CLEVER: Cross-Layer Error Verification, Evaluation and Reporting

Rafael Kioji Vivas Maeda
School of Electrical Engineering
Federal University of Minas Gerais,
MG, Brazil

rafaelkioji@gmail.com

Frank Sill Torres
Graduate Program in Electrical Engineering
Federal University of Minas Gerais
Av. Antônio Carlos 6627, 31270-901, Belo Horizonte, MG, Brazil

franksill@ufmg.br

ABSTRACT

The continuously progressing technology scaling is not only the source for increasing performance and complexity of embedded systems, but also the reason for rising reliability concerns of the underlying hardware. Consequently, numerous techniques emerged in recent years to mitigate wide range of error causes. Single layer methods, though, proved to be limited in providing an effective and efficient way to address errors in the system perspective. This paper proposes the new technology Cross-Layer Error Verification, Evaluation and Reporting (CLEVER), which is capable of gathering errors and system information on different levels of abstraction to evaluate the remaining useful lifetime of the system and its current state. Thus, CLEVER permits to overcome errors due to time-dependable and abrupt deterioration effects in complex embedded systems with a reasonable overhead. The presented system architecture was designed using a system description language and integrated in a simulation platform. Simulation results indicate the feasibility of this technology.

Categories and Subject Descriptors

B.8.1 [**Reliability, Testing, and Fault-Tolerance**]: Management

General Terms

Reliability

Keywords

Dependable embedded systems, cross-layer, management

SBCCI '14 September 01 - 05 2014, Aracaju, Brazil

ACM 978-1-4503-3156-2/14/09 ...$15.00.
http://dx.doi.org/10.1145/2660540.2660978

1. INTRODUCTION

Current embedded systems play an important role on humans daily life. Today's embedded systems are composed by a quite diverse of devices ranging from discrete components to Multiple Processors System-on-Chips (MPSoCs). Though technology advances provide integrated circuits with continuously decreasing feature size and increased complexity, errors caused by a broad range of sources rise as well. This amount creates a critical scenario around the overall reliability of the system. Many efforts have been made to address all kinds of errors, like approaches to detect and recover from radiation induced errors [8], process variability problems [10], aging [13] and many more. However the vast majority of solutions are focused on a single abstract layer and do not provide an effective and efficient way to handle errors in a system perspective, as already identified by [4].

This paper presents the new technology Cross-Layer Error Verification, Evaluation and Reporting (CLEVER), which is capable of gathering errors and system information on different levels of abstractions to evaluate the remaining useful lifetime of the system and its current state. It provides a method detached from the main target system processor to reduce the overall overhead. Also using cross-layer evaluation results in a cost-effective solution for systems with high complexity.

A similar approach for gathering the system's reliability is the Self-Monitoring Analysis and Reporting Technology (SMART). It is a standardized specification adopted by the Hard-Disk (HD) driver industry which defines a bunch of reliability attributes to be monitored aggregating the main characteristics of HDs. Using the collected information, SMART is able to warn when the HD is about to fail enabling some backup strategy before it happens. However, SMART is a HD specific technology and is not suitable for complex embedded systems. Another similar work is [6], which presents a framework of a monitoring scheme of attributes with reduced overhead for reliable embedded systems. The latter is limited to monitoring and doesn't include any reliability management strategy.

The rest of the work is structured as follows. Section 2 presents preliminary concepts as the base knowledge for this article. Section 3 discusses the main concept of the CLEVER technology. Section 4 and section 5 presents the proposed architecture of the CLEVER technology and the results of the architecture achieved so far, respectively. Fi-

nally, section 6 concludes this paper.

2. PRELIMINARIES

This section covers the basic concepts and underlying assumptions used in this work.

2.1 Terminology

There is typically three different concepts used when dealing with threats for reliability: failure, error and faults. When an incorrect service is delivered, it is said that a failure occurred. Failures occur when errors reach the service interface and alter the interface. A fault is the adjudged or hypothesized cause of an error [**?**]. Hence, error verification stands for the continuously monitoring of system state correctness.

