LUND UNIVERSITY

Embedded Systems and FPGAs for Implementation of Control Oriented Models, Applied to Combustion Engines

Wilhelmsson, Carl

2009

Embedded Systems and FPGAs for Implementation of Control Oriented Models

Applied to Combustion Engines

Embedded Systems and FPGAs for Implementation of Control Oriented Models

Applied to Combustion Engines

Carl Wilhelmsson

Division of Combustion Engines
Department of Energy Sciences
Lund University
Lund, November 2009

Till min Ella och min familj

Division of Combustion Engines
Department of Energy Sciences
Lund University
Box 118
SE-221 00 LUND
Sweden

ISBN 978-91-628-7957-0
ISSN 0282–1990
ISRN LUTMDN/TMHP--09/1068--SE

Printed in Sweden by Media-Tryck.
Lund, 2009

Abstract

Performance demands put on combustion engines are ever increasing, e.g. demands on emissions and fuel consumption. The increased demands together with new combustion concepts increase the need for feedback engine and combustion control. Mathematical models are considered important in order to implement high performance feedback control, as well as to perform diagnostic functions in vehicles.

Various implementation platforms which can be used to implement mathematical models in vehicles are described in this this thesis; embedded processors, FPGAs and ASICs. Which of these implementation platforms to choose must be decided based on the intended application and current demands on performance. Embedded systems, ASICs and FPGAs are discussed based on literature found in the field, covering a wide span of considerations. Furthermore a number of considerations which are important when implementing algorithms and logic in embedded processors and FPGAs are described.

The theory is put into practice in the thesis, implementing a heat release calculation on an FPGA and developing and implementing a NO_x model in an embedded processor. To be able to implement a fast NO_x model several techniques were used. Parts of the model were tabulated, difficult operators such as division were avoided and the properties of fast C code was kept in mind.

This thesis combines the areas of automatic control, electronic hardware design and development of embedded software, and applies it to combustion engine control. The work undertaken indicate different possibilities when implementing high speed control oriented models in FPGAs and embedded processors. This thesis aims to fill a gap between state space models, common in automatic control, and high fidelity physical models, commonly used for simulation, by providing a method to develop high fidelity control oriented models which are low in computation demand and implementable in FPGAs and embedded processors.

Acknowledgements

The first person I would like to thank for his support and understanding is Per Tunestål. Per has been an great mentor for me and he has always provided help and encouragement when needed, he has simply done a perfect job as my supervisor and without him I would not have been able to write this thesis. Per is also a good friend and we have shared a lot of great experiences in Japan, California, the rest of the US and back home in Sweden.

I want to thank professor Rolf Johansson for his untiring support when writing modeling and control oriented papers. Anders Widd has been my 'colleague' at the automatic control department, his contribution to the NO_x model is greatly appreciated. Professor Anders Rantzer has been my assistant supervisor and the contacts with him have always been positive. The help of Leif Andersson has been valuable while typesetting using the wonderful LaTeX document preparation system. Krister Olsson has helped acquiring computers and what ever goes along, he has also continuously provided sly analysis of the surrounding world. Gunvi, Maj-lis, Nina and Ingrid have all helped me to keep the unavoidable paperwork going.

The department has a lot of young employees and the atmosphere is openminded and pleasant. I especially want to thank Thomas for good friendship on and off work, sharing loads of lunch breaks and a great interest in performance motorbikes. Andreas, Håkan, Leif and Jari were all very good friends at work before they moved on to new challenges. Clement, Sasa, Magnus, Thomas and I had a nice time in NYC during what came to be my last conference journey. Patrick was in Japan and joined the department much in the same way I did, I wish him the best of luck in the future. The technicians, especially Bertil and Tom have helped me a lot in the workshop, keeping up my interest in workshop work through different projects such as a diving torch.

I want to thank my friends at Toyota, especially Moriya Hidenori for tutoring me both on and off work in Japan 2004, putting his own life in second hand sharing a fantastic half year with me in Susono. It was a long

time ago but the memories are precious and the stay in Japan is what came to cause my PhD studies. Yanagihara Hiromichi, also with Toyota, helped me to get to Japan and has maintained contact with me ever since, keeping an eye on me and my research. Ohata Akira, my supervisor at Toyota during that time deserves appreciation as well.

Finally I want to thank my whole family which is very precious to me. I know you wonder what I really do, from time to time I am not really sure my self... I want to thank my love Ella for sharing her life with me, Ella has patience with me, my stubbornness, curiosity and restlessness in a way which I never even hoped to find. Ella, I hope to share the rest of my life with you. My brothers, Nils and Per, as well as my parents, Leif and Ingrid, are very important persons in my life, they have always been and will always be. My 'father-in-law' Thomas keeps feeding the wild hogs so that I can stay at the desk and focus on work.

Friends are important to have, in good times and in bad. I have good friends who deserve attention, Erik, Ola and Stefan among others.

To those not mentioned here I am not less grateful.

Carl

Contents

Preface

Scope

To improve control and on board diagnosis of combustion engines mathematical models are needed. These mathematical models need to be computed in some sort of electrical system on board the vehicle. The common method to implement computations in electronic systems is to use some sort of processor (known from e.g. personal computers). Processors are flexible and easy to develop software for but they are not suited for all types of problems. In some cases where the algorithms are very computationally demanding alternatives to processors may be a better solution. The Field programmable Gate Array (FPGA) is an intermediate between processors and the non-reconfigurable Application Specific Integrated Circuits (ASICs). FPGAs consist of electronic hardware which can be reconfigured and they may well provide a good platform for implementation of demanding algorithms, especially in combination with processors and ASICs.

When implementing a mathematical model in electronic computation systems it is important to reform the model so that it can be computed efficiently. Increasing the computational efficiency decreases the computational load and reduces the requirements and, thus, the cost of the electronic system. If the properties of the computation system are kept in mind from the beginning it is possible to steer the development in the right direction at an early phase. This reduces the risk of ending up in a dead end when moving on from the development phase to the implementation phase.

This work was conducted to study how to efficiently implement control oriented models in embedded processing systems (embedded systems) and FPGAs. Two different mathematical models were implemented as case studies. The first model, a heat release analysis, was mathematically rewritten in order to be implemented more easily in an FPGA environment. The second

model was a NO_x emission model which was developed with the intention to implement it on an embedded processor. Both model and resulting algorithm were developed with computational efficiency in mind. This work was performed to indicate a possible development practice which can be used to develop efficient physical models in embedded systems.

Limitations

This work was conducted to show possible methods to develop control oriented and mathematical models which are computationally efficient but still maintain fidelity. To do that some background knowledge about the various electrical systems had to be explored and explained, even though no claim is made of providing an exhaustive description of these systems. Embedded systems and FPGAs are very complex systems and it is not possible to fully describe these systems in this work. The description given should be understood as an 'appetizer' describing parts which are considered important for the concept studies performed. Similarly, this work was not intended to develop combustion engines as such and the engine background provided in the thesis is hence very rudimentary.

The implementation of the heat release was intended as a concept study, showing the potential performance when implementing control related algorithms on FPGAs. It was not intended as a state of the art heat release analysis which is why development work ended when a functional version was finished; there is potential to develop the FPGA based heat release much further in terms of precision.

The NO_x model was developed with the intention to obtain high computational efficiency in an embedded (processor) system. The contribution is hence a control-oriented model and should be read in the context of online on-board modeling. It is *not* an effort to exhaustively model combustion physics and chemistry. The main intention with the model was to show a method to develop high fidelity physical models which are implementable on embedded systems and some limitations were hence applied; EGR was not taken into account and the validation of the NO_x model was performed on a smaller data set than would be desirable if the actual model would be the most important part of the result.

Contribution

This work aims to fill the gap which exists between developers of models and control algorithms and embedded programmers or electronic hardware designers. It shows how a model can be developed and implemented in a controlled manner, maintaining the original precision of the model and obtaining a computationally efficient end result through keeping the implementation platform in mind during the process.

To be able to select an appropriate implementation platform for on board mathematical models it is important to know which options exist and the strong and weak points of the different options. To this end, properties of embedded processors and FPGAs were studied in detail. A literature review was performed which indicates differences and similarities between embedded systems and FPGAs.

In automatic control theory there are systematic approaches to reduce the complexity of models. In this work, an alternative approach is used, the model is developed with the limitations of an embedded systems in mind from the very beginning. Using detailed knowledge of the model as well as detailed knowledge of embedded systems, the intended implementation platform, it is possible to develop and implement a model which is computationally efficient without sacrificing model precision. The novelty with this approach is that the complete development flow, starting with mathematical concept and ending with a fixed point version of the model implemented in C, is carried out with the important implementation limitations in mind. Using such a strategy means that the 'gap' which frequently exists between algorithm developers and embedded programmers is bridged, enabling a better end result. The resulting model excels in terms of computation speed, true inter engine cycle performance was obtained using a 'normal' embedded processor, while still producing the correct output.

If the model would be too computationally demanding, even when developed for an efficient implementation from the beginning, it is possible to use another implementation platform such as FPGA or ASIC. An FPGA or ASIC implementation can be made much faster than the corresponding processor implementation. The implementation of a heat release algorithm in an FPGA is one of the contributions in this theses, showing performance which was several orders of magnitude faster than previous similar work.

Outline

This thesis is written as a monography which means that no previously published papers are included in the thesis. Nevertheless material from previous publications is covered. All publications by the author that have contributed to this thesis are listed under the caption 'Attributed Publications' bellow.

The first chapter encountered by the reader is the introduction, putting the work into context by giving the background to combustion engine feedback control. There are two sections about internal combustion engines. Internal combustion engines as such are described very briefly in the first section. The second section describes combustion engine feedback control. Heat release analysis, cylinder pressure measurement and NO_x models are described in detail in the same section.

The second chapter describes embedded systems and Field Programmable Gate Arrays. The properties of embedded processors are described in the first section. The second section describes FPGAs in somewhat more detail, since FPGAs represent a newer and less known environment than embedded processors. The history, the architecture of the actual device as well as different architectural considerations on the design level and finally design tools and methods are discussed. There is a section that describes the similarities and differences between embedded processors, FPGAs and ASICs. Developing algorithms intended for embedded systems or FPGAs requires a number of special design considerations and those are treated in the last section of the second chapter.

The third chapter deals with the work presented in the first and second papers covered by this thesis, namely an FPGA implementation of a heat release analysis algorithm. The chapter describes the experimental setup, the design tools used, the test environment as well as the algorithm used and its actual implementation on the FPGA. Finally, the performance of the final system is described and discussed. This work was intended as a 'proof of concept' study.

The fourth chapter describes the work which was undertaken to develop and implement an on board implementable NO_x model. These results were originally presented in paper three and four among the attributed papers. Firstly the model is described in general, including model assumptions, mathematical concept and equations. Secondly the algorithm which was developed to implement the model is described; The different changes which had to be made in order to implement the model efficiently, e.g. tabulating parts of the model. A description of the experimental platform (embedded system) follows. Finally the results from the tests with the original model as well as the the embedded implementation of the model are described and discussed. Some concluding remarks end the thesis.

Attributed Publications

Model based engine control using ASICs

Engine Control, Simulation and Modeling (E-COSM) - Rencontres Scientifiques de l'IFP

Carl Wilhelmsson, Per Tunestål, Bengt Johansson
Division of Combustion Engines, Department of Energy Sciences, Faculty of Engineering, Lund University

Poster presented by the first author and Per Tunestål at the New Trends in Engine Control, Simulation and modeling, Rueil-Malmaison, France, October 2006

FPGA Based Engine Feedback Control Algorithms

Fédération Internationale des Sociétés d'Ingénieurs des Techniques de l'Automobile (FISITA) Technical paper F2006P039

Carl Wilhelmsson, Per Tunestål, Bengt Johansson
Division of Combustion Engines, Department of Energy Sciences, Faculty of Engineering, Lund University

Presented by the first author at the 31st FISITA World Automotive Congress, Yokohama, Japan, October 2006

A Physical Two-Zone NO_x Model Intended for Embedded Implementation

Society of Automotive Engineering (SAE) Technical Paper 2009-01-1509

Carl Wilhelmsson, Per Tunestål, Bengt Johansson
Division of Combustion Engines, Department of Energy Sciences, Faculty of Engineering, Lund University

Anders Widd, Rolf Johansson
Department of Automatic Control, Lund University

Presented by the first author at SAE World Congress, Detroit, MI, USA, 2009

A Fast Physical NO_x Model Implemented on an Embedded System

Engine Control, Simulation and Modeling (E-COSM) - Rencontres Scientifiques de l'IFP

Carl Wilhelmsson, Per Tunestål
Division of Combustion Engines, Department of Energy Sciences, Faculty of Engineering, Lund University

Anders Widd, Rolf Johansson
Department of Automatic Control, Lund University

Presented by the first author at the New Trends in Engine Control, Simulation and modeling, Rueil-Malmaison, France, October 2009

Related Publications

Combustion Chamber Wall Temperature Measurement and Modeling During Transient HCCI Operation

Society of Automotive Engineering (SAE) Technical Paper 2005-01-3731

Carl Wilhelmsson, Andreas Vressner, Per Tunestål and Bengt Johansson
Division of Combustion Engines, Department of Energy Sciences, Faculty of Engineering, Lund University

Gustaf Särner, Marcus Aldén
Division of Combustion Physics, Faculty of Engineering, Lund University

Presented by the first author at the Power train & Fluid Systems Conference & Exhibition, San Antonio, TX, USA, October 2005

The Effect of Displacement on Air-Diluted Multi-Cylinder HCCI Engine Performance

Society of Automotive Engineering (SAE) Technical Paper 2006-01-0205

Jari Hyvönen, Carl Wilhelmsson, Bengt Johansson
Division of Combustion Engines, Department of Energy Sciences, Faculty of Engineering, Lund University

Presented by Bengt Johansson at the SAE 2006 World Congress & Exhibition, Detroit, MI, USA, April 2006

Operation strategy of a Dual Fuel HCCI Engine with VGT

Japanese Society of Automotive Engineering (JSAE) Technical Paper 20077035, SAE Technical Paper 2007-01-1855

Carl Wilhelmsson, Per Tunestål, Bengt Johansson
Division of Combustion Engines, Department of Energy Sciences, Faculty of Engineering, Lund University

Presented by the first author at the JSAE/SAE International Fuels & Lubricants Meeting, Kyoto, Japan, July 2007

An Ultra High Bandwidth Automotive Rapid Prototype System

Proc. International Federation of Automatic Control (IFAC), pp. 563-570, Technical Paper AAC07-057

Carl Wilhelmsson, Per Tunestål, Bengt Johansson
Division of Combustion Engines, Department of Energy Sciences, Faculty of Engineering, Lund University

Presented by the first author at the Fifth IFAC Symposium on Advances in Automotive Control, Aptos, CA, USA, August 2007

Control-Oriented Modeling of Homogeneous Charge Compression Ignition incorporating Cylinder Wall Temperature Dynamics

Proc. 9th International Symposium on Advanced Vehicle Control, pp. 146-151

Anders Widd, Rolf Johansson
Department of Automatic Control, Lund University

Per Tunestål, Carl Wilhelmsson
Division of Combustion Engines, Department of Energy Sciences, Faculty of Engineering, Lund University

Presented by the first author at 9th International Symposium on Advanced Vehicle Control, Kobe, Japan, 2008

1

Introduction

As the reader may know internal combustion engines have been the dominating energy source in mobile applications for something like a century. The reader is probably also familiar with the great threat to the environment which is posed by mankind's wasteful use of energy. Internal combustion engines are a part of this energy waste and their contribution to the environmental harm is exaggerated by the fact that most normal types of combustion engines emit both carbon dioxide (CO_2) and other harmful compounds like oxides of nitrogen (NO_x), hydrocarbons (HC), carbon monoxide (CO) and paticulate matter (PM). Due to environmental issues, the green house effect and increasing fuel prices, there is a strong urge to improve the internal combustion engine. A very important factor for improving the environmental and economical performance of engines is feedback control of various engine parameters.

1.1 The Internal Combustion Engine

Traditionally there have been two different kinds of engines, the Otto engine (the normal spark ignited gasoline engine) and the Diesel engine. Obviously there are enormous amounts of results and written publications regarding these two engine types and the best place in literature to start for the interested reader would be [Heywood, 1988]. This section briefly describes the Diesel engine and the Homogeneous Charge Compression Ignition (HCCI) engine which seem to be the most promissing combustion concepts in a world where demands on engine performance are ever increasing. The author makes no claims of writing an exhaustive description of combustion engines as such. Evenso they have to be briefly discussed in order to give the reader an idea of the frame within which this work has been undertaken. Introducing the reader to the topic will also reveal the relevance of this work.

The Diesel Engine

Diesel engines are becoming increasingly popular due to their high efficiency and thus low fuel consumption. The Diesel engine cycle involves compression of intake air during the compression stroke which increases the temperature and pressure. Fuel is injected directly into the cylinder after the compression stroke has almost completed. Diesel combustion is hence of diffusion type; [Andersson, 2008] gives a good introduction to Diesel combustion. The Diesel engine has high thermodynamic efficiency due to lean operation and high compression ratio. Fuel (to work) conversion efficiency is hence about 20% higher for a Diesel engine, compared to a corresponding gasoline engine. The main drawback with Diesel engines is higher emissions of NO_x and PM, which is caused by locally high temperatures and local excess of fuel in the fuel spray. Removing these emissions is more difficult in Diesel engines than in gasoline engines. A three way catalyst, which is used to remove almost all harmful emissions from gasoline engines, can not be used since Diesel engines are run with an excess of air (more air than needed to fully combust the fuel). There are a number of techniques to reduce the emissions of Diesel engines, including special catalysts and traps removing NO_x or PM. Essentially there is a tradeoff between NO_x emissions, PM emissions and efficiency in a Diesel engine. The legislation has been less strict for Diesel engines compared to gasoline engines for a long time but new legislation which is uniform for both gasoline and Diesel engines is however on the way forcing development of cleaner Diesel engines. Meeting future emission legislation for Diesel engines will require control of the actual engine and the combustion, as well as peripheral systems such as catalysts or exhaust gas recirculation.

The HCCI Engine

The Homogeneous Charge Compression Ignition engine was first suggested by [Onishi *et al.*, 1979]. It can best be understood as a hybrid between the traditional Otto and Diesel engines. Pure HCCI engines are operated with a homogeneous mixture of fuel and air, as an Otto engine. However unlike Otto engines, there is no throttling of the intake air, and there are no spark plugs. The fuel mixture is instead ignited by the increased temperature originating from compression of the intake charge, as in a Diesel engine. In theory this operation principle combines the high efficiency from Diesel engines with the low emissions from Otto engines. Soot emission is avoided because of the homogeneity of the mixture and the absence of locally rich combustion zones. Nitrogen oxide emission is avoided because of the decreased peak in-cylinder temperature due to the diluted operation of the engine and the absence of stoichiometric zones. In practice HCCI combustion can be obtained in a large number of ways, each with different benefits/drawbacks

compared to traditional Otto and Diesel engines.

Even though it has many good features, the HCCI engine also has some limitations. The operational principle unfortunately suffers from very high combustion rates, causing noise as well as wear on engine hardware. Another issue with the HCCI principle is low combustion efficiency at low load. This causes high emissions of unburned hydrocarbons and carbon monoxide. Since the HCCI combustion process is unstable in many operating points feedback combustion control is needed to operate an HCCI engine in parts of its operating range. Such combustion control can be performed in numerous ways using different actuators and sensors.

1.2 Combustion Engine Feedback Control

Internal combustion engines have, ever since they were first developed, been under feedback control. In fact feedback control of combustion engines is an 'enabler' for the success of the entire engine concept. An engine which can not deliver a controlled amount of energy at a controlled engine speed is of no use. In the beginning feedback control was achieved using mechanical means such as Watt's speed governor. The performance demands put on combustion engines are however increasing rapidly and the importance of combustion engine feedback control is increasing with the performance demands.

[Kiencke and Nielsen, 2000] as well as [Guzzella and Onder, 2004] provides very interesting sources of information regarding automotive control systems, particularly concerning mathematical modeling and feedback control of the engine and driveline. Mathematical models of various internal phenomena are commonly used when performing automotive control. Models are mainly used for three purposes; calibration, analysis and control. Physical models are often preferred over 'black-box' (empirical) models due to their more general nature, provided that assumptions and simplifications are well chosen. The calibration effort when porting a well designed physical model from e.g. one engine to another is much less than the corresponding effort to port a black-box model. Also, a good physical model may well be used outside its proven area of verification if the assumptions are still valid in the new context.

An early view of state of the art gasoline engine control was provided by [Powell, 1993]. For the Diesel engine similar work has just recently been undertaken, e.g. [Lewander *et al.*, 2008] who attempted cylinder pressure based control of a modern Diesel type combustion concept, using model predictive control. Many interesting results, starting with [Olsson *et al.*, 2001], have been published on HCCI control using various feedback control techniques. [Shaver *et al.*, 2004] utilizes a physically based model for HCCI con-

trol. Results using identified models instead of physically based ones have been published e.g. by [Bengtsson *et al.*, 2006], using Model Predictive Control (MPC) with an identified model for HCCI control.

Many of these authors used model assisted controllers, meaning that mathematical models of the combustion process and/or other sub phenomena in the engine system are contained in the control system, being continuously updated to reflect the current state of the engine system. The point is that embedded mathematical models play an increasingly important role in engine control to improve the performance of gasoline and Diesel engines, and to be able to implement new combustion concepts, e.g. HCCI. The nonlinear and unstable nature of HCCI combustion and other low temperature combustion concepts, together with the fact that actuators with high control authority sometimes are missing poses a great challenge to researchers in the field.

