World Automation Congress © 2010 TSI Press.

SPACE DIVISION OF PROCESSING POWER FOR FEED FORWARD AND FEED BACK CONTROL IN COMPLEX PRODUCTION AND PACKAGING MACHINERY

STEFAN AUST, HARALD RICHTER
Department of Computer Science
Clausthal University of Technology
Julius-Albert-Str. 4
38678 Clausthal-Zellerfeld, Germany
e-mail: stefan.aust|harald.richter@tu-clausthal.de

ABSTRACT— Since production and packaging machinery with hundreds of sensors and actuators get faster and more complex at the same time, new demands come up to the real-time capacity of the machinery's controller. This article takes an analysis of timing problems caused by the commonly used time-slicing of the CPU as a task scheduling strategy and its effects on the real-time behavior of complex PLC programs. Furthermore, previous studies on reconfigurable FPGA based controller hardware are discussed. In conclusion we suggest a design approach for a scalable multi-processor architecture based on FPGA technology. Thereby the main focus of our design is set to the real-time capacity of the multi-task controller.

Key Words: PLC; PAC; real-time processing; multi-processor; FPGA

1. INTRODUCTION

Since the advent of electro servo motors as distributed drives for complex machinery, the operating speed of the mechanical parts of such machines has increased by 2-3 orders of magnitude. Accordingly, the number and the speed of the machines' sensors and actuators have reached several hundred devices and cycle times that are below the sub-millisecond range. Examples for this development can be found in machines for production as well as for packaging throughout factory automation. All machines get faster and more complex in order to become more efficient with respect to costs. This progress demands more from the machine's hard- and software than a standard Programmable Logical Controller (PLC) or Programmable Automation Controller (PAC) can deliver, resulting in severe problems for the PLC programmer to fulfill real-time constraints. Real time in this context means the due execution of tasks according to a predefined time line, called machine schedule. Beside this schedule, the timely response on concurrently upcoming events is also mandatory for machine control. To fulfill these real-time constraints becomes exponentially more difficult for schedules that are faster and more complex than ever. Manufacturers in automation technology such as Allen Bradley try to catch up with innovative concepts like ControlLogix [1].

This article describes a method for feed forward and feed back control of complex high-speed machines via PLC/PACs under many timing constraints by means of multiple soft-core processors implemented in a Field Programmable Gate Array (FPGA). The objective is to substitute the time division of the resources in a single CPU of a PLC by space division of the FPGA's chip area. A FPGA can accommodate as many soft processors as user tasks must be executed in real time. Our proposal uses the fact that a Virtex-4 FPGA [2] from Xilinx, for example, can host up to 21 MicroBlaze [3] soft processors executing 21 tasks truly in parallel. Most recent FPGAs such as Virtex-6, e.g. can implement already several hundred 32 bit soft processors on one chip, allowing the same amount of user tasks running simultaneously. By this, real time execution of user applications is without time-sharing of tasks in one CPU, thus alleviating the burden of finding a working task schedule. In the next section, the time division multiplex (i.e. time sharing) and scheduling algorithms that are state-of the-art in feed forward and feed back control of contemporary machines are described.

2. STATE-OF-THE-ART

Conventional PLCs use single processor hardware, and it is the task of the PLC programmer to find a working schedule for task execution by means of time-sharing the CPU resources. This schedule must satisfy timing constraints of all PLC tasks as boundary condition: some instructions, such as the switching of an infeed lock, can be performed within a time interval of a few milliseconds. Other instructions, e.g. the visualization of messages to the operator, can be delayed for several hundred milliseconds without any harm. There are three task types that differ in their time strictness and are planned by advanced reservation in the time schedule. All tasks consist of a list of PLC instructions, which are executed sequentially from beginning to end (= PLC cycle). After cycle completion, the processing of the list repeats from the beginning without any stops. Such a preprogrammed execution is called continuous task [4]. Each PLC program contains exactly one continuous task. The second task type runs periodically triggered at fixed points in time. The last type is the hardest to implement: asynchronous triggered events, e.g. triggered by sensors, must be reliably and duly executed concurrently to other tasks. Time or event triggered tasks run once from beginning to the end and stay suspended until the next trigger occurs (figure 1) [4].

