

DISSERTATION

APPLICATIONS OF FIELD PROGRAMMABLE GATE ARRAYS FOR
ENGINE CONTROL

Submitted by

Matthew Viele

Department of Mechanical Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2012

Doctoral Committee:

Advisor: Bryan D. Willson

Anthony J. Marchese

Robert N. Meroney

Wade O. Troxell

ABSTRACT

APPLICATIONS OF FIELD PROGRAMMABLE GATE ARRAYS FOR ENGINE CONTROL

Automotive engine control is becoming increasingly complex due to the drivers of emissions, fuel economy, and fault detection. Research in to new engine concepts is often limited by the ability to control combustion. Traditional engine-targeted micro controllers have proven difficult for the typical engine researchers to use and inflexible for advanced concept engines. With the advent of Field Programmable Gate Array (FPGA) based engine control system, many of these impediments to research have been lowered.

This dissertation will talk about three stages of FPGA engine controller application. The most basic and widely distributed is the FPGA as an I/O coprocessor, tracking engine position and performing other timing critical low-level tasks. A later application of FPGAs is the use of microsecond loop rates to introduce feedback control on the crank angle degree level. Lastly, the development of custom real-time computing machines to tackle complex engine control problems is presented.

This document is a collection of papers and patents that pertain to the use of FPGAs for the above tasks. Each task is prefixed with a prologue section to give the history of the topic and context of the paper in the larger scope of FPGA based engine control.

The author of this study founded, built up, and eventually sold Driven Inc., a company dedicated to the implementation of FPGAs in engine control. As a result, this study spans a decade of time where we see the first few papers related to FPGA

based engine control, and concludes with FPGA based engine controllers being the de facto standard for advanced combustion research.

ACKNOWLEDGEMENTS

I would like to thank John Silvestri at Gamma Technologies for allowing the use of GT-SUITE over the many years it took to complete this work.

DEDICATION

To my dad who taught me to never give up on a project, no matter how long it takes.

TABLE OF CONTENTS

1	Introduction	1
1.1	The history of electronic engine control with a focus on FPGAs	1
1.1.1	Automotive Microcontrollers	1
1.1.2	FPGA	3
1.2	FPGAs in engine control	5
1.3	Advantages of FPGA based engine controllers	10
1.4	Impact of Drivven FPGA based engine controllers	12
2	Engine Control Co-Processing	16
3	SAE2005-01-0067	19
3.1	Synopsis	19
3.2	Introduction	19
3.3	Overview	21
3.4	Technology Background	22
3.4.1	LabVIEW	22
3.4.2	Compact RIO	22
3.4.3	FPGA	23
3.5	Research and I/O Module Creation	24
3.5.1	Fuel Injector Driver Module	24
3.5.2	Spark Module	27
3.5.3	Input Module	27
3.6	Mapping	28
3.6.1	Mapping Method	29
3.6.2	Data Collection	30
3.6.3	Post Processing	31

3.7	Engine Controller	34
3.7.1	Basic Engine Control Algorithms	36
3.7.2	Speed-Density	36
3.7.3	Alpha-N	37
3.7.4	MAF	39
3.7.5	Transient Compensation	40
3.7.6	Closed Loop	41
3.7.7	Fuel Calculation	41
3.7.8	Transient Fuel Compensation	44
3.7.9	Spark Control	45
3.7.10	Rev Limiter	47
3.7.11	Startup and Idle Control	47
3.7.12	Other I/O	48
3.7.13	User Interface	48
3.8	FPGA Fuel and Spark Control	49
3.9	Conclusion	55
3.10	Acknowledgments	55
3.11	Contact	56
4	ICEF2011-60225	57
4.1	Synopsis	57
4.2	Introduction	57
4.2.1	Why a Free-Piston	58
4.2.2	Breif Free-Piston History	58
4.3	ATS HiPerTEC Engine	60
4.4	Control System Overview	64
4.4.1	Hardware	64
4.4.2	Software	66

4.5	Engine Position Tracking	70
4.5.1	Types of Outputs	73
4.5.2	Free-Piston EPT	74
4.6	Control System Operation	74
4.6.1	Valve Operation and Timing Control	84
4.7	Conclusion	88
4.8	Future Work	89
4.9	Acknowledgments	89
5	Digital Signal Processing and Control	90
6	ICEF2010-35119	92
6.1	Synopsis	92
6.2	Introduction	93
6.3	System Overview	94
6.4	Next-Cycle Control Description	97
6.5	Next-Cycle Results	98
6.6	Same-Cycle Requirement	101
6.7	Heat Release Calculations	103
6.8	Conclusion	113
6.9	Future Work	117
6.10	Acknowledgments	117
7	Custom Computing Machines	118
8	FCCM2012	121
8.1	Synopsis	121
8.2	Introduction	122
8.3	Related Work	125
8.4	Background	127
8.4.1	One-Dimensional Computational Fluid Dynamics	127

8.4.2	Precision Timed Architecture	129
8.5	Design Flow and Architecture	130
8.5.1	System Description and Design Flow	131
8.5.2	System Hardware Architecture	134
8.5.3	Software Design	138
8.6	Experimental Results and Discussion	140
8.6.1	Setup	141
8.6.2	Timing Requirement Validation	142
8.6.3	Resource Utilization	143
8.7	Conclusions and Future Work	145
9	ICES2012-81138	147
9.1	Synopsis	147
9.2	Introduction	148
9.3	Computational Model	150
9.3.1	One-Dimensional Computational Fluid Dynamics	151
9.3.2	Mechanical Subsystem	154
9.3.2.1	Mechanical Equilibrium Governing Equation	154
9.3.2.2	Numerical Integration	155
9.3.2.3	Modeling a Hydraulic Damper	157
9.4	Real-Time Implementation	160
9.4.1	Field Programmable Gate Arrays	160
9.4.1.1	Deterministic Processors	161
9.4.2	System Description	162
9.4.3	System Implementation	164
9.5	Optimization	166
9.5.1	Worst-Case Computation Time	167
9.5.2	Maximize Threads Per Core	167

9.6	Conclusion and Further Work	169
9.7	Acknowledgement	170
10	Patent 7,991,488	171
10.1	Synopsis	172
10.2	Claims	172
10.3	Description	173
10.3.1	Field	173
10.3.2	Background	173
10.3.3	Breif Description of the Drawings	174
10.3.4	Description	177
11	Discussion	223
11.1	Ever Increasing System Complexity	224
11.2	Hardware/Software Architecture for Reusable IP Blocks	227
11.2.1	MPC565 H-Bridge implementation case study	229
11.2.2	Flexible Injector Design	231
11.2.3	No Peripheral Interrupts	231
11.3	Blurring of the Hardware/Software divide	233
11.4	Model based control and model co-processors	235
11.5	System Building Tools	236
11.6	Re-inventing the wheel: the case for open source research software	240
12	Conclusions	242
	Bibliography	243

LIST OF FIGURES

1.1	Example Verilog Code	6
1.2	Example LabVIEW FPGA Code	6
3.1	Finished Motorcycle	21
3.2	RIO Architecture	23
3.3	Prototype Modules	25
3.4	PFI Current Profile	26
3.5	Low Side Switch	26
3.6	Bike glove box with cRIO installed	30
3.7	Accepted Data Points	32
3.8	Fuel Pulse Width Standard Deviation	33
3.9	Factory and Custom ECU installation	35
3.10	Mass air flow	38
3.11	Transient Enrichment	40
3.12	Injector voltage compensation	44
3.13	Spark advance table	46
3.14	Graphical user interface	50
3.15	N-M pattern	52
3.16	N+1 Pattern	52
3.17	Module architecture	54
4.1	Linear Opposed Free-Piston Generator Unit	59
4.2	HiPerTEC Configuration	61
4.3	HiPerTEC 4-Stroke Cycle Positions	61
4.4	8-Chamber HiPerTEC Tested with Control System	62
4.5	Simplified System Layout Diagram	67

4.6	HiPerTEC Installed in Test Cell systems to the left and right	68
4.7	Diagram of Engine Code	69
4.8	Main Control Program Loops and Core Assignment	70
4.9	36-2 (N-M) Camshaft Encoder Pattern	71
4.10	N+1 Camshaft Encoder Pattern	72
4.11	Free-Piston Encoder Pattern.	75
4.12	Engine Position with Cranking, After-Start, and Warm-up Phases Labeled	77
4.13	Engine Position SYNC	78
4.14	HiPerTEC Cranking Phase	79
4.15	HiPerTEC Starter - Pneumatic rotary actuator and release mechanism .	81
4.16	Final Stroke of Cranking	82
4.17	HiPerTEC Warm-up Phase	85
4.18	HiPerTEC Intake Valve	86
6.1	Engine Control Overview	96
6.2	DCAT Data Flow Structure	97
6.3	Mass Fraction Burned Open Loop Control	99
6.4	Mass Fraction Burned Next-Cycle Control	100
6.5	3 bar BMEP, 1800 RPM, 33% EGR	102
6.6	Heat Release Comparison: DCAT with filer, FPGA without Filter	107
6.7	Heat Release Comparison with Filter	108
6.8	Combustion Sensing	109
6.9	Same-Cycle Calculation Flow	110
6.10	Heat Release - After Delay	111
6.11	Injector Current - After Delay	112
6.12	EGR Sweep Open Loop Control	114
6.13	EGR Sweep - Same-cycle Heat Release	115

6.14	EGR Sweep - Same-cycle Injection	116
8.1	First Order Difference	129
8.2	Design Flow	131
8.3	High Level System Diagram	132
8.4	Detailed System Diagram	134
8.5	System of PRET Cores and Interconnects	136
8.6	Execution of Nodes at Each Time Step	139
9.1	GT-SUITE 4-Cylinder Unit Pump Example	149
9.2	GT-SUITE 4-Cylinder Common-Rail Example	150
9.3	First Order Difference	153
9.4	Schematic of Hydraulic Damper	157
9.5	FPGA Size over Time	162
9.6	High Level System Diagram	163
9.7	Detailed System Diagram	165
10.1	Patent Fig. 1A	198
10.2	Patent Fig. 1B	199
10.3	Patent Fig. 1C	200
10.4	Patent Fig. 1D	201
10.5	Patent Fig. 1E	202
10.6	Patent Fig. 2	203
10.7	Patent Fig. 3A	204
10.8	Patent Fig. 3B	205
10.9	Patent Fig. 3C	206
10.10	Patent Fig. 3D	207
10.11	Patent Fig. 4	208

10.12	Patent Fig. 4B	209
10.13	Patent Fig. 4C	210
10.14	Patent Fig. 4D	211
10.15	Patent Fig. 5	212
10.16	Patent Fig. 6A	213
10.17	Patent Fig. 6B	214
10.18	Patent Fig. 7A	215
10.19	Patent Fig. 7B	216
10.20	Patent Fig. 7C	217
10.21	Patent Fig. 7D	218
10.22	Patent Fig. 7E	219
10.23	Patent Fig. 7F	220
10.24	Patent Fig. 8	221
10.25	Patent Fig. 9	222
11.1	Product Development Curve	227
11.2	ECU Pin Reuse Matrix	228
11.3	Simplified H-bridge driver schematic	230
11.4	Comparison of H-bridge implementations	230
11.5	Overview of model optimization process	236
11.6	FPGA model generation process	237
11.7	Model code generation process	239

LIST OF TABLES

3.1	Sensor and Actuator Characteristics	28
3.2	Engineering Data	29
4.1	HiPerTEC Specifications	60
4.2	Control System Components	65
6.1	50% burn duration location (CAD ATDC)	98
6.2	COV of 50% burn location and Peak Pressure	101
6.3	50% burn duration location (CAD ATDC)	113
8.1	Equations for Supported Types	130
8.2	Library of Computational Node Elements	133
8.3	Computational Intensity of Supported Types	141
8.4	Number of Occupied Slices per Core on the Virtex 6 (xc6vlx195t) FPGA.	143
8.5	Total Resource Utilization of Examples Synthesized on the Virtex 6 (xc6vlx195t) FPGA	145
9.1	Equations for Supported Types	154
9.2	Library of Computational Node Elements	165
9.3	Computational Intensity of Supported Types	165
9.4	Number of Occupied Slices per Core on the Virtex 6 FPGA (XC6VLX195T).	168
9.5	Total Resource Utilization of Examples Synthesized on the Virtex 6 FPGA (XC6VIX195T).	168

NOMENCLATURE

Acronym	Description
A/D	Analog to Digital Converter
ARM	Acorn RISC Machines or Advanced RISC Machines
ASIC	Application Specific Integrated Circuit
ASME	American Society of Mechanical Engineers
ATS	Applied Thermal Sciences (company)
ATDC	After TDC
BDC	Bottom Dead Center
BMEP	Break Mean Effective Pressure
BRAM	Block RAM
BTDC	Before TDC
CAD	Crank Angle Degrees
CAT	Caterpillar Corporation
CI	Compressions Ignition
CFD	Computational Fluid Dynamics
CNG	Compressed Natural Gas
COTS	Commercial Off-The-Shelf
COV	Coefficient of Variation
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
CR	Compression Ratio

Acronym	Description
cRIO	Compact RIO
DAQ	Data AcQuisition
DCAT	Drivven Combustion Analysis Toolkit
DI	Digital Input
DI	Direct Injection
DMA	Direct Memory Access
DPF	Diesel Particulate Filter
DRAM	Dynamic RAM
ECU	Engine Control Unit
EGO	Exhaust Gas Oxygen sensor (O_2 sensor)
EGR	Exhaust Gas Recirculation
EPT	Engine Position Tracking
ERC	Engine Research Center
ESD	Electro-Static Discharge
ESHB	Engine Synchronous H-Bridge
FCCM	Field-programmable Custom Computing Machines
FFT	Fast Fourier Transform
FIFO	First In First Out
FPE	Free-piston Engine
FPGA	Field Programmable Gate Array
FTP	File Transfer Protocol
GFI	Gaseous Fuel Injectors
GFLOP	Giga FLoating OPerations per second
GM	General Motors
GNU	GNU is Not Unix

Acronym	Description
GPGPU	General Purpose GPU
GPU	Graphics Processor Unit
GUI	Graphical User Interface
HCCI	Homogeneous Charge Compression Ignition
HDL	Hardware Description Language
HEGO	Heated Exhaust Gas Oxygen Sensor
HIL	Hardware In the Loop
HiPerTEC	High Performace Toroidal Engine Concept
IAD	Included Angle Degrees
IC	Integrated Circuit
I.C.	Internal Combustion (engine)
IEEE	Institute of Electrical and Electronics Engineers
IIR	Infinite Impluse Response
IIT	Indian Institute of Technology
ILP	Integer Linear Programming
IMEP	Indicated Mean Effective Pressure
IP	Intellectual Property
IO	Input / Output
ISA	Instruction Set Architecture
LTC	Low Temperature Combustion
LUT	Look-Up Table
MAC	Multiply And Accumulate
MAF	Mass Air Flow
MAP	Manifold Air Pressure or Manifold Absolute Pressure
MAT	Manifold Air Temperature or Manifold Absolute Temperature

Acronym	Description
MBT	Minimum advance for Best Torque
MFB	Mass Fraction Burned
MHz	Million cycles per second
NI	National Instruments
NOP	No OPeration
OEM	Original Equipment Manufacturer
ORNL	Oak Ridge National Laboratory
OS	Operating System
PC	Personal Computer
PCB	Printed Circuit Board
PCCI	Premixed Charge Compression Ignition
PCI	Peripheral Component Interconnect
PCP2	Peripheral Control Processor 2
PFI	Port Fuel Injection
PID	Proportional Integral Derivative
PLA	Programmable Array Logic
PLD	Programmable Logic Device
PRET	PREcision Timed
PWM	Pulse Width Modulated
PXI	PCI eXtensions for Instrumentation
RAM	Random Access Memory
RCCI	Reactivity Controlled Compression Ignition
RIO	Reconfigurable IO
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory

Acronym	Description
RPM	Revolutions Per Minute
RT	Real-Time
SAE	Society of Automotive Engineers
SI	Spark Ignited
SIMD	Single Instruction - Multiple Data
SPI	Serial Peripheral Interface
SwRI	Southwest Research Institute
TDC	Top Dead-Center
TDI	Turbo Diesel Injection
TI	Texas Instruments
TPU	Time Processing Unit
UEGO	Universal Exhaust Gas Oxygen sensor
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuits
VE	Volumetric Efficiency
VR	Variable Reluctance
VVA	Variable Valve Actuation
VW	Volkswagen

Chapter 1

INTRODUCTION

Modern engines are increasingly dependent upon complex control systems to achieve their performance and emissions goals. Following Moore's law for ever increasing performance, engine controllers have become ever more complex and capable. In the span of a few decades engine controller units (ECUs) moved from 8-bit controllers with low on-chip integration to modern 32-bit floating point processors. Presented herein is a logical next step in ECU evolution, using a field programmable gate array (FPGA) as the core of the ECU allowing both more flexibility and new capabilities in ECU design.

As a result of, and enabled by, this complexity engine controller design now supports higher level of abstraction. This means that very high level symbolic and dataflow languages are used to define the control, and that formal layering is employed in the software to abstract the application programmer from the hardware.

1.1 The history of electronic engine control with a focus on FPGAs

1.1.1 Automotive Microcontrollers

Automotive microcontrollers differ from industrial microcontroller in a few major respects. First is temperature, automotive grade parts are generally -40C to 125C, as opposed to industrial parts that are generally -40C to 85C. A more important difference is in the built-in peripherals. Automotive microcontroller have specialty I/O for automotive specific tasks like dedicated protocol engines or the TPU and PCP2 units described below.

Microcontroller-based electronic engine controllers first started appearing in the late 1970s. By the early 1980s microcontrollers were the standard for engine control.

In 1981 the entire Domestic market GM fleet was microcontroller based. (Bereisa, 1983) These were 8-bit controllers of the class that the Motorola 6800 exemplifies. They implemented throttle body injection, spark advance, and EGR generally through lookup tables.

In 1983 Motorola introduced the 68HC11 with numerous on-board peripherals. This processor and its cousins throughout the industry enabled consolidation of numerous discrete components into a single chip. These, however, remained slow 8-bit processors that were largely coded in assembly language.

By 1990 the Motorola 68332 was released for the automotive market. This was an automotive variant of the popular 68000 series processor used in the Apple Macintosh of the day. This processor was a full 32-bit processor enabling the use of higher level programming languages like C. One major innovation was the Time Processing Unit (TPU).

The TPU offloaded the processor from handling timing critical tasks like fuel and spark commands. It allowed easier abstraction of the higher level software by reducing the intensity of interrupts to the main processor. This in turn simplified the task of implementing multi-port fuel injection. The TPU itself was a special purpose processor with special purpose instructions allowing it to do complex counter-timer operations in single instructions. It was not particularly fast, running at some fraction of the processor clock, but the ability to do quick timer manipulations without CPU intervention allowed complex control of multi-port injection systems. A similar system was offered by Infineon with its Peripheral Control Processor (PCP2) in its Tri-Core processor line.

Both the TPU and PCP2 were programmed using proprietary assembly languages. Given the concurrent nature of the I/O coprocessing tasks, these proved difficult to program and debug. This timing coprocessor approach remains the state of the art in high volume production ECUs to this day.

By 2000 Motorola released its PowerPC variant of automotive microcontroller in the MPC5xx family. This was an evolution of the 68332, but with a PowerPC core and a larger mix of peripherals. The big advance in this processor was the hardware floating point allowing model-based control (called algorithmic control in other industries) to be developed without the need of complex integer math overflow checking.

1.1.2 FPGA

Digital systems, consisting of boolean logic and latches, are the basic building blocks of modern processors. These systems may be discrete gates and latches in their own ICs soldered down to a circuit board or a collection of logic grouped in to a single chip. Chip manufacturers can group these logic gates in to large systems creating things like microprocessors. In fact the prefix “micro” denotes that the majority of the processing functions are grouped in to a single IC. Still, nearly every modern digital system consists of a collection of major components like processors and some special purpose peripheral. These are usually connected using “glue logic”, a set of logic gates the engineer uses to interface one component to another.

Starting in the late 1960s IC companies started making “generic” logic ICs that could be custom configured at the factory. These allowed engineers to group their “glue logic” or other complex function in to a single IC assuming that their design could fit in the number of gates and other constraints of the parts. Because these were programmed at the factory they were only applicable to high volume applications.

In 1975 Harris Semiconductor, renamed Intersil in 2000, introduced the IM5200, the first programmable logic array (PLA) programmable by the user instead of the factory. These PLA still needed to be programmed in an external programming fixture before being installed on a printed circuit board. Programmable devices

in this era were on the order of 10 inputs and 10 outputs with any combination of AND/OR logic between the inputs and outputs with a latch at each output and feedback from outputs in back in to the logic array. Programming was done in PALASM, a simple low level programming language. These were practical for implementing complex state machines, but did not have enough internal memory to implement large numbers of counters and timers.

In the mid-1980s the number of available gates was increased by the introduction of the Complex Programmable Logic Device (CPLD). These were effectively multiple PLAs tied together in a single IC. The downside of this is that the symmetry was gone and gate level mapping of the device by the programmer became much more complex. Programming languages like ABEL were introduced to help program these devices. Similarly, designs could be created in “Schematic Capture” just like PCBs and hierarchies of designs with mixed schematic and text logic were possible.

Field Programmable Gate Arrays (FPGAs) accomplished similar objectives as CPLDs with some noted differences in approach. FPGAs can be thought of as being RAM-based and need to be programmed at bootup as opposed to ROM-based PALs and CPLDs that have their programs active once power is applied. FPGAs operate faster and have much higher densities. In the early 1990s FPGAs could be had with tens of thousands of gates. By early 2000s devices with a million gates were available. Today’s FPGAs have as many as 20 million gates.

New programming languages were developed in order to handle such large designs. VHDL and Verilog are the major text-based languages used in FPGA design. Both have their origins in Application Specific Integrated Circuit (ASIC) design flow. Because ASICs can cost millions of dollars for a prototype run and have multi-month design turns there is a huge emphasis on getting it right the first time.

As such, both languages have built-in simulation constructs to allow the designer to simulate part or all of the design.

VHDL and Verilog are complex and specialized languages with learning curves that make them impractical for casual use. VHDL and Verilog are, if care is taken with linked libraries, device and supplier independent languages. These languages, along with their more primitive cousins, are known as Hardware Description Language (HDL). Numerous projects have been implemented to provide a higher level of abstraction. C to HDL has been done numerous times with limited acceptance because the sequential nature of C is in direct conflict with the parallel nature of the HDLs. Figure 1.1 shows a snippet of Verilog code.

Two proprietary high-level languages have been well adapted to FPGA code generation. Simulink by The Mathworks and LabVIEW by National Instruments are dataflow languages that can exploit inherent parallelism of the FPGA. These languages are used widely outside of the FPGA scope by system level programmers like those in engine research. An example of LabVIEW FPGA code is shown in Figure 1.2

FPGA code described in the case studies presented in this document are either Verilog or LabVIEW FPGA. Verilog is used when re-use and device independence is required, or when the target FPGA is embedded in a subsystem or module. LabVIEW FPGA is used for system level or application level design.

1.2 FPGAs in engine control

A survey of the literature shows that while FPGAs were mentioned in relation to automotive applications as early as the mid-1990s these were mostly in the areas of network communications. Karen Parnell from Xilinx (Parnell-Xilinx, 2002) outlined the general course of FPGAs in production applications to date. Specifically their

```

// *****
// FX2_CLKOUT Domain
// *****

// Infer Dual Port Distributed RAMs for FX2DataPacket.
// This is just one side which does the writing.
// We are assuming that no writing will be performed to this memory
// while we are reading from this memory in the DART_SCI_CLK domain.
always @(posedge FX2_CLKOUT) begin
    if (!FX2_WR && FX2_A15) begin
        FX2DataPacket[FX2_ADDRESS] <= FX2_DATA;
    end
end

// Generate UartTrigger so that the slower DART_SCI_CLK can pick it up.
// FX2_CLKOUT is faster than DART_SCI_CLK.
always @(posedge FX2_CLKOUT) begin
    if (!FX2_WR && FX2_A15 && (FX2_ADDRESS == (UART_PACKET_SIZE - 1))) begin
        UartTriggerTimer <= 0;
    end
    else if (UartTriggerTimer < UART_TRIGGER_TIME) begin
        UartTriggerTimer <= UartTriggerTimer + 1;
        UartTrigger <= 1;
    end
    else begin
        UartTrigger <= 0;
    end
end

// Tickle watchdog on WatchdogTickle rising one shot
always @(posedge FX2_CLKOUT) begin
    if (WatchdogTickle_ROS) begin
        WDTimer <= 0;
        WDTIMEOUT <= 0;
    end
    else if (WDTimer > WDTIMEOUT) begin
        WDTIMEOUT <= 1;
    end
    else begin
        WDTimer <= WDTimer + 1;
    end
end

```

Figure 1.1: Example Verilog Code

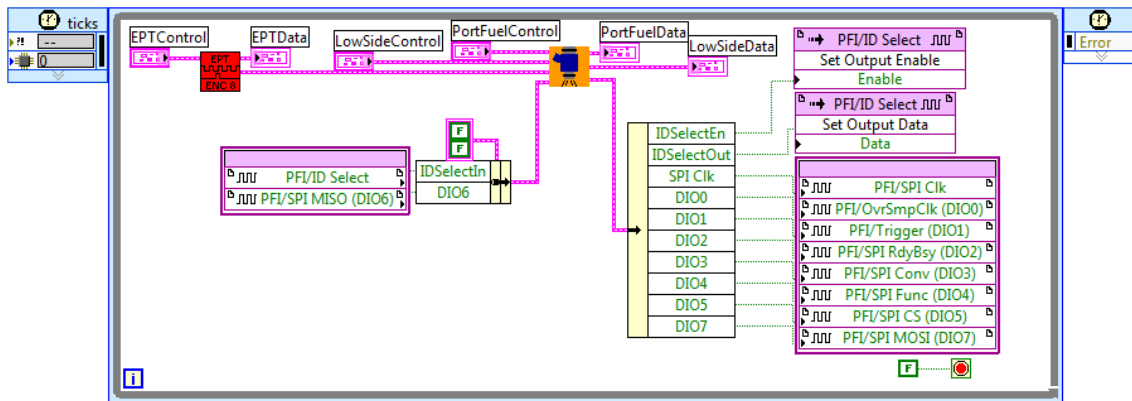


Figure 1.2: Example LabVIEW FPGA Code

large employment in infotainment (in cabin electronics like radio, navigation, cell phone, etc.) and interface components.

The 2004 SAE paper by this author (Viele et al., 2004) titled, “A PC and FPGA Hybrid Approach to Hardware-in-the-Loop Simulation,” is the first SAE paper to feature an FPGA as the main I/O processor for an HIL system. It outlined using an FPGA to generate all the engine synchronous signals that were consumed by Woodward’s various production ECUs. It also captured critical features of all the key engine control outputs.

A similar effort was published the year later by TVS Motor (Rohini et al., 2005). This paper compared a generic Digital Signal Processor board, a dSpace Hardware in the Loop (HIL) simulation machine, and National Instruments FPGA based architecture similar to the one used in (Viele et al., 2004). Ultimately they chose the FPGA based approach because of both the flexibility of programming and because that flexibility led to reduced hardware, and therefore, cost savings.

Opal-RT, dSpace, A&D Technology, and ETAS all produced papers along the same theme over the next few years. Now FPGAs are the de facto standard for I/O boards in HIL applications, being offered by all major HIL vendors as the basis for academic and industrial projects.

The included paper, “Rapid Prototyping an FPGA-based Engine Controller for a 600cc Super-Sport Motorcycle,” is the earliest published SAE reference to using an FPGA as a major reconfigurable part of the system architecture. The contribution of this paper is described in detail in Section 2.

Outside of Drivven, a company this author founded to promote FPGA based engine controller, there are a few major clusters of FPGA based engine control research. Major engine control services companies dSpace (Schuette et al., 2005), A&D (Jiang et al., 2007), Ricardo (Beaumont et al., 2006), and SwRI (Wang and Sarlashkar, 2007) all have FPGA based engine control efforts to some degree. The

SwRI platform is an outgrowth of the FPGA based prototyping platform this author initiated when he was employed there in the late 1990s.

Delphi released an interesting paper on an FPGA based controller for cylinder pressure feedback development. (Schten et al., 2007) This attempted to span the gap between research hardware and production hardware. This was a fully embedded box capable of real-time cylinder pressure analysis.

There are a few academic projects that built ground-up FPGA based engine controllers. These have tended to be student projects with single publications. For instance, a team of Brazilian students built a low cost FPGA based controller for student competitions and described it in (D'Angelo et al., 2006). In this setup they mirror the same three-level architecture that is outlined in the first included paper with a host PC, a real-time controller, and an FPGA to do engine position tracking.

The majority of FPGA based engine research work uses the same National Instruments hardware and software that Drivven uses, but with in-house developed engine position tracking software and sensor/actuator interfaces. Examples of this can be found at IIT Madras(Padala et al., 2008) and Texas Tech (He et al., 2005). The three major centers of independent FPGA based engine controller research are University of Bologna, Lund University, and the University of Windsor.

Enrico Corti's group at the University of Bologna focuses on FPGA based engine control to optimize combustion phasing, primarily in high-performance SI engines. In (Corti and Soller, 2005) they look at an engine control with integrated combustion analysis based on the same NI FPGA platform Drivven uses. This first paper describes the framework that they used in a number of subsequent papers on combustion feedback and control. They normally use the higher power PXI-based controllers that have the processing power to analyze combustion analysis signals online. They were used in (Corti et al., 2007) and (Corti et al., 2008b). For these systems the FPGA is primarily used to synchronize with the engine.

In (Corti et al., 2008a) they transitioned to the Drivven FPGA software for low-level engine synchronous I/O control and FPGA based drive electronics. This paper describes a tractor pull engine. While not specifically mentioned in the paper, the related videos (<http://it.youtube.com/watch?v=g2fevZVI5uk>) show the tractor with advanced control not only won the contest, but did it with little black smoke.

In (Corti and Forte, 2011) and (Cavina et al., 2011) they use the FPGA based architecture to implement cycle-to-cycle control of high-performance SI engines.

Lund University is another major center for the independent development of FPGA based engine controllers led by Per Tunestl and Bengt Johansson. They used NI based FPGA hardware initially to control a Variable Valve Actuation (VVA) system (Trajkovic et al., 2006). The FPGA read engine position from an optical encoder and generated commands to the VVA system. This work was extended in (Persson et al., 2007), (Trajkovic et al., 2007), and (Trajkovic et al., 2008) continuing to use the FPGA base controller.

This same controller concept was applied to a broader range of controlled (Persson et al., 2008), (Wilhelmsson et al., 2009), and (Borgqvist et al., 2011).

Professor Zheng at the University of Windsor has taken the approach of using many NI Compact RIO FPGA based devices through his lab to perform various functions of control and data acquisition. While this may be sub-optimal from a system perspective, it allowed him to have many students tackling many smaller problems. In (Zheng et al., 2006) he examines an adaptive fueling strategy using FPGA based controls to control fuel, boost, EGR, and other engine parameters.

His numerous publications in this area leverage the common theme of FPGA based engine controls allowing flexibility and rapid control prototyping.

(Kumar et al., 2007) (Zheng et al., 2007) (Kumar and Zheng, 2008)
(Asad and Zheng, 2008) (Zheng et al., 2009a) (Zheng et al., 2009c)
(Asad et al., 2009) (Tan et al., 2009) (Asad and Zheng, 2009)
(Zheng et al., 2009b)

Delphi has written a pair of important papers on the practical uses and limits of FPGA in production vehicles, (Lumpkin and Gabrick, 2006) and (Gabrick et al., 2006). These have been on the practical aspects of packaging and reliability of FPGAs, not just their software applications.

To date FPGA adoption in volume production engines has been limited by two major factors. The first is part cost. The cost of an automotive microcontroller continues to be less than the cost of an FPGA that can perform the requisite tasks. While the ease of programming FPGAs reduces the non-recurring engineering costs, the part price difference for multi-million unit runs is still greater. The second is temperature. Automotive microcontrollers operate up to a package temperature of 125C, where Xilinx and Altera FPGAs can only support a 125C junction temperature. While the relation of junction to package temperature is dependent upon how much heat the FPGA is generating, a rule of thumb is 10 to 20 degrees C. Nonetheless, defining the specifications for their new engine platform, Lamborghini called for FPGAs in their engine control. (Ceccarani et al., 2005)

1.3 Advantages of FPGA based engine controllers

We are starting to see FPGAs in low volume applications like controllers for stationary engines and motorsports. There are a few factors in these applications that overcome the cost and temperature limitations.

First is flexibility. An FPGA based engine controller has the ability to be changed to suit different applications without hardware changes. This is exemplified

in the paper presented in 4. This is even more important in a non-modular ECU where changes to the I/O require a completely new controller.

The second advantage is portability and layering. When an FPGA goes obsolete or code needs to be migrated to a different ECU, the FPGA code is fully portable to a different FPGA. The device drivers can be identical for the real-time code. This is in contrast to a traditional processor where the I/O is different in each processor family and generation. This difference means major rewriting and testing of I/O software, something that can take man-years.

Another advantage is system architecture. As ECUs transition to higher level languages like Simulink and LabVIEW they are less able to handle interrupt events because these disrupt the dataflow. Similarly modern processors with large caches, branch prediction, etc., take a larger performance hit for each interrupt than their less complex predecessors. The ability to drive many small engine synchronous or periodic tasks to the FPGA reduces the number of interrupts to the real-time CPU allowing the code to be simpler and fit better to dataflow languages.

The paper in Section 2 demonstrates a fully functioning FPGA based engine controller, the first SAE paper published on this topic. However, it is not the first FPGA based engine controller, since it is unknown which product that was because internal ECU architecture is not often published. A system, similar in concept, was developed by Viele and Dase during their tenure at SwRI some years before, but the FPGA software was entirely different in concept and execution and did not support interpolation or other critical components of a proper ECU time coprocessor. Through opening up and inspecting a series of ECUs, we know that FPGAs have seen production in low volume, off-road applications as well as race ECUs, but the exact nature of their function is unknown to this author.

1.4 Impact of Drivven FPGA based engine controllers

The company Drivven was formed in the mid-2000s by Matthew Viele and Carroll Dase. The company's goal was to take engine specific FPGA IP and I/O hardware and combine it with commercial off the shelf (COTS) National Instruments FPGA Hardware and Software, to create a complete engine rapid control prototyping environment for IC engines. A secondary goal was to market FPGA IP to engine control OEMs to use in volume production vehicles, but they were hampered by the slow adoption of FPGAs by the likes of Bosch and Delphi due to the thermal and cost limitations outlined above.

The advantage of FPGA based controllers over more conventional setups involving processors and counter/timer boards was its symmetry and modularity of I/O combined with the flexibility of the FPGA for configuration. Similar modular controllers were available from dSpace and ETAS in this time frame, but both of them had strong restrictions on configurations of cam/crank patterns and I/O mix requiring the researcher to build some sort of custom interface for unconventional engines.

As a result, Drivven's work in simplifying FPGA based engine controllers, they have been widely adopted. Since the first Drivven deployment in a research setting in 2006, there have been around 100 publications based on the research made possible by these controllers. A sampling of the published research based upon these controllers is below. All of these projects use an FPGA based engine position tracking software and at least one I/O module that uses an FPGA to precisely control I/O.

Delphi has a number of modular control systems to do both Gasoline Direct Injection (Sellnau et al., 2012b), (Sellnau et al., 2012a), (Sellnau et al., 2011) and Diesel (Yun and Sellnau, 2008) work. In addition to the combustion work, they

worked with this author to develop a new method of measuring trapped burned gas residual fraction. This author jointly published (Sellnau et al., 2009) based upon the earlier analytical work and experimental data from (Sinnamon and Sellnau, 2008).

PTT, the Thai national oil company, has a system to run dual fuel diesel/CNG engines. They have documented their research (Chatlatanagulchai et al., 2010c), (Chatlatanagulchai et al., 2010d), (Chatlatanagulchai et al., 2011b), (Chatlatanagulchai et al., 2011a), (Chatlatanagulchai et al., 2010b), (Chatlatanagulchai et al., 2010a), and (Yaovaja et al., 2011).

Steve Ciatti at Argonne National Lab has written, in addition to the joint publication included here (Viele et al., 2010), a number of papers using gasoline-like fuels in a diesel engine, (Subramanian and Ciatti, 2011), (Ciatti and Subramanian, 2011), (Ciatti et al., 2010) using the FPGA based controller.

The National Research Council Canada has a novel adaptation of a single cylinder CAT engine. They published PCCI work in (Dumitrescu et al., 2010).

Oak Ridge National Laboratory has nearly every test cell running a Driven controller. They have published experiments ranging from spark-assisted HCCI (Weall and Szybist, 2011) to ethanol CI combustion (Szybist et al., 2011), (Szybist et al., 2010a), to University of Wisconsin style RCCI (Cho et al., 2011), (Curran et al., 2010), (Wagner et al., 2011), (Curran et al., 2011a), (Curran et al., 2011b), (Briggs et al., 2011) to 6-stroke engines (Szybist et al., 2010b).

As mentioned above, the group from University of Bologna used a Driven controller to implement a tractor pull controller (Corti et al., 2008a).

The University of Wisconsin ERC has a number of projects running Driven controllers and has been prolific in publishing research based upon these controllers. Without going into details, the following is a list of technical papers, presentations, and theses based on the Driven FPGA based controller work.

(R.Kaddatz, 2011)	(Kim, 2009)	(Swor, 2009)
(Dunbeck, 2009)	(Staples, 2008)	(Kokjohn et al., 2011a)
(Kim et al., 2010)	(Dunbeck and Reitz, 2010)	(Swor et al., 2010)
(Kokjohn et al., 2011b)	(Kokjohn et al., 2010)	(Kim et al., 2009)
(Sung et al., 2009)	(J et al., 2009)	(Kokjohn et al., 2009)
(Hanson et al., 2010)	(Splitter et al., 2010)	(Splitter et al., 2010)
(Kokjohn and Reitz, 2010)	(Hanson et al., 2011)	(Splitter et al., 2011)
(Tess et al., 2011)	(Staples et al., 2009)	

Tim Jacob's group at Texas A&M has a Deere diesel and a GM 1.9L diesel with Drivven controllers. His focus is on biodiesel HCCI. He and his students have published the following.

(Bittle et al., 2011b)	(Bittle et al., 2010c)	(Knight et al., 2010)
(Bittle and Jacobs, 2011)	(McLean and Jacobs, 2011)	(McLean and Jacobs, 2011)
(Song et al., 2011)	(Tompkins et al., 2011)	(Bittle et al., 2010b)
(Bittle et al., 2010a)	(Jacobs, 2011)	(Tompkins and Jacobs, 2011)
(Evans et al., 2010)	(Schmidt et al., 2011a)	(Schmidt et al., 2011b)
(Bittle et al., 2011a)	(Knight et al., 2011)	

Wayne State uses the FPGA to trigger a camera and laser system in conjunction with the FPGA. This system can closely control the phase of the injection with that of the camera and laser. They have published a number of studies based upon this system.

(Florea et al., 2012)	(Zha et al., 2012)	(Yu et al., 2012)
(Zha et al., 2011)	(Zha et al., 2010)	(Zha and Jansons, 2011)
(Jansons et al., 2011)	(Jansons et al., 2010a)	(Jansons et al., 2010b)
(Jansons et al., 2009)		

Richard Stobart's group at Loughborough University has published a bit of work on advanced control of heavy duty diesel engine fuel systems using Drivven systems (Deng et al., 2010) (Deng et al., 2011) (Winward et al., 2010).

University of Michigan recently commissioned a system and has published (Manofsky et al., 2011).

Alexander Sappok published about his DPF sensor (Sappok et al., 2010) using a Drivven controller at ORNL.

Curtis Stovall (Stovall, 2008) and Kevin Norman (Norman, 2007) both published master's thesis from Colorado State University using early Drivven controllers.

While this list is not exhaustive it is intended to show some of the flexibility of these controllers and the impact they have had on the engine research community.

Chapter 2

ENGINE CONTROL CO-PROCESSING

The basic FPGA engine controller architecture consists of three parts: a real-time processor that performs engineering calculations; an FPGA to perform time and angle critical I/O functions; and host display that can be used to view and tune parameters.

The real-time processor is responsible for carrying out a number of state machines, equation solvers, PID control loops, and lookup tables. The fastest control loops execution frequency is on the same order as crankshaft rotation frequency, with optional slower loops for functions with longer time constants if processor speed is insufficient. A $5ms$ loop is used in all the included papers for the real-time loop execution.

The real-time loop first converts data from FPGA counts to engineering units. It then performs the necessary calculations and determines the desired outputs. Finally, it converts the data back to FPGA counts and writes the data to the FPGA for execution.

The FPGA operates on a much faster timescale, in our case, 40MHz. With every FPGA loop the FPGA performs all of its tasks in parallel. These tasks include:

- Read cam and crank signals.
- Track engine position by referencing the last known cam/crank pulse and extrapolating based on engine speed.
- Feed current engine position to all the engine synchronous sampling and output blocks.

- The engine synchronous output blocks like fuel or spark monitor the engine position and commands from the real-time code to decide their current state.
- Pulse Width Modulated blocks update their state based on frequency and duty cycle.
- Module I/O blocks communicate between the main FPGA and logic in the I/O modules to send parameters or get status.

Complicated I/O blocks in the FPGA allow the processor to run with little interruption. For instance, let's consider a speed input block for reading shaft speed. The input needs to not only measure the time between pulses with sufficient resolution, but also detect overflow in the event of a stopped pulse and pick up immediately when the shaft starts moving again. It may also need to reject glitches caused by ESD or spark noise. In a microprocessor application this may require interrupts to the CPU to process all the different exceptions as well as glitch-free pre-scaler changing if the dynamic range is too large. In the FPGA the bit resolution of the timer can be adjusted as well as providing flags to handle the overflow and restart cases. Also a shift filter or up-down counter can be placed before the speed input to reject noise glitches and tuned in software.

Lastly, the host interface allows display of indicators from the controller as well as updating of controls. Data types supported include floating point, integer, arrays, lookup tables, boolean and enumerated types. The controller should be able to connect and disconnect from the host PC. Calibrations should be able to be saved, loaded, and version controlled with differencing. This is accomplished through Drivven's CalVIEW software which is similar in functionality to ETAS INCA or dSpace ControlDesk.

FPGAs were also used in the modules themselves. The main system FPGA sends configuration commands to the modules and they, in turn, run through the

appropriate sequence of actions when a fuel or spark command is sent from the LabVIEW FPGA.

The following paper was used to summarize Drivven's first major project. This was done when Drivven consisted of this author and Carroll Dase. The paper outlines using an FPGA based controller first to map an engine then later using the same FPGA to control the engine.

This paper covers the first implementation of Drivven's Engine Position Tracking as well as the initial publication of Drivven's FPGA based engine I/O modules.

This paper was presented at the 2005 SAE Congress and won an award for best presentation of the session.

For this paper the specific contributions of this author are:

- The concept and initial implementation of engine position tracking in an FPGA.
- The design and build of the FPGA based cRIO modules used to generate fuel and spark signals.
- The high-level algorithms for engine control.

Reprinted with permission from SAE Paper No. 2005-01-0067 (c) 2005 SAE International. Further use or distribution is not allowed without permission from SAE.

Chapter 3

SAE2005-01-0067

Rapid Prototyping an FPGA-based Engine Controller for a 600cc Super-Sport Motorcycle.

The paper was written by Matthew Viele and Carroll Dase of Drivven, presented at SAE Congress 2005

3.1 Synopsis

Two main goals exist for prototype engine control systems. One goal is to research specific areas of engine control or behavior such as fuel delivery or exhaust emissions. Another goal is to prototype an engine controller for aftermarket applications or Original Equipment Manufacturer (OEM) production. In either case, engineers often face the challenge of creating a prototype controller for an already existing OEM engine with little or no knowledge of the control strategies embedded within the OEM controller. This paper will discuss an FPGA-based system used to map the behavior of an OEM controller as well as function as the prototype controller. The FPGA was used to track the angular position of the crankshaft and generate fuel and spark control signals synchronously to the rotation of the crankshaft, as well as to acquire analog and digital sensor data.

3.2 Introduction

Modern engines rely on sophisticated electronics and software. Production engine control units (ECUs) read sensors, drive actuators, and execute the algorithms to control the engine. These ECUs are highly specialized devices that perform their

tasks with as little extra hardware as possible to minimize cost. However, this efficiency makes them poor systems for research and development. Therefore, research oriented ECUs are created, having much more computing power and I/O flexibility than production ECUs, but with a much higher cost and larger size.

At Drivven, Inc., we used the CompactRIO (cRIO) platform from National Instruments to develop such a research ECU for a Yamaha YZF-R6 super-sport motorcycle, shown in Figure 3.1. We applied our FPGA IP to control fuel and spark and designed cRIO-compatible hardware modules for interfacing to the various sensors and actuators.

Before the cRIO could be used to control the motorcycle, a base control application needed to be developed and calibrated for many different operating points. A common method of calibration is to install the engine on a dynamometer and optimize control parameters at hundreds of steady-state and transient operating points. If the engine is a production engine, the OEM has already performed this calibration with great accuracy and at great expense. When engineers are performing research with an OEM engine (as in our case here), it is unlikely that the calibration data will be available to them. Therefore, it may be advantageous to record sensor and actuator behavior while running the engine under control of the OEM ECU. This exercise is called controller mapping and can be much less time consuming than calibrating control parameters from scratch. It is worth noting that mapping exercises can be much more reliably performed on a motorcycle ECU, in which most of the control algorithms are open-loop, than on an ECU found in a modern automobile, where many of the control algorithms are closed-loop.

Once the mapping exercise was completed, the data was reformatted to fit the engine control application developed for the cRIO. The cRIO then was wired to a replacement ECU connector and took over control of all tasks performed by



Figure 3.1: Finished Motorcycle

the OEM controller. The cRIO became a full-authority engine and vehicle control system.

3.3 Overview

This project was broken up into three major phases. The first phase involved researching the specifics of how the bike worked and how we would control it. It also required the manufacture of custom hardware.

The second phase involved mapping of the motorcycle ECU. This mapping involved tapping in to every signal we felt relevant and logging its behavior over the full operating range. In this phase the OEM controller was still running the bike while the cRIO was just monitoring the I/O signals.

The last phase involved actually taking over control using our software and FPGA IP implemented on the cRIO. The factory ECU was removed and the cRIO connected to the ECU mating connector.

3.4 Technology Background

Before we discuss the project further, it is important to understand the technologies involved. With the exception of the top level engine management software, FPGA IP, and I/O modules, the entire project was carried out using National Instruments COTS hardware and software products. This platform has shown to be a worthy alternative to custom-developed hardware, or products from companies such as ETAS and dSPACE. (Tsai et al., 2003)

3.4.1 LabVIEW

LabVIEW is a graphical programming language that has been available since the 1980s. It evolved from its roots in process automation to a number of other engineering and scientific markets. Unlike Simulink LabVIEW is very focused on the data acquisition and control of real world, real-time I/O, whereas Simulinks core is a simulation environment. However, their market segments have overlapped significantly in recent years, and both packages interface well with each other. (Aceituna, 2002)(Kulkarni and Hoekstra, 2002)(Viele et al., 2004)(Cummings, 2003)

3.4.2 Compact RIO

The Compact RIO or cRIO is a new product for National Instruments. It combines a Pentium class processor with a large Xilinx FPGA in a rugged chassis with locations for plug in I/O modules. The modules interface directly to the FPGA, allowing a very generic module interface and rapid module development. This topology is shown in Figure 3.2.

The cRIO always runs two programs in parallel. One is a real-time control application running on the processor, written with LabVIEW RT. The other is a FPGA application written in LabVIEW FPGA. Good design practices for this ar-

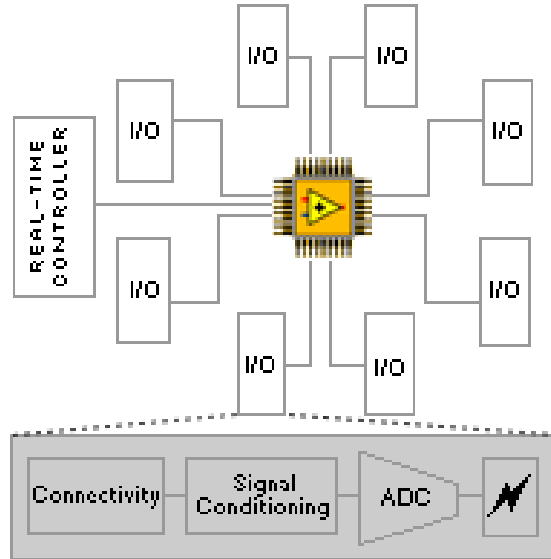


Figure 3.2: RIO Architecture

chitecture (and something you will hear us repeat often) dictate that the real-time program be made as simple and slow as possible, with loop times in the several milliseconds. The FPGA program should handle all the high speed, timing-critical I/O tasks and have data ready to present to the real-time application when requested.

3.4.3 FPGA

An FPGA or Field Programmable Gate Array is a device to implement any sort of digital logic. It can be considered as a large fabric of AND gates, OR gates, and Flip Flops. A large enough collection of gates can be used to implement any digital circuit including I/O, communication bus interfaces, and even entire microprocessors.

Two traditional methods exist to create FPGA designs. They are the text-based programming languages VHDL and Verilog. These languages are roughly analogous to C for microprocessors. They, along with other similar languages, are collectively called HDL or Hardware Description Languages. HDLs are quite powerful but also require high levels of expertise to program.

Higher-level tools are available to simplify this practice. These include The Xilinx System Generator Toolkit for Matlab, Celoxicas various C compilers, and LabVIEW FPGA. For our project, LabVIEW FPGA was used because of the seamless integration with the cRIO toolchain. All of the low level FPGA code, or IP cores, are written in Verilog and wrapped in a graphical LabVIEW FPGA “HDL Node” block, allowing the code to be used in non-LabVIEW systems like production engine controllers.

3.5 Research and I/O Module Creation

Because the cRIO was months from being released for sale when this project was started, no modules were available for us to use – even for the mapping exercise. Even if modules were available, unique automotive requirements are unlikely to be met by commercial off the shelf hardware.

Existing I/O designs from previous projects were repackaged and upgraded to provide robust automotive modules specifically for the cRIO platform. The motorcycle required a fuel injector driver module, a spark driver module, and an analog and digital input module. Other modules, like our H-bridge module, knock module, or UEGO module, will eventually be used on the bike as the project evolves further. Figure 3.3 shows two of the modules used on the bike.

3.5.1 Fuel Injector Driver Module

Automotive-style fuel injectors generally can be categorized into two electrical classifications: saturation and “peak/hold.” A saturation injector solenoid is able to withstand full battery voltage across it for an extended period of time and allows simple drive circuitry, but at the cost of reduced valve opening and closing performance. A peak/hold injector solenoid requires a large amount of current to open the valve and significantly less to hold it open. This current profile is shown in



Figure 3.3: Prototype Modules

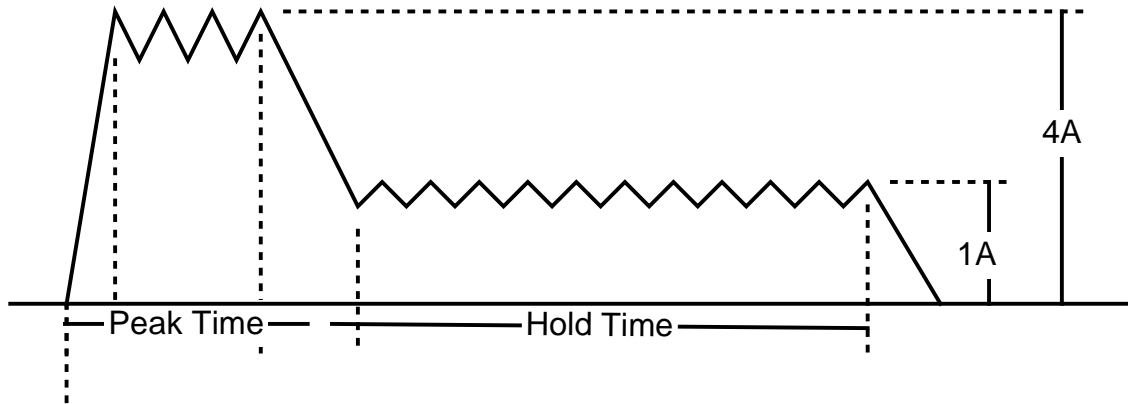


Figure 3.4: PFI Current Profile

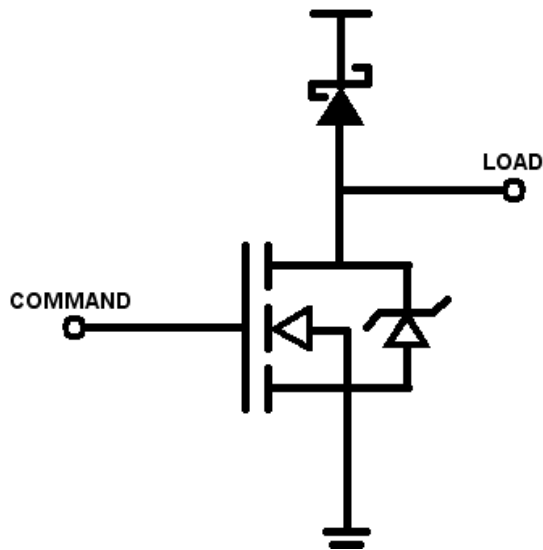


Figure 3.5: Low Side Switch

Figure 3.4. This requires specialized drive circuitry but typically allows for moving a larger needle in the injector valve. In either case special hardware is used to ensure that the current stops quickly in the injector solenoid when it is turned off so that the valve shuts quickly and repeatedly.

Most other high-current automotive actuators are driven by what are called low side switch drivers. Typical low side switch driver applications are relays, wastgate valves, EGR valves, fuel pumps and fuel regulators. Figure 3.5 shows the general topology of a low side switch driver circuit.

Our fuel injector driver module has four peak/hold fuel injector drivers and four low side switch drivers.

3.5.2 Spark Module

A spark module is used to drive the coil that generates the spark across the spark plug gap. It functions by allowing current to flow through the primary side of the coil, then interrupting it suddenly. When the current is interrupted, the stored energy is transformed into a high voltage across the secondary coil and dissipated in the form of a spark.(Bos, 2000) Generating the spark creates a large kickback voltage in the range of several hundred volts. Care must be taken not to disrupt the other circuits when this happens or to dissipate this energy prematurely.

Our spark module had eight spark driver channels, even though only four were needed on this project. Both fuel and spark modules have electrical isolation. Although isolation is not required in production controllers, a research controller is far more prone to mal-wiring related problems.

3.5.3 Input Module

The analog and digital input module, which will be referred to as the AD Combo module, is used to monitor nearly all types of automotive sensors. It has 22 single ended analog inputs, two VR sensor inputs, and two digital hall effect type inputs, as well as regulated sensor power supplies.

All of the modules were designed with cost sensitive, automotive-quality components, in order to achieve similar I/O circuit behavior between R&D and possible future production FPGA-based controller implementations

Table 3.1: Sensor and Actuator Characteristics

Sensor / Actuator	Characteristics / Type
Crankshaft Position	Variable Reluctance
Camshaft Position	Digital Hall Effect
Throttle Position	Linear Analog
Intake Air Pressure	Linear Analog
Barometric Air Pressure	Linear Analog
Intake Air Temperature	Analog Thermistor
Coolant Temperature	Analog Thermistor
Vehicle Speed	Digital Hall Effect
Fuel Injector Solenoid	12 Ohm, Low-Side
Ignition Coil Primary	0.3 Ohm, CDI

3.6 Mapping

In any project involving replacing an OEM ECU, it is important to know where to start. It is critical to create a baseline application which controls the engine in the same way that the OEM ECU did for all of the conditions relevant to the testing being preformed. Only after a solid baseline is achieved should features be added or changed. Baselining was the extent of the next phases of the motorcycle project. The next step is to map the OEM ECU. Rarely are researchers privy to the original control application and calibration, so some work is required. Mapping is done by tapping in to the existing ECU wiring and monitoring the data going in and out.

Table 3.1 lists the sensors and actuators that interfaced to the motorcycle and the type of interface circuit that was needed. For the mapping effort, sensors were monitored on the primary AD Combo module and actuators were monitored on the secondary AD Combo module. This division allowed the primary AD Combo module configuration to remain the same for the sensors throughout the project.

Table 3.2 lists the quantities that were sensed for each input. It is important to generate a calibration of all sensors. It is common to use the same sensor calibration for all sensors of a given type on a vehicle. That was the case for the pressure sensors

Table 3.2: Engineering Data

Engineering Data	Range / Units
Fuel Pulse-Width	0 - 7 Milliseconds
Spark Advance	0 - 50 Deg BTDC #1
Throttle Position	0 - 100%
Intake Air Pressure	0 - 110 kPa Absolute
Barometric Air Pressure	0 - 110 kPa Absolute
Intake Air Temperature	-20 - 120 Deg C
Coolant Temperature	-20 - 120 Deg C
Vehicle Speed	0 - 160 MPH
Engine Speed	0 - 16000 RPM
Fuel Pulse Start Angle	0 - 720 Deg Absolute
Fuel Pulse End Angle	0 - 720 Deg Absolute

and temperature sensors. These were calibrated by removing a sensor and logging the response to known temperatures and pressures. The throttle was assumed to be linear and the end points were measured. The rest of the inputs had calibrations that were derivable due to their nature and could be entered directly into the program. Our FPGA IP was used to track the angular position of the crankshaft and capture the angular positions and durations of various fuel and spark events.

3.6.1 Mapping Method

Before a mapping exercise could be started, it was critical to understand what is planned to be mapped. Due to the lack of exhaust oxygen sensors in the I/O list of Table 3.1, it was apparent that the OEM controller implemented an open-loop fueling strategy. It is known that the OEM controller does not consider the transmission gearing, so that full coverage of the map can be achieved by simply covering the full range of engine speed and load, regardless of vehicle speed.



Figure 3.6: Bike glove box with cRIO installed

3.6.2 Data Collection

One option was to pull the engine out of the bike and install it on a dynamometer. This would have been time consuming and expensive. It was decided to attempt to map the engine in the vehicle first, and examine the quality of the data. If this provided sufficient coverage of most operating points while running the engine in the motorcycle, then this costly procedure could be avoided.

A logging rate of 5 milliseconds for each channel was used because that would fill the available 10 megabytes of internal flash storage of the cRIO in about 20 minutes. Therefore, the data on the bike would have to periodically be uploaded to another computer. An 802.11 wireless link was installed on the motorcycle. This would allow an engineer in a chase vehicle to upload the data files to a laptop without the need to make a physical connection. The engineer could also monitor the data in real-time as long as the chase vehicle was within a few hundred feet of the motorcycle. The cRIO installation and 802.11 wireless interface are shown in Figure 3.6.

A LabVIEW application on the laptop automated the process of making a wireless ftp connection to the motorcycle, uploading data, deleting the data from the cRIO, and analyzing the data for coverage of operating points. This application analyzed all of the data collected up to that point in the mapping exercise and determined where there was sufficient coverage and where more engine operation was needed.

Our operating coverage map consisted of about 700 points, where engine speed ranged from 0 to 16000 RPM in 500 RPM bins and throttle position ranged from 0 to 100% in 5% bins. Each engineering parameter of interest, such as spark advance, fuel pulse-width, etc., had its own operating map for its respective data to be stored and processed. Ninety percent map coverage was achieved in two hours of testing. It was concluded that this was sufficient coverage to allow use of the data by filling in the sparse gaps via interpolation.

3.6.3 Post Processing

Post processing was handled by two different LabVIEW applications. The first application, which was performed on the laptop in the chase vehicle, examined each data point to determine whether it could be accepted as a steady-state operating point. A point was considered acceptable if it and the 10 previous points all were contained in the same speed versus load bin. Figure 3.7 shows how many points were collected over the range of speed and load.

