
Design of Cyber-Pysical Systems

Department of Computer Science

TU Kaiserslautern

# Project Report

# Comparison of IEEE Std. 1666, Accellera Std. AMS and TLM extensions using a throttle control unit case study

Thiyagarajan
Purusothaman

January 06, 2014

*Supervisor:*
Prof. Dr. Christoph Grimm,
TU Kaiserslautern

# Abstract

In a modern vehicle, all ECUs consists of distributed control system working concurrently and communicating through networking technologies. To replicate this in a test environment, the open or closed loop lab-car is used to test the ECUs with respect to customer requirements. A lot of time elapses for setting up a test environment for HW/SW developments.The need for faster time to market and high return on investment in automotive industry motivates this work about HW/SW co-design. An executable specification of requirement for system and architectural exploration is developed in this Project work. The challenges are realization of physical models in testing or virtual domain without compromising the accuracy and quality. The following Project work explores and compares the need for abstraction and dedicated models of computation for distributed embedded computers. A unified novel method to HW/SW co-design of mixed-signal and mixed domain system using SystemC, TLM and its AMS extensions is explained. This method also simplifies system design to abstract computation and communication using Discrete Event (DE), Timed Data Flow (TDF), Dynamic TDF (DTDF), Linear Signal Flow (LSF), Electrical Linear Network (ELN) and Transactional Level modeling(TLM) MoCs. These dedicated MoCs enables mixed system modeling faster and more elegant with high quality and accuracy benefits. To prove the safety and reliability of this methodology, an Electronic Throttle Control (ETC) system is developed and experimented as a virtual prototype.

# Motivation

Today's Embedded systems combine digital hardware and software with analog/mixed-signal components such as sensors, transceivers, or ADC/DAC. For architecture level design, such systems are modeled using IEEE Std. 1666 (SystemC). In the lectures of my advisor, Prof. Grimm, I also learned about ongoing processes in IEEE DASC to extend IEEE 1666 to also cover Accellera/OSCI Standard SystemC TLM and AMS. Both SystemC (IEEE 1666) TLM and AMS extensions offer quite different means to model communication at a high level of abstraction. The TLM extensions are used for modeling asynchronous transfer of data packets from a communication master to a slave. The AMS extensions are used for (synchronous, timed) data-flow modeling of communication systems. In this project work I would like to compare the ability, expressive power, and simulation performance of SystemC and its extensions TLM and SystemC-AMS. To illustrate the benefits of TLM and AMS/timed data flow, a throttle valve control unit is modeled. This involves modeling a SystemC TLM wrapped processor, with SystemC-AMS peripherals like Analog Digital Converter (ADC), Pulse Width Modulation (PWM) and Throttle Valve Actuator model. The combined components forms a virtual prototyping platform for Throttle valve control unit.I am able to conceive this concept by understanding fundamentals of Architecture of Digital Systems by Prof. Dr.-Ing. Wolfgang Kunz.Through this work I would like to give answers to the following questions:

- How similar and different are data flow model of computation (foundation of the AMS std.), and transaction level modeling (foundation of the TLM std.)?
- Can we easily couple models using the AMS' data flow, and TLM std. e.g. by converters? Which errors do occur? Or is it maybe possible to even describe DTDF and TLM as special cases of a more general way to communicate? If so, which changes for future versions of IEEE Std. 1666 can be made? What is the impact on simulation performance?
- Knowing about similarities between TDF and TLM, which useful properties from data flow (e.g. schedulability) can be transferred under which conditions?
- And what all benefits of using SystemC AMS for Mixed Signal and mixed domain modeling of Automotive Embedded Systems.

# Abbreviations

| | |
|---|---|
| **HW** | Hardware |
| **SW** | Software |
| **ETC** | Electronic Throttle Control |
| **TDF** | Timed Data Flow |
| **DE** | Discrete Event |
| **LSF** | Liner Signal Flow |
| **CPU** | Central Processing Unit |
| **ELN** | Electrical Liner Network |
| **TLM** | Transaction Level Modeling |
| **MoC** | Model of Computation |
| **ETC** | Electronic Throttle Control |
| **PI** | Proportional / Integral |
| **PWM** | Pulse Width Modulator |
| **ECU** | Electronic Control Unit |
| **ASIC** | Application Specific Integrated Circuit |
| **DLL** | Data Link Layer |
| **DSP** | Digital Signal Processor |
| **AMS** | Analog Mixed Signal Systems |
| **EDA** | Electronic Design Automation |
| **ESL** | Electronic System Level design |
| **FPGA** | Field Programmable Gate Array |
| **FSM** | Finite State Machine |
| **GPP** | General Purpose Processor |
| **HLS** | High-Level Synthesis |
| **IP** | Intellectual Property |
| **RTOS** | Real Time Operating System |
| **FFT** | Fast Fourier Transform |
| **LPF** | Low Pass Filter |
| **PHY** | Physical Layer |
| **RF** | Radio Frequency |
| **SDF** | Synchronous Data Flow |
| **SoC** | System on Chip |

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 HW/SW Co-Design with Virtual Prototypes

The electronic control units (ECU) for real-time automotive applications are designed and implemented with mixed signal components . These systems are heterogeneous in nature and utilizes microprocessors, micro-controllers and digital signal processors for executing real time tasks concurrently. Generally, the features of a system is implemented as software and is flexible to change the features over time. The hardware components of the system are added to increase the performance and concurrent activity of system. Some examples of applications of embedded controllers are [23]

- *Consumer Electronics*: microwave ovens, cameras, compact disk players.
- *Telecommunications*: telephone switches, cellular phones.
- *Automotive*: engine controllers, anti-lock brake controllers.
- *Plant Control*: robots, plant monitors.

The embedded controllers are versatile in many heterogeneous embedded systems. These digital controllers determines the quality of product. Often embedded system industry collaborate with technology partners to increase the quality and quantity with less investments.

### 1.1.1 Problem Statement and Objectives

Design of embedded systems can be subject to many different types of constraints, including timing, size, weight, power consumption, reliability, and cost. Current methods for designing automotive embedded systems requires to specify and design hardware and software separately with these constraints. A specification, often incomplete and written in non-formal languages, is developed and sent to the hardware and software engineers. HW/SW partition is decided a priori and is adhered to as much as is possible, because any changes in this partition may necessitate extensive redesign. Designers often strive to make everything fit in software, and off-load only some parts of the design to hardware to meet timing constraints. The problems with these design methods are :-

- Lack of a unified hardware-software representation, which leads to difficulties in verifying the entire system, and hence to incompatibilities across the HW/SW boundary.
- A priori definition of partitions without deeper understanding of system, leads to sub-optimal designs.
- Lack of a well-defined design flow, which makes specification revision difficult, and directly impacts time-to-market.

Based on above difficulties there is limitation for faster system development for a requirement.This is due to non availability of sufficient and unified environment for architecture exploration and HW/SW co-design. Advent of modeling of Digital and Mixed Signal systems with SystemC and Its Extensions, helps the system designer, HW/SW engineers to exploit this methodology to do better system design for safety and reliability of Automotive System.

System modeling related issues :-

- Often system designers lacks knowledge about multi domain modeling.
- Non unified environment leads to simulation under performance.
- Unclear understanding of MoCs for system design, no guidance and not able to choose DE or AMS based modeling and safe handling of virtual simulation parameters like time steps, sample rate, delays, dynamic features of MoCs.
- Availability of virtual prototyping techniques not clear to system designers.

Objectives of this work for system modeling and HW/SW co-design are :-

- To provide a concrete understanding of different MoCs of SystemC and its AMS extensions for heterogeneous systems modeling.
- To guide multi domain modeling for engineers.
- Develop a executable specification for System exploration and HW/SW co-design for a ETC
- Experiment the interaction between DE and AMS MoCs with TLM interface and guide the designers with results.
- Compare the benefits of system modeling with dedicated MoCs using an ETC case study.
- Illustrate systematically the modeling advantages of using dynamic features and TDF in real time automotive embedded system.

These factors substantiate the need for a Virtual Prototyping platform for Architecture exploration in the initial phase of the system development. With this the hardware and software partitioning can be well analysed with scientific methodology. As this is a virtual prototype, the hardware cost can be greatly reduced. The virtual prototype can be redesigned many times with very minimal cost and time until the architecture exploration satisfies the customer requirement. Even there are lot of tools and virtual

platforms available in market, a unified environment with possibility to model the different components of a dynamically varying automotive system is greatly appreciated.

### 1.1.2 Hardware and Software Views of Parallelism

To understand system level modeling, it is essential to understand the difference in approach to parallelism taken in hardware and software design [12].

A hardware engineer, typically writing in a Hardware Description Language (HDL) such as Verilog, SystemC or VHDL, describes a design as a collection of parallel activities, which communicate via shared data. The parallel activities are always (Verilog), sc_module (systemC) or process (VHDL) blocks. The shared data structures are wires or signals.

This follows very naturally the way that physical hardware behaves. There is no one flow of control of all parallel components are active at the same time, with their individual flow of control.

By contrast, a software engineer typically describes parallelism in a design as a number of threads, which pass flow of control between them. The threads communicate by a number of mechanisms (message passing or remote procedure call for example), but although there is logical parallelism, only one thread is ever physically active at one time.

This follows naturally the behaviour on a conventional uni-processor CPU, where there is a single program counter indicating the next instruction to execute, and so only one flow of control. Even with modern multiprocessors, this is still a natural way of programming for the software engineer, because the number or threads or processes will often exceed the number of processor cores available.

### 1.1.3 Modeling Hardware in Virtual Prototyping System

A simple way to model hardware is via a round-robin, which updates the state of each component as time advances. Each component is represented as a software function. A master clock function calls each component function in turn when the clock advances for example on each clock edge. The wires between the components are represented as variables shared between the components. A number of tools (e.g. ARC VTOC, ARM RealView SoC Designer, Carbon SpeedCompiler, Verilator) use this approach to cycle accurate modeling [12].

With its close parallel of the way hardware is designed with languages such as Verilog and VHDL, this approach has merit for detailed modeling. It is well suited to cycle accurate modeling where every hardware register and wire must be accurate.

Efficiency demands that not every HDL process or always is built as a separate function. Automated tools which generate cycle accurate models in SystemC from HDL can often reduce complex designs to a small number of functions executed on each cycle.

For less detailed models, the overhead in calling each component whenever time advances cannot be justified.

The solution is to model each component only when it has something to do. The individual components communicate by sending messages requesting data be transferred between

each other. The exchange of messages is called a transaction, and the approach Transaction Level Modeling (TLM).

This mirrors the way hardware behaves at the high level, where functional blocks communicate by reading and writing across buses. Towards above discussed facts, the following explanation about SystemC, TLM, AMS and a SystemC Processor models helps in understanding the building blocks of a Virtual Prototyping System(VPS).

### 1.1.4 Simulations: Speed Vs. Accuracy Trade-off

The introduction of AMS abstraction methods, which are facilitated by the models of computation of the SystemC AMS extensions [14], addresses the need to significantly improve simulation speed and efficiency of the AMS descriptions. This marks the beginning of a new era in which AMS and digital HW/SW systems can finally be simulated together.

In terms of simulation accuracy [5], the objective is to capture basic functionality relevant for the intended use case and to validate this against the specification, including AMS behavior. As these AMS system aspects are evaluated as part of a virtual prototype that uses a high-level (e.g., TLM) description of the digital HW/SW system, it is not expected that analog physical effects (e.g., parasitics and substrate noise) should become part of the AMS system model. Instead, dynamic linear and static non-linear behavior related to amplification, mixing, filtering, and so on of analog signals can be captured quite naturally and easily, without having a major impact on the simulation performance.

To enable inclusion of AMS descriptions in virtual prototypes, simulation speed should be comparable to the simulation performance when using TLM. Note that the abstraction and associated accuracy for the AMS description at the system level and circuit level are completely different, because different use cases are served and thus a true one-to-one comparison of the results cannot be given. This being said, Figure 1.1 gives first-order indications toward selection of the most applicable modeling strategy and modeling language.

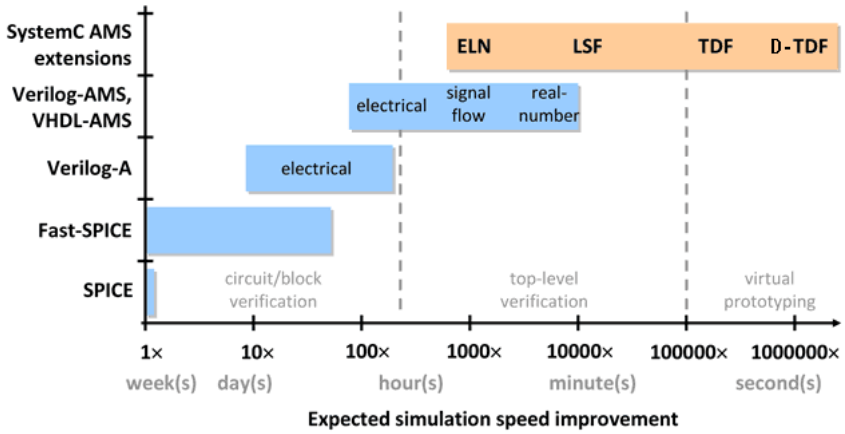


Figure 1.1: Expected simulation speed for different modeling languages compared to SPICE simulation.

Fast-SPICE solutions available in the market today show an improvement of a factor of 5-50 (compared to SPICE) and thus are very valuable for circuit/block verification. The simulation speed improvement offered by Fast-SPICE solutions makes them unsuitable for integration into a virtual prototype. The use of conventional AMS behavioral modeling techniques (e.g., Verilog-A, Verilog-AMS, and VHDL-AMS) including real-number modeling is beneficial for top-level verification, bringing a simulation speed improvement between 100 and 10,000. However, this is still not sufficient for virtual prototyping (e.g., to boot an operating system and to control a software-defined radio baseband processor and analog front-end).

The TDF model of computation offered by the SystemC AMS extensions facilitates a very efficient simulation approach, as TDF models are processed at discrete time points without using the dynamic scheduling of the discrete-event kernel of SystemC. This makes TDF the most powerful modeling style for the creation of AMS descriptions in virtual prototypes.

For a wireline communication system , a Fast-SPICE simulation for the analog front-end takes approximately 15 hours to simulate 1ms of real time. By creating an abstracted description using the SystemC AMS extensions, the simulation takes 1.5 hours to simulate 1 second of real time. Although a true comparison cannot be made because a different abstraction and accuracy have been applied, the values show a ratio of 10,000 in simulation speed.

In the case of applying baseband modeling, a dedicated data type is defined as a specialized class to store the complex envelope for the individual carrier frequencies. This abstraction method is applied with success in a case study presented at the DATE Conference 2010 . The simulation results show that the transient simulation up to a bandwidth of 2.4 GHz for the transmission of 1,000 bits takes 63 seconds. When replacing the signal type double with the specialized data type for baseband modeling, the simulation bandwidth is reduced according to the bandwidth of the complex envelope, resulting in a simulation time of 36ms. This brings a speed improvement of a factor of 1,750.

## 1.2 Automotive System : An overview

Due to everlasting development in the field of Micro Electronics, sensors and logic systems, a new car has dozens of computers that control everything from the airbags and brakes to the lights and entertainment system. However, when someone refers to "the car's computer," they are probably referring to the engine control module (ECM). The ECM generally employs the most powerful (and expensive) micro-controller in the vehicle. Engine control modules determine where to set the throttle, how much fuel to inject into the cylinders, and when to fire the spark plugs. In many vehicles this controller also regulates the electric power distribution, provides the on-board diagnostics, and communicates with a number of other automotive systems to share information it obtains from various sensors. Engine control modules take data from a wide variety of analog sensors, digitize this information and use it to calculate the proper engine settings. The results of these calculations are converted to actuator settings and both digital and analog outputs from the module are used to operate these actuators. The diagram below illustrates some of the primary sensors and actuators employed by the engine control module.