2.2 Metrics

The reliability of a system can be quantified as Mean Time To Failure (MTTF), which is the average time that a system runs until it fails. Furthermore, the failure rate λ expresses the probability that a system fails in a given time interval. The failure rate is a time-dependable value and usually modeled by the bathtub curve or Weibull Analysis, which divide the failure rate of a system in the three phases: infant mortality, useful life, and wear-out. The phase infant mortality is a relatively short period that is characterized by an initially high failure rate which is rapidly decreasing. The majority of failures in this phase are due to manufacturing defects and assembly errors. The following useful life is a long period with a low and near constant failure rate. During the final phase wear-out, the failure rate increases again based on the degrading effects of the material and accumulated damages of the system.

The Remaining Useful Lifetime (t_{RUL}) defines the expected time until the system fails and is an estimated during runtime of the system [1]. Usually, the effects of a recalculated t_{RUL} are notable in the wear-out period of the failure rate curve. Figure 1 depicts an example comparing a actualized Remaining Useful Lifetime $t_{RUL_{new}}$ after the old failure rate $\lambda_{old}(t)$ had been corrected to the new function $\lambda_{new}(t)$ during runtime of the system [11]. The value λ_{acc} determines the specified maximum acceptable failure rate of the system.

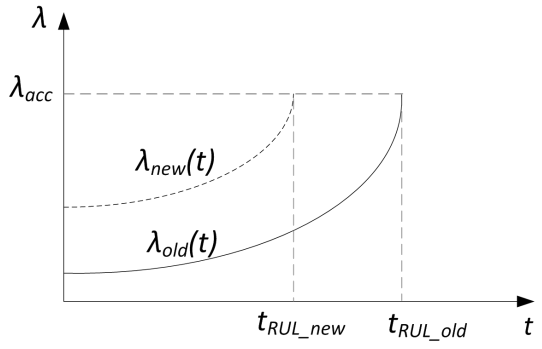

Figure 1: Figure : Adaptation of failure rate $\lambda(t)$ and Remaining Useful Lifetime t_{RUL} [11]

2.3 Prognostics and Health Management

Prognostics and Health Management (PHM) is a method that concentrates on determination and extension of the remaining useful lifetime of a system and its components. Thereby, data extracted by monitoring at runtime are applied to analyze the current state of the system and predict its t_{RUL}. Figure 2 illustrates the advantages of this approach compared to designs that solely rely on common statistically reliability analysis [9]. The graph compares the statistically expected lifetime with the time to failure when a system is under severe and mild usage. As it can be seen, prognostics can increase the reliability or increase the revenue of a system by readjusting its expected end of life. Further, due to the knowledge of the t_{RUL} measure can be taken to increase the lifetime, e.g. by exchanging components or readjusting the system load.

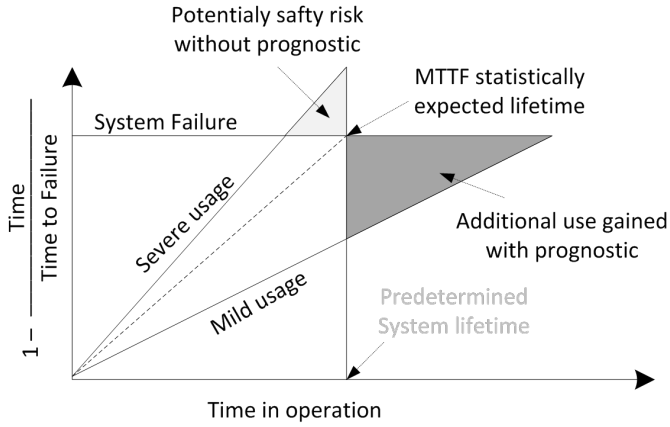

Figure 2: Figure : Comparison of time to failure of a system for different scenarios [9]

