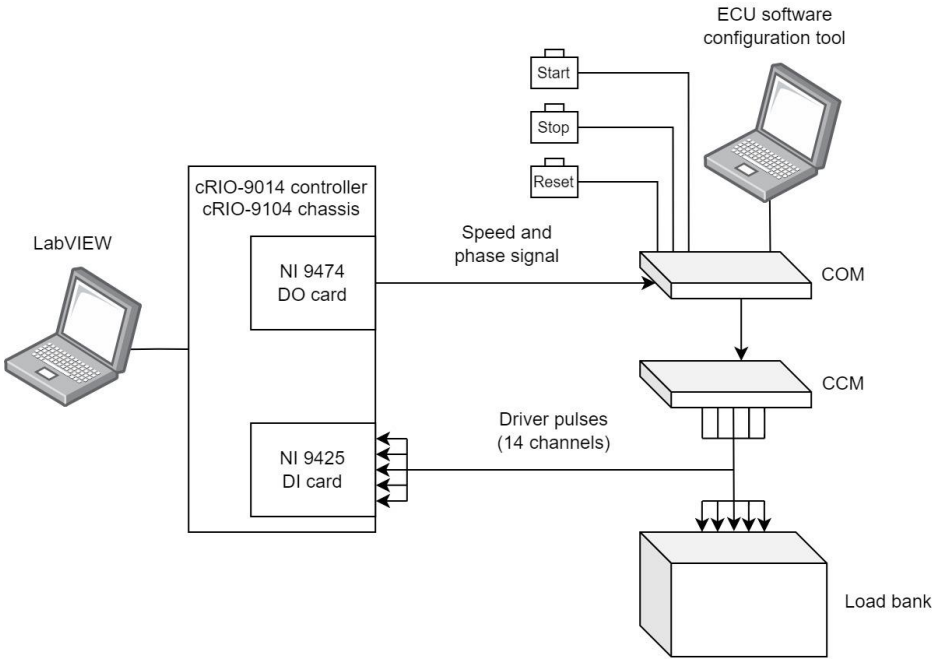

Figure 16. Setup of the laboratory system using an ID-swap for testing the proposed treatment

The configuration of the driver channels is checked manually from the ECU configuration software tool, so that the channels currently being measured could be allocated accordingly to the testing report. This is done in GUI (Figure 17), which has the following functionalities:

1. Section for filling in the test specification. The *Engine name* field is used to identify the engine in the test report. The number of cylinders and their arrangement for different engine types, e.g. in-line or V engine, are defined in the C*ylinder order* field. The *Angular displacement* field is used to input the expected angle of the fuel injection in each cylinder. *Engine rotation* direction is selected

in the corresponding field. *Test RPM array* field is for inputting different rpm values that are going to be tested to check that t_i is behaving consistently at different speeds. Similarly, *Test angle array (BTDC)* is for inputting different BTDC values subjected for testing.

2. *Path and file name* selection for the test report.
3. Tab selection for organizing different parts in the GUI. *Input data* is for the main test specifications. *Speed settings* can be used to adjust engine speed. *Preferences* has radio button selection for three different testing modes: run engine a given number of cycles, run engine until time has elapsed, and run until stop is pressed.
4. *BTDC selector* is for choosing currently configured BTDC value in the ECU.
5. *Driver channel selector* is for choosing which driver channels are assigned in the CCM being tested.
6. Button for starting and stopping speed reference signal.
7. Button to start test execution and generate test report.

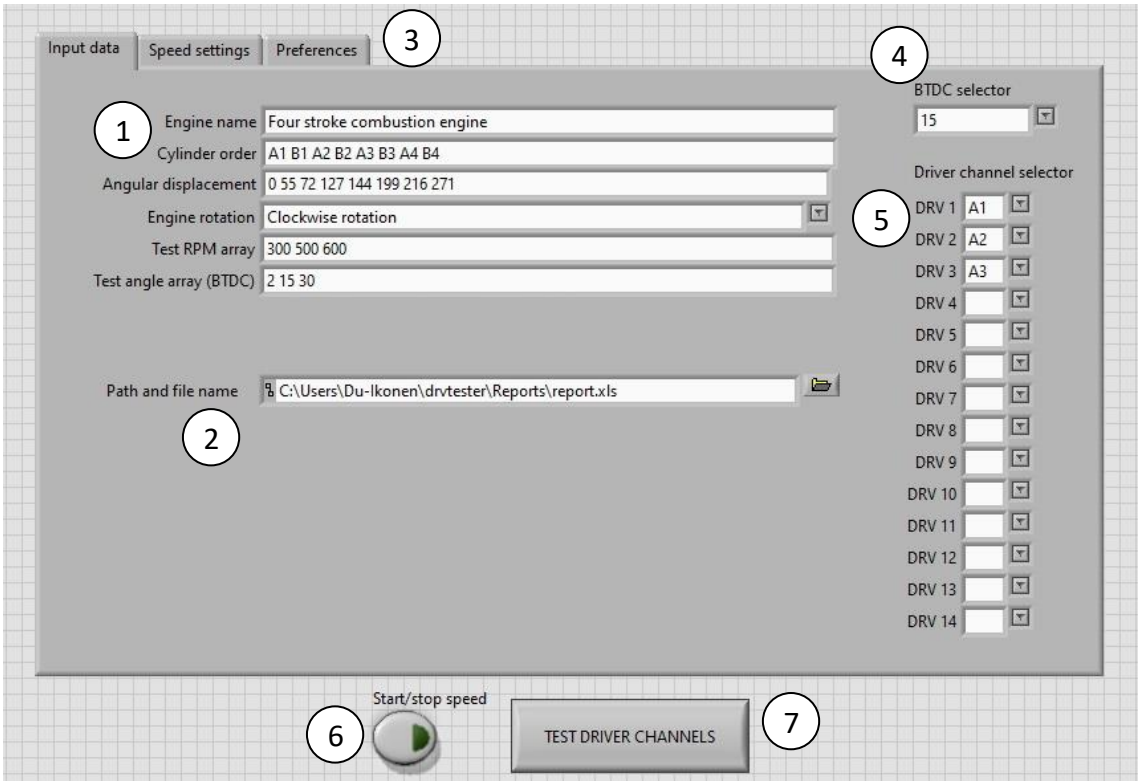

Figure 17. GUI mockup shows fields for inputting information of the test specification and buttons for starting the test