Cylinder Pressure Measurements

Cylinder pressure is a very powerful measurement signal when conducting engine feedback control and modeling. [Tunestål, 2000] quotes the late Professor A. K. Oppenheim who stated that cylinder pressure is like "the heartbeat of the engine" and measuring it is like carrying out "engine cardiology". The author recognizes this to be an excellent explanation of the importance of the cylinder pressure signal. From cylinder pressure all sorts of important parameters can be computed, and cylinder pressure is the most important measurement in this context.

Cylinder pressure is typically measured using a piezoelectric pressure transducer. The piezoelectric effect causes a quartz crystal to give away a small charge when exposed to an external force. Such a pressure transducer is typically connected to a charge amplifier which converts the small charge generated by the piezoelectric effect to a measurable voltage. A charge amplifier however can not be constructed without some leakage current and the leakage current will cause a drift in the DC level of the pressure signal. Due to this drift the output signal from the charge amplifier has to be treated in order to obtain the correct absolute level of the pressure. Several methods can be used to calculate the correct absolute pressure, [Randolph, 1990] accounts for three different commonly adopted methods. [Tunestål, 2007] offers a detailed explanation of how to treat the signal from a cylinder pressure transducer and intruduces a fourth, very accurate, method.

Heat Release Analysis

If the pressure within a cylinder is known it is possible to calculate the released heat within that cylinder using thermodynamic equations as shown by [Gatowski *et al.*, 1984], performing a Heat Release (HR) analysis accounting for losses. Losses accounted for includes heat transfer to the combus-

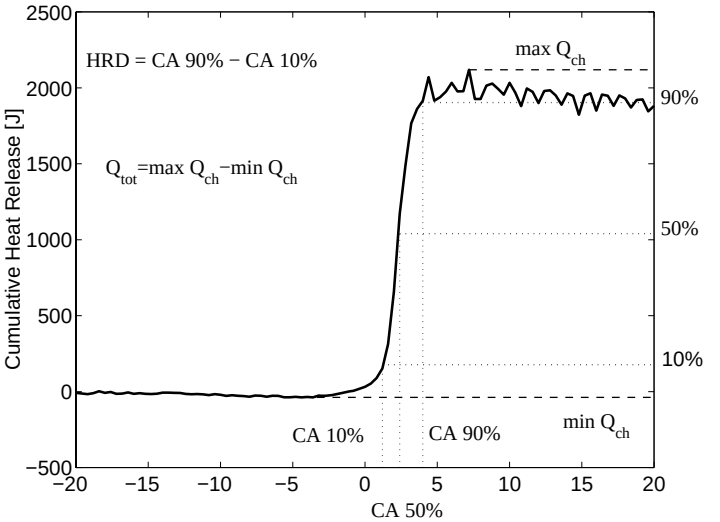

Figure 1.1 A typical heat-release trace (the integral of Eq (1.1)) with the important combustion phasing (CA50%) indicated. CA50% is defined as the instance when half of the total heat has been released (half of the combustion has taken place). Bottom axis in the figure has the unit Crank Angle Degree meaning that CA50% has the same unit.

tion chamber walls and mass loss caused by leakage past the piston rings. Heat transfer losses are often computed based on the results presented by [Woschni, 1967]. For feedback control purposes the different losses are often neglected as discussed by [Bengtsson *et al.*, 2004] The reason being that combustion phasing (see Fig. 1.1), which is the most important feedback control candidate calculated using HR analysis, can be calculated with enough accuracy even neglecting these losses. Controller complexity and hence execution time can in this way be reduced. This 'simplified' calculation is shown in Eq. (1.1), where p represents cylinder pressure, V

$$\frac{dQ}{d\theta} = \frac{\gamma}{\gamma - 1} p \frac{dV}{d\theta} + \frac{1}{\gamma - 1} V \frac{dp}{d\theta} \tag{1.1}$$

[Tunestål, 2000] develops a preintegrated form of Eq. (1.1) which has some numerical benefits compared to the original form of the heat release. Furthermore an auto-tuning heat release method was recently developed in [Tunestål, 2007], avoiding the main drawback with [Gatowski *et al.*, 1984] and [Woschni, 1967] which is *ad hoc* parameter tuning. This thesis contributes by implementing the preintegrated heat release in an FPGA, aiming for very high computation speeds.

NO_x Models

Controling Diesel combustion and/or after treatment systems is a challenge and mathematical models which are able to compute e.g. NO_x or PM formation and which can be implemented in the computing systems available in vehicles may be needed. To model PM emissions is more difficult than modeling NO_x emissions. Evenso there is, to this day, according to [Guzzella and Onder, 2004], no physical time resolved (non average value) NO_x model simple enough for on-board vehicle use. The reason is that physical NO_x models generally require a lot of complex computations which frequently are iterative by nature, e.g. nonlinear equation systems describing equlibrium chemistry. Engine controllers often do not have that much 'spare' computation time (obviously) and on board NO_x formation models are out of the question using conventional implementation platforms and models.

Several authors have made attempts at finding NO_x models which are physical (or at least semi physical) and still usable in engine control units e.g. [Andersson *et al.*, 2006] and [Ericsson and Westerberg, 2006], the second being based on the first. The claim of [Andersson *et al.*, 2006] is 'realtime' performance of the partially precalculated multi zone model. The evaluation platform was however a (for the time) high spec desktop computer, avoiding many of the obstacles present in on-board computation systems. The main idea with the models of the two authors is to compute the adiabatic flame temperatures in the different zones (more than two). These temperatures are then compensated in different steps according to partly physical, partly empirical relationships which are precomputed and stored in tables. The relationships are intended to correct the burned zone temperature for incomplete combustion, wall heat transfer and dissociation. Another NO_x model was developed by [Egnell, 1998] who utilizes conservation of energy iteratively to find a burned zone temperature. The iterative energy balance is solved partly in advance and tabulated as a function of global λ and EGR content. The chemistry part of the model, which is the part that computes acctual NO_x formation is based on an equlibrium chemistry model found in [Bensson and Whitehouse, 1979]. This thesis expands the previos work with a novel NO_x model which is inspired by previous publications but has a format making it much more computationally efficient.

2

Embedded Systems and FPGAs

2.1 Embedded Systems

The 'normal' processor technology is well known to many people, if not through its internal operation it is known for being the 'heart' of personal computers. A processor is a sequential device which executes a program consisting of a number of instructions. The number of clock cycles it takes the processor to compute a certain instruction differs depending on the architecture of the processor. Best case is one instruction per clock cycle but this is not the average rate of instruction completion. A processor is normally programmed using a high level programming language, each high level language instruction consists of several low level ones which in turn are executed on the processor. Hence, a high level instruction would normally need many clock cycles to complete.

The benefits with such an operational principle is that the processing device is very general. A large number of different programs can be composed using different instructions in different orders and it is possible to describe what the processor is supposed to do on a behavioral level, using a high level programming language. Many different problems can be solved on the same device without modifications to the actual hardware, the different programs can just be stored in a memory.

An Introduction to Embedded Systems

Embedded systems is a broad term which is used in many different contexts. The term is used to describe information processing devices (processors) embedded into products where the processor is not the main part of the product, even though it may be a necessary part. The presence of an

embedded processor is typically not obvious for the end user of the product. Computers (PC, server or computing cluster) are hence *not* embedded systems. There is a rich literature about embedded systems e.g. [Marwedel, 2003] or [Vahid and Givargis, 2002]. Examples of embedded systems could be mobile phones, DVD players, GPS receivers or ABS controllers and engine control units in cars. The last application is the one targeted in this thesis. There are many common properties which loosely define embedded systems, the properties which are relevant in this context are;

- Embedded systems are information processing systems embedded in a larger product.
- Embedded systems are often connected with the environment using sensors and actuators.
- Embedded systems have to be efficient, from a code size, run time or cost perspective.
- Embedded systems are often dedicated to one, single, application.
- Embedded systems must meet realtime constraints.
- Embedded systems are typically reactive, meaning that they respond to their environment and must execute at a pace decided by the environment.

Embedded processors typically perform tasks like control, user interaction and information flows handling. Humans in developed countries have daily contact with a large number of embedded processors (in the range of 50 or more) and embedded processing devices are present in virtually any product which has an electronic system.

Design Tools used in Embedded Systems

The tools used to develop software intended for embedded systems are essentially the same as the ones used to develop software for other processor environments. Programming languages such as C, C++ or JAVA can be used to write code describing the intended bahaviour of the program. Compilers are then used to make computer readable code from the human readable behavioral (or high level) code. Different compilers have to be used depending on which processor technology is targeted. Commonly the GNU C compiler is used when developing programs for embedded processors. The GNU C compiler exists in many different versions intended for different processor technologies.

Embedded processors frequently have some sort of operating system installed, most commonly rudimentary versions of Linux. It is however not necessarily so that an embedded processor uses an operating system. Many

Figure 2.1 An FPGA circuit.

embedded processors only run one single application which is made as one single C program. Another thing which is special with embedded processors is that user input and output is limited, hence some sort of communication has to be made available during development in order to allow debugging and software download.

2.2 FPGAs

An FPGA does *not* execute instructions at all as a processor does. An FPGA is a net of logic components which can be configured (and reconfigured) in a way so that the device performs a specific task. Input is presented to the FPGA device through input signals. The input signals propagate through the logic and internal connections of the device and finally the result is present on the outputs of the device. One strength of the FPGA technology compared to the processor is the inherited suitability for problems which are of parallel nature. Another strength of using FPGAs is that they commonly outperform a corresponding processor implementation in terms of e.g. speed. This is due to the fact that the actual design is made using the hardware directly and the overhead which is introduced in a processor in order to fetch and decode instructions etc. is avoided.

There are a large number of different considerations to make when implementing an algorithm or logic in FPGAs, such as hardware selection and design tool selection. [Todman *et al.*, 2005] and [Compton and Hauck, 2002] have published surveys of FPGA design considerations covering these topics, this section provides a brief overview.

FPGA History

The FPGA was first invented in the mid 1980s by Ross Freeman, who also was one of the founders of the large FPGA company Xilinx. Early FPGAs can be regarded as, often larger, versions of a similar device called Complex Programmable Logic Device (CPLD). The CPLD on the other hand is a larger and more modern version of the Programmable Array Logic (PAL). Leaving the rudimentary PAL out of the picture it can be said that it is not only the size that differs between the FPGAs and the CPLDs, the architectures differ as well. FPGAs have a more flexible architecture than CPLDs. FPGAs often feature a more complex interconnect between the internal units than CPLDs. Another difference might be that FPGAs often contain other components than pure logic functions e.g. distributed memory, adders, multipliers or other similar components, in many cases increasing the performance of the FPGA compared to the CPLD. FPGAs have evolved rapidly since the first ones and modern FPGAs can host designs with an equivalent gate count of many million gates. More and more complex peripheral devices are added to FPGAs, e.g. processor cores, Digital Signal Processing (DSP) blocks, even mixed mode FPGAs exist containing analog and partly analog parts, e.g. Analog to Digital Converters or analog filters.

Hardware Level Architecture

The internals of an FPGA typically consist of a large number of different (more or less) configurable functional units linked together by a net of configurable interconnect, together sometimes called a reconfigurable fabric (shown in Fig. 2.2). The functional units are 'were it happens' meaning that the actual logic is implemented using these functional units. The reconfigurable interconnect is used to transport intermediate results between the different reconfigurable units. Different FPGAs have different architectures for their reconfigurable fabric. Two extremes can be identified; fine grained and coarse grained architectures. The same kind of distinction in architecture exists both for the reconfigurable interconnect and the functional units. Selecting between fine grained and coarse grained interconnect and functional units essentially is a tradeoff between flexibility which gains from a fine grained architecture and speed or overhead (efficiency) gaining from a more coarse grained architecture. A fine grained fabric can be adapted to a larger variety of different tasks as the configuration possibilities are more detailed. A more coarse grained fabric on the other hand is not as adaptable but will be more efficient for the problems where they are well suited, as the existing hardware is more specialized for these cases.

A fine grained functional unit consists of a multiple input lookup table which can be configured to implement any logic function. These functional units are put together in clusters which in turn are interconnected via par-

allel connections with neighboring clusters. The reconfigurable interconnect provides global signaling to clusters positioned in other parts of the FPGA circuit. A coarse grained architecture would typically be multipliers, different kinds of arithmetic or logic units or large shift registers. These coarse grained blocks are considerably more efficient for their specific task compared to performing the same task using a number of fine grained lookup tables. The coarse grained blocks might on the other hand not be of any use at all if the application does not include e.g. a large multiplication. A reconfigurable fabric can be either homogeneous, meaning that the complete fabric consists of the same kind of functional units, or heterogeneous. A heterogeneous fabric contains different kinds of functional units instead of one type. A heterogeneous architecture can contain multipliers, adders and of course distributed processor cores, all this to complement the basic functional units and to increase performance. Modern FPGAs are normally heterogeneous, containg both fine grained and coarse grained blocks on the same circuit. The best mix between different functional units strongly depends on the intended application and the FPGA vendors supply different variants, which variant represents the best fit must be decided from time to time.

The reconfigurable interconnect connecting the different units within an FPGA can, as the functional units, either be fine or coarse grained. In a fine grained interconnect structure it is possible to control the routing wire by wire but with a coarse grained structure it would only be possible to route a bundle of wires per control bit. As before a coarse grained structure is less flexible but demands less overhead than the more flexible fine grained interconnect structure. On modern FPGAs there are normally several different communication meshes with different granularity forming some sort of overlapping 'network' on the FPGA. The intention with this overlapping communication network is mainly to decrease the power consumption of the chip (by decreasing the fan out of the logical units) as well as to enable a more efficient routing on the chip.

FPGA Design Tools and Methods

Basic FPGA design tools are essentially same as digital ASIC tools. Today there are two largely different 'levels' at which FPGA (as well as digital ASIC) design is carried out. One way to perform FPGA design is to use some sort of hardware descriptional language like VHDL. Hardware descriptional languages are fairly complicated to work with since the designer has to describe how to connect digital logic in order to obtain a certain functionality. When writing a computer program the programmer describes the actual behavior of the program, rather than which digital logic to use to achieve the desired bahaviour. Performing circuit design is hence a more complicated task than writing a computer program why it takes more time

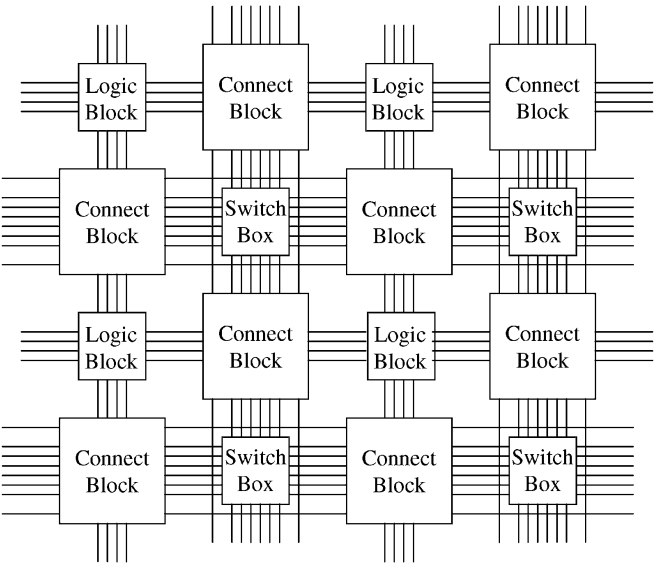

Figure 2.2 A typical generic reconfigurable fabric with switching units and functional units or logic blocks.

and more specialized knowledge. In recent years FPGA and ASIC technology have evolved very rapidly while the development of the design tools has not evolved as much. The consequence is that it is getting increasingly difficult to utilize the full performance of digital circuits, one speaks of the *design gap*. To meet the design gap new tools which enable circuit design on a higher level of abstraction (behavioral level) are emerging. The term for these tools is EDA, Electronic Design Automation. Typically these tools use commonly known programming languages such as C or C++ to perform circuit design. Before continuing it is crucial that the reader understands that even though traditional programming languages such as C, C++ or JAVA are used for high level hardware design, *no* program is run on the FPGA. Code written in these languages is synthesized into a connection scheme used for the reconfigurable fabric within e.g. an FPGA.

Low Level Design The most common design method for FPGAs or ASICs is to use some sort of hardware description language for describing the circuit at Register Transfer Level (RTL). Typically one of the two hardware description languages VHSIC Hardware Design Language (VHDL), (VHSIC =

Very High Speed Integrated Circuit) or the Verilog hardware descriptional language are used. VHDL is exhaustively described in [IEEE, VASG: VHDL Analysis and Standardization Group, 2007] and Verilog is described in the same manner in [IEEE, P1800, System Verilog Work Group, 2001]. Performing design at register transfer level means that the intended application is described using flip flops with combinatorial logic in between. These flip flops are usually triggered by the global clock and the result is hence a synchronous digital circuit implementing the intended functionality. VHDL or Verilog code can be written in a normal text editor much the same way as any other programming language. In most cases a more complete design tool would however probably be used. For most FPGA devices vendor specific toolboxes exist, normally allowing the designer to design circuits either purely from VHDL (or Verilog) or using a mix of VHDL code and some sort of graphical interface allowing the designer to partly draw the design using wires and different building blocks. These building blocks can either be defined by personal VHDL code or defined in advance in libraries, such blocks could be gates or flip flops. Once the code is finished it is synthesized to a netlist which in turn is treated by some sort of tool (a back end tool) which is specifically developed for adapting the netlist to the actual FPGA or ASIC technology used. The back end tool generates a physically realizable form of the design from the netlist. In the case of FPGAs the physically realizable form would typically be the configuration data which is to be loaded onto the actual FPGA (the bitstream).

High Level Design High level design tools typically generate an FPGA design based on a high level programming language or some other behavioral description of the intended design. [Todman *et al.*, 2005] shows examples using C, C++ or JAVA as host languages. These tools work according to three different principles. They are either annotation and constraint driven, annotation free or work according to the source directed compilation principle. The main idea with the annotation and constraint-driven approach is to use annotations and constraints in combination with the source code in the original language to generate the design. The annotation and constraint driven approach has the benefit that the source code originally intended for a normal microprocessor would only need minor modifications to be used as description code for hardware. Compilers which are annotation free also exist. General C, C++ or JAVA code can be used, meaning that no code revision is needed regardless of if the code is intended for hardware or for a micro processor. Annotation-free compilers have the great benefit that code can be moved from a processor core to an FPGA without modifying it at all, a property which is ideal when designing mixed processor hardware systems. The source directed approach on the other hand adapts the host language to better suit the FPGA environment for example by extending the

language with suitable operators and types.

Another type of high level design tools are those based on Data Flow Graphs. Data Flow Graphs (DFG) are typically used within the automatic control (technical computing) and DSP communities. There exist a number of DFG based FPGA tools which are specialized to suit development of DSP like systems, these tools can of course be used for other purposes as well. One tool which has to be mentioned here is Simulink (Simulink is an extensive plug-in tool for Matlab), Simulink is based on the idea of DFGs and is hence very well suited for development of DSP and automatic control systems. Since Simulink is a tool which is already very well established within the technical computing and DSP community and extensively used for other purposes than FPGA development many of the large FPGA manufactures have made plug in versions of their tool chains adapted for use together with Simulink, enabling hardware design within the well known and easy to use Simulink environment. Besides Simulink there are other tools which are based on the DFG principle which are specialized in or adapted to DSP design for FPGAs.

2.3 Embedded Processors, FPGAs and ASICs

FPGAs are said to be a 'poor mans ASIC' meaning that FPGAs are mainly used when it is considered too expensive to develop an ASIC with the same functionality, which is true in one sense. FPGAs are an attractive way to obtain near ASIC performance even in applications where traditional ASIC design is not feasible. However, FPGAs also have a number of benefits over traditional ASIC, the startup cost is much smaller and so is the time it takes to complete a design and have the products available on the market. In addition, FPGAs have some completely new features compared to ASICs. FPGAs can be reconfigured and the content of FPGAs can hence easily be changed during the lifespan of the system. Some FPGAs even have the capability to be reconfigured at runtime, meaning that it is possible to change the logic inside the FPGA on the fly depending e.g. on operation mode.

Naturally this added flexibility comes at a cost, according to [Kuon and Rose, 2007] FPGAs consumes about $18-35\times$ as much silicon area, $14-87\times$ as much dynamic power while it is $3.4-4.6\times$ slower than the corresponding ASIC implementation. Nevertheless FPGAs can be expected to increase their importance as a high performance digital platform. Probably not mainly as a platform for high performance pure digital logic but as a versatile implementation platform for e.g. embedded processing systems. A heterogeneous FPGA chip including embedded processor kernels large amounts of memory and reconfigurable FPGA fabric, all embedded on a single chip (so called system on chip) is a very attractive solution for many applications.

In many FPGA systems, current and future, some sort of processor kernel might well be present. Several different motivations exist for mixing processor(s) and FPGA(s) in a single system. FPGAs, or synchronous digital logic, are not well suited for all types of tasks, spontaneous, iterative tasks with variable length, e.g. loops or control of data flow, are difficult to implement efficiently in an FPGA. Such tasks are preferably implemented on a processor while computationally expensive, time critical and parallel tasks benefit greatly from an FPGA implementation.

