Embedded Systems Architecture

Sérgio de Jesus Duarte Dias

Departamento de Informática, Universidade do Minho
4710 - 057 Braga, Portugal
sdias@idite-minho.pt

Abstract. When a computer is an integral part of a larger system, where it intensively interacts with the surrounding environment and frequently deals with real-time constraints, it is considered an embedded system. This communication describes the embedded system world: architecture, classification, characteristics, system limitations, and trends. It also discusses architectural features that cannot exist in hard real time control.

1 Introduction

Whenever the world microprocessor is mentioned, it tends to think in a desktop or laptop PC running an application such as a word processor or a spreadsheet. While this is a popular application for microprocessors, it is not the only one and the fact is most people use microprocessors indirectly in common objects and appliances without realising it.

The embedding of microprocessors into equipment and consumer appliances started before the appearance of the PC and consumes the majority of microprocessors that are made today [1]. In this way, embedded microprocessors are more deeply ingrained in everyday life than any other electronic circuit. A high-end automobile may have 100 microprocessors, but even inexpensive cars today use multiple microprocessors. Some of these microprocessors do very simply things such as detect weather seat belts are in use, others control critical functions such as the ignition and braking systems. Mobile phones contain more processing power than a desktop processor of few years ago. Many toys contain microprocessors and there are even kitchen appliances such as washing machines, refrigerator or toasters.

1.1 Classification

An embedded system is a microprocessor-base system that is built to control a function or range of functions and is not programmed by the end user in the same way that a PC [1]. An embedded system has only one application. It can have several functionalities, but they belong to the same application [2]. One washing machine has several wash programs (functionalities), but the embedded application (functionality of system) is wash and cannot be changed to cook.

Embedded systems can be divided into two different subclasses [3]: embedded controllers and embedded data-processing systems.

Embedded controllers are dedicated to control functions. They are control flow dominated and react to external events [4]. Therefore, embedded controllers are very often called reactive systems. Reactive systems typically respond to incoming stimuli from the environment by changing its internal state and producing outputs result. Normally they support a set of modes and settings and their real-time constraints are often in the range of milliseconds. Therefore, the performance required usually varies from low to moderate. For this reason microcontrollers are very often sufficient to implement an embedded controller. Ex-

amples of embedded controllers are automobile and industrial applications or home appliances (e.g. wash machines)

Embedded data-processing systems are dedicated to data communication and processing. Therefore, they are very often called transformational systems. These systems are data flow dominated and often they are real-time systems executing a special function within a predefined window. They require a much higher performance compared to embedded controllers. Therefore, microcontrollers are not sufficient and more powerful microprocessors (very often DSP – Digital Signal Processing or ASIPs - (Application-Specific Instruction-set Processor) and ASICs (Application-Specific Integrated Circuit) are required. Examples of embedded data-processing are: audio and video applications or wireless communication.

Embedded systems are often implemented by heterogeneous systems consisting of dedicated and programmable parts [3]. Therefore, they are well known as hardware/software systems where "hardware" represents the dedicated hardware parts, and "software" the programmable processors.

Heterogeneous systems contain dedicated hardware parts (ASICs) and programmable embedded processors. These processors may be, for example, DSP (Digital Signal Processing) or RISC (Reduced Instruction Set Computer) processors. In addition, RAM (Random Access Memory) is required for storing data and ROM (Read Only Memory) for program code and constant tables. Peripherals (A/D converter, D/A converter and I/O unit) are necessary for communicating with the environment. A network of busses and wires connects all these components. Heterogeneous systems can be integrated on single board or on single chip (SOC) [5]. Single-board systems are integrated on a board containing components, like ASICs, ASIPs, processors and memories. Single-chip systems are integrated in one single ASIC containing processor cores, dedicated parts and memories. The processor cores of a single-chip are provided in the component library of the ASIC technology. The advantages of single-chips solutions compared to single-board solutions are the increased performance and reliability. In addition, the power consumption and the manufacturing costs are reduced. On the other hand, single-chip systems have a larger chip size and therefore debugging those systems become much harder.

2 Embedded Processors

It tends to distinguish between microcontrollers and microprocessors, even thought there's no hard and fast definition. Viewed very simply, you often find microcontrollers associated with the embedded domain and microprocessors with desktop arena [6].

Microcontroller can be considered as self-contained systems with a processor, memory, and peripherals so that in many cases all that is needed to use them within an embedded system is to add software [1]. Microcontrollers are usually available in several forms.

