LUND UNIVERSITY

Field Programmable Gate Arrays and Reconfigurable Computing in Automatic Control

Wilhelmsson, Carl

2007

ISRN LUTMDN/TMHP—07/7050—SE
ISSN 0282-1990

Field Programmable Gate Arrays and Reconfigurable Computing in Automatic Control

Carl Wilhelmsson

Thesis for the Degree of Licentiate in Engineering

Division of Combustion Engines
Department of Energy Sciences
Lund University

Field Programmable Gate Arrays and Reconfigurable Computing in Automatic Control

Field Programmable Gate Arrays and Reconfigurable Computing in Automatic Control

Carl Wilhelmsson

Division of Combustion Engines
Department of Energy Sciences
Lund University
Lund, December 2007

Till min Ella och min familj

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

ISSN 0282–1990
ISRN LUTMDN/TMHP--07/7050--SE

Printed in Sweden by Media-Tryck.
Lund 2007

Acknowledgements

The first person I would like to thank for his support and understanding is Per Tunestål. Per has as my supervisor been an excellent mentor for me in the process of 'maturing' academically. 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 hoped to find before I meet her. My brothers and parents, supporting me now and before are very important persons in my life, have always been and will always be. The rest of my family are also precious to me. Friends are important to have, in good times and in bad, I have good friends which deserves appreciation, you know who you are. The department has a lot of young employees and the atmosphere is open minded and good. I especially want to thank Thomas for good friendship on and off work, sharing a great interest in motorbikes. Andreas, Håkan, Leif and Jari have been very good friends at work before they moved on to new challenges. Professor Bengt Johansson deserves an acknowledgement for setting up an adequately founded and creative research environment, it is inspiring to work in collaboration with large and market leading companies. I also want to thank professor Rolf Johansson and professor Anders Rantzer for providing a lot of important automatic-control ideas and support. The help of Leif Andersson has been valuable while typesetting this thesis. The technicians have helped me a lot in the workshop keeping up my interest in workshop work in different ways. 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. Yanagihara Hiromichi, also with Toyota, helped me to get to Japan and has maintained contact with me since, keeping an eye on me and my research. Finally to those not mentioned here I am not less grateful.

Carl

Preface

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

An Ultra High Bandwidth Automotive Rapid Prototype System

International Federation of Automatic Control (IFAC) 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

Peripheral 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

Outline

This thesis has the form of a monograph meaning that no papers are included in the end. Never the less material from previous publications are covered by it, the covered publications are the ones listed under the title 'publications' above. The first section encountered by the reader would be the introduction, covering the motivation of this work, putting the work into its context and setting the focus of the thesis. Following the motivation are two sections which, very briefly, touch internal combustion engines as such and, a bit more in detail, how to carry out combustion engine feedback control.

The second chapter describes, in detail, the Field Programmable Gate Array (FPGA), the first section generally, the second it's history. The third section of the chapter describes architectural and design considerations, both the architecture of the actual device as well as different architectural considerations on the design level are discussed. Tools and design methods are described in the fourth section, covering topics as the basic steps carried out by a design tool, low level design, different high level design tools and their corresponding pros and cons. Second to last, to illustrate the power and applicability of the FPGA technology, a flavor of FPGA applications are offered the reader. The chapter is ended with a summary.

Implementing feedback-controllers in an FPGA environment takes special considerations devoted one chapter, Chapter three. Giving an introduction to the topic of FPGA implemented controllers, compared with microcontroller implemented ones. Continuing on to the second section presenting considerations to be made generally implementing digital control, including special considerations for the FPGA environment. Section three discusses the practical issue of internal word-length optimization, Section four handles the issues arising when implementing highly over-sampled control systems and Section five describes considerations which have to be made regarding parallelization and re-formulation of control algorithms in order to make them efficient in a FPGA implementation. A flavor of control applications implemented in FPGAs follows with the intention to give the topic legitimacy and engage the reader, lastly this chapter is ended with a chapter

summary.

Chapter four deals with the work presented in the first and second paper 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, of course, the outcome meaning performance of the final system. This work was intended as a 'proof of concept'.

Approaching the end of this thesis, second to last, a chapter describing an intended 'rapid prototype' system, featuring FPGA hardware and with the capabilities of implementing controllers dealing with very fast feedback control loops. This system is mainly intended for combustion engine feedback control experiments, but the ideas can be reused for similar problems.

Some concluding remarks ends the thesis.

Contents

1

Introduction

1.1 Motivation

As the reader may know internal combustion engines have been the main energy source in mobile applications for something like a century. The reader is probably also familiar with the great threat to the environment of our planet which are 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 worsened 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), hydrocarbon (HC), and carbon-monoxide (CO). Due to environmental issues, the green house effect and increasing fuel prices, there is of course a strong urge to improve the internal combustion engine. One part which is regarded as an important factor for improving the environmental and economical performance of engines is feedback control of various engine parameters.

The author makes no claims of writing an exhaustive description of combustion engines as such. Even so they have to be briefly discussed in order to give the reader an idea of the frame within which this work has been undertaken. It is also important to introduce the reader to the topic, enabling understanding of the relevance of this work.

Instead of an exhaustive description of combustion engines the Field Programmable Gate Array (FPGA) is described. FPGA internals, design, benefits and drawbacks are described, with a feedback-control perspective in mind. The topic targeted with this thesis is how to use the FPGA as a potentially powerful tool for feedback control, both generally speaking and regarding its application in the combustion engine feedback control field.

For most feedback control solutions the capabilities of 'normal' processor systems are more than enough and the price performance ratio of processors

is good. Even so there are potential benefits in using Field Programmable Gate Arrays for feedback control and especially feedback control of internal combustion engines. The benefits could be implementation of feedback control loops in timescales not yet possible, and not expected to be possible in a very long time. Potential benefits could also be the possibility to use models more complex than possible with conventional techniques in feedback control loops. Economically speaking FPGAs have the potential to outperform conventional techniques. FPGA systems can hence potentially serve as powerful tools for feedback control of, among other the internal combustion engine.

1.2 The Application, the Internal Combustion Engine

Traditionally there have been two different kinds of engines, the Otto engine (the normal 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]. Instead of the Otto or Diesel engine a third type of internal combustion engine principle will be discussed below. This third engine type is called Homogeneous Charge Compression Ignition (HCCI) and was first suggested by [Onishi *et al.*, 1979]. The HCCI engine 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 as opposed to 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 originating from Diesel engines with the low emissions originating from Otto engines. 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.

The HCCI Engine

Despite intense research efforts in recent years the HCCI engine is still less known than its relatives the Otto and the Diesel engine. Nevertheless it seems to be one of the more promising engine concepts of the future, combining high efficiency with ultra-low emissions of nitrogen oxides. When the concept of HCCI was first reported by [Onishi *et al.*, 1979] is was primarily an attempt to reduce emissions of unburned hydrocarbons (HC) and improve part load efficiency of two stroke engines. After Onishi et al. published their results it took a while before further HCCI publications emerged,

when they did the publication rate increased. Important early publications are for example [Najt and Foster, 1983], [Thring, 1989] and [Stockinger *et al.*, 1992].

The operation principle of the HCCI engine makes it possible to increase the efficiency compared to the Otto engine due to the avoidance of throttling losses. At the same time the high soot and nitrogen oxide emissions of the diesel engine are avoided. 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 undiluted zones.

HCCI combustion can be achieved in numerous ways, both in two and four stroke engines. High compression ratio can be applied [Haraldsson *et al.*, 2002], the inlet air can be pre heated [Martinez-Frias *et al.*, 2000]. HCCI combustion can be induced by unconventional valve strategies that retain hot residuals [Milovanovic *et al.*, 2004] and the octane number of the fuel can be altered [Olsson *et al.*, 2001] to modulate the ignition temperature. Since the HCCI combustion process in many operating points is unstable, 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.

Even though it has many good features, the HCCI engine also has some limitations besides the, just described, need for feedback combustion control. The operational principle unfortunately suffers from very high combustion rates, causing noise as well as wear of 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.

1.3 Combustion Engine Feedback Control

Internal combustion engines have, every since they were first developed been under control/feedback-control. In fact control/feedback-control of combustion engines is an 'enabler' for the success of the entire engine technique. An engine which can not deliver a controlled amount of energy at a controlled engine speed is of no use. The control tasks to be carried out by the engine controller depends greatly on the engine type and the performance requirements of the engine. This very brief section will mainly be devoted to what is considered state-of-the art control technology in the HCCI field, being a more challenging control task than state-of-the art gasoline or Diesel engine control. An early view of state-of-the art gasoline engine control was provided by [Powell, 1993]. For the Diesel engine however similar publications are sparse or completely lacking, such work has just recently been

undertaken. [Kiencke and Nielsen, 2000] provides a very interesting view for readers mainly interested in more production near combustion engine feedback control, as well as feedback control within other parts of the automotive domain (suspension, steering, breaking and driveline).

HCCI Engine Control

Besides development of the actual HCCI principle much work has been carried out addressing the great task of developing feedback combustion control systems for the HCCI engine. The highly non-linear nature of HCCI combustion together with the fact that actuators with high control authority are lacking poses a great challenge to researchers in the field. Furthermore, as indicated by Figure 1.1, the HCCI engine represents an unstable system which needs feedback control for successful operation. The two most important variables to control in an HCCI engine are combustion phasing and engine load. Combustion phasing is simply a number describing when during the engine cycle combustion takes place and engine load is a number describing how much work the engine develops.

Many interesting results, starting with [Olsson *et al.*, 2001], have been published attempting HCCI control using various feedback control techniques. Some of these attempts to utilize 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 the engine system. Interesting results has been published, for example by [Shaver *et al.*, 2004] who utilizes a physically based model for HCCI control (meaning combustion phasing and output work). Results using identified instead of physically based models for HCCI control have for example been published by [Bengtsson *et al.*, 2006] using Model Predictive Control (MPC) with an identified model for HCCI control (Bengtsson et al. also covers other interesting topics in the area of HCCI control). The point being that mathematical models contained in the control system are playing an increasingly important role in the attempts of tackling the difficult HCCI control problem. A fact which also is true in the attempts of improving control and performance of the Otto and Diesel engines.

Cylinder Pressure Measurements

Cylinder pressure is a very powerful measurement signal when conducting engine feedback control, [Tunestål, 2000] quotes Professor A. K. Oppenheim who said that the 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 the cylinder pressure both combustion phasing and

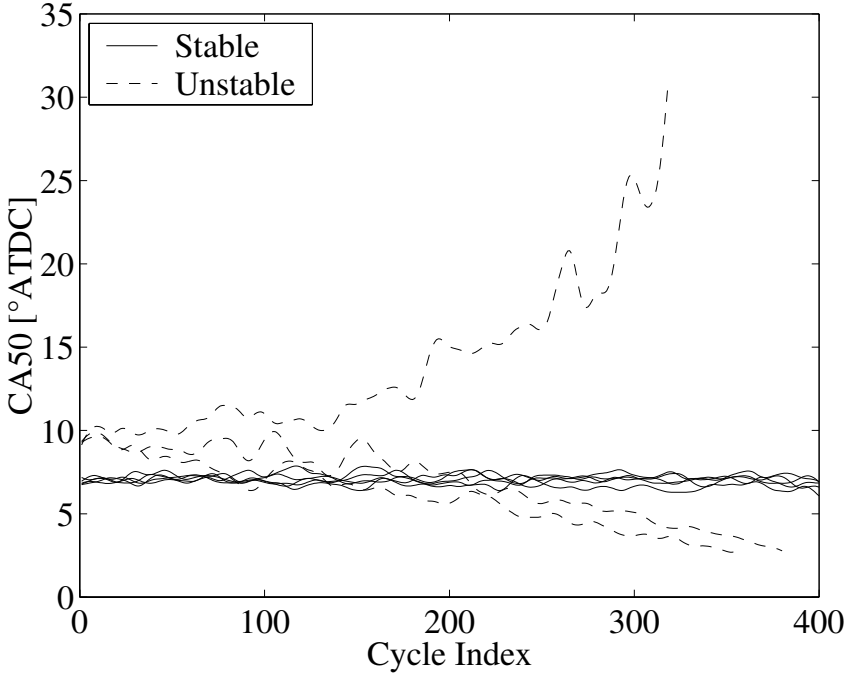

Figure 1.1 HCCI combustion instability, combustion phasing is shown as function of cycle-number for different cases. Combustion-phasing feedback-controllers are switched off at cycle zero, note that some cases are stable (combustion phasing is maintained) while others change combustion phasing spontaneously. Figure found in [Olsson *et al.*, 2002]

engine load can be calculated, as well as other important parameters. 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 piezo-effect to a measurable voltage. A charge amplifier however can not be constructed without some leakage current, 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 ways;

- The cylinder pressure at bottom dead center of the intake stroke of the engine equals the pressure of the intake manifold.
- The average cylinder pressure during the exhaust stroke is equal to the back pressure in the exhaust system.
- The compression is poly-tropic and the poly tropic exponent is known and fixed.

Tunestål offers a far more detailed explanation of how to treat the signal from a cylinder pressure transducer, both in general and specifically using the third method above. In [Tunestål, 2007] the same author shows a heuristic approach which, using non linear least-squares estimation, finds both the poly tropic exponent and the DC level of the pressure signal simultaneously.

Heat Release Analysis

If the pressure within a cylinder is known it is possible to calculate the released heat within that cylinder after each engine cycle using thermodynamic equations as shown by [Gatowski *et al.*, 1984], performing a Heat Release (HR) analysis. The HR model of Gatowski et al. accounts for losses as well, losses included are losses through heat transfer to the combustion chamber walls and losses originating from mass loss caused by leakage past the piston rings. Heat transfer losses are calculated based on the results presented by [Woschni, 1967] while the model for crevice losses is developed by Gatowski et al. For feedback control purposes the different losses are often neglected as discussed by [Bengtsson *et al.*, 2004], the reason being that combustion phasing (see Figure 1.2) 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 visible in Equation 1.1 which is a version of Equation 9 in Gatowski et al., neglecting losses.

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

Furthermore [Tunestål, 2007] has recently expanded his work, as previously noted.The benefit from Tunestål is that the explicit heat transfer model and model for losses over the piston rings used by Gatowski et al. no longer are needed since the method of Tunestål includes these effects implicitly. The main drawback with the approach taken by Gatowski et al., which is parameter tuning, is in this way avoided. The models included in Gatowski et al. and Woschni need to be parametrically tuned to fit every specific application. Using the method of Tunestål it is possible to avoid the non-heuristic

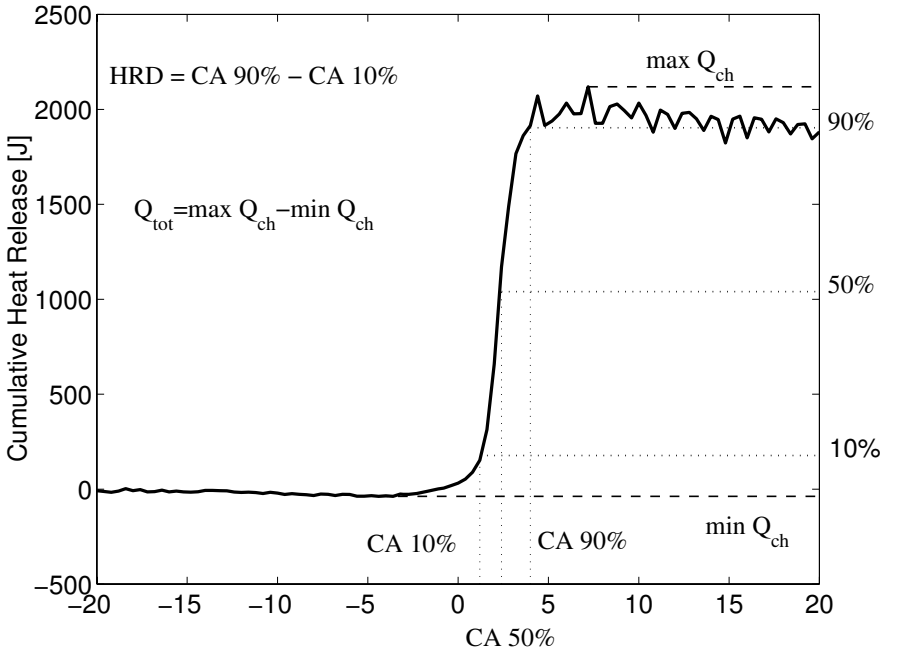

Figure 1.2 A typical heat-release curve (the integration of Equation 1.1) with the important combustion phasing, defined as the instance when half of the total heat has been released (half of the combustion has taken place), indicated (CA50%). Bottom axis in the figure has the unit Crank Angle Degree meaning that CA50% has the same unit. Figure found in [Tunestål, 2000].

and completely 'ad-hoc' tuning procedure associated with traditional HR analysis according to Gatowski et al.

2

The Field Programmable Gate Array (FPGA)

2.1 FPGA Fundamental Description and its Processor Comparison

The 'normal' processor technology is well known to many people, if not through its internal operation it is know for being the 'heart' of a normal personal computer. The essence of 'normal' processor technology is not explained in detail here, but some generalizing statements are made in order to help the reader understand the difference between an FPGA and a processor.

A processor is a sequential device which executes a program consisting of a number of single instructions, the same unit must in this case be able to handle many different instructions. The 'power' of each instruction and the number of clock cycles it takes the processor to compute one instruction differ depending on the architecture. The 'best case' is one instruction per clock cycle but this is not the average rate of instruction completion. It should also be noted that a processor normally is programmed using a 'high level' programming language, each high level language instruction consists of several 'low level language' instructions hence of course a high level instruction would normally need many clock cycles to complete its operation on the processor. The benefits with this operational principle are that the processing device is very general and a large number of different programs, used to solve different problems can be run on the same device without modifications to the actual device.

An FPGA on the other hand does *not* execute instructions at all, an FPGA is an net of logic components which can be connected in a way so that the

device performs a specific operation of varying complexity. Inside the FPGA information is transferred to the different sub-nets by electric signals. If we compare the FPGA with the processor we find that *no* instructions are executed on the FPGA, instead the input is presented to the FPGA device through input signals. The input signals propagates through the FPGA using the internal connections of the device and finally the result is present on the outputs of the device. Normally, one result per clock cycle can be guaranteed on the output with an FPGA design. Another 'strength' of the FPGA technique compared to the processor is the inherited suitability for problems which are of parallel nature. To illustrate this with an example; if a specific computational problem consist of two different parts which first have to be calculated independently the two intermediate results are then summed before the final result is obtained. For simplicity it is also assumed that these two different parts are equally complex to calculate. The FPGA can in this case calculate the two different intermediate results simultaneously due to the fact that two independent signal paths will be implemented in the FPGA and the final result is obtained after a simple addition of the two results. The processor would in this case need to compute the intermediate results one at a time before it calculates the final sum, since the intermediate results are similarly complex the processor would in this case need at least twice the time of the FPGA to complete the calculation but probably more depending on which instructions are involved. This generalized simple example illustrates the benefits with the parallelism featured in the FPGAs and how it can be used.

The parallel nature of the FPGA in combination with the fact that the internals are customized completely for the application makes it a precious tool for systems where very high computational power is needed. Even if processors continue to evolve with the current rate they will fall short of performance. In many cases a, even state of the art, processor would not be enough and to make matters worse a state of the art processor is very expensive, power thirsty and is hence not an option.

2.2 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 a, often larger, version 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) and the size 'ranking' from smallest to largest would be: PAL, CPLD, FPGA. 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

Figure 2.1 An FPGA circuit.

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. Modern FPGAs can host designs with an 'equivalent gate count' of many million gates. They now a days contain more and more complex peripheral devices, e.g. processor cores, Digital Signal Processing (DSP) blocks, even 'mixed-mode' FPGAs exist containing analog and partly analog parts for example Analog-to-Digital Converters (ADC) or analog filters. The FPGA have evolved to become a flexible, cost effective and high power device suitable for a wide variety of applications.

