A Virtual Prototyping Platform for Real-time Systems with a Case Study for a Two-wheeled Robot

Daniel Mueller-Gritschneder, Kun Lu, Erik Wallander, Marc Greim, Ulf Schlichtmann
Institute for Electronic Design Automation, TU Muenchen, Germany
daniel.mueller@tum.de, kun.lu@tum.de, ulf.schlichtmann@tum.de

***Abstract*—In today's real-time system design, a virtual prototype can help to increase both the design speed and quality. Developing a virtual prototyping platform requires realistic modeling of the HW system, accurate simulation of the real-time SW, and integration with a reactive real-time environment. Such a VP simulation platform is often difficult to develop. In this paper, we propose a case-study of autonomous two-wheeled robot to show how to develop a virtual prototyping platform rapidly in SystemC/TLM to adequately aid in the design of this instable system with hard real-time constraints. Our approach is an integration of four major model components. Firstly, an accurate physical model of the robot is provided. Secondly, a virtual world is modeled in Java that offers a 3D environment for the robot to move in. Thirdly, the embedded control SW is developed. Finally, the overall HW system is modeled in SystemC at transaction level. This HW model wraps the physical model, interacts with the virtual world, and simulates the real-time SW by integrating an Instruction Set Simulator of the embedded CPU. By integrating these components into a platform, designers can efficiently optimize the embedded SW architecture, explore the design space and check real-time conditions for different system parameters such as buffer sizes, CPU frequency or cache configurations.**

Keywords*—*Virtual Prototyping, Transaction Level Modeling, Real-time Constraints, Embedded Systems

I. INTRODUCTION

Today, real-time systems are designed in a top-down flow. For example, a high-level model of the system is developed in MATLAB to simulate the functionality of the control algorithms. The finished algorithms are then implemented either in the SW or HW component of the model of the embedded system. The implementation may cause certain new effects that may limit the quality of the control such as execution time of the SW, communication delays or buffer overflows. Often, such issues only become obvious when a prototype of the system is available because RTL simulation of the system is often too slow to investigate the system for many seconds of real-time. Additionally, the simulation of the control algorithms requires a reactive environment. In an in-the-loop-fashion, the testbench must read the outputs of the control system and, based on these, produce the new inputs for it. This requires the designer to model not only the embedded system but also the controlled actors, the sensors, which are read by the control system, and the physics of the controlled system.

In this work we present an integrated SystemC/TLM/Java virtual prototyping platform for an autonomous two-wheeled robot. The robot acts as an inverted pendulum that has strongly instable behavior and produces hard real-time-constraints on the control algorithm. Additionally, the robot can move autonomously by following a red line on the floor. This requires the system to run a computationally intense computer vision algorithm in addition to the control to stabilize the robot.

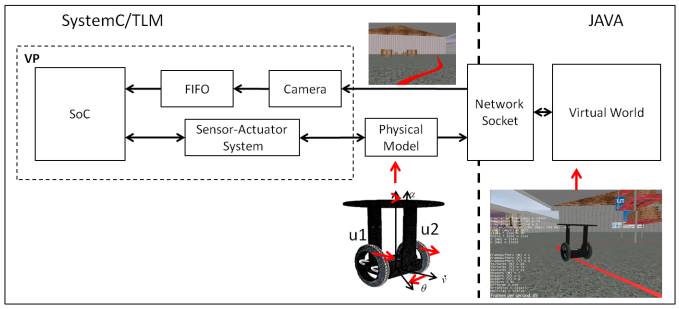

Figure 1. Overview of the VP of the two-wheeled robot

An overview of the Virtual Prototype (VP) of the robot is given in Fig. 1. The virtual world is implemented in Java. It can be used to visualize the motion of the robot and produces the camera inputs. The SoC for the control, the camera and FIFO buffer as well as the actuator and sensor system are modeled in SystemC at transaction level. Additionally, a physical model is implemented in SystemC that models the dynamics of the robot with a set of differential equations (DEs).