There are five different ways of integrating a processor core with the reconfigurable fabric of an FPGA, Fig. 2.3 shows four of them identified by [Compton and Hauck, 2002], the last one was added by [Todman *et al.*, 2005]. Communication between the processor and the reconfigurable fabric may be established through the standard Input/Output (I/O) units of the processor. The FPGA is then called a 'standalone processing unit'. There are two types of intermediate structures, meaning that the processor can access the reconfigurable fabric without having to use the standard I/O units. The variant called 'attached processing unit' is, communication wise, a bit slower than the version called 'Co processor' which features direct communication between the processor core and the reconfigurable fabric. The fourth architecture, called FU (functional unit), features a reconfigurable fabric which is present on the same chip as the processor. The functional unit can be connected to the internals of the processor in different ways. Such an architecture enables very high communication speed between the processor core and the reconfigurable fabric. The fifth possible way to connect a processor and reconfigurable fabric resembles the fourth architecture, but instead of adding a piece of reconfigurable logic to a processor, a processor is added (implemented) onto the reconfigurable fabric of an FPGA. The two last versions of FPGA processor connection schemes can be altered in two ways, the processor can be either a hard core meaning that a fixed part of the device contains a processor which is constructed on the same chip as the reconfigurable fabric, or a soft core. Soft core processor meaning that the processor structure is implemented on the reconfigurable fabric, the complete chip hence consists of reconfigurable fabric, but on a piece of that reconfigurable fabric a processor is constructed using the reconfigurable fabric for its implementation. One of the benefits with the latest two design structures is that it is possible to customize the actual processor internals, e.g. to add custom instructions to the instruction set of the processor. This principle is called soft instruction processors or flexible instruction processor.

The nature of the applications does of course decide which of the five different architectures above that is to prefer. [Compton and Hauck, 2002] provides a very good description of the benefits and drawbacks with the different structures. Using Architecture one for example the communication between the processor and the reconfigurable fabric is slow, hence this ar-

chitecture is suitable for systems where large chunks of work can be treated by the reconfigurable fabric independently. The attached processing unit of architecture two has the same properties as a normal processor would have had in a multi processor system. Architecture three, the Co processor architecture, is typically also capable of performing calculations independently from the processor core but it has access to the same memory and other facilities as the processor core. Architectures four and five would be best suited when communication between the processor and the fabric needs to be vigorous, for example when the reconfigurable fabric is used to customize the processor in some way and communication hence needs to be very efficient. According to [Todman *et al.*, 2005] systems according to Architecture one are the most common in commercial FPGA platforms.

Combining reconfigurable logic with embedded processors can surely be powerful. There is however not one single best way to put together these systems. Deciding whether to use a conventional embedded processor, a mixed custom system implemented either on an FPGA or on an ASIC must be made depending on the intended application, production volume, algorithm complexity, speed grade etc. In an application where speed or power consumption is not the main focus a normal embedded processor may be enough, providing the cheaper and more flexible solution. If speed is of importance but the production volume is low an FPGA system might be the best way to go. And when demands on speed as well as power consumption are large and the product will be produced in large volumes ASICs would probably be the way to go. Each of these technologies have their application window but it is safe to conclude that the application window of FPGAs will increase in the future. Design of embedded systems are gradually developing from being just design of software intended for processors embedded in products to some sort of software/hardware co design.

2.4 Algorithmic Considerations for Embedded and FPGA Implementation

Many of the considerations which have to be made when implementing algorithms using embedded processors and micro controllers are similar to the ones which have to be made using FPGAs and ASICs. Some considerations and tradeoffs are however different. Handling of dynamic fixed point number format, removing 'troublesome' operators and extracting parallelism in the algorithms are suggestions given by [Monmasson and Cirstea, 2007] aiming for a more efficient implementation of (control) algorithms using FPGAs. Basically the same things are important when implementing algorithms in embedded processors, besides extracting parallelism in the al-

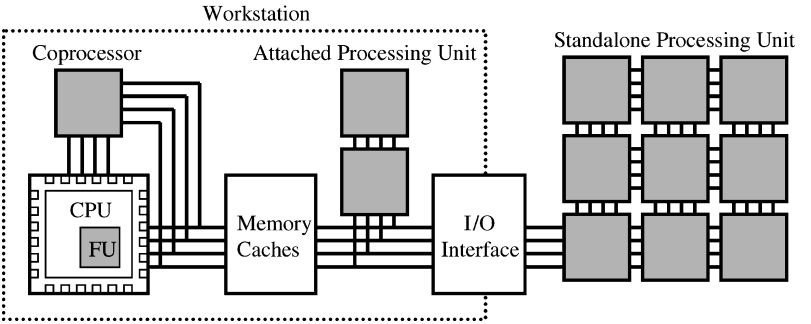

Figure 2.3 Four different architectures commonly used in mixed processor/hardware systems. The different architectures have different properties regarding for example communication speed and flexibility and are hence suitable in different situations.

gorithm which is not needed in a single processor environment.

Algorithm Reformulation

Implementing efficient algorithms in embedded systems, DSP or FPGA environments takes a lot of consideration since the infrastructure in these systems commonly is more limited compared to desktop computers. Hence, the algorithm has to be written to suit the intended target system. The best way would be to keep the limitations of the final system in mind throughout the algorithm implementation process, commonly algorithms are however adapted to the system after they are fully developed. Examples could be that embedded processors lack floating point support or that the word length available to represent numbers is limited. Certain operators are more demanding e.g. multiplication and division, these operators should be replaced with less demanding operators if possible. Multiplication and division by numbers which are powers of two can be replaced with bit shift. Complex operations such as square root, natural logarithm etc. are preferably computed offline and tabulated. These measures can be taken even if the system used actually has the possibility to perform the desired computations, in order to gain efficiency. Frequently floating point arithmetics e.g. takes more time than fixed point arithmetics, especially if the processor/hardware lacks floating point support. The execution time which is saved in this way could mean the difference between being able to use a cheaper low-spec processor or a more expensive high-spec processor.

Avoiding difficult arithmetics is a good idea even when using FPGAs. Even if dedicated digital logic could be designed to compute e.g. troublesome operators, it will take a significant effort to implement unnecessarily

difficult logic. The simplified version of an algorithm would still be more efficient even in an FPGA environment.

Number Representation

Internal number representation is important whether using fixed-point numbers for algorithm implementation or algorithms implemented with floating-point numbers. Floating-point numbers are convenient to use since the programmer or designer does not need to worry about which scaling different variables have. However, floating point number representation is frequently less efficient than the corresponding fixed point number representation due to the fact that floating point numbers come with a larger over head. This can be said both from a calculation speed point of view and from a chip area point of view, both using embedded systems and FPGAs. Hence algorithm implementation is preferably made using fixed point arithmetics in these environments, to gain speed and save resources.

Implementing algorithms in FPGAs provides the possibility to arbitrarily select the internal number representation within each step of the calculation. The same is not possible in processors where it does not make sense to use a word length which isn't a multiple of the 'natural' word length (word size) of the processor architecture used. In FPGAs or ASICs it is on the other hand possible to find an optimal fixed point representation for the internal data path, given an algorithm. An optimal internal data path means a data path which supplies sufficient accuracy using minimal internal word length. Finding the optimal word length within each step of the calculation is a difficult optimization problem which is subject to research efforts, e.g. [Hervé *et al.*, 2005] and [Cantin *et al.*, 2006]. Excessive word length consumes memory, time, power and chip area. [Cantin *et al.*, 2002] states that in complex DSP designs, as much as 50% of the design time is spent deciding the internal number format in different steps of the algorithms. It is obvious that it is desirable to automate the time consuming and error prone task to decide internal number representation of an algorithm. The above mentioned publication contain a survey of ways to automate the search for an optimal internal number representation and [Todman *et al.*, 2005] also discusses this subject. Normally the word length of the integer part is decided based on the dynamic range needed, deciding the word length of the fractional part takes some more consideration. Three different approaches can be used to select the fractional word length, it can be decided based on analysis of DFGs, it can be decided based partly on analytical methods and partly based on simulations and it can be selected based on methods relying only on simulations. Deciding the number representation of an algorithm based on simulation is performed by firstly simulating it using floating point number representation (so called 'golden model'). Fractional word length is then decided based on the simulation and a fixed point simulation is run. The outcome

of the fixed point simulation is compared to the golden model e.g. using an error function. The process of deciding fractional word length, performing fixed point simulation and evaluating the outcome can be iterated e.g. until system specifications are met. The criterion for stopping the iteration can be expressed as an error function value or as a limit of the difference between the fixed point and floating point simulation.

The simulation based methodology has the drawback that it does not guarantee that no overflow will occur or that the specification will be met for any other cases than the ones reflected by the 'test bench' used. The reason is that the test bench in most cases can not reflect all possible inputs encountered during the complete lifetime of the system and performing simulations using a test bench that covers the complete possible set of inputs would be too time consuming. There are a few ways to improve the performance of the test bench in order to make it more likely that most of the input dynamics are taken into account during the simulations. Pseudo random inputs can be used in the test bench. In some cases it might be possible to calculate the mean and standard deviation of each operand and, based on the results, add bits to the data path to avoid overflow. It is possible to increase the constraint more than necessary to gain some 'margin' for overflow. One way is to perform 'cascade simulation', where a possible data path is found using a very limited test bench. A test bench that is as complete as can be tolerated is then used to verify this data path for a large part of the input range. The gain with 'cascade simulation' is that it is not necessary to perform every data path iteration using an extensive test bench thus saving simulation time.

Extracting Parallelism

Extracting parallelism in an algorithm is a way to speed up computations in FPGAs or ASICs. If there are parallel structures in the algorithm, meaning two or several parts of the algorithm which can be computed independently at the same time, it is possible to implement these as parallel computation branches on FPGAs or ASICs. The same thing is possible in multiple processor embedded systems but not in single processor systems. Implementing algorithms in parallel is beneficial from a computation time point of view but it has the drawback that more hardware is often needed compared to a non parallel approach.

If the designer has detailed background knowledge of the algorithm it is possible to manually extract parallel branches. However, there are also a number of automated tools, as described in [Damaj, 2006], which extracts parallelism in algorithms. The benefit with automated tools is that they are able to detect parallelism at levels which are very difficult to detect, even with detailed knowledge about the algorithm.

2.5 Summary

This chapter covers embedded processors and FPGAs. The term embedded systems was defined and several properties common for embedded systems were discussed. FPGAs were described a bit more in detail, being the less known system. Strengths, issues, design considerations and design tools have been discussed. An FPGA is a reconfigurable hardware device and it works in the same way as any electric circuit; signals propagate through it. Comparing FPGAs to embedded processors it is found that the embedded processor is a more general device since it can carry out a set of general instructions, this property makes the processor flexible and easy to develop software for. An FPGA is more difficult to develop designs for, the possible performance of the implementations is however greater using FPGAs.

Design of software intended for embedded processors are typically performed using some sort of behavioural computer languare such as C or C++. When designing applications for FPGAs the connection between different components on the FPGA is specified using some sort of hardware descriptional language. Hence FPGA design is carried out using a structural description rather than a behavioural one, tools are however emerging which enable behavioural FPGA design using e.g. C.

This chapter ended with a number of algorithmical considerations which are suitable to keep in mind when implementing logic in FPGAs as well as algorithms in embedded processors. When implementing computations in these environments it is important to find an efficient representation of the internal numbers. To find a good number representation can be very difficult and time consuming. It is also beneficial to remove operators which are more demanding e.g. division and to try to rewrite the intended algorithm so that it can be efficiently implemented e.g. using precomputed lookup tables.

The following two chapters presents two different case studies implementing control related algorithms utilizing the methods described above to perform efficient implementation of the algorithms.

3

An FPGA Implemented Heat Release Computation

For successful control of e.g. HCCI (and other low-temperature combustion concepts) it is the common opinion that a closedloop combustion control system with fairly large complexity is needed as described in Chapter 1. One or more models/calculations will need to be maintained online by the closedloop combustion control system. This and the ever increasing complexity of normal engine control systems constitute the background to the proof of concept study to be presented in this chapter. From the previous chapter it is understood that an FPGA could well serve as a very useful tool for implementing e.g. closedloop combustion control systems and their strengths are extra valuable in situations when speed is important. The intention with this chapter is to show this possibility and at the same time implement a part of a future closedloop combustion control system, namely the heat release analysis. The reasons for implementing a HR computation rather than any other computation are twofold. The HR, or rather the combustion timing CA50% which is calculated from the HR, is considered the most important feedback variable in engine control in general and in HCCI control in particular. HR calculation is, from an automotive perspective, regarded as a computationally expensive operation which in many cases can not fit within the highly loaded normal engine control units. It is hence desirable to show a method to calculate HR in a fast and accurate way, not loading the engine control unit and with a speed unmatchable by normal engine control units or other processor based embedded systems. The potential of FPGA implemented computations/models and the concept of using FPGAs for control oriented modeling/computation is simultaneously proven by this proof of concept study.

3.1 Experimental Setup

The experimental setup necessary for these experiments consists both of software and hardware. The hardware parts are made up of an FPGA prototype board, a signal simulator simulating the engine pulses and cylinder pressure and an expansion module for the FPGA board featuring Analog to Digital Converter (ADC) and Digital to Analog Converter (DAC) and some surrounding circuitry infrastructure. The FPGA system is connected to a Personal Computer through a JTAG cable, enabling display of debugging data, FPGA/PC co simulation and reconfiguration of the FPGA. The design of the FPGA configuration and the hardware co simulation were carried out in Matlab/Simulink with the aid of a Simulink toolbox supplied by Xilinx, 'Xilinx System Generator DSP' (SGDSP). In order to generate an FPGA design from the Simulink diagrams the Xilinx development suite 'Xilinx ISE' was also needed.

FPGA System

The main part of the setup is the experimental card fitted with the FPGA. The card is a Commercial Off The Shelf (COTS) product supplied by 'Avnet', the (rather long) name of the card is 'Memec Xilinx Virtex-4 LX XC4VLX25-SF363 LC Kit'. As understood the card holds a 'Xilinx Virtex-4 LX XC4VLX25-SF363 LC' FPGA, which holds 24,192 logic cells, 168 Kb distributed RAM memory and a maximum clock speed of 100 MHz. Besides the above mentioned the card holds a number of peripheral devices; 16 Mb Serial Flash for FPGA configuration, 64 Mb SRAM and a 'P160' expansion header, among other things.

Desired FPGA configuration is either loaded directly onto the FPGA or stored in a serial flash memory. If the configuration is stored on to the serial flash it is automatically reloaded onto the FPGA at power up. There are of course other solutions for the FPGA reconfiguration that may be more suitable in production automotive applications. Configuration both of the serial flash and directly of the FPGA are carried out through the JTAG interface.

The necessary ADC and DAC were not included on the basic FPGA board. To gain access to the analog world, a circuit board was fitted in the 'P160' expansion header. This board was, as the FPGA board, supplied by 'Memec/Avnet' and named '160 Analog Kit'. The board featured dual ADCs and dual DACs all made by 'Burr-Brown', the ADC was a high speed 12 bit pipelined (a refinement of the successive approximation technique) ADC, the DAC also had a resolution of 12 bit. The clock signal was supplied from the FPGA which runs at 100 MHz, limiting maximum ADC_{clk} to 50 MHz and DAC_{clk} to 100 MHz. The surrounding circuitry, on the ADC channels, consisted of an input buffer, a low-pass filter and a single ended to differential amplifier. The differential amplifier had an output buffer and a low-pass

output filter. The expansion card was less suited for the intended application than expected at the time of purchase and modifications were made to the input circuitry of the AD channels. The issue was high-pass filter action on the input which removed the relatively low frequency that was of interest in this application, the DAC channels were however left without modification.

The total FPGA system price was at the time of purchase $\approx$ €700 and must hence be considered as a low cost system, it was nevertheless a high performance system!

Design Tools

As briefly mentioned earlier the design of the FPGA HR algorithm was carried out in Matlab/Simulink with the help of SGDSP. SGDSP contains a number of Simulink blocks implemented in both Simulink and VHDL. Using these blocks it was possible to generate VHDL from a Simulink diagram. FPGA layout with the help of Simulink in DSP applications was discussed by [Todman *et al.*, 2005]. Please note that it was not possible to implement standard Simulink blocks in the FPGA, it was necessary to possess a VHDL implementation of the blocks to implement, which was somewhat limiting. Running the hardware co simulation (compare Hardware In the Loop simulation HIL) during the debug process, it was however possible to implement standard Simulink systems on the PC, communicating data with the FPGA through the JTAG interface. This was a handy feature during the debugging process. After co simulation of the system, generic VHDL files could be created from Simulink, processed by the relevant FPGA design tools and downloaded to the serial flash.

Test Environment

Desktop tests were carried out during the development. The interface to the simulated/real engine consisted of three signals, Crank Angle Degree Pulses (CADP), Top Dead Center Pulses (TDCP) and analogue cylinder pressure p. Current engine position was assumed to be measured with 0.2 Crank Angle Degree (CAD) accuracy, the engine (simulator) hence produced 5 CADP every physical CAD. Besides CADP the angle sensor was assumed to produce one TDCP every time the engine completed a thermodynamic cycle, i.e. two revolutions for a four stroke engine. Fig. 3.1 provides an overview of the experimental setup. This engine interface is similar to the setups of [Olsson *et al.*, 2001], [Bengtsson *et al.*, 2004] and [Bengtsson *et al.*, 2006].

During the development and testing p was simulated using recorded cylinder pressure traces, both a motored and a fired trace were available. The basis for p were pressure traces recorded from the 12 l Scania engine used by [Olsson *et al.*, 2001], geometric data of this engine can be found in Table 3.1. The engine simulator simulated CADP, TDCP and p synchronously at an engine speed of 1200 rpm. p was simulated with a vertical resolution of 8

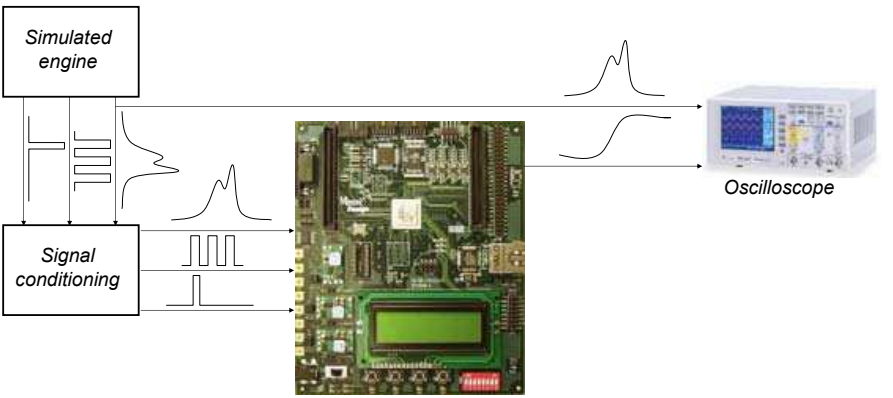

Figure 3.1 FPGA experimental system overview.

bit, horizontal resolution was 720 samples, that is one sample each CAD. Digital to analog conversion was carried out with the help of a R/2R ladder and the output of the device was buffered. The original pressure trace was somewhat distorted due to the limitations of the engine simulator which was developed in house.

3.2 FPGA Layout

The design that was implemented in the FPGA can be viewed as a DSP design, no processor core was present in the system. It would however of course be possible to implement the HR algorithm on the reconfigurable fabric of a mixed processor/hardware system. The DSP design approach was selected due to the relative simplicity of the calculations and the system.

Algorithm

A net HR (Q_{HR}^{net}) calculation was implemented on the FPGA, that is the HR calculation disregards heat transfer losses, crevice losses and blow by. Heat transfer losses are typically caused by convective energy loss to the combustion chamber walls. Crevice losses are caused by trapping air and fuel mixture in the crevices between the piston and the cylinder wall, thus getting cooled down and avoiding combustion. Blow by is fuel and air that escapes the cylinder past the piston rings down to the crank case. The reason for neglecting all losses in this work was to somewhat simplify the implementation. Since [Bengtsson *et al.*, 2004] finds that Q_{HR}^{net} is sufficient for feedback purposes this simplification was regarded as legitimate.

The calculation of Q_{HR}^{net} was carried out in a non conventional manner. The conventional way to calculate Q_{HR}^{net} is through the integration of Eq. (1.1) as described by [Gatowski *et al.*, 1984]. For signal processing applications it is however inconvenient to include the pressure derivative in the calculation since the process of differentiating a measured signal infers severe noise issues, especially in combination with a high 'over-sampling' rate. Instead of using Eq. (1.1) it was possible to calculate Q_{HR}^{net} through Eq. (3.5). Eq. (3.5) originates from the conservation of energy and is motivated through Eq. (3.1)-Eq. (3.4). This promising optional method to calculate the HR was first presented in [Tunestål, 2000].

$$\left.\begin{array}{l} dU = dQ - dW \\ dU = nC_v dT \\ dW = pdV \end{array}\right\} \Longrightarrow nC_v dT = dQ - dW \Longrightarrow dQ = nC_v dT + pdV \quad (3.1)$$

$$pV = n\widetilde{R}T \Longrightarrow \frac{1}{n\widetilde{R}} d(pV) = dT \quad (3.2)$$

$$C_v = \frac{\widetilde{R}}{\gamma - 1} \Longrightarrow nC_v dT = \frac{n\widetilde{R}}{\gamma - 1} dT \overset{(3.2)}{\Longrightarrow} nC_v dT = \frac{1}{\gamma - 1} d(pV) \quad (3.3)$$

$$dQ = \frac{1}{\gamma - 1} d(pV) + pdV \quad (3.4)$$

$$\begin{aligned} Q &= \frac{1}{\gamma - 1} \int_{\theta_{start}}^{\theta} d(pV) + \int_{\theta_{start}}^{\theta} pdV = \\ &= \frac{1}{\gamma - 1} (p(\theta)V(\theta) - p(\theta_{start})V(\theta_{start})) + \int_{\theta_{start}}^{\theta} p(\theta) \frac{dV}{d\theta} d\theta = \\ &= \underbrace{\frac{1}{\gamma - 1} p(\theta)V(\theta) + \int_{\theta_{start}}^{\theta} p(\theta) \frac{dV}{d\theta} d\theta}_{\text{Calculated on-line}} - \underbrace{\frac{1}{\gamma - 1} (p(\theta_{start})V(\theta_{start}))}_{\text{Estimated constant, added off-line}} \end{aligned} \quad (3.5)$$

Implementation

Fixed point arithmetics was used throughout the implementation of the HR calculation although no automatic word-length determination was used (or available in Simulink at the time). Fixed point settings were determined *ad hoc* by studying internal signals during the implementation, no large simulation with the purpose of finding optimal fixed-point settings was carried out. To avoid racing phenomena the different internal computational branches were synchronized by adding unit delays and Simulink-specific synchronization blocks in the data path. Keeping Eq. (3.5) in mind, two multiply operations, $p(\theta)V(\theta)$ and $p(\theta)dV/d\theta$, were computed in parallel within the FPGA. The integration (summation) acting on $p(\theta)dV/d\theta$ was by necessity computed subsequently and $p(\theta)V(\theta)$ had to be delayed one sample to make up for the time it takes to update the summation, all to avoid racing. Finally, the two parts of Eq. (3.5) are added to obtain the part of the result which is calculated online. When Eq. (3.5) was implemented the parts of the equation that were known in advance were not calculated online in order to simplify the equation as much as possible, thus gaining speed. Instead they were mapped as a function of current engine CAD and stored in the distributed RAM of the FPGA, this goes for V and $dV/d\theta$. The algorithm output was not active during the complete engine cycle, Q_{HR}^{net} was enabled between 300 CAD to 400 CAD (a complete engine cycle is 720 CAD).