Prognostics can be data-driven, model-based, or a combination of both [12]. Thereby, data-driven prognostics relate previously determined data of reference implementation of the systems with current data, applying machine learning techniques or pattern recognition. In contrast, model-based or physics-of-failure approaches calculate the cumulative damage due to various failure mechanisms based on the knowledge about the physics of the degradation of individual components [2]. Hybrid approaches combine both strategies, whereas data-driven approaches are used for calibration of the physical models while model-based techniques define failure criteria and thresholds for the data-driven methods [14].

2.4 Modeling Embedded Systems

Due to the increasing complexity of embedded systems, design methods based on higher abstraction layers have a gaining significance. The principal idea is the maximization of automatized development steps following deterministic rules to increase the productivity [7]. A widely used layer model for embedded system is the separation in following levels: system, processor, logic, and circuit [3]. The system level is the highest abstraction layer and focuses on the architecture of the embedded system. The following processor level is dedicated to the processing elements of the system, while the logic level represents the logic blocks and its in-

teraction. The circuit level is the lowest abstraction layer and includes the integrated elements like standard cells and electronic components.

Each layer requires Models of Computation (MoC) which describe the mechanism to perform the computations [5]. Typical examples of MoCs are shared memory, Petri-Nets, and discrete events. Furthermore, several languages exist for the description of MoCs, whereas SystemC is widely used for discrete event based MoCs and modeling of communication mechanisms [7]. The latter usually is based on Transaction-level modeling (TLM), which is a high-level approach for modeling communication among modules.

3. CLEVER

This section presents the concept of the new approach and its principal architecture.

3.1 Overview

Prognostics and Health Management (PHM) of embedded systems is of increasing importance due to diminishing technology sizes, rising number of error sources, and increasing system complexity. Rising complexity is additionally the motivation for solutions that cover several abstraction layers. Beyond that, applicable techniques require straightforward system integration, and tolerable performance penalties.

The proposed Cross-Layer Error Verification, Evaluation and Reporting (CLEVER) is a technology that faces these requirements by combining monitoring module, prognostic computation, and communication interfaces. The characteristics of each element are as follows:

- The monitoring module consists of a collection of system specific sensors that gather characteristic data on several abstraction-layers. This includes physical values like temperatures and voltages, error values like events of overvoltage or radiation based transient faults [8], and communication failures. It should be noted the type of sensors depends on the target application.
- The prognostic computation is executed on a proprietary processing unit, in the following named as the CLEVER-PU, and processes the gathered information. Thereby, hybrid prognostics shall be applied to determine the remaining useful lifetime of the system and its components.
- The communication interfaces manage the communication between the sensors and the CLEVER-PU as well as the communication between the CLEVER-PU and higher instances, like the processing unit of the embedded system. This includes the definition of new communication protocols as well as the integration in existing protocols.

Further, CLEVER is designed as universal approach for embedded system, i.e. it is not limited to a specific problem or application. Consequently, the decision of applied sensors and the integration of the CLEVER has to be made at design time of the system.

3.2 Principal Architecture

Figure 3 depicts the principal architecture of the technology CLEVER. The processing unit CLEVERâĂŚ-PU, integrated in a microprocessor or FPGA, is supported by volatile (RAM) and non-volatile memory (NVM). Further, each component of the target embedded system is connected with one or several sensors that monitor physical data, error values, and/or communication failures. The sensors communicate via a proprietary protocol with the arbiter (omitted in this figure) inside the CLEVER-PU.

