An Experiment on Exploiting Virtual Platforms for the Development of Embedded Equipments

P. Cuenot1[3], E. Jenn1[4], E. Faure2, N. Broueilh2 , E. Rouland1[5],
1: IRT Saint-Exupéry, 118 route de Narbonne, 31042 Toulouse, France
2: ASTC France, 42 avenue du Général De Croute, 31100 Toulouse, France

Abstract: *Virtual engineering* methods and tools based on simulation have become a privileged mean to reduce time-to-market and product cost. However, design and verification activities still need to be improved to manage the ever increasing complexity of electronic products and their interactions with heterogeneous environments. In particular, an important challenge is to master the real time properties of the product composed of interacting hardware and software components.

In this paper we propose a pragmatic approach to use virtual platforms to verify gradually and accurately the properties of a system under design. We illustrate the approach on an example.

Keywords: Virtual platform, SystemC, Verification. Simulation, Heterogeneous environment.

1. Introduction and Motivation

New embedded electronic systems impose a new leap in product integration. The progress of the System on Chips (SoC) market share clearly illustrates this need. In parallel, embedded electronic systems or Electronic Control Units (ECU) need to be optimized to integrate new functions usually developed using analog parts at an acceptable cost.
Basically, the challenge is to maximize the usage of SoCs while keeping reasonnable development costs. Embedded electronic systems are usually reactive: they are part of some control loop closed through their environment. The temporal properties of the elements of this loop are of prime importance as they determine the overall performance of the implemented function (response time, stability, etc.). Accordingly, those properties must be carefully monitored and the product (including drivers, IPs, sensors, actuators…) shall be carefully designed to comply with those properties. In this context, being able to simulate a complete control chain within its environment is of major interest, and the benefits of the approach are as high as it can be applied soon, continuously and "smoothly" during the development process.

Towards this goal, there is a strong need to move from segregated hardware and software simulation to system-level simulation. This requires the modelling and simulation of the system's components and environment at various levels of abstraction, representativeness, and accuracy. The SystemC language and simulation kernel [1] provide such capabilities.

SystemC is supported by an open source ecosystem organized in the context of Accelera Systems Initiative [1]. Recently, the Electronic Design Automation tools industry (EDA) extended commercial offers by integrating SystemC. The tool environment offers modelling services on the top the language, scripting capabilities to automate test execution, co-simulation interface to interact with external tools or actual electronic prototypes, etc.

Despite (or because of) the large offer of technical methods and tools, some methodological guidance is required to ensure a safe, high-quality, and cost-effective development and verification process.

Therefore, we (i) propose an iterative process to optimize the use of SystemC, (ii) apply it on part of a system, and (iii) show its benefits. Focus is placed on verification activities carried out with the system's environment to demonstrate the preservation of the components properties during the successive development phases.

Our paper is structured as follows. Section 2 introduces the virtual platform concept and gives an overview of some significant related works. Section 3 presents the proposed approach for an iterative development and verification process. In section 4, the experimental setup is explained and section 5 gives some evaluation results. Finally, the conclusion reports feedback on methods and tools evaluation by application of the process and draws perspective for future work.

2. Related Work

A virtual platform is a hardware simulator executing embedded software. The hardware simulator is usually built on top of an Instruction Set Simulator (ISS) of the processor connected via buses to memory and peripherals such as timers, general I/O, communication interface, etc. The ISS may be implemented in SystemC or in any other general purpose language.

In a typical configuration, communications are modelled using SystemC Transactional Level Modelling

[3] Seconded from Continental Automotive France, 1 avenue Paul Ourliac, 31036 Toulouse France
[4] Seconded from Thales Avionics, 105 avenue du Général Eisenhower, 31100 Toulouse, France
[5] Seconded from ACTIA Automotive, 5 rue Jorge Semprun, 31400 Toulouse, France

(TLM) in order to achieve high simulation speed. Peripherals and memory are register accurate, they communicate according to the TLM paradigm, and their behaviour is implemented in SystemC/C++. The application software is the actual code compiled for the target processor. The SystemC non-preemptive simulation kernel orchestrates the execution of all components of the platform.

Among typical examples of virtual platforms, we can mention the Infineon Tricore™ SoC [2] based on the QEMU ISS or the SockROCKET LEON3 virtual platform developed by ESA [3] implemented in Python with SystemC/TLM buses and peripherals. Note that the ISS can be developed by silicon suppliers from proprietary architecture modelling languages, such as Freescale's ADL/uADL [4], and then be integrated with peripherals to build a complete platform.

Virtual platforms have been experimented on various industrial use cases, such as automotive power train applications [6]. Those experiments have demonstrated the need for appropriate methods during the development of the platform components and their integration.

In the industry, the different uses of virtual platforms map to the organisation of the supply chain: design and verification of SoC on the silicon suppliers side (component), design and verification of software on the equipment suppliers side (system).

On the system side, virtual platform are used to estimate the performances of hardware and software architectures, and perform early verification activities. In particular, debug and verification phases are simplified thanks to the high observability and controllability of virtual platforms compared to real hardware. Furthermore, using virtual platforms moves the simulated hardware parts out of the critical path: hardware development phases can start once the definition of the hardware has matured and has been (virtually) validated.

Finally, virtual platforms also provide:

- A high configurability and flexibility allowing a particular platform to be configured and elaborated within minutes *provided that the models are available*.
- A capability to integrate models with heterogeneous abstraction levels (in particular temporal abstractions thanks to the versatility of SystemC/SystemC-TLM/ SystemC-AMS)

In its Return On Investment (ROI) analysis on electronic system level design, Synopsys [5], one of the main EDA tool supplier, announces cost reduction opportunities by reducting the number silicon iterations and a productivity increase of x2-x5 for software development.