2.3 FPGA Architecture and Design Considerations

[Todman *et al.*, 2005] and [Compton and Hauck, 2002] have both published excellent surveys of FPGA design considerations, dealing with FPGA hardware, FPGA design, design tools, design flow and design architectures. There are a large number off different considerations to make when designing for FPGAs both regarding hardware selection, design tool selection as well as selecting an appropriate design architecture, an overview follows. Figures 2.2 and 2.3 were adapted from Todman et al. while Figure 2.4 was found in Compton and Hauck.

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'. The functional units is 'were it happens' meaning that the actual logics is implemented in the 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, one can speak of either fine grained or coarse grained architectures, both in the case of the reconfigurable interconnect and in the case of the functional units. Selecting between fine grained and coarse grained interconnect/functional units essentially is a trade-off between flexibility (which gains from a fine grained architecture) and speed/overhead (gaining from a more coarse grained architecture). A fine grained fabric can better be adapted to 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 much faster for the problems where they are well suited.

For an architecture with fine grained functional units a functional unit typically is 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 parallel connections between the closest neighbours of each cluster, and via the reconfigurable interconnect for signaling to clusters positioned in other parts of the FPGA circuit. For the coarse grained architecture the lookup table would be replaced by for example a multiplier block, and Arithmetic Logic Unit (ALU) or a memory blocks. These coarse grained blocks would perform considerably better on their specific task than the fine grained lookup tables would, but the coarse grained blocks might not be of any use for some applications. Modern FPGAs normally contain both fine grained and coarse grained blocks in the same circuit, the fine grained reconfigurable fabric is often complemented by for example an ALU or other more coarse grained components with the intention to increase the speed for some common operations. Figure 2.2 shows the structure of fine grained versus coarse grained functional units.

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 see Figure 2.3 for visualisation. As before a coarse grained structure is less flexible but demands less overhead than the more flexible fine grained interconnect structure.

The interconnect together with the functional units forms the reconfig-

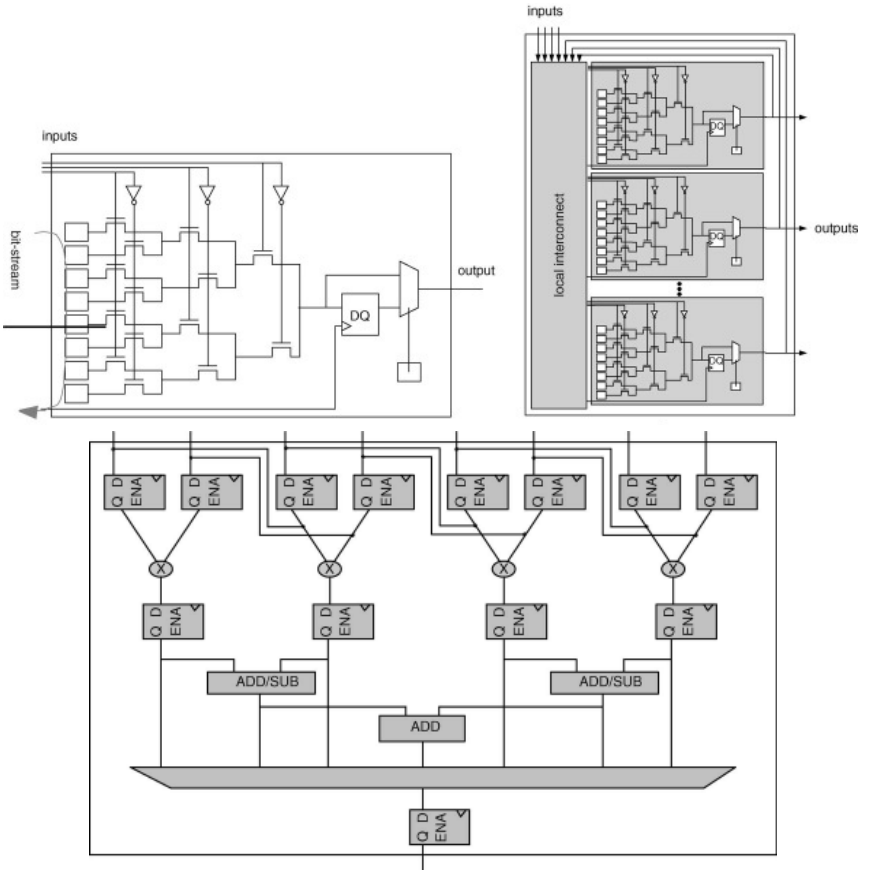

Figure 2.2 Coarse-grained versus fine-grained functional units, found in [Todman *et al.*, 2005]. A fine grained three-input look-up table is shown above left, a cluster with fine grained look-up tables above right. A Altera DSP block is, being a coarse-grained unit, shown below.

urable fabric as shown in Figure 2.4. 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 typically contains different kinds of functional units instead of one type, a heterogenous architecture can contain ALUs, multipliers, processor cores and of course distributed memory, all this to complement the basic functional units and to increase performance.

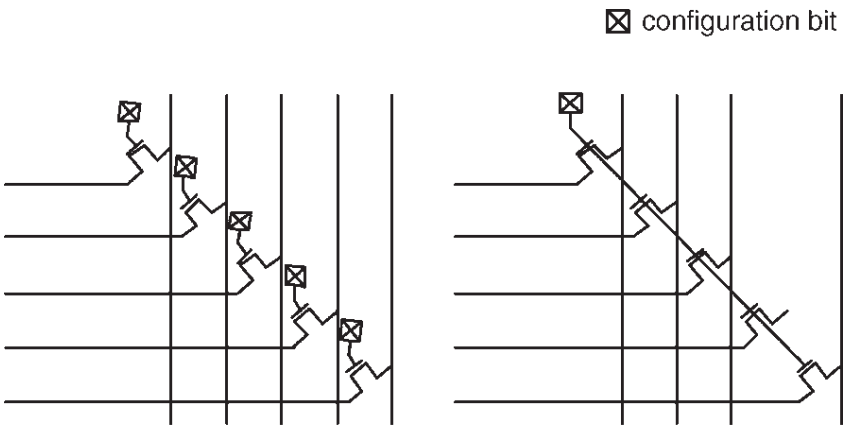

Figure 2.3 Coarse-grained versus fine-grained routing structures, also found in [Todman *et al.*, 2005]. In coarse-grained structures, shown right, a number of signal-lines are controlled as a unit. In a fine-grained structure on the other hand a smaller number or even a single signal is configured individually as shown left.

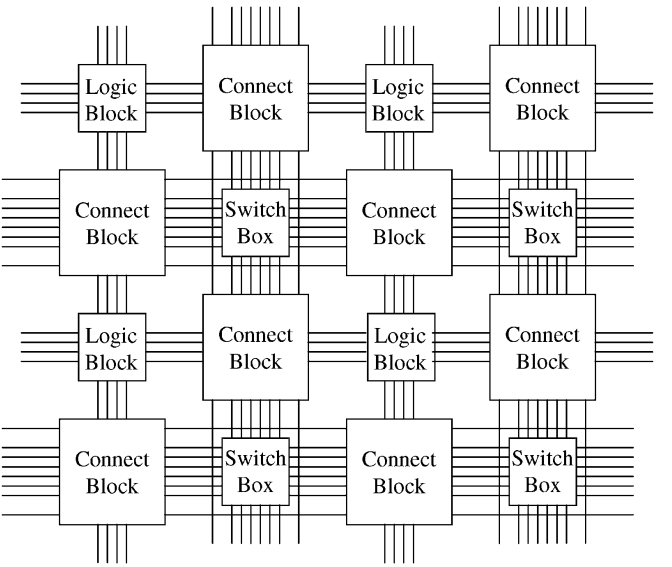

Figure 2.4 A typical generic reconfigurable fabric with switching units and functional-units or logic blocks. Figure found in [Compton and Hauck, 2002].

Design Level Architectures

Besides the architecture of the actual FPGA which is not so much a concern of the FPGA application designer, the architecture of the actual design is of great interest. It is of course possible to design a complete full hardware system residing only using the reconfigurable fabric of the FPGA (no type of processor core is present in the system). In many systems however some sort of processor core might well be present. Several different motivations exist for mixing processor(s) and FPGA(s) in a single system. The FPGA is not well suited for all types of tasks, iterative tasks with variable length, e.g. loops or control of dataflow, are difficult to implement efficiently in an FPGA. Such tasks are preferably implemented on a processor while computationally expensive, time critical and/or 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, Figure 2.5 shows the five methods, four of them identified by [Compton and Hauck, 2002] and the last one (number five in the figure) added by [Todman *et al.*, 2005]. Architecture one features communications between the processor and the reconfigurable fabric through the standard Input/Output (I/O) units of the processor. Architecture two and three show intermediate structures, meaning that the processor can access the reconfigurable fabric without having to use the standard I/O units. Architecture two is, communication-wise, a bit slower than Architecture three as architecture three features direct communication between the processor core and the reconfigurable fabric using a 'coprocessor' structure. The fourth architecture features a reconfigurable fabric which is present on the same chip as the processor, such an architecture does enable very high communication speed between the processor core and the reconfigurable fabric. Architecture number five resembles number four, but instead of adding 'a piece' of reconfigurable logic to a processor, a processor is added (implemented) on-to the reconfigurable fabric of an FPGA.

Using the last two architectures 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 made 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, it is possible to add custom instructions to the processor instruction set which is able to make custom calculations with only one call. This feature can in some applications give significant speedup as easily understood if the reader imagine a certain computational problem where a

number of arithmetic operations are required on the operands, and where the same calculation is carried out over and over again by the processor. If a normal processor is used the number of instructions needed is (in best case) equal to the number of arithmetic operations to be carried out, a customized processor on the other hand may be able to carry out all of the operations with a singe call and the speedup will hence be equal to the number of operations in this simplified example. 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 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 architecture is suitable for systems where large 'chunks' of work can be treated by the reconfigurable fabric independently. The reconfigurable unit of Architecture two has the same properties as a normal processor would have had in a multi processor system. Architecture three, the coprocessor 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 has. 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. A drawback with the soft instruction processor approach is however that it is more difficult to utilize the possibility and parallelism with such a tight coupling to the processor. According to Todman systems according to Architecture one are the most common in commercial FPGA platforms.

It should also be noted that it is not every FPGA system which features a processor core even though it would be possible to include one. It is for example possible to design larger applications where the complete functionality is implemented entirely on the reconfigurable fabric. Such designs can be used to verify the functionality of actual Application-Specific Integrated Circuits (ASICs) or in cases when ASIC like performance is needed but it is too expensive to make a real ASIC.

2.4 FPGA Design Tools and Methods

[Compton and Hauck, 2002] has written an excellent section about configuration of FPGA devices, Compton and Hauck explains the three design flows visible in Figure 2.6. Added to this [Todman *et al.*, 2005] covers more in detail a number of tools implementing the fully automated approach and describes some tools based on data flow graphs (e.g. Simulink). Selecting

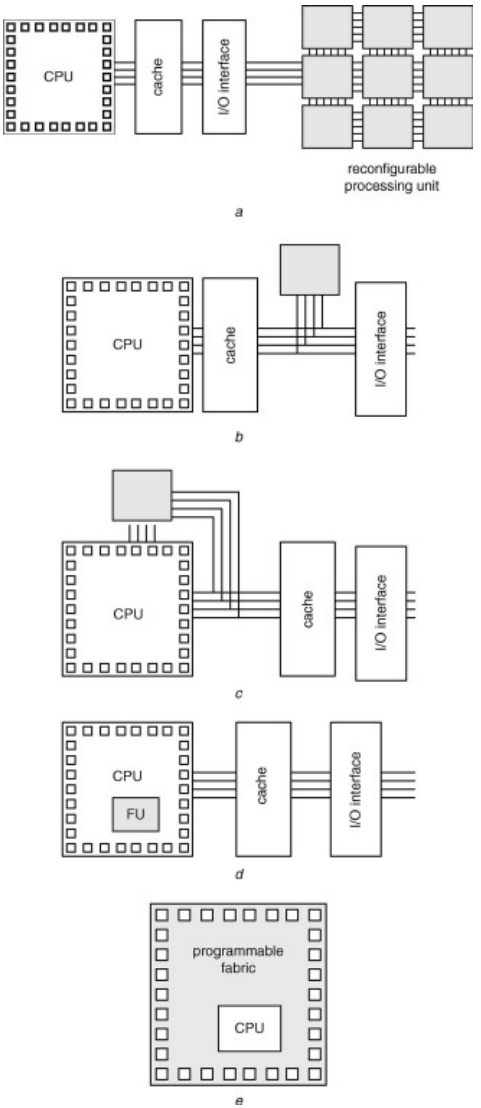

Figure 2.5 Five different architectures commonly used in mixed processor/hardware systems. The different architectures have different properties regarding for example communication-speed and flexibility and are hence suitable in different situations. Figure found in [Todman *et al.,* 2005].

a manageable and efficient FPGA design software is an important point which can ease the effort of implementation significantly. Which tool that is best suited depends largely on the type of implementation to be carried out, the skill of the hardware designer and the performance requirements of the final outcome. A fully manual approach (right in Figure 2.6) will, according to Compton and Hauck, give circuit-designs of very high quality, it however puts high demands on the developer regarding background knowledge about circuit design in general and also knowledge of the specific device in use. A fully automated approach on the other hand simplifies implementation greatly, the outcome however is often less efficient in terms of power consumption, speed and occupied chip area. It might however be worth its price to use a fully automated approach since it makes the FPGA technology available to a larger number of developers and since even a less optimal design still often has large gains compared to, for example a processor implementation. All these three design methods can be divided into separate steps as shown in Figure 2.6. These different steps will be described in the following sections. 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 for example an FPGA.

Basic Implementation Steps

Technology Mapping During technology mapping what basically is done is that the large design is broken down to small sub functions. Each sub function in turn is translated to actual logic on the FPGA device, meaning that each sub function is translated into the 'basic units' of the specific FPGA in use (e.g. look-up tables). The large design which is input for the technology mapping is described either at gate-level or at element-level, originating either directly from the designer if it is a fully manual design or from the design tool if it is some sort of automated design. This step is of course heavily device dependent since different devices contains different logic and are built according to different architectures.

Place & Route Subsequent to the technology mapping in the tool-chain is the place & route step. The place & route tool basically places the logic (assigns a location on the actual chip) corresponding to the different sub functions obtained from the technology mapping. A place & route tool needs to make difficult trade-offs carrying out the placement of the different sub functions, to do this successfully a technique called floor-planning is often applied before the actual detailed placement. The idea behind floor-planning is to analyse the sub functions and find which sub functions need

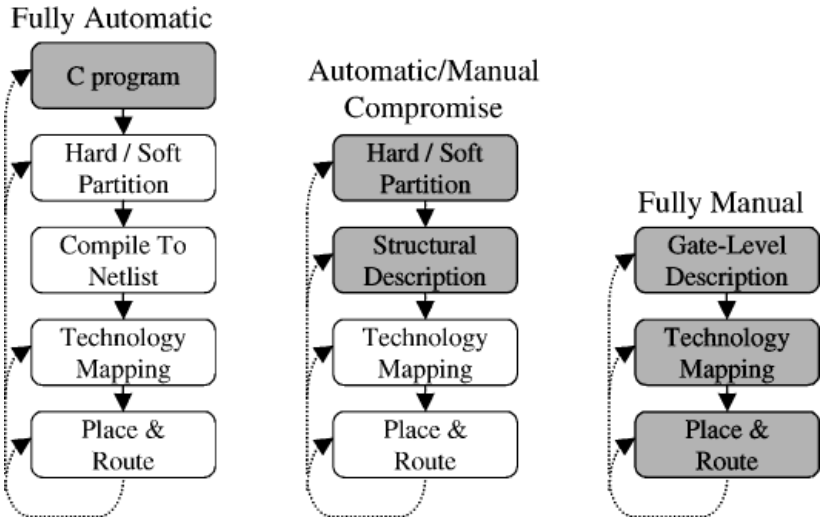

Figure 2.6 Three different FPGA design-flows, implementing an algorithm using different levels of automation. Different tools and implementation approaches demand different amounts of manual intervention by the designer/user, grey denotes manual efforts of some sort in the corresponding step. Figure found in [Compton and Hauck, 2002].

to communicate frequently with each other. Such sub functions are grouped together in clusters enabling a higher intercommunication rate within these clusters. When floor-planning is finished (deciding the global placement) a more detailed placement is carried out by the placement tool. In the last step the different components positioned in different parts of the FPGA are interconnected by the routing tool. As is the case with placement and floor-planning, routing is a very difficult trade-off. Maximum FPGA clock-speed is decided by the longest signal path within the design (since all signals need to arrive within one clock cycle). It is not only important to the keep the longest route within the FPGA as short as possible. Routing resources (signal lines within the FPGA) are limited and care must be taken to optimize the usage of this resource. On top of it all, to be able to route a design in a good manner, it is essential that the placement tool has made a reasonably good placement.

Low Level Design

The 'lowest' level of circuit design would be the fully manual approach, meaning that the designer has to decide which of the FPGA internal components to use and how to connect them with each other, producing a design at element-level. Another option is that the designer makes a net-list containing components at gate-level which describes the circuit functionality. As earlier mentioned the fully manual method gives high performance designs, it should however only be used for smaller designs and only by designers with high skill and good knowledge about the actual FPGA device in use.

Next 'level' of low level design would be the usage of a structural design language such as VHSIC Hardware Design Language (VHDL), (VHSIC = Very-High-Speed Integrated Circuit) or the Verilog hardware descriptional language. 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]. These languages are used to describe a hardware design from building-blocks such as gates, flip-flops or other similar components. When the VHDL or Verilog code is compiled by e.g. an VHDL compiler the structural description in VHDL is translated to a gate or element-level design, technology mapping and placed & route is then carried out based on the compiled VHDL code. VHDL or Verilog can be programmed 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 design 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 draw the design using 'wires' and different building blocks, either defined by personal VHDL code or defined in advance by libraries, such blocks could be gates or flip-flops.

Both the fully manual design method and design using a structural design language puts demands of significant knowledge on the designer, regarding hardware design and regarding the actual FPGA device. A programming environment consisting of an editor and compiler for structural design languages in combination with some graphical interface, probably bundled into a large toolbox together with the technology mapping and place & route tools and normally supplied by the FPGA vendor, would as understood by the author by far be the most common FPGA design option.

High Level Design Tools

High level design tools are here used as a term for the two leftmost design flows in Figure 2.6, the fully automatic approach and the semi automatic

approach. A fully automated approach is, compared to the fully manual design method, much more convenient to use and even not so experienced engineers can complete an FPGA design using one of the tools implementing a fully automatic approach. The drawback is as noted previously that the designs may be less efficient than a design which is made completely manually. Semi automated approaches are, simply put, any mix between fully automated and fully manual designs, an example could be where smaller interface or control parts are made manually while a larger or more complex algorithmic part is designed using an automated tool.

Hardware Compilers high level design tools typically generate an FPGA design based on a high level programming language. [Todman *et al.*, 2005] shows examples using C, C++ or Java as 'host' languages. These tools work, as identified by Todman et al., 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 constrains 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 review is needed regardless of if the code is intended for the hardware or for a micro processor. Annotation-free compilers have the great benefit that code can be 'moved' from an processor core to a 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.