It is a good practice to first define requirements and functions before starting to design the system. This is a task similar to defining nonfunctional requirements and properties in the DSR method proposed by Wieringa, but as a part of the engineering process, this is done in a more detailed way and some descriptions about implementation are added. The proposed automatic testing system included the following properties, which were validated by experts in the third design cycle iteration:

- Engine package chosen for testing automation development: 20-cylinder V gas engine,
- Measures timing from all 14 channels from one COM,
- Timing measured with the NI 9425 DI module,
- Array of loads to simulate injector,
- Customisable number of measurements,
    - Different BTDC timings, and
    - Different rpm,

- Automatically fills timing test Excel workbook,
- Has a LabVIEW user interface from which the test can be selected and run.

The block diagram of an automatic testing system is presented in Figure 18. The software solution is implemented as two different targets, PC and FPGA, running in parallel. Time critical speed signal generation and measurement of timing are handled in FPGA by *counting ticks* (clock cycles). It will take several ticks to run the *Detect driver pulse rising edge timing* function that reads the DI channel. This function needs to return the number of ticks it takes to execute one loop, as well as the number of loop iterations it takes to run from 0 ° crankshaft angle to the first rising edge of the injection pulse. The PC target has functions for Excel report generation, GUI, and not time-critical calculations, such as unit conversions. The *Ticks to ms conversion* function obtains timing in seconds by multiplying the loop time in ticks and the number of loops returned from the *Detect driver pulse rising edge timing* function on the FPGA target with the period of time duration in one configurable FPGA frequency cycle.

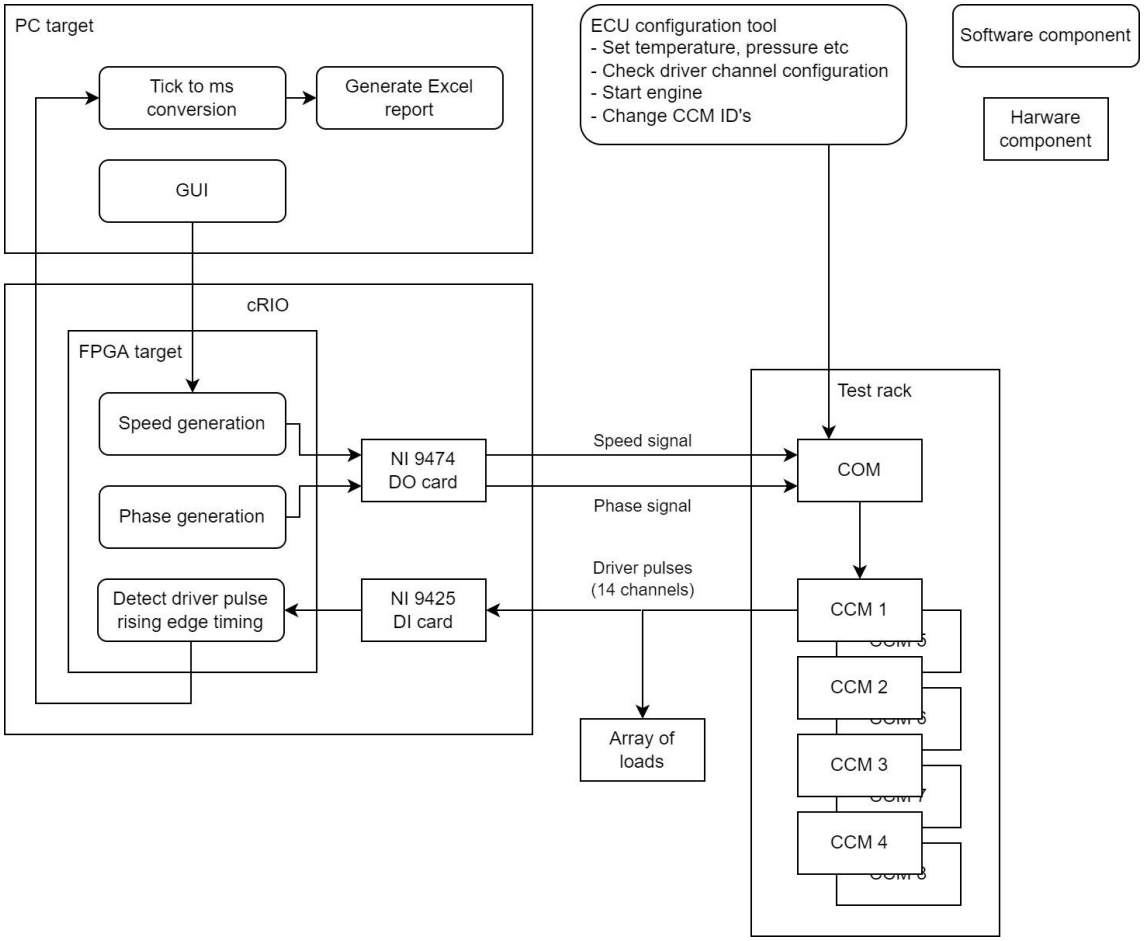

Figure 18. The block diagram of the technical proposed implementation shows the most important software and hardware components in the system

5 Results

This chapter highlights some of the significant key elements that occurred during the implementation of the testing system. The first part presents the achieved design goals and discusses the design practices used in them. The second part focuses on the FPGA implementation part. The third part explains how the interference problem from the driver channel was solved. The results of the laboratory measurement of the testing system accuracy and analysis are presented in the fourth part. Finally, in the fifth part, some future development suggestions, as well as limitations of the research are discussed.