The advantages of developing such a VP are that the complete system behavior becomes transparent. We can investigate implementation effects such as communication delays, SW execution times on the real-time control and also check for implementation problems, e.g. a buffer overflow in the frame FIFO of the camera. Additionally, the VP can be used for debugging because any register of the Instruction Set Simulator (ISS) of the CPU, or HW blocks, as well as the memory can be traced. To optimize the control algorithm, the computation can also be traced together with the system state such as velocity or pitch angle. Due to the implementation at transaction level, simulation performance is very high compared to RTL simulation, 40s simulation time can be simulated with around 9min execution time. It is demonstrated in this paper how such a VP can be developed fast and efficiently for the case study of the two-wheeled robot. In the future, this work can act as a prototyping platform for other systems by including their physics and more sensor and actor

978-3-9815370-0-0/DATE13/©2013 EDAA

models. To the knowledge of the authors, such a platform for SystemC that combines a virtual world with a TLM embedded system model with actors and sensors as well as a model of the physics is not yet available.

The remainder of the paper is structured as follows. In section II, the related work is discussed. In section III, the simulation testbench is presented. The robot control system is presented in section IV. We present an evaluation of the case study in section V. Section VI concludes.

II. Related Work

In the area of virtual prototyping of real-time systems, there exist many approaches that contribute to the modeling and simulation of embedded systems, e.g. [4-7]. Also, there exist many tools to simulate and refine models for control applications that also model the physics of the controlled system and featuring a virtual environment, e.g. [8-10].
However, in this paper it is shown how to achieve such a simulation platform quickly using SystemC and a Java rendering engine. This has the advantage that SystemC has a free simulator and good support for fast simulating TLM models that can still be very near to the implementation, e.g. featuring cycle-approximate timing. Also, many system modules are freely available. SystemC is also a C++ library such that other non-electronic models, e.g. for actors or for the physics can be easily integrated.

III. The Prototyping Simulation Platform

For the virtual prototype of the two-wheeled robot, we developed a reactive simulation platform, which is outlined in this section.

A. Virtual World

The camera inputs for the robot are produced by a virtual world that was implemented in Java with the JMonkey Engine [3]. The Java program is connected to the SystemC simulation by a network socket. Several cameras can be placed into the environment to generate the camera data for the robot and display the movement of the robot from a follower position.

B. Physical Model

The two-wheeled robot acts as an inverted pendulum. The physics are described by a set of differential equations (DEs). The state variables $\mathbf{x}$ are the velocity, orientation angle θ, orientation angle rate $d\theta/dt$, pitch angle α, pitch angle rate $d\alpha/dt$ as well as the position x and y of the robot [1]:

$$d\mathbf{x}/dt = \mathbf{f(x)} + \mathbf{g(x)}\,\mathbf{u} \quad with\ \mathbf{x}=[x\ y\ \theta\ \alpha\ d\alpha/dt\ v\ d\theta/dt]^T \quad (1)$$

The inputs $\mathbf{u}=[\ u_1\ u_2]^T$ are the motor torques of the right and left motor. We use the Forward Euler method for the integration of the DEs:

$$\mathbf{x}(z+1) = \mathbf{x}(z) + \Delta t\ (f(\mathbf{x}(z)) + g(\mathbf{x}(z))\ \mathbf{u}(z)) \quad (2)$$

This is a discrete time model of the physics. Equation (2) must be evaluated periodically in a time step Δt for the simulated time points $t=\Delta t \cdot z$. This can be easily implemented as a SystemC thread that updates the robot state variables dependent on the current motor torques in fixed time steps before calling the wait function *wait(delta_t,SC_NS)*.

C. Sensor/Actuator System

The Sensors and actors connect the physical model to the embedded system. The modeled tilt sensors and gyroscopes can read the angles and angle rates from the physical model. They store these values in registers, which can be read by the embedded system via a system bus as shown in Fig. 4. The motors are torque controlled. The embedded system can write the desired torque values to registers in the motor controllers. The motor model computes the physical torque values from the desired values and current motor state including effects such as friction and back-emf force. The computed physical motor torque values are then supplied as inputs to the physical model. The sensors and actors are connected via TLM sockets to the system bus and physical model. The system bus socket would be a real bus interface of the modules while the connection to the physical model is part of the analog environment model.