To reach such a ROI, the virtual platform design flow shall be optimized so as to (i) minimize the cost of models and (ii) maximize the credit one may take from verifications performed using those models. An approach consists to use simulation models as a form of an "executable design artifact" that is refined all along the development process, from the implementation agnostic logical level down to the physical software and hardware levels. Obviously, this approach is conditionned by the quality of the model and particularly on the properties preserved by the model. Those qualities must be clearly stated and become part of a contract binding the provider and the user of the model. A specific care shall be taken on timing properties of the hardware model and of the environment impacting the overall system behaviour.

The Socket project [7] has proposed such a design flow for the development of critical embedded SoCs. The development steps maps to the SystemC/TLM modelling styles. This allows to co-simulate hardware and software, and to introduce and verify properties gradually (in particular temporal properties). The top level design of the SoC architecture is focused on hardware bus traffic optimization, not on the complete set of properties allocated to the SoC. In this project, the modelling of the environment is limited since the systems considered are not reactive.

The COMPLEX project [8] uses UML/MARTE to model and explore the design space of embedded systems. Power and performance are the exploration criteria. The various abstractions of the processor micro-architecture, of the SoC internal buses, etc. allow an early and efficient simulation. Those abstractions also impair accuracy and raise sensitivity problems (in particular with target compiler optimization versus native compiler and internal SoC bus communication control). Compared to Socket, COMPLEX placed focus at system level where interconnected ECUs communicate through communication buses. The precision in low-level modelling is not considered besides the use of existing SystemC component libraries. Additionally the verification activities are not formalized.

With respect to the previous works, our approach is aimed at covering a larger modelling spectrum, from high level models down to behavioural hardware models. Emphasis is placed on verification of real-time properties considering the ECU's environment.

3. An Approach for the Development and Use of Virtual Platforms

To benefit the virtual platform's "good properties", the objective is to (i) minimize the development cost of the virtual platform and (ii) maximize the usage of the virtual platform.

To achieve the first objective, one shall (i.a) maximize the reuse of existing models (possibly reusing them from previous developments), (i.b) maximize the automatic generation of platform models (or skeletons) from existing design models, (i.c) develop simple models covering the necessary and sufficient features and details with respect to the validation and verification objectives. These topics will be addressed in Section 4.d.

To achieve the second objective, the strategy is two-pronged: (ii.a) the "most expensive" problems shall be identified and tackled first, (ii.b) the model shall be detailed up to the point where the necessary and sufficient precision and accuracy are obtained to solve (ii.a).

Here, we propose an informal approach somewhat similar to a Failure Mode and Effect Criticality Analysis (FMECA) applied on a development process. This approach takes into account: the cost of evaluations including the cost of model development, the cost of model execution and the risk inherent to a "sloppy" evaluation. According to this interpretation, the frequency of occurrence of an error is related to the quality of the measures (its accuracy and precision). The gravity integrates the potential impact of the error on the development process (e.g., the number and nature of the activities to be redone), and the impact on the product under design. Once the analysis is complete, reducing the overall risk consists to reduce the probability of occurrence of the evaluation error (e.g., by refining the model in order to account for phenomena that have a significant impact on the behaviour of the system), (ii) reduce the gravity of the error by improving the robustness of the process (e.g., by introducing margins), (iii) detect evaluation error by adding additional checks.

In this paper, focus is placed on the temporal properties, so the trade-off between cost and accuracy/precision is essentially focused on the modelling of temporal aspects. We consider the three levels — or programming styles —defined by SystemC-TLM: untimed (U or *functional*), loosely-timed (LT), and approximately-timed (AT). As the design improves, the verification objectives get more and more focused and may eventually require a fine grain temporal modelling. This change is determined by the property to be verified.

Failure modes

Contrary to hardware devices, there is no common, standardized list of failures at functional level. Candidate failure modes are identified and selected on the basis of expertize, experience, etc. Here, focus is placed on temporal errors such as under and over estimation of delays, non-compliance with SW/HW interaction sequencing or hardware design constrainst. The objective is to determine which verification technique to apply by considering the design faults, their effect on the system and the cost of the detection / mitigation means.

Criticality

Our estimation of criticality is (roughly) estimated by a cost. The principle is trivial: the cost of the verification means shall be somehow commensurate or related to the direct and indirect costs of the error. The cost of the error is roughly estimated by its criticality, which depends on its probability of occurrence and the costs of its effects including the cost of detection means (technical and human) and the cost of error elimination (redesign and refactoring,…).

Error propagation

With respect to a classical FMECA, the approach differs because design errors propagate across design artefacts (space) and across design phases (time). In particular, an error in the dimensioning of a parameter in phase i may impact the design choices done in phase $i + 1$, and so on.

Any verification approach is basically aimed at preventing such propagation by (i) detecting design errors as soon as possible in the design process and (ii) as soon as possible in the dependency chain that relate design artefacts.

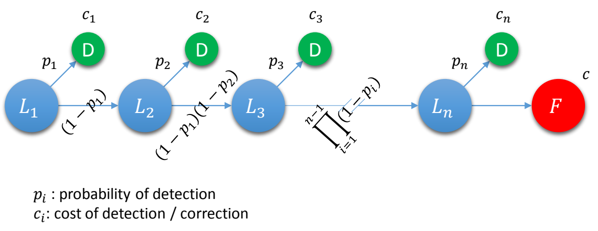

c_1
c_2
c_3
c_n
D
D
D
D
p_1
p_2
p_3
p_n
c
L_1
L_2
L_3
L_n
F
$(1-p_1)$
$(1-p_1)(1-p_2)$
$\prod_{i=1}^{n-1}(1-p_i)$
p_i : probability of detection
c_i: cost of detection / correction

Figure 1: FMECA overview

This "approach" has been applied on some simple functions of our experimental setup. The corresponding use cases are described in section 4.b and 4.c.

