IL2212 EMBEDDED SOFTWARE

Reflection Assignment 2

Henrik Borg

February 20, 2019

The reflection assignment shall train the students for the 'open' questions, but also improve the writing skills of the students. The participation in the reflection exercises is **mandatory***. Answer the questions* ***individually*** *in electronic form (about 3-4 pages). Be also careful about the quality of your writing.*

NOTE: There is not a single correct answer, in fact for many of these question there exist even multiple different opinions in industry and the research community. It is important that you start to think about the problems in order get a better understanding of the problems related to area of embedded software development. *Please, revisit the slides of the corresponing lectures before answering the question!*

Print your document *five* times and bring the printed copies to the seminar!

Deadline for submission of answers: Wednesday, February 20, 2019.

1 Part A - Models and Design Languages

1.1 What do you view as the largest benefits of the theory on models of computation? How does this theory relate to the classical real-time theory (based on periodic tasks)?

MoCs means that programs and systems will be able to be produced/designed in a systematic way in a top down approach. If the foundation for the MoCs are developed in a good way (do I dare to say the right way) the programs/systems will be produced in a very **predictable and formal analyzable** way. This predictability will lead to systems that are "easily" analysable. Easy in a mathematical point of view, leading to formaly analyzable systems.

This analyzability is what we want to have then we apply our classical real-time theory. **This analyzability will give us the predictability that our classical real-time theory tries to give us.**

1.2 Will we see different programming or modelling languages as future design languages for embedded systems? What is required from such a language?

Maybe not all, but many of the **programming languages** that we have now can be used for describing and generating different parts of the systems. But not for the whole systems. They are either to unpredictable or way to cumbersome to model the whole system in a predictable and efficient way. Or both to unpredictable and to cumbersome.

The **modelling languages** of today is a litle hard for me to spoke for. I beleive that I do not know enough about them. But if it would be easy enough to use them in a MoCs way I think they would be used more and be spoken more about in that regard.

What can I see in the crystal ball that will come in the "near" future? I do see that we will get **a set of modelling languages** that are easy enough to work with, and will produce highly predictable systems, all that done in a mathematically analyzable way. And either we can use a **subset of existing programming languages** or we will get a **new set of programming languages** for developing the low level parts.

Parts that will be mathematically analyzable and predictive. If we choose to use a **subset of existing programming languages** we will get new programming models, or paradigms, how to do the design in a formaly analyzable and predictive way. I expect from a modelling language to have extensions (loadable per project?) connected to different HW platforms there the designer can choose among different low level solutions to common problems and situations. These predifined solutions will ease the formal analyze and reduce the amount of verification (if needed by forced standards).

One quite big problem I can see is how to solve some certain kinds of problems. It is not allways possible to design a perfect predictable and easily analyzable system in a formal way. Such a easy thing as a extra AD-conversion might be needed *will* effect the time execution and prolong it. But the result of that extra AD-conversion might result in a premature ending of that job due to some error handling. Result: unpredictible execution time with some statistical properties.

2 Part B - Design Methodologies

2.1 Do you think a correct-by-construction design flow for real-time systems is feasible? What is required for such a flow? What are the largest challenges?

Yes, I think it is. But maybe not for all kinds of systems. A system with hard deadlines and with more or less unpredictable execution times, even the slightest can be really tough to realize on a hard squeezed system. Therfore I beleive that some systems might not be fully realizable in a correct-by-construction design.

To be able to fulfill this goal I can mostly see one approach using a single or just a few tools:

- The designer must be able to give all the constraints needed by the system. This should be done in such a way that it is easy to write them and easy to get a good overview of them. Without an easy overview it will the hard to change/correct them later if needed.
- Points below should all be done automatically.
- The requested functionality must be devided into submodule.
  - The submodules be devided into futher submodules.
  - Continue deviding into further submodules till we get close enough to the underlying HW so that we can have a analyzable and synthesizable description of it.
  - From here we can analyze the system bottom up.
  - Then all submodules of a module is formaly analyzed, the module can be formaly analyzed.
  - Continute to the top module.
- Then an error or warning is generated that a submodule does not fulfill its constraints, then the tool can either halt and present the message, or collect the messages and present them then done.
- If needed, the tool can halt and ask questions to the operator/designer.
- If all goes well the system has been formaly analyzed and can be automatically syntesized.

This kind of tool can calculate if more than one processor is needed, and give the schedule or type of scheduler, how many units of each resource is required and so on.

The above steps are clearly similar to HW design, such as the use of VHDL design.

The probably greatest challenge would be high density wide distributed system. High density as in high utilisation per processor, often with many tasks. Wide distributed as with long (in time) communication channel, not unusual with stochastic characteristics. These kind of systems is hard to predict. Probability theory is needed to do estimations.

Another problem I almost always see: For the financial department all is about reducing cost and increase yield, in the short term. They will do what they can to keep the HW cheaper then is good for the SW developer team, and therefore most systems will become unnecessary sqeezed and they will later blame the engineers that things take to long time to design. It is hard to convince them that the products will be cheaper (in the long run) to design if they buy a new kind of system that just a few companies are using. And it won't be easier to convice them then the prize tag will be so big that they will get short of breath. This problem will prolong the time extencively till we get good tools for correct-by-construction. Why? Almost only the academic departments are willing to pay for the development of systems with these kind of new thinking.

2.2 Applications in many industrial domains tend to be very large and distributed in nature (i.e. automotive, avionics, industrial automation). What do you view as the most important design aspects that would allow/support short development cycles at manageable costs (design, testing, validation)?

In this answer I will not mension thed importance of getting the correct information before the design of the system. That were part of the previous seminar.

Desgin: In this step we should focus on were it is logical to place different tasks. I.e. the tasks for automatic braking system for analyzed dangerous obsticles should not be placed in the entertainment controlling unit.

One, or a few if necessary, exlucive communication bus for the most critical tasks. I.e. the automatic braking system for analyzed dangerous obsticles.

Extra EMC and ESD protection of those critical communication buses will reduce the risk of failure, resending of information etc. And by that, it will reduce the number of possible failures and therefore the number of test cases to run.

Due to the above mentioned separation,both the **testing** and the **validation** will be easier and less expencive for the most critical parts of the combined system.

The simpler system the easier it will be to make changes and to analyze and validate the outcome of made changes. This means that the above separation will also lead to shorter development cycles.

Of course this separation will increase the HW cost of the system, if it is done to extensively. Therefore I do suggest this should be done more extensively on the really critical parts, i.e. autonomous braking, than for the less critical parts, i.e. always showing the correct speed of the vehicle and the parts that are more nice to have, i.e. entertainment systems.

3 Comments and references

Much of my reasoning come from my own professional experience and personal thory, developed over long time and inspired from several courses at KTH Royal Institute of Technology and many articles.

One of the more inspiring "article" is the lecture slides about "Correct-by-Construction Design of Embedded Systems", especially slide 17 through 19 and slide 23 through 24, in the course IL2212 Embedded Software given at KTH Royal Institute of Technology.

The courses IL2217 Digital konstruktion med HDL (course in VHDL), 6B2471 Telecommunication (mobile networks and birthand death processes in statistics)and 6B3745 Tillmpad statistik och signalteori (probability theory with stochastic processes) given at KTH Royal Institute of Technology.