5.1 Design Implementation

All features defined in the third treatment validation of the testing system were successfully developed. Many parts of the code were divided into *SubVIs* to increase code readability and to speed up programming by reusing the code. Moreover, SubVIs makes the code easier to debug, troubleshoot, document, and track changes (National Instruments, 2014, p. 84). SubVIs are LabVIEW equivalent of subroutines in text-based programming languages that enable modular programming structure.

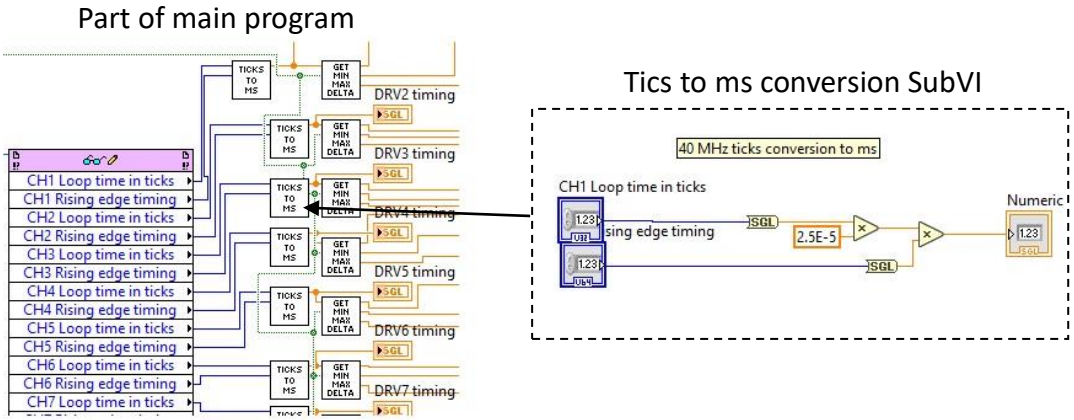

Figure 19. Using *Tick to ms* subroutine in the LabVIEW graphical programming environment

A ready-made speed simulation code was utilized by integrating it into the testing system. This program used *crank degrees per tick* (clock cycle) for the engine speed. The code was modified by adding a function that enables the engine speed to be entered in units of rpm, as in the testing templates. This function converts rpm to crank degrees per tick (clock cycle) and vice versa and keeps them updated. Rpm to crank degrees per tick (clock cycle) is obtained by:

$$\omega_{\mathrm{clk}} = 6 \cdot \omega_{\mathrm{rpm}} \cdot 10^{-7}\,, \tag{6}$$

where ω_{rpm} is crankshaft angle rpm. From equation (6) rpm is obtained:

$$\omega_{\mathrm{rpm}} = \frac{\omega_{\mathrm{clk}} \cdot 10^{7}}{6}\,, \tag{7}$$

where ω_{clk} is the rpm crank degrees per tick (clock cycle).

Several improvements to the GUI were made to enhance and speed up its operability. In Figure 17, the *BTDC selector* and *Driver channel selector* drop-down menus update dynamically as the user fills the *Test angle array (BTDC)* and the *Cylinder order* fields.

When these fields are filled with numbers, the program parses them into an array and uses that array as a data source for dropdown menus. In this way, the user can select BTDC values and cylinders more easily and faster. The first time a new engine package is put to the test, the program will generate a test report for a new Excel workbook. When testing continues after changing BTDC values or CCM IDs, the program will continue to fill the same Excel workbook that was created for the same engine package. Steps for using the testing system are described in the Appendix.

The system automatically generates an Excel report (see Figure 20). The column *difference deg* shows the difference of the *Measured timing (ms)* and the *Theoretical timing (ms)* columns. The screen shot shown in Figure 20 was taken during the development stage in a situation where the theoretical timing values of the timing were incorrect w.r.t. the real ECU configuration. The excel sheet highlights a result in red when the difference is greater than 0.1 ° so that timings exceeding the required accuracy would be easier to notice.

,00	271	B4	85,33		-85,33	-256,00	
	Timing (BTDC)	15					Timi
	Speed(rpm)	600	Theoretical	Measured			Spe
eg	Cyl TDC	Cyl	timing (ms)	timing (ms)	difference (ms)	difference deg	C
,00	0	A1	195,83	197,23	1,40	5,03	
,00	72	A2	15,83	57,23	41,40	149,04	
,00	144	A3	35,83	137,23	101,40	365,02	
,00	216	A4	55,83		-55,83	-201,00	
,00	55	B1	11,11		-11,11	-40,00	
,00	127	B2	31,11		-31,11	-112,00	
,00	199	B3	51,11		-51,11	-184,00	
,00	271	B4	71,11		-71,11	-256,00	

Figure 20. Example of test results as an Excel table

Driver channels require an electrical load that simulates fuel injectors to be wired to them before any pulse can be measured. For this purpose, an enclosure containing electrical reactors was utilized to simulate fuel injectors used in laboratory tests that could be easily connected to CCMs. The DIs measuring pulse timing were connected

parallel to loads of the same circuit. In total, 14 electrical reactors were used as a load array. The reactor model was *Hammon Manufacturing Heavy Current Chassis Mount 195e20*, having the following electrical properties suitable for the simulation of the fuel injectors:

- Inductance is 2,5 mH, and
- Resistance 0,022 Ω.