Hardware/Software Partitioning systems where both reconfigurable hardware and a microprocessor are present and both components are to be used together the intended application must in some way be partitioned between hardware and software. For this purpose there are different possibilities, the system can either be manually partitioned. Each part is then developed using corresponding tools, if a hardware compiler is used for the hardware part either of the three types of hardware compilers could be used. Selecting an annotation and constraint driven or an annotation-free compiler would however give the largest flexibility to move code between the hardware and microprocessor. A manual approach will in any case not be very flexible regarding moving source code between the different parts. To solve this, compilers which co-target mixed micro processor/hardware systems have

been developed and examples are given both in Todman et al. and [Compton and Hauck, 2002]. These compilers often use the possibility of co-design and automatic partitioning, making partitioning and hardware design more 'transparent' for the developer (who probably is more familiar with the micro processor environment). Using for example annotation-free compilers the developer can either choose to let the compiler decide the partitioning, or the developer can decide how to partition the code between the microprocessor and the hardware.

It should also be said that hardware/software partitioning is tightly coupled both to parallelization and to the hardware compiler in use. Hardware compilers in some cases co-optimize or co-decide both the parallelization and the hardware software partitioning. Many parallelization tools by nature also implicitly decide the hardware/software partitioning during the process of deciding which parts of the code is suitable to pipeline and make parallel for putting in hardware, in order to off-load the microprocessor.

Parallelization To fully utilize the potential performance gains from using reconfigurable hardware, parallel and pipelined structures should be used as much as possible in the design. It is important to note that parallelization in this context is an issue connected with hardware/software partitioning and hardware compilers, not a completely independent step. Parallelization of 'high-level' code is of course not an issue for the manual design method since it is included explicitly in the design since structure and timing of the circuit is totally up to the hardware designer (who of course should try to use parallel and pipelined structures as much as possible even in the manual design). Using some sort of hardware compiler it is however more difficult to extract parallel and pipelined parts from the (by nature) sequential source code in C, C++ or Java. And again there is basically the two possibilities of handling this manually or automatic. Several approaches exists for automatic extraction of parallel and pipelined structures and again Todman et al. and Compton and Hauck describes the most common ones. Automatic parallelization and pipelineing of inner loops, parallelization of common instruction paths (rarer paths are run on a processor) or parallelization of all loops through control flow diagrams of the complete source code are a few worth mentioning. As for fully manual parallelization the programmer typically would have to define parallel areas within some part of the source code, either by annotations or by using parallel threads similar to 'normal' parallel programming.

Data Flow Graph Tools 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, they are of course suitable for other design purpose tools 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 already is 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 toolchain adapted for use in 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.

Using FPGAs for DSPs or other similar technical computing applications has a lot of nice features regarding for example speed, jitter and accuracy. One distinct feature with DSP like implementations in FPGAs is the non-fixed word-length of the internal number representation, something that can be a strength if handled correctly and a barrier of implementation if not. A DSP is normally a processor based device and has hence fixed internal word-length as processors do. When implementing DSP-like applications in an FPGA however this limitation of the internal number representation no longer exist It is possible for the designer to decide how many bits to use for internal number representation in each step of the algorithm. There is an optimal internal number representation for each step of a given algorithm, this optimal representation will give the system best accuracy and noise reduction, even more accuracy than possible with a fixed word-length DSP. Those word-lengths can be found through-out the design, it is however difficult to find them, this is in fact an NP-hard optimisation problem. [Cantin *et al.*, 2002] has written a survey of ways to automate the search for the optimal internal number representation and Todman et al. also discuss this subject, not all DFG based FPGA development tools support automated approaches on the word-length optimization problem. For example the Simulink based tool for Xilinx devices does not, as stated by Todman et al. and as experienced by the author and described in Chapter 4. Algorithmic settings must in the Xilinx case be selected by the designer in every calculation step, making work in this environment much more difficult for the designer. It is strongly recommended that a tool with automated word-length optimization is used since it makes implementation significantly easier and more efficient, especially if the designer is less experienced with working in the FPGA domain.

2.5 A Flavor of Application

Since FPGAs have been around for quite some time there exists a variety of scientific results on FPGA applications and design. This brief section is intended as a flavor (not a complete description) of applications and areas where reconfigurable computing technology has been applied successfully, in order to highlight the potential power of reconfigurable hardware. Enhancing feedback control system performance is, being the topic of this thesis, devoted a complete chapter (Chapter 3) and is hence not accounted for in this section.

A large application area for these systems is applications treating images or image streams (video). A good reason for using reconfigurable hardware in this kind of applications is that they are extremely computationally expensive. Heavy computations combined with the fact that many tasks have to be performed on-line puts very high demands on computational power of such systems. Examples of results in this area are [Djemal *et al.*, 2005], [Jörg *et al.*, 2004] and [Tao *et al.*, 2005] who all have implemented different FPGA or mixed FPGA processor systems in the domain.

Another very interesting application area is FPGA based super computers, meaning computers consisting of one or many FPGAs connected together in a matrix. The design within the FPGAs is totally customized to one specific calculation or simulation task and the FPGAs are able to, in a very rapid manner, solve the special problem which they are designed for. The 'secret' is of course heavy parallelization and pipelineing achieving outstanding performance and even price/performance. Very interesting results are published by among others [Jones *et al.*, 2006] and [Belletti *et al.*, 2006]. Such computers clusters are emerging as a serious competition to traditional processor based super computing clusters. The possibility also exists to make systems with one or more FPGAs to accelerate frequent or intensive tasks in a normal PC.

Different types of DSP applications are also common. Implementing DSPs and DSP-like systems in FPGAs are a very promising idea which is covered in an survey by [Tessier and W., 2000].

2.6 Chapter Summary

This chapter has covered the FPGA and reconfigurable hardware technology. Strengths, issues, design considerations and design tools have been discussed. The chapter started with a brief history about the FPGA which was invented by Ross Freeman in the mid 1980s and originating from similar devices such as the CPLD and PAL. An FPGA is a reconfigurable hardware device and works in the same manner as any digital/electric circuit. Com-

paring it to a processor it is found that a processor is a more general device since it can carry out a set of general instructions, an FPGA can only cope with the task it is specially designed for, it does on the other hand outperform the processor technology on that specific task.

There are two different 'levels' which are significant during FPGA design. One is the hardware-level meaning the architecture of the actual hardware of the device. Two significantly different architectures are distinguishable, a fine-grained architecture and a coarse-grained architecture. A fine-grained architecture typically consist of a large number of look-up tables which can be configured to carry out an arbitrary task, the routing within a fine grained architecture can typically be controlled with a large precision. A coarse-grained architecture on the other hand typically consist of more high level blocks which are hence not as general as the fine-grained look-up tables, the control of the routing is also less precise. The benefit with a coarse grained architecture is that it will perform better in speed, power consumption and chip area when it fits the intended application. Architecture on the design level have been covered as well and systems consisting of both a processor core and reconfigurable fabric has been shown. It has been described in which situations different interconnections between the processor core and reconfigurable fabric are suitable.

FPGA design tools and methods have been thoroughly described. Low level design tool steps consisting of the technology mapping and the place & route step and low level design is carried out with the help of structural design languages. It was found that a design at this low level gave the fastest and most efficient circuits but is extremely demanding and only suitable for highly skilled designers and smaller designs with very high performance requirements. High level design tools were described as well, high level tools often consist of hardware compilers which generate FPGA designs from programs in the well known C, C++ or Java languages. These languages are suitable to use for normally skilled designers and in systems which are mixed FPGA/processor systems since code easily can be migrated between the different subsystems, or the different subsystems can be co-targeted. Hardware compilers should be used for larger designs when speed, area and power consumption requirements are less strict. Using hardware compilers and mixed systems, trade-offs regarding hardware software/partitioning and parallelization arises (parallelization is an important factor for low level designs as well). Systems can either be manually partitioned between the hardware and the software or partitioned by the hardware compiler, if the compiler is able to co target both the processor and the hardware. Parallelization can also be carried out either manually or automatically, different approaches exist for automatic partitioning, for example hardware acceleration (parallelization and pipelineing) of inner loops within the program. Data Flow Graph based tools (e.g. Simulink) and FPGA design using these

tools were presented. The special issue of internal word-length optimization associated with algorithms within FPGAs was also mentioned, having the potential benefit of a solution with better performance than a fixed word-length algorithm. The optimal solution to the word-length problem can be found automatically but it is computationally intensive. The author strongly recommended an automatic approach since the word-length optimization problem is difficult to solve manually. Last but not least some success stories were presented.

3

FPGAs in Feedback Control Applications

3.1 Introduction

Very promising results are shown applying FPGAs in the field of automatic control, as a platform for implementing automatic-control systems. Results are emerging dealing specifically with implementation of different controllers using FPGAs as well as general ideas regarding implementation of control systems using FPGAs, [Monmasson and Cirstea, 2007] deals with design of industrial control systems using FPGA techniques and also very briefly compares it with control implementation using micro-controllers. Furthermore extensive work has been carried out by many different authors addressing the task of implementing well-known control algorithms in the new FPGA environment, evaluating methods, gains and drawbacks in doing so. An FPGA has properties which in many ways makes it the ideal component for control system implementation, some of them are mentioned in Monmasson and Cirstea. Three main reasons can be identified as particular strengths of FPGAs for feedback-control implementation;

- Increased controller performance, speed and robustness
- Reduced controller price due to improved price-performance ratio
- Reduced system power consumption and space requirement (crucial in embedded system applications)

The main benefit using FPGAs is the enormous increase in speed compared to conventional techniques, Figure 3.1, from Monmasson and Cirstea, illustrates a typical situation using FPGA for control compared to either DSP

or normal micro-controllers. The figure illustrates a situation where a micro-controller is fully utilized, meaning that there is no 'spare' computation time available, with the periodic task of executing and updating a feedback control algorithm. The drawbacks with the 'full utilization' are reduced margins for instability caused either by insufficient controller bandwidth, influence of jitter or both. Added to this there is no possibility to increase the complexity of the control algorithm. Such a scenario is likely even using a state-of-the-art micro-controller due to the generic structure of micro-controllers (processors), performing good in average but bad on the specific problem. The case for a DSP performing the same task would be slightly better, of course, since the DSP architecture is better suited, adapted, to this kind of task. While using an FPGA the computation time of the feedback control algorithm typically would be less than or equal to the time needed to perform the actual sampling. The time 'margin' gained from using FPGA-technology could be used in different ways, increasing controller bandwidth, gaining stability margin could be one. It would be possible to 'over sample' the system, in some cases increasing controller performance. Many control problems suffer from lacking ability of the implementation platform. Controllers and control-oriented models have to be simplified, suffering in performance. FPGAs, to a large extent, increase the abilities of the implementation platform, at least from a calculation speed point of view. More complex and computationally intense control strategies can be used, with better accuracy. The spare time can also be utilized for other calculations, for example other control loops residing in the same system. FPGAs would, compared to micro-controllers and DSPs and depending on the properties of the algorithm to implement, offer performance only possible to match using fully analog control systems. At the same time the problems associated with analog control systems such as component ageing, related parameter drift and the subsequent risk of controller instability are avoided due to the digital nature of an FPGA. An interesting note to make is that, as an alternative to 'traditional' analog control systems, a control system could be implemented using a device called Field Programmable Analogue Array (FPAA). FPAAs are related to FPGAs and are best described as the analogue counterpart of FPGAs. These devices are not covered by this dissertation but for example [Hall *et al.*, 2005] gives an insight in the world of FPAAs.

When it comes to system price there are a few sub-reasons which promotes an FPGA implemented approach over an micro-controller one. The price over performance ratio of an FPGA is superior to that of an micro-controller. This is easily understood from the fact that there, in a micro-controller, are a lot of 'go along' hardware not needed to solve the task but included in the device, thus increasing the price of it. Using an FPGA it is possible to only include the logic actually needed, hence a smaller device can be used. Adding to the price difference is the general and sequential

nature of a micro-controller which can not compete in performance compared to the parallel and customizable nature of an FPGA, drastically increasing performance as earlier thoroughly discussed. Hence a less powerful and cheaper device is needed. Related to the flexible internal nature of an FPGA it is, in larger systems, possible to include the exact mix of hardware needed on a single FPGA, even such components which typically would reside externally. Such an approach is called a System-on-a-Chip (SoC) solution and removes the need for much differing hardware greatly simplifying hardware-integration efforts, decreasing component cost and decreasing implementation time, 'time-to-market', all improving the economical performance. Monmasson and Cirstea also argues another reason giving shorter 'time-to-market' using FPGA devices compared to micro-controller and DSP devices; Significant effort is needed to hand-tune and optimize algorithms residing in DSPs in order to obtain the calculation speeds needed. Normal micro-controllers on the other hand need a lot of real-time considerations and probably a real-time operating system in order to implement feedback controllers. Although these remarks are true, they are weak as pro-FPGA arguments. Implementation, especially of control-algorithms, in an FPGA environment is far from straight-forward. Difficult considerations regarding, among other things, parallelization and number-format issues have to be made, probably increasing implementation effort to the same range as micro-controller implementations. Some of these considerations are noted previously and some are highlighted in this chapter. Concluding the discussion; economical performance of FPGAs exceed microcontrollers for matching performance, 'time-to-market' may gain somewhat from using FPGAs but is not the main strength of FPGAs.

Last but not least it should be said that FPGA technology potentially saves both space and power. The reduced power-consumption and space requirement are both in some sense related with the SoC approach, making it possible to include much of the external components on the FPGA. And again, the customized nature of the device, decreasing the device performance requirement adds to the benefit in power consumption.

3.2 Implementation of Digital Controllers

Implementing control and signal-processing algorithms either using micro-controllers or using FPGAs takes a lot of practical considerations, different considerations for different implementation platforms. In the well known micro-controller-case algorithmical considerations would for example have to be made depending on the performance of the device, depending on if the device holds floating-point support or if a real-time operating system is present on the device. The literature with in the topic of implementation

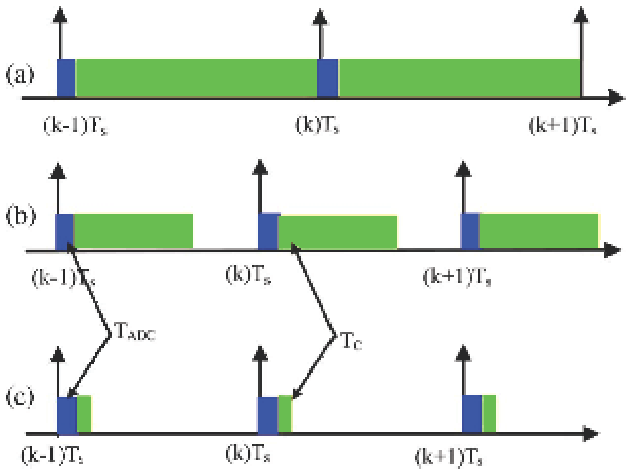

Figure 3.1 Probable time consumption of a control algorithm running on a micro-controller, top, a DSP, middle, and an FPGA, bottom. Blue denotes time needed to perform one sample and green denotes time needed to perform computations. It is noticeable how the microcontroller typically needs all the time available between two samples to compute the controller (this would of course depend on design considerations). The DSP would probably use less time to compute the controller compared to the micro-controller, leaving more margin for stability. The FPGA on the other hand would in most cases not need more time to compute the controller than it takes to sample the system. Gained time can for example be used to increase system stability in some way. The figure was originally found in [Monmasson and Cirstea, 2007].

of control/feedback-control algorithms using conventional computers (e.g. micro-processors and DSPs) is quite extensive, [Åström and Wittenmark, 1997] is an excellent reference piece on the topic. Besides the task of performing the discretization of the controller algorithms in a manner which assures maintained properties of the algorithm many other implementation specific considerations has to be made. There are, in order to successfully implement a digital controller, essentially a union of four different parameters which needs to be balanced/full-filled.

- System frequency and controller frequency (bandwidth)
- Control-system sampling-speed
- Accuracy of controller parameters (word-length and number representation)
- Parameter conditioning

Starting from the top it is important that the bandwidth of the controller is high enough to guarantee stability in the closed-loop system. The sampling speed must be selected so that the desired controller bandwidth can be maintained (in the range of 10-30 times the bandwidth of the closed-loop system). The limit of controller sampling speed is in many cases the calculation capacity of the device used for implementation. If the sampling-speed is much higher than the bandwidth of the controller, special considerations have to be made to avoid for example noise issues. Maintaining controller stability when using limited-word-length arithmetics (which always is done in the real-world case) is another issue which needs to be handled to guarantee performance of the closed-loop system. Another thing to keep in mind implementing digital controllers is that it is important to use well-conditioned parameters, avoiding excessively large or small parameters especially in combination with 'normal' sized ones. There are mainly two reasons for avoiding ill-conditioned parameters, over/under-flow in the algorithms and noise sensitivity.

Algorithmic Considerations for FPGA Implementation

Some of the considerations needed when implementing digital control using micro-processors and DSPs are shared by the corresponding FPGA-controller implementation and some considerations and trade-offs are different. Handling of dynamic fixed-point number format, removing 'troublesome' operators and extracting parallelism in the algorithms are suggestions given by Monmasson and Cirstea aiming for a more efficient implementation of control systems using FPGAs. These points sum up in two main categories, 'Word-length Optimization and Internal Number representation' and 'Parallelization and Algorithm Reformulation'. The (in some sense) limited word length and above all the very high sampling-speeds reachable using FPGAs raises one additional issue when implementing for example control systems. The issue is how to handle a system sampling speed which is significantly higher than the bandwidth of the sampled signals/systems, a topic which is often forgotten describing this type of applications. To remedy this situation the, within digital-control and digital signal processing, commonly used z-transform can be replaced with a transform called the δ-transform. Using the δ-transform and the associated δ-operator gives numerical advantages reducing controller sensitivity to round-off effects and quantization issues of controller parameters. The δ-transform also limits the issues connected with the z-transform at high sample-speeds and especially at very high over-sampling speeds. This is the last (but not least important) consideration implementing high-bandwidth control systems using FPGAs, accounted for in this section.

3.3 Word-length Optimization, Internal Number Representation and Parameter Conditioning

The internal-number representation and optimization issue are important whether using a fixed-point implementation or using algorithms implemented with floating-point number representation. Floating-point numbers are convenient to use within algorithms, however a floating-point number representation is in many ways less efficient than the corresponding fixed-point number representation, using floating-point numbers some algorithmical operations for example needs more calculation steps. This can be said both from a calculation speed point of view and from a chip-area point of view and, both using microcontrollers and especially using FPGAs, implementation would preferably be made using fixed-point arithmetics to save chip-area, power and to gain speed.

Quantization effects implementing controllers in micro-controller and DSP environments are well known, [Åström and Wittenmark, 1997] deal with implementation of control algorithms accounting for quantization effects and the impact of limited precision in number representation on controller performance. Åström and Wittenmark also show a way to model the effects of quantisation using 'linear analysis' adding noise sources either of known or stochastic nature depending on which effect to model. These disturbances are then handled as any other disturbance in the system and the controller can be designed to cope with the quantization effects.