Acceptable data points, for each parameter of interest within each operating bin, were stored in temporary arrays until all collected data was sorted. Then the data in each bins array was averaged and a standard deviation was calculated. The average and standard deviation was placed into an operating map for each parameter of interest. The result was a collection of average data, for each parameter of interest, sorted into speed versus throttle (load) tables. Figure 3.8 shows an

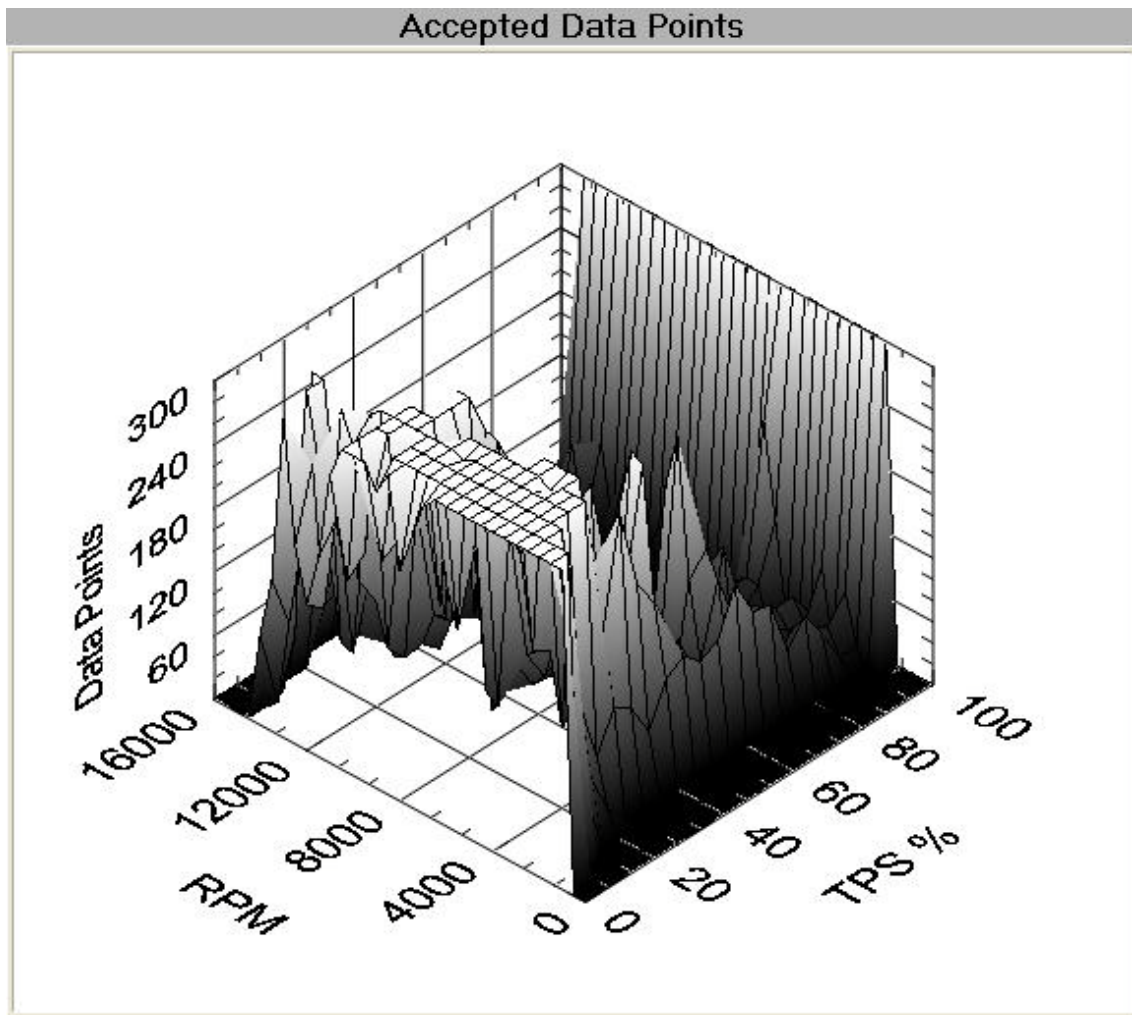


Figure 3.7: Accepted Data Points

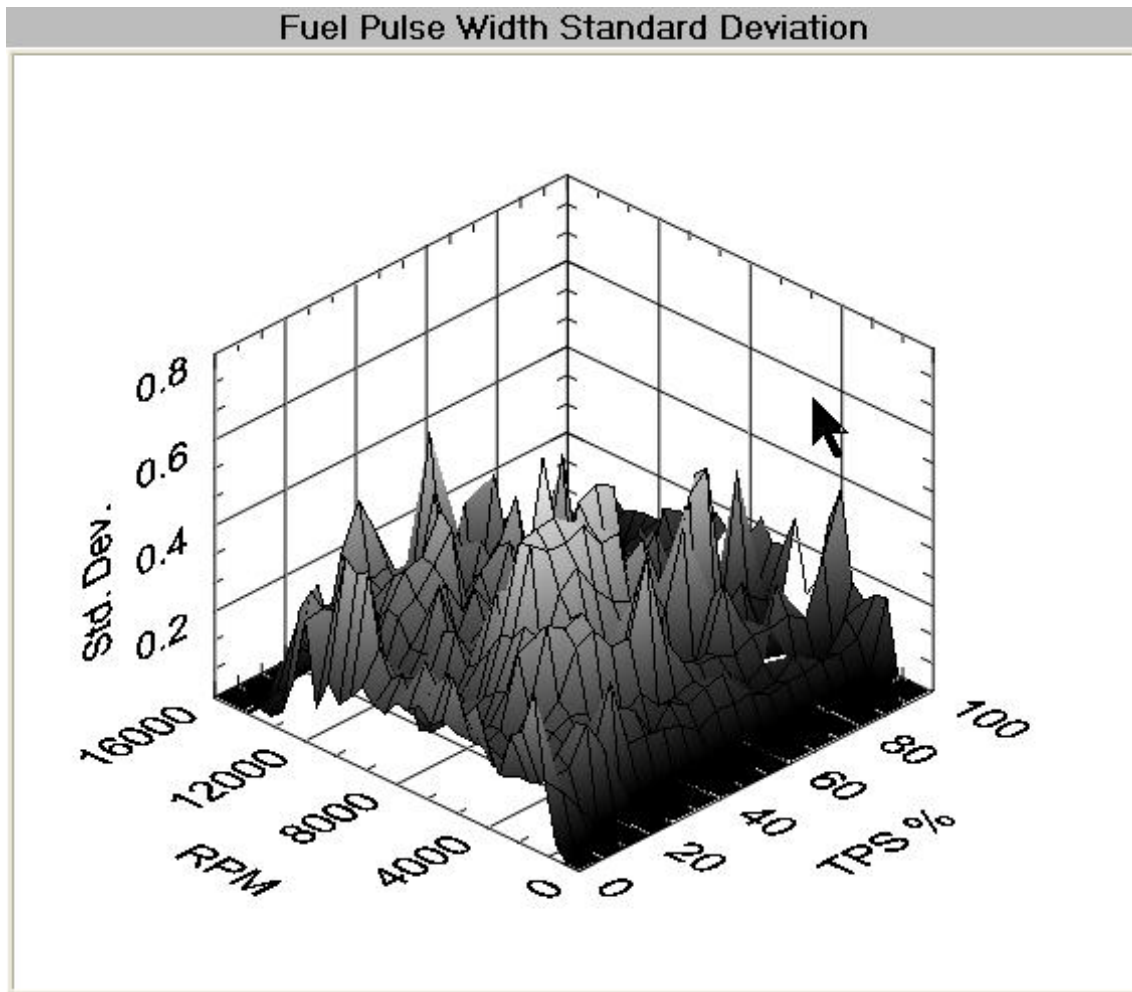


Figure 3.8: Fuel Pulse Width Standard Deviation

example of a standard deviation map for fuel pulse-width. The standard deviation maps provided us an indication of consistency at each operating point for each engineering parameter.

In order to visualize the data and application was built that shows each parameter in a three dimensional surface as well as two dimensional slices of the surface. The application allowed the two dimensional slices to be modified graphically. Data was manually fixed where inconsistencies were found or points were missing. The resulting modified, two dimensional arrays of data then were ported to the control application as calibration tables. The display for the control application also showed the calibration tables as three dimensional surfaces, which could be modified graphically while the engine was running. The fuel pulse-width map was then converted to an air mass map, based on stoichiometric fueling. The air mass table was also corrected to standard temperature and pressure. This air mass table was used as a beginning calibration for our alpha-N fueling strategy.

3.7 Engine Controller

It is important to be able to go between the stock engine controller and the development engine controller quickly. This is important when the engine is not running as it should. Putting the stock ECU back in place will quickly determine if the problem lies in the engine or the controller. The upper picture of Figure 3.9 shows the factory ECU installed under the motorcycle seat. The bottom picture shows the factory ECU removed, with an adapter harness, built in house, connecting the cRIO to the factory wiring harness. In order to achieve this convenient interchangeability, a spare OEM ECU was acquired for the purpose of utilizing the ECU connector in our adapter harness.

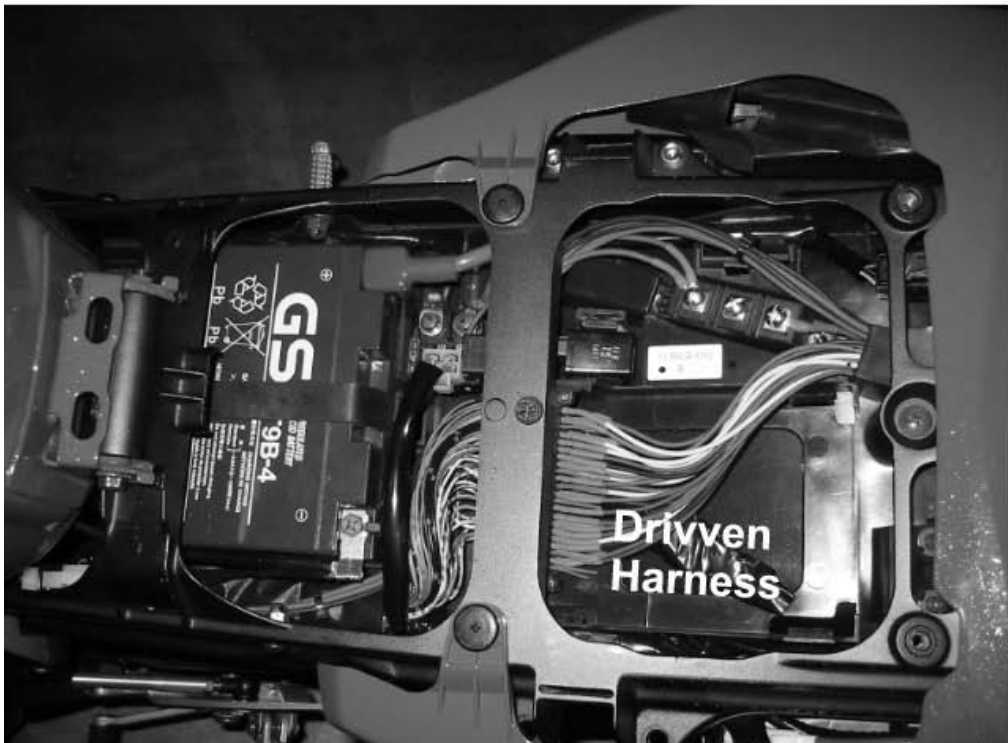


Figure 3.9: Factory and Custom ECU installation

3.7.1 Basic Engine Control Algorithms

For gasoline engines, there are a number of basic engine control algorithms that can be employed depending on the application. This section describes the control algorithms available and why we chose the ones we did.

Mobile gasoline engines generally use a “fuel follows air” approach to fuel control. This means that the driver determines how much air is brought in (via the throttles), and fuel is calculated to match. This provides for a constant air/fuel ratio and better emissions.

There are three general air calculation methods: Speed-Density, Alpha-N, and Mass Air Flow. They often are used in combination with one another, and control will transition from one mode to another over the speed load range.

The motorcycle used a combination of Alpha-N and Speed-Density. Speed-Density was used at idle and small, part throttle conditions while Alpha-N was used at open throttle conditions.

3.7.2 Speed-Density

As the name implies, this air calculation method requires the computation of engine speed and intake air density. The basic equation is below. (Heywood, 1988)

$$\dot{m} = \eta_v(N)\rho_a(T_i, p_i)V_d = \frac{\eta_v V_d p_i}{R_a T_i} \quad (3.1)$$

Where \dot{m} is the mass flow rate of air in kg/stroke. That is how much mass of air goes in to the cylinder each time the intake valve is opened. An alternate form of this equation could be used to derive the mass flow per second. This is common for throttle-body injected systems like carburetor replacement applications or natural gas engines. The terms p_i , T_i , and N refer to Manifold Absolute Pressure (MAP), Manifold Absolute Temperature (MAT), and Engine Speed in firings per second,

respectively. For this equation, MAP is in Pa and MAT is in K. R_a is the universal gas constant for air and is 286.9 J/KG*k.

The volumetric efficiency term η_v , often abbreviated as VE, is used to correct for pumping losses and intake/exhaust tuning effects. It is not strictly an efficiency because with proper intake and exhaust tuning, numbers greater than 1 are possible. Volumetric efficiency is a function of a number of engine parameters including engine speed, engine temperature, humidity, intake pressure and exhaust pressure. Normally most of these terms are ignored and corrected for in closed loop control. In normally aspirated engines the volumetric efficiency table is a function of RPM. In a boosted engine it is a function of both RPM and MAP. For many engines, a value of 0.85 is usually close enough to get the engine running and begin tuning.

Speed-Density is used on many applications because it is inexpensive, and the only calibration work for steady state operation is the VE table.

3.7.3 Alpha-N

Alpha-N engine control calculates air flow purely as a lookup table of throttle position (Alpha) and engine speed (N). Alpha-N control algorithms are simple and fast. However, they are very calibration intensive because each engine operating point must be calibrated any time a modification is made that affects the volumetric efficiency of the engine.

Alpha-N is often used as a backup strategy in case of a sensor failure in a MAF or Speed-Density strategy. Alpha-N also is used in many after-market, high performance and race engine applications. Aftermarket engine controllers can be applied to a wider variety of engines by using an Alpha-N fueling strategy, which requires fewer sensors. Also, high performance and racing engines typically will employ camshafts that have significant overlap timing of intake and exhaust valve actuation. This can significantly reduce the variability provided by a MAP sensor

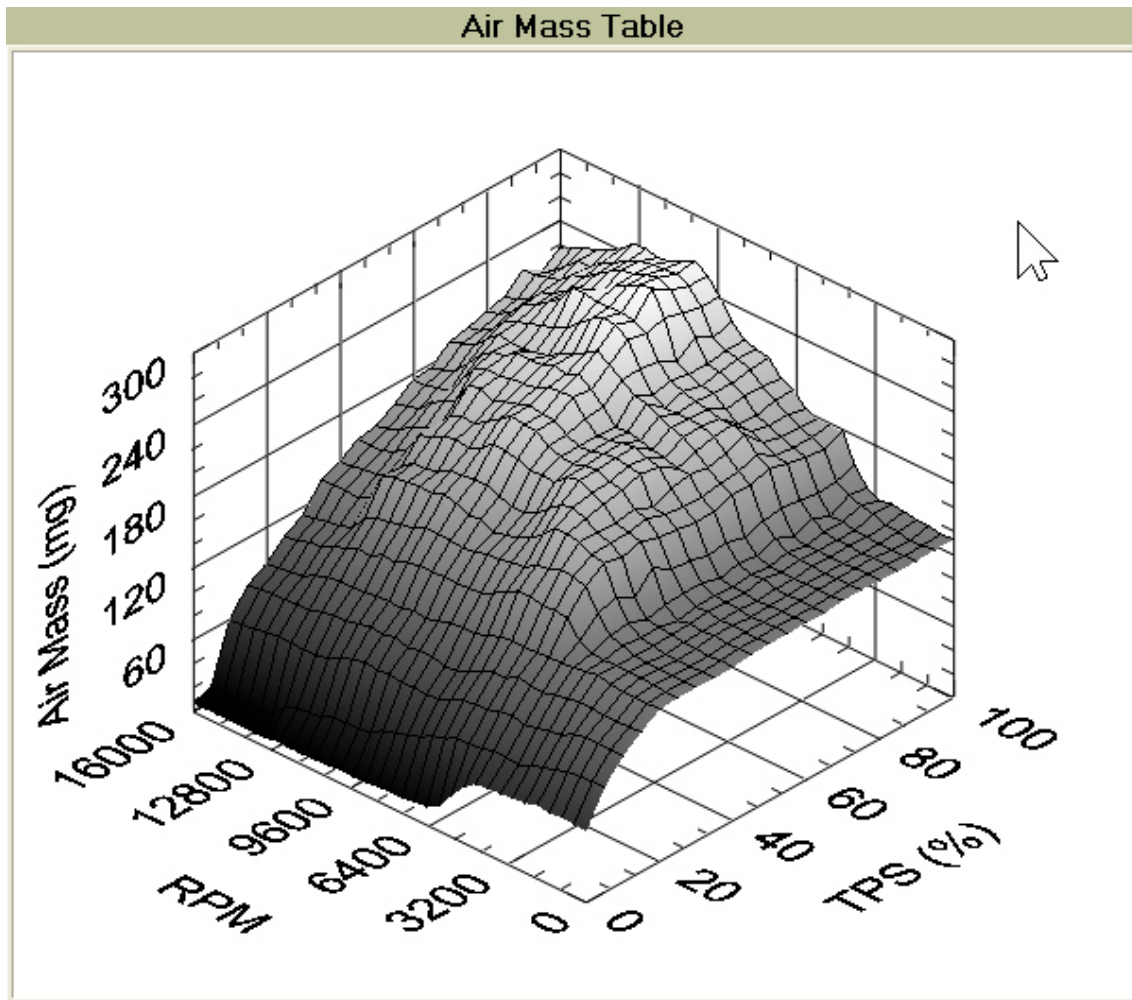


Figure 3.10: Mass air flow

over the entire engine load range. The result can be that the MAP sensor will read atmospheric pressure at throttle positions as low as 30%. Therefore, throttle position must be used as the primary indication of engine load for higher throttle angles.

Our high performance motorcycle engine utilized both an intake air pressure sensor and a throttle position sensor. We had to deal with similar pressure sensing issues described above due to high performance camshafts. Therefore we implemented a Speed-Density algorithm at throttle positions below 20% and engine speeds below 3000 RPM. This provided us with consistent engine operation at idle and small part throttle conditions, which would respond to fluctuations in air pressure and temperature. Above these operating points we transitioned to an Alpha-N air mass table. An air mass was extracted from the table, corrected for actual air temperature and barometric pressure, and a resulting stoichiometric fuel quantity was calculated. This meant that we had to rely heavily on the data acquired for fuel pulse-width, as collected during the mapping phase. We only needed to calibrate our volumetric efficiency table, used in Speed-Density mode, for engine speeds below 3000 RPM. This calibration was a short trial and error process.

Figure 3.10 shows the Alpha-N air calculation table derived from the mapping exercise.

3.7.4 MAF

Mass Air Flow (MAF) is a technique of directly measuring \dot{m} of air into the engine. MAF sensors are relatively expensive and too large to be practical on a motorcycle application. However, they do provide a relatively calibration-free method of calculating air flow to an engine.

The other danger with MAF sensors is that they cannot determine the direction of flow. If the manifold allows occasional backward air flows towards the throttle

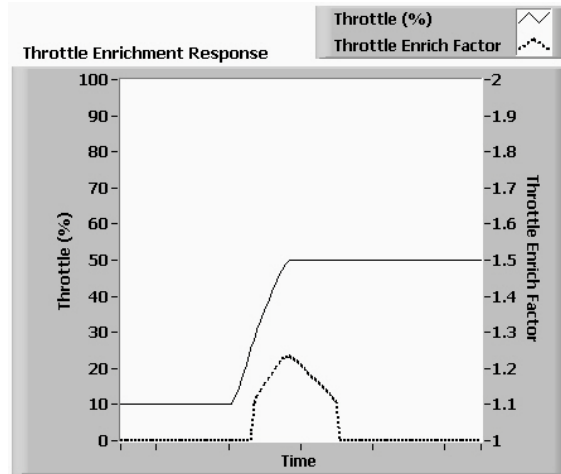


Figure 3.11: Transient Enrichment

(not uncommon at low speed), the resulting calculated air quantity will be higher than it should.

3.7.5 Transient Compensation

Transient compensation is by far the trickiest aspect of intake air estimation and fuel flow calculation. Many papers have been written on the subject and it is beyond the scope of this paper to describe more advanced methods.(Hendricks et al., 1996)

Fortunately, if you only need to make power and are not constrained by emissions targets, there are some simple approaches that work well.

The method we used creates an enrichment value. This value is added on to the calculated air mass. The enrichment value is added to by the first derivative of throttle position. It is subtracted from at a constant rate. Figure 3.11 shows an example air addition factor as a result of a throttle transient.

3.7.6 Closed Loop

In order to meet emissions requirements, cars need to run at precise air-fuel ratios in order to take advantage of exhaust catalysts. Carburetors and open loop fuel injection systems are not sufficiently accurate to control fuel for this because they do not have any feedback of the actual air-fuel ratio.

Because the motorcycle does not need to pass strict emissions tests required of passenger cars, it implemented an open loop fueling system. If emissions were more critical, it would have utilized an exhaust oxygen sensor in the form of either a “switching O₂ sensor” or a “wide band O₂ sensor” to close the loop.

Switching sensors only tell the controller whether it is operating rich or lean of stoichiometry. They do not tell how much. A ramp and jump back type strategy is used to control fuel with this sort of sensor.

If it is desirable to precisely operate away from stoichiometry, then the only option is to use a wide-band sensor like the Bosch LSU4.2. This requires a much more complex sensor interface than the switching sensors but provides proper feedback to use a simple PID controller to deliver optimized fuel.

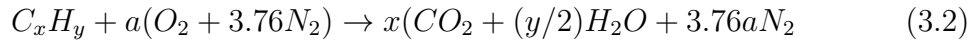
With either sensor it is possible to determine errors in the Speed-Density VE table over the life of the engine. The table then may be corrected on the fly to give better results for open loop operation.

3.7.7 Fuel Calculation

Once the mass of air is calculated, a mass of fuel must be calculated for a desired air-fuel ratio. This is a three-part calculation. First, we must determine the desired air-fuel ratio for the given operating condition and fuel being burned. Second, we must calculate the mass of fuel appropriate for the air-fuel ratio and air

mass. Third, the characteristics of the fuel injector must be taken into account to calculate an injection pulse-width command for the injector driver.

Stoichiometry is defined as the ratio of reactants to products in a chemical reaction. If fuel is combined with air at exactly the stoichiometric ratio and ideal combustion is achieved, then all reactants (fuel and air) will be consumed and only products CO₂, H₂O and N₂ will remain. For a hydrocarbon fuel given by C_xH_y the stoichiometric reaction can be expressed as (Turns, 2000)



where

$$a = x + y/4 \quad (3.3)$$

From this, we can determine the stoichiometric air-fuel ratio as

$$(A/F)_{stoich} = \left(\frac{m_{air}}{m_{fuel}} \right)_{stoich} = \frac{4.76a}{1} \frac{MW_{air}}{MW_{fuel}} \quad (3.4)$$

We then define the term equivalence ratios Φ and λ as

$$\Phi = \frac{1}{\lambda} = \frac{(A/F)_{stoich}}{(A/F)} = \frac{(F/A)}{(F/A)_{stoich}} \quad (3.5)$$

As a note, the symbol Φ is often used in academic textbooks, while λ is often used in industry papers and documentation.

If you are not constrained by using a three-way exhaust catalyst, then it is reasonable to try increasing power or fuel economy by adjusting the air-fuel ratio.

Once the desired phi is determined and the constants for fuel properties are entered, then the mass of fuel can be calculated by

$$m_{fuel} = \lambda(A/F)_{stoich}m_{air} \quad (3.6)$$

Finally, the fuel mass must be converted to an injector pulse-width. This is the amount of time in which current is driven through the injector solenoid. This can be calculated from the volumetric flow rate of the injector as a function of fuel pressure to the injector valve. The fuel density must also be known. The fuel pressure typically is tightly regulated by a small mechanical system attached to the fuel injector rail, so the pressure can be assumed constant.

Another significant factor in the calculation of final pulse-width is the amount of time required for the injector valve to fully open after the injection command starts. This open-time is a function of battery voltage supplied to the injectors. Because we did not have access to a fuel injector flow-bench, we carried out a low-budget experiment on the motorcycle to collect injector open-time data for various battery voltages. This involved using a current probe to monitor the current to the injector on an oscilloscope. We also monitored the voltage drop across the injector solenoid and the battery voltage. During an injection event, as the current to the solenoid rises, an inflection in the current trace can be seen before the current reaches its maximum level. This inflexion point represents the time in which the valve opens. The time from the beginning of the solenoid voltage drop to the inflection point is the injector open-time. We started the test with a fully charged battery. We cranked the engine just long enough to capture the first few injection events on the oscilloscope and then immediately stopped cranking the engine before the engine was able to start. We took notes of the battery voltage and injection open-time from the captured oscilloscope traces. After performing this procedure several times, the battery voltage level gradually decreased to a point where the engine would not crank or the ECU would not stay alive. We were able to take several consistent

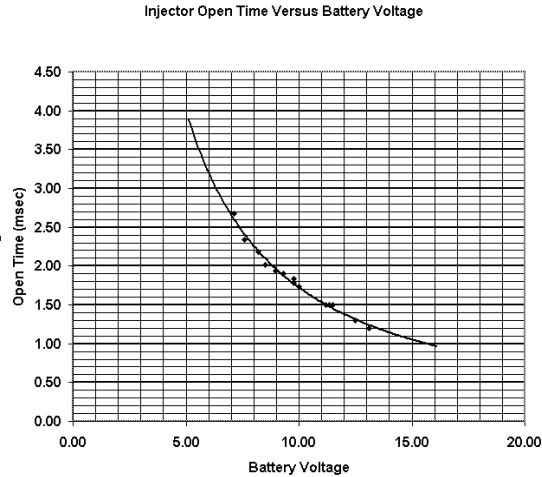


Figure 3.12: Injector voltage compensation

measurements from a battery voltage of 14 volts down to 7 volts. This data allowed us to create the plot shown in Figure 3.12. During engine control, the injector open-time was pulled from a table based on battery voltage. This time was multiplied by a factor of 0.67 to account for a lower fuel flow rate during injector open-time.

$$PW_{fuel} = k_{open} * OpenTime + \frac{m_{fuel}}{\dot{m}_{injector}} \quad (3.7)$$

Equation 3.7 shows the fuel pulse-width calculation. Once the pulse-width is calculated, it is handed off to the FPGA for execution. The FPGA tracks engine position and fires the injectors at the desired crankshaft angle and for the desired pulse-width, without any further interaction with the CPU.

3.7.8 Transient Fuel Compensation

Transient fuel compensation is difficult and beyond the scope of this project. We simply used the throttle transient air compensation as a fudge-factor to encompass both manifold-filling of air and wall-wetting of fuel. Nonetheless it is worth touching on the subject.

One common model of transient fuel flow is called the $\tau - x$ model.(Leone et al., 1997) It states that some portion (x) of each injection event goes into the cylinder, and the rest goes to the puddle that sits on the valve. There is an evaporation rate of the puddle (τ) as a function of coolant temperature. Because there is no simple measure of either τ or x , this is difficult to calibrate.

3.7.9 Spark Control

Spark timing is a simple function of speed and load - throttle position in our case. For an application such as ours, it is always desirable to operate at MBT timing. MBT timing is the Minimum advance for Best Torque.(12. Guerrier and Cawsey, 2004) Unfortunately, in many cases, MBT timing may be beyond the knock limit of the engine. Control systems utilizing knock sensors attempt to advance to MBT timing until knock is detected and then retard until knock is eliminated.

Our motorcycle did not come with knock sensors, so the factory calibration was apparently conservative enough to allow it to run on premium pump gasoline without knocking.

We mapped the spark timing of the OEM ECU simply by capturing the crank-angle of each spark event and logging the latest value every 5 milliseconds. Our LabVIEW data analysis application sorted the data into a speed versus load table that was used directly in our control application. The resultant map is shown in Figure 3.13.

Ignition coil dwell-time also was extracted from a table based on battery voltage, since the amount of dwell to achieve repeatable spark-energy depended on battery voltage. The spark timing and dwell-time are then handed off to the FPGA for execution. The FPGA tracks crankshaft position and commands the ignition coils for the desired dwell-time and terminates the command at the desired spark timing angle, without any further interaction with the CPU.

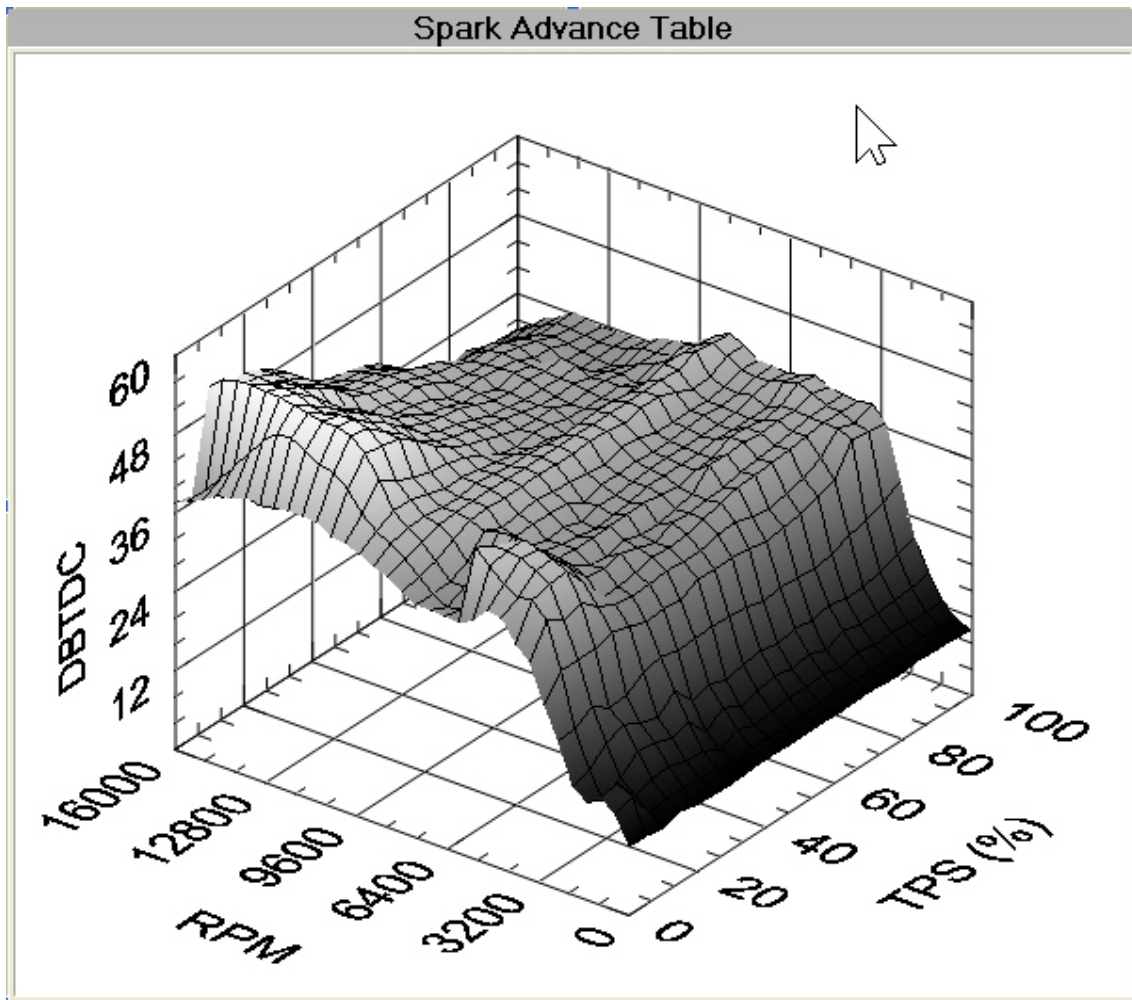


Figure 3.13: Spark advance table

3.7.10 Rev Limiter

In performance application a rev limiter is required so that the operator cannot damage the engine by allowing it to overspeed. It is important, however, that the rev limiter not be too severe so that it will not upset the torque delivered by the engine.

Our rev limiter has two parts: spark retard and fuel/spark kill. Spark retard retards timing by a tunable amount when the engine speed reaches 15000 RPM. We found that retarding spark timing by about 20 degrees BTDC smoothly reduced torque and prevented over-revving.

If the engine speed exceeds 15500 RPM, (red-line indicated on the OEM tachometer) despite the retarded timing, then we go ahead and kill both spark and fuel.

3.7.11 Startup and Idle Control

Getting an engine to start and idle for the first time is usually a trial and error process. We did not formally perform any mapping of the startup and idle conditions while under OEM ECU control. Startup and idle are primarily affected by the operating temperature of the engine. In our case, that indicator was coolant temperature. Due to this relationship, tuning for startup and idle must be performed over several days time in order to arrive at a good calibration. This is because engines can take several hours to cool to ambient temperatures. Therefore, we performed tuning for these conditions whenever we began a new session of tuning with a cold engine.

Idle control is not a difficult task on a motorcycle because there are no accessory loads such as air conditioning and power steering. Also, the electrical load for a motorcycle is typically very consistent. On passenger cars, an idle air control valve

or a fly-by-wire throttle perform the function of idle control. Our motorcycle used individual, mechanical, butterfly throttle bodies for each cylinder.

We implemented individual startup and idle fuel enrichment factors, which were multiplied by the final fuel quantity calculation from the Speed-Density mode. These factors were based on coolant temperature. They were very effective in starting and idling the engine consistently. We were able to find their minimum and maximum limits and then tune them for consistent starting and idling, at least in the limited temperature ranges we encountered. We monitored a signal (provided to the OEM ECU) that indicated when the start button was pressed. Startup fuel enrichment was made active during that time.

As a note, idle engine speed control could also be more effective for this application by adjusting spark advance. We did not implement this.

3.7.12 Other I/O

While fuel and spark are the two primary outputs of this controller, there are additional engine and supervisory functions that must be performed. For example, the motorcycle has an air induction valve, which is controlled as a function of throttle position and coolant temperature, to bypass fresh air from the intake air box to the exhaust, aiding the catalyst in the muffler. Also, the fuel pump, headlights, and radiator fan are controlled by the factory ECU. Our system replicated the ECUs control of these actuators. There are also a number of safety related signals to monitor in order to prevent the engine from running while the side stand is down or the bike is tilted past a certain angle.

3.7.13 User Interface

Besides being a graphical programming environment, LabVIEW provides an excellent user interface. Figure 3.14 is a screen shot of the user interface we de-

veloped. This interface has the ability to adjust all of the calibration values of the engine control code on the fly. The data communicated between the laptop-based interface application and the cRIO real-time control application were transported via the TCP/IP protocol over a wireless link. A LabVIEW wizard was able to generate LabVIEW code for communicated parameters and tables via TCP/IP. The wireless link proved to be very beneficial and could be maintained over short distances (about 300 feet) between a chase vehicle and the motorcycle while being tested on the road. Breakdowns in the wireless link did not affect the operation of the real-time control application.

Because the mapping exercise provided an excellent calibration for our control application, we did not get many opportunities to use our interface for significant calibration work. However, as we add features in the future, the ability to change the calibration and its visualization will be critical.

3.8 FPGA Fuel and Spark Control

The previous sections alluded to the cRIO FPGA managing all of the I/O tasks, primarily fuel and spark. This section will discuss the FPGA IP cores implemented for this application.

FPGA technology has made great advances in the last few years in all of the critical categories of density, speed, and cost. As a result, FPGAs recently have been embraced by the consumer electronics industry, providing superior flexibility and reduced development time over conventional microcontroller designs.

Automotive applications have tougher cost and temperature requirements for electronics components, but FPGA manufacturers are beginning to meet these demands. OEMs are already using FPGAs considerably in automotive telematics devices.¹³

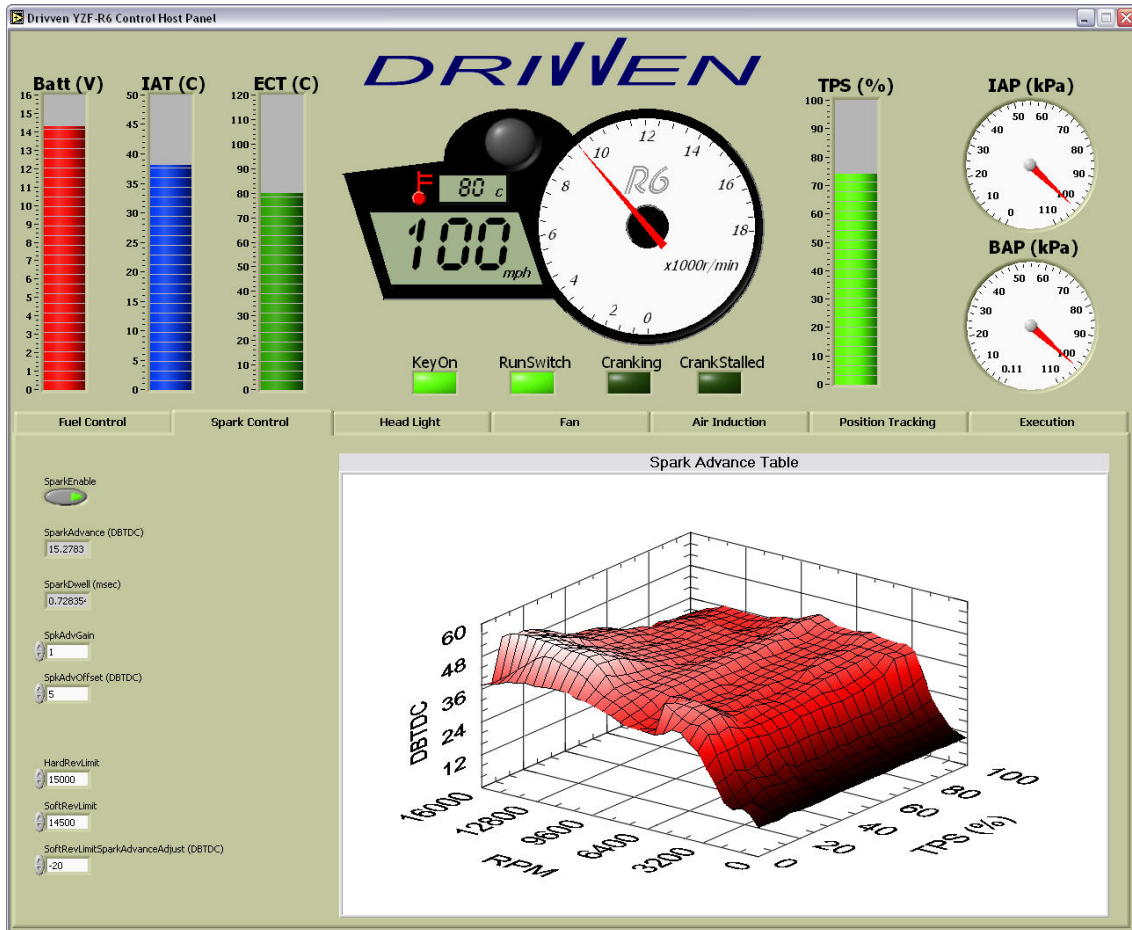


Figure 3.14: Graphical user interface

We have found that FPGA technology is far superior to traditional microcontrollers even specialty automotive microcontrollers for prototyping a powertrain control system. This is due to the ability to add or remove any I/O peripheral block, only limited by the size of the FPGA. Also, the software algorithms used to drive the IP blocks are very portable between hardware platforms which utilize an FPGA, relieving the concerns for microcontroller obsolescence and migration to different CPU environments.

We developed a library of FPGA IP blocks, or cores, for a variety of automotive powertrain tasks. The primary IP cores that are implemented in this engine controller are an engine position tracking (EPT) block surrounded by four identical port fuel injector cores and four identical spark cores. The EPT cores in our library are designed for specific crank/cam trigger pattern types. For example, one of the EPT cores tracks position from a crank trigger wheel having evenly spaced teeth and one or two adjacent missing teeth. This pattern is commonly referred to as an N-M pattern (Figure 3.15) because it has N evenly spaced teeth with M missing adjacent teeth. A very popular production example of this pattern is a 60-2 pattern. This pattern also can be accompanied by a single cam trigger tooth during every other tooth-gap for engine phase information on four-stroke engines. Another EPT core in our library tracks position from a pattern typical of optical encoders, such that there are N evenly spaced teeth per engine cycle accompanied by another single pulse per cycle. This EPT core is often applied to research engines having an optical encoder mounted to the crankshaft or camshaft (Figure 3.16). This latter pattern example is what we encountered on our project motorcycle, having four evenly spaced teeth on the crankshaft and a single tooth on the camshaft. The core was configured appropriately for the number of teeth.

The EPT core can be configured to calculate the angular position of the crankshaft to any desirable resolution as long as the clock rate to the core is sufficient. We con-

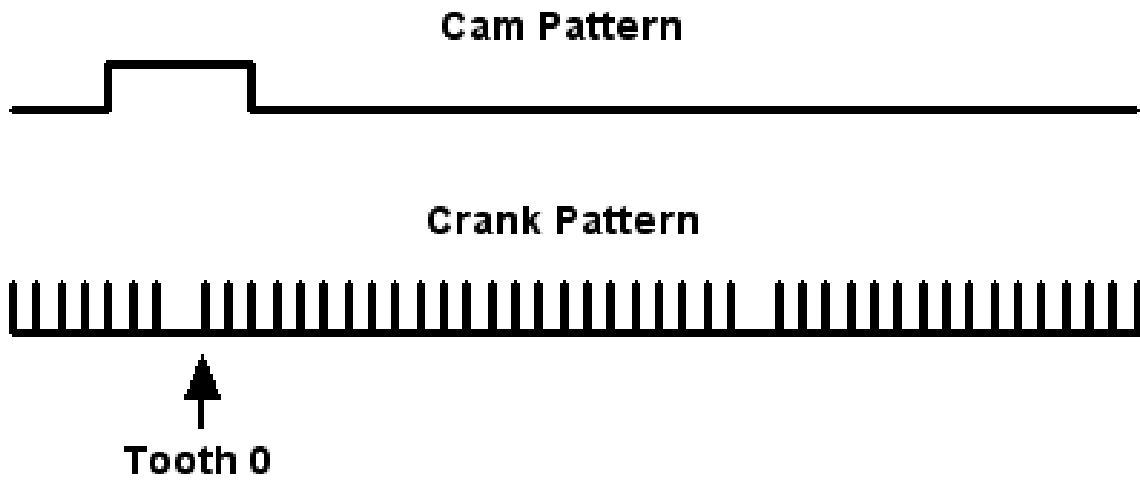


Figure 3.15: N-M pattern

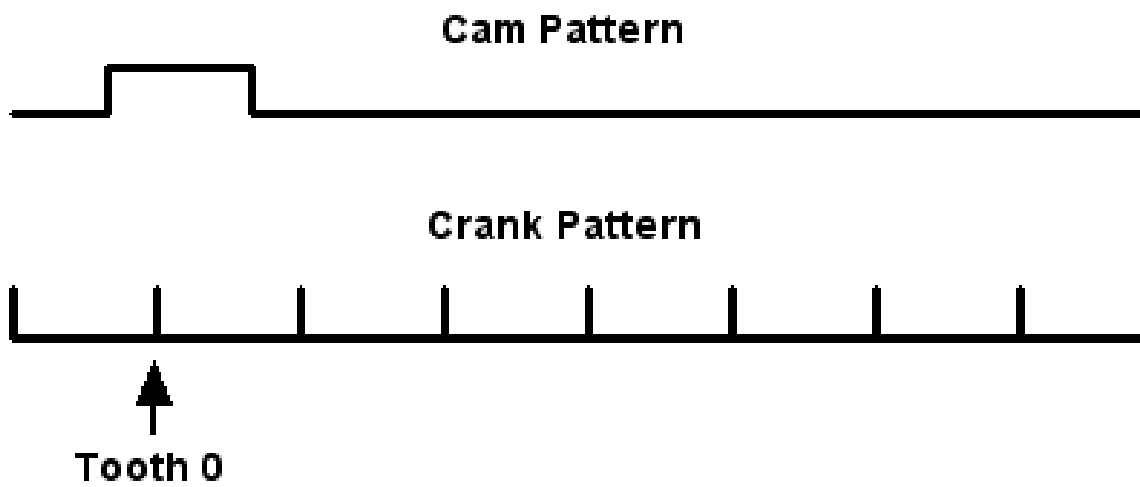


Figure 3.16: N+1 Pattern

figured the EPT core for this application to provide a position resolution of 0.3 degrees. This position, along with other supervisory signals, is routed to each fuel and spark core. In addition, the EPT cores report the time period between the most recent two trigger teeth, the time period over a requested multiple of teeth, and a variety of signal fault conditions.

The Port Fuel core generates fuel command pulses to the fuel injector driver in both angle and time domains. It delivers a separate configurable pulse at the beginning of each main pulse for the purpose of commanding peak/hold type driver circuits. However, this feature was not needed for this project because the injectors are of the high impedance type. The core can be dynamically configured to generate fuel commands having a specified start or end angle, along with a specified duration, where duration has the priority over position during transients. The core will also deliver additional fuel pulses within the same cycle, after the first pulse has ended, if the CPU requests additional fuel to be injected. After studying the data collected from the ECU mapping phase, we determined that the production controller was commanding fuel pulses with a start angle and duration. This was due to the fact that the fuel start-angle measurements, according to engine speed and load, showed a more deliberate three dimensional surface. The factory ECU did not show to be commanding multiple fuel injections per cycle. A separate Direct Fuel core in our library provides multi-pulse injection features for common-rail diesel applications, providing up to five individually tunable injection events per cycle.

The Spark core generates commands for an ignition coil driver in both angle and time domains, according to a requested dwell-time and spark angle. The end-angle of the command has priority over dwell-time as long as a minimum dwell-time has been satisfied. This core also supports multiple re-strike pulses following the main spark pulse.

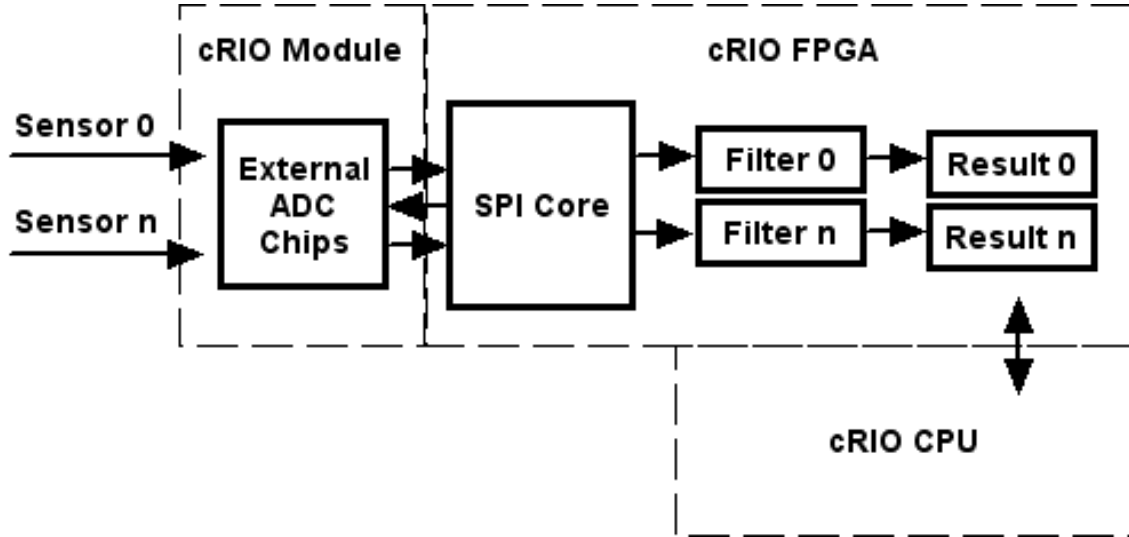


Figure 3.17: Module architecture

A Pulse Period core was used to measure the time period between pulses from the vehicle speed sensor. The CPU could always read the latest period value stored in the FPGA.

Finally, a Serial Peripheral Interface (SPI) core was implemented to retrieve the Analog-to-Digital (A/D) results from the AD-Combo modules ADC chips. All of the motorcycles analog sensor voltages were sampled via this SPI interface, including some signals that were digital in nature. Each channel was sampled at a rate of 2 kHz and filtered using a 5th order low-pass FIR filter having a cutoff frequency of 200 Hz. Each channels filtered result was stored in FPGA-based registers for the CPU to read at any time. This is shown if Figure 3.17.

The use of the FPGA to handle all of the I/O related tasks allowed the CPU to execute a single 10 millisecond loop without any I/O related interrupts. Asynchronous I/O related interrupts cause modern, high performance, pipelined processors to operate much less efficiently than they are capable. Our goal was to optimize the separation of tasks between the CPU and FPGA so that each device was being

utilized most efficiently. We are confident that this goal was achieved in this control system.

3.9 Conclusion

This project showed the viability of an FPGA based research engine controller. It demonstrated that the cRIO, executing LabVIEW RT, is an excellent choice of platform for both engine data acquisition and control. It showed that our team was able to do a complete controller replacement project in just three man-months.

We also showed the ability to map the behavior of an OEM ECU while operating the engine in the vehicle on the road. Using the mapped data as a calibration for the cRIO engine controller provided engine performance that was very acceptable, without any dynamometer time. Experienced riders could not note any significant differences between OEM ECU control and cRIO control.

This project motorcycle will be a future platform for use in developing additional hardware I/O and software modules.

Eventually, the LabVIEW code for this and other Drivven projects will be available under a semi-public license as part of a Drivven OpenECU project.

3.10 Acknowledgments

We would like to thank the entire National Instruments Compact RIO development team for their excellent and timely support.

We would like to thank Chuck Nance for his expert motorcycle riding skills and assistance with mapping the OEM ECU. Chuck is also a very skilled custom wiring harness fabricator. Observing his work on past projects at Southwest Research Institute held us to a higher standard of building a wiring harness for this project.

We would like to thank Leslie Nance for doing an excellent job of keeping up with Chuck while driving the chase vehicle for the mapping exercise.

3.11 Contact

Carroll G. Dase is the president of Drivven, Inc. He has designed research and production powertrain control hardware and software for Southwest Research Institute, Motorola, and Woodward. He has a BS in ME from the University of Texas. He can be contacted at cdase@drivven.com. <http://www.drivven.com>

Matthew Viele is a vice president of Drivven, Inc. He has designed research and production controllers for Southwest Research Institute and Woodward. He has designed research and production ion sensing systems and holds numerous patents in the field. He has a BS in electrical engineering from the University of Oklahoma and a MS in Computer Science from the University of Texas at San Antonio. He is pursuing a Ph.D. in mechanical engineering from Colorado State University. He can be contacted at mville@drivven.com.

Chapter 4

ICEF2011-60225

A Novel Approach to Free-Piston Engine Control Using an FPGA Based Control System

This paper was presented at the ASME Internal Combustion Engine Division Fall Technical Conference in Morgantown West Virginia, Oct 2-5, 2011. The paper was written by Matthew Viele and Carroll Dase of Drivven, and Eric Shorey and Karl Hoose of Applied Thermal Sciences.

4.1 Synopsis

The variable stroke length of the free-piston engine poses an interesting problem for the controls engineer. At a low level, the control system must be able to track piston position and address changes in top and bottom dead center positions. To accomplish this, a new FPGA (field programmable gate array) based engine position tracking software was developed, along with a simple method of mapping from a conventional engine control system to a free-piston control system.

The tracking software was integrated into a complete rapid prototyping control system that was responsible for all control actions of the engine. The control system was laboratory tested on the HiPerTEC, an opposed, free-piston engine with a circular piston arrangement (as opposed to linear free-piston engines) developed by Applied Thermal Sciences, Inc (ATS). The control system has been demonstrated to run 8 cylinders up to an effective speed of 2,200rpm in spark ignition mode.

4.2 Introduction

4.2.1 Why a Free-Piston

From a thermodynamic viewpoint, higher combustion pressures offer the capability of generating more work which translates into higher thermal efficiencies. Some novel engines, particularly free-piston engines, have shown higher thermal efficiencies than that of conventional diesel engines because of the extremely high combustion pressures found in these engines (Blarigan, 2002). With a variable compression ratio (CR), FPE can use low CR mode for starting and switch to high CR mode for improved efficiency. Numerous investigators have shown that a CR in the 35:1 range is attainable (Flynn, 1957)(Mikalsen and Roskilly, 2008b). In addition to thermodynamic efficiency gains, there is the potential to increase the mechanical efficiency by removing the wasteful slider-crank mechanism (Chinitz, 1969) and eliminating the friction caused by the piston side thrust (Amann, 1987)(Ciulli, 1992)(Ciulli, 1993).

4.2.2 Breif Free-Piston History

Free-piston engines were tested at length by a few major European and American manufactures, such as General Motors, Ford, and International Harvester, back in the 1950s and 1960s. Most of the testing configurations took the form of inward-compressing 2-stroke diesel-cycle gasifiers where the free-piston engine gasifier exhaust was used to drive a turbine to extract work. Flynn observed thermal efficiencies in the 43% range (Flynn, 1957), which is high even for todays engines. This was attributed to the high compression ratio of the engines which reduces the entropy gain during the combustion process. Flynn also demonstrated insensitivity to fuel types (Flynn, 1957). Unfortunately, free-piston engines never progressed from the testing phase into major production, largely due to the complexities of extracting mechanical power from the free-piston designs and controlling piston motion. Mod-

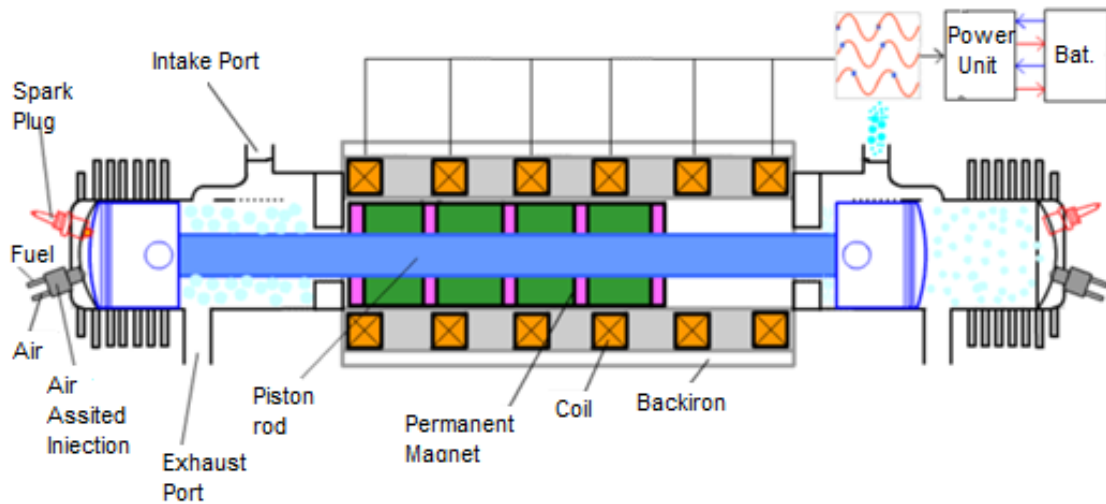


Figure 4.1: Linear Opposed Free-Piston Generator Unit

ern microprocessor based controls have permitted certain of the complexities of the FPEs to be overcome.

By eliminating the slider-crank mechanism of the conventional I.C. engine, the piston becomes “free”. Its motion is no longer governed by the rotation of the crank, but instead by the forces acting on the pistons from the cylinder gases and external loads. This introduces a complexity in engine control that was not easily addressed in early free-piston engines. However, modern microprocessor based controls have opened the door for engineers to revisit the advantages of the free-piston engine (Mikalsen and Roskilly, 2008a). Recent interest in free-pistons has developed due to the need for significant improvements in fuel efficiency and the attention of hybrid vehicle applications. The growing interest has led to new configurations such as the linear free-piston generator shown below (Figure 4.1), where a linear dual free-piston engine drives a reciprocating generator to produce electricity (Van Blarigan, 2002)(Carter and Wechner, 2003)(Mikalsen and Roskilly, 2008a). This interest has also led to the development of the HiPerTEC, an alternative arrangement to the linear FPE.

Table 4.1: HiPerTEC Specifications

HiPerTEC Specification	
Seam Diameter	10in
Displacement	1.8L
Bore	2.5in
Stroke	2.75in
Chambers	8
Max Geometric Compression Ratio	40:1
Calves per Chamber	1 intake/1 exhaust

4.3 ATS HiPerTEC Engine

The HiPerTEC design was conceived to incorporate the simplicity and compactness of the toroidal and Wankel engines, and the excellent balancing of the Bradshaw Omega engine (Chinitz, 1969)(Ciulli, 1993). Further, the HiPerTEC includes unique characteristics to take advantage of operating methods found to exhibit superior performance, such as variable compression ratio. ATS has successfully demonstrated the operation of the HiPerTEC with a ported, 2-stroke prototype and a valved, 4-stroke prototype. The focus of this paper will be the control of the valved, 4-stroke prototype throughout its start-up and warm-up phases of operation.

The HiPerTEC employs a toroidal geometry with eight chambers (corresponding to cylinders in conventional I.C. engines) separated by eight double-faced pistons. The engine torus shape is made up of two parts, one outer ring and one inner ring. The torus is cut (not like a bagel) resulting in two rings as shown in Figure 4.2. These rings reciprocate back and forth and make up the actual chamber walls of the engine. Four pistons, having the internal cross-sectional area of the torus, are connected to the concave (inside) wall of each ring (eight pistons total) at 90 degree intervals. Thus, eight chambers are made when the two rings are fitted together to form the torus shape; see Figure 4.2 and Figure 4.4.

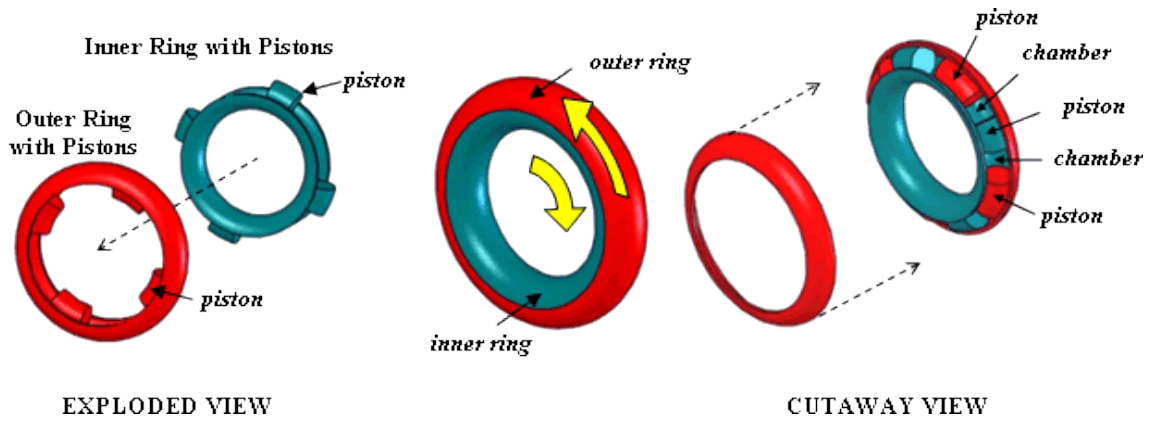


Figure 4.2: HiPerTEC Configuration

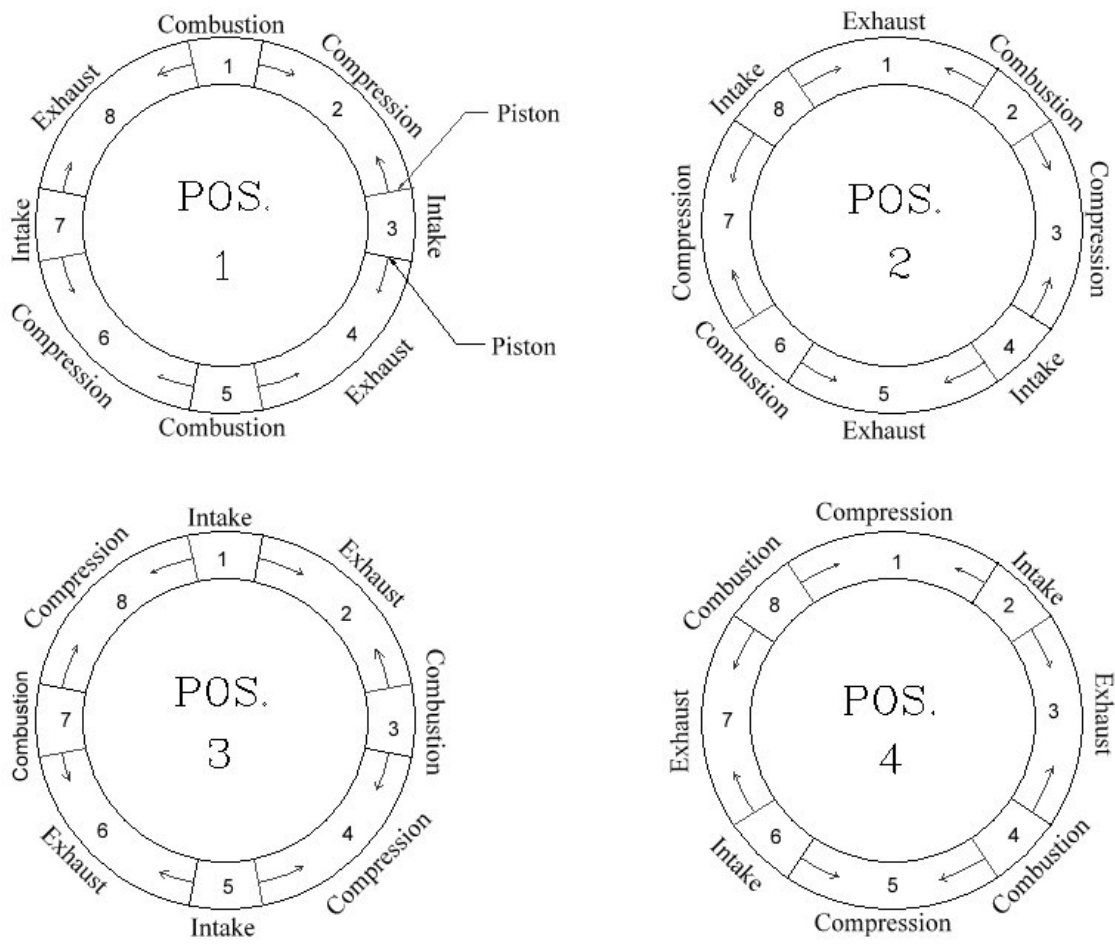


Figure 4.3: HiPerTEC 4-Stroke Cycle Positions

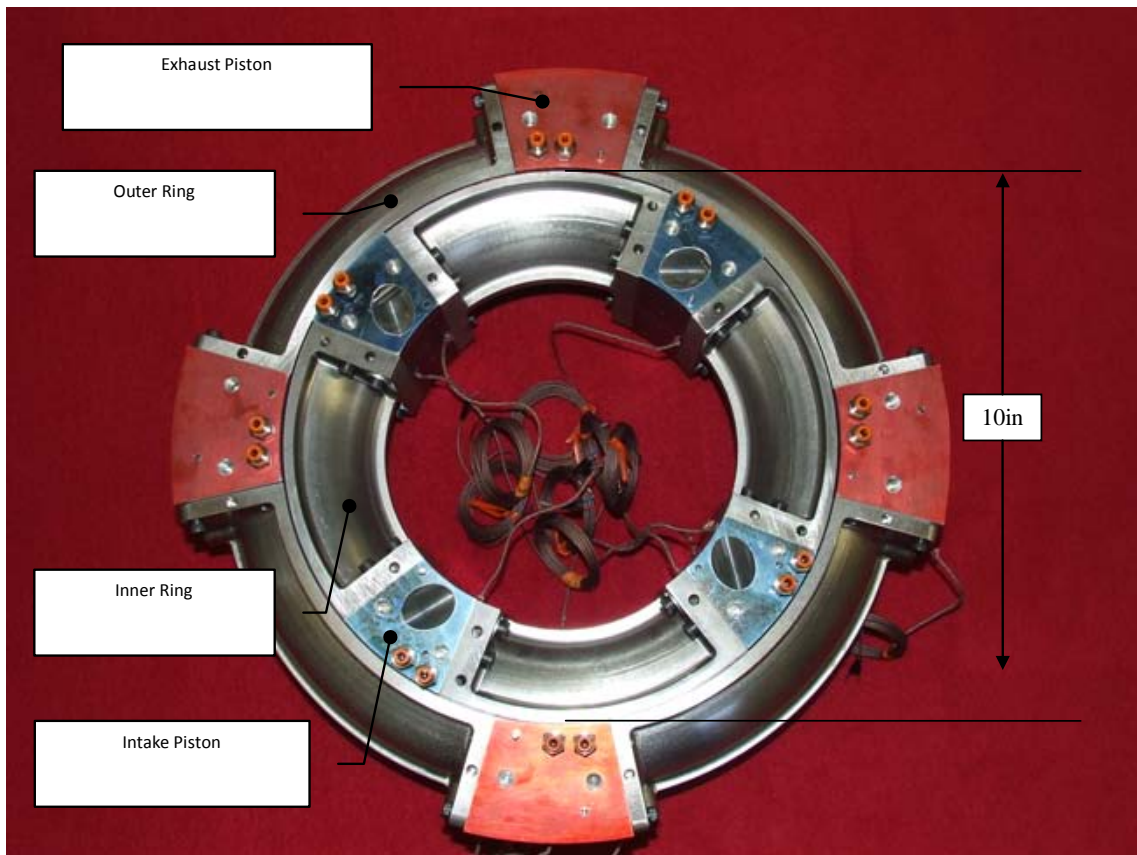


Figure 4.4: 8-Chamber HiPerTEC Tested with Control System

Each chamber is bounded by two pistons; one piston connected to the inner ring and one connected to the outer ring. During operation, pressure changes occurring in a chamber will act against one piston connected to the inner ring and one connected to the outer ring. When combustion occurs in a chamber, the two pistons bounding the chamber will be forced apart, causing the rings to move in opposite directions (counter rotating). Four chambers will decrease in volume, while the remaining four will increase in volume. This is illustrated in Figure 4.3 with four positions representing the 4-stroke cycle operation. (Note that the pistons in Figure 4.3 are depicted as single lines.)