5.2 Computing Time from FPGA Clock Cycles

In the initial design the *Ticks to ms conversion* was implemented in FPGA target. In an attempt to synthesize the FPGA model that had 14 DI channels, operation would have used 107 % of FPGA resources, which led to a synthetization failure. The *Ticks to ms conversion* were moved to the PC target since they are not time critical. In the final design, 69,6 % of the FPGA resources were used.

LabVIEW software functions allow access to low-level hardware timing parameters, which enables good resolution for timing measurements. The principle of timing measurement is based on counting how many ticks (clock cycles) it takes to run the software loop that measures timing of an input pulse. Because of this, it is important to know how long it takes to run a software loop when implementing timing measurement in digital logic. It takes several ticks to read a DI channel and to run a function that gets the pulse timing. For this purpose, counters can be used to count how many ticks it takes to run one loop. This is utilized when measuring timing in seconds. While the FPGA operating frequency is known, the number of ticks can be converted into seconds. The number of clock cycles was obtained with the code shown in Figure 21. The feedback node stores the counter in the previous while loop iteration. The difference between the tick counter in the current and the previous iterations returns the number of ticks that a loop takes to complete.

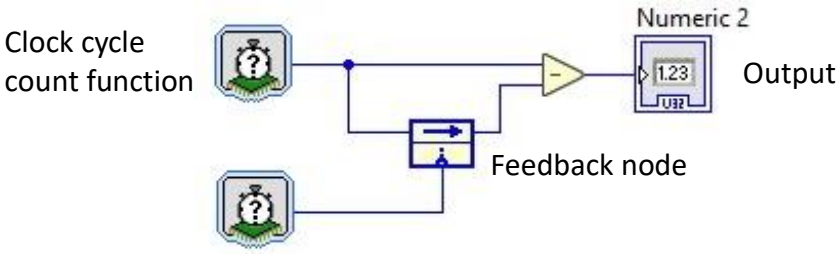

Figure 21. Obtaining the number of clock cycles needed to read a DI-channel

It is possible to adjust how often each iteration of loop structure will run, but in case of injection pulse timing measurement, the loop was set to run as fast as possible. However, the FPGA clock frequency sets a limit for this. The FPGA clock rate is configured to run in the default setting of the 40 MHz frequency. This means that one clock cycle takes 25 ns. When the number of clock cycles that each loop takes to run is known and the number of loops it takes from 0 degree angle to the first rising edge, it is possible to get the fuel injection pulse timing t_{dp} in seconds:

$$t_{dp} = n_{clk} \cdot 25 \cdot 10^{-9} \cdot n_{loop}\,, \tag{8}$$

where n_{clk} is the number of clock cycles that a loop takes to run, and n_{loop} is the number of loop iterations from 0 ° angle to the first rising edge of the fuel injection pulse.

5.3 Pulse Interference Elimination

Next, the number of times a while loop runs from starting from zero angle to the first rising edge must be acquired. Before starting programming this function, a written description of its intended operation is created. The description of the fuel t_i measurement function is as follows:

1. Start loop counter from 0 when a new 720 ° revolution starts.
2. When a rising edge is detected, assign the loop counter to a variable.

During the development of the software, it was noticed that the driver channel has interference peaks, as shown in Figure 22. While reading DI rising edges, the program would trigger to a wrong pulse. The problem was noticed when the measured timing did not match the expected timing, which was then confirmed by measuring pulses with an oscilloscope. Possible sources of interference were proposed to be caused by several alternative reasons including a flyback voltage spike caused by sudden interruption of the current supply to the coils in the electrical reactors, or an electromagnetic interference from the cables close to each other in the test setup, or current leakage from electronics, or a combination of several of these mentioned reasons.

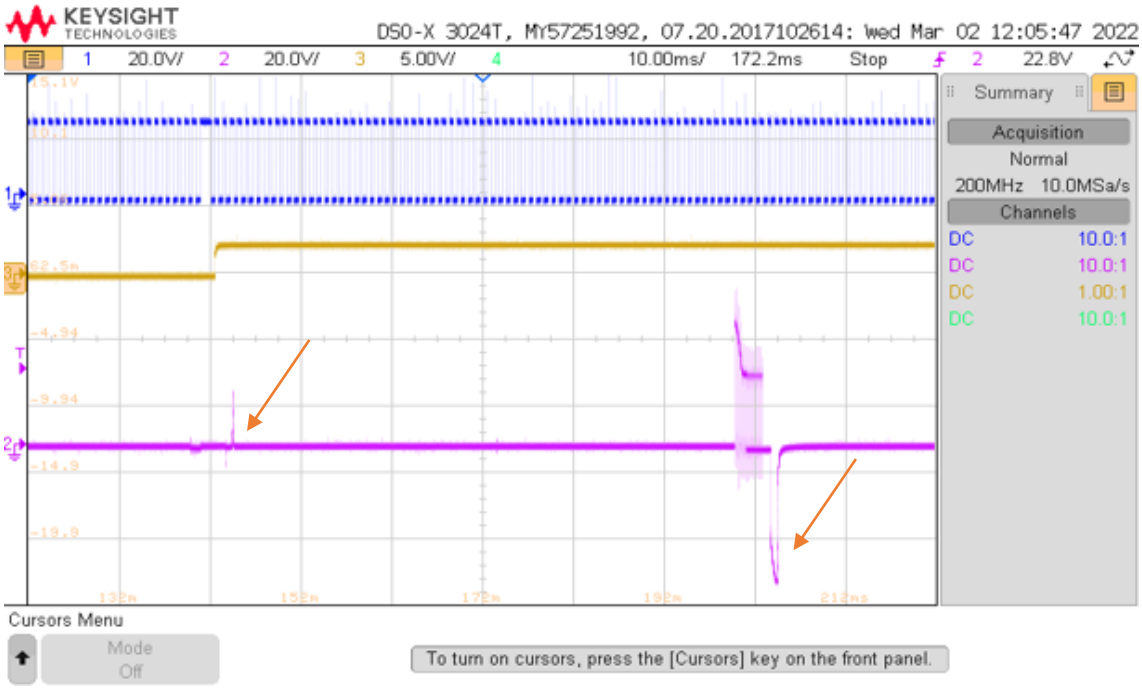

Figure 22. Pulse interferences shown by orange arrows

Triggering of the start of the injection pulse was updated to the code. The expected fuel injection angle is known, and the user should input the starting value for pulse triggering, for example, 30 ° before the expected pulse. This delays the start of measurement triggering and bypasses any interferences. Measurement triggering starts when a new crankshaft angle revolution starts, and the measurement start threshold

angle is passed. The updated description of the fuel t_i measurement function is as follows:

1. Start loop counter from 0 when a new 720 ° revolution starts.
2. When the threshold of the start of the measurement is passed and a rising edge is detected, assign a loop counter to a variable.

The LabVIEW code to obtain the timing of the fuel injection pulse is shown in Figure 23. Fuel injection DI is a burst of pulse-width modulated voltage pulses, and this function delays the starting of measurement to bypass interference and returns the timing of the first rising edge in that voltage burst. The function has three inputs that are the *crankshaft angle* (1), the *measurement start threshold* (2), and the *driver channel DI* (3). There is one output that is a *loop counter* (4), which is used to calculate the t_i in seconds. The *feedback node* (5) compares the crankshaft angle with the previous crankshaft angle value in the program iteration loop, and if the new value is less than the previous value, the comparison returns true. The *Select structure* (6), which is the LabVIEW equivalent of the if-else statement, counts loop iterations by incrementing the counter value by one on each iteration and resets the counter to zero when the new 720 ° cycle of the crankshaft angle starts. DI of the injection pulse state is registered only when the crankshaft angle is equal to or greater than (7) the measurement start threshold to filter interference. The combination of the feedback node and greater than expression (8) detects the rising edge of the DI. The *reset-set flip-flop* (9) keeps track of whether a new 720 ° cycle of the crankshaft angle has started and resets when DI of the injection pulse occurs. The loop count is assigned to the output of the function (10) when the first rising edge of the injection DI occurs for the first time in the 720 ° crankshaft angle cycle (11). Finally, the loop count can be used to calculate the injection time in seconds.

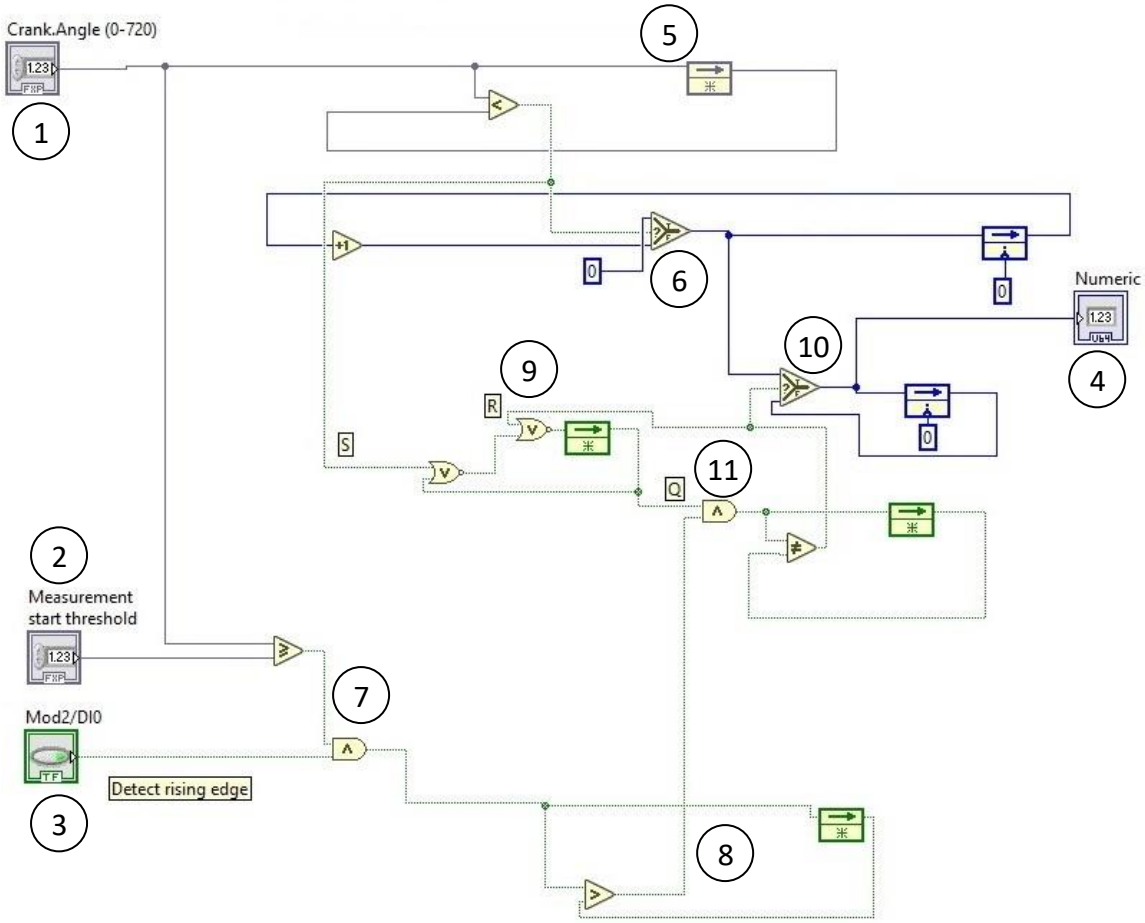

Figure 23. Code to obtain the pulse timing of the fuel injection with the threshold of the start of the measurement

5.4 Preliminary Testing of the Measurement Accuracy

Measurement functionality was tested by measuring fuel injection pulses with the testing system and with an oscilloscope and then comparing results from each (see Table 1). The first column *Test* shows the index of the measurement and the driver channel from which it was measured. Two different channels were tested to see if there would be any difference in the behaviour of the channels. The second column *Automatic testing system* shows the result of the timing measurement done with the automatic testing system as read from the GUI of the testing system. The third column *Oscilloscope* shows the timing obtained with the oscilloscope read from the screen of the oscillo-

scope. The fourth column *Δ* shows the difference between the automatic testing system and the oscilloscope measurements. The speed signal and the engine were stopped and restarted between each test.