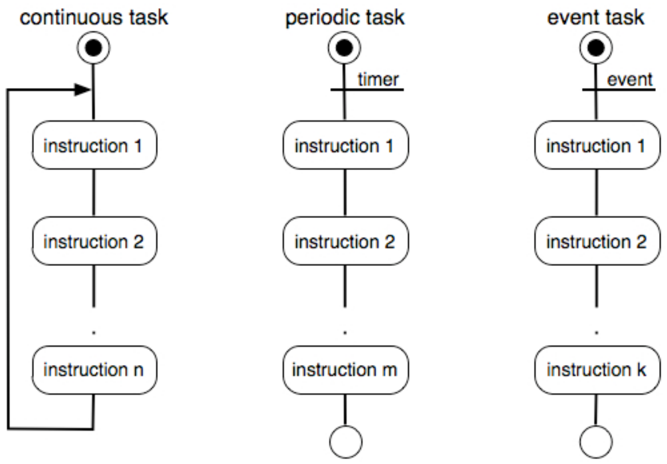

Figure 1: Task cycle model: continuous task, periodic task, event task

2.1 Time Division Multiplex of CPU (Time Sharing)

Since standard PLC hardware operates sequentially, task execution always follows a specific scheduling strategy, such as fixed-priority preemptive scheduling. Fixed-priority scheduling means that the PLC programmer has assigned the priority of the task during configuration phase of the PLC software. Other strategies such as Earliest Deadline First (EDF), Round Robin (RR) or Shortest Job First (SJF) exhibit similar problems.

In fixed-priority preemptive scheduling, the continuous task is allocated the lowest priority. Other tasks get higher priorities related to their timing constraints. If a triggered task is activated, the processing of the low priority task is suspended until the task cycle is completed. Two problems arise with this strategy: It must be guaranteed that the low priority task still meets its real-time requirements despite waiting. Plus it must be ensured that during execution of a high priority task, any other task with the same or higher priority may cause interruption. This results in multiple time-slicing of the CPU's computing power. Therefore, a careful allocation of task priorities by the PLC programmer must be accompanied by a thorough analysis of the worst-case execution times of all tasks that are running concurrently. This is accomplished in state-of-the-art technology via commercially available scheduling analysis tools e.g. SymTA/S [5] that instrument the user software and that count CPU cycles. However, for low quantity or individual production as known from machinery the complex timing analysis is not an efficient solution.

The risk of a low priority task being interrupted in a time critical moment by any high priority task is often circumvented in today's technology, in that way critical parts of the task are delegated to smaller subtasks with higher priority. However, because of this outsourcing of responsibility, inter task communication is needed which is performed via shared variables. The shared variables are called controller tags. This means that variables that were previously task-local (program tags), are turned into global data. As a consequence, additional programming and testing expense is inferred, clarity and portability of the software is decreased.

a)

Task	Trigger	Priority	Data Memory	
Continuous task	n/a	P0	program tags	controller tags
Periodic task	100 ms	P1	program tags	
Periodic task	20 ms	P2	program tags	
Event task	sensor	P3	program tags	

b)

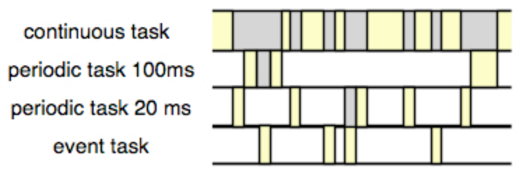

Figure 2: Example of task priorities and the resulting CPU time-sharing

In figure 2a), the allocation of low and high prioritization to tasks is shown in a sample PLC program. In figure 2b), the resulting time division multiplex of the CPU power is depicted with negligence of the OS influence on the schedule. Yellow bars show a running task. Grey areas indicate the duration while a task has been suspended because of interruption by a high priority task. As one can see, the interruptions of the continuous task leads to significant delays for that task. In addition to task delays, task overlaps can occur. A task overlap means that the execution of a time- or event-triggered task will be delayed by one or multiple interrupts of any other high priority task, so that a new start trigger occurs before the previously started execution of the task has been completed. Figure 3 gives an example of the described problem.