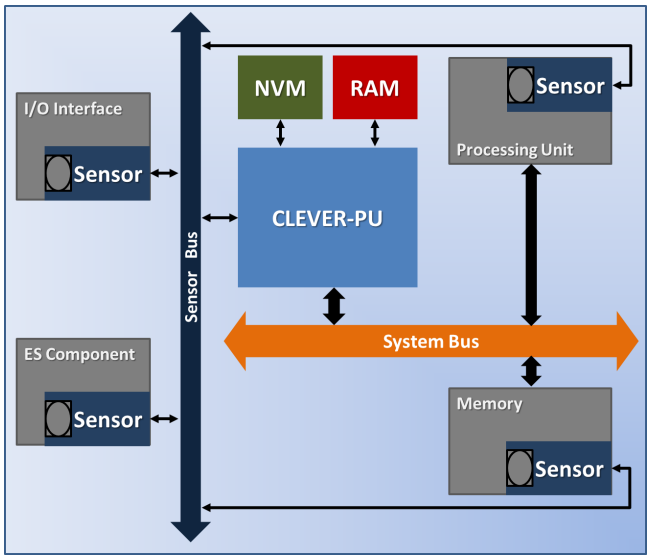

Figure 3: Figure : Principal Architecture of CLEVER

Figure 4 shows the principal Hardware/Software (HW/SW) interface of CLEVER. The CLEVER-PU executes the prognostics applying the monitored data and communicates via the system bus with the processing unit of the targeted embedded system. Further, a system application will be able to read-out data of the CLEVER-PU and to be informed if the system enters in critical states. Consequently, the actual performance of the embedded system is nearly unaffected by the new technology as most of the processing is executed on the CLEVER-PU.

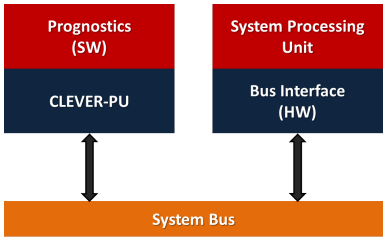

Figure 4: Figure : Principal HW/SW interface with Embedded System

3.3 Scope

The scope of this work is the presentation of the architecture as well as the communication interfaces of the new technology CLEVER. A System Description Language is used to

provide means for architecture exploration, functional evaluation, and early performance analysis.

4. SYSTEM DESCRIPTION

This section details the the hardware as well as the software architecture of the proposed technology. This includes the specifications at behavioral and transaction level, an overview of the created SystemC model, and tools oriented for debugging purposes.

4.1 Design Strategy

Hereby, a top-down strategy has been chosen with the specification of the main functional and non-functional characteristics as initial step. This is followed by the hardware/software partitioning regarding the needs of the system elements. The next step is the design of the transaction level structures, which provides the principal data flow. Concluding, hardware and software could be refined using a concurrent approach.

4.2 Hardware architecture

The proposed hardware architecture uses proprietary components to provide a flexible model and enable different implementations. Figure 3 shows the main components of CLEVER. The basic classes of components are the sensors, memories, the CLEVER-PU, and the system bus.

The collection of sensors used to monitor features and errors of the system are connected to the CLEVER-PU via an internal arbiter entity. A priority scheme may be adopted accordingly to the target system needs. Using a priority scheme, critical features being monitored may have higher priority.

The sensors are divided in two parts: the communication component and the sensing device itself. The communication part must be compliant with CLEVER protocols, while the sensing device is designed specifically for the monitored properties. Figure 5 depicts the logic partitioning of the sensor.

Besides the communication and sensing part, a buffering technique is required to increase the communication performance of the sensors. The buffer stores the most recent information produced by the sensing element until enough data has been collected for a efficient transmission. Further, since the sensing device and the communication element may have different clock frequencies the buffer is also used to synchronize data crossing the two clock domains.

Memories, located at the top of figure 3, are used to store volatile and permanent data. The volatile memory, named as *RAM* (Random Access Memory), is used to store processing data needed by the evaluation algorithms while the non-volatile memory, named as *NVM*, is used as the main storage component enabling the creation of a database for sensors statistics.