Devices for Prototyping or Low Volume Production Runs. These devices use nom-volatile memory to allow the software to be downloaded and returned in the device. Usually it is used UV erasable EPROM (Electrical PROM), but EEPROM (Electrical Erasable PROM) is also used. Some microcontrollers used a special package with a special piggy-back socket on the top of the package to allow an external EPROM to be plugged in for prototyping. This memory technology replaces the ROM on the chip allowing software to be downloaded and debugged. The device can be reprogrammed as needed until the software reaches its final release version. The use of nom-volatile memory also makes these

devices suitable for low volume production runs or where software may need customisation and thus preventing moving to a ROMed version.

Devices for Low to Medium Volume Production Runs. Instead of use UV EPROM or EEPROM it is used OTPs (One Time Programmable). These devices use EPROM instead of the ROM but instead of using the ceramic package with a window to allow the device to be erased, it was packaged in a cheaper plastic pack and thus was only capable of programming a single time. These devices are cheaper than the prototype versions but still have the disadvantage of programming. However, their lower cost made them a suitable to producing ROM device.

Devices for High Volume Production Runs. For high volumes, microcontrollers can be built already programmed with software in the ROM. To do this, a customer supplies the software to the manufacture that then creates the masks necessary to create the ROM in the device. The advantage to the customer is that the costs are much lower than using prototyping or OTP parts and there is no programming time or overhead involved. The downside is that there is usually a minimum order based on the number of chips that a wafer batch can produce and an upfront mask charge.

3 System Limitations

Most embedded systems place severe constraints on the processor in the terms of requirements for size, weight, power, cooling, performance, reliability and manufacturing constraints [2], [3], [6].

Physical size and Weight. The physical aspects of the final system can vary greatly depending upon the application. An industrial control system for assembly line may be designed to fit into a standard-size rack with no strict limitations on weight. A handheld device typically has tight requirements on both size and weight that can ripple through the entire system design.

The problem with size and restrictions is that high performance processing systems tend to be larger and heavier than slower systems. At the CPU (Central Processor Unit) level, a processor that has a large number of pines take up large printed circuit board area. At the system level, a design that needs cache memory controller chips and large amounts memory takes even more printed circuited board area.

Performance. In Embedded Systems not only instructions-per second rating is important, other factors which are vital to systems performance include interrupt response characteristics, context switching overhead and I/O performance.
Two important characteristics in embedded applications are predictability and determinacy. Predictable systems have behaviour at run-time that as understandable from examination of the original source code program; Deterministic systems have instructions whose behaviour does not vary.

Power Consumption. Power is important in battery-powered systems and is often important in other applications as well. It is important to distinguish *energy* and *power*. Power is energy consumption per unit time. Heat generation depends on power consumption that

results in increased cooling requirements. Battery life, on other hand, most directly depends on energy consumption.

Thus, most embedded microprocessors have three different modes: fully operational, standby or power down, and clock-off [2]. *Fully operational* means that the clock signal is propagated to the entire processor, and all functional units are available to execute instructions. In *standby* mode, the processor is not actually executing an instruction, but all its stored information is still available—for example, DRAM (Dynamic RAM) is still refreshed, register contents are valid, and so forth. In the event of an external interrupt, the processor returns (in a couple of cycles) to fully operational mode without losing information. In *clock-off* mode, the system has to be restarted, which takes nearly as long as the initial start-up.

Most new processors focus on reducing power consumption in fully operational and standby modes. They do so by stopping transistor activity when a particular block is not in use. For that reason, such designs connect every register, flip-flop, or latch to the processor's clock tree. The implementation of the clock therefore becomes crucial, and it often must be completely redesigned.

Cost. The target cost or purchase price for the system is almost always a consideration. Cost typically has two major components: *manufacturing costs* includes the cost of components and assembly; *nonrecurring engineering costs* include the personnel and other costs of designing the system.

Reliability. Embedded processing Applications are notorious for extreme operating conditions, especially in automotive and military equipment. The processing system must deal with vibration, shock, extreme heat and colt, and perhaps radiation. In remotely installed applications, such as spacecraft and undersea applications, the system must be able to survive without field service technicians to make repairs.