The time resolution is also of high algorithmic interest and this has to be noted. The described setup had the properties of an asynchronous system since the engine delivers CADPs at one clock speed (which varies with the engine speed) and the FPGA board ran at a totally different one, meaning that the FPGA clock was asynchronous with the CADP. This is an unconventional approach in an engine control system. The described calculation of Q_{HR}^{net} demands, as understood from Eq. (3.5), some synchronization between the calculation of Q_{HR}^{net} and the engine position (in order to retrieve the correct V and $dV/d\theta$). This synchronization was provided through a CADP counter, a simple incremental counter which was reset on the rising edge of TDCP. Overrun detection was also provided on this CADP counter in order to detect issues with the CADP and TDCP. The output of the CADP counter indicated the current position of the engine and it was used as index for the tabulated values of V and $dV/d\theta$. In this manner the FPGA system had all the information needed for the calculation of Q_{HR}^{net} without synchronizing FPGA clock pulses with the CADP.

The clock speed of the ADC was, as previously noted, 50 MHz. This was hence the sampling rate of p. This is by far a higher sampling rate than the update speed of V and $dV/d\theta$, meaning that a number of Q_{HR}^{net} samples were calculated in vain. The outcome was a system that calculates Q_{HR}^{net} based on the same p, V and dV values several times, a system with very high rate of

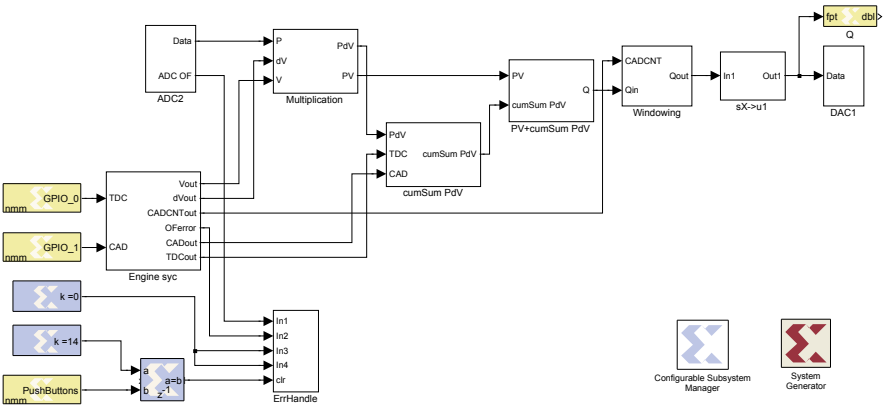

Figure 3.2 The DFG implementing the HR. The figure describes the top level of the implemented algorithm/system, it is important to note that each block at this level contains a lot of logic's. The two different parallel branches making up the computation are not obvious from the top view.

over sampling. The parallel nature of the FPGA however still made this the preferred way to perform the design. The FPGA simply outputs Q_{HR}^{net} samples at 50 MHz no matter what, meaning that the inferred latency, counted in number of FPGA cycles was constant regardless of the engine speed. The commonly used method to synchronize the engine and the control electronics by clocking the control electronics from engine driven pulses (i.e., CADP) was not used here since it would completely destroy the largest benefit of using the FPGA, which is low computational latency. It was not at all necessary to synchronize the clock of the FPGA with the CADP since it was possible to maintain sync with the engine with help of the CADP counter. Disregarding the synchronous thinking between the engine and FPGA board enables very low latencies without any major drawback.

3.3 Experimental Results

The major result of this investigation was of course the successful implementation of the described HR algorithm in the FPGA environment. The implementation was very small in terms of occupying resources on the FPGA, and an equivalent gate count of 14400 gates which means that approximately $2-4\%$ of the resources on the FPGA were needed. The performance of the implementation is a point that can not be stressed enough, a p sample that arrives to the FPGA from the ADC is, considering the timescale of an engine,

Number of Cylinders	6
Swept Volume	11 705 [cm^3]
Compression Ratio	18:1
Bore	127 [mm]
Stroke	154 [mm]
Connection Rod	255 [mm]

Table 3.1 Geometric properties of the Scania engine.

$p_{lsb} = 1464.84$ [Pa]	$V_{lsb} = 5.05 * 10^{-7}$ [m^3]
$dV_{lsb} = 1.74 * 10^{-9}$ [m^3/CADP]	$Q_{lsb} = 3.85$[J]

Table 3.2 the resolution of the input and output variables.

calculated immediately. Immediately in this context means 12 FPGA clock cycles, $FPGA_{clk} = 100$ MHz $\rightarrow$ 120 ns. This, in other words, means that when the ADC has delivered a sample on the FPGA pins it takes 120 ns before the FPGA has delivered the corresponding Q_{HR}^{net} sample on the pins of the DAC! In 120 ns, an engine revolving at 1200 rpm moves 0.000864 CAD. If the engine were revolving at 24000 rpm it would move 0.01728 CAD! The latency between an arriving p sample and the output of the corresponding Q_{HR}^{net} sample is in other words negligible. Since it is possible to measure the Q_{HR}^{net} with as high frequency as p and concurrently with p, it is well motivated to call the system a 'virtual Q sensor' meaning that Q_{HR}^{net} is calculated the instant p is measurable. It is difficult to measure the latency through the FPGA with standard measuring equipment, the calculation of the latency is based on the fact that it is possible to extract precise latency information from the Simulink layout. To give an indication to the numbers mentioned Fig. 3.3 is included. Fig. 3.3 shows p output from the engine simulator together with the output of the FPGA (or virtual Q sensor), the fist cycle is a motored cycle and the following cycle is a fired one. p and Q_{HR}^{net} are synchronously sampled, the sampling is not halted and no data is cut away between the two corresponding cycles. It is easily understood from Fig. 3.3 that Q_{HR}^{net} is calculated with very low latency with respect to p.

Besides very low latency the system features a very high throughput. If the engine would be able to produce pressure at a rate high enough it would be possible to perform 12 complete (120*5 points) HR analysis each CAD at 1200 rpm. The limit of the throughput is the ADC conversion speed.

The last point of the results is the correctness of the output. To verify

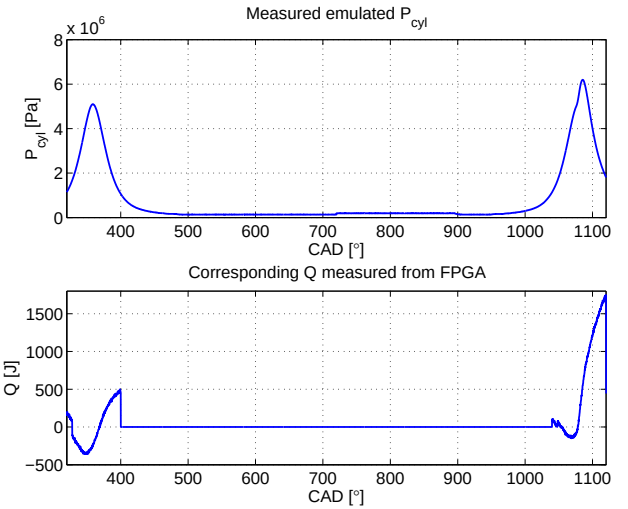

Figure 3.3 Output from the FPGA system when the combustion is suddenly switched on, showing true in cycle performance. Q_{HR}^{net} was computed between 300 CAD and 400 CAD.

the correctness 100 cycles are sampled to the PC. Due to poor quality of the pulses originating from the engine simulator some cycles however had to be removed due to sudden steps in the middle of the calculation. The number of disregarded cycles are in the range of 20-30 cycles depending on the mood of the engine simulator. Fig. 3.4 shows all the sampled cycles that are not damaged. As understood from the figure most of the calculated cycles hold a high enough signal quality to be used in a feedback control loop. Fig. 3.5 shows the average of the cycles shown in Fig. 3.4. It also shows the corrected values calculated offline using Matlab. As understood from the figure the FPGA implemented algorithm was able to calculate Q_{HR}^{net} accurately enough. When FPGA output and Q_{HR}^{net} computed offline using Matlab, were compared to a Q_{HR}^{net} computed using Eq. (1.1) they were found to agree as visble in Fig. 3.5.

3.4 Discussion

The results in Fig. 3.3 - Fig. 3.5 show that implementation of Eq. (3.5) was successful. As previously noted some cycles were removed from the results before presentation. The intention of the final system was of course that every cycle would be calculated perfectly, which also is achievable, better edge

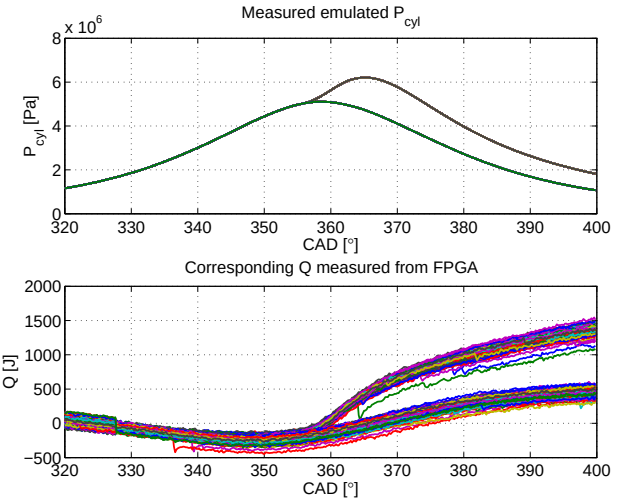

Figure 3.4 Non average results corresponding to figure 3.5. Q_{HR}^{net} was computed between 300 CAD and 400 CAD.

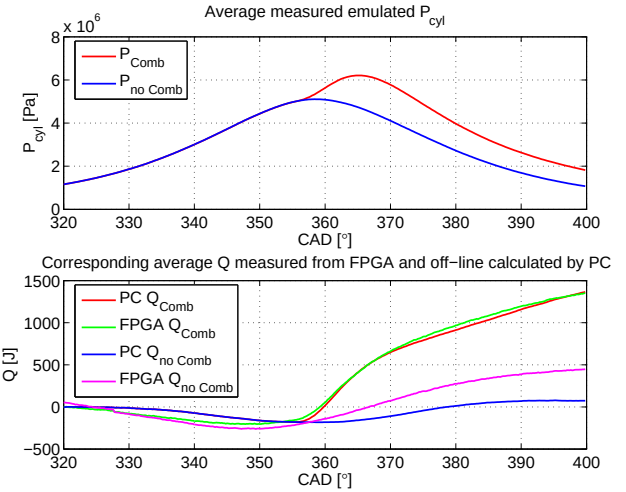

Figure 3.5 Average output from the FPGA system compared to the corresponding values corrected in the PC.

detection logic in combination with better quality of the positioning pulses would do enable this. The reason for the erroneous cycles are considered mainly to be issues with the engine simulator. As it was difficult to find a suitable signal simulator, the simulated engine had to be developed in house, and even though it performs acceptable on average some cycles are not true to the real engine and it is these cycles that were removed. The cycles were removed based on a derivative threshold of the measured Q_{HR}^{net}. As shown in Fig. 3.4 some bad cycles do however still persist (bad cycles meaning cycles that have a sudden jump in the middle of the signal).

Besides the issue with bad cycles an offset problem was present, in Fig. 3.4 there appears to be different bias on the different cycles of Q_{HR}^{net}. This offset problem was explained by the ADC circuitry infrastructure which was intended for usage in very high frequency systems. The signal paths to the ADC hence blocked the very low frequency cylinder pressure signal. The input circuitry thus had to be replaced by an in house alternative which unfortunately suffered from a slight problem with the input bias which explains the 'cycle-to-cycle' variation of Q_{HR}^{net} in Fig. 3.4.

Removing erroneous data and computing average over the correct cycles gave Fig. 3.5. Fig. 3.5 shows that the FPGA system despite the noted issues on average performs well. The difference between Q_{HR}^{net} calculated offline using Matlab and Q_{HR}^{net} calculated by the FPGA was not large, at least not in the case with combustion. Q_{HR}^{net} in the motored case proved less consistent between the Matlab computations and the FPGA. The explanation for this was considered to be the bias present on the input of the ADC. This bias issue will strike differently depending on the signal level. Since the signal level is lower in the motored case the error will also be larger.

Most of the difference between the corrected Q_{HR}^{net} and the FPGA Q_{HR}^{net} can be explained by the above noted problem. The fact that fixed point numbers had to be used in the implementation is not thought to account for any large error in the signal as motivated by Table 3.2. From the table it is clear that despite the limited input word length of only 12 bit the resolution (represented by the value of the Least Significant Bit (LSB)) was enough. p_{lsb} was calculated based on a 60 bar p_{max} assumption. If p_{max} would be 200 bar it would give $p_{lsb} \approx 0.05$ bar).

Even though the resolution was regarded as accurate enough the FPGA environment still caused some problems during the implementation phase. System generator DSP is a tool that should be used with care; both FPGA internal number representation and the internal timing of the FPGA were very difficult to manage. The transparency of the tool is simply not good enough to enable the designer to design the FPGA layout in a controlled manner. Many issues had to be solved *ad hoc* causing delays in the design work.

3.5 Summary

An FPGA implementation of a HR model was performed and discussed. The HR was selected before other models since combustion phasing calculated from the HR is considered to be the most important feedback candidate for combustion control. A reformulated HR was implemented, removing the pressure derivative from the equation due to noise issues. The computation was implemented on a Xilinx FPGA residing on an experimental card holding the necessary peripheral hardware. Matlab/Simulink with the Xilinx plug in tool 'System Generator DSP' was used for the development. Simulink and System Generator are promising tools for FPGA design but they appeared to be not completely mature at the time, making the implementation task more difficult. The outcome was nevertheless an FPGA implemented HR model having an extraordinary performance. One pressure sample is computed and the corresponding HR sample is output from the FPGA within $120ns$, the throughput of the system is $50MHz$, limited by the AD converter. Considering the time scale of an engine the HR is computed almost instantaneously and the term 'virtual HR sensor' is well motivated.

4

Development and Embedded Implementation of a Physical NO_x Model

4.1 The Model

The model developed in this chapter is a two-zone model, one burned zone in which the NO_x formation takes place and one unburned zone composed of only air, as indicated in Fig. 4.1. The burned zone temperature was computed using knowledge about the number of moles in the different zones and the global temperature as well as the temperature of the unburned zone, under the assumption of isentropic compression, rather than using an iterative energy balance approach. Using this method the physical interpretation can be maintained while the algorithm is significantly simplified.

If the pressure in the cylinder is known, which is a basic condition for much combustion engine related modeling, it is possible to compute the global temperature as well as the temperature of the unburned zone using a number of assumptions. Furthermore it is possible to compute the number of moles in the burned zone using conventional heat release analysis. It is also possible to compute the total number of moles in the combustion chamber and hence the number of moles in the unburned zone. When the number of moles in the unburned zone, burned zone and globally are known as well as the temperature globally and in the unburned zone it is possible to compute the temperature of the burned zone. A more extensive model of the combustion is hence not needed to compute the temperature of the burned zone. It is possible to avoid a direct computation of burned zone temperature and the related complex and iterative numerical solution of an

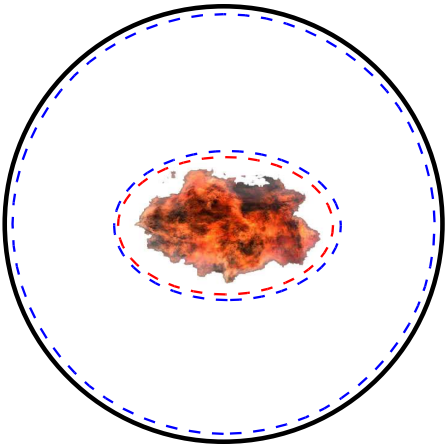

Figure 4.1 The model works according to a two-zone concept. NO is formed in the inner, burned, zone marked in red. The unburned zone is marked in blue.

energy balance normally used to determine burned zone temperature and mole content.

In this context a common simplification is that NO_x emissions are mainly composed of nitrogen oxide, NO, [Heywood, 1988]. Hence notation is that NO is modeled, rather than NO_x, and that NO emission is equivalent to NO_x emission. NO formation results from high temperatures in the gases resulting from the combustion, hence the burned zone temperature of the combustion products is a key variable for computing NO emissions. The methods for computing NO, knowing the burned zone temperature, are fairly well known and straight forward, [Egnell, 1998], [Heywood, 1988] and [Bensson and Whitehouse, 1979]. The algorithm from [Egnell, 1998] was used, but it has been significantly rewritten to increase precision and reduce numerical problems. It has also been corrected for varying burned zone volume. The final step is to solve the differential equation which gives the NO content of the burned zone and recalculate the local NO content to a global one.

Assumptions

All of the assumptions made are listed below for clarity, they are mentioned where needed in the model description as well. Their impacts on the result are discussed later on.

- The ideal gas law is valid for each zone separately and for the complete combustion chamber.
- The pressure is uniform throughout the cylinder.

- The gas temperature just after Inlet Valve Close is roughly equal to the intake temperature.
- The number of moles in the combustion chamber is not significantly changed by the combustion.
- The temperature of the unburned zone is assumed to vary according to an isentropic (polytropic) relationship.
- The ratio of specific heats, γ is assumed to be constant in the unburned zone.
- The heating value of the fuel is known.
- Iso-octane is used to model Diesel-fuel.
- The combustion efficiency is 100%.
- Combustion takes place at a constant local λ, which is assumed to be known.
- Species in the burned zone are in equilibrium.
- NO_x emissions mainly consist of nitrogen oxide, NO.

Zone temperatures and number of moles

One of the novel features with this model compared to earlier work is the computation of burned zone temperature. A two-zone approach together with uniform pressure and the ideal gas law for each zone individually as well as for the complete combustion chamber (the sum of both zones) allows computation of the burned zone temperature according to Eq. (4.2). The proof is formed in two steps; first Eq. (4.1) is formed by applying the ideal gas law to the burned zone and the unburned zone. Using Eq. (4.1) and the ideal gas law for the complete combustion chamber together with the observation that the sum of the volumes of the burned and unburned zones must equal the total volume of the combustion chamber it is possible to form Eq. (4.2). We now have an equation for the temperature of the burned zone, T_{bz}, that depends on the *number of moles* in the complete combustion chamber, n_g, in the burned zone, n_{bz}, and in the unburned zone, n_{uz} as well as the global *temperature* in the combustion chamber, T_g, as well as the *temperature* of the unburned zone, T_{uz}.

$$\left.\begin{aligned} pV_{bz} &= n_{bz}\widetilde{R}T_{bz} \\ pV_{uz} &= n_{uz}\widetilde{R}T_{uz} \end{aligned}\right\} \Longrightarrow n_{uz}T_{uz} + n_{bz}T_{bz} = \frac{p(V_{uz} + V_{bz})}{\widetilde{R}} \tag{4.1}$$

$$\left.\begin{aligned} &n_{uz}T_{uz} + n_{bz}T_{bz} = \frac{p(V_{uz}+V_{bz})}{\widetilde{R}} \\ &V_{uz} + V_{bz} = V_g \\ &pV_g = n_g\widetilde{R}T_g \\ &n_{uz} = n_g - n_{bz} \end{aligned}\right\} \Longrightarrow$$

$$\begin{aligned} &n_{uz}T_{uz} + (n_g - n_{bz})T_{bz} = \\ &= \frac{p(V_{uz}+V_{bz})}{\widetilde{R}} = \frac{pV_g}{\widetilde{R}} = n_gT_g \Longleftrightarrow \\ &n_gT_g = (n_g - n_{bz})T_{uz} + n_{bz}T_{bz} \Longleftrightarrow \\ &T_{bz} = \frac{n_gT_g - (n_g - n_{bz})T_{uz}}{n_{bz}} \end{aligned} \tag{4.2}$$

$$\left.\begin{aligned} n_0 &= \frac{p_0V_0}{\widetilde{R}T_0} \\ n_g &= n_0 \end{aligned}\right\} \Longrightarrow T_g = \frac{pV}{n_0\widetilde{R}} \tag{4.3}$$

$$T_{uz} = T_{g0}\left(\frac{p}{p_{g0}}\right)^{\frac{\gamma-1}{\gamma}} \tag{4.4}$$

Given the cylinder pressure and assuming that the gas temperature in the cylinder just after Inlet Valve Close (IVC) is roughly equal to the intake temperature it is possible to compute the initial state in the cylinder. The initial state here refers to initial global temperature, T_0, and the total number of moles of air in the cylinder at IVC, n_0. The number of moles in the entire combustion chamber n_0 can be computed using the ideal gas law based on a reference point just after IVC. Assuming that the number of moles is not significantly changed by combustion the global number of moles is known from n_0 throughout the complete cycle. This assumption is motivated through [Egnell, 1998] where it e.g. is stated that natural gas combustion at $\lambda = 1.5$ changes the number of moles in the combustion chamber by approximately 0.5% which is negligible in this context. Furthermore the global temperature can be computed from the cylinder pressure and the global number of moles using the ideal gas law, all according to Eq. (4.3).

