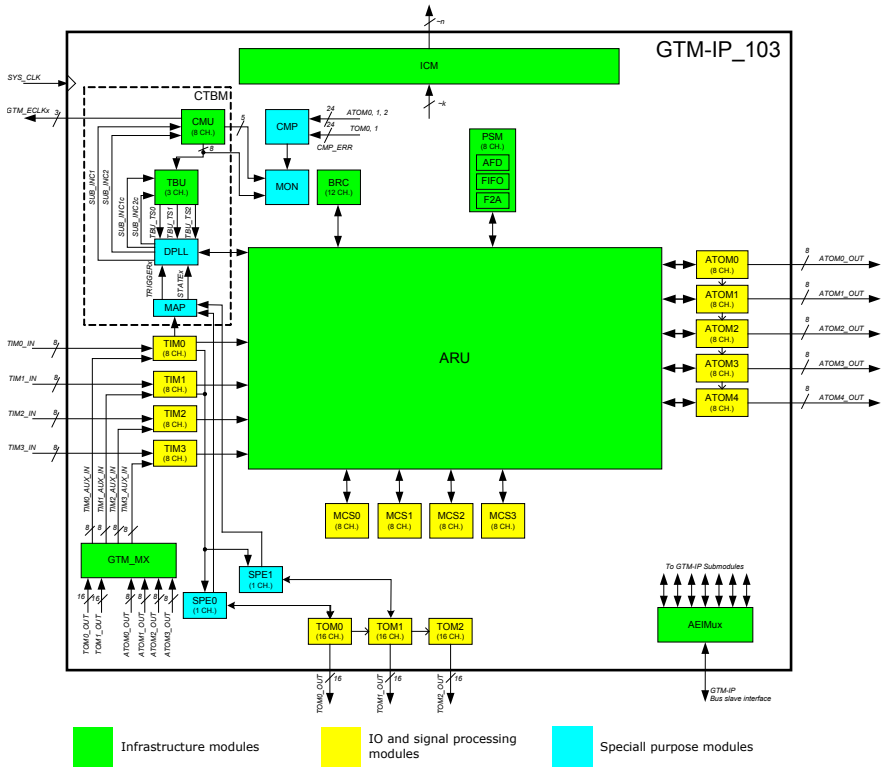
CHALMERS



Exploring the Generic Timer Module's Feasibility for Truck Powertrain Control

Master of Science Thesis in Embedded Electronic System Design

DAN LARSSON

JONAS HEMLIN

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Gothenburg, Sweden, June 2015

Exploring the Generic Timer Module's Feasibility for Truck Powertrain Control

DAN LARSSON
JONAS HEMLIN

Supervisor: Magnus Stålesjö, Volvo Group Trucks Technology
Supervisor: Sally A. McKee, Department of Computer Science and Engineering
Examiner: Per Larsson Edefors, Department of Computer Science and Engineering

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Gothenburg
Telephone +46 (0)31-772 1000

Cover: Overview of the Generic Timer Module IP developed by Robert Bosch GmbH.

Typeset in LaTeX
Department of Computer Science and Engineering
Gothenburg, Sweden, June 2015

Abstract

Timer modules are employed in ECUs to perform real-time signal processing in truck powertrains. The architecture of these modules usually has either a hardware centric or a software centric approach. In 2014 Robert Bosch GmbH released the Generic Timer Module (GTM), which has an architecture that combines both approaches, in the form of signal processing hardware modules and small embedded RISC cores. As it is fairly new on the market no publicly available evaluations of the GTM exist. This thesis examines the GTM's feasibility to perform all timer-module tasks in Volvo Group Truck Technology powertrain ECUs. We do this by developing and implementing two proof-of-concept designs for the most demanding of these tasks, the control of fuel injection. With the results from tests performed on the designs we show that the GTM, with the help of DMA, can perform the tasks at hand. Volvo's existing ECUs are used as the reference system which utilize another timer module, the enhanced Time Processing Unit, against which the GTM is compared.

Keywords: timer module, Generic Timer Module, enhanced Time Processing Unit, fuel injection, current waveform, angle clock.

Acknowledgements

We would like to take the opportunity to thank our supervisor at Volvo Group Truck Technology, Magnus Stålesjö, for sharing his expertise in the area and his time invested during this thesis. Also, we wish to express our gratitude to our advisor at Chalmers University of Technology, Prof. Sally A. McKee, who especially supported us in the art of technical writing. Additionally, we thank Alistair Low, Johan Thorsen, and Agne Holmqvist for their support regarding the hardware.

Dan Larsson and Jonas Hemlin, Gothenburg, Sweden, June 2015

Contents

List of Figures v

List of Tables vii

1 Introduction 1
- 1.1 Aim . . . 2
- 1.2 Scope . . . 2
- 1.3 Thesis structure . . . 3

2 Fuel injection control systems 4
- 2.1 Fuel injectors . . . 4
- 2.2 System control . . . 5
  - 2.2.1 Engine rotary position feedback . . . 6
  - 2.2.2 Injectors . . . 7
  - 2.2.3 Fuel pressure . . . 9

3 Architectures 10
- 3.1 Enhanced Time Processing Unit . . . 10
- 3.2 Enhanced Modular Input/Output Subsystem . . . 13
- 3.3 Generic Timer Module . . . 14
  - 3.3.1 Modules . . . 14
  - 3.3.2 Device configurations . . . 19
  - 3.3.3 GTM reference model . . . 19
  - 3.3.4 GTM generation 3 . . . 20
- 3.4 AURIX . . . 20

4 Tasks performed by timer modules in the EMS 22
- 4.1 Fuel injector control system . . . 22
  - 4.1.1 Angle clock . . . 22
  - 4.1.2 Needle valves and spill valves . . . 23
  - 4.1.3 Remaining rail valves . . . 25
- 4.2 Other tasks . . . 26

5 Handling of the EMS' timer-module tasks using the GTM 28
- 5.1 Requirements . . . 28
- 5.2 Angle clock . . . 30
  - 5.2.1 Micro tick generation . . . 31

5.2.2 Engine speed and base angle calculation . . . 31
5.3 Needle valves and spill valves . . . 32
5.3.1 Current waveform design . . . 33
5.3.2 Input signal capture . . . 35
5.3.3 Processing . . . 35
5.3.4 Output signal generation . . . 37
5.3.5 Worst case response times . . . 38
5.4 Remaining rail valves . . . 40
5.5 Other tasks . . . 41
5.6 Prototype . . . 41
5.6.1 Hardware . . . 41
5.6.2 Testing . . . 42

6 Results **44**
6.1 Angle clock . . . 44
6.2 Needle valves and spill valves . . . 45
6.3 Resource utilization . . . 48

7 Discussion **50**
7.1 GTM architecture . . . 50
7.2 Angle clock . . . 51
7.3 Needle valves and spill valves . . . 52
7.4 Ethics . . . 54
7.5 Future work . . . 55

8 Conclusion **56**

Bibliography **57**

List of Figures

2.1 Illustration of the fuel delivery system's components in a six-cylinder Volvo diesel engine. Figure taken from a Volvo internal document [1], with modified number coding. 5
2.2 Illustration of a very simple fuel injector and its basic parts. 6
2.3 Typical positions of the cam shaft, crank shaft, and flywheel in an engine. On the flywheel the slits used for rotary position feedback can be seen. Partly modified figure originally taken from a Volvo internal document [1]. 7
2.4 Example of a current waveform driving a solenoid fuel injector. 8

3.1 High level block diagram of the enhanced Time Processing Unit in dual-engine configuration. Figure taken from Freescale [2]. 11
3.2 Block diagram of an enhanced Time Processing Unit engine. Figure taken from Freescale [2]. 12
3.3 Overview of the Generic Timer Module architecture, device 103 and IP version 1.5.5.1. Figure taken from Robert Bosch GmbH [3] with added color coding. 15

4.1 Example of the logic signals generated from crank and cam sensors around the TDC event of cylinder four in a six-cylinder engine. 23
4.2 Typical waveform for the Delphi F2 fuel injector. The waveform is generated by driving dummy loads, resulting in that the waveform characteristics are affected and the waveform is not within the required boundaries. Figure taken from a Volvo internal document [4]. . 24
4.3 Simplified illustration of the drive circuit for one injector bank. 25
4.4 Example of the current waveform and drive stage signals generated by the PHPWM function. 26

5.1 Current waveform parameters for the requirements of a Delphi F2 fuel injector. 28
5.2 Block diagram of proposed design for engine rotary position feedback using the GTM 30
5.3 Block diagram of the GTM-only design for needle valve and spill valve control using the GTM 33
5.4 Block diagram of the Hybrid design for needle valve and spill valve control using the GTM 34
5.5 Overview of the hardware setup used for development and testing. . . 42

6.1 Capture of three injector pulses that have been scheduled on the crank slit zero interrupt to start 45 degrees later. 45
6.2 Capture of a current waveform generated by the GTM-only solution with cursors indicating the vital points. 46
6.3 Capture of a current waveform generated by the Hybrid solution with cursors indicating the vital points. 46
6.4 Zoom-in on Figure 6.2 showing the latency introduced by the GTM-only solution. 47

List of Tables

3.1 Two device configurations of the GTM with the number of modules (channels) they contain and their respective ARU RTT. 19

5.1 Typical values for parameters in Figure 5.1 given by Delphi [5]. . . . 29
5.2 Measured worst case times for how long it takes to reach the boundaries in the different operation sections of a F2 fuel injector. 29
5.3 Utilization of MCS channels in one MCS instance for one bank. . . . 36
5.4 WCRT for the modules used in the presented designs, assumed that all digital modules operate at the maximum clock frequency of 100 MHz. 38
5.5 The number of instruction cycles present in the critical path during the four interesting WCRTs. 39

6.1 Description of the different signals shown in Figure 6.1, 6.2, 6.4, and 6.3. 44
6.2 Measured worst case latencies on both designs and the estimated WCRTs. 47
6.3 Configuration values which differentiates between the designs. 48
6.4 GTM-resource utilization for our two designs on a GTM device 103. The number of available channels is placed within parenthesis after the module acronym. 49

1
Introduction

The Electronic Control Units (ECUs) controlling a modern truck's powertrain perform many tasks with strict real-time requirements. Reliably performing these tasks requires deterministic timing behavior which a general purpose processor lacks. Additionally, some tasks demand a higher throughput than a general purpose processor can commonly sustain [6]. Thus, the microcontroller within the ECU employs *timer modules* as peripheral circuitry to aid the main processor in performing the time-critical portions of tasks.

Timer modules can be divided into two main approaches: hardware-centric and software-centric. The hardware-centric approach consists of signal processing Input Output (IO) units controlled by a Central Processing Unit (CPU). The software-centric approach has simple IO units, but it also includes a processing core that commonly uses a specialized instruction set optimized for timing operations. The processing core allows for a more versatile operation, making it more independent of the host CPU than the hardware-centric approach. In the latter the host CPU has to serve many interrupts while controlling the module. The disadvantage of the software-centric approach is that it has a lower time resolution.

The latest powertrain ECUs developed by Volvo Group Trucks Technology[1] include two timer modules to aid the main CPU: the hardware-centric enhanced Modular Input/Output Subsystem (eMIOS) [7] and the software-centric enhanced Time Processing Unit (eTPU) [2], both developed by Freescale Semiconductor[2]. Volvo is in early stages of evaluating microcontroller family alternatives for the next generation of the ECUs. Several of these microcontrollers include a timer module called Generic Timer Module (GTM), which is an Intellectual Property (IP)-core developed by Robert Bosch GmbH[3]. The GTM aims to combine the two approaches using an interesting architecture and it has units with signal processing capabilities as well as a processing core. This approach together with a versatile design give the GTM potential to replace both the eMIOS and the eTPU in the next ECU generation. If it is possible to replace the currently used timer modules, development is simplified as expertise then can be focused on one timer module. Additionally, relying on the GTM as timer module is one step towards microcontroller-platform independence. This as the GTM is not tied to one microcontroller company thanks to that it is distributed as an IP-core. The GTM's potential has been identified by two major microcontroller suppliers to the automotive industry (Infineon Technologies[4] and

[1]Volvo Group Trucks Technology is hereafter referred to as Volvo.
[2]Freescale Semiconductor is hereafter referred to as Freescale.
[3]Robert Bosch GmbH is hereafter referred to as Bosch.
[4]Infineon Technologies is hereafter referred to as Infineon.

Freescale) which includes a GTM in their state-of-the-art microcontrollers.

The GTM was released in a chip in early 2014 [8]. No evaluation reports have been released presenting the GTM's capabilities, and Volvo has not yet evaluated the timer module. Other companies may have evaluated the GTM, but if so they are likely bound by disclosure agreements or do not want to publish such information. This thesis is thus the first publicly available report evaluating the GTM's capabilities.

1.1 Aim

This thesis assesses the suitability of the GTM to perform all timer-module tasks in the current generation of Volvo powertrain ECUs. The goals are to present a proof-of-concept design of how the GTM can be employed to perform the most demanding tasks currently performed by the eMIOS and the eTPU, and to produce a proof-of-concept prototype that shows that the proposed design works as intended. Also, we evaluate if the GTM has enough resources to perform all timer-module tasks for the ECUs.

1.2 Scope

We limit the proposed design to the most demanding tasks, as it is not possible to evaluate the GTM's compatibility with all tasks performed by the timing modules on all ECUs within the project timespan. We choose to focus on fuel injection, because those tasks are the most demanding with respect to response time, processing time, and resource usage. Also, most other tasks can be seen as subsets of the fuel injection tasks in terms of functionality and all remaining tasks are even less demanding. As a result, verifying that the GTM can perform fuel injection gives a strong indication that the GTM can perform all tasks executed by the timer circuits.

Similarly, we limit the evaluation of resource usage to cover only one ECU. Volvo employ two ECUs in their powertrain, the Engine Management System (EMS) and the Aftertreatment Control Module (ACM). Due to the distribution of tasks between the ECUs the evaluation only needs to cover the EMS; the ACM performs fewer and less complex tasks. Also, the EMS performs the tasks related to fuel injection. Taking the task distribution to consideration, verifying that the GTM has resources to serve all timer tasks executed in the EMS also verifies that the GTM has enough resources to also replace the timer modules in the ACM.

In this thesis we use the current Volvo powertrain ECUs as our reference point. We thus assume that the tasks executed by the timer modules in the current ECUs will stay the same in the next generation of ECUs, an assumption which may not be accurate. The assumption was made as the tasks for the next generation ECUs are not yet available.

Two of the microcontroller families including the GTM was available at the start of this thesis: the AURIX family developed by Infineon, and the Qorivva family developed by Freescale. Within the Qorivva family the model Matterhorn [9] includes a GTM device, whereas all models in the AURIX family includes a GTM. As

this thesis evaluates the GTM, the choice of microcontroller affects the analysis to a small degree, but a microcontroller from the AURIX family was chosen as it was of greater interest for Volvo.

1.3 Thesis structure

In the next section we introduce the fundamentals of a fuel injection control system together with its basic components. Then we continue to describe the architecture of relevant timer modules as well as the architecture of the microcontroller used in the prototype. After that we cover how Volvo's current system performs the tasks that we have created designs for. In the section after we present our proposed designs, some details about the implementation and how input stimuli were generated. We then continue with showing our results in the following section. Lastly, a discussion and a conclusion ties it all together.

2

Fuel injection control systems

In fuel injection systems of combustion engines, the fuel is pressurized and delivered to the combustion chamber through an injector. The combusted air-fuel mixture is prepared by multiple injection events, where the start of injection and the quantity of fuel injected is controlled by an actuator valve. Additionally, the injection events take place at a predefined fuel pressure and the injection process starts when the piston is in a specific position. An ECU manages injection by controlling the fuel pressure, deriving the piston's position and operating the injector.

The fuel pressure needed for injection is produced by one or several high pressure pumps. There exist several possible configurations for how the pump(s) are connected. One configuration is to have one pump per injector. This pump can either be incorporated in the injector, called Unit Injector System (UIS), or the pump and injector can be separated, called Unit Pump System (UPS). Another configuration is the Common-Rail (CR) system, which means that all injectors are connected to a rail and share the same pressurized fuel. Normally, only one pump is employed to pressurize the rail, but designs with several pumps exist. One such solution called *F2* from Delphi has injectors with integrated pumps, where all pumps together pressure the common rail [10].

In Figure 2.1 an F2 fuel injection system in a Volvo engine is illustrated together with all parts leading up to it, thus constituting a whole fuel delivery system. It begins with the fuel tank (1) from which a low-pressure fuel pump (2) sucks fuel through a filter (3). The fuel then passes through piping to the F2 systems high-pressure fuel pumps (4) that pressurizes the common rail (5). All injectors (5) then take fuel from this rail. For safety reasons there is a dump valve (7) that can release the pressure in the rail when needed. The dump valve, the high-pressure pumps, and the injectors are all controlled using an Volvo developed ECU (8), which is cooled by incoming fuel.

2.1 Fuel injectors

Figure 2.2 illustrates a very simple fuel injector. Pressurized fuel sprays through the holes in the nozzle (1) into the combustion chamber when the needle (2) is lifted off its seat. The lift is controlled by the electromagnetic field that is created when current flows through the solenoid coil (3), which causes the metallic core (4) to move linearly through the coil. The needle is attached to the core, and so it lifts when the core starts to elevate. When current stops flowing, the core and the needle are pushed down by the spring (5), which ends the injection event. More advanced

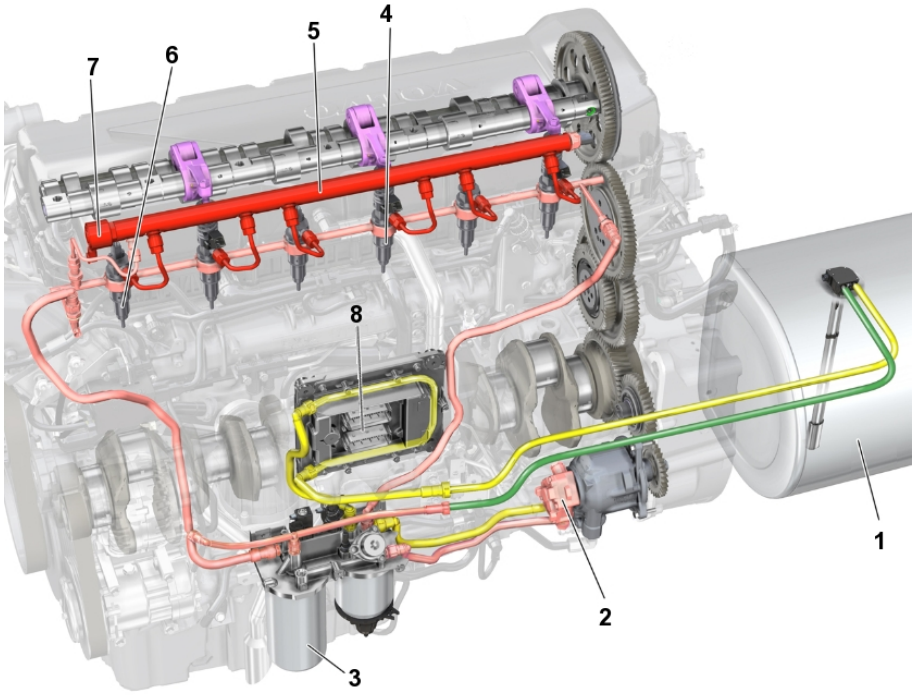


Figure 2.1: Illustration of the fuel delivery system's components in a six-cylinder Volvo diesel engine. Figure taken from a Volvo internal document [1], with modified number coding.

designs may decouple the core from the needle and use the solenoid actuator as a hydraulic servo for moving the needle [11]. Such designs allow lower operating currents and have the benefit of reducing mechanical vibrations transferred from the core to the needle. As the solenoid controls the needle movement that affects the fuel influx, the described injector can be called a *needle valve*.

