MODERN SYSTEM ARCHITECTURES IN EMBEDDED SYSTEMS

T.Korhonen, PSI, Villigen, Switzerland

Abstract

Several new technologies are making their way also in embedded systems. In addition to the FPGA technology which has become commonplace, multi-core CPUs and I/O virtualization (among others) are being introduced to the embedded systems. In this paper we review the trends and discuss how to take advantage of these features in control systems. Some potential application examples are discussed.

THE DOMINANT TRENDS

Many of the technical advances in the computing industry are driven by the needs of the applications in the general information technology, Internet and the web, telecommunication and the entertainment industry. Common for these markets are that the data volumes are increasing, as well as need for of data communication bandwidth and processing power. To respond to these demands, the computer and electronics industries have come up with several technology solutions.

The embedded and control system applications are in some areas in the leading edge of technology, but in other areas the technology used in the general computing is coming into the applications with a delay. This delay is to a large extent due to the nature of the embedded market: the application life cycle is much longer than in the general computing market.

The general trends of the computing market are not essentially new but have practically always been to a) provide more computing power and communication bandwidth and b) use the hardware investment in the most efficient way. In the next chapters we take a look at these solutions and what they could bring to the field of control systems. This is not always obvious and it is a time-consuming process to implement this into the embedded infrastructure.

NEED FOR SPEED

Computers are never fast enough. Each advance in the processor technology has taken us a step further, but as soon as a solution has become available, a number of applications have arisen that use up all the available power and require even more.

As increasing speed just by making the CPU clock rates higher has become impractical, the major method is now increasing the parallelism. This is happening in a number of ways, most apparent is putting multiple processors in one unit. However, this is not the only way to implement parallelism.

Speed by Parallelization

Starting at the lowest level of parallelization, FPGAs have been around for some time and have become very common in embedded applications, too. The recent applications have started to use them as processing engines instead of glue logic. An FPGA can be the ultimate tool for fine-grained parallel processing if the tasks in hand match the model. The downside of using FPGAs is that the application development is work-intensive, as the abstraction level of the tools is low. To implement an application in a FPGA the problem needs to be well constrained. FPGA is also a fairly expensive solution in many cases.

In the processor world the obvious way of introducing parallelism is to increase the number of processing units and divide the computing tasks between them, Multi-core CPUs have emerged in the recent years, being now the standard in server and desktop machines but have also recently entered the embedded market. There are essentially two different directions in multicore implementations: having a number of identical cores (typical in desktop computers) or having a mix of general and special purpose processors on the same chip, like the TI OMAP[1], which combines a (dual-core) ARM CPU,DSP and GPU on the same chip. This kind of heterogeneous multicore processors could be very interesting in applications that need floating-point and mathematical operations in the embedded level.

As an example of a multicore CPU targeted for the embedded market, the QorIQ SoC series by Freescale [2] is displayed in Fig. 1. In addition to a variable number of processor cores, it also integrates specialized function accelerators on the same chip, for instance a TCP packet accelerator that implements a part of the TCP packet processing in hardware. It also integrates Gigabit Ethernet transceivers and three PCI express endpoints plus a number of specialized function controllers like an encryption engine.

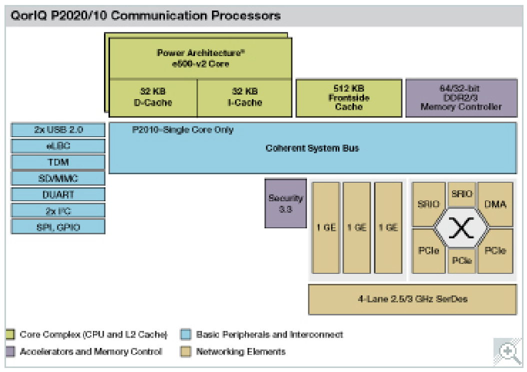

Figure 1. Freescale QorIQ® P2020 block diagram

Interconnects

To make advantage of the multiple processing units, the units need to communicate efficiently with each other. This has traditionally been implemented with multidrop buses. However, a bus scales very badly with the increasing number of processing units. For this reason, the interconnects are more and more going to the direction of serial point-to-point links. In dedicated applications these links have been used already a long time. General-purpose interconnect protocols are now being included directly in processors so it makes a lot of sense to use them in embedded applications, too. The most prominent interconnect protocols are Ethernet (obviously) and PCI Express. The other protocols cater for specialized needs and what will happen with them in the long term is not that obvious. The two prominent protocols are however likely to have a large share in applications and be widely supported long into the future.

Controls Applications

A typical embedded application consists of tasks for I/O and some level of processing. As the processing power in the low level has grown, more and more applications have migrated to the low level. This has several advantages, as the data can be pre-processed right down at the source and only the relevant data needs to be passed on to the higher level. To achieve this in a low level embedded processor, there are a number of options. One can implement applications in a FPGA and take advantage of the real parallelism possibilities and very hard real time capabilities. This can however be very labour-intensive.