Since this engine design has eight chambers, two chambers will be going through the same stroke in the cycle simultaneously. In position 1, chambers 1 & 5 are beginning the power (combustion) stroke where the two pistons bounding chamber 1 are close together, or at TDC. Combustion is initiated forcing the two pistons to move in opposite directions. All inner ring pistons move together and all outer ring pistons move together. Chambers 2 & 6 are in the compression stroke, 3 & 7 are in the intake stroke, and 4 & 8 are in the exhaust stroke. Note that the outer and inner rings rotate a maximum of 45 degrees (assuming pistons with zero thickness and an infinite compression ratio) with this eight chamber arrangement. The maximum rotation angle of the rings is dependent upon the number of engine chambers and physical sweep of the pistons.

Position 2 represents full expansion, BDC, of chambers 1 & 5. At this point, chambers 1 & 5 are starting the exhaust stroke, 2 & 6 the power stroke, and so on around the engine. The chambers undergoing a power stroke are providing the energy to carry out the strokes in the other chambers. There are always two chambers undergoing a power stroke for each engine stroke. This process continues until all the chambers go through a complete 4-stroke cycle; the cycle then repeats continuously.

Power extraction can be achieved by driving a turbine with the exhaust gases, or generating electrical power with a generator, or mechanically through a motion conversion unit. ATS has extracted mechanical power via a motion conversion unit (MCU) of its own design. This unit uses two mechanical diodes to convert the reciprocating motion from the engine drive shaft to continuously rotary motion at the output of the MCU.

4.4 Control System Overview

4.4.1 Hardware

The HiPerTEC ECU (hardware and software) was operated as a prototyping ECU in a laboratory environment. It is comprised of a PXI embedded controller with a forty channel, high-speed, multifunction DAQ module, a user-programmable FPGA module, powertrain control modules for driving things like ignition coils and fuel injectors, and various sensors and actuators as detailed in Table 4.2. The embedded controller serves as the main processor of all control actions of the engine and collection of necessary sensor information. To assist the embedded controller, an FPGA module is used to handle synchronous timing of fuel injection, spark ignition, throttle position, and valve operation. This module is primarily responsible for tracking engine position and ensuring critical events happen at the desired engine position. Unloading this task from the main controller reduces the performance requirements of the hardware and software to a level necessary for deterministic control. It is from this module that all powertrain control modules receive their inputs for driving system devices.

Specifically, the ECU is responsible for control of the following subsystems (simplified layout shown in Figure 4.5): the engine position tracking system, the spark ignition system, the fuel delivery system, the air intake system, and the starter sys-

Table 4.2: Control System Components

Hardware	Description	Function
PXI-8770	Quad-core 2.2GHz Pentium controller	Real-time control and data analysis
PXI-1042	8-slot PXI chassis	Physical Chassis
PXI-7813R	FPGA Module	Engine position tracking and synchronous actuation
PXI-6255	40ch DAQ Module	High-speed DAQ
NI 9411	High-speed DI	Engine and valve position
D000017	Throttle Driver	Control intake throttle
D000017-ESHB	Engine Synchronous H-bridge	Controls valves
D000006	PFI Driver	Actuates fuel injections
D000012	Spark Driver	Fires spark plugs

tem. The engine position tracking system is made up an optical, incremental, rotary encoder and high-speed digital input module for the FPGA module. The encoder is geared to the main gear attached to the outer ring of the engine. As the engine rotates, the encoder generates A, B, and Z (index) pulses that are decoded at the FPGA level for position tracking. Quadrature decoding is used to maximize positional resolution for the timing of events. The spark ignition system is made up of two spark driver modules, eight inductive ignition coils, and eight spark plugs. The fuel (gasoline) delivery system is made up of two PFI driver modules, eight high-impedance PFIs, a fuel pump, a fuel filter, and a pressure regulator. The air intake system is made up of one output of a throttle driver module, one electronic throttle body, an electrically driven supercharger (to enable future 2-stroke operation), four ESHB driver modules (for controlling pilot stage of poppet valves), sixteen 4-way solenoid valves, eight exhaust valves, eight intake valves, and two pilot valve supply shut-off valves. The starter system is made up of the other output of the throttle driver module, one 4-way solenoid valve, and a pneumatic rotary actuator. Fig-

ure 4.6 shows the HiPerTEC installed in the test cell with all of its sensors and actuators.

Along with controlling all the actuators, the ECU also functions as the DAQ system. Forty differential analog channels are sampled at 5 kHz apiece for acquiring data such as chamber pressure (for each chamber), manifold pressure, mass air flow, mass fuel flow, air intake temperature, exhaust gas temperature, and piston temperature. This data, along with engine position, throttle position, and all the timing signals for valve control, fuel injection, and spark ignition, is collected for use in the control actions as well as engine operation analysis.

4.4.2 Software

The control code was written in NI LabVIEW and executed as three distinct programs; see Figure 4.7. The user interface program controls requests from the user, such as setting the base timing values, changing the maps for fuel injection pulse width and spark advance, enabling and disabling subsystem functionality, recording data, and initiating the starting routine. This program is implemented as a sequential state machine, running in a single loop on a non-deterministic operating system.

The main control program handles the requests from the user interface program, retrieves and processes the sensor information, determines the control actions that need to occur, and sends the appropriate information to the FPGA. This program interfaces with the code running on the FPGA, but does not execute the control actions directly. This program is implemented as parallel loops with each loop running a sequential state machine tailored to a specific engine subsystem; see Figure 4.8. By taking advantage of the quad-core processor and real-time operating system, all loops can deterministically run in parallel on the embedded controller, also illustrated in Figure 4.8.

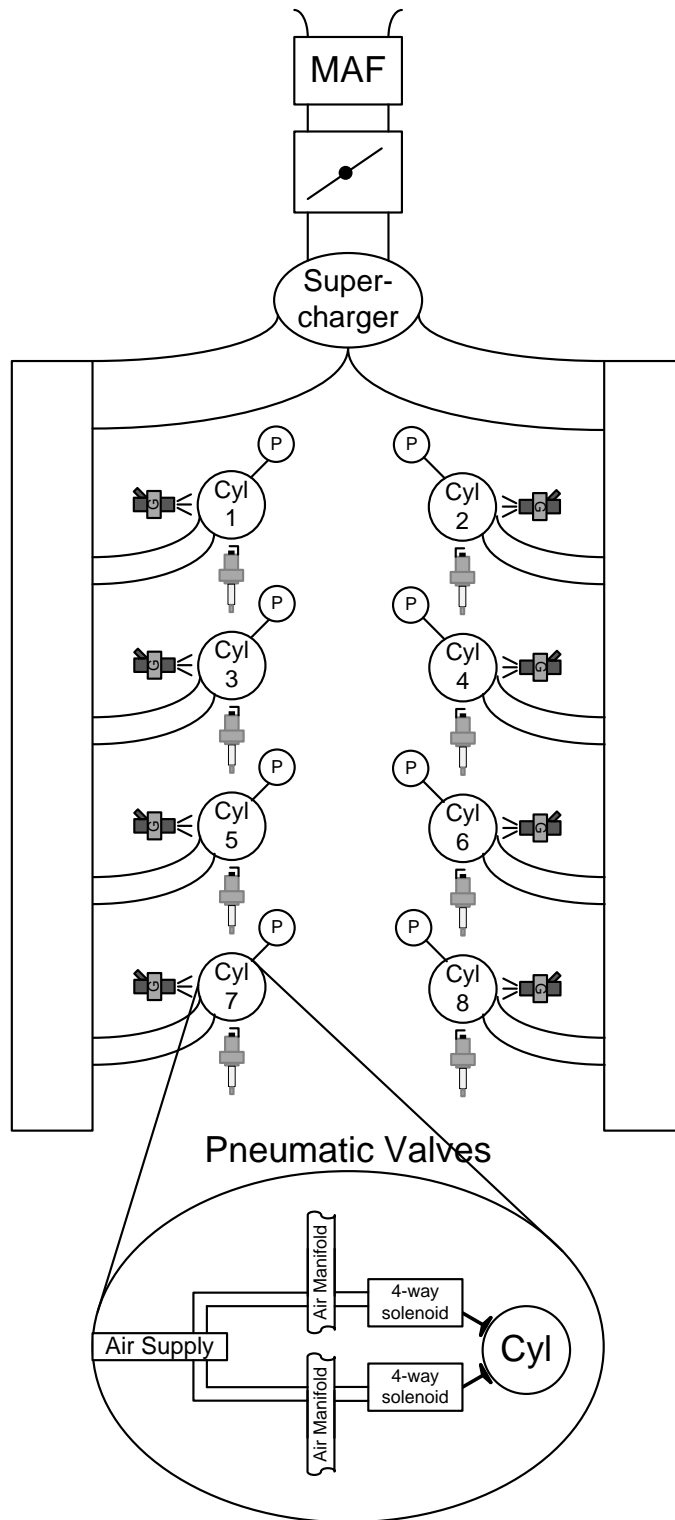


Figure 4.5: Simplified System Layout Diagram

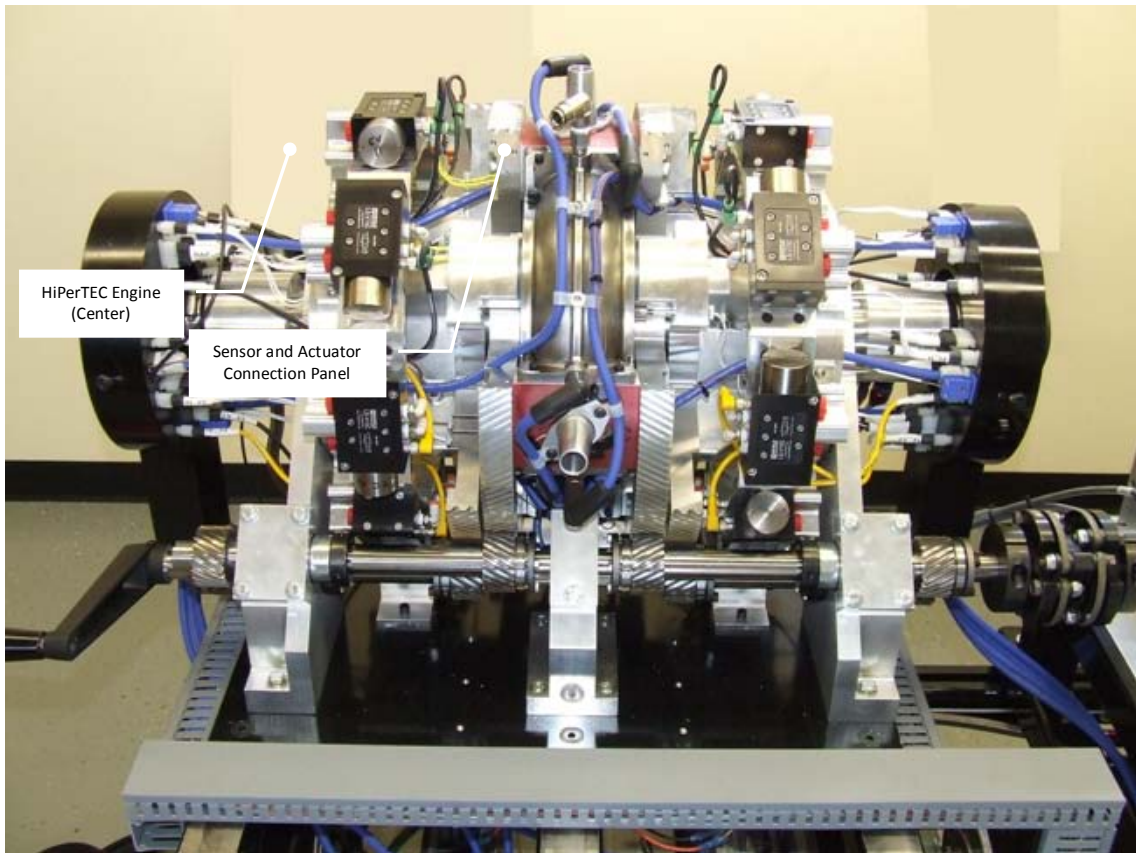


Figure 4.6: HiPerTEC Installed in Test Cell systems to the left and right

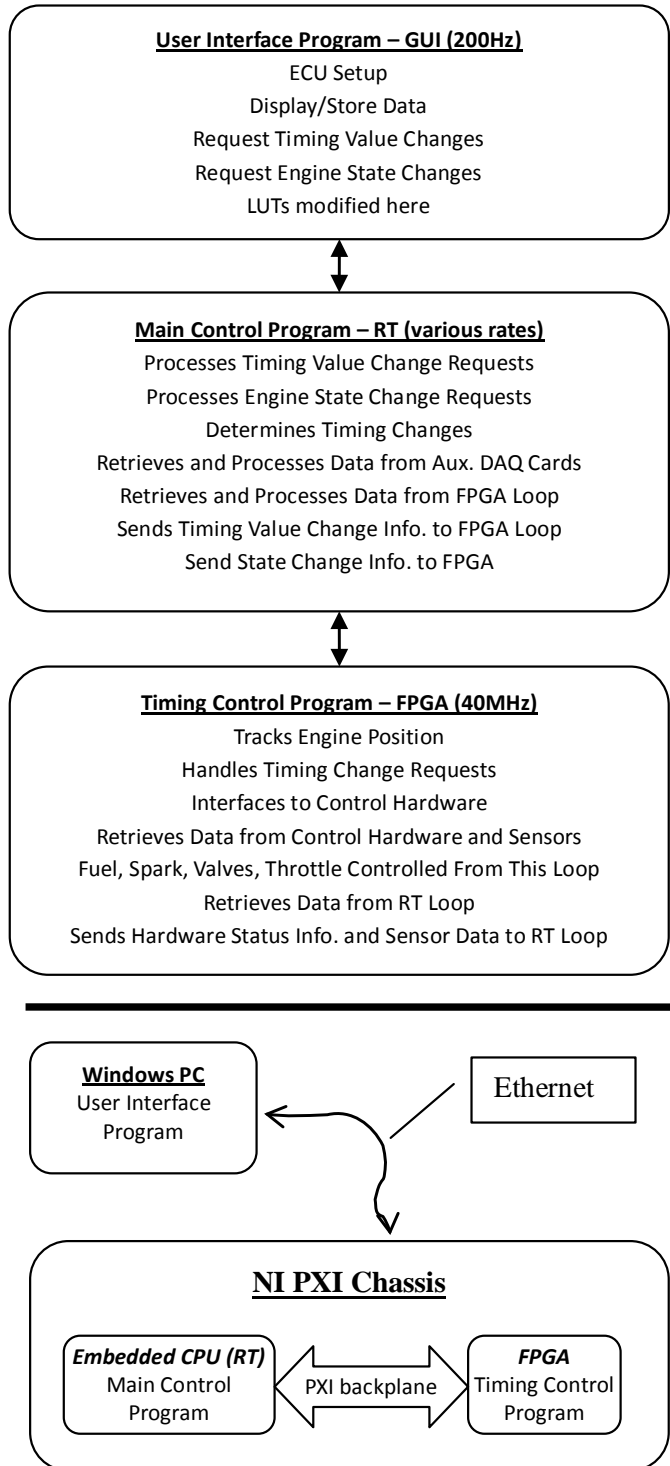


Figure 4.7: Diagram of Engine Code

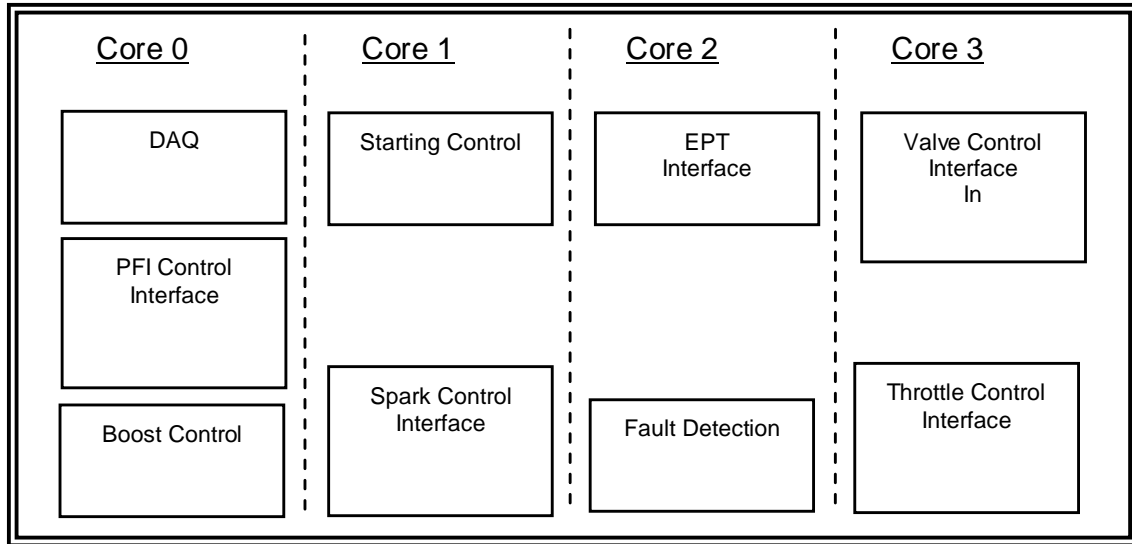


Figure 4.8: Main Control Program Loops and Core Assignment

The timing control loop tracks the engine position and ensures proper timing, interfaces to the control hardware, processes requests from the main control program, and sends hardware status and sensor data to the main control program. This program is implemented as a Single Cycle Loop (SCL) at a clock rate of 40 MHz.

4.5 Engine Position Tracking

In a conventional 4-stroke slider-crank engine the engine position is tracked by measuring a camshaft and crankshaft position sensor. Because the crankshaft revolves at twice the rate of the camshaft, the camshaft sensor is usually only used to determine engine phase (compression vs. exhaust stroke). All precise timing calculations are done based on the camshaft sensor(s).

A variety of camshaft encoder patterns are available on engines, but the vast majority can be broken in to three classes. The most common is N-M where N represents the tooth spacing (N=36 for 10 degree spacing) and M the number of missing teeth. So a 36-2 tooth pattern would have 34 teeth set on 10 degree centers with two adjacent missing teeth as shown in Figure 4.9.

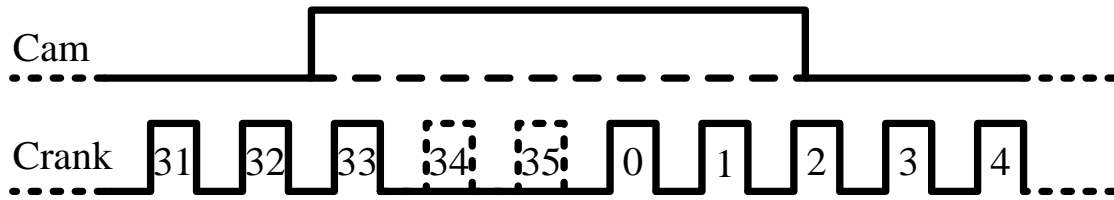


Figure 4.9: 36-2 (N-M) Camshaft Encoder Pattern

An N+1 pattern is shown in Figure 4.10 and typically has fewer teeth. It is not uncommon to find this pattern on the cam of a heavy duty engine acting as a backup to the N-M on the crank.

A pattern commonly found on research engines and in heavy duty stationary engines is the “encoder” pattern. This pattern consists of N evenly spaced teeth with a single reset pulse on the crank. Because a camshaft often experiences minor changes in phase compared to the crankshaft, especially on timing belt engines, a crank sensor to mark TDC is still common.

Optical shaft encoders used in research often provide A, B, and Z outputs. The Z output is a once per rev pulse. The A and B outputs contain N evenly spaced pulses that are offset by 90 degrees. This configuration is called quadrature and allows shaft direction to be determined by looking at the level of the B line on a level transition of the A line and vice-versa.

Generally the position pickup is at lower resolution (10CAD) than what the ECU wants (0.1CAD). Actual engine position is extrapolated in real-time from previous real tooth edges and an extrapolated crank angle position is derived. This is used to feed synchronous output blocks.

Because the speed of engine position tracking and output generation is in the sub-microsecond range, it is usually done with a dedicated co-processor like the Freescale TPU. TPUs and similar counter-timer architectures have limited flexibility and are difficult to program and debug. An alternative is to use an FPGA. In

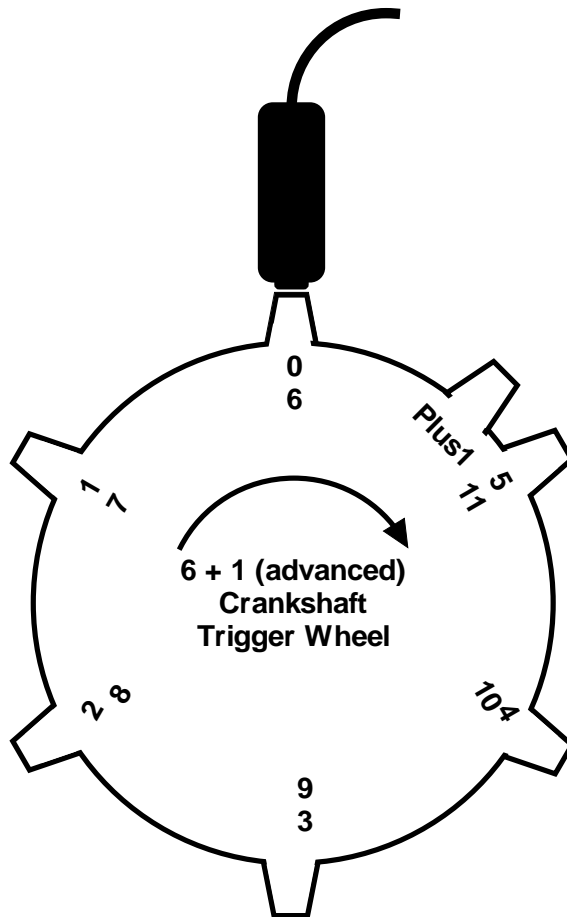


Figure 4.10: N+1 Camshaft Encoder Pattern

this case a Drivven EPT (Engine Position Tracking) core was run on a National Instruments FPGA module.

The Drivven EPT software extrapolates the number of teeth on an actual engine by a power of two (2,4,8,16, etc). For instance, the above 36-2 encoder could be extrapolated by $2^5=32$ to get 1152 crank angle ticks (CAT) per crank rev or 2034 CAT per 4-stroke cycle. All calculations are done every 25ns. All crank angle synchronous events (fuel injection, spark, etc.) are defined to happen exactly on CAT edges. LabVIEW RT level functions are provided to map from Crank Angle Degrees (CAD) to CAT and back.

The extrapolation mechanism is subject to errors due to engine acceleration and deceleration, both gross and in-cycle. In the case of deceleration the extrapolated pulses will run out and the CAT state machine will suspend operation until the next physical tooth is recognized. In the case of acceleration a new physical tooth is recognized before the extrapolated CAT have run out. In this case the CAT is advanced at the maximum allowable speed. This ensures that every CAT is seen by all the device I/O drivers (Spark, fuel, etc.) exactly one time and at as close to the theoretically perfect time as possible with what can be known with the given tooth pattern.

4.5.1 Types of Outputs

Three primary types of pulse generation are available based on their start and end conditions: Angle-time, Angle-Angle, and Time-Angle. Different physical actuators map better to different types of command sequences.

Angle-Angle sequences are the simplest and as the name implies they are defined by their start and end angle CAT. These functions map well to valve timing events or to knock window generation.

Spark events are time-angle events where an RT-level dwell time is specified in ms. The current engine speed is used to approximate a start angle where Dwell will begin. When the end angle is reached the spark will be initiated regardless of the actual dwell time achieved.

Fuel injection may either be angle-time or time-angle based on the desire of the system designer. Time-Angle events are often used to try to get the injection to end just before intake valve opening. In this case time is the dominant variable and will take precedence over end angle when determining when to turn off the injector.

4.5.2 Free-Piston EPT

The HiPerTEC engine uses a variation of the “encoder” pattern as shown in Figure 4.11. This configuration uses a quadrature encoder scheme to determine engine position. Because the valvetrain is cam-less there is no cam shaft sensor. The control system simply assigns phase during the cranking process and commands the valves accordingly.

When the piston makes a change of direction, some of the encoder pulses beyond the direction-change point will not be observed. When the direction change is detected, the EPT will rapidly advance through cranks angle ticks in the un-observed section until it catches up to the real engine location, allowing the specification of angles within the unobserved region.

4.6 Control System Operation

As previously mentioned, the focus of this paper is on the start-up and warm-up phases of engine operation. Figure 4.12 shows a plot of the engine position throughout these phases. Figure 4.13, Figure 4.14, and Figure 4.16 show zoomed sections of this figure.

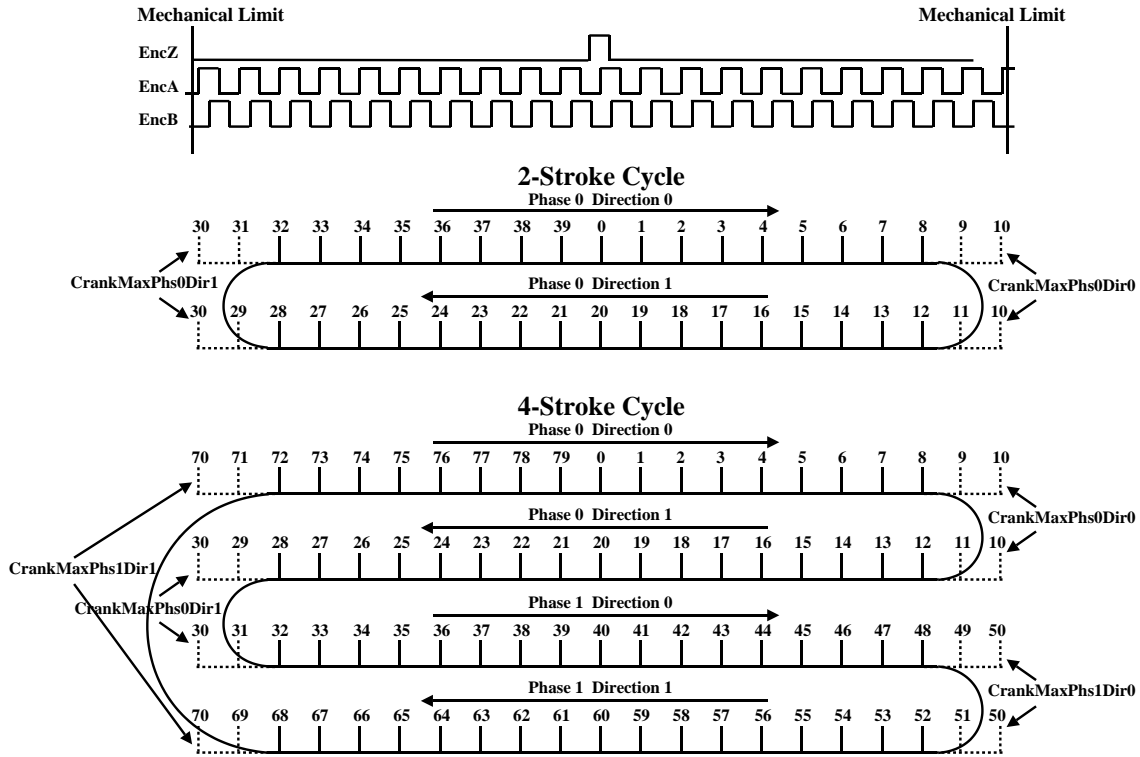


Figure 4.11: Free-Piston Encoder Pattern.

At this stage in development, the after-start and warm-up phases are not being controlled directly by the ECU. These phases are merely recognized during post-test analysis and control parameters are manually adjusted based on what is observed. The engine runs in open-loop mode throughout these phases, with control values selected prior to a test and manually adjusted during the test. As development progresses with the engine and ECU, these phases will be studied in more detail so as to gain control of their behavior. On the contrary, the cranking phase is completely controlled by the ECU and is discussed in more detail below.

The reciprocation of the engine drive shafts precludes the use a standard automotive starter. Instead, a single vane pneumatic rotary actuator (FPE starter) has been employed to reciprocate the engine and bring the desired air/fuel mixture into the chambers; see Figure 4.15. Before the starting sequence is initiated, the engine is rotated into position to allow synchronization (SYNC) of the EPT sys-

tem. After the engine is in position, the starting sequence is initiated through the User Interface program. The air supply valves for the pilot valves are opened, the throttle position and boost are set to the desired level, the supercharger is turned on, and the air solenoid for the FPE starter is energized. This causes the engine to rotate and SYNC the engine position (Figure 4.13) and marks the beginning of the cranking phase. Once SYNC is active, the valves begin to open and close at the set engine position.

Engine synchronous events have been defined in terms of included angle degrees (IAD). IAD is the angular displacement between two opposed pistons that bound one chamber. Since the relative motion of the pistons is only 32 degrees, the following conversion is employed to map IAD to CAD for the purposes of comparison to conventional engines as well as to satisfy the device driver software.

$$\text{Max.HiPerTECStrokeAngle} = 32\text{deg}(IAD)$$

$$\text{Rotationfor1StrokeofConventionalEngine} = 180\text{deg}(CAD)$$

$$\frac{180CAD}{32IAD} = 5.625 \frac{CAD}{IAD}$$

During the cranking phase (Figure 4.14), the Main Control program monitors the engine position and switches the state of the FPE starter to stop the engine and rotate it in the other direction. This turnaround position was specified empirically to optimize the initial combustion event, which is typically a tradeoff between maximizing stroke length and maximizing engine speed. Also during cranking, the supercharger maintains boost in the intake, and air is brought into the chambers, compressed, and then exhausted, but without ignition.

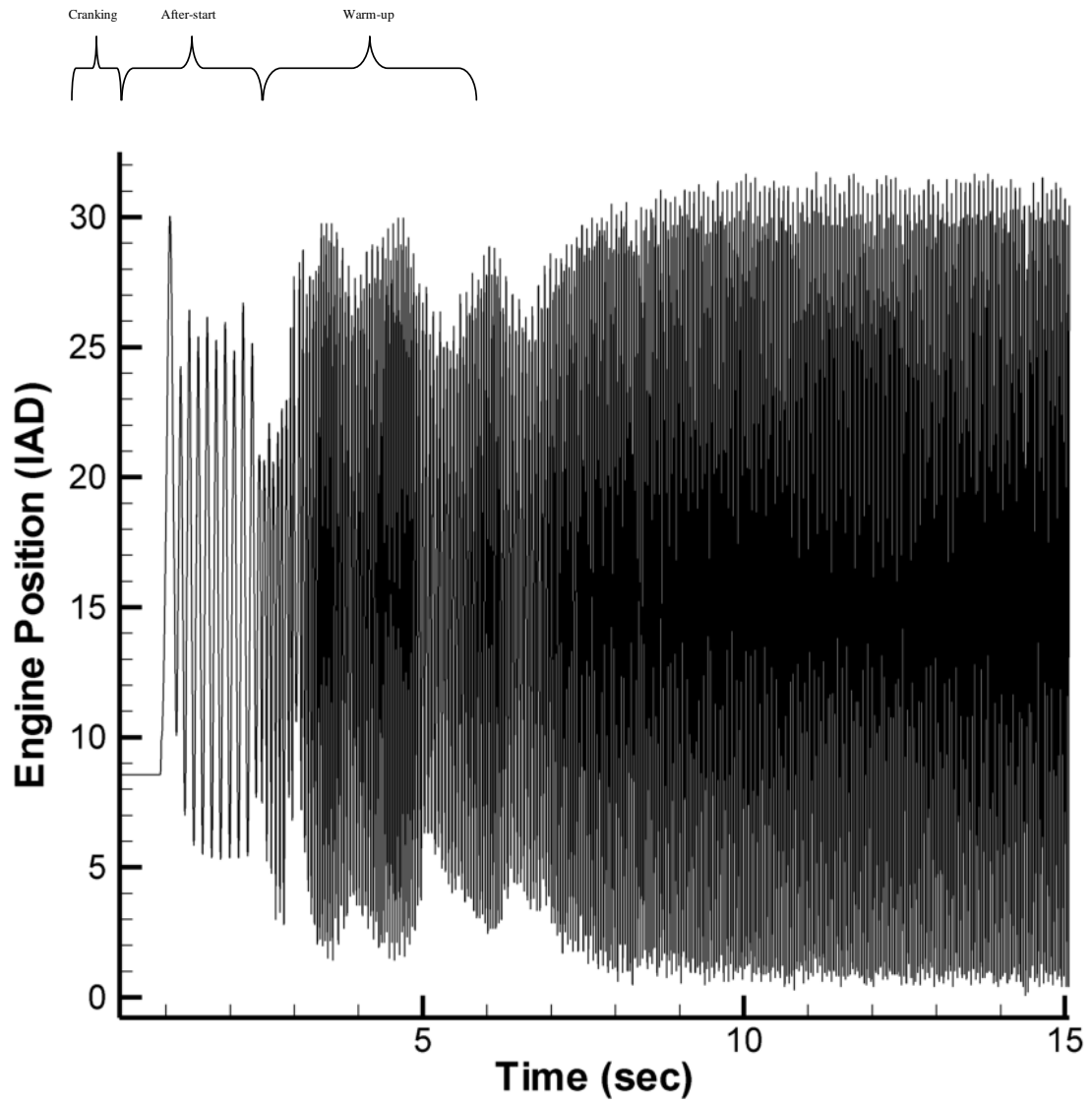


Figure 4.12: Engine Position with Cranking, After-Start, and Warm-up Phases Labeled

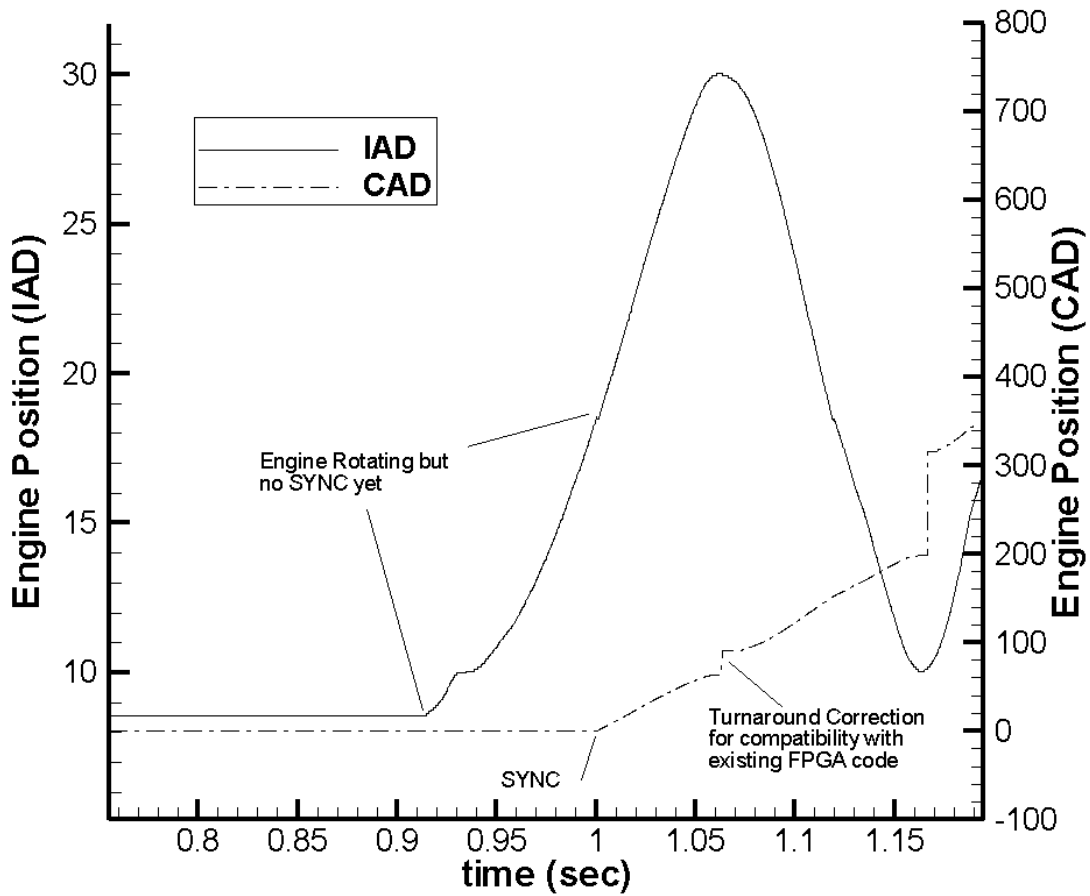


Figure 4.13: Engine Position SYNC

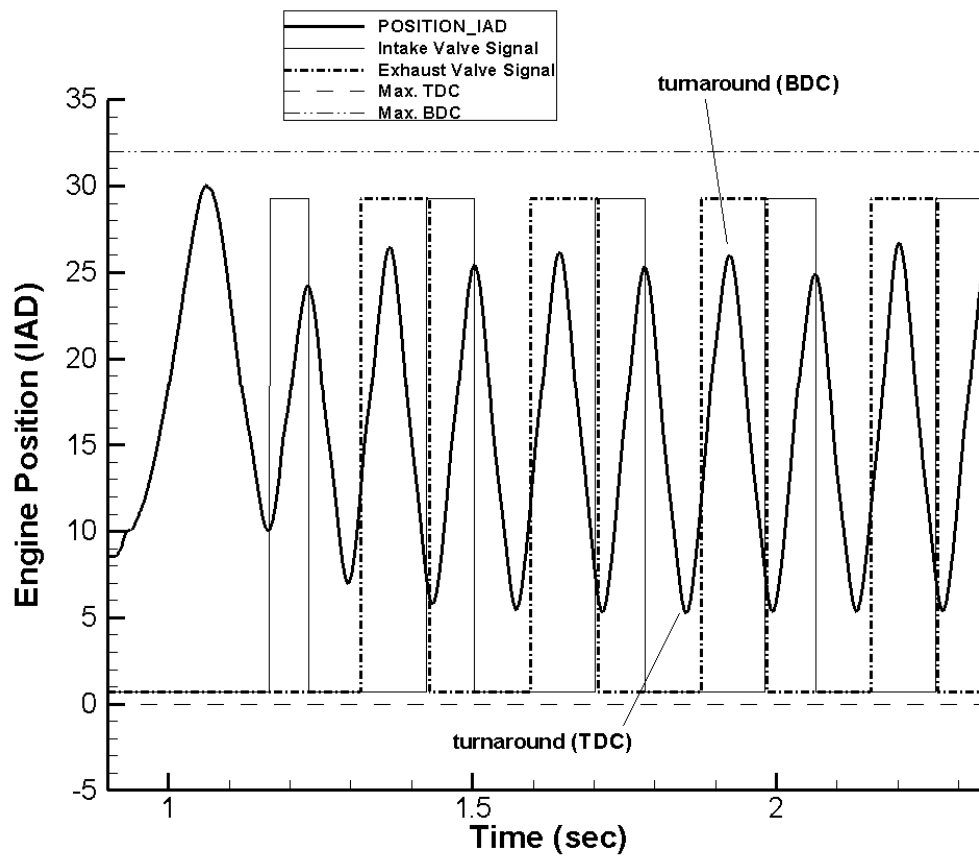


Figure 4.14: HiPerTEC Cranking Phase

Due to the way this FPE starter works, the ignition system cannot be enabled until flow into the engine has been established and the desired starting chamber has been charged with fuel and air. Unlike the conventional automotive starter, the FPE starter does not “kick-out” when a combustion event overdrives it. Maintaining the positional relationship between the engine and the starter is critical to avoid cranking the engine past TDC or BDC; allowing the starter to “kick-out” would require it to automatically reconnect on a no-start, which would require automatic alignment in relation to the engine position. While this would be the preferred method for reliability and robustness, manually resetting the starter after each no-start was found to be sufficient for laboratory work.

To eliminate the additional load from the starter on the first combustion event and thereafter, the starter is decoupled from the engine with a release mechanism as shown in Figure 4.15. This means that on the stroke before the first combustion event, the FPE starter drives the engine to a certain position and decouples from the engine, while the engine continues towards the ignition point and TDC. This process is illustrated in Figure 4.16. On a no-start the starter is manually coupled back to the engine.

The engine position for the first ignition event is set by knowing the chamber pressure and the final engine position after it has been decoupled from the starter. Both of these values were determined by cycling the engine through the starting sequence without fuel and examining the pressure and position traces and observing the peak chamber pressure and stopping position of the engine after being released from the starter. The optimal ignition position is set to correspond with peak chamber pressure prior to reaching TDC. The startup event can be optimized further by modifying the release angle of the starter which in turn changes the pressure/time profile of the engine through the first stroke.

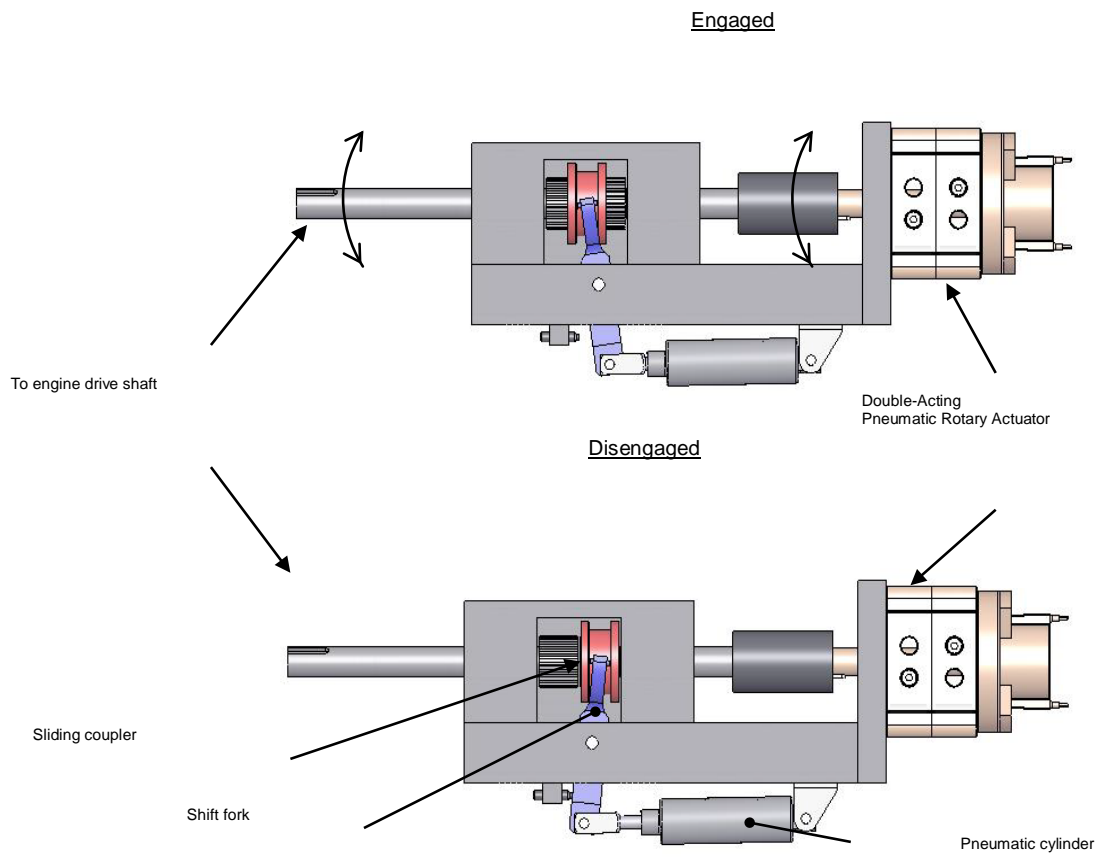


Figure 4.15: HiPerTEC Starter - Pneumatic rotary actuator and release mechanism

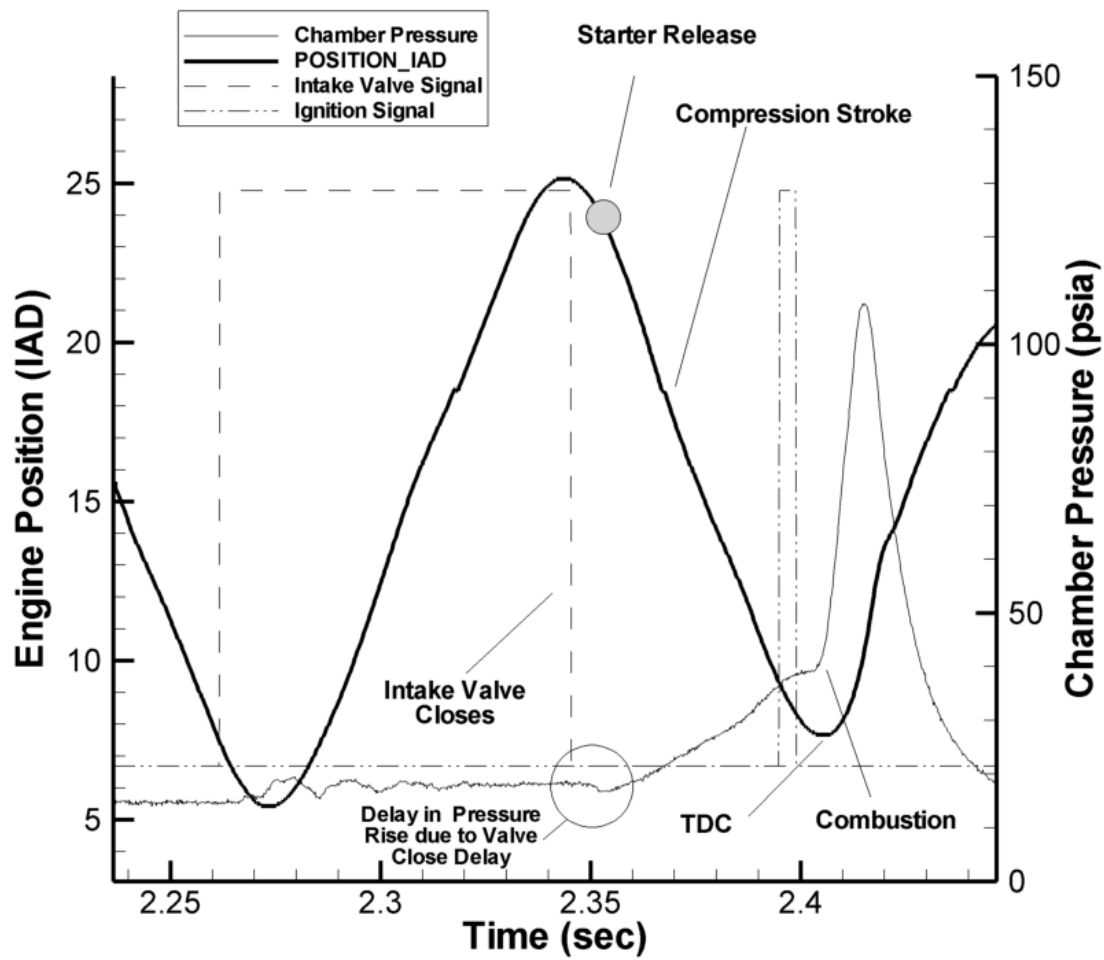


Figure 4.16: Final Stroke of Cranking

TDC and BDC for the HiPerTEC refer to the position of direction change of piston motion. This represents the minimum or maximum chamber volume, respectively, before piston turnaround. For the HiPerTEC, the TDC and BDC positions can vary for each stroke. To eliminate complications in setting timing values based on variable TDC positions, the extreme TDC location has been utilized. The extreme TDC position would represent the minimum IAD and minimum chamber volume bounded by two opposed pistons. Likewise, the extreme BDC position would represent the maximum IAD and the maximum chamber volume. TDC can also be referred to as face-to-face as this is the position where the two opposed pistons meet.

During cranking the throttle position, boost pressure and valve timing are kept constant to minimize the variation in air flow rate. Stroke length is allowed to vary naturally as this has little effect on mass flow during cranking. Gasoline is injected on the final cycle of cranking and is also based on a target equivalence ratio and estimated intake air mass. This is done on the final cycle to avoid flooding during cranking.

Following the first combustion event, the combustion pressure drives the pistons in opposite directions and triggers after-start phase control by the ECU. In after-start control the gasoline pulse width is ramped down towards the equivalence ratio set for warm-up (1.0) and a separate spark timing table is utilized until the stroke length stabilizes. Spark advance is open-loop based on engine position until the stroke length stabilizes, at which point it is switched to speed dependence. For control operations the cycle-averaged engine speed is used and is the independent variable for use in lookup tables.

When referring to the speed of the HiPerTEC, frequency is preferred because RPM refers to the time involved for a revolution of the crankshaft which is not present in this FPE. Instead, what should be reported is the time for the HiPerTEC

to complete two strokes or one complete cycle of piston motion. This is equivalent to one revolution of the crankshaft and allows direct comparison with conventional engine speeds. For example, if the HiPerTEC were running at 40 Hz this would be equivalent to 2400 RPM of a conventional engine as illustrated below:

$$40 \frac{\text{cycle}}{\text{s}} * \frac{2\text{stroke}}{\text{cycle}} * \frac{1\text{rev}}{2\text{stroke}} * \frac{60\text{s}}{1\text{min}} = 2400 \frac{\text{rev}}{\text{min}}$$

For easy comparison, engine speed in this paper has been given in terms of equivalent crankshaft RPM instead of FPE frequency.

Once the engine passes through the after-start phase, which is around three to five seconds long, the warm-up phase begins. As illustrated by Figure 4.17, this phase is characterized by a stabilized stroke length and engine speed. The ECU maintains the same throttle position and level of boost as the after-start phase throughout this phase. Spark timing is controlled in open-loop mode and advanced or retarded based on engine speed. The gasoline pulse width is also controlled in open-loop mode and is determined by the air mass brought into the chamber and the equivalence ratio. A MAF sensor measures the flow rate of air brought into the intake manifold during the intake stroke and this flow rate is integrated over the time of one stroke to determine the mass of air that entered the chamber. The gasoline pulse width is then calculated using the air mass, equivalence ratio, and stoichiometric fuel/air ratio for gasoline. Closed-loop control of the air/fuel ratio has not been employed, as engine warm-up (steady state temperature) has not yet been achieved.

4.6.1 Valve Operation and Timing Control

Because the HiPerTEC is cam-less, an alternative valve actuation method was needed. Previous work with pneumatic valve actuation provided the necessary

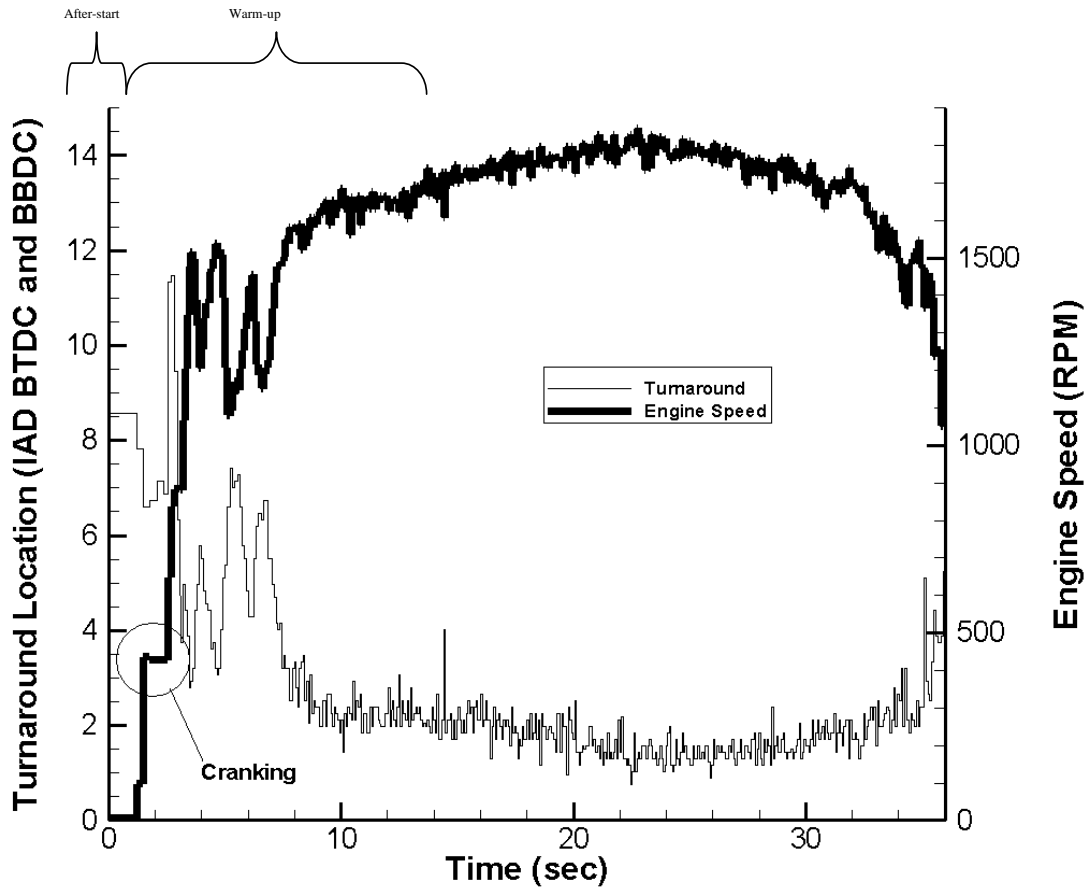


Figure 4.17: HiPerTEC Warm-up Phase

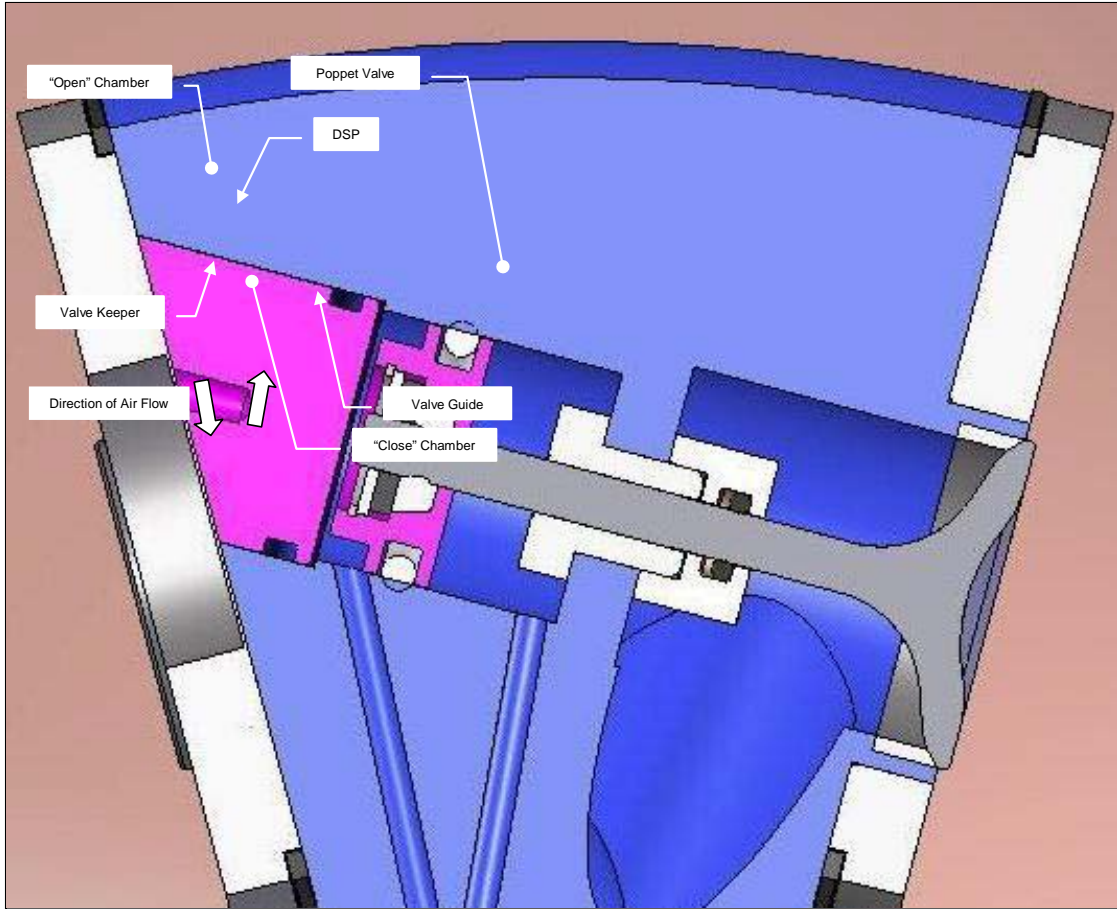


Figure 4.18: HiPerTEC Intake Valve

ground work to develop a pneumatic valve actuation system. Conventional poppet valves from a motorcycle engine were modified to fit the HiPerTEC. A double-sided piston (DSP) is used for both opening and closing the valve, as shown in Figure 4.18. Compressed air controlled by a 5-port 4-way proportional solenoid valve is used to actuate the valve in both directions resulting in spring-less operation.

To open the valve, a signal is sent from the ESHB module, via the FPGA, to move the solenoid to one extreme position. In order to drive the DSP an engine-synchronous bi-polar driver was required. While Drivven did not have such a product in their portfolio they were able to exploit the flexibility of the FPGA and quickly

generate code to turn normal throttle driver modules in to engine synchronous bipolar actuator drivers.

Air from a regulated supply (140 psi) pressurizes the opening chamber of the DSP and lifts the poppet valve off its seat. The opening chamber remains pressurized and the valve open, until the valve needs to be closed. At that time a signal is sent to the solenoid to move it to the other extreme position. This causes the opening chamber to begin exhausting and the closing chamber to begin charging. No movement of the poppet valve occurs until the force on the valve from the closing chamber is greater than that from the opening chamber. Once the force from the closing chamber prevails, the valve closes against its seat. Open and close valve positions are set by physical stops and valve bounce due to high velocity at landing has not be addressed. It is recognized that much work has been done on cam-less valve actuation (Moran, 2003) and there is certainly room for improvement in this implementation.

Because these valves are actuated differently than cam driven ones, valve timing is also done differently. There is a finite delay between the time the signal to switch the valve state is sent and when the action actually occurs; see Figure 4.16. This delay is insignificant at low engine speeds (≤ 400 RPM), but compensation is needed as the engine speed increases. The total delay is a result of many factors, with a few of the major ones being: engine chamber pressure, inertia, DSP chamber charging and exhausting time (flow rate into and out of DSP) and friction. The total delay is on the order of 5ms from when the signal is sent to when the valve starts to open or close. This value is a composite of valve bench testing and results from computer simulations that included inertia, engine chamber pressure, and valve stem friction at various simulated engine frequencies (5Hz to 83Hz).

Valve “open” timing values are entered as the desired engine position for start of open. Valve “close” timing values are entered as the desired engine position

where the valves are fully closed. To properly set the delay for the closing events, 2ms is added to account for the time to close the valve, making a total delay of 7ms. This delay and the opening delay are factored into the timing in order to ensure the valve events happen at the desired position as the engine speed changes. To do this a linear relationship is used to determine the advance that is needed. It is acknowledged that the engine speed is not constant throughout the entire stroke and therefore a higher order relationship or LUT may be needed in different regions of the stroke and under load. However, the current approach has proven to be adequate for the speed range encountered during no-load warm-up.

The above mentioned linear relationship, relates valve actuation delay, engine speed, and valve timing advance. A delay value for the opening and closing of each valve is specified based on how they are performing. This allows fine tuning of the gas exchange process on a chamber-by-chamber basis, much like the tuning of fuel pulse width. At the middle of each stroke, the FPGA (Timing Control program) calculates the instantaneous speed. This speed is then paired with corresponding valve events for that stroke and an advance value is calculated. The desired engine position minus the advance sets the engine position at which the open/close signals need to be sent from the ESHB module. It is the primary function of the FPGA to track the engine position and send this signal at the proper time. Valve timing values are updated once per cycle.

4.7 Conclusion

A prototype version of the HiPerTEC engine was successfully run on a Driven control system. The low-level control of a myriad of novel actuators as well as the tracking of a dynamically varying piston position was accomplished. The engine was operated in an open-loop fashion and datasets were built to allow the development of a model-based control strategy.

Valved, 4-stroke, free-piston engine operation was shown. This is believed to be among the first for free-piston engines (Mikalsen and Roskilly, 2008b) and demonstrates control and combustion techniques that were never before available to free-piston engines.

4.8 Future Work

While the low-level control of actuators was proved out in this work, much attention to the high-level control is required and will be enabled by a more thermally robust prototype, currently in the design phase. A full transient control application will be developed for use with this engine as it approaches final deployment.

4.9 Acknowledgments

We would like to thank Rick Eason for his initial code development work and Ryan Gauthier for laying the ground work in understanding the cycle operation of the HiPerTEC. Also, thanks to Ken Hoose and Mitch Cyr for their help in developing the LabVIEW code.

Chapter 5

DIGITAL SIGNAL PROCESSING AND CONTROL

Engine controllers have traditionally used some off-chip peripherals to implement specialized signal processing. The most common of these is Knock detection. Knock ICs from TI/National Semiconductor (LM9011, 2009), Bosch (Bosch CC195, 1996), and Freescale (Freescale, 2001) are all readily available, but have limitations in filter type and the decision tree for knock detection.

More recently, systems have used dedicated digital signal processing chips, chips optimized for MAC instructions, critical for FFTs and other signal processing math. These programmable solutions offered better configurability of specialty applications and were applied to off-highway applications.

These same algorithms can be implemented in an FPGA. With a large FPGA this can be simply a subcomponent of the larger design. Since these knock algorithms require input from the engine position tracking software, a discrete solution requires many additional lines between the processor and IC. These lines and their associated complexity are eliminated in the pure FPGA solution.

While knock is a well understood and handled problem from the controls side, there are new areas where similar approaches can be used. The foremost of these is cylinder pressure analysis. Cylinder pressure analysis is available in most research engines and is now standard on a number of passenger cars today like the GM 2.0L I-4 Turbo-Diesel and the VW 2.0L TDI (Dorenkamp and Gruber, 2008). Cylinder pressure feedback for real-time control has been examined in research for years, but as it becomes a reality in production vehicles it has become a central part of numerous control strategies.

The following paper examines two versions of real-time cylinder pressure feedback. It is a follow-on work to (Viele and Quillen, 2009) that examined the computational feasibility of "next-cycle control" in the Driven DCAT architecture. The first is the common case of "next-cycle" or possibly multi-cycle control. In this scenario, cylinder pressure is analysed over the course of a cycle and the results are used in the calculation for the next cycle. While this takes dedicated ECU hardware, it is well within the realm of standard automotive digital signal processors.

The second variant examined exploits some of the capabilities only offered in FPGAs and custom ICs. In this version cylinder pressure is sampled every $10\mu s$, heat release calculations are performed, and results are acted on by the injection system. In the case examined we controlled the inter-pulse delay of a multi-pulse diesel injection system. This paper simply explored the mechanism by which same-cycle control can be implemented and demonstrated the ability to do it in a standard FPGA-based research controller. Similar work is ongoing at several universities to take this to the next step and show emissions and operating regime improvements based on same-cycle control.

For this paper the specific contributions of this author are:

- Design of the combustion analysis system architecture.
- Same-cycle design, concept, and algorithms.

Kristopher Quillen was responsible for coding the DCAT software.

Steve Ciatti was responsible for running the engine and collecting the data.

Reprinted with permission for ASME.