Fuel injectors can be based on piezoelectric elements instead of solenoids, but this thesis focuses on the latter: injectors based on piezoelectric elements are not used in Volvo truck engines, and thus they are out of scope.

An *injection event* comprises a *peak phase* and a *hold phase*. During the peak phase, the core and needle reach their elevated positions, and during the hold phase they maintain these. It is desirable that the elevated position is reached quickly during the peak phase. This requires a high current flow through the solenoid coil. The hold phase uses a lower current level, as the core only need to keep its position.

2.2 System control

The system controls the injectors and the fuel pressure. To control the latter, the system must know the current pressure, and to operate the injectors the engine's rotary position must be known. In this section we first describe how the rotary position is acquired and then we cover how the injectors are controlled. Finally, we describe how the fuel pressure is regulated.

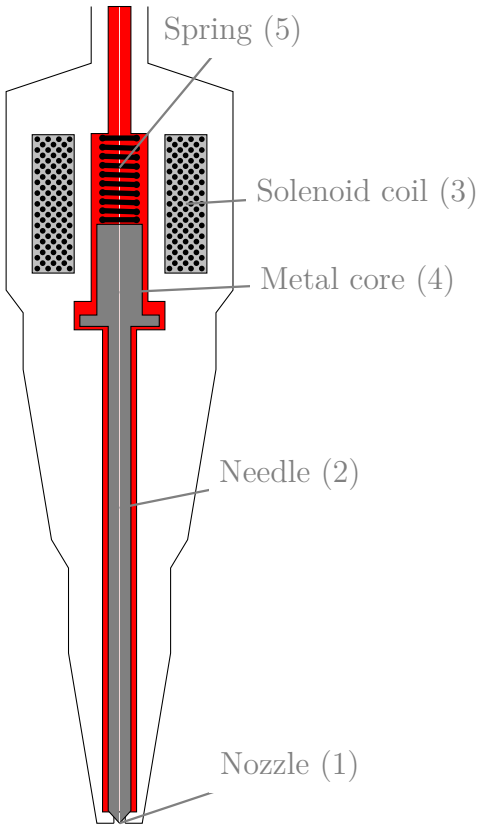


Figure 2.2: Illustration of a very simple fuel injector and its basic parts.

2.2.1 Engine rotary position feedback

For the ECU to be able to determine when to start an injection event a high-resolution feedback of the piston's position is needed, information that is given from the engine's rotary position. Low-resolution feedback can be gained by a sensor that detects slits evenly distributed around the flywheel attached to the end of the crank shaft. Figure 2.3 shows the position of the crank shaft (1) and flywheel (2) in an typical engine. To increase the resolution, the slit period can be divided into smaller steps, called *micro ticks*. As the micro ticks need to be generated before the period of the current slit is known, the period has to be estimated. Also, the engine rotation speed is rarely fixed, which means that the micro tick generation has to handle slit period estimations that are inaccurate due to deceleration and acceleration. Deceleration is usually managed by pausing the micro tick generation when the total number of ticks for one slit period has been generated, and acceleration is managed by generating the remaining micro ticks from the latest slit period during the next period.

The micro ticks can be used as a clock signal for a counter, then called an *angle clock*. By looking at the number of micro ticks that have passed, it is possible to know with high precision where in the cycle the engine currently is. At startup when the starter begins to turn the engine, it is in an unknown position. Thus, before injection of fuel can start, the angle clock needs to be synchronized. Part of the synchronization is performed by detecting periodically missing slits on the

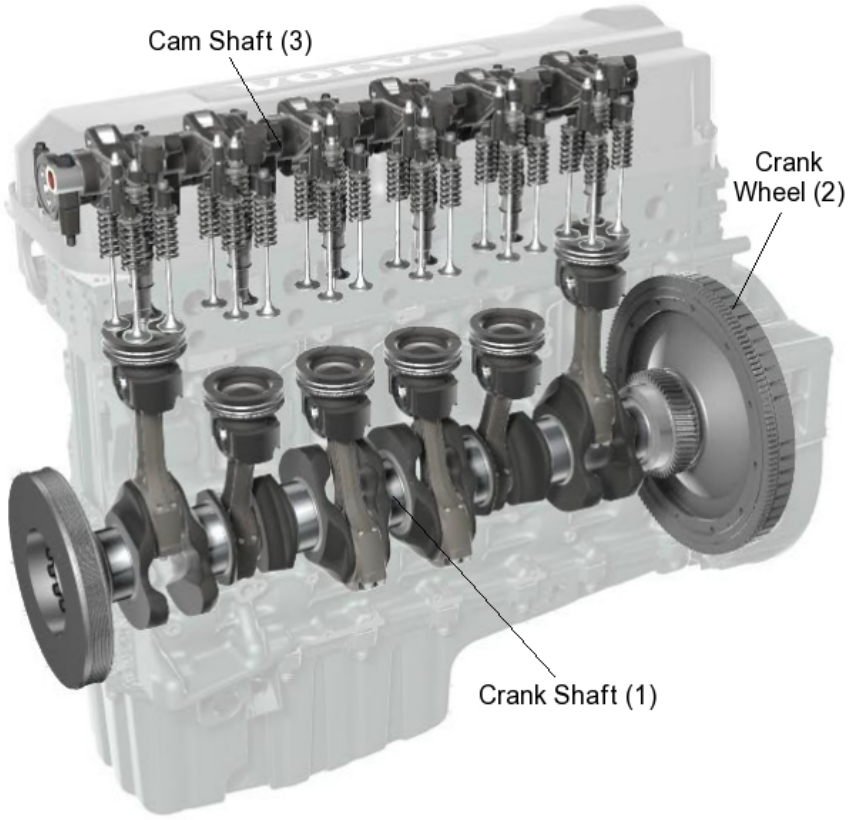


Figure 2.3: Typical positions of the cam shaft, crank shaft, and flywheel in an engine. On the flywheel the slits used for rotary position feedback can be seen. Partly modified figure originally taken from a Volvo internal document [1].

flywheel. However, as one engine cycle consists of two full crank shaft revolutions in a four-stroke engine, an additional rotational reference is needed to synchronize with the engine's rotary position. The additional reference is normally provided by a sensor that detects teeth distributed around a wheel attached to the cam shaft, a shaft that only does one full revolution for one engine cycle. Figure 2.3 shows a typical position of the cam shaft (3) in an engine. The cam wheel commonly has far fewer teeth than the crank shaft has slits. When a special pattern in the teeth is detected, the engine's rotary position is known and the angle clock can be fully synchronized.

2.2.2 Injectors

The needle valve is controlled by the ECU. The ECU controls the current flowing through the needle valve solenoid by feeding it voltages, which is usually done through switching Field Effect Transistors (FETs) connected to the injector's high and low sides. On the low side, the FET breaks the injector's ground connection. For the high side, two FETs are commonly employed, enabling the ECU to apply either a low voltage (normally *battery* voltage) or a higher voltage (usually called *boost* voltage). The main advantage of the boost voltage is that it enables a shorter current rise time, resulting in a faster opening of the injector. To keep a steady

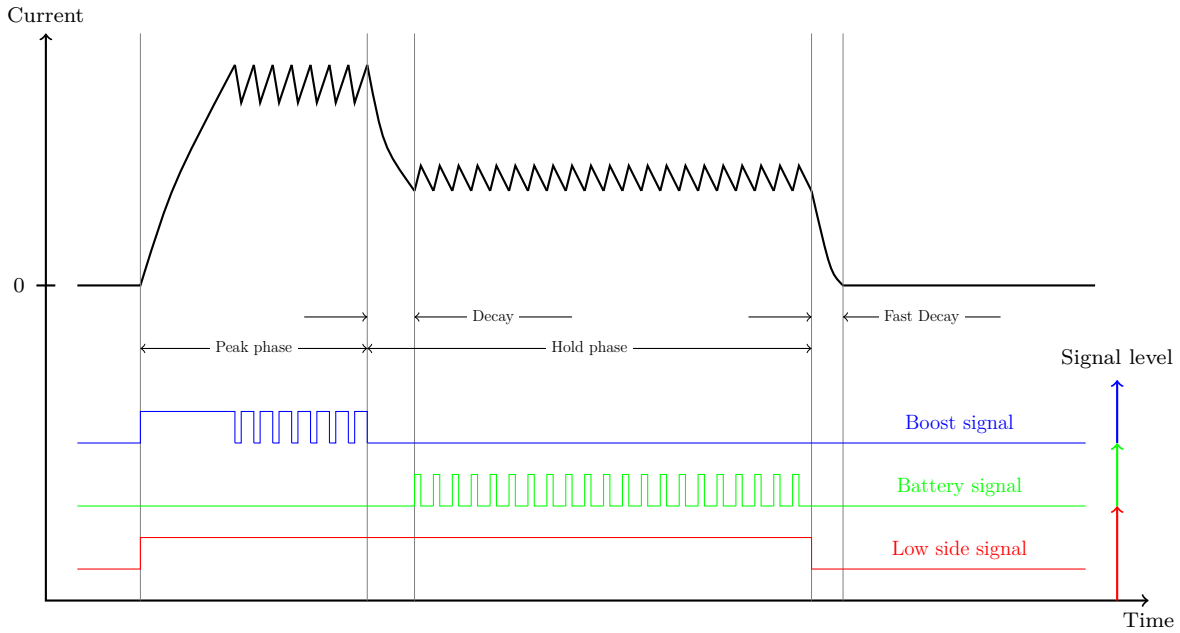


Figure 2.4: Example of a current waveform driving a solenoid fuel injector.

current flowing through the solenoid the high side voltage is rapidly switched on and off, where the on time is called a *chop*. This technique gives the best result if current feedback can be used. If no current feedback is available, Pulse Width Modulation (PWM) can be used instead to keep a somewhat steady current level. When switching the high side off completely the current level is dropped and if a more rapid decay is desired the low side can be switched off as well. Figure 2.4 illustrates a current waveform for a solenoid injector along with the ECU's output signals. The duration of the time where current is flowing through the solenoid is called an *envelope*.

It takes some time for the electromagnetic field to move the metallic core to its elevated position. When the core is close to the desired level, less power is needed to sustain the electromagnetic field due to that the inductance changes as the core moves. As a consequence, the solenoid requires that voltage is applied a longer period or more often for the current to rise the same amount during the first part of the peak phase than during the later part of the peak phase. This dynamic behavior has to be considered when designing the appearance of a current waveform.

The goal of the fuel injection is to create a homogeneous air-fuel mixture with a desired air-fuel ratio. As the mixture properties affect emission levels, engine performance, fuel consumption and combustion noise [11] (p60), it is important for the current waveform to be precisely controlled to deterministically produce a mixture with the needed properties [12]. One effective method to aid fuel atomization is to split one injection into several injection events [13][1]. However, the injected fuel quantity is non-linear when the injection event's pulse width is shorter than a certain threshold [14]. A lower minimum injection quantity within the linear area enables

[1] In Japanese, statement claimed by [12]

an increased number of injection events, which makes it easier to reach a higher fuel atomization. On the other hand, if the ECU is unable to control the current waveform with the required accuracy, the minimum injection quantity is increased, which worsens the fuel atomization due to the fact that injection events have to be longer and fewer in number.

2.2.3 Fuel pressure

For all CR configurations there are one or several solenoid valves used to regulate the injection pressure. In a UIS or multi-pump CR system such valves are called *spill valves*. They are controlled in the same manner as an injector needle valve and is used to inhibit the spill outlet of the pump, allowing a pressure to be built up. All CR systems have a rail-pressure sensor for feedback, and for safety reasons they have a valve that can dump the rail pressure into the fuel tank, called *dump valve*. In the single-pump CR system an additional valve, called *inlet valve*, is employed to manage the influx to the fuel pump. The dump valve and inlet valve do not have to be as precisely controlled as an injector needle valve since their function is less demanding. Even so, they are preferably controlled using current feedback, but necessarily not in the same way as a spill valve.

3
Architectures

This chapter provides basic information about the different hardware architectures relevant to the thesis. Firstly, the eTPU and the eMIOS are covered to give a background to what the GTM is supposed to replace. Secondly, we present the GTM architecture together with information about device configurations and upcoming versions. Finally, we introduce AURIX, which is the microcontroller architecture used in the prototype.

3.1 Enhanced Time Processing Unit

The eTPU is a co-processor that has high resolution timing capabilities. This is mainly achieved by a short latency from the occurrence of an event to the start of event servicing, processing capabilities that reduce the need of intervention from the CPU, and that each function has dedicated I/O units and timer circuits. Due to these features it can perform waveform generation based on real-time input events without the CPU's intervention, which makes it suitable for injector control.

Figure 3.1 present the block diagram of an eTPU dual-core configuration. The processing core's of the eTPU, called *engines*, employ RISC and have an instruction set that include instructions for handling and processing time events. The engines fetch instructions from the Shared Code Memory (SCM) and have access to the Shared Parameter RAM (SPRAM), where application data and parameters reside. The CPU can access an engine's registers, the SPRAM, and the SCM. An eTPU engine comprises of two *time base* counters with angle clock logic, 32 timer channels, a scheduler, a microengine and a host interface. The following paragraphs give an introduction to the eTPU engines' parts, which are illustrated in Figure 3.2.

Time bases. Two 24-bit time base counters are shared by all timer channels and are used to produce timed events. Time base 1 can be clocked either externally by a signal passed through a digital filter or internally by the system clock divided, by a factor of 1 to 256. Time base 2 can also be clocked externally and internally, but it also has a third mode. This is the angle clock mode which, when supported by software, can keep track of the angle of a toothed wheel. In a dual engine configuration of the eTPU a time base can shared with the other engine.

Timer channels. An engine has 32 I/O timer channels. A channel includes hardware for input signal processing, hardware for output signal generation, and timing

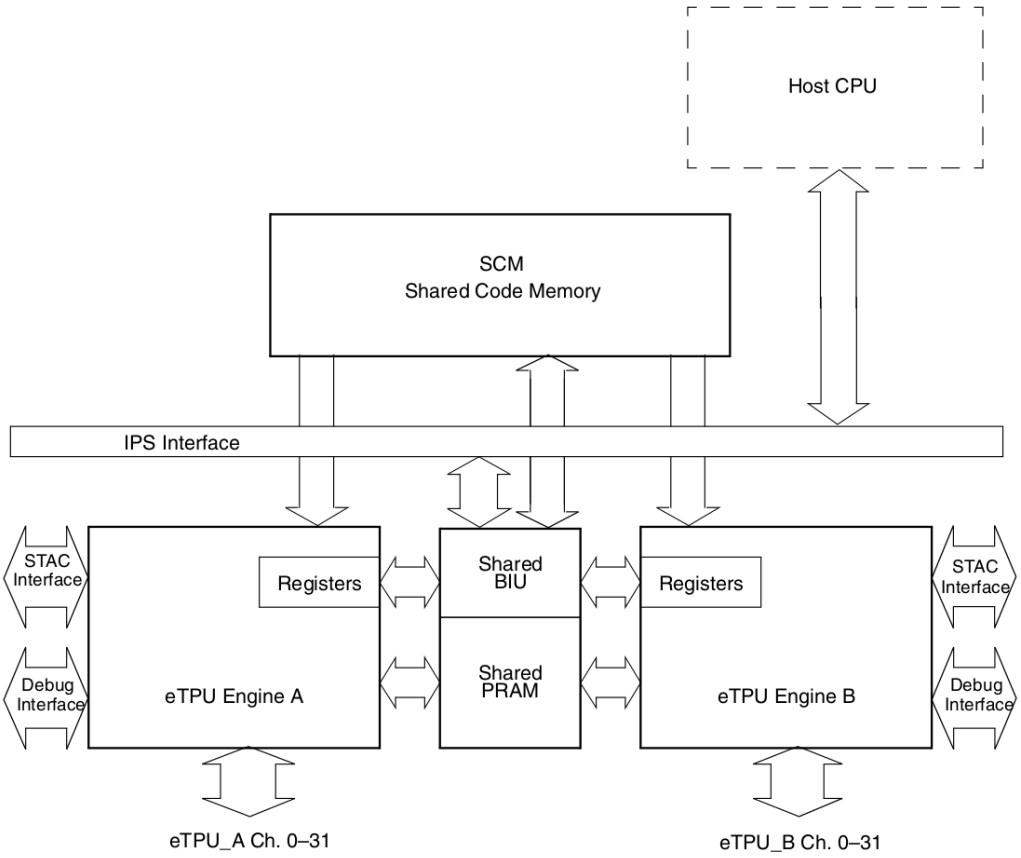


Figure 3.1: High level block diagram of the enhanced Time Processing Unit in dual-engine configuration. Figure taken from Freescale [2].

event logic. Depending on the implementation, the channels' input and output signals be tied to one pin. The hardware for input signal processing includes a digital filter that removes noise. There are two types of timing events, input pin transition and time base match, the latter meaning that a time base counter reached or exceeded a predefined compare value.

The timer logic includes two sets of one 24-bit capture register and one 24-bit match register, enabling a channel to support four different timing events. A capture register holds the captured time base value upon an input pin transition and a match register holds the value to compare with a time base in order to generate a time base match event. The channel mode dictates when a timing event is enabled and what action should be taken when an enabled timing event is generated. The actions possible include enabling/disabling of other timing events, a change in the output level, or a service request can be issued.

A channel can send 32 different service requests to the scheduler. The request sent depends on internal flags, what timing event or specific combination of timing events that have occurred, requests from other channels (called link requests), and service request triggers received from the CPU. The set of service routines, called threads, executed by the microengine upon a service request from a specific channel is called a *function*. The instructions constituting the service routines of the function reside in the SCM.

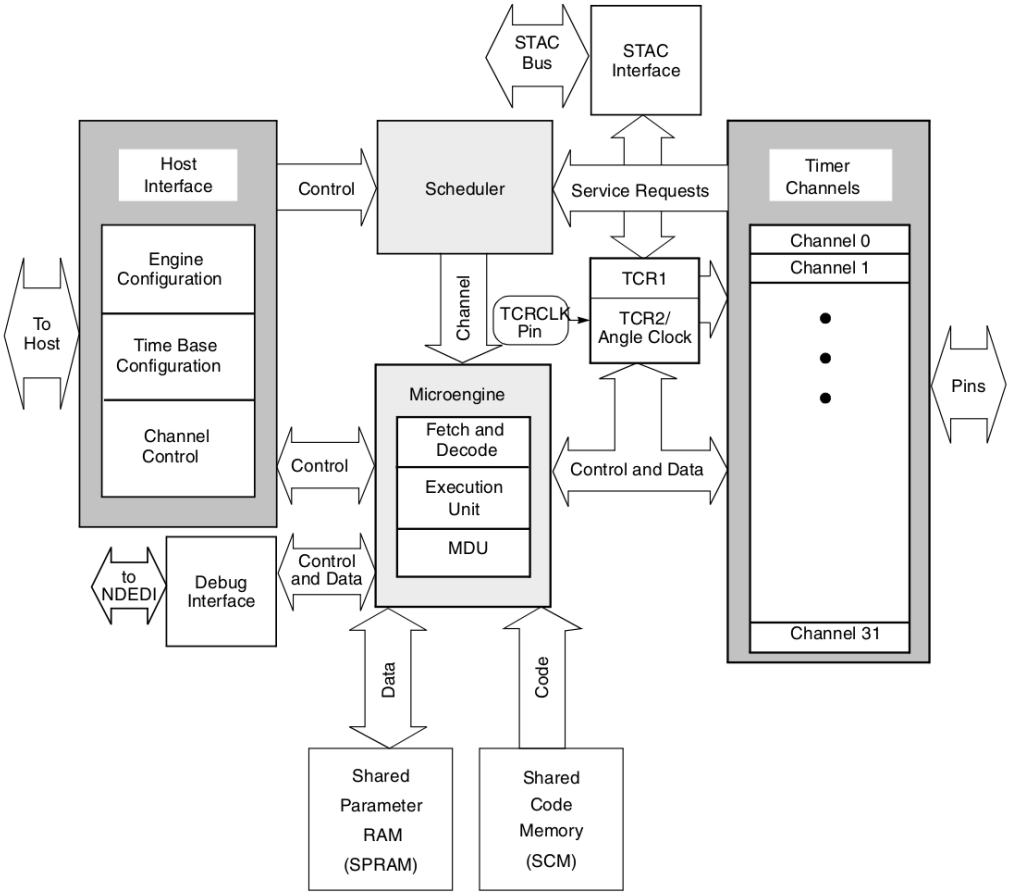


Figure 3.2: Block diagram of an enhanced Time Processing Unit engine. Figure taken from Freescale [2].