As mentioned in 2.4 the designer has, when implementing algorithms in FPGAs, the possibility to select the internal number representation within each step of the calculation (almost) arbitrarily. Before designing the controller so that it can handle the ever present round-off and quantization issues according to above it is important to make an effort in finding the best internal data-path in order to minimize these effects, maximizing possible controller performance. Selecting the optimal word-length within each step of the calculation is, as briefly discussed in 2.4 a hard optimization problem which is subject to research efforts. Excessive word-length consumes chip-area and power when using FPGAs, using DSPs excessive word-length decrease the performance of the algorithm and consumes power. [Cantin *et al.*, 2002] state that in complex DSP designs, as much as 50% of the design time is spent on deciding the internal number format in different steps of the algorithm, probably similar values hold for complex FPGA algorithm designs since the problem essentially is the same. It is obvious that it is desirable to select the minimum word-length possible, still guaranteeing controller/algorithm performance and it is also obvious that it is desirable to automate the time-consuming and error-prone task to decide the internal number representation of an algorithm. Methods to perform such an opti-

mization automatically are for example shown in Cantin et al. and Cantin et al., although mainly intended for the DSP environment these results can be translated to FPGA based applications. Normally the integer part word-length of the fixed-point representation is decided based on the dynamic range needed, to decide the fractional word-length however takes more consideration. Three different approaches can be used to select the fractional word-length, the fractional word-length 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. Cantin et al. used word-length determination based on simulations. An algorithm was first simulated using floating-point number-representation. Fractional word-length was then decided based on a minimization procedure, a fixed-point simulation was run and the outcome evaluated using a error function. The result from the evaluation function was used by the minimization procedure, updating the decision on which fractional-length to use, nine different minimization procedures were tested. The process of deciding the fractional word-length, performing the fixed-point simulation and evaluation of the outcome was iterated until system specifications were met. The criterion for stopping the iteration could either be expressed in an error function value or as a limit on the difference between the fixed-point and floating-point simulation. Cantin et al. evaluates these heuristics on twelve different algorithms commonly used for digital signal processing. For the purpose nine different minimization procedures were used. Being a simulation and evaluation-function based method the suggested heuristics shares one drawback with other cost-function based methods, the sensitivity for a well formulated cost/minimization-function. Cost-function based heuristics (for example neural-networks or genetic algorithms) are generally not able to perform better optimizations than the evaluation function enables them to do. Other drawbacks with the simulation based methodology as described later by Cantin et al. is that this method does not guarantee that no overflow will occur or that the specification will be full filled for any other case than the one reflected by the 'test-bench' used. The reason is that the test-bench in many cases can not reflect all possible inputs to the system encountered during the complete lifetime. Performing simulations using a test-bench covering 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 possible overflow. It is possible to increase the constraint more than necessary to gain some 'margin' for overflow. And one way could be to perform 'cascade

simulation', when a possible data-path is found using a less extensive test-bench an as complete test-bench as possible is used to verify this data-path for a large part of its 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. Using these extensions Cantin et al. claims that the proposed method produces rapid and accurate solutions to the internal word-length optimization problem.

The last issue to be discussed here is parameter conditioning, i.e. how to avoid ill-conditioned parameters. The conditioning and scaling of algorithm internal parameters can affect noise and overflow sensitivity. Lets illustrate this phenomenon with an example. If we have an algorithm like Equation 3.1 below, u is the input and y is the output. Equation 3.2 shows the typical discretization of Equation 3.1. The mathematical notation is somewhat sloppy. Studying Equation 3.2 one realizes that $\frac{B}{\Delta}$ increases as the sampling rate increases ($\frac{B}{\Delta} \to \infty$ as $\Delta \to 0^+$), which raises two related and severe issues. One problem is that $\frac{B}{\Delta}$ sooner or later will grow to such numbers so that it can not be represented with the word length available. At which sampling rate this happens does of course depend on which word length is used. The second issue is caused by the fact that $\frac{B}{\Delta}$ is now far larger than A. Consequently noise present in the right term of Equation 3.2 will have significant impact, risking computational overflow in the complete equation and opening a 'noise leak' into the overall result of the equation. Signal-to-noise-ratio of a discrete-time derivative decreases with increasing sample rate. An intuitive explanation for the phenomenon is that the signal of interest does not change enough between two samples of u, hence the discrete-time *difference* is small. At high sampling rates $\frac{B}{\Delta}$ consequently has to be large to maintain the original relations of the equation However due to the large amplification of the second term the noise of the difference ($(1 - q^{-1})u$) will be amplified far more than the noise present at u, making the equation useless at high sample rate due to excessive noise. Ill-conditioned parameters can occur, and cause noise in many situations and parameters can be ill-conditioned even though they are analytically correct.

$$y = Au + Bu', A = B \tag{3.1}$$

$$u = Au + \frac{B}{\Delta}(1 - q^{-1})u \tag{3.2}$$

3.4 Over-Sampling and Limited Precision, Digital Control using the δ-transform

Implementing digital controllers does, as the reader for sure understands by now, take a lot of consideration and [Åström and Wittenmark, 1997] does, as previously noted, thoroughly describe the process of implementing digital controllers. There is however one aspect not mentioned in Åström and Wittenmark which is getting more and more relevant with the development of computers, FPGAs and other high performance devices used for automatic control. Implementing digital controllers on high definition micro-controllers and FPGAs, as previously noted, enables bandwidths not dreamed of just ten years ago and consequently the conventional way of discretizing controllers is not developed with very high controller bandwidths in mind. Since implementation of high-bandwidth controllers is the scope of this publication it is important to investigate which effects this higher bandwidth has on the discretization process of controllers. Can the conventional implementation methodologies using z-transform or shift operators always be used? Experiences made by the author during the development of the FPGA-implemented heat-release analysis shown in Chapter 4, and the literature in the field, for example [Middleton and Goodwin, 1986], [Goodwin *et al.*, 1992] and [Goodall and Donoghue, 1993] indicates that so is not the case. Goodwin et al., for example, state that; "Most traditional digital signal processing and control algorithms are inherently ill-conditioned when applied in situations in which data are taken at sampling rates that are high relative to the dynamics of the underlying continuous-time-process being sampled." the previous section touches this subject when discussing ill-conditioned parameters and related issues, making an example with Equation 3.1 and Equation 3.2. Traditionally the option of 'over-sampling' a system did often not exist due to limitations of the hardware used for controller implementation. New hardware removes this boundary and it is possible to over-sample even systems which have very high-frequency dynamics. Since over-sampling gives rise to a host of additional problems a logical question is; "why over-sample?". Goodwin et al. give an answer to that question; "Low speed methods are usually associated with a performance penalty, but have been necessitated by technological limitations. Recent advances have made these limitations less important". It is important to remember that closed-loop control systems are, by necessity, open-loop between samples. Effects as inter-sample ripple (see Åström and Wittenmark p111) sometimes causing mechanical oscillations and a reduced stability-margin are typical consequences if sampling-frequency is selected in the low-end of what is possible from a control perspective. If such a system is 'over-sampled' precision and robustness are improved. High-bandwidth communication sys-

tems is another example of an application where over-sampling occur. The format of these systems (the underlying information-carrying frequency) in some cases enforces high over-sampling rates of control-loops. In the case of the high-speed heat-release analysis described in Chapter 4 issues with synchronisation, differing and non-constant time bases made over-sampling necessary.

What is then a high sampling speed, what is the definition of 'over-sampling' in these contexts? The commonly used definition of over-sampling is when sampling at a speed significantly higher than the sampling speed decided by the Nyquist criterion. If a system is sampled at a rate, $f_s = \beta \times f_N$ where f_s is the sample frequency and f_N is the Nyquist frequency, the common notation would be that the system is over-sampled by the factor β. A computer-controlled-systems perspective however demands a higher sample-frequency than the Nyquist-criterion indicates in order to guarantee performance. A rule of thumb given in Åström and Wittenmark p110 states that a feedback-control-system shall use a sample-speed between 10 and 30 times the bandwidth of the open-loop system. This combined with the fact that Goodall and Donoghue introduces the rule-of-thumb that, when the sampling frequency is ten times the bandwidth of the system (or in this case the cut-off frequency of a filter) or higher, special considerations should be made, makes it reasonable to state that a system which is sampled at a rate more than ten times the bandwidth of the open-loop system is over-sampled. Hence, if $f_s > 10 \times f_{-3dB}$, the system is here said to be over-sampled from the perspective of digital-control.

What are then these considerations which have to be made using over-sampling in a digital control system? Middleton and Goodwin, Goodwin et al. and Goodall and Donoghue all suggess the usage of the so-called δ-operator and the corresponding δ-transform instead of using the normal shift-operator, q, and the z-transform. Getting back to the previous example, a implementation of Equation 3.1 using the δ-operator rather than the shift-operator (as in Equation 3.2) is shown in Equation 3.3. Comparing Equation 3.2 and Equation 3.3 one realizes that the explicit dependence between parameter scaling and sampling rate is now accounted for implicitly by the δ-operator. Both A and B now maintain their original values solving the previously highlighted parameter conditioning issues.

$$y = Au + B\delta u \tag{3.3}$$

Goodwin et al. shows and motivates a 'unified calculus' applicable in automatic-control based on the δ-transform/operator with the intention to develop the corresponding system theory. The authors deals with the topic of finding δ-transform representations for commonly used automatic-control techniques, showing that using the δ-transform gives advantages from a

number of different perspectives. Important theoretic and system-theoretic concepts such as integration, generalized matrix exponent, stability boundary, frequency response and state-space models are redefined using the δ-transform. For the interested reader these definitions, motivations and proofs can be found in Goodwin et al. and some of them as well in Middleton and Goodwin however these are not further discussed here. The definition of the δ-operator is however shown in Equation 3.4, the operator is as understood from the equation defined in the same way as an Euler-estimation of a time derivative. Goodwin et al. gives a number of reasons motivating this way of defining the δ-operator; flexibility, continuity and implementability. Flexibility meaning that it is desirable to have an operator equally flexible as the shift operator q. It is also desirable to be able to transform between a z-transform representation and a δ-transform representation and consequently it should be possible to change between q and δ which it according to Equation 3.5 is. Continuity in the sense that Equation 3.6 shall be fullfilled and implementability meaning that the operator shall be directly implementable in software (or for that matter, hardware).

$$\delta X(t) = \left\{ \begin{array}{ll} \frac{d}{dt}(\cdot): & \Delta = 0 \\ \frac{X(t+\Delta)-X(t)}{\Delta}: & \Delta \neq 0 \end{array} \right\} \tag{3.4}$$

$$q = 1 + \Delta\delta \tag{3.5}$$

$$\lim_{\Delta \to 0^+} \delta(\Delta) = \delta(0) \tag{3.6}$$

Which are then the benefits from using the δ-transform/operator? One obvious benefit is that δ converges towards a continuous time derivative, $\frac{d}{dt}$, as sampling-time, Δ, is decreased, as $\Delta \to 0$, enabling a 'smooth transition' between continuous and discrete time as δ is a discrete-time Euler-estimate of a derivative. The discrete-time sampled system hence, using the δ-transform, converges to the underlying continuous-time system as $\Delta \to 0$. Many issues related to highly over-sampled control-systems are solved through this fact. The higher the sample rate, the more 'like continuous-time' the corresponding discrete-time system representation will be. One easy way to explain the reason for this is that δ is a version of q which is 'weighted' against sampling-time and hence is equally large and precise regardless of sampling time.

Related to the fact that δ converges towards $\frac{d}{dt}$ is the fact that controller stability increases with sampling speed. Closed-loop pole assignment for example will, according to Goodwin et al., be better conditioned using δ. Using the shift-operator and z-transform the poles and zeros will converge to the (unstable) point $1 - 0j$ as $\Delta \to 0$. However using δ the poles and zeros

will converge towards their continuous-time counterparts as $\Delta \rightarrow 0$, maintaining (and probably increasing) controller stability margin as the sample rate increases. Using the z-transform the closed-loop poles would have to be within the unit-circle (with the exception of the negative real-axis to avoid ringing) to guarantee a stable closed-loop system. However, using the δ-operator, the closed-loop poles would have to be within a circle centered at $-\Delta^{-1}$ having a radius of Δ^{-1}. Slow sampling hence has the effect that it limits the stability region and as sampling-speed increases the stability region grows. The discrete-time stability domain approaches the continuous-time one, in which the poles would need to be in the left half plane to guarantee stability.

The original motivation for using δ was however not its superior characteristics at high sampling and over-sampling rates even though this has turned out to be maybe the most important and attractive feature for the future. δ was largely introduced because of the numerical benefits when using finite word length number representation. Using δ, in many cases, gives better numerical properties than using the common z-transform and shift-operator and the benefits of using δ increases with sample rate when the shift-operator degrades. The numerical benefits are thoroughly accounted for in the literature, for example Middleton and Goodwin and [Wu *et al.*, 2000] focuses on the numerical benefits of δ and they are also mentioned in Goodwin et al. As previously discussed word-length and numerics needs a lot of consideration implementing controllers using finite word length number representation. It is for example necessary to select the correct internal number-representation (integer and fractional bit lengths). Middleton and Goodwin show that using the δ-transform for the discretization improves both the problem with round-off noise and imperfections in the coefficient representation, whether using a fixed-point or floating-point representation of the controller internals. Consequently it is possible to decrease the word lengths used in the calculations compared to those needed for z-transform implementations, saving power and chip area while maintaining performance and reducing the discussed issues at high sample frequencies. The determination of optimal word length as previously discussed will hence be somewhat different using the δ-transform. Wu et al. shows that an optimal word length δ based controller performs better than the corresponding shift-operator based one while using less bits for number representation. In Wu et al. a measure for the performance of controllers, implemented using finite word length, is developed. This measure is used to find the optimal internal word lengths for the controller design. It can be used both for a δ representation and for a shift-based implementation. Wu et al. states that it is always possible to, given a word length, find a δ representation which outperforms the corresponding shift-based implementation.

Some results can be found in the literature indicating that the δ-operator

performs well in practise. For example Goodall and Donoghue have implemented a digital filter (low-pass) using the δ-transform/operator. Goodall and Donoghue state that when using the common z-transform and shift operator the numerical problems start as noted when sampling at ten times the filter cut-off frequency and they become severe when sampling at about 100 times the filter frequency. Using the δ-operator Goodall and Donoghue were however able to successfully implement a filter which samples at a rate 32000 times the cut-off frequency of the filter. [Chen *et al.*, 2000b] use the heuristics developed in Wu et al. and deploys it on a PID controller comparing a δ approach with a shift-operator approach for a span of different sampling rates. Chen et al. find that the δ implementation in most cases, as expected, needs less bits for the number representation maintaining the same performance and it is at the same time superior to the shift-based implementation at high sampling frequencies.

Which are then the drawbacks with the δ-transform/operator compared to the shift-operator. One obvious drawback is that a δ implementation is a bit more difficult to implement, the shift-operator is very simple to implement (just a delay of one sample and a subtraction). Consequently there is, as noted by Goodwin et al., a computational overhead coming from using the δ-operator instead of the shift operator. This overhead is by Goodwin et al. regarded as small compared to the total number of computations needed in these contexts. The major drawback of the δ approach would probably be the fact that the shift-operator is well established, commonly used in discrete time implementations. The δ-operator is not. Maybe the need for high frequency performance will renew the interest in the δ-transform in the future.

3.5 Parallelization and Algorithm Reformulation

It is, as the reader for sure has understood by now, the parallelization which makes the FPGA truly more powerful than a micro-controller. If parallel structures are not utilised the great gain in performance is not obtained either. This is important to keep in mind when developing algorithms considered for FPGA implementation, for control as well as for other purposes. As an example Equation 3.2 can, according to Figure 3.2, be implemented in two ways, sequentially as described by I or parallel as described by II. It is obvious even in this small example that the parallel implementation in II is much faster than the sequential one in I and even more so since multiplication in many cases is more demanding than addition. This effect gets more pronounced if the problem is larger and contains more parallel branches. Not all problems can be parallelized. If for example B would depend on $Au(t)$, $B = f(Au(t))$, then $B(u(t) - u(t-1))$ can not be calculated before

the calculation of $Au(t)$ is finished and the complete equation would have to be calculated sequentially reducing the maximum possible performance (Δ is here assumed to be one). It is important to exploit such parallelism when found in algorithms. Studying for example Figure 4.2, which describes the DFG implementation of the algorithm performing the high-bandwidth heat-release analysis described in Chapter 4, it is found that it contains two different calculation paths which are summed together before obtaining the final result (this is the case even though it is not completely obvious from the figure). The high-speed heat-release analysis hence uses a parallel structure, similar to the one described by the example, in order to obtain its high performance.

Parallelization is touched earlier in 2.4, more from an programming and FPGA design point of view. Using high-level FPGA design languages parallel structures will be extracted implicitly with the risk of missing structures existing on the algorithm level. The designer has to make sure that algorithm paths which are truly parallel are implemented in a way so that they end up as parallel paths on the FPGA. How to do this is difficult to state generally since it would depend greatly on which tool is used. It would for example come 'almost automatically' when implementing algorithms using Simulink since the DFG structure of Simulink promotes parallel calculation paths. Using a programming-language based tool, for example Handel-C or VHDL, it would probably require some more attention since the implementation format is more sequential in its nature as opposed to DFG based tools. In either case the designer should be aware of which parallel paths that exist in the algorithm and if it is desirable to implement them parallel or sequentially.

It is not in every case desirable to use the 'full-parallelism' available in an algorithm for a number of different reasons. Parallelization is in some cases a trade-off between speed, chip-area and power-consumption as indicated by for example [Zhao *et al.*, 2005], attempting PID control (which also is described subsequently in this chapter) using different degrees of parallelization of the algorithm. For a single-channel loop both a fully parallel design and a design which only uses one adder and one multiplier, performing all the calculations needed in a time-multiplexed manner is attempted. Using multiple-channels Zhao et al. attempts both channel-level serial design (all loops share one controller), multi-channel serial (all loops share one controller which in turn share only one adder and one multiplier) and a fully parallel design. The objective is of course to save area and power. Zhao et al. finds that due to the large over-head introduced by the control logics performing 'context-switching' parallel designs are to prefer when implementing a more moderate number of loops. Having a large number of loops however some sort of serial design would save both power and chip-area at the expense of speed, the results of Zhao et al. indicate the trade-off between

chip-area, power consumption and speed.

When performing parallelization of algorithms it is both important and reasonable to have a well balanced approach, heavy computations should be parallel with branches of a number of less demanding computations. Logically the over-all computation will not finish faster than the slowest branch does and it makes no sense to waste resources in order to finish a part of the computation quickly having to wait for the rest to finish. This leads to another angle of parallelization, 'racing', implementing parallelized algorithms in FPGAs (or ASICs for that matter) it is important to make the design 'race free'. The 'race' phenomenon arises when parallel branches compute two different sub-parts which both are needed in sub-sequent parts of the algorithm. If one of these branches is faster than the other the two sub-results will be available at different time instances, 'older' values will be combined with 'newer' ones in the sub-sequent computations. Implementing control-algorithms issues with racing tend to be less severe due to relatively high sample rates and a continuous behaviour of the sampled system, the values in the different paths of the calculation probably will be fairly similar. Racing should never the less be strictly avoided due to the risk of errors which are difficult to trace. Under the described circumstances some method of synchronization should be deployed, guaranteeing that output originating from the parallel branches are 'belonging'. Optimally these branches should also take equally long time to compute, not to waste resources implementing too many and un-balanced branches, remember that routing resources also occupy area! The lesson is hence that, to avoid issues with racing and not to waste resources, parallel branches within an algorithm should be synchronized and well-balanced.

Algorithm reformulation is another topic which is important in order to get an efficient controller implementation. [Monmasson and Cirstea, 2007] does, besides showing a calculation example, discuss this topic and mentions the COordinate Rotation Digital Computer (CORDIC) algorithm as an example. In order to implement a design as efficiently as possibly it is wise to try to simplify the algorithm, to remove operations which are chip-area and/or time consuming and to try to minimize the number of calculations made on-line. Not very different from implementations in an DSP/micro-controller environment in other words. Simplifications, assumptions and re-modeling of the algorithms are a good idea whether they are considered for a micro-controller or an FPGA implementation. CORDIC is one approach for simplifying among others trigonometric functions commonly used and ways to implement the CORDIC algorithm in FPGAs are suggested by [Andraka, 1998]. Which operations that are to be considered as difficult to implement, and hence candidates for removal/simplification depends greatly on the architecture of the actual FPGA devise used. Commonly, large adders or multipliers or similar logic are present in FPGAs (for example Alteras DSP

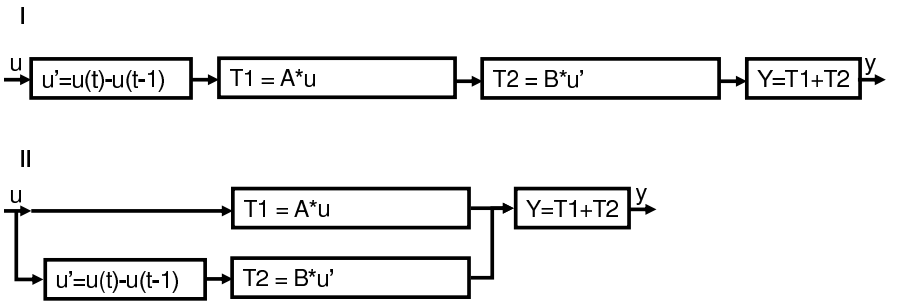

Figure 3.2 Flow diagrams describing two different possible ways to implement and parallelize Equation 3.2. Method I is the fully serial one, the one that would be run upon a processor. It is clearly the case that method II would compute faster given that the different operations takes equal time since II is implemented parallel.

block), hence for example a large multiplication does not have to be time and chip consuming by default.