Unburned zone temperature (T_{uz}) is computed using a reference point during the cycle and the assumption of an isentropic relationship, Eq. (4.4), similarly to the previously mentioned study. The temperature and pressure reference (T_{g0} and p_{g0}) used to compute the temperature of the unburned zone are acquired just before the combustion has started to reduce the effect of losses, e.g. heat losses and mass losses. The ratio of specific heats, γ is assumed to be constant.

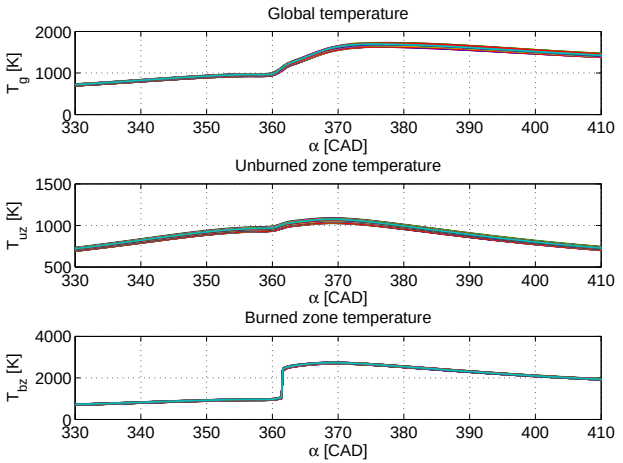

Figure 4.2 In-cycle zone temperatures of the three different zones resulting from the described method, 100 cycles.

The number of moles in the burned zone, n_{bz}, is obtained from heat release analysis. The global temperature, the unburned zone temperature and the burned zone temperature for one operation point is shown in Fig. 4.2. Corresponding average temperature traces of the different zones are shown in Fig. 4.3.

Number of moles in the burned zone One way to compute the number of moles in the burned zone is to use heat-release analysis to compute the amount of released energy. Assuming that combustion does not change the number of moles significantly, it just 'moves' moles from the unburned zone to the burned zone, it is possible to compute the number of moles in the burned zone from the heat release. It does, however, require some further assumptions. The fuel energy must be known and the combustion efficiency is assumed to be 100%. The local air/fuel ratio at which combustion takes place, λ, must also be known. Knowing the exact local λ is difficult, λ_{cal} is hence introduced as a tuning parameter in the model in a fashion similar to [Egnell, 1998]. Hence combustion is assumed to take place at a constant local λ. Note that λ_{cal} is a calibration variable even though it has a physical interpretation. The intention is to keep λ_{cal} fixed at the value which gives the best NO prediction on average for a number of data points, and which is within physically justified limits. The heat release equation used, Eq. (4.5), represents an integrated version of the apparent heat release equation in [Gatowski *et al.*, 1984] assuming constant γ. It was first presented in [Tunestål, 2000] and was previously applied in Chapter 3.

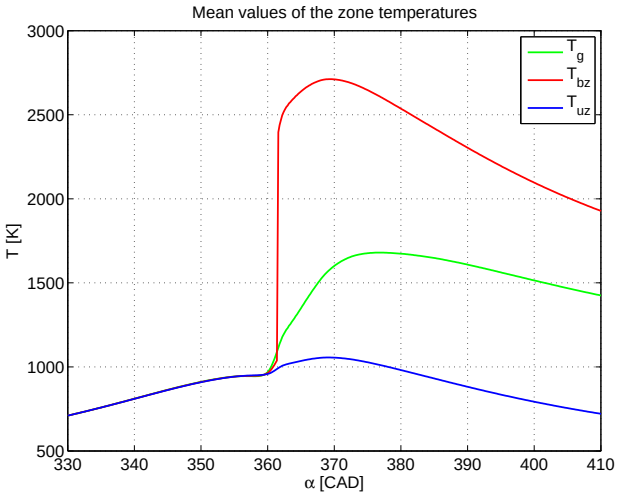

Figure 4.3 100 cycle *average* in-cycle zone temperatures of the three different zones resulting from the method described.

$$Q = \frac{1}{\gamma - 1} p(\alpha) V(\alpha) + \int_{\alpha_{start}}^{\alpha} p(\alpha) \frac{dV}{d\alpha} d\alpha - \frac{1}{\gamma - 1} (p(\alpha_{start}) V(\alpha_{start})) \tag{4.5}$$

$$\left.\begin{aligned} n_{bz} &= n_f^{bz} + n_a^{bz} = \frac{m_f^{bz}}{M_f} + \frac{m_a^{bz}}{M_a} \\ m_a^{bz} &= m_f^{bz} \lambda \alpha_{safr} \\ m_f^{bz} &= \frac{Q}{Q_{LHV}} \end{aligned}\right\} \Longrightarrow$$
$$n_{bz} = \frac{Q}{Q_{LHV} M_f} (1 + \frac{M_f \lambda \alpha_{safr}}{M_a}) \tag{4.6}$$

Both heat losses to the combustion chamber walls and the effect of varying γ are neglected in this step, in the sense that they are not explicitly modeled, in order to decrease the computational load. The heat release was then normalized against the energy content of the injected fuel. In this way the

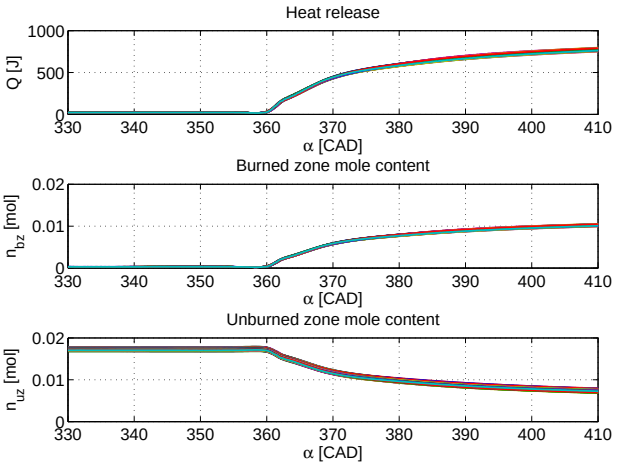

Figure 4.4 The heat release and number of moles in the burned and unburned zones for 100 cycles.

transient response of the heat release can be maintained as well as a full physical interpretation (meaning that the effect of heat losses is taken into account implicitly).

Provided an accurate heat-release analysis, it is possible to compute the number of moles in the burned zone from Eq. (4.6) which uses average molar masses of fuel and air together with local, burned zone λ (λ_{cal}) and the stoichiometric air/fuel ratio (α_{safr}). Resulting mole content of the burned zone and unburned zone together with the corresponding heat release are shown in Fig. 4.4 and every variable needed to compute the burned zone temperature (T_{bz}) is hence available.

Computation trigger The computation of the number of moles in the burned zone can not be started directly after IVC and this is the case for some of the other computations as well. The reason is that the burned zone contains zero moles until combustion has started and for example Eq. (4.2) will hence include a division by zero. Hence, a computation trigger had to be introduced in order to avoid numerical problems before combustion has started. For that purpose, the time instance when 10% of the total heat has been released (CA10%) was used. Triggering computation by the start of combustion is also sound from a physical point of view; no NO can be formed until combustion has started.

Burned Zone Composition

To be able to compute NO formation rate (using Eq. (4.23)) the concentrations of a number of species in the burned zone have to be known. It is possible to compute those based on the assumption that corresponding species have reached chemical equilibrium. To compute NO emissions using the Zeldovich mechanism the equilibrium concentration of free oxygen (O), nitrogen (N_2) and nitrogen oxide (NO) must be known (see Eq. (4.23)). These concentrations can be computed as functions of temperature, pressure ratio and local λ (λ_{cal}) by taking dissociation reactions into account. The dissociation reactions (meaning the reverse reactions) are considered important since they provide the most important source of free oxygen available for NO formation after the combustion event. The approach of [Egnell, 1998] and [Bensson and Whitehouse, 1979] was used, two reactions, Reaction 4.7 and Reaction 4.8, relative Gibbs energy and carbon, hydrogen, oxygen and nitrogen balances are put together to form the nonlinear equation system with six unknown variables shown in Eq. (4.9)-Eq. (4.14).

$$\mathrm{CO} + \mathrm{H_2O} \leftrightarrow \mathrm{H_2} + \mathrm{CO_2} \tag{4.7}$$

$$\mathrm{CO} + \frac{1}{2}\mathrm{O_2} \leftrightarrow \mathrm{CO_2} \tag{4.8}$$

$$\frac{a_{\mathrm{CO_2}} a_{\mathrm{H_2}}}{a_{\mathrm{CO}} a_{\mathrm{H_2O}}} = K_{p1} \tag{4.9}$$

$$\frac{1}{a_{\mathrm{O_2}}}\left(\frac{a_{\mathrm{CO_2}}}{a_{\mathrm{CO}}}\right)^2 = \frac{P}{a_{\mathrm{CO_2}} + a_{\mathrm{CO}} + a_{\mathrm{H_2O}} + a_{\mathrm{H_2}} + a_{\mathrm{O_2}} + a_{\mathrm{N_2}}} K_{p2} \tag{4.10}$$

$$n = a_{\mathrm{CO_2}} + a_{\mathrm{CO}} \tag{4.11}$$

$$m = 2(a_{\mathrm{H_2O}} + a_{\mathrm{H_2}}) \tag{4.12}$$

$$2\lambda\left(n + \frac{m}{4}\right) = 2a_{\mathrm{CO_2}} + a_{\mathrm{CO}} + a_{\mathrm{H_2O}} + a_{\mathrm{O_2}} \tag{4.13}$$

$$2\lambda\left(n + \frac{m}{4}\right) = \frac{a_{\mathrm{N_2}}}{3.773} \tag{4.14}$$

Where n is the number of carbon atoms and m is the number of hydrogen atoms in an arbitrary fuel; $\mathrm{C}_n\mathrm{H}_m$ (iso-octane was used with $n = 8$ and $m = 18$). P is normalized (cylinder) pressure ($P = \frac{P}{P_0}$, $P_0 = 1.013 \cdot 10^5$ Pa) and a_i is a dimensionless number describing the equilibrium concentration of species i. The two equilibrium constants, K_{p1} and K_{p2} can be computed from polynomials (the algorithm is explained in detail in [Bensson and Whitehouse, 1979]). The fact that the partial pressure of each species has a relation to K_{p1} and K_{p2} gives Eq. (4.9) and Eq. (4.10). It is also a fact

that the number of carbon, hydrogen, oxygen and nitrogen atoms involved in the reactions should remain unchanged. Hence it is possible to state six equations Eq. (4.9)-Eq. (4.14) which would have to be solved, either at runtime or in advance using precomputation/tabulation.

In order to enable the solver to find a good solution Eq. (4.9)-Eq. (4.14) were significantly rewritten using basic algebra. Performing this action gives Eq. (4.15) and Eq. (4.16) which, for the solver, have a more suitable format than the previous six equations. In this way the solver can produce a more accurate solution in much shorter time.

$$\frac{a_{CO_2}(\frac{m}{2} - a_{H_2O})}{(n - a_{CO_2})a_{H_2O}} = K_{p1} \tag{4.15}$$

$$\begin{aligned}(\frac{a_{CO_2}}{n - a_{CO_2}})^2 &= \frac{1}{(n + \frac{m}{4})\lambda - \frac{1}{2}a_{CO_2} - \frac{1}{2}n - \frac{1}{2}a_{H_2O}} = \\ &= \frac{PK_{p2}^2}{\frac{1}{2}(3n + m - a_{CO_2} - a_{H_2O}) + 8.546(n + \frac{m}{4})\lambda}\end{aligned} \tag{4.16}$$

Solving Eq. (4.15) and Eq. (4.16) gives the dimensionless equilibrium concentrations of CO_2 and H_2O from which CO, H_2, O_2 and N_2 can be computed.

All of the variables needed to compute dNO/dt are however not yet known. N_2 is known but O and NO have to be found. This requires models for the reactions forming O from O_2 (Reaction 4.17) and NO from O_2 and N_2 (Reaction 4.19). Eq. (4.18) and Eq. (4.20) represent an approach similar to [Egnell, 1998].

$$O_2 \leftrightarrow O + O \tag{4.17}$$

$$\begin{aligned}c_O^e &= \frac{K_{pO}\sqrt{c_{O_2}^e}}{\sqrt{\widetilde{R}T}} = [K_{pO} = 3.6 \cdot 10^3 e^{(\frac{-31090}{T})}] = \\ &= \frac{3.6 \cdot 10^3 e^{(\frac{-31090}{T})}\sqrt{c_{O_2}^e}}{\sqrt{\widetilde{R}T}}\ [\,]\end{aligned} \tag{4.18}$$

$$O_2 + N_2 \leftrightarrow 2NO \tag{4.19}$$

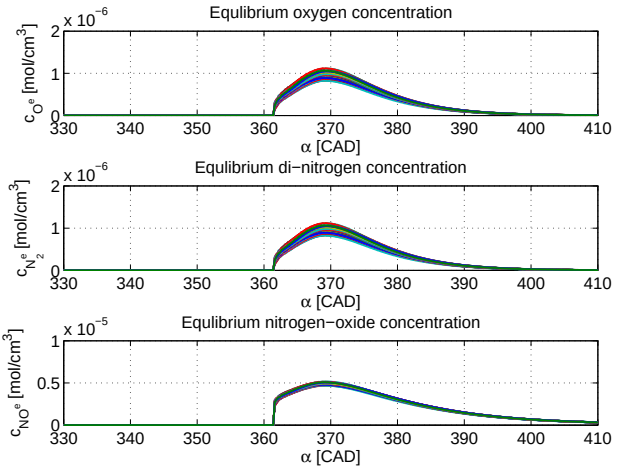

Figure 4.5 Equilibrium concentration of the three different species, needed to compute NO formation rate.

$$c^e_{\mathrm{NO}} = \sqrt{K_{p\mathrm{NO}} c^e_{\mathrm{O}_2} c^e_{\mathrm{N}_2}} = [K_{p\mathrm{NO}} = 20.3e^{(\frac{-21650}{T})}] = \\ = \sqrt{20.3e^{(\frac{-21650}{T})} c^e_{\mathrm{O}_2} c^e_{\mathrm{N}_2}}\ [\,] \tag{4.20}$$

All equilibrium concentrations were computed with an angular resolution of 0.2 CAD within each cycle, the resulting values are shown in Fig. 4.5. Knowing equilibrium concentrations of O, NO and N_2 allows computation of dNO/dt.

NO formation

Once the equilibrium concentrations of the different species are known it is possible to compute the NO_x formation rate using the Zeldovich mechanism shown in Eq. (4.23). The Zeldovich mechanism which models Reaction 4.21 and Reaction 4.22 is commonly used for NO_x modeling.

$$\mathrm{O} + \mathrm{N}_2 \leftrightarrow \mathrm{NO} + \mathrm{N} \tag{4.21}$$
$$\mathrm{N} + \mathrm{O}_2 \leftrightarrow \mathrm{NO} + \mathrm{O} \tag{4.22}$$

$$\frac{\partial c_{NO}}{\partial t} = \frac{15.2 \cdot 10^{13} e^{\frac{-38000}{T}} c_O^e c_{N_2}^e (1 - (\frac{c_{NO}}{c_{NO}^e})^2)}{1 + \frac{7.6 \cdot 10^{13} e^{\frac{-38000}{T}} c_O^e c_{N_2}^e}{1.5 \cdot 10^9 e^{\frac{-19500}{T}} c_{NO}^e c_O^e} (\frac{c_{NO}}{c_{NO}^e})} \tag{4.23}$$

During this work it was however discovered that the original rate equation based on the Zeldovich mechanism, Eq. (4.23), as presented by [Egnell, 1998] and [Heywood, 1988] actually is not valid if the volume of the zone it is applied to varies! For a two-zone model it is obviously the case that the volume of the burned zone increases as combustion progresses and more and more moles are included in the burned zone. Eq. (4.23) hence had to be rewritten according to Eq. (4.24) (which from now on is denoted the *modified Zeldovich mechanism*) in order to be valid even when the volume of the burned zone varies.

$\partial c_{NO}/\partial t$ is given by the original Zeldovich mechanism (Eq. (4.23)); dV/dt and V represent volume derivative and volume, respectively, of the *burned zone* computed using the ideal gas law and the previously computed temperatures and mole numbers of the burned zone. Solving the *modified Zeldovich* in Eq. (4.24), which can be done for example using a simple Euler method, gives the NO concentration (the unit is mole/cm^3) of the *burned zone*! Typical NO formation and NO concentration traces belonging to 100 consecutive engine cycles can be found in Fig. 4.6. However, it is the concentration of NO in the exhaust gases that is to be determined by the model. Since NO tailpipe emissions commonly are measured as a mole-based fraction the best way to compute overall NO is to compute the number of moles of NO in the burned zone and divide it by the total, global, number of moles. The procedure is shown in Eq. (4.25). This actually is the final step of the NO model. The fraction of NO in the exhaust gases is now known throughout the engine cycle, a typical NO formation case for the same 100 cycles as shown in Fig. 4.6 is shown in Fig. 4.7.

$$\left.\begin{aligned} \frac{dc_{NO}}{dt} &= \frac{\partial c_{NO}}{\partial t} + \frac{\partial c_{NO}}{\partial V}\frac{dV}{dt} \\ \frac{\partial c_{NO}}{\partial V} &= -\frac{n_{NO}}{V^2} = -\frac{c_{NO}}{V} \\ c_{NO} &= \frac{n_{NO}}{V} \end{aligned}\right\} \Longrightarrow$$

$$\frac{dc_{NO}}{dt} = \frac{\partial c_{NO}}{\partial t} - \frac{c_{NO}}{V}\frac{dV}{dt} \tag{4.24}$$

$$X_{NO} = \frac{V_{bz} c_{NO}}{n_g} \tag{4.25}$$

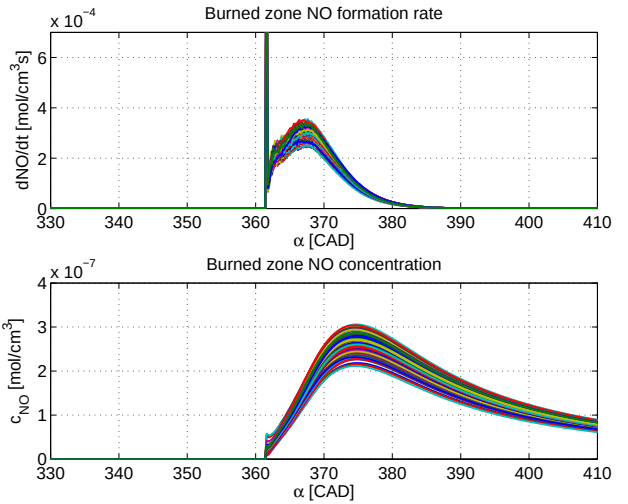

Figure 4.6 Burned zone NO formation rate and concentration belonging to 100 consecutive cycles.

4.2 The Algorithm Implementing the NO Model

To be able to successfully develop the NO model, the precomputed version of the model and finally the algorithm implementing the model (both using floating point and fixed point numbers) it was necessary to divide the development into sub stages which were iterated according to Fig. 4.8. One stage was completed when the resulting design performed as well as the previous stage. If some fundamental issue was encountered the original model was modified and reverified before it was possible to proceed down the stages again. Using such a development flow was necessary to assure correct operation of the final algorithm. During the implementation process the original model was converted to an *algorithm* which can be implemented for example in the C programming language.

Implementation of the model described above took quite some consideration. The aim was to be able to model NO *during the cycle*. Having to wait for the cycle to complete in order to normalize the heat release and compute the trigger point ($CA10\%$) was hence never an option. To make it possible to compute the model during the cycle the model had to be made causal, meaning that it does not use measured values from the future.

Implementing algorithms in embedded systems often takes special considerations as described in Chapter 2. In such environments the infrastruc-

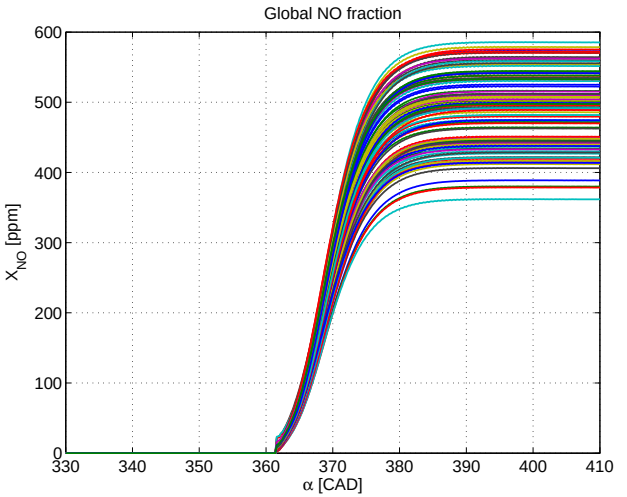

Figure 4.7 Total fraction of NO within 100 engine cycles.

ture is more limited and computations hence must be adapted. For the sake of limiting hardware use and maximizing speed, simplifications have to be made to the model when developing the algorithm. Such simplifications were made e.g. selecting filter coefficients so that they are power-of-two and computing $CA12.5\%$ instead of $CA10\%$, avoiding one division. Last but not least as large part of the model as possible should be precomputed and stored in memory to reduce the computational load at runtime.