Angle clock. The eTPU's angle clock is a combination of hardware and software. Dedicated logic, the second time base counter, and timer channel 0 are used together with the function associated with timer channel 0. The micro tics are generated as explained in Section 2.2.1, where the slit period is estimated by software. Missing slits on the flywheel can be treated as regular slits or the micro tics for the slit before the missing ones can be increased to make up for the missing slits. Software has to choose the desired action every time, right before the missing slits, meaning that a profile of the slits has to be kept. Additionally, the software synchronizes the angle clock to the slit profile.

Scheduler. The schedulers job is to determine which service request the microengine should service next. It employs a scheduling mechanism that has a primary and a secondary priority scheme. A channel's priority can be set to one of three levels. The primary scheme decides which of the three priority levels that the secondary scheme should pick a service requests from. The request chosen then gets served by the microengine. To decide which priority level to choose, the primary priority scheme uses time slots where each slot indicates one of the priority levels. Seven time slots are used where two are dedicated to medium priority channels, one to low priority channels and the remaining to high priority channels. If no channel

with the chosen priority level has issued a service request, another slot is chosen. A time slot ends when one service routine has finished execution, and a service routine executing can not be interrupted by any event other than a force end event issued by the CPU. The secondary priority scheme prioritizes the channel with the lowest channel number. Starvation of high numbered channels is eliminated by starting a new servicing cycle on the channels with the same priority level only when all channels on that priority level which require service has been served.

Microengine. An eTPU engine's microengine consists of two pipeline stages: instruction fetching, and execution. When no service routine is executed, it is in an idle state. Execution of one instruction takes two clock cycles, with the exception of multiply, division, and Multiply and Accumulate (MAC) instructions or if an access conflict occurs in the SPRAM. However, several resources within the microengine can be utilized in parallel, and if no access conflict occurs and if the MAC and Divide Unit (MDU) is not overloaded, the throughput is one instruction per two clock cycles. One instruction can perform three out of four types of operations: Arithmetic Logic Unit (ALU)/ or MDU operations, SPRAM operations, channel configuration or control operations, and flow control operations.

Host interface. The host interface gives the CPU control of the eTPU operation. Initialization of the eTPU is performed by the CPU which assigns a function and scheduling priority to each timer channel, as well as initialize the SCM. The initialization sequence is finished when the CPU enables eTPU access to the SCM, which disables CPU access. Operation is then started when the CPU enables the timer channels. To further control operation, the CPU can trigger service requests and modify a channel's configuration. During operation channels can spawn two types of request targeted towards the CPU: channel interrupt request and data transfer request. The requests are issued within a executing service routine. The channel interrupt request is serviced by the CPU and the data transfer request is serviced by a Direct Memory Access (DMA) module.

3.2 Enhanced Modular Input/Output Subsystem

The eMIOS is a hardware timer module developed by Freescale. It has 32 channels that each have a set of configuration registers, a 24-bit counter, and 24-bit wide registers for capturing or comparing counter values. There are also five global 24-bit counters that can be used as common time bases.

All channels in the eMIOS can be configured to perform one out of 18 available functions. The functions include different kinds of PWM generation, PWM period or duty cycle measuring, single input/output event generation, and General Purpose Input Output (GPIO).

3.3 Generic Timer Module

The Generic Timer Module is an IP-module developed by Bosch. It is designed to be modular allowing different hardware vendors to choose a configuration that fits their needs. The main task of the GTM is to offload work from the CPU by handling fast repetitive tasks and minimizing the number of interrupts needed to be raised. This is achieved with a number of programmable hardware modules that after initialization can work independently, without the need to constantly interrupt the CPU.

This section will expand on the first generation of the GTM architecture as this was the only generation readily available and suitable in terms of IO ports when the thesis was carried out. The final version, 1.5.5.1, of the first generation was released in early 2014 and shortly after this the first microcontroller including a GTM became available on the market. At the end of 2014 generation 2 was released [8]. However, due to this generation only being released in small device configurations it did not fit the needs for this work. The third generation was not available for this work as it had not yet been released, but the relevant improvements this generation will bring are brought up in Section 3.3.4.

Central to the GTM architecture is the Advanced Routing Unit (ARU) which transfer data between all modules connected to it. For example, a common signal path in the GTM is where a Timer Input Module (TIM) channel captures an input event, then transfers information about the event through the ARU to a Multi Channel Sequencer (MCS) channel for processing. When finished, the MCS sends output characteristics over the ARU to an ARU-connected Timer Output Module (ATOM) channel that generates an output signal. By using a module called the Signal Multiplexer (GTM-MX) it is possible for the GTM to generate input events to itself by internally routing an ATOM or Timer Output Module (TOM) output to a TIM input.

3.3.1 Modules

The modules that the GTM architecture builds upon can be divided into three groups. The first group, green in Figure 3.3, contains infrastructure modules that provide base functionality needed by the other modules. The second group, yellow in the figure, consists of four different modules for signal generation and processing. The last group, cyan in the figure, include modules for application specific and safety related tasks. Below we give more details about the different modules in the GTM.

ICM. Even though the GTM aims to reduce the number of interrupts to the CPU almost a thousand different interrupts can be raised by the GTM's modules. To lower the number of interrupt lines needed to the CPU the Interrupt Concentrator Module (ICM) concatenates related interrupts into common interrupt lines. Due to this, the CPU has to read a status register inside the relevant GTM module to determine the actual interrupt source when a interrupt is raised.

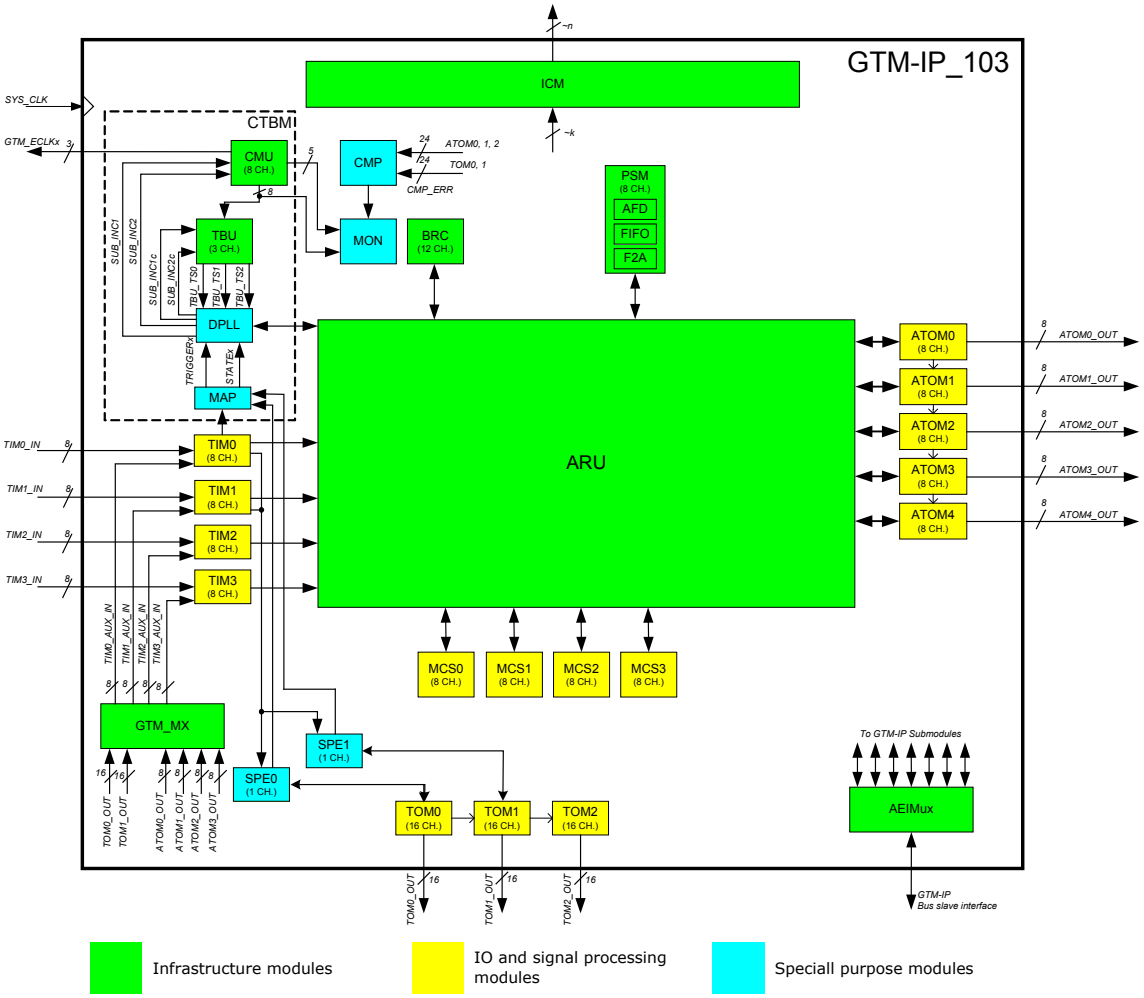


Figure 3.3: Overview of the Generic Timer Module architecture, device 103 and IP version 1.5.5.1. Figure taken from Robert Bosch GmbH [3] with added color coding.

CMU. The GTM is clocked by an external clock called *SYS_CLK* with a maximum frequency of 100 MHz, but to be able to generate signals and events with specific timing there is a need for configurable clock signals. It is the Clock Management Unit (CMU) which provides this in the GTM. The CMU has one global clock divider that multiplies the SYS_CLK with a rational number, where the rational number must be less or equal to one. The resulting clock $\frac{n}{m} \times \text{SYS\_CLK}, n \leq m$, is distributed to the rest of the GTM's modules as their clock source. To achieve other clock rates the CMU has eight clock enable signals. These are created as integer multiples of the global clock which results in them all being synchronized. The CMU also has eight clock enable signals with fixed dividers. These signals can be driven by any one of the eight user configurable signals.

TBU. The Time Base Unit (TBU) counts the number of pulses from a selected CMU clock enable signal. The produced counter value can then be accessed by the GTM's other modules, thus providing a global time base with the same resolution as the selected clock enable signal. As the maximum clock frequency is 100 MHz the

highest resolution achievable is $\frac{1}{100 \times 10^6} = 10\,\text{ns}$. A total of three counters exist in the TBU and all three can trigger on any of the CMU clock enable signals, in addition, two of the counters can be set to count the number of micro ticks generated by the Digital Phase Locked Loop (DPLL), possibly providing an angle clock instead of a time base.

ARU. For the GTM to unload the CPU it has to be able to work independently and the different modules thus need to be able to communicate with each other. This internal communication is what the ARU is responsible for. The ARU transfers data between all connected modules in a round robin fashion and all transfers takes two cycles of the global clock. This design ensures a deterministic communication between all modules with a worst case transfer latency, called *ARU Round Trip Time (RTT)*, that only depends on the number of ARU-connected data consumers. The width of the data transmitted is 53 bits; two 24-bit words and five configurable control bits.

All data producers have a specific address from which a consumer can read from, and as all transfers are destructive there can only be one consumer for each producer. However, this can be circumvented by using the Broadcast Module (BRC) to broadcast the same data to several consumers, with the downside of a higher latency.

MCS. For signal processing the GTM has a scaled down RISC core. The core consists of a five stage pipeline with eight hardware threads, called *tasks*, that have their own work registers and program counter. The tasks also have access to a shared trigger register, about which more detail will be given later. The pipeline's ALU operates on 24-bit wide operands and can perform addition, subtraction, and basic logic operations, but lacks support for multiplication and division. Most instructions take one instruction cycle to execute, except for memory accesses and some flow control instructions. Data can be shared between all tasks within one MCS instance via the shared Random Access Memory (RAM), but the ARU has to be used to share data between tasks in different MCS instances.

The available memory for data and code in one MCS instance is by default 6 KB. It is configured as a 4 KB and a 2 KB page which can be configured to belong to a neighboring instance. Thus the memory available to one instance can range from 2 KB to 10 KB. Independent of the memory configuration all available memory is shared between all tasks in one MCS instance.

Which of the eight tasks in one MCS instance that should get to execute next can be decided upon in two ways. The first is round robin scheduling where all eight tasks and the CPU get one instruction slot each round. The CPU gets one slot so it can access the MCS memory and registers. Running this scheme, one instruction cycle takes nine clock cycles as all tasks get scheduled independent of if it is sleeping or not. The other scheduling schema is similar to round robin, but sleeping tasks does not get scheduled. Running this accelerated scheduling schema one instruction cycle can take one to nine clock cycles, but with a minimum latency of five clock cycles due to the pipeline. A task can be sleeping for three different reasons: it is disabled, it executes a ARU read or write instruction, or it have executed a wait

instruction.

ARU read instructions can be either blocking or non-blocking while the ARU write instructions are only blocking instructions. A blocking instructions will sleep the task until data has been received or sent through the ARU. The non-blocking read instructions will sleep the task until the ARU arbiter has moved past the data destination read from, resulting in a worst-case sleep time of one ARU RTT.

The mentioned wait instruction that puts a task to sleep is the *Wait Until Register Match (WURM)*. After executing a WURM the task will sleep until the targeted register holds a specified pattern. Together with the shared trigger register the WURM instruction enables a task to sleep until another task, or the CPU, wakes it by setting its specific bit in the trigger register. This triggering only works one way and a task can not put another task to sleep, so for enabling/disabling of a task the CPU has to be used.

All instructions to be executed in the MCS have to be written in assembler code, as there is no high-level language compiler available. The assembler produces a C array of 32-bit integers that represents the machine code. This array then has to be written into the MCS memory by the CPU at system startup, after which the MCS tasks can be enabled to start execution.

AEI. To configure one of the GTM's modules the CPU writes configuration bits to hardware registers and RAM locations in the respective module. This is performed using the Generic Bus Interface (AEI) which connects to the CPU's bus via a bridge module. The MCS is not able to access the AEI, meaning that it can not perform any configuration of the modules. All write or read accesses by the CPU over the AEI have lower priority than each module's own access.

PSM. Another way to transfer data to all ARU-connected GTM modules other than over the AEI is to use the Parameter Storage Module (PSM). The PSM offers an interface between the ARU and the CPU with a buffer in between. The buffer can either run in First In First Out (FIFO) mode or be used as a ring buffer. When running as a ring buffer the same data sequence will continuously be served to the ARU enabling, for example, generation of a periodic complex output signal. There can be a variable number of PSMs in the GTM and each PSM has eight configurable buffers with corresponding interfaces to the ARU and CPU. The buffer also has a mechanism for raising interrupts when certain configurable fill levels have been met, e.g. full or empty. In addition to transfer data from the CPU into the GTM, the PSM can also be used the other way around.

TIM. The TIM handles input to the GTM and consists of eight internal channels that can each process one input signal. It is connected to the ARU as a data producer, and depending on the mode of operation a channel will write different kinds of data to the ARU. The different modes include input edge counting, PWM signal characterization, input transition time stamp capturing, conversion of parallel data to serial data, or periodic sampling. Each channel have an internal filter module that can be activated. This filter can perform various types of glitch filtering and time out detection, independent of the selected mode of operation.

TOM. For generation of PWM and Pulse Count Modulation (PCM) signals a TOM can be used. Each module instance has 16 channels that all have a 16-bit counter, two 16-bit compare registers with accompanying shadow registers. The rate of the counter can be selected from one of the CMU clock enable signals with fixed divider. The two compare registers should be loaded with period and duty cycle of the signal to be generated. To load new values into the compare registers without disturbing ongoing operation the shadow registers can be used. The TOM can be configured to load the values in the shadow registers at the beginning of the next period, or it can be set to wait for a trigger signal from a neighboring channel.

ATOM. The TOM can only be loaded with data over the AEI, so to be able to generate an output signal with the MCS or the PSM an ATOM has to be used. It has 8 internal channels instead of 16, and apart from having the same basic functionality as the TOM, and being connected to the ARU, the ATOM has some additional features. For one, the internal registers are 24 bits wide instead of 16 bits, allowing for a greater range of signals to be created. The compare registers can also compare against the TBU's global time bases, in contrast to only the internal counter, giving the possibility to produce output based on the angle clock or a global time base.

The ATOM also has a mode of operation in which it is configurable what action to take when a compare event occurs. Possible actions include different combinations of output transitions and combinations of enabling/disabling of the other compare register. When running in this mode and a compare event occurs the value from two of the TBU time bases is captured in the ATOM's shadow registers. These values then have to be read, either by the CPU or over the ARU, before new compare values can be written to the ATOM.

DPLL. The DPLL has the purpose of generating a fast switching signal from a slower signal. One application is to drive an angle clock keeping track of the rotary position of a combustion engine, as explained in Section 2.2.1. It has two inputs which could be used either separately to maintain rotary position of two engines or they could be used in conjunction for one engine. In the latter case one input signal works as a backup that can be seamlessly switched over to, in case the other fails.

The created fast switching signal is used to trigger one of the global time bases in the TBU, which then can be used as an angle clock. The number of micro ticks generated for each revolution are fixed independent of acceleration or deceleration. Deceleration is handled by pausing the micro tick generation when all ticks have been generated for the current period. For acceleration there are two strategies that can be used: the first is to generate all remaining micro ticks as fast as possible before beginning with the ticks for the next period. The other is to add all remaining micro ticks to those that will be generated in the next period, before calculating the frequency for micro tick generation.

The period to use for micro tick generation is a prediction calculated using data of up to two full revolutions old input events. The DPLL can also calculate time predictions for when a specific angle will occur in context of the global time base. This is possible as the DPLL is configured with a profile of the input signal which it

uses together with the two revolutions worth of saved input data. Synchronization is performed by manually detecting the current rotational position and then setting a pointer to the correct position in the profile.

MAP. When the DPLL should be used to monitor the rotary position of a Brushless Direct Current (BLDC) engine the Input Mapping Module (MAP) needs to be used. When one of the hall effect sensors in a BLDC engine, usually three, changes output, an input event has to be passed to the DPLL. This is where the MAP comes in and maps the output from all three sensors to one DPLL input. The MAP is also used when the DPLL should track other signals than from a BLDC engine. In these cases the MAP just statically routes two TIM channels' output to the DPLL's inputs.

SPE. The Sensor Pattern Evaluation (SPE) module can in conjunction with a TOM and a TIM drive a BLDC motor, but as this work not include any BLDC engines, this module will not be covered in more detail.

CMP. In safety critical applications error detection is vital. The Output Compare Unit (CMP) can be used to compare the output of neighboring channels in the TOMs and the ATOMs to detect a malfunction. As this thesis does not focus on safety this module will not be used.

MON. Similar to the CMP the Monitoring Unit (MON) is used in safety critical applications and will not be used in this thesis.

3.3.2 Device configurations

For the GTM to fit a multitude of applications it exist in several device configurations. The main difference between the configurations is in the number of MCS, TIM, TOM, and ATOM instances. As all of these modules except for the TOM are connected to the ARU they affect the ARU RTT, and in extension the worst case transfer latency. In Table 3.1 two different GTM device configurations with their respective number of modules and ARU RTT is displayed.

Table 3.1: Two device configurations of the GTM with the number of modules (channels) they contain and their respective ARU RTT.

