Energy-aware MPSoC for Real-time Applications with Space-Sharing, Adaptive and Selective Clocking and Software-first Design

Stefan Aust and Harald Richter
Dept. of Computer Science
Clausthal University of Technology
Clausthal-Zellerfeld, Germany
stefan.aust|harald.richter@tu-clausthal.de

Abstract—**Energy-awareness is an important criterion for many mobile appliances such as (smart)phones and handhelds. It is also indispensable for electronic controller units in cars for example. Unfortunately, low energy consumption and high-computing power exclude each other. With the proposed methods of space-sharing, adaptive and selective clocking and software-first design, both goals can be reached simultaneously. Space-sharing is an alternative to time-sharing for multi-task controllers in real-time systems that significantly simplifies task scheduling. With space-sharing, there is no need for worst-case execution-time analysis. Furthermore, adaptive and selective clocking, together with a software-first design reduce the controller's energy consumption to the absolute minimum. The results described herein were achieved by a set of measurements made at a single-chip multiprocessor system called MPSoC1 that implements space-sharing on one FPGA and by a second system in software-first design called MPSoC2 that implements adaptive and selective clocking.**

Keywords-real-time system; low power; multiprocessor system on chip (MPSoC); worst-case execution time (WCET); space-sharing; adaptive and selective clocking; software-first design.

I. Introduction

A MPSoC is a parallel computer on a single silicon chip that may contain between two and several hundreds of processing units. In the case of up to ten units, we may speak of a multi-core processor, otherwise of a many-core CPU. Typically, a multi-core processor employs an on-chip first-level cache that is shared between all cores for interprocess communication. If more cores than about 10 are present on the same chip, shared-memory can not be used any more because of the memories bandwidth saturation and other communication means have to be used. Many-core CPU have therefore an on-chip static or dynamic interconnection network for interprocess communication that is normally not real-time capable.

Furthermore, multi- and many-core CPUs are usually implemented by a full-custom-design chip. But during the last years, the capabilities of Field Programmable Gate Arrays (FPGAs) have been increased so much that they are in many cases an alternative to full-custom chips. Every state-of-the-art FPGA can accommodate already now hundreds of processing units, i.e., cores, together with a static or dynamic interconnection network. However, the disadvantage of such FPGA solutions is that each core is less powerful as in the full-custom design due to low clockrates and that less memory is available on chip. This can be overcome by a higher number of cores and by software that is coded in the parallel programming style, together with a proper intertask synchronization.

In daily life, there are many mobile appliances that demand high computing power, but have low energy resources only. This creates a contradiction because high computing power normally means high energy consumption as well. Additionally, precious energy must be invested to cool-down these devices. As a consequence, either the usability or the operational time is short. An other fact is that thermal dissipation limits the life expectancy of electronic devices because of aging processes in the semiconductor material. The pn-junctions in the transistors deteriorate with increasing heat exposure. Finally, heat dissipation always means low energy efficiency, which is the opposite of Green IT. Because of these issues, energy-awareness is important for mobile systems and for ECUs in cars as well.

The concept of space-sharing was introduced first by the authors in [1]. Based on space-sharing we suggested in [2] the usage of MPSoCs on a single FPGA to provide high computing power for energy-limited real-time applications. Space-sharing instead of time-sharing eliminates the problem of finding and guaranteeing a proper time schedule for multiple tasks that are needed to meet the prescribed functionality under a given set of time constraints. Furthermore, it allows also a better energy efficiency in embedded systems if combined with adaptive and selective clocking and a so-called software-first design because this will reduce dynamic power dissipation. As a result, space-sharing is able to reduce energy consumption and to produce high computing power.

The paper is organized as follows: In section II, an overview of the state-of-the-art of time-sharing and space-sharing is given. In section III, the architecture of the MPSoC1 that implements space-sharing in one FPGA is presented. In section IV, measurement results of the energy consumption of MPSoC1 are given. In section V, MPSoC2 is presented, together with methods for lower energy consumption. In section VI, MPSoC1 is compared to MPSoC2 with the same user application that illustrates the benefits of the proposed methods. Section VII draws a conclusion of the achieved results and gives an outlook to future work.

II. State-of-the-Art of Time-Sharing and Space-Sharing

Real-time applications demand the time-sliced execution of a sequence of tasks on a single processing unit under the boundary condition of given time limits or at given points in time. A common method to evaluate the soft or hard real-time character of an application is the so-called worst-case execution-time (WCET) analysis. This analysis is made for the processing unit that executes that tasks by calculating how long each task will need to complete in the worst case [3].