Chapter 6

ICEF2010-35119

Next-Cycle and Same-Cycle Cylinder Pressure Based Control of Internal Combustion Engines

This paper was presented at the ASME Internal Combustion Engine Division Fall Technical Conference in San Antonio, TX Sept 12-15, 2010. The paper was written by Matthew Viele and Kristopher Quillen of Drivven and Steve Ciatti of Argonne National Laboratory.

6.1 Synopsis

This paper reviews next-cycle and same-cycle control techniques developed by Drivven and implemented at Argonne National Laboratory on a General Motors 1.9L common rail diesel engine. Next-cycle control involves measuring cylinder pressure engine-synchronously, performing calculations, and using the complete result in the control algorithm for the fueling event in the next engine cycle. For this control method, injection timing was manipulated to maintain a specified 50% burn location and was investigated in both single-cylinder and multi-cylinder modes. The engine was tested in steady state with step changes in input parameters such as EGR rate.

Similar to next-cycle control, same-cycle control involves measuring cylinder pressure engine-synchronously, performing calculations, and making fueling decisions based on partial results within the same engine cycle. For this control method, injection pulse spacing was manipulated to optimize the heat-release calculated angle-by-angle. Next-cycle and same-cycle control both have the capability of enhancing production engine control while next-cycle control also has great benefits as a calibration aid.

6.2 Introduction

For many years combustion analysis systems have been a fundamental part of engine research and development (Zhao and Ladommatos, 2001). These systems use a number of cylinder pressure sensors and a crank shaft coupled timing device (typically an optical encoder) to extract data from the engine. The analysis system records and displays this data. It then performs calculations on the data to provide derived calculations of interest. Much of engine development is based on optimizing one of these parameters while keeping others within suitable ranges.

In the past few decades electronic control has been a requirement to make an engine function so as to meet regulations and design criteria. A broad range of sensors and actuators are available to the engine designer. Factors in choosing sensors and actuators include the type of engine, the emissions and performance targets, and the engine cost per unit.

Drivven produces one of the few hybrid systems available that allows both combustion analysis and engine control in the same hardware and software package. This system was originally developed to allow control parameters of the ECU to be logged with the corresponding engine state as measured by the combustion analysis system. This has the advantage over the normal two-system approach because it uses a common interface and common set of acquired files to make post processing easier.

Earlier work shows that values computed in the combustion analysis system, called Drivven Combustion Analysis Toolkit (DCAT), can be used as input parameters to the controller side of the system and ensure that data is always available in time to be used for control (Viele and Quillen, 2009). Beyond calculating the values to be used in the next combustion cycle, DCAT algorithms were adapted to run on the FPGA in real-time to modify the control of the same-cycle.

This paper is mainly involved with demonstrating the next-cycle control and the same-cycle control. This is accomplished by comparison of the standard open loop control strategy to the next-cycle and same-cycle control strategies while varying engine conditions.

It is worth differentiating next-cycle-control systems from auto-balancing systems common on large industrial engines. Auto-balancing systems have been around for years, but due to lack of integration in to the control system as well as older processors, will average a large number of cycles before making a correction to engine operation. These systems can balance engine performance cylinders in the long run, but cannot perform cycle-to-cycle optimization to reduce combustion instability.

While the use of cylinder pressure based engine control algorithms is not new (Miller and Isermann, 2001)(Mueller et al., 2000)(Sellnau and Matekunas, 2000)(Fritz et al., 1996), earlier techniques were primarily focused on reduced complexity algorithms rather than conventional analysis (Guzzella and Onder, 2001).

6.3 System Overview

The experiments were preformed on a General Motors 1.9L, 4-cylinder, common rail diesel engine. The cylinders are instrumented with piezoelectric pressure sensors. The engine is equipped with all the standard stock engine sensors.

The ECU is a Drivven full authority engine controller. The system is composed of the following hardware components:

- NI PXI-1042 8-slot chassis
- NI PXI-8106 2.16GHz dual core real-time controller
- NI PXI-7813R 3M gate FPGA
- NI PXI-6123 8 channel, simultaneous sampling analog input

- NI 9215 4 channel, simultaneous sampling analog input
- Drivven AD Combo Module
- Drivven Low-Side Module
- Drivven DI Module
- Drivven PFI Module

The Intel Core 2 Duo based controller provides a sufficient processing power to accomplish advanced calculations without interrupting the control algorithms. The PXI-6123 simultaneous sampling A/D card is used to sample the cylinder pressure sensors for DCATs next-cycle control functionality. The NI-9215 simultaneous sampling A/D module is used to sample the cylinder pressure sensors for the same-cycle control. The FPGA is used to control all the Engine IO and EPT.

Conventional processors like the Intel processor used have high clock rates and data processing throughput as well as excellent programming language support making them great for next-cycle control type applications where data may be collected over a cycle then averaged. Unfortunately they have high latency and low turnaround time making them unsuitable for same-cycle control where decisions must be made on a crank angle by crank angle basis.

An FPGA is a programmable device used to implement generic digital logic. In this case it is used as an I/O co-processor. This particular FPGA implementation has a latency of 25ns allowing very fast control decisions. Adding more code to an FPGA will not increase the latency, but will increase the size of FPGA required to run the program. This program stretched the limits of the 3M Gate FPGA used in this experiment, though significant code improvements are possible. Another shortfall of the FPGA is that floating point calculations take much more space, so all calculations for same-cycle-control were done in fixed point.

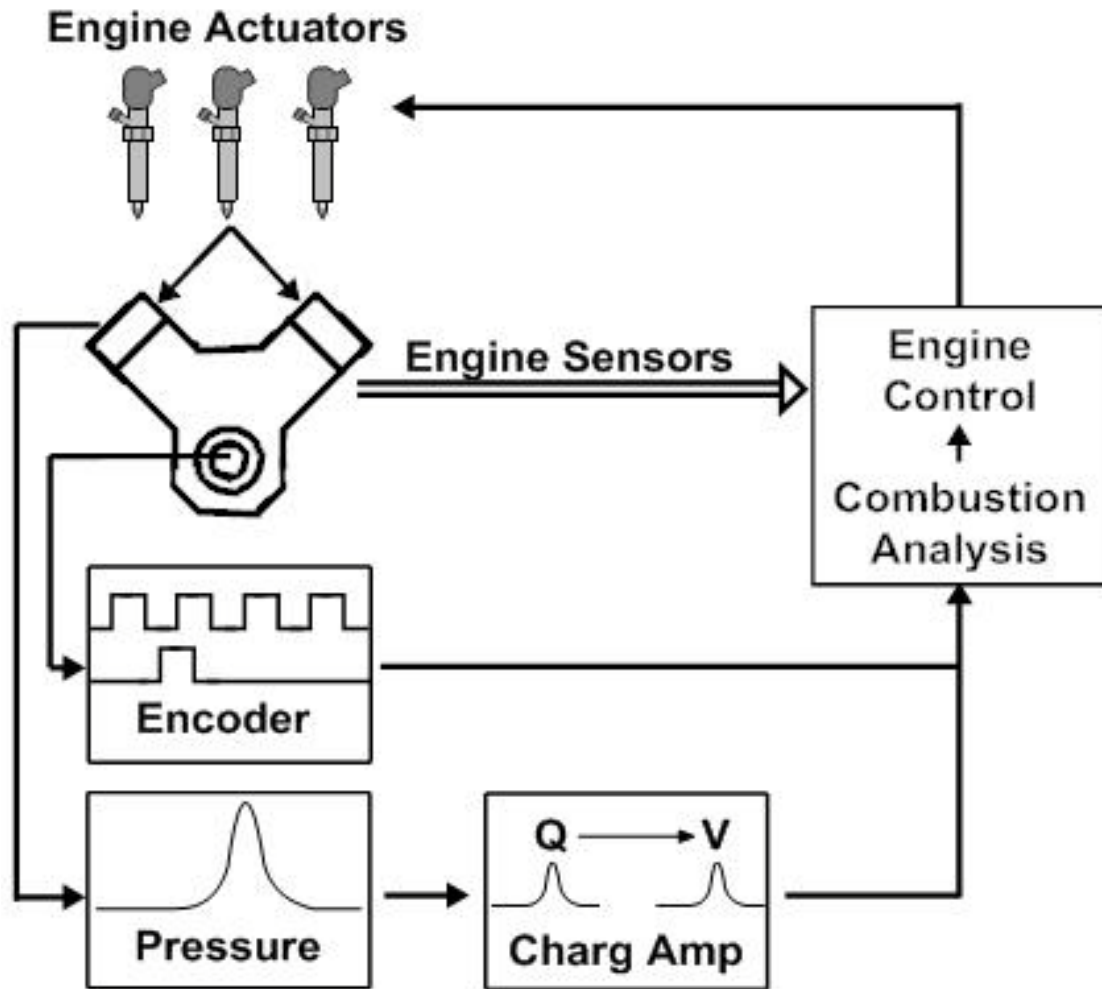


Figure 6.1: Engine Control Overview

The hardware is controlled using National Instruments LabVIEW software with the CalVIEW and DCAT add-ons. CalVIEW is used for calibration management and user interface while DCAT is used for combustion analysis. The system is set up to allow engine control and analysis in the same hardware package. (Figure 6.1) Therefore, it provides the ability to do advanced closed loop feedback control algorithms base on the combustion analysis.

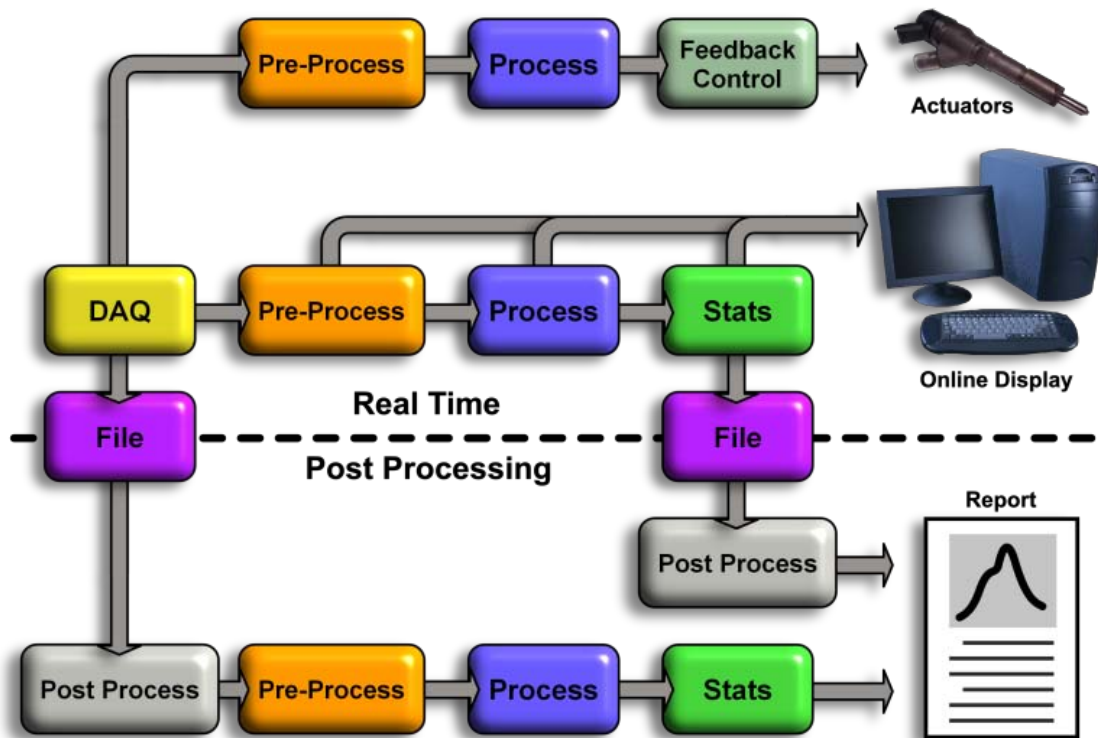


Figure 6.2: DCAT Data Flow Structure

6.4 Next-Cycle Control Description

Next-cycle control algorithm provides a short, high priority, calculation path and the framework to allow control algorithm to adapt based on the combustion results of the previous cycle, as shown in Figure 6.2 (DCA, 2009). The framework is set up to estimate the angle where the calculations finish ensuring that the calculated values are ready for next-cycle thus both ensuring on-time calculations and giving data as to how much margin is available for more calculations.

Next-cycle control may be based on any parameter calculated in DCAT depending on the goals of the testing and control strategy. E.g. Maintain a desired peak pressure by adjusting injection or spark timing. Maintain a desired power setting (IMEP) by adjusting the fuel quantity. Maintain a desired 50% MFB by adjusting the spark timing.

Table 6.1: 50% burn duration location (CAD ATDC)

	Open Loop	Next-cycle
0% EGR	11.2	13.0
10% EGR	11.3	13.0
23% EGR	11.6	13.0
27% EGR	11.9	13.0

6.5 Next-Cycle Results

For this paper, the next-cycle control was set up to maintain a 50% mass fraction burned location of 13.0 CAD ATDC. The calculated 50% MFB is controlled using a simple PID controller to adjust the open loop start of main injection timing. At the open loop conditions, the 50% MFB resides at 11.2 CAD ATDC and changes by 0.7 CAD as the EGR changes, as seen in Table 6.1. Next-cycle-control is able to maintain the desired 13.0 CAD ATDC setting across the range of possible EGR settings for this engine.

Figure 6.3 shows a plot of the MFB without next-cycle control shows the difference in the 50% MFB at the different EGR rates.

Enabling the next-cycle control quickly modifies the start of injection to the desired 50% MFB setting and maintains it as conditions change, as shown in Figure 6.4.

Further experimentation was done with next cycle control to analyze stability. Table 6.2 shows the effects of next cycle control for various operating conditions. Both the COVs of the 50% burn location and peak pressure are shown. Since cylinder 4 on this engine was instrumented for optical access and not pressure it is not shown in this data.

As expected as with high EGR the COV increases. Since these are different speed/load points a direct correlation is not expected. Note that in all cases the next cycle control implementation produced a significant decrease in COV of 50%

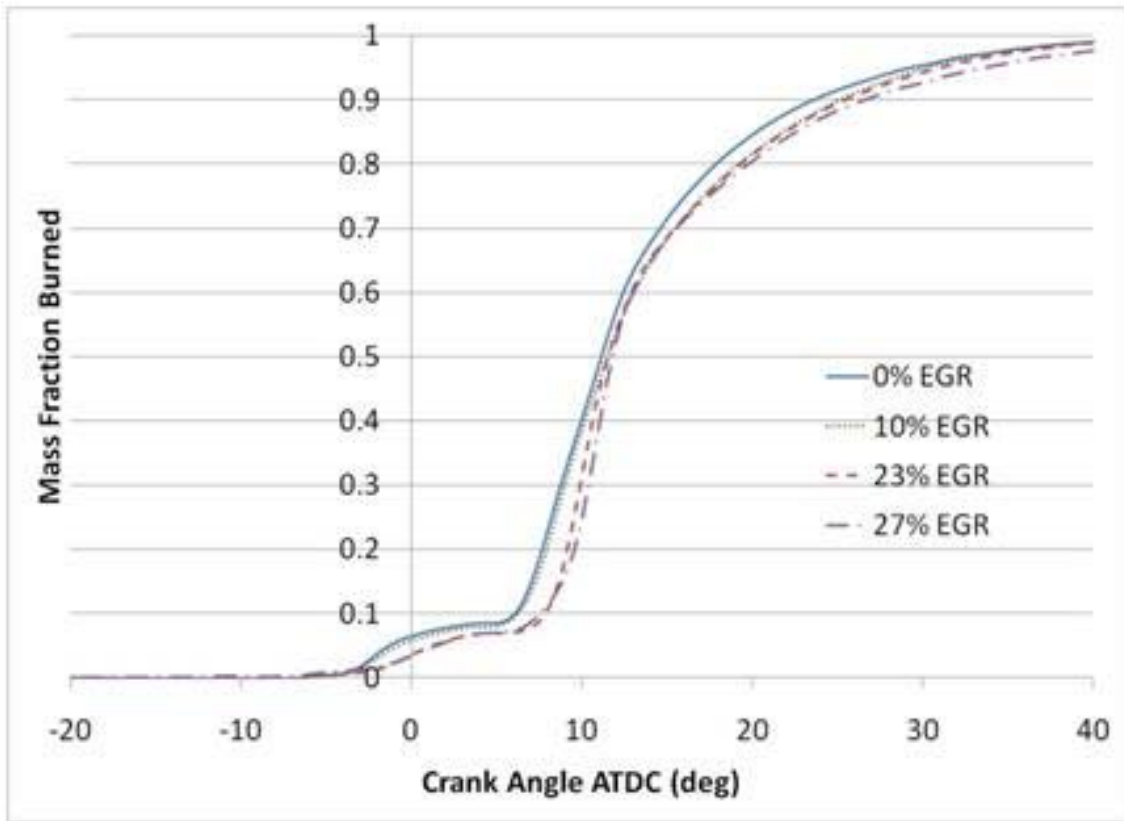


Figure 6.3: Mass Fraction Burned Open Loop Control

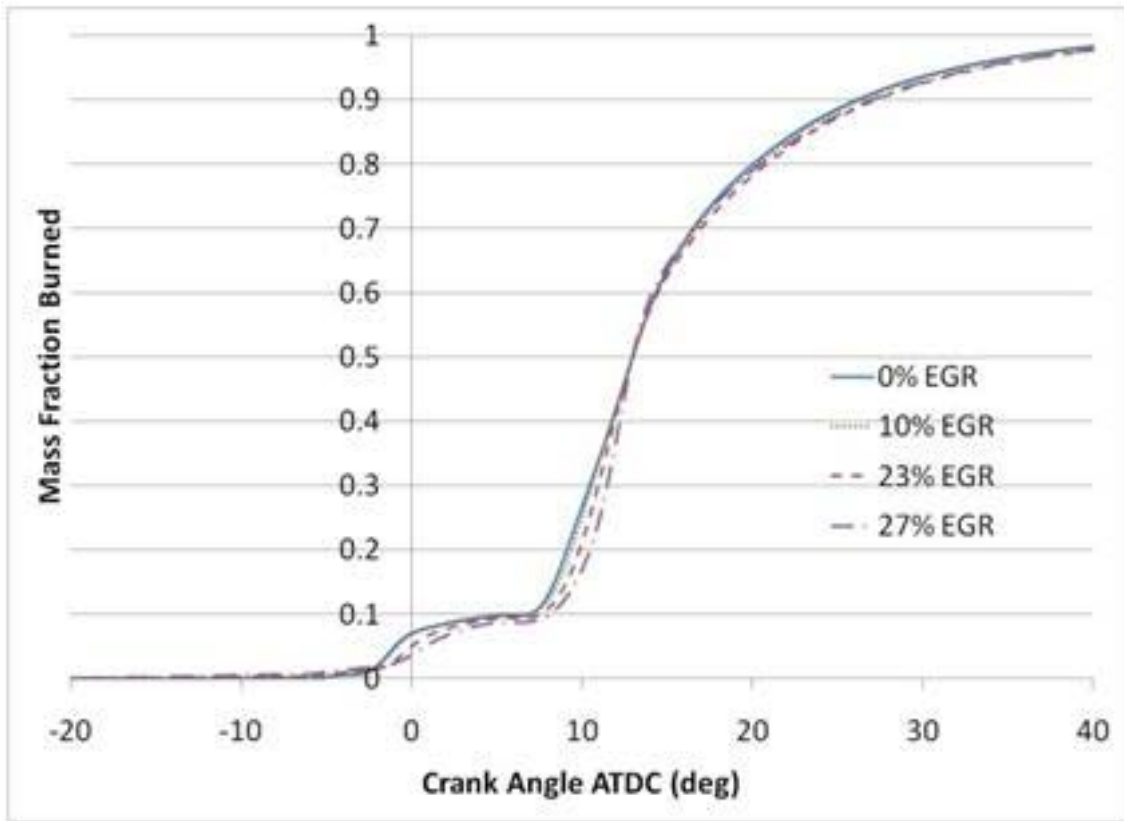


Figure 6.4: Mass Fraction Burned Next-Cycle Control

Table 6.2: COV of 50% burn location and Peak Pressure
 10 bar BMEP, 2500 rpm, 10% EGR

10 bar BMEP, 2500 rpm, 10% EGR				
		Cylinder 1	Cylinder 2	Cylinder 3
COV of 50% MFB	Open Loop	1.42	1.43	1.12
	Closed Loop	1.03	1.01	1.04
2.25 bar BMEP, 3000 rpm, 18% EGR				
		Cylinder 1	Cylinder 2	Cylinder 3
COV of 50% MFB	Open Loop	2.65	2.57	5.10
	Closed Loop	1.73	2.19	2.49
3 bar BMEP, 1800 rpm, 33% EGR				
		Cylinder 1	Cylinder 2	Cylinder 3
COV of 50% MFB	Open Loop	2.03	2.40	4.57
	Closed Loop	1.53	1.90	2.64

burn location. The tuning parameters of the next cycle control PID were optimized for a point not shown on this table, so further room for improvement is available through speed/load specific optimization or through the use of more advanced control techniques.

Plots of the mass fraction burned for the various operating points can be found in Figure 6.5. Note that with increased EGR and stock timing the 50% burn location is delayed for all cylinders, but quite a bit more for cylinder 3. EGR mal-distribution to cylinder 3 is a known issue on this engine because of compromises in the EGR system due to packaging. With next cycle control not only is 50% burn location kept on target, it is kept to the same target on all cylinders. It can also be seen, especially on cylinder 3 that much of the variation is reduced. No emissions data was taken in these tests, but emissions are generally reduced with lower COV.

6.6 Same-Cycle Requirement

Same-cycle control requires that combustion analysis calculations happen in on a sub-crank angle basis. Two issues can prevent this happening. First, the calculations require time to complete. Second, any delay in accessing shared resources

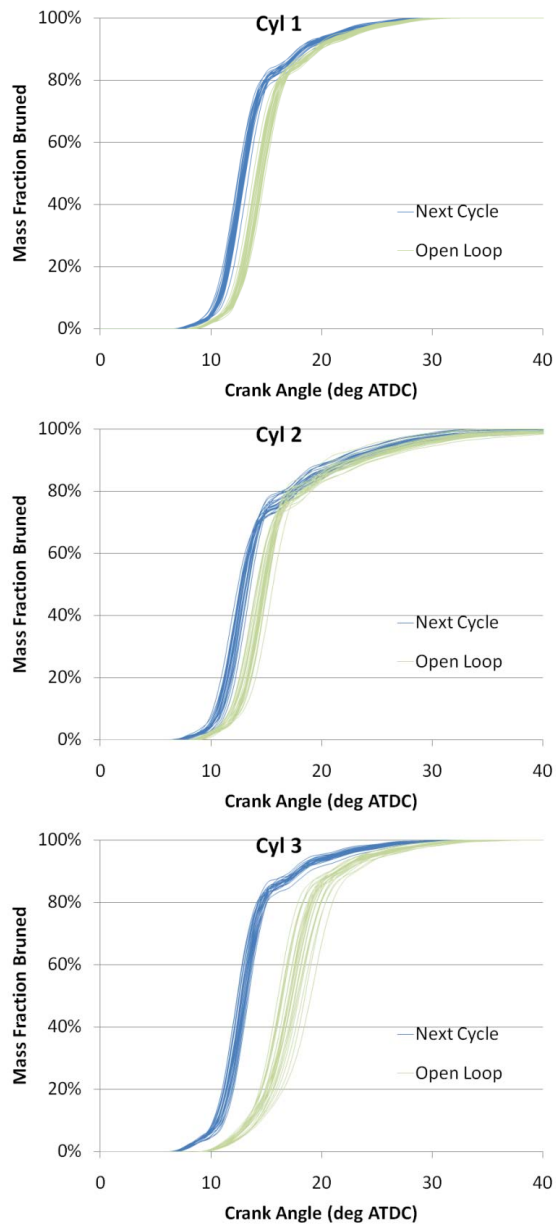


Figure 6.5: 3 bar BMEP, 1800 RPM, 33% EGR

will cause a delay. For these reasons, the FPGA is ideal for same-cycle control. The FPGA provides parallel execution and deterministic response time. However, the FPGA has limited resources, no floating point calculations, and large compile times.

Our target for this project was to have the same-cycle control algorithm use no more than 1M of our 3M gate FPGA. This would leave the rest of the FPGA to do its normal functions of firing injectors and reading I/O. The control code without same-cycle control uses 1.5M gates. In the end the same cycle control used nearly 2M gates and some of the normal control functions needed to be eliminated or simplified in order to fit everything in to the existing FPGA board. Further optimization of the FPGA code will be done in a later phase of the project.

The practical limit of executions speed for same-cycle-control is dominated by the analog sample rates available in off-the-shelf hardware. In our case we were limited to 100kS/sec/chan. Using the default 40MHz FPGA clock rate this gave us 400 clock ticks to complete any iterative calculations which was more than enough.

6.7 Heat Release Calculations

The cylinder pressure sampling may be triggered in one of two ways. The first way, engine-synchronous sampling, is to trigger the sampling at specified crank angles. The second way, time-based sampling, is to trigger the sampling based on a fixed frequency clock. Each method has advantages and disadvantages.

Engine-synchronous sampling simplifies requirements on the FPGA because it allows cylinder volumes to be pre-calculated and stored in memory on the FPGA. A simple lookup is required to have a precise volume corresponding to each pressure measurement. The minimum sample time is a function of the maximum hardware sample rate and the maximum engine speed. Therefore, at slow engine speeds, the time between calculations may increase beyond acceptable limits.

Due to the inherent noise in combustion and the pressure sampling process some amount of filtering is required. If noise of a specific time-base frequency is to be filtered out then the filter coefficients of filters applied to engine synchronous data need to be updated as a function of engine speed.

Time-based sampling avoids the issues with changing engine speeds. The data is always calculated at the same rate regardless of the engine speed. The main disadvantage to this sampling method is calculating the volume for each sample. The volume calculations contain trigometric calculations that are both slow and resource intensive in the FPGA. Therefore, it is not desirable to calculate volume at every sample. However, a lookup table may be used. The table should be able to accept any value of crank angle up to the maximum resolution of the engine position tracking software, typically, 8k-65k (1CAD to 0.1CAD) depending on the encoder pattern and EPT version used. However, it is impractical to make a lookup table of that size in the FPGA memory. The other options available are to use a smaller look up table and interpolate or use the closest value. These introduce errors but should provide acceptable results if implemented correctly.

After analysis of both sampling methods it was determined that for this engine and FPGA setup engine-synchronous sampling fit better in the FPGA, but that this should be re-evaluated for different engines and FPGA configurations as they are implemented.

Because math operations in the FPGA take up FPGA gates. Operations may be optimized for speed or size, but since only a few hundred clock cycles are available an optimization for speed was chosen. This makes math operations very “expensive” in terms of FPGA space. Divide operations take quite a bit more room than multiplies and non-linear calculations that are solved by Taylor series expansions are out of the question. As a result the equations implemented should be optimized for FPGA

use by pre-calculating values to minimize computation space and eliminate non-supported operations.

In our case volume and its derivative as well as the polytropic coefficients are known before the same-cycle control algorithm is started and the associated math can be pre-computed to a series of constants.

The Rassweiler and Withrow (Grimm and Johnson, 1990) heat release method was considered. (Equation 6.1) This method is not desirable due to the exponent calculation at every sample.

$$\frac{dQ}{d\theta} = \left(\frac{1}{n-1} \right) * V * \frac{dP_c}{d\theta} \quad (6.1a)$$

$$\frac{dP_c}{d\theta} = P_i = P - i - 1 * \left(\frac{V_{i-1}}{V_i} \right)^n \quad (6.1b)$$

The pressure ratio (Sellnau and Matekunas, 2000) calculations are similar to the Rassweiler and Withrow calculations as they contain an exponential calculation. (Equation 6.2) However, the exponent may be avoided by pre-calculating the motoring pressure in the real-time system and using a lookup table. This case then becomes the simplest calculation of heat release. However, it is not commonly used and may be difficult to compare with standard heat release methods.

$$PR = \frac{P}{P_m} \quad (6.2a)$$

$$P_{m,i} = P_{m,i-1} * \left(\frac{V_{i-1}}{V_i} \right)^n \quad (6.2b)$$

The single zone heat release (Heywood, 1988) calculations without heat transfer are fairly simple. (Equation 6.3) When using the engine-synchronous sampling

method with a constant polytropic coefficient, much of the equations can be replaced with pre-calculated constants.

$$\frac{dQ}{d\theta} = \left(\frac{1}{n-1} \right) * V * \frac{dP}{d\theta} + \left(\frac{1}{n-1} \right) * P * \frac{dV}{d\theta} \quad (6.3)$$

The single zone heat release with heat transfer requires many added complications (Heywood, 1988). (Equation 6.5) Some of the additional calculations include calculating the estimated in cylinder gas temperature, typically using the ideal gas law (Cengel and Boles, 2002). (Equation 6.4) The polytropic coefficients should be replaced with a temperature based correlation of the ratio of specific heats (Gatowski et al., 1984) (Gatowski et al., 1984). The cylinder area must also be known or calculated for each sample. Last, the heat transfer coefficient needs to be determined. Several methods of estimating the heat transfer correlations are commonly used but are computationally expensive (Hohenberg, 1979)(Annand, 1963)(Woschni, 1967).

$$T = \left(\frac{T_{IVC}}{P_{IVC} * V_{IVC}} \right) * P * V \quad (6.4)$$

$$\frac{dQ}{d\theta} = \left(\frac{1}{\gamma-1} \right) * V * \frac{dP}{d\theta} + \left(\frac{\gamma}{\gamma-1} \right) * P * \frac{dV}{d\theta} + \frac{dQ_{ht}}{d\theta} \quad (6.5a)$$

Engine-synchronous sampling with the single zone heat release calculations but without heat transfer were chosen for this paper due to the relatively simple calculations used that are also of a robust and standard nature. All of the following heat release results were calculated using the single zone without heat transfer method.

The heat release results calculated in the FPGA were found to match those calculated in the Pentium controller using DCAT with floating point calculations. (Figure 6.5) The main difference between the two data sets is that DCAT applies

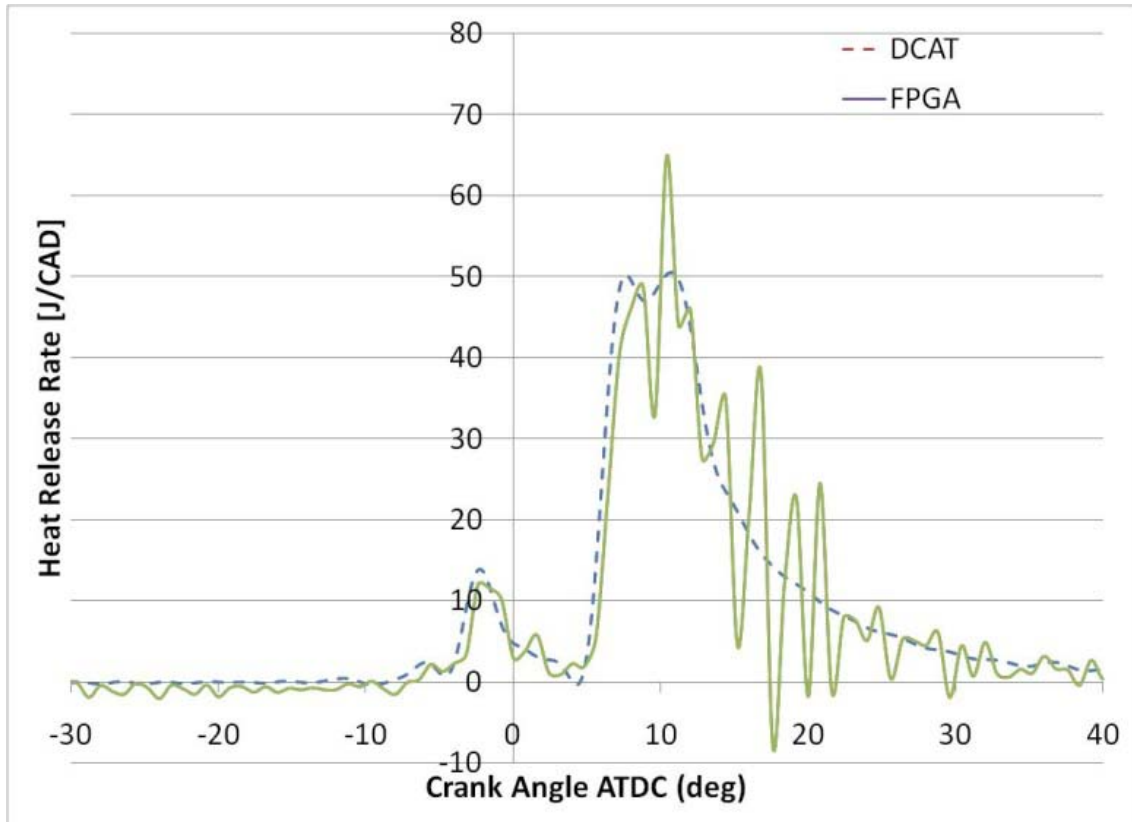


Figure 6.6: Heat Release Comparison: DCAT with filter, FPGA without Filter

a mild, zero-phase-shift filter to the pressure waveform before calculating the heat release. The calculated values in DCAT contain a similar amount of noise to the raw FPGA calculations with the filter turned off. Some of the other differences include the analog sampling hardware and the FPGA use of fixed point calculations.

The noise on the heat release results needs to be removed in order for the end of combustion to be determined in a reliable and robust manner. Therefore, a filter was applied to the pressure data and the calculated heat release. (Figure 6.6) A simple first order IIR filter was used on each sample as it came in. This was found to produce adequate results. The IIR filter causes a phase shift and delays the ability of the system to detect the end of combustion by the amount of the phase shift.

The heat release results are used to determine the end of combustion so that the next injection event may be scheduled. For this, an adaptive trigger arming method

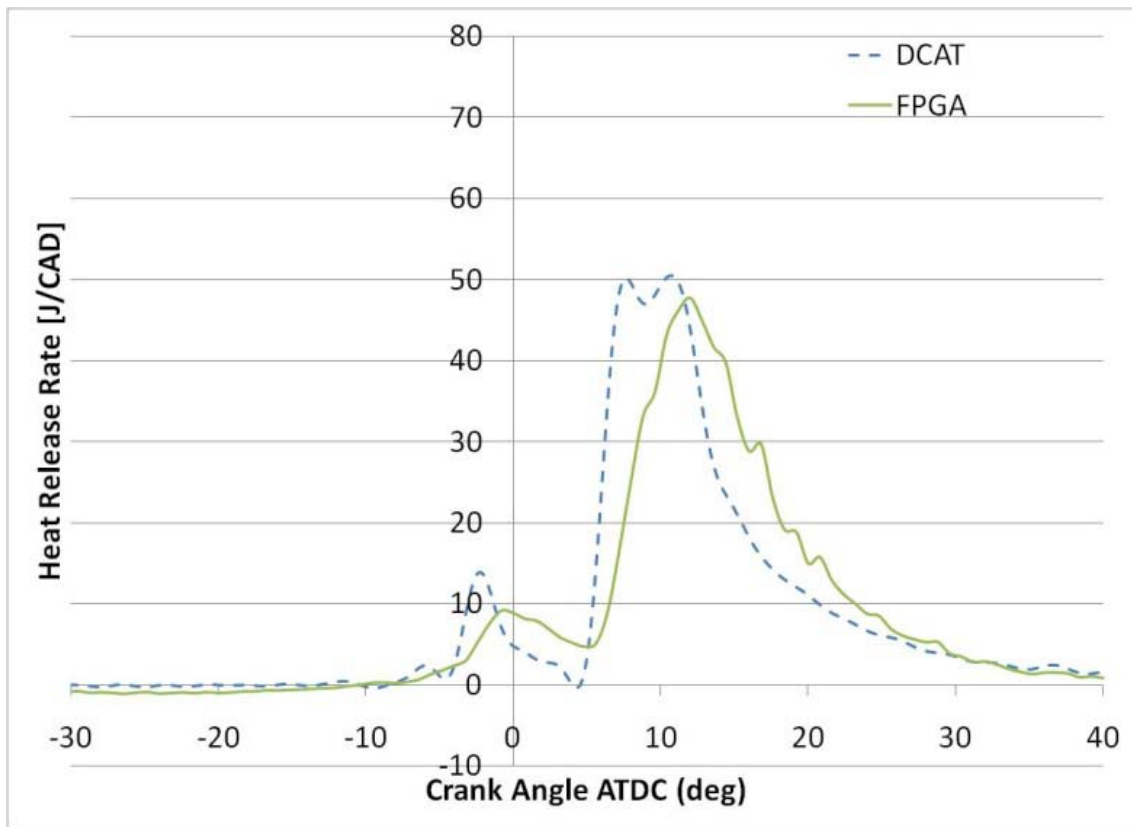


Figure 6.7: Heat Release Comparison with Filter

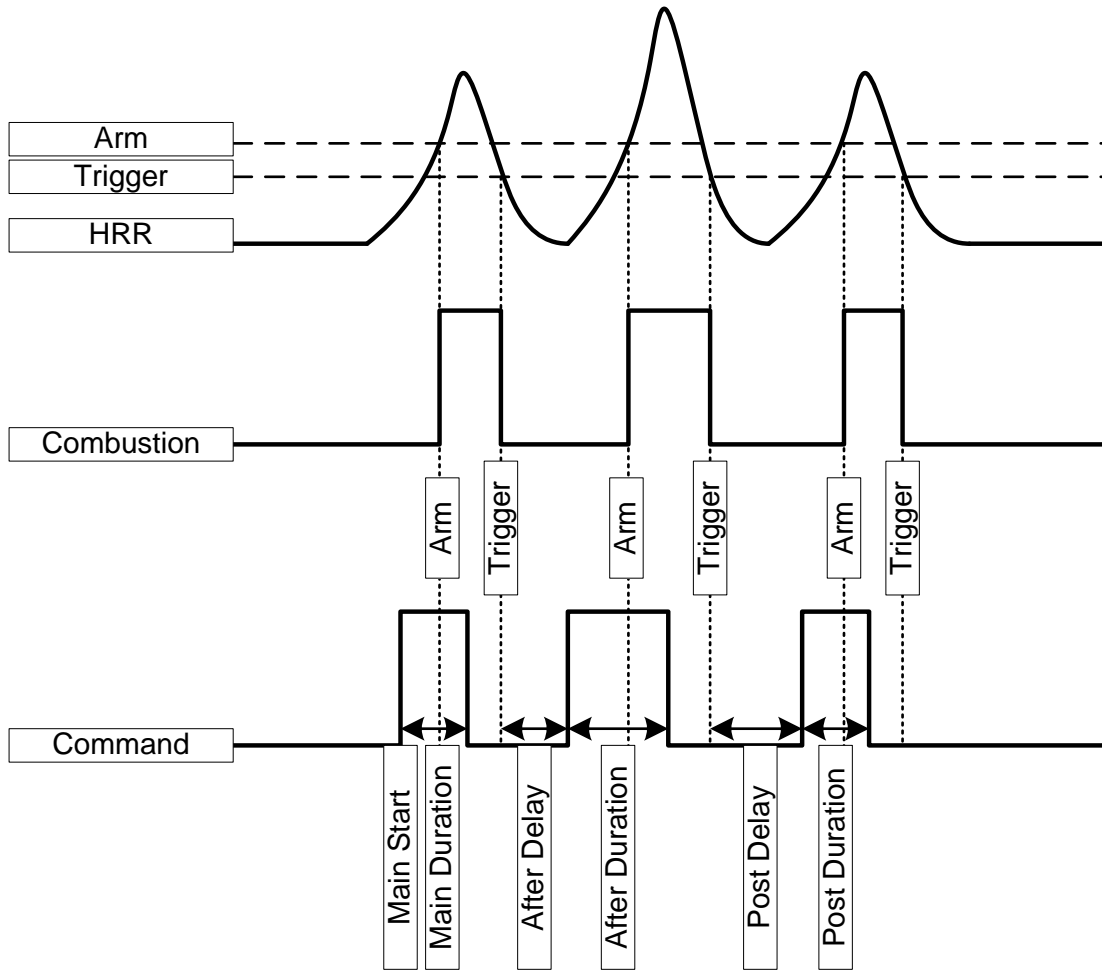


Figure 6.8: Combustion Sensing

was implemented, as shown in Figure 6.7. For each firing event we require the filtered heat release to go above the arming threshold then when it comes back down below the trigger threshold a timer is triggered to start the subsequent injection event.

The same-cycle control calculations can be broken down into several steps. The calculation flow can be seen in Figure 6.8.

The desired execution speed was met. With some reduction non-essential functions the complete FPGA code was made to fit in the systems 3M gate FPGA. A performance of 180 ticks for a 4 cylinder engine was achieved for the calculations themselves. The results of the calculations are available to the control 580 ticks

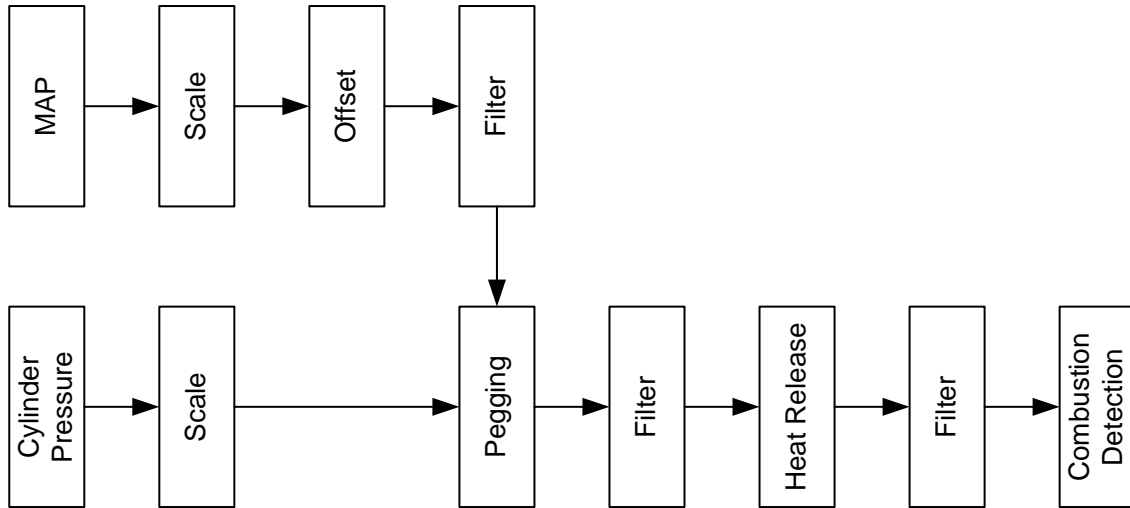


Figure 6.9: Same-Cycle Calculation Flow

(400ticks for DAQ and 180 for calculations) or 14s after the sample trigger. In future versions this will be pipelined to operate in parallel and only take 400 ticks.

The injection effects of the injection delay were tested at a partial load condition and 1500 RPM using two injections. The after delay was set to 0.3 and 0.6 ms which should correspond to a 2.7 and 5.4 CAD delay, respectively. The results of the test show that the expected delay was achieved in Figure 6.9 and Figure 6.10. This showed that the arming and triggering and heat release calculations were behaving properly.

To further experiment on this same cycle control a series of EGR sweeps were performed. The same-cycle-controller was configured to not fire a subsequent injection event until sometime after the first injection event has mostly burned out. No claims of performance or emissions improvements are made with this strategy; simply it is used to demonstrate the proper operations of same-cycle-control in a safe and well known operating region. By increasing EGR, combustion speed decreased (Stone, 1999) and thereby a longer inter-pulse delay was expected. This was verified by both measuring 50% burn locations and by looking at the raw injection current.

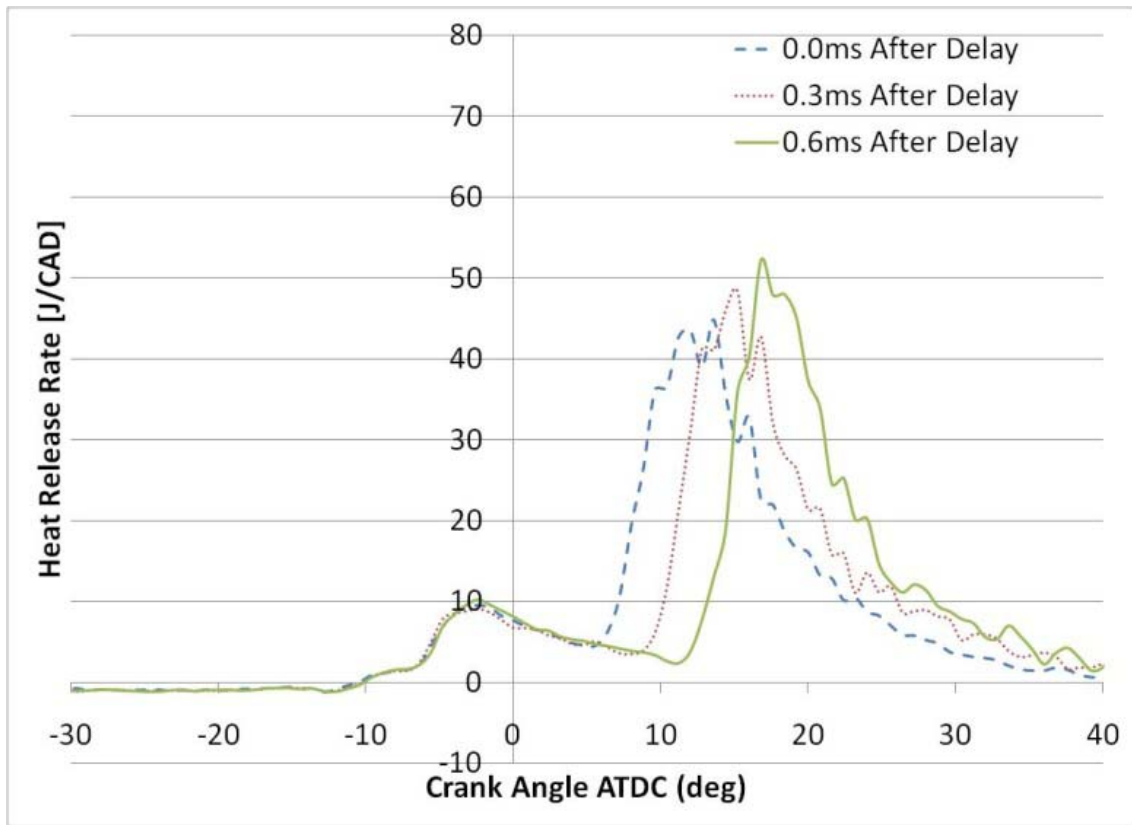


Figure 6.10: Heat Release - After Delay

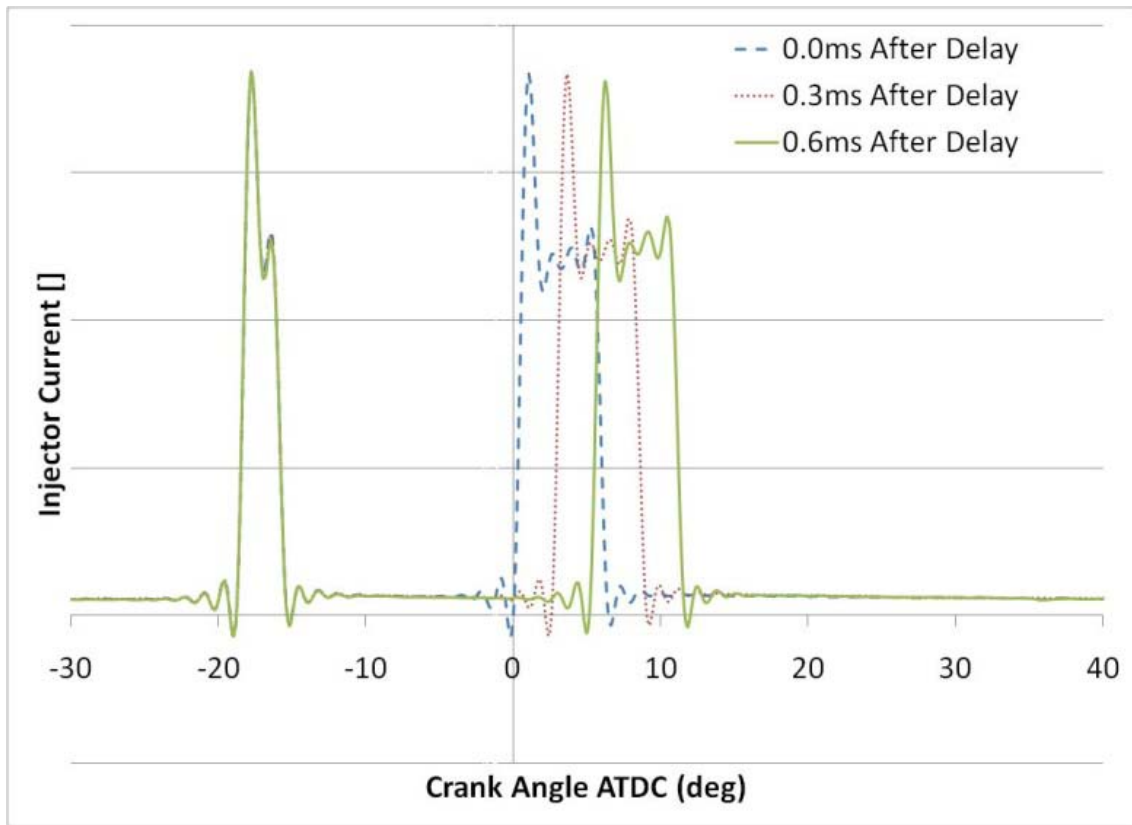


Figure 6.11: Injector Current - After Delay

Table 6.3: 50% burn duration location (CAD ATDC)

	Open Loop	Same-cycle
0% EGR	11.2	12.1
10% EGR	11.3	12.4
23% EGR	11.6	15.0
27% EGR	11.9	17.8

In Table 6.3 we can see that increasing EGR using open loop control (no change to injection timing) results in little change to 50% burn location. However, with same cycle control, the combustion events become stretched out, resulting in significant movement in 50% burn location.

Figure 6.11 and 6.12 show the heat release for the open loop and same-cycle control EGR sweeps. With the open-loop plots, a step change in combustion characteristics was observed from 10% to 23% EGR rate, then a leveling off after that. In the same-cycle plots there is a similar effect, except that the same-cycle controller reacts to the change in combustion characteristics by retarding timing. Plots of the injector current can be found in Figure 6.13 where a steady pilot pulse with dynamically varying main and post injection pulse timings occurred.

6.8 Conclusion

Next-cycle-control was shown to be able to control 50% burn locations over a broad range of engine operating conditions. It was shown to reduce cyclic instability in the conditions tested. Next cycle control shows promise for production applications to reduce emissions by keeping the engine operating at the targeted combustion conditions. It also shows promise as a calibration aid to help calibrators account for cylinder-to-cylinder differences in air, fuel, and EGR delivery. Lastly it is a potential calibration aid for doing large datasets by allowing the operating

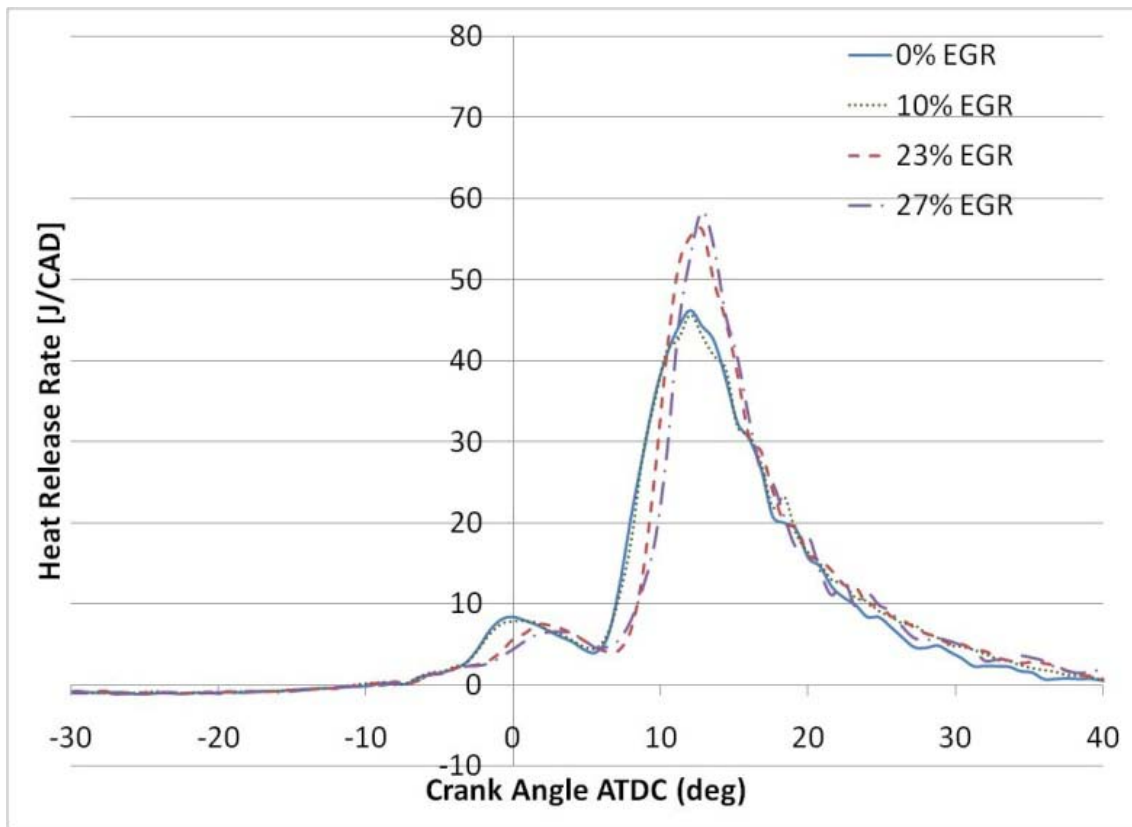


Figure 6.12: EGR Sweep Open Loop Control

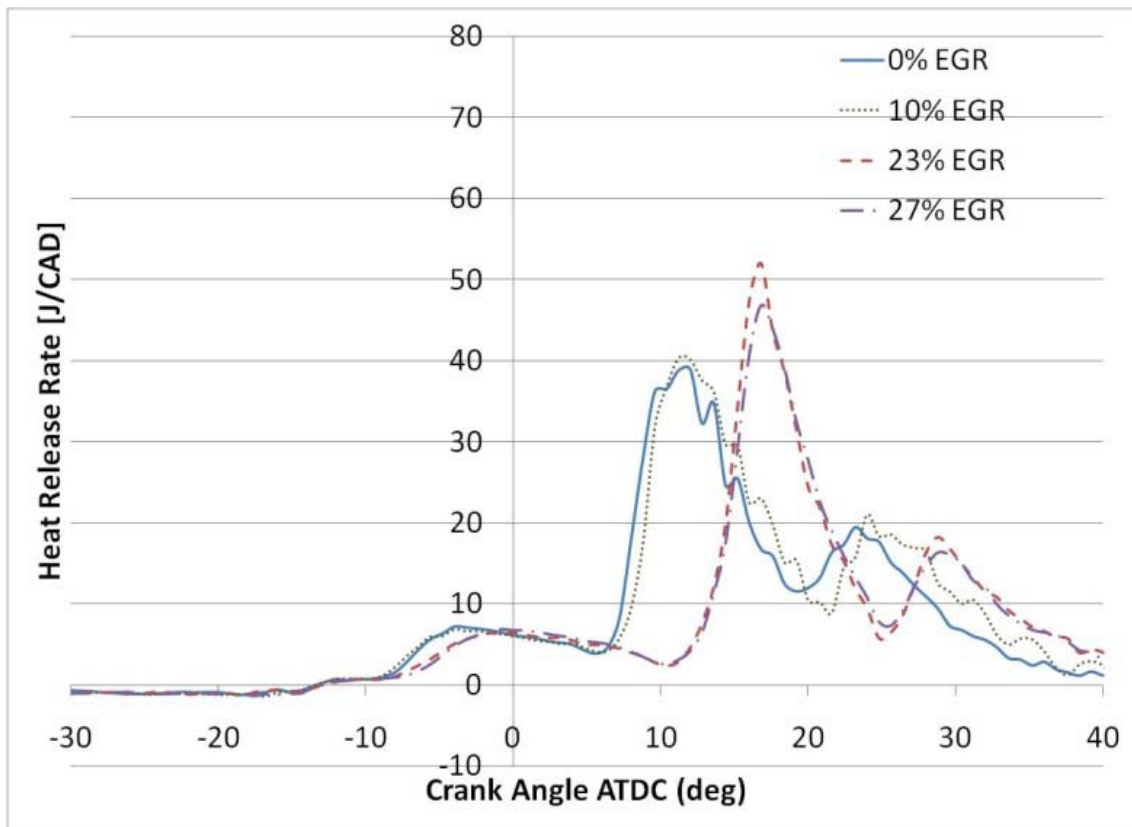


Figure 6.13: EGR Sweep - Same-cycle Heat Release

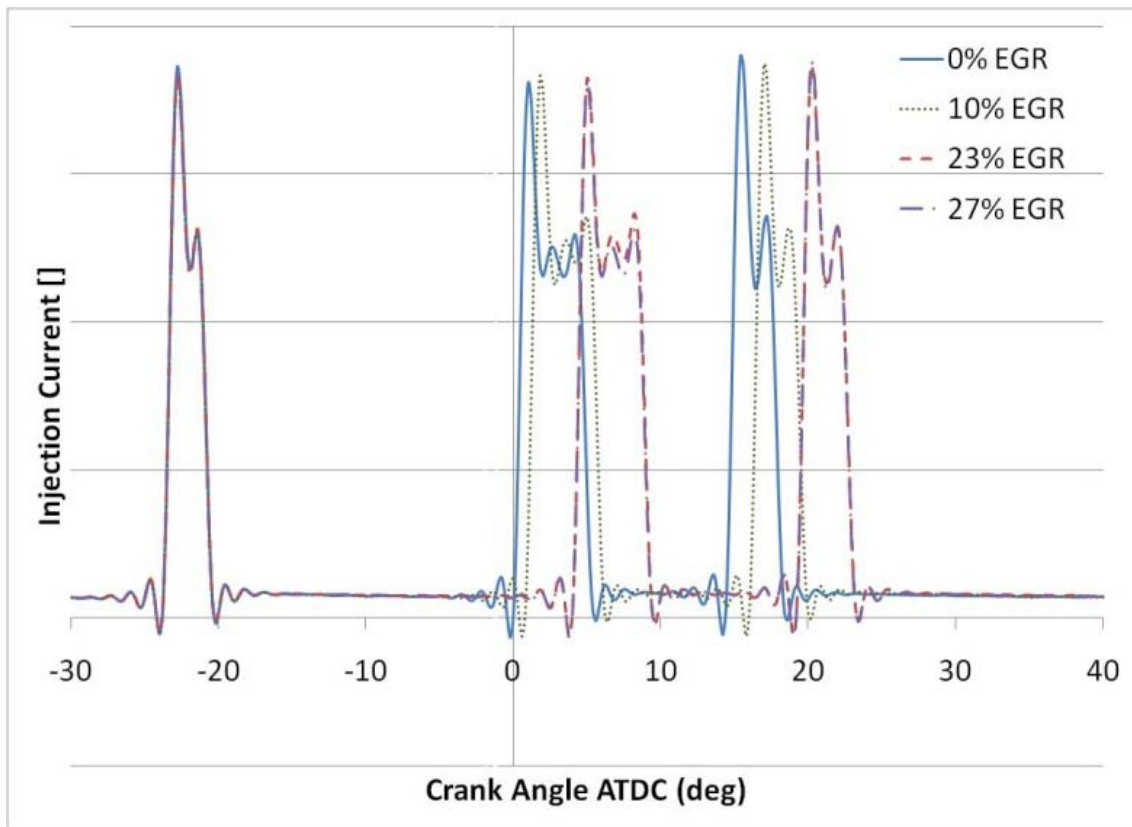


Figure 6.14: EGR Sweep - Same-cycle Injection

to directly control engine operating parameters like 50% burn location instead of having to manually set injection timing to achieve the desired results.

The same-cycle control algorithm was shown to correctly calculate the heat release and react to it on a sub-crank angle basis. The same-cycle control was able to robustly detect the end of combustion and modify the injection delay based on the calculated values over a variety of engine operating points.

6.9 Future Work

This project was intended to define and implement robust next-cycle and same-cycle control code. The experiments done were intended to test the algorithms themselves but not to demonstrate and particular improvement in engine performance or emissions.

Future work will be to evaluate potential applications of these technologies, specifically in the area of same-cycle-control. We hope to investigate improved HCCI stability as well as misfire-restrike on SI engines.

Work to simplify, optimize, and fully document the same-cycle-control software will be performed so that it can be included as a standard part of the DCAT software and available to DCAT users to perform their own experiments based on this approach.

6.10 Acknowledgments

We would like to thank the staff at Argonne National Laboratory, especially Swaminathan Subramanian, for collecting the experimental data.

We would also like to thank Carroll Dase from Driven for supporting device driver changes required for running the experiments.

Chapter 7

CUSTOM COMPUTING MACHINES

The previous papers examined necessary tasks performed by more conventional means and showed improvements in their implementation by use of FPGA technology. It is the intention of this section to examine something new and different in engine research: custom, highly-parallel, computing machines dedicated to real-time simulation. This real-time simulation is to be used in control to generate virtual sensors where physical sensors would be impractical because of cost, packaging, or some other concern. These simulations would be implemented in the FPGA fabric of upcoming FPGA based engine controllers.

This topic overlaps a number of areas, but there are no directly competing research projects. The related areas are:

- Commercial 1D CFD solvers used to design engines.
- Real-time CFD implementations for visualization (computer games).
- FPGA and GPU-based CFD accelerators.
- Non-CFD based engine models.

Commercial 1D-CFD solvers are generally used in off-line mode to design engines. GT-Power approaches real-time simulation by using the off-line simulation to train a neural-net then running the neural-net in real-time.

Much of the real-time CFD referred to in the literature is for computer game simulation. For instance, waves on a lake or movements of fog or mist. These are different from the approach described here in several respects. First the timescale is much slower, $25ms$ or more for games compared to about $5\mu s$ for fuel systems. Second, they are not hard-real time applications and are allowed to degrade gracefully.

Thirdly, they can have high latency because they are not part of a control system allowing fast, but high overhead, PC and GPU hardware to be applied. Because of these differences gaming and similar CFD models can implement large 2D and 3D models in real-time.

There are a number of projects using FPGAs and General Purpose Graphics Processor Units (GPGPUs) to accelerate large CFD models. In these systems each computational element in the FPGA or GPGPU processes hundreds or thousands of nodes. Because these systems have many more execution units than conventional processors and because these computations units are well tailored to the types of applications the overall speedup from conventional processors is quite large.

For both gaming CFD and non-real time CFD accelerators, the GPU is a more popular technology than FPGAs. The reason for this is that FPGAs are general purpose and consume more silicon area than GPUs for a hardware multiply instruction. This makes FPGAs more expensive per Giga Floating Operation Per Second (GFLOP) than FPGAs.

The downside of GPUs is they are an outgrowth of Single Instruction Multiple Data (SIMD) machines. As a result, they want to work on large sets of identically defined nodes with variation only in the node parameters. In the 1D-CFD models examined the mix of nodes is large and the space between different types of nodes is small. This means that whole sections of the GPU would need to be consumed to implement a single node type. Further, the loading and initializing of the GPU has a high overhead (hundreds of clocks) that precludes the super low latencies of the custom-built FPGA hardware examined in these papers.

