Developing and testing distributed CAN-based real-time control-systems using a single PC

Anders Moeller[*†] Per Aberg[•] Fredrik Loewenhielm[*] Jakob Brundin[*] Mikael Nolin[*†] Jakob Engblom[‡–]

[*] CC Systems, www.cc-systems.com

[•] ESAB, www.esab.com

[‡] Virtutech, www.virtutech.com

[†] Maelardalen Real-Time Research Centre, Maelardalen University, www.mrtc.mdh.se

[–] Information Technology Department, Uppsala University, www.it.uu.se

Abstract

Developing and testing of distributed embedded real-time control-systems is known to be very challenging due to the difficulties of debugging these systems in a target environment (e.g. due to weak monitoring capabilities and lack of powerful debugging tools).

The simulation technology described in this industrial experience paper is a toolbox aimed to improve the development and testing of distributed, CAN-based, embedded real-time control-systems. When using our technology, a complete control-system can be developed and tested without, or with only partial, access to target hardware. This is achieved by replacing target hardware dependent operations (e.g. device driver and operating system calls) with simulated equivalences that allow execution in a regular PC environment using regular PC programming tools. Thus, powerful PC tools for debugging, automated testing, fault injections, and dynamic modelling of the target machine, are made available for the embedded systems engineer. Complex dynamic behaviours can be studied in the simulated environment, without access to the target hardware, e.g. allowing single stepping through scenarios.

Simulating the complete system also facilitates customer tests and end-user evaluation of the system in an early phase of system development. It also shortens the turn around time for change, test, and evaluation, because development can be performed on a single PC instead of a full target system.

1 Introduction

Development and testing of software for distributed embedded real-time control systems is an area with great potential for improvements in terms of efficiency, quality, and time-to-market. One reason for this is that debugging of embedded systems in their target environment is challenging, e.g., due to the lack of powerful debugging tools and the difficulties of monitoring the internal software behaviour. Another reason is that access to target hardware is often a limiting factor in many software development projects – causing unnecessary delays.

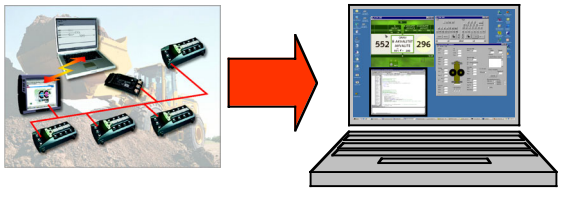

Figure 1: *CCSimTech - a toolbox that improves embedded control system development and application engineering.*

The simulation technology, CCSimTech, described in this paper is a toolbox that can be used to improve and simplify development of software for embedded systems. This is facilitated by replacing all target hardware dependencies in the software

with simulated equivalences - making a complete distributed embedded system executable and testable in a single PC (figure 1).

By using CCSimTech, software can be developed without, or with only partial, access to target hardware. In addition, powerful PC tools for debugging, automated testing, and fault injection can be used to guarantee the functional behaviour of the software. This, in turn, simplifies the introduction of target hardware, since the correct functional behaviour of the software is already validated.

2 Embedded System Setting

In order to describe the context in which the simulation technology is used, we outline some common and typical solutions and principles used in the design of the distributed embedded real-time control systems.

The system architecture can be described as a set of computer nodes called Electronic Control Units (ECUs). These nodes are distributed throughout the system to reduce cabling, and to provide local control over sensors and actuators (typical systems are vehicles, mobile machines and industrial automation). The nodes are interconnected by one or more communication bus forming the network architecture of the distributed system. When several different organisations are developing ECUs, the bus often acts as the interface between nodes. The communication buses are typically low cost and low bandwidth, such as the Controller Area Network (CAN) [2] or the Local Interconnect Network (LIN) [1].