In embedded systems with time-sharing operation, delays in task execution can occur that have several reasons, such as 1.) matching the deadlines of all tasks by one processor, which forces concurrency, i.e., competition between tasks, 2.) exchanging intermediate results between tasks because of interprocess communication, and 3.) interruption of one task by an other task of higher priority. In order to avoid intolerable delays in task execution, a suitable scheduling must be found that guarantees desired operation under all circumstances. Several task scheduling strategies have been created to solve this problem. These are mainly priority scheduling, earliest deadline first or round robin [4]. Furthermore, WCET analysis grows exponentially with the number of tasks because the amount of if- then-else branches span up a tree of execution paths, which must be all traversed to find the longest path. Because of the fact that WCET is a NP-complete problem, many commercial and open-source analysis-tools have been developed to ease WCET analysis [5], [6]. With space-sharing, there is no need for task scheduling and thus also not for WCET analysis.

In space-sharing, every task gets its own processing element, which is a core or a whole CPU that is part of an on-chip parallel computer [1]. Additionally, every interrupt service routine and every device driver gets its own processing element as well. Thus it does not happen that one task is interrupted by an other. This means that the number of processing elements must match the cumulative number of all tasks (Fig. 1).

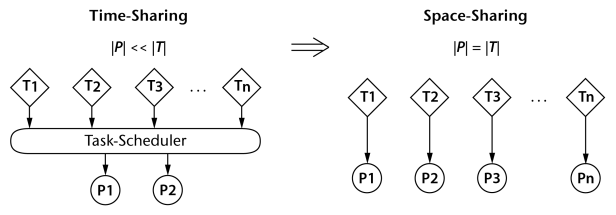

Figure 1. Time-sharing vs. space-sharing.

Furthermore, by this means every task gets its own local memory where it resides with program and data. As a consequence, tasks are never competing for the same resources. However, in case of interprocess communication the execution of a task can still be delayed if the task must wait until a corresponding task has calculated a required intermediate result. This problem is known as task synchronization. It can not be solved by WCET analysis but by proper parallel programming.

The combination of storage and computing element is called processor-memory-module (PMM). Several or many PMMs are coupled by a static or dynamic interconnection network that is on the same chip and real-time capable. Finally, PMMs can be connected to peripheral devices such as sensors, actuators, hard disk, network interface or external memory by the chip's IO pins.

Space-sharing requires that the FPGA has enough resources to accommodate all needed components, and it requires that practical methods exist for allocating tasks to processors, as well as for automatic chip synthesis due to the number and structure of PMMs. It requires also that simple means exist to compile code and to debug it for every PMM, and that intertask communication occurs in real-time.

III. Architecture of MPSoC1

Fig. 2 shows the architecture of MPSoC1 that we have implemented on various FPGAs from Xilinx. MPSoC1 consists of a configurable number of PMMs and memory sizes and a multistage interconnection network that has switches of size 2×2. Each PMM can be directly coupled to an I/O pin or can access peripheral resources via the interconnection network. Because of the spatial isolation of PMMs, local program code and data are protected from unintended overwriting by other tasks. This means that interprocess communication is either possible by passing messages through the network or by shared variables that reside in a global (external) memory. Furthermore, we managed to develop a VHDL program that synthesizes a real-time capable interconnection network of configurable size, together with an arbitrary number of PMMs. Furthermore, there exist simple procedure calls for intertask communication and for clock-rate setting.

The processing elements in our implementation are soft-core processors. For these processors, compilers and debuggers exist, but for each PMM code must be compiled and debugged separately. Open issues are therefore by which software tools programming and debugging of the parallel computer can be accomplished as a whole, how tasks are allocated automatically by a middleware to PMMs, what operating system can be run on the FPGA, and whether the number of logic cells on the FPGA is sufficient to accommodate all tasks. These problems are especially imminent as soon as very complex real-time applications should be solved by space-sharing or as soon as complex device drivers are needed to access peripherals. For example, the

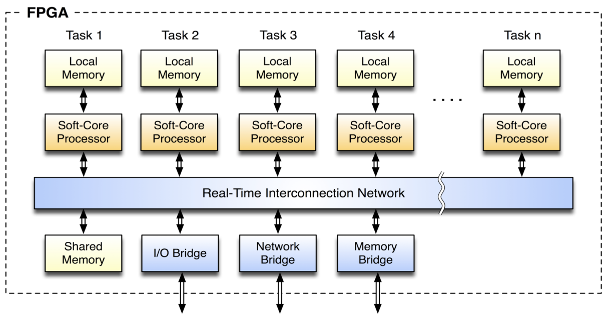

Figure 2. MPSoC architecture on a FPGA.