Although cars did not have engine control modules for the first 80 - 90 years after the gasoline engine was invented, today's cars would not be able to meet modern fuel efficiency and emissions requirements without them. Improvements in engine control algorithms, data collection and data communication continue to be a major reason that cars are more efficient and less polluting with each new model year. Some vehicles allow the driver to make trade-offs between power and fuel economy by simply activating a switch that causes the ECM to run different engine control subroutines. There are also various programmable ECMs that are available to give car enthusiasts a great deal of control over how their engine will perform in various situations. *Today's ECMs generally employs 32-bit multi-core microcomputers with a few megabytes of memory clocked at speeds between 32 MHz and 300 MHz. They generally communicate with other electronic modules using one or more CAN bus interfaces. In cases where the engine control function and the transmission control function are combined in the same module, the module is generally referred to as a power-train control module (PCM).*

### 1.2.1 A general schematic of ECM

The nerve center of an electronic system is the control unit. Figure 1.2 shows the system blocks of a Motronic engine management system. All the open-loop and closed-loop algorithms of the electronic system run inside the control unit. The heart of the control unit is a microcomputer with the program memory(flash EPROM) in which is stored the program code for all functions that the control unit is designed to execute. The input variables for the sequence control are derived from the signals from sensors and set-point generators. They influence the calculations in the algorithms, and thus the triggering signals for the actuators. These convert into mechanical variables the electrical signals that are output by the microcomputer and amplified in the output stage modules. This could be mechanical energy generated by a servomotor (power-window unit), for example, or thermal energy generated by a sheathed-element glow plug.

Many systems have a mutual influence on each other. For example, it may sometimes be necessary to not only have the electronic stability program carry out a braking intervention in the event wheel spin but also to request that the engine management system reduce torque and thus counteract wheel spin. Similarly, the control unit for the automatic transmission outputs a request to the engine management system to reduce torque during a gear shift and thereby promote a soft gear change. To this end, the systems are networked with each other, i.e. they are able to communicate with each other on data buses (e.g. CAN, LIN).

### 1.2.2 Motronic engine management system

"Motronic" is the name of an engine management system that facilitates open- and closed-loop control of gasoline engines within a single control unit. There are Motronic variants for engines with intake-manifold injection (ME Motronic) and for gasoline direct injection(DI Motronic). Another variant is the Bifuel Motronic, which also controls the engine for operation with natural gas.

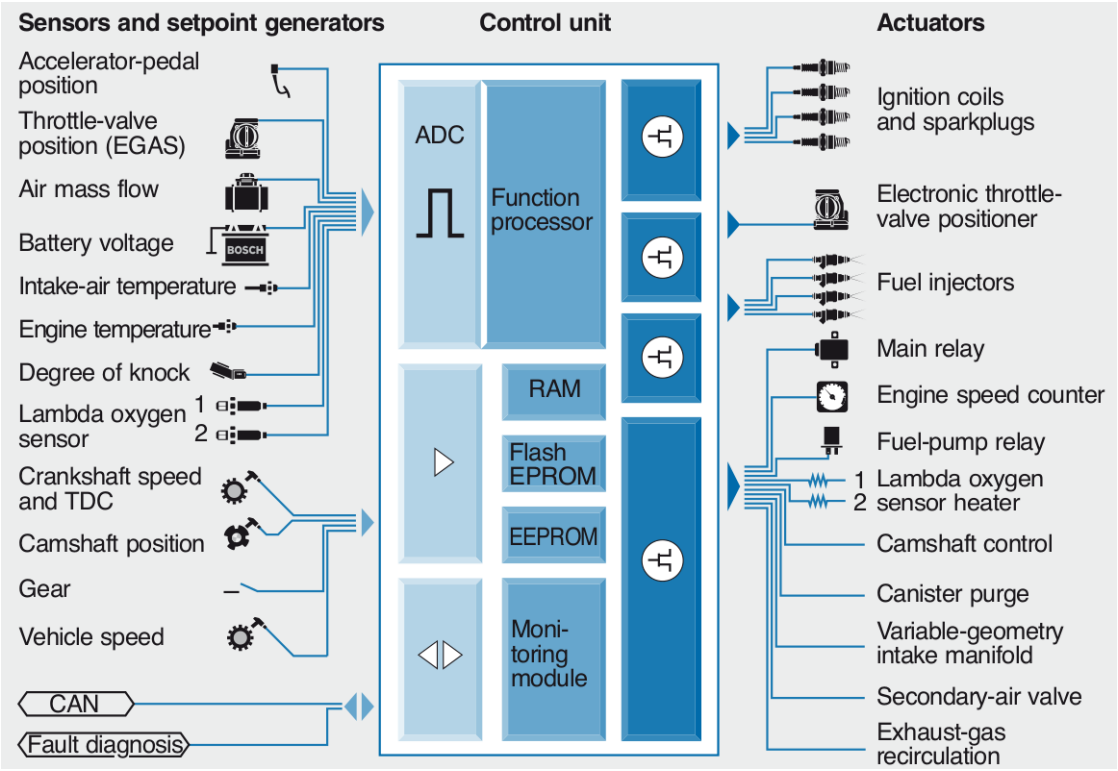


Figure 1.2: Function modules of an electronic system [13].

### 1.2.3 System description

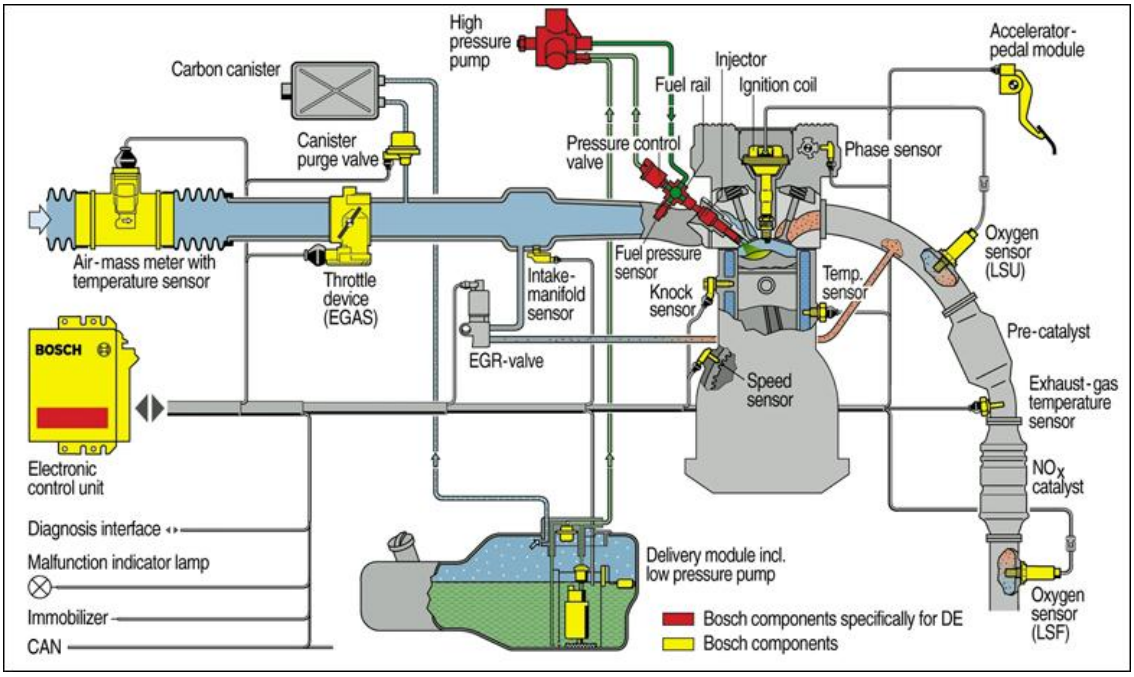


Figure 1.3: System layout of the Bosch direct injection system [13].

**Functions** The primary task of the Motronic engine management system is:

- To adjust the torque desired and input by the driver depressing the accelerator pedal.
- To operate the engine in such a way as to comply with the requirements of even more stringent emission-control legislation.
- To ensure the lowest possible fuel consumption but at the same time.
- To guarantee high levels of driving comfort and driving pleasure.

**Components** Motronic comprises all the components which control and regulate the gasoline engine (Figure 1.3). The torque requested by the driver is adjusted by means of actuators or converters. The main individual components are:

- The electrically actuated throttle valve (air system): this regulates the air-mass flow to the cylinders and thus the cylinder charge
- The fuel injectors (fuel system): these meter the correct amount of fuel for the cylinder charge
- The ignition coils and spark plugs (ignition system): these provide for correctly timed ignition of the air-fuel mixture present in the cylinder

Depending on the vehicle, different measures may be required to fulfill the requirements demanded by the engine-management system (e.g. in respect of emission characteristics, power output and fuel consumption). Examples of system components able to be controlled by Motronic are:

- Variable camshaft control:it is possible to use the variability of valve timing and valve lifts to influence the ratio of fresh gas to residual exhaust gas and the mixture formation
- External exhaust-gas recirculation: adjustment of the residual gas content by means of a precise and deliberate return of exhaust gas from the exhaust train (adjustment by the exhaust-gas recirculation valve)
- Exhaust-gas turbo-charging:regulated supercharging of the combustion air(i.e. increase in the fresh air mass in the combustion chamber) to increase torque
- Evaporative emission control system: for the return of fuel vapours that escape from the fuel tank and are collected in an activated charcoal canister

**Operating variable acquisition** Motronic uses sensors to record the operating variables required for the open and closed-loop control of the engine (e.g. engine speed, engine temperature, battery voltage, intake air mass, intake-manifold pressure, Lambda value of the exhaust gas). Set-point generators (e.g. switches) record the adjustments made by the driver (e.g. position of the ignition key, cruise control).

**Operating variable processing** From the input signals, the engine ECU detects the current operating status of the engine and uses this information in conjunction with requests from auxiliary systems and from the driver (accelerator pedal sensor and operating switches) to calculate the command signals for the actuators.

### 1.2.4 Air system

A specific air-fuel mixture is required to achieve the desired torque. For this purpose, the throttle valve (Figure 1.4) regulates the air necessary for the mixture formation by adjusting the metering orifice in the intake port for the fresh air taken in by the cylinders.

Figure 1.4: Throttle device with potentiometric position feedback [20].

This is effected by a DC motor integrated in the throttle device that is controlled by the Motronic control unit. The position of the throttle valve is fed back to the control unit by a position sensor to make position control possible. This sensor may be in the form of a potentiometer, for example. Since the throttle device is a component relevant to safety, the sensor is designed with redundancy. The intake air mass (air charge) is recorded by sensors (e.g. hot-film air-mass meter, intake-manifold pressure sensor).

### 1.2.5 Fuel system

The control unit calculates the fuel volume required from the intake air mass and the current operating status of the engine (e.g. intake-manifold pressure, engine speed), and also the time at which fuel injection should take place. In gasoline injection systems with intake manifold injection, the fuel is introduced into the intake duct upstream of the intake valves. To this end, the electric fuel-supply pump delivers fuel (primary pressure up to approximately 450 kPa) to the fuel injectors. Each cylinder is assigned a fuel injector that injects the fuel at intermittent intervals. The air-fuel mixture in the intake passage flows into the cylinder during the induction stroke. Corrections are made to the injected fuel quantity, e.g. by the Lambda control (Lambda oxygen sensor) and the canister purge (evaporative- emissions control system). With gasoline direct injection, fresh air flows into the cylinder. The fuel is injected directly into the combustion chamber by high-pressure fuel injectors where it forms an air-fuel mixture with the intake air. This requires a higher fuel pressure, which is generated by additional high-pressure pump . The pressure can be variably adjusted (up to 20 MPa) in line with the operating point by an integrated fuel-supply control valve.

# Chapter 2

# Case Study: Modeling a Throttle Valve Control Unit

## 2.1 Electronic Throttle Control

### 2.1.1 System Description of ETC

The electronic throttle control (ETC) system is a drive-by-wire system in which the direct linkages between the accelerator and the throttle or the steering wheel and the steering gear are replaced with pairs of sensors and actuators. ETC systems have existed for more than a decade but have only entered mass production in the past few years. The systems are entering the passenger car market in high-end vehicles, in which manufacturers want to enhance the driver's experience by dynamically adjusting pedal to throttle position transfer function in response to changes in the temperature, altitude, or vehicle speed.[10]

Main parts of the ETC are described in Figure 2.1:-

1. Throttle Valve
2. DC motor
3. Gear Assembly with return spring
4. Wiper for position tracking
5. Potentiometer track

Electronic throttle control systems are usually packaged as one assembly with a motor, springs, and throttle position sensors all fit into the throttle body. Figure 2.2 shows a simplified diagram of the internals of an electronic throttle body. In the center are the throttle bore and the plate. As a safety mechanism, the spring provides a torque to close the throttle when the motor is off. The equilibrium position of the spring in some electronic throttle bodies is set to a small positive angle and the throttle is used to control idle speed. The motor on the left end of the throttle shaft actuates the throttle and the potentiometer on the right end of the shaft is the throttle position sensor.

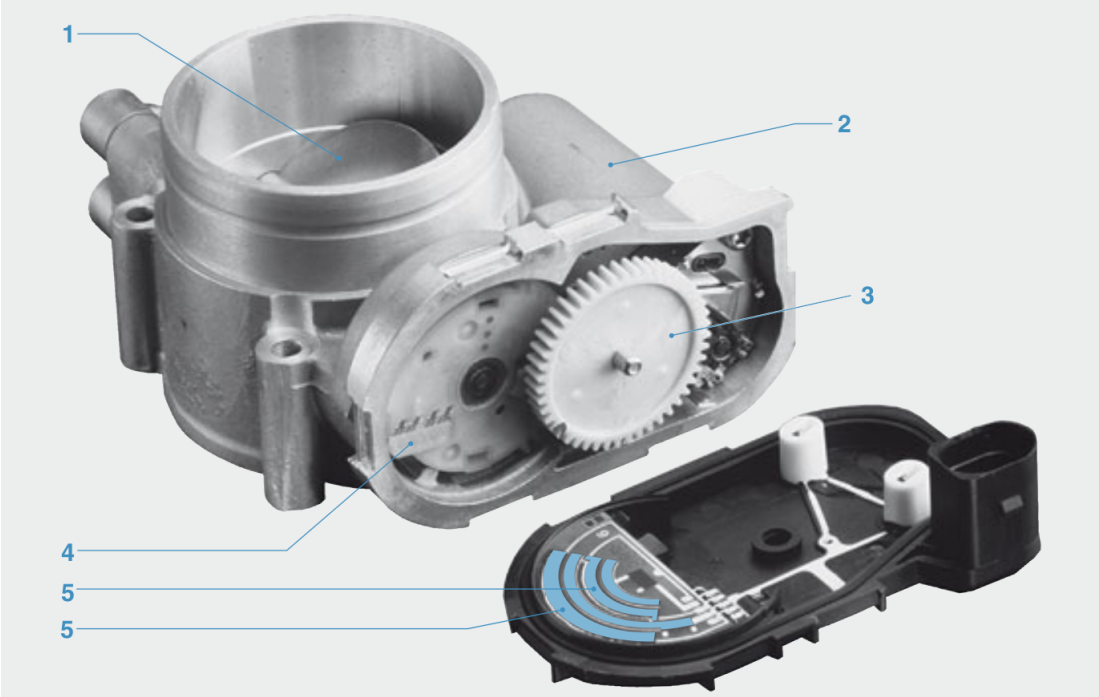


Figure 2.1: Electronic throttle control valve [13].

