Exploiting Non-intrusive Monitoring in Real-Time Embedded Operating Systems*

Ricardo C. Pinto, José Rufino
Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa
LaSIGE - Navigators Research Team
{ricardo.pinto, jmrufino}@ciencias.ulisboa.pt

ABSTRACT

Monitoring in embedded system software can have several uses, ranging from system characterization to run-time verification (RV). Traditional monitoring techniques require code instrumentation, imposing an overhead on system execution both in performance and timeliness. In real-time systems this is exarcebated by the need of new worst-case execution time estimation and schedulability analysis. In this paper we discuss how monitoring can be exploited in real-time embedded operating systems, via non-intrusive mechanisms.

General Terms

Run-time Verification, Real-time Embedded Operating Systems, Non-intrusive Monitoring

1. INTRODUCTION

Exploitation of Cyber-Physical Systems (CPSs) requires adaptive behaviour in the presence of faults. A contributing solution comes from Run-time Verification (RV), where the application could assess the CPS state and act accordingly, e.g. enable safe operation modes. The basis of RV consists in monitoring system state, which can be described through: values of variables; execution of functions and procedures; input/output activities. All these are materialized by the reading/writing of specific memory addresses, or ports.

Monitoring data collection usually requires software modifications (instrumentation). In the realm of real-time embedded operating systems, it poses obstacles: performance, due to resource scarcity; timeliness, due to the need of Worst-Case Execution Time (WCET) re-evaluation and schedulability analysis, possibly deeming the system as unschedulable. A non-intrusive approach to monitoring voids these shortcoming.

*This work was partially supported by the EC, through project IST-FP7-STREP-288195 (KARYON) and by FCT, through project PTDC/EEI-SCR/3200/2012 (READAPT), through LaSIGE Strategic Project PEst-OE/EEI/UI0408/2014, and Individual Doctoral Grant SFRH/BD/72005/2010.

EWiLi'14, November 2014, Lisbon, Portugal.

2. MONITORING ARCHITECTURE

A non-intrusive Observer Entity (OE) [6] is the crux of the monitoring system, targeting a System-on-a-Chip (SoC) architecture with a central bus for data exchange. The OE is embedded in a SoC system based on a SPARC LEON3 processor [1] and connected via the SoC Advanced Microcontroller Bus Architecture (AMBA) [2] (*see* Figure 1). The AMBA bus data is monitored and compared with a set of configured observation points. Upon detection an event is created, time-stamped and the pertaining information relayed to an external system, for storage and processing.

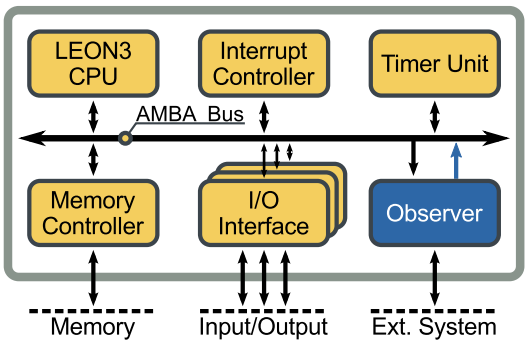

Figure 1: LEON3 System-on-a-Chip with Observer Entity

The OE architecture is composed by several functional blocks: **Bus Interface**, allowing the OE to peek the bus activity and access the configuration via the bus (fallback method); **Event Observer**, which compares the current bus operations against a set of configured observation points stored in the **Observer Configuration**; an **External I/O Interface**, to send the detected event data to an external system and receive the observation points to be configured.

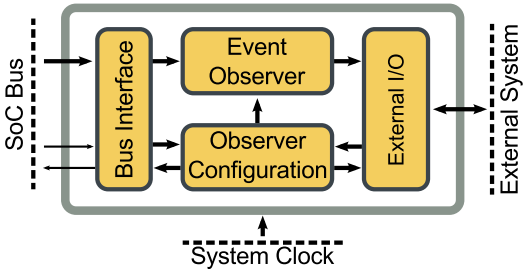

Figure 2: Observer Entity Architecture

3. OBSERVATION POINTS

Observation points for monitoring are extracted from a binary application file using a combination of off-the-shelf and specially-crafted tools. The extracted entities are called symbols, e.g. functions and variables, and are shown in Figure 3. The memory addresses (`Value`) corresponding to symbols (`Name`) are extracted.

```
Name          Value    Class    Type     Size     Section
f            |40001924|   T   |    FUNC|00000058| .text
main         |4000197c|   T   |    FUNC|0000002c| .text
atoi         |40002140|   T   |    FUNC|0000001c| .text
global_var   |4000606c|   D   |  OBJECT|00000004| .data
__bss_start  |40006bc0|   B   |  NOTYPE|        | .bss
```

Figure 3: Map of Symbols of a C Application

After symbols are extracted, they are selected and the corresponding memory address is used to create a configuration file for the OE. The flow is depicted in Figure 4.