Table 1. Comparison of the timing test results of the laboratory test with the testing system

Test	Automatic testing system (ms)	Oscilloscope (ms)	Δ
1 (CH3)	137,2289	137,2261	0,0028
2 (CH3)	137,2289	137,2259	0,0030
3 (CH3)	137,2289	137,2256	0,0033

Test	Automatic testing system (ms)	Oscilloscope (ms)	Δ
4 (CH1)	197,2276	197,2265	0,0011
5 (CH1)	197,2315	197,2270	0,0045
6 (CH1)	197,2276	197,2265	0,0011

The results were further analysed by calculating the mean, *standard deviation* (SD), and *standard error* from the timing test results. The population SD is obtained by:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}, \tag{9}$$

where $\bar{x}$ is the arithmetic mean, x_i is the value of the sample and n is the total number of samples. The standard error (SE) is obtained by:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}, \tag{10}$$

where σ is the population SD and n is the total number of samples. The results of the mean, SD, and standard error are shown in Table 2 for both driver channels.

Table 2. Mean, standard error, and accuracy of the test result

Driver channel 3	**Automatic testing system (ms)**	**Oscilloscope (ms)**	**Δ**
Mean	137,2289	137,2259	0,0030
Standard deviation	0	0,0002	
Standard error	0	0,0001	
Accuracy	137,2289 ± 0,0000	137,2259 ± 0,0001	

Driver channel 1	**Automatic testing system (ms)**	**Oscilloscope (ms)**	**Δ**
Mean	197,2289	197,2267	0,0022
Standard deviation	0,0018	0,0002	
Standard error	0,0011	0,0001	
Accuracy	197,2289 ± 0,0011	197,2267 ± 0,0001	

The difference in the mean values (*Δ*) in Table 2 suggests that the oscilloscope detects the signal about 2–3 µs earlier than the automatic testing system. From this laboratory setup, it cannot be determined which instrument is closer to true value since the true value is unknown. Therefore, it cannot be resolved which instrument is more accurate. The test results of the automatic testing system from driver channel 3 are exactly the same with eight significant figures, so SE is zero while in driver channel 1 SE is 1,11 µs while the oscilloscope SE is 0,1 µs. The sample size of the observations is too small to determine the precision of the instruments.

Since accuracy or precision cannot be reliably clarified, the highest value of differences between test measurements is used as a point of reference. The highest difference value is 4.5 µs from Table 1 which can be rounded to 5 µs. Therefore, the accuracy of the testing system is ±5 µs. This is less than half of the required update rate for the measurement, which is 11 µs. As a result, the measurement was found to be sufficiently accurate in the preliminary testing.

5.5 Limitations

5.5.1 Future Development Suggestions

The most important improvement for the automatic testing system would be to develop communication between the testing system and ECU TS, which would increase the level of automation. A communication API would enable reading the driver channel configuration so that the user would not need to do this manually. Moreover, the BTDC values could be changed and uploaded to the modules. As a result, the whole test report could be generated with a single press of a button. The details of cRIO and ECU TS communication require further investigation. LabVIEW supports ActiveX technology, which could be possibly used for this purpose.

The tester needs further development to complete some missing features and minor bug fixes. In the test *Preferences* tab, different testing modes are disabled. These would enable a more useful testing state. However, ECU TS also includes a reporting component, which would be another way to implement reporting than the current Excel reporting.

Changing the rpm speed stops the speed signal for one second. This will trigger an emergency stop on a running engine. The one second stop was made as a workaround to a problem that made measurement go out of sync when a new rpm was inputted. The development package with which the software was developed did not have this issue. A different kind of solution needs to be developed to prevent the measurement from going out of sync. Also, the engine package will automatically emergency stop the engine if the speed instantly goes from 0 to 750 rpm. A ramping speed pulse would need to be created.

The driver channel tester detects the first rising edge after a zero angle. For future development, different types of pulses should be investigated to check if the rising edge

detection is a valid way to test their timing. For example, in the ignition channel, the timing of the last falling edge is important. Different pulses to consider are listed in page 52.

Since fuel injection pulses have interference, measurement can trigger an incorrect rising edge. The source of interferences should be investigated, but suspected reasons were flyback voltage spike, electromagnetic interference, or current leakage from electronics. Flyback voltage can be eliminated by connecting a flyback diode to the circuit. To eliminate electromagnetic interference, cables in the laboratory setup would need to be replaced with shielded cables and re-arranged in a manner that 230 V AC power cables used in the setup would have minimum distance of 30 cm to the DC cables. Suspected current leakage from electronics cannot be fixed easily since it would probably require replacing the electronics, which would not be reasonable since the ECU used in the test is found to work from electronics respect, even sensitive measurements could be interfered with by some minor current leakages. In addition to fixing hardware related problems, software-based solutions are recommended. Pulse filtering would be a useful addition for later use since other sources of interference could emerge later, and thus it would make the testing system more reliable. A suitable filtering technique could be filtering the signal by its duration. The duration of the signal must exceed the length of some adjustable variable to be registered for a valid measurement.