### 2.1.2 Drive by Wire

The electronic throttle control (ETC) system presents two interesting problems in one application. From the automatic control perspective, the throttle plate has non-linear dynamics, so control of the throttle plate position is most appropriately addressed by the use of non-linear control theory. Of equal concern is the development of software, which implements the stabilizing control law. The controller software is an embedded application on a power-train microprocessor and it must meet high reliability and safety requirements. In this project, a complete ETC system will be designed that considers the hardware, software, and control theory design problems. The alternative to the drive-by-wire electronic throttle system is the standard pull cable throttle with return spring. This is still the predominant solution in use in passenger cars. There are a number of ways in which the electronic throttle system performs better than the mechanical linkage. The only disadvantage of the drive-by-wire solution may be the cost.
A natural concern about removing the mechanical linkage between the accelerator pedal and the throttle mechanism is that the non-mechanical system might be inherently less safe and less reliable. The drive-by-wire system can, in fact, be more reliable particularly when considering problems with sticky and dirty throttle bodies. The electronic system can adapt to the friction in the system in order to maintain the accelerator pedal tracking performance. Since the system makes an estimate of the friction, it can also diagnose dirty, sticky, or otherwise worn out hardware. Hardware and software redundancy is also used to maintain a very high level of reliability. Integration of various engine and vehicle control systems can be accomplished with a single ETC system and can offset the additional cost of the ETC hardware. Cruise control, idle control, engine over-rev. protection and traction control features might all need to modify the throttle position. With the ETC system, the switching or blending of control algorithms occurs in software

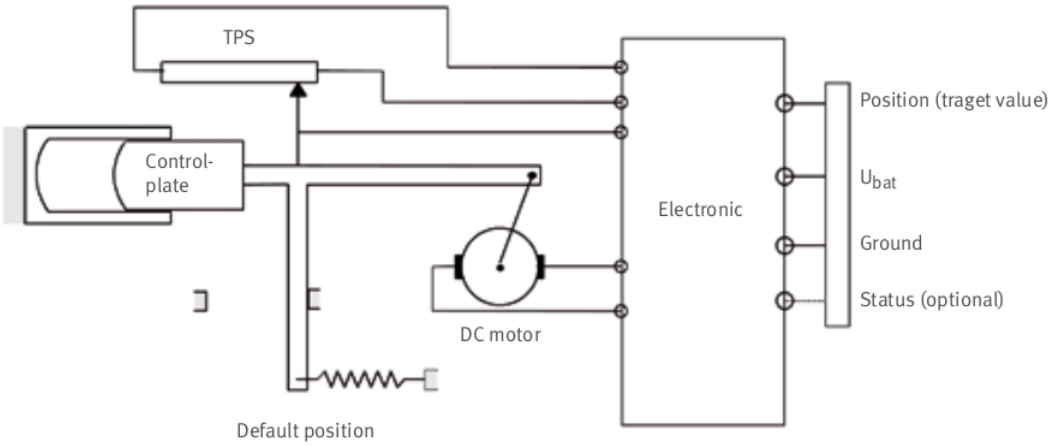


Figure 2.2: simplified layout of ETC

and there is no need for separate actuation components for each feature. There are also advanced features that can be accomplish only with the ETC system. If multiple throttles are used, sophisticated engine power management can shut down individual cylinders and, in doing so, increase the efficiency of the power cycles in the other engine cylinders. The replacement of the connection between the driver's foot and the throttle plate with software allows the designer to adjust the pedal-to-plate transfer function. For instance, initial pedal travel can correspond to smaller throttle plate motion compared with pedal travel closer to the wide open-throttle (WOT) position. This transfer function can also be adjusted for vehicle speed or altitude to make the engine feel more responsive to the driver.

## 2.2 Model of Throttle Valve Mechanics

Many of the actuators in automotive control systems is a DC motor. It can provide rotational motion along the axis of Rotor. The electric equivalent circuit of the armature,the free-body diagram of the throttle valve and a return spring are shown in the Figure 2.3.

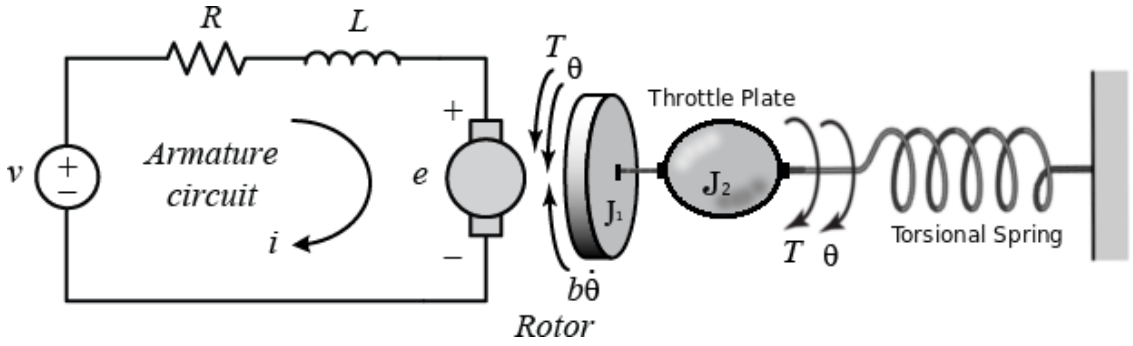


Figure 2.3: Free body diagram of DC motor

In general, the torque generated by a DC motor is proportional to the armature current and the strength of the magnetic field. In this work, a constant magnetic field is assumed

for automotive actuators and, therefore, that the motor torque is proportional to only the armature current i by a constant factor $K_t$ as shown in the equation below. This is referred to as an armature-controlled motor [18].

Lets define the variables of the model.

- **J** moment of inertia of the throttle plate and DC motor
- **b** motor viscous friction constant
- $K_e$ electromotive force constant
- $K_t$ motor torque constant
- **R** electric resistance
- **L** electric inductance
- $\theta$ angle of rotation of throttle valve
- **V** applied voltage to motor
- **i** current in armature circuit

$$T = K_t \times i \tag{2.1}$$

The back emf, e, is proportional to the angular velocity of the shaft by a constant factor $K_b$.

$$e = K_b \times \dot{\theta} \tag{2.2}$$

In SI units, the motor torque and back emf constants are equal, that is, $K_t = K_b$; therefore, we will use K to represent both the motor torque constant and the back emf constant.

The Figure 2.3 shows two inertia $J_1$ of rotor and $J_2$ of the throttle blade. It can be combined as a single inertial component $J = J_1 + J_2$ as both are connected to same mechanical source (DC motor). Thus based on Newton's 2nd law and Kirchhoff's voltage law the throttle mechanics can be represented as in Equation 2.3 and 2.4.

$$J\ddot{\theta} + b\dot{\theta} = K \times i \tag{2.3}$$

$$L\frac{di}{dt} + Ri = V - K\dot{\theta} \tag{2.4}$$

Applying the Laplace transform, the above modeling equations can be expressed in terms of the Laplace variable 's'.

$$s(Js + b)\theta(s) = KI(s) \tag{2.5}$$

$$(Ls + R)I(s) = V(s) - Ks\theta(s) \tag{2.6}$$

Apart from preceded equations, a throttle valve has a return spring that opposes the torque generated by DC motor as in Figure 2.3. The return spring is a torsional spring type, that twists in inward direction when DC motor is powered ON. And as this is a return spring, it untwists when there is no power applied to DC motor. The Equation 2.6 relates the twisting torque $T$ with its spring constant $K_t$.

$$K_t = \frac{\pi G D^4}{32L} \tag{2.7}$$

$$T = K_t \times \theta \tag{2.8}$$

where $\theta$ is the net angular twist in the element, G, is the shear modulus of elasticity, D of coil, $T$ is the torque causing the twist, and $K_t$ is the torsional spring constant. The initial position is $\theta = 0$ at the spring position where there is no torque in the spring.

The torque generated and opposed should be equivalent, and within the safe operating area of the DC motor. This is an important requirement for throttle valve motor design. when the throttle device is not applied any supply, the load current is zero.When we switch on the value varies from 2A to 12A. Its designed with this condition that when a fully opened position (100%) is reached its 10% below the maximum current. Thus a boundary condition is as in Equation 2.9.

$$\begin{aligned} (2A \leqslant i \leqslant 10A) \\ (0^\circ \leqslant \theta \leqslant 90^\circ) \end{aligned} \tag{2.9}$$

In this system,the opening position is mapped to the load current of DC motor. The term related to back emf is neglected as throttle plate rotates between 0° and 90°. By solving equation 2.5 and 2.6 for opening position, a transfer function in Equation 2.10 is used to describe the system with no back emf as the throttle valve never rotates 360°. It gives in terms of angular velocity to the applied voltage. But the system under design opens and closes with a specified angle. By integrating the Equation 2.10, an open loop function for opening angle for applied voltage is arrived in Equation 2.11.

$$\frac{\dot{\theta}(S)}{V(S)} = \frac{K}{(Ls+R)(Js+b)} \tag{2.10}$$

Hence,

$$\frac{\theta(S)}{V(S)} = \frac{K}{s(Ls+R)(Js+b)} \; [\frac{rad}{V}] \tag{2.11}$$

### 2.2.1 Abstract modeling based on Experimental Data

By substituting the experimental values in transfer funtion 2.11. The Dynamic behaviour of the system can be implemented with the help of AMS MoCs. In order to accelerate the system development, the dynamic behaviour of ETC is abstracted.The Potentiometer position feedback is used to identify the position of the throttle plate. But here for abstract modeling and a new novel technique of load current sensing is done. In reality, current sensor in armature circuit or hall effect sensor can be used. This methodology is more

realistic in comparison with modeling a position feed back. A typical throttle valve opens rotationally between from 0° to 90°, which is termed as 0% to 100% opening of valve. Based on opening of the valve the load current varies from 2A to 10A and this parameter is used for modeling the system behaviour of the throttle valve.

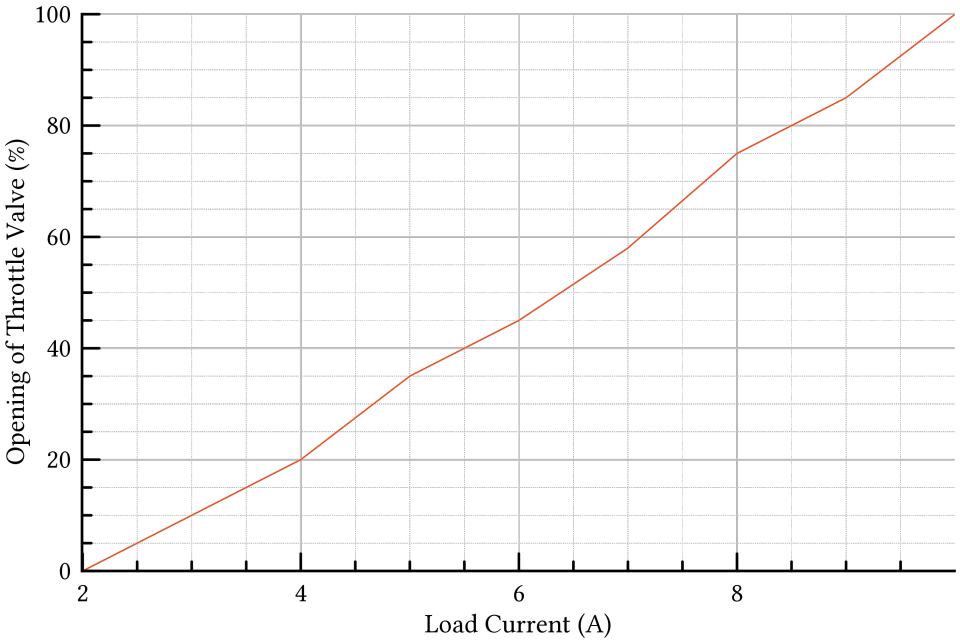


Figure 2.4: Throttle valve load current vs opening position

## 2.3 Functional Components of ETC

### 2.3.1 PI Control Algorithm

The automotive ECUs have many control algorithms running in parallel. sometimes running in a distributed network with control values passed over network (CAN, FlexRay, Lin.., etc.). The most predominantly used algorithm is PI control algorithm. A PI controller has two parts, a Proportional and an integral term, a schematic is shown in Figure.

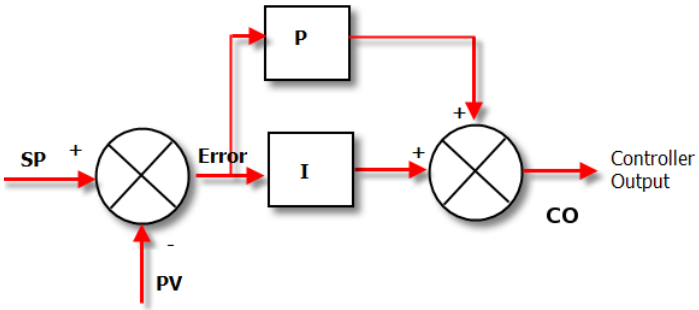


Figure 2.5: PI controller

$$CO = CO_{bias} + K_p \times e(t) + \frac{K_i}{T_i} \times \int e(t)d(t) \tag{2.12}$$

Where,
$CO$ - controller output value
$CO_{bias}$ - controller bias or null value
$e(t)$ - current controller error, defined as (SP-PV)
$SP$ - set point
$PV$ - measured process variable
$K_p$ - Proportional gain
$K_i$ - Integral gain
$T_i$ - reset time, a tuning parameter

The proportional term of the PI algorithm, $K_p \times e(t)$, adds or subtracts from $CO_{bias}$ based on the value of control error $e(t)$ at each time $t$. As when control error $e(t)$ increases or decreases, the control output $CO$ also varies immediately and proportionately. The past history and current trajectory of the controller error have no influence on the proportional term computation.

The integral component of the PI algorithm is the last term of the equation 2.12. While the proportional term considers the current size of e(t) only at the time of the controller calculation, the integral term considers the history of the error, or how long and how far the measured process variable has been from the set point over time. Integration is a continual summing. Integration of error over time means that summing up the complete controller error history up to the present time, starting from when the controller was first switched to latest point of execution.

The reset time tuning parameter, $T_i$:

- It provides a separate weight to the integral term so the influence of integral action can be independently adjusted.
- It is in the denominator so smaller values provide a larger weight to (i.e. increase the influence of) the integral term.
- It has units of time so it is always positive.

In this work the PI control algorithm is implemented with different MoCs for System and architecture exploration with different MoCs. In DE domain, its implemented as CPU process that executes at a period of 1us to 10us. The AMS extensions provides a lot of library for implementing the DSP functions and control algorithms.

***Challenges of PI Control***
There are challenges in employing the PI algorithm:

- The tuning parameters namely $K_i$,$K_p$, and $T_i$ interact with each other and their influence must be balanced by the system designer.
- The integral term tends to increase the oscillatory or rolling behaviour of the process response.

For initial system design the PI control was implemented with LSF MoC. With the help of AMS technical library its very elegantly implemented within short time. While moving towards discrete domain it has to be executed periodically. But the mixed MoC virtual prototype helps the designer to estimate the time of schedule for PI algorithm.

### 2.3.2 Pulse Width Modulator for Actuator Control

The pulse width modulator (PWM) is also an integral part of the feedback control loop and needs to be properly modeled for control system design. A constant frequency modulator can be modeled by a constant gain, but such linear time invariant model is only valid up to half the carrier frequency. Beyond this frequency, the modulator response to a perturbation in the reference may be affected by side-band components of the perturbation (when the perturbation frequency is a multiple of one half of the carrier frequency) and cannot be modeled by a constant gain. A variable-frequency modulator can also be modeled by a DC gain with a leading phase. The DC gain changes with the ON and OFF time, but the variation is small and can be ignored in practice. The leading phase adds to the phase margin of the control loop and is beneficial for control stability.