Two papers and a patent on this topic are included in this chapter. Both papers have been accepted for publication, one to IEEE, the other to ASME.

For these two papers this author was responsible for:

- The concept of 1D CFD as well as the detailed framing of the problem.

- Heterogeneous core concept and model assignment.
- The math describing the 1D CFD as well as the C-code and optimization.
- the GT-Power comparisons.
- The multi-core and FPGA system framework.
- Benchmarking and optimizing all C-code.

Isaac Liu was responsible for the adaption of the PRET architecture to support multi-processor and conducting the benchmarking of FPGA area.

Gerald Wang was responsible for the automatic optimization algorithms and the spring-mass-damper model formulation.

Chapter 8

FCCM2012

This paper starts with examining the basic computational framework required for real-time CFD and outlines the necessary equations. It then provides a simple example benchmarked against a textbook example to prove the correctness of the equations and correct implementation of the small example. It proceeds to discuss in detail the software and hardware architecture required to implement this system. The paper concludes with benchmarks of the implementation of two models based on GT-SUITE examples.

A Heterogeneous Architecture for Evaluating Real-Time One-Dimensional Computational Fluid Dynamics on FPGAs

This draft paper has been accepted for publication in the proceedings of the The 20th Annual IEEE International Symposium on Field-Programmable Custom Computing Machines. It will be presented in Toronto, Canada in April, 2012. This paper was written by Matthew Viele of Drivven, Isaac Liu and Edward Lee from UC Berkeley, and Guoqiang Wang and Hugo Andrade from National Instruments.

8.1 Synopsis

Many fuel systems for diesel engines are developed with the help of commercial one-dimensional computational fluid dynamics (1D CFD) solvers that model and simulate the behavior of fluid flow through the interconnected pipes off-line. This paper presents a novel framework to evaluate 1D CFD models in real time on an FPGA to improve fuel pressure estimation and close the loop on fuel delivery, allowing for a cleaner and more efficient engine. The real-time requirements are defined by the physics and geometry of the problem being solved, which determines

how long a time step should be. In this framework, the interconnected pipes are partitioned into individual sub-volumes that compute their pressure and flow rate every time step based upon neighboring values. We use timing-based synchronization and multiple Precision Timed (PRET) processor cores to ensure the real-time constraints are met. Leveraging the programmability of FPGAs, we use a configurable heterogeneous architecture to save hardware resources. Several examples are presented along with the synthesis results on a Xilinx Virtex 6 FPGA. The results demonstrate the resource savings and scalability of our framework, confirming the feasibility of our approach – solving 1D CFD models in real time on FPGAs.

8.2 Introduction

In order to meet ever tightening worldwide emissions standards diesel engines are becoming more complex. In particular the diesel engine’s fuel system which must now support as many as 5 injections per cylinder event Drivven (2009). The fuel system consists of a high pressure pump, fuel injectors, and a network of connecting pipes known commonly as the "fuel-rail". Each time an injection event happens, pulsations are sent through the fuel supply rail. The high pressure, around 2000 bar, in the fuel system is often generated by a piston pump that also induces pulsations. These pulses need to be damped and/or modeled before the subsequent injection event in order to ensure a correct amount of fuel injection Bauer (2004).

Currently most fuel systems use an ad-hoc model of fuel pressure for subsequent injection events Winward et al. (2010). Since many fuel rails are developed in commercial one-dimensional computational fluid dynamics (1D CFD) solvers like GT-SUITE GT-Suite (2007), it seems a natural approach to use the same modeling technique to model their behavior in real time. To meet the stringent real-time requirement, the solution obtained on-line usually ignores second order effects such as cavitation and thermal gradients that are taken into account in the GT-SUITE

calculations. The second order effects are small but important for designing a well-behaved system. There is the salient distinction between an off-line research oriented approach like GT-SUITE and a real-time approach like the one presented here. So long as the real-time code is sufficiently accurate as to allow improved fuel pressure estimation, it can close the loop of fuel delivery, allowing for a more precise air/fuel ratio control as well as a cleaner and more efficient engine.

1D CFD is used when the system to be evaluated can be described as a network of pipes. The advantage of 1D CFD over its 2D and 3D cousins is a greatly reduced number of nodes to be solved and simplified equations in each node. This makes it common for use in simulating transient operation of internal combustion engines Sellnau et al. (2009). It also makes it possible to solve these problems in real time using a highly parallel approach. We specifically examine the area of fluid flow, but heat transfer, mechanical dynamics, and electrical circuit simulation all represent similar problems. In each of these problems, it may be possible to represent the set of equations to be solved as a graph where each node of the graph represents a physical quantity to be modeled, such as a sub-volume of fluid in a pipe. The information path communicated by nodes is represented as the interconnect of the graph.

1D CFD fits into the class of problems that are heterogeneous and micro-parallel. The micro-parallel modifier emphasizes small, dedicated portions of the problem being solved in each computational element while the heterogeneous modifier means that there are a number of distinct node types. This distinguishes them from homogeneous, micro-parallel problems often found in image processing, which lend themselves to graphics processor unit (GPU) and SIMD solutions with large common memories Ylvisaker et al. (2006). At each time step, each node reads the neighboring data from the previous time step and computes the new data to be sent

to its neighbors. In this periodic and parallel execution of all nodes, the performance of the system is limited by the node with the longest execution time.

It is important to understand that when modeling physical quantities, such as fluid flow, the time step is determined by the granularity of the application. For an explicit solver with a fixed time step, it is required that the solver run faster than the speed of information flow. This is expressed as

$$\frac{\Delta t}{\Delta x}a = C, \tag{8.1}$$

where a is the wave speed, C is the Courant number, Δt is the time step, and Δx is the spatial discretization step. For stability the Courant number needs to be less than 1 and a number below 0.8 is recommended GT-Suite (2007). For instance, if a fluid has a wave speed a of $1 \text{ cm}/\mu\text{s}$ and a discretization length Δx of 1 cm , then we require a time step Δt of less than $1 \mu\text{s}$. The discretization length of a pipe network is dominated by its smallest sub-volume. For a diesel fuel system, a 1 cm discretization length is common. For our purposes in this paper, we treat the speed of sound (wave speed) as 1500 m/s Tat and Gerpen (2003). We require adequate performance so that the slowest node can complete in Δt .

The advent of caches, out-of-order executions, branch prediction, and other performance improvements in modern processors has improved their average execution speed at the cost of determinism. As a result, worst-case execution time analysis often gives imprecise and overestimated results. For hard real-time systems, this causes overprovisioning of hardware resources in order to guarantee that no timing violations occur. The Precision Timed (PRET) architecture Lickly et al. (2008) is a processor architecture designed to provide timing determinism and good worst-case performance. It contains multiple hardware threads with predictable timing and

utilizes timing instructions to gate execution time of code blocks on the hardware threads.

This paper presents a novel framework for real-time execution of a 1D CFD solver on a Field Programmable Gate Array (FPGA) using PRET cores. We map the heterogeneous computational nodes onto the hardware threads of multiple PRET cores. The deterministic execution time ensures that each node completes within the specified time step and the timing instructions ensure the synchronization of data communication between threads and cores. We show that a timing deterministic design allows us to minimize the synchronization overhead and processor core footprint, while our heterogeneous evaluation architecture further optimizes the FPGA area and leads to a practical and scalable solution.

8.3 Related Work

The computing power of FPGAs has enabled accelerated simulation in many application domains. In this paper, we focus on real-time CFD problems. FPGAs have been used to accelerate off-line 2D and 3D CFD computations with millions of nodes Smith and Schnore (2004). In these examples, there is no real-time constraint and the number of computational nodes is huge, which makes a common practice to reuse FPGA elements by many fluid nodes. Soft real-time CFD has been used for video games for quite a while Yu et al. (2009). These cases differ from our application in several important respects. First, they operate on the order of milliseconds (e.g. $25ms$) as opposed to micro-seconds (e.g. $5\mu s$) in our case. Second, the soft real-time simulation results just need to look good to game players, so accuracy is not all that important. Lastly, being soft real-time, they can be allowed to miss a deadline and degrade gracefully if they cannot complete the calculations in time.

From a hardware architecture perspective, we need to consider evaluating our problem via the alternatives of traditional processor or GPU. Traditional desk-

top processors like Intel’s Pentium have a great deal of non-deterministic behavior brought about by caches, out-of-order instruction executions, etc. GPUs are gaining ground in scientific computing because of the large total throughput they offer. While in aggregate a GPU can outperform an FPGA-based implementation, it has some disabling limitations Kothapalli (2011). GPUs do well with relatively homogeneous tasks. E.g., NVIDIA’s GTX 280 has 30 streaming multiprocessors. Each streaming multiprocessor is effectively a 32-channel SIMD processor. Communication to the global memory takes hundreds of cycles. Our application has many different types of nodes interconnected and it requires an ultra-low latency. This is why we chose not to pursue a GPU-based approach.

Besides their powerful computing capability, FPGAs have additional advantages. They can contain other structures unrelated to the CFD code like actuator control logic or sensor interfaces that can be connected to the correct part of the CFD model with a single-cycle latency.

There may exist different implementation options for our application on FPGAs. Without leveraging the benefits from PRET cores, we could attempt the problem in discrete FPGA blocks. In order to make the application fit in a practical FPGA like the one we tested, we would need to re-use the hardware multipliers, adders, and other functional units. This would require a state machine to run it and begins to look a great deal like a processor. Along this line, a natural inclination would be to explore a Microblaze-based architecture. The Microblaze is not threaded, so in order to optimize our application for space, we would need to implement some sort of sharing mechanism to execute multiple computational nodes on each Microblaze, which may bring challenges in implementation complexity.

After exploring possible implementation options, we concluded that existing tangential efforts bound the space of our problem, but none of them is appropriate to real-time modeling of fuel systems. Instead, a deterministic threaded soft-core

processor is the right architecture for the job and the PRET cores, with good compiler support, offer a flexible solution. To the best of our knowledge, we believe this is the first attempt to attack real-time CFD on this timescale and complexity of problem.

8.4 Background

8.4.1 One-Dimensional Computational Fluid Dynamics

Solving 1D CFD problems begins with the Navier Stokes equations for compressible flow. We construct a library of computational elements for the type of pipe segments we use in the form of first order finite difference equations Desantes et al. (1999). We start with momentum equation (8.2) and continuity equation (8.3):

$$\frac{P_x}{\rho} + \dot{V} + \frac{fV}{2D}|V| = 0, \quad (8.2)$$

$$a^2V_x + V \left(\frac{P_x}{\rho} + g \sin \alpha \right) + \frac{P_t}{\rho} = 0, \quad (8.3)$$

where P is pressure, ρ is fluid density, V fluid velocity, f is the Darcy-Weisbach friction factor, D is pipe diameter, a is the wave speed, and $g \sin \alpha$ is the directional force of gravity. A dot ($\dot{\cdot}$) over a variable indicates the total derivative with respect to time. Subscripts x and t indicate partial differentiation along the pipe length and with respect to time, respectively.

These equations can be expanded and simplified by leaving out the body force and convective terms because these are negligible given the pressure and flow regime. A method of characteristic solution is used to explicitly evaluate the pressure and flow at the next time step through a first order finite difference method. The left graph in Fig. 8.1 shows the evaluation of pressure and flow at point I , where I is i at $t_0 + \Delta t$, based on the pressure and flow of the adjacent points at time t_0 . The

equations to be evaluated at each point are

$$V_I = \frac{1}{2} \left[V_{i-1} + V_{i+1} + \frac{1}{a\rho} (P_{i-1} - P_{i+1}) - \frac{f\Delta t}{2D} \beta \right] \quad (8.4)$$

$$P_I = \frac{1}{2} \left[P_{i-1} + P_{i+1} + \rho a (V_{i-1} - V_{i+1}) - \frac{\rho a f \Delta t}{2D} \beta \right] \quad (8.5)$$

where $\beta = (V_{i-1}|V_{i-1}| + V_{i+1}|V_{i+1}|)$.

One critical part in evaluating the value at point I is that there is a fixed relationship among Δx , Δt , and the wave speed a such that $a \leq \Delta x/\Delta t$. Δx is fixed by the geometry of the problem and a is fixed by the properties of the working fluid such as $a = \sqrt{K/\rho}$, where K is the bulk modulus. Therefore, Δt is defined. All computations must complete within Δt and the results must be posted in exactly Δt . Because the wave speed varies and the geometry of the problem may not work out evenly for all pipe segments, a modified method is implemented with an interpolation step. This is shown by the right graph in Fig. 8.1 and described by equations

$$\zeta_R = \zeta_i - \theta a(\zeta_i - \zeta_{i+1}) \text{ and } \zeta_L = \zeta_i - \theta a(\zeta_i - \zeta_{i-1}),$$

where θ represents the amount of interpolation desired and subscript R (resp. L) denotes the right (resp. left) point of i . These equations are evaluated for both pressure $\zeta = P$ and velocity $\zeta = V$. While we use this method to give calculation leeway, it is important to realize that this adds complexity to every block in the system as well as decreases Δt .

Let $B = a\rho/A$ and $E = \rho f \Delta x/2DA^2$, where A is the pipe cross sectional area. Plugging them into (8.4) and (8.5) and further rearrangement give the following characteristic equations:

$$C_p = P_{i-1} + Q_{i-1} (B - E|Q_{i-1}|),$$

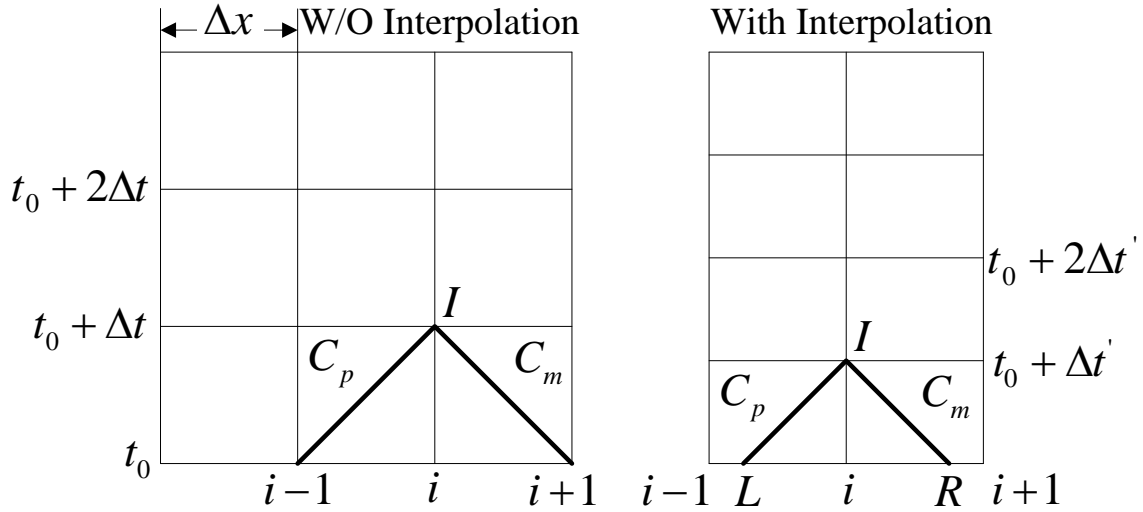


Figure 8.1: First Order Difference

$$C_m = P_{i+1} - Q_{i+1} (B - E|Q_{i+1}|),$$

where Q is the flow rate along the pipe and subscript p (resp. m) denotes the plus (resp. minus) branches of the characteristic equation.

Table 8.1 shows the equations for each of the supported flow elements. From these flow elements we can generate a network of flow elements that represents our fuel system. The B_{nd} subscript denotes a boundary condition. C_v is the flow coefficient which is a function of: Q_0 the nominal open flow, P_0 the downstream pressure, and τ the fraction the valve is open. Table 8.1 shows the equations for each of the supported flow elements. Based on them, we can generate a network of flow elements that represents our fuel system.

8.4.2 Precision Timed Architecture

For real-time applications that need timing determinism, Edwards and Lee Edwards and Lee (2007) propose Precision Timed (PRET) architectures. These architectures are designed for timing predictability and determinism, rather than average-case performance. Lickly et al. Lickly et al. (2008) present a multi-threaded imple-

Table 8.1: Equations for Supported Types

Type	$P_I =$	$Q_I =$
Pipe segment	$\frac{(C_p+C_m)}{2}$	$\frac{(P_I+C_m)}{B}$
Imposed pressure	P_{Bnd}	$\frac{(P_{Bnd}-C_m)}{B}$
Imposed flow	$C_m+B \cdot Q_{Bnd}$	Q_{Bnd}
Valve	$C_p-B \cdot Q_I$	$-B \cdot C_V + \sqrt{(B \cdot C_V)^2 + 2 \cdot C_V \cdot C_p}$ $C_V = \frac{Q_0 \tau}{2 \cdot P_0}$
Cap	$C_p-B \cdot Q_I$	0
Pipe ‘‘T’’	$\frac{C_{p1} + \frac{C_{m2}}{B_2} + \frac{C_{m3}}{B_3}}{\sum_{j=1,3} \frac{1}{B_j}}$	$-\frac{P_I}{B_1} + \frac{C_{p1}}{B_1}; -\frac{P_I}{B_2} + \frac{C_{m2}}{B_2};$ $-\frac{P_I}{B_3} + \frac{C_{m3}}{B_3}$

mentation of the PRET architecture using a thread-interleaved pipeline and scratchpad memories. The thread-interleaved pipeline removes data and control hazards within the pipeline by interleaving multiple hardware threads in a predictable round robin fashion. Scratchpad memories are single-cycle access on-chip memories which are managed in software. They are typically used in place of a cache to improve on predictability because the contents on the scratchpads are transparent in software.

Along with the predictable architecture, Lickly et al. Lickly et al. (2008) also introduce an instruction to control the temporal behavior of programs. The deadline instruction in Lickly et al. (2008) gives programmers a way to specify a lower bound on execution time for a specific code block, guaranteeing that a code block will not complete until at least the specified execution time. Lickly et al. (2008) also outlines a producer consumer example that synchronizes communication through the use of deadline instructions to ensure an ordering between shared data accesses. Our implementation of the 1D CFD solver leverages this instruction to synchronize communication across computational nodes and align the computation with real-world events.

8.5 Design Flow and Architecture

8.5.1 System Description and Design Flow

The process of generating a system is outlined in Figure 8.2. Starting from a description of the flow system and our library of elements we can create a graph to describe the system. The flow system will also dictate the maximum time step of the system. With this information we can instantiate processors and interconnects tailored to our needs and apply the library code to them. From there we can build and deploy our system.

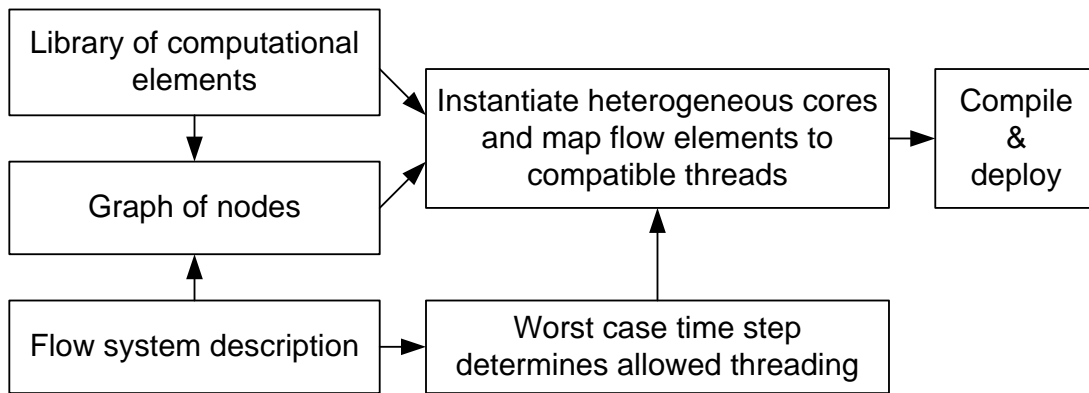


Figure 8.2: Design Flow

Fig. 8.3 shows an overview of a representative system for modeling fuel rails. The 1D CFD model is bounded inside the dashed rectangle. External to that is the real-world sensor and actuator interfaces that provide boundary conditions or consume model output variables. The small blue squares inside the dashed rectangle represent the network of flow elements. In a practical simulation of a diesel fuel system, the total number of flow elements can range from around 50 to a few hundred.

Each pipe element is a computational node, and their graphical representation is shown in Table 8.2. The top 3 rows of the table represent the flow elements described in Table 8.1. *Mechanical calculation* elements compute the inputs to valve, defined flow, and defined pressure blocks. They serve as an interface between the

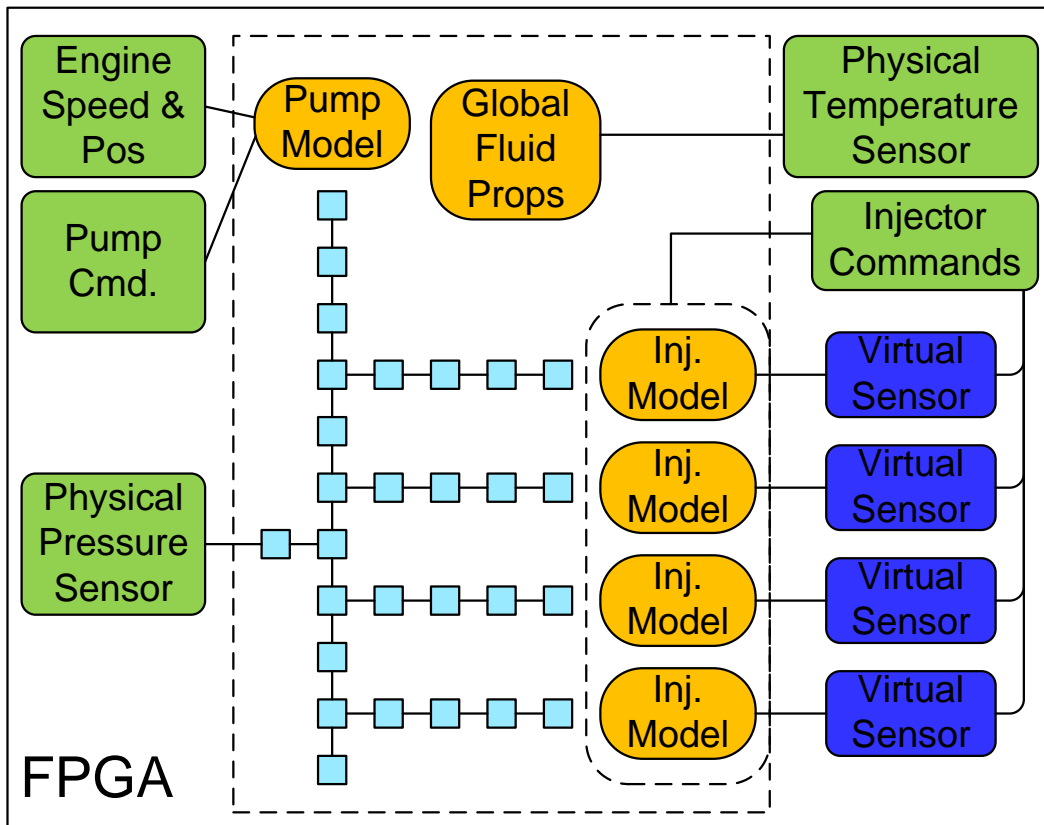












Figure 8.3: High Level System Diagram

Table 8.2: Library of Computational Node Elements

Pipe segment		Cap	
Imposed pressure		Imposed flow	
Pipe “T”		Valve	
Mechanical calculation		Global calculation	
Global distribution		Output	

flow model and the real world. *Global calculation* elements are used to compute the temperature dependent variables of density and wave speed. They post their data to a global distribution node, broadcasting it to all other nodes. Blocks with white backgrounds in the last row of Table 8.2, i.e., *Global distribution* and *Output* denote that they are implemented directly in FPGA fabric, not mapped to a processor thread. Global distribution elements are purely used to indicate a distribution of input value to each of the computational elements in our model. Output elements are used when data needs to be communicated out of the model to other parts of the FPGA.

For illustrative purposes, we show a simplified sample pipe network with an imposed flow input (P1) in Fig. 8.4. Fluid travels through a few pipe segment nodes (P2 and P3) to a “T” intersection (P4), where it splits off to a second branch of the network. The “T” node is also measured by the outside world (D1) through a output port. Flow going up the new leg ends in a cap (P8), while flow continuing down the original path exits the system through a valve (P6). The system is assumed to be at uniform temperature and values based on the temperature dependent variables of density and wave speed are computed by global calculations (G1, G2, and G3) and delivered by global distributions (GD1, GD2, and GD3) to each of the computational elements every time step for use in the subsequent time step.

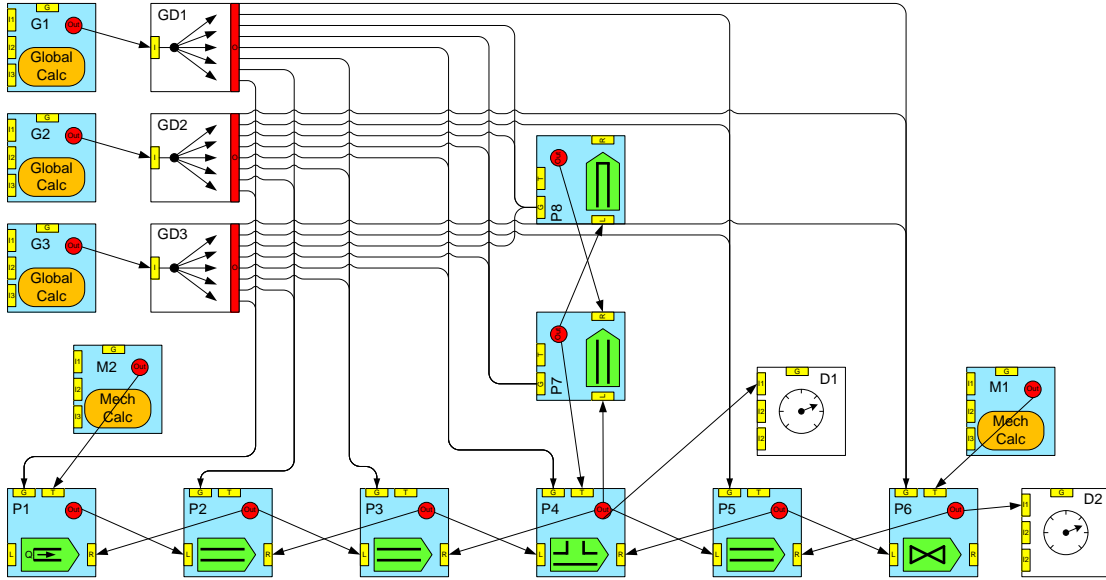


Figure 8.4: Detailed System Diagram

8.5.2 System Hardware Architecture

The hardware architecture of real-time 1D CFD evaluation consists of multiple PRET cores connected through point-to-point connections and a global distribution circuit. Fig. 8.5 shows a block-level view of the hardware architecture. As mentioned in Section 8.4.2, each core consists of hardware threads interleaved through the pipeline in a predictable round robin fashion. Computational nodes are mapped onto hardware threads, instead of each being implemented as its own core. This provides two major advantages. First, multi-threaded architectures maximize the throughput over latency. Multi-threaded architectures hide the latencies of multi-cycle operations, such as floating point operations or memory operations. When a hardware thread is waiting on these operations to complete, other threads can continue to execute in the pipeline. E.g., in our implementation, floating-point addition and subtraction appear as single cycle instructions in timing analysis because the floating-point latency is hidden through the execution of other thread contexts within the pipeline.

Second, the thread interleaved pipeline design allows for a simpler pipeline design. Since data and control hazards are no longer present in the pipeline, the logic used for handling them can be stripped out, greatly reducing the cost of the core. Furthermore, multiple threads share the same datapath, so the cost of adding threads is far less than adding a core, further reducing the cost of the system. We discuss in more details the trade-offs involving adding threads later in Section 8.6. The memory footprint required for each node is small enough, roughly a hundred assembly instructions, so that the scratchpad is sufficient for memory use and no main memory is needed.

Only basic single-precision floating point operations (add/sub/multiply) are needed for most nodes. But some require more complicated operations: the valve element uses floating point square root and the “T” element uses floating point divide, as shown in Table 8.1. However, these elements typically represent only a few percent of the overall system. In our complex example presented later, the common rail fuel system, there are 234 nodes: only 5 nodes are “T”s (requiring division) and 4 node are valves (requiring square root), which is approximately 4%. To save on hardware resources, we could use software emulation for the complex operations. However, our system is bounded by the slowest computational element, so the performance hit from using software emulation for these small percent of nodes would limit the overall real-time performance. But if we add hardware units on all the cores, it would be wasted on most cores with a homogeneous multi-core approach. Instead, we adopt a heterogeneous multi-core approach and provide several configurations of the core that includes different floating point hardware units. We only synthesize hardware accelerators to the cores that require them. Since the number of cores which need hardware accelerators is small, we still get the throughput improvement from adding hardware support without the huge resource overhead. This justifies substantial resource savings, which we show in Section 8.6.

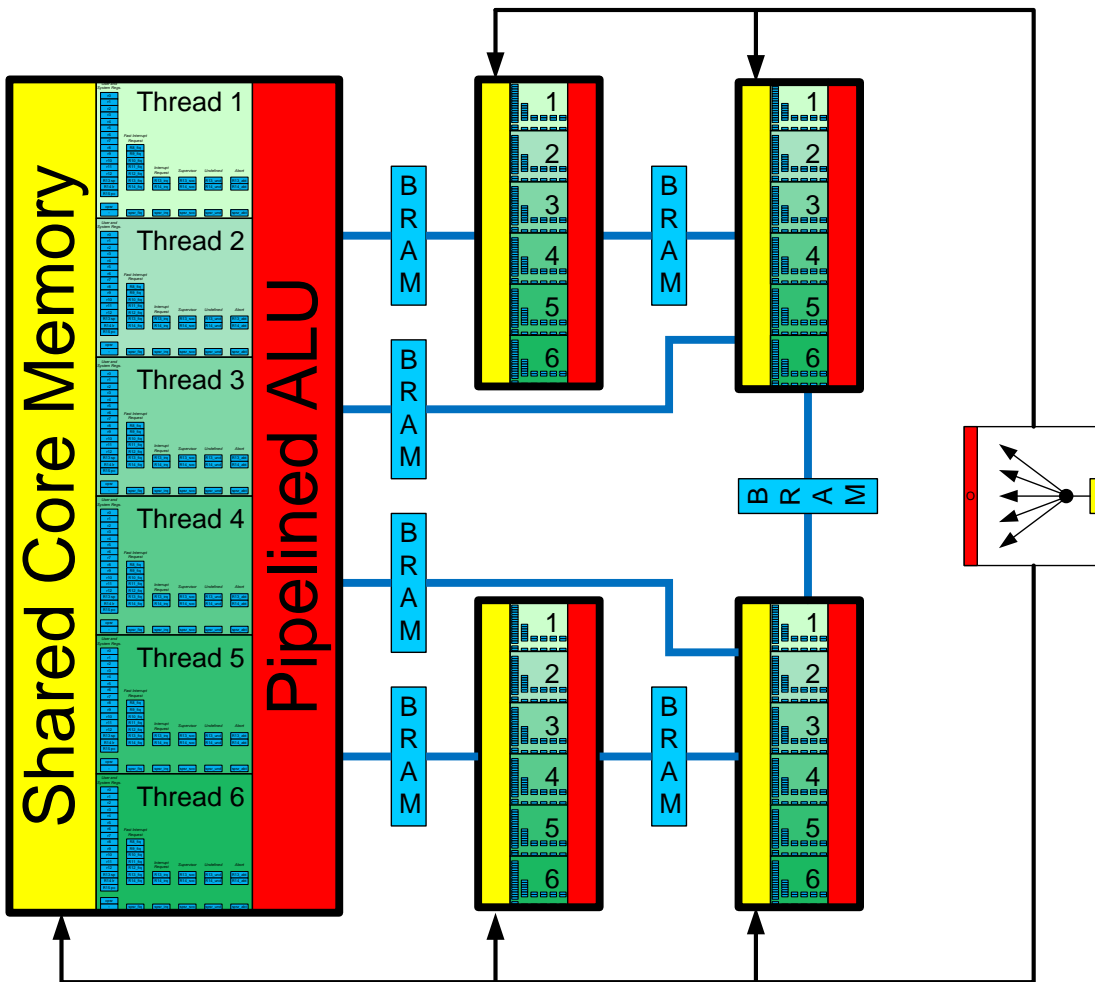


Figure 8.5: System of PRET Cores and Interconnects

Each node in our system requires one to four input ports along with one to four output ports depending on the number of neighboring connections connected to its neighboring nodes. One input port is dedicated as the global port, which always receives broadcasts from the global distribution circuits. Since only local communications occur across nodes, only point-to-point communication channels need to be established. Nodes mapped to the same core (intra-core communication) can communicate through the shared local memory within the core as shown in Fig. 8.5. Nodes mapped to different cores (inter-core communication) communicate through the point-to-point interconnect as illustrated in Fig. 8.5. We use shared dual-port Block RAMs (BRAMs) for our inter-core communication. This serves two purposes. First, it provides single-cycle deterministic communication, as BRAM access is single cycle. This allows the timing analysis to be simplified, as there is no hardware protocol that needs to be accounted for when accessing data through the inter-core communication channels. More importantly, the timing analysis for each node is now independent of the node mapping; both intra- and inter-core communication mechanisms are single-cycle, as they both access BRAMs. If this were not the case, then the timing analysis needs to assume all communications are inter-core communications, i.e., the longer of the two. Second, by using the dedicated BRAM blocks on the FPGA for interconnects, we save the logic slices to be used for computation nodes. This is useful because the limiting resource in our implementation is logic slices, not BRAMs, as justified later in Section 8.6. Each core only requires a small number of BRAMs to be used for registers and scratchpads, so the BRAM utilization ratio is far less than the logic slice utilization ratio. At each time step only two words, pressure and flow rate values, are transferred from a node to each of its neighbors. Our time periodic execution of computation nodes (described next in Section 8.5.3) ensures that we only need a buffer size of one for each of the words. Because the communication bandwidth is small, we only need one BRAM block to

establish an interconnect that allows all threads from one core to communicate with all threads on the other.

All of our flow elements have a dependency on density and wave speed that are functions of temperature. Temperature is assumed to be the same throughout the system, so these parameters are computed in a single computational element and broadcast to all pipe elements through the global distribution circuit as illustrated in Fig. 8.4. Leveraging this, the global distribution circuit is implemented by a single broadcaster that writes to dedicated memories local to each core. This broadcast receiving memory is synthesized to a small dual-port BRAM, with a read-only side connected to the core, and a write-only side connected to the broadcaster. This memory is shared amongst all threads in a core so all threads can access the global values. This architecture allows us to save on the resources needed to implement a full fledged interconnect routing system or any network protocol to be used for broadcasting.

8.5.3 Software Design

We implement the equations in Table 8.1 in the language C and compile it with the GNU ARM cross compiler gnu (2012) to run on our cores. Columns 2-6 in Table 8.3 show the number of Multiply, Add/Subtract, Absolute Value, Square Root, and Divide operations required by each computational element after optimization. Each computational node is executed on a hardware thread and data is exchanged only at the boundaries of the time steps to avoid data races. Fig. 8.6 shows an example timeline view of the operations for each node. The execution for computational nodes (top in Fig. 8.6) during each time step consists of three phases: (1) Read in the pressure and flow rate values from neighbors as well as global values; (2) Compute the output values; (3) Send output values to neighbors to be used for next time step. The global and mechanical calculation nodes do not need to read data

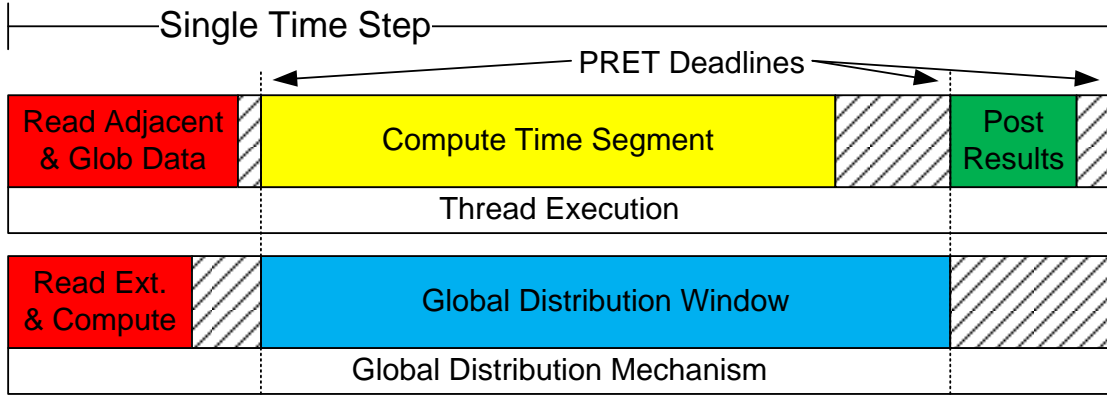


Figure 8.6: Execution of Nodes at Each Time Step

from other nodes, but might gather data from physical sensors and broadcast or send data to other nodes.

The computation done by each node consists of only a single path of execution, voiding the need for complex software analysis. We leverage the PRET precise architectural timing analysis, which provides exact execution time for the computation in the nodes. Data synchronization is handled by the synchronized periodic communication points, which enforces an ordering between the writing and reading of shared data. This voids the need of any explicit synchronization methods, removing any overhead and unpredictability for communication. These properties allow us to statically obtain an exact execution time for each computation node, which we show in the last two columns of Table 8.3. A *Thread cycle* is defined as a thread's perceived clock cycle. To get the physical execution time, multiply the thread cycles by the number of hardware threads in the pipeline to get processor clock cycles, then convert the clock cycles to physical time according to the processor clock speed.

In addition to statically assuring that the worst-case execution time meets the timing constraints specified, we also need to enforce that node executions remain synchronized. Our approach uses specialized timing instructions provided by the PRET architecture to enforce the synchronized communication points for all nodes.

Fig. 8.6 shows the program synchronization points that our timing instruction enforces. The hatched area in the figure denotes slack time that is generated by the timing instructions. When a timing instruction is decoded, it first enforces the previously specified timing constraint, then it specifies a new timing constraint for the next code block. In the code, one timing instruction is used during initialization to specify the first timing constraint. Then, timing instructions are inserted at the end of each code block. When the code block completes and the timing instruction is reached, the processor enforces the previously specified time bound by stalling if needed. Once the specified execution time is reached, the code will continue execution and the next timing specification is set. Each timing instruction takes 2 cycles because it manipulates a 64-bit value representing time. For our computational elements, 3 timing instructions are used each computation iteration, thus 6 cycles of overhead are introduced per time step. The overhead is already included in our execution time analysis presented in Table 8.3. The same effect can possibly be achieved with no overhead using instruction counting and NOP insertions. This can certainly be done on any deterministic architecture such as PRET. However, NOP insertion is both brittle and tedious. Any change in the code would change the timing of the software and insertions need to be adjusted to ensure the correct number of NOPs is added. Designs now are mostly written in programming languages like the language C and compiled into assembly, making it extremely difficult to gauge the number of NOPs needed at design time. The timing instructions allow for a much more scalable and flexible approach. In a system with heterogeneous nodes and different execution times, the timing instructions allow us to set the same timing constraints in all nodes regardless of its contents.

8.6 Experimental Results and Discussion

Table 8.3: Computational Intensity of Supported Types

Type	Without Interpolation / With Interpolation					
	Mul	Add/Sub	Abs	Sqrt	Div	Thread cycles
Pipe segment	10 / 18	5 / 13	2 / 2	0 / 0	0 / 0	81 / 51
Imposed pressure	6 / 10	3 / 7	1 / 1	0 / 0	0 / 0	50 / 38
Imposed flow	5 / 9	3 / 7	1 / 1	0 / 0	0 / 0	51 / 40
Valve	13 / 17	5 / 9	1 / 1	1 / 1	0 / 0	64 / 55
Cap	4 / 8	2 / 6	1 / 1	0 / 0	0 / 0	48 / 39
Pipe “T”	16 / 28	13 / 25	3 / 0	0 / 0	4 / 4	111 / 72

8.6.1 Setup

We use three examples to evaluate our framework. Our first example is a simple waterhammer example taken from Wylie and Streeter Wylie and Streeter (1978). It is similar to the one shown in Fig. 8.4, but without the “T” element and the nodes that branch up. This example contains an imposed pressure, 5 pipe segments, a valve, and two mechanical input blocks which provide both the reference pressure and the valve angle as a function of time. We use this simply as a sanity check for the correctness of functionality of our framework.

The second and third example cover two common diesel injector configurations: the unit pump and common rail. The data for configuring these cases was taken from reference examples provided by Gamma Technologies’ GT-SUITE software package GT-Suite (2007). The unit pump is much like the simple waterhammer case in that there are no branches in the system. The input is a defined flow specified by an electronically controlled cam driven pump. The output is a single valve. There are a total of 73 fluid sub-volumes in this system. The common rail example is more complex where the topology is roughly that described by the computational elements in Fig. 8.4. It has a total of 234 sub-volumes, including 5 “T” intersections and 4 valves. Both the GT-SUITE-based models use a 1 *cm* discretization length,

which, using a 1500 m/s wave speed, and a stability factor of 0.8 yields a $5.33\mu\text{s}$ time step to complete our worst-case instructions for the slowest computational element.

We synthesize all our cores and interconnects on the Xilinx Virtex 6 xc6vlx195t with speed grade 3. Each Virtex-6 FPGA logic slice contains 4 LUTs and 8 flip-flops, and this FPGA contains 31,200 logic slices and 512 18-*KB* BRAMs. Each PRET core is clocked at 150 MHz and has 6 threads. All floating point units are generated from the Xilinx Coregen tool xil (2012) and are configured to use the least amount of logic slices possible to meet the timing constraint. Our current PRET implementation uses an ARM-based ISA, thus our C code is compiled using the GNU ARM cross compiler gnu (2012) with the optimization compiler flag set to level 3. For these examples, we used a mapping heuristic that grouped nodes requiring same computations onto the same core. In the sections below we will show that this heuristic allows us to save hardware resources by synthesizing less floating point units.

8.6.2 Timing Requirement Validation

We need to ensure that the worst-case computational element can meet the timing requirements for our examples. A hardware context switch occurs every processor cycle, and threads are scheduled in a round robin order. Given a 150 MHz clock rate, each thread essentially executes at 25 Mhz . Thus, each thread cycle converted to physical time is 40 ns long. The unit pump and common rail have a requirement of $5.33\mu\text{s}$, which gives us 133 thread cycles to complete the computation each time step. Table 8.3 shows that the “T” element, which takes 111 thread cycles with interpolation, is the worst-case node. For the simple waterhammer example, a bigger discretization Δx is used, which leads to a bigger time step than that of the two complex examples. This validates that we can safely meet the timing requirements, ensuring the correctness of functionality.

8.6.3 Resource Utilization

Table 8.4: Number of Occupied Slices per Core on the Virtex 6 (xc6vlx195t) FPGA.

Threads per core	6	8	9	16
Fixed point only	572	588	764	779
Basic float	820	823	1000	1022
Float with sqrt	987	992	1146	1172
Float with div	1039	1051	1231	1237
Float with div & sqrt	1237	1249	1403	1413

Table 8.4 shows the resource usage for different configurations of a core. The synthesized area results consist of the cores, interconnects, and the global distribution circuit. This shows the direct impact of our framework. We include the fixed point configuration only for reference purposes, as it doesn’t contain any floating point units. The baseline configuration used in our implementation is the “basic float”, which contains a floating point add/subtractor, a floating point multiplier, and float to fix conversion units. The “sqrt”, “div” and “sqrt & div” configurations add the corresponding hardware units onto the “basic float” configuration. Besides the effect of hardware units, we also show the area impact of adjusting the thread count on a single core.

An interesting observation is that the area increase is approximately proportional only to the number of bits required to represent the thread count. E.g., 6 and 8 threads, which require three bits to represent, have a similar area usage. But once a 9th thread is introduced, the used area noticeably increases, but remains similar for up to 16 threads. This can be explained by understanding the architecture of multi-threaded processors. Multi-threaded processors maintain independent register sets and processor states for each thread, while sharing the datapath and ALU units amongst all threads. The register sets are synthesized onto BRAMs, so the number of bits used to encode thread IDs will determine how big of a BRAM is used for the

register set. The size of the muxes used to select thread states and registers is also determined by the number of bits encoding the thread IDs, not the actual number of threads running. As a result, increasing the core thread capacity can potentially reduce the number of cores required to fit a fixed number of nodes because it is possible to increase the thread count with only a small increase of area. However, since hardware threads share the processor pipeline, adding threads slows down the running speed of the individual threads. Nonetheless, for applications that have sufficient slack time or require faster performance, adjusting the number of threads could lead to a valuable improvement. Our implementation uses 6 threads, which is the maximum number of threads allowing us to meet our timing constraint for flow elements.

Comparison of the resource usage for 6 threads on a core to the “basic float” configuration gives that square root uses roughly 20.3% more slices and division uses roughly 26.7% more. A core with both square root and division would use roughly 50.8% more slices. These are estimates because the slices occupied might vary slightly based on how the synthesis tool maps LUTs and flip flops to logic slices. But they give an intuition to the resource difference used for each configuration.

Each core uses 7 BRAMs: 3 for the integer unit register set (3 read and 1 write port), 2 for floating point register set (2 read and 1 write port), 1 for the scratchpad, and 1 for the global broadcast receiving memory.

The actual resource impact can be seen from Table 8.5, which shows the total slices occupied when the three examples we implemented are synthesized. In the homogeneous (hom. suffix) configuration, all the cores contain the square root and divide hardware. In the heterogeneous (het. suffix) configuration, only necessary cores contain square root and divide, the rest use the basic float configuration.

For the simple waterhammer example, since only 2 cores are used, the savings is less noticeable. But as the application size scales up, the resource savings become

Table 8.5: Total Resource Utilization of Examples Synthesized on the Virtex 6 (xc6vlx195t) FPGA

Example		Nodes	Cores / Conn.	Slices / BRAM	
				Absolute	Relative (%)
Water Hammer	het.	12	2 / 1	1805 / 15	5.7 / 2.1
	hom.			2379 / 15	7.6 / 2.1
Unit Pump	het.	73	13 / 12	10566 / 103	33.0 / 15.0
	hom.			16635 / 103	44.0 / 15.0
Common Rail	het.	234	39 / 38	29134 / 311	93.4 / 45.0
	hom.			N/A	

more apparent. The homogeneous approach uses roughly 1.5 times the number of slices our heterogeneous approach uses, which is consistent with the findings of Table 8.4. This proved to be critical for the 234-node common rail example, as only our heterogeneous architecture could implement the design on the xc6vlx195t FPGA while the homogeneous design simply could not fit. These results also reflect our decision to use a heuristic that groups nodes with the similar computation together. By doing so, we can synthesize less hardware computation units overall, saving hardware resources.

Table 8.5 also shows the BRAM usage for the implemented examples. Each interconnect uses 1 BRAM and each core uses 7 BRAMs. We see that the BRAM utilization ratio is far below the logic cell utilization, validating our design choice of using BRAMs for interconnects and broadcasts.

8.7 Conclusions and Future Work

In this paper we presented a novel framework for solving a class of heterogeneous micro-parallel problems. Specifically we showed that our approach is sufficient to model a diesel fuel system in real time using the 1D CFD approach on FPGAs. We used the PRET architecture to ensure timing determinism and implement a

timing based synchronization of a multi-core system. We set up a configurable heterogeneous architecture that leverages the programmability of FPGAs to efficiently synthesize designs for efficient area usage. Our results show ample resource savings, proving that our approach is practical and scalable to larger and more complex systems.

We plan to continue to extend this work along several lines. From the application perspective, we continue to add more flow element types to our library and compare our results to more complex flow systems. We also plan to examine more closely the integration of mechanical and electrical nodes in our library. For the hardware architecture, we can to explore multi-rate timing of nodes to allow for differences in electrical, fluid, and mechanical timesteps.

Chapter 9

ICES2012-81138

Remote sensing of fuel systems using real-time 1D CFD

This draft paper has been accepted for publication in the proceedings of the ASME Internal Combustion Engine Division Spring Technical Conference in Torino, Italy. This paper was scored with honors and recommended for journal publication by the reviewers. This author will present the paper in Italy in May. This paper was written by Matthew Viele of Drivven, Isaac Liu from UC Berkeley, and Guoqiang Wang and Hugo Andrade from National Instruments, and Bryan Willson from Colorado State University

This paper is similar to the above IEEE paper, but with more focus on the models and less on the processor and FPGA implementation. It adds a section on mechanical and electrical co-simulation.

9.1 Synopsis

Many modern engine systems are designed using one-dimensional computational fluid dynamics (1D CFD). This same technique can be used to model these systems in real time. This real-time model can be used to create virtual sensors in places where due to environmental or cost reasons physical sensors would not be practical. Achieving real-time performance of the CFD model requires more throughput than is available on single processor systems, so an Field Programmable Gate Array (FPGA) is employed. By employing an FPGA, we can synthesize and reconfigure our system to ensure determinism and lower resource usage. We instantiate several dedicated processing cores for parallel processing of sub-volumes. The number of cores can be configured to support up to 500 fluid volumes, more

than enough for common 1D CFD models used in engines. This paper evaluates the feasibility of such a system and evaluates the complexity of such models against the GT-SUITE simulation software.

9.2 Introduction

For years engine designers have been using computer simulation to model diesel fuel systems (Kolade et al., 2003). Over the past decade commercial 1D CFD design software like GT-SUITE has become the tool of choice instead of custom code. Independently the engine control team builds models of the engine subsystems to implement in the controller. This is required because both the fuel injectors and piston style pumps for common rail fuel injectors cause pulsations in the fuel rail that need to be modeled or damped before the next injection event (Bauer, 2004; Winward et al., 2010). In modern common-rail diesel injection systems, it is expected that there are as many as 5 pulses per cylinder event (Drivven, 2009). With the advent of large Field Programmable Gate Arrays (FPGAs), we attempt to merge the two and allow the same basic model used to design the engine to be used in real-time as part of the engine controller. This allows us to use the 1D CFD model in real-time to measure properties like pressure and flow in places where installing a physical sensor would be difficult or cost prohibitive.

To achieve this we instantiate in the FPGA a large number of “soft-core processors” where each processor is responsible for a single node of the model. For instance, if a pipe is modeled with 100 subvolumes then we would assign each subvolume to one of 100 processors. The practical result of this is that something like a diesel common-rail fuel system can be modeled with around 200 soft-core processors on a single chip. Note that we use the term processors here, but later we will discuss multi-threaded processors and more precisely say that the systems will be implemented on 200 soft-core processor threads.

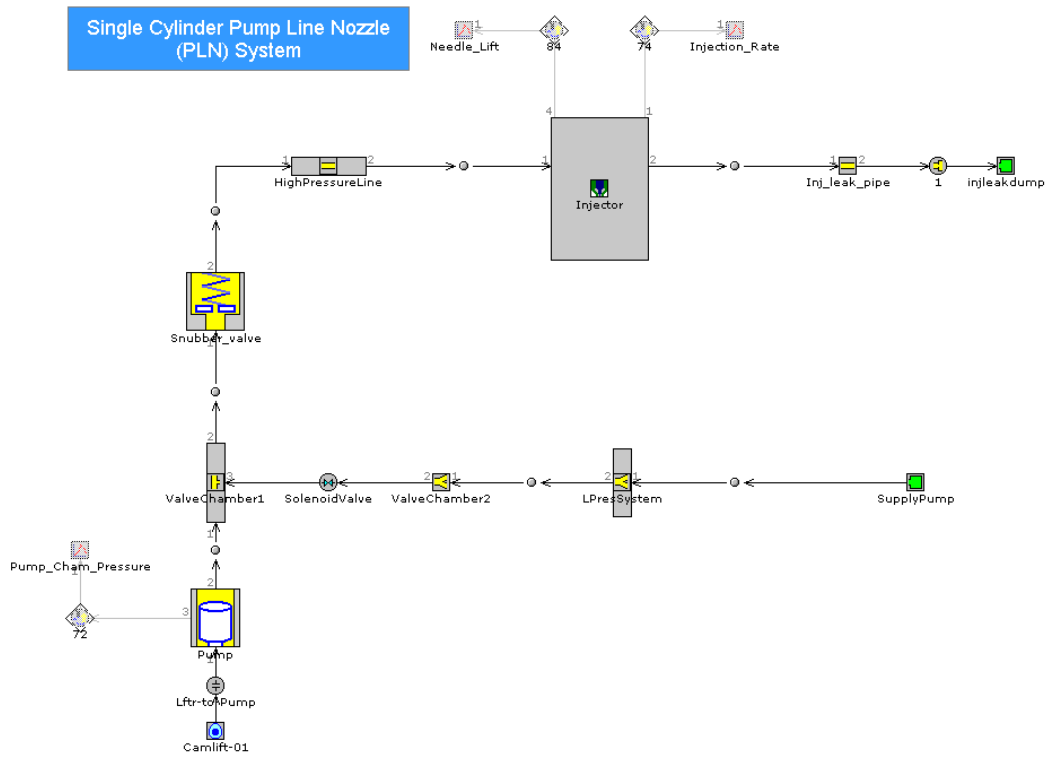


Figure 9.1: GT-SUITE 4-Cylinder Unit Pump Example

We have focused our initial efforts on fuel systems instead of air systems for a number of reasons. First, the basic equations are simpler as pressure and flow are the only communicated variables and no state or reaction equations are needed. Second, fuel systems tend to have smaller discretization lengths and higher wave speed making them more interesting to examine in the context of our low-latency processor architecture. Lastly, the number of non-flow elements required to model a fuel system is much smaller than an air system which requires a combustion model and possibly turbo models. This same architecture can be applied to air systems by using a different, and larger library of components.

We focus our efforts on matching our problem to two examples provided by Gamma Technologies GT-SUITE application: Figure 9.1 shows a simple unit pump

example for a single cylinder while Figure 9.2 shows a common-rail fuel system for a hypothetical 4-cylinder diesel engine.

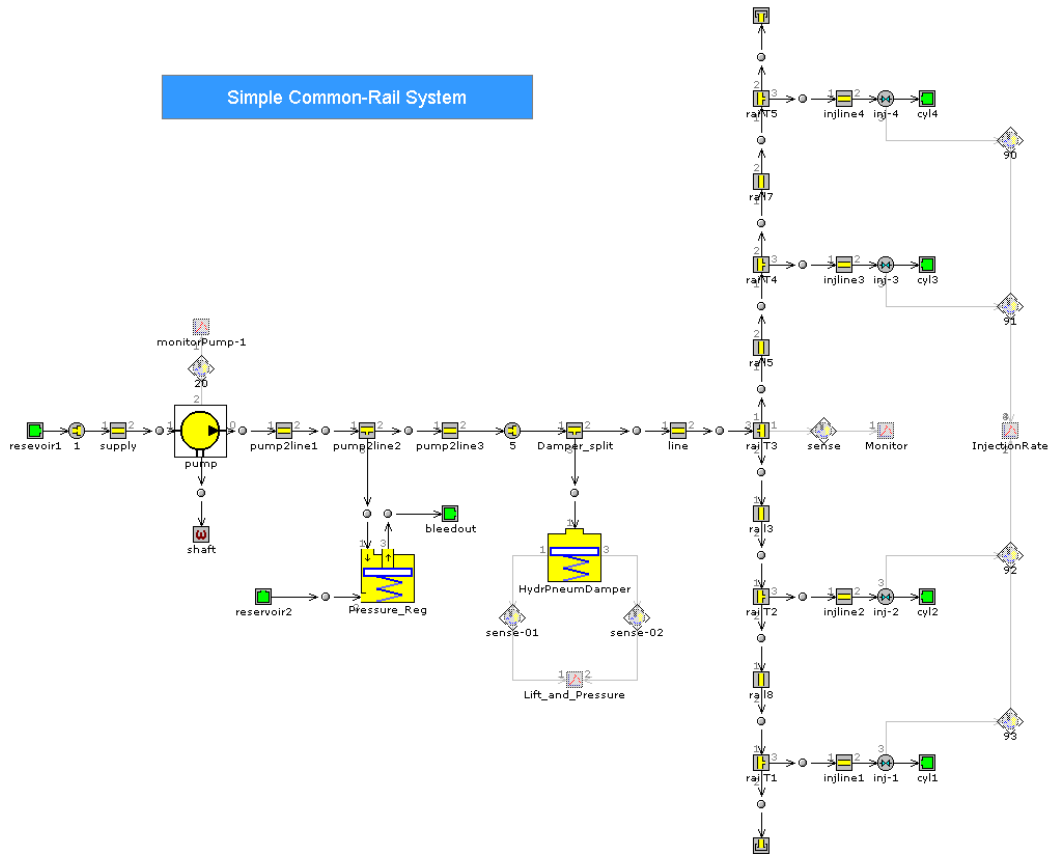


Figure 9.2: GT-SUITE 4-Cylinder Common-Rail Example

9.3 Computational Model

The bulk of any fuel system model will consist of one-dimensional fluid dynamics nodes. However in most practical system there are dampers or valves whose actions are influenced by the fluid flow. In this section we build up the library of computational elements used in our system.

9.3.1 One-Dimensional Computational Fluid Dynamics

Solution of 1D CFD problems begins with the Navier-Stokes equations for compressible flow (Wylie and Streeter, 1978). We then build a library of solution elements for the type of pipe segments we use in the form of first order finite difference equations (Desantes et al., 1999). We start with momentum equation

$$\frac{P_x}{\rho} + \dot{V} + \frac{fV}{2D}|V| = 0 \quad (9.1)$$

and continuity equation

$$a^2V_x + V \left(\frac{P_x}{\rho} + g \sin \alpha \right) + \frac{P_t}{\rho} = 0 \quad (9.2)$$

where P is pressure, ρ is fluid density, V fluid velocity, f is the Darcy-Weisbach friction factor, D is pipe diameter, a is the wave speed, and $g \sin \alpha$ is the directional force of gravity. A dot ($\dot{\cdot}$) over a variable indicates the total derivative with respect to time and the “ x ” and “ t ” subscripts respectively indicate partial differentiation along the pipe length and with respect to time.

These equations can be expanded and simplified by leaving out the body force and convective terms because these are negligible given the pressure and flow regime. This leads us to the following equations in the form $L = L_1 + \lambda L_2$:

$$L_1 = \frac{P_x}{\rho} + V_t + \frac{f}{2D}V|V| = 0 \quad (9.3)$$

and

$$L_2 = \frac{P_t}{\rho} + a^2V_x = 0. \quad (9.4)$$

A method of characteristic solution is used to explicitly evaluate the pressure and flow at the next step through a first order finite difference method. Figure 9.3

shows the evaluation of pressure and flow at point $I, t_0 + \Delta t$ based on the pressure and flow of the adjacent points at time t_0 . The equations to be evaluated at each point are

$$V_I = 0.5 \left[V_{i-1} + V_{i+1} + \frac{1}{a\rho} (P_{i-1} - P_{i+1}) - \frac{f\Delta t}{2D} (V_{i-1}|V_{i-1}| + V_{i+1}|V_{i+1}|) \right] \quad (9.5)$$

and

$$P_I = 0.5\rho \left[\frac{1}{\rho} P_{i-1} + P_{i+1} + a(V_{i-1} - V_{i+1}) - \frac{af\Delta t}{2D} (V_{i-1}|V_{i-1}| + V_{i+1}|V_{i+1}|) \right]. \quad (9.6)$$

One critical part in evaluating the value at point I is that there is a fixed relationship between Δx , Δt , and the wave speed a such that $a = \Delta x/\Delta t$. Δx is fixed by the geometry of the problem and a is fixed by the properties of the working fluid such as $a = \sqrt{K/\rho}$, where K is the bulk modulus. Therefore, Δt is defined as above. All of our computations must be complete within Δt and the results must be posted at exactly Δt . Because the wave speed varies and the geometry of the problem may not work out evenly for all pipe segments a modified method is implemented with an interpolation step. This is shown in Figure 9.3 and described by equations

$$\zeta_R = \zeta_i - \theta a(\zeta_i - \zeta_{i+1}) \text{ and} \quad (9.7)$$

$$\zeta_L = \zeta_i - \theta a(\zeta_i - \zeta_{i-1}), \quad (9.8)$$

where θ represents the amount of interpolation desired. These equations are evaluated for both pressure $\zeta = P$ and velocity $\zeta = V$. While we use this method to give

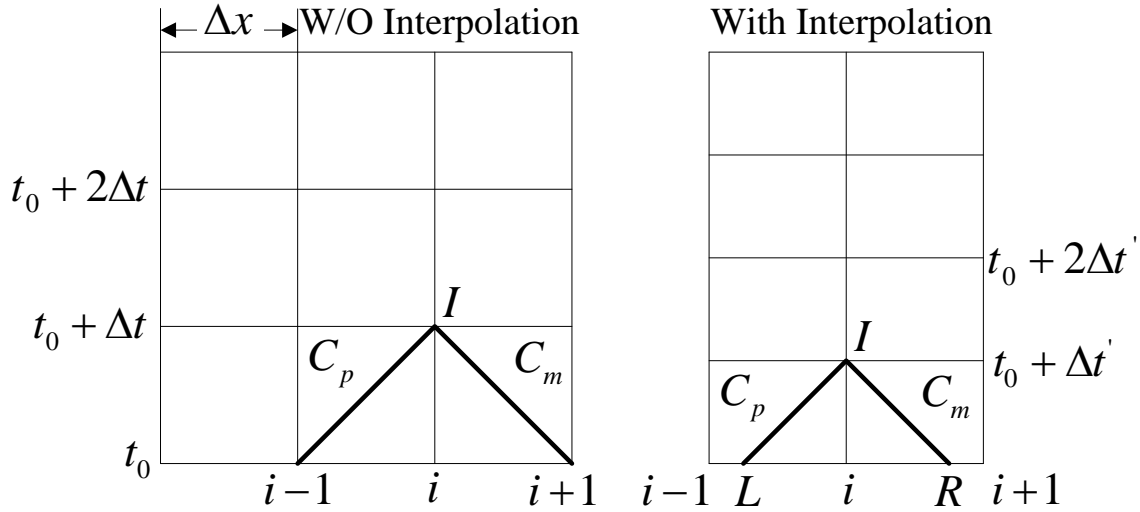


Figure 9.3: First Order Difference

our calculations leeway it is important to realize that this adds complexity to every block in the system as well as decreases the Δt we have to work with.

In order to evaluate the system of pipes we define a few types of computing nodes that correspond to different pipe elements. Specifically, the types are: 1) Pipe Segment; 2) Imposed pressure upstream, representing the pressure sensor on the fuel system; 3) Imposed mass flow into the pipe, representing a pump; 4) Valve at downstream end of pipe, representing an injector; 5) Cap at downstream end of pipe; and 6) “T” intersection. Different types of computing nodes execute different algorithms on the same hardware.