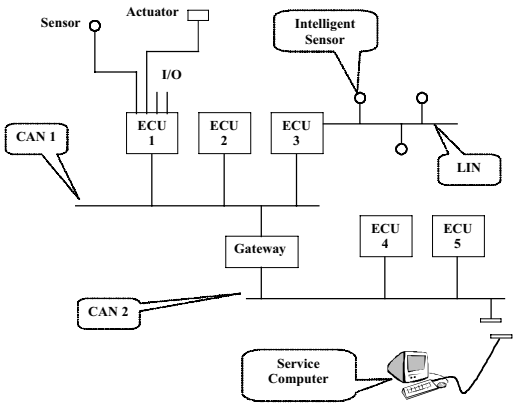

Figure 2: *Example of embedded system network architecture.*

In the example shown in figure 2, the two CAN buses are separated using a gateway. This is an architectural pattern that can be used for several reasons, e.g., separation of criticality, increased total communication bandwidth, increased fault tolerance, compatibility with standard protocols [3], etc. Also, safety critical functions may require a high level of verification, which is usually very costly. Thus, non-safety related functions might be separated to reduce cost and effort of verification. Services provided by the network could also include synchronisation and provide fault tolerance mechanisms.

Sophisticated run-time debugging environments are typically infeasible in the real target environment, as the nodes are too resource constrained to provide good support for debugging.

3 Motivation

Traditionally, embedded system software is tested using target hardware in a test-bed laboratory. The system is debugged using trace print-outs and the IO values are checked using oscilloscopes or even a multimeter (see figure 3). These tests are often labour intensive, time-consuming, inaccurate, and demands full access to all target hardware.

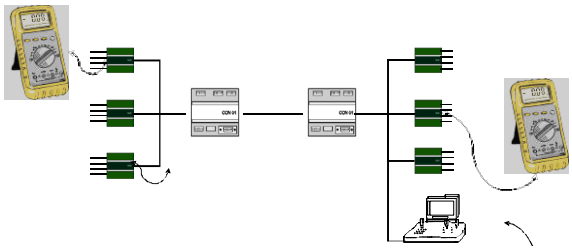

Figure 3: *Traditional embedded system testing*

There are several means by which the simulation technology improves the system development process, but the primary objective is to shorten the turn around time for change, test, and evaluation in system development.

Also, hardware and software can be developed in parallel. Hence, the software development can be started before the hardware even exists [4][6][7][8]. Moreover, software development becomes more efficient since every developer does not need access to (the often) limited hardware units or target machines, as the complete distrib-

uted system can be executed on a single PC.

Furthermore, a simulation-centric design process [4] enables development of hardware-independent software, since it encourages the use of platform independent software (e.g. layered software with replaceable device drivers). This makes the software developed easier to reuse and decreases the cost of future systems.

Hardware integration problems can be tackled early, or in some situations even be avoided, because functionality tests of software components can be done while simulating the system.

When testing the system using the simulation technique, complicated dynamic behaviours can be studied in a slowed-down environment, and slow behaviours can be speeded up. Moreover, some test sequences are hard, or even impossible, to test using the target machines (e.g. overheating or collision detection). Using simulation, such dangerous or difficult test cases can be explored.

In addition to the use in software development, simulation can be applied in other areas and activities, like, e.g., (*i*) customer tests and end-user evaluations, (*ii*) integration in commercial products/systems, e,g, in diagnostics applications and aftermarket tools, (*iii*) inexpensive training systems and simulators can be built based on the simulation technology, and (*iv*) marketing and sales support tools, e.g., presentations of a product/system.

4 Simulation and Test Environments

Using CCSimTech, three conceptually different types of simulation environments (complete system simulation, mixed simulation, and target hardware tests) become available to develop and test embedded control systems. Using these three test environments, combined with automated test-tools and an environment model of the system to control, we get a complete test solution for embedded systems and give rise to a simulation-centric development process. This approach has proven successful in many different software projects, and has been used by, e.g., CC Systems for over ten years.

4.1 Complete System Simulation

In a complete system simulation, all nodes, control panels, and environment models are simulated in a single PC. This is the most commonly used test technique by CCSimTech users, since no target hardware is needed. Figure 4 presents a complete system simulation.