Back to the important question of the ROI of such multi-viewpoint, multi-level approach, the following questions shall be answered:

- What is the cost of developing (all) those additional models? The answer obviously depends on their complexity. Hence, the development effort for a SystemC-TLM timed model of a simple timer is lower than one men×month, while a complex ISS can require more than twenty men×months.
- If models are develop by some third party, how do we formalize the "contract" that binds the model provider to the model user? Stated differently, how can we specify the domain to obtain significant results with the virtual platform model?
- Finally, how does this additional cost compares to the gain due to the early validation?

In the sequel of the paper, we propose to answer those questions by applying the virtual platform approach on a small example: an autonomous rover.

4. The experimental setup

In order to evaluate the proposed development process and supporting tools, and estimate its actual benefits, we use it for the development of a small mobile vehicule, or "rover", named "TwIRTee".

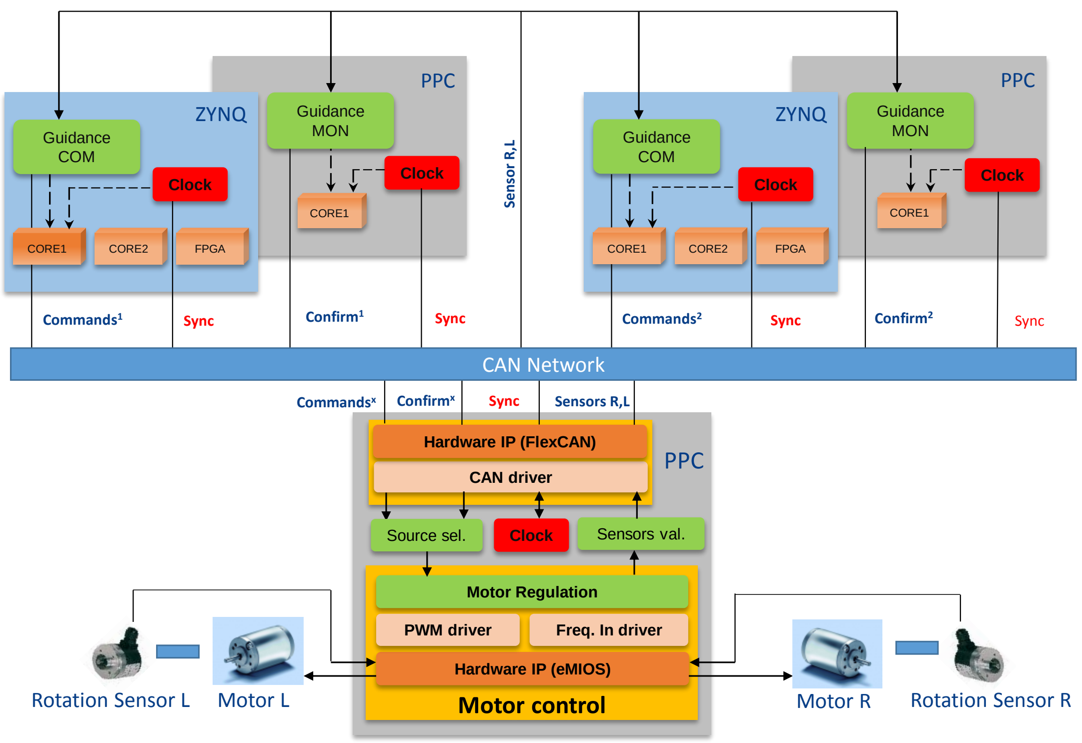


Figure 2: Overview of the TwIRTee equipment

a. The TwIRTee demonstrator

TwIRTee is a three-wheeled autonomous rover fitted with a camera and various other sensors (odometry, positioning,…). Its operational role is very simple: (i) move itself on some predefined tracks from a point A to a point B (a "mission") while avoiding other rovers.

TwIRTee is developed within the INGEQUIP project at the Toulouse *Institut de Recherche Technologique* (IRT) Saint-Exupéry. IRTs are new research structures established under the auspices of the French *Agence Nationale de la Recherche* (ANR) aimed at favouring the transfer of innovation from laboratories to industries.

TwIRTee is designed so as (i) to cover the major topics addressed in the project namely: early validation, architecture exploration, performance prediction, and formal verification. Furthermore, it is aimed at covering issues, functional and architectural elements specific to three industrial domains: automotive, space, aeronautics.

Accordingly, the missions, functions and the architectural elements are determined so as to tackle or exercise one or several issues: for instance, the "localization" function relies partially on imaging so as to exercise hardware / software space exploration and co-design; the highly redundant architecture provides the experimental setup to perform early performance evaluations (including dependability).

An overview of the TwIRTee plaftorm is given on Figure 2: the computing platform is composed of 2 COM/MON channels that host the main "mission" functions and one channel dedicated to power supply generation and motor control. A clock synchronization protocol is implemented and distributed on each ECU communicating by the CAN network.

The rover displacements is achieved by the motor control ECU controlling a two-wheeled powertrain composed of 2 CC motors, 2 reduction gearboxes, 2 quadrature encoders, 2 wheels. The setpoint for motor regulation is selected from the 2 commands (COM) and monitoring (MON) channels.

The methods introduced in Section 3 are applied on two simple functions: the clock synchronization (Fck) and the PWM motor control (F_{PWM}).

b. Clock synchronization

The clock synchronization protocol is used to provide the rover's computation units with a "common" time reference. The protocol has been proposed in [9]. It is built on top of the CAN network.