Rearranging the equations again and making the substitutions $B = a\rho/A$ and $R = \rho f \Delta x / 2DA^2$, where A is the cross sectional area of the pipe, and Q is the flow rate along the pipe, we get the simplified characteristic equations

$$C_p = P_{i-1} + Q_{i-1} (B - E|Q_{i-1}|) \text{ and} \quad (9.9)$$

$$C_m = P_{i+1} - Q_{i+1} (B - E|Q_{i+1}|). \quad (9.10)$$

Table 9.1: Equations for Supported Types

Type	$P_I =$	$Q_I =$
Pipe Seg.	$\frac{(C_p+C_m)}{2}$	$\frac{(P_I+C_m)}{B}$
Imp. P	P_{Bnd}	$\frac{(P_{Bnd}-C_m)}{B}$
Imp. Q	$C_m+B \cdot Q_{Bnd}$	Q_{Bnd}
Valve	$C_p-B \cdot Q_{In}$	$-BC_V + \sqrt{(BC_V)^2 + 2C_V C_P}$ $C_V = \frac{(Q_0 \tau)^2}{2 \cdot P_0}$
Cap	$C_p-B \cdot Q_{In}$	0
Pipe "T" Generic	$\frac{\frac{C_{p1}}{B_1} + \frac{C_{m2}}{B_2} + \frac{C_{m3}}{B_3}}{\sum_{j=1,3} \frac{1}{B_j}}$	$-\frac{P_I}{B_1} + \frac{C_{p1}}{B_1}$ $-\frac{P_I}{B_2} + \frac{C_{m2}}{B_2}$ $-\frac{P_I}{B_3} + \frac{C_{m3}}{B_3}$
Pipe "T" Standard	$\frac{\frac{C_{p1}}{B_1} + \frac{C_{m2}}{B_2} + \frac{C_{m3}}{B_1}}{\sum_{j=1,3} \frac{1}{B_j}}$	$-\frac{P_I}{B_1} + \frac{C_{p1}}{B_1}$ $-\frac{P_I}{B_2} + \frac{C_{m2}}{B_2}$ $-\frac{P_I}{B_1} + \frac{C_{m3}}{B_1}$
Pipe "T" Simple	$\frac{C_{p1}+C_{m2}+C_{m3}}{3}$	$-\frac{P_I}{B} + \frac{C_{p1}}{B}$ $-\frac{P_I}{B} + \frac{C_{m2}}{B}$ $-\frac{P_I}{B} + \frac{C_{m3}}{B}$

Table 9.1 shows the equations for each of the supported pipe elements. From these pipe elements we can generate a network of pipes that represents our fuel system. The Bnd subscript denotes a boundary condition. C_v is the flow coefficient of Q_0 , the nominal open flow, P_0 the downstream pressure and τ the fraction the valve is open.

9.3.2 Mechanical Subsystem

We examined two mechanical subsystems for this model: the diesel fuel injector and the flow damper.

9.3.2.1 Mechanical Equilibrium Governing Equation

Common rail system injector modeling has been studied intensively in literature (Bianchi et al., 2002b)(Bianchi et al., 2002a)(Sprich, 2011)(Dongiovanni and Coppo,

2010). In the following, we briefly summarize the mathematical model that captures the mechanical dynamics of devices in an injector.

There are three moving injector components: control valve, needle, and control piston. They can be modeled as mass-spring-damper assemblies. Their mechanical dynamics can be described via the conventional governing equation of a mass-spring-damper assembly:

$$m_i \frac{d^2 x_i}{dt^2} + \beta_i \frac{dx_i}{dt} + k_i x_i + F_{0,i} = F_i, \quad (9.11)$$

where m_i is the mass of the moving device (i can be control valve, needle, or control piston), x_i the device position, β_i the damping coefficient, k_i the spring stiffness, $F_{0,i}$ the total spring pre-load, and F_i is the external force.

Depending on their position, the moving device damping coefficient (β_i), spring stiffness (k_i), and spring pre-load ($F_{0,i}$) can be computed using relevant damping coefficients and spring stiffnesses. The external force can be due to interaction force between two adjacent masses, pressure, or electromagnetic action. The interaction forces can be expressed as a function of corresponding device position displacement as well as effective spring stiffness and viscous damping coefficient.

9.3.2.2 Numerical Integration

There exist different approaches to numerically solve the set of mechanical equilibrium differential equations, e.g., the Euler method, the trapezoidal method, the Runge-Kutta method, etc. In our study, we take the classical Runge-Kutta Method (a.k.a. RK4) to solve the set of ordinary differential equations since this method gives good numerical accuracy (Desantes et al., 1999)(Sprich, 2011).

Given the following first order ordinary differential equation,

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0 \quad (9.12)$$

the RK4 method used for integration can be described as follows:

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$t_{n+1} = t_n + h,$$

$$k_1 = hf(t_n, y_n),$$

$$k_2 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right),$$

$$k_3 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right),$$

$$k_4 = hf(t_n + h, y_n + k_3),$$

where h is the time step for integration.

However, notice that the mechanical equilibrium governing equation (9.11) is much more complex than (9.12) in that (9.11) is a second order differential equation and it has more than two variables in the expression of first order of derivative. The second order derivative can be solved via introducing the following intermediate variable,

$$\frac{dx_i}{dt} = z_i. \quad (9.13)$$

With this, (9.11) is transformed into

$$m_i \frac{dz_i}{dt} + \beta_i z_i + k_i x_i + F_{0,i} = F_i. \quad (9.14)$$

Taking the expression of the other terms into account, (9.14) is essentially

$$\frac{dz_i}{dt} = f_i(t, z_i, x_i, x_{j_1}, \dots, x_{j_{N_i}}), \quad (9.15)$$

where j_1, \dots, j_{N_i} are neighboring interacting masses of mass i . Ordinary differential equation (9.15) can be solved using the RK4 method.

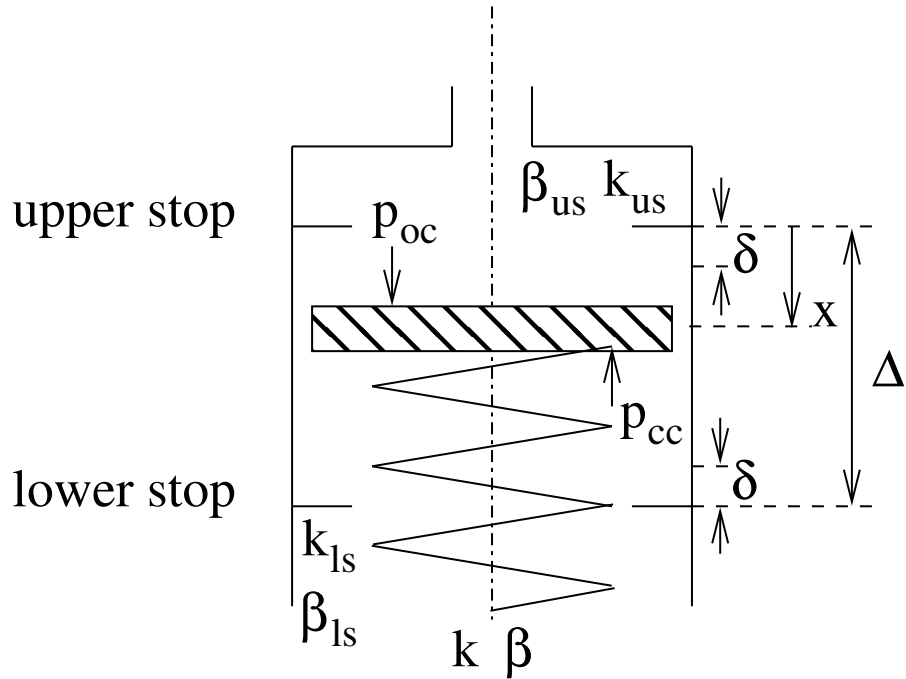


Figure 9.4: Schematic of Hydraulic Damper

9.3.2.3 Modeling a Hydraulic Damper

Figure 9.4 shows the schematic of a hydraulic damper. To model its dynamics, the general governing equation (9.11) simplifies to

$$m \frac{d^2 x}{dt^2} + \beta \frac{dx}{dt} + kx + F_0 = F. \quad (9.16)$$

The model consists of one mass with its value to be the sum of the mass of the diaphragm and a fraction of the spring. F_0 is the pre-load of the spring. The

external force can be computed as follows:

$$F = \begin{cases} (p_{oc} - p_{cc})A - k_{us}x - \beta_{us} \frac{dx}{dt}, & x < 0 \\ (p_{oc} - p_{cc})A - \beta_{us} \frac{dx}{dt}, & 0 \leq x \leq \delta \\ (p_{oc} - p_{cc})A, & \delta < x < \Delta - \delta \\ (p_{oc} - p_{cc})A - \beta_{ls} \frac{dx}{dt}, & \Delta - \delta \leq x \leq \Delta \\ (p_{oc} - p_{cc})A - k_{ls}(x - \Delta) - \beta_{ls} \frac{dx}{dt}, & \Delta < x, \end{cases} \quad (9.17)$$

where A is the cross sectional area of the damper and δ stands for the maximum gap to model damping for both upper and lower stops. Substituting F in (9.11) with the above expression gives

$$m \frac{d^2x}{dt^2} + \bar{\beta} \frac{dx}{dt} + \bar{k}x + F_0 - (p_{cc} - p_{oc})A = 0, \quad (9.18)$$

where

$$\bar{\beta} = \begin{cases} \beta + \beta_{us}, & x \leq \delta \\ \beta, & \delta < x < \Delta - \delta \\ \beta + \beta_{ls}, & \Delta - \delta \leq x \end{cases} \quad (9.19)$$

and

$$\bar{k} = \begin{cases} k + k_{us}, & x < 0 \\ k, & 0 \leq x \leq \Delta \\ k + k_{ls}, & \Delta < x. \end{cases} \quad (9.20)$$

The three cases for $\bar{\beta}$ are used to specify the difference in damping near the end stops. Similarly we treat the end stops as stiff springs so we have three regions for

\bar{k} as well. With (9.13), (9.18) becomes:

$$\frac{dz}{dt} = \frac{-1}{m} (\bar{\beta}z + \bar{k}x + F_0 - (p_{cc} - p_{oc})A). \quad (9.21)$$

With appropriate initial conditions (e.g., $x = 0, z = 0$), RK4 gives

$$\begin{aligned} z_{n+1} &= z_n + \frac{1}{6} (k_1^z + 2k_2^z + 2k_3^z + k_4^z), \\ k_1^z &= \frac{-h}{m} (\bar{\beta}z_n + \bar{k}x_n + F_0 - (p_{cc} - p_{oc})A), \\ k_2^z &= \frac{-h}{m} \left(\bar{\beta} \left(z_n + \frac{k_1^z}{2} \right) + \bar{k} \left(x_n + \frac{k_1^x}{2} \right) + F_0 - (p_{cc} - p_{oc})A \right), \\ k_3^z &= \frac{-h}{m} \left(\bar{\beta} \left(z_n + \frac{k_2^z}{2} \right) + \bar{k} \left(x_n + \frac{k_2^x}{2} \right) + F_0 - (p_{cc} - p_{oc})A \right), \\ k_4^z &= \frac{-h}{m} (\bar{\beta} (z_n + k_3^z) + \bar{k} (x_n + k_3^x) + F_0 - (p_{cc} - p_{oc})A), \\ k_1^x &= k_2^x = k_3^x = hz_n, \end{aligned}$$

and

$$x_{n+1} = x_n + hz_{n+1}. \quad (9.22)$$

For the purposes of our real-time code we can greatly simplify the above equations by expanding the equations and then grouping all the constants. For given ranges of x the values $\bar{\beta}$ and \bar{k} are constant. This yields

$$z_{n+1} = C_1 z_n + C_2 x_n + C_3 (F_0 - (p_{cc} - p_{oc})A), \quad (9.23)$$

where C_n are constants for a specific operating region of the problem as defined in (9.17) and are selected from a table at the beginning of each time step.

In order to examine the fuel injector we modify the force equation slightly.

$$z_{n+1} = C_1 z_n + C_2 x_n + C_3 (F_0 - (p_{cc} A_1 - p_{oc} A_2) - \epsilon) \quad (9.24)$$

Equation (9.24) is similar to Equation (9.23) except that we have split the area terms to handle separate areas and have added an electrical force ϵ . We make the simplifying assumption for our computation calculations in Table 9.3 that ϵ is a constant that is either on or off, but that A_2 is an arbitrary function of x .

9.4 Real-Time Implementation

The process of implementing a model on real-time hardware is summarized by

1. Design a fuel system using the components described in the library;
2. Determine the timing constraints on the system;
3. Feed in graph of system and timing constraints to the optimizer;
4. Take the optimized graph, calibration constants, and initial values into the code generator;
5. Integrated model into I/O system;
6. Build target and deploy.

9.4.1 Field Programmable Gate Arrays

FPGAs are generic digital logic that can be configured as application specific digital hardware. In automotive research applications they have been traditionally used as timing coprocessors to execute timing critical operations such as controlling fuel and spark timing (Viele et al., 2011). They have also been employed as digital

signal processors to process knock or cylinder pressure (Quillen et al., 2010) and respond with lower latency than would be available with a CPU.

More generically, FPGAs can be thought of as doing many simple tasks in parallel as opposed to one complex task at a time. As a processor’s program gets more complex its loop time slows down, but as an FPGA’s program get more complex it requires more FPGA “gates”. FPGAs are sized in terms of logic gates or some similar metric to define how complex a program they can execute.

For our purposes, one of the most interesting things about FPGAs is their ability to instantiate processors inside them. While these do not rival desktop processors like Intel’s Pentium they can be tailored to perform specific tasks well and we can create many of them along with specialty I/O hardware to bring I/O latency and cycle times well below what we expect on desktop processors.

FPGAs grow in performance like DRAM and follow Moore’s law for performance by doubling in size every 18 months(Trushard, 2010). Figure 9.5 shows the performance of FPGAs over time and give a sense of how the application we describe can scale.

9.4.1.1 Deterministic Processors

The advent of caches, out-of-order executions, branch prediction, and other performance improvements in modern processors have improved their average execution speed at the cost of determinism. In order to bring the worst-case executing time of our system closer to the average execution time we base our soft-core processors on the The Precision Timed (PRET) Architecture (Lickly et al., 2008). It contains multiple deterministic hardware threads, and timing instructions to gate execution time of code blocks on the hardware threads. Thus we can ensure that our worst-case timing constraints are met, making them suitable for hard real-time applications.

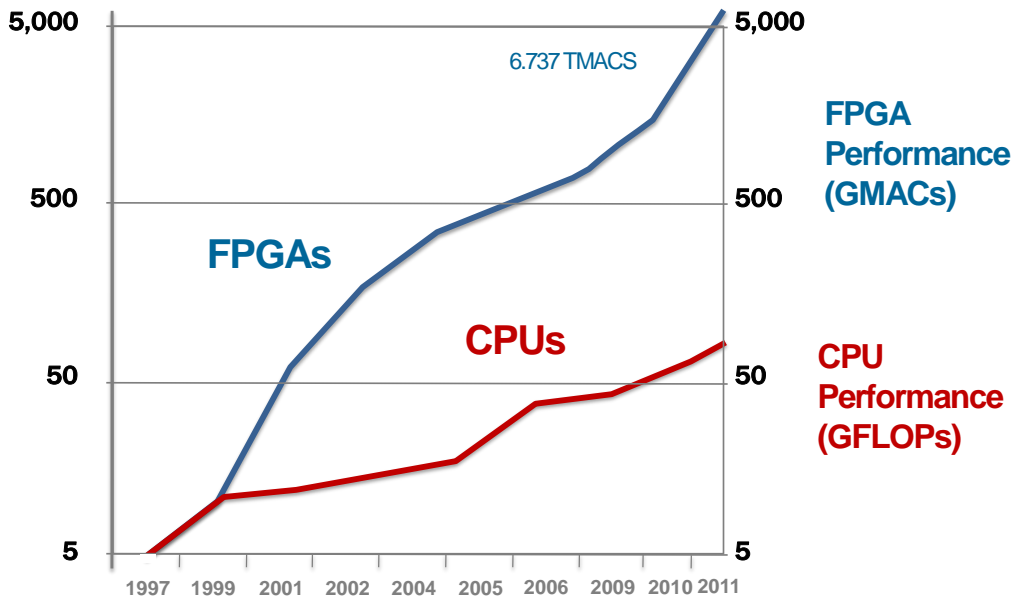


Figure 9.5: FPGA Size over Time

With this processor core as a foundation we make several additions to meet the needs of 1D CFD. First we use a heterogeneous approach to hardware optimization (Liu et al., 2011). For instance processors that are used to compute valve position have hardware square root instructions, but processors that are used for simple pipe segments do not.

9.4.2 System Description

Figure 9.6 shows an overview of a representative system for modeling fuel rails. The 1D CFD model is bounded inside the dashed rectangle. External to that is the real-world sensor and actuator interfaces that provide boundary conditions or consume model output variables. The small blue squares inside the dashed rectangle represent the network of pipes. In a practical simulation of a diesel fuel system the total number of pipe elements can range from around 50 to a few hundred.

Each pipe element is a computational node, and their graphical representation is shown in Table 9.2. The top 3 rows of the table represent the pipe elements

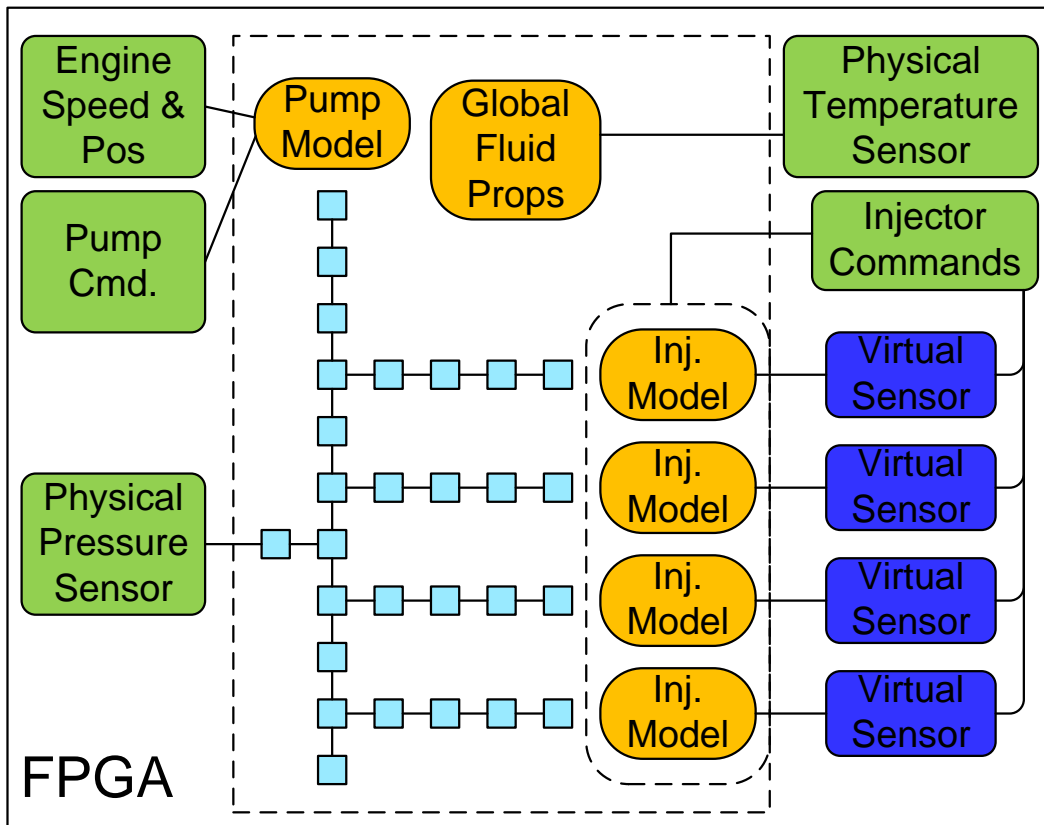


Figure 9.6: High Level System Diagram











described in Table 9.1. Mechanical calculation elements compute the inputs to valve, defined flow and defined pressure blocks. They serve as an interface between the flow model and the real world. Global calculation elements are used to compute the temperature dependent variables of density and wave speed. They post their data to a global distribution block, broadcasting it to all nodes. Blocks with white backgrounds are implemented directly in FPGA fabric, not mapped to a processor thread. The Global Distribution node is purely used to indicate a distribution of input value to each of the computational elements in our model. The Display node is used when data needs to be communicated out of the model through a register or FIFO to other parts of the FPGA.

For illustrative purposes, we show a simplified sample pipe network with an imposed flow input in Figure 9.7. Fluid travels through a few pipe segment nodes to a “T” intersection, where it splits off to a second branch of the network. The “T” node is also measured by the outside world through a display port. Flow going up the new leg ends in a cap, while flow continuing down the original path exits the system through a valve. The system is assumed to be at uniform temperature and values based on the temperature dependent variables of density and wave speed are delivered to each of the computational elements every time step for use in the subsequent time step. For the purposes of system analysis we assume that flow always enters through the left and exits to the right and possibly top of the network element. This lets us describe the system as a directed graph.

9.4.3 System Implementation

Each of the pipe elements described in Table 9.2 is hand coded in C. Each pipe element is mapped to a hardware thread on the underlying PRET architecture. The number of thread cycles each takes to complete is shown in Table 9.3. Dedicated floating point square root and division hardware is added to processors where threads

Table 9.2: Library of Computational Node Elements

Pipe		Cap	
Imposed Pressure		Imposed Flow	
Pipe "T"		Valve	
Mechanical Calc.		Global Calc.	
Global Dist.		Display	

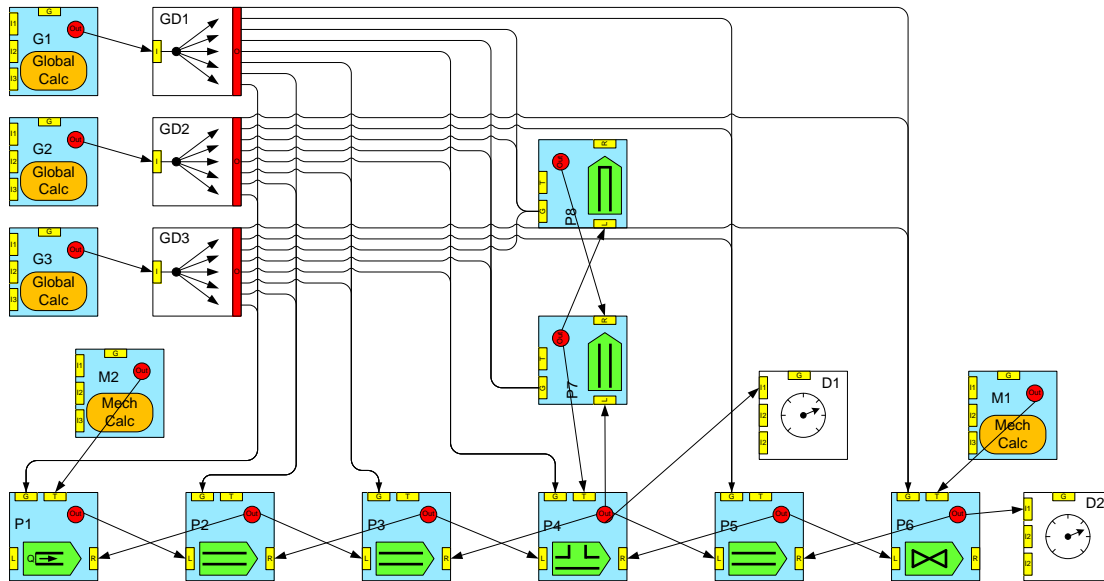


Figure 9.7: Detailed System Diagram

Table 9.3: Computational Intensity of Supported Types

Type	Without Interpolation / With Interpolation					
	Mul	Add/Sub	Abs	Sqrt	Div	Thread cycles
Pipe Seg.	10 / 18	5 / 13	2 / 2	0 / 0	0 / 0	81 / 51
Imp. P	6 / 10	3 / 7	1 / 1	0 / 0	0 / 0	50 / 38
Imp. Q	5 / 9	3 / 7	1 / 1	0 / 0	0 / 0	51 / 40
Valve	13 / 17	5 / 9	1 / 1	1 / 1	0 / 0	64 / 55
Cap	4 / 8	2 / 6	1 / 1	0 / 0	0 / 0	48 / 39
"T" Gen.	16 / 28	13 / 25	3 / 0	0 / 0	4 / 4	111 / 72
"T" Std.	15 / 27	13 / 25	3 / 3	0 / 0	3 / 3	106 / 67
"T" Smp.	14 / 26	11 / 23	3 / 3	0 / 0	1 / 1	109 / 58
Mech-Dmpr.	5	11	0	0	0	45
Mech-Inj.	9	14	0	0	0	66

require these operations. Also shown are our mechanical elements the mechanical damper and fuel injector. Both of these represent single spring-mass-damper systems with fixed area and variable area respectively for the pressure force to act on. It is important to note that more complex system masses can be created by attaching the mechanical elements together in a similar fashion to the flow elements.

The timing constraints of the system is determined by the wave speed and discretization length of our model. The execution time of each time step is determined by our slowest element, which needs to meet the timing constraint defined above.

In our simulation model we are running the mechanical system at the same rate as the fluid system. GT-SUITE varies the time step, by default by up to a ratio of 5 mechanical time steps to one fluid time step. We experimented with this ratio in GT-SUITE and found trivial variation in the output. It is worth noting, however, that the mechanical system implementation is small enough that a ratio of 2:1 may be easily achieved on our model, though we have not yet implemented this feature.

9.5 Optimization

While the cost of FPGA gates is rapidly decreasing, large designs like a full common-rail model still push the bounds of FPGA parts that are readily available. In order to reduce the size of the FPGA we employ several optimization steps.

1. Minimize the amount of code used by the worst-case computational element;
2. Make use of heterogeneity of processor cores, i.e. only instantiate operators that are required by the allocated threads;
3. Balance threading and clock speed of each core to minimize FPGA circuit area;
4. Ensure the number of interconnects is not excessive.

9.5.1 Worst-Case Computation Time

It is important to understand that when modeling physical quantities, such as fluid flow, the time step is determined by the granularity of the application. For an explicit solver that uses a fixed time step, it is required that the solver run faster than the speed of information flow. This is expressed in (9.25), where a is the wave speed and C is the Courant number. For stability the Courant number needs to be less than 1 and a number below 0.8 is recommended (GT-Suite, 2007).

$$\frac{\Delta t}{\delta x} a = C \quad (9.25)$$

Our discretization length of $1cm$ and wave speed of $1500m/sec$ with a margin of 0.8 give us a $5.33\mu s$ time step. As discussed below we run a 20MHz thread clock giving us 106 clock cycles per time step, just enough to execute the Standard Pipe “T” configuration shown in Table 9.3, which is the slowest element in the two examples we studied.

9.5.2 Maximize Threads Per Core

The PRET architecture is a multi-threaded architecture that contains several hardware threads which are interleaved through the pipeline in a round robin fashion each cycle. The hardware threads are non-interfering and cannot effect the execution of each other. The more threads we design into the processors, the more concurrency we can support on the FPGA. But all threads on each processor share the processor resource, so the latency of each thread reduces as we increase the number of threads. Due to the FPGA implementation, noticeable area increase is only observed when we increase across the powers of two (Liu et al., 2011). We can see this in Table 9.4 where we benchmarked different processor configurations. For example, when we increase from 8 threads to 9 threads.

Table 9.4: Number of Occupied Slices per Core on the Virtex 6 FPGA (XC6VLX195T).

Threads per Core	6	8	9	16
Fixed Point Only	572	588	764	779
Basic Float	820	823	1000	1022
Float w/sqrt	987	992	1146	1172
Float w/div	1039	1051	1231	1237
Float w/div & sqrt	1237	1249	1403	1413

Table 9.5: Total Resource Utilization of Examples Synthesized on the Virtex 6 FPGA (XC6VIX195T).

Example	Nodes	Cores	Slices / BRAM (%)
Unit Pump	73	13	10566 / 103 (33% / 15%)
CR System	234	39	29134 / 311 (93.4% / 45%)

The additions of a floating square root and floating point division unit increases the area of a single processor on an FPGA by 50%. The heterogeneity of the processors in our system design allows us to greatly reduce resource usage.

In (Liu et al., 2011) we examined the area consumed by our models on a Xilinx Virtex 6 (part number XC6VLX195T), as this was the smallest FPGA we could make this particular example run on. Table 9.5 shows resources used by our two systems.

We bechmarked the number of 8 thread cores we could practically implement on a Xilinx Virtex 6 FPGA (part number XC6VVSX315T), which is the middle of the size range in the Virtex 6 family. We found that we could implement 45 processor cores with single precition floating point, sqrt, and divide hardware on the FPGA using 96% of its area. That corresponds to 360 threads total, because we have 8 threads on each core. We also examined the case where we filled the FPGA with processor cores that had the same floating point math except for the hardware sqrt and divide. In that case we could fit 63 cores onto the FPGA using 94% of the

slices, slices being a representation of the generic logic portion of the FPGA. That is 504 total runnable threads. Our heterogeneous approach uses a mix of several cores types, but with the preponderance being the basic cores. This means that we expect on this size FPGA to be able to run fuel injection configurations of over 400 computational nodes.

9.6 Conclusion and Further Work

We have presented a practical system to implement 1D CFD problems in real-time. We did this by building a library of coupled flow and mechanical elements that we could evaluate on soft-core processors. We benchmarked these against examples provided in GT-SUITE and determined how many processor threads we would need and how fast they would need to run. We then showed that using a heterogeneous collection of PRET processors we could implement these on a Xilinx XC6V1X195T.

We examined the maximum size of model that we can fit in a middle of the road Xilinx Virtex 6 part and found that we can fit in the neighborhood of 400 nodes depending on the mix of the actual problem. This should be adequate for many practical fuel injection applications.

In order to make this a practical solution the design process needs to be automated. We need to be able to generate not only the layout but the calibration constants and initial conditions for each computational element. This must be done by compiling from another simulation environment because it is far too tedious and error prone to do by hand. This is the next major task ahead of us.

We have verified a building blocks of a system (Liu et al., 2011). Next we need to fully automate the model generation process so we can easily validate our system against more complex models and finally start testing on real physical systems.

We have shown mechanical models that can run at a 2:1 step ratio from our fluid system, but we need to incorporate a mechanism to allow multi-rate communication.

9.7 Acknowledgement

Thanks to John Silvestri at Gamma Technologies for his support over the years as we developed this concept.

Chapter 10

PATENT 7,991,488

The following patent was filed early in this research with this author as the sole inventor and was issued August 2011. It patents the idea of FPGA-based, real-time 1D-CFD. The crucial part of the patent is the 1:1 or near 1:1 relationship between solver nodes (physical part of the problem) and computational elements (processors running equation solvers).

This patent describes a system running in real-time with one or more inputs and outputs. Inside the system is an array of computational elements. Each element is solving a small number of fluid nodes connected in a 1D-CFD arrangement.

This patent covers the ideas explained in the previous two papers.

Apparatus and method for use in computational fluid dynamics

10.1 Synopsis

An apparatus includes a parallel computation unit including an input port and an output port and a one-dimensional computational fluid dynamics model. The input port is configured to sample at a time t_1 a boundary condition signal for the one-dimensional computational fluid dynamics model and the output port is configured to provide an output signal before the boundary condition signal is sampled at a time t_2 .

10.2 Claims

What is claimed is:

1. An apparatus comprising: a plurality of parallel computation units configured to implement a one-dimensional computational fluid dynamics model for controlling a physical system, wherein each parallel computation unit comprises a respective plurality of computation units, and wherein each parallel computation unit is associated with a respective node of the one-dimensional computational fluid dynamics model; wherein the plurality of parallel computation units are configured to implement the one-dimensional computational fluid dynamics model to receive a first sample of a boundary condition signal for the one-dimensional computational fluid dynamics model at a time t_1 , wherein the boundary condition signal represents a first physical variable sampled at a first location in the physical system; and wherein the plurality of parallel computation units are configured to implement the one-dimensional computational fluid dynamics model to generate an output signal representing the first physical variable at a second location in the physical system, wherein the second location is not sampled in the physical system, and wherein the

output signal is usable for controlling the physical system; wherein the plurality of parallel computation units are configured to implement the one-dimensional computational fluid dynamics model to generate the output signal before a second sample of the boundary condition signal is received at a time t_2 .

2. The apparatus of claim 1, wherein the difference between the time t_2 and the time t_1 is between ten microseconds and ten milliseconds.

3. The apparatus of claim 1, wherein the apparatus is implemented as a field programmable gate array that implements the plurality of parallel computation units.

4. The apparatus of claim 1, wherein the physical system is an engine, wherein the apparatus is implemented as an engine control unit that controls the engine.

10.3 Description

10.3.1 Field

The subject matter of the disclosure relates to computational fluid dynamics and, more particularly, to real time computational fluid dynamics.

10.3.2 Background

Computational fluid dynamics uses mathematical methods to solve problems that include fluid flow. An exemplary problem in the field of computational fluid dynamics is the problem of predicting the pressure at any point along a fuel rail of an operating diesel engine as a function of time. A real time solution to this problem would enable the design and manufacture of improved engines. These improved engines would provide higher performance and lower pollution levels than engines available today. At this time, predictions of the real time operation of engines are obtained by running simulations using computational fluid dynamics models on a

supercomputer or workstation. Unfortunately, the predictions that result from such simulations require hours of supercomputer time to predict a few seconds of engine operation. They are not performed in real time. Finally, when these predictions are incorporated in a real time engine control system they do not yield the desired results.

10.3.3 Breif Description of the Drawings

The disclosed embodiments may be understood with reference to the following drawings, in which like elements are indicated by like numbers. These drawings are provided to illustrate selected embodiments of the disclosure and are not intended to limit the scope of the claims.

FIG. 1A shows a block diagram of an apparatus including a boundary condition signal for a one-dimensional computational fluid dynamics model and a parallel computation unit to receive the boundary condition signal and provide an output signal in accordance with some embodiments.

FIG. 1B shows a timing diagram illustrating the relationship between the sampling time t_1 and the sampling time t_2 of the boundary condition signal shown in FIG. 1A and the output signal shown in FIG. 1A in accordance with some embodiments.

FIG. 1C shows a block diagram of an apparatus including the apparatus of FIG. 1A included in a field programmable gate array in accordance with some embodiments.

FIG. 1D shows a block diagram of an apparatus including the apparatus of FIG. 1A included in an engine control unit in accordance with some embodiments.

FIG. 1E shows a block diagram of an apparatus including the apparatus of FIG. 1A and including a reciprocating engine model included in the one-dimensional computational fluid dynamics model in accordance with some embodiments.

FIG. 2 shows a flow diagram of a method including receiving a sensor signal and processing the sensor signal using real time computational fluid dynamics methods to generate a virtual sensor signal in real time in accordance with some embodiments.

FIG. 3A shows a block diagram of an apparatus including a real time computational fluid dynamics model coupled to a sensor in accordance with some embodiments.

FIG. 3B shows a block diagram of an apparatus including the apparatus shown in FIG. 3A and including a fuel pressure sensor in accordance with some embodiments.

FIG. 3C shows a block diagram of an apparatus including the apparatus shown in FIG. 3A and including a one-dimensional model in accordance with some embodiments.

FIG. 3D shows a block diagram of an apparatus including the apparatus shown in FIG. 3C and including a diesel fuel rail model in accordance with some embodiments.

FIG. 4A shows a block diagram of an apparatus including a diesel engine coupled to the engine control unit in accordance with some embodiments.

FIG. 4B shows a block diagram of an apparatus including the apparatus shown in FIG. 4A, an injector, and an integrator to process the virtual sensor signal to form the real time control signal for the injector in accordance with some embodiments.

FIG. 4C shows a block diagram of an apparatus including the apparatus shown in FIG. 4A and a field programmable gate array included in the virtual sensor model in accordance with some embodiments.

FIG. 4D shows a block diagram of an apparatus including the apparatus shown in FIG. 4A and a vehicle that includes the diesel engine and the engine control unit in accordance with some embodiments.

FIG. 5 shows a flow diagram of a method including receiving one fuel pressure signal at a virtual sensor model and processing the one fuel pressure signal to generate a fuel pressure signal for each of a plurality of injectors in real time in accordance with some embodiments.

FIG. 6A shows a block diagram of an apparatus including a real time computational fluid dynamics model, and a combustion model to communicate with the real time computational fluid dynamics model in accordance with some embodiments.

FIG. 6B shows a block diagram of the apparatus shown in FIG. 6A further including a one-dimensional computational fluid dynamics model, a reciprocating internal combustion engine model, a look-up table, and a field programmable gate array in accordance with some embodiments.

FIG. 7A shows a block diagram of an apparatus including an engine model, an inertial model, and a real time computational fluid dynamics air system model to communicate with the engine model in accordance with some embodiments.

FIG. 7B shows a block diagram of an apparatus including the apparatus shown in FIG. 7A and further including a reciprocating internal combustion engine model, an intake model, and a plurality of nodes included in the intake model in accordance with some embodiments.

FIG. 7C shows a block diagram of an apparatus including the apparatus shown in FIG. 7A coupled to an engine control unit in accordance with some embodiments.

FIG. 7D shows a block diagram of an apparatus including an engine model, a combustion model, and a real time computational fluid dynamics air system model to communicate with the engine model in accordance with some embodiments.

FIG. 7E shows a block diagram of an apparatus including the apparatus shown in FIG. 7D and further including a reciprocating internal combustion engine model, an exhaust model, and a plurality of nodes included in the exhaust model in accordance with some embodiments.

FIG. 7F shows a block diagram of an apparatus including the apparatus shown in FIG. 7D coupled to an engine control unit in accordance with some embodiments.

FIG. 8 shows a flow diagram of a method including running a real time one-dimensional computational fluid dynamics engine model in a field programmable gate array in accordance with some embodiments.

FIG. 9 shows a flow diagram of a method including generating field programmable gate array code automatically for a one-dimensional computational fluid dynamics engine model from code that is not real time code, and running the field programmable gate array code for the one-dimensional computational fluid dynamics engine model in the field programmable gate array in accordance with some embodiments.

10.3.4 Description

The following discussion sets forth numerous specific details to provide a thorough understanding of the disclosure. However, those of ordinary skill in the art, having the benefit of this disclosure, will appreciate that the subject matter of the disclosure may be practiced without these specific details. In addition, various well-known methods, procedures, components, software, and circuits have not been described in detail in order to focus attention on the features disclosed.

FIG. 1A shows a block diagram of an apparatus 100 including a boundary condition signal 102 for a one-dimensional computational fluid dynamics model 104 and a parallel computation unit 106 to receive the boundary condition signal 102 and provide an output signal 108 in accordance with some embodiments. The parallel computation unit 106 includes an input port 110 and an output port 112. The input port 110 receives the boundary condition signal 102. The output port 112 provides the output signal 108.

The boundary condition signal 102 is not limited to a signal representing a particular physical variable. One-dimensional computational fluid dynamics models can be formed to process boundary condition signals for any variable of interest in the system being modeled or any variable that can be derived from the variables included in the system being modeled. One-dimensional computational fluid dynamics models can describe a network of pipes. A network includes any piping intersection configuration. One-dimensional computational fluid dynamics models include linked models, for example, a mechanical check-valve model. Exemplary boundary condition signals processed in computational fluid dynamics engine models include pressure signals, temperature signals, air quality or composition signals, and air/fuel ratio signals. In some embodiments, models convert real world boundary events to boundary conditions. Air quality includes the chemical species and thermodynamic properties included in the air or similar gas. For example, a pressure pulse is generated when an injector is opened.

The one-dimensional computational fluid dynamics model 104 is formed to include in the model the physical variable represented by the boundary condition signal 102 and allow prediction of the value of the variable at a location in the model that is not sampled in the physical system. For example, for a diesel engine fuel rail that includes one pressure sensor to generate a boundary condition pressure signal, the one-dimensional computational fluid dynamics model 104 can process the boundary condition signal 102 for pressure and predict the pressure at any point along the rail in real time.

The computational fluid dynamics model 104 is not limited to using a particular solution method. Exemplary solution methods include finite element, finite volume, finite difference, and spectral methods. In the finite element method, each node is weighted before integration to guarantee continuity. In the finite volume method, the conservation equations are included in integral form and are discretized to a set of

algebraic equations that are then solved. In the finite difference method, at each grid point the differential conservation equation is approximated by replacing the partial derivatives by approximations in terms of the nodal values of the functions. In the spectral method, the differential equations are solved using Fourier methods. In some embodiments, a solution method is selected in which the speed of the solution is determined by the Courant number. The Courant number is a parameter used in the stability analysis of finite difference equations such as algebraic equations used to approximate partial differential equations in the computational fluid dynamics model 104.

The parallel computation unit 106 receives the boundary condition signal 102 at the input port 110 and provides the output signal 108 at the output port 112. The output signal 108 is a virtual signal generated through the processing of the boundary condition signal 102 by the parallel processing unit. The output signal 108 is a virtual signal because it is obtained through computation performed in the parallel computation unit 106 rather than through a measurement obtained from the physical system. For example, if the boundary condition signal 102 represents the actual pressure at a first point along a diesel fuel rail, then the output signal 108 is a virtual signal that represents the pressure at an second point along the diesel engine fuel rail that is not measured or sampled.

In operation, the boundary condition signal 102 is sampled or received by the parallel computation unit 106. At a time t_1 , the parallel computation unit 106 samples or receives at the input port 110 the boundary condition signal 102. The parallel computation unit 106 provides the output signal 108 at the output port 112 before the boundary condition signal 102 is sampled at a time t_2 . The time t_2 occurs after the time t_1 .

In some embodiments, the difference between the time t_2 and the time t_1 is between about ten microseconds and about ten milliseconds. A difference between

the time t_2 and the time t_1 of more than about ten milliseconds is too long for real time control of physical systems such as reciprocating engines. A difference between time t_2 and t_1 of less than about ten microseconds is too short to provide sufficient processing time for the parallel processing unit 106 to generate the output signal 108 in real time for physical systems such as reciprocating engines. In some embodiments, the difference between time t_2 and time t_1 is slightly less than the Courant number. The Courant number is substantially equal to the speed of sound divided by a volume element in a computational fluid dynamics model.

The apparatus 100 is not limited to use in connection with a particular system or industry. The apparatus 100 can be applied to systems that include intake air flow dynamics, exhaust flow dynamics, exhaust recirculation flow, hydraulic modeling in anti-lock braking systems and steer-by-wire systems, pump/motor control in hydraulic hybrid vehicle systems, biomedical flow systems, petrochemical systems, and heat transfer systems.

FIG. 1B shows a timing diagram 114 illustrating the relationship between the sampling times t_1 and t_2 of the boundary condition signal 102 shown in FIG. 1A and the output signal 108 also shown in FIG. 1A in accordance with some embodiments. As shown in FIG. 1B, the boundary condition signal 102 when sampled or received at the time t_1 has a value of y_1 and when sampled or received at the time t_2 has a value of y_2 . The output signal 108 is provided by the parallel processing unit 106, shown in FIG. 1A, at the output port 112, shown in FIG. 1A, at a time t_3 and has a value y_3 . The time t_3 occurs after the time t_1 and before the time t_2 .

FIG. 1C shows a block diagram of an apparatus 116 including the apparatus 100 shown in FIG. 1A included in a field programmable gate array 118 in accordance with some embodiments. A field programmable gate array (FPGA) is an electronic device that includes programmable logic units and programmable interconnects. The programmable logic units can be programmed to provide logic

functions, complex combinational functions, and memory functions. Exemplary logic functions provided by FPGAs include AND, OR, XOR, and NOT. Exemplary complex combinational functions provided by FPGAs include decoders and mathematical functions including mathematical functions suitable for use in forming the one-dimensional computational fluid dynamics model 104. Exemplary memory functions include primary and complementary storage as provided by flip flops flip-flops and dynamic random access memory circuits. The programmable interconnects can be programmed in a manufacturing environment or in the field after delivery of the product to a customer. Methods of programming interconnects in FPGAs include electrical methods and optical methods. Field programmable gate arrays can be converted to application specific integrated circuits in which the programmability provided by the field programmable gate array has been reduced. Thus, application specific integrated circuits can be substituted for field programmable gate arrays.

FIG. 1D shows a block diagram of an apparatus 120 including the apparatus 100 shown in FIG. 1A included in an engine control unit 122 in accordance with some embodiments. The engine control unit 122 is configured to send and receive signals to an engine. The engine control unit 122 is not limited to a control unit for controlling a particular type of engine. Exemplary engines suitable for control by the control unit 122 include diesel engines, gasoline engines, alternative fuel engines, and hybrid engines powered by fossil fuels and renewable fuels. Exemplary alternative fuels include natural gas and biofuels, such as methanol, ethanol, and hydrogen.

The engine control unit 122 is not limited to being formed from a particular type of electronic component. Discrete circuits and integrated circuits, including processors, such as complex instruction set processors and reduced instruction set processors, application specific integrated circuits, and software are components and technologies suitable for use in forming the engine control unit 122.

The engine control unit 122 is not limited to being formed using a particular packaging technology. Exemplary packaging technologies suitable for use in connection with the fabrication of the engine control unit 122 include multi-carrier modules, card or board packages, and encapsulated or hermetically sealed packages. Combinations of packaging technologies can also be used in forming the engine control unit 122. The engine control unit 122 is not limited to a single unitary package. In some embodiments, the engine control unit 122 is a distributed engine control system distributed among a plurality of packages.

FIG. 1E shows a block diagram of an apparatus 124 including the apparatus 100 shown in FIG. 1A and including a reciprocating engine model 126 included in the one-dimensional computational fluid dynamics model 104 in accordance with some embodiments. The reciprocating engine model 126 is a model of an engine whose crankshaft is turned by pistons moving up and down in a cylinder.

FIG. 2 shows a flow diagram of a method 200 including receiving a sensor signal (block 202) and processing the sensor signal using real time computational fluid dynamics methods to generate a virtual sensor signal in real time (block 204). The sensor signal is provided by a sensor, such as a temperature or pressure sensor, in real time. In some embodiments, the method 200 further includes processing the virtual sensor signal to generate an injector control signal. An injector control signal can control an injector, such as an injector included in a diesel engine. An injector delivers a controlled amount of material, such as diesel fuel, to a process chamber. In some embodiments, processing the sensor signal using computational fluid dynamics methods to generate the virtual sensor signal in real time includes interpolation. Interpolation is the estimation of a numerical value between two given numerical values. The interpolation is not limited to a particular method. Exemplary methods of interpolation include linear interpolation, polynomial interpolation, and spline interpolation.

FIG. 3A shows a block diagram of an apparatus 300 including a real time computational fluid dynamics model 302 coupled to a sensor 304 in accordance with some embodiments. The real time computational fluid dynamics model 302 includes an input port 306 to receive the sensor signal 308 and an output port 310 to provide a virtual sensor signal 312 in real time. The real time computational fluid dynamics model 302 includes a plurality of parallel computation units 314 to generate and provide the virtual sensor signal 312. In some embodiments, each of the plurality of parallel computation units 314 includes a logic unit, a memory unit, a math unit, and interconnects. The plurality of parallel computation units 314 are coupled in series at the interconnects. The virtual sensor signal 312 is generated in real time by the computational fluid dynamics model 302.

The sensor 304 provides a real time sensor signal to the real time computational fluid dynamics model 302. The sensor 304 is not limited to a sensor for sensing a particular physical variable. Exemplary sensors suitable for use in connection with the apparatus 300 include pressure, temperature, and chemical sensors. In some embodiments, the sensor 304 is sampled at a rate of between about 100 Hz and about 100 kHz. Sampling at a rate of less than about 100 Hz is too slow to control high performance systems in real time. Sampling at a rate of more than about 100 kHz does not permit processing a virtual sensor model in real time. Sampling includes sampling performed at the real time computational fluid dynamics model or sampling and conversion of an analog sensor signal to a digital signal outside the computational fluid dynamics model.

In some embodiments, the ratio of the plurality of parallel computation units 314 to nodes is low. A node is a computation point in the real time computational fluid dynamics model 302. A low ratio is a ratio close to about one. A low ratio of the plurality of parallel computation units 314 to nodes enables real time calculation of the virtual sensor signal 312. Each of the plurality of parallel computation units

314 computes a new value for the variable of interest for one node in the real time computational fluid dynamics model 302. This method of computation permits generation of the virtual sensor signal 312 in real time.

FIG. 3B shows a block diagram of an apparatus 316 including the apparatus 300 of FIG. 3A and a fuel pressure sensor 318 in accordance with some embodiments. The fuel pressure sensor 318 generates a fuel pressure signal that can be virtualized to provide a fuel pressure signal for any point in the combustion chamber. Virtualization includes providing a fuel pressure value for a location not monitored by a fuel pressure sensor.

FIG. 3C shows a block diagram of an apparatus 320 including the apparatus 316 shown in FIG. 3B and including a one-dimensional model 322 in accordance with some embodiments. The one-dimensional model 322 can be modeled as a pipe. The properties of a fluid within the pipe vary only along the direction of the pipe. Real time computational fluid dynamic calculations can be performed on the one-dimensional model in real time. For the model, the space in the pipe is divided into many small volumes. The volumes have a known geometry. The fluid, such as a fuel, contained in these volumes has specific properties, such as compressibility, density, and viscosity, for example. Entry and exit conditions, for a fuel rail pipe model, are defined by the engine speed, injection events, and a pressure-regulating valve. Each of these events can be modeled.

FIG. 3D shows a block diagram of an apparatus 324 including the apparatus 320 shown in FIG. 3C and including a diesel fuel rail model 326 in accordance with some embodiments. In operation, the diesel fuel rail is maintained at high pressure during operation of the engine. Modeling the pressure at each of the injectors along the diesel fuel rail permits precise delivery of fuel which results in improved performance and reduced hydrocarbon emissions.

FIG. 4A shows a block diagram of an apparatus 400 including a diesel engine 402 coupled to an engine control unit 404. The diesel engine 402 is an internal-combustion engine that receives a spray of fuel after the start of the compression stroke and ignites the spray of fuel through the use of the heat of compressed air. The diesel engine 402 includes a sensor 406 to provide a sensor signal to the engine control unit 404. In some embodiments, the sensor 406 includes a pressure sensor. The engine control unit 404 is coupled to the diesel engine 402 and includes a virtual sensor model 408 to receive the sensor signal from the sensor 406. The engine control unit 404 performs a real time computational fluid dynamics calculation to generate a virtual sensor signal 410 for use in forming a real time engine control signal 412 to control the diesel engine 402.

FIG. 4B shows a block diagram of an apparatus 420 including the apparatus 400 shown in FIG. 4A, an injector 422, and an integrator 424 to process the real time control signal 412 or the virtual sensor signal 410 for the injector 422. The injector 422 is a device for metering fuel to a combustion chamber in an engine. The integrator 424 is a device or algorithm that applies the mathematical operation of integration to a signal. For example, in some embodiments, the integrator 424 integrates the virtual sensor signal 410 or the real time control signal 412.

In some embodiments, the pulse width of a control signal delivered to the injector 422 by the integrator 424 is controlled by integrating the instantaneous fuel delivered until it reaches the desired quantity. An exemplary real time engine control signal 412 includes a signal that represents the instantaneous fuel delivered to the injector 422. The instantaneous fuel delivered is a function of the instantaneous pressure at the injector. Delivered fuel is added each time step until a quantity of fuel is reached. Extrapolation techniques are used to predict the exact shutoff time at a temporal resolution greater than the rate at which the real-time computational fluid dynamics model runs.

FIG. 4C shows a block diagram of an apparatus 430 including the apparatus 400 shown in FIG. 4A and a field programmable gate array 432 included in the virtual sensor model 408. The field programmable gate array 432 is a electronic device that includes programmable logic units and programmable interconnects.

FIG. 4D shows a block diagram of an apparatus 440 including the apparatus 400 shown in FIG. 4A and a vehicle 442 that includes the diesel engine 402 and the engine control unit 404. The apparatus 440 is not limited to a particular type of vehicle. Exemplary vehicles suitable for use in connection with the apparatus 440 include trucks, cars, trains, planes, and ships.

FIG. 5 shows a flow diagram of a method 500 including receiving one fuel pressure signal at a virtual sensor model (block 502) and processing the one fuel pressure signal to generate a fuel pressure signal for each of a plurality of injectors in real time (block 504). In some embodiments, receiving the one fuel pressure signal at the virtual sensor model includes sampling the fuel pressure signal substantially periodically with respect to engine angle. In some embodiments, the method 500 further includes integrating each of the fuel pressure signals. In some embodiments, the method 500 further includes processing the fuel pressure signal for each of the plurality of injectors to generate an injector pulse width for controlling each of the plurality of injectors. In some embodiments, processing the fuel pressure signal for each of the plurality of injectors to generate an injector pulse width for controlling each of the plurality of injectors, as shown in FIG. 5, includes applying computational fluid dynamics methods in processing the fuel pressure signal.

FIG. 6A shows a block diagram of an apparatus 600 including a real time computational fluid dynamics model 602 and a combustion model 604 to communicate with the real time computational fluid dynamics model 602 in accordance with some embodiments. The real time computational fluid dynamics model 602 includes an input port 606 to receive a sensor signal 608 and an output port 610 to provide

a virtual sensor signal 612. The combustion model 604 communicates combustion information to the real time computational fluid dynamics model 602.

A communication channel 614 provides for communication between the real time computational fluid dynamics model 602 and the combustion model 604. The communication channel 614 includes any method, device, or system for exchanging information. In some embodiments, the information communicated between the real time computational fluid dynamics model 602 and the combustion model 604 is digital information. In some embodiments, the information communicated between the real time computational fluid dynamics model 602 and the combustion model 604 is analog information. The information may be coded or uncoded. Coded information can include fewer bits than the starting information or more bits than the starting information. In a software system, the communication channel 614 includes a variable or a location in a memory shared between the real time computational fluid dynamics model 602 and the combustion model 604.

The real time computational fluid dynamics model 602 predicts the fluid flow and physical properties of the system being modeled. An operating diesel fuel rail in a diesel engine is an exemplary system for modeling in the apparatus 600. In some embodiments, the real time computational fluid dynamics model 602 operates at a frequency of between about 200 hertz and about 1000 kilohertz. Frequencies of between about 200 hertz and about 1000 kilohertz are suitable for modeling a diesel engine fuel rail. In some embodiments, the real time computational fluid dynamics model 602 operates at a frequency slightly greater than required by the Courant number. The Courant number is substantially equal to the speed of sound divided by a volume element in the computational fluid dynamics model.

The real time computational fluid dynamics model 602, in some embodiments, receives information related to the state of the system being modeled. Exemplary information received, for example by a computational fluid dynamics model for a

an engine powered by combustion, includes engine speed, engine load, turbo speed, air/fuel ratio, manifold pressure, and manifold temperature, and exhaust state.

The combustion model 604 simulates chemical reactions in which substances combine with oxygen and release heat energy. In some embodiments, the combustion model 604 includes a model of burning a fuel, such as diesel fuel, in the presence of oxygen to produce heat. The chemical reactions in a combustion process are rapid. Thus, a system to simulate a combustion reaction in real time, includes computing elements and software capable of calculating the necessary physical variables in real time.

In operation, the real time computational fluid dynamics model 602 receives the sensor signal 608 at the input port 606. The combustion model 604 communicates information relating to the combustion process to the real time computational fluid dynamics model 602. The real time computational fluid dynamics model processes the sensor signal 606, such as a pressure signal generated from a pressure sensor in a fuel rail of a diesel engine, and information provided by the combustion model 604 to generate the virtual sensor signal 612 at the output port 610. The virtual sensor signal 612 includes, for example, the pressure value in a diesel fuel rail at a location not monitored by a sensor. The apparatus 600 provides a virtual sensor signal 612, such as a pressure signal, that can be provided to the system being modeled in real time to improve the performance. Performance is improved by reducing undesired gas emissions or using less fuel to produce the same power.

FIG. 6B shows a block diagram of an apparatus 615 including the apparatus 600 shown in FIG. 6A and further including a one-dimensional computational fluid dynamics model programmable gate array 622 in accordance with some embodiments. The apparatus 600 included in the apparatus 615 includes the sensor signal 608, the real time computational fluid dynamics model 602 including the input port

606 and the output port 610, the virtual sensor signal 612, and the combustion model 604.

The one-dimensional computational fluid dynamics model 616, in some embodiments, is included in the real time computational fluid dynamics model 602. The one-dimensional computational fluid dynamics model 616 enables calculation of physical variables in real time. One example of the one-dimensional fluid dynamics model 616 is a pipe. A pipe including a series of computational nodes located along the length of the pipe is one model suitable for modeling a fluid, including liquid and gas fluids, in some engine configurations.

The reciprocating internal combustion engine model 618, in some embodiments, is included in the real time computational fluid dynamics model. The reciprocating internal combustion engine model 618 includes a reciprocating engine model and an internal combustion engine model. A reciprocating engine converts pressure to rotating motion using one or more pistons. A piston is a sliding element that fits within the bore of a cylinder. In an internal combustion engine gases expand to create pressure that causes movement of the piston in the bore of the cylinder. The exothermic reaction of a fuel with an oxidizer causes expansion of the gases in a combustion chamber.

The look-up table 620, in some embodiments, is included in the combustion model 604. The look-up table 620 includes information related the combustion process. For example, in some embodiments, the look-up table 620 includes temperature and pressure at a location in a combustion chamber at discrete points in time during the combustion process. Look-up tables can provide information at a rate that enables real time operation.

The field programmable gate array 622, in some embodiments, is included in the real time computational fluid dynamics model 602. The field programmable gate array 622 includes computation units or nodes including software to calculate

the value of physical variables at nodes in the one-dimensional computational fluid dynamics model 616.

In operation, the real time computational fluid dynamics model 602 receives the sensor signal 608 at the input port 606. The combustion model 604 including the look-up table 620 communicates combustion information to the real time computational fluid dynamics model 602 over the communication channel 614. The real time computational fluid dynamics model 602 including the one-dimensional computational fluid dynamics model 616 and the reciprocating internal combustion engine model 618 running in the field programmable gate array 622 provide the virtual sensor signal 612, such as pressure signal, at the output port 610.

FIG. 7A shows a block diagram of an apparatus 700 including an engine model 702, an inertial model 704, and a real time computational fluid dynamics air system model 706 to communicate with the engine model 702 in accordance with some embodiments. The engine model 702 includes an input port 708 to receive an input signal 710 and an output port 712 to provide an output signal 714.

The real time computational fluid dynamics model 706 communicates with the engine model 702 over a communication channel 716. The communication channel 716 is not limited to a particular type of communication channel. Any system, medium, or method capable of transmitting information between the engine model 702 and the real time computational fluid dynamics air system model 706 is suitable for use in connection with the apparatus 700. In some embodiments, a variable in a software program or a memory location in a computer system is the communication channel 716.

The input signal 710 received at the input port 708 of the engine model 702 includes one or more engine control signals. Exemplary engine control signals include actuator control signals, such as throttle control signals, injector control signals, and spark control signals.

The engine model 702 is not limited to a model of a particular type of engine. Exemplary engines suitable for modeling and use in the apparatus 700 include diesel engines and non-diesel engines. A gasoline engine is an exemplary non-diesel engine suitable for modeling in the apparatus 700. The engine model 702 is suitable for use in connection with a hardware-in-the-loop system. A hardware-in-the-loop system provides a system and method for testing an engine control unit without an actual engine.

The inertial model 704 included in the engine model 702 provides information related to the dynamic operation of the engine being modeled. For example, in some embodiments, the inertial model includes a torque model that provides information related to the amount of force required to rotate the crankshaft of an engine. In some embodiments, the inertial model 704 includes an engine speed model that provides information related to the rotation rate of the engine.

The real time computational fluid dynamics air system model 706 includes a computational fluid dynamics model of the air system included in the engine model 702. In some embodiments, the real time computational fluid dynamics air system model 706 is a one-dimensional model. Exemplary elements that may be included in the real time computational fluid dynamics air system model 706 include an intake model and an exhaust model for the engine being modeled. In some embodiments, the real time computational fluid dynamics air system model 706 includes a catalytic converter model and a turbocharger model. In some embodiments, a virtual sensor signal is generated for information that cannot be obtained using a sensor. For example, the ratio of exhaust gas recirculation/mass of fresh air in a diesel engine can be provided through the real time computational fluid dynamics air system model 706. Other virtual quantities that can be provided include the pressure difference across a turbo and charge air quality.

In operation, the engine model 702 receives the input signal 710 at the input port 708. The input signal 710 includes one or more engine control signals, such as engine actuator signals. The real time computational fluid dynamics air system model 706 performs a real time computation for variables included in the engine air system and communicates the results to the engine model 702. After processing the information received from the real time computational fluid dynamics air system model 706 and the engine control signals, provided at the input port 708, the engine model 702 provides the output signal 714, including one or more engine signals such as engine speed, at the output port 712.

FIG. 7B shows a block diagram of an apparatus 720 including the apparatus 700, show in FIG. 7A, and further including a reciprocating internal combustion engine model 618 included in the engine model 702, an intake model 724 included in the real time computational fluid dynamics model 706, and a plurality of nodes 726 included in the intake model 724 in accordance with some embodiments. The reciprocating internal combustion engine model 618 includes the reciprocating engine model and the internal combustion engine model described above. The intake model 724 includes the plurality of nodes 726 to model the intake system of the engine being modeled. Each node in the plurality of nodes 726 includes a computation unit that includes software and hardware to compute a computational fluid dynamics variable at a node location in the intake model 724.

In operation, the engine model 702 receives the input signal 710 at the input port 708 and a communication from the real time computational fluid dynamics air system model 706 via the communication channel 716. The communication includes information, such as virtual sensor signals, relating to the output of the intake model 724 generated by the plurality of nodes 726. The engine model 702 processes information from the inertial model 704, the reciprocating internal combustion engine

model 618, and the received information to generate the output signal 714 at the output port 712.

FIG. 7C shows a block diagram of an apparatus 730 including the apparatus 700, shown in FIG. 7A, coupled to an engine control unit 732 in accordance with some embodiments. The engine control unit 732 includes an output port to provide the input signal 710 to the engine model 702. The engine control unit 732 includes an input port 736 to receive the output signal 714 from the engine model 702. The apparatus 730 is sometimes referred to as a hardware-in-the-loop system and enables testing of the engine control unit 732. In operation, the engine model 702 receives, at the input port 708, the input signal 710, such as an actuator signal, from the engine control unit 732, receives via the communication channel 716 information, such a pressure or temperature information related to the air system, from the real time computational fluid dynamics air system model 706, and provides the output signal 714, such as a virtual pressure signal, to the engine control unit 732, at the output port 712.

FIG. 7D shows a block diagram of an apparatus 740 including an engine model 702, a combustion model 604 included in the engine model 702, and a real time computational fluid dynamics air system model 706 to communicate with the engine model 702 in accordance with some embodiments. The apparatus 740 includes all the elements of the apparatus 700, shown in FIG. 7A and described above, except the inertial model 740. Further, the apparatus 740 includes the combustion model 604 not explicitly included in the apparatus 700 shown in FIG. 7A.

The combustion model 604 simulates chemical reactions in which substances combine with oxygen and release heat energy. In some embodiments, the combustion model 604 includes a model of burning a fuel, such as diesel fuel, in the presence of oxygen to produce heat. The chemical reactions in a combustion process are rapid.

Thus, a system to simulate a combustion reaction in real time, includes computing elements and software capable of calculating the physical variables in real time.