Real-Time / Reactive Operation: Real-time system operation means that the correctness of a computation depends, in part, on the time at which it is delivered. In many cases the system design must take into account worst-case performance. Predicting the worst case may be difficult on complicated architectures, leading to overly pessimistic estimates erring on the side of caution. The Signal Processing and Mission Critical example systems have a significant requirement for real time operation in order to meet external I/O and control stability requirements. Reactive computation means that the software executes in response to external events. These events may be periodic, in which case scheduling of events to guarantee performance may be possible. On the other hand, many events may be aperiodic, in which case the maximum event arrival rate must be estimated in order to accommodate worst-case situations. Most embedded systems have a significant reactive component

4 Real-Time Architectures Constraints

Many embedded applications have to perform in real time – if the data is not ready by a certain deadline, the systems break. In some cases, failure to meet a deadline is unsafe and can even endanger lives. In other cases, missing a deadline does not create safety problems but does create unhappy customers – missed deadlines in printers, for example, can result in scrambled pages [2].

There are many reasons than an architectural feature is included in a processor. Today, usually the feature is added in order to support general purpose computing, which usually means engineering workstation programs. The list of architectural features that are inappropriate for hard real time embedded control systems seems similar to a catalogue of modern processor design techniques [7]. Most of these features break two important performance rules of hard real time control applications: predictability and determinacy.

Cache Memory. Data and instruction cache memories are the biggest sources of unpredictability and indeterminacy. Modern compiler technology is just beginning to address the issue of scheduling instruction cache misses. Because the time required to process a cache miss is approximately an order of magnitude slower than the time required for a cache hit, significant execution speed degradation takes place even a small percentage of cache misses. Figure 1 shows relative execution time variation that may take place with varying number of cache misses. In hard-real systems, often the worst case of all cache misses must be assumed for time-critical sections, resulting in designing hardware with only fast static memory chips that render the cache management hardware superfluous.

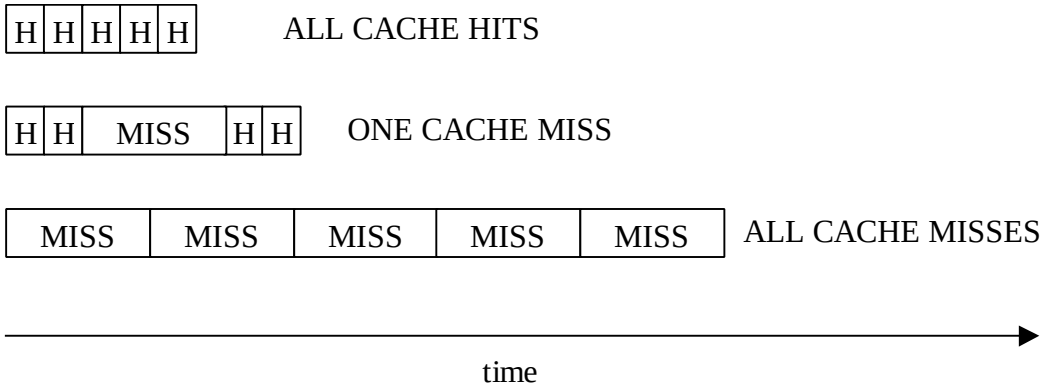

Fig. 1. Example of cache misses causing variable execution time.

Variable Length Execution Time of Instructions. Many instructions (usually on CISC systems) can take a variable number of clock cycles to execute, often depending on the data input to the instruction.

Low Semantic Content of Instructions. Low semantic content of instructions greatly increases the number of instructions that must be executed to accomplish a given task. This, in turn, increases demands for high-speed memory. A requirement for large amounts of high speed memory in turn either requires the use of cache memory or a significant expense in making all program memory out of cache-grade memory chips for high guaranteed system performance.

Write Buffers. Write buffers allow the CPU to perform a write to memory without pausing for an actual memory cycle to take place. The problem is that write must then wait for spare bus cycles. If no spare bus cycles are forthcoming, the processor must be stalled when the write bus overflows. Additional stalls can be caused if a memory read could possibly correspond to a memory location that has yet to be updated by the write buffer. Interaction be-

tween the write buffer and cache misses for instruction and data fetches can cause indeterminacy in program execution.

Pipelines. Deep instruction pipeline increases interrupt response latency. Even if the first instruction of an interrupt service routine can be inserted into the pipeline within on clock of the interrupt being asserted, it still takes several clock cycles for the instruction to pass thought the pipeline and actually do its job. A pipeline also requires some sort of handling of data dependencies and delays for memory access, which results either in compiler-generated nops, instruction rearrangement, or hardware-generated pipeline stalls. All of these dependency-resolution techniques decrease predictability and determinacy, or both.

