Software Engineering in Embedded Systems

Stephan Heidinger

February 28, 2012

Contents

0.1 Introduction 1
0.1.1 Organisation 1
0.2 Preliminaries / Foundations 1
0.2.1 Embedded Systems 1
0.2.2 Motivation 1
0.3 Embedded Systems Design 2
0.3.1 Problems 2
0.3.2 Design Steps 3
0.4 Architectural Patterns 3
0.5 Timing Analysis 3
0.6 Real-time Operating Systems 3
0.7 Examples / Case Studies 3
0.8 Assessment 3
0.9 Conclusion 3

0.1 Introduction

This report will introduce into some characteristic features of developing software in *Embedded Systems*.
do some introduction

0.1.1 Organisation

0.2 Preliminaries / Foundations

0.2.1 Embedded Systems

Before talking about how to develop software for *Embedded Systems* I would like to establish what Embedded Systems are. Unfortunately there is no strict definition. But when looking into the subject, one finds certain points surfacing in many definitions and descriptions:

- Embedded Systems respond to a physical world.
- Embedded Systems have to respond in real time.
- Embedded Systems often have only little resources.
- Embedded Systems often run on special purpose hardware.
- Embedded Systems use real-time operating systems.b

Therefore it is convenient to stick to Sommervilles definition of *Embedded Systems*:

> *"An* embedded software system *is part of a hardware/software system that reacts to events in its environment. The software is 'embedded' in the hardware. Embedded systems are nominaly real-time systems."* [Som11, p. 561]

Embedded Software is the part of an Embedded System, that consists of software.

0.2.2 Motivation

Why would I want to talk about Embedded Systems? Well, they are everywhere. When we look around ourselves, we'll quickly realize, how many Embedded Systems are there actually and that there are probably even more of them than regular computers. Among them we could find phones, routers, burglar alarms, coffee machines, any automated system in a car like airbags or distance warners and many more.
Upon realizing how many Embedded Systems we use in our daily life, we certainly realize, that embedded systems must be quite important. Personally I choose this topic, because prior to starting my studies in Constance I did an internship producting a monitoring device for detectors used in the neutron spallation source SINQ at the PSI in Villigen, Switzerland.

0.3 Embedded Systems Design

0.3.1 Problems

Because of the special circumstances of Embedded Systems we are faced with some problems, that are not important, or at least not as important in regular software.

Deadlines: Embedded Systems have to react in real time. Therefore they have to meet certain deadlines upon which results have to be ready or certain actions be taken. Deadlines are probably the most important problem we are faced with, when developing Embedded Software. Therefore Emebedded Systems can be divided into one of two categories depending on the results to meeting a deadline.

Hard Software Systems are systems where the whole programm will fail, when a deadline is not met. This includes i.e. safety critical applications like airbags, ejection seats, water level control systems, ...

Soft Software Systems are systems, where the result will degrade when deadlines are not met. Eventually the system will fail with an increasing number of unmet deadlines. This includes i.e. signal processing, signal transmission, ...

Environment: Embedded Systems have to respond to a physical world. This physical world is not a single state world. Rather it is constantly changing, which has to be taken into account when developing Embedded Systems. We may need to react to multiple events at the same time and also need to verify that a result is still valid upon producing this result. This could best be achieved with a concurrent design, but when we encounter really short deadlines, concurrent languages may not be fast enough.

Continuity: In many cases Embedded Systems run continuously, so they never terminate. Therefore Embedded Software has to be reliable, because it is not feasible to just restart them when encountering an error. Additionally we may need to be able to update the software while it is running.

Direct Hardware Interaction: As Embedded Systems do a wide variety of work, we will encounter a similar variety of specialized and uncommon hardware, i.e. detonators in an airbag, special sensors, special output devices. In some cases this hardware may even be designed especially for our system. It is therefore very probable, that we need to develop drivers for this hardware along with the Embedded System to be able to use it.
In some cases, when our system cannot possibly meet some deadlines, it is advisable to implement some functions not in software, but in hardware, as this is generally faster.

Safety & Reliability: Embedded Systems are in many cases responsible for the well-being of living creatures, i.e. airbags, ejection seats, handle dangerous material, i.e. in a nuclear power plant, or are otherwise used in processes with potentially dangerous to catastrophic results upon failure. These failures may then lead to high costs, either economicaly or in (human) life. To reduce these risks special care needs to be taken to ensure correctness of such systems.

0.3.2 Design Steps

Certain decissions about hardware, i.e. its performance, costs, power consumption (especially in mobile devices), strongly affect the overall performance of the system. As these parts are not easily exchangeable in Embedded Systems they have to be given early consideration and Sommverille does not encourage a top-down approach. He also emphasizes strongly, that there is no standard system design process for Embedded Systems. Instead of that, he proposes some design steps, that are sensible to use. Still, as there is no standard approach, some of these steps may not be feasible for a certain system and one has to think about which steps to use and which to drop.

Plattform Selection

Special Purpose Hardware

Stimulus/Response Analysis

Timing Analysis

Algorithm Design

Data Design

Process Scheduling

0.4 Architectural Patterns

0.5 Timing Analysis

0.6 Real-time Operating Systems

0.7 Examples / Case Studies

0.8 Assessment

0.9 Conclusion

Bibliography

[Som11] Ian Sommerville. *Software Engineering*, volume 9. ed, international ed. Pearson, Boston, Munich, 2011.

Appendix