D. Camera System

The camera system is implemented in SystemC. It models the camera and the frame FIFO to store the pixels. The camera receives frames from the Java environment for a certain camera position that depends on the state of the robot given by the physical model. The data as if the camera is fixed on the robot, i.e. that the camera also is tilted up and down when the robots leans forward or backward for acceleration or de-acceleration.

E. Coupling of SystemC and Java

The SystemC simulation and Java rendering task run as separate processes. An important issue is the synchronization between the Java world and the SystemC simulation as well as the data exchange. The Java program requests the current values of the state variables together with a time stamp from the physical model of the SystemC simulation via the network socket. With it, the robot is moved to the new position in the virtual world. The Java program also notifies the SystemC simulation about the time point when the next frame from the camera is available. As soon as it is reached, the SystemC simulation stops to synchronize. The Java engine saves the view from the camera associated with the robot and sends the pixel data to the SystemC camera module via the network socket. The SystemC simulation waits until the data is available before advancing the simulation time further. The frame rate must be known in both environments.

IV. Control System

A. Control Algorithm

The control algorithm of [1] was implemented to stabilize the robot. It consists of a two-level controller. The lower level

controller balances the robot for a desired pitch angle α_r and heading angle by controlling the motor torques. The higher level controller uses α_r as gas pedal to set the desired velocity. In addition, we have developed a fast path recognition algorithm that determines the orientation angle and velocity for the control algorithm. It was implemented to follow a red line on the floor. For this, the red channel is extracted from the camera frame. Then a threshold on the red channel is applied to generate a binary image as shown in figure 2. Two markers are shifted from left to right and vice versa over the image starting at the most bottom line to detect the red line and remove any other objects.

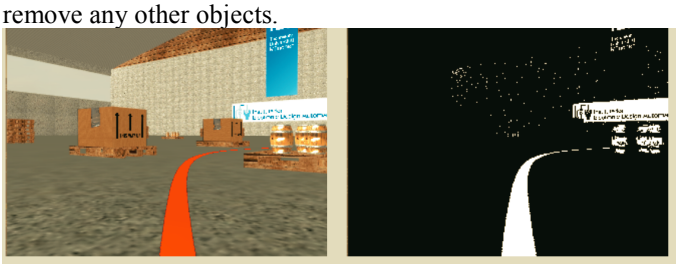

Figure 2. Red channel extraction and threshold function

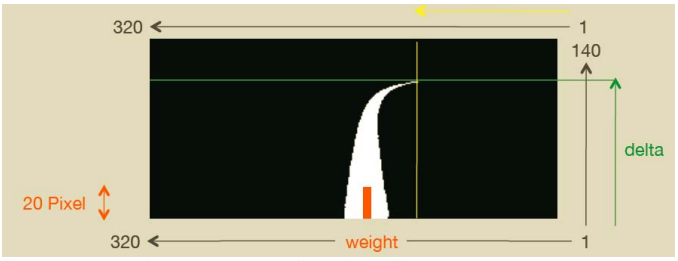

Figure 3. Control parameters to follow the red line

After the line is extracted, the controller keeps the line in the center of the image as shown in figure 3, which results in the robot following the red line. Two other parameters are used to detect an upcoming curve in order to slow down the robot in advance.

B. *SW/HW System*

The transaction level model (TLM) of the control system is illustrated in figure 4. We use the Or1ksim Instruction Set Simulator (ISS), which models the open-source Or1200 CPU. The ISS is written in C and supplies a SystemC wrapper and cycle-approximate timing including cache and memory access times [2]. The loosely-timed modeling style with blocking transactions is used to achieve high simulation performance.
The TLM was created with the target in mind that the modules inside the shaded box may be implemented as embedded system on a FPGA. For this, the used OpenRISC processor is available as Softcore. A custom GNU toolchain is available for this processor and a library, e.g., with math functions and for interrupt and exception handling. The SW runs bare-metal on the CPU without operating system.