Multiple PWM converters can be connected in parallel or in series to form a modular design with scalable current or voltage capacities. In such modular systems, interleaving offers an opportunity to reduce the combined input or output harmonics through harmonic cancellation among different modules. The traditional symmetric interleaving method eliminates all but N-multiples of the carrier harmonic for N modules connected in parallel or in series. Additionally, the harmonic cancellation effect is such that, when a carrier harmonic is eliminated, all of its side-band harmonics are eliminated as well. Asymmetric interleaving makes it possible to selectively reduce different carrier harmonics and their side-band components to achieve different objectives, such as minimisation of EMI filter size.

*Peak current control is a special form of control used in many automotive actuators.* The peak current varies based on the PWM duty cycle (Figure 2.6).

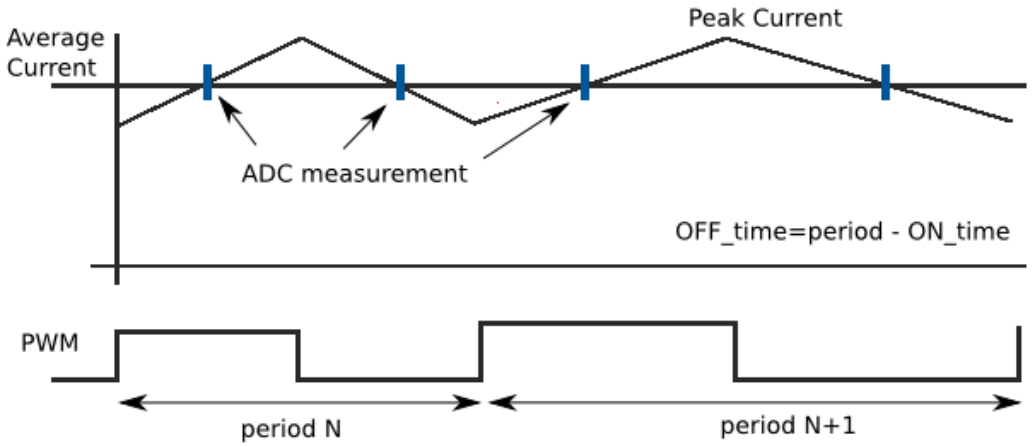


Figure 2.6: PWM ON time vs peak current

Further towards implementation, Given,

$$V = L\frac{di}{dt}$$
$$I(t) = I(t_0) + \frac{1}{L} \times \int V(t)dt \tag{2.13}$$

Rearrange,

$$\frac{L}{V} \times (I(t) - I(t_0)) = dt \tag{2.14}$$

Redefining **dt** with PWM on time,

$$PWM_{ON} = \frac{L}{V} \times (I_{desired} - I_{actual}) \tag{2.15}$$

For most applications, the input voltage can't change very quickly-because of large input-filter capacitors. Therefore, its not needed calculate the time-consuming divide operation (L/V) for every execution of the control algorithm. To reduce the computational workload, you can share this calculation among a number of PWM cycles. By treating the (L/V) term as a quasi-constant, the remainder of the duty-cycle calculation becomes trivial. By not using an analog comparator for PWM termination, the analog comparator becomes available for detecting unexpected severe load-current transients or output over voltage conditions.

It's important to select the optimum method for digital current control for automotive applications. For a systematic analysis,the of usage of different MoCs [2] are modeled with DTDF and TDF. The PWM generation is implemented in Dynamic TDF MoC and also experimented together with a Discrete Event SystemC module. When [2] using the conventional TDF model of computation defined in the SystemC AMS 1.0 standard, the AMS computations are executed at fixed discrete time steps while considering the input samples as continuous-time signals. Since the PWM block has an almost discrete-event behaviour, the need to have very steep ramps for the signal transitions at its output imposes the use of very small time steps. A fine-grained time grid is essential for a correct overall response of the system, as it needs to meet the accuracy constraint (time constants) of the PI controller and the Driver + DC motor. However, a too-fine-grained time grid will reduce the simulation performance, as the number of time steps and thus AMS computations will increase.

Alternatively, the PWM could be modeled as a pure SystemC discrete-event model. But this makes the simulation less efficient due to the use of the dynamic scheduling of SystemC (evaluate/update cycle) instead of the more efficient TDF static scheduling. Furthermore, it will introduce unnecessary synchronization between TDF and the SystemC discrete-event model of computation.

By introducing Dynamic TDF for this application, the computation of the motor control loop is only triggered four times per pulse period by changing the TDF time step (Figure 2.7): first at the start of the rising edge, second at the end of the rising edge, third at the end of the voltage pulse plateau, and fourth at the end of the falling edge. Each time, the PWM adjusts the scheduling of the next activation based on the duty cycle sampled at its input during the rising edge. With the pulse output active, energy is supplied to the power driver of the DC motor resulting in an increase of the current. This process repeats

itself while the system reaches its steady state. Note that the PWM output signal (vdrv) represents a continuous-time waveform and thus has finite-slope edges.

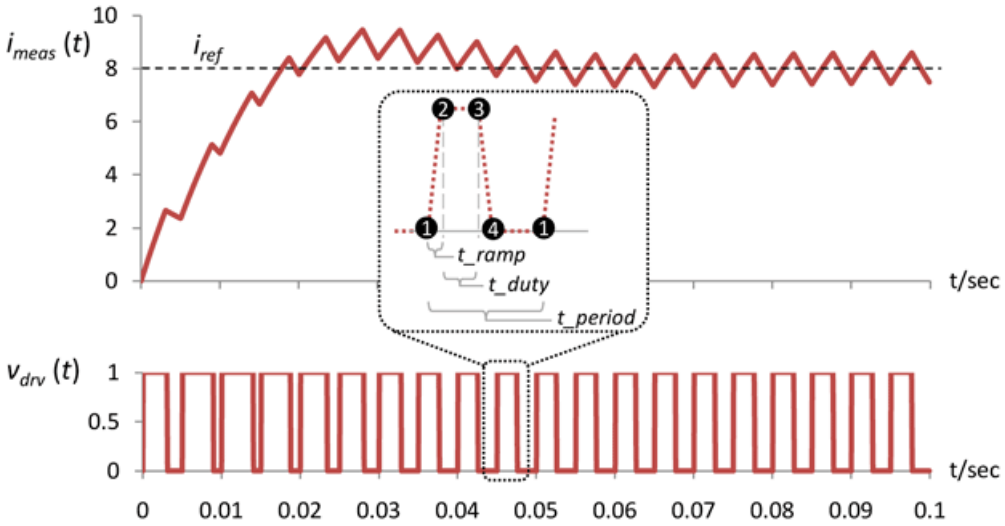


Figure 2.7: Step response of the motor control loop using Dynamic TDF with four activations per period [2]

To efficiently model the PWM pulse with a varying pulse width, the time step attribute will be changed such that the PWM pulse's rising edge and falling edge are included resulting in only four activations of the PWM TDF module per period, as shown in Figure 2.7.

### 2.3.3 Analog to Digital Converter(ADC)

In a closed loop digital control system, analog to digital converters are very important component in converting the physical value of the system into samples of discrete values. A digital processor consumes these values and produces new control values with control algorithms executing as a task in RTOS. To model such an embedded system, TUV-AMS library is used and developed in our Cyber Physical Systems group. The library has set of components for analog and digital components for automotive system design.

A general n-bit ADC converts continuous signals to N bit discrete digital numbers. The ADC is implemented as a DTDF model which is able to generate conversions dynamically to accelerate simulation performance 4.1 4.2. The reference voltage can be set with parameter uref. The optional parameters gain_e and offset_e allows to simulate the static errors like gain error and offset error. Their unit is LSB. The parameter gain_e defines the maximal gain error and respectively offset_e defines the maximal offset error. By the instantiation of the model a value between -gain_e LSB and +gain LSB (or -offset_e LSB and +offset_e LSB) will be selected automatically according to the uniform distribution. The number of the output ports (resolution of the converter) can be set with the template N. Please note that the highest bit is used to express the algebra sign of the output value. So, one extra bit must be reserved for it. For instance, N should be set to 4 when a resolution of 3 bit is expected.

Class definition:

template <int N> adc(sc_core::sc_module_name n,double uref, double gain_e,double offset_e);

Interfaces:

sca_tdf::sca_in<double> in; sca_tdf::sca_out<sc_bv<N> > out;

| Parameter | Type | Default Value | Description |
|---|---|---|---|
| n | sc_module_name | " " | instance name |
| uref | double | 1.0 | reference voltage |
| gain_e | double | 0.0 | Maximum gain error |
| offset_e | double | 0.0 | Minimum offset error |

Table 2.1: ADC parameters definition

Example:
adc<4> i_adc ("adc",2.0,0.5,0.7);
i_adc.out(signal_out);
i_adc.in(signal_in);

## 2.4 Executable System Specification in DE and AMS MoCs

In this work different MoCs are and based on introduced model [2.3], following is the application of MoCs for Mixed domain modeling. As discussed about automotive system in previous section 1.2, consists of basically a processing unit, generates output based on system behaviours.To accelerate the development of System and HW/SW co-design a throttle control system is modeled as a subsystem. An Electronic throttle control was introduced for precise control of AIR/FUEL mixture ratio. This mixture is calculated based on the state of the system and operating environment. To arrive at architecture definition and software partition, this methodology helps initially with a executable system model. With this we can explore the system and identify the required components systematically.

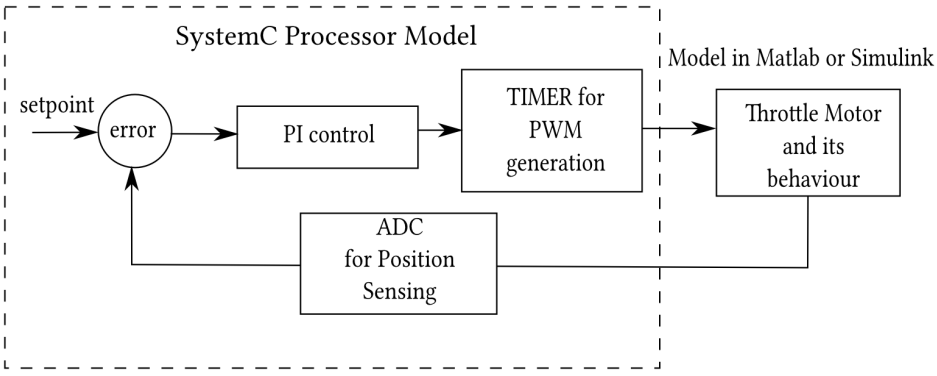


Figure 2.8: A conventional approach of mixed domain modeling

In a conventional SystemC environment, the only digital parts modeled (eg.,processor,IP models) and mixed signal parts as an abstract discrete model. When there is need to

interact with dynamic behaviour of the system, a plant model is developed in MATLAB, Simulink, Silver or Saber. With these software a co-simulation environment is created with SystemC DE models. A conventional design flow of such a system is shown in Figure 2.8. The model is very abstract in discrete domain, and also a mathematical equivalence is very difficult to describe. The interaction between different software is not open and execution semantics are not known to the user and thus system optimization is limited and unrealistic. As these modeling software are proprietary and governed by patents in commercial market, the quality of modeling is a critical quality issue. Hence any system exploration in a commercial software can not used for HW/SW partitioning unless a huge license is purchased. The DE MoC of SystemC is able to model describe the discrete domain components. And for mixed signal components the use of dedicated MoCs are well established by SystemC AMS extensions .

To understand use of dedicated MoCs in a ETC example, an executable specification for System specification is implemented (Figure 2.9). With this a partition is done with different components needed for ETC. To formalise the specification, different modules of the system is assigned a MoC. The assignment of MoC is trivial. After many iterations the best MoCs for the ETC application is formalised. A PI control is needed to proactive control the output variable based on set-point and hence a LSF is used.

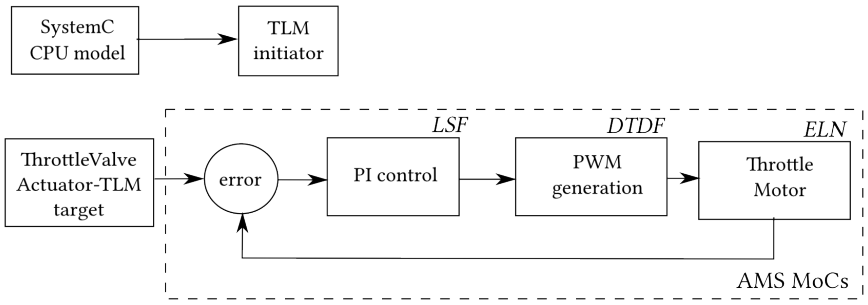


Figure 2.9: Executable specification of ETC

The throttle valve model discussed in section 2.2 is implemented as ELN. But same can be implemented as TDF with sca_tdf::sca_ltf_zp class to represent Laplace transform of any automotive system.

Based on the analysis with system implementation, HW/SW partitioning is established. The PI control algorithm needs much tuning during development and also entire life cycle of product. PI control can be tuned as a software process. The PWM part is a hardware module that creates output waveform based on the control value from PI algorithm. The PWM can also be implemented as a SW process and can toggle the microprocessor pin as PWM output. With this a system specification is concluded with different MoCs and can be analysed further with architecture exploration in next chapter. In system design process this also acts as a formal representation in accordance with customer requirements.

In the field of system engineering, the need for a formal representation of requirements is highly appreciated. For high quality and low failure rates in safety critical, this has to be integrated into the product development life-cycle 3.3. Often this benefits in automotive industry with high return on investments and reduced time to market with less re-engineering cost.