For the processing with a CPU, the obvious way is to try to take advantage of the multiple cores. A relatively simple way is to take advantage of a symmetric multiprocessing (SMP) OS and let it take care of distributing the load between processors. This can work quite well when the workload consists of different tasks that have little or no relation to each other. However, for embedded applications this is often not the case. The other possibility is then to dedicate computing cores for different tasks, or define certain tasks to run on different cores. This can help in management and partitioning of the workload.

The SoCs that provide different specialized cores can also have interesting possibilities. For instance the OMAP processors with their special processors could be efficient when the workload is a mixture of heavy floating point and matrix operations plus running general purpose tasks (I/O handling, network tasks etc., for which a general purpose core like an ARM is well optimized. Dedicating coprocessors for different tasks would also enable applications where a tight connection between a modelling environment and an embedded controller is useful. The modelling can happen in an workstation and the modelled application can be run on a dedicated core, where it would not need to disturb the general purpose tasks, and restarted as wanted.

Using GPUs as coprocessors for doing intensive matrix manipulations is also a possibility that is becoming more and more common.

QUEST FOR EFFICIENCY

Even if the price of computing equipment has come down, it still always costs too much. One of the long-standing ideas (and implementations) of taking the most benefit of the installed equipment has been to share the computing resources between several applications. Instead of each service running on its own machine, sharing the time of the hardware by running several things in parallel can improve the utilization of the hardware.

Virtualization

However, even more than optimizing the use of the hardware the problems facing IT departments now are physical space in the data center, power/cooling costs, system maintenance and management. The performance of servers has also increased to a point where there is a large amount of idle capacity. With virtualization, IT departments can utilize that spare capacity rather than adding a new physical server to support a new environment.

Virtualization is traditionally implemented with the help of a hypervisor that controls and shares the computing resources at a low level. The hypervisor is usually a thin software layer below the guest operating system. As the number of cores in CPUs increases, the task of a hypervisor gets more and more demanding and it self starts to become a bottleneck for the effective load sharing. Thus the functions of a hypervisor are increasingly being implemented in hardware. The need is most obvious and pressing in the area of I/O.

I/O Virtualization

The concept of I/O virtualization [3] is fairly recent and its meaning may not be immediately obvious. What this means is the implementation of the tasks of a software hypervisor in hardware to improve the efficiency. When many guest systems want to share the access to I/O modules the supervisor becomes a bottleneck when all the accesses have to be arbitrated. For instance, the SR-IOV standard within PCI express defines so called virtual functions in the devices. The guest systems can be given direct access to the hardware and the need for data manipulation by the hypervisor is eliminated.

SHARE AND CONQUER

Being a discipline with long development cycles, most of the software infrastructure in the control systems field is still oriented towards the single-processor model. Only recently activities to address have been initiated; some of them presented also in this conference. For instance the handling of timing of real-time processes: in a single-core system

The rise of open-source software has had a big impact on the way systems are built. Direct access to the source code has enabled implementation of functionality that was previously an area where only proprietary solutions existed. In addition, several working groups are defining open standards to make the utilization of multi-core systems easier. A notable standardization body is the Multicore Association [4], which has several working groups working on standardization of APIs in different areas. For instance MCAPI working group provides a standardized API for communication and synchronization between closely distributed cores and/or processors in embedded systems. The working group has formed two subgroups. One is working on ‘zero copy’ functionality, including bidirectional interaction between ‘application and application’ using shared memory and bidirectional interaction between ‘application and driver’. A second subgroup is focused on interoperability.

The MTAPI™ working group is charged with creating an industry-standard specification for an API that supports the task coordination on embedded parallel systems.

The Tools Infrastructure Working Group's key objective is to improve the interoperability between the different tool solutions for the development of embedded multicore systems, has been predominantly continuing its work on the Common Trace Format (CTF) specification. The goal of this work is seamless correlation of data coming from different types of tracers (e.g. software instrumentation, hardware instruction/data trace) as well as divergent execution contexts on multiple cores or systems.

PUTTING IT TOGETHER

Case for Parallelization and Multi-Core

There is no other way than going to multi-core – practically all modern CPUs have more than one core and the number will increase. However, the (real-time) control system software that runs on embedded processors has been written using single-core machines and while one can introduce operating systems that provide SMP support out of the box, just letting the OS to do the task assignment will cause some surprises in how the system behaves. The timing of tasks will inevitably change if tasks that have been assumed to be sequentially scheduled, are running truly parallel in separate cores.

Having said that, there are several cases where having multiple cores available can bring true advantages.