Superscalar Instruction Issue: Microprocessors that implement "Superscaler" instructions execution, in which multiple instructions may be issued in a single clock cycle. Unfortunately, he number of instructions that may be issued depends on the types of instructions, the available hardware resources, and the execution history of the program. All these factors make it very difficult to determine any single instruction's execution time.

Pure Load/Store Architectures. Load/store RISC architectures can by their very nature make no provision for atomic read/modify/write instructions. This generally required the addition of external hardware for implementing semaphores, locks, and others inter-process communication.

Brach Target Buffers. This is a special case of an instruction cache, in which past program execution history is used to cache instructions at branch targets for quick access. This type of instruction cache operates is a manner dependent on input data, and so is beyond the ability of compilers to manage effectively. Branch target buffers are sometimes used in conjunction with branch prediction strategies, in which the compiler encodes a guess as to whether a particular branch is likely to e taken into the instruction itself, causing either the branch target or the next consecutive instruction to be fetched before the outcome of the branch is resolved. The actual time to perform a branch then depends on whether the branch target buffer value corresponds to the guess made by compiler

Prefetch Queues. Prefetch queues greatly affect the predictability of execution because the execution time for an instruction depends on whether or not the preceding instructions were slow enough at allow the prefetch queue to accumulate new instructions. Thus, determining whether a particular instruction will execute quickly from the prefetch queue or slowly from program memory require cycle counting of several preceding instructions to see if spare memory cycles would have been available for the prefetch queue to fill. This cycle counting is subtly affected by data-dependent execution paths through the program, as well as the number of wait states or cache misses in program memory. For example, if there is latency for filling an empty prefetch queue, adding a one-cycle wait state that causes the prefetch queue to be emptied may add more than one clock cycle to program execution time.

Virtual Memory. The use of virtual memory implies the use of a cache memory to perform address translation, which has the same problems as the other caches. If a disk-paging system is used, the problem of speed degradation can become much worse than for simple cache misses.

Autonomous I/O Processors. The use of autonomous I/O processors and DMA (Direct Memory Access) hardware that steel bus cycles can cause nom-determinism as the processor is stalled for bus accesses. Consequently, it is sometimes desirable to perform I/O directly to CPU.

5 Conclusions

Most embedded systems are based in 8 bits microcontrollers (e.g. 8051, PIC, 68HC11…), since they are inexpensive and the same chip can integrate a processor, a memory and peripherals. These microcontrollers start becoming obsoletes, but instead of emerging new microcontrollers architectures, old general-purpose processors start to reappear. Processors in end lifecycle like 80486 from Intel or MC6840 from Motorola offer more performance than is usually available from a traditional microcontroller. These processors can be combined with memory (RAM and ROM), peripherals such as parallel and serial ports, DMA controllers and interface logic in the same chip (SOC). Therefore, these devices are more suitable for embedded systems by reducing the hardware design task and costs. Modern 32-bit RISC processors become to apply the same technique, to come into the embedded marketplace.

This new type of architecture radically changes the embedded system designs. The performance, and size are no longer a big constraint. On the other hand, system debugging is strongly punished.

Embedded system design is much more complex than programming PCs because we must meet multiple design constraints, including performance and costs. Designing embedded systems is not the sum of hardware and software development, but the relationship of hardware architecture with the software application.

References

[1] Heath, Steve: Embedded System Design. Newmes, (1997)

[2] Wolf, Wayne: Computer as Components – Principles of embedded computing systems design. Morgman Kaufmann Publishers, (2001)

[3] Niemann, Ralf : Hardware/Software Co-Design for Data Flow Dominated Embedded Systems. Kluwer Academic Publishers, (1998)

[4] Machado, Ricardo, Metodologias de Desenvolvimento em Projectos de Engenharia de Computadores no Suporte à Implementação de Sistemas de Informação Distribuídos Não Convencionais (Industriais). Tese de Doutoramento, Escola de Engenharia, Universidade do Minho, (2001)

[5] Brás, Arnaldo: Systems on Chip: Evolutionary and Revolutionary Trends. 3rd Internal Conference on Computer Architecture (ICCA'02)**,** (2002)

[6] Schlett Manfred: Trends in Embedded-Microprocessor Design. Computer Aug 1998 Vol. 31, No 8, pp 44-49

[7] Koopman, Philip: Design Constraints on Embedded Real Time Control Systems. Systems Design & Networks Conference, (1990)