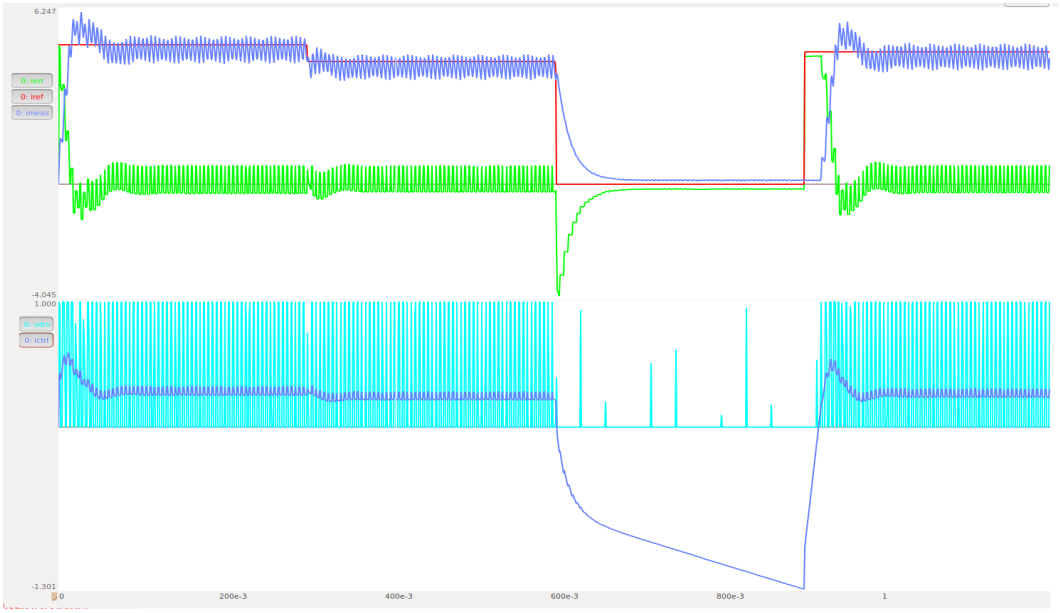


Figure 2.10: Response of the ETC system model

# Chapter 3

# Experiments,Results and Discussion

## 3.1 Throttle Valve Controller as a Mixed MoC Virtual Prototype

In the initial phase of the system design, for system exploration an executable prototype is build. The Figure 2.9 shows an overview of the system. The AMS extensions with dedicated MoCs for heterogeneous systems helps the system identification task simpler. A closed loop control was implemented completely with SystemC AMS. After the analysis with the output in executable specification, it triggered new thoughts for architecture definition. To understand the behaviour of different MoCs a more refined virtual prototype was built as in Figure 3.1. In automotive industry usually hardware and software partitioning is done based on the customer requirement. As there are different sensors and actuators connected to the ECUs. An executable prototype in this work helps the system designer to decide components to be partitioned. Based on the experimentation with different configuration of MoCs its often able to exploit best fit for the system design. The language semantics available with AMS extensions benefits cost sensitive markets to develop system models in a unified platform. To understand the benefits of a unified environment and better use of Mixed MoC virtual prototype. The electronic throttle valve control unit (ETC) is developed as a mixed domain virtual prototype. To support this methodology, the fore coming results substantiates the benefits of AMS extensions of SystemC as a unified modeling environment. In automotive ECUs, tasks are scheduled to run periodically based on ECU's global timer and also there task list for dynamic variation of engine speed. With this view the following experimental framework uses discrete event simulation for periodic tasks coupled with AMS MoCs for variable time step tasks. There are upto nine different experiments are conducted to analyse and describe the system design with AMS MoCs. The modules simulated in Discrete Event MoC can be partitioned to SW based on the feasibility study with virtual ECU prototypes. Going further with dynamic scheduling techniques TDF MoC, possibility to design system with dynamic activation of modules either in Hardware or Software. Often in life critical embedded systems even interrupts are polled in a periodic window. But this fact is based on past experience. The advent of multi core micro-computers system its now possible to re-

engineer certain fundamental assumptions. Thus these experimentations helps to refine these conventional system design techniques and evolve to a modern paradigm of complex embedded systems with data flow oriented processors.

### 3.1.1 Use cases and semantics of different MoCs

The objective of Dynamic TDF is to offer mechanisms to dynamically change key TDF properties such as time step, rate, or delay. This efficiently overcomes the limitations of the TDF MoC. These new features should also support the successive transition from abstract, functional-level modeling to refined mixed-signal architecture modeling with ideal and 'non-digital' properties. In that respect, we consider the following relevant use cases for automotive domain modified based on Annex A.2 in [2].

- *Abstract modeling of sporadically changing signals* that are constant over long time periods and that cannot be modeled by sampled, discrete-time signals in an efficient way. A particular application is Key ON/OFF cycle with T15 (terminal 15 switch of ECU) and wakeup switch in ECU. In order to model power down cycle and other after run properties of ECU SW, it is required to be able to specify a condition that enables/disables the execution of the AMS computations.

- *Abstract description of reactive behavior*, for which computations are executed for events or transactions, such as an CAN message arrival, which triggers activation of new task in ECU. Typical applications are sensor systems, in which crossing a threshold will cause an action. An event-triggered modeling approach for these systems would be more natural and also more efficient. A reactive synchronization mechanism would be beneficial, especially in cases where these AMS systems are integrated together with TLM-based virtual prototypes (DE) [3.12, 3.11]. This avoids the penalty of introducing fine-grained computations by using small time steps to detect the actual event.

- *Capture behavior where frequencies (and time steps) change dynamically.* This is the case for applications executed based on VCO, PLL, PWM, or clock recovery circuits of ECU, which are often controlled by analog or digital signals. Modeling oscillators with variable frequencies (e.g., clock recovery) or capturing jitter is not possible when using constant time steps. In order to allow modeling of such systems, it is required to be able to change the time step continuously during simulation. A typical scenario would be SW tasks for ratio-metric correction of ADC reference voltage in ECU when the system supply chain varies due to battery voltage variation. Another example is in ECU start up sequence, needs simulating of tasks with variation of PLL, VCO or clock of micro-controller.

- *Modeling systems with varying (data) rates*, which are changed during operation and thus during simulation. This is the case, for example, when communication systems are described at high levels of abstraction. To perform cross-layer optimization and to evaluate the correctness of a particular signal-processing algorithm, both the physical layer (PHY) as well as media access control (MAC) as part of the data link layer (DLL) need to be modeled. An example of such systems are ECU network with different BUS technologies (CAN, FlexRay, Lin ..), in which parameters such as data rates, bus-off sensing and synchronisation between different BUS.

### 3.1.2 Conversion between different MoCs

The different use case substantiates the benefits of a Dynamic TDF for reactive and dynamic ECU systems. There is a need to use different MoCs to model other parts of the system. For example the DSP algorithms needs a constant time step. In order to facilitate the interaction between different MoCs. There are conversion channels and ports to make the modeling task simpler and more technically feasible. The AMS standard provides this conversion with its infrastructure semantically. Given below is code example for a low pass filter (LPF) in TDF and ELN MoCs with converters. (Annex A.1 in [2])

```
SCA_TDF_MODULE(lp_filter_tdf)
{
sca_tdf::sca_in<double> in;// converter port for SystemC input
sca_tdf::sca_out<double> out;
sca_tdf::sc_in<double> gain; // converter port for SystemC input
sca_tdf::sca_ltf_nd ltf; // computes transfer function
sca_util::sca_vector<double> num, den; // coefficients
   void initialize()
   {
      num(0) = 1.0;
      den(0) = 1.0;
      den(1) = 1.0 /( 2.0 * M_PI * 1.0e4 );
   }
   void processing()
   {
      out.write( ltf( num, den, in, gain.read() ) );
   }
SCA_CTOR(lp_filter_tdf) {}
};
```

Listing 3.1: LPF model with converters from DE to TDF

```
SC_MODULE(lp_filter_eln)
{
sca_tdf::sca_in<double> in; // Input from TDF MoC
sca_tdf::sca_out<double> out; // Output to TDF MoC

sca_eln::sca_node in_node, out_node; // node declarations
sca_eln::sca_node_ref gnd; // reference node
sca_eln::sca_r *r1; // resistor
sca_eln::sca_c *c1; //capacitor
sca_eln::sca_tdf_vsource *v_in; //converter TDF -> voltage
sca_eln::sca_tdf_sink *v_out;  //converter voltage -> TDF
   SC_CTOR(lp_filter_eln)
   {
      v_in = new sca_eln::sca_tdf_vsource("v_in", 1.0); // scale factor 1.0
         v_in->inp(in); // TDF input
         v_in->p(in_node); // is converted to voltage
         v_in->n(gnd);
      r1 = new sca_eln::sca_r("r1", 10e3); // 10 kOhm resistor
         r1->p(in_node);
         r1->n(out_node);
      c1 = new sca_eln::sca_c("c1", 100e-6); // 100 uF capacitor
         c1->p(out_node);
         c1->n(gnd);
      v_out = new sca_eln::sca_tdf_sink("v_out", 1.0); // scale factor 1.0
         v_out->p(out_node); // filter output as voltage
```

```
        v_out->n(gnd);
        v_out->outp(out);  // here converted to a TDF signal
    }
};
```

Listing 3.2: Similar implementation of LPF in ELN MoC

Based on the example of LPF its upto the system designer's interest to choose the right MoCs. In this work the PWM and ADC modules are modeled in DTDF and TDF MoCs. To study the usage of TLM, the DE signal is copied to transaction payload and used for PI control algorithm. In the code the initiator and targets are connected between the ADC, and CPU as given below.

```
/*CPU for PI control alorithm*/
cpu process1("process1");
process1.clk(EOC_sig2); //  <- -dynamic activation of CPU for  data flow
     oriented processing
process1.clkout(EOC_sig2);

/*ADC to sense the throttle position */
adc<16> adc_2("adc_2", 10, 0.0, 0.0, 0); // Module_name , Reference
    Voltage, gain_e, offset_e, nl_m
adc_2.in(imeas);
adc_2.out(out_ad);

/* Instantiating and connecting the TDF to DE converter */
conv_tdf_sc<16> tdf_sc2("tdf_sc");
tdf_sc2.inTDF(out_ad);  /*END SystemC-AMS*/
//SystemC Discrete Domain Signals->interface using TLM technique
tdf_sc2.outDE(ad_out_de2);
tdf_sc2.EOC(EOC_sig2); //

adc_data_t *adc_ini = new adc_data_t("adc_data_t");
adc_data_r *adc_tgt = new adc_data_r("adc_data_r");
adc_ini->clk(clk1);
adc_ini->meas_bv(ad_out_de2);
adc_ini->ref_bv(ad_out_de1);
adc_ini->socket.bind(adc_tgt->socket);
```

Listing 3.3: code snippet for propagating dynamic behaviour to DE

## 3.2 ETC prototype with DTDF MoC

As stated in previous discussions the following virtual prototype is designed and implemented with open source SystemC and AMS extensions. The electronic throttle controller (ETC) is implemented with following key components.

- CPU - processes control algorithms and executes firmware tasks
- Reference source - to test the prototype with a step input
- Clock - to activate periodic modules (DE)
- Throttle Motor and Mechanics - a plant model based on 2.2

- Converters - to couple different MoCs
- ADC - to sense the physical value from the throttle valve plant model
- PWM - to generate PWM signal for plant model

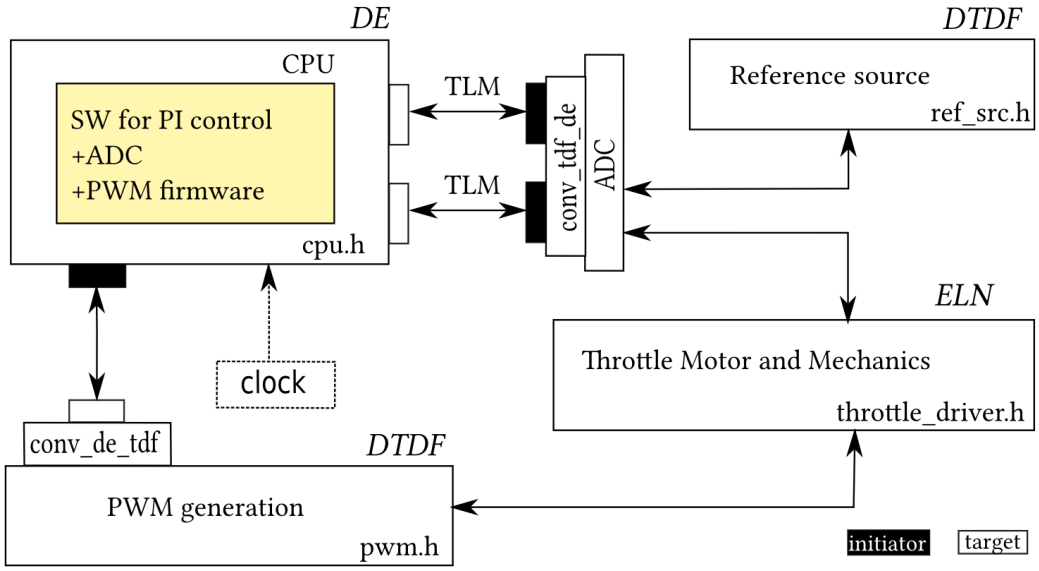


Figure 3.1: ETC executable prototype with DTDF

The Figure 3.1 illustrates the simulation framework used for this study. During the initial phase of system model establishment in previous section 2.4 the PI control is not tuned and unusable. After applying some control system techniques [22] [25] an approximate value for the PI control parameters ($K_p$,$K_i$) are assigned. Based on the system performance and modifying the PI parameters a smooth response with less than ± 2% error band achieved in Figure 3.1. When moving towards the usability analysis of the model, we cannot directly use the PI parameters of the model. It depends on factors like scheduling time, dynamic activations of model and execution time. But the values used in 2.4 helps us as a starting value without applying scientific approaches. Thus with these values and study of ETC system level specification, we are able to use the modeling experience for architectural exploration in following section. Initially Dynamic TDF clusters are explored with this framework and further section compares this results with TDF clusters.

*To favour the dynamic TDF analysis with this frame work, the dynamic activation is implemented in PWM module. In a set of AMS clusters, when a cluster is modeled to set time step dynamically this time propagates in all AMS MoCs. This scenario is extracted in the ADC module as a signal through output port. whenever an end of conversion happens in ADC, it outputs the time as a signal. The CPU is assigned to process the control algorithm either in positive or negative edge of this signal "EOC_sig2" (see listing 3.1.2). Note ADC cluster is one of the AMS clusters in the framework which accepts the time propagation.*

### 3.2.1 Experimental Results with DTDF,DE and TLM MoCs

With same values for $K_p$ and $K_i$ from previous experiment 2.7, the simulation of ETC was further continued for 1s, during 0.3s a set-point of 50% is set. In control system analysis, the system is tested by applying a step response. The different configuration of the framework are the DTDF clusters in Figure 3.1 are allowed to execute dynamically with DTDF semantics. The need for DTDF and its semantics are explained in section 2.3.2. The CPU is executed with a constant clock frequency between 1ns and 100us to study the system exhaustively. The decision to make clocked CPU to interact with a dynamic TDF is to resemble an automotive environment. In which the system varies dynamically based on environment conditions and ECUs executes periodic task in a static schedule. The usage of DTDF in other parts of the simulation framework helps to accelerate the simulation in terms of number of samples and execution time of the model. This is made possible with dynamic PWM, which triggers other parts of the framework to execute dynamically.

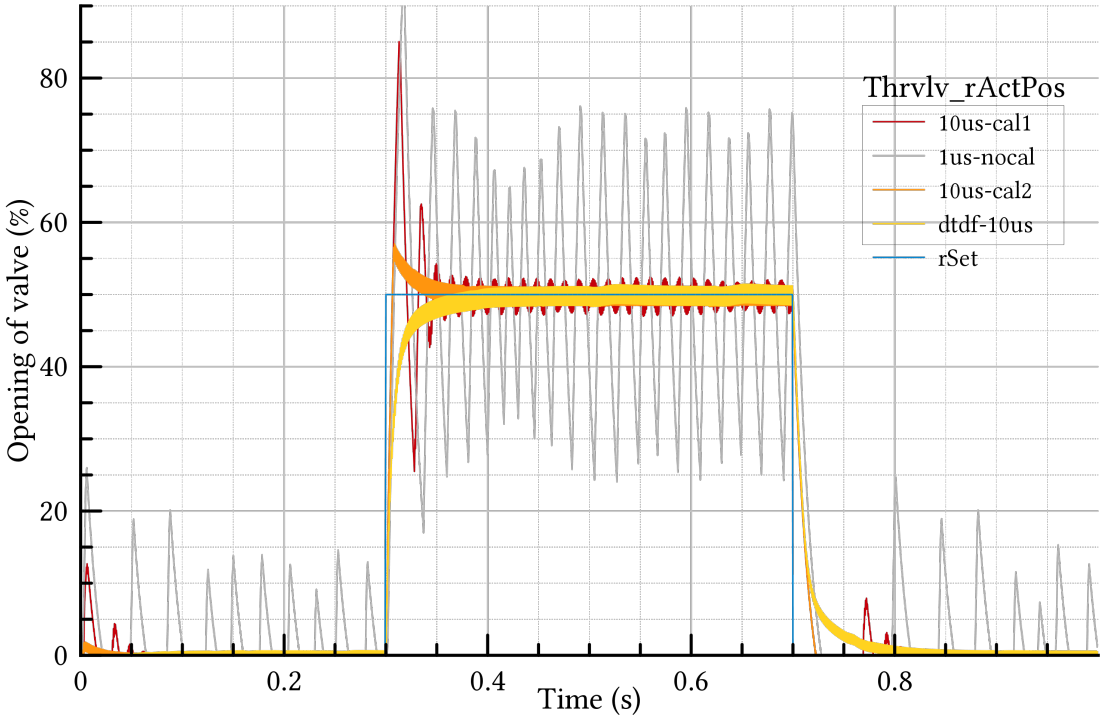


Figure 3.2: Comparison of responses with DTDF

Before explaining about technical features, in figure 3.2, the different graphs are plotted with all TDF clusters with Dynamic features. But the CPU module is executed as clocked process and is depicted by following legends 10us-cal1, 10us-cal2, 1us-no cal. While in "dtdf-10us" experiment the CPU is activated dynamically based on ADC conversion rate. And this ADC conversion rate is controlled by time propagation of PWM module with DTDF MoC. In these graphs "ThrVlv_rActPos" means throttle valve actual positions and "rSet" means throttle valve set-point.