Data Streaming Application

One of the applications we are working on is a processing system for a low-level RF controller. The basic structure of the system is as shown in Fig. 2. The system collects data from several ADCs, processes them and does feedback of the RF system. The platform to do this is the IFC_1210 board that has recently been developed in collaboration between PSI and IOxOS SA [5]. The board is based on a PCI express communication infrastructure, on which several processing units are connected. The board has a powerful Virtex-6 “Central” FPGA, a dual-core Freescale P2020 QorIQ SoC CPU unit. The board in in VME format, to be compatible with the large existing installation base at PSI, but with an innovative use of the P0 connector allows PCI express links to be routed through the backplane, allowing us to create a multi-board system by connecting several boards together. The VME bus can be used as a control plane interface, but the system is not limited by the bus speed.

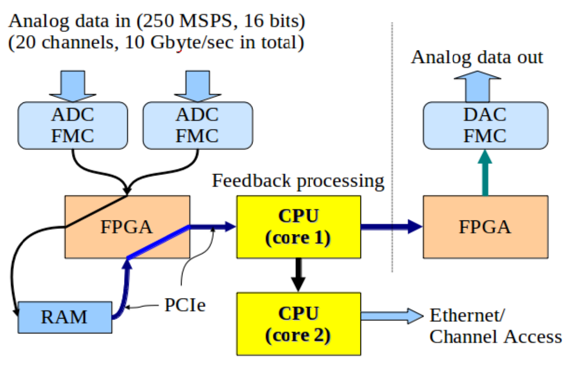

Figure 2. A data streaming and processing application for low-level RF control

The bulk of the raw data processing in this system takes place in the FPGA, however the control algorithms are at least in the first stage planned to be run on the general purpose PowerPC cores, that are much more suitable for interactive development. This makes the prototyping cycle much shorter. FPGAs do not easily lend them for scripting, even with the most modern tools.

The idea here is to take advantage of the multi-core CPU by clear partitioning of the tasks. At least in the development stage one could even take advantage of virtualization in the sense that the algorithms could run on a desktop machine which is connected to the field hardware via a long-distance PCI express link. The plan is to accelerate the development cycle by supporting direct access from modelling tools for developing and testing the feedback algorithms. The preferred tool for this is Matlab/Simulink. We plan to integrate it with the IFC_1201 board that we can directly download code from the tool to the system under test.

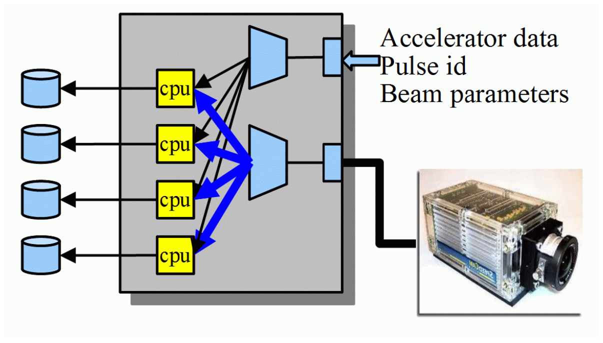

Figure 3. Striping data acquisition

Striping Data Acquisition

One conceivable application could be high-speed acquisition of data, e.g., images from a high-speed sensor to be processed in parallel. When the data cannot be processed fast enough by a single unit, it can be streamed to multiple units and the results stored in parallel (see Fig. 3). Many experiments in physics and X-ray FELs could fit this kind of model where each “event” is in principle an individual unit. If the imaging device would support I/O virtualization, the load could be effectively shared between multiple units. Usually the sensor data needs also to be merged with data from the accelerator (e.g., pulse number, beam characteristics, etc.) to enable later analysis of the data.

Case for Virtualization

For controls applications, configuration management is probably the most compelling reason why one would consider virtualization.

The trend has been to put the computing closer to the equipment. With the fast interconnects and protocols available, one could think of revising the trend. It might be feasible to put the computing infrastructure together in a server room instead of spreading it out on the field. This would have the advantages that the management becomes easier, granularity of allocating processing power to the applications can be improved: not all I/O processors need to be the dimensioned according to the highest requirement, and on the other hand more power can be allocated where it is needed. In a sense this has already been happening since the EPICS software has allowed us to run the IOC core software on a server machine. Moving from the "soft IOC" would involve adding the direct (virtualized) I/O to the system and one could even implement real-time systems on a remote server. Using PCI express as the interface protocol, the infrastructure for device drivers would be easily usable in such an environment.

CONCLUSIONS

The advances in computing sometimes enter the embedded world with a delay. It is also often not obvious how to take advantage of the technologies that have been targeted to a different domain. However, after all the requirements are not that different, and by embracing the technology they can bring a lot of benefits to the embedded applications.

REFERENCES

[1] OMAP Application Processors; http://www.ti.com/omap.
[2] Freescale Corporation, www.freescale.com
[3] I/O Virtualization specifications. http://www.pcisig.com/specifications/iov/.
[4] Multicore association, www.multicore-association.org
[5] IOxOS SA, www.ioxos.ch