In operation, the engine model 702 receives the input signal 710 at the input port 708. The input signal 710 includes one or more engine control signals, such as engine actuator signals. The real time computational fluid dynamics air system model 706 performs a real time computation for variables included in the engine air system and communicates the results to the engine model 702 via the communication channel 716. After processing the information received from the real time computational fluid dynamics air system model 706 and the input signal 710, the engine model 702 provides the output signal 714, including one or more engine signals such as engine speed, at the output port 712.

FIG. 7E shows a block diagram of an apparatus 750 including the apparatus 740 show in FIG. 7D and further including a reciprocating internal combustion engine model 618 included in the engine model 702, an exhaust model 754 included in the real time computational fluid dynamics model 706, and a plurality of nodes 726 included in the exhaust model 754 in accordance with some embodiments. The reciprocating internal combustion engine model 618 is described above and includes the reciprocating engine model and the internal combustion engine model described above. The exhaust model 754 includes the plurality of nodes 726 to model the intake system of the engine being modeled. Each node in the plurality of nodes 726 includes a computation unit that includes software and hardware to compute a computational fluid dynamics variable at a node location in the exhaust model 754.

In operation, the engine model 702 receives the input signal 710 at the input port 708, and information generated by the exhaust model 754 via the plurality of nodes 726 from the real time computational fluid dynamics air system model 706 via the communication channel 716. The engine model 702 process the combustion model 604 information, the reciprocating internal combustion engine model 618

information, and the received information to generate the output signal 714 at the output port 712.

FIG. 7F shows a block diagram of an apparatus 760 including the apparatus 740 shown in FIG. 7D, coupled to an engine control unit 762 in accordance with some embodiments. The engine control unit 762 includes an output port 764 and an input port 766. The output port 764 of the engine control unit is coupled to the input port 708 of the engine model 702. The output port 712 of the engine model 702 is coupled to the input port 766 of the engine control unit. The apparatus 760 is sometimes referred to as a hardware-in-the-loop system and enables testing of the engine control unit 732 without an actual engine.

In operation, the engine control unit 762 provides the input signal 710, such as an actuator signal, to the engine model 702. The engine control unit 762 provides the input signal 710 at the output port 764. The engine model 702 receives the input signal 710 at the input port 708. The engine control unit 762 receives the output signal 714 from the engine model 702. The engine model 702 provides the output signal 714 at the output port 712. The engine control unit 762 receives the output signal 714 at the input port 766.

The engine model 702 receives the input signal 710 at the input port 708. The input signal 710 includes one or more engine control signals, such as engine actuator signals. The real time computational fluid dynamics air system model 706 performs a real time computation for variables included in the engine air system and communicates the results to the engine model 702 via the communication channel 716. After processing the information received from the real time computational fluid dynamics air system model 706, the combustion model 604, and the input signal 710, the engine model 702 provides the output signal 714, including one or more engine signals such as engine speed, at the output port 712.

FIG. 8 shows a flow diagram of a method 800 including running a real time one-dimensional computational fluid dynamics engine model in a field programmable gate array (block 802). In some embodiments, the field programmable gate array is replaced by an application specific integrated circuit. Generally, an application specific integrated circuit replaces the field programmable gate array when production quantities of the model are required, such as when the model is included in a production vehicle, such as a passenger car or industrial truck. A field programmable gate array is converted to an application specific integrated circuit by removing some of the programmable features of the field programmable gate array. The method 800 is useful in systems that simulate the engine being modeled. A simulation system that includes the method 800 can be configured to provide simulated actual sensor signals and virtual sensor signals. Such a simulation system is useful for applications such as testing an engine control unit when the engine is unavailable.

In some embodiments, the method 800, further includes configuring a hardware-in-the-loop test system including an engine control unit coupled to the real time one-dimensional computational fluid dynamics engine model in the field programmable gate array. In operation, an engine control unit provides engine control signals to an engine, such as a diesel engine. A hardware-in-the-loop test system enables testing an engine control unit when an actual engine is unavailable for testing, such as in the early design phases of a new engine or when the cost of providing an actual engine is high. The engine is replaced by the real time one-dimensional computational fluid dynamics engine model in the field programmable gate array and perhaps other models, such as combustion and inertia/torque models.

In some embodiments, the method 800 further includes testing the engine control unit by sending signals to the real time one-dimensional computational fluid dynamics engine model running in the field programmable gate array and receiv-

ing signals from the real time one-dimensional computational fluid dynamics engine model running in the field programmable gate array.

FIG. 9 shows a flow diagram of a method 900 including generating field programmable gate array code automatically for a one-dimensional computational fluid dynamics engine model from code that is not real time code (block 902), and running the field programmable gate array code for the one-dimensional computational fluid dynamics engine model in the field programmable gate array (block 904). Code can be generated automatically by converting a non-real time simulation model into real time simulation model that can run on a field programmable gate array.

In some embodiments, the method 900 further includes testing an engine control unit by sending signals to the real time one-dimensional computational fluid dynamics engine model running in the field programmable gate array and receiving signals from the real time one-dimensional computational fluid dynamics engine model running in the field programmable gate array. Exemplary signals provided by the engine control unit include throttle command, turbo boost command, spark commands, and injector commands. These are processed by the model running on the field programmable gate array to produce sensor values, such as throttle position, manifold pressure, manifold temperature, engine speed, and coolant temperature, that are provided to the engine control unit.

The disclosed embodiments have been provided to illustrate various features of the disclosure. Persons skilled in the art of computational fluid dynamics, having the benefit of this disclosure, will recognize variations and modifications of the disclosed embodiments, which none the less fall within the spirit and scope of the appended claims.

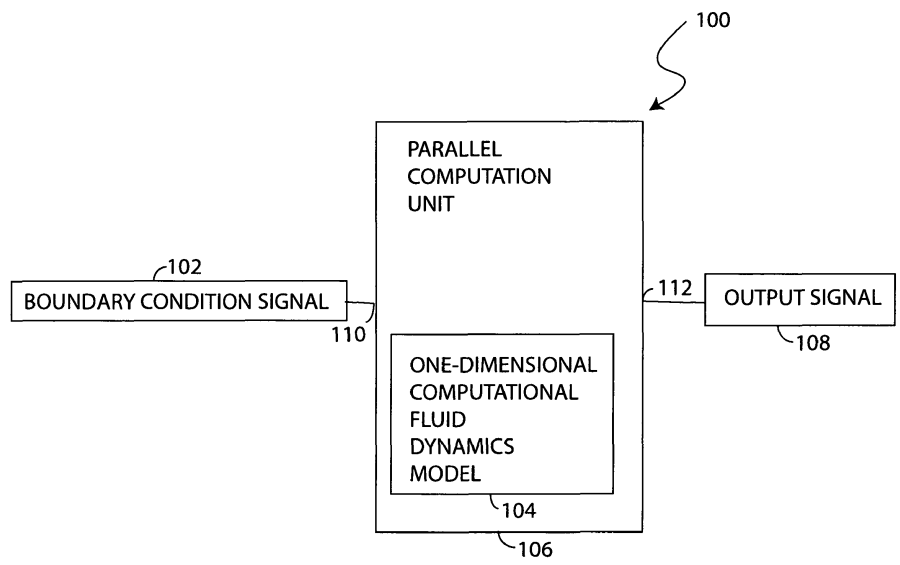


FIG. 1A

Figure 10.1: Patent Fig. 1A

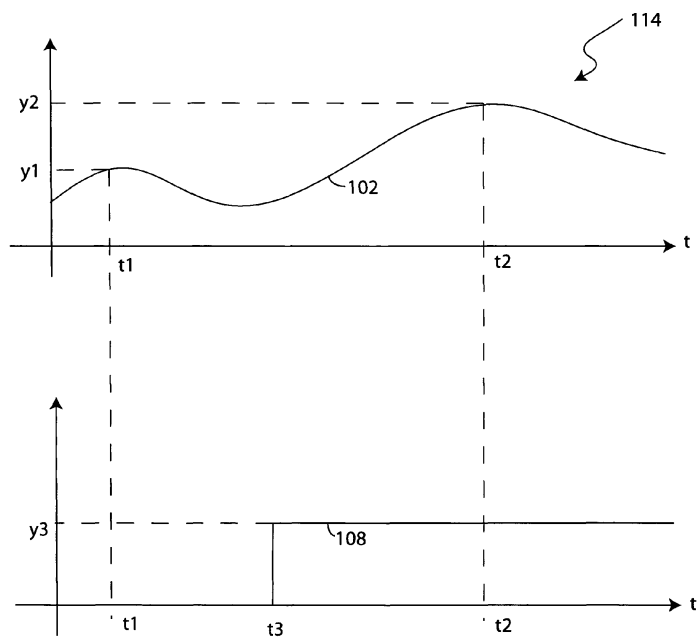


FIG. 1B

Figure 10.2: Patent Fig. 1B

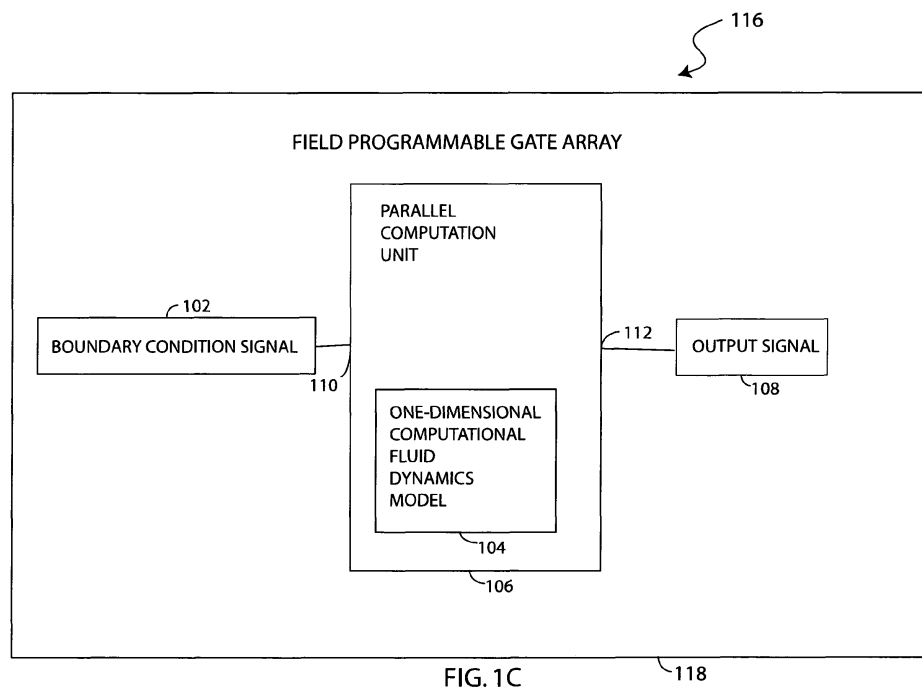


FIG. 1C

Figure 10.3: Patent Fig. 1C

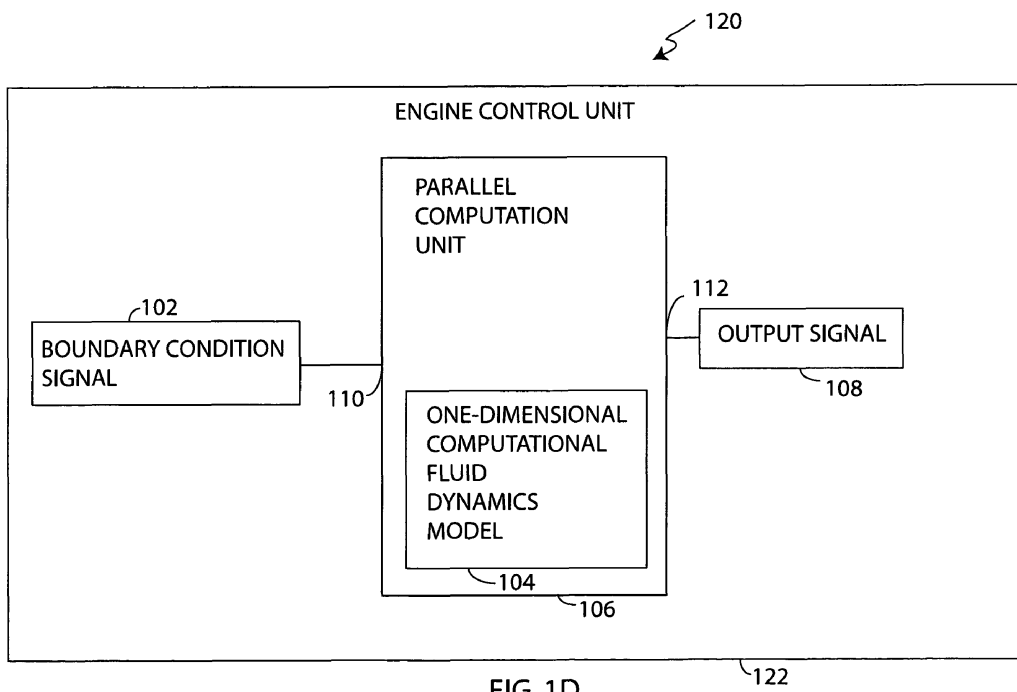


FIG. 1D

Figure 10.4: Patent Fig. 1D

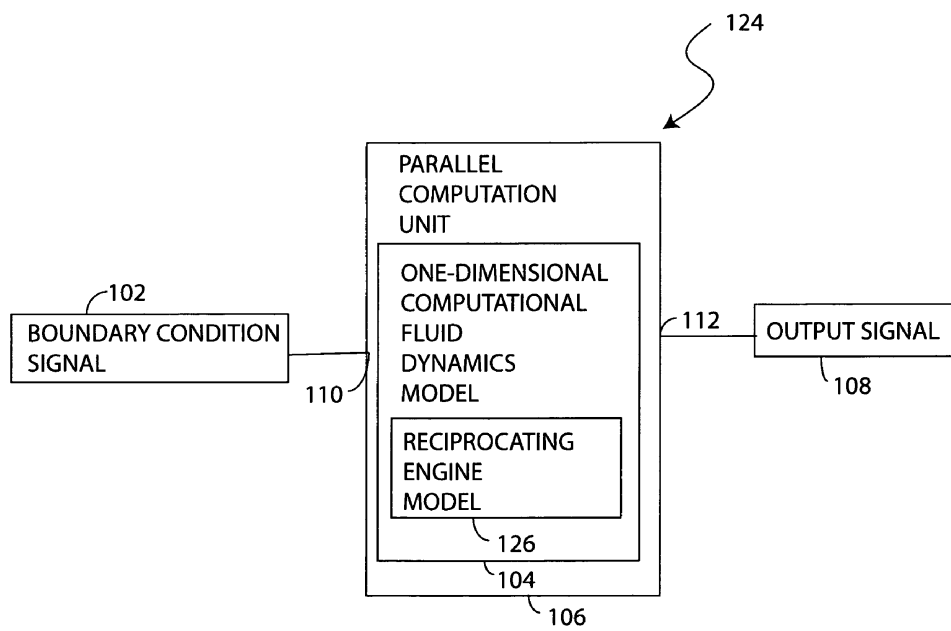


FIG. 1E

Figure 10.5: Patent Fig. 1E

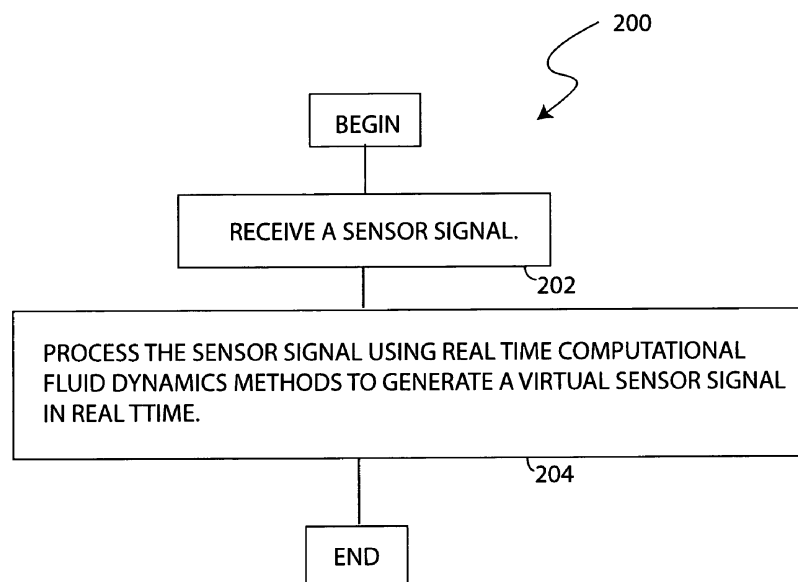


FIG. 2

Figure 10.6: Patent Fig. 2

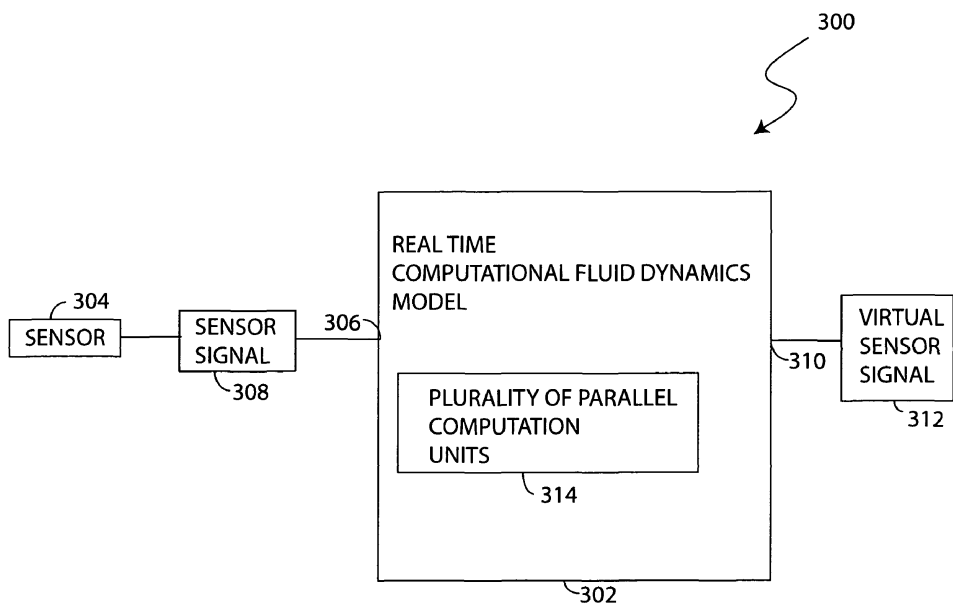


FIG. 3A

Figure 10.7: Patent Fig. 3A

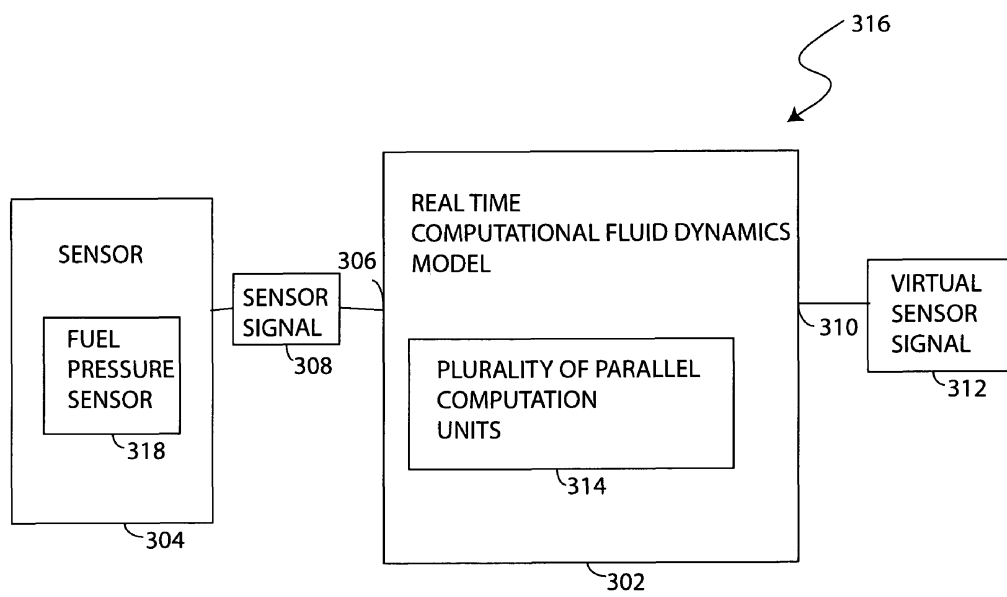


FIG. 3B

Figure 10.8: Patent Fig. 3B

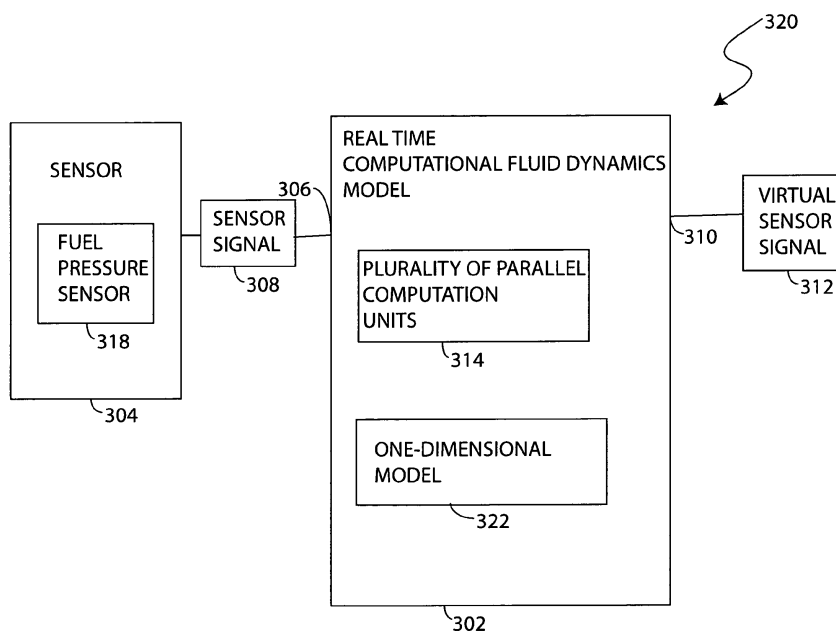


FIG. 3C

Figure 10.9: Patent Fig. 3C

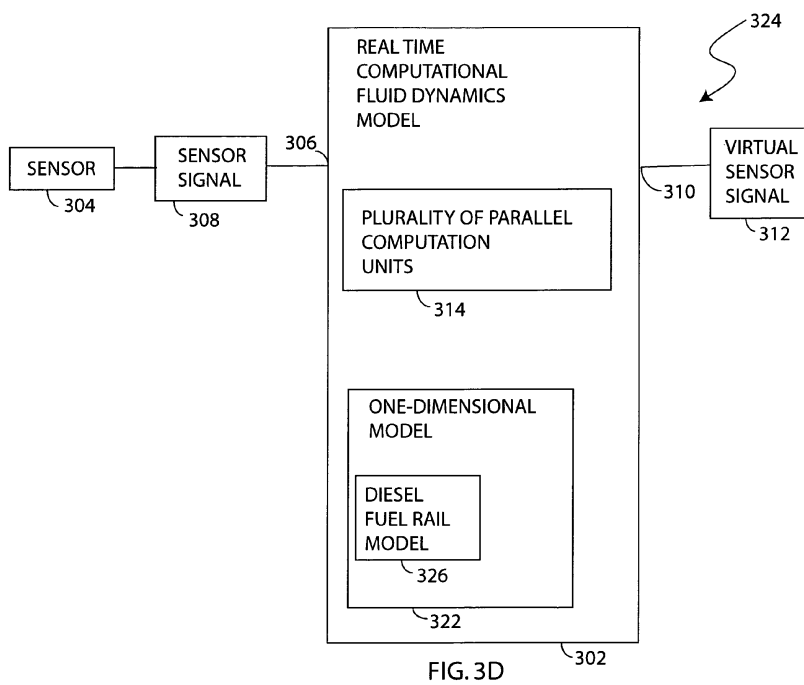


Figure 10.10: Patent Fig. 3D

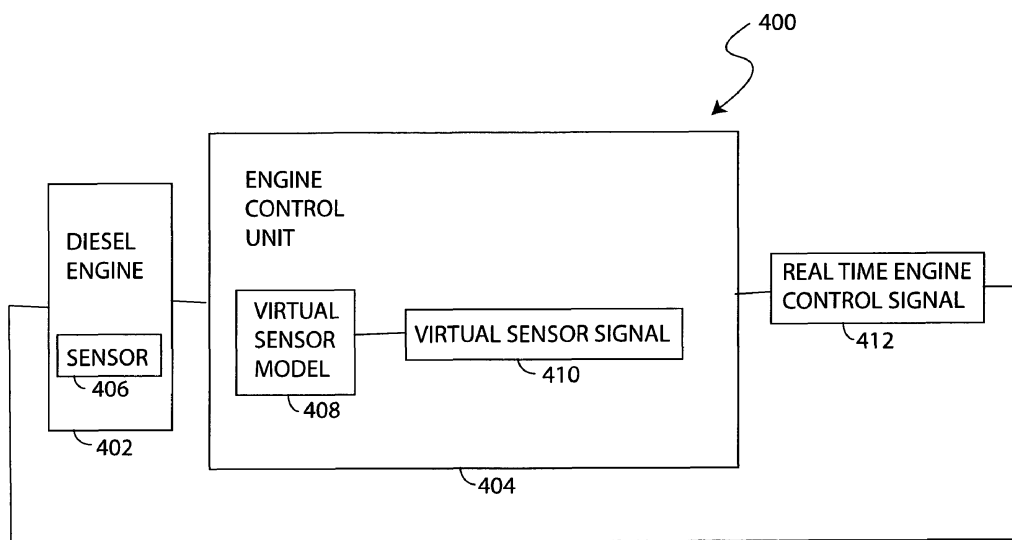


FIG. 4A

Figure 10.11: Patent Fig. 4

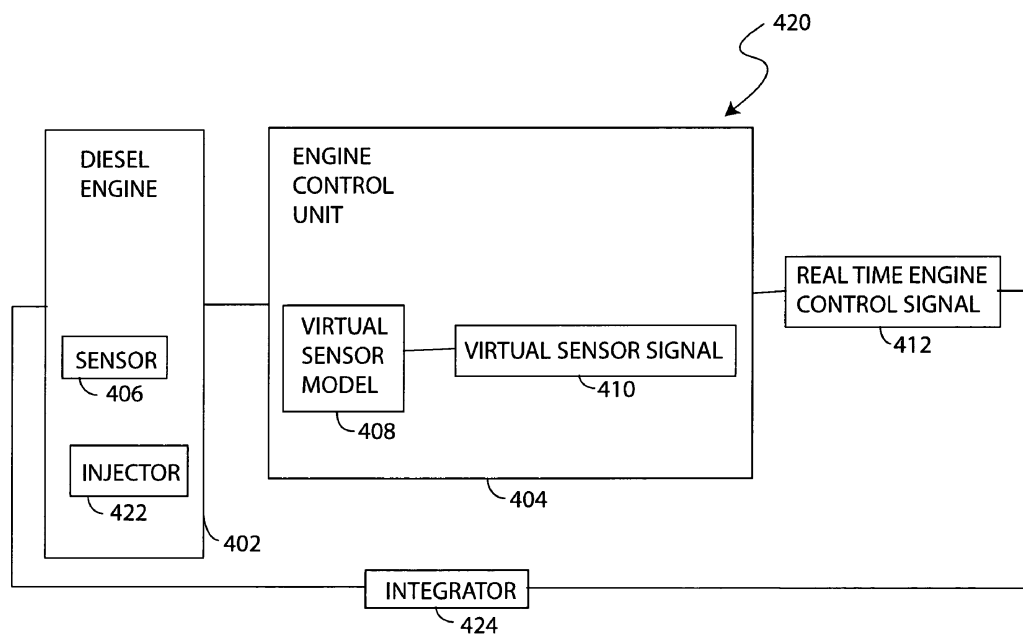


FIG. 4B

Figure 10.12: Patent Fig. 4B

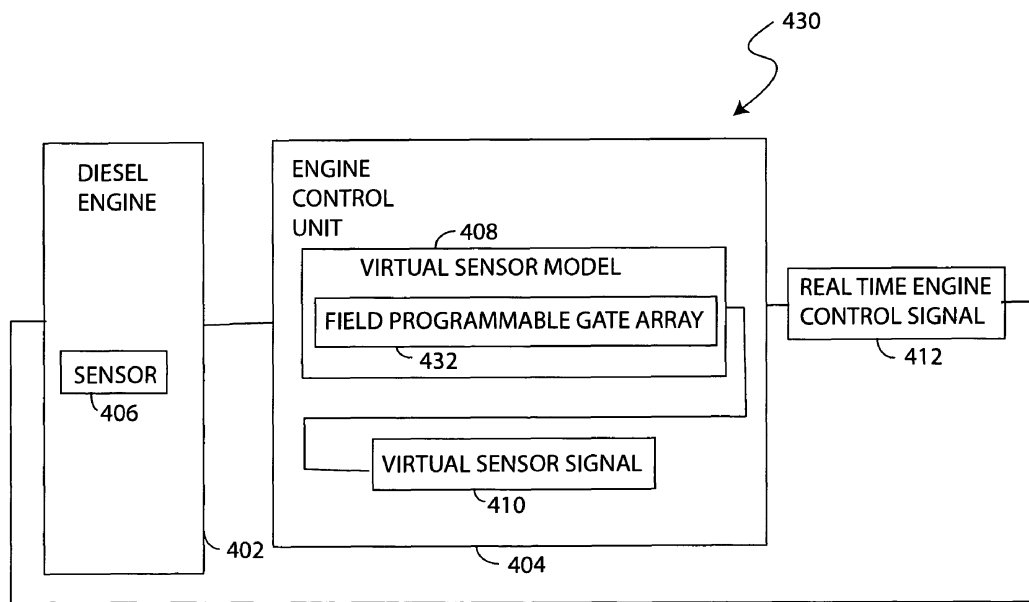


FIG. 4C

Figure 10.13: Patent Fig. 4C

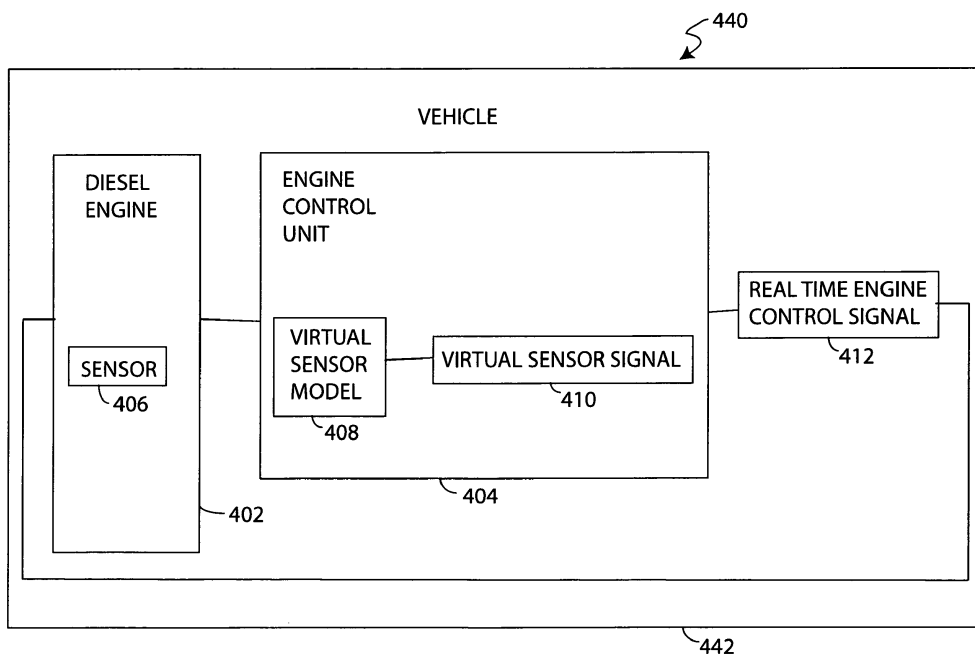


FIG. 4D

Figure 10.14: Patent Fig. 4D

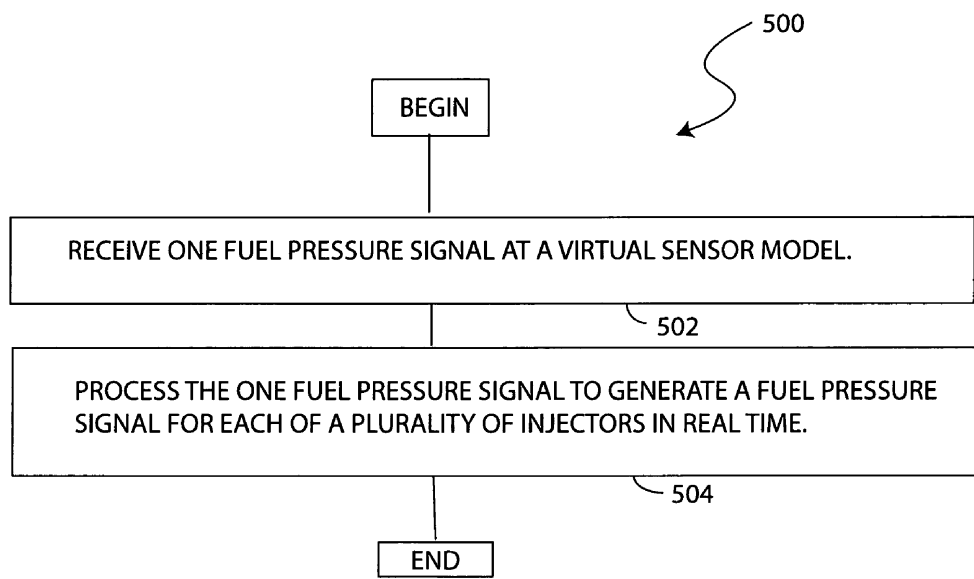


FIG. 5

Figure 10.15: Patent Fig. 5

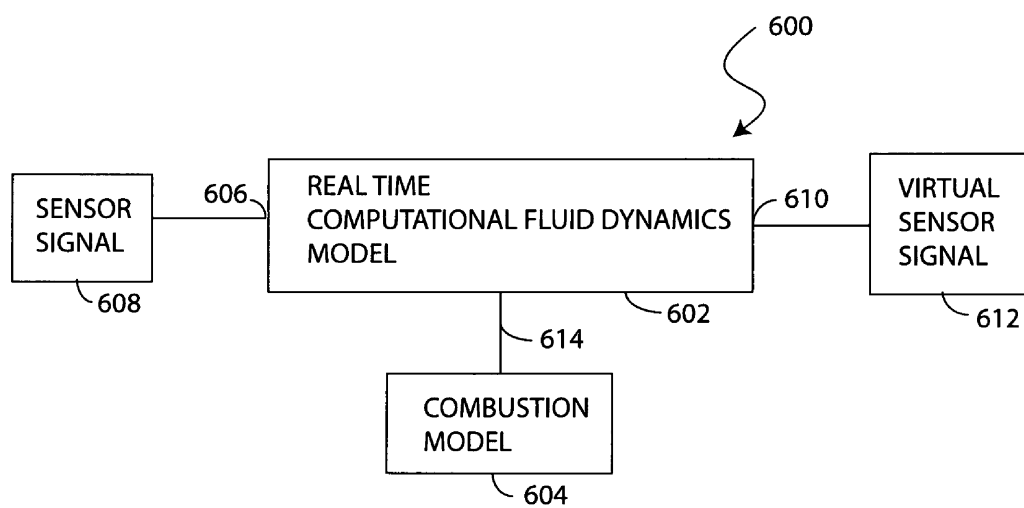


FIG. 6A

Figure 10.16: Patent Fig. 6A

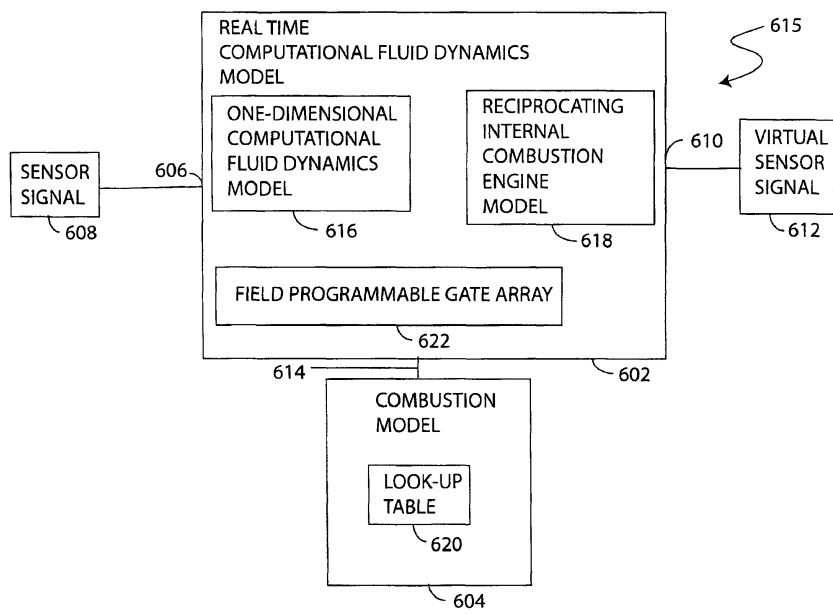


FIG. 6B

Figure 10.17: Patent Fig. 6B

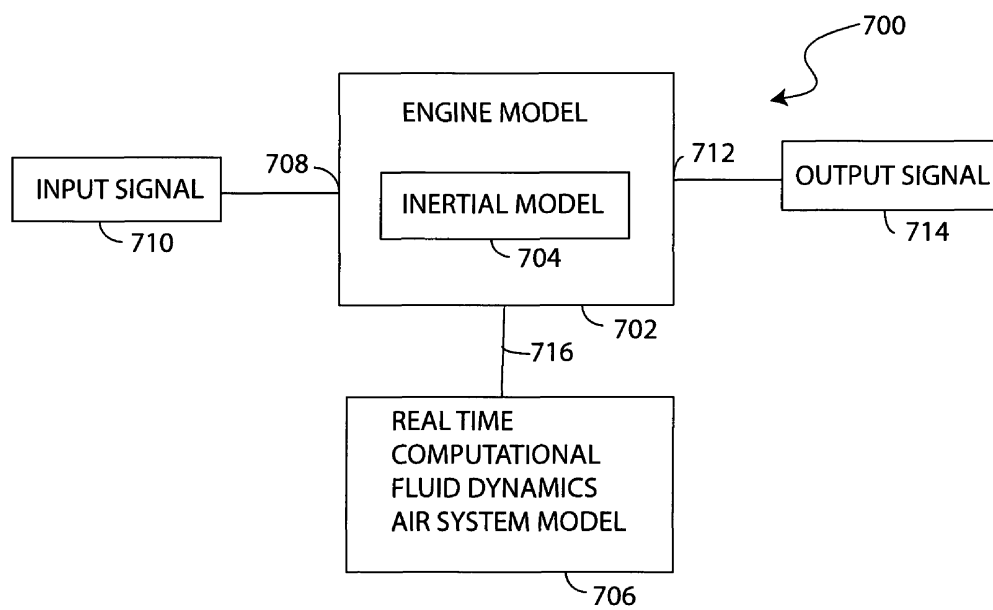


FIG. 7A

Figure 10.18: Patent Fig. 7A

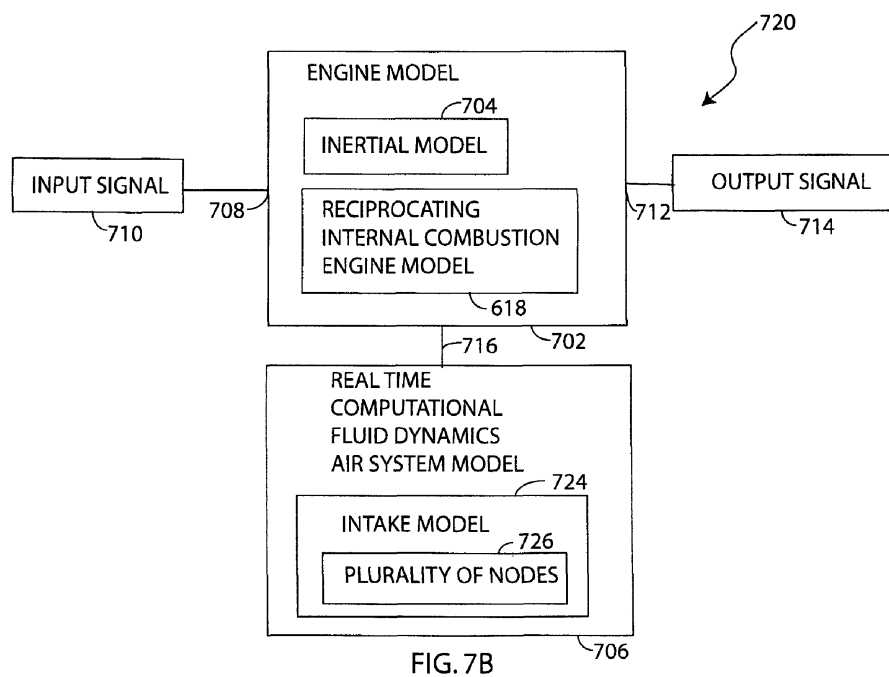


Figure 10.19: Patent Fig. 7B

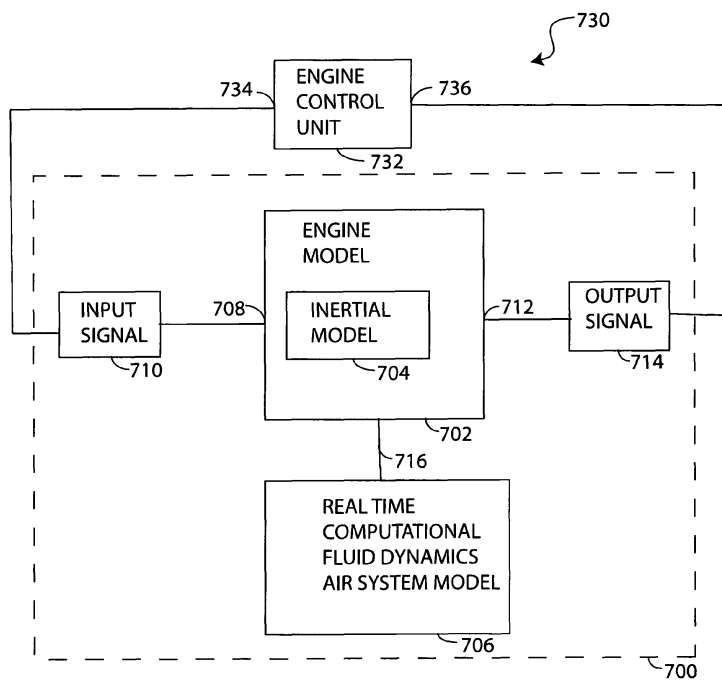


FIG. 7C

Figure 10.20: Patent Fig. 7C

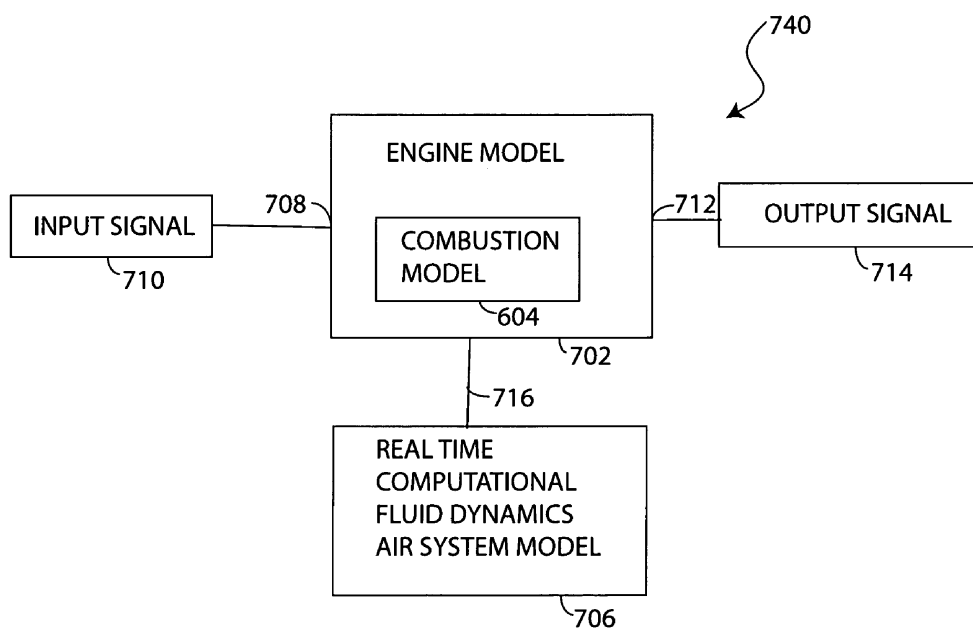


FIG. 7D

Figure 10.21: Patent Fig. 7D

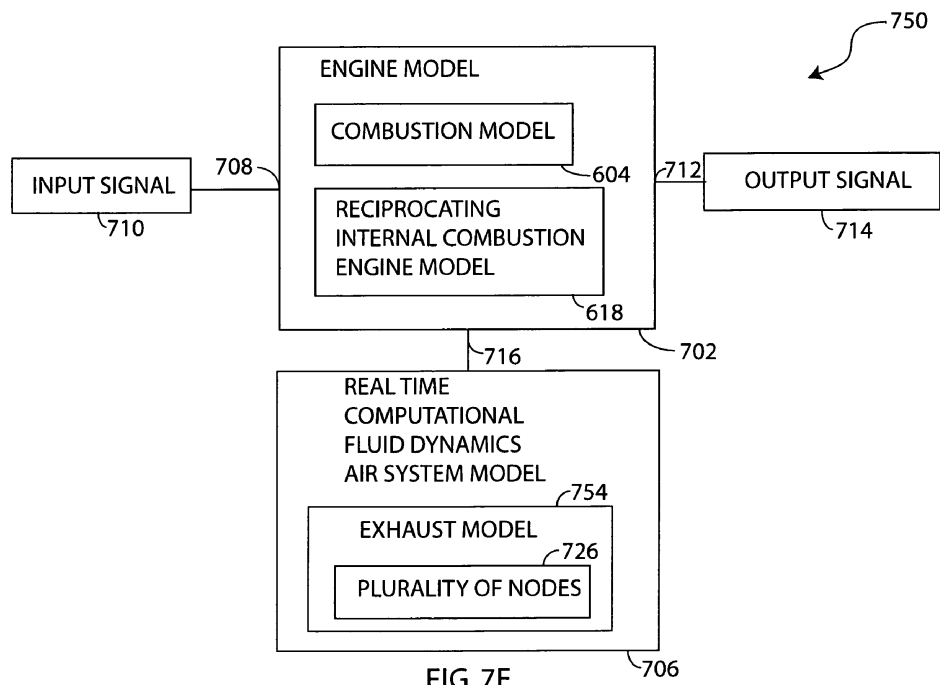


FIG. 7E

Figure 10.22: Patent Fig. 7E

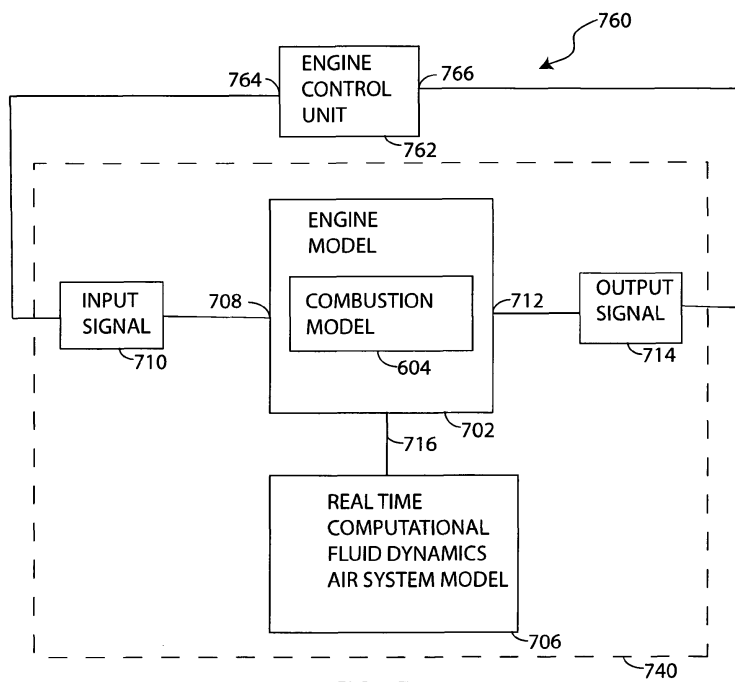


FIG. 7F

Figure 10.23: Patent Fig. 7F

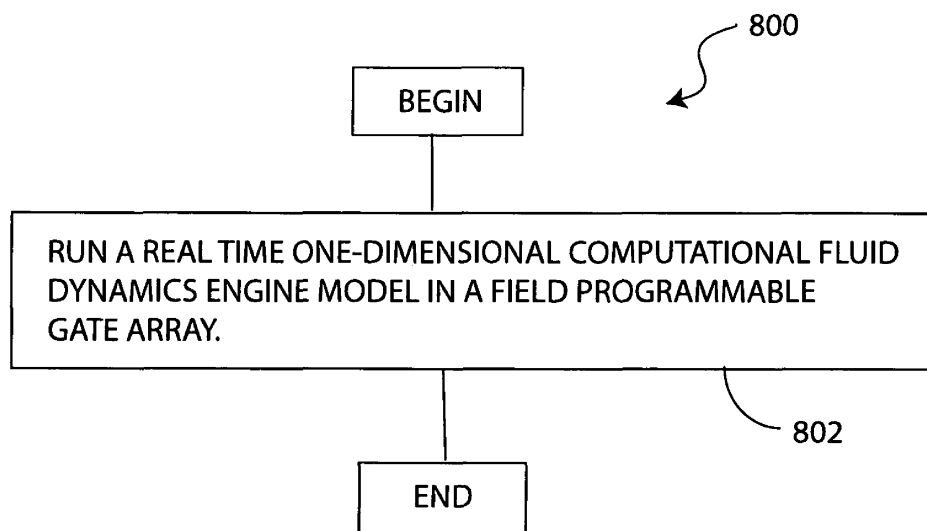


FIG. 8

Figure 10.24: Patent Fig. 8

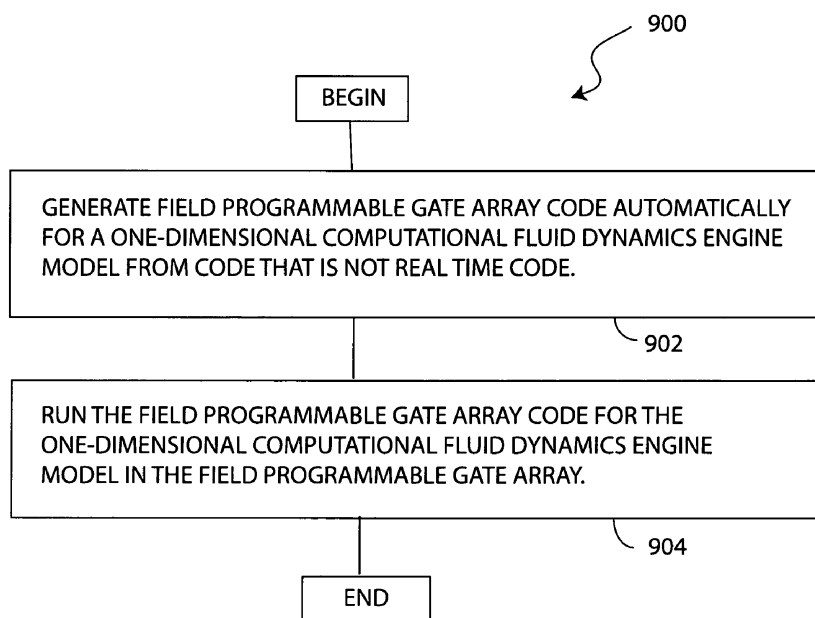


FIG. 9

Figure 10.25: Patent Fig. 9

Chapter 11

DISCUSSION

So far this paper has looked back at the development of FPGA based engine control through a collection of discrete publications. In this section we will step back and look at the broader picture.

FPGA based ECUs are now ubiquitous for advanced engine research and are starting to gain traction for low volume production. There are two major factors driving the use of FPGAs in research application.

The first is convenience. Well designed FPGA-based controllers are easier to use. They offer symmetry between channels and among devices not found in more traditional setups. They also can be setup to very logically partition code between low-level and high level code. Small tweaks are able to be made to the I/O platform by novice users without needing to dive in to the arcane depths of processor peripheral architecture.

The second driver is expanded capabilities. This manifests itself both in truly novel capabilities like custom computing machines and in more mundane ways like the ability to instantiate an oscilloscope inside the FPGA to help debug internal signals.

The downside of FPGAs in engine control is higher cost and lower temperature operation than conventional processors. In research applications there is no weighting for these negatives, but as volumes go up it becomes more of an issue.

The temperature problem can be approached in several ways. First, there are specialty FPGA manufacturers that make automotive temperature parts. These parts tend to cost more and not be supported in many 3rd party tool-chains.

Another approach to look at the systems view and determine if automotive (125C) temperature is required or if industrial (85C) parts will suffice. Setting the ECU away from the engine either on the skid or in the cowl may allow for the use of industrial temperature parts, though with increased packaging difficulties and costs.

In addition, parts can be screened for performance and/or operated at lower clock frequencies. Part information about performance de-rating for temperature is available for some parts. Should the demand for high-temperature parts be sufficient manufacturers will find a way to build high-temperature parts.

The cost issue should sort itself out in time. As Moores law predicts the cost of both processors and FPGAs will continue to decrease. At some point the incremental costs to switch to an FPGA is less than the benefits. The same thing happened with the switch from 8 to 32-bit processors to support programming in C, or the introduction of floating point in order to support model based control. When this will happen is uncertain, Drivven was founded on the assumption that this break over point was imminent, but we still have a way to go. Still, the path is clear, if not the rate of travel.

11.1 Ever Increasing System Complexity

Lets look at some additional drivers of complexity that will lead towards adoption of FPGAs in first large engines and then more mainstream applications.

Complexity of engine hardware and algorithms will increase to meet the demands of new emissions standards. Some of the major drivers of this will be:

- The availability of low-cost in-cylinder pressure sensors in production vehicles.
- Adoption of advanced combustion techniques that require cylinder pressure feedback for stability.
- The advances in fuel injector technology.

- Electrification of sub-systems like cam-phasers and superchargers.

Taking the technology drivers one at a time; some of the potential for low-cost pressure sensors was shown in Chapter 6. This was done with high-cost research grade sensors. Lower quality production sensors add additional complication/opportunity because the signals are less linear and prone to problems like thermal shock. We can expect huge growth in this area because direct cylinder pressure measurement and real-time analysis reduces the calibration effort in releasing an engine to production. Further, advanced combustion techniques centered around various forms of Low Temperature Combustion (LTC) are inherently unstable and require in-cylinder feedback for them to operate.

Advanced fuel injection systems and techniques offer great promise for advanced combustion systems. Chapter 7 discusses one of the shortcomings of such systems and a potential way to address it. From a higher level we see that advanced injection systems require not only the most complex actuation electronics they also require the most advanced feedback system to optimize.

Electrification of subsystems provides not only the first order goals of mild-hybridizations of conventional power-trains, it also increases system flexibility. Electrical systems have much faster response times than their hydro-mechanical cousins. In order to exploit this control loops with response times on the order of a few crank angle degrees need to be implemented. This sort of loop rate is too fast for a single CPU burdened with numerous other control activities.

Because emissions regulations are applied to high-volume engines first then to more specialized engines we can see further in to the future for low-volume engines. We know that the same basic technology that was used to clean up diesel trucks is starting to be applied to locomotive and will eventually be applied to ship engines.

Several key differences apply to the larger engines.

- We know what worked on the high volume engines.
- Large, low volume engines have higher per-unit cost than mass production engines, making the relative cost of high-power computing less of a design tradeoff.
- These engines tend to have small generalized engineering teams.

Engine control will be, if it is not already, too complex to be effectively calibrated using arrays of multi-dimensional maps. Advanced modeling of air and fuel systems will be required. This modeling is too complex for most small companies to take on, so they will need to go to outside sources for tools and components of their engine system. While it is not always the case, the lasting trend is for a single powerful central ECU and dumb actuators as opposed to numerous smart actuators. The ability to plug a supplier's software models and electrical driver/sensor interfacers into an ECU without major retooling will be a major advantage in terms of required engineering effort and reduced system complexity.

Projects like OSEK and AUTOSAR are attempts in this direction on the software side, more on this later. On the hardware side the greater control flexibility of an FPGA based ECU allows the ECU to adapt to varying requirements with less (and sometimes no) changes.

In figure 11.1 (courtesy of National Instruments) a typical embedded deployment curve is shown with progress from large PC based controllers through modular embedded controller to single board controllers and finally to system-on-chip integration. One additional advantage of the large engine area is that we do not have to go all the way to the tail of the development curve because of the low volumes and high costs of large engines it may be practical to go to "production" with modular embedded controllers to trade away engineering design costs with per unit hardware costs.

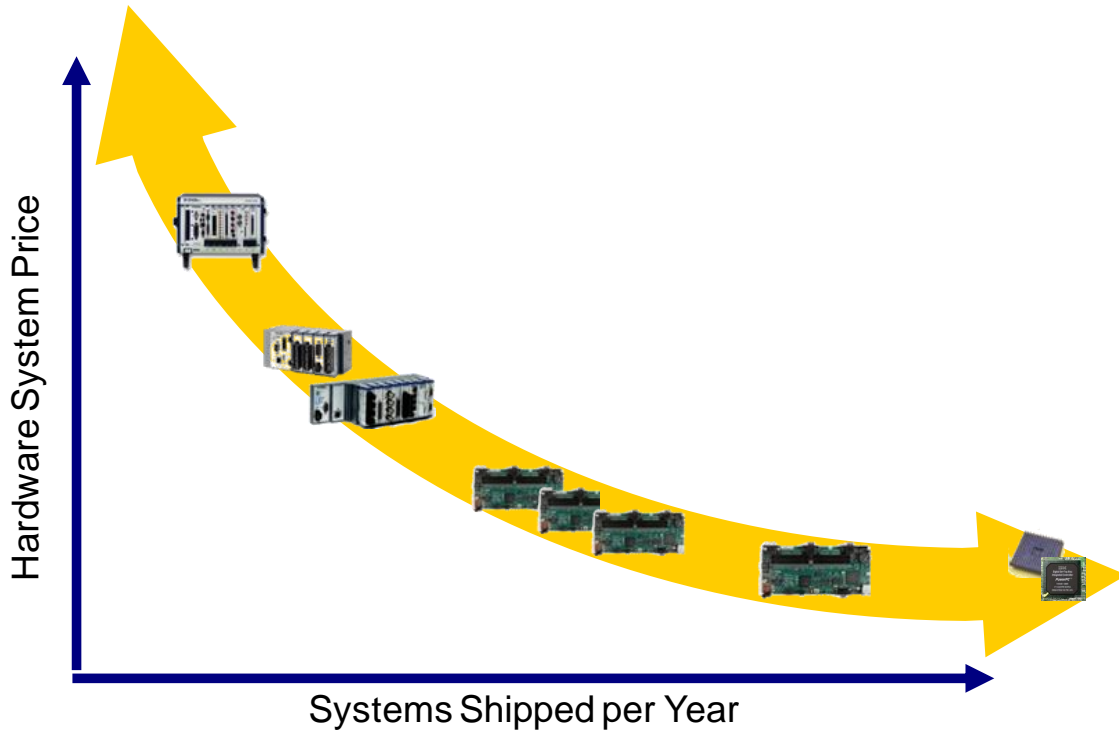


Figure 11.1: Product Development Curve

11.2 Hardware/Software Architecture for Reusable IP Blocks

In this section we use a series of case studies to illustrate some advantages of FPGA based design.

Because of the high development cost of a low-volume ECU, pin re-use becomes important in order to support the maximum variety of engines. In figure 11.2 a matrix of primary function and alternate function is listed. This table looks at the common circuits used for various I/O types and what subset of functionality is available in hardware. For instance, a low-side switch designed to drive a wastegate or fuel pump solenoid can be re-used as a switch input by wiring it in such a way that the short-circuit detection circuitry is used to identify if the switch is open or closed.

Realizing this full re-use matrix is difficult with a microcontroller because while the hardware is similar, the software may not be and the processor peripheral may

	0-5V	Thermistor	Batt Input	Potentiometer	Switch Input	Synchronous Analog	VR	Hall	Diff I/O	Serial Comm	Knock	Cylinder Pressure	Sensor Power	Low Side	High Side	MPRD	PFI	Unip Sol DI	Bip Sol DI	Unip Piezo DI	Bip Piezo DI	H-bridge	MIL	CAN	MODBUS	RS-485	Inductive Spark	CD Spark	Smart Coil	UEGO	NOx
Configurable Analog In	Green	Green	Green	Green	Green	Green					Green	Green																			
Direct Analog In	Green										Green	Green																			
VR/Hall					Yellow		Green	Green																							
Protected DIO								Green	Green																Green	Green			Green		
Sensor Supply												Green																			
Low Side					Yellow							Green																			
High Side													Green											Yellow							
MPRD													Green		Green																
PFI													Green		Green		Green							Yellow							
Unip Sol DI													Green		Green		Green														
Bip Sol DI													Green		Green		Green		Yellow												
Unip Piezo DI													Green		Green		Green		Yellow												
Bip Piezo DI													Green		Green		Green		Yellow		Green										
H-bridge													Green		Green		Green		Yellow		Green										
CAN																								Green							
Inductive Spark																											Green		Yellow		
CD Spark																											Green				
UEGO																														Green	
NOx																															Green

Figure 11.2: ECU Pin Reuse Matrix

not easily support this function. Moving to an FPGA-based approach makes realizing the full re-use matrix easy.

FPGAs have several limitations mentioned earlier, but the biggest is cost. In high-volume production of similar scale a dedicated chip will always cost less than a general purpose chip. There are, however, some mitigating circumstances. With a dedicated automotive microcontroller, for example the Freescale MPC565, there are features on the chip that you must pay for even if you never use. For instance the J1850 byte datalink controller that had fallen out of favor by the time that chip was released. Worse yet, if you need more of a peripheral than it supports, say an additional timer or counter, you are out of luck. While the offered count of 48 TPU channels seems like quite a few, the Driven implementation of a bipolar injector driver consumes 10 TPU-like channels per injector making external logic required to drive a six cylinder engine.

11.2.1 MPC565 H-Bridge implementation case study

An H-bridge circuit like the one shown in figure 11.3 is a common automotive circuit used to driver electronic throttles and other bidirectional valves.

A project I once worked on was a large low-volume ECU. I crafter the I/O to cover a number of engines and applications, then had to map the I/O to an MPC565 processor. Outputs like fuel injectors and sparks naturally fell to TPU channels. Low side PWM switches fell to the PWM channels, etc. By the time I got to the three H-bridges I did not have enough of any one type of processor resource to handle all three H-bridges. As a result one H-bridge was driven from the TPU, another from the MPWMSM (MIOS14 Pulse Width Modulation Sub Module) controller, and a third from the MDASM (MIOS14 Dual Action Submodule). Figure 11.4 diagrams the complexity of this system versys an FPGA based approach. Green blocks show identical code/hardware, yellow show similar and red show entirely different hardware.