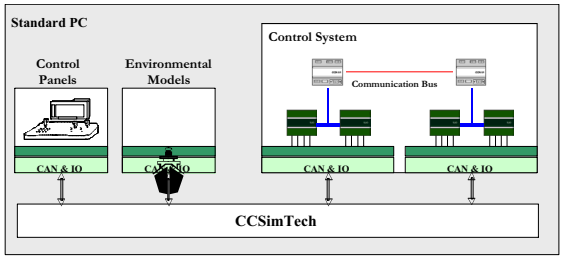

Figure 4: *An embedded control system simulated in a single PC*

The nodes communicate using a simulated field bus, e.g., CAN. IO-signals between the nodes and the external devices (sensors, actuators etc) are simulated using IO-channels in the PC. The necessary internal target hardware devices of the nodes (EEPROM, Flash, PLD, Power monitor etc.) are also simulated. These simulation devices (e.g. CAN, LIN, Flash Memories, RS232, etc.) are implemented as reusable software components. These components are at the heart of the simulation technique and are implemented to support both internal communication between the simulated hardware nodes and communication with different devices in the PC. The technical issues of CCSimTech are further described in Section 5.

To make the simulation-centric development process more complete, we have integrated support for communication with 3rd party tools. These tools can, e.g., be used to build environment models or control panel simulations. Also, automated test tools, both for unit testing and for system tests, can be used together with CCSimTech.

The environment models enable simulation of, e.g., mechanical, thermal, and electrical devices, and these models are connected to the simulated hardware nodes using simulated IO-ports facilitating a complete system simulation.

A system panel is typically used to control the start and stop of the simulated nodes.

4.2 Mixed Simulation

In a mixed simulation setting, some target hardware nodes are used together with the simulated system, see figure 5. This type of simulation is mainly used for two purposes; firstly, it enables integration of target hardware and hardware related software in a controlled step-by-step procedure, secondly it enables test of the physical communication between nodes. Furthermore, testing of IO-ports is possible.

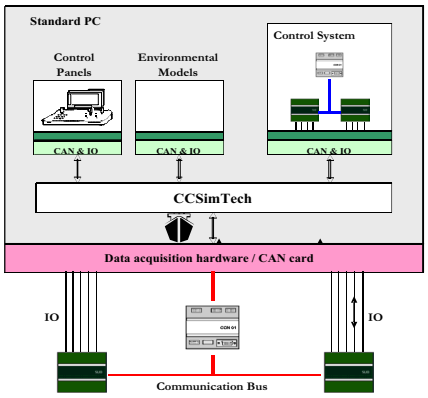

Figure 5: *An overview of a mixed simulation setting*

The environmental models and the control panels are the same as for complete system simulation. Also, if automated test-tools are used, the same test scripts can be used regardless of whether complete system simulation or a mixed setting is used. Hence, hardware-in-the-loop tests are facilitated using the described approach for simulating embedded systems.

Real target nodes can be connected to the other nodes via, e.g., a PC CAN-card. A simulated CAN-bus then connects to the simulated nodes in the PC. Furthermore, IO-ports on the external nodes are connected to data acquisition hardware in the PC (e.g. National Instruments PCI eXtensions for Instrumentation (PXI)) and connected to the simulated nodes and environmental models.

4.3 Target Hardware Tests

During the final phase of the simulation-centric development process, all target hardware nodes are used. However, the control panel components can still be simulated on the PC and connected to the external nodes through data acquisition hardware in the PC.

To facilitate hardware-in-the-loop testing, the target system hardware communicates with the environmental model.

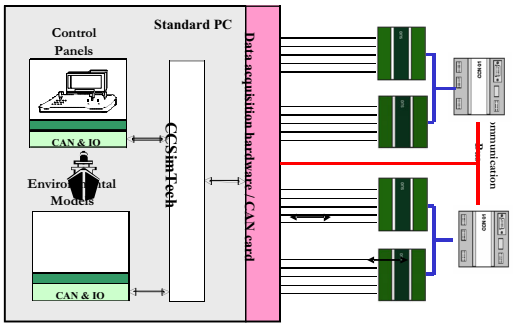

Figure 6: *An overview of a setting for target hardware tests*

The environmental models and control panel simulations as used in fully simulated tests are reused for target system simulation.

5 CCSimTech

As described, CCSimTech enables control-system software to be developed and tested without access to target hardware. This is achieved by replacing all target hardware dependent operations (e.g. device driver and real-time operating system calls) with simulated equivalences. These equivalences are the core of CCSimTech. They are implemented as a set of software components (figure 7), simulating the behaviour of, e.g., CAN, LIN, IO, RS232, EEPROM, and Flash.