Device name	MCS	TIM	TOM	ATOM	ARU RTT
Device 103	4 (32)	4 (32)	2 (30)	5 (40)	730 ns
Device 104	6 (48)	6 (48)	5 (80)	9 (72)	1130 ns

3.3.3 GTM reference model

Bosch maintain a reference model of the GTM bundled together with a simulation environment. The GTM Reference Model (GTM-RM) makes it possible to develop

and test code for the GTM on a computer without any additional hardware. A full trace of almost all internal GTM signals and register can be viewed after a simulation has finished. This in conjunction with the possibility to make printouts to a log file during simulation makes the GTM-RM very suitable for initial development and testing. However, there can be some small differences between how a real GTM and the simulated one function, so final verification of a design using the GTM has to be performed on the actual hardware. Especially when it comes to verification of timing requirements as the simulation tool is not cycle accurate.

3.3.4 GTM generation 3

The next generation of the GTM-IP that Bosch will release is the generation 3. It will bring several large improvements to mainly the ARU, and the MCS [15]. Also, the ability to run at a clock speed of 200 MHz and clock gating on a cluster level have been added. The clock gating allows unused parts of the GTM to be shut of, resulting in a decrease power consumption. The division into clusters is done on a module level where each cluster contain one of each module type, for as long as there is instances. E.g. the first cluster contain a PSM while the second does not as there only is one PSM.

The improvements made to the MCS include the addition of a parallel multiplication instruction and a serial division instruction. Next is that each MCS task will have the ability to share a neighboring task's registers, enabling data exchange between tasks without the need to use the memory. Another large improvement is that a MCS module will be able to access the AEI in its own cluster. This makes it possible to use the MCS for configuration of the other modules in the same cluster. The final relevant improvement of the MCS is the ability to receive interrupts from the other modules in the same cluster. These last two reforms will increase the GTM's independence from the CPU.

In the ARU the possibility to customize the arbitration sequence has been added. The connected data destinations are still served in a round robin fashion but it is now possible to insert specific data destinations at user defined intervals in the arbitration sequence. This results in these data destinations getting served more frequently than the otherwise fixed ARU RTT. It is also possible to specify the complete arbitration sequence in a PSM from which the ARU then loads it.

There are other improvements brought by generation 3 but as they do not have any significance for this thesis they will not be brought up.

3.4 AURIX

AURIX is a family of microcontrollers developed by Infineon. It is aimed at the automotive industry and is designed to offer high performance and high safety. It exists in various configurations in terms of memory size and number of cores. At most it has three cores of Infineon's TricCore architecture. For the developed prototype we used an AURIX TC2X7 development board with a AURIX TC277[1]

[1] AURIX TC277TF-64-F200S-CA [16]

microcontroller. This model has in addition to three processing cores a GTM device 103 and a multitude of other peripherals. The peripherals of interest for this thesis are the Analog to Digital Converter (ADC), the asynchronous serial interface, and the DMA module. We will therefore give some more detail about these here.

VADC. For conversion of analog voltages to digital values the Versatile ADC (VADC) in the AURIX can be used. It has 64 channels divided into groups of 8. Each group has its own converter with a resolution of up to 12 bits. The channels are served in order and can be individually shut off to let the others work faster. Each channel can also be set to what is called *Fast Compare Mode (FCM)*. In this mode the converter will not produce a full result but only a flag bit indicating if the voltage is over/under a user specified value. In this mode it is also possible to specify a hysteresis to minimize glitches when the voltage is close to the specified level.

The time it takes to perform a conversion depends on several factors, however, the fastest conversion time is achieved when running FCM. Running this mode as fast as possible a conversion time of 100 ns is possible. The time between each conversion depends on how the channels are triggered. If running the auto-scan trigger mode, all active channels get triggered in a continuous sequential order, and the latency between each conversion becomes 800 ns. When single trigger mode is used the latency can range from 100 ns to 800 ns. A feature of single trigger mode is that several channels can be grouped together and then triggered by a single signal.

DMA. The DMA module can be used to transfer data between peripherals in the AURIX without the CPU's intervention. It consists of 64 channels that can be individually configured with what is called a *transaction control set (TCS)*. When a channel gets triggered, one out of two move engines loads the channels TCS and performs the transfer. The source and destination address stored in the TCS can be automatically increased or decreased to enable transfer of complete memory blocks. Also, the TCS can include a pointer to another TCS for the move engine to load when it is finished with the current transfer. This feature makes it possible to make a sequence of transfers from memory addresses that are scattered throughout the address space.

Triggering of a channel can be performed either by software or by hardware. The software triggering is performed by the CPU and can for example be used to initiate a block transfer of data in the background. For hardware triggering the various interrupts of all the peripherals in the AURIX can be used. For example a finished VADC conversion can trigger a transfer of the result to the memory of a MCS in the GTM.

Communication interface. To communicate with the AURIX, one of its several different communication interfaces can be used: including Ethernet, CAN, FlexRay, I2C, and an asynchronous/synchronous serial interface. The last interface supports different communication protocols like: LIN, SPI, and ASC. In all protocols data transmission and data receiving is done through buffers, and the CPU can be notified of new data received either by interrupt or by polling.

4

Tasks performed by timer modules in the EMS

The ACM and EMS ECUs focused on in this thesis employ an eMIOS and an eTPU, where the latter is in a dual core configuration. As mentioned the EMS perform the tasks regarding fuel injection which this thesis focuses on. The timer modules handle all sensors and actuators within Volvo's fuel injection control system and will be elaborated on in Section 4.1. All other tasks performed by the timer modules in the EMS are briefly described in Section 4.2 to give a holistic view of the ECU.

4.1 Fuel injector control system

Volvo's fuel injector control system described here is our reference system when evaluating the GTM. This system can work with many different configurations regarding injectors and high-pressure fuel pumps. One commonly used configuration is the Delphi F2 system. Volvo mainly employs eTPUs to perform all complex time critical tasks within the control system, such as generating current waveforms. The eMIOS only performs less complex tasks like PWM signal generation. The CPU's main task during runtime is to to feed the timer modules with configuration data and commands. Other tasks that the CPU performs are gathering input parameters for computing the configuration data, and running diagnostics, but as this thesis is focused at the timer module's responsibilities we will not discuss the CPU's tasks further. The following sections describe the timer module tasks regarding fuel injector control.

For safety reasons and for the possibility of dynamic calibration, the eTPU performs self diagnostics. If for example a short circuit is detected, the affected function's logic signals are deactivated within 1 µs, resulting in a graceful degradation. Furthermore, fault codes are set for easier maintenance in the event of a fault.

4.1.1 Angle clock

In Volvo's system the eTPU keeps an angle clock for engine rotary position feedback, as explained in Section 2.2.1, and the same part also calculates the engine speed. There are 60 slits, minus the missing ones used for synchronization, around the flywheel. For a four-cylinder engine two slits are missing for each half of the flywheel circumference, and for a six-cylinder engine two slits are missing for each third of the flywheel circumference. Each slit, and the missing ones, are further divided into

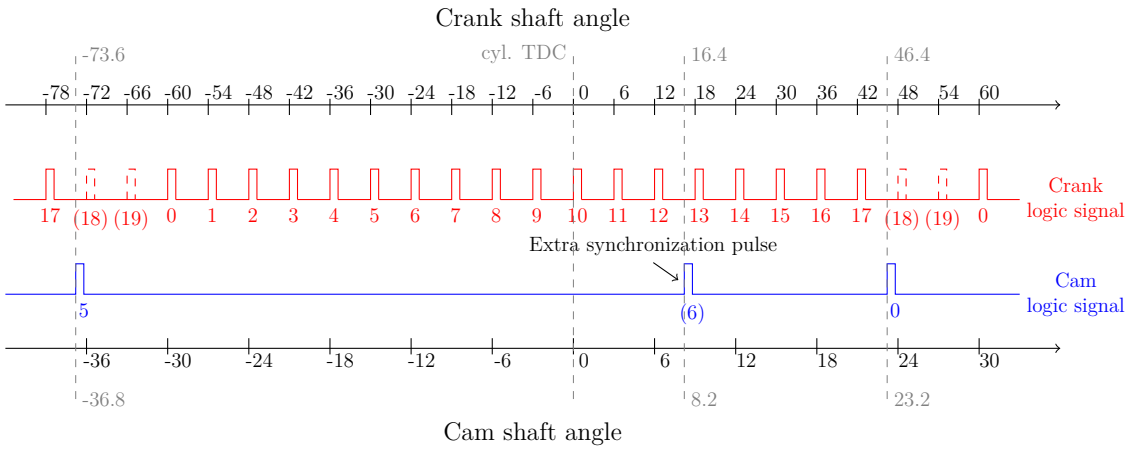


Figure 4.1: Example of the logic signals generated from crank and cam sensors around the TDC event of cylinder four in a six-cylinder engine.

512 micro ticks, resulting in a resolution of $\frac{360°}{60\times512} = \frac{3}{256}°$. For the cam wheel there is one tooth per cylinder distributed evenly around the wheel, plus one extra tooth placed 15 degrees before the tooth indicating the first cylinder. The extra tooth is there to enable for synchronization. Figure 4.1 displays the input signals generated by the cam and crank sensors around the Top Dead Center (TDC) event of cylinder four in a six-cylinder engine.

The angle clock logic in the eTPU, presented in Section 3.1, only has one input, which are connected to the crank sensor. In the event that the crank sensor would break, the Volvo system can fall back to using the cam wheel feedback for timing injection events and engine speed calculations. However, this results in lower precision as the cam sensor is not connected to the micro tick generating logic and injection events will have to be scheduled in the time domain. The engine speed will also be less accurate as there are less input events produced by the cam sensor than by the crank sensor. If instead the cam sensor breaks down, the micro tick generation continues to work, but synchronization becomes problematic as there is no way to detect which cylinder is active.

In the eTPU the engine speed is calculated as an average over 20 crank slits. When summing the period for these slits together an error could be introduced if the period for the missing slits were included, as it is three times the normal length. Volvo have solved this by virtual slits at the location for the missing slits. These virtual slits are given the same period as the period for preceding slits.

4.1.2 Needle valves and spill valves

For control of the needle valves and spill valves the eTPU generates high precision waveforms, which are a function called *Peak and Hold Current Waveform Generation (PHCWG)*. The drive signals for the solenoids are created as explained in Section 2.1 and the regulation of the waveform is aided by current feedback. The battery voltage in the system is 24 V and the boost voltage, provided by a capacitor, is 48 V. The current feedback signal is supplied by hardware as a voltage that has

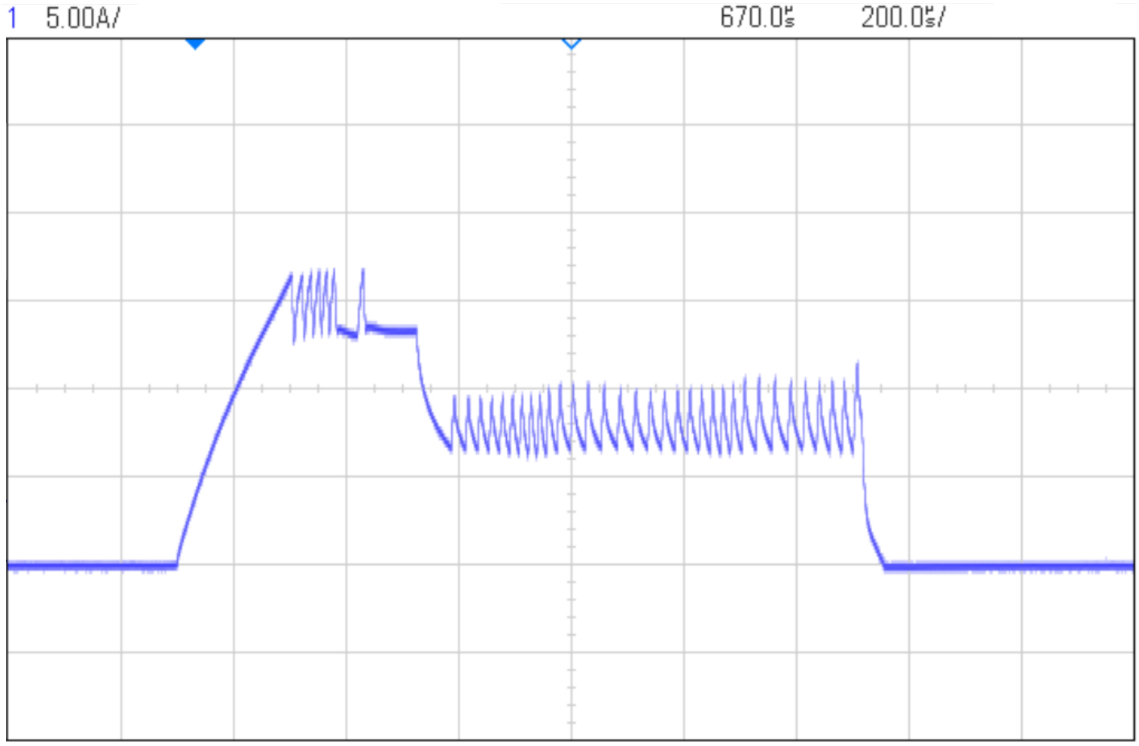


Figure 4.2: Typical waveform for the Delphi F2 fuel injector. The waveform is generated by driving dummy loads, resulting in that the waveform characteristics are affected and the waveform is not within the required boundaries. Figure taken from a Volvo internal document [4].

a linear relationship to the current flowing through the solenoid. Two comparators are fed with the current feedback signal and the one-bit comparator output switches level when the feedback signal passes a set reference voltage. The reference voltage is created by a low pass filtered PWM signal generated by the eMIOS. One comparator's reference value is set to the peak current and is called the *peak comparator*. The other comparator's reference voltage is set to the level of the hold phase and is called *hold level comparator*. This comparator also has a logic signal called Hold Level Control (HLC) which adds a voltage offset to the reference value when activated. The added offset makes the comparator's switch level to rise to that of the peak phase.

The Volvo system has a feature that enable an automatic switch between boost and battery voltage. The main purpose of this feature is to slow down the rapid decay during the beginning of the peak phase when the boost voltage is switched off.

At run time the CPU feeds the timer module commands comprising of a list of instructions that the timer module follows sequentially to produce a desired current waveform. The possibility to combine simple instructions in an arbitrary order enables for a wide variety of waveforms to be created. Figure 4.2 shows a current waveform generated by the Volvo system for a F2 injector.

Diagnostics are performed both in software and hardware. The software diagnostics comprises mainly of checks of the time it takes to perform a subpart of the

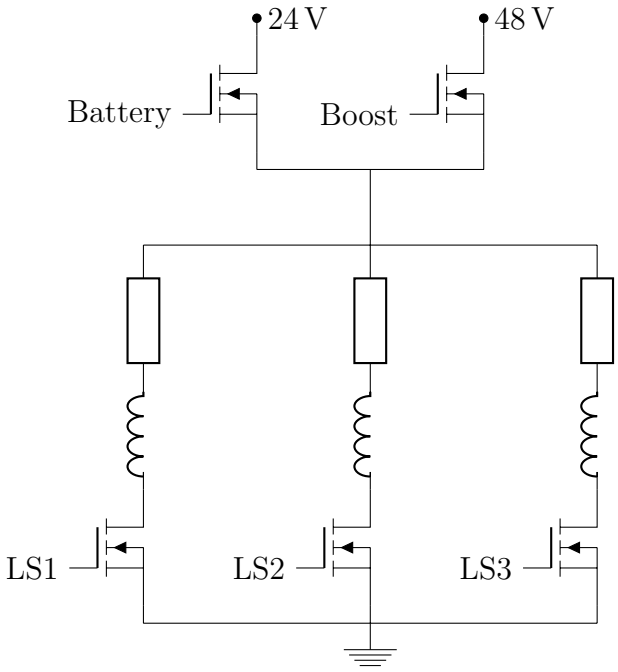


Figure 4.3: Simplified illustration of the drive circuit for one injector bank.

envelope with the intent of detecting abnormal current behavior. The hardware diagnostic detects short circuits for the high side FETs, and if this happens a one-bit digital signal (called *High Side Short Circuit Detection (HSSD)*) alerts the timer module, which then shuts down the affected drive stage.

As all of the system's injectors and spill valves are not used simultaneously, they can be grouped together to lower the number of IO pins and drive stages needed. Volvo has four groups with three PHCWG tasks in each, called a *bank*. This enables up to 12 unique solenoids to be operated, with a maximum of four in parallel. All solenoids in one bank share the same connection for the high side (Battery and Boost) but have their own low side connection (LS1, LS2, and LS3). Figure 4.3 shows an illustration of one bank. Each bank also share the peak comparator, the hold level comparator, and the HSSD signal. Together with the HLC signal this makes for a total of 9 IO pins dedicated to each bank. In this value the four PWM signals which give the reference voltage used by the comparators (*trim* signals) are not accounted for, as they are shared between the banks.

4.1.3 Remaining rail valves

The dump valve and the inlet valve employed in CR configurations are operated differently depending on the used fuel injection system. The dump valve can be either mechanically or electronically controlled, whereas the inlet valve is always electronically controlled. Further text assume that the dump valve is operated electronically, as of otherwise a timer module is not involved. An electronically operated dump valve can either be controlled like a fuel injector or by a PWM signal. On the other hand, the inlet valve is always controlled by a PWM signal. When PWM is used the drive stage consists of two FETs, one low side for connection

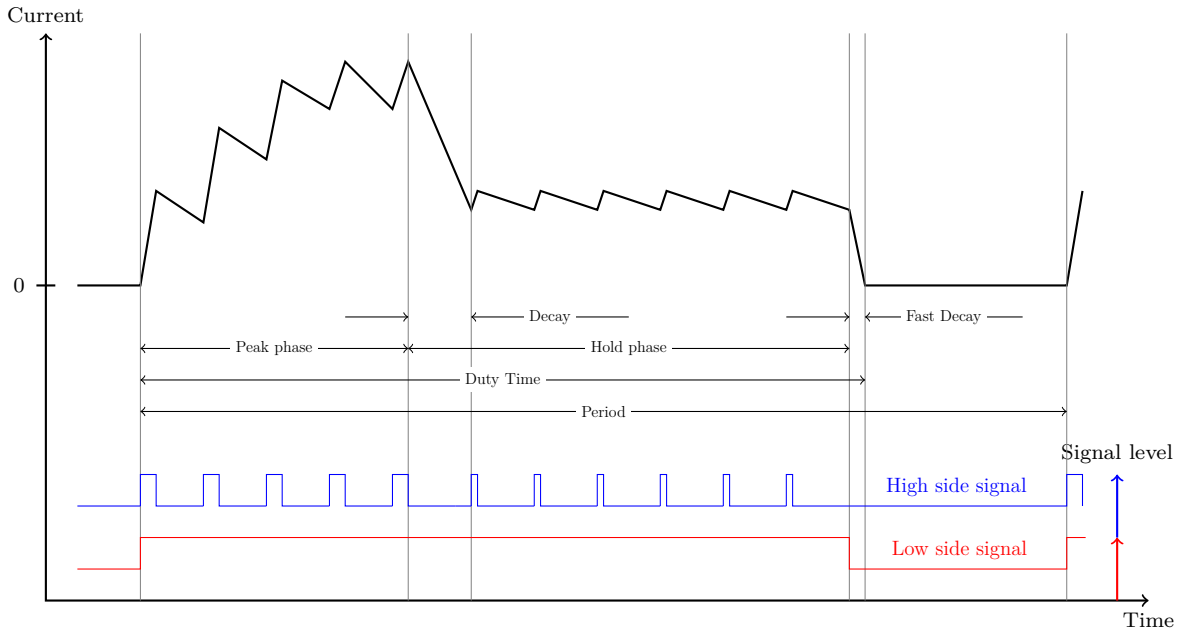


Figure 4.4: Example of the current waveform and drive stage signals generated by the PHPWM function.

to ground, and one high side for connection to battery voltage. The PWM signal is then created by switching the high side FET while having the low side FET activated.