Note that when we need to set the CPU to execute at 1us, all other DTDF clusters should have a initial time of 1us. This is a limitation set by AMS to DE kernel, without which the simulation could run but without any reasonable results. Based on AMS SW ar-

chitecture with different time settings in DTDF clusters, the simulation kernel decides the initial time as minimum of all clusters. And this time step is propagated across clusters. That ensures synchronisation with different clusters. To illustrate this behaviour in Figure 3.2, the throttle response plot "1us-no cal" has a rugged curve with oscillations. During this experiment the CPU is executed at 1us clock and other modules as dynamic TDF. The reason behind this is there is no sync between the PI control and DTDF clusters. The oscillatory behaviour can be reduced by calibrating the PI parameters. But the essential understanding between DE and DTDF cluster interaction is the PI control executes faster and always produces new control value. And this leads to perturbation in the system, before the system settles with last control value.

*The TLM MoCs are implemented to send the transaction between the DE domain and DTDF domain. When the new control value is generated by CPU, its immediately sent in next clock event to the target using the initiator socket. Hence this perturbation is not due to the synchronisation of TLM MoC.* And this assumption is much concrete in forecoming experiments as there is no modification in TLM's time settings. In Figure 3.2, the response for "10us-cal 1" and "10us-cal 2" illustrates that its a calibration issue. We could see the improved response of the system with calibration change between "10us-cal 1" and "10us-cal 2" for a same activation time of 10us. Thus the mechanism 3.2 explained before helps to create new control values when the system evolves dynamically with improved simulation performance.

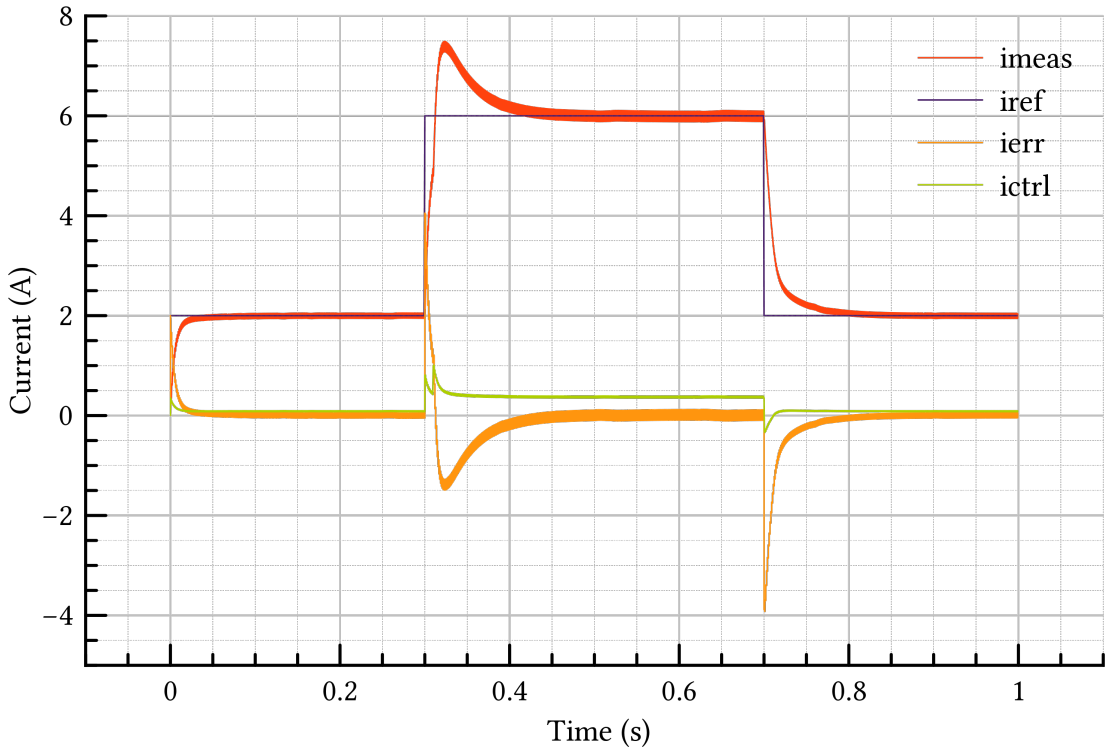


Figure 3.3: Throttle valve response with DTDF (1us)

The Figure 3.3 illustrates the variation of system parameters over simulation period. Due to repeated calibration we are able to achieve a smooth performance. By understanding the period of execution of the control algorithm we are able to get better response in Figure 3.4. The smooth response without any peak overshoot is due to dynamic TDF

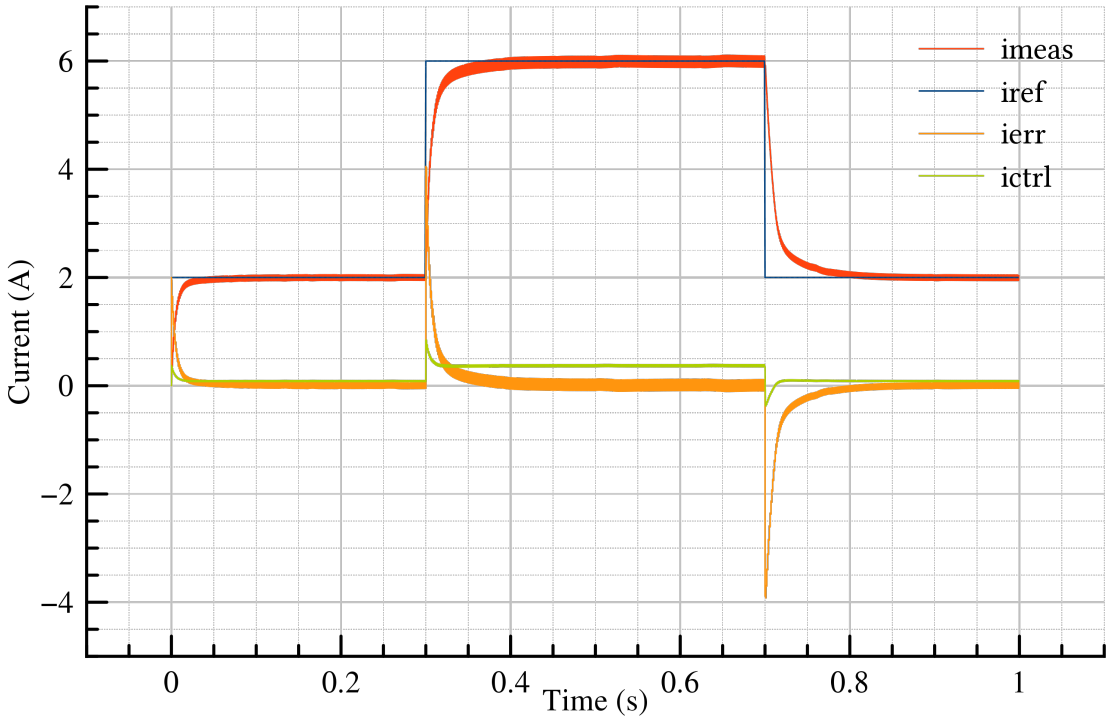


Figure 3.4: System response after refinements (DTDF-10us)

features. *This graph motivates the need for a data-flow oriented embedded system design. The system is able to achieve a settling time of ±2% of error band. The CPU process executes just in time whenever a new data conversion occurs in ADC module dynamically.* Moreover the calibration time required for this experiment with dynamic CPU activation is very less compared to other experiments. Thus an automotive system simulation with AMS not only helps system design in unified environment but also helps calibration and optimization, It motivates this new paradigm to shift towards data flow oriented embedded controllers. This is in fact not advised for a safe critical embedded systems. But with advancements in this work helps us better system developments.

The following graphs in Figures 3.5 and 3.6 compares the error between dynamic activation of CPU with 1us and 10us. This helps system designer to understand the importance of module activation in a DTDF cluster. Usually there is an understanding that when we have fine grained simulation time step it could process the algorithm for better results. Its not the case for a control oriented systems as in automotive ECUs. The benefit having optimal time step and its influence on system and simulation performance is illustrated with this experiments. For a detailed comparison of different DTDF experiments, the tables 4.1 and 4.2 with rows 1 to 5 can be referred.

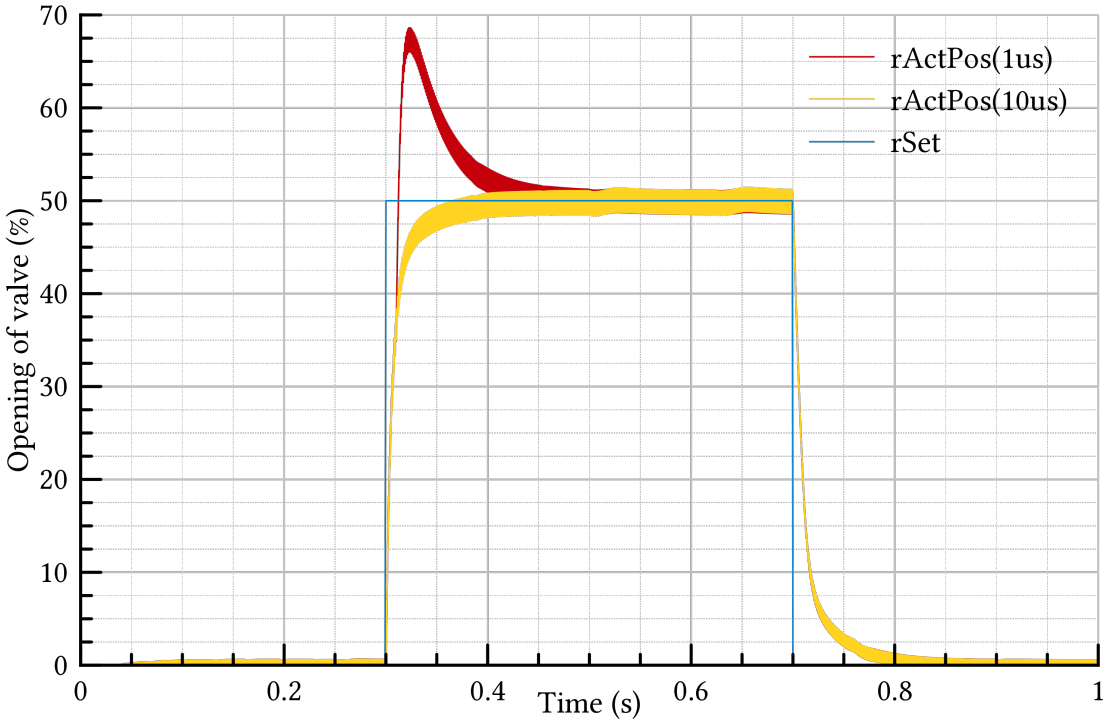


Figure 3.5: Opening of Valve for Step input(DTDF-10us)

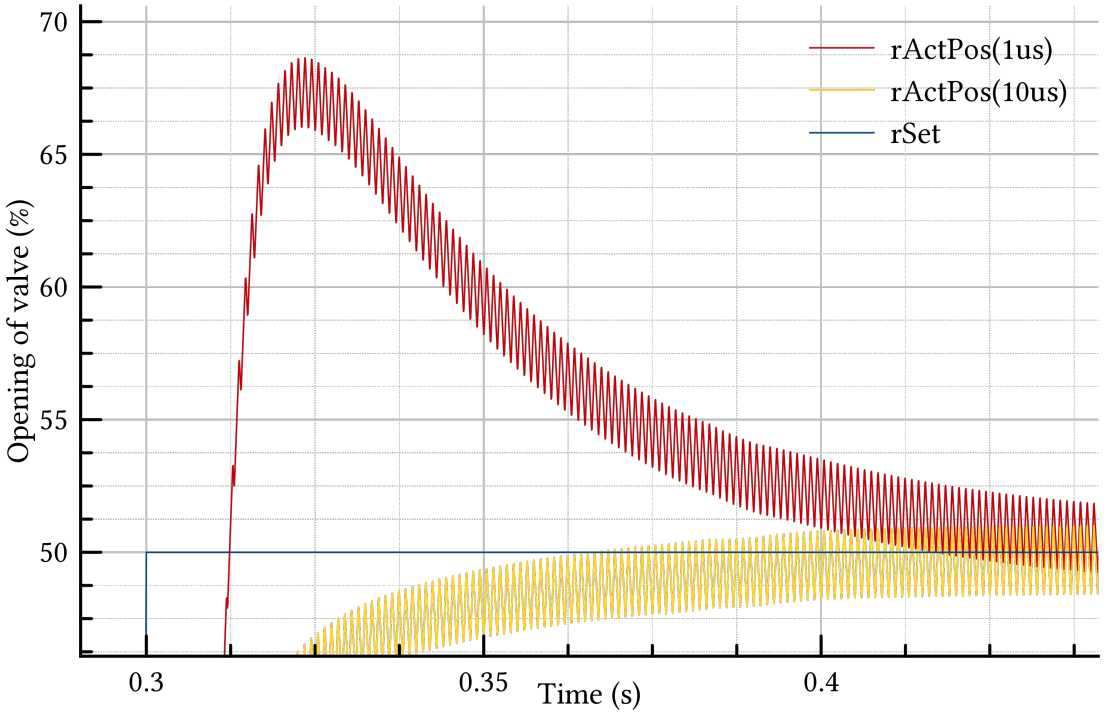


Figure 3.6: Response of DTDF for 1us and 10us initialisation

## 3.3 ETC executable prototype with TDF and DE MoCs

The SW for automotive embedded system involves activation of task by event or timer. In many automotive ECUs its advised to activate the task periodically with out much interaction of interrupts. But its very unclear about the reactive behaviour of the ECU. The two major tasks are engine synchronous and time synchronous tasks. The engine synchronous tasks are activated dynamically in relation to variation of engine speed. while the time synchronous tasks are activated periodically based on a global HW timer. Thus a TDF based clusters are also needed to model the automotive ECUs completely. With this view this framework executes all modules in TDF MoCs. The following experimental frameworks look similar to section 3.1 with some modification in the CPU and PWM modules. In previous frame work the dynamic feature of TDF is implemented in PWM and the whole other modules are coded to accept the dynamic requests from PWM. Even the CPU is able to receive the dynamic behaviour. Here in this the CPU module modification is to accept the clock signal. The PWM module is modified to resemble a hardware timer and it works based on duty cycle and pre scaler. The PWM module receives these values from CPU as an outcome of PI control action and switches ON/OFF its output. when the switch is ON its sending a "12V" to the throttle valve plant model or else no signal a "0V". The throttle valve is a ELN based implementation and can directly accept these voltage signals.

