Chapter 18
Framework-Based Debugging for Embedded Systems

Gokhan Tanyeri
Clarinox Technologies Pty Ltd

Trish Messiter
Clarinox Technologies Pty Ltd

Paul Beckett
RMIT University, Australia

ABSTRACT

Debugging embedded systems is almost guaranteed to cause headaches. Embedded systems, and especially portable embedded systems, are becoming increasingly complex and have unique constraints that make them hard to debug. Traditional static debugging tools provided by the embedded development tool chains are important but are only part of the story. Time-dependant issues cannot be debugged by such tools. Embedded environments have to provide efficient mechanisms for managing a range of issues such as thread interaction, control of timers, semaphores and mutexes, IPC message passing, event handling, and finite-state machine organizations. This chapter looks at issues of escalating complexity in modern heterogeneous embedded systems and their impact on debugging techniques and advocates a framework approach to manage this complexity. Using the ClarinoxSoftFrame® Suite framework as an illustrative example, this chapter describes how a modular and open approach to debugging can aid the rapid development of robust wireless-enabled embedded systems that employ a variety of operating systems and platforms. The overall objective in this type of approach is to leverage prebuilt code infrastructure plus existing development skills as much as possible, thereby avoiding the need for engineering staff to learn and re-learn a range of compilers, operating systems, and the like. Overall, debug time can be greatly reduced by improved visibility into the complex interactions between cooperating processes within the code. Collateral benefits can include a reduction in the size of the necessary development team with a reduction in skills specialization.

DOI: 10.4018/978-1-4666-6194-3.ch018

INTRODUCTION

A recent industry survey of embedded systems designers (UBM Tech, 2013) found that more than half the respondents were involved in debugging hardware and/or software and they spent nearly a quarter of their development time on test and debug (see also: (Cadene, 2013)). When the survey asked the question: "If you could improve one thing about your embedded design activities, what would it be?", debugging tools came out on top as one of the most important – equal to compilers and/or assemblers. This is hardly a surprise. Tools for test and debug have been the single most requested area of improvement in embedded design support for many years (Nass, Sept 2, 2007).

Of course, the scope of the debugging process is much more wide-spread than just embedded systems software. However, embedded systems and particularly portable systems are becoming increasingly complex and more often than not have unique constraints which make them hard to debug. In contrast to general purpose computer software design, a primary characteristic of embedded environments is the sheer number of different platforms available to the developers, encompassing CPU architectures, vendors, operating systems and their variants. Embedded systems are, by definition, not general-purpose designs. They are traditionally developed for a single task or small range of tasks and the platform is chosen specifically to optimize that application. Not only does this make life tough for embedded system developers, it also makes the debugging and testing of these systems harder, since different debugging tools are needed for each unique platform (Yagi et al., 2009).

Trends such as the "Internet of Things" are driving explosive growth in the embedded market so that companies that are working in areas such as the *smart watch*, *smart glass* and the various examples of flying gadgets that are proliferating have to look for much more integrated solutions. These systems will tend to encompass a variety of CPUs, Real Time Operating Systems (RTOS) and wired/wireless interface technologies that can cause their code sizes to grow exponentially. There is clearly a need for tools and libraries to be able to cope with these demands.

While the ultimate goal may be to develop code that has zero issues, software engineers must live in the real and imperfect world where system complexity is such that invariably issues will occur. The fact that debugging remains the problematic and costly indicates that they must absorb the idea of ever increasing complexity, prepare for the challenge from day one and acquire more and better debugging tools to keep pace with the increasing demands on new developments. To do this, they need all the help they can get to close the so-called "complexity gap" (Raccoon, 1995). Embedded designers can choose to either loathe the frustration of what can easily become a lengthy and tiresome task or otherwise see it as a challenging game to be fought and won.

This chapter advocates a "framework" approach to debugging complex embedded systems by "plugging" the debugging tools into the framework thereby leveraging an appropriate level of prebuilt code infrastructure without incurring excessive overheads. An approach to debug built around this plug-in framework will encompasses the use both high and low level (static and dynamic) debugging tools and built-in unit testing based on an integrated debugging architecture, the seamless movement of run-time code between desktop and target system, the use flexible memory management tools, profilers and code coverage tools and the reuse of proven and well-tested software components.

Using a commercial tool, the ClarinoxSoftFrame® (Farokhzad, Tanyeri, Messiter, & Beckett, 2010), as an illustrative example we emphasize how a modular and open approach such as this can aid the rapid development of robust, wireless-enabled embedded systems by providing a means to encapsulate and hide any underlying platform differences, both hardware and adjacent software