Since all the H-bridge needed in its simplest form was one-pin to set direction and a PWM pin to set current this setup worked. The problem was making them symmetric. All three control PWM sources had different timebases, this meant that though some had 16-bit PWM values, the unified device driver that controlled all three devices could only support 8-bit precision.

Another problem was the limited subset of advanced features available to all channels. Ideally the H-bridge direction pin would be synchronized with the PWM command so direction changes occur seamlessly. Since this was not available on anything but the TPU driven H-bridge an alternate external IC was selected that could handle this condition, but cost more and had poorer performance.

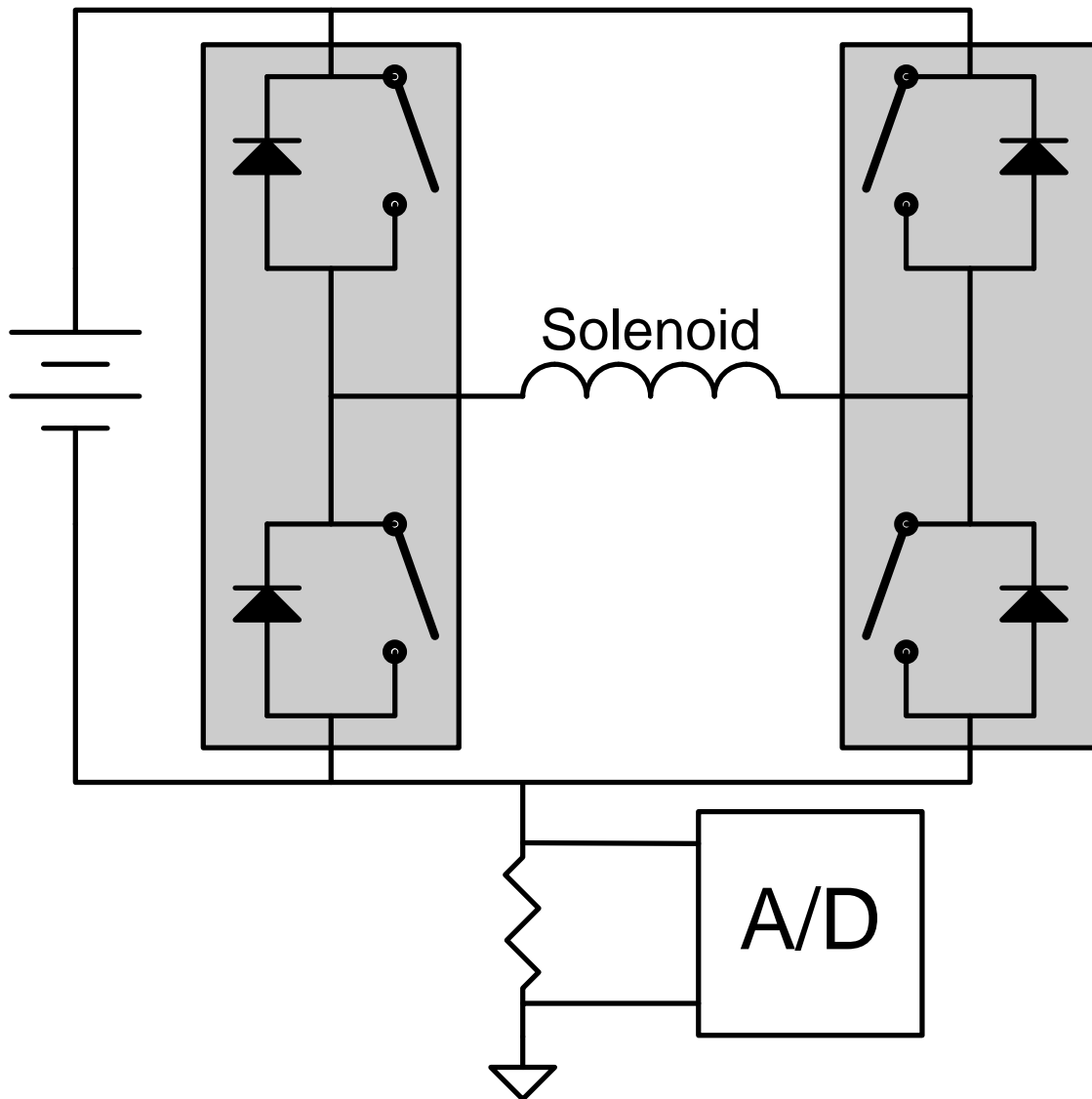


Figure 11.3: Simplified H-bridge driver schematic

Application	Application	Application	Application	Application
Abstraction Layer	H -> TPU	H -> MPWM	H -> MCT	H -> FPGA H
Device Driver	TPU H-control	MPWM & DIO	MDASM & DIO	FPGA H-control
Chip I/O	TPU	MPWM & DIO	MDASM & DIO	FPGA
I/O Circuit	Smart H Circuit	Smart H Circuit	Smart H Circuit	Simple H Circuit
Pin	H1	H2	H3	H1-H3

uP Based H1
uP Based H2
uP Based H3
FPGA Based H

Figure 11.4: Comparison of H-bridge implementations

11.2.2 Flexible Injector Design

One trend we see is a ratcheting up of processing and injector technology. Take diesel fuel injectors for example. For decades simple mechanical injectors were all that was available. Starting with the first electronically controlled injectors the ability to multiply inject became available. Two injection were common at first, but the incremental cost for more injection with a digital control system were trivial so 5 became the minimum expected of a control system. The increased number of injections made turn on and turn off times of injectors more critical, this lead to piezo based injectors with faster response times. Both of these systems still used a basically analog control of the individual pulses with some minor adjustment capability for the basic injection profile. As the costs of digital components came down the profile control converted to digital. This has lead to an explosion in the last few years of complex pulse sequences where each pulse in a multi-pulse event has a different voltage and current profile. Similarly this advanced control capability has led injector designers to develop designs that could not work without advanced software control. Take the Delphi DFI3 that needs to select a correct voltage profile on a pulse-by-pulse basis to keep the injector from tearing itself apart. (<http://delphi.com/manufacturers/auto/powertrain/diesel/crfs/directact/>)

11.2.3 No Peripheral Interrupts

One of the key benefits of an FPGA based architecture is the ability to tailor the I/O peripheral exactly to is application without requiring processor intervention. Taking the H-Bridge example again, lets say we want to do current control. In the common configuration where we keep the high side on and PWM the low side current is only flowing through the sense resistor when the low-side is on. This means the sense resistor value should only be sampled when the low-side is on.

In an MPC565 this requires sending the PWM command to the trigger pin of the Quad A/D Converter module (QADC) A/D converter or triggering an interrupt every PWM event. The former is limiting because the QADC has a small number of trigger lines, the latter is fine if that is all the processor is doing. Unfortunately, the number of tasks desiring interrupts on a modern engine control processor is closer to 100, so interrupting continuously is taxing on the software and creates numerous inter-module dependencies.

With an FPGA-based design the I/O component is precisely tailored to the application, so interrupting the processor is not needed. This is good because interrupt latency has grown with processor speed and programming language abstraction. This means that while the throughput of in-line code has gone up by a factor of 100 since the first automotive processors in the 1980s the time it takes to service an interrupt is roughly the same.

Compounding the interrupt problem are large development teams and high level languages. With the popularization of model-based control in ECUs, the high level applications are increasingly written in high-level dataflow languages like Simulink and LabVIEW. Suppliers are expected to deliver both simulations and control models of their components in these languages, along with normal technical data. These models and languages lend themselves to strait-line execution with no interrupts.

Further, the days of a single programmer writing most of the code for an ECU are over. Now large development teams build and validate larger parts of the code independently. Components like operating systems and network stacks come from 3rd party vendors. The current design process requires significant effort mapping all these components to a specific hardware platform.

An alternate path, one with an FPGA at its core, has vertical slices of architecture from the application all the way to the sensor or actuator. Instead of teams building horizontal wafers of the system, teams can build vertical slices. They

can design, model, and optimize the application, device driver, hardware, and sensor/actuator package. This package, once done, can be moved from one design to the next. Because most of the complex logic is in the FPGA fabric the I/O circuits are able to implemented with more generic parts. This leads to fewer problems with obsolescence.

The FPGA itself, if correctly written, is proof against obsolescence. While the life of any specific FPGA is on the order of a few years, similar to that of a processor, upgrading to a newer FPGA is seamless. A fully static FPGA design that does not use manufacturer specific blocks is just a compile away from a new device. A similar upgrade from one processor family to another requires several man-years of development.

11.3 Blurring of the Hardware/Software divide

Xilinx introduced the VirtexPro line of FPGAs, with hard PowerPC cores, in the early 2000s and more recently the Zynq line with a hard ARM core. These combined the best of both worlds with the speed of a direct silicon implementation of a processor and the peripheral flexibility of an FPGA.

The PowerPC 405 and 440 cores had a fairly ridged structure, somewhat limiting the flexibility of implementation. Xilinx's soft core Microblaze architecture on the other hand allowed a great deal of flexibility. It used an open source GNU toolchain so that both the core and compiler could be tailored to meet a specific need. The Microblaze fell short in that being a soft core processor it was an order of magnitude slower than its directly implemented brethren.

The Xilinx Zynq solves many of these shortcomings. It's arm architecture supports a co-processor instruction allowing extensibility for specific functions, but also the speed of a hard processor. Because ARM is a popular core choice across

the embedded industry and supported by all the top FPGA manufacturers we can expect this combination to have more staying power than the Virtex Pro line.

The ability to extend the processor for specific applications is not new to automotive. The Motorola 6833x processor line supported a dedicated lookup table instruction not found on the parent 68000 series processor. We exploit a similar concept with the heterogeneous core architecture of the custom computing machine papers.

Taking this process a little further we can envision a system on chip where entire subsections of the design are carried out by their own discrete execution units. An extreme example of this is the real-time CFD model presented earlier. In this case a section of the FPGA is used to model the whole fuel system. Another section of the FPGA may be used to model the air system.

Similarly we can envision an entire closed loop model for valve controller where the desired timing per cylinder event is set by the high level controller and implemented by a high-speed control model acting independently in a separate part of the chip. Today similar concepts are used with discrete controller units, but these are limited by network speed Cheever et al. (2012).

Another example of subsystems that can be given their own FPGA blocks are network communications. This is done today to a certain extent with Can and Flexray modules and of-course Ethernet hardware. In all these cases though interrupts are required to move data to and from the processor and do the higher level processing. While some of this can be alleviated with advanced linked-list DMA controllers they still face challenges of only being able to handle the simplest network traffic without CPU intervention and are extremely chip dependent. An FPGA based version of this architecture allows complete network packets to be processed, data scaled, and place in the correct mailboxes without CPU intervention.

Because FPGAs come in various size/cost options in the same physical package it is possible to have a more complex unit for debugging and smaller chip for deployment. For example a development ECU may have a larger FPGA that implements knock detection with an FFT and allows real-time plotting and data capture. For the deployment version a much more compact integer FIR time-domain filter implementation may be used. With a processor based design external hardware like the Motorola MKICKS or Driven DCAT is required to do the FFT based knock tuning.

11.4 Model based control and model co-processors

Modern engine control is model-based control. By this I mean that while the feedback still tends to be of the PID variety, there is a great deal of linearization and feed-forward based on low-order models of the physical system. Since models want to run at different rates, some on conventional timebases, others stepping based on cycles or crank angle degrees, it may be advantageous to look at multi-processor solutions.

In this paradigm data is taken and processed using the mailbox mechanisms from something like AUTOSAR, but instead of being processed as a thread in a multi-threaded OS on a single processor it is processed on its own dedicated processor.

This simplifies the modeling a great deal. Their determinism is enhanced by not having engine synchronous and time-based events occur on the same processor. This means that worst case performance of algorithms is easy to determine.

Once worst case instruction execution is known, along with the worst-case hard real-time requirement, the execution speed of the processor can be determined. From this the area of the processor can be minimized by either changing how the various

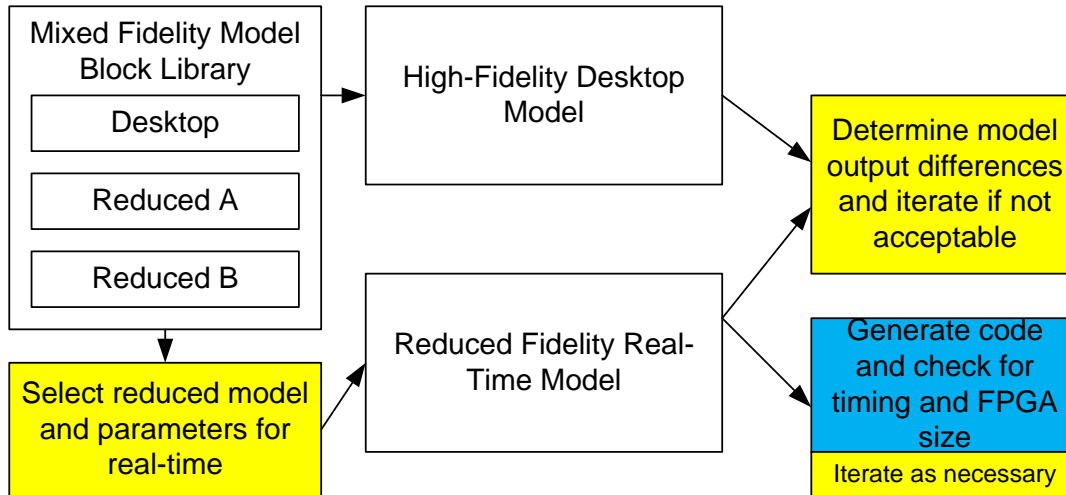


Figure 11.5: Overview of model optimization process

elements are structured or by threading it in to other processors like the PRET processors described in the custom computing machine papers cores.

11.5 System Building Tools

To round out the work in chapter 7 on custom computing machines and turn it in to a useful tool we need to create a set of systems building tools. These tools allow the generation of real-time FPGA based models without the need for huge amounts of human intervention in the process.

To make this process a reality we will need to convert from an industry standard, but closed source modeling environment like GT-SUITE to an open-source one like Modellica. From there we need to generate a new set of model blocks to support real-time. Each of these blocks will have different versions supporting different trade-offs between fidelity and speed, with the full fidelity blocks possibly not supporting explicit real-time solutions. This is outlined in figure 11.5 where the yellow blocks are iterative user interaction and the blue block represents the automated processes described below.

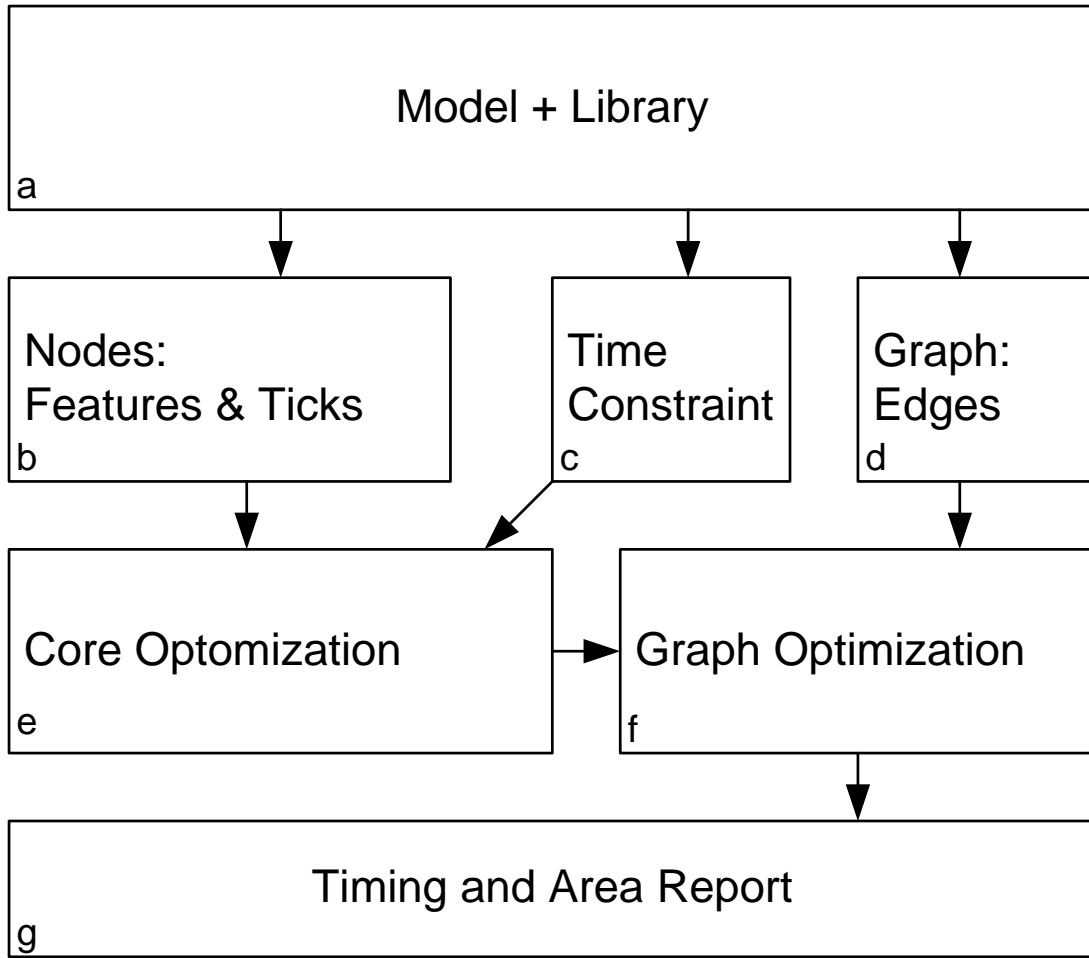


Figure 11.6: FPGA model generation process

The user can build a high-fidelity model with as accurate as possible a simulation of the real system. Next a reduced fidelity model is built from the reduced fidelity blocks and possibly reduced system constraints. The results are then compared against the high-fidelity model to determine if the reduced fidelity model is sufficiently accurate. If it is then the model is passed through the system generation tools to determine if it is possible to execute it in real time and how large and FPGA would be required.

Figure 11.6 details the process of generating the first intermediate step based on the model and library blocks chosen (block a). For each element used in the library we know how many thread cycles it would take to execute on a PRET or

similar architecture as described in the custom computing machine papers above, we also know the list of resources used (block b). From the model we know the time constraints (block c) and can decide the amount of multiplexing of threads per core we are allowed (block e). Examining the model we can construct a graph of interactions between model blocks. This graph is then optimized to ensure that most of the model communication is between cores instead of across them to keep communication using the “free” BRAM (block f). In the case of multi-chip communication the graph must be optimized to segregate the model across chips where the allowed communication channels are limited to only one or two graph edges.

Once this is done the timing and initial area report are generated (block g) allowing the user to know if the model can execute in real-time and an approximate FPGA area required.

Lastly the process of code generation is outlined in figure 11.7. This takes the optimized graph from the previous step (block c) as well as the model (block a) and library (block b) from the first steps. The model and library are used to fill in the constants and tunable parameters of the model so they can be optimized by the C compiler (block d). The interconnects are defined by the graph layout, the code generation tool then fills in the appropriate code for each interconnect to direct the block where to get values from and where to publish them (block e).

Once this is done the final C code and associated make file are generated for each processor thread (block f). They are then all compiled in to a collection of object files (block g).

The compiled C code, compiled FPGA code (block h) and external FPGA code are all linked together, placed and routed, so that they can be deployed on to a target and the model can be executed in real time.

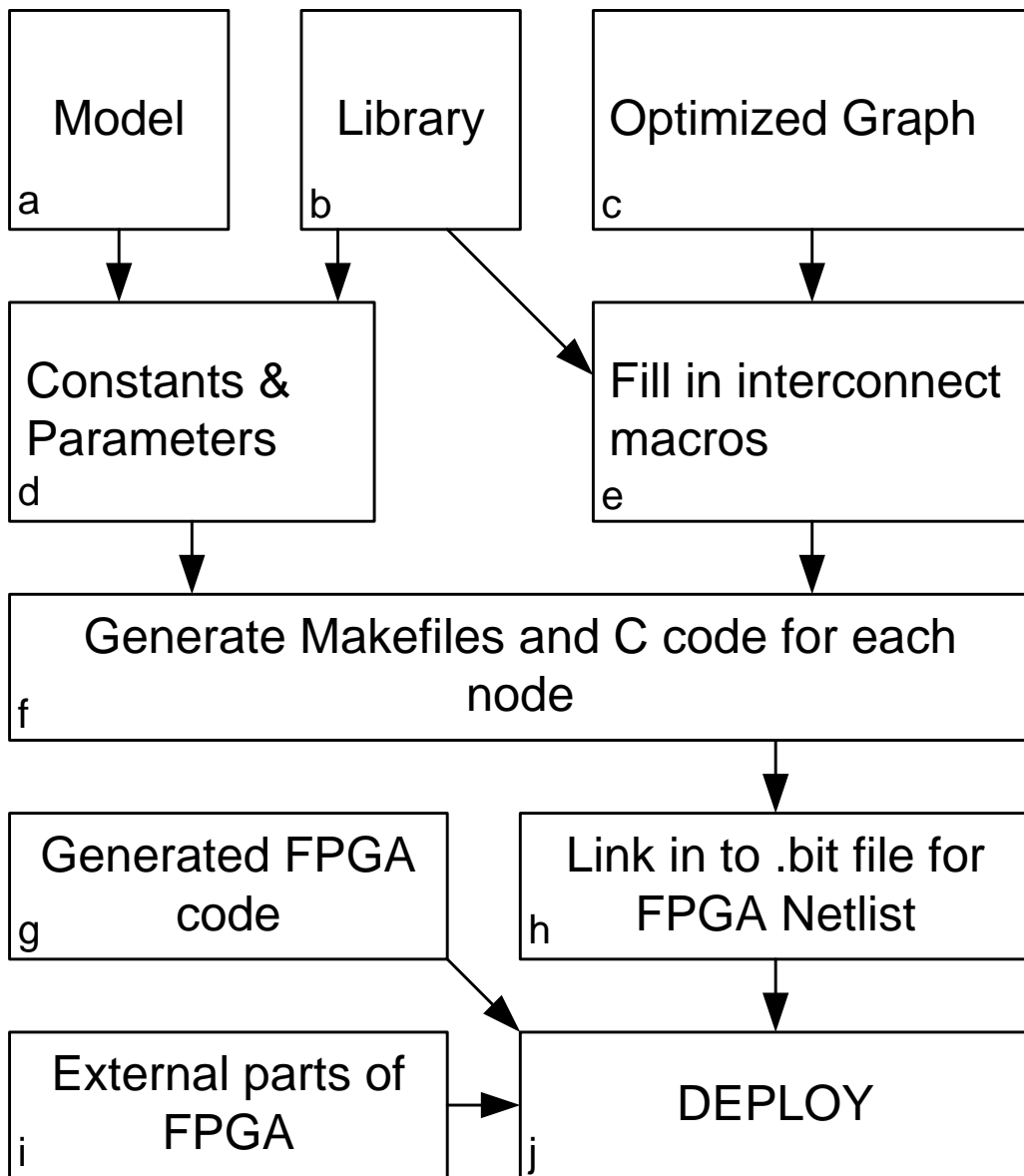


Figure 11.7: Model code generation process

11.6 Re-inventing the wheel: the case for open source research software

One final note. The automotive software industry is very secretive. On top of this there are few text books with any detail on engine control algorithms. As a result we see many researchers, especially in an academic setting, developing the same algorithms.

Following the same path of the operating system industry years ago, two paths are recommended. The first is the publication of practical text books on engine control software. The current publications tend to be either shallow like Bosch (2006) and Ribbens (2012) or fairly abstract Gueezlla and Onder (2009) . A concrete text with worked examples would greatly cut down on the learning curve that most researchers have to go through.

The second prong of this effort is, again following what happened decades ago with the OS industry, open source application software and standard software interfaces. While we see standardization efforts with OSEK/VDX (<http://www.osekvdx.org/>) and later AUTOSAR (<http://www.autosar.com>) these standards are both complex and voluntary. AUTOSAR outlines a nice mailbox framework for intermodel communication and tools to generate it, but it has some drawbacks. The AUTOSAR framework to describe a PWM is over 20 pages and still leaves most of the implementation details up to the user. More complex tasks like multi-pulse injector control are left, so far, TBD with the standard.

Across the research community we see a huge duplication of effort, and it is safe to assume this continues throughout industry. Many man-years are spent implementing the same basic I/O functions by different institutions. The same goes for basic engine control routines like a simple speed-density fuel control algorithm. Even simple models like wall-wetting and manifold filling are built and rebuilt.

We have seen a number of projects in this area over the years. The DIY-EFI project based on a Motorola 68332 and the RTEMS project was one of the first, but floundered for lack of hardware.

The Megasquirt project contains much of the core ingredients, but is targeted to the hobbyist market and has traditionally used such low-end processors and tools that they have been quite limited in what they can achieve.

Some level of work is being done in this area at the academic level, Stobart et al. (2011) but so far it is just a start without a broader base of active users like most successful open-source projects. Promoting this approach on both a hardware and software level is a long-term goal of this author.

Chapter 12

CONCLUSIONS

This dissertation covered major advances in FPGA based engine controllers over the past decade. Three major areas of FPGA applications were discussed:

- FPGAs as engine control coprocessor.
- FPGAs for advanced engine control.
- FPGAs as custom computing machines.

The topic of FPGAs as engine control coprocessors is now fairly mature from an advanced research perspective. The basic architecture for the Driven EPT has been stable for 7 years. Tweaks to the internal code like those to support free piston engines or start-stop applications where the engine rolls back are incorporated occasionally.

The focus in this area has shifted and is now focused on two areas. The first is complex I/O. As diesel and gasoline direct injection systems get more complex the need for more complex pulse sequencing increases. Conversely, with the ability to shift more complexity in to the control electronics, actuators are being designed that require much more complex controls. The second branch of research is combining the FPGA and processor on the same chip. This is exemplified by the Xilinx Zynq series of FPGAs with dual core ARM processors embedded in them and “soft” coprocessor peripherals.

The use of FPGAs for advanced engine control as outlined in the included paper is primarily focused in rapid feedback from in-cylinder combustion sensors. This field had been largely gated by the lack of production cost/quality cylinder pressure

feedback sensors. Now that these sensors are becoming commonplace, research in to algorithms and tools to exploit their availability has increased dramatically. This is expected to be a major research focus for years to come and the FPGA is expected to be a core enabling technology.

FPGA based custom computing machines in engine control is still a very advanced research topic. Years of work still lie ahead to transition the 1D-CFD application and similar custom computing machines from complex research devices requiring expert setup to appliance-like devices that "just work" like the engine control coprocessor FPGA software. Still, the path is clear that due to lowering costs and increasing capability of FPGAs, these types of custom specialty processors will eventually become standard subsystems in the larger engine control picture.

This author is proud to have been at the forefront of research in each of these areas and expects to continue to contribute in this area for years to come.

Bibliography

- (2000). *Bosch Automotive Handbook*. Robert Bosch.
- (2009). Dcat user's manual. [http://www.drivven.com/visitor_download/Manuals/DCAT User's Manual.pdf](http://www.drivven.com/visitor_download/Manuals/DCAT%20User's%20Manual.pdf).
- (2012). *Core generator guide*. Xilinx.
- (2012). GNU ARM Toolchains.
12. Guerrier, M. and Cawsey, P. (2004). The development of model based methodologies for gasoline ic engine calibration. In *SAE Technical Paper 2004-01-1466*.
- Aceituna, D. (2002). Algorithm design using labview. In *SAE Technical Paper 2002-01-1465*.
- Amann, C. A. (1987). *How Shall We Power Tomorrows Automobile?* Plenum Press, New York.
- Annand, W. J. D. (1963). Heat transfer in the cylinders of reciprocating internal combustion engines. *Proc. I. Mech E.*, 177(36):973–989.
- Asad, U. and Zheng, M., Han, X., Reader, G., and Wang, M. (2009). Fuel injection strategies to improve emissions and efficiency of high compression ratio diesel engines. *SAE International Journal Fuels Lubrication*, 1(1):1220–1233.
- Asad, U. and Zheng, M. (2008). Real-time heat release analysis for model-based control of diesel combustion. In *SAE Technical Paper 2008-01-1000*.
- Asad, U. and Zheng, M. (2009). Efficacy of egr and boost in single-injection enabled low temperature combustion. *SAE International Journal of Engines*, 2(1):1085–1097.

- Bauer, H. (2004). *Diesel-Engine Management*. Society of Automotive Engineers, 3rd edition.
- Beaumont, A., Lemieux, J., Battiston, P., and Brown, A. (2006). Design of a rapid prototyping engine management system for development of combustion feedback control technology. In *SAE Technical Paper 2006-01-0611*.
- Bereisa, J. (1983). Applications of microcomputers in automotive electronics. *Industrial Electronics, IEEE Transactions on*, IE-30(2):87–96.
- Bianchi, G. M., Falfari, S., Pelloni, P., Filicori, F., and Milani, M. (2002a). Development of a dynamics model for studying the 1st generation of common rail injectors for hsd diesel engines. In *SAE Technical Paper 2001-01-013*.
- Bianchi, G. M., Falfari, S., Pelloni, P., Kong, S.-C., and Reitz, R. D. (2002b). Numerical analysis of high-pressure fast-response common rail injector dynamics. In *SAE Paper 2002-01-0213*.
- Bittle, J. and Jacobs, T. (2011). On the effect of injection pressure on two-stage ignition behavior of low temperature diesel combustion. In *2011 7th US National Combustion Meeting, Atlanta, Georgia*.
- Bittle, J., Knight, B., and Jacobs, T. (2010a). Demonstration of low temperature combustion for simultaneous and substantial reductions in nox and soot in a medium-duty diesel engine. In *Central States Section of the Combustion Institute*.
- Bittle, J., Knight, B., and Jacobs, T. (2010b). Efficiency considerations of diesel premixed charge compression ignition combustion. In *2010 Directions in Engine-Efficiency and Emissions Research Conference*.

- Bittle, J., Knight, B., and Jacobs, T. (2010c). Investigation into the use of ignition delay as an indicator of low-temperature diesel combustion attainment. *Combustion Science and Technology*, 183(2):138–153.
- Bittle, J., Knight, B., and Jacobs, T. (2011a). Heat release parameters to assess low temperature combustion attainment. In *SAE Technical Paper 2011-01-1354*.
- Bittle, J., Knight, B., and Jacobs, T. (2011b). Two-stage ignition as an indicator of low-temperature diesel combustion. *Combustion Science and Technology*, 183(9):947–966.
- Blarigan, V. (2002). Rapid combustion electrical generator. Technical report, Sandia National Laboratories. Chicago, IL, April 23-24.
- Borgqvist, P., Tunestl, P., and Johansson, B. (2011). Investigation and comparison of residual gas enhanced hcci using trapping (nvo hcci) or rebreathing of residual gases. In *SAE Technical Paper 2011-01-1772*.
- Bosch, R. (2006). *Gasoline Engine Management*. Wiley, 3rd edition.
- Bosch CC195 (1996). *Bosch CC195 Knock Sensor IC Specification*. Bosch Semiconductor.
- Briggs, T., Cho, K., Curran, S., Kim, J., Nafziger, E., and Wagner, R. (2011). High efficiency clean combustion in multi-cylinder light-duty engines. Technical report, 2011 DOE Hydrogen Program and Vehicle Technologies Merit Review.
- Carter, D. and Wechner, E. (2003). The free piston power pack: Sustainable power for hybrid electric vehicles. In *SAE Technical Paper 2003-01-3277*.
- Cavina, N., Corti, E., Poggio, L., and Zecchetti, D. (2011). Development of a multi-spark ignition system for reducing fuel consumption and exhaust emissions of a high performance gdi engine. In *SAE Technical Paper 2011-01-1419*.

- Ceccarani, M., Cacciatore, D., and Fusco, N. (2005). Development guidelines for a lamborghini high performance engine. In *SAE Technical Paper 2005-24-095*.
- Cengel, Y. A. and Boles, M. A. (2002). *Thermodynamics: An Engineering Approach*. McGraw Hill, Inc., Boston, MA.
- Chatlatanagulchai, W., Pongpanich, N., Wannatong, K., and Rhienprayoon, S. (2010a). Quantitative feedback control of air path in diesel-dual-fuel engine. In *SAE Technical Paper 2010-01-2210*.
- Chatlatanagulchai, W., Rhienprayoon-PTT, S., and Wannatong-PTT, K. (2011a). Sliding mode control of air path in diesel-dual-fuel engine. In *SAE Technical Paper 2011-01-0917*.
- Chatlatanagulchai, W., Rhienprayoon-PTT, S., Yaovaja, K., and Wannatong-PTT, K. (2010b). Air/fuel ratio control in diesel-dual-fuel engine by varying throttle, egr valve, and total fuel. In *SAE Technical Paper 2010-01-2200*.
- Chatlatanagulchai, W., Yaovaja, K., Rhienprayoon, S., and Wannatong, K. (2010c). Air-fuel ratio regulation with optimum throttle opening in diesel-dual-fuel engine. In *SAE Technical Paper 2010-01-1574*.
- Chatlatanagulchai, W., Yaovaja, K., Rhienprayoon, S., and Wannatong, K. (2010d). Gain-scheduling integrator-augmented sliding-mode control of common-rail pressure in diesel-dual-fuel engine. In *SAE Technical Paper 2010-01-157*.
- Chatlatanagulchai, W., Yaovaja, K., Rhienprayoon-PTT, S., and Wannatong-PTT, K. (2011b). Fuzzy knock control of diesel-dual-fuel engine. In *SAE Technical Paper 2011-01-0690*.

- Cheever, G., Sullican, C., Schten, K., Punater, A., and Erickson, C. (2012). Design of an electric variable cam phaser controller. In *SAE Technical Paper 2012-01-0433*.
- Chinitz, W. (1969). Rotary engines. *Scientific American*, pages 90–99.
- Cho, K., Curran, S., Prikhodko, V., Sluder, C., Parks, I., James, E., and Wagner, R. (2011). Experimental investigation of fuel-reactivity controlled compression ignition (rcci) combustion mode in a multi-cylinder, light-duty diesel engine. Technical report, Oak Ridge National Laboratory (ORNL); Fuels, Engines and Emissions Research Center.
- Ciatti, S. and Subramanian, S. (2011). An experimental investigation of low-octane gasoline in diesel engines. *Journal of Engineering for Gas Turbines and Power*, 133:092802.
- Ciatti, S. A., Subramanian, S., Das Adhikary, B., and Aggarwal, S. (2010). A study of low-cetane kerosene fuel in diesel engines. In *Central States Section of the Combustion Institute*.
- Ciulli, E. (1992). A review of internal combustion engine losses - part 1: Specific studies on the motion of pistons, valves and bearings. *Proc. Instn Mech. Engrs*, 206:223–236.
- Ciulli, E. (1993). A review of internal combustion engine losses - part 2: Studies for global evaluations. *Proc. Instn Mech. Engrs*, 207:229–240.
- Corti, E., Cazzoli, G., Rinaldi, M., and Solieri, L. (2008a). Fast prototyping of a racing diesel engine control system. In *SAE Technical Paper 2008-01-2942*.
- Corti, E. and Forte, C. (2011). Real-time combustion phase optimization of a pfi gasoline engine. In *SAE Technical Paper 2011-01-1415*.

- Corti, E., Moro, D., and Solieri, L. (2008b). Measurement errors in real-time imep and rohr evaluation. In *SAE Technical Paper 2008-01-0980*.
- Corti, E., Moro, D., and Sollieri, L. (2007). Real-time evaluation of imep and rohr-related parameters. In *SAE Technical Paper 2007-24-0068*.
- Corti, E. and Sollieri, L. (2005). Rapid control prototyping system for combustion control. In *SAE Technical Paper 2005-01-3754*.
- Cummings, R. (2003). Tool integration from design to test. In *SAE Technical Paper 2003-01-1204*.
- Curran, S., Cho, K., Briggs, T., and Wagner, R. (2011a). Drive cycle efficiency and emissions estimates for reactivity controlled compression ignition in a multi-cylinder light-duty diesel engine. In *ASME Technical Paper Number ICEF2011-60227*.
- Curran, S., Hanson, R., Barone, T., Storey, J., and Wagner, R. (2011b). Rcci operation on a light-duty multi-cylinder engine using sme b20 with gasoline and ethanol blends. In *National Biodiesel Board Biodiesel Technical Workshop*.
- Curran, S., Hanson, R., Cho, K., Barone, T., Wagner, R., Kokjohn, S., and Reitz, R. (2010). Range extension and multi-cylinder performance of rcci. In *Proceedings of the 7th U.S. National Combustion Meeting*.
- D'Angelo, M., Giacomini, R., and Saotome, O. (2006). A flexible three-level architecture for engine control modules. In *SAE Technical Paper 2006-01-2734*.
- Deng, J., Winward, E., Stobart, R., and Desai, P. (2010). Modeling techniques to support fuel path control in medium duty diesel engines. In *SAE Technical Paper 2010-01-0332*.

- Deng, J., Xue, Y., Stobart, R., and Maass, B. (2011). Fuel path control of a diesel engine at the operating point of the low load and medium speed. In *Control and Decision Conference (CCDC), 2011 Chinese*, pages 747–751. IEEE.
- Desantes, J., Arreglt, J., and Rodriguez, P. (1999). Computation model for simulation of diesel injection systems. In *SAE Paper 1999-01-0915*.
- Dongiovanni, C. and Coppo, M. (2010). Accurate Modelling of an Injector for Common Rail Systems. In Siano, D., editor, *Fuel Injection*, chapter 6. InTech.
- Dorenkamp, R. and Gruber, M. (2008). U. s. light duty clean diesel - volkswagen/audi met the technical challenge. In *Directions in Engine-Efficiency and Emissions Research (DEER) Conference*.
- Drivven (2009). *DI Driver Module Kit User's Manual*. Drivven. Rev. E.
- Dumitrescu, C. E., Neill, W. S., Guo, H., Hosseini, V., and Chippior, W. L. (2010). Fuel property effects on pcci combustion in a heavy-duty diesel engine. *ASME Conference Proceedings*, 2010(49446):255–264.
- Dunbeck, P. and Reitz, R. (2010). An experimental study of dual fueling with gasoline port injection in a single-cylinder, air-cooled hsd diesel generator. In *SAE Technical Paper 2010-01-0869*.
- Dunbeck, P. B. (2009). An experimental study of dual fueling with port injection in a single cylinder air cooled hsd diesel engine. Master's thesis, University of Wisconsin - Madison.
- Edwards, S. A. and Lee, E. A. (2007). The case for the precision timed (PRET) machine. In *Proceedings of DAC '07*, pages 264–265.

- Evans, I., Bowen, P., Kay, P., King, J., Knight, G., and Schmidt, L. (2010). Characterisation of a bio-ethanol direct injection spray under sub-zero conditions. In *Europe 2010, 23rd Annual Conference on Liquid Atomization and Spray Systems*.
- Florea, R., Zha, L., Jansons, M., Taraza, D., and Henein, N. (2012). Reactivity stratification investigation of ethanol/ulsd dual-fuel combustion using formaldehyde plif. SAE Technical Paper Draft 12PFL-0951.
- Flynn, G. (1957). Observations on 25,000 hrs of free-piston engine operation. *Transactions of SAE*, 65.
- Freescale (2001). *PROSAK IC - PROgrammable Stand Alone Knock*. Freescale Semiconductor.
- Fritz, M., WeiB, E., Alberter, G., and Hettich, G. (1996). Diesel engine management with a new housing technology. In *SAE Technical Paper 960043*.
- Gabrick, M., Nicholson, R., Winters, F., Young, B., and Patton, J. (2006). Fpga considerations for automotive applications. In *SAE Technical Paper 2006-01-0368*.
- Gatowski, Balles, Chun, Nelson, Ekchian, and Heywood (1984). Heat release analysis of engine pressure data. In *SAE Technical Paper 841359*.
- Grimm, B. M. and Johnson, R. T. (1990). Review of simple heat release computations. In *SAE Technical Paper 900445*.
- GT-Suite (2007). *GT-Suite Flow Theory Manual*. Gamma Technologies, 7.1 edition.
- Gueezlla, L. and Onder, C. (2009). *Introduction to Modeling and Control of Internal Combustion Engine Systems*. Springer, 2nd edition.
- Guzzella, L. and Onder, C. (Springer-Verlag, Berlin). *Introduction to Modeling and Control of Internal Combustion Engine Systems*. 2004.

- Hanson, R., Kokjohn, S., Splitter, D., and Reitz, R. (2010). An experimental investigation of fuel reactivity controlled pcci combustion in a heavy-duty engine. *SAE International Journal of Engines*, 3(1):700.
- Hanson, R., Kokjohn, S., Splitter, D., and Reitz, R. (2011). Low load investigation of reactivity controlled compression ignition (rci) combustion in a heavy-duty engine. In *SAE Technical Paper 2011-01-0361*.
- He, X., Leslie, A., Halmari, J., Cordaway, A., Maxwell, T., and Parten, M. (2005). Development of real-time control for a hydrogen powered hybrid electric vehicle. In *SAE Technical Paper 2005-01-0023*.
- Hendricks, E., Chevalier, A., M., J., Sorenson, S., Trumpy, D., and Asik, J. (1996). Modeling of the intake manifold filling dynamics. In *SAE Technical Paper 9600037*.
- Heywood, J. B. (1988). *Internal Combustion Engine Fundamentals*. McGraw Hill, New York, USA.
- Hohenberg, G. (1979). Advanced approaches for heat transfer calculations. In *SAE Technical Paper 790825*.
- J, K., SW, P., M, A., RD, R., and K., S. (2009). Experimental investigation of intake condition and group-hole nozzle effects on fuel economy and combustion noise for stoichiometric diesel combustion in a hsd diesel engine. In *SAE Technical Paper 2009-01-1123*.
- Jacobs, T. (2011). Novel opportunities for reduced biodiesel nox and pm without aftertreatment through the use of in-cylinder low-temperature combustion regimes. In *National Biodiesel Boards Conference and Exposition*.

- Jansons, M., Florea, R., Zha, K., Estefanous, F., Florea, E., Taraza, D., Bryzik, W., Henein, N., and Hoogterp, L. (2009). Optical and numerical investigation of pre-injection reactions and their effect on the starting of a diesel engine. In *SAE Technical Paper 2009-01-0648*.
- Jansons, M., Florea, R., Zha, K., and Florea, E. (2011). The combined effect of hcho and c2h4 addition on combustion in an optically accessible diesel engine fueled with jp-8. *SAE International Journal of Engines*, 4(1):2048.
- Jansons, M., Florea, R., Zha, K., and Gingrich, E. (2010a). The effect of hcho addition on combustion in an optically accessible diesel engine fueled with jp-8. *SAE International Journal of Fuels and Lubricants*, 3(2):671.
- Jansons, M., Zha, K., Florea, R., Taraza, D., Henein, N., and Bryzik, W. (2010b). Effect of swirl ratio and wall temperature on pre-injection chemiluminescence during starting of an optical diesel engine. *SAE International Journal of Engines*, 2(2):173–185.
- Jiang, S., Medonza, D., Furukawa, S., and Smith, M. (2007). Design and implementation of an integrated development environment consisting of engine rapid control prototyping and real time vehicle simulation. In *SAE Technical Paper 2007-01-0515*.
- Kim, J. (2009). Experimental investigation of effects of intake pressure and advanced concept nozzles on stoichiometric diesel combustion with a three-way catalyst for emissions reduction. Master's thesis, University of Wisconsin - Madison.
- Kim, J., Reitz, R., and Park, S. (2010). Improvements in the performance and pollutant emissions for stoichiometric diesel combustion engines using a two-spray-angle nozzle. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 224(8):1113–1122.

- Kim, J., Reitz, R. D., Park, S. W., and Sung, K. (2009). Reduction of no_x and co emissions in stoichiometric diesel combustion using a 3-way catalyst. *ASME Conference Proceedings*, 2009(43406):389–397.
- Knight, B., Bittle, J., and Jacobs, T. (2010). Efficiency considerations of later-phased low temperature diesel combustion. In *ASME Technical Paper ICEF2010-35070*.
- Knight, B., Bittle, J., and Jacobs, T. (2011). Characterizing the influence of egr and fuel pressure on the emissions in low temperature diesel combustion. *SAE Technical Paper*, pages 01–1354.
- Kokjohn, S., Hanson, R., Splitter, D., Kaddatz, J., and Reitz, R. (2011a). Fuel reactivity controlled compression ignition (rcci) combustion in light-duty and heavy-duty engines. In *SAE Technical Paper 2011-01-0357*.
- Kokjohn, S., Hanson, R., Splitter, D., and Reitz, R. (2010). Experiments and modeling of dual-fuel hcci and pcci combustion using in-cylinder fuel blending. *SAE International Journal of Engines*, 2(2):24.
- Kokjohn, S., Hanson, R., Splitter, D., and Reitz, R. (2011b). Fuel reactivity controlled compression ignition (rcci): a pathway to controlled high-efficiency clean combustion. *International Journal of Engine Research*, 12(3):209.
- Kokjohn, S. and Reitz, R. (2010). Investigation of charge preparation strategies for controlled premixed charge compression ignition combustion using a variable pressure injection system. *International Journal of Engine Research*, 11(4):257–282.

- Kokjohn, S. L., Swor, T. A., Andrie, M. J., and Reitz, R. D. (2009). Experiments and modeling of adaptive injection strategies (ais) in low emissions diesel engine. In *SAE Technical Paper 2009-01-0127*.
- Kolade, B., Boghosian, M. E., Reddy, P. S., and Gallagher, S. (2003). Development of a general purpose thermal-hydraulic software and its application to fuel injection systems. In *SAE Paper 2003-01-0702*.
- Kothapalli, K. (2011). The gpgpu phenomenon : Understanding its scope, applicability, and its limitations. ICDCN, 2011.
- Kulkarni, R. and Hoekstra, G. (2002). Labview fpga in hardware-in-the-loop simulation applications. Technical report, National Instruments Corporation White Paper.
- Kumar, R. and Zheng, M. (2008). Fuel efficiency improvements of low temperature combustion diesel engines. In *SAE Technical Paper 2008-01-0841*.
- Kumar, R., Zheng, M., Asad, U., and Reader, G. (2007). Heat release based adaptive control to improve low temperature diesel engine combustion. In *SAE Technical Paper 2007-01-0771*.
- Leone, D. M., Dodge, L. G., Shouse, K. R., Grogan, J., and Weeks, R. W. (1997). Model-based control and cylinder-event-based logic for an ultra-low emissions vehicle. In *SAE Technical Paper 970531*.
- Lickly, B., Liu, I., Kim, S., Patel, H. D., Edwards, S. A., and Lee, E. A. (2008). Predictable Programming on a Precision Timed Architecture. In *CASES '08: Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems*, pages 137–146, New York, NY, USA. ACM.

- Liu, I., Viele, M., Wang, G., Andrade, H., and Lee, E. A. (2011). A heterogeneous architecture for evaluating real-time one-dimensional computational fluid dynamics on fpgas. In *submission to FCCM 2012*.
- LM9011 (2009). *LM9011 Electronic Ignition Interface*. Texas Instruments.
- Lumpkin, E. and Gabrick, M. (2006). Hardware/software design and development process. In *SAE Technical Paper 2006-01-0170*.
- Manofsky, L., Vavra, J., Assanis, D., and Babajimopoulos, A. (2011). Bridging the gap between hcci and si: Spark-assisted compression ignition. In *SAE Technical Paper 2011-01-1179*.
- McLean, J. J. and Jacobs, T. (2011). Brake fuel conversion efficiency study in a medium-duty diesel engine using variations in injection timing. In *2011 7th US National Combustion Meeting, Atlanta, Georgia*.
- Mikalsen, R. and Roskilly, A. (2008a). The design and simulation of a two-stroke free-piston compression ignition engine for electrical power generation. *Applied Thermal Engineering*, 28(5-6):589–600.
- Mikalsen, R. and Roskilly, A. (2008b). *A review of free-piston engine history and applications*. Elsevier.
- Moran, R. (2003). *Variable Valvetrain System Technology*. Society of Automotive Engineers, Inc.
- Mueller, R., Hart, M., Truscott, A., Noble, A., Kroetz, G., Eichoff, M., Cavalloni, C., and Gnielka, M. (2000). Combustion-pressure-based engine management system. In *SAE Technical Paper 2000-01-0928*.
- Mller, N. and Isermann, R. (2001). Control of mixture composition using cylinder pressure sensors. In *SAE Technical Paper 2001-01-3328*.

- Norman, K. (2007). The development of a turbocharged race engine application. Master's thesis, Colorado State University.
- Padala, S., Bansal, A., and Ramesh, A. (2008). Studies on an air assisted gasoline direct injection system for a two-stroke engine. In *SAE Technical Paper 2008-28-0048*.
- Parnell-Xilinx, K. (2002). Reconfigurable vehicle. In *SAE Technical Paper 2002-01-0144*.
- Persson, H., Hultqvist, A., Johansson, B., and Remn, A. (2007). Investigation of the early flame development in spark assisted hcci combustion using high speed chemiluminescence imaging. In *SAE Technical Paper 2007-01-0212*.
- Persson, H., Sjholm, J., Kristensson, E., Johansson, B., Richter, B., and Alden, M. (2008). Study of fuel stratification on spark assisted compression ignition (saci) combustion with ethanol using high speed fuel plif. In *SAE Technical Paper 2008-01-2401*.
- Quillen, K. P., Viele, M., and Ciatti, S. A. (2010). Next-cycle and same-cycle cylinder pressure based control of internal combustion engines. *ASME Conference Proceedings*, 2010(49446):635–645.
- Ribbens, W. (2012). *Understanding Automotive Electronics*. Butterworth-Heinemann, 7th edition.
- R.Kaddatz, J. (2011). Experimental investigation of dual-fuel rcci operation in a light-duty engine. Master's thesis, University of Wisconsin - Madison.
- Rohini, B., Das, H., Kavitha, K., and Dhinagar, S. (2005). Development of a low cost engine simulator for hil testing. In *SAE Technical Paper 2005-26-040*.

- Sappok, A., Bromberg, L., Parks, J., and Prikhodko, V. (2010). Loading and regeneration analysis of a diesel particulate filter with a radio frequency-based sensor. In *SAE Technical Paper 2010-01-2126*.
- Schmidt, L., King, J., Stokes, J., Mullineux, J., Ramasamy, C., Nazri Amiruddin, A., Evans, I., Kay, P., Heikal, M., and Begg, S. (2011a). Validation of a cfd model of a hollow-cone spray with gasoline fuel blends. In *SAE 2011 Technical Paper 2011-01-0379*.
- Schmidt, L., Seabrook, J., Stokes, J., Zuhdi, A., Faizan, M., Begg, S., Heikal, M., and King, J. (2011b). Multiple injection strategies for improved combustion stability under stratified part load conditions in a spray guided gasoline direct injection (sgdi) engine. In *SAE Technical Paper 2011-01-1228*.
- Schten, K., Ripley, G., Punater, A., and Erickson, C. (2007). Design of an automotive grade controller for in-cylinder pressure based engine control development. In *SAE Technical Paper 2007-01-0774*.
- Schuette, F., Berneck, D., Eckmann, M., and Kakizaki, S. (2005). Advances in rapid control prototyping - results of a pilot project for engine control. In *SAE Technical Paper 2005-01-1350*.
- Sellnau, M., Sinnamon, J., Hoyer, K., and Husted, H. (2011). Gasoline direct injection compression ignition (gdci)-diesel-like efficiency with low co2 emissions. *SAE International Journal of Engines*, 4(1):2010.
- Sellnau, M., Sinnamon, J., Hoyer, K., and Husted, H. (2012a). Development of full-time gdci for high efficiency and low nox and pm emissions. Aachen Colloquium 2012, Aachen, Germany, October.

- Sellnau, M., Sinnamon, J., Hoyer, K., and Husted, H. (2012b). Full-time gasoline direct injection compression ignition for high efficiency and low nox and pm emissions. *SAE Technical Paper Draft 2012-01-0384*.
- Sellnau, M., Sinnamon, J., Oberdier, L., Dase, C., Viele, M., Quillen, K., Silvestri, J., and Papadimitriou, I. (2009). Development of a practical tool for residual gas estimation in ic engines. *SAE Technical Paper*, pages 2009-01-0695.
- Sellnau, M. C. and Matekunas, F. A. (2000). Cylinder-pressure-based engine control using pressure-ratio-management and low-cost non-intrusive cylinder pressure sensors. In *SAE Technical Paper 2000-01-0932*.
- Sinnamon, J. F. and Sellnau, M. C. (2008). A new technique for residual gas estimation and modeling in engines. In *SAE Technical Paper 2008-01-0093*.
- Smith, W. and Schnore, A. (2004). Towards an rcc-based accelerator for computational fluid dynamics applications. *The journal of Supercomputing*, 30(3):239–261.
- Song, H., Tompkins, B., and Jacobs, T. (2011). An analytical model to estimate nitric oxide emissions for a diesel engine. In *2011 7th US National Combustion Meeting, Atlanta, Georgia*.
- Splitter, D., Hanson, R., Kokjohn, S., and Reitz, R. (2011). Reactivity controlled compression ignition (rci) engine operation at mid and high loads with conventional and alternative fuels. In *SAE Technical Paper 2011-01-0363*.
- Splitter, D., Kokjohn, S., Rein, K., Hanson, R., Sanders, S., and Reitz, R. (2010). An optical investigation of ignition processes in fuel reactivity controlled pcci combustion. *SAE International Journal of Engines*, 3(1):142.

- Sprich, T. K. (2011). Determination of the Pressures within an Injector - Experimentation and Model Validation. Master's thesis, University of the Witwatersrand, Johannesburg, Gauteng Province, Republic of South Africa.
- Staples, L. R. (2008). An experimental investigation into diesel engine size-scaling parameters. Master's thesis, University of Wisconsin - Madison.
- Staples, L. R., Reitz, R., and Carl Hergart, C. (2009). An experimental investigation into diesel engine size-scaling parameters. In *SAE Technical Paper 2009-01-1124*.
- Stobart, R., Guo, X., Bartley, G., and Stacey, S. (2011). Exploring the value of open source in si engine control. In *SAE Technical Paper 2011-01-0702*.
- Stone, R. (1999). *Introduction to Internal Combustion Engines*. SAE International.
- Stovall, C. W. (2008). Utilization of the split engine concept for lean nox adsorber regeneration. Master's thesis, Colorado State University.
- Subramanian, S. and Ciatti, S. A. (2011). Low cetane fuels in compression ignition engines to achieve ltc. In *ASME Conference Proceedings*. ASME. ICEF2011-60014.
- Sung, K., Kim, J., and Reitz, R. (2009). Experimental study of pollutant emission reduction for near-stoichiometric diesel combustion in a three-way catalyst. *International Journal of Engine Research*, 10(5):349–357.
- Swor, T. (2009). Experimental investigation of adaptive injection strategies through low and high pressure split injections. Master's thesis, University of Wisconsin - Madison.
- Swor, T., Kokjohn, S., Andrie, M., and Reitz, R. (2010). Improving diesel engine performance using low and high pressure split injections for single heat release and two-stage combustion. In *SAE Technical Paper 2010-01-0340*.

- Szybist, J., A.D. Youngquist, T. B., Storey, J., Moore, W., Foster, M., and Confer, K. (2011). Ethanol blends and engine operating strategy effects on light-duty spark-ignition engine particle emissions. *Energy & Fuels*, 25(11):4977–4985.
- Szybist, J., Foster, M., Moore, W., Confer, K., Youngquist, A., and Wagner, R. (2010a). Investigation of knock limited compression ratio of ethanol gasoline blends. In *SAE Technical Paper 2010-01-0619*.
- Szybist, J., Nafziger, E., and Weall, A. (2010b). Load expansion of stoichiometric hcci using spark assist and hydraulic valve actuation. *SAE International Journal of Engines*, 3(2):244–258.
- Tan, Y., Zheng, M., Reader, G., Han, X., and Wang, M. (2009). An enabling study of diesel low temperature combustion via adaptive control. *SAE International Journal Fuels Lubrication*, 2(1):750–763.
- Tat, M. E. and Gerpen, J. H. V. (2003). Measurement of biodiesel speed of sound and its impact on injection timing. Technical report, Department of Mechanical Engineering, Iowa State University. Prepared under NREL subcontract ACG-8-18066-01 for the National Renewable Energy Laboratory.
- Tess, M., Lee, C.-W., and Reitz, R. (2011). Diesel engine combustion size scaling at medium load without egr. In *SAE Technical Paper 2011-01-1384*.
- Tompkins, B. and Jacobs, T. (2011). Low temperature combustion with biodiesel. In *National Biodiesel Boards Conference and Exposition*.
- Tompkins, B., Song, H., and Jacobs, T. (2011). Particulate matter emissions from late injection high egr low temperature diesel combustion. In *2011 7th US National Combustion Meeting, Atlanta, Georgia*.

- Trajkovic, S., Milosavljevic, A., Tunestl, P., and Johansson, B. (2006). Fpga controlled pneumatic variable valve actuation. In *SAE Technical Paper 2006-01-0041*.
- Trajkovic, S., Tunestl, P., and Johansson, B. (2008). Investigation of different valve geometries and valve timing strategies and their effect on regenerative efficiency for a pneumatic hybrid with variable valve actuation. *SAE International Journal Fuels Lubrication*, 1(1):1206–1223.
- Trajkovic, S., Tunestl, P., Johansson, B., Carlson, U., and A., H. (2007). Introductory study of variable valve actuation for pneumatic hybridization. In *SAE Technical Paper 2007-01-0288*.
- Trushard, J. (2010). Bringing FPGA Design to Application Domain Experts. Keynote for The 2010 International Conference on Field-Programmable Technology (FPT'10).
- Tsai, G., Wu, Y., Chen, B., and Chuang, H. (2003). Rapid prototyping ecu of a si engine with fuel injection and ignition control. In *SAE Technical Paper 2004-01-0419*.
- Turns, S. R. (2000). *An Introduction to Combustion*. McGraw Hill, Boston, USA.
- Van Blarigan, P. (2002). Advanced internal combustion electrical generator. Technical report.
- Viele, M. and Quillen, K. P. (2009). An analysis of next cycle control based on cylinder pressure derived calculations. In *ASME Technical Paper ICEF2009-14109*.
- Viele, M., Quillen, K. P., and Ciatti, S. A. (2010). Next-cycle and same-cycle cylinder pressure based control of internal combustion engines. *ASME Conference Proceedings*, 2010(49446):635–645.

- Viele, M., Shorey, E. E., Dase, C. G., Hoose, K. V., and Light, K. H. (2011). A novel approach to free-piston engine control using an fpga based control system. *ASME Conference Proceedings*, 2011(60225).
- Viele, M., Stein, L., Gillespie, M., and Hoekstra, G. (2004). A pc and fpga hybrid approach to hardware-in-the-loop simulation. In *SAE Technical Paper 2004-01-0904*.
- Wagner, R., Curran, S., Hanson, R., Barone, T., Kokjohn, S., and Reitz, R. (2011). Addressing the challenges of rcci operation on a light-duty multi-cylinder engine. In *Directions in Engine-Efficiency and Emissions Research (DEER) Conference*.
- Wang, J. and Sarlashkar, J. (2007). Engine crankshaft position tracking algorithms applicable for given arbitrary cam- and crank-shaft position signal patterns. In *SAE Technical Paper 2007-01-1597*.
- Weall, A. and Szybist, J. (2011). The effects of fuel characteristics on stoichiometric spark-assisted hcc. In *ASME Technical Paper ICEF2011-60122*.
- Wilhelmsson, C., Tunestal, P., Widd, B., and Johansson, R. (2009). A physical two-zone nox model intended for embedded implementation. In *SAE Technical Paper 2009-01-1509*.
- Winward, E., Deng, J., and Stobart, R. (2010). Innovations in experimental techniques for the development of fuel path control in diesel engines. *SAE International Journal of Fuels and Lubricants*, 3(1):594.
- Woschni, G. (1967). A universally applicable equation for the instantaneous heat transfer coefficient in the internal combustion engine. In *SAE Technical Paper 670931*.
- Wylie, E. B. and Streeter, V. L. (1978). *Fluid transients*. McGraw-Hill.

- Yaovaja, K., Chatlatanagulchai, W., Wannatong, K., and Rhienprayoon., S. (2011). Air-path control of diesel-dual-premixed-charged-compression-ignition (df-pcci) engine by using supervisory fuzzy control system. In *TSME International Conference on Mechanical Engineering*.
- Ylvisaker, B., Essen, B. V., and Ebeling, C. (2006). A type architecture for hybrid micro-parallel computers. *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on*, 0:99–110.
- Yu, Q., Neyret, F., Bruneton, E., and Holzschuch, N. (2009). Scalable real-time animation of rivers. In *Computer Graphics Forum*, volume 28, pages 239–248. Wiley Online Library.
- Yu, X., Zha, K., Florea, R., and Jansons, M. (2012). Comparison of in-cylinder soot evolution in an optically accessible engine fueled with jp-8 and ulsd. SAE Technical Paper SAE Paper 12PFL-072.
- Yun, H. and Sellnau, M. (2008). Development of premixed low-temperature diesel combustion in a hsd diesel engine. In *SAE Technical Paper 2008-01-0639*.
- Zha, K., Florea, R., and Jansons, M. (2010). Comparison of soot evolution using high-speed cmos color camera and two-color thermometry in an optical diesel engine fueled with b20 biodiesel blend and ultra-low sulfur diesel. *ASME Conference Proceedings*, 2010(ICEF2011-60146).
- Zha, K., Florea, R., and Jansons, M. (2011). Soot evolution with cyclic crank-angle-resolved two-color thermometry in an optical diesel engine fueled with biodiesel blend and ulsd. *ASME Journal of Engineering for Gas Turbines and Power*, GTP-11-1388.

- Zha, K. and Jansons, M. (2011). Novel implementation of two-color soot temperature measurement in optical diesel engine with high-speed cmos digital color camera. In *ICARAME 2011 International Conference on Advanced Research and Applications in Mechanical Engineering*.
- Zha, K., Yu, X., Florea, R., and Jansons, M. (2012). Impact of biodiesel blends on in-cylinder soot temperature and concentrations in a small-bore optical diesel engine. SAE Technical Paper Draft 12PFL-0176.
- Zhao, H. and Ladommatos, N. (2001). *Engine Combustion Instrumentation and Diagnostics*. SAE International.
- Zheng, M., Kumar, R., and Reader, G. (2006). Adaptive fuel injection tests to extend egr limits on diesel engines. In *SAE Technical Paper 2006-01-3426*.
- Zheng, M., Kumar, R., Tan, Y., and Reader, G. (2009a). Heat release pattern diagnostics to improve diesel low temperature combustion. *SAE International Journal Fuels Lubrication*, 1(1):1242–1258.
- Zheng, M., Tan, Y., Mulenga, M., and Wang, M. (2007). Thermal efficiency analyses of diesel low temperature combustion cycles. In *SAE Technical Paper 2007-01-4019*.
- Zheng, M., Tan, Y., Reader, G., Asad, U., Han, X., and Wang, M. (2009b). Prompt heat release analysis to improve diesel low temperature combustion. In *SAE Technical Paper 2009-01-1883*.
- Zheng, M., Wang, M., Reader, G., Mulenga, M., and Tjong, J. (2009c). An improvement on low temperature combustion in neat biodiesel engine cycles. *SAE International Journal Fuels Lubrication*, 1(1):1120–1132.