IL2206 EMBEDDED SYSTEMS

Reflection Assignment 1

Badi Mirzai, badim@kth.se

August 2020

1 Reflection Assignments for Seminar 1

1.1 Give three new examples (not mentioned previously in the lectures or lecture notes) for typical saftey-critical embedded systems from different application domains.

When Scuba diving, divers need to take into consideration the change in pressure as they emerge up to and down from the surface of the ocean. This is why divers today heavily rely on diving computers that guides them in safety. The diving computers must therefor work properly under strict conditions in order to give correct information for diver. An embedded system that doesn't work correctly here (or stops working) might cause decompression sickness for the diver or even cause death if it's giving misleading information. This is because the release in pressure too fast causes the Nitrogen in the body being dissolved in liquid/bubble form which causes permanent problems. The diving computer therefore needs to work under high pressure and be able to make calculations without any mistakes and therefore must meet the worst case specifications under the worst case circumstances with safety margins.

Embedded systems are found in the home, specifically in the kitchen such as the microwave and oven that needs to be reliable and safe all the time. If the embedded systems in these systems are not safety critical, they can cause fire or an explosion due to the high temperature and electrical power needed to generate the heat. Today people heavily rely on these systems, especially with the development of internet of things in the kitchen evolving. It might be scary to think how wrong it could go if your kitchen-equipment at home would not be safety-critical.

Safety-critical embedded systems are critical for tools used in oil platforms, where an error in it's embedded system could cause an explosion and cause many deaths. It could also cause leakage of oil which could be catastrophic for the environment and nature.
Another example of safety-critical systems are in space exploration, where astronauts rely on embedded systems that work under all circumstances. A failure in the spaceships embedded system might cause an explosion in the spaceship and therefor cost the lives of the astronauts.

1.2 What are the main problems and challenges in the design process of safety-critical embedded systems?

One of the biggest challanges when it comes to designning safety-critical embedded systems is to meet their requirements while staying within the constraints of the problem. This requires good understanding in different levels of domains, from high level software to the low level in the hardware. When designing the system to meet the safety specifications, one must understand the limitations of the hardware even when starting at the higher level of abstraction. In the design process, one needs to be able to understand from high level design to the low level hardware design. This takes a lot of knowledge and it might be hard to make the entire system work together while meeting the requirement constraints. The higher the abstraction of the design, the more flexibility we have. The flexibility gets narrowed down as the abstraction decreases, which is why the designer needs to have a relative good intuition of the hardware limitation so it gets narrowed down to the right point.

A big challenge in the design process is to make the system work at all times and efficiently (with respect to power consumption and time effectiveness). The system cannot fail during its usage, or have any unexpected time delays or lack of memory, power failure etc. For example, the diving computer must be able to compute the correct time for the diver to be at its depth while not risking to suffer from power failure. If too little memory is unavailable, then important information might be overwritten, causing in incorrect information for the diver. The designer should also take into consideration how much hardware it should implement so that the system can be fast enough, while still maintaining a reasonable price for the cost of the product. This is a important trade-off between the cost and performance.

The most important thing though is to meet the worst case specifications at all times, which can be hard to know what is might be.

It is important not to forget that the systems should be safe from a cyber-security perspective so that no one from the outside can control and do harmful things with the system through different peripherals.

1.3 Discuss relevant properties for an embedded computing platform designed for safety-critical systems. Give characteristics of the individual components (processor, memory, interconnection network, peripherals) that build the platform for a safety-critical embedded systems?

The microprocessor is critical in making computations and executing the functions and different tasks. The speed of the processor can be vital in life-critical situation, since speed will play an important role. Microprocessors can be programmed in software which gives flexibility but also need to fetch and allocate memory during its execution. The processor execute its instructions by first fetch instruction from the (external or cache) memory, decodes this instruction before it can execute it. There is of course the trade-of between the processor speed and the cost of the product, which needs to be taken into consideration as mentioned before.

Peripherals: Processors needs to be able to connect to the I/O and get the necessary input. Important that the processor is doing other things instead of waiting for the input, but needs to be able to efficiently take in the input whenever needed while saving the job it did before and returning to it when Input is done. This switch between these tasks might take a lot of time. Need to prioritize different inputs, for example the braking in a car might be more important that the performance of the music in the car.