Independent of how the valves are controlled the rail pressure is needed as decision basis for controlling them. This pressure is acquired by an ADC measuring the signal from the pressure sensor connected to the rail. The ADC is triggered by the eTPU as regular intervals during the engines rotation. When the ADC has produced a result the value is transfered to the eTPU by DMA.

The characteristics of the PWM used to drive the valves are based on the Root Mean Square (RMS) value of the current going through the valve. This special kind of PWM is called *PWM RMS*. In contrast to the control of injectors the needed current feedback is provided by an ADC instead of hardware comparators. The results from the ADC are transferred to the eTPU with DMA, after which the eTPU computes the RMS value of the current. Every 10 ms the CPU reads the calculated value and uses it together with the rail pressure sensor data to derive new PWM characteristics. These are then sent to the eTPU, which switches the FETs according to the given characteristics.

4.2 Other tasks

All of the remaining tasks handled by the EMS' timer modules are less complex than the majority of those performed in the fuel injection control system. These tasks comprise of four types: PWM signal characteristics measurement, PWM signal generation, ADC triggering and data accumulation, and generation of a signal called

Peak and Hold PWM (PHPWM).

The PHPWM produces a simpler current waveform for solenoid control than the PHCWG employed for controlling fuel injectors and spill valves. Two differences is that PHPWM control the solenoid without current feedback and employ no boost voltage. As no current feedback is used the signals that compose the peak phase and hold phase can be generated by simple PWM. The peak phase is created using a higher duty cycle than during the hold phase. Figure 4.4 presents an example of a PHPWM pulse.

The same function that performs ADC triggering for reading the rail pressure sensor is also responsible for triggering other ADCs that collect data from other pressure sensors. As such, these sensors are also read every 15th degree of the engines rotation and the results is transfered to the CPU with DMA.

Summarizing all the remaining timer unit tasks there are a total of 22 PWM signals generated by the eMIOS and the eTPU. The eMIOS measures the characteristics of four PWM signals, and two PHPWM tasks are executed by the eTPU. A reason for the eTPU to handle the PHPWM is that short circuit diagnostics have to be performed to ensure safety; an ADC measures the solenoid current level periodically and DMA transfers the result value to the eTPU which terminates the task if the current rises to high. Additionally there is an eTPU function that triggers the readout of all pressure sensors.

5

Handling of the EMS' timer-module tasks using the GTM

In this chapter we present our proof-of-concept design for how the GTM can be employed to perform the timer-module tasks currently performed by the EMS. We focus on fuel injector control and angle clock functionality as these are the most demanding tasks that the system performs, but we will also give a less detailed design proposal for how the remaining tasks can be handled. Additionally, the produced proof-of-concept prototype is described together with how the input stimuli for testing have been produced.

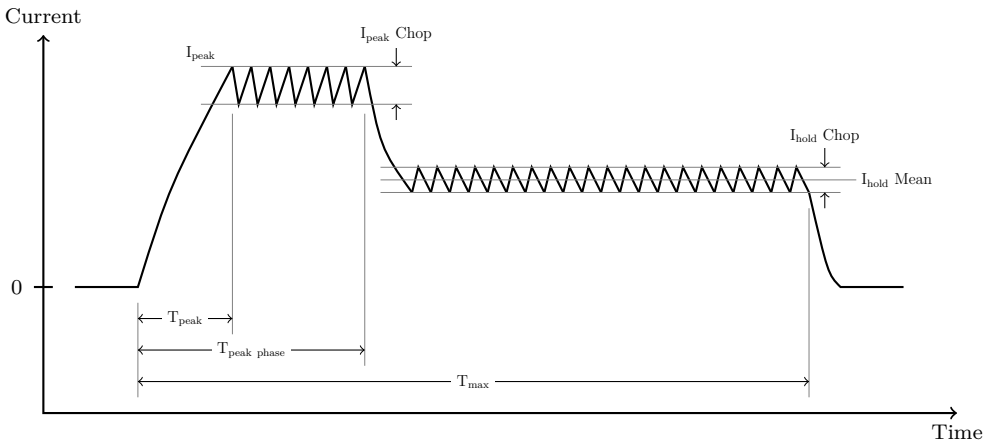


Figure 5.1: Current waveform parameters for the requirements of a Delphi F2 fuel injector.

5.1 Requirements

For Volvo's current system to function correctly there are a lot of requirements it has to fulfill. We have not created our design to follow all of those requirements as it is a proof-of-concept and not a full drop-in replacement. However, we have derived requirements for our design based on Volvo's system. The functional requirements on our design are that given the same input and output signals it should produce

Table 5.1: Typical values for parameters in Figure 5.1 given by Delphi [5].

Parameter	**Value**
I_{peak}	16 A ±0.2 A
I_{hold} Mean	8 A ±0.2 A
I_{peak} Chop	4 A max
I_{hold} Chop	2 A ±0.2 A
T_{max}	10 ms
$T_{peakphase}$	400 µs
T_{peak}	170 µs

Table 5.2: Measured worst case times for how long it takes to reach the boundaries in the different operation sections of a F2 fuel injector.

Section	**Rise Time**	**Fall Time**	
		Normal decay	**Battery decay**
Peak	3.5 µs	3 µs	3.8 µs
Peak phase	5 µs	4 µs	5.5 µs
Hold phase	5 µs	11 µs	N/A

an equivalent result compared to the existing system, described in Section 4. All remaining requirements on our design is given below.

The appearance of the current waveforms used for needle valve and spill valve control are tightly defined by the manufacturer. We aimed at being able to generate a correct current waveform for the Delphi F2 fuel injector [10], as it was the most demanding injector, in terms of timing requirements, used by Volvo at the time. Figure 5.1 shows typical waveform parameters for a F2 injector and Table 5.1 holds the values for the parameters given by Delphi [5].

Table 5.2 displays worst case measurements that we have performed on a real F2 injector. The values are the times it take to reach the boundaries in an injection event's two phases, as well as how long it takes to reach I_{peak} in the beginning of the peak phase. The first peak is separated from the rest of the peak phase in these measurements as the current drops very fast after the initial rise to I_{peak} and it can be treated individually. When obtaining the rise times, boost voltage was applied for the peak and the peak phase measurements, while battery voltage was applied during the hold phase. In the specification from Delphi no voltage is applied during decay, and the measurements for this use case are denoted with *normal decay*. However, if battery voltage is applied when the boost FET is switched off during the peak phase, the fall times become longer, giving the system more time to react. Such fall times are obtained if Volvo's automatic switch between boost and battery voltage is used. Measurements performed using this kind of feature are denoted *battery decay* in Table 5.2

The times in Table 5.2 give an indication of the timing requirements on our design; our design has to stop the current from falling or rising, within the specified times. The times also include a 0.8 µs switching time of the high side FETs. As an example,

subtracting the switching time leaves 10.2 µs for performing the chain of subtasks, from current level detection to output activation, when the current is dropping after a chop in the hold phase. This value we denote as the Worst Case Response Time (WCRT) of our design in that specific subpart of the current waveform. The other WCRT for the remaining current waveform's subparts is calculated in the same way. As we employ battery decay in our designs the WCRT for the peak and the peak phase becomes 3.0 µs and 4.7 µs, respectively.

5.2 Angle clock

In Figure 5.2 an overview for how maintaining an angle clock and performing engine speed extraction can be done using the GTM. The DPLL module performs the largest part in maintaining the angle clock, while the MCS handles engine speed calculation and helps with converting absolute angles to relative angles for the angle clock. The design is configured to trace the same kind of signals as described in Section 4.1.1 and to produce the same resolution.

Some of the angle clock related tasks handled by the eTPU in Volvo's solution are not possible to handle with the GTM alone. Self diagnostics, error handling, and synchronization need to be performed by the CPU. However, the DPLL has a large number of status flags, error flags and interrupt sources that make such tasks less complex. But as self diagnostics and error handling are not part of the angle clock's base functionality we did not include them in the design.

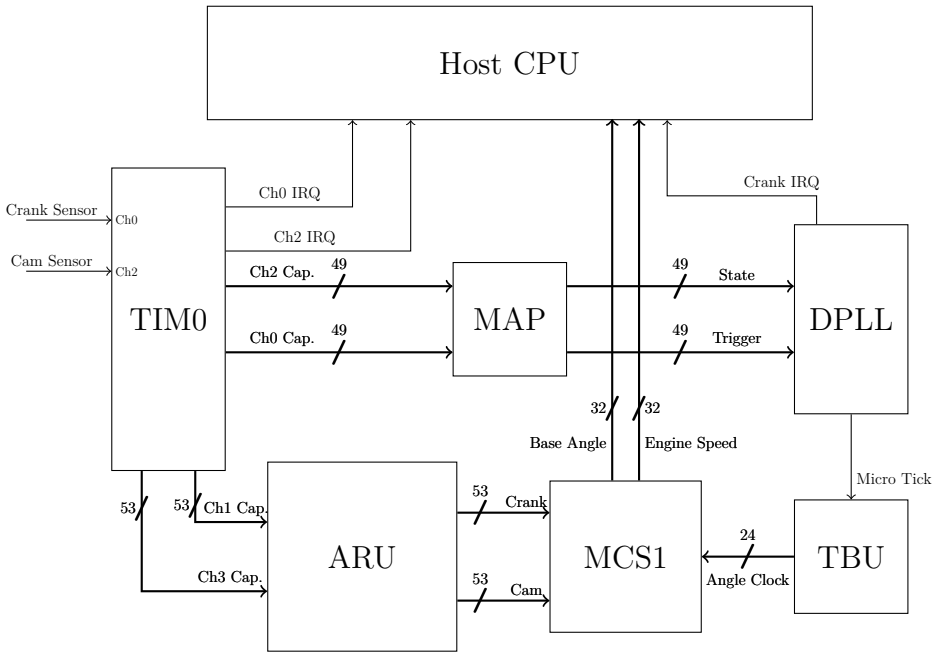


Figure 5.2: Block diagram of proposed design for engine rotary position feedback using the GTM

5.2.1 Micro tick generation

To generate micro ticks in the DPLL the crank and cam sensor outputs first have to be captured. This is done using TIM0 channel 0 and 2 for the crank and the cam signals, respectively. The channels are configured to filter away glitches on the input signals, and when a valid sensor output is captured, the time stamp for the edge and the added delay due to the filter, is transfered to the MAP module. TIM channel 0 is routed straight through to the trigger input on the DPLL and TIM channel 2 to the state input. The MAP can multiplex any of the TIM's channel 1 to 5 to either the state input or as an input for rotational direction indication. However, the direction indication was not used as the Volvo system currently does not have any sensors for this.

When the DPLL receives input from the MAP one of two things will happen: if all micro ticks for the preceding slit period have been generated it continues to the next step; otherwise, the micro ticks not yet produced are generated in quick succession before moving on. The next step is to calculate a prediction for the length of the next slit period. The calculations are done using time stamps from past input events and the configured profile of the input signal. Using the predicted period and the number of micro ticks to be generated, a frequency for the micro tick generation during the next slit period is attained. Each micro tick pulse is passed to the TBU which in turn uses them to trigger the counter used as the angle clock.

Before the DPLL module starts to generate micro ticks it has to be synchronized. Synchronization is performed by setting one memory pointer for the trigger input and one for the state input. The pointers hold the address for the engine's current position in relation to the stored input signal profiles. To detect the engine's current position, interrupts indicating a new input edge from TIM channel 0 and 2 are used. When a new rising edge is received on the cam input the period of the last tooth is calculated. If it is less than half the previous period the extra cam tooth has been found and we set the state pointer accordingly. The same approach is used for the crank slits, but for them the CPU looks for a period that is twice as long instead of half. When the DPLL is synchronized it is configured to raise an interrupt on the first crank slit after each missing slit pair (crank pulse number zero in Figure 4.1). This interrupt can later be used to schedule injection events for each cylinder.

In the event that the crank or cam sensor fail, the design can handle this gracefully. When an expected crank slit fails to occur the DPLL will raise an interrupt. The CPU can then handle it by reconfiguring the DPLL to use the cam input instead. This will result in lower accuracy than when running with the crank as the cam teeth are further apart, but the DPLL will continue to produce micro ticks to the angle clock. If instead the cam sensor breaks down, operation can continue as usual until the engine stops, then on restart the synchronization will be problematic. However, these functions have not been included in the design as they are not part of the basic functionality.

5.2.2 Engine speed and base angle calculation

Engine speed is calculated in the MCS based on data received from two TIM channels. In the TIM module it is possible to feed a channel with the same input as

its preceding channel. We use this to connect the crank sensor output to channel 1 and the cam sensor output to channel 3. Channel 1 then measures the period of the crank signal and channel 3 captures the angle clock value at the rising edges of the cam signal. Both channels then write the results to the ARU. In the MCS one channel reads the crank slit period measurements from the ARU and puts it in a 32 entry large ring buffer. The period for the missing slit pair gets discarded as the MCS is unable to divide it by three. A moving average of all entries in the ring buffer is then calculated. Even though the MCS can not perform division this is possible as the number of entries in the buffer is a multiple of two, which enables division by shifting right. The result is then converted to RPM and written to memory from where the CPU can access it.

The 24-bit counter that constitutes the angle clock does not reset after a complete engine cycle. Therefore, when an injection event is scheduled with an absolute angle it has to be converted to a relative angle clock value. For this to be possible a MCS channel keeps a variable called *base angle*. The base angle holds the counter value for the beginning of the current engine cycle and it is added to an absolute angle clock value to get the relative value. The MCS channel tracks the cam input signal by reading it from the ARU and updates an internal counter for each new input edge. When the first edge of the engine cycle appears, the MCS channel updates the base angle memory field with the captured angle clock value.

5.3 Needle valves and spill valves

This section presents two ways of performing the control of needle valves and spill valves. As for the angle clock, these designs are produced with the aim of providing basic functionality; they are proof-of-concept designs. We have therefore not included all diagnostics that is performed in the Volvo system, nor do we support instructions in the sense they are used in the Volvo system; instead we have one hard coded injection event which can be modified by a list of parameters. The first design is optimized to produce as low response time as possible, while only employing the GTM, hence it is called the *GTM-only design*. The aim for the second design is to achieve a lower response time then the GTM-only design and to do this are other resources from the targeted AURIX microcontroller in addition to the GTM utilized. This design is called the *Hybrid design* and is used to evaluate if it is desired that a design with a lower response time than the GTM-only design operates the Delphi F2 fuel injector.

We begin this section with brief introductions to the two designs. Then the design of the current waveform is presented. Next we present the signal path of the designs beginning with how input signals are captured, how they later are processed and how we have utilized the processing core, and then we present how output signals are generated. Finally we estimate the WCRTss of both designs.

The block diagram of the GTM-only design is shown in Figure 5.3. The current feedback signal is processed by hardware comparators as in the Volvo system, and each comparator output is captured by a TIM channel. The comparator data is then transported to the MCS via the ARU. The MCS then uses the comparator data together with time bases as a decision basis for deriving commands to send to

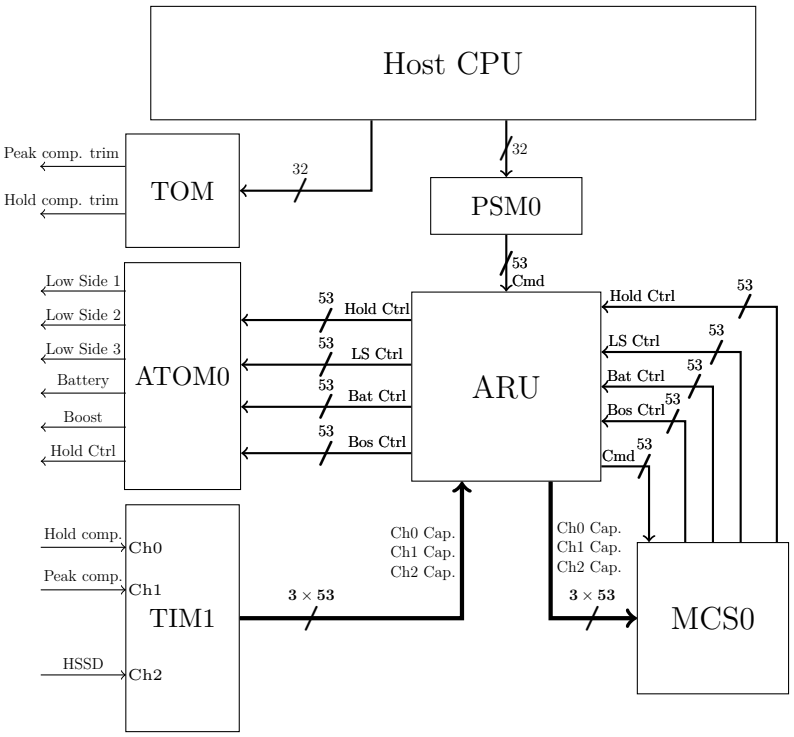


Figure 5.3: Block diagram of the GTM-only design for needle valve and spill valve control using the GTM

ATOM channels via the ARU. The ATOM channels then produce the signals feed to the FET drive stage.

In the Hybrid design, presented in Figure 5.4, we employ DMA which is included in the AURIX microcontroller family. The difference outside the MCS instance relative the GTM-only design is that the transfer of TIM results is performed by DMA instead of the ARU. Additionally, DMA is employed to safely shut down a bank which has encountered a short circuit or an open circuit.

5.3.1 Current waveform design

This section explains how we designed the current waveform to stay within the waveform boundaries specified in Section 5.1. Both designs use the same current waveform design allowing for an easier comparison of the results. We aimed to produce a waveform as close to the one specified by Delphi in Figure 5.1.

At the beginning of an envelope the specified Low Side (LS) signal as well as the boost and the battery signals are activated. Recall that the interlock suppresses the battery signal when both the boost and the battery signals are active. The applied signals result in that a boost voltage is applied to the solenoid and current starts to rise. When the peak comparator indicates that the current has reached I_{peak} the boost signal is deactivated. To keep a steady current level in the peak phase the boost signal is normally activated as fast as possible when the hold level comparator indicates that the current has dropped below the hold level. This is not the case for the first time the current drops after the initial rise to I_{peak}, where the boost

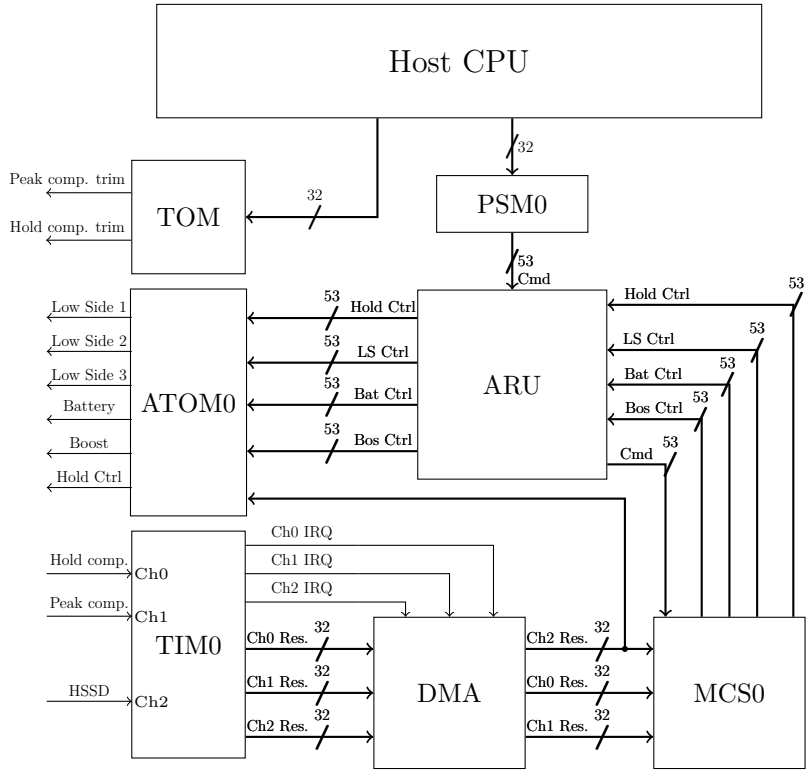


Figure 5.4: Block diagram of the Hybrid design for needle valve and spill valve control using the GTM

signal is instead activated directly. This is done as the short fall time (Table 5.2) does not allow the GTM-only design to execute the number of instructions needed to respond to the hold level comparator's result in time.