Principles

The common time reference – or virtual clock (vc) – satisfies the precision, rate and accuracy properties. The *precision* property states that no two synchronized virtual clocks may differ by more than a given value. For instance, at any time t, any virtual clock shall show a time "less than 100μs" from any other virtual clock. More formally, if nodes k and l participate to the protocole:

$$\exists \delta_v : |vc_k(t) - vc_l(t)| \leq \delta_v \qquad \text{(P1)}$$

In practice, the achievable precision depends on two main parameters Γ_{tight} and $\Gamma_{\text{max agree}}$:

$$\delta_v = \delta_{vi} + 2\rho T \qquad \text{(P2)}$$

$$\delta_{vi} \geq (1+\rho)\Gamma_{\text{tight}} + 2\rho\Gamma_{\text{max agree}} \qquad \text{(P3)}$$

where

- T is the resynchronization period, ρ is the physical clock drift
- Γ_{tight} is the network propagation delay (around 10μs), $\Gamma_{\text{max agree}}$ is the agreement delay which depends in particular on the number of tolerated faults and on the background traffic of higher priority.

First, let's analyse the failure modes, their "probabilities" of occurrence, and their effects.

Failures

- F_{F1M1}: Erroneous $\Gamma_{\max agree}$, underestimation
- F_{F1M2}: Erroneous $\Gamma_{\max agree}$, overestimation
- F_{F2M1}: Erroneous Γ_{tight}, underestimation
- F_{F2M2}: Erroneous Γ_{tight}, overestimation

Probabilities of occurrence

- O_{F1M1}: VERY HIGH, because (i) many factors contribute to the communication times, and (ii) the dynamic behaviour of CAN is not trivial (see [10] and [12]).
- O_{F1M2}: LOW, because the mode for fault F1 is much likely F1M1.
- O_{F2M1}: VERY LOW because this parameter is part of the specification.
- O_{F2M2}: VERY LOW *(*same reason as F2M1).

For space reason, we do not consider failure modes F2 any longer.

Effects and cost impact

- E_{F1M1}: Actual accuracy lower than expected. Cost impact is MODERATE: (i) the system being redundant (MASTER/SLAVE, COM/MON) many functions rely on the synchronization precision in particular via discrepancy margins, confirmation times are affected. However, the risk for a large effect on the synchronization is low thanks to the ρ[1] factor in (P2).
- E_{F1M2}: Actual accuracy greater than expected. Cost impact is LOW: the network and CPU loads are higher than necessary but no rework is expected. (Note that, generally speaking, the impact could be very high because it could lead to select and oversized computing platform and / or network. In our very case, there is no risk of such effect).

In the absence of dedicated detection means, F_{F1M1} and F_{F1M2} are likely to be detected only in operations.

Probability of detection

- D_{F1M1}: VERY LOW because $\Gamma_{\max}$ is a worst-case situation that is difficult to observe in operation.
- D_{F1M2}: VERY LOW because the effects are hardly visible.

Cost of correction

- C_{F1M1}: VERY HIGH
- C_{F1M2}: VERY HIGH

[1] ρ is in the range of 10^{-6} s/s

From the combination of a MODERATE cost, a VERY LOW detection probability and a VERY HIGH detection / correction cost, we decided that it was worth adding a new detection means with a MODERATE cost and HIGH detection coverage.

Consequently, we decided to test the clock synchronization protocol on a bit-level virtual model of the CAN capable of simulating the actual effects of background traffic and error occurrences.

(Note: we consider that the behaviour of the network in the presence of fault is simulated. However, in the particular case of CAN, analytical models of the CAN bus latency are already available (e.g., [10]), but simulation models allow to consider faults models of arbitrary complexity.

c. PWM motor control

Power is delivered to the rover motors via a H-bridge (4 transistors, see Figure 3) controlled by a PWM signal. The PWM duty cycle is computed by the motor regulator from the speed set point provided by the rover guidance controller and the actual speed measured using the wheels' optical encoders. The wheels' optical encoders generates quadrature signals acquired as frequencial input and then integrated for speed determination.

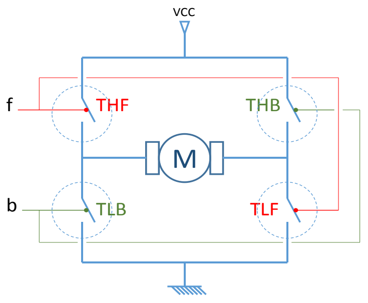


Figure 3: Motor H-Bridge control

The PWM controller was assessed under two perspectives: PWM timing and PWM design constraints both having effect on real time properties.

PWM control generation

The PWM generation device hold the following functional requirements (P4 - non exhaustive list)

- Frequency and resolution step of PWM control: 10kHz frequency with a 0.1 µs step)
- Latency for application of new PWM value (next cycle)
- Immediate desactivation of active state of the PWM (lower than 1 ms)

Compliance to the previous requirements may be achieved in many different ways, including:

- Pure software implementation toggling an output port (resolution will be certainly an issue)

- Pure hardware specific implementation with single register interface and fixed frequency (too specific solution)
- Mix mode with a simple timer, control by software interrupt and latch of an output (resource will be an issue)
- Hardware dominant solution with multiple register interface but "reasonable" or limited resource (most realistic in this basic example)

Attached to selected scenario we have also a list of design constrainst (C4) stemming from domain experience, available resources, etc. For instance :

- Register shall be 32-bit wide. At most 4 registers must be necessary (duty cycle value, frequency value, cycle counter and register control)
- The timer frequency range shall be in [100Hz,100KHz] (10kHz for motor control)
- The effect of the frequency and duty cycle change shall only occur at the next PWM cycle

For the evaluation of the motor controller we will consider the failure mode "wrong design" (F2M1).