3.6 A Flavor of Application

With the intention to illustrate the potential power of FPGAs in control applications and to highlight some interesting implementation attempts, this section briefly describes a few different examples of feedback-control algorithms implemented on FPGAs. Three different control algorithms, using different design considerations and methodology, implementation of PID (Proportional-Integral-Derivative) control, implementation of MPC and implementation of Neural Networks (NN) are presented. Typically controllers which needs a lot of on-line computational power and/or which are intended to be run at extra ordinary bandwidths are considered for FPGA implementation.

PID controllers

PID controllers and variants of them (P, PI, and PD controllers) are the most important controllers and it is the most commonly used control strategy. In control problems where the open-loop system is at least reasonably linear and where no feed-forward is needed PID controllers may often be good enough. The possibility of implementing PID controllers using FPGAs has been investigated for example by [Chen *et al.*, 2000a] and as earlier mentioned by [Zhao *et al.*, 2005]. The motivations for and gains with implementing PID controllers using FPGAs vary. Generally speaking since PID

controllers are not extremely computationally intensive the gain of using FPGAs for PID control would be largest when deploying multiple control-loops or loops with extreme bandwidth requirements, for example control within digital communication systems or controlling switched power supplies [Yousefzadeh *et al.*, 2006]. Depending on the architecture of the FPGA implementation the benefits from using FPGAs in the control loop vary as discussed earlier from the horizon of the work of Zhao et al. using different degrees of parallelization when implementing PID controllers and evaluating their corresponding properties. It is also worth to note that the motivation for using FPGAs in the by Zhao et al. undertaken work was constraints put on the system by the environment in terms of speed, space and power-consumption, implementing a multi-loop control system for a small-scale robot. There are cases where PID control using FPGAs could be useful even when bandwidth requirements are limited, for flexibility or to guarantee the real-time requirements could be examples of such cases. Chen et al. used an FPGA for implementation of PID control partly for its flexibility, using it as a 'rapid-prototype system', and partly because of the alternatives available probably would not be able to cope with the control task at hand.

Model Predictive Controllers

MPC is a method of feedback control increasingly used. MPC has previously been used mainly in the petrochemical industry and in similar applications (large systems with slow dynamics) for process control. MPC is well suited for this type of applications since it can handle constraints, it is at the same time computationally expensive. Lately however there is, as reported by [Maciejowski, 2002], an increasing interest to deploy MPC on processes with a higher bandwidth, the reason being that performance of processing devices increase rapidly and MPC can hence be applied in situations previously not possible. As indicated by its name an MPC controller contains a model of the plant. The model can be obtained either from physical/mathematical modeling efforts or through some sort of identification experiment. This model is used to iteratively predict the plant output over a limited time-horizon, predicting the state of the process and how the trajectory of the state depends on the controller output. Using this prediction the optimal controller output is determined with the help of a 'cost-function'. One reason for the popularity of the MPC control strategy is its ability to minimize the 'cost' taking constraints into account. Not surprisingly MPC controllers are typically implemented using processors, fairly high-power devices are needed which in the case of a petrochemical plant is no issue. In such a case it is possible to devote a high-performance computer for one control-loop. In other cases it is not practically feasible to use a high-performance processor, in const sensitive applications alternative implementation platforms would have to be used. Processor implemented

MPC-controllers have previously been shown effective for control of the HCCI engine [Bengtsson *et al.*, 2004] and MPC is hence of special interest in this thesis considering the intended application for FPGA-based feedback-control, the field of automotive control. It is reasonable to believe that Model Predictive Controllers (MPC) implemented on an FPGA platform is a 'winning concept'. MPC has properties making it ideal in many difficult control situations, at the same time the on-line computations needed are time-consuming, an issue possibly solved by the customized high-speed nature of FPGAs.

Results have been published showing MPC implementation in an FPGA environment, for example [Ling *et al.*, 2006] and [Bleris *et al.*, 2006]. MPCs can hence be applied in applications previously not possible due to bandwidth requirements, cost issues or system size limitations. Ling et al. presents an approach using Matlab/Simulink together with Handel-C (Celoxica DK design suite) for implementation and verification of an MPC controller. Translation between Matlab and Handel-C were performed manually and a Simulink 'test-bench' was used to test the MPC both using software simulations and using the FPGA implemented logics and hardware-in-the-loop. The authors used single-precision floating-point numbers for implementation. The result of the work in Ling et al. is mainly the successful implementation of an "reasonable sized" MPC controller on an FPGA, Ling et al. shows promising performance. The system implemented by Ling et al. is a full-hardware system (compare Chapter 2 and Figure 2.5), there is also the possibility to use a coprocessor or mixed-system approach, implementing custom hardware parts in the FPGA letting it perform time consuming computations while a processor core takes care of data-flow control and similar tasks. Bleris et al. uses this approach, implementing the bulk of the repetitive and demanding computations such as matrix calculations in the FPGA. In the case of Bleris et al. the hardware and software parts are co-designed using 'CodeWarrior', Matlab/Simulink and real-time workshop. Using a co-design methodology the complete spectrum between a full hardware and a full software implementation can be made easily depending on the needs. It should be noted that MPC, being an iterative method, probably is better suited for a mixed processor/hardware system than a full hardware one. If a mixed system is used flexibility may be gained without decreased efficiency. The main result of Bleris et al., as well as of Ling et al., is the successful implementation of MPCs in FPGA and mixed environments, showing that the possibility to employ MPC control at bandwidths not before possible now exists. These results open the field for MPC control of a large number of interesting and difficult high bandwidth applications and control problems.

Neural Controllers & Neural Networks

Another application considered interesting for implementation in FPGAs is NN or Artificial Neural Networks (ANN). NNs are a mathematical modeling methodology inspired by the central nervous system of humans. A NN consists of a number of simple processing elements interconnected in a network. The internal structure of NN is explained by Figure 3.3. Further detail regarding the algorithm can be found in [Russell and Norvig, 1995]. NNs have previously proved efficient on a number of different classification, modeling, data-processing and last but not least (and partly covering the mentioned topics) automatic control problems. [Hunt *et al.*, 1992] deals with automatic control using NNs. Using NNs has proven especially efficient when handling large and complex datasets or datasets based on observations, 'black-box models' and in cases where it is difficult or impractical to use 'normal' modeling techniques. One large drawback with neural networks is that they are quite demanding, computationally speaking. However, the properties of NNs make them in many ways suitable for FPGA or ASIC implementation. Not surprisingly there have been attempts at implementing Neural Networks on FPGAs, [Li *et al.*, 2006] and [Gironés *et al.*, 2005] are two of them. The contribution of Gironés et al. shows a way to pipeline one commonly used NN algorithm. Implementing NNs using a pipelined version of the algorithm and using hardware Gironés et al. claim to take much better advantage of the inherent parallelism of NNs than a corresponding software implementation would do, which is probably the case. The structure of NNs are inherently parallel, as indicated by Figure 3.3, and the computations in the NN are carried out by fairly simple processing elements at least from a mathematics point of view. The 'fairly simple' processing elements are unfortunately non-linear functions, typically a sigmoid function or similar which are inconvenient to implement in hardware. Gironés et al. discusses different methods to implement this non-linear function digitally (in hardware) which is by Gironés et al. considered as the main problem for digital implementation of NNs. Exploring the parallelism, implementing pipelined calculations and finding a suitable digitalization of the NNs the opinion of Gironés et al. is that an FPGA implementation of NNs is preferred over a software one, and from a calculation speed point of view that is probably the case. Li et al. also attempts implementation of NNs in FPGAs, also aiming for a significant speed-up by exploiting the inherent parallelism. Li et al. identifies a multiply and accumulate operation in the NN algorithm as being the main computation and hence being the one to put extra consideration on for FPGA implementation. Li et al. also identifies the routing resources to be a bottle-neck when implementing algorithms with such a high grade of parallelity as a NN. To utilize the routing resources more efficiently temporary data (which is used by a large number

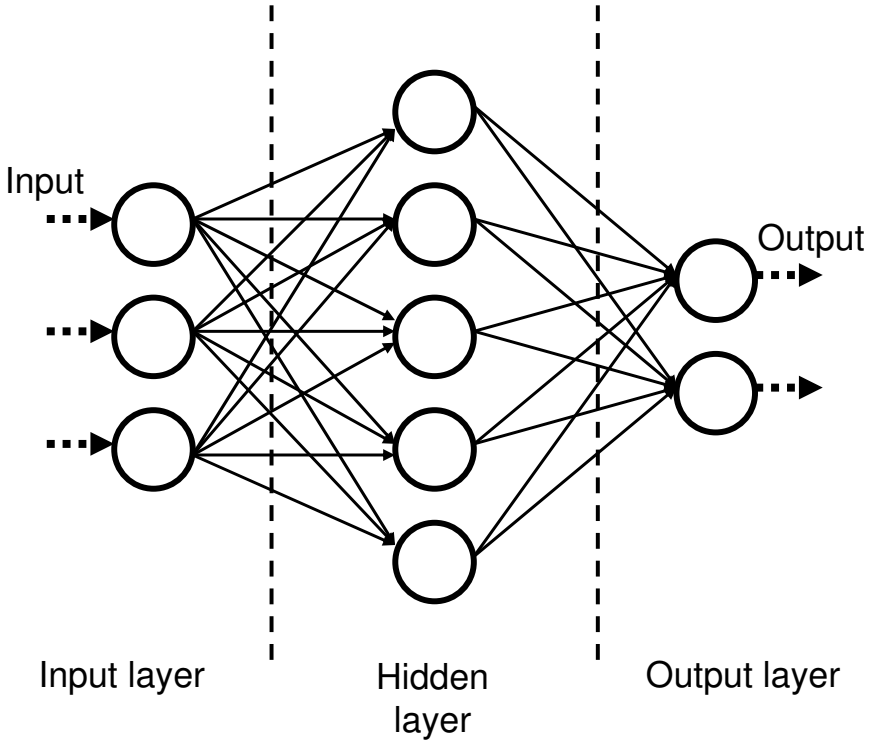

Figure 3.3 A typical outline of a Neural Network. The circles denotes the processing elements and the arrows the interconnects. The number of layers, units and the connection pattern can be varied giving the network different properties.

of computational units) are stored globally rather than in local RAM memory. The numeric precision needed for NNs are further discussed. Li et al. cites [Holt and Hwang, 1991] who finds that NNs should be implemented using at least 16-bit fixed point numbers to avoid decreased performance of the NN. The results of Gironés et al. and Li et al. indicate that there are large gains implementing NNs on FPGAs and it is safe to say that very interesting results will continue to emerge in this field.

3.7 Chapter Summary

This chapter discusses the topic of implementation of feed-back controllers and related algorithms within an FPGA environment. Starting with an in-

troduction discussing the benefits and drawbacks with FPGA implemented controllers. Three main strengths are identified, increased controller speed and robustness, increased price-performance ratio and reduced power-consumption. In many cases the time it takes to compute a controller algorithm using micro-controllers or DSPs is long enough to fill up the complete time-span between two corresponding samples, leaving no 'extra-time' for other computations or for having stability margin. The same logics implemented on an FPGA on the other hand would possibly not take more time to compute than the time it takes to sample the system. The time margin gained can be used in many ways, the most interesting would be to increase controller complexity, over-sample the system, thus increasing controller bandwidth and leaving more 'margin' in time, increasing the robustness. Total System cost in many cases gains from FPGA implementation, having an improved price performance ratio and enabling the designer to consider many sub-systems implemented on one device, a SoC solution.

Implementing control logics in a digital environment takes a lot of considerations about among other things sample rate and numerical precision. Regardless of whether implementation is carried out in a micro-controller, DSP or FPGA, some of these considerations were discussed. Control algorithms have to be discretized in a manner such that their continuous-time performance is maintained in discrete time. Criteria regarding the discrete controller sample rate compared to the system bandwidth would have to be full filled, the digital number-representation word length both for the data path and controller parameters has to be selected in a way so that controller performance is not degraded significantly when discretized.

On top of the 'normal' consideration which has to be made for any controller implemented in discrete-time special considerations have to be made implementing controllers using FPGAs. How to select the internal number format in an optimal way is more difficult in an FPGA case having the option of freely choosing the actual word length within each algorithmical step of the calculation. The issue of selecting an optimal internal number representation was presented and a number of solutions were suggested.

The internal numbers can either be represented using fixed-point or floating-point number representation. A fixed-point representation is in many ways more efficient than a floating-point representation saving chip area and increasing speed. A floating-point number representation is however much more flexible and convenient during algorithm implementation. One important issue is how to handle round-off and quantization effects within a control algorithm in an 'optimal' manner not wasting chip area or precision. In fact a large part of the design time is spent on this problem and some sort of automation is desirable. Some methods based on simulation were described, and they can possibly be used to automate the process.

Another consideration which becomes especially important implement-

ing controllers on FPGAs, as the bandwidth normally should be higher with FPGA based systems, is the issues connected with over-sampling and especially over-sampling in combination with limited number-representation precision. As explained traditional discretization methodologies of automatic control and filtering, using the shift operator, is ill-conditioned when applied in situations with high over-sampling rates and limited word length.

The solution presented is to use a divided-difference operator (which essentially an Euler-estimation of a derivative), the δ-operator, instead of the common shift-operator. Promising results obtained using the δ-operator in situations with very high over-sampling rates as well as the details of the δ-operator were accounted for. A rule-of-thumb indicating when the δ-operator should be used says that when sampling ten times the system bandwidth or more it is suitable to use the δ-operator instead of the shift-operator.

It is interesting to over-sample even though it infers problems since it is possible to increase controller performance by over-sampling, in some cases the format of the system may even enforce over-sampling. The benefits with the δ-operator are that as sampling rate increases the discrete time representation approaches the continuous time behaviour. Consequently the controller stability margin increases as the sampling rate increases which is not the case with the normal shift-operator. The original motivation behind δ and another benefit with this discretization method is its, compared to the shift-operator, superior numerical properties. It is actually possible to decrease the word length still maintaining numerical precision using the δ-operator.

Second to last the very important question of algorithm parallelization and reformulation was discussed. It is noted that it is the possibility to perform operations in true-parallel which makes the FPGA the powerful device it is. In order to obtain the gains in calculation speed it is hence essential to develop algorithms which can benefit from this parallel structure. The user has to make sure that branches in an algorithm which can be implemented in parallel actually are implemented as parallel computation paths. How to parallelize would depend on which development tool is being used. It is important to try to minimize sequential dependences on the algorithm level, and the user has to manually make sure that an algorithm is parallelized correctly on the 'algorithm-level'. The automated approaches that exist mainly work on the 'code-level' making it difficult for the tool to find global parallel paths in the sequential code. This is an important point in implementing controllers on an FPGA.

An algorithm shall not only be parallelized, it should be parallelized in a well balanced manner meaning that parallel sub-results shall finish at the same time-instance. This is important for two reasons, not to waste chip-area implementing an algorithm 'un necessarily parallel' and to avoid racing

within the data-path. Racing occurs when a sub-result belonging to one time instance is used in calculations together with a sub-result belonging to a completely different time instance, the corresponding result will then 'not belong in time'. Racing is less severe in control applications than in many other applications due to continuity of the underlying system but should never the less be strictly avoided.

Besides to consider algorithm parallelization it is important to avoid difficult operators such as trigonometry and exponentials implementing controllers in an FPGA environment, as well as in a micro-controller environment. Having to implement difficult operators it is advisable to use some method to approximate these functions in a way that saves chip area and time, the CORDIC is one approach discussed.

Lastly three different application examples were presented and discussed from the views of a number of different researchers. PID control, MPC control and Neural Networks all implemented in FPGAs were shown indicating possibilities, powers and special considerations using FPGAs in practical control applications.

4

An FPGA Implemented Heat Release Model

For successful control of HCCI (and most other low-temperature combustion concepts) throughout the useful operational area it is the common opinion that a closed-loop combustion control system with fairly large complexity is needed as described in Chapter 1. One or more models will need to be maintained on-line by the closed-loop 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 chapters it is understood that an FPGA could well serve as a very useful tool for implementing closed-loop 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 closed-loop combustion control system, namely the HR analysis. The reasons for implementing a HR computation or model rather than any other model are twofold. The HR, or rather the combustion timing, CA50%, being 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 and it is by many not considered to be possible to calculate the HR fast and accurate using the engine control unit for an foreseeable future. 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 precision and speed un-matchable by normal engine control units. The potential of FPGA-implemented models and the concept of using FPGAs for control, especially automotive control is simultaneously proven by this proof of concept study.

4.1 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 board simulating the engine pulses and cylinder pressure, an expansion module for the FPGA board featuring ADC and Digital-to-Analog Converter (DAC) and some surrounding circuitry adjusting signal levels etc. 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 are 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' is necessary.

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 448 user I/O ports. The board holds support circuitry to develop a complete system. Maximum clock speed of the FPGA is 100 *MHz* and the clock signal is supplied by an oscillator present on the board, although there is also a possibility to provide a custom oscillator. Besides the above mentioned equipment the card holds a number of peripheral devices; 16 *Mb* Serial Flash for FPGA configuration, 64 *Mb* DDR SRAM, on-board oscillators, 'P160' expansion header, JTAG programming/configuration port, 10/100 Ethernet, Alpha numeric LCD panel, RS232 port, LEDs, pushbuttons, DIL switches and General Purpose I/O pinout.

Desired FPGA configuration is either loaded directly onto the FPGA or stored in a serial flash memory produced by Atmel. If the configuration is loaded on to the serial flash it is automatically reloaded onto the FPGA on power-up with the help of a CPLD. There exist of course other solutions for the FPGA re configuration that might be more suitable in automotive applications, this approach is however flexible in a development environment. Configuration both of the serial flash and directly of the FPGA are carried out through the JTAG interface with the help of a Xilinx 'Parallel Cable 4'.

Not included on the basic FPGA board is the ADC and DAC converters. To gain access to the analogue world, a 'bolt-on' circuit board that fits in the 'P160' expansion header was used. This board is, as the FPGA board supplied by 'Memec/Avnet' and is named '160-Analogue-Kit'. The board features dual AD and DA channels. Both AD and DA converters are made by

'Burr-Brown'. Maximum clock speed of the AD, $ADC_{clk} = 53MHz$, the DA handles $DAC_{clk} = 165MHz$ maximum. Clock signal is supplied from the FPGA via the 'P160' header, the FPGA runs at 100 MHz limiting maximum ADC_{clk} to 50 MHz and DAC_{clk} to 100 MHz. The surrounding circuitry, on the AD channels, consist of an input buffer, a low-pass filter, a single ended to differential amplifier and latches. The DA has the same latches, an output buffer and a low-pass output filter. The expansion card was, considering the application, less 'bolt on' than expected at the time of purchase and modifications were made to the input circuitry of the AD channels in order to adapt the channels to suit the relatively low frequency that is of interest in this application, the DA channels are however left untouched.

The total FPGA system price was at the time of purchase $\approx$ €700 and must hence be considered as a low cost system, it is never the less a high performance system!