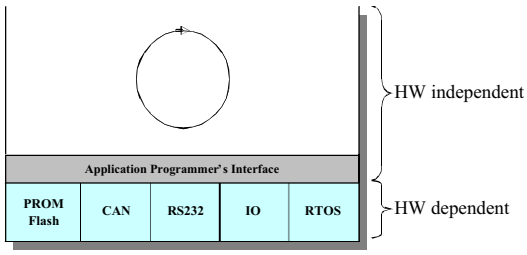

Figure 7: *Simulated system software overview*

It is important to emphasize that the software executed on top of CCSimTech (i.e. the HW independent application in figure 7) uses the same source code as the target hardware system does, except that the device drivers and the operating system

are replaced. The application programmer's interface, used to communicate with the environment, is the exact same regardless of whether the control-system is simulated or running on target hardware. This means that the target source code can be tested and debugged in a very efficient way, compared to black-box testing on hardware. Furthermore, powerful tools for debugging, automated testing, fault injection, and dynamic modelling of environmental models of the target machine, available for PC's, can be used to guarantee the functional behaviour of the software.

5.1 CCSimTech Limitations

The technology does not cover all aspects of system development, and some issues must be specifically addressed:

- ✓ Hardware device drivers must still be developed and tested in the target hardware.
- ✓ Timing properties are different in the simulated environment, compared to the target environment (further discussed in section 5.4).
- ✓ Operating system properties like process management and resource handling between processes must be considered and properly handled in the simulation (e.g. stack sizes, mutex- and semaphore handling).

5.2 Structure and Components

Software for embedded systems is often divided into a hardware independent part and a hardware dependent part (figure 7). Using CCSimTech, the hardware independent part is compiled under Windows together with simulated equivalences (implemented as reusable software components) of the hardware dependent parts (e.g. device drivers). These simulated equivalences are the core of the simulation technology and can be categorised into two different groups.

The first group simulates actual hardware components (using a device driver API) and includes components used for simulating; (i) buses (e.g. CAN, LIN, JI708, and RS485), (ii) IO (e.g. Analogue, Digital, PWM, PULSE), (iii) Serial communication (e.g. RS232), and (iv) components simulating memory devices (e.g., EEPROM and Flash). The second group of software components is used for simulation of the features of operating systems (e.g. OSE[1], RTXC[2] and RUBUS[3]) and time synchronisation properties (further described in section 5.4).

The same simulated components are used in all projects, and thus the effort to put up a new simulated environment is small.

5.3 Integration with COTS tools

The simulation technique can be interfaced from 3rd party software tools that can be used for, e.g., complex environmental modelling and creation of realistic user interfaces. In order to integrate these tools with CCSimTech, the software tool has to be able to interface the simulated communication.

Modelling of the system that is to be simulated is an important aspect of the simulation, since the better the model reflect the real system, the better simulation substitute testing on the real system. There exist several graphical tools for designing the models of the system behavior, e.g., MATLAB/Simulink[4] or LabVIEW[5]. In such modeling tools the conventional programming in e.g., C/C++ is replaced, or supplemented, by a graphical interface from which complex models can be created using libraries of predefined and user-defined functions and modules.

Using a graphical modeling tool for modeling has several advantages over conventional programming. Firstly, the graphical environment uses a view, similar to the logical architecture of the real system. Secondly, model components can be easily reused, which reduces the time to create new models. Third, the predefined component libraries enable creation of complex systems with a smaller effort.

In LabVIEW the simulated communication can be interfaced using specific model components that execute C-code during execution. The C-code is used for sending and receiving information through the C-API's. Connectivity between the simulation

[1] OSE; www.ose.com

[2] Quadros Systems; www.quadros.com

[3] Arcticus Systems; www.arcticus.se

[4] MathWorks; www.mathworks.com

[5] National Instruments; ni.com

technique and models created in other modeling tools, e.g., MATLAB/Simulink and SystemBuild/MATRIXx can be achieved using a similar approach.

Using GUI tools like Macromedia Director[6] realistic user interfaces can be created and connected to the simulation technique. This can be used for the marketing purposes or for facilitate cheap training systems.

5.4 Time Accurate Simulation

Distributed control systems are often timing critical, meaning that the behaviour of the control system (and the controlled system) is dependent on the time when actions are taken. Hence, when simulating a distributed control system it is often important to mimic also to timing behaviour of the real system (i.e. without a representative timing behaviour the functional behaviour will not be representative).

