IL2212 EMBEDDED SYSTEMS

Reflection Assignment for Seminar 2

Weikai Zhou

February 22, 2022

1. **What are the main advantages and disadvantages of (a) the models of computation theory (Chapter 5 in the lecture notes) and (b) the real-time theory (Chapter 3 in the lecture notes)? Is it meaningful to combine models of computation theory with the classical real-time theory as part of an industrial design flow for safety-critical embedded multiprocessor software systems? Give suggestions!**

   The advantages of models of computation theory are that MoCs are formally defined and they can be mathematically analyzed and predicted, which enables the development of tools for simulation, performance analysis and design space exploration, formal verification and synthesis and code generation. That is to say it is a formally analyzable and predictable model. Besides, there are different kinds of MoCs for different functionalities. For example, the synchronous MoC forms the base for the family of synchronous languages and also synchronous hardware, the data flow MoC supports the development of streaming media applications, the discrete-event MoC is used for simulation tools like VHDL and Verilog simulators.

   The disadvantages of models of computation theory are that for some of the languages like ForSyDe and Lustre, there seem to be a few people using them and it lack the forum and community to discuss about it. Thus the updating of them can be slow. And as its name suggests, it is mainly used for modeling the system. It gives the abstraction and data flow of the systems and people can implement and test the functionality of the system. However, it is hard to give the detailed implementation of the system.

   The advantages of real-time theory are that each task has its own priority and generally the priorities vary from task to task so that the concurrent feature can be enabled in the way that the CPU will switch between different tasks according to their priorities, And the tasks can be finished within their deadlines. Therefore, the concurrent feature is achieved with a single CPU. Besides, real-time theory provides protect for critical sections using components like semaphores and resource access protocols. It also supports the schedule of aperiod and sporadic tasks. And we can use real-time theory to calculate the worst case response time and check the schedulability of the tasks.

   The disadvantages of real-time theory are that how to decide the priority of each task is. In class, we mainly talked about Rate-Monotonic Scheduling (RMS) Algorithm and Earliest Deadline First (EDF) Algorithm. The RMS has some problem that it will not figure out a feasible schedule for some task sets. Although EDF can solve this problem, it requires additional knowledge of the deadlines of each tasks and needs to re-compute every time when a job is released to update the priority, which will consume more resource. Another disadvantage is that the switch between different tasks according to their priorities may need extra computational time and decrease the efficiency of the system.

   Yes, it is meaningful. Generally, they will be responsible for different parts of the design flow. For example, the MoC will be mainly responsible for abstracting the system and giving the data flow of the system. And we can verify the functionality of the system. While it is difficult for us to us MoC theory to analyze the schedule, response time and so on of the system. Then, we are going to use real-time theory. It helps us to judge whether the tasks are schedulable or not. And we can also use it to get the worst response time, which can help us analyze the worst total time needed to the system to execute, which is a important part of safety-critical embedded multiprocessor software system.

2. **Chapter 6 presents design space exploration (Section 6.1) and software synthesis (Section 6.2). Are these phases existing in current industrial design flows? If yes, how are they conducted? What do you see as the biggest challenge for design space exploration? Are the *general principles and rules for synthesis* in Section 6.2.2 really general and can be applied to all target architectures and models of computation?**

   Yes, I think they exist in current industrial design flow.

   Design Space Exploration (DSE) aims to identify efficient and suitable implementation in an early stage of the design process. It includes different parts like architecture instance, applications, mapping, performance analysis, performance numbers and so on. Each step is a single mapping and is analyzed using a suitable performance analysis method like simulation based methods, analytical methods and hybrid methods. Different steps are connected to form a iterative process. Therefore, for each step, we can iteratively analyze it again and again with different implementation methods to find out the best one. Therefore, it can be generally understood that it searches the possible designs to find suitable implementations. And ForSyDe has its own DSE tool DeSyDe, which is similar to the process discussed above.

   Synthesis generally is translating the MoC into another target technology, which can be a target language or platform consisting of hardware or software. For software synthesis, we generally transform ForSyDe into C code in class discussion. First, we should translate the process network into a static schedule with sufficient buffer space. Then we convert each process constructor into a corresponding C-code template. Finally, we translate each function into the corresponding C-code. The work is done from general structure to detailed implementation.

   The biggest challenge for design space exploration is that it can be very large for complex embedded systems. As mentioned above, DSE is a iterative process. It searches the possible designs and conduct performance analysis to find suitable implementations. With the increase of complexity of embedded systems nowadays, the iterative method will take much longer time for all possible designs. Besides, the performance analysis is also difficult. We have simulation-based methods, which is time consuming and impossible to simulate all possible input and internal state combinations. We have analytical methods, which are often pessimistic like analysis for WCET and difficult to find good analytical models for like power. We also have hybrid methods, for which how to combine simulation-based methods and analytical methods are hard to determine.

   The general principles and rule for synthesis are a) a correct implementation of the process network; b) a correct implementation of all process constructors in the target technology; c) a correct translation of all function arguments and parameters. I think they are really general and can be applied to all target architectures and MoCs. The principles are a top to down format, meaning that is first constructs the structure, then processors and finally detailed functions. And it also considers the differences among different languages and platforms. Some of the MoCs may not be transformed into target technology due to the reason that the target technology does not support certain functions or libraries. Therefore, the principles and rules highlights correctness, which gives the freedom of implementing them use the native functions and libraries of the target technology as long as they are correct.

3. **To what extent can the idea for a correct-by-construction design flow (as given in Figure 1.5) for embedded real-time multiprocessor software systems, where several applications shall be implemented on a shared platform, be automated? What are the largest challenges? Is this flow suitable? Are important parts missing in the flow of Figure 1.5?**

   It depends on the selection of the underlying formal base. If we choose formal modeling technique and predictable architectures, we can achieve this to a high extent. While if we use informal modeling technique ore non-predictable architectures, it will be more difficult to be automated. This is due to the nature of correct-by-construction design flow, which requires the formal analyzability and predictability. The test and performance analysis of different parts can be highly automated. We can pre-design testing platforms and set our constraints to test and verify our design. The synthesis can also be automated for some languages and platforms. For example, ForSyDe.Deep can help generate VHDL files while Lustre can generate C file.

The largest challenge may be how to combine all the different parts together to implement the final system. In the whole design process, we divide the system into different parts. And for these parts, we may also have different teams for different jobs including design, verification, testing and so on. For example, if we find this part of design has low performance, how should we figure out the solution to this problem, which will involve the cooperation of testing and designing. Another example may be that even though all the subsystems pass their own performance testing, when we put them together and the whole system may fail the test. It may be because of the shared resources other other facts. And it can be hard to detect the location of the problem and find the solution of the problem.

Therefore, the flow is suitable generally, which shows almost whole design process. However, it still lacks some detailed parts like how should we handle the problems that may occur in the final combination of each part. And the Design Space Exploration can also be extended about how the performance analysis are done and what steps should be taken when an error or problem is found.