Design Tools

As briefly mentioned earlier the design of the FPGA HR algorithm is carried out in MATLAB/Simulink with the help of SGDSP. SGDSP contains a number of Simulink blocks that are already implemented in VHDL, and using these blocks it is possible to generate VHDL from a Simulink diagram. FPGA layout with the help of Simulink in DSP applications is discussed by [Todman *et al.*, 2005]. Please note that it is not possible to implement 'standard' Simulink blocks in the FPGA, it is necessary to possess a VHDL implementation of the blocks to implement (a number of such blocks comes with SGDSP). This is somewhat limiting. When running the 'hardware co-simulation' (compare Hardware-In-the-Loop simulation 'HIL') during the debug process, it is however possible to implement 'standard' Simulink systems on the PC communicating data with the FPGA through the JTAG interface, this is a handy feature during the debugging process. Note that it is necessary to possess a copy of the Xilinx ISE design suite in order to use SGDSP since SGDSP uses the underlying ISE tools to generate the design files. After co-simulation of the system, generic VHDL files are created from Simulink, processed by the relevant FPGA design tools and downloaded to the serial flash. When the design finally resides on the serial flash the FPGA system is 'stand alone' from the computer.

Test Environment

Desktop tests were carried out during the development. The 'interface' to the simulated/real engine consists of three signals, Crank Angle Degree Pulses (CADP), Top Dead Center Pulses (TDCP) and analogue cylinder pressure P_{cyl}. Current engine position is assumed to be measured with 0.2 Crank Angle Degree (CAD) accuracy, the engine hence produces 5 CADP every physical CAD. Besides CADP the angle sensor is assumed to give one TDCP

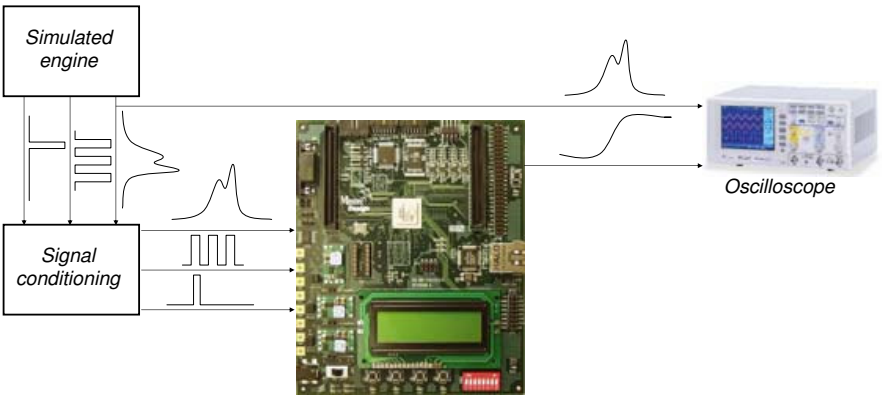

Figure 4.1 FPGA experimental system overview, from [Wilhelmsson *et al.*, 2006].

every time the engine has revolved two complete revolutions. Figure 4.1 provides an overview of the experimental setup, this engine 'interface' concurs with the setups in [Olsson *et al.*, 2001], [Bengtsson *et al.*, 2004] and [Bengtsson *et al.*, 2006].

During the development and testing P_{cyl} was simulated by recorded cylinder pressure traces, both a motored and a fired trace were recorded. The basis for P_{cyl} are pressure traces recorded from the 12 l Scania engine used by Olsson et al., geometric data of this engine is noted in Table 4.1. A 'in house' developed device simulating CADP, TDCP and P_{cyl} synchronously at an engine speed of 1200 rpm was used for system tests during the development. P_{cyl} was simulated with a vertical resolution of 8 bit, horizontal resolution was 720 samples, that is one sample each CAD. DA conversion was carried out with the help of a R/2R ladder and the output of the device was buffered. The original pressure curve was somewhat distorted due to the limitations of the engine simulator.

4.2 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, see Chapter 2 for properties of a mixed system. The DSP design approach was selected due to the relative simplicity of the calculations and the system. If the system is expanded to a complete engine control system a processor core might be added.

Algorithm

A net HR (Q_{HR}^{net}) calculation was implemented in the FPGA, that is the HR calculation disregards heat transfer losses and crevice losses. Heat transfer losses are caused by convective energy loss to the combustion chamber walls. Crevice losses are caused by trapping fuel-air mixture in the crevices between the piston and the cylinder wall, thus avoiding combustion. The reason for neglecting crevice and heat transfer 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 is regarded as legitimate.

The calculation of Q_{HR}^{net} is carried out in a non-conventional manner. The conventional way to calculate Q_{HR}^{net} is through the integration of Equation 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 very high 'over-sampling' rate. Instead of using Equation 1.1 it is possible to calculate Q_{HR}^{net} through Equation 4.5. Equation 4.5 origins from the conservation of energy and is motivated through Equation 4.1-4.4. This is a promising optional method to calculate the HR are showed in [Tunestål, 2000].

$$\left.\begin{aligned} dU &= dQ - dW \\ dU &= nC_v dT \\ dW &= pdV \end{aligned}\right\} \Longrightarrow nC_v dT = dQ - dW \Longrightarrow dQ = nC_v dT + pdV \quad (4.1)$$

$$pV = nRT \Longrightarrow \frac{1}{nR} d(pV) = dT \quad (4.2)$$

$$C_v = \frac{R}{\gamma - 1} \Longrightarrow nC_v dT = \frac{nR}{\gamma - 1} dT \overset{(4.2)}{\Longrightarrow} nC_v dT = \frac{1}{\gamma - 1} d(pV) \quad (4.3)$$

$$dQ = \frac{1}{\gamma - 1} d(pV) + pdV \quad (4.4)$$

$$\begin{aligned} Q &= \frac{1}{\gamma - 1} \int_{\theta_{start}}^{\theta} d(pV) + \int_{\theta_{start}}^{\theta} pdV = \\ &= \frac{1}{\gamma - 1} (p(\theta)V(\theta) - p(\theta_{start})V(\theta_{start})) + \int_{\theta_{start}}^{\theta} p(\theta) \frac{dV}{d\theta} d\theta = \\ &= \underbrace{\frac{1}{\gamma - 1} p(\theta)V(\theta) + \int_{\theta_{start}}^{\theta} p(\theta) \frac{dV}{d\theta} d\theta}_{\text{Calculated on-line}} - \underbrace{\frac{1}{\gamma - 1} (p(\theta_{start})V(\theta_{start}))}_{\text{Estimated constant, added off-line}} \end{aligned} \quad (4.5)$$

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). To avoid racing-phenomena the different internal computational branches were synchronized by the addition of unit-delays and Simulink-specific synchronization blocks in the data-path. Keeping Equation 4.5 in mind, two multiply operations, $p(\theta)V(\theta)$ and $p(\theta)\frac{dV}{d\theta}$, were computed in parallel within the FPGA. The integration (summation) acting on $p(\theta)\frac{dV}{d\theta}$ 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 Equation 4.5 are added to obtain the part of the result which is calculated on-line.

When Equation 4.5 was implemented the parts of the equation that were known in advance were not calculated on-line in order to simplify the equation as much as possible, thus gaining speed. Instead they were mapped as a function of current engine CAD in the distributed RAM of the FPGA, this goes for V and dV. As basis for the volume maps the geometry listed in Table 4.1 was used (since this engine is the basis for the pressure traces used in the test environment).

The time resolution is also of high algorithmic interest and it has to be noted. The described setup had the properties of an asynchronous system since the engine delivers CADP:s 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 non-synchronous with the CADP. This is an unconventional approach in an engine control system. The described calculation of Q_{HR}^{net} demands, as understood from Equation 4.5, some synchronization between the calculation of Q_{HR}^{net} and the engine position (in order to retrieve the correct V and dV). 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 which were kept in the RAM memory. 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 AD converter was, as previously noted 50 MHz, this was hence the sampling rate of P_{cyl}. This is by far a higher sampling rate than the update speed of V and dV, 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 over-sampling. The parallel nature of the FPGA however still made this the preferred way to perform the design. The FPGA

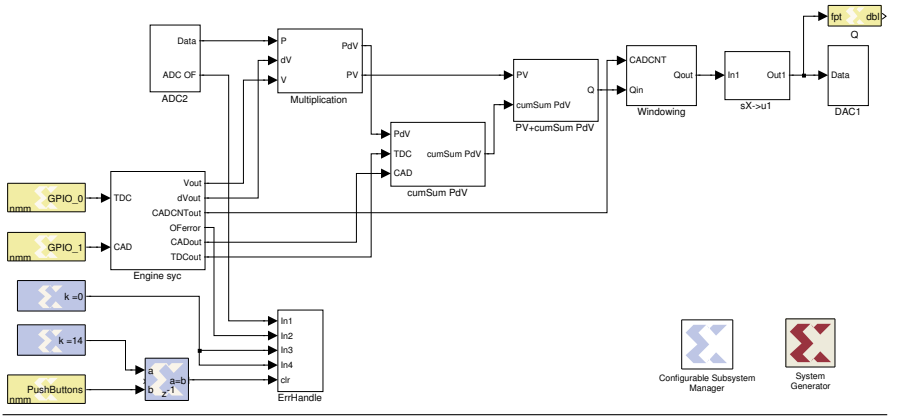

Figure 4.2 The DFG implementing the HR, from [Wilhelmsson *et al.*, 2006]. The figure describes the 'top-level' of the implemented algorithm/system, it is important to note that each block at this level contains a lot of logics. The two different parallel branches making up the computation are not obvious from the top-view.

Number of Cylinders	6
Swept Volume	11705 cm^3
Compression Ratio	18 : 1
Bore	127 mm
Stroke	154 mm
Connection Rod	255 mm

Table 4.1 Geometric properties of the Scania engine.

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 totally destroy the largest benefit of using the FPGA, i.e. 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.

$P_{lsb} = 1464.84\ Pa$	$V_{lsb} = 5.05 * 10^{-7}\ m^3$
$dV_{lsb} = 1.74 * 10^{-9}\ \frac{m^3}{CADP}$	$Q_{lsb} = 3.85\ J$

Table 4.2 the resolution of the input and output variables.

4.3 Experimental Results

The major result of this investigation was of course the successful implementation of the described HR algorithm in the FPGA environment. The performance of the implementation is a point that can not be stressed enough, a P_{cyl} sample that arrives to the FPGA from the AD converter is, considering the timescale of an engine, calculated immediately. Immediately in this case means 12 FPGA clock cycles, $FPGA_{clk} = 100\ MHz \rightarrow 120\ ns$! This, in other words, means that when the AD converter 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 DA converter! 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_{cyl} 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_{cyl} and concurrently with P_{cyl}, it is motivated to call the system a 'virtual Q sensor' meaning that Q_{HR}^{net} is calculated the moment P_{cyl} 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 Figure 4.3 is included. Figure 4.3 shows P_{cyl} 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_{cyl} 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 Figure 4.3 that Q_{HR}^{net} is calculated with very low latency with respect to P_{cyl}.

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' (meaning 120*5 point) HR analyses each CAD at 1200 rpm! The limit of the throughput is the AD conversion speed.

The last point of the results is the correctness of the output. To verify the correctness 100 cycles are sampled to the PC. Due to poor quality of the pulses originating from the simulated engine some cycles however had to be removed due to sudden steps in the middle of the calculation. The num-

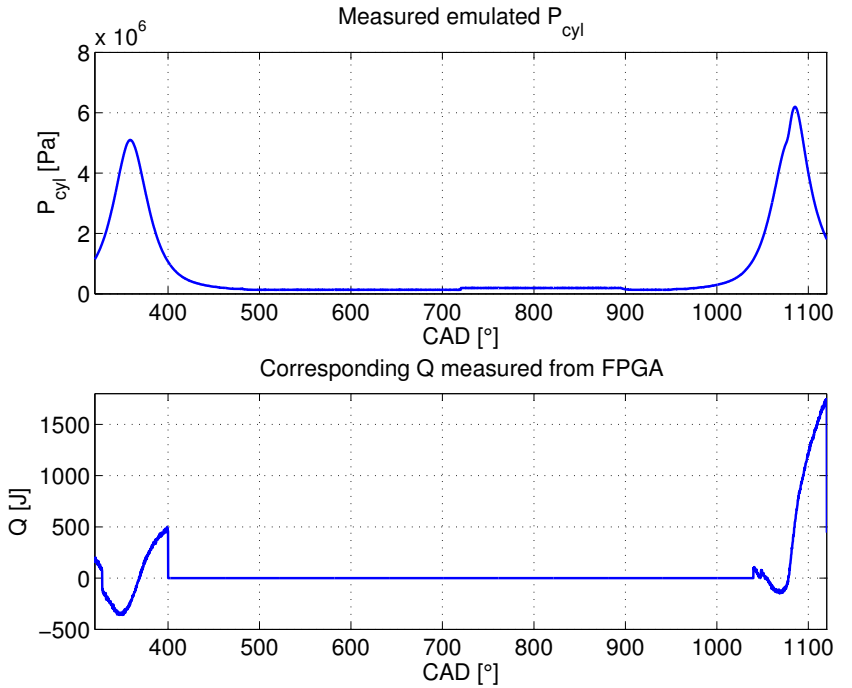

Figure 4.3 Output from the FPGA system when the 'combustion' is suddenly switched on, showing true 'in-cycle' performance. From [Wilhelmsson *et al.*, 2006].

ber of disregarded cycles are in the range of 20-30 cycles depending on the 'mood' of the simulated engine. Figure 4.4 shows all the sampled cycles that are non damaged, as understood from the figure most of the calculated cycles holds a high enough signal quality to use in a feedback control loop. Figure 4.5 shows the average of the cycles shown in Figure 4.4. It also shows the 'corrected' values calculated off-line by the PC and Matlab. As understood from the figure the FPGA implemented algorithm is able to calculate Q_{HR}^{net} accurately enough. If the FPGA output and off-line Matlab calculated Q_{HR}^{net} are compared to a Q_{HR}^{net} calculated from Equation 1.1 consistency is again found. Q_{HR}^{net} according to Equation 1.1, is however not shown in order not to blur the figure.

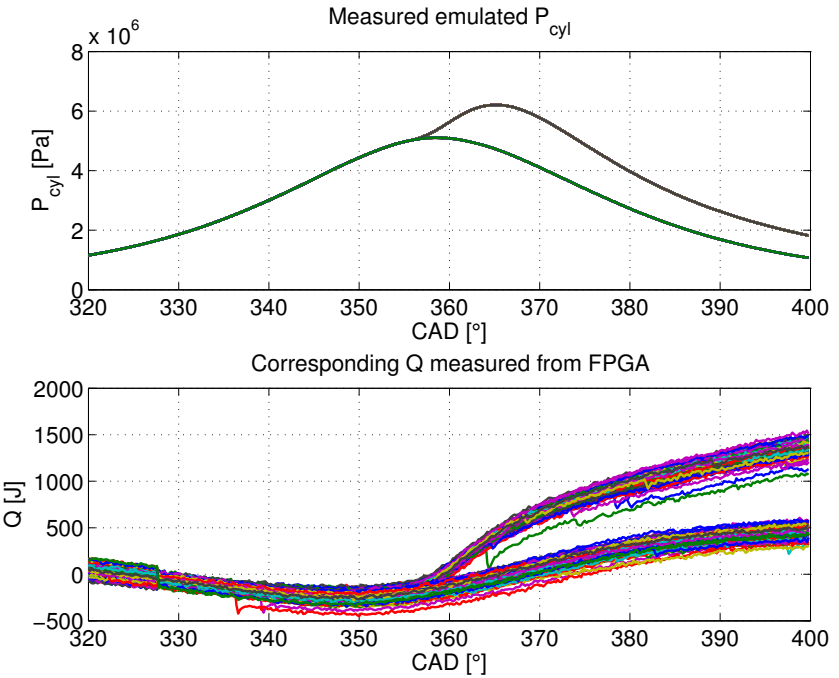

Figure 4.4 Non average results corresponding to figure 4.5, from [Wilhelmsson *et al.*, 2006].

4.4 Discussion

The results in Figure 4.3 - Figure 4.5 show that implementation of Equation 4.5 was successful. As previously noted some cycles are removed from the results before presentation. The intention of the final system is of course that every cycle will be calculated perfectly, this is also achievable, better edge detection logics in combination with better quality of the positioning pulses will do the trick. The reason for the erroneous cycles are mainly thought to be issues with the simulated engine. As it was difficult to find a suitable signal simulator, the simulated engine had to be developed in house, and even though it performs well 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 visible in Figure 4.4 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' there appears to be an offset prob-

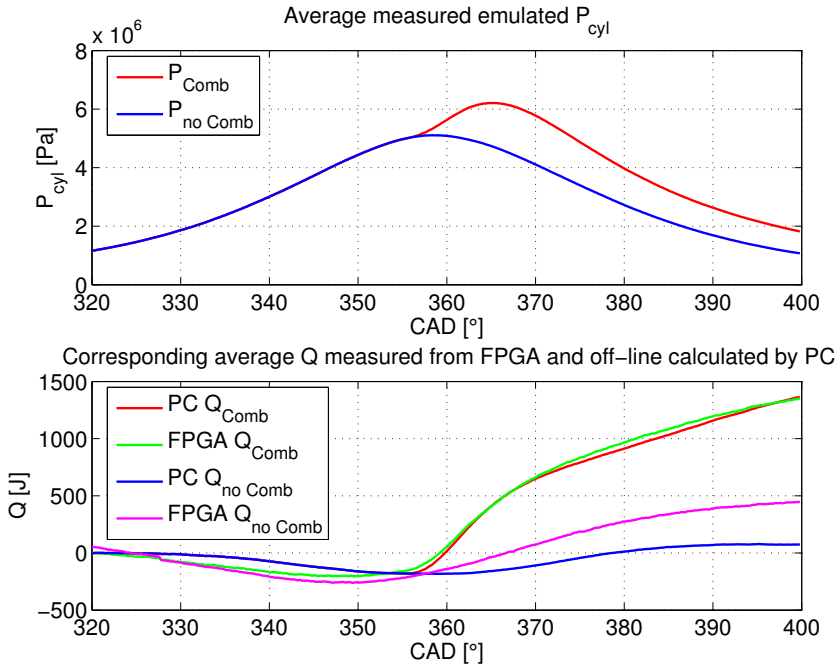

Figure 4.5 Average output from the FPGA system compared to the corresponding values corrected in the PC, from [Wilhelmsson *et al.*, 2006].

lem, in Figure 4.4 there appears to be different bias in the different cycles of Q_{HR}^{net}. This offset problem is explained by the nature of the AD converter connected to the FPGA. The AD converter was purchased as an expansion card suitable for the expansion header on the FPGA board, the case was however that the input circuitry on the AD converter card was intended for usage in very high frequency systems and the signal paths to the AD converter hence blocked the very low frequency cylinder pressure signal. The input circuitry hence 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 Figure 4.4.

The two noted issues are unfortunate and an effort will be made to correct them for future versions of the system. When the erroneous data was removed as described and the remaining cycles were averaged Figure 4.5 emerged. Figure 4.5 shows that the FPGA system despite the noted issues on average performs well. The difference between Q_{HR}^{net} calculated by the PC off-line and Q_{HR}^{net} calculated by the FPGA is not large, at least not in the case with 'combustion'. Q_{HR}^{net} in the 'motored' case is less consistent between

the PC calculations and the FPGA, the explanation for this is thought to be the bias issues on the input of the AD converter. 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.

The difference between the corrected Q_{HR}^{net} and the FPGA Q_{HR}^{net} is mainly thought to 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 4.2. In the table it is clearly visible that despite the limited input word-length of only 12 bit the resolution (known by the value of the Least Significant Bit (LSB)) is, by far, enough (P_{lsb} is calculated on a 60 bar P_{max} assumption, if $P_{max} = 200$ bar are used $P_{lsb} \approx 0.05$ bar).

Even though the resolution is 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 the FPGA internal number representation and the internal timing of the FPGA are 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, FPGA design is very sensitive to the available tools regardless of its application.

