1 Introduction

A critical need in developing embedded applications is to thoroughly test them for functionality and dependability before entering mass production. However, with modern complex applications the issue of cost becomes important as access to and changes in hardware are costly. In this article we describe how simulation as a means of testing can help reducing the effort associated with embedded application development without compromising the quality of the application.

This goal is achieved by using suitable tools and development process which allow us to write the simulation version of the application using target language and libraries, and reduce the need for hardware access during simulation testing. Most importantly, testing in simulation mode provides significantly improved flexibility, controllability and observability while the accuracy of results (as compared to testing on hardware) is not compromised.

2 Overview of TTP/C and C-Sim

This section provides a brief introduction to the TTP/C time-triggered protocol and C-Sim simulation tool. The time-triggered architecture (TTA) [6] was developed as a basis form building distributed systems for safety-critical real-time applications. A TTA application is deployed on nodes (containing a communication controller plus host computer) connected to a shared bus.

The nodes communicate, via the shared bus, using the TTP/C protocol [4] which is based on the TDMA (Time Division Multiple Access) access method. This guarantees a predictable time behavior of the nodes connected to the bus. The protocol meets the requirements for SAE class C applications (safety critical automotive applications). It has been designed to tolerate any single physical fault in any one of its constituent
parts (node, bus, etc.) without any impact on the communication activity of a properly configured cluster. To the application, TTP/C provides a global synchronized time and consistent membership service.

C-Sim is a C language library and accompanying toolset which allows the user to create his own pseudo-parallel processes and data elements with the ability to be queued. In its approach and features it is much inspired by the SIMULA language and libraries. It provides predefined functions and macros for process life manipulation, which allow to change the schedule and state of a living process using co-operative multitasking, and deriving user defined queue-able types similar to SIMULA’s LINK and HEAD. The simulation of the whole system is executed by invoking the predefined csim_step() function periodically, typically in the main() function, after at least one process is activated.

The main advantages of using C-Sim for simulation are the performance that can be achieved using the low-level C language, and the wide base of applications that were already written and can be used directly in the simulation. Thus transformation of the application from C-Sim version to the final one requires very little effort and does not change its synchronization and timing properties.

3 A C-Sim Model of a Simple TTP/C Application

A simulation model of TTP/C has been created within a project of the EC 5th Framework program, Fault Injection for Time Triggered Architecture (FIT). The aim was an experimental evaluation of the TTA specified FT properties using several forms of fault injection (FI), including simulation. The C-Sim of TTP/C has therefore been verified during the FIT project by numerous FI experiments.

Our test drive application for the TTP/C (sine wave measurement) comprised four nodes that perform the same functionality — measurement of an external signal source (generates the sine-wave signal). After measurement, the value is transformed to give range and precision and sent to other nodes. Three nodes of the application form a single fault-tolerant unit (FTU). The nodes are deterministic replicas and before committing the output, they perform a majority voting about their computed value. The fourth node provides no fault-tolerance property concerning permanent faults, but it should recover its output once a stream of transient faults has stopped.

The simulation version of the sine wave application uses the same base code as the production version. It comprises a C function measure_task() which defines the functionality a task periodically executed by the node. A main function that creates a working model of TTP/C network for the sine wave application first creates the network of nodes (controllers) on which the task is installed and their interconnections via channels. Then, it runs the simulation which interleaves the tasks on the nodes in model time.

The task and controller calls operating system functions; for application testing their implementation was replaced by equivalent simulation versions. Similarly, the testing was done first on a simulation implementation of the TTP/C protocol, which was developed as part of the project. This work helped to uncover several errors in the TTP/C specification and hardware implementation.
4 Application Development using Simulation

From our experiences from the development and use of the C-Sim model of TTP/C, we can outline the steps in safety-critical application development.

1. Create a (C-Sim) simulation model of the application environment (from both reusable components, e.g. models of car parts, and newly developed ones, namely modeling the physical characteristics of a concrete car model).

2. Obtain a (C-Sim) simulation model of the communication subsystem(s) in all variants (TTP/C v0.1 and v1.0, FlexRay).

3. Write the control applications (the C-Sim versions) for embedded computers in variants for all communication protocols.

4. Test and evaluate these applications (esp. with respect to functionality and dependability) in a simulation mode using the environment model, debug and redesign as necessary.

5. Select the optimal combination of application design and communication subsystem.

6. Test the (working) application on real communication hardware subsystem using simulation with the model environment, debug and redesign as necessary.

7. Using the physical prototype of the environment (e.g. car), test the non-simulation application version in a real setup, correct bugs as necessary.

8. Create the production version of the application.

Steps 4 and 5 are aimed at implementing correct synchronization and network communication of the applications [7] forming the cluster. Steps 6 and 7 then adjust this functioning application to the quirks introduced by real hardware.

This process has several advantages: (1) through simulation modeling [2] it reduces the need to create or use real hardware models (which are expensive) before the application functionality is stable, (2) it eliminates the risk of losing expensive hardware in case of serious failure, and (3) it supports any combination of application/hardware and simulation/production version in the testing process.

5 Lessons learned

Perhaps the practically most encouraging experience was the acknowledgement of the importance of our approach by a major vehicle manufacturer — the Volvo Technology Corporation. We are thus confident that it is suitable for production application development.

Before the application can be implemented, the developers need access to the specifications and simulation models of all components external to the application. Developing such models has the added benefit of verifying the respective specifications or physical prototypes.

Last but not least, simulation testing would much benefit from suitable tools. The fault injection testing done as the key part of our work required us to write all the fault injection and test generation code by hand. Much better productivity would be obtained if these activities could be automated. Currently, we have one such tool development as a work in progress.
Acknowledgement

This research was supported by TBD.

References