The time of a chop is determined by the configuration data that the CPU supplies to the GTM at initialization. The chop time is provided to an ATOM which deactivates the boost signal after the specified time duration. As mentioned, longer chops are needed at the beginning of the peak phase for the current to rise the same amount as later in the peak phase. We counter this behavior by decreasing the length of the boost chops for a specified number of chops. The remaining boost chops then have the same length.

Battery voltage is applied between boost pulses by employing a feature such as Volvo's automatic switch between boost and battery voltage. This battery voltage has a different effect on the current level in the beginning and at the end of the peak phase due to the change in inductance when the metallic core is lifted. In the beginning the battery voltage will only slow down the current drop. However, as the peak phase progresses the battery voltage will eventually not only slow the drop, but instead make the current rise. To counter that the current elevates too high, the battery signal is deactivated if the peak comparator indicates that I_{peak} has been reached. The battery signal is then activated again after the hold level comparator indicates that the current has dropped below the hold level and the boost signal has been activated. During the decay when the battery voltage is switched off, the peak comparator is not monitored, as the current can not rise.

A busy-wait loop is executed after the boost signal is deactivated to prevent the battery signal from being switched off. The reason to do this is to wait for the current to decay below I_{peak} before monitoring the peak comparator's result. Without the busy-wait, the battery voltage would be switched off resulting in a rapid decay instead of a slow decay. The disadvantage of waiting before monitoring the peak comparator is that a delay is introduced when the current does not decay and the battery voltage needs to be turned of.

When the peak phase has ended, both high side FETs are switched off and the HLC signal is transitioned to lower the reference voltage supplied to the hold level comparator. During the hold phase no voltage is applied between the chops. Also, as the chop length relative to the current level increase is close to linear throughout the hold phase, the chop lengths can be static. When the hold phase is finished the battery and the LS FETs are switched off, leading to a fast decay of the current and the end of the envelope is reached.

5.3.2 Input signal capture

This part of the task consists of how the one bit comparator output signals are captured and transported to the processing core. In both designs the comparator output signals are connected to one TIM channel each. The TIM channels' output in the GTM-only solution is supplied to the ARU and is transferred to the MCS when an ARU read instruction is executed. However, in the Hybrid design the TIM channels' output data is transferred to the MCS' memory by DMA, trigged by interrupts from the TIM channels.

The TIM channels are configured to produce a result as soon as an edge transition is detected on the input signal. All form of processing is kept to a minimum; the most simple input signal characteristics are captured. The mode employed counts the incoming edge transitions and produces a result when a chosen number of edges has been detected. We set this number to one edge. By this configuration we estimate that a result is produced within 50 ns. In the Hybrid design when a result is produced, the interrupt that triggers a DMA transfer is also raised.

The HSSD signal is, in the GTM-only solution, handled in the same way as a comparator output signal. However, the Hybrid design takes care of it differently. The HSSD signal is initially in its inactive state and as soon a TIM result is provided it is certain that the HSSD signal has transitioned, indicating that a short circuit has occurred. Thus, we use the new result interrupt raised to trigger a DMA transfer that disables the ATOM and MCS channels employed. The disabling of the ATOM channels also deactivates all outputs.

5.3.3 Processing

The MCS perform processing based on parameters given from the CPU. Together with data from the comparators and the two time bases, commands are sent to the ATOM channels. The MCS module is configured to schedule the tasks using round robin to ensure deterministic performance when the remaining MCS channels are used for other tasks. Table 5.3 presents how many, and for what, the MCS channels

are used, depending on design solution.

Table 5.3: Utilization of MCS channels in one MCS instance for one bank.

MCS channel	GTM-only design	Hybrid design
0	Master	Master
1	HSSD	-
2	Peak comp.	-
3	Hold level comp.	-
4	LS	-
5	Boost	-
6	Battery	-
7	-	-

In the GTM-only solution seven MCS channels are used: one channel handles the signal generation logic (*master channel*), one channel fetches the HSSD signal value from a TIM channel and reacts to it, two channels fetch comparator data from the TIM channels, and three channels send commands to to ATOM channels. Everything regarding this task could have been done in one channel, but to get the lowest latency possible we decided to split up the tasks.

The MCS channels fetching comparator data ensure that the latest data written to the ARU by the TIM channels are provided to the master channel as fast as possible. If the master channel itself fetched this data it would add a time jitter to the time-critical control loops, as the ARU fetch instruction blocks the channel until the arbiter in the ARU has passed the desired ARU read address.

To safely shut down the FETs when the HSSD signal is activated the three channels sending ATOM commands are needed. Without those channels the master and the HSSD channel would both send commands to the ATOM channels. As it is not possible for a MCS channel to perform several instructions atomically, such as checking a shut down variable and sending an ATOM command, there is a high risk that the master channel would send an ATOM command activating a FET when the HSSD channel already has deactivated all FETs. Also, it is not possible for a MCS channel to disable other MCS channels. Thus must only one MCS channel sends commands to one ATOM channel to maintain correct behavior.

Fewer MCS channels are needed in the Hybrid solution. The channels fetching comparator data are not needed as DMA is employed to provide input data to the MCS. Additionally, the three channels sending commands to the ATOM channels are not needed either as DMA is employed to disable the master channel and the ATOM channels. The HSSD channel can be discarded as the emergency shut down is triggered by a TIM channel. This result in that only the master channel is needed.

The following paragraphs will describe what is performed in each MCS channel for the two designs.

Master channel The master channel fetches parameters, which are stored in a PSM by the CPU, performs all processing based on the given commands, and constructs the ATOM channel commands. When a master channel retrieves from

the PSM all necessary parameters to produce an injection event the channel proceeds to derive ATOM commands based on the provided parameters. In the GTM-only design the ATOM commands are stored in memory which the LS, boost, and battery channels then load and send to the ATOM channels. In contrast, the Hybrid design writes the commands to the ATOM channels directly over the ARU.

LS, Boost, and Battery channels These three channels are used in the GTM-only design and are simply used as an interface between the master channel and the ATOM channels. The master channel stores an ATOM command in memory and then triggers one of the three channels. The triggered channel then loads the command and then writes it to the corresponding ATOM over the ARU. The MCS channel then goes back to sleep, awaiting a new trigger. However, before the triggered channel writes data to the ATOM channel, it check a termination variable to see whether any termination of the current envelope have been requested. If so, an ATOM command for deactivating the output signal is written instead of the provided command, and afterwards the MCS channel disables itself.

Comparator channels The GTM-only design has two MCS channels which fetch the captured comparator results from the TIM channels to the MCS instance. These channels execute blocking ARU instructions and when the MCS channel has received the data, the channel stores it in memory where the master channel can access it.

HSSD channel In the GTM-only design the HSSD channel fetches, with a blocking ARU instruction, the result from the TIM channel connected to the HSSD signal. The acquired data is then checked in order to determine if the HSSD signal has been activated. If so, an ongoing envelope is terminated and the whole injector bank is shut down. The GTM-only solution terminates a ongoing envelope by first setting the previous mentioned termination variable, and then triggering the LS, battery, and boost channels.

5.3.4 Output signal generation

Each of the output signals fed to the drive stage and the HLC signal are generated by an ATOM channel. The ATOM channels which produce the signals to the drive stage are configured to work in a mode where they compare the value in an internal register against a time base. When a compare match occurs the output signal is transitioned according to the loaded ATOM command. The HLC signal can be generated without any compare matches and the ATOM producing the HLC signal is configured in a simpler mode where one of the bits in the ATOM command simply determines the output level. The commands and the compare values are received through the ARU.

The compare capability is mostly used to perform a chop. In such case, the compare value supplied together with the ATOM command defines a time stamp in the future for when the high side should be deactivated. In all other cases it is desired that the output is deactivated as fast as possible, for example when the current has dropped below the hold level. Then the compare value sent is zero, which triggers a

compare event and switches the output as soon as the ATOM channel receives the command.

The voltage reference signals supplied to the comparators are generated by TOM channels. These PWM signals are not changed by the MCS, or by the CPU after they have been initialized. Thus they can be created by TOM channels instead of ATOM channels.

5.3.5 Worst case response times

The cases where the response time calculations are relevant for our designs are when a new input event is received which results in that an output signal shall be transitioned. Such input events are generated by the comparators and there are four cases where the WCRT is interesting: when the current initially reaches I_{peak}, when the current drops below the hold levels in the peak phase and the hold phase, and when an emergency shut down must be performed due to an activation of the HSSD signal. An estimation of the individual WCRT of all modules in our designs are presented in Table 5.4. The value provided for DMA is for one transfer and it is assumed that the transfer is served immediately.

The response time deactivating the boost signal after the initial peak is not regarded as an interesting WCRT. This is beacuase the peak comparator's reference value can be lowered if more time is needed for the system to react. Although, the comparator level affects how long the busy-wait loop needs to be and how high the battery voltage is allowed to elevate the current level. Why we nonetheless discard this response time as unimportant is due to the current's slow rise time when I_{peak} is reached initially. If the reference value has to be lowered, it results in a substantial longer time for the system to react to the initial peak than it affects the current elevation of the battery voltage and the busy-wait loop length. All thanks to the slow rise time.

Table 5.4: WCRT for the modules used in the presented designs, assumed that all digital modules operate at the maximum clock frequency of 100 MHz.

Module	WCRT of module
DMA	$\approx$ 300 ns per transfer
TIM	$\approx$ 50 ns
ARU	730 ns per transfer
MCS	90 ns per instr. cycle
ATOM	$\approx$ 50 ns

If an input event triggers an output transition, the WCRTs for the designs are calculated by adding the WCRT of the modules passed from the comparators until the output signal is transitioned. The WCRT of the GTM-only design thus includes the WCRT of the TIM, two times the ARU, the MCS, and the ATOM. Summing these together, minus the WCRT of the MCS, becomes approximatively 1.6 µs. The WCRT of the Hybrid design when excluding the WCRT of the MCS becomes 1.1 µs, as one ARU transfer is replaced by one DMA transfer. Depending on the available

time until a current waveform boundary will be violated, different numbers of instruction cycles can be executed. The WCRT calculated here, which excludes the WCRT of the MCS, is referred to as the *base WCRT*.

Table 5.5 present the number of instruction cycles in the critical path for the four interesting WCRTs and in parenthesis is the maximum number of instruction cycles that can be performed while still meeting the requirements. The number of instruction cycles assume that all time that is not part of the base WCRT is used for execution, which is not the case in the peak phase and hold phase as it will take some time before the current has dropped below the hold comparator's reference value. The table also shows our estimated WCRTs, as well as how far the values are from the requirements is in parenthesis.

The type of WCRT estimation explained above is a naive approach and the calculated value is likely to be longer than the real WCRT. What has not been taken into account is that ARU's arbiter will not necessarily be in a random position when the ARU is used a second time in the critical path. This is because the arbiter continuously polls all data destinations in a specified order independently of whether the destination want data or not, but as the poll order is not defined in any document we have access to, a correct WCRT is hard to calculate. Therefore we here present WCRT estimations based on the naive approach.

The GTM-only design has more instructions in its critical path than the Hybrid design due to the fact that the tasks are divided between several MCS channels. Relative to the Hybrid design, five more instruction cycles are needed to fetch input data and eleven more to transfer derived ATOM commands. These numbers can not be applied to the emergency shut down action, as it is designed differently.

Table 5.5: The number of instruction cycles present in the critical path during the four interesting WCRTs.

	GTM-only design		**Hybrid design**	
Section	**Est. WCRT**	**Instr. cycles**	**Est. WCRT**	**Instr. cycles**
First drop	2.8 µs (-0.2)	14 (16)	1.8 µs (-1.2)	3 (16)
Hold & Peak phase	3.8 µs (-0.9)	25 (34)	1.9 µs (-2.8)	9 (42)
HSSD	3.1 µs (+2.1)	17 (0)	0.35 µs (-0.65)	- (-)

We mentioned that during the decay from the initial peak the GTM-only design did not have time to take the hold level comparator's result into account. The current decay has to be countered by the GTM within $3.8\,\mu s - 0.8\,\mu s = 3.0\,\mu s$, when the FET switch delay has been subtracted. This leaves time for 1.4 µs of instruction execution that equals 15 instruction cycles. This is not enough instruction cycles, as the same number as are used in the hold phase, 25, would be needed. In the solution adopted by both designs the hold level comparator is not monitored and the ATOM command which activates the boost signal is dispatched directly. This results in the GTM-only design needing 14 instruction cycles and the Hybrid design needing three. The critical path for this solution no longer starts with an input event. Instead the response time is measured from when the boost signal is transitioned to its inactive state. In both designs the ARU has to be employed twice, as a readback

must be performed on the boost ATOM before the new ATOM command can be sent, resulting in the designs having the same base WCRT of 1.5 µs. The estimated WCRT of the GTM-only design then becomes $1.5\,\mu s + (90\,ns \times 14) \approx 2.8\,\mu s$ and the Hybrid design's becomes $1.5\,\mu s + (90\,ns \times 3) \approx 1.8\,\mu s$.

The WCRT is longer in the peak phase than in the hold phase as also the peak comparator is monitored, but the WCRT that affects the designs ability to keep the current within the specified boundaries is the same. As the peak comparator is monitored during decay with activated battery signal, the critical path is longer than during decays without applied voltage. The WCRT which affects the system the most is where the current drops most rapidly, and during the peak phase this is when no voltage is applied during the decay. As a result, even though the WCRT is larger during the battery decays, the WCRT affecting the current drop the most is when no voltage is applied. The number of instruction cycles in the critical path during the rapid decays are identical with those employed in the hold phase, thus the estimated WCRT becomes identical as well. The GTM-only design uses 25 instruction cycles and the Hybrid design nine, which correspond to estimated WCRTs of $1.6\,\mu s + (90\,ns \times 25) \approx 3.8\,\mu s$ and $1.1\,\mu s + (90\,ns \times 9) \approx 1.9\,\mu s$, respectively.

The total WCRT requirement during the hold phase for the design is large compared to the other two cases. The time available for execution in the MCS is 8.6 µs and 9.3 µs for the GTM-only design and the Hybrid design, respectively. These long times are well within the capabilities of both designs.

The requirement to deactivate the output signals within 1.0 µs when the HSSD signal has been activated is not possible for the GTM-only design. A total of 14 required instruction cycles and the base WCRT of 1.6 µs results in an estimated WCRT of 3.1 µs. However, the Hybrid design meets the requirement, if the DMA channel is served in time, as only one transfer is needed to disable and deactivate all the ATOM channels. One additional transfer is used to disable the master channel, but is not a part of the WCRT as the outputs already have been disabled. The estimated WCRT then consists of one TIM WCRT and one DMA transfer WCRT which becomes 0.35 µs. The time it takes for the ATOM outputs to switch after a successful transfer is neglected.

5.4 Remaining rail valves

Currently, one way Volvo control influx and dump valves in the fuel injection system is by PWM RMS, as explained in Section 4.1.3. The influx valve could also be controlled like an injector, in a similar way to that described in Section 5.3.

To use the GTM to generate a PWM RMS signal would not be possible as the MCS can not perform multiplication or division in hardware, and thus the RMS calculations would be extremely slow. One solution to this is to move the RMS calculation into the CPU. To not flood the CPU with readings from the ADC a MCS channel would be responsible for collecting a small set of data points that the CPU then could processes in a batch. Data collection would involve triggering of an ADC at specific places of the generated PWM pulse and then receive the result via DMA. The MCS would also need to control two ATOM channels for the actual PWM signal output.

For the rail pressure acquisition an ADC needs to be triggered synchronous to the engines rotation. This could be done with an ATOM channel that is set to PWM generation using the angle clock as time base. The ADC in turn, triggers a DMA transfer when it has finished its conversion.

5.5 Other tasks

Left uncovered are all the tasks brought up in Section 4.2. To perform PWM signal generation, a TOM channel can be used, and to obtain PWM characterizations a TIM channel can be used, since the TIMs and the TOMs can perform the same input and output functions as the eMIOS channels can, respectively.

The PHPWM is more advanced and also requires more resources. One approach to generate such a signal is to use an MCS channel to control two ATOM channels. The MCS would receive pulse parameters from the CPU via a PSM over the ARU, or the CPU could update pulse length and period directly in the MCS' memory. Another solution without using an MCS channel could be to use three ATOM channels, one for the low side, one for the high side, and a last one to trigger a duty cycle reload for the high side ATOM channel. The high side channel would, when receiving a trigger, fetch a new duty cycle and period configuration from a PSM configured in ring buffer mode. The CPU controls the pulse length and period of the PHPWM pulses by changing the period and duty cycle in the low side and the trigger channels.

Triggering of the ADCs for the remaining pressure sensors could be done the same way as for the rail pressure sensor; an ATOM is configured to produce a pulse every 15th degree based on the angle clock.

5.6 Prototype

We have implemented the design proposed in Section 5, forming a proof-of-concept prototype. The purpose of the prototype was to show that the design is capable of producing the correct functional behavior as well as to verify the timing requirements. Thanks to the GTM's modular design with replicated hardware, control of one injector and a working angle clock are enough to show that the GTM can perform the intended tasks. Adding support for more injectors would not affect the performance of the already existing parts in the GTM, as long as there are enough modules available.

5.6.1 Hardware

Figure 5.5 gives a overview of the hardware setup we used for development and testing. Volvo's current system runs on the hardware platform called EMS, which is developed and produced by TRW in close cooperation with Volvo. The EMS includes the FETs required for driving solenoids with battery and boost voltage. It also has the circuits needed to measure the current going through these solenoids and for short circuit detections in the high side FETs (HSSD). The current sensing

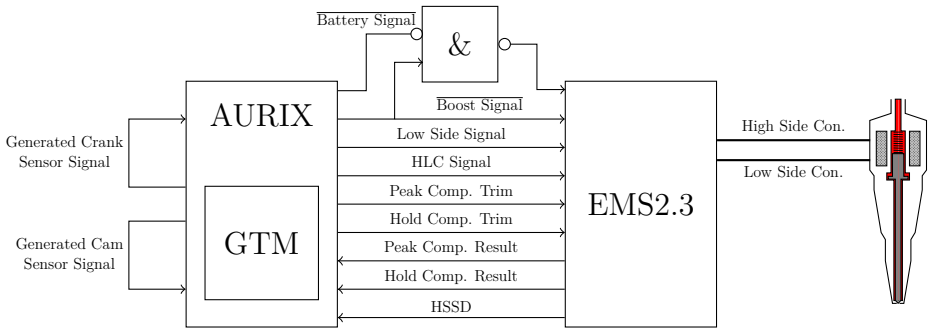


Figure 5.5: Overview of the hardware setup used for development and testing.

circuit is combined with comparators that signal when the the current going through the solenoids passes a specified threshold.

As mentioned in Section 3.4 we used an AURIX TC2X7 development board as the hardware platform for the prototype. To this board we connected one each of the above mentioned signals, allowing us to drive one real injector. The EMS microcontroller was loaded with software that left its connections floating to remove any potential interference with the AURIX's logic signals.