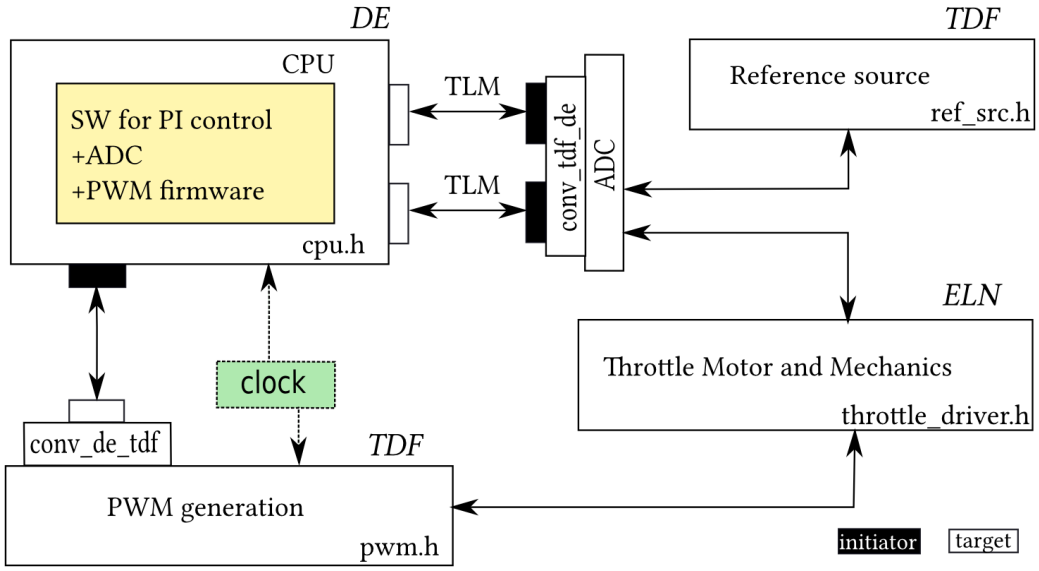


Figure 3.7: ETC executable prototype with only TDF

To understand the benefits of using TDF for control algorithms the following experiments are done. With same calibration values in this frame work the number of time steps and execution parameters are tabulated in tables 4.1 and 4.2.

## 3.4 Comparative study of TDF,DTDF and TLM MoCs

During this study the ETC framework with TDF features are tested with different time steps. In the graph "tdf-10us-cal1" means TDF clusters with 10us time step with

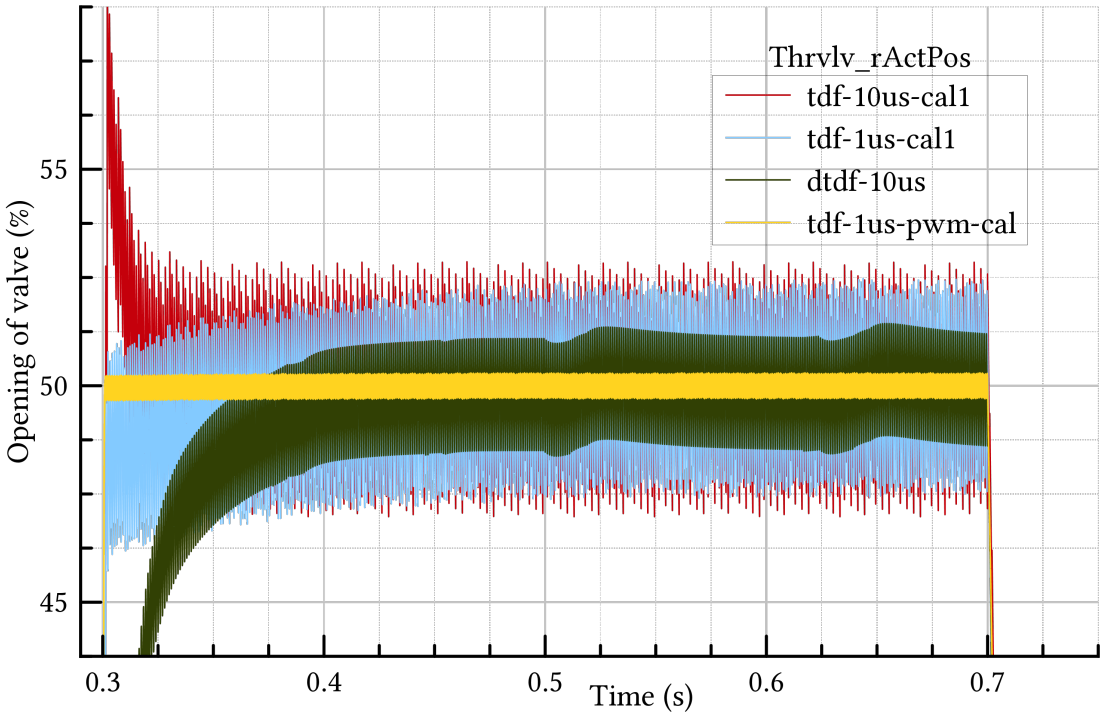


Figure 3.8: Comparison of settling time for TDF and DTDF MoCs

calibration 1, "tdf-1us-cal1" means TDF clusters with 1us time step with calibration 1,"tdf-1us-pwm-cal" means TDF clusters with 1us time step with calibration 1 and modification in PWM pre-scale value, and "dtdf-10us" means DTDF clusters with 10us initial time step. Based on this configuration for time step in TDF clusters, the plot shows that "tdf-1us-pwm-cal" minimum error and settling time. Even if it looks to a ideal plot, but it depends on the type of automotive system under modeling. For example some high end vehicles which travels with a speed of 350 Km/hr, needs the ETC to settle faster. In these vehicles, actuators should have a settling time of less than 10ms.This experiment guides us to choose an optimal time step for a required level of abstraction. In table 4.1 and 4.2 from row 6 to 9 explains the trade off between the accuracy system model with other parameters like simulation time, number of time steps and execution time.

The figure 3.9 and 3.10 shows the digital traces of two frame works. Based on the "clkout1" signal in figure 3.9 its explicit that DTDF able to generate control values exactly after ADC conversion. Thus a simulation set-up like this could accelerate automotive ECU network simulation. Automotive ECU network has multiple control algorithms, so a modeling paradigm like this would speed-up the simulation time. The trace for TDF gives information about a periodic CPU activation with clock cycle of 10us. Not surprisingly the DTDF simulation shows same dynamic behaviour with less number of activations. With the cost of more activations TDF able to achieve less error and settling time. But sometimes more idealistic nature is not needed for modeling, as we try to replicate a exact system in virtual domain. Less realistic is often means less abstractions. Moreover we can even improve the settling time in DTDF as similar to TDF at the cost of increased calibration time.

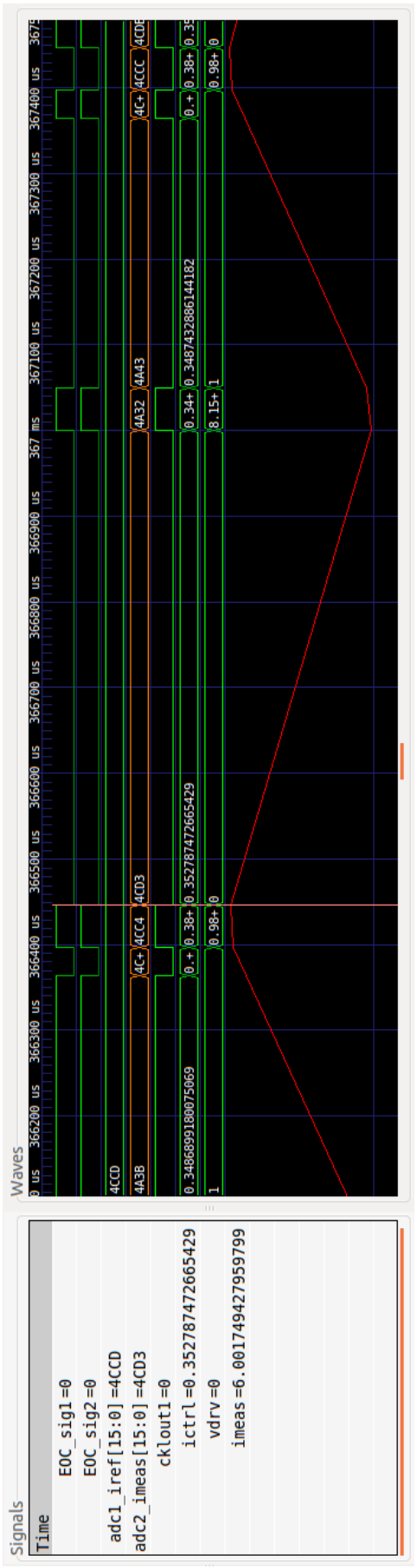


Figure 3.9: Variable CPU activation in DTDF simulation

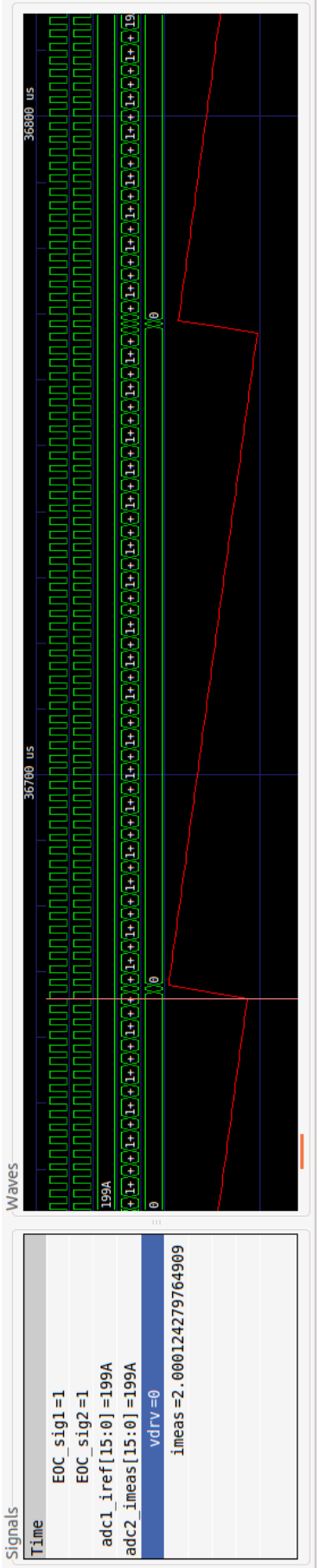


Figure 3.10: Digital trace of TDF simulation of ETC

## 3.5 Towards real world implementation of a Throttle valve control System

The approach explains the usability of virtual models in previous sections towards implementation. Using the TLM based approach as discussed in ETC models we can abstract the communication between different ECUs. As ECU networks are real time reactive networks we can apply the real time constraints of the system in virtual prototype. When these models are virtualized with real time parameters we can directly use the software or hardware designs out of this methodology. The important task of the system engineers is to map corresponding system components with commercial hardware or partition in software. To explain more clearly in Figure 3.11 a real hardware ECU has memory, Registers and communication interface, and even many other things like sensors and actuators. In a principle, the software interacts only with memory, registers and communication interface. By abstraction in space and communication we are able to generate a virtual prototype which is equivalent to real ECUs. When the space and communication is abstracted correctly with real parameters, the software in this virtual platform can be modeled to execute in final ECU hardware. Also the usability of the control parameters also depends highly on accuracy of the plat model (here Throttle valve mechanics).

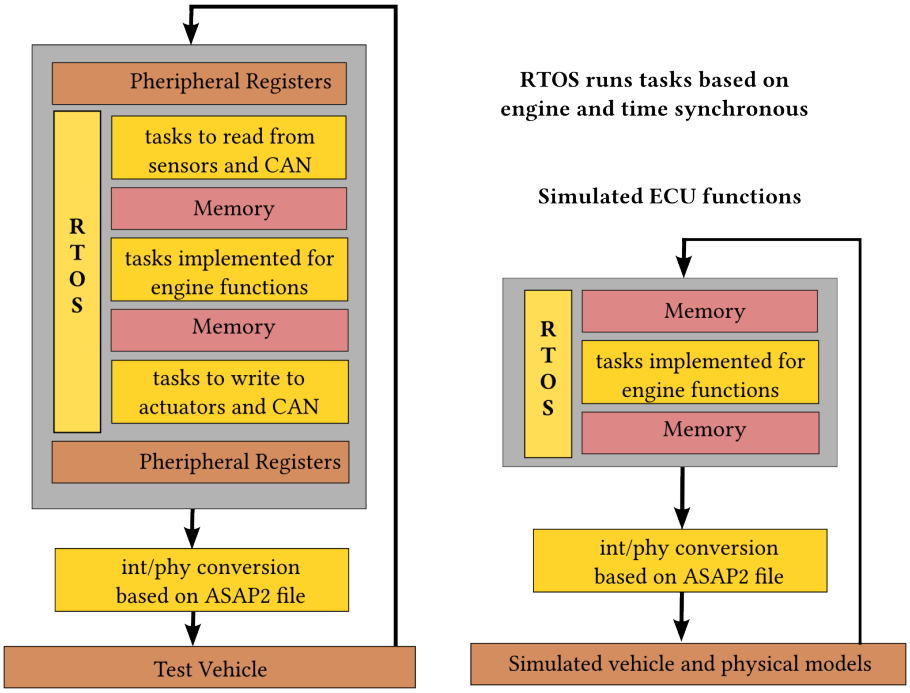


Figure 3.11: Real world ECU vs virtual prototype

Thus with experimental analysis motivates us to develop automotive applications with this methodology. The Figure 3.12 and 3.13 is a real world equivalent of previously discussed virtual prototype. Sometimes the virtual ETC model can be linked with real or emulated system through interface like CAN bus. A typical implementation consumes data over CAN and processes it in virtual ECU and sends back the control values back to real time network (Figure 3.13).

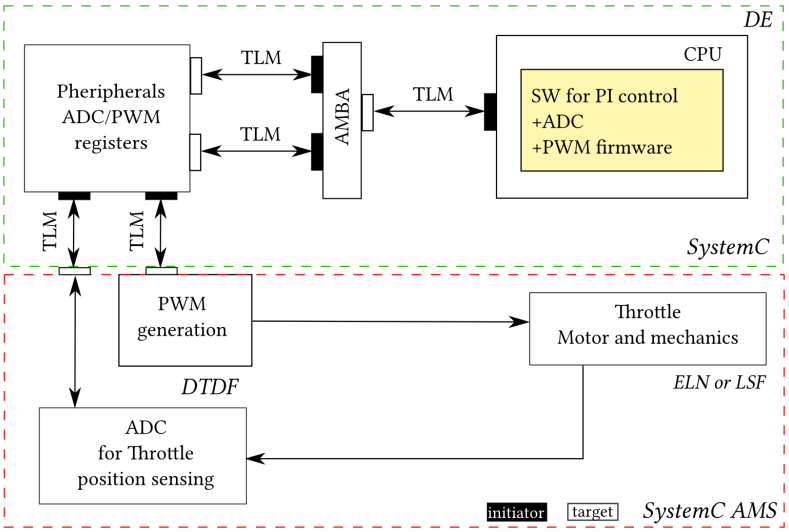


Figure 3.12: Real world equivalent virtual prototype

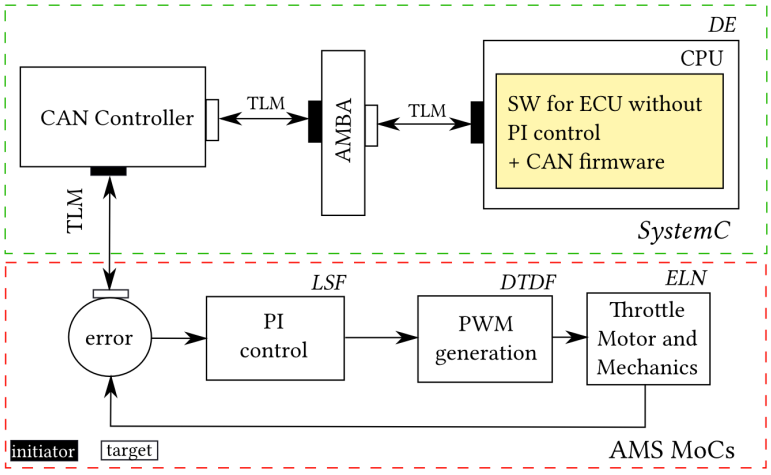


Figure 3.13: Throttle valve controller virtual prototype

The level abstract and accuracy of the system models is upto the interest of the designer. If there is a need to develop only software out of the executable specification. As in Figure 3.13 only SW is developed and for testing and calibration the virtual simulation environment is used. The advantages of this abstraction is faster test bench set-up with reduced level of abstraction and a unified environment.