Making the Model Causal

As mentioned previously the NO computation was triggered by $CA12.5\%$ (as opposed to $CA10\%$ for the original model) and the heat release was normalized against the total energy of the fuel injected. To perform these procedures two points were needed; the minimum and maximum point of the heat release trace. When computing the NO model off-line, using recorded data, this will not cause a problem. However since the computations were to be carried out during the cycle issues arose with the important maximum and minimum points of the heat-release. A naive way to solve this was to just use the maximum and minimum points from the previous cycle. Doing so did however give rise to some practical issues; oscillations occurred in the two variables. The oscillations could be removed using third-order low-pass FIR filters on the corresponding variables. Selecting a very short step response, no amplification and filter coefficients such that division could be avoided the two filters had virtually no negative impact on the algorithm as

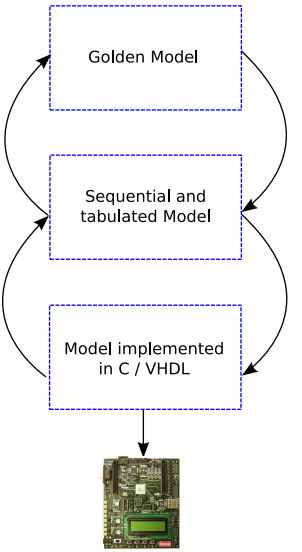

Figure 4.8 The development flow used to successfully implement the model and algorithm.

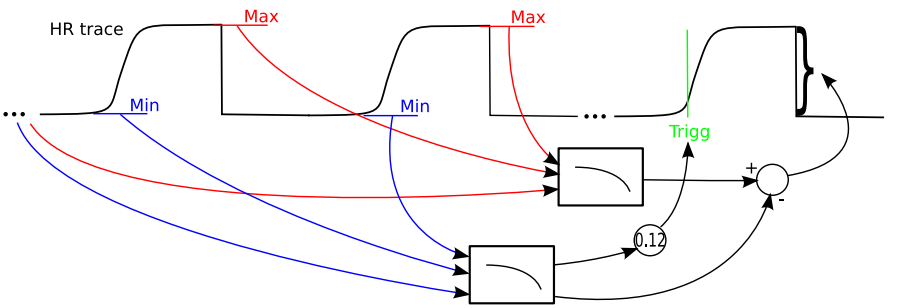

Figure 4.9 The figure explains the operational principle of the causal heat release including low-pass filters.

such. The filters successfully removed the issues with oscillations and made it possible to develop a causal version of the algorithm.

Precomputation and tabulation

Since high computation speed is an absolute requirement complex computation such as the nonlinear equation system (Eq. (4.15) and Eq. (4.16)), the equations used to compute the equilibrium concentration of O and NO

(Eq. (4.18) and Eq. (4.20)) as well as the Zeldovich mechanism (Eq. (4.23)) have to be avoided at runtime. Even though it may be possible to implement a fast solver in some sort of electronic hardware it is much easier and significantly faster to tabulate the result and store it in memory. This action was performed using the observation that dc_{NO}/dt actually is a function of only three variables; burned zone temperature, the pressure ratio in the cylinder and current NO concentration. By Eq. (4.23) $\partial c_{\mathrm{NO}}/\partial t$ is a function of T_{bz}, c_{NO}, c^e_{NO}, $c^e_{\mathrm{N_2}}$ and c^e_{O}. The equilibrium concentrations $c^e_{\mathrm{N_2}}$, c^e_{O} and c^e_{NO} are in turn functions of T_{bz}, P and c_{NO} only. Hence it is possible to tabulate $\partial c_{\mathrm{NO}}/\partial t$ as a function of T_{bz}, P and c_{NO}. Since the number of input variables was not too large it was possible to use precomputed values to avoid solving the nonlinear equations and Eq. (4.23) at runtime. In this context pressure ratio, P, refers to the ratio between cylinder pressure and atmospheric pressure.

It is very important to point out that the model still remains a physical model,regardless of the fact that parts of the computations needed for the model are precomputed. Tables are extensively used in many forms of advanced computer arithmetics and they are just a complementary method of performing computations which, in this case, is much faster.

The equations involved were however nonlinear and performing this pre-computation yields results looking like the surface shown in Fig. 4.10. An attempt was made to tabulate these functions using evenly (linearly) spaced breakpoints as a grid, the table look-up was performed using rounding towards nearest value. Using such simple logic proved not to work very well and a second attempt was made using a linearly spaced table together with trilinear interpolation. Even though it significantly improved the result it was not good enough. At the same time trilinear interpolation comes at a high computational cost.

One possibility was to use unevenly spaced table breakpoints and exponential spacing between the breakpoints would have been a logical choice considering Fig. 4.10. Exponential spacing would however have the drawback that it would require significant computation effort to compute the indices when performing look-up in the table at runtime. As an intermediate approach, avoiding the difficulties of computing exponential indices, piecewise linear polynomials were used. The polynomials were selected in an *ad hoc* manner studying different surface plots similar to Fig. 4.10. It is important to try to keep the order of the polynomials as low as possible to reduce the number of tests needed to implement the algorithm (thus reducing program size or hardware consumption). Another factor to take into account producing these tables is the fact that most devices have limited amounts of memory available to store the tables. Naturally we end up in an precision/consumption trade-off situation when increasing the degree of the

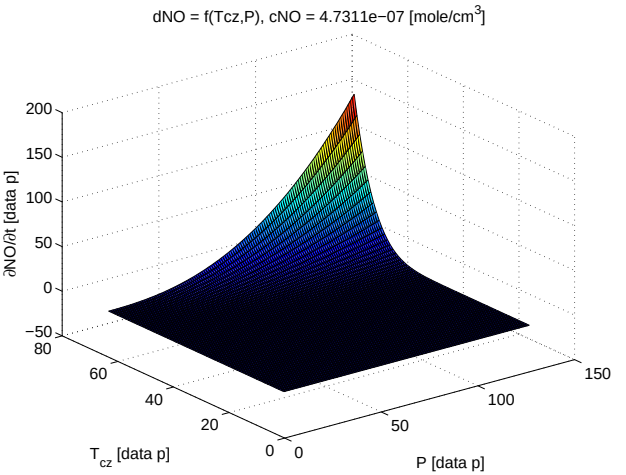

Figure 4.10 The NO formation-rate table visualized as a function of burned zone temperature and pressure ratio, at constant NO concentration.

polynomials and/or the length of the tables. The three different polynomials which were used as look-up indices are shown in Fig. 4.11.

It was not only the NO formation mechanism that had to be tabulated, the isentropic relationship (4.4) was also too complex to be computed at runtime and thus Eq. (4.4) was tabulated as a function of p/p_{g0}. The combustion chamber volume and volume derivative were also tabulated as functions of crank angle. In these cases normal linear tables with round-to-nearest look-up was sufficient.

4.3 Experimental Platform

Engine Data

Five data points consisting of some 150 cycles each were used for the validation of the algorithm. The data was obtained on a single-cylinder version of a passenger-car sized Volvo D5 diesel engine (0.5 l displacement per cylinder). The setup is presented in detail in [Horn *et al.*, 2008]. The data points were taken at increasing loads with varying intake-pressure, no external or internal Exhaust Gas Recirculation was present. The fuel was standard (Swedish MK1) Diesel fuel and the injection strategy was single injection with constant end of injection and varying start of injection. Global λ hence varied

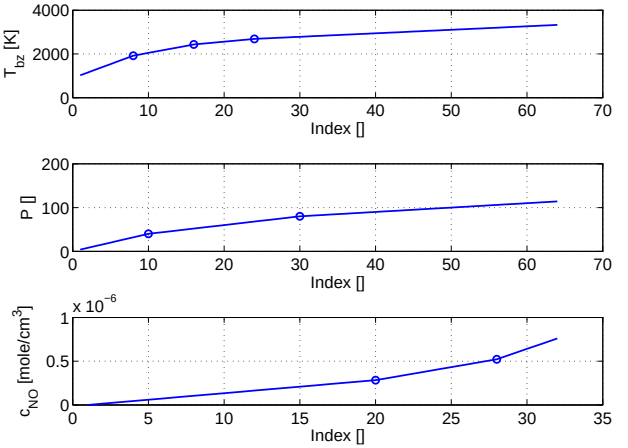

Figure 4.11 The three different index variables used in the look-up table, circles denote a change of polynomial.

between about 4 and 1.5 according to Fig. 4.12. The reader should note that even though the three last data-points had the same lambda, the load varied between the points since the intake pressure differed (due to turbo charging). One could say that the data-points represent a common Diesel load sweep.

Simulation Environment

All computations and simulations were carried out using Matlab running on a normal desktop PC. When developing the tabulated version of the model (the algorithm (ALG)) the original NO model was used as a reference (as a 'golden-model' (GoM)). In this way it was possible to compare internal variables of the GoM to those of the ALG, ensuring correct operation of the ALG. In a sense the GoM hence was a part of the setup when developing the ALG.

Embedded system

As embedded implementation platform, a development platform hosting an ARM-9 processor (an Atmel AT91SAM9260 device) was used. The processor was a 180 MHz 16/32-bit RISC processor featuring 64 MB SDRAM and 512 MB Flash memory. The ARM processor family is very common in cost sensitive applications such as mobile-phones and other similar products with embedded processors. A Linux kernel version: 2.6.26.3 was present on the processor and the algorithm was implemented in C. Two different versions

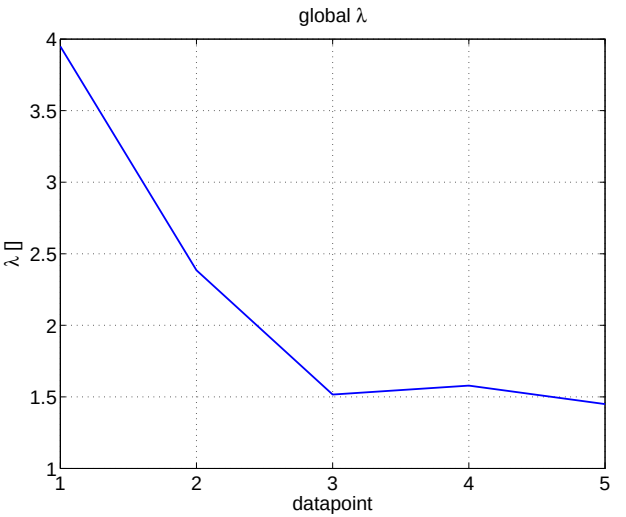

Figure 4.12 Actual measured, global, λ.

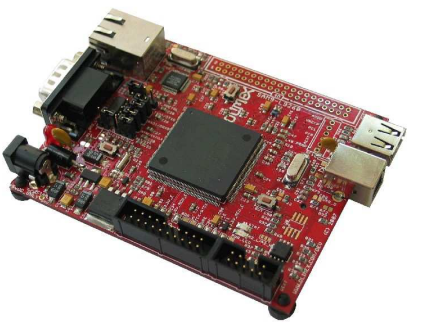

Figure 4.13 The development board used for algorithm test.

were implemented, one version using fixed-point arithmetics and one using floating point. Applications were compiled using the GNU based 'CodeSourcery G++' tool chain, data input and output was taken care of using files. Execution times were measured for 100 cycles (360 000) samples at a time using the C 'time.h' library.

4.4 Results and Performance

The five data points described above were used to test the performance of the model. The only parameter available for 'tuning' the model, λ_{cal}, was swept with high resolution between 0.95 and 1.2. NO computation was performed using both the model and the algorithm with these λ_{cal} values, the results are shown in Fig. 4.14. Also included in the figure are dots and squares indicating the actual measured NO emission for each data point and a vertical line indicating the average λ_{cal} giving the highest agreement with the measured NO values of the five data points. The average λ_{cal} giving best agreement between measured NO emission and those computed by the model is 1.088. The corresponding best λ_{cal} value computing NO using the algorithm was 1.096. Using the model and a λ_{cal} value of 1.088 gives a maximum error of about 30% and an absolute average error of roughly 20%. Considering the coarse nature of the model and the very complex nature of the physical process this must be considered acceptable at least for qualitative control and diagnostics purposes.

The previously presented figures (Fig. 4.2-Fig. 4.7) shows crank angle resolved data originating from data point 3 (which was randomly selected) using a λ_{cal} of 1.088. Comparing Fig. 4.3, Fig. 4.6, and Fig. 4.7 on crank angle base with the work of [Egnell, 1998] or even with [Nishida, 2006] shows that the transient response of internal variables as well as NO formation of the developed model corresponds well to previous models and in-cycle measurement of NO, which is encouraging.

The maybe most important part of the results is without doubt the successful development and implementation of the causal and fast version of the NO model. The algorithm was evaluated by running two different versions on the embedded ARM processor. One version was implemented using single precision (32 bit) floating point number representation and the other using 32 bit fixed point numbers. The two different versions of the algorithm were applied to a representative data point (data point 3) and NO formation rate and concentration were stored to files and evaluated against the output of the original reference model ('golden model'). The floating point C code implementing the model is found in Appendix C for reference.

Algorithm Accuracy

Fig. 4.15 show the results from to the two different versions of the algorithm sample-by-sample and cycle-by-cycle. The results were also compared to the output of the GoM. These comparisons can be found in Fig. 4.16 and Fig. 4.17. Fig. 4.16 shows 90 cycles average corresponding to Fig. 4.15 compared to the GoM. The top two plots of Fig. 4.17 shows the fractional error between the causal version of the heat release and the non-causal version of the GoM, compared sample-by-sample and cycle-by-cycle. The middle

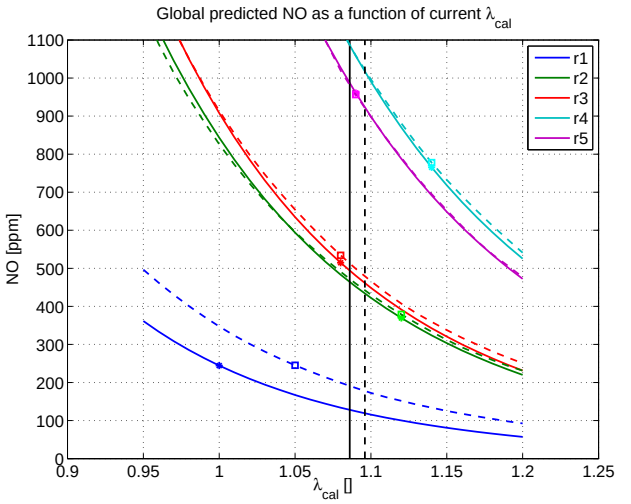

Figure 4.14 The exhaust gas NO content as a function of current λ used by the model (solid lines) and the algorithm (dashed lines).

plots of Fig. 4.17 shows the fractional error of final NO concentration output, again compared sample-by-sample and cycle-by-cycle to the GoM. The bottom part of Fig. 4.17 show 90 cycle average fractional error of NO concentration, NO formation rate and heat release, compared to the GoM.

Algorithm Performance

The intended use of the suggested model/algorithm implies that computation speed needs to be very high. Benchmark runs were performed on the embedded ARM processor to measure the time consumption of the algorithm. Each version of ALG was compiled both with and without optimization for speed (the gcc -O3 option). When optimizing for speed the size of the resulting binaries are increased and the program hence consumes more memory. The different versions were tested upon the ARM processor and the results are visible in Tab. 4.1. Time consumption for 100 consecutive cycles containing 3600 data points each was measured with a time resolution of 0.01 s. The table shows three different values for each version of ALG. The first column shows total execution time for all 360000 samples. Column two, labeled 1/700, shows the execution time average over the scope when ALG is somewhat active (between 300 CAD and 440 CAD). Outside this scope only a few very simple operations are performed, such as conditional tests and variable declarations. Within this scope parts of ALG are active, such as the heat release and temperature computations. The last column,

Table 4.1 Execution times for one sample, measured on the ARM processor.

Algo. Ver.	$t_{exec} \pm 0.01$s	$\frac{1}{700} \pm 0.14\mu$s	$\frac{1}{400} \pm 0.25\mu$s
Fix. -O0	0.84 s	12.0 μs	21.0 μs
Fix. -O3	0.53 s	7.6 μs	13.3 μs
Flt. -O0	2.02 s	28.9 μs	50.5 μs
Flt. -O3	1.52 s	21.7 μs	38.0 μs

Fix. = 32-bit fixed point, Flt. = 32-bit floating point.
-O0 = 'Normal' compilation, -O3 = Optimized for speed.

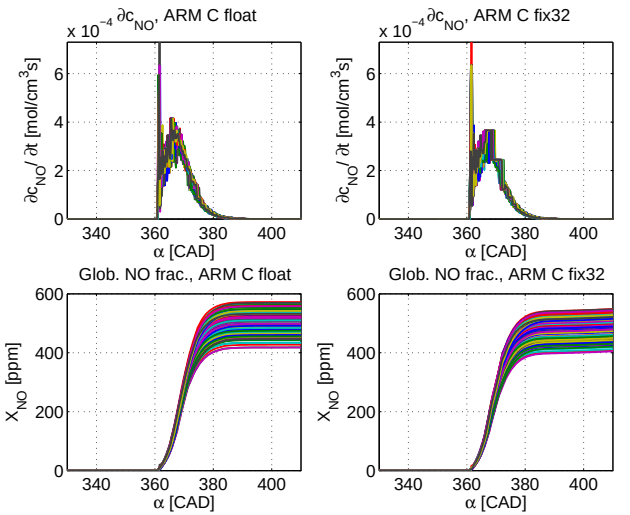

Figure 4.15 The two upper plots show cycle resolved NO formation rate computed using tabulated C algorithm, floating point (left) and fixed point (right). The two lower plots show corresponding global concentration of NO.

labeled 1/400, shows the execution time average over the scope where the actual NO computations are carried out (approximately between 360 CAD and 440 CAD). A representative sample average computation time, valid for the complete ALG, would hence be something in between the values in column two and three.

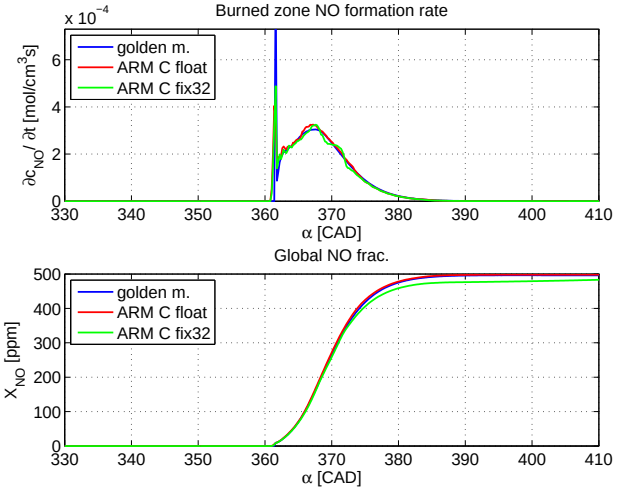

Figure 4.16 Upper part shows 90 cycles average NO formation rate, C algorithm compared to golden model. Lower part shows corresponding global concentration of NO.

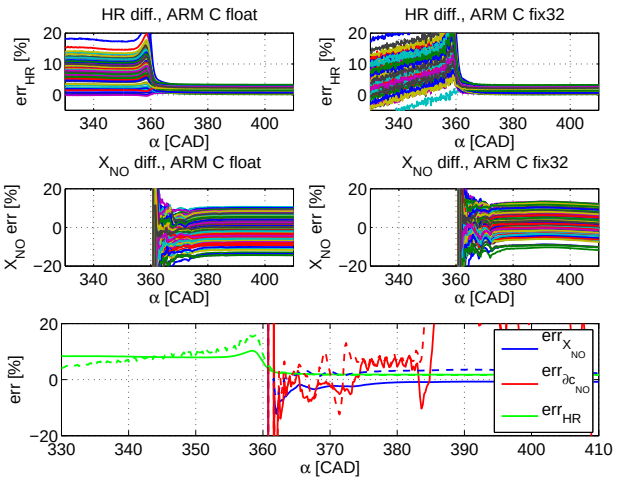

Figure 4.17 Sample by sample and average error between the golden model and the C algorithm. Left plots show floating point and right ones show fixed point. Upper plots show heat release, middle plots show global NO fraction. Lower plot shows 90 cycles average errors, floating point in solid and fixed point in dashed lines.

4.5 Discussion

Models are, naturally, rarely perfect. Assumptions which have to be made represent simplifications of the real world and will hence infer errors. How large errors can be tolerated varies depending on the purpose of the model. The errors reported for this model are considered as acceptable for control and embedded applications. Similarly to the physical world where NO formation depends exponentially on combustion λ, the modeled NO formation depends exponentially on λ_{cal} as seen in Fig. 4.14. The NO model is thus very sensitive to this setting which explains the maximum absolute error of 30% even though best calibration λ only differs at most by 0.1 (9%) between the different points. In the light of the exponential dependence between NO and λ_{cal} and considering the relatively coarse nature of the model the 20% average error must be considered as a good result.