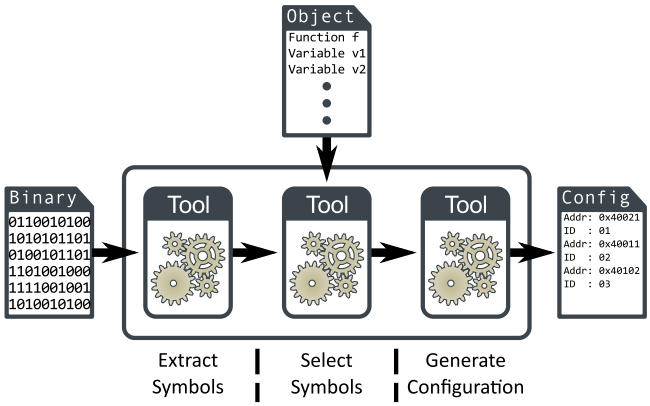

Figure 4: Observation Point Extraction Flow for Symbol-based Monitoring

The tools referred in Figure 4 are: GNU Binutils tools [3], such as `nm`, for symbol extraction; standard Unix tools, such as `grep`, for symbol selection and `awk` for formatting the selected symbols into a configuration file for the OE.

4. EXPLOITATION

Exploitation of monitoring data can be split in three timeliness profiling categories: **Scheduling Analysis** enabling the analysis of a given task set schedule as provided by analytic tools; **WCET Measurement**, assessing the correctness of WCET estimates obtained from external tools; **Run-time Verification**, to assess the correctness of execution at run-time, by the application.

4.1 Scheduling Analysis

Cheddar [7] is a tool providing schedulability analysis for a set of real-time tasks, outputting the resulting schedule. Monitoring data can be used to perform experimental evaluation of the analytic results provided by Cheddar. Experimental results can be filtered and fed to Grasp [5], a tool enabling visualization of real-time system execution, namely task execution and switching. Composing both tools enables a work flow for designing and validating task schedule.

4.2 WCET Measurement

The existence of data pertaining to timeliness profiling of the system can provide insight into task WCET characterization, and assist the activities in the design-cycle. WCET values provided by tools are pessimistic, and can be off the real task execution values by an order of magnitude [4]. Monitoring of task execution times allows to get a realistic magnitude of WCET value, which can then be fed into the schedulability analysis tools.

4.3 Run-time Verification

Non-intrusive monitoring data can be fed into verification techniques, in order to achieve RV. Given that CPSs need to interact with the physical environment, uncertain by nature, the usage of RV is a valuable asset to enable adaptive behaviour of embedded applications. For example, an unmanned aerial vehicle (*drone*) may require a level of thrust until the aircraft has reached a certain altitude. Both variables - thrust and altitude - can be monitored, and fed into a RV mechanism coupled with the fault detection, isolation and recovery mechanisms. Such an approach improves on the current state-of-the-art by its non-intrusiveness and flexibility.

5. CONCLUSIONS

The availability of a non-intrusive monitoring infrastructure can enrich an embedded application at all stages of its lifecycle, by allowing the collection of execution data from the system. Such data can be exploited to assess properties pertaining to system performance and timeliness; scheduling analysis and Worst-Case Execution Time (WCET) measurement; and serve as a supporting basis for Run-time Verification (RV) techniques.

6. REFERENCES

[1] Aeroflex Gaisler A.B. *GRLIB IP Library User's Manual*, Apr. 2014.

[2] ARM Limited. AMBA Specification, May 1999.

[3] Free Software Foundation. GNU Binutils.

[4] J. Garrido, J. Zamorano, and J. A. de la Puente. Static Analysis of WCET in a Satellite Software Subsystem. In *OASIcs-OpenAccess Series in Informatics*, volume 30. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[5] M. Holenderski, M. van den Heuvel, R. J. Bril, and J. J. Lukkien. Grasp: Tracing, visualizing and measuring the behavior of real-time systems. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, pages 37–42, 2010.

[6] R. C. Pinto and J. Rufino. Towards Non-invasive Run-time Verification of Real-time Systems. In *Proceedings of the 2014 European Conference on Real-time System - Work in Progress Session*, ECRTS '14. Euromicro, 2014.

[7] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar: a Flexible Real Time Scheduling Framework. In *ACM SIGAda Ada Letters*, volume 24, pages 1–8. ACM, 2004.