# Chapter 4

# Conclusion

## 4.1 Automotive System Modeling with AMS and TLM extensions

In this project work, an unified automotive system modeling and HW/SW co-design methodologies were explored with the help of SystemC and its AMS and TLM extensions. In the initial phase of the work, a formal semantics of discrete and continuous domain computational systems were researched for its applicability in heterogeneous systems. This scientific approach helps in mapping the MoCs for different system modeling tasks. Understanding of computational models in terms of model of computation (MoC) helps further with TLM approach for modular design of communication between heterogeneous embedded systems. And its influence on communication between modules and abstraction imposed on system modeling is studied with a SOC model example. During the analysis of a simple SOC model, the need for a dedicated MoCs for Analog and Mixed Signal modeling is explored. Due to limitations of SystemC kernel for mixed signal and mixed domain systems, AMS extensions are studied for usability in automotive system modeling with multi physics environment. The AMS MoCs are based on historical Kahn Process Network, Synchronous Data Flow networks, its technical usage is applied for electronic system level modeling. From which the different levels of abstraction are studied and illustrated for HW and SW development. In automotive system development we need a unified approach from specifying the requirements and till final product release, and it helps the reduction of engineering cost and faster time to market. In order to accelerate the heterogeneous automotive embedded system a virtual prototype is needed. This involves system definition from a formal way and able to prove its equivalence with customer requirements.

At this phase we are able to substantiate the need for a virtual prototype and its difficulties about modeling for multi domain and multi physics environment. But as started with a fundamental research about the execution semantics of the different MoCs of AMS, its important to investigate the state of art SystemC and its extended MoCs for this work. The different MoCs semantically refereed and benefited for this work are discrete event (DE), Timed Data Flow (TDF), Dynamic TDF (DTDF), Linear Signal Flow (LSF) and Electrical Linear Network (ELN). As with this extensions its able to study the semantics and its helpful for system exploration. The work continues to the next phase with a

system modeling task for an automotive subsystem. To illustrate the benefits of newly investigated MoCs, a case study was made with an Electronic Throttle Control (ETC). The ETC is very important subsystem in a modern vehicle for maintaining air/fuel ratio in turn fulfils emission norms.
The modeling of ETC starts with collection of requirements, and modeling same towards a real world equivalent virtual prototype (VP). This VP was developed from the abstract system design and refined for equivalence with the specification. The Throttle valve mechanics and its system behaviour were mathematically modeled and this is used as a plant model. The other functional components of the system were identified. For example here the need for the PI control algorithm is identified. The foremost important parameter for modeling the system parameter is identified, which is the load current of the throttle motor. With this the control algorithm is build to pro-actively control the process variable as load current as a throttle opening position. After this point in the project work, we are able to develop a executable system specification by using major contributions from AMS extensions2.4. This executable model helps us to understand the system with theoretical computer science. Based on the formalisms of MoCs and moving to the higher level of abstractions, and able to plugging in the ideas for HW/SW co-design.

## 4.2 System Model Refinements and HW/SW Partitioning

With a deeper understanding of system model and HW/SW co-design methodologies, the ETC case study was refined by assigning different MoCs based on the system modeling and validation experience in section 2.4. The HW/ SW partitioning is done at this point to choose right simulation semantics to accelerate the model refinements. A CPU model is used to execute SW algorithms and other firmware tasks essential in an embedded system. The refinements involved moving the PI control algorithm towards DE for reconfigurability. This is to be done because the PI control has to be calibrated throughout the life cycle of the product. Hence partitioned towards a SW process in CPU. The PWM is modeled as a DTDF MoC as it can dynamically activate computations in the throttle valve model. This important feature is well explored in terms of simulation time steps and execution time. And the same is studied without dynamic features. The difference between these two techniques helps a system designer to choose right features of simulation based on the system requirement. Here its able to describe the interest of abstraction based on different scale of expectation on quality and accuracy of system model. Regarding the sensing for the control parameters an ADC is plugged into the simulation framework to resemble an equivalent automotive ETC. This ADC is modeled with and with out dynamic features of the TDF. To facilitate these functional components communicate together in the simulation frame work the converters channels and TLM interface are introduced and experimented.

### 4.2.1 Simulation with Different MoCs

The System refinements and HW/SW partitioning helped me to develop a simulation frame work by assigning dedicated MoCs. The framework in Figure 3.1 and 3.7 are used for nine different experiments and results are tabulated in tables 4.1 and 4.2. The table gives an understanding of implementation in DTDF and TDF semantics. It

experimentally illustrates that DTDF has better simulation performance in comparison with TDF. Regarding the control related exploration in Table 4.2, TDF seems to have less error and settling time but at a cost of increased model execution time and immensely great number of time steps. The best results and permitted values for automotive controllers could be a DTDF execution of Exp.No:5 with an execution time of 5.42ms with an error of 1.94% and settling time of 25.9ms. In automotive system development to model the ECU network this table could guide in choosing the right semantics for modeling the system as a unified virtual prototype.

| **Exp.No** | **System Implementation** | **PI Controller** | **t_step (us)** | **t_step(N)** | **Execution Time (s)** |
|---|---|---|---|---|---|
| 1 | DTDF | Dynamic | (initial)10 | 5072 | 1.88 |
| 2 | DTDF | Dynamic | (initial)1 | 5495 | 15.84 |
| 3 | DTDF(after Cal) | Dynamic | (initial)1 | 5134 | 15.96 |
| 4 | DTDF | Clocked | 10 | 4698 | 5.42 |
| 5 | DTDF(after Cal) | Clocked | 10 | 4530 | 5.42 |
| 6 | TDF | Clocked | 1 | 1000000 | 50.68 |
| 7 | TDF(after Cal) | Clocked | 1 | 1000000 | 51.32 |
| 8 | TDF | Clocked | 10 | 100000 | 5.82 |
| 9 | TDF(after Cal) | Clocked | 10 | 100000 | 5.94 |

Table 4.1: Performance analysis of different MoCs based on execution parameters

| **Exp.No** | **Kp** | **Ki** | **Maximum Error (%)** | **Settling Time(ms)(4%)** |
|---|---|---|---|---|
| 1 | 0.178 | $1.256637061\times10^{-3}$ | 2.88 | 80.1 |
| 2 | 0.178 | $1.256637061\times10^{-3}$ | 37.28 | 132.3 |
| 3 | 0.186 | $1.256637061\times10^{-3}$ | 2.64 | 82.2 |
| 4 | 0.178 | $1.256637061\times10^{-3}$ | 70.1 | 275.3 |
| 5 | 2.666 | $31.415926536\times10^{-6}$ | 1.94 | 25.9 |
| 6 | 0.6666 | $31.415926536\times10^{-6}$ | 4.96 | 335.3 |
| 7 | 0.6666 | $31.415926536\times10^{-6}$ | 0.6 | 0.75 |
| 8 | 0.6666 | $314.15926536\times10^{-6}$ | 18.66 | $\gtrapprox$ 4% error band |
| 9 | 0.6666 | $31.415926536\times10^{-6}$ | 5.74 | $\gtrapprox$ 4% error band |

Table 4.2: Performance analysis of different MoCs based on control parameters

## 4.3 TLM as a subset of TDF

The TLM interface with a special case of TDF also studied. The TDF clusters are a synchronous data flow tokens with tagging in time which consumes or produces data samples with all *data types defined by SystemC standard*. This work experimented by communication between these TDF modules and able send tokens of any data type. TLM, when viewed as subset of the TDF could transact only *char* data type under semantic wrapping of standard generic payload. In automotive and heterogeneous system modeling its obvious to use different representation of simulation parameters. For example the

different parameters of a PI control needs a **double** value to maintain the accuracy and quality of the model. To be successful tool in EDA and ESL domain, these language and virtual simulation methodology should enable all possible representation of data. To validate this idea an implementation is done to send the *double* data type control value from CPU to other module. As a TLM transaction can accept and send char data type, an implementation specific workaround was done for this project work to send a double data-type through the generic payload. In this work, a *union* based data packing of *double* and *char* variable is done. In software when the control value is needed to accessed the double variable used. And as when TLM transactions are executed, the *char* variable of union is passed to the generic payload. This kind of work around needed when coupling the samples from TDF or DTDF modules between DE domain modules. This is viewed as a limitation from the side of AMS MoCs rather than justifying a standard for memory mapped BUS architectures modeling. But defining the TLM as subset could overcome limitations of TLM for heterogeneous system modeling. This could explore the use of TLM together as a standardised subset with TDF MoCs. As the TLM used for Loosely or approximate timed modeling technique, when coupling with TDF architecture could increase the simulation performance. In literatures [8] for TLM and AMS interactions are explained as read/write mechanisms with FIFOs . But for an automotive control related use-cases its not a good practice to use FIFOs for modeling control related modules. The FIFO which hold a last stored value, the control action goes with a delay based on size of FIFO. For an expected system response for ETC, value produced by CPU is sent asynchronously at every new computation of CPU 3. This imposes many usage difficulties to synthesize developer specific implementation during a multi domain modeling. When this is formally standardized in TLM or AMS could benefit both digital and analog world. This convergence of standards could helps us to avoid non standard EDA and ESL activities in heterogeneous space. This work greatly helps in guiding the interaction between DTDF, TDF and TLM MoCs with this ETC case study.

## 4.4 Data Flow Driven Heterogeneous Systems and Future Work

From the requirements, refinements and HW/SW co-design of electronic throttle control (ETC), its clear from the experiments that dynamic TDF outperformed with other MoCs. This project work of dynamic activation of control algorithms based on data flows motivated me to design system based on this paradigm rather than event or time driven. From the experiments 3.10 its evident that dynamic CPU processes ADC samples just in exact time and produces new control values and send to PWM module. When we map this feature to an architecture in real world we can further move towards a new novel method of Data Flow Driven Heterogeneous Systems.

### 4.4.1 Advancements towards Data Flow Oriented Modeling

In many life critical embedded systems its not a good methodology to design based on event or data flow. This impose a critical schedulablity problem in operating systems. As we cannot predict the system behaviour before deploying in environment. In automotive ECUs, the scheduling of task is periodic and engine synchronous. There exists a dynamic

data flow when the engine speed varies dynamically this also in-turn activates dynamic tasks. But for system modeling and virtual pro-typing we can still use this dynamic behaviour for increased simulation performance and in turn motivates faster architectural explorations.

This work motivates further to implement more automotive components with the discussed AMS and DE MoCs and experiment with this new paradigm. This could help us to explore more modern and state of future architectures for modern engineering needs.

**source code** https://github.com/thiagsaran/etc.git

# Bibliography


[1] *IEEE Standard for Standard SystemC® Language Reference Manual*, 2012. IEEE Std. 1666 (Rev 2005).

[2] *Standard SystemC® AMS extensions 2.0 Language Reference Manual*, 2013.

[3] Banerjee Amal and Sur Balmiki. *SystemC and SystemC-AMS in Practice*. Springer, 2014.

[4] A. T Bahill and S. J Henderson. Requirements development, verification, and validation exhibited in famous failures. *Systems Engineering*, 8:1–14, 2005.

[5] Martin Barnasconi. *Whitepaper: SystemC AMS Extensions-Solving the Need for Speed*. Open SystemC Initiative, 2010. http://www.accellera.org/resources/articles/amsspeed/.

[6] Black.D.C, Donovan.J, Bunton.B, and Keist.A. *SystemC: From teh Ground Up. Second Edition*. Springer, 2010.

[7] Markus Damm. *Crossing Modeling Paradigms in System Models*. PhD thesis, University of Kaiserslautern, 2014.

[8] Markus Damm, Jan Haase, and Christoph Grimm. Towards co-design of hw/sw/analog systems. *Design Methodologies for Secure Embedded Systems and Lecture Notes in Electrical Engineering* ., 78:1–24, 2011.

[9] Markus Damm, Jan Haase, Christoph Grimm, Fernando Herrera, and Eugenio Villar. Bridging mocs in systemc specifications of heterogeneous systems. *EURASIP J. Embedded Syst.*, 2008:1–16, 2008.

[10] Gary Nichols Daniel McKay and Bart Schreurs. Delphi Electronic Throttle Control Systems for Model Year 2000; Driver Features, System Security, and OEM Benefits.ETC for the Mass Market Technical report. Technical report, Delphi Automotive Systems, 2000.

[11] Johan Eker, Jorn Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorffer, Sonia R. Sachs, and Yuhong Xiong. Taming heterogeneity — The Ptolemy approach. *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*, 91(1):127–144, January 2003.

[12] Embecosm Ltd. *Building a Loosely Timed SoC Model with OSCI TLM 2.0, A Case Study Using an Open Source ISS and Linux 2.6 Kernel*, 2013. http://www.embecosm.com/appnotes/ean1/ean1-tlm2-or1ksim-2.0.html.

[13] Robert Bosch GmbH. *Bosch Automotive Electrics and Automotive Electronics*. Springer, 2014.

[14] C. Grimm, M. Barnasconi, A. Vachoux, and K. Einwich. An Introduction to Modeling Embedded Analog/Mixed-Signal Systems using SystemC AMS Extensions, June 2008.

[15] William J. Palm III. *System dynamics*. The McGraw-Hill Companies and Inc, January 26, 2009.

[16] G. Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Information processing*, pages 471–475, Stockholm, Sweden, Aug 1974. North Holland, Amsterdam.

[17] Edward Ashford Lee and David G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, C-36(1):24 – 35, 1987.

[18] Prof.Bill Messner and Prof. Dawn Tilbury. *Control Tutorials for MATLAB and Simulink*, 2013. http://ctms.engin.umich.edu/CTMS.

[19] Thomas M. Parks, Thomas M. Parks, and Thomas M. Parks. Bounded scheduling of process networks. Technical report, 1995.

[20] Pierburg GmbH. *Product photos from wesite of Pierburg GmbH*, 2013. http://www.kspg.com/en/products/.

[21] QTronic Stuttgart. *Silver Virtual ECU simulation software*, 2013. http://www.qtronic.de/en/silver.html.

[22] Jing-Chung Shen. New tuning method for pid controller. In *Control Applications, 2001. (CCA '01). Proceedings of the 2001 IEEE International Conference on*, pages 459–464, 2001.

[23] University of Berkeley. *Hardware/Software Codesign Group*, 2013. http://embedded.eecs.berkeley.edu/Research/hsc.

[24] A. Vachoux, C. Grimm, and K. Einwich. Systemc extensions for heterogeneous and mixed discrete/continuous systems. In *International Symposium on Circuits and Systems, Kobe, Japan*, May 2005.

[25] Qing-Guo Wang, Tong-Heng Lee, Ho-Wang Fung, Qiang Bi, and Yu Zhang. Pid tuning for improved performance. *Control Systems Technology, IEEE Transactions on*, 7(4):457–465, 1999.