The interaction between the PWM and the H-bridge (the device that physically controls the power signal) raises another possible failure mode. As the H-bridge provides no overcurrent protection, care shall be taken not to switch transistors located on the same side of the bridge on "at the same time" in order to avoid shortcuts. This situation can be prevented by introducing a *silence time* between the commutations of opposite transistors. The duration shall in particular take into account the transistor switching time. The respective failure mode are noted F1Mx.

The following property shall hold:

$$\forall\, tb_i, tf_j :\ \left|tb_i - tf_j\right| \geq \Delta t_{dz} \qquad \text{(P5)}$$

Where tb_i (resp. tf_i) be a time at which signal "backward" (resp "forward") is active, and Δt_{dz} is the duration of the "silent zone" where no transistor is active.

Failures modes

- F_{F1M1}: Erroneous Δt_{dz}, underestimation
- F_{F1M2}: Erroneous Δt_{dz}, overerestimation
- F_{F2M1} : Wrong design

Probability of occurrence

- O_{F1M1}: LOW
- O_{F1M2}: LOW
- O_{F2M1}: MEDIUM (component are selected to fulfill all forecoming applications)

Effects

- E_{F1M1}: During a rotation sense change, the opposite MOSFETs of the H-bridge transistors are on at "the same time". Cost impact is VERY HIGH: this configuration is basically a shortcut. Depending on the duration of the shortcut, the dissipated energy may leads to the destruction of the two transistors and the power supply. No other function is affected by the error.
- E_{F1M2}: During a rotation sense change, the PWM signal is delivered slightly later to the H-bridge. Cost impact is NEGLIGIBLE: the rotation of the wheel is slightly delayed which has no further impact on the rover's capabilities.
- E_{F2M1}: In case of wrong design with non capable component the function performance must be downgraded. In the worst case, a complete redesign may become necessary. Cost impact is HIGH because the problem will be detected during verification but it will have an impact on overall product planning.

In the absence of dedicated detection means, F_{F1M1} and F_{F1M2} are likely to be detected only in operations. F_{F2M1} will be detected lately during product verification meaning high effort for redesign.

Probability of detection

- D_{F1M1}: LOW. Depending on the duration of the "short-circuit", the number of forward/backward commutations, the error may stay undetected for a long time. However, it may eventually reduce drastically the lifetime of the transistor / power supply.
- D_{F1M2}: VERY LOW.
- D_{F2M1}: HIGH (as processus for verification are mature)

Cost of correction

- C_{F1M1}: LOW. The correction is basically a modification of a constant in the PWM management code.
- C_{F1M2}: LOW (same as C_{F1M1})
- C_{F2M1}: HIGH to VERY HIGH

For the "silence time underestimation" fault model, the combination of a VERY HIGH cost, a LOW detection probability and a LOW correction cost leads the introduction of a new detection means. This means shall have a MODERATE cost and HIGH detection coverage.

In practice, we created a virtual platform to host the PWM driver software and implemented a system-C observer to check (P5).The virtual platform shall provide a representation of time "compatible" with the property at stake. Here, we used a SystemC AT model of the PWM hardware driver.

For the "wrong design" fault model, the combination of HIGH cost, HIGH detection and HIGH (to VERY HIGH) correction cost leads to the introduction of a new detection means aimed at securing later development phases. As the functional requirement (P4) and design constrainst (C4) are expressed in terms of hardware and software interactions, the virtual platform shall at least provide a LT abstraction.

To facilitate the verification of the functional properties (P4) a functional model is developed (initiating also the test bench environment). From this level, requirements and associated properties are refined and allocated. During the decomposition process, some requirements may become “contracts” binding the different components of the motor control chain.

Figure 4 depicts some of these contracts:

- The PWM controller shall ensure a silence-period greater than Δt_{dz} *and* the H-bridge transistors shall switch is less than (e.g.,) $\Delta t_{dz}/100$ (P5).
- The power dissipated by the electrical motor shall be less than 15W. Therefore, the PWM duty cycle shall never be greater a given ratio for a given time. This contract propagated from motor power dissipation constrainst is not considered in the article.

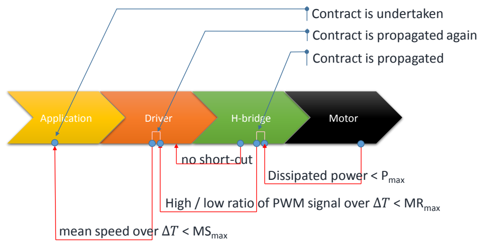


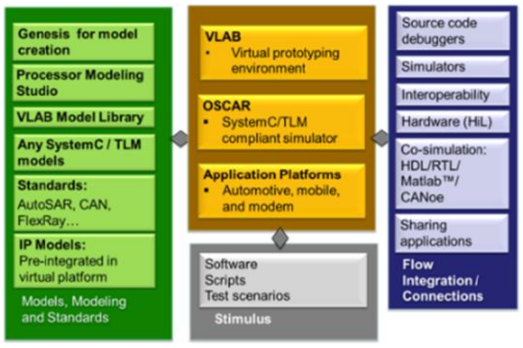
Figure 4: Motor drivers contracts

Another contract defined during the PWM controller design binds the hardware and software. It concerns the static and dynamic definition of the interface (see property C4). This contract is verified at the LT and AT modelling levels.

Section 5.b describes the use of the contract verification on the PWM controller.

d. The virtual platform

VLAB™[2] is used to develop the models and build virtual platforms. It provides a programmable and interactive environment for the assembly, configuration, programming and operation of electronic system level simulations, such as virtual system prototypes and virtual platforms, as well as other applications (see Figure 5) . A virtual platform integrates together simulation models and other simulation objects, scripts, tools, test and infrastructure software, and target software.