Both, volatile and non-volatile memory, may be employed as on or off-chip memories. A tradeoff exists between these two technologies regarding routing effort on system level, memory size and cost. Selecting on-chip memory may decrease the effort needed to route the Printed Circuit Board (PCB) of the target system, however it will have less storage capacity at constant costs. Further, the selection of memory speed has a significant impact on CLEVER performance. Depending on memory technology speed chosen, the evaluation algorithm may take more time to process data.

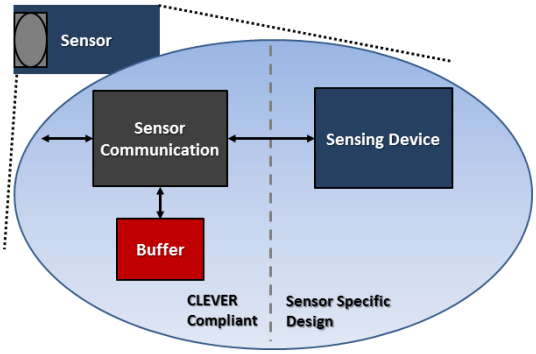

Figure 5: Logic partitioning of sensor. Sensors are composed by a communication part and the sensing device itself.

All components transactions are coordinated by the CLEVER-PU. This block is responsible for generating and attending all transactions requested by other elements. Also, this block has the purpose to execute the evaluation algorithms. Using an entity detached from the target system main processor reduces the total overhead added to the main system routines. This is a crucial characteristic for both soft and hard real-time embedded systems [7].

Figure 6 shows the block diagram of the CLEVER-PU. The proposed interconnection provides a multiplexing transaction which permits high performance. The RAM Transaction Logic block deals with transaction requests among storage elements, RAM, main target system processor and sensors communication. All operations within the non-volatile memory is coordinated by Storage Logic. Communications with the target system main processor is driven by the Driver Logic block. The Reliability R&A Logic is the module responsible for reporting and analysis of the system state. Finally, the Arbiter decides which sensor will have permission to transfer its data.

4.3 Software architecture

Besides the hardware definitions, some services must be provided at software level, e.g. the interaction between the main target system processor and CLEVER-PU, which requires a driver software to handle its communication. Also, some features are easier to implement in software rather than in hardware, e.g. graphical user interface for statistical reporting.

To fulfill these requirements the proposed CLEVER architecture defines an application running within the target system processor. Its main purpose is to handle all states needed to keep communication active and to enable the reliability reporting. This approach allows a low coupling between the system target applications and the underlying CLEVER driver by providing a high level interface using a communication method.

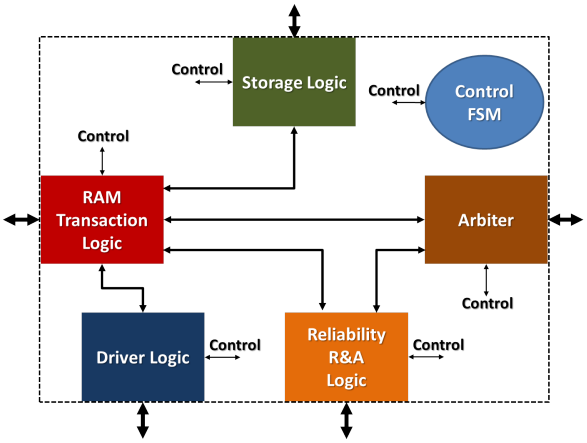

Figure 6: CLEVER-PU block diagram. Interconnection similar to a cross-bar to provide higher data throughput and multiplexing.