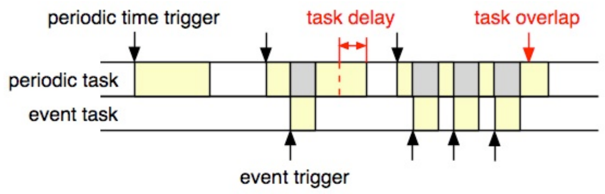

Figure 3: Periodic task delay (left) and task overlap (right)

From a certain number of tasks on, the PLC is no longer able to meet all timing requirements. In these cases, it becomes necessary to distribute the PLC program over multiple controllers residing in the same PLC or over multiple PLCs of the production or packaging machine. However, as a consequence of the task distribution additional inter processor or inter PLC communication has to be established in order to update the values of the shared variables [6]. Due to the inter processor or inter PLC communication and because of the update process, the number of shared variables as well as their value update rate is very limited.

To summarize state-of-the-art technology: In case of a single processor PLC, the strategy of fixed-priority preemptive scheduling means, that only the task with the highest priority has a deterministic schedule. Second to none, this task has a guaranteed response time to sensor signals while running in a task mix. All other tasks may be interrupted arbitrarily. It must be mentioned that other scheduling strategies show similar problems. The skill is to find any schedule that meets all timing constraints.

3. SPACE DIVISION MULTIPLEX OF FPGA (SPACE SHARING)

3.1. Reconfigurable Hardware

For state-of-the-art PLC hardware the total memory capacity is the only software-related feature. All previously described timing requirements have been considered by the real-time operating system and its task scheduling strategy. Reconfigurable computing hardware respecting the PLC program becomes feasible by using FPGA technology. Various studies on reconfigurable PLC hardware were presented during the last years. Most of these studies translate PLC programs directly into hardware description languages (HDL) for FPGA programming. Translations can be found for ladder diagram [7], instruction sequence [8], sequential function chart [9], and finally for language independent Petri net as formal specification of logic [10,11,12]. The advantage of all these is the implementation of program logic that works very fast plus, as required, truly parallel. However, from our view the automatic conversion of complex PLC programs into any synthesizable hardware design seems to be full of risks. Furthermore, every program change results in a new hardware design. This will cause interminable hardware synthesizes.

For complex machinery control we suggest the use of soft-core processors instead. A detailed analysis of reconfigurable computing hardware with regard to hardwired technology vs. software-programmed microprocessors was presented in [13]. Soft-core processors provide the flexibility of FPGA hardware while downloading to memory can easily change program logic. An appropriate design approach is shown in [14]. Since PLC instructions are commonly used, synthesized hardware elements, such as soft-core processors, can be standardized. The risk of synthesis failure can be minimized this way.

3.2. Multiple Soft-Core Processors

Our concept of feed forward and feed back control for complex machinery in factory automation is based on defining multiple soft-core processors on a FPGA. A soft processor is a CPU specification written in hardware description language such as VHDL [15] that is compiled into a set of fuse matrices and loaded onto the chip. Multiple soft processors can coexist on the same FPGA because each of them consumes only a small percent of the chip area. FPGAs allow the flexible creation and allocation of multiple soft processors with respect to the individual PLC program requirements. Since even complex PLC programs consist of not more than 10-20 tasks, it is possible to statically assign a soft processor to each task to avoid scheduling problems otherwise arising from the time division multiplex of computing resources. With space division multiplex, each task can run without any competition on its soft processor. Furthermore, a real-time OS kernel as task support, depending on and configurable by the PLC programmer, can enhance the soft processor. Because every executable task permanently possesses its own processor, real-time multi-tasking can be implemented by hardware exclusive of any scheduling strategy. The time scheduling problem is translated into an available-area-on-chip problem. Therefore, we denote our concept as space division multiplex. The block diagram in figure 4 gives a generic architectural sketch of the concept.