To capture and simulate the exactly correct timing of a distributed system is practically impossible. Instead one has to make do with a "good enough" modelling of the timing behaviour of the real system. Then, what is good enough? In CCSimTech timed events (such as delays and alarms) are triggered at the correct simulated time. To achieve this, we *synchronize* the simulated nodes based on the amount of target time they have accumulated [5]. To support this synchronisation, *breakpoints* are inserted into code running on the PC. The time it would take *the target system* to execute the code between breakpoints is stored in the breakpoints. Each simulated node has its own local time counter which is incremented when the code reaches a breakpoint.

However, in CCSimTech the execution time of application code is not modelled between these breakpoints - instead the application is executed in the speed provided by the PC. For multitasking systems this means that the simulation may not reproduce the exact same task switch and interleaving patterns as the real-system. However, the time accuracy provided by CCSimTech is enough to detect most timing related problems (such as errors caused by timeouts and watchdogs) and to allow mixes of simulated and real nodes to cooperate.

Certain low-level timing problems may be difficult to detect in CCSimTech:

- ✓ Delays caused by overload in nodes.
- ✓ Race-conditions that occur when multiple tasks simultaneously tries to perform the same action (such as locking a shared resource).
- ✓ Errors that only occur during special preemption patterns (e.g. due to failure of protecting shared resources).
- ✓ Errors that only occur when unsynchronised tasks sends messages in a special order.
- ✓ Bus overload conditions, as we do not model the amount of traffic on the CAN bus.
- ✓ Timing problems caused by messages being resent on the CAN bus because of electrical interference or other factors causing communications problems.

While representing an important class of problems, that are often difficult to detect and solve, the simulation accuracy needed to address them is beyond the scope of CCSimTech. To address these problems much more heavy-weight solutions such as clock-cycle accurate instruction level simulators *(*like the ARMulator from ARM [10]) or sophisticated systems for event recording needs to be used [9].

The simulated time in CCSimTech can be set to progress in real time. In this case the simulated system will (on average) execute at the same pace as would the real system. This mode makes it possible to run a mix of real nodes and simulated nodes together. However, the progress of simulated time can be slowed down, speeded up (the maximum speed is limited only by the speed of the PC), or even completely stopped. The slowing down or stopping of the simulated time is very useful when debugging. In this case an erroneous course of events could be slowed down, or stepped though, and all intermediate states could be analysed in a debugging tool. Speeding up simulated time is very useful when performing automated test sequences. In this case a tedious scenario could be simulated in a short time, e.g. facilitating efficient regression testing.

[6] Macromedia, www.macromedia.com

6 Application - ESAB

ESAB is one of the world's largest producers of welding consumables and equipment, with customers in, e.g., transport and off-road vehicle industry, the offshore and shipbuilding industry, power, process and construction industries (figure 8 shows a typical ESAB welding system).

ESAB have used traditional methods to test and debug their control-system functionality. The functional behaviour of the software was tested by measuring the IO-values on the electronic control units (ECU's) – a time-consuming and complicated method. Increased customer requirements on system functionality, leading to shorter release time-cycles, made it necessary for ESAB to look for new ways to improve efficiency and quality in software development and test. ESAB's choice was to introduce CCSimTech in the development process.

Figure 8: *Welding equipment with a CAN-based, distributed control-system.*

A schematic overview of a typical ESAB welding control-system, in its basic configuration, is illustrated in figure 9. This distributed, CAN-based, system has three ECU's; i) the weld data ECU, managing overall control of the welding process and HMI, ii) the power source ECU, controlling current and voltage in the power source, iii) the wire feed ECU, controlling the feed of welding wire.

When ESAB introduced CCSimTech, all target hardware dependencies in the control-system source code were identified and replaced with simulated system calls. After replacing these calls, the system became ready for execution in a PC.

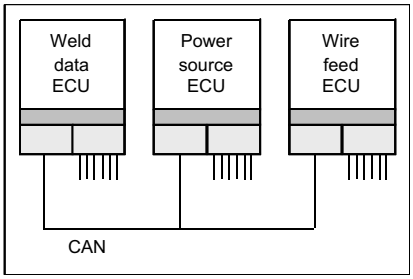

Figure 9: *ESAB welding equipment – system overview.*

The results for ESAB where that the product development cycle and time-to-market, were shortened. Also, the software quality was enhanced (fewer software bugs in the target system). Today, ESAB uses CCSimTech in more or less every new development project and has expressed a keen interest in using future improvements and additional components in order to further improve the software quality and decrease time-to-market.