Of the FPGA resources, 69.6 % was used. For future development, if all channels were tested at once, it could be useful to explore whether the FPGA code can be optimised. One possibility for that would be to move the rpm and crank degree conversions from the FPGA target to the PC target. These two functions were used to convert the rpm to the crank degrees and vice versa so that the user could input the engine speed in the GUI in both formats. These are not time critical functions and could be moved to a PC target without any compromises to the system operation. Conversion functions use division operations which have high FPGA resource consumption.

A useful research question for the future would be the calculation of the *return on investment* (ROI) of the automatic testing. When the automated testing system would be completed, it would be natural to study how much it saves time. In this study, the time used to perform the tests using an automatic testing system could be compared to the execution of the same tests done manually. The time used to develop the automation system would need to be taken into account to evaluate the feasibility of developing the automation. Additional variables could be the quantity, accuracy, and consistency of tests which affect the quality of testing which possibly creates more value than only looking at the cost savings of automation.

5.5.2 Research Limitations

In the laboratory setup to test the accuracy of the measurement, too few samples were taken so that the accuracy and precision of the testing system could not reliably be determined. Based on the preliminary tests, there was an offset of 2–3 µs between the mean values of the sample observations made with the oscilloscope and the testing system. It is unclear whether this is systematic or not, and a larger sample size could possibly be used to determine that. If the problem would seem to be systematic, the accuracy issue could be resolved by using a testing instrument with a known accurate control pulse duration, and then comparing both the testing system and oscilloscope to that. A higher sample size would most likely have been useful for determining the precision with higher confidence.

Generalizing the result of the answer to the knowledge question “What would the architecture be like for an automated testing system for the verification of engine fuel injection?” has both strengths and limitations. While the upper-level principles and hardware choices are valid for other ECUs and systems, on a detailed software level, they cannot be utilized in other applications. Software is highly application specific, and therefore e.g. the proposed setup could not be utilized for the testing of other ECUs. When developing an automated testing system for other ECUs than the one presented

in this thesis, first, it would require good understanding of the details of the system for which it is developed, and second, software would need to be developed from the beginning. Also, it is most likely that the configuration of the cRIO hardware modules would need to be changed.

Results of the second knowledge question, “Are there hidden and unexpected technical problems involved in the design?”, also most likely cannot be generalized to other applications. Pulse interference, lack of ECU TS communication, and minor bug fixes are probably problems that are limited to the specific ECU studied in this work. These kinds of issue are highly application specific and therefore results cannot be used for forming a general rule. In this work, ECU TS was an integral part of the testing technology architecture and working practices. Similar structures may or may not be relevant to other testing applications.

6 Conclusion

The design of an automatic testing system for fuel t_i was successfully created. All non-functional requirements and properties set for the design were met. Testing was carried out using one of the latest ECU configurations and the measurement accuracy of the automatic testing system was ±5 µs, which was sufficient. The measurement precision could be improved even further by choosing a DI module with a higher update rate. This would not increase the cost of the system. Moreover, the developed testing system automatically generates an Excel report.

When considering the design science knowledge question about the design of the architecture for the automatic testing system, (Figure 10 and Figure 18) can be generalized to develop a timing testing system for any ECU or electronic system with input and output. NI cRIO proved to be a good hardware and software platform since it is relatively easy to use, highly customizable, reliable, and has high-speed performance for precise measurements. The modularity and programmability of the cRIO and LabVIEW solution enables flexibility for further development and modifications. Therefore, cRIO can be recommended as a platform for developing automatic timing testing systems.

The literature review suggests that there is a finite amount of studies made about fuel t_i testing. According to them, high precision can be achieved with automatic testing, which is similar to the results in this study. Another similarity is that a good software architecture is crucial for developing functional testing automation, as stated by Graham and Fewster. It is important that the developer should become familiar with the details of the testing procedure. A good way to achieve this is to first personally spend some time doing the manual testing to understand the specific details in it.

Developing extensive automatic testing is complicated and time consuming. Understanding both the broad view of the system and small details together, makes the development of testing challenging. While systems are evolving with ever increasing speed, testing should be designed to be flexible enough so that it would not become

obsolete too quickly. Because developing testing is time and resource consuming, efforts put into it should be put in proportion to the benefits that the automation brings. Critical applications related to, for example, safety should be thoroughly tested. In addition, large projects that have hundreds of developers and are under constant development benefit from automatic testing. As an example, this work shows a feasible testing automation architecture and discusses the issues to be taken into account while designing it.

References

A. Keawtubtimthong, D. Koolpiruck, S. Wongsa, Y. Laoonual, & A. Kaewpunya. (2010). Development of engine control technique for flex-fuel motorcycle. *ECTI-CON2010: The 2010 ECTI International Confernce on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, 159–162.

Boyang Du & L. Sterpone. (2016). An FPGA-based testing platform for the validation of automotive powertrain ECU. *2016 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 1–7. https://doi.org/10.1109/VLSI-SoC.2016.7753553

Donev, J., & Afework, B. (2019). *Energy Education—Four stroke engine [Online]*. https://energyeducation.ca/encyclopedia/Four_stroke_engine.

Dustin, E. (2002). *Effective Software Testing: 50 Specific Ways to Improve Your Testing 1st Edition* (1st ed.). Addison-Wesley Professional.

Dustin, E., Garret, T., & Gauf, B. (2009). *Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality*. Addison-Wesley Professional.

Eteläpää, A. (2021). *WCD-20 spark diagnostics* [University of Vaasa]. https://urn.fi/URN:NBN:fi-fe202102104377

F. Juan & M. Xian-Min. (2009). Research on fuel injection intelligent control system. *2009 4th IEEE Conference on Industrial Electronics and Applications*, 2782–2785. https://doi.org/10.1109/ICIEA.2009.5138716

Graham, D., & Fewster, M. (2012). *Experiences of test automation: Case studies of software test automation* (2nd ed.). Addison-Wesley.