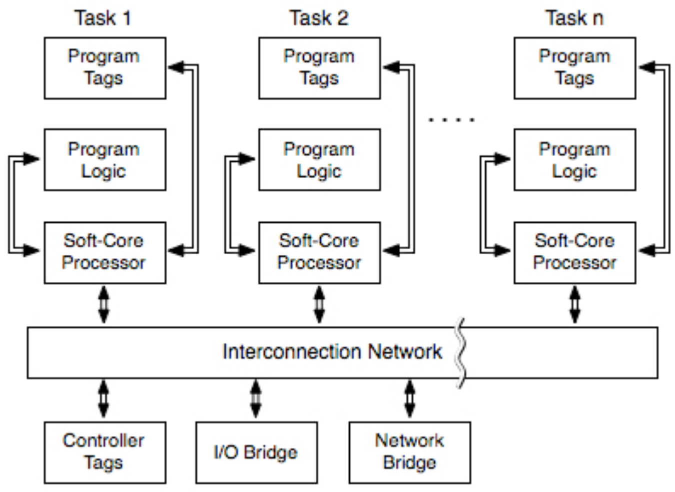

Figure 4: Generic architecture of the space division concept

User instructions and user data are distributed to the processors' local memories. All task-internal variables (program tags) are stored there too. A shared memory is available for global data (controller tags) that can be safely accessed by mutual exclusion, implemented in hardware. Mutex [16] is a standard feature for mutual exclusion in Spartan/Virtex-FPGAs from Xilinx, for example. For scalability reasons, an interconnection network instead of an ordinary bus couples all processors/global modules. Examples for suitable interconnection networks can be found in [17]. The FPGA's capacity and its external RAM also limit the amount and sizes of processor/memory modules. They must be configured in the application software. The proprietary Xilinx MicroBlaze [3] for instance is a processor description with 32-bit word length. For many applications in factory automation the Xilinx Picoblaze [18] with 8-bit word length would be sufficient. It is important to notice that in the description of the soft processor every processor element that would introduce indeterminism in the number of CPU cycles a task needs for completion has to be avoided. This means, due to predictability in strict real-time systems, caches, branch prediction, pipelining and superscalar instruction-level parallelism should not be allowed in the soft processors in order to prevent from read-after-write, write-after-read and write-after-write conflicts and resource conflicts within the processor [19]. As a consequence such soft-core processors as needed for space division multiplex do not provide high computing power. This is especially true in the view that soft-core processors cannot be clocked above a few hundred Mega Hertz, which is rather slow in comparison with full-custom commodity

CPUs from Intel or AMD, for example. But the distribution of the PLC program to multiple processors, which work in parallel instead of time-slicing, compensates the lower clock rate. However, the main characteristic of PLCs and PACs is their real-time capability and not its sheer floating point performance, although both correlate with each other. In factory automation, computing power is more important for image processing than for feed forward or feed back control.

With the use of standardized components for processing hardware the potential risk of problems during hardware synthesis tends to be zero. The shown multi-processor architecture of figure 4 has been tested on Xilinx FPGA type Spartan-3 S1000 with 17,280 logic cells available. Using 8K-block memory for instructions and local data, 35% of the chip is allocated. Figure 5 shows the numbers of logic cells per chip available in the Xilinx FPGA families of Spartan and Virtex. The latest Virtex-7 shows approx. 10x more logic cells than the Virtex-4 chip. It can be expected that the numbers of logic cells per chip will increase as well, so that further soft-core processors will become technically feasible within the next years.