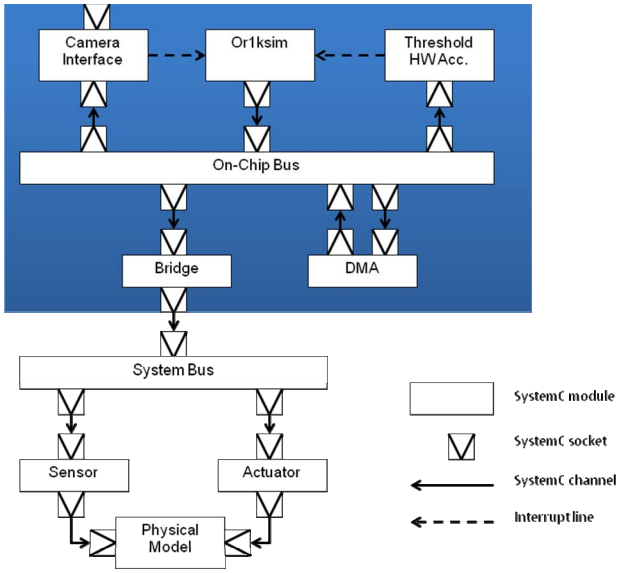

Figure 4. Transaction level model (TLM) of the control system

V. EVALUATION OF THE VP

In the experiments, a test environment with a red line is used to simulate the modeled embedded system of the two-wheel robot with different details and configurations. The simulation results are then used to validate the control algorithm and to check real-time constraints.

A. *Functional model without execution delays*

SystemC offers the possibility to refine models in several steps. In our case study, we first implemented the control algorithms as SystemC modules. The control algorithms are executed in an endless loop in a SystemC thread process once at the beginning of each control period. There are no execution delays such that the desired values are computed in zero time after the sensors are read. From the evaluation, we could quickly extract that the stabilization control of the robot must have a 3ms control period. The control period for the controller that sets velocity and orientation by detecting the red line must be at least 100ms, resulting in a frame rate of 10 frames/s. Execution time was almost real-time with 40s for 40s simulation time.

B. *Loosely-timed virtual prototype without HW acceleration*

In the second step of the case study, the control algorithms were implemented to run on the ISS with an interrupt and timer based SW architecture. Communication delays were also added to the model. It shows that the delays had some impact on the control as shown in figure 5. As can be seen there is higher oscillation in the orientation angle at 15s, at which the robot is turning fast. This oscillation was not seen in the untimed model.
It also shows that the real-time constraints could only be met for a CPU frequency of 200MHz, which may be hard to realize for a FPGA implementation. Additionally, when all control algorithms run on the CPU, no data could be fetched from the camera during the non-preemptive execution of the stabilization control. This added a new real-time constraint

because the stabilization control required to finish before an overflow of the frame FIFO buffer happens. Otherwise data for every frame is lost, because the stabilization control is executed three times in the duration of one frame period. Execution time was 4:49 min for 40s simulation time.

C. *Loosely-timed virtual prototype with HW acceleration*

From an in-depth analysis, it could be detected that the time to fetch a frame from the camera and extract the channel as well as run the threshold function is very time-consuming to run on the CPU. These tasks can also be easily implemented in HW. HW acceleration was added to the system as shown in figure 4. The direct memory access controller (DMAC) is configured to extract the red channel from the pixel data coming from the camera interface and to transfer the pixel data line by line to the threshold HW accelerator. It compares the red value of the pixel with a threshold and produces the binary image. The CPU only needs to fetch the binary image resulting in far less accesses due to a compression from 24bit per pixel to 1 bit per pixel. The HW accelerator issues an interrupt to the CPU when a line of a frame is available. Since the DMAC also acts as initiator on the bus, arbitration delays were added to the bus accesses of the DMAC and CPU in this model.
Investigation of this configuration resulted in a tolerable frequency of 100MHz. Figure 5 also shows that this lead to an improvement of the control with less oscillation in the orientation angle at the simulation time of around 15s. Execution time was 9:48 min for 40s simulation time.