J. Mahboob & J. Coffman. (2021). A Kubernetes CI/CD Pipeline with Asylo as a Trusted Execution Environment Abstraction Framework. *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, 0529–0535. https://doi.org/10.1109/CCWC51732.2021.9376148

J. Zhou, G. Ouyang, & M. Wang. (2010). Common Rail Direct Injection Diesel Engine Control Strategy Validation Research. *2010 International Conference on Digital Manufacturing & Automation*, *1*, 387–390. https://doi.org/10.1109/ICDMA.2010.399

Jena, A. K., Das, H., & Mohapatra, D. P. (2020). *Automated Software Testing: Foundations, Applications and Challenges*. Springer Singapore Pte. Limited. https://link.springer.com/book/10.1007/978-981-15-2455-4

Johannesson, P., & Perjons, E. (2021). *An Introduction to Design Science* (P. Johannesson & E. Perjons, Eds.; Second Edition). Springer International Publishing. https://doi.org/10.1007/978-3-030-78132-3

Khair, M. K. & Jääskeläinen, H. (2020). *Diesel Fuel Injection*. DieselNet. https://dieselnet.com/tech/diesel_fi.php

Knol, W., Coombes, P., & Couvert, G. (2022). *Spark plugs: Discovering Denso Technology*. Denso. https://www.denso-technic.com/images/document/ignition/en/spark-plugs-manual-en.pdf

Latarche, M. (Ed.). (2020). *Pounder's Marine Diesel Engines and Gas Turbines (Tenth Edition)*. Butterworth-Heinemann.

https://www.sciencedirect.com/book/9780081027486/pounders-marine-diesel-engines-and-gas-turbines

M. Venkatraman & G. Devaradjane. (2010). Experimental investigation of effect of compression ratio, injection timing and injection pressure on the performance of a CI engine operated with diesel-pungam methyl ester blend. *Frontiers in Automobile and Mechanical Engineering -2010*, 117–121. https://doi.org/10.1109/FAME.2010.5714814

Mitroglou, N., Gavaises, M., & Arcoumanis, D. (2012). Spray stability from VCO and a new Diesel nozzle design concept. In IMechE (Ed.), *Fuel Systems for IC Engines* (pp. 279–290). Woodhead Publishing. https://doi.org/10.1533/9780857096043.7.279

National Instruments. (2014). *NI LabVIEW for CompactRIO Developer's Guide*. *Generic*. https://www.ni.com/pdf/products/us/fullcriodevguide.pdf

Papalambros, P. Y. (2015). Design Science: Why, What and How. *Design Science*, *1*, e1-undefined. https://doi.org/10.1017/dsj.2015.1

Pensar, J., & Storbacka, M. (2007). UNIC – The reliable solution for robust industrial controls. *Wärtsilä Technical Journal*, 40–44.

Q. Kang, Z. Xie, Y. Liu, & M. Zhou. (2017). A fuel injection control SoC for Diesel Engine Management System. *2017 IEEE 12th International Conference on ASIC (ASICON)*, 969–972. https://doi.org/10.1109/ASICON.2017.8252639

S. Wu, Y. Dong, K. Yu, & X. -Q. Tang. (2020). Fuel engine injector control based on multiple search approach. *2020 19th International Symposium on Distributed Com-*

puting and Applications for Business Engineering and Science (DCABES), 100–103. https://doi.org/10.1109/DCABES50732.2020.00034

T. Stratoudakis. (2021). *Introduction to LabVIEW FPGA for RF, Radar, and Electronic Warfare Applications*. Artek House. https://www.proquest.com/docview/2515890798/4B25567FC4D344F0PQ/3

V. Vasquez Lopez, J. M. Echeverry Mejia, & D. E. Contreras Dominguez. (2017). Design and Statistical Validation of Spark Ignition Engine Electronic Control Unit for Hardware-in-the-Loop Testing. *IEEE Latin America Transactions*, *15*(8), 1376–1383. https://doi.org/10.1109/TLA.2017.7994782

Wärtsilä. (2019). *Wärtsilä 46DF product guide*. Wärtsilä Finland Oy. https://www.wartsila.com/docs/default-source/product-files/engines/df-engine/product-guide-o-e-w46df.pdf

Wärtsilä. (2022). *Wärtsilä 26 product guide*. Wärtsilä Finland Oy. https://www.wartsila.com/docs/default-source/product-files/engines/ms-engine/product-guide-o-e-w26.pdf

Wieringa, R. J. (2014). *Design Science Methodology for Information Systems and Software Engineering*. Springer Berlin / Heidelberg. https://link.springer.com/book/10.1007/978-3-662-43839-8

Appendix

Steps for Using the Testing System

1. Start ECU configuration software tool
2. Upload engine packet to the testing rack and ensure that the upload was successful for every module
3. Set the correct simulated temperature, pressure, and other safety limit parameters in the ECU configuration software tool
4. Fill in the test information in the automatic testing system
    a. Engine name
    b. Cylinder order array
    c. Angular displacement array
    d. CW/CCW rotation select
    e. Fill test rpm array
    f. Fill test BTDC array
    g. Timing in ms or deg selection
    h. Stepless intake and exhaust valve selection
5. Check driver channel configuration from the ECU configuration software tool and set right module and driver channels in the automatic testing system
6. Start engine in ECU configuration software tool
7. Start speed in automatic testing system
8. Click the button “Test driver channels” in the automatic testing system
9. Change to the next CCM ID in the ECU configuration software tool and upload it again and repeat steps 5-8 until all CCM driver channels have been tested
10. Save test report