Genesis for model creation
Processor Modeling Studio
VLAB Model Library
Any SystemC / TLM models
Standards: AutoSAR, CAN, FlexRay…
IP Models: Pre-integrated in virtual platform
Models, Modeling and Standards
VLAB
Virtual prototyping environment
OSCAR
SystemC/TLM compliant simulator
Application Platforms
Automotive, mobile, and modem
Software
Scripts
Test scenarios
Stimulus
Source code debuggers
Simulators
Interoperability
Hardware (HiL)
Co-simulation: HDL/RTL/ Matlab™/ CANoe
Sharing applications
Flow Integration / Connections

Figure 5: Tool organization

The development of models does require a solid expertise understanding SystemC concept and language (object oriented). Moreover, the iterative refinements of models may be achieved by different persons with different coding skills.

The tool framework facilitates the creation of models thanks to a Genesis library compliant with SystemC and IP-XACT standards. It allows to capture model structure and connectivity and to automatically create C++ skeleton for the implementation of the behavioral part of the model. This library is also available in Python meaning models can be developed and debugged in a much faster and simpler way than in C++.

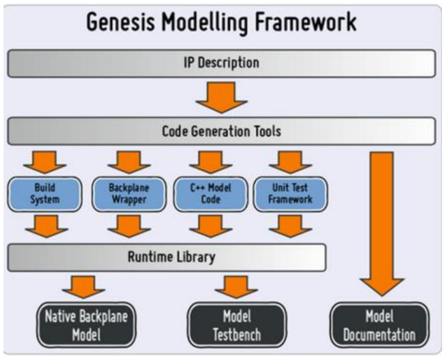


Figure 6: Genesis Framework

Finally, the tool provides an integrated and interactive execution platform leveraging again Python capabilities. It provides a simple yet efficient user experience for assembling virtual platform, debugging virtual platform (setting HW & SW breakpoints) and scripting test scenario (including fault injections in a non intruisive way) .

Modelling and assembling a virtual platform in Python are the key functionality used to support the implementation of the case study.

5. Implementation approach

a. Clock synchronization

We used ASTC VLAB™ “CAN toolbox” to create a model of our computation and communication infrastructure (5 nodes) and to inject faults during the execution of the clock synchronization protocol. A CAN node comes with two models, at token and bit levels. The toolbox provides an API for the configuration and control of nodes, including activation, frame transmission and frame reception. Python scripting allows to access directly to API, to send CAN frame and control the bit engine in order to introduce error in the CAN frame. These features have been used to exercice the clock synchronization protocol in situations where bit-level errors lead to multiple retransmissions of the same message. Such situations would have been difficult to obtain using the actual hardware.

[2] VLAB^{TM} is a product of ASTC

b. The Motor control

The PWM controller scenario, as elaborated in section 4.c, is realized using VLAB^{TM}. Some parts of the complete model were developped using SystemC and Python modeling capabilities while some other parts were directly taken from the hardware library (for TLM transactor or MPC5674F AT models). The modelling scope for contract verification of the PWM controller is enlarged to the overall motor controller feature. For the sake of the demonstration, we consider that the C code of the motor controller was generated from the same Simulink® model used to design and validate the controller.

The development phase of the motor controller corresponds to the three predefined modelling style U, LT and AT.

Untimed model

The Untimed model is a functional model. It describes how the PWM signal is generated from the rotation speed setpoint and the actual speed measured from the optical encoders (see Figure 7 below). The Hardware Abstraction Layer API (HAL) is defined at this level. It allows to implement a static interface between application layer and driver abstraction using physical data interface with the motor regulation algorithm (% for PWM and speed in km/h for integration of quadrature signal).

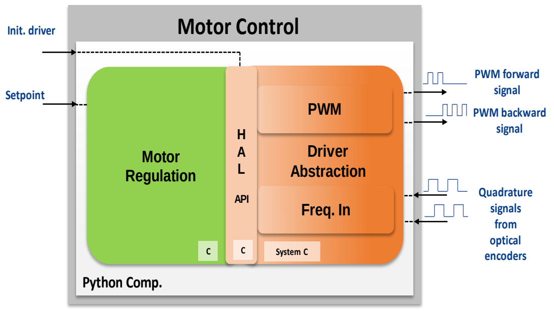


Figure 7: Functional model of Motor Control

The PWM controller is integrated in the driver abstraction SystemC module by implementing two specific C++ methods for the definition of the API (*init_mc_driver() and put_mc_pwm(int32 value)*).
The SystemC module integrates the driver abstraction with the PWM controller and the motor regulation. It declares two public methods *init_mc() and put_setpoint(value)* to allow access to PWM controller and to write into the setpoint data.
It includes one thread (or method) activated every 10 ms calling *c_mc_ctrl() for regulation* control activation.
Using the toolset, the SystemC motor control model is first wrapped up into a python module which is then imported into the tool environment as a new component of the virtual platform. To use this newly defined component, the user first instantiates and initializes it. Then, he/she sets the setpoint value using to the python API *(driver_obj=vlab.get_instance("MC").obj, driver_obj.init_mcr(), driver_obj.put_setpoint(value))* .

This first level of modelling is a necessary step to build the subsequent models. We take benefit of it to perform a first verification of the design with respect to properties P4 and P5.

To do so, a test environment (or testbench) is built as shown on Figure 8. In particular, property P5 (dead zone) is checked using a dedicated SystemC/python component (P5 checker on the Figure 8) that is connected to PWM output signals via a monitor component). The test driver generates a scenario where wheels are moved forward and backward.