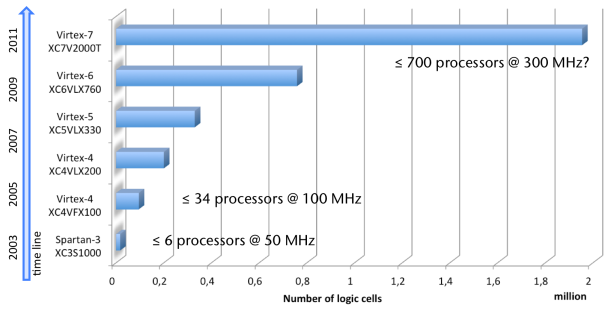

Figure 3. Evolution of Xilinx FPGAs within the last few years.

ECU software in modern cars comprises hundreds of tasks and thousands of intertask communications between ECUs. The questions arising from that issues will be discussed in the next subchapters. It will be shown that space-sharing is a usable alternative to time-sharing for many real-time applications.

A. *FPGA Evolution*

As shown in Fig. 3, FPGAs have advanced significantly with respect to the number of logic cells and internal memory within the last few years, and this trend will continue. For example, a Virtex-4 XC4VFX100 FPGA has less than 5 percent of the capacity of the latest Virtex-7 XC7V2000T FPGA. The same holds for on-chip memory, which has reached up to 70 Mbits per chip. In consequence, a Virtex-7 FPGA from Xilinx for instance is able to host hundreds of PMMs, albeit with a much lower processor clock rate compared to full-custom designed processors and with smaller local memories. However, a clock of 200 MHz and a program and data store of several dozens of KByte is feasible for a Virtex-7 FPGA for about 2-300 PMMs, which is sufficient for many feed forward and feed back control algorithms. Also single-chip controllers have no higher clock frequencies. The reason for that is the necessity for GHz and GBytes in embedded systems is low.

B. *Soft-Core Processors*

A soft-core processor exists as a set of FPGA logic cells that are synthesized via a hardware description language such as VHDL or Verilog. In our tests, we have used a publicly available processor description from Xilinx called MicroBlaze [7]. On a Virtex-5 PFGA, for example, each single MicroBlaze consumes 2-3 % of the chip's logic cells and memory. A MicroBlaze implements an in-order, non-superscalar, 32-bit RISC CPU with a clock rate of up to 200 MHz. Because of its simple RISC architecture, it is possible to predict the CPU cycles needed for a given real-time task more easily than in the case of a fully-featured CPU. MicroBlazes can even be synthesized without caches and branch prediction, which allows to calculate exactly the execution time for every task. Application software can be developed in C or C++ because compilers are existing in the Xilinx EDK toolset, from other vendors and open sources as well.

C. *Interconnection Network*

For interprocessor communication we used a multistage interconnection on-chip network (MINoC), which is based on the Beneš-network and which establishes real time communication paths between PMMs. Messages are transferred in a point-to-point manner or as multicast or broadcast. All three communication types are realized by $(2N \cdot log_2 N - 1)$ 2×2 switches that can transfer data either as a through ("=") or as a crossed ("x") connection or as broadcast from one switch input to both outputs. Each input port to the interconnection network is equipped with a message FIFO to decouple message creation in a PMM from message transfer and delivery in the network. The FIFOs store the messages that are destined for a specific output port as long as that port is occupied. The FIFO depth is small in order not to impair real-time capability that would be caused by messages that are waiting too long in a FIFO. Since the interconnection network works in circuit switching mode, a direct path through the network from sender to receiver is established for every communication type as long as the communication persists. The network can connect every input with every output at any time, as long as no two inputs want to be transferred simultaneously to the same output port. Such a situation is considered as bad task synchronization.

The most important feature of the interconnection network is its non-blocking character, which is a consequence of its topology. This feature comes from the fact that alternative paths can be switched through the network during run-time. The network routing algorithm was developed by one of the authors [8], [9] and [10]. Later the routing was improved by the other author [11]. The improved routing algorithm is able route N paths from inputs to outputs within one clock cycle [12] by means of combinatorial logic that is implemented in AND, OR and NOT gates. The non-blocking character of the network is mandatory for real-time communication. Otherwise connection requests would be delayed because of network-internal conflicts, and no upper time limit could be guaranteed for message latency.