3.3. Coprocessor Unit

FPGA technology gives the ability to implement processor units with instruction sets customized to their application [14,20]. Soft-core processors could be automatically built out of standard elements with regard to the instructions used in the PLC program. This seems to be practicable in future.

Another option is the use of standard microprocessors, such as MicroBlazes, as processor unit. The implementation of customized coprocessors in addition can perfectly set up the instruction set to the needs of a user task. Secondly, coprocessor units made in hardware enhance the computing performance. The performance increase is shown as comparison spatial vs. temporal computation in [21]. The coprocessor unit is an option if the real-time performance of the standard soft-core processor is too slow for a given user task and if it is not possible to distribute the task onto multiple processors. An example where a coprocessor unit makes sense is in PID feed back control for motion control as shown in figure 6. Here, the coprocessor unit can enhance the standard soft processor to accelerate the PLC's performance in proportional, integral and derivative arithmetic. The coprocessor can be linked to the soft processor via asynchronous message passing by means of a bidirectional FIFO memory in order to achieve the widespread mailbox communication. For Xilinx FPGAs for example, two proprietary Fast Simplex Link (FSL) interfaces can be used as mailbox FIFOs [22]. Since PLC programs are mostly repetitive in their software execution, coprocessors are effective in speeding-up the task. Multiple coprocessors for a single PLC CPU that work together in a master/slave method are also possible. In case of MicroBlazes, up to eight coprocessor units can be linked to a microprocessor via FSL. The effectiveness of this method is impressively demonstrated in IBM's cell processor, for instance.

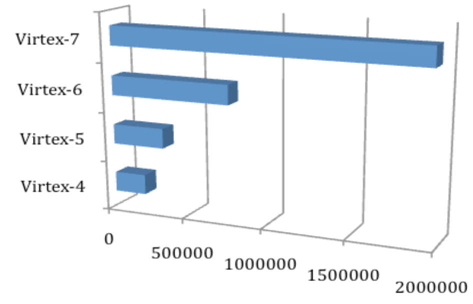

Figure 5: Numbers of logic cells available in Xilinx FPGAs

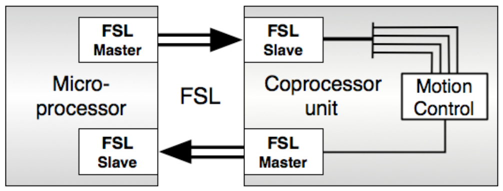

Figure 6: Soft processor with application-specific instruction set as PLC coprocessor

4. CONCLUSION AND FUTURE WORK

The growing complexity of today's machines for factory automation implies hundreds of sensors and actuators, together with significantly reduced cycle times for real-time task execution. This induces new requirements to the PLCs as well as PACs. The currently used single processor architecture with preemptive priority task scheduling in the PLC is not able to meet future needs. Additionally, the distribution of the user application onto multiple CPUs in the PLC or onto multiple PLC in the machine is difficult and confusing and requires additional skills by the PLC programmer. We propose a concept to avoid scheduling problems and to alleviate parallel programming by static allocation of tasks to soft processors on a 1:1 basis. By the use of a multi soft processor architecture, the scheduling problem in time resolves into a problem of available space on the chip. We call this space division multiplex. Furthermore, FPGAs allow flexible soft processor architectures with customized features tailored to the PLC program. In addition, commercial tools can reliably detect the needed number of soft processer execution cycles for a given task because all nondeterministic parts of the CPU such as the branch prediction unit can be skipped in the processor's description. For reasons of scalability an interconnection network will replace the commonly used processor bus. In future, we will develop a software tool that automatically derives the number of needed soft processors with respect to bit length and clock frequency from the user code to support the distribution of the application on the chip's resources.