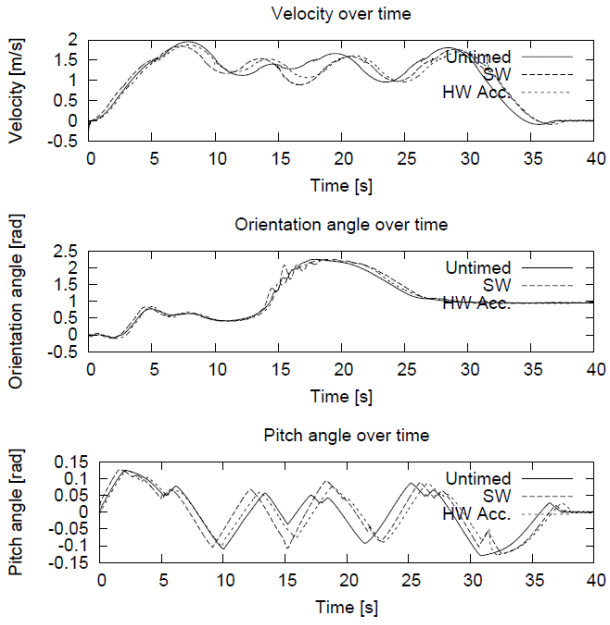

Figure 5. State variables for 40s of simulation

D. *Discussion of the Results*

In this case study, some interesting insights were made. It was possible to run the SystemC simulation in the GDB debugger. This could also be used to debug the embedded SW, because it was possible to trace any register value of the ISS and HW together with bus accesses. Additionally, the degree of detail of the modules could be easily configured. Physical effects, such as motor friction can be switched on or off together with effects of the embedded system such as cache access times or additional delays due to conflicts on the shared on-chip bus. This allows investigating the impact of these effects on the control separately or in combination to identify trouble makers. Also quick design space exploration was possible as was shown with the addition of the HW accelerator. A main advantage identified was the high simulation performance that allows re-evaluating the system quickly after a change or for different parameter values.

VI. CONCLUSION

In this paper, a case study for a SystemC virtual prototyping platform for a two-wheeled robot is shown. It integrates not only the model of the embedded real-time system, but also the physical model and the virtual environment. Therefore it offers transparent system behavior for SW development, verification of real-time constraints, investigation of implementation impact on control quality and fast design space exploration. In the case study, this approach has successfully simulated the tested real-time scenarios and aided in making design decisions.

ACKNOWLEDGMENT

This work is party sponsored by the German Federal Ministry of Science and Education (BMBF) in the project SANITAS (16 M 3088).

REFERENCES

[1] K. Pathak, J. Franch and S. Agrawal “Velocity and Position Control of a Wheeled Inverted Pendulum by Partial Feedback Linearization,” in IEEE Transactions on Robotics, Vol.21, No.3, June 2005.

[2] J. Bennett, “Building a Loosely Timed SoC Model with OSCI TLM 2.0: A Case Study Using an Open Source ISS and Linux 2.6 Kernel,” http://www.embecosm.com/appnotes/ean1, retrieved 09.03.2012.

[3] JMonkey OpenGL Gaming Engine: http://jmonkeyengine.com/

[4] V. Zivojnovic and H. Meyr, “Compiled HW/SW co-simulation,” in ACM/IEEE Design Automation Conference (DAC), 1996.

[5] W.Ecker,S.Heinen,andM.Velten,“Using a data flow abstracted virtual prototype for HdS-design,” in IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC), 2009, pp. 293–300.

[6] P. Gerin, X. Guerin, and F. Petrot, “Efcient Implementation of Native Software Simulation for MPSoC,” in Design, Automation and Test in Europe (DATE), 2008.

[7] J. Ceng, W. Sheng, J. Castrillon, A. Stulova, R. Leupers, G. Ascheid, and H. Meyr, “A High-Level Virtual Platform for Early MPSoC Software Development,” in International conference on Hardware/Software codesign and system synthesis, 2009.

[8] K. Popovici, F. Rousseau, A. Jerraya, and M. Wolf, “Embedded Software Design and Programming of Multiprocessor System-on-Chip: Simulink and SystemC Case Studies”, Springer, 2010.

[9] de Melo, L.F.; de Souza Cervantes, S.G.; de Franca, J.A.; Koyama, M.H.; , "Dynamics design and simulation for mobile robots navigation systems," in Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on , vol., no., pp.2133-2138, 22-25 Feb. 2009

[10] Osorio, F.; Wolf, D.; Castelo Branco, K.; Pessin, G., “Mobile Robots Design and Implementation: From Virtual Simulation to Real Robots,” in IDMME - Virtual Concept 2010