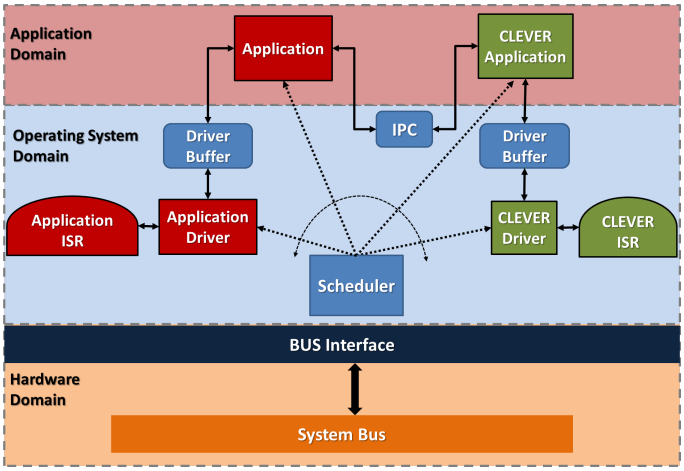

Figure 7: Software achitecture. Arrangement of clever software entites at the target processor

Figure 7 depicts the proposed method for the arrangement among CLEVER software entities and system target applications. The advantage of this architecture approach is the low coupling between CLEVER and the target system, as mentioned earlier. Any changes at CLEVER routines require the update of the communication protocols, achieving high flexibility for bug fixes and new releases. In some cases a lower overhead communication may be needed - like hard real-time embedded systems making use of CLEVER reporting feature. For such cases, an API (Application Programming Interface) may be designed to provide direct communication to the main CLEVER processor.

4.4 Transactions description

Transactions is defined as the exchange of information between devices. In order to enable communication, transactions must attend to a protocol specification. The main characteristics the protocol must define include:

- sequence of messages, describing in which order data and control packets are transmitted;
- contention detection scheme, since multiple devices may be connected to the same communication bus a method to detect whether the bus is idle or not is important to mitigate collisions;
- error detection, as collisions and other effects may cause errors at the communication channel, a technique like CRC (Cyclic Redundancy Check) may be used to detect errors;
- scheme for retransmissions, since the communication channel are prone to errors retransmission of lost or corrupted messages is essential.

Further. the communication component of the sensors has to support multiple sensors connected to the same bus. This reduces the number of the required I/Os, and permits a higher flexibility regarding the number of connected sensors.

Additionally, the sensor communication has to support polling and interruptions. Polling schemes are used by the CLEVER-PU to request the sensors current state while interrupt method are used by sensors to report the CLEVER-PU when the buffer is full, due to its limited size. Also, interrupts may be used to register the device to the list of sensors.

Finally, the interaction with protocols on higher level of abstraction has to be supported. Since multiple monitoring sensors are used within the embedded system and each sensor has its own characteristics like sampling rate, level of criticality, and reporting frequency needs, a efficient way to collect all relevant data from sensors is required.

4.5 Behavioral description

A transaction multiplexing scheme were used to pipeline memory access. Since different elements (arbiter, driver and storage components) need different access rate to the memory the multiplexing improves data throughput. The system described multiplex memory access among all active modules – driver logic, reliability reporting and analysis module and sensors – and also among passive modules – NVM and RAM.

4.6 TLM to MSC tool - tlm2msc

The proposed system was modeled using SystemC to extract results and verify the correctness of the described architecture. A SystemC executable model provides few different ways to monitor the system behavior. One approach is using tracing: generating the waveform of specified signals. Another approach may be through messages printed at the output (console). Also, the user may configure the default output of the executable model to be dumped into a file.

Using waveform to monitor its behavior is a more intuitive approach. However, as the number of the signals increases (for large communication bus, and dense synchronization signals between devices) this kind of analysis may becomes less efficient. On the other side, debugging through printed messages doesn't give an intuitive way to visualizes concurrent signals - since the messages are printed in a sequential