To close the control loop, a very simple model of the motor / encoder devices is developed and integrated in the platform. In this model, the frequency of the quadrature signals is considered directly proportional to the value of the PWM duty cycle (this is a good approximation as far as the frequency of the PWM signal is sufficiently high). This model is used to get a first insight on the performances of the regulation (response time, stability). In the future we will investigate co-simulation with Simulink® dynamic of motor itself.

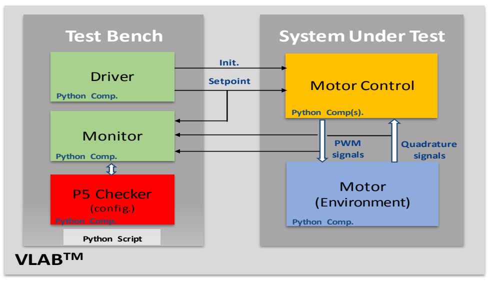


Figure 8: Test bench environment

To observe the results of the P5 verification, a trace is placed on the P5 checker diagnosis output port using tool tracing capabilities. See the trace of the scenario result in Figure 11.

Loosely Timed model

The Loosely Timed Model is a structural and behavioral refinement of the Functional Model.

It introduces several hardware and software components:

- The timer IP abstraction generates the clocks used to generate the PWM signal. This IP is modelled in systemC. It integrates all design constraint defined in C4 property (register size, frequency range).
- The driver abstraction provides the interface to the timer IP. This software component is defined in C. It is interfaced with the timer abstraction IP via a Hardware Software Interface (HSI). It integrates the C4 HSI contract (static and sequence interface).
- A SystemC bus driver interface allows to access to the timer IP peripheral. The C to SystemC interface complies with the TLM2 loosely timed standard. It is available in the tool component library.

This structural and behavioural refinement preserves the properties demonstrated on the functional model. It is used to express the Hardware Software Interface (HSI) contract (C4).

The HSI is implemented using five 32 bits registers:

- CNT running counter register (range of 24 bits) with 10ns resolution (range from 6Hz to 100 MHz, capable for 10KHz with 0.1% of precision)
- A and B compare registers for PWM control (range of 24 bits)
- CCR control register with enable bit (UCPREN bit 6), polarity bit (EDPOL bit 24) and fixed buffered mode (OPWFMB matching eMIOS definition bit 25-31)
- CSR status register with overrun bit (FLAG bit 31)
- Sequence transfer for buffered register A,B at the end of the PWM cycle and guarantee of B always greater than A.

The objective is not to implement the complexity of an hardware IP like the eMIOS, but build a simplified IP fulfilling the fonctionnality with respect of a reduced HSI accessed via TLM2 access (untimed).

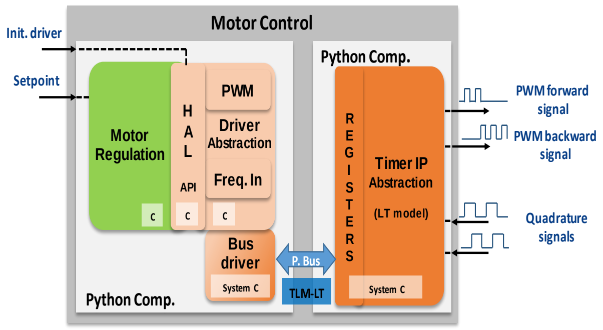


Figure 9: LT model of the Motor Control

The motor control is now split in two python components as depicted in Figure 9. This split allows the observability using VCD. The python components are then instanciated with the test bench architecture as in Figure 8. For the testbench, the interface for the motor control is identical as the functional model. The P5 checker can also be reused to demonstrate the dead zone contract. See Figure 11 for trace of the results.

The HSI interface contract can be demonstrated thanks to respect of the API resolved during virtual platform building and sequence demonstrated by instrumentation capability to trace TLM transaction (register access and value transported).

Approximated Timed model

The last model integrates approximated timing on bus communication and on hardware IP resource component access (internal definition and average processing time in hardware IP). Model can also complete the register interface when extra control registers are necessary for pre-implementation constraint, such as merging several LT models into a more complex and configurable hardware component. In any case the predefined HSI contracts shall be preserved: at least the same registers are used to interact with the component, according to the same sequence (protocol).

A use case more relevant than timer IP could be an image processing hardware accelerator merging LT models with modeling of internal timing interconnection.

In our experiment, the LT timer IP is mapped to the MPC5674F eMIOS AT model. The Timer IP is connected to a bridge and to the memory bus of the PowerPC core via the peripheral bus (see Figure 10). The target software is the binary software, including final drivers, application software compiled for the µC target.

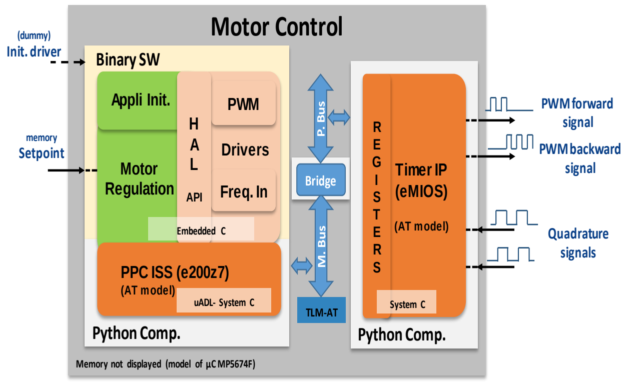


Figure 10: AT Model of the Motor Control

The test bench is only slightly adapted with respect to the one used at the more abstract levels. A python transactor component is inserted and connected to the same test bench driver. It integrates one method for target memory introspection using debugging API feature of the tool environment (*write_memory()*) and a second one to dummy the motor control initialization as it is now performed by the µC start-up code. The simulation scenario generator and the property checker component do not change. The P5 checker can be reused for the validation of the contract defined in the FMECA process. The functional property P4 of the PWM control is verified. The C4 contract on the HSI register access and sequence is also verified.