REFERENCES

1. Rockwell Automation, "ControlLogix System User Manual," July 2008.
2. Xilinx, "Virtex-4 Family Overview," September 2007.
3. Xilinx, "MicroBlaze Processor Reference Guide," October 2009.
4. Rockwell Automation, "Logix5000 Controllers Tasks, Programs, and Routines," July 2008.
5. Symtavision GmbH, www.symtavision.com [checked: 2010-02-12].
6. Rockwell Automation, "Logix5000 Controllers Produced and Consumed Tags," July 2008.
7. I. Miyazawa, T. Nagao, M. Fukagawa, Y. Ito, T. Mizuya, and T. Sekiguchi, "Implementation of ladder diagram for programmable controller using FPGA," in *Proc. 7th IEEE Int'l Conf: Emerging Technologies and Factory Automation (ETFA'99)*, vol. 2, 1999, pp. 1381-1385.
8. Shuichi Ichikawa, Masanori Akinaka, Ryo Ikeda and Hiroshi Yamamoto, "Converting PLC instruction sequence into logic circuit: A preliminary study," in *Industrial Electronics, 2006 IEEE International Symposium*, July 2006, Vol. 4, pp. 2930-2935.
9. M. Ikeshita, Y. Takeda, H. Murakoshi, N. Funakubo, and I. Miyazawa, "An application of FPGA to high-speed programmable controller -development of the conversion program from SFC to Verilog -," in *Proc. 7th IEEE Int'l Conf Emerging Technologies and Factory Automation (ETFA'99)*, vol. 2, 1999, pp. 1386- 1390.
10. P. Nascimento, P. Maciel, M. Lima, R. Sant'ana and A. Filho, "A partial reconfigurable architecture for controllers based on Petri nets," in *Proceedings of the 17th symposium on Integrated circuits and system design*, Pernambuco, Brazil, September 2004, pp. 16-21.
11. M. Adamski and J. L. Monteiro, "From interpreted Petri net specification to reprogrammable logic controller design," in *Proc. IEEE Int'l Symp. Industrial Electronics (ISIE 2000)*, vol. 1, 2000, pp. 13-19.
12. A. Wegrzyn and M. Wegrzyn, "Petri net-based specification, analysis and synthesis of logic controllers," in *Proc. IEEE Int'l Symp. Industrial Electronics (ISIE 2000)*, vol. 1, 2000, pp. 20-26.
13. K. Compton and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," in *ACM Computing Surveys*, Vol. 34, No. 2, June 2002, pp. 171–210.
14. D. Gawali, V.K. Sharma, "FPGA Based Micro-PLC Design Approach," in *International Conference on Advances in Computing, Control, & Telecommunication Technologies, 2009. ACT '09,* Trivandrum, India, December 2009, pp. 660-663.
15. U. Heinkel *et al.*, "The VHDL Reference," John Wiley & Sons, 1st Edition, Chichester, 2000
16. Xilinx, "XPS Mutex (v1.00c)," June 2009.
17. H. Richter, "Verbindungsnetzwerke für parallele und verteilte Systeme," Spektrum Akademischer Verlag, Heidelberg, 1997.
18. Xilinx, "PicoBlaze 8-bit Embedded Microcontroller User Guide," January 2010.
19. A. S. Tanenbaum, "Computerarchitektur," Pearson Studium, 5th Edition, Munich, 2006.
20. M. Xu, F. Ran, Z. Chen, S. Kang and R. Li, "IP Core Design of PLC Microprocessor with Boolean Module," in *Conference on High Density Microsystem Design and Packaging and Component Failure Analysis*, Shanghai, June 2005, pp. 1-5.
21. A. DeHon and J. Wawrzynek, "Reconfigurable Computing: What, Why, and Implications for Design Automation," in *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, 1999, pp. 610-615.
22. Xilinx, "Fast Simplex Link (FSL) Bus v2.11b," June 2009.