The EMS which we had access to did not have Volvo's feature that enable an automatic switch between boost and battery voltage and thus we implemented our own version of the feature. Our implementation was made of 7400 logic gates and was placed in the signal path between the GTM and the EMS drive stage. The implementation allows the boost and the battery signals to both be active at the same time, but only one of the high side FETs is activated. The boost signal has precedence and suppresses the battery signal. As soon the boost signal is deactivated the battery signal will propagate to the drive stage as normal.

5.6.2 Testing

Before the prototype was moved to the hardware platform it was tested on the GTM-RM simulator. This allowed for fast initial development as the software platform allowed for more insight in the operation of the GTM than the hardware would have. It also made sure any initial errors did not result in broken hardware. When the behavior of the prototype running in the GTM-RM was satisfactory the prototype was moved to the AURIX board on which further testing was carried out.

Stimuli for the cam and crank sensor inputs was needed to test the behavior of the angle clock. The stimuli was generated by the GTM itself with the use of two PSM channels in ring buffer mode and two ATOM channels reading values from the buffers over the ARU. The ring buffers were populated with PWM data to produce cam and crank logic signals representing an engine speed of 5000 RPM. To gain more realistic signals, a 5% sinusoidal variation of the engine speed with three times the period were added. This is to simulate the speedup of the engine after each combustion as well as the slowdown in between.

The way of generating cam and crank sensor signals works for both the hardware platform and the GTM-RM, but for the injection control input stimuli two different solutions were used. When testing on the GTM-RM no real injector drive hardware

could be connected, and as the signal generation is feedback driven, interrupts from the ATOM channels were used to control TOM channels emulating hardware signals. For the testing performed on the prototype a Delphi F2 injector was connected via the EMS drive stage and the resulting behavior was inspected on an oscilloscope. To remove the need for recompilation and downloading of the software between every test, the asynchronous serial interface of the ARUIX TC277 was utilized to send commands to the prototype and to receive debug printouts.

6
Results

In this chapter we present how the prototype performed with the input stimuli described in Section 5.6.2. The prototype's performance is presented using several figures and Table 6.1 give a description of the signals seen in these figures. Last we also show the resource utilization of the prototype.

Table 6.1: Description of the different signals shown in Figure 6.1, 6.2, 6.4, and 6.3.

Analog signal	**From**	**To**	**Description**
3 (Blue)	EMS	Injector	Current passing through injector
1 (Orange)	EMS	Injector	Voltage over injector
Logic signal	**From**	**To**	**Description**
D6	GTM	EMS	Hold level control
D12	EMS	GTM	Peak comparator flag
D11	EMS	GTM	Hold level comparator flag
D5	EMS	GTM	HSSD (active low)
D4	GTM	EMS	Battery (active low)
D3	GTM	EMS	Boost (active low)
D2	GTM	EMS	Low side
D1	GTM	GTM	Cam sensor
D0	GTM	GTM	Crank sensor

6.1 Angle clock

To verify the functionality of the angle clock, injection events were scheduled on specific angles. The injections were queued by the CPU when it received the crank-slit zero interrupt from the DPLL and they were set to start 20 degrees before the TDC of the current cylinder. This was done for one full revolution of the crank shaft resulting in three injections 120 degrees apart. The length of the injections were set to 30 degrees long, which at 5000 RPM corresponds to 1.2 ms.

Figure 6.1 shows the result from the above test. As it can be seen, all three injections start slightly after the 7th crank signal for each cylinder, corresponding to 20 degrees before TDC. The length of the injections are five slits long, which corresponds to $5 \times 6 = 30$ degrees.

For verification of the engine speed calculation crank and cam signals were generated for three different RPM; 1000, 2500, and 5000, to which the calculated values were compared. For all three speeds the calculation was off by 0.5%, and the reason for this will be brought up in Section 7.

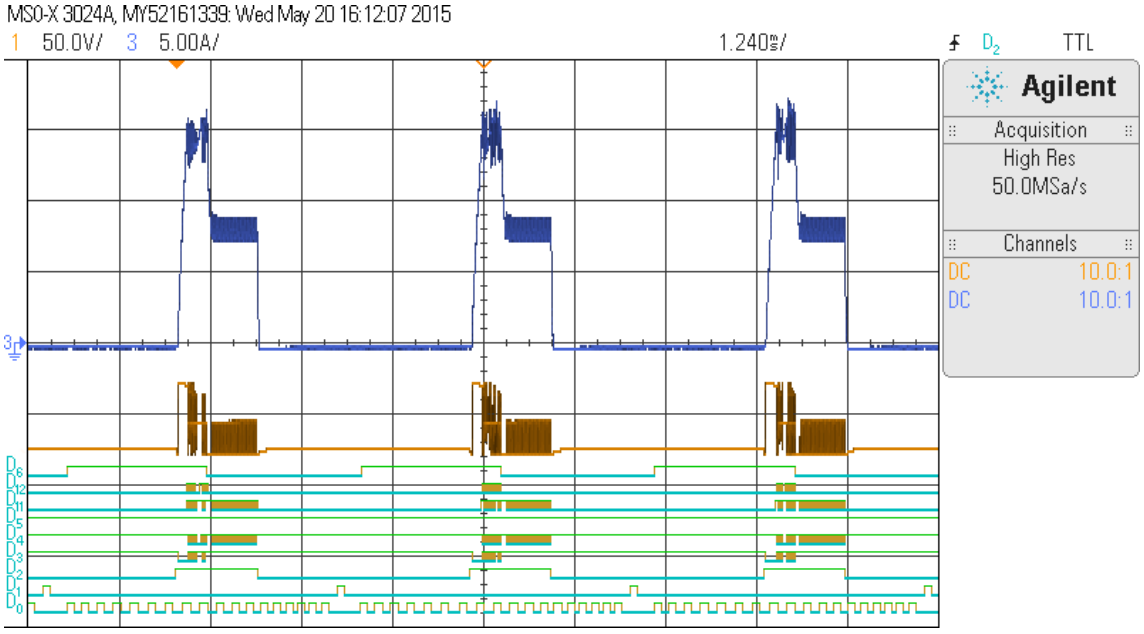


Figure 6.1: Capture of three injector pulses that have been scheduled on the crank slit zero interrupt to start 45 degrees later.

6.2 Needle valves and spill valves

For the control of needle and spill valves two different designs were presented in Section 5.3 and in this section we show how both designs performed.

A typical current waveform produced by the GTM-only design and the Hybrid design are displayed in Figure 6.2 and Figure 6.3, respectively. In the figures, cursors have been placed to indicate the four I_x requirements from Table 5.1 in Section 5.1. As seen, the GTM-only design meets all these, with the exception of I_{peak} chop for which it can be seen that the ripple is a roughly 1 A larger than the required maximum of 4 A. The Hybrid design meets all the I_x requirements with a small margin for the I_{peak} chop. The time requirements have not been marked, but it can be seen that these are fulfilled for both designs.

The busy-wait loop, mentioned in Section 5.3.1, was set to 2 µs. As seen in both waveforms, this wait time was long enough for the peak flag to go low and thus prevent the battery voltage from being switched off, but it also introduces an elevation of the current in the later part of the peak phase, after the boost voltage is switched off, which is not desired.

The latencies measured correspond with those mentioned in Section 5.3.5: how long time it takes for the system to activate the boost signal again after I_{peak} is reached initially; how long time it takes during the peak phase and hold phase until

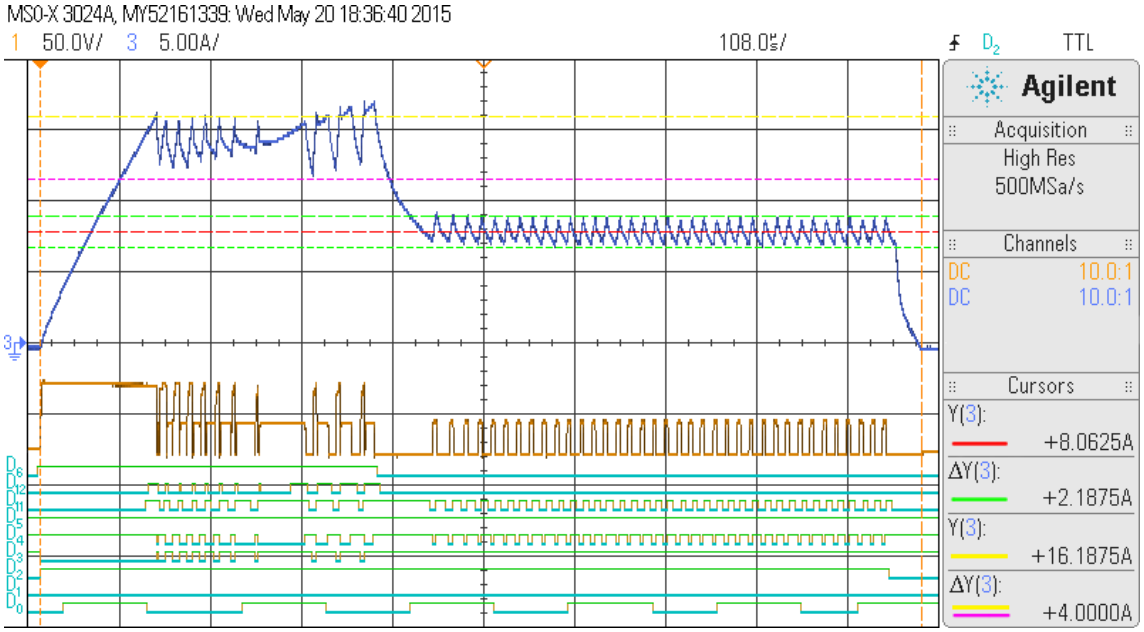


Figure 6.2: Capture of a current waveform generated by the GTM-only solution with cursors indicating the vital points.

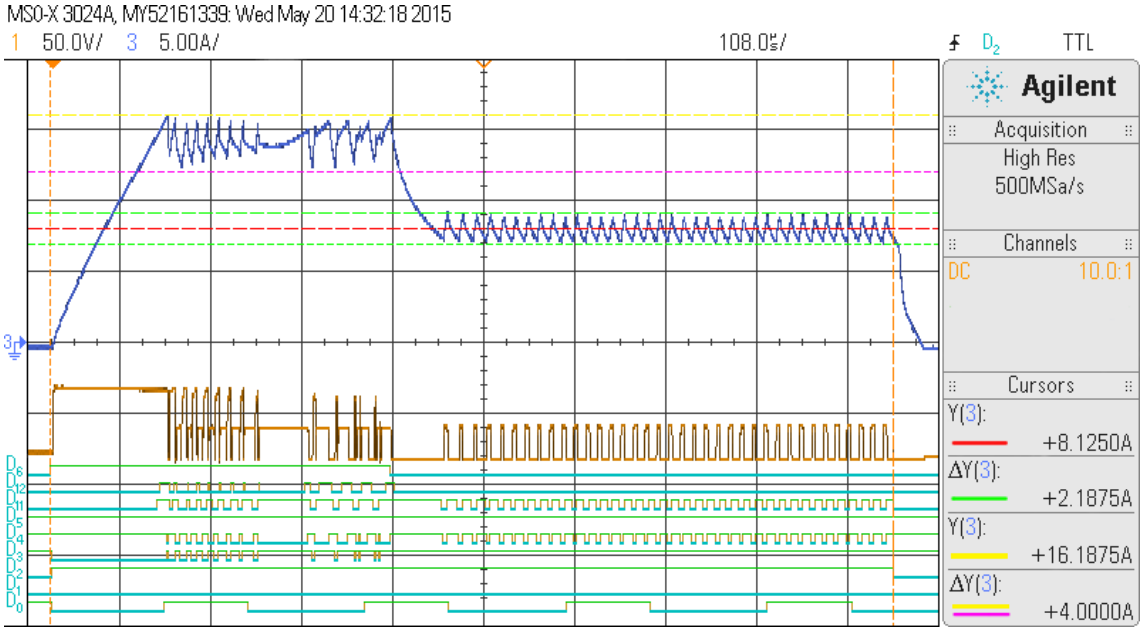


Figure 6.3: Capture of a current waveform generated by the Hybrid solution with cursors indicating the vital points.

the high side is activated after the current has dropped below the hold level; and how long time it takes to perform an emergency shut down when the HSSD signal is activated. Figure 6.4 shows an example of how the latency is measured for the peak phase and hold phase. The latencies for each design are summarized in Table 6.2. Recall that the peak phase and the hold phase has the same critical WCRT, and by the same argumentation as in Section 5.3.5, the critical latencies are the same for

the peak phase and hold phase. Thus the values for the two phases are the same in this table as well. The values in parenthesis indicate how close the measurements or estimated WCRTs are to the available time presented in Table 5.2, with the switching time for the FET removed.

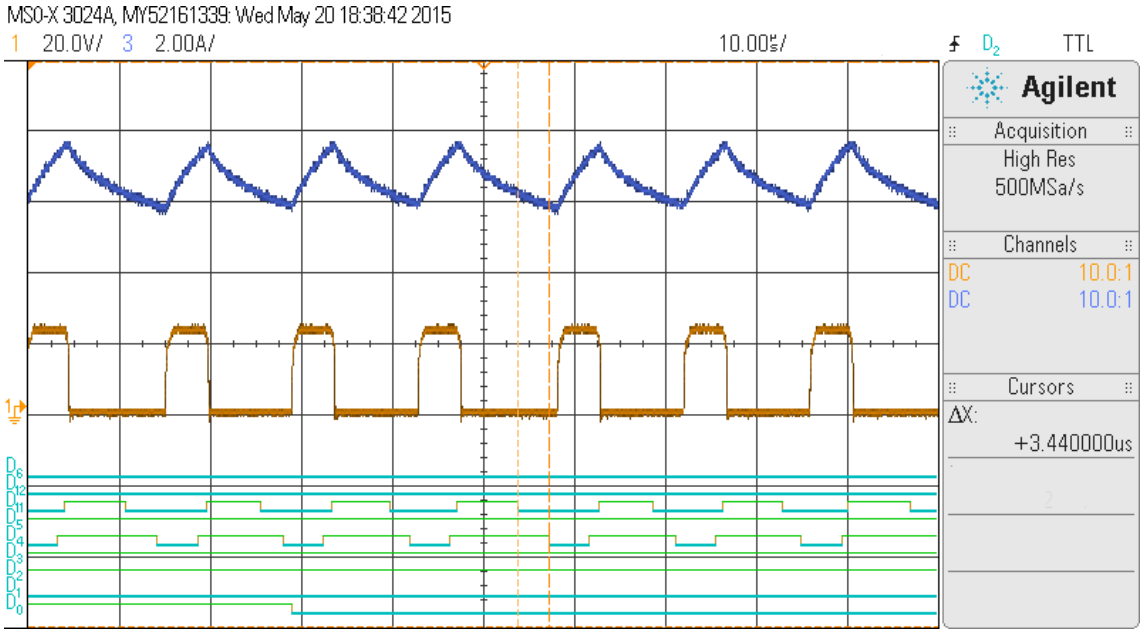


Figure 6.4: Zoom-in on Figure 6.2 showing the latency introduced by the GTM-only solution.

Table 6.2: Measured worst case latencies on both designs and the estimated WCRTs.

Section	GTM-only design		Hybrid design	
	Latency	Est. WCRT	Latency	Est. WCRT
First drop	2.3 µs (-0.7)	2.8 µs (-0.2)	0.82 µs (-2.2)	1.8 µs (-1.2)
Hold & Peak phase	3.4 µs (-1.3)	3.8 µs (-0.9)	1.8 µs (-2.9)	1.9 µs (-2.8)
HSSD	2.0 µs (+1.0)	3.1 µs (+2.1)	0.34 µs (-0.66)	0.35 µs (-0.65)

The measured latencies are all shorter than the estimated WCRT, some are quite accurate while others differ as much as 50% from the estimations. The largest reason for this result is, as discussed, that the ARU arbiter is not always in a random position when it is employed the second time in a critical path. To give a better understanding of the arbiter problem we highlight the measurement for the Hybrid design's first drop. After the command to deactivate the boost signal has been transfered by the ARU, the arbiter will always be in the same location when the MCS executes the readback command. Evidently the time between the ARU arbiter serving the MCS write address and the ATOM write address is large enough to let the readback be served on the same arbiter round trip. Yet, the time span is still small enough to let the MCS to also execute the ARU write for boost reactivation

before the beginning of the next arbiter round trip. These circumstances enables all three ARU transfers to be performed on one round trip time, thus the latency is always the same. The additional 820 ns − 730 ns = 90 ns on the response time comes from the latency introduced by the ATOM and the actual transfer over the ARU, as well as inaccuracy in our time measurements.

The waveforms from the two designs are produced with nearly identical configurations. The comparators have the same reference values and the length of the phases are identical; the length of the peak phase is set to 400 µs and the length of the envelope to 1 ms. What differs between the designs are the values regarding the length of a peak chop. As the GTM-only solution has longer response time, the current decays more than in the Hybrid design and the chop must thus be longer in the GTM-only design for the current to reach I_{peak}. A longer chop time is not needed during the hold phase as the longer latency only lowers the I_{hold} Mean level and does not affect the peak-to-peak value. The configuration values relating to the peak chop are presented in Table 6.3.

Table 6.3: Configuration values which differentiates between the designs.

	GTM-only design	**Hybrid design**
Peak Chop	9.0 µs	8.5 µs
Decrease per Peak Chop	0.8 µs	0.6 µs
Number of Decreases	9	7

6.3 Resource utilization

Table 6.4 shows the utilization of the GTM's resources for the designs presented in the previous chapter. The utilization is calculated based on a GTM device 103, as it was the configuration available in the prototype's microcontroller. For the GTM-only design the usage of MCS channels is 100% if it is assumed that the PHPWM function is implemented as the MCS-less approach presented in Section 5.5. Otherwise, there would not be enough MCS channels and the larger device 104 would be needed to realize the design. Recall that the two trim signals are shared by two banks, thus only four TOM channels are needed for needle and spill valve control. Concerning the pressure sensor measurement it is enough with one ATOM channel to trigger all ADC conversions as groups of VADC channels in the AURIX can be triggered by a single interrupt source.

The MCS memory usage is moderate. If the GTM-only design is employed for the needle and spill valve control, 1.4 KB of MCS memory is utilized. If instead the Hybrid design is used, 0.9 KB is needed. These memory values correspond to 23% and 15% usage of the default 6 KB combined data and code space for one MCS instance, respectively. The angle clock with its two MCS channels use 0.4 KB of memory which corresponds to 7% of 6 KB.

The resource usage outside of the GTM for our designs compared to the current EMS system are almost the same. The difference is that the Hybrid design needs

12 more DMA channels. Additionally, any changes in execution time demands on the CPU is hard to determine, but our estimation is that during runtime the needs should be about the same between the current system and a GTM system. To determine the differences more exact, we leave for future work.

Table 6.4: GTM-resource utilization for our two designs on a GTM device 103. The number of available channels is placed within parenthesis after the module acronym.

Function or design	**MCS (32)**	**TOM (32)**	**ATOM (40)**	**TIM (32)**	**PSM (8)**
Angle clock	2	0	0	4	0
GTM-only / Hybrid	28 / 4	4	24	12	4
PHPWM	0	0	6	0	2
PWM RMS	2	0	4	0	0
PWM generation	0	22	0	0	0
PWM measuring	0	0	0	4	0
Sensor measuring	0	0	1	0	0
GTM-only total	32 (100%)	26 (82%)	35 (88%)	20 (63%)	6 (75%)
Hybrid total	8 (25%)				

7
Discussion

In this chapter we first elaborate about strengths and weaknesses regarding the GTM. Next the angle clock is covered, where we relate our design to how it differentiates toward Volvo's or a general design employing the eTPU. The last discussion is about the needle and spill valve control, where we evaluate our two designs and reason about design decisions.