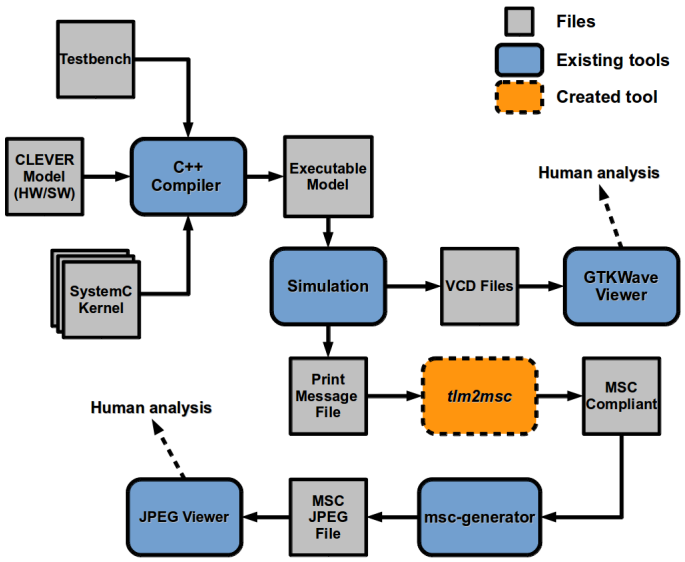

Figure 8: Simulation flow using SystemC and *tlm2msc*.

manner. However, printed messages may be used to give a more simplified version of transactions happening at the moment.

A combination of both, debugging through waveforms and printed messages may become more powerful. Hence the tool *tlm2msc* was created. It converts the printed message used to show transactions events to a message sequence chart which provides a better timing sense. Combining the waveform (with detailed bus signals) and the message chart (highlighting transactions) debugging becomes easier.

Figure 8 shows a diagram describing the simulation flow used to validate the SystemC model. Square blocks are files, continuous rectangle blocks are pre-existing tools and the dashed rectangle block is the created tool. The inputs of the flow are the SystemC Kernel libraries, the CLEVER model and the testbench modules. These inputs are compiled using a C++ compiler resulting in an executable model. The executable model may generate waveform signals (dumped into VCD files) and print messages saved in a file. Then, two ways may be used to ease analysis: visualizing VCD files using waveform viewer and inspecting the message sequence chart generated by the print messages in combination with *tlm2msc*.

5. RESULTS

A system consisting of the CLEVER architecture and a few number of sensors was modeled to evaluate the proposed model. This section presents the testbench architecture, simulations events and the main results extracted from the simulation platform.

Figure 9 shows the testbench architecture realized to verify the proposed approach. The dashed components are elements of the testbench. Several monitoring and stimulus signals were used to check the system's behavior. Communication channels were modeled using TLM.

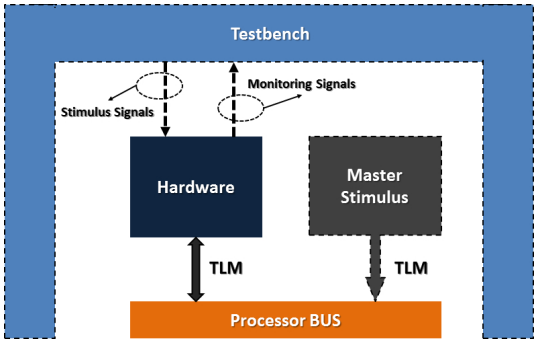

Figure 9: Testbench architecture model.

The testbench simulates the target embedded system by generating error events. Figure 10 shows a situation in which the sensor detects fault conditions and stores its data internally. Consequently, the arbiter unit of the CLEVER-PU polls the sensor and receives all fault events occured within a period of time. Also figure 10 depicts several control messages sent to sensor to start and finish communication.

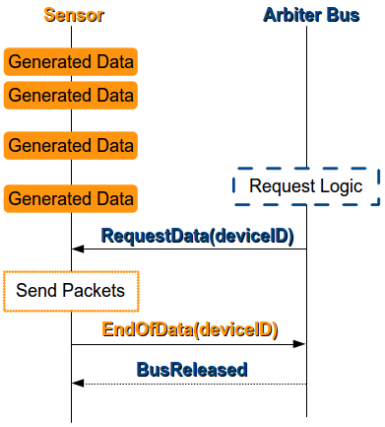

Figure 10: Message Sequence Chart for a few exception events detected by a sensor.