7 Related Work

The work presented here is essentially an API-level simulation of a distributed embedded system. This solution is also known as a host-compiled solution, as the code is compiled for the host machine and not the target.

Real-time operating system companies like WindRiver [6] and Enea [7] make API-level simulators available for their operating systems. The solution presented here has stronger support for both networked and discrete I/O than their solutions, and is also independent of the operating system used.

Another level of simulation of distributed systems is to use instruction-set simulation to execute the actual target binaries. This gets a step closer to the real target. Regarding the use of an operating system in this context, it can either be emulated (operating system calls are executed directly on the host) or executed for real. The emulated solution allows the IO and networking to be simulated in a way similar to CC SimTech.

The operating system code can also be run on the simulator, generating a full-system simulator [8]. This makes it necessary to construct simulation models of the actual IO and networking hardware used, as well as the appropriate processors and

boards. The solution is thus less generic, but on the other hand, system behaviour very close to the real hardware can be obtained, including timing and bus contention.

8 Conclusions and Future Work

In this paper, we describe an approach to improve software development for embedded distributed real-time systems. We describe the technology, CCSimTech, and how it can be used to facilitate a simulation-centric software development process. The technique can be used during all phases of embedded system software development, all the way from a system simulated in a single PC using automated-test tools and environmental models to hardware integration tests. We also describe the technique as used by ESAB,

Our plans for future work include improvements of performance issues when simulating a large system (typically in the range of twenty – thirty simulated nodes). Also, we look at extending the technique with a number of new software components (e.g. other communication buses).

References

[1] LIN – Protocol, Development Tools, and Software for Local Interconnect Networks. In 9th International Conference on Electronic Systems for Vehicles, October 2000. Baden-Baden, Germany

[2] International Standards Organisation (ISO). Road Vehicles – Interchange of Digital Information – Controller Area Network (CAN) for High-Speed Communication, November 1993. vol ISO Standard 11898

[3] CiA. CANopen Communication Profile for Industrial Systems, Based on CAL, October 1996. CiA Draft Standard 301, rev 3.0, http://www.canopen.org

[4] S James. Simulation-centric processes for aerospace. Courtesy of Embedded Systems Programming, Embedded.org, http://www.embedded.com

[5] J. Engblom and M. Nilsson. Time Accurate Simulation – Making a PC behave like an 8-bit embedded CPU. Technical Report number 2002-024, Dept of Information Technology, Uppsala University, July 2002

[6] WindRiver VxSim data sheet, WindRiver Inc, 1999 (www.windriver.com)

[7] OSE Soft Kernel data sheet, Enea Embedded Technology, 2004 (www.ose.com)

[8] J. Engblom, Full System Simulation, Proceedings of the European Summer School on Embedded Systems (ESSES 2003), 2003.

[9] D. Sundmark, A. Möller and M. Nolin. Monitored Software Components – A Novel Software Engineering Approach. In Proceedings of the 11th Asian-Pacific Conference on Software Engineering, Workshop on Software Architectures and Component Technologies, Busan, Korea, November 2004

[10] ARM (Advanced Risc Machines) Ltd. WWW Homepage. www.arm.com.

Authors:

Anders Möller,
Fredrik Löwenhielm,
Jakob Brundin,
Mikael Nolin
CC Systems
Fyrisborgsg.5 754 50 Uppsala Sweden
Phone: +46 702 17 48 29
Fax: +46 18 12 38 85
{Anders.Moller; Fredrik.Lowenhielm;
Jakob.Brundin; Mikael.Nolin@cc-systems.se
www.cc-systems.com

Per Åberg
ESAB
695 81 LAXÅ Sweden
Phone: +46 584 84 100
Fax: +46 584 13 222
Per.Aberg@esab.com
www.esab.com

Jakob Engblom
Virtutech
Norrtullsg.15 113 27 Stockholm Sweden
Phone: +46 8 690 07 20
Fax: +46 8 690 07 29
jakob@virtutech.com
www.virtutech.com