The simulation trace typical of every refinement level is depicted below in the Figure 11.

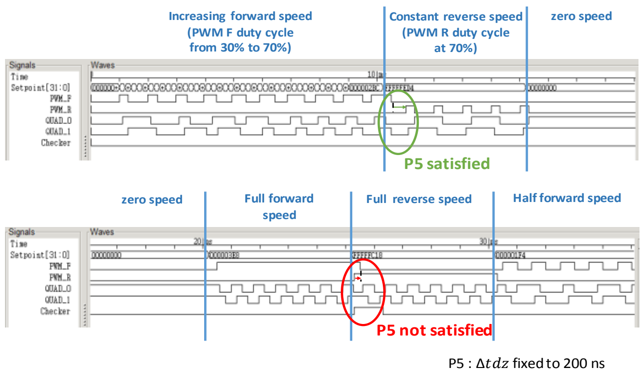


Figure 11: Trace of dead zone validation

6. Conclusions and future work

In current SystemC ecosystem, the adequacy of model representativity for verification objective and the quality of the simulation models curbs the large deployement of virtual platform. Our approach provides a first level of guidance and formalization of the modeling process.

The “FMECA-like” approach introduces – to some limited extend, however – objective criteria to determine whether a refined model is necessary or not. It allows to start the modelling phase at the adequate abstraction level for the verification of the selected property. The formalization of the development and the test of an architecture compliant with SystemC modelling standard allows a sound design covering hardware software codesign process. We demonstrated that test elements can be reused all over the model refinement. Moreover, the contract approach provides backbone for decomposition and guarantee on the coverage of the requirements.

The tool automation features simplify the platform models generation for component interfaces description and component bindings. However, for the creation of a new component model, the remaining manual operations (code implementation, platform integration and test campaign) still represent around 90% of the development costs. In actual practice the elaboration of simulation models follows a typical top-down approach based on successive abstractions. Such refinement has been formalized through the control motor modelling experiment. More important, the models have been continuously verified. In our case, the development and testing of two additional abstracted models is only estimated at 10 to 20% of the overall effort.

Balanced to the model development effort, the proposed methodology with virtual platform provides strong benefits on the overall product development. The contract and the model based communication facilitate inter domain communication and improve the design quality of the product. It helps to reduce significantly the re-work iterations. The effort gained can be estimated at 10 to 20%. This was demonstrated in our experimentation and also on porting the OS Trampoline [15] on MPC5674F microncontroller for twIRtee. The early hardware software integration by simulation reduces operational set up and validation.

The approach proposed in this paper will be experimented for the development of other parts of the INGEQUIP project demonstrator. In particular, the camera-based line tracking function will be developed according to the same approach. This function requires a lot of computing power, focus will be placed on the issue of hardware/software design space exploration.

7. Acknowledgement

The authors thank all people and industrial partners involved in the Ingequip project. This work is supported by the French Research Agency (ANR) and by the industrial partners of IRT Saint-Exupéry Scientific Cooperation Foundation (FCS): Actia Automotive, Airbus, Airbus Defence and Space, ASTC Design Partners, Continental Automotive, SAGEM, Systerel, Space Codesign Systems and Thales Avionics.

8. References

[1] SystemC core langage definition, Accelera Systems initiative standards, http://accellera.org/downloads/standards/systemc

[2] B. Koppelmann and all, “An open and Fast Virtual Platform for Tricore^{TM} based SoC Using QEMU, DVCON Europe 2014

[3] ESA SockROCKET virtual platform, http://www.esa.int/Our_Activities/Space_Engineering_Technology/Microelectronics/SoCROCKET_Virtual_Platform_-_SystemC

[4] Freescale ADL, uADL open source project, http://opensource.freescale.com/fsl-oss-projects/

[5] Franck Schirrmeister, Synopsys, “System Level Market trends”, 2010 Synopsys Interoperability forum

[6] P. Cuenot, N. Tavernier, JM Talbot, “Embedded Software V&V using virtual platforms for Powertrain application”, ERTS 2008 Toulouse, France

[7] V. Lefftz, J. Bertrand, H. Casse, C. Clienti, P. Coussy, L. Maillet-Contoz, P. Mercier, P. Moreau, L. Pierre, et E. Vaumorin, « A design flow for critical embedded systems », in 2010 International Symposium on Industrial Embedded Systems (SIES), 2010, p. 229–233.

[8] F. Herrera, H. Posadas, P. Penil, E ; Villar, F. Ferreo, R. Valencia, G. Palemro « The COMPLEX methodology for UML/MARTE Modeling and design space exploration of embedded systems”, Journal of System Architecture, volume 60, issue 1, Jan 2014.

[9] L. Rodrigues, M. Guimarães, and J. Rufino, ‘Fault-tolerant clock synchronization in CAN’, in Proceedings of the 19th Real-Time Systems Symposium, Madrid, Spain, 1998, pp. 420–429.

[10] I. Broster and A. Burns, ‘Timely use of the CAN protocol in critical hard real-time systems with faults’, in Real-Time Systems, 13th Euromicro Conference on, 2001., 2001, pp. 95–102.

[11] Electronic Reliability Design Handbook, US Department of Defense, MIL-HDBK-338B, Oct. 1998.

[12] Road Vehicles – Controller area network (CAN) – Part 1: data link layer and physical signalling’, ISO/TC 22/SC 3N, 7-May-2014

[13] http://cache.freescale.com/files/32bit/doc/ref_manual/MPC5674FRM.pdf

[14] VLAB™, ASTC, http://www.vlabworks.com/

[15] Trampoline OpenSource RTOS projet, http://trampoline.rts-software.org/