7.1 GTM architecture

The Generic Timer Module assessed in this thesis is a new approach to how a timer module works. Its modular design makes it easy to add and remove features in a system, without affecting the existing ones. Compared to the eTPU, in which the addition of a new task may cause the other ones to fail, as all 32 channels in the eTPU shares one execution pipeline. On the other hand, the eTPU has a much more powerful instruction set than the GTM which, for example, can not perform division or multiplication. This makes the signal processing capabilities in the GTM's processing core a bit more limited. However, the GTM's different IO modules all have hardware accelerated signal processing capabilities on their own, while in the eTPU all processing has to be done with code.

When performing real-time tasks with short deadlines as those assessed in this thesis the latency introduced by the ARU is a large problem. To use the GTM for what it is intended, offloading the CPU, data has to be transported internally between its modules using the ARU. One could use the CPU to move data between the modules, but that would defeat the purpose of the timer module. However, in the upcoming generation of the GTM it will be possible to circumvent this latency issue. This generation will also address the issue of not being able to perform division or multiplication in the MCS, which will make the processing capabilities of the MCS much more powerful.

An additional downside with the GTM is that there is a fairly limited number of ways a MCS channel can be triggered. The possible ones are to write to the trigger register with the CPU or with another MCS channel in the same instance. Issuing a ARU read instruction will also put the MCS channel to sleep, but then only the channel read from is able to wake it. This means that if we want to, for example, monitor more than one input we have to use the non-blocking ARU instruction and poll all inputs in a loop. On this point the eTPU has the upper hand as it is very flexible in how, and by what conditions, it can be triggered. However, the next generation GTM will improve on this as well, with the possibility to receive

interrupts in the MCS.

When it comes to latency and throughput we believe the GTM and eTPU have different strengths. The GTM can, thanks to its design with independent hardware modules, sustain a higher throughput and a more deterministic latency. On the other hand can the eTPU achieve a lower minimum latency, but if to many events happen at once the latency can jitter a lot. An example of this is if we setup both timer modules to receive 32 inputs at the same time. The GTM would process all 32 inputs in parallel and the latency for all signals would therefore only depend on their own execution time. For the eTPU the highest priority channel's latency would also only depend on its own execution time, but the lowest priority channel's latency would be bigger than the execution times of the 31 other channels combined.

7.2 Angle clock

In Section 6.1 we mentioned that the engine speed calculation returns a speed offset of 0.5%. Recall that we have added a sinusoidal variation to the generated cam and crank signals with a period equal to that of 20 slits. As the engine speed is calculated as an average of the last 32 slit periods and we do not include the period for the missing slits or the one before them, we get an average variation that is non-zero. The mentioned period is not added as it is three times bigger than the normal periods and the MCS can only perform division with denominators that are a power of two. One solution to this problem could be to combine the period for the missing slits with the period for the slit before. The new value gained would then hold the period of four slits, and can thus be shifted right two steps to get an average of them. If this average then is used instead of the period for the missing slits, the error will become extremely small.

For the proposed angle clock design we have dedicated two MCS channels, one for engine speed calculation and one for base angle updating. As can be seen in Section 6.3 this will in conjunction with the GTM-only design for needle and spill valve control result in 100% usage of the MCS channels. This could pose a problem if some more task where to be added. However, in such case the base angle and the speed calculation MCS channels could be combined, freeing up one channel for other work. Combining the channels would not have a significant effect on the calculations due to the relatively long time scale; at 5000 RPM there is 200 µs between the input events, allowing for more than two thousand instruction to be executed. The reason they are separated in our design is that it lowers the complexity and the code becomes more modular.

The micro tick generation in the GTM is mainly performed by the DPLL. As the DPLL is a hardware module that has been specifically designed to handle tasks like this, the user gets much functionality from the start. Also, an all out hardware implementation can handle high rotational speeds while still producing micro ticks with an accurate frequency. On the other hand, an all out hardware design that should handle diagnostics, error detection, and input anomalies can become very complex. The error detection in theDPLL is to a large degree autonomous, but it depend on timeout values that need to be updated by the CPU as the engine speed changes. Also, synchronization needs to be managed by the CPU, increasing its

workload, but as it is a one-time task performed at system startup it may not be a large concern.

In the eTPU, several parts of the angle clock implementation are also handled by software. Yet, the eTPU executes this software on its own, thus it lays no burden on the CPU and it becomes more self-sufficient. This will also be the case for the next generation GTM as the MCS channels then will be able to receive interrupts and use the AEI. It will then be possible to handle synchronization and error detection completely in the GTM.

One advantage with the DPLL is that it generates very accurate predictions of the upcoming slit period using up to two full revolutions of old input data. This translates into the generated micro ticks being very accurate in terms of their placement. Advanced period predictions are possible with the eTPU as well, but it then has to be performed in software and will thus consume valuable execution time. Another advantage of the DPLL is that it can seamlessly switch from generating micro ticks based on the crank or the cam input signals. This results in a smaller loss of accuracy in the event of a crank sensor failure than for the eTPU which then has to switch to time based operation.

Based on our findings during this work we believe that the GTM can cater for Volvo's need in terms of angle clock functionality. The same can be said about engine speed calculations if the suggested fix is applied to decrease the error, as the accuracy then will be on par, or even slightly better, with the accuracy achieved in the eTPU. This is because the eTPU solution creates two virtual slits based on the period of earlier slits while our design, with the added fix, uses the actual slit period for the missing slits.

7.3 Needle valves and spill valves

All requirements are fulfilled regarding the current waveform, except that the GTM-only solution has a to high ripple during the peak phase. We believe that by tweaking parameters and adding a new feature that is used in the peak phase, it will be no problem to keep the current within the ripple boundary. The feature we propose is to not apply boost voltage after that the battery voltage has elevated the current to I_{peak}, but instead always apply battery voltage when the current has dropped either for a static amount of time or when it has decayed past the hold level. When the current then reaches I_{peak}, the voltage is switched off. This new feature will not introduce a longer critical path during the first part of the peak phase when battery decay is employed if it is assumed that the master channel does not detect that I_{peak} is reached until the battery voltage elevates the current. This assumption was held during our experiments as seen in the result figures.

It may not be needed to have a system with shorter response time than the GTM-only design when producing the current waveform. When observing the current waveforms for the two designs, it is clear that the reduced latency for the Hybrid design did not affect the waveform appearance to a higher degree. Thus it may not be necessary to strive for a design with a shorter latency than the GTM alone can produce.

As indicated by our results both designs have the potential to produce correct

current waveforms. When we instead look at the 1 µs latency requirement for emergency shutdown, the Hybrid design has 0.66 µs to spare. This extra time may be needed as DMA is used and there may be another ongoing transfer when the HSSD signal is activated. Yet, if special care is taken when the system is designed, it should be possible to ensure that the DMA is served in time. On the other hand, the GTM-only has a 2 µs long latency, making it unfit to use in a real system. However, as the design otherwise is capable of generating a correct current waveform it could still be used if we introduced HSSD handling using DMA. This would also have the added benefit of freeing up the four MCS channels used for monitoring HSSD signal and sending data to the ATOM channels. Also, by not using these channels we would not need to trigger them from the master channel, thus removing 11 instruction cycles from the critical path and lowering the WCRT with 990 ns.

In our designs we have not included all software diagnostics performed in the Volvo system, this as our designs are a proof-of-concept and focused on basic functionality. Still, we are positive that the diagnostics can be included without affecting performance, for example, it can be inserted at the busy-wait loop after reaching I_{peak}, or during a chop. As can be seen in Table 5.5 there is room for a few additional instructions in the critical path for both designs and it would thus be possible to insert some degree of diagnostics there as well.

In their current state both of the proposed designs have a fairly low memory utilization. The GTM-only design only uses about 23% of the default available memory, while it occupies seven out of eight MCS channels in one instance. For the Hybrid design the memory usage per MCS channel is considerably higher, 15%, but four replicas of the design would still fit in the default memory of one MCS instance. However, the addition of diagnostics and error handling would claim additional memory space. Still, the Hybrid design has room left for 155 more instructions for each bank, which we believe will be more than enough for the addition of diagnostics and error handling. If it is not, additional memory can be taken from other MCS instances, or the banks can be partitioned into different MCS instances.

The designs' latencies vary relatively much due to the time jitter introduced by the ARU. As the ARU arbiter likely is in an arbitrary position when a MCS channel execute an ARU read or write, a delay is introduced in the interval from zero to one ARU RTT. As a result, the response time is not static which leads to the waveforms produced becoming inconsistent. However, the jitter can be circumvented when transferring commands to ATOM channels by setting the compare match to occur at least one ARU RTT from the time when the ATOM command is provided to the ARU. The disadvantage of preventing the jitter in this way is that the system average response time will be higher, but it may be preferable as the waveform appearance becomes more consistent.

If the Hybrid design was targeted towards a third-generation GTM it could be designed without the use of DMA, while still having the same GTM resource utilization. Additionally, the response time would be shorter than for our current design. An improvement brought by the third generation that would aid the transfers of the TIM results is the possibility to customize the ARU arbitration sequence. Using this, the transfers from the TIM channels to the MCS master channel could be done over the ARU, but with a much lower latency than today. Using such a

solution would make it possible for the master channel not to use busy-wait loops when waiting for input, but non-blocking ARU instructions instead, lowering power consumption. In the third generation, the MCS has an interface to the AEI and it can be used to control the ATOM channels, eliminating one ARU RTT of latency. Another possibility to aid TIM result transfers is to utilize interrupts from the TIM channels to notify the MCS when they receive an input edge transition. Then the MCS channel could access the TIM result via the AEI. This would remove the need for the ARU completely, but may not be the best possible resource utilization. Also, interrupts may introduce unwanted jitter.

If a lower response time is required from our designs, a option is to employ the accelerated scheduling scheme instead of the round robin scheme. The accelerated scheduling will cause the response time to be dependent on the number of executing channels. For the Hybrid design the majority of MCS channels is unused. If we assume that no more MCS channels will be needed than those we estimated, we could partition two tasks for each MCS instance. At the worst it would then be only two channels and the CPU that competes for the instruction cycles, thus decreasing the length of an instruction cycle from 90 ns to 30 ns. Also for the GTM-only design the accelerated scheduling scheme would be able to lower the latency. This is because six of the used MCS channels are sleeping while they are waiting for input events, or output commands. Thus, by employing the accelerated scheme the nominal time between two instruction cycles for a MCS channel could be decreased to 30 ns also for this design. The disadvantage of using the accelerated scheduling scheme is that execution time jitter will appear.

As brought up in this section, the task of needle and spill valve control is possible to perform with the GTM if it is aided by DMA. The subtask that has to be aided by DMA is the emergency shutdown. The disadvantage of employing DMA is that it is a source of indeterministic behavior. In our experiments we encounter no such problems as DMA move engines always were available when needed. When designing a system as the EMS, thorough planing is vital to ensure that all DMA transfers are served within each deadline. It is not simple to guarantee that the deadlines are fulfilled and it would be better if a solution was adopted that did not rely on DMA for time critical transfers. By this argumentation we do not want to employ DMA more than needed. We thus propose that the improved GTM-only design discussed above is the best design choice. Also, we believe that this design has the best compromises between waveform consistency, response time, and resource utilization.

7.4 Ethics

The GTM has potential to become an automotive industry standard. The major reasons for this are that the GTM: is distributed as an IP-core which makes it microcontroller-platform independent, has an architecture that probably can replace the conventional two types of timer modules, and is under continuous development. Also, Bosch is not on unfamiliar territory regarding providing an industry standard and has provided this before where Controller Area Network (CAN) [17] is an example.

If the GTM becomes the standard timer module within the automotive industry, it will lead to both advantages and disadvantages. An advantage will be that knowledge and development only have to be concentrated on one timer module. The disadvantage if the GTM becomes a standard is that it will lead to less competition between different timer modules, which may inhibit the research progress within the field. Also, when one company holds the major market share it will be hard for other companies to come into the market.

7.5 Future work

This thesis is an initial study of the GTM to conclude if it possibly can replace the currently used timer modules within Volvo powertrain control. Our research show that the GTM can perform the most real-time critical task within powertrain control and this result indicates that all other timer tasks within the system probably can be performed in the GTM as well, as their response time requirements are lower. To be certain that the GTM indeed can perform all timer tasks within powertrain control for the Volvo system, further evaluation is required.

We made our study of the GTM with focus on response time and evaluations targeting other areas are needed before it is clear that the GTM is a good option for next generation of Volvo ECUs. An area which is important to cover is how the GTM can comply with the functional safety standard ISO 26262 [18], as a revision of the standard that also includes medium and heavy duty vehicles is expected to be released in 2018. Other areas to cover in future analyses are for example performance and CPU resources needed.

8

Conclusion

In this thesis we have evaluated the feasibility to use the Generic Timer Module for control of timer-module tasks in a truck powertrain. To verify our findings we developed two proof-of-concept designs for how the GTM could be used to generate the signals needed for driving a Delphi F2 fuel injector. We also implemented the designs in a hardware proof-of-concept prototype to verify their functionality.

We have found that the GTM is very capable and can produce deterministic behavior, but it struggles in the 1 µs time scale that this thesis is aimed at. This conclusion is drawn from our results, which indicate that the first-generation GTM would not be able to meet the timing requirements for all tasks on its own. It was able to generate a correct current waveform, but the response time for emergency shutdown when a short circuit occurred were too long. However, with the aid of a DMA module to reduce the latency, the GTM was able to meet all the requirements.

Bosch, who develop the GTM, are releasing a third-generation GTM that will be available on the market in a few years, and with the improvements this generation brings the GTM will be able to meet the requirements setup in this thesis without aid.

Bibliography

[1] Volvo Truck corporation, "Diesel Engine Fundamentals Classroom Training," 2007, internal document, Slideshow presentation.

[2] Freescale Semiconductor Inc., "Enhanced time processing unit (eTPU) preliminary reference manual," May 2004, Rev. 1, Accessed: 2015-04-08. [Online]. Available: http://cache.freescale.com/files/32bit/doc/ref_manual/ETPURM.pdf

[3] Robert Bosch GmbH, "GTM-IP Specification," Jun. 2013, Rev. 1.5.5.1, Accessed: 2015-04-08. [Online]. Available: http://www.bosch-semiconductors.de/media/pdf_1/ipmodules_1/timer/GTM-IP_Specification_v1551.pdf

[4] Volvo Group Trucks Technology, "Developer reports for P3151E_VGT," internal document, revision BSW3.14F.62.

[5] Delphi Diesel Systems, "Product Design Specification," jan 2015, Internal document.

[6] J. Cheever and D. Penrod, "Fuel injector control system and component for piecewise injector signal generation," Jun. 2014, EP Patent App. EP20,130,193,798, Accessed: 2015-04-01. [Online]. Available: http://www.google.com.ar/patents/EP2738375A2?cl=en

[7] Freescale Semiconductor Inc., "Mpc5674f microcontroller reference manual," Feb. 2015, Rev. 7, Accessed: 2015-05-11. [Online]. Available: http://cache.freescale.com/files/32bit/doc/ref_manual/MPC5674FRM.pdf

[8] Robert Bosch GmbH, "Bosch GTM-IP Overview," Nov. 2014, AE/PJ-IPT 2014-11-13 Robert Bosch GmbH, v.3.1.1.

[9] Freescale Semiconductor Inc., "Fact Sheet Qorivva MPC5777M MCU," 2014, Rev. 1, Accessed: 2015-06-10. [Online]. Available: http://cache.freescale.com/files/32bit/doc/fact_sheet/MPC5777MFS.pdf?fpsp=1

[10] Delphi, "Delphi F2E High Pressure Heavy Duty Diesel Common Rail System," Accessed: 2015-04-08. [Online]. Available: http://delphi.com/docs/default-source/old-delphi-files/cbefef53-e98a-4b4d-be04-57ea6536ede0-pdf

[11] K. Reif, Diesel Engine Management: Systems and Components. Wiesbaden: Springer Gabler, 2014.

[12] R. Kusakabe, M. Abe, H. Ehara, T. Ishikawa, T. Mayuzumi, and T. Miyake, "Injection quantity range enhancement by using current waveform control technique for DI gasoline injector," SAE Int. J. Engines 7(2), pp. 560–567, Jan. 2014.

[13] Y. Sukegawa, Y. Kihara, N. Youko, and K. Kumano, "Examination of low emission technology for gasoline direct injection engines by means of split injection," Society of Automotive Engineers of Japan (JSAE) annual Meeting 2008-08-02, 2012, 52-52Technical Paper 2012-01-0082, 51-52, In Japanese.

[14] M. Abe, N. Maekawa, Y. Yasukawa, T. Ishikawa, Y. Namaizawa, and H. Ehara, "Quick response fuel injector for direct-injection gasoline engines," Journal of Engineering for Gas Turbines and Power, vol. 134, no. 6, p. 62803, 2012, Accessed: 2015-03-11. [Online]. Available: http://dx.doi.org/10.1115/1.4005995

[15] Robert Bosch GmbH, "Bosch GTM-IP Generation 3," Nov. 2014, AE/PJ-IPT 2014-11-14 Robert Bosch GmbH.

[16] Infineon Technologies AG, "AURIX TC27x C-Step User's Manual," Dec. 2014, v2.2.

[17] Robert Bosch GmbH, "CAN Specification," Stuttgart, 1991, Ver. 2.0, Accessed: 2015-06-08. [Online]. Available: http://www.kvaser.com/software/7330130980914/V1/can2spec.pdf

[18] The International Organization for Standardization, "ISO 26262 Road vehicles — Functional safety," Geneva, 2011.

Glossary

ACM Aftertreatment Control Module.

ADC Analog to Digital Converter.

AEI Generic Bus Interface.

ALU Arithmetic Logic Unit.

ARU Advanced Routing Unit.

ATOM ARU-connected Timer Output Module.

BLDC Brushless Direct Current.

Bosch Robert Bosch GmbH.

BRC Broadcast Module.

CAN Controller Area Network.

CMP Output Compare Unit.

CMU Clock Management Unit.

core Processing unit that performs execution of code.

CPU Central Processing Unit.

CR Common-Rail.

Delphi Delphi Technologies, Inc..

DMA Direct Memory Access.

DPLL Digital Phase Locked Loop.

ECU Electronic Control Unit.

eMIOS enhanced Modular Input/Output Subsystem.

EMS Engine Management System.

eTPU enhanced Time Processing Unit.

FCM Fast Compare Mode.

FET Field Effect Transistor.

FIFO First In First Out.

Freescale Freescale Semiconductor.

GPIO General Purpose Input Output.

GTM Generic Timer Module.

GTM-MX Signal Multiplexer.

GTM-RM GTM Reference Model.

HLC Hold Level Control.

HSSD High Side Short Circuit Detection.

ICM Interrupt Concentrator Module.

Infineon Infineon Technologies.

IO Input Output.

IP Intellectual Property.

LS Low Side.

MAC Multiply and Accumulate.

MAP Input Mapping Module.

MCS Multi Channel Sequencer.

MDU MAC and Divide Unit.

microcontroller One or more processing cores in a package including other peripherals, like memory and IO modules.

MON Monitoring Unit.

PCM Pulse Count Modulation.

PHCWG Peak and Hold Current Waveform Generation.

PHPWM Peak and Hold PWM.

PSM Parameter Storage Module.

PWM Pulse Width Modulation.

RAM Random Access Memory.

ring buffer Buffer that loops around at the end forming a circular array..

RISC Reduced Instruction Set Computing.

RMS Root Mean Square.

round robin Scheduling schema were all involved parts are served in order, independent of need for the scheduled resource.

RTT Round Trip Time.

SCM Shared Code Memory.

SPE Sensor Pattern Evaluation.

SPRAM Shared Parameter RAM.

TBU Time Base Unit.

TDC Top Dead Center.

TIM Timer Input Module.

TOM Timer Output Module.

TRW TRW Automotive.

UIS Unit Injector System.

UPS Unit Pump System.

VADC Versatile ADC.

Volvo Volvo Group Trucks Technology.

WCRT Worst Case Response Time.

WURM Wait Until Register Match.