Memories are important for storing, fetching and writing information, which is why is it important to use different memories for different purposes. It is always important not to run out of memory, but cost puts a limitation on what kind of memories that can be used. Registers are the fastest type of memory. Registers might not be the place you want to store important information without backup, since they are a type of volatile memory meaning that the information is not stored permanently and will be lost when device is turned off. Registers are good for the CPU when it's doing computations, but the system should be able to store the data back to other types of memory after its execution.
Although there are different levels of cache memory, it's usually integrated in the same chip with the micro processor so that it can fetch important data that it might need. This is important, because this data transfer between CPU and cache (hopefully) reduces the time it takes to find and fetch data. But there is no guarantee that the data that the CPU is looking for is at the Cache memory. If the Chahe is full then different strategies can be used, either randomly replace new memory in the cache or prioritize the ones that have been recently used. The system needs to prioritize which data in the cache should be overwritten when fetching new data from main memory, while making sure that the information is not lost in the process. This is usually done by the Cache controller, that handles if data should be fetched from Cache or main memory. The designer should take this into consideration so that safety-critical information is in the main memory in case it will be lost in the cache.

A 2-way set-associative Cache can be used to avoid conflicts in memory locations when mapping from the main memory during the fetching. If the system needs to guarantee a certain worst case performance, then one might consider using scratchpad memories, which gives a direct mapping between the scratchpad and cache. This will guarantee a better worst case performance in memory fetching, but the average performance (hit rate) might be worse. A good hit rate in the cahce memory (when finding what the CPU is looking for in the Cache) gives a good average performance. If the objective is to have a high hit rate, then a full associative cache might be worth considering, since it gives higher hit rate with high flexibility. However, this gives a trade-off for the time it takes to find the data in the cache, since all lines needs to be visited. A lot of hardware is therefor needed and thus this becomes practical when the cache memory is smaller.

The interconnection network connects the different components of an embedded system (CPU, memory, I/O, DSP, etc.), where information between them needs to be transferred efficiently and reliably. There are different ways to do this, and of course will there be trade-off between performance and cost. Data-Bus structures gives a lower cost, but might not be able to deliver high performance in the speed of the data transfer and is therefor not good in high scalability. This is solved by more advanced architectures such as Network on Chip (NoC), where there are many resources connected by switches.
Since the bus is a shared information link between different components, it can only handle one data transfer at a time and thus not possible to work in parallel. This means however that all the components connected to the bus can read the information that is being shared, which might be an security issue if you don't want important information to be accessed by other components. A synchronous bus divides the time into clock cycles and can thus run faster since it doesn't require much logic. However, this means that devices will share the same clock and can suffer from clock scew (different distances of the components from the clock source resulting in different arrival time). On the other hand, a asynchronous bus doesn't need a clock and can therefor connect different devices. This also removes clock scew and allows data delays which might make it more possible to have components further away and lower criteria on the data transfer.

Since a bus can only handle one data transfer at a time and not all devices should have access to the bus, a bus master and slave is used. The master can control on their own with the bus and the slave must get permission from the master by responding to its request. If there are multiple masters, a arbiter is used to decide which master would have access to the bus at a given time to ensure that there is one data transfer at a time. The priority of which master should be chosen can be done in different ways, for example with priority or fairness. This is important since different bus masters might be much more important in a safety-critical system which can otherwise lead to problems. This of course depends on what system you have and what the objective is.

The I/O peripheral looks for the right input (for example keyboard pressed) and then goes to the right memory location for the next steps. It is important that it is robust to outside noise while being safe for not being manipulated by some external system that can access internal things through the I/O.

The Direct Memory Access peripheral handles the I/O so that the CPU can focus on other things. The DMA is also a peripheral which uses both slave and master port for working together with the CPU when transferring data.

1.4 Safety-critical systems execute a lot of software. Think about and discuss how the timely behaviour of the software can be supported by the hardware.

Cache memory close to the CPU will result in lower time for the processor to access the memory at the chache (provided that the memory does not need to be fetched from the main hard drive). The larger the cache memory, the lower the probability that its memory is full and therefor will make less memory needed to be moved from the main drive to the cahce.
Multicore CPU with their own cache make it possible for them to run in parallel so it does not wait for delays. This means that you can have multiple software functions that run in parallel and this increases time effectiveness.

Depending on the hardware specifications/objective, one might prefere to use a specific programming language that best supports the the hardware. For example, if we have little memory, then it might be a bad idea to use a high level language since the compiling of complex functions might take up too much of the memory.

A more complex software language will give more flexibility for what can be done and it’s certainly easier for the programmer, but it does however require more computations and better hardware to execute some of those tasks which can be more energy consuming.