From this point on, data are forwarded from arbiter bus to the storage units. It should be noted that this simulation solely concentrates on the interaction between the system sensors and CLEVER. Hence, any prognostics and health management tasks are not included here. The proposed architecture was validated by simulation. In future works, PHM algorithms will be integrated in the developed environment, which will then serve for performance, communication, and overhead estimations.

6. CONCLUSION

This work proposes the novel technology CLEVER, which is able to gather errors and system information on different levels of abstractions enabling an overall cost-effective solution to evaluate the remaining useful lifetime and the current state of today's complex embedded systems. The system architecture was validated using system description language and simulations indicate its feasibility. Future works will concentrate on the analysis of the architecture using PHM on the simulation platform, which will serve for per-

formance, communication and overhead estimations. Further, the CLEVER will be applied to an embedded system for analysis on a real system.

Acknowledgment

This work has been supported by the Brazilian agencies CNPq, CAPES, FAPEMIG, and PRPq/UFMG.

7. REFERENCES

[1] D. Banjevica. Remaining useful life in theory and practice. *Metrika*, 69:337–349, 2009.

[2] J. L. et al. Model-based prognostic techniques. *IEEE Systems Readiness Technology Conference*, 2003.

[3] D. D. Gajski. System-level design methodology. *Computer Society Workshop on VLSI '98*, 1998.

[4] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, , and N. Wehn. Reliable on-chip systems in the nano-era: lessons learnt and future trends. *Proceedings of the 50th Annual Design Automation Conference (DAC '13), ACM, New York, NY, USA*, (99), 2013.

[5] R. Janka. Specification and design methodology for real-time embedded systems. *Kluwer Academic Publishers*, 2002.

[6] B. K. Kim, K. Lee, S. Y. O. nad, and S.-G. Kim. A monitoring composition approach for reliable embedded systems. *Advanced Science and Technology Letters - AITS, Hangzhou, China*, 2013.

[7] P. Marwedel. Embedded system design: Embedded systems foundations of cyber-physical systems. *Springer*, 2011.

[8] R. Possamai Bastos, F. Sill Torres, G. Di Natale, M. Flottes, and B. Rouzeyre. Novel transient-fault detection circuit featuring enhanced bulk built-in current sensor with low-power sleep-mode. *Microelectronics Reliability*, 2012.

[9] R. R., S. H., and C. J. Feasibility study of a rotorcraft health and usage monitoring system (hums): Results of operator's evaluation. *NASA CR-198446*, February 1996.

[10] B. F. Romanescu, M. E. Bauer, S. Ozev, and D. J. Sorin. Novel transient-fault detection circuit featuring enhanced bulk built-in current sensor with low-power sleep-mode. *Proceedings of the 5th conference on Computing frontiers (CF '08). ACM, New York, NY, USA*, pages 129–138, 2008.

[11] E. Suhir. Remaining useful lifetime (rul): Probabilistic predictive model. *International Journal of Prognostics and Health Management*, 2:5, 2011.

[12] G. Vachtsevanos, F. Lewis, M. Roemer, A. Hess, and B. Wu. Intelligent fault diagnosis and prognosis for engineering systems. *Wiley*, 2006.

[13] H. Zhang, L. Bauer, M. Kochte, E. Schneider, C. Braun, H.-J. Imhof, M.E.and Wunderlich, and J. Henkel. Module diversification: Fault tolerance and aging mitigation for runtime reconfigurable architectures. *Test Conference (ITC), 2013 IEEE International*, 1(10):6–13, Sept. 2013.

[14] H. Zhang, R. Kang, and M. Pecht. A hybrid prognostics and health management approach for condition-based maintenance. *IEEE International Conference on Industrial Engineering and Engineering Management*, pages 1165–1169, 2009.