4.5 Chapter 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 feed-back candidate for combustion control. A re-formulated 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 never the less an FPGA implemented HR model having an extra-ordinary 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.

5

FPGA Based Rapid Prototype System – a Suggestion

For developers of automotive control, prototyping and initial tests are a hassle. Commercial solutions are available but the price and especially the price/performance ratio opens the field for more cost effective solutions. Automotive rapid prototype systems seen so far are mainly processor based systems with standard interrupt driven measurement and actuation. Control systems based on high time resolution measurements of for example cylinder pressure are difficult to implement using these systems, neither is it possible to implement controller loops with an extremely high bandwidth in combination with expensive algorithms. According to, among others, [M. Oki *et al.*, 2006] the bandwidth of actuators, meaning fuel injectors (such as piezoelectric fuel injectors) is currently increasing. In order to utilize the performance gains of the fuel injectors, new non-conventional engine control and rapid-prototype systems have to be developed. Conventional engine control systems typically have a performance-grade which enables them to control the engine from one cycle to the next, at best. Until now this has not been an issue since normal actuators so far have lacked capability beyond cycle-to-cycle control. The new high bandwidth fuel injectors in combination with a high bandwidth control system would however give interesting combustion feedback control possibilities. Using FPGAs as a base for such a system is an interesting and maybe enabling option to consider. This chapter sketches one possible architecture of such a system and which devices that could be included.

5.1 Automotive Control Rapid Prototyping

Simulink based development environments for rapid prototyping of automotive control have been previously attempted and commercial products exist on the marked from among others the company dSpace. Besides dSpace, IFP in France have a solution enabling rapid prototyping and regression tests both on the software and system level. Other rapid prototype systems are for example presented by [Viele *et al.*, 2005] showing a solution based upon National Instruments products, the CompactRIO system, in combination with an FPGA module. In this case LabView is chosen as programming/FPGA configuration environment. The approach taken by Viele et al. is a more conventional one and even though the theoretical possibility to implement algorithms, not only pure logic, in the FPGA exists it is not utilized. The system is used for implementation of a 'standard' control unit on a motorcycle. It is also reasonable to believe that LabView would not be the first choice for complex and high performance FPGA controller and algorithm implementations. [Beaumont *et al.*, 2006] shows one approach where an 'off the shelf' processor based rapid prototype system from the company Ricardo is fitted with an FPGA based I/O system. The intention of Beaumont et al. seems to be to use the FPGA as a coprocessor to assist the processor-based rapid prototype system with data acquisition, analog to digital conversion and data pre-processing. The architecture is less suited for feedback control loops implemented purely in the FPGA and very-high bandwidth control since actuators are connected to slow subsystems and thus not directly accessible from the FPGA.

5.2 The Setup

The over-all idea with the system proposed in this chapter is to combine very flexible parts, featuring high flexibility in implementation but limited capacity, with less flexible but very high performance parts. The parts regarded as flexible would be an x86 PC running the Matlab/Simulink xPC-Target environment which is flexible in the sense that it is very easy to re-implement and change controllers and supporting software/algorithms residing in the xPC-Target environment. The part of the system regarded as less flexible would be reconfigurable hardware (FPGA) which has to be configured using special tools featuring limited or no rapid prototyping capabilities. The resulting system will be referred to as Engine Dyno Controller (EDC), the reason being that it is a control system suitable for controlling the complete dyno setup, not only the engine.

Figure 5.3 gives an overview of the engine dynamometer setup. The figure describes a 'typical' engine control design setup and the devices present

in the figure would be present in most setups in one form or another. The center of the setup is of course the actual engine, which also is the actual plant for the control engineer. The engine generates energy during operation, this energy needs to be taken care of, a device capable of maintaining the rotational speed is hence necessary, i.e. an engine dynamometer. In this case the dynamometer consists of an electric motor in combination with control electronics. The EDC system would need the capability to demand an engine speed setpoint from the dynamometer in order to carry out transient identification and validation experiments. Preferably control data for the dynamometer would be transferred through MODBUS to the control electronics of the electric motor, RS-485 will be used as the physical layer for the MODBUS communication.

In each cylinder of the engine a piezoelectric cylinder pressure transducer will be present. The pressure transducers would be connected to charge amplifiers which convert the electronic charges originating from the transducers to voltages. Besides analog outputs for pressure, the charge amplifiers have the possibility of control related communication. In reality however the control parameters of the charge amplifier is very seldom changed, and this communication path will hence not be implemented. Each analog output of the charge amplifier is connected to two different analogue I/O inputs of the EDC.

A vast variety of other devices might also be connected to the engine setup to perform different measurements, for example thermocouples, air/-fuel ratio sensor(s), emission measurement devices, non-cylinder pressure transducers and the crank angle counter (angular positions sensor). These devices are typically sampled by the EDC (apart from the crank angle counter) either by an analogue I/O device (preferable) or in some cases by digital communication (GPIB, RS-XXX, or ethernet). The crank angle counter communicates through parallel digital channels physically carried on fiber optics. Based on the inputs from the sensors the control engineer wants to perform two tasks:

- Implement and/or validate automatic control logics
- Collect sensing data for post processing

In order to implement control logics actuators are, of course, needed. In this case there are mainly two types of actuators with significantly different bandwidths. The high bandwidth kind are typically piezoelectric fuel injectors, spark coils and inductive fuel injectors. Low bandwidth actuators would typically be stepper motors, controlling for example throttle positions and turbo settings (for a non fixed turbo).

In the end it is of course the engine control logics which is of main interest however, in order to have an efficient experimental setup, the peripheral

devices needs to be controlled by the same system as the engine control logics resides in. If they are, it enables the control engineer to perform identification and transient experiments on for example the engine speed (if the controller can communicate with the dynamometer). It also enables the control engineer to perform experiments on associated control problems, for example boost-pressure control or other air-path control. Besides capability to communicate with 'need to use' devices always present in the system it is of utmost importance that it is quick and easy to plug in new devices, sensors or actuators which may be needed in future control experiments.

5.3 Loop Overview

The main idea with the described system is to have the possibility of two control loops with significantly different bandwidths as indicated in Figure 5.1. Color marking distinguishes the two different loops in the figure, the loop marked in black would be the high bandwidth loop carrying out control tasks with high bandwidth requirements e.g. injection control. It can also assist the low bandwidth loop (marked in grey) with the cycle-to-cycle control of, for example, engine load. The high bandwidth loop, indicated in Figure 5.2, is found to consist of; the engine, cylinder pressure sensor (CPS), charge amplifier (CAMP), high frequency AD converter (HFADC), the FPGA board and last but not least the injection system (communication protocol between the components vary). The low bandwidth loop, noted in grey in Figure 5.1 and still with Figure 5.2 in mind, would consist of; the engine, the cylinder CPS, the CAMP, the framed ADC (FADC), the x86-PC and the FPGA. A FADC is a ADC which stores a defined number of samples, one frame, in local memory before handling the complete frame at the same time to the PC. Also note that the FPGA board is the master controller of the injection system and the output of the low bandwidth combustion control loop hence has to pass through it. Pressure sampling in the high bandwidth loop is asynchronous to the engine revolution which means that some synchronisation algorithm is needed in order to implement the injection control and other algorithms based on the engine crank angle. The low bandwidth loop on the other hand is clocked synchronously with the engine revolutions and synchronisation is hence built-in.

The high bandwidth loop is complemented by the x86-PC running xPC-Target environment. Implementation of controllers in the high bandwidth loop is time consuming but controller implementation in xPC target is not. The existence of the PC hence enables the designer to, in a very rapid manner, implement controllers with the support of a large amount of plug in modules for I/O. The xPC-Target environment will be used for implementation of user interface, implementation of the low-bandwidth control of the

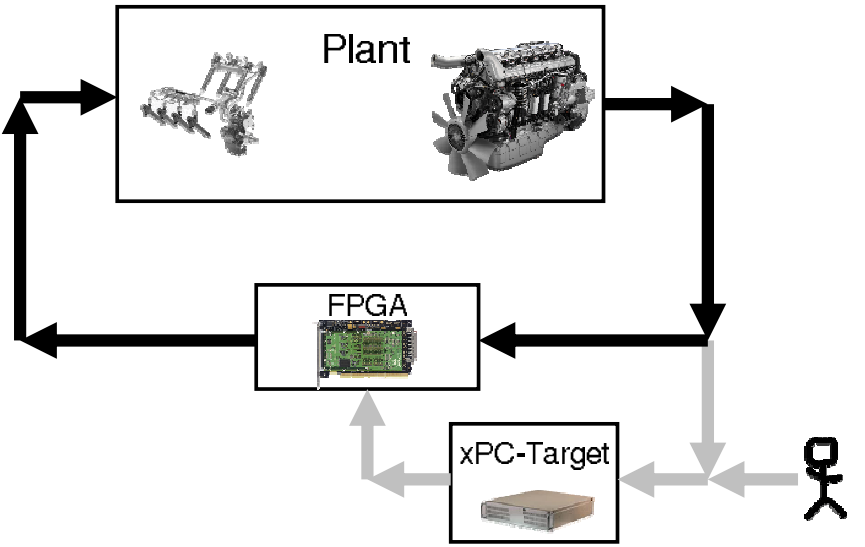

Figure 5.1 Overview of the two different loops intended to have two significantly different bandwidths, from [Wilhelmsson *et al.*, 2007]. The loop consisting of black arrows will have a high bandwidth but be less flexible consisting of an FPGA. The loop consisting of grey arrows is intended to be more flexible consisting of a PC being configured in Simulink. Speed performance will however be decreased using a PC and Simulink.

combustion (cycle-to-cycle controllers) and the low bandwidth control of peripheral devices, such as air path control. For clarity, the control loops handling the peripheral control are not indicated in Figure 5.1. Both the HFADC and the FADC could act as inputs for these loops, normally however the FADC would act as an input since peripheral control tasks have lower bandwidth requirement. It would typically be implemented in the Simulink environment. Protocols like RS-XXX, GPIB and ethernet would be used for communication with actuators.

5.4 Loop Bandwidth

The bandwidth of the two loops in absolute terms would of course be determined by the slowest component included in each loop. To determine

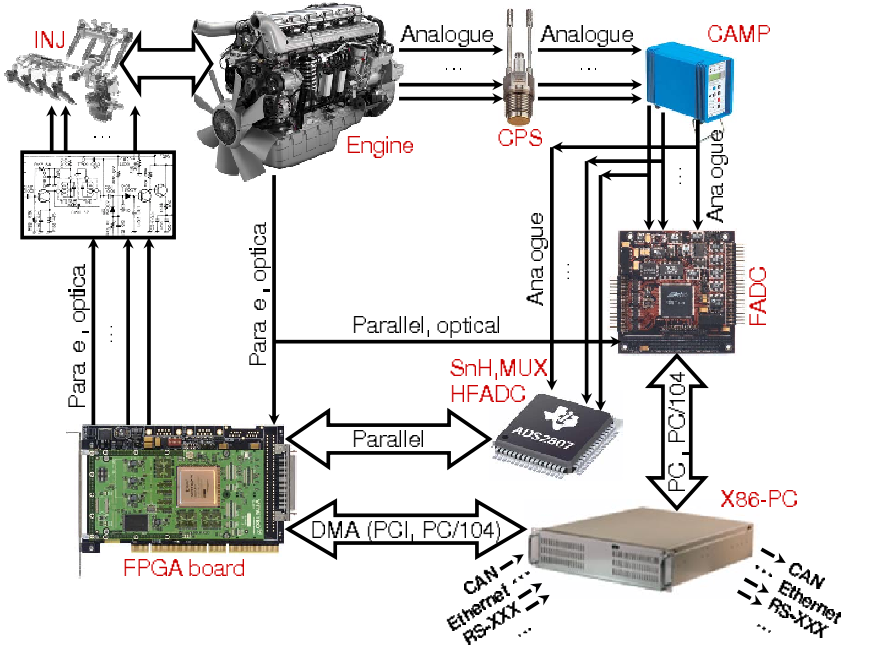

Figure 5.2 Overview of the components within the suggested control system, from [Wilhelmsson *et al.*, 2007].

the bandwidth of the two different loops the bandwidth of each component hence must be found, starting with the high bandwidth loop.

The clock frequency of the FPGA would be at least $100MHz$ depending on device selection, FPGA clock would hence not be an issue. Clock frequency of the HFADC would, again depending on device, range between a few kHz up to $100MHz$. As input the HFADC takes the output signal from the CAMP which in turn takes the output from the CPS. CAMP cut-off frequency would be in the range of $200kHz$ and CPS natural frequency in the range of $40 - 200kHz$ all according to manufacturer specifications. The last link in the HF loop is represented by the fuel injectors. Work published by [M. Oki *et al.*, 2006] indicates the available bandwidth of diesel fuel injectors. It is found that the time it takes the injector to lift the nozzle needle to half, from the instance the command signal is given, is $200\mu s$ and f_{-3dB} is hence $\approx 5kHz$.

Appropriate loop bandwidth can be decided in many different ways, if the Nyquist criterion is used the sampling frequency of the HFADC would be decided to $2 * 200kHz = 400kHz$ with the cut-off frequency of the CPS in mind, or $2 * 5kHz = 10kHz$ based on injector bandwidth. However, since

this system is intended for control purposes another sample rate criterion should be used ensuring control system performance. [Åström and Wittenmark, 1997] suggests a sampling frequency between 10 to 30 times the loop bandwidth. Loop bandwidth is limited to the injector bandwidth of $5kHz$ and an appropriate minimum sampling/control frequency would hence be $50 - 150kHz$, if all six cylinders together are sampled by the same ADC it has to be able to sample at $6 * 150kHz = 900kHz$. For the high speed loop the Nyquist frequency will consequently be $150/2 = 75kHz$. A fairly high loop bandwidth in combination with the fact that the FPGA clock frequency is, compared to the sampling frequency, significantly higher enables large amounts of calculations to be carried out between each CPS sample. For example in order to implement complex controllers (MPC etc), with such high bandwidth requirement, in multiple cylinders simultaneously the very high throughput, parallelism and high clock speed of the FPGA is regarded as essential and the use of the FPGA system is hence motivated. Preferably the HFADC is operated synchronously with the FPGA clock. The FPGA should drive the HFADCs and in this way traditional real-time and synchronisation issues are avoided. The performance of the FPGA board is the enabling factor, letting the systems perform very complex calculations between each sample, without violating the bandwidth of the loop.

The low bandwidth xPC target based loop shares some components with the high bandwidth one, the CPS, CAMP, the FPGA and the injection system. In the case of the low bandwidth loop the FADC would be clocked by the crank shaft pulses, a method which can be regarded as the 'standard' method. The sampling frequency will hence depend on current engine speed, making anti-aliasing filtering more difficult. For simplicity an anti-aliasing filter with cut-off frequency decided by maximum engine speed would be used. Maximum clock speed of the FADC is decided by the crank angle sensor resolution and the maximum physical engine speed. Results are published using different angular resolution. One pulse each CAD is frequently used, in this case a resolution of five pulses each CAD as used by among others [Olsson *et al.*, 2001] and [Wilhelmsson *et al.*, 2006] is preferred. Maximum engine speed is selected to be $6000rpm$ and with five crank pulses each physical CAD the crank pulses will arrive at a frequency of $180kHz$ according to Equation 5.2, at maximum engine speed and frequencies up to $180/2 = 90kHz$ are resolvable according to the Nyquist criterion. Consequently the crank pulses will arrive at a frequency of $24kHz$ at the minimum engine speed of $800rpm$ (according to Equation 5.1) still using five crank pulses each CAD, Nyquist frequency for minimum engine speed is hence $12kHz$. The desire is to be able to handle at least a 6 cylinder engine at maximum engine speed of $6000rpm$ preferably with a time resolution of five samples each CAD.

$$f_{min}^{CAD} = \frac{800\ [n/min]}{60\ [s/min]} * 360\ [CAD/n] * 5\ [cad^{-1}] = 24\,000\ [Hz] \tag{5.1}$$

$$f_{max}^{CAD} = \frac{6000\ [n/min]}{60\ [s/min]} * 360\ [CAD/n] * 5\ [cad^{-1}] = 180\,000\ [Hz] \tag{5.2}$$

5.5 Suitable EDC Hardware

The suggested electronic hardware setup is indicated in Figure 5.2, the system is intended to be built around a PC (lower right in the figure). There are a number of different communication interfaces which could be used between the x86 computer and peripheral components, for example PCI, PC/104 or ethernet. Which interface that is best suited for the application is mainly a question which I/O devices that hold support for Simulink/xPC-target. The support for framed sampling in xPC-Target is somewhat limited and one has to choose the I/O device with this in mind. As for the FADC there are mainly two suitable devices supported, either an analog I/O module from 'Diamond Systems' named 'Diamond-MM-32-AT' or one from 'United Electronic Industries (UEI)' named 'PD2-MFS-2M/14'. The Diamond Systems device is a 16-bit one, using the PC/104 bus, the device holds framed sampling support in xPC-target and is capable of a sample frequency of 200 000 sps. However with this sampling frequency it is possible to handle an engine speed of 5556rpm in a 6 cylinder engine (as desired) only if a resolution of 1 CAD is used (see Equation 5.3). The Nyquist frequency would then be $\approx 16700Hz$ at maximum engine speed (Equation 5.4).

$$n_{max}^{Diamond} = \frac{\frac{200\ 000[s^{-1}]}{6[]}}{360[n^{-1}]} * 60[s/min] \approx 5556\ [n/min = rpm] \tag{5.3}$$

$$f_N^{Diamond} = \frac{\frac{200\ 000[s^{-1}]}{6[]}}{2} \approx 16\,667\ [Hz] \tag{5.4}$$

The other option, the UEI device, is a 14-bit device using the PCI bus and capable of 2 $Msps$. Using the UEI device it is possible to sample five times each CAD for a six cylinder engine up to $\approx$ 11 000rpm according to Equation 5.5 or handling the engine speed criterion (6000rpm) for up to a ten

cylinder engine according to Equation 5.6 which, in combination with the fact that the UEI device uses the more modern and faster PCI bus, would make the UEI device the preferred one even though its resolution is 2 bit less than the Diamond Systems device. Framed sampling is an essential feature in order to implement the system with cylinder pressure based cycle-to-cycle combustion control with the cycle-to-cycle controllers implemented in xPC target. Due to the desired time resolution of cylinder pressure measurements the processor would not be able to maintain the controllers if the sampling were taken care of in an interrupt driven sample to sample manner.

$$n_{max}^{UEI} = \frac{\frac{2M[s^{-1}]}{6[]}}{1800[n^{-1}]} * 60[s/min] \approx 11\,111\ [n/min = rpm] \tag{5.5}$$

$$n_{max}^{UEI} = \frac{\frac{2M[s^{-1}]}{10[]}}{1800[n^{-1}]} * 60[s/min] \approx 6667\ [n/min = rpm] \tag{5.6}$$

Besides the I/O module connected to the PCI or PC/104 bus to work with xPC-Target one more ADC will be present in the system, the HFADC sampling values to the FPGA. That ADC will share the input of the FADC in an analogue manner and will have a high specification, a data resolution of $16bit$ and a maximum sample frequency of up to $25MHz$ is possible. Minimum sample frequency for this ADC is $150 * 6kHz = 900kHz$. A high spec ADC furthermore provides capability for system expansion by addition of high speed signals. The high speed ADC will not run on a higher clock frequency than necessary. The selected ADC could preferably be LTC2203 from Linear Technology (other options from Linear in the sampling range of $10 - 105Msps$ are LTC2207, LTC2206, LTC2205, LTC2204, LTC2203, or LTC2202). A MUX and sample-and-hold circuit might be needed as well. The outputs of the HFADC will be communicated in a parallel manner to the digital I/O of an FPGA development board.