It is very difficult to fully verify this work against engine testbed data which would be desirable. The reason is of course that it is very difficult to measure the NO actually present in the cylinder during combustion. One way to perform such a measurement was proposed in [Nishida, 2006], but this type of equipment was not available to the authors. It is neither possible to compare end NO of each modeled cycle to NO concentration measured in the exhaust gas on an engine testbed since NO values measured represent average values from a number of different engine cycles due to mixing of exhaust gases. It is also a fact that a typical NO instrument takes approximately 2 s to respond to a change in concentration (in 2 s at least 40 cycles have occurred in the engine). The only feasible way to verify an in-cycle NO model against testbed data is to compare the average, modeled, end NO value to the measured engine-out NO. This must be done over a time span where the engine maintains constant operating parameters. These are the reasons why the ALG was compared to the GoM which in turn was developed and compared to experimental, testbed, average data.

Taking a closer look at Fig. 4.16 and Fig. 4.17 it is obvious that the tabulated ALG is a sufficient implementation of the original NO model. The output from the ALG (global NO ratio) is in-fact within about $\pm$15% compared to the GoM, compared cycle-by-cycle and sample-by-sample. On average the ALG performs within a few percent of the GoM as visible in Fig. 4.17, bottom part. This goes for both the floating-point and fixed point version of the ALG. The table look-up method also worked very well on average as indicated by Fig. 4.16, upper part, and Fig. 4.17, lower part. Agreement between the NO formation rate from the GoM and the one computed using the exponential table (using the ALG) is excellent both for the floating and fixed point version of the ALG.

The Model

Sensitivity to Assumptions It is very difficult to state which of the assumptions that has the largest impact on the performance of the model. During the work it was however obvious that the model is very sensitive to variables that affect the total number of moles included in the burned zone, for example λ_{cal} to which model-out NO shows exponential dependence. The total heat release (giving the number of moles in the burned zone) is also very important in order to get acceptable results from the model, which was the reason for normalizing the heat release with the total energy content of the injected fuel.

The error coming from assuming that combustion does not change the number of moles in the combustion chamber are regarded as very small. [Egnell, 1998] quantifies the error coming from neglecting the combustion effect on the number of moles to about 1.5% @ $\lambda = 1$.

Assuming 100% combustion efficiency and a known energy content of the fuel are reasonable assumptions which should have a very small impact on the final result. Diesel engines commonly have a very high combustion efficiency and they are run on Diesel fuel with known heating value. Furthermore the injection system has information about the injected fuel amount.

Some of the model assumptions are not discussed further since they are considered to be common practice in the field. These are for example assuming chemical equilibrium, assuming that the ideal gas law is valid and assuming the NO_x mainly consists of NO.

It should be pointed out that the impact of errors and assumptions in the model is that the best calibration λ is shifted away from it's physically correct value which is considered to be one. The fact that the best calibration λ is very close to one strongly indicates that the assumptions made are reasonable.

Calibration Local λ Commonly some variable(s) is(are) introduced in models which allows calibration of a model output, meaning that the model output can be adjusted to fit empirical data. If the number of parameters that are introduced can be kept small it is beneficial for the validity of the model and for the model calibration effort. In the suggested approach actually only one variable is available for calibration, λ_{cal} (the modeling local λ). It is very important to point out that λ_{cal} used in the model is *not* the same as the global λ for all the gas entering the engine. λ_{cal} is nor the same as the actual *physical* local λ categorized by [Dec, 1997]. λ_{cal} has a physical interpretation even though it is a calibration variable. It can be said to reflect the average lambda at which combustion takes place, or rather the average λ of the gases which enter the burned zone. Turbulence will introduce gases

which do not take part in combustion (not adding to the heat release) into the burned zone and since there is no account for turbulence in the model this phenomenon will be accounted for through a slightly leaner average λ_{cal}. A best calibration value of $\lambda_{cal} = 1.088$, according to Fig. 4.14, holds some promise regarding the validity of the model. Best λ_{cal} is significantly richer than the real measured global lambda in the five data-points, indicating that the model captures the stratified nature of Diesel combustion. It is also not very far from 1.0 which according to [Dec, 1997] is the *physical* local λ maintained during Diesel diffusion combustion.

Clearly calibration λ can not be expected to *exactly* match the values physically expected, the calibration λ will be affected by the validity of many different assumptions and will include many effects. Hence even though calibration λ has a full physical interpretation it will not obtain fully physically correct values! It is important to note that calibration λ should *not* exceed the *global* λ, shown in Fig. 4.12. If it exceeded the global λ the use of a stratified, two-zone, model would have been questionable. Such a case would probably indicate that some of the assumptions or computation (for example the heat-release analysis) were erroneous.

Heat release and Mole Numbers The format of the heat release needed to compute the number of moles in the burned zone (Eq. (4.5)) differs significantly from the 'standard' heat release equation (see [Gatowski *et al.*, 1984]). Using a heat release analysis which neglects all the effects of losses (heat losses, crevice losses and losses due to combustion inefficiency) and neglects the varying γ of the combustion zone has the drawback that the value of the result may differ significantly from the value expected from the amount of fuel injected. The precision of the heat release actually appeared to be a severe problem since it is very important to be able to compute the number of moles in the combustion zone with reasonable accuracy. This issue was solved by normalizing the output of the heat release against the amount of fuel actually injected. The amount of fuel actually injected was computed using fuel flow, it could also be computed from the injector signals with high accuracy. Normalizing the heat release proved to be an efficient method to keep the transient response of the heat release while obtaining reasonable values. The normalization is a very efficient way to reduce the effects that heat losses and a varying γ have on the computations, using a value already available in an engine systems (the injector signal).

Algorithm Trigger Another interesting point of discussion is how the computation trigger ($CA10\%$ in the GoM and $CA12.5\%$ in the ALG) affects the result. Using a computation trigger might have some impact on the results, the best candidate for computation trigger would obviously be CA5% which however proved to be too sensitive to noise for practical use. Instead CA10%

was selected as a more stable approach with the drawback of reduced accuracy during the early combustion. It is a physical fact that there is no NO formation before combustion has started since the temperature increase due to compression is not high enough. It is only when combustion has started that the burned zone temperature is high enough to form NO. The burned zone is furthermore infinitesimal during the earliest stages of combustion and hence very small amounts of NO is formed initially. Hence, it is not necessary to start the computation of NO before combustion has progressed for a little while. The peak present in the computation of $\partial \mathrm{NO}/\partial t$ in Fig. 4.6, Fig. 4.15 upper parts and Fig. 4.16 is not regarded as an artifact of the method to start the computations. This peak is instead considered to be a result of the chemistry model as such. Early in the cycle there is *no* NO present in the cylinder, since $\partial \mathrm{NO}/\partial t$ is a function of among others current c_{NO} by Eq. (4.23), it is given that NO formation rate is high when current NO concentration initially is low (zero). Triggering NO computation has two clear benefits, numerical issues caused by zero-division events are avoided and computation work is saved! We have found no drawbacks with the use of $CA10\%$, $CA12.5\%$ as computation trigger, considering the very small size (number of moles) in the burned zone before CA10%. Naturally this method is independent of IVC and which valve strategy is applied. As long as there is combustion, the model will trigger when the actual combustion takes place and hence when NO is formed.

Sensitivity to γ The ratio of specific heats, γ, is an important variable in this modeling context. The true and correct way to handle γ is a matter of dispute in modeling communities. Assuming an ideal gas, γ is a function of temperature and it is in many cases important to include this dependence on temperature.

In this work the fluids are assumed to be ideal gases, γ is however assumed to have a constant value! The reason for doing so in this case is to spare computation time and memory. γ was used in two places in the model, one place is in the heat release which is considered to be the place where a varying γ has its largest impact since the temperature in the burned zone is high. The issue with varying γ in this case is however solved by normalizing the heat release with the amount of injected fuel.

γ was also used to compute the temperature of the unburned zone. The unburned zone is however significantly colder than the burned zone. In the temperature range of the unburned zone γ does not vary enough to affect the results. The model as such is hence not significantly affected by the assumption of a constant γ and being able to assume a constant γ actually was important to obtain a fast model. In the tabulated version of the model it would be possible to take account for a varying γ without decreasing computation speed, since the unburned zone temperature expression was pre-

computed and tabulated. It would however add one dimension to the table which was used and the result is not considered to be affected enough to motivate this added overhead.

Modified Zeldovich Rate Expression Realizing that a modification to the commonly used and well known Zeldovich rate expression causes some confusion a short motivation in the form of logic reasoning follows. Consider a volume containing a certain number of moles of NO. Let us assume that, at current conditions, no NO is either formed or consumed meaning that $dc_{\mathrm{NO}}/dt = 0$ according to the original Zeldovich mechanism. At such a situation we have a fixed number of NO molecules in a fixed volume. If the volume changes but the formation of NO molecules still is zero an issue with the original Zeldovich rate expression arises. In such a case the *actual* physical NO *concentration* will decrease but the original Zeldovich expression as presented in [Heywood, 1988] will still compute the NO concentration change to zero since the difference in volume is not included in the expression!

From this reasoning it undoubtedly follows that the original Zeldovich rate expression is only valid for constant volume. If the volume varies it has to be complemented with a term describing the concentration change due to volume change. This explains the development of the *modified* Zeldovich rate expression.

Since previous contributions in the area use a multi-zone approach it is highly likely that each zone does not vary significantly in volume, the number of zones just increases. The original Zeldovich rate expression has hence been valid in those cases.

Modeling EGR Modern Diesel engines commonly use Exhaust Gas Recirculation (EGR) to reduce emissions of among others NO_x. Any NO_x model should hence preferably capture the impact of EGR on NO_x emissions. It would be interesting to know how difficult it would be to adapt the developed model to include the effects of EGR. The answer to this question is that it would not take much effort to include the effect of EGR. The use of EGR will basically have an impact on the parts of the model which compute the number of moles in the burned zone, n_{bz}, and the global number of moles, n_g. When using EGR there will be a number of moles of non reactive gas present in the burned zone which will increase the total number of moles present in the burned zone, thus lower the burned zone temperature, T_{bz}. Hence, the burned zone temperature computations will have to be updated so that it includes the number of moles of non reactive gas entering the burned zone together with the reactive gas.

The start temperature of the charge would also change when using EGR. Start temperature estimation would have to be changed accordingly so that

it computes a correct start temperature of the gas which is important to be able to compute the global number of moles, n_g.

Since the factors that determine NO formation are computed using an equilibrium chemistry model it is not *directly* affected by the use of EGR. The equilibrium chemistry model (and hence the NO formation) depend only on T_{bz}, p and λ (see e.g. [Eriksson, 2004]). This is a very good thing since it means that the precomputed NO formation table which was used to implement the algorithm is valid even when using EGR, as long as combustion still takes place at the same local λ.

EGR influences combustion by lowering the burned zone temperature and if the computation of the number of moles in the burned zone and the global number of moles is updated so that they reflect these changes the model can be made to work even when using EGR.

The Algorithm

Causal Heat Release The causal version of the heat release appears to work very well compared to the non causal version used in the GoM. So is the case both compared sample by sample and on average as evident from the upper and lower parts of Fig. 4.17. The results from the causal heat release are always within 3% of GoM during combustion, the average error is 2%. Note that the fairly large fractional error before 360 CAD depends on division by zero when computing the fractional error. It seems to be safe to conclude that the causal heat release of ALG works almost as well as the non causal one of GoM. The computation of burned-zone temperature is however highly sensitive to errors in the heat release and a part of the difference in final NO between ALG and GoM on cycle base, seen in the middle of Fig. 4.17, can surely be attributed to these few percent of deviation in the heat release. The causal heat release was however necessary in order to compute in-cycle NO

Look-up Tables The use of look-up tables was crucial to obtain an ALG which was able to compute in the desired time frame. Using the piecewise linear polynomials to index the NO formation table it was possible to compute NO formation with sufficient accuracy. Attempts made using normal linearly spaced tables were not able to produce such good result with respect to agreement with the GoM or signal quality of the formation rate. The exponentially spaced table surely was a key component in developing the ALG. Some switching noise was present on the NO formation rate signal (Fig. 4.15 upper part), meaning the noise which is caused by discrete steps in the table. However none of the noise propagated through to the resulting c_{NO} signal (Fig. 4.15 lower part). The reason for this is that the Euler method used to solve Eq. (4.24) has strong low-pass characteristics and the switch-

ing noise present on the NO formation-rate signal is hence less of a problem, it is the NO concentration (c_{NO}) and fraction (X_{NO}) that is of interest.

This method of precomputation and tabulation should *not* be confused with the type of *empirical* calibration tables commonly used in the automotive industry. It is very important to point out that the model remains physical even though parts of it are precomputed and stored in tables. The table performs exactly the same mapping between the input variables and the output variable as the original function would have done, the only difference is that it is precomputed and hence faster. Neither is the extrapolation property, which is the strength of physical models, lost. The model maintains its capability of extrapolation and there is no need for exchanging the table e.g. when changing engine.

C Algorithm A very important part of the result is the successful implementation of ALG in C. Developing the tabulated version of ALG took quite a lot of consideration. Implementing ALG in C also took some time, especially the fixed point version. Implementing arithmetics using fixed point numbers is complicated, internal variables must have a correct fraction length in order to maintain enough resolution to produce correct results, at the same time overflow has to be avoided during all steps of the computations. In this work the fraction length (number of bits used to represent the decimal part) were selected for best precision based on simulations of a number of different data points in Matlab. These simulations were performed based on 16-bit word length but the final implementation was performed using 32-bit word length, thus overflow is highly unlikely (there are 16 spare bits). The 16 spare bits surely leaves the door open for further optimization, either to increase the precision of the fixed point version or maybe to reduce the word length of the fixed point version to 16-bit while still maintaining about the same numerical performance.

Computation Speed and Memory Consumption The goal with this work was to develop a model and algorithm of such a format and with such low demands on computational power so as to be computable in an embedded system. The goal was to handle a crank angle resolution of 0.2 CAD and a maximum engine speed of 4500 rpm (corresponding to the maximum engine speed of a modern passenger car Diesel engine). This means that ALG needs to compute one sample faster than $1/(4500 \cdot 1800/60) \approx 7.4$ μs, in order to finish before the next sample arrives. The result from the tests on the embedded ARM processor in Tab. 4.1 indicate that ALG almost meets the goals. The fixed-point version of ALG was able to compute in-cycle NO formation with the desired crank angle resolution up to about $60/(1800 \cdot 11 \cdot 10^{-6}) \approx 3000$ rpm (the most optimistic way to compute gives 4400 rpm and most pessimistic gives 2500 rpm). The floating-point version

of ALG would however not be able to cope with engine speeds higher than about $60/(1800 \cdot 30 \cdot 10^{-6}) \approx 1100$ rpm. The fixed point version of ALG is more than twice as fast as the floating point one, which was expected since no hardware floating point support was available in the compiler. Using floating point the processor has to perform every operation twice, once for each of the two words used to represent a floating point number. Using fixed point numbers each operation only has to be performed once since a single number is used. To compute the model at the desired time of 7.4 μs several things could be attempted, one thing that probably would increase computation speed would be to remove the Linux OS present on the processor. Another thing that could be done is to slightly reduce the crank angle resolution. The easiest would however be to use a slightly faster processor or maybe even to implement the model using an FPGA, which may be more cost effective.

Memory is another limiting factor in this context. The NO formation-rate table as well as volume, volume derivative and the table used for the isentropic relationship needs to be stored in memory. Table values were stored using 32 bit numbers and NO formation table size was $64 \cdot 64 \cdot 32 = 131072$ points. The table would consequently need $2^6 \cdot 2^6 \cdot 2^5 \cdot 32 \approx 4.12$ Mb, the other tables together would need $(2 \cdot 3600 + 256) \cdot 32 \approx 0.24$ Mb. The size of the binaries is about 0.80 Mb which means that the current model occupies about 5.16 Mb = 0.65 MB which is a negligible part of the memory available on typical embedded processors, as well as the processor used to generate the results. With some more work it is likely possible to further reduce the size of the NO formation table about two times to 2.06 Mb = 0.26 MB by reducing the pressure ratio resolution to five bits.

4.6 Summary

This chapter suggested a model which can be used to compute the NO emissions of a Diesel engine. The model, or algorithm implements a physically correct NO model which uses a two zone approach to model the stratified nature of Diesel combustion. A novel approach for computing the temperature of the burned zone was developed. The temperature computation is based on the ideal gas law applied to the different zones in combination with assumption of isentropic compression of the unburned zone. Knowing the temperature of the burned zone it is possible to compute the NO formation rate. The well-known Zeldovich rate expression has for that purpose been modified in order to be correct when the control volume (burned zone) changes its volume.

The only 'tuning parameter' within the model is the local λ and the model provides best agreement with measured data when the local λ is

within physically reasonable values (λ is close to one). On the five data points available an absolute average error of 20% was obtained. Even though this number is not to be considered as a very good result compared to extensive multi-zone models it is, using the model, possible to get quantitative information regarding the instantaneous NO content in the cylinder. Comparing with complex multi-zone models is unfair considering the computation times needed to complete the different models.

The most important part of the result was to show that it was possible to develop an algorithm implementing the physical NO model on an embedded system. A causal version of the heat release had to be developed. In order to obtain high computation speeds the NO formation rate had to be tabulated as a function of pressure ratio, current NO concentration and current burned zone temperature. Two different versions of the algorithm were developed, one version using floating-point numbers and one version using fixed-point numbers. The resulting algorithms were tested on an embedded processor and execution times were measured and evaluated. The fixed-point algorithm proved to be able to compute in-cycle NO concentration 'online' up to an engine speed of at least 3000 rpm and with a resolution of 0.2 CAD. The floating-point version, being less than half as fast due to the use of floating point arithmetics, could do the same up to engine speeds of about 1000 rpm. Even though the model used fairly large tables it did not consume a lot of memory. The present model implementation occupies a total of 0.65 MB of memory.

A large part of the methodology used are applicable in other situations where it is desirable to implement fast versions of physical models in embedded environments. This work can be said to over bridge the traditional gap between the detailed physical models and the less detailed, state based, control oriented models.

5

Concluding Remarks

Performance demands put on combustion engines are ever increasing, e.g. demands on emissions and fuel consumption. The increased demands together with new combustion concepts increase the need for feedback engine and combustion control. Mathematical models are considered important in order to implement high performance feedback control, as well as to perform diagnostic functions in vehicles. At the same time economical considerations enforce the usage of low fidelity control hardware, limiting the implementation possibilities due to lack of computational power.

This thesis combines the areas of automatic control, electronic hardware design and development of embedded software, and applies it to combustion engine control.

Various options which can be used to implement mathematical models in vehicles are described; embedded processors, FPGAs and ASICs. Embedded processors are processors embedded in products e.g. engine control units. An FPGA is a large network of electronic components (an electronic circuit) which is reconfigurable and ASICs are large electronic circuits which are not reconfigurable. Which of these implementation platforms to choose must be decided based on the intended application and current demands on performance. An embedded processor is suitable in many applications; it is flexible and easy to develop software for. The resulting implementation does however not have the performance of the corresponding FPGA or ASIC implementation. In recent years FPGAs have developed, from being a device mainly used to implement grids of 'glue-logic', to a very flexible device which can be used to develop complex hardware systems residing on one circuit. FPGAs are well suited for usage in cost and performance sensitive applications. Embedded systems, ASICs and FPGAs have been discussed based on literature found on the topic covering a wide span of considerations.

Furthermore a number of considerations which are important when implementing algorithms and logic in embedded processors and FPGAs were

described. Rewriting the algorithms while keeping the properties of the implementation platform in mind is important in order to decrease the demands on computational effort. Another very important issue when developing mathematical models intended for FPGAs and embedded systems is to find a suitable internal number representation. Using fixed point digital number representation gives more efficient results. Two different mathematical models were implemented as concept studies in order to illustrate the methodologies described and to make use of the theory.

An FPGA implementation of a reformulated but completely equivalent heat release calculation was shown. The implementation was capable of computing a heat release sample within 120 ns which is to be considered as immediate in a combustion engine context. Heat release analysis is within the automotive community often considered too computationally demanding to be implemented on board a vehicle. This proof of concept study indicates the achievable performance of FPGAs.

A NO_x model was developed implemented on an embedded processor. The output from the model was intended for closed loop Diesel combustion control. To be able to implement a fast NO_x model several techniques were used. Parts of the model were tabulated, difficult operators such as division were avoided and the properties of fast C code was kept in mind. The performance goal of the model was intra cycle performance, i.e. to compute NO_x formation during the same cycle it occurs with a high time resolution. The goal was almost achieved by the fixed point version of the NO_x algorithm which was able to compute in-cycle NO_x formation up to engine speeds of about 3000 rpm. The floating point version of the algorithm was not able to cope with engine speeds higher than about 1100 rpm, i.e. less than half as fast as the fixed-point implementation. To further increase the computation speed of the model several things could be attempted; Removing the Linux OS present on the processor, slightly reducing the time resolution or to use a slightly faster processor, it may even be a good option to implement the model using an FPGA. Even so, the developed NO_x model computes fast on an embedded processor and on-board, intra-cycle implementation of the NO_x model was possible!

The work undertaken indicated the possibility to implement high speed control oriented models in FPGAs and embedded processors. Methods have been devised to develop physical models which have very low demands on computational power. It is important to keep the limitations of the implementation environment in mind during the entire development process. This thesis aims to fill a gap between state space models, common in automatic control, and high fidelity physical models, commonly used for simulation, by providing a method to develop high fidelity control oriented models which are low in computation demand and implementable in FPGAs and embedded processors.