The FPGA development board present in the lower left of Figure 5.2 will be the center of the system. Communications between the FPGA card and the x86 PC will be carried out using Direct Memory Access (DMA), which enables very high speed communication in a manner which is fairly simple to implement. Thus the FPGA board will have to be connected to one of the data buses of the PC. Devices suitable for this are for example the 'Virtex-5 LX110 PCI Development Board' from Vmetro which can be connected to the PCI bus or the 'TS-104-3001' from GE Fanuk Embedded Systems, which handles the PC/104 bus.

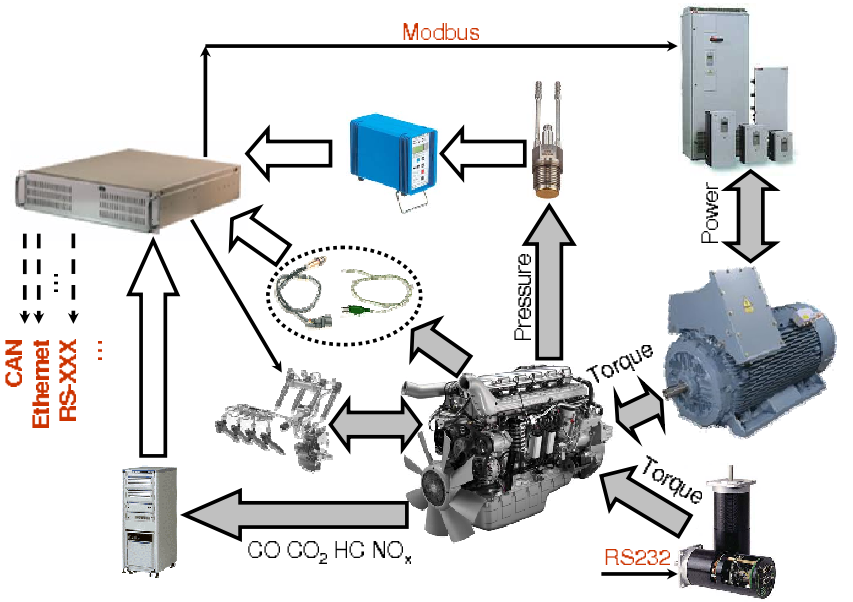

Figure 5.3 Overview of a typical engine dynamometer setup, from [Wilhelmsson *et al.*, 2007]. Grey arrows denote physical relationships, white arrows represent sensor-information flow and black is control-signal flow.

5.6 Software/Hardware Configuration

The configuration of the rapid prototype system is of course a key issue, if the configuration process is too difficult and time consuming the prototyping may prove to be too slow. The configuration of the cycle-to-cycle controllers which are run on xPC-Target are of course carried out in Simulink with the relevant block sets. It should hence be possible for the control engineer to implement the cycle-to-cycle controllers in a rapid manner. Configuration of the target computer (downloading of firmware) as well as operator interface data will be carried out over ethernet.

Design of FPGA configurations however tend to be less rapid than the graphical programming of the cycle-to-cycle controllers in Simulink. There exists rapid prototype tools for FPGA environments as well, for example the Xilinx toolbox System generator DSP used in [Wilhelmsson *et al.*, 2006]. System generator, being a plug-in tool to Simulink, features the same graphical programming environment as Simulink. Altera and other manufacturers of EDA tools also have options similar to System generator DSP either to use

in combination with Simulink or stand alone. Previous experiences made by the authors [Wilhelmsson *et al.*, 2006] however point in the direction of using tools more specialised on FPGA configuration. The reason is that tools like System generator DSP are not regarded to supply a high enough grade of automation and they appear not to be completely mature. In combination with a lack of transparency it is in some cases, according to previous experiences, very difficult to successfully carry out larger FPGA designs. The approach selected is instead to implement the main part of the intended design in a better suited and more complete design tool like the Xilinx ISE. Xilinx ISE is a toolbox containing the complete tool-chain for implementation in Xilinx FPGAs. Other FPGA manufacturers provides similar tools for their FPGAs. With this it is not said that every part of the design has to be carried out in pure VHDL, the possibility of carrying out graphically based designs/design parts still exists in ISE and the other similar tools. It is also possible to design modules in Simulink with the help of system generator DSP and 'plug them in' to a skeleton system designed in Xilinx ISE or similar. Design of logics designated for the FPGA is never the less more time consuming than implementing logics designated for an xPC-target system in Simulink.

5.7 Chapter Summary

A novel suggestion on the structure of an Engine Dyno Controller has been presented. The aim for the EDC is to have the normal capabilities regarding control of axial equipment in combination with the possibility of implementing advanced pressure based combustion-control logics. The complete system will have the capability of performing rapid prototyping of cycle-to-cycle controllers with the help of Simulink and xPC-Target. The system will also have the capability of implementing control logics with a high enough bandwidth to perform experiments with higher requirements than cycle-to-cycle control, using for example piezoelectric fuel injectors at their full bandwidth.

6

Concluding Remarks

6.1 Summary

New combustion engine principles increase the demand of feed-back combustion control, at the same time economical considerations currently enforces the usage of low-end control hardware limiting the implementation possibilities. Significant development is simultaneously and continuously carried out within the field of Field Programmable Gate Arrays (FPGAs). In recent years FPGAs has developed, from being a device mainly used in to implement grids of 'glue-logic' to something of a flexible 'dream device' in cost and performance sensitive applications. It is not solely the development of FPGA devices which has made the FPGA the promising implementation platform it is. Development of software tool-sets and design methodologies is as important as the development of the device as such.

FPGAs and FPGA design has been thoroughly discussed based on literature found on the topic covering a wide span of considerations. Architectural consideration and corresponding pros and cons have been discussed, both on the FPGA design level and FPGA hardware level, as well as different design tools and design considerations.

Using FPGAs as implementation platform for controllers and feed-back control is a rewarding topic. The literature in the field has been scouted and a number of points especially important for FPGA implementation of control logic, such as word-length, parallelization and high-speed sampling issues have been highlighted. A short survey of application examples from the literature was presented.

Implementation of a heat-release algorithm within an FPGA has been described. Combustion phasing, calculated from the heat-release, is considered as a key feed back variable for closed-loop combustion control. A successful FPGA implementation of a reformulated but completely equivalent

heat-release calculation has been shown. Matlab/Simulink in combination with an FPGA-specific plug-in tool was used to carry out the implementation. The resulting system was capable of computing a heat-release sample within 120ns which is to consider as immediate within the context of combustion engines. System through-put was 50MHz.

Finally a suggestion of a 'rapid-prototype' automotive control system was presented. Rapid-prototyping within automotive control is usually a difficult practical problem. Many of the existing solutions lack the ability to implement control-loops at high bandwidths, especially in combination with complex control logics. The intention with the suggested system is to be able to implement loops at two different bandwidths with different implementation flexibility. Putting an FPGA in the loop it is possible to implement for example extensive model based controllers.

The author hopes that this thesis has illuminated FPGAs as an suitable implementation platform for control applications. Its possibilities, drawbacks and potential application hopefully have been made clear to the reader.

6.2 Future Works

A lot of interesting work remains to be done in this area. One rewarding thing to do would be to make a more complete examination of suitable ways to implement well-known controllers in FPGAs and mixed FPGA/processor systems. How to partition the controllers between hardware and software, finding δ-transform representations of the controllers and the parallelization level are examples of matters which should be examined. Are there optimal implementation structures for controllers and, if so, which are they and which controllers have highest benefits from being implemented on FPGAs. Preferably this kind of study should be illustrated on a number of real control problems.

Another very interesting area waiting to be explored is the topic of handling real-time issues using mixed processor/FPGA systems. Will the traditional real-time handling methodologies still be valid when parts of the logics are FPGA accelerated? Is it possible to utilize the parallelism of FPGAs in some way to increase the utilization of processors without violating the real-time demands? Another dimension on this topic is the very interesting possibility of 'run-time reconfiguration' of FPGAs, intentionally left out of the thesis. Which will the real-time issues be if it is possible to rapidly change the FPGA configuration depending on task?

In the automotive domain a lot can be done, the obvious is to complete the rapid prototype system described in Chapter 5. High bandwidth injection control of Diesel engines, using FPGAs, can be attempted, moving the control horizon from 'between-cycle-control' to 'in-cycle-control'. It is suit-

able to survey the FPGA area using 'automotive-control goggles' to illustrate what is performed in other areas and how these gains can be utilized within automotive control.

Lastly it can be interesting to deploy different automotive-control related models on FPGAs, to off-load the normal ECU or to model things previously regarded as too complex. One example could be an NO_x model implemented using a neural network on an FPGA.

7

Bibliography

Andraka, R. (1998): "A survey of cordic algorithms for FPGAs." In *Proceedings of ACM/SIGDA conference*, pp. 191–200.

Åström, K. J. and B. Wittenmark (1997): *Computer-Controlled Systems*. Prentice Hall.

Beaumont, A., J. Lemieux, P. Battiston, and A. Brown (2006): "Design of a rapid protoryping engine management system for development of combustion feedback control technology." In *SAE*, number 2006-01-0611.

Belletti, F., I. Campos, A. Maiorano, S. Gavir, D. Sciretti, A. Tarancon, J. Velasco, A. Flor, D. Navarro, P. Tellez, L. Fernandez, V. Martin-Mayor, A. Sudupe, S. Jimenez, E. Marinari, F. Mantovani, G. Poll, S. Schifano, L. Tripiccione, and J. Ruiz-Lorenzo (2006): "Ianus: An adaptive FPGA computer." *Computing in Science & Engineering*, **8:1**, pp. 41–49.

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.

Bleris, L. G., P. D. Vouzis, M. G. Arnold, and M. V. Kothare (2006): "A co-processor FPGA platform for the implementation of real-time model predictive control." In *Proceedings of the 2006 American Control Conference*. IEEE.

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.

Chen, R.-X., L.-G. Chen, and L. Chen (2000a): "System design consideration for digital wheelchair controller." In *IEEE Transactions on Industrial Electronics*, vol. 47, pp. 898–907.

Chen, S., R. Istepanian, J. Wu, and J. Chu (2000b): "Comparative study on optimizing closed-loop stability bounds of finite-precision controller structures with shift and delta operators." In *Systems & Control Letters*, vol. 40, pp. 153–163. Elsevier.

Compton, K. and S. Hauck (2002): "Reconfigurable computing: A survey of systems and software." In *ACM Computing Surveys*, vol. 32.

Djemal, r., D. Demigny, and R. Tourki (2005): "A real-time image processing with a compact FPGA—based architechture." *Journal of Computer Science*, **1:2**, pp. 207–214.

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.

Gironés, R. G., R. C. Palero, J. C. Boluda, and C. A. S. (2005): "Fpga implementation of a piplined on-line backpropagation." *Journal of VLSI Signal Processing*, **No 40**, pp. 189–213.

Goodall, R. M. and B. J. Donoghue (1993): "Very high sample rate digital filters using the δ operator." In *IEEE proceedings*, vol. 140.

Goodwin, G. C., R. H. Middleton, and H. V. Poor (1992): "High—speed digital signal processing and control." In *Proceedings of the IEEE*, vol. 80, pp. 240–259. IEEE.

Hall, T. S., C. M. Twigg, J. D. Gray, P. Hasler, and D. V. Anderson (2005): "Large—scale field—programmable analoge arrays for analog signal processing." In *IEEE Transactions on Circuits and Systems*, vol. 52.

Haraldsson, G., J. Hyvönen, P. Tunestål, and B. Johansson (2002): "HCCI combustion phasing in a multi cylinder engine using variable compression ratio." In *SAE*, number 2002-01-2858.

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

Holt, J. L. and J.-N. Hwang (1991): "Finite precision error analysis of neural network electronic hardware implementations." In *Neural Networks, IJCNN-91-Seattle*, vol. 1, pp. 519–525.

Hunt, K. J., D. Sbarbaro, R. Zbikowski, and P. J. Gawthrop (1992): "Neural networks for control systems: a survey." *Automatica (Journal of IFAC)*, **28:6**.

IEEE, P1800, System Verilog Work Group (2001): *1364—2001 IEEE standard Verilog hardware description language*. Number ISBN 0-7381-2826-0.

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

Jones, M., Z. Nakad, P. Plassman, and Y. Yi (2006): "The use of configurable computing for computational kernels in scientific simulations." *Future Generation Computer Systems*, **22:1-2**, pp. 67–79.

Jörg, S., J. Langwald, and N. M. (2004): "FPGA based real—time visual servoing." In *Proceedings of the 17th International Conference on Pattern Recognition (ICPR04)*, vol. 1, pp. 749–752. IEEE.

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

Li, A., Z. Wang, Q. Li, and Y. Wan (2006): "A reconfigurable approach to implement neural networks for engeneering application." In *Proceedings of the 6th World Congress on Intelligent Control and Automation*. IEEE.

Ling, K. V., S. P. Yue, and M. J. M. (2006): "A FPGA implementation of model predictice control." In *Proceedings of the 2006 American Control Conference*. IEEE.

M. Oki, S. M., Y. Toyoshima, K. Ishisaka, and N. Tsuzuki (2006): "180mpa piezo common rail system." In *SAE*, number 2006-01-0274.

Maciejowski, J. M. (2002): *Predictive Control With Constraints*. Prentice Hall.

Martinez-Frias, J., S. Aceves, D. Flowers, J. Smith, and R. Dibble (2000): "HCCI engine control by thermal management." In *SAE*, number 2000-01-2869.

Middleton, R. H. and G. C. Goodwin (1986): "Improved finite word length characteristics in digital control using delta operators." In *IEEE Transactions on Automatic Control*, vol. 31. IEEE.

Milovanovic, N., R. Chen, and J. Turner (2004): "Influence of the variable valve timing strategy on the control of a homogeneous charge compression (HCCI) engine." In *SAE*, number 2004-01-1899.

Monmasson, E. and M. N. Cirstea (2007): "FPGA design methodology for industrial control systems — a review." In *Transactions on Industrial Electronics*, vol. 54. IEEE.

Najt, P. and D. Foster (1983): "Compression-ignited homogeneous charge combustion." In *SAE*, number 830264.

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

Olsson, J.-O., P. Tunestål, B. Johansson, S. Fiveland, R. Agama, M. Willi, and D. Assanis (2002): "Compression ratio influence on maximum load of a natural gas fueled hcci engine." In *SAE*, number 2002-01-0111.

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.

Powell, J. D. (1993): "Engine control using cylinder pressure - past, present and future." *Journal of Dynamic Systems Measurement and Control - Transactions of the ASME*, **2B:115**, pp. 343–350.

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

Russell, S. and P. Norvig (1995): *Artificial Intelligence A Modern Approach*. Number ISBN 0-13-790395-2. Prentice Hall.

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.

Stockinger, M., H. Schäpertöns, and P. Kuhlmann (1992): "Versuche an einem gemischansaugenden mit selbszündung." In *MTZ*, number 53.

Tao, L., M. Zhang, A. Livingston, and V. Asari (2005): "A multi—sensor image fusion and enhancement system for assisting drivers in poor lighting conditions." In *Proceedings of Applied Imagery and Pattern Recognition Workshop (AIPR05)*, pp. 106–113. IEEE.

Tessier, R. and B. W. (2000): "Reconfigurable computing for digital signal processing: a survey." *Journal oc VLSI Signal Processing*, **No 28**, pp. 7–27.

Thring, R. H. (1989): "Homogeneous-charge compression-ignition (HCCI) engine." In *SAE*, number 892068.

Todman, T., G. Constantinides, S. Wilton, O. Mencer, W. Luk, and C. P.Y.K. (2005): "Reconfigurable computing: Architechtures and design methods." In *IEE Proceedings - Computers and Digital Techniques*, vol. 152, pp. 193–207. IEEE.

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, Department of Mechanical Engineering.

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.

Viele, M., L. Stein, M. Gillespie, and G. Hoekstra (2005): "Rapid prototyping an fpga-based engine controller for a 600cc super-sport motorcycle." In *SAE*, number 2005-01-0067.

Wilhelmsson, C., P. Tunestål, and B. Johansson (2006): "Fpga based engine feedback control algorithms." In *Proceedings of the FISITA world automitive congress*, number F2006P039. FISITA.

Wilhelmsson, C., P. Tunestål, and B. Johansson (2007): "An ultra high bandwidth automotive rapid prototype system." In *Proceedings for the Fifth IFAC Symposium on Advances in Automotive Control*, pp. 563–570.

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

Wu, J., S. Chen, G. Li, R. Istepanian, and J. Chu (2000): "Shift and delta operator realisations for digital controllers with finite word length considerations." In *IEE Proceedings, Control Theory Application*.

Yousefzadeh, V., N. Wang, Z. Popovic, and D. Maksimovic (2006): "A digitally controlled dc/dc converter for an rf power amplifier." In *IEEE Transactions on Power Electronics*, vol. 21, pp. 164–172.

Zhao, W., B. W. Kim, A. C. Larson, and R. M. Voyles (2005): "FPGA implementation of closed-loop control system for small—scale robot." In *Proceedings of the 12th International Conference on Advanced Robotics, ICAR '05*, pp. 70–77. IEEE.

A

Abbreviations

ADC	Analog-to-Digital Converter
ALU	Arithmetic Logic Unit
ANN	Artificial Neural Network
ASIC	Application-Specific Integrated Circuit
CA10%	Crank Angle of 10% heat release
CA50%	Crank Angle of 50% heat release
CA90%	Crank Angle of 90% heat release
CAD	Crank Angle Degree
CADP	Crank Angle Degree Pulse (not necessarily with on-degree interval)
CAMP	Charge Amplifier
CO	Carbon monoxide
CO_2	Carbon dioxide
CORDIC	COordinate Rotation Digital Computer
COTS	Commercial Off-The-Shelf
CPLD	Complex Programmable Logic Device
CPS	Cylinder Pressure Sensor
DAC	Digital-to-Analog Converter
DC	Direct Current
DFG	Data Flow Graph
DMA	Direct Memory Access
DSP	Digital Signal Processor
EDC	Engine Dyno Controller
FADC	Framed Analog-to-Digital Converter
FPGA	Field Programmable Gate Array
HC	Un specified hydrocarbon fuel
HCCI	Homogeneous Charge Compression Ignition
HFADC	High Frequency Analog-to-Digital Converter
HR	Heat Release
I/O	Input/Output

JTAG	Joint Test Action Group (standard test access port)
LSB	Least Significant Bit
MPC	Model Predictive Control
Msps	Mega-samples per second
NN	Neural Network
NO_x	Nitric Oxides
PAL	Programmable Array Logic
PID	Proportional-Integral-Derivative (Controller)
SGDSP	System Generator DSP
SoC	System-on-a-Chip
TDCP	Top Dead Center Pulses
VHDL	VHSIC hardware description language
VHSIC	Very-High-Speed Integrated Circuits
x86	the instruction set of the most commercially successful processor architecture defined by Intel in 1978

B

Symbols

β	Over-sampling rate
δ	An operator and transform corresponding
Δ	Sampling time
γ	Specific Heat Ratio
θ	Crank Angle
ADC_{clk}	ADC clock frequency
C_v	Molar specific heat at constant volume
DAC_{clk}	DAC clock frequency
f_{-3dB}	Cut-off frequency, bandwidth
f_{max}^{CAD}	Maximum crank angle pulse frequency
f_{min}^{CAD}	Minimum crank angle pulse frequency
f_N	Nyquist frequency
f_s	Sample frequency
$FPGA_{clk}$	FPGA clock frequency
n_{max}^X	Maximum engine speed using hardware 'X'
n	Number of moles
P_{cyl}	Cylinder pressure
P_{lsb}	Pressure resolution (value of LSB)
P_{max}	Maximum pressure
$P,\ p$	Pressure
Q_{HR}^{net}	Net heat released from combustion
Q	Heat
q	The shift-operator
R	Universal gas constant
T	Temperature
t	Time
U	Internal energy
W	Mechanical work
V	Volume

LUND UNIVERSITY