6

Bibliography

Andersson, M., J. B., H. A., and C. Nöhre (2006): "A real-time NO_x model for conventional and partially premixed diesel combustion." In *SAE*, number 2006-01-0195. Detroit.

Andersson, Ö. (2008): *Handbook of Combustion*, vol. 3, chapter Diesel Combustion. Wiley-VCH Verlag GmbH & Co. KGaA, Weinheim.

Bengtsson, J., R. Johansson, P. Strandh, P. Tunestål, and B. Johansson (2004): "Closed-loop combustion control of homogeneous charge compression ignition (hcci) engine dynamics." *International journal of adaptive control and signal processing*, **18:2**, pp. 167–179.

Bengtsson, J., R. Johansson, P. Strandh, P. Tunestål, and B. Johansson (2006): "Hybrid control of homogeneous charge compression ignition (HCCI) engine dynamics." *International journal of control*, **79:5**, pp. 422–448.

Bensson, R. S. and N. D. Whitehouse (1979): *Internal Combustion Engines*. Pergamon Press, Oxford.

Cantin, M.-A., Y. Savaria, and P. Lavoie (2002): "A comparison of automatic word length optimization procedures." In *IEEE International Symposium on Circuits and Systems (ISCAS 2002)*, vol. 2, pp. 612–615. IEEE, New York.

Cantin, M.-A., Y. Savaria, D. Prodanos, and P. Lavoie (2006): "A metric for automatic word-length determination of hardware datapaths." In *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, number 10, pp. 2228–2231. IEEE, New York.

Compton, K. and S. Hauck (2002): "Reconfigurable computing: A survey of systems and software." In *ACM Computing Surveys*, number 2. Danvers.

Damaj, I. W. (2006): "Parallel algorithms development for programmable logic devices." In *Advances in Engineering Software*, number 37, pp. 561–582. Maryland Heights.

Dec, J. E. (1997): "A conceptual model of di diesel combustion based on laser-sheet imaging." In *SAE*, number 970873. Detroit.

Egnell, R. (1998): "Combustion diagnosis by means of multi-zone heat release analysis and NO calculation." In *SAE*, vol. 981424. Dearborn.

Ericsson, C. and B. Westerberg (2006): "Modelling diesel engine combustion and NO_X formation for model based control and simulation of engine and exhaust aftertreatment systems." In *SAE*, number 2006-01-0687. Detroit.

Eriksson, L. (2004): "CHEPP - a chemical equilibrium program package for matlab." In *SAE*, number 2004-01-1460. Detroit.

Gatowski, J. A., E. N. Balles, K. M. Chun, F. E. Nelson, J. A. Ekichian, and H. J. B. (1984): "Heat release analysis of engine pressure data." In *SAE*, number 841359. Detroit.

Guzzella, L. and C. Onder (2004): *Introduction to Modeling and Control of Internal Combustion Engine Systems*. Springer, Berlin, Heidelberg.

Hervé, N., D. Ménard, and O. Sentieys (2005): "Data wordlength optimization for FPGA synthesis." In *Signal Processing Systems Design and Implementation*, pp. 623–628. IEEE, New York.

Heywood, J. B. (1988): *Internal Combustion Engine Fundamentals*. McGraw-Hill, New York.

Horn, U., E. Rijk, R. Egnell, O. Andersson, and B. Johansson (2008): "Investigation on differences in engine efficiency with regard to fuel volatility and engine load." In *SAE*, number 2008-01-2385. Chicago.

IEEE, P1800, System Verilog Work Group (2001): *1364—2001 IEEE standard Verilog hardware description language*. IEEE, New York.

IEEE, VASG: VHDL Analysis and Standardization Group (2007): *1076C-2007 IEEE Standard VHDL Language Reference Manual*. Number ISBN 0-7381-5523-3. IEEE, New York.

Kiencke, U. and L. Nielsen (2000): *Automotive Control Systems - For Engine, Driveline and Vehicle*. Springer, Berlin, Heidelberg.

Kuon, I. and J. Rose (2007): "Measuring the gap between FPGAs and ASICs." In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, number 2. IEEE, New York.

Lewander, M., B. Johansson, P. Tunestål, N. Keeler, N. Milovanovic, and P. Bergstrand (2008): "Closed loop control of a partially premixed combustion engine using model predictive control strategies." In *9th International Symposium on Advanced Vehicle Control, AVEC'08*. Kobe, Japan.

Marwedel, P. (2003): *Embedded System Design*. Kluwer Academic Publishers, Dordrecht.

Monmasson, E. and M. N. Cirstea (2007): "FPGA design methodology for industrial control systems — a review." In *Transactions on Industrial Electronics*, number 4. IEEE, New York.

Nishida, K. (2006): "Combustion and emission formation processes in d.i. diesel engine under various injection strategies." In *FISITA*, number F2006P278. Yokohama.

Olsson, J.-O., P. Tunestål, and B. Johansson (2001): "Closed-loop control of an HCCI engine." In *SAE*, number 2001-01-1031. Detroit.

Onishi, S., S. Jo, K. Shoda, P. D. Jo, and S. Kato (1979): "Active Thermo-Atmosphere Combustion (ATAC)—a new combustion process for internal combustion engines." In *SAE*, number 790501. Detroit.

Powell, J. D. (1993): "Engine control using cylinder pressure - past, present and future." *Journal of Dynamic Systems Measurement and Control - Transactions of the ASME*, **No 115**, pp. 343–350.

Randolph, A. L. (1990): "Methods of processing cylinder-pressure transducer signals to maximize data accuracy." In *SAE*, number 900170. Detroit.

Shaver, G. M., J. C. Gerdes, and M. Roelle (2004): "Physics-based closed-loop control of phasing, peak pressure and work output in hcci engines utilizing variable valve actuation." In *Proceeding of the 2004 American Control Conference*, vol. 1, pp. 150–155. Boston.

Todman, T., G. Constantinides, S. Wilton, O. Mencer, W. Luk, and C. P.Y.K. (2005): "Reconfigurable computing: Architectures and design methods." In *IEE Proceedings - Computers and Digital Techniques*, number 2, pp. 193–207. IEEE, New York.

Tunestål, P. (2000): *Estimation of the In-Cylinder Air/Fuel Ratio of an Internal Combustion Engine by the Use of Pressure Sensors*. PhD thesis, University of California, Berkeley.

Tunestål, P. (2007): "Self tuning cylinder pressure based heat release computation." In *Proceedings for the Fifth IFAC Symposium on Advances in Automotive Control*, number AAC07-30, pp. 183–189. IFAC, Aptos, CA.

Vahid, F. and T. Givargis (2002): *Embedded Systems Design*. John Wiley & Sons, Hoboken.

Woschni, G. (1967): "A universally applicable equation for instantaneous heat transfer coefficient in the internal combustion engine." In *SAE*, number 670931. Detroit.

A

Abbreviations

ADC	Analog-to-Digital Converter
ALG	Algorithm
ASIC	Application-Specific Integrated Circuit
CAD	Crank Angle Degree
CADP	Crank Angle Degree Pulse (not necessarily with on-degree interval)
COTS	Commercial Off-The-Shelf
CPLD	Complex Programmable Logic Device
DAC	Digital-to-Analog Converter
DC	Direct Current
DFG	Data Flow Graph
DSP	Digital Signal Processor
EGR	Exhaust Gas Recirculation
FIR	Finite Impulse Response (filter)
FPGA	Field Programmable Gate Array
GoM	Golden Model
HC	Un specified hydrocarbon fuel
HCCI	Homogeneous Charge Compression Ignition
HR	Heat Release
I/O	Input/Output
IVC	Inlet Valve Close
JTAG	Joint Test Action Group (standard test access port)
LSB	Least Significant Bit
MB	Megabyte
Mb	Megabit
MPC	Model Predictive Control
PAL	Programmable Array Logic
PC	Personal Computer
PM	Particulate Matter
rpm	Revolutions Per Minute

RTL	Register Transfer Level
SGDSP	System Generator DSP
TDCP	Top Dead Center Pulses
VHDL	VHSIC hardware description language
VHSIC	Very-High-Speed Integrated Circuits

B

Symbols

α, θ	Engine crank position (crank angle degree)
α_{safr}	Stoicometric air fuel ratio
γ	Ratio of specific heats
λ	Relative air-fuel ratio
λ_{cal}	Burned zone lambda, model calibration variable
CO	Carbon monoxide
CO_2	Carbon dioxide
H_2	Hydrogen
H_2O	Hydrogen oxide (water)
N, N_2	Nitrogen
NO	Nitrogen oxide
NO_x	Nitric oxides, including both NO and NO_2
O, O_2	Nitrogen
ADC_{clk}	ADC clock frequency
a_{XX}	Dimensionless equilibrium concentration of species XX
$CAXX\%$	The time instance for XX% of total heat release
c_{NO}	Concentration of NO
C_v	Molar specific heat at constant volume
c^e_{XX}	*Equilibrium* concentration of species XX
DAC_{clk}	DAC clock frequency
$FPGA_{clk}$	FPGA clock frequency
K_{pX}	Equilibrium constants
M_a	Mole mass of air
M_f	Mole mass of fuel
m_a^{bz}	Mass of air in the burned zone
m_f^{bz}	Mass of fuel in the burned zone
n	Number of moles
n_0	Number of moles at a zero datum point

n_{bz}	Number of moles in the burned zone
n_a^{bz}	Number of moles of air in the burned zone
n_f^{bz}	Number of moles of fuel in the burned zone
n_g	Global number of moles
n_{NO}	Number of moles of NO
n_{uz}	Number of moles in the the unburned zone
P	Pressure *ratio*
p_0	Pressure at a zero datum point
p_{cyl}, p	Cylinder pressure
p_{g0}	Global pressure at a zero datum point
p_{lsb}	Pressure resolution (value of LSB)
p_{max}	Maximum pressure
Q	Released heat
Q_{HR}^{net}	Net heat released from combustion
Q_{lhv}	Lower heating value (of fuel)
$\widetilde{R}$	Molar gas constant (ideal gas constant)
T	Temperature
T_0	Temperature a zero datum point
T_{bz}	Temperature of the burned zone
T_g	Global temperature
T_{g0}	Global temperature at a zero datum point
T_{uz}	Temperature of the unburned zone
t	Time
t_{exec}	Execution time
U	Internal energy
V	Volume
V_0	Volume at a zero datum point
V_{bz}	Volume of the burned zone
V_g	Volume of the total combustion chamber
V_{uz}	Volume of the unburned zone
W	Mechanical work
X_{NO}	Fraction of NO (on mole base)

C

C Code Listing, Floating-Point NO_x Model

```
//++Inclusion++
#include <stdio.h>

//++Definitions and globals++
//Computation "windows" (INTs)
#define minHRScope 1500//Min of cycle window
#define maxHRScope 2200//Max of cycle window
#define pegPMin 1492//Min point for HR peg point
#define endWind 2184//End window, used for max search
#define algoUpdaP 1//CAD point for cycle change (variable update)
#define nRLock 1152//Crank angle where to lock nR
#define cycleLen 3600//The length of an engine cycle (resolution)
#define isenPTLock 1732//CAD point for isentrpo lock

//Constants, float
#define qTrigLim 0.125//NO computation start Q threshold (12.5%)
#define oneDivGamGMin1 2.7778//Constant 1/(gam-1)
#define qToLambdaConst 1.3319e-05//Constant used to comp. nbz from Q
#define invR 0.1203//Constant 1/R
#define p0 1.0130e+05//Normal atmos. pres. (for computing pres. rat)
#define r 8.3144720//Universal gas constant R
#define isenTabAmpli 64.0//Amplification for index in isentrop
#define calcQOffs 16//Offset on Q trace, 16J

//Constants memory maps, read from file
static float V[3600];//Engine volume as function of CAD
static float dV[3600];//Engine derivative volume as function of CAD
static float isenTab[256];//Isentrop Y value (temperature)
static float dNO[32][64][64];//NO formation rate
```

```
//In cycle variables
//State variables
static float nR = 0;//For global number of moles and temperature
static float ng = 0;//Global number of moles
static float genStartNOx = 0;//Point where to start comp. NO (CAD)
static float oldCNO = 0;//NO concentration in previous sample
static float oldVcz = 0;//Volume of combustion zone, previous samp.
//HR related
static float cumP = 0;//Cumulative pressure (Per HR)
static float pegP = 0;//HR P*V peg point (Per HR)
static float nextMaxP = 0;//Max HR point, used for normalization
static float nextMinP = 0;//Min HR point, used for normalization
static float qNormConst = 0;//Normalization constant, current cycle

//inter cycle variables
static float isenPLock = 0;//Isentrpoic lock pressure
static float isenTLock = 0;//Isentropic lock temperature
//HR related
static float maxPFilt[4];//Vector; HR norm. max point filter
static float filtMaxP = 0;//Current filtered max point
static float filtMinP = 0;//Current, (not filtered), min point

//++Function declarations++
//Returns the current NO formation rate, a func. of conc.,
// burned zone temp. and current pressure ratio.
float lookupNO(float currCNO, float currTcz, float currRatP);

//Takes "one step" with the model, input is engine position (CAD),
//cylinder pressure, intake temperature, fuel amount belonging to
//current cycle and the time between CADS (engine speed).
float iterateModel(int CAD, float cylP, float inT, ...
                   float qf, float dt);

//Reads the maps needed (V, dV, isentrop, dNO) from .dec files.
//Returns success or failure, 1,0.
int installMaps(void);

//Main function performs the actual test run
int main(void);

//++The code++
float iterateModel(int CAD, float cylP, float inT, ...
                   float qf, float dt)
{
  //Variable declarations and initiations
  //HR related
  float constCylPV, cylPV, dVP, realCurrQ, q;
```

```
realCurrQ = 0;
cylPV = 0;
constCylPV = 0;
q = 0;

//Moles, temperatures and volume
float nb, nuz, tg, tuz, tbz, pVDivR, vcz, pczVcz;
nb = 0;
nuz = 0;
tg = 0;
tuz = 0;
tbz = 0;
vcz = 0;
pVDivR = 0;

//The isentrop
float isenPRat, interpIsen, theIsentrop;
int isenTabPos;//Look up table position
isenPRat = 0;
isenTabPos = 0;
interpIsen = 0;
theIsentrop = 0;

//NO related
float ratP, cNO, dNOdt, dNOdV, xNOtot;
genStartNOx = 0;
pczVcz = 0;
ratP = 0;
cNO = 0;
dNOdt = 0;
dNOdV = 0;
xNOtot = 0;

//New cycle clear the states and update inter-cycle states
if(CAD == algoUpdaP)
  {
    //Compute the filtered max point, HR
    filtMaxP = (maxPFilt[0]*.125) + (maxPFilt[1]*.125) + ...
      (maxPFilt[2]*.25) + (maxPFilt[3]*.5);
    //Update filter state
    maxPFilt[3] = maxPFilt[2];
    maxPFilt[2] = maxPFilt[1];
    maxPFilt[1] = maxPFilt[0];
    maxPFilt[0] = nextMaxP;

    //Compute min point, HR
    filtMinP = nextMinP;
```

```
    //Compute normalization constant
    qNormConst = qf/(filtMaxP-filtMinP);

    //Update state variables for new cycle
    //HR
    cumP = 0;
    pegP = 0;
    nextMaxP = -1000;
    nextMinP = 1000;
    //Moles, temperatures, volume and isentrop
    nR = 0;
    ng = 0;
    isenPLock = 0;
    isenTLock = 0;
    //NO algorithm
    genStartNOx = 0;
    oldCNO = 0;
    oldVcz = 0;
  }

//Compute the global number of moles
if(CAD == nRLock)
  {
    nR = cylP*V[CAD]/inT;
    ng = nR*invR;
  }

//These variables are used several times = precomputed
if(CAD > pegPMin && CAD <= maxHRScope)
  {
    cylPV = cylP*V[CAD];
    constCylPV = cylPV*oneDivGamGMin1;
  }

//Compute the HR datum point
if(CAD == minHRScope)
    pegP = constCylPV;

//Compute HR, moles and global temp
if(CAD > minHRScope && CAD <= maxHRScope)
  {
    //Update cumulative sum, HR
    dVP = cylP*dV[CAD];
    cumP = cumP + dVP;
    //Compute Q according to Per T method, HR
    realCurrQ = constCylPV + cumP - pegP;
```

```
    //Compute final Q, HR
    q = (qNormConst*realCurrQ - filtMinP)+calcQOffs;

    //Compute the moles
    nb = q*qToLambdaConst;
    nuz = ng-nb;

    //Global temperature
    tg = cylPV/nR;
  }

//Compute current cycle HR max point, for normalization, HR
if(CAD == maxHRScope)
  nextMaxP = realCurrQ;

//Find current cycle minimum HR point for Q "pegging", HR
if(CAD > minHRScope && CAD <= maxHRScope && realCurrQ < nextMinP)
  nextMinP = realCurrQ;

//Determine NOx algorithm start point based on prev. Qmax
if(CAD > 1751 && genStartNOx == 0 && q > qTrigLim*filtMaxP)
  genStartNOx = CAD;

//Lock for the isentropic relationship
if(CAD == isenPTLock)
   {
    isenPLock = cylP;
    isenTLock = tg;
  }

//Run the NO formation algorithm
if(genStartNOx != 0 && CAD >= genStartNOx && CAD <= maxHRScope)
  {
    //Isentr. pressure ratio
    isenPRat = cylP/isenPLock;
    //Compute position in isentropic table
    isenTabPos =  (int)(isenTabAmpli*isenPRat);
    //Perform the isentropic look up
    interpIsen = isenTab[isenTabPos];
    //Compute the unburned zone temperature
    tuz = (interpIsen*isenTLock);

    //Compute burned zone temperature
    pVDivR = ng*tg - nuz*tuz;
    tbz = pVDivR/nb;

    //Compute pressure ratio
```

```
        ratP = cylP/p0;

        //Perform NO look up!
        dNOdt = lookupNO(oldCNO, tbz, ratP);

        //Compute volume of combustion zone
        pczVcz = nb*r;
        pczVcz = pczVcz*tbz;
        vcz = pczVcz/cylP;

        //Compute cNO volume derivative
        dNOdV = ((oldCNO/vcz)*(vcz-oldVcz));

        //Compute current burned zone NO concentration
        cNO = oldCNO + dNOdt*dt - dNOdV;

        //Compute overall NO concentration
        xNOtot = (vcz*cNO)/ng;

        //Update state, Store variables
        oldVcz = vcz;
        oldCNO = cNO;
      }

    //Return the computed fraction of NO
    return xNOtot;

  }

  //NOTE this must be changed when the map is changed
  float lookupNO(float currCNO, float currTcz, float currRatP)
  {

    int cNOPos, TczPos, ratPPos;

    //Current cNO lookup position
    cNOPos = 0;
    if(currCNO < 2.83122062683105e-07)
      cNOPos = (int)(currCNO*67108864);
    else if(currCNO < 5.21540641784668e-07)
      cNOPos = (int)(((currCNO-3.12924385070801e-07)*33554432) + 20);
    else
      cNOPos = (int)(((currCNO-5.81145286560059e-07)*16777216) + 28);

    //Combustion zone temperature look up position
    TczPos = 0;
    if(currTcz < 1920.00)
```

```
    TczPos = (int)((currTcz-1024.00)*0.0078);
  else if(currTcz < 2432.00)
    TczPos = (int)((currTcz-1984.00)*0.0156) + 8;
  else if(currTcz < 2688.00)
    TczPos = (int)((currTcz-2464.00)*0.0312) + 16;
  else
    TczPos = (int)((currTcz-2704.00)*0.0625) + 24;

  //Pressure ratio look up position
  ratPPos = 0;
  if(currRatP < 40.00)
    ratPPos = (int)(currRatP-4.00)*0.25;
  else if(currRatP < 80.00)
    ratPPos = (int)(((currRatP-42.00)*0.5) + 10);
  else
    ratPPos = (int)(currRatP-81.0)+30;

  //Limit to avoid segmentation fault
  if(cNOPos > 31)
      cNOPos = 31;
  else if(cNOPos < 0)
      cNOPos = 31;

  if(TczPos > 63)
      TczPos = 63;
  else if(TczPos < 0)
    TczPos = 0;

  if(ratPPos > 63)
      ratPPos = 63;
  else if(ratPPos < 0)
    ratPPos = 0;

  //Perform and return the actual look up!
  return dNO[cNOPos][TczPos][ratPPos];
}

int main(void)
{

  //Variable declarations
  int CAD;

  float cylP, xNO, inT, qf, dt;

  //Intake temperature and energy flow point 3
  inT = 3.828398478260870e+02;
```

```
    qf = 8.128702457102349e+02;

    //Engine speed dependent (time between two samples)
    dt = 2.7760e-05;

    //Install tables and check so that we were successfully
    if(installMaps() == 0){
      printf("Error installing maps\n");
      return 0;
    }

    //Open input and output file
    FILE *fidIn;
    FILE *fidOut;
    fidIn = fopen("theData3.dec","r");
    fidOut = fopen("outputData.dec","w");

    //Check if files are open and we are ready to rock!
    if(fidIn != NULL && fidOut != NULL)
      {

        //Loop until no more data points
        while(!feof(fidIn))
          {
            //Read the values from the data file
            fscanf(fidIn,"%i %E",&CAD,&cylP);

            //Run the model
            xNO = iterateModel(CAD, cylP, inT, qf, dt);

            //Store output to file
            fprintf(fidOut,"%E\n",xNO);
          }
      }
    else
      {
        printf("Error reading input data\n");
        return 0;
      }

    //Close files
    fclose(fidIn);
    fclose(fidOut);

    return 1;

  }
```