D. Design Methodology and Operating System

In controllers with time-sharing, tasks are sequenced by a task scheduler that is part of a real-time operating system (RTOS). The RTOS provides additionally for device drivers, memory protection, interrupt handling and interprocess communication. However, most of these RTOS functions are obsolete in space-sharing. The only task per PMM can be executed in stand-alone mode if some prerequisites are fulfilled. These are: 1.) a communication library is provided that implements point-to-point, multicast and broadcast for message passing via the interconnection network, 2.) semaphores for shared memory access via a global memory are existing. 3.) Reading and writing sensors and actuators by the PMM is performed via specific device driver tasks in dedicated PMMs. The latter is possible as long as no complex devices such as graphic, hard disk or network controllers must be read and written. In that case, either a RTOS kernel must be employed additionally that has stripped-off all unnecessary features because of memory limitations in the PMM, or external memory that is on the FPGA board must be engaged.

The design methodology for application code in space-sharing is similar to that of parallel programming: first the application must be partitioned into several tasks. Then the needed interprocess communication must be defined, and code and communication libraries must be bounded together. Finally, the parallel program must be tested. In contrast to parallel programming, the number of PMMs, the structure of every PMM and the interconnection network must by synthesized for the target FPGA in space-sharing before program test. Furthermore, space-sharing allows to resize the local memory to the requirements of each task. Since soft processors are used, the processor architecture can be adapted to software requirements as well, for example by an additional coprocessor as hardware accelerator. Xilinx EDK allows to configure in detail every MicroBlaze as needed. Because of that, computing hardware depends on the application software, which we call software-first design.

E. Energy-awareness in Space-Sharing

According to [13], the total power consumption P of a semiconductor chip consists of static power dissipation P_{stat} and dynamic power dissipation P_{dyn} and it must hold:

$$P = \sum P_{stat} + \sum P_{dyn} \tag{1}$$

In the following, it will be discussed how the dynamic fraction of the FPGA power consumption can be reduced. The static power cannot be altered by chip users because it is caused by leakage currents inside of the semiconductor [14]. Dynamic power dissipation, however, arises from switching activities of transistors. Equation 2 determines the dynamic power dissipation as a function of supply voltage V, clock frequency f and chip capacitance C [15].

$$P_{dyn} = \sum C \cdot V^2 \cdot f \tag{2}$$

According to Eq. 2, there are three options for reducing dynamic power dissipation:

- lower switching capacitance C
- lower supply voltage V
- lower switching frequency f

The switching capacitance depends on the production process of the chip and cannot be controlled by FPGA users. The supply voltage can be controlled by an adaptive power supply for the whole chip. However, the switching frequency can be controlled by an adaptive clock for every PMM. Such clocks can be implemented by a central clock generator for the chip and by individual clock dividers at every PMM. If the divided output clock can be disabled then the PMM can also be stopped and restarted arbitrarily because no DRAM cells are inside of a FPGA that must be refreshed. Furthermore, only that area on the FPGA-chip must be clocked at all that is needed for space-sharing. The rest of the chip will dissipate only static power, which is much less. Finally, because of the fact that every PMM has its own clock the PMM's dynamic power dissipation can statically be reduced until the clock has reached the lowest possible periodicity that the user application allows. Additionally, dynamic power dissipation can be reduced adaptively if the clock divider is controlled by the PMM's task. Phases with high computing requirements are clocked faster by the task itself than phases with low requirements or with slow reaction times. The task knows when these phases occur because it is programmed be the user.

Finally, as long as the PMM waits for input from a peripheral device the clock can be switched off totally and restarted again by that peripheral when data are delivered. Such power saving potential is not available for embedded systems with time-sharing because their clock rates do not depend on the tasks. It would be too risky for high-speed tasks if the clock rate would be decreased by time-shared low speed tasks. Furthermore, if the clock is switched-off in time-sharing systems all tasks must rest for ever because an individual switch-off and easy hardware restart by a peripheral is not possible. Thus power dissipation remains in general the same in time-sharing systems unless the supply voltage is reduced. The regulating of the supply voltage is practiced in every laptop, for example. However, space-sharing can adopt voltage regulation as well. In Fig. 4, individual clocking is shown together with voltage regulation.

IV. Measurement Results of the Energy Consumption of MPSoC1

A. Test Setup

During our tests, we used three commercial FPGA evaluation boards with Xilinx Spartan-3, Virtex-4 and Virtex-5 FPGAs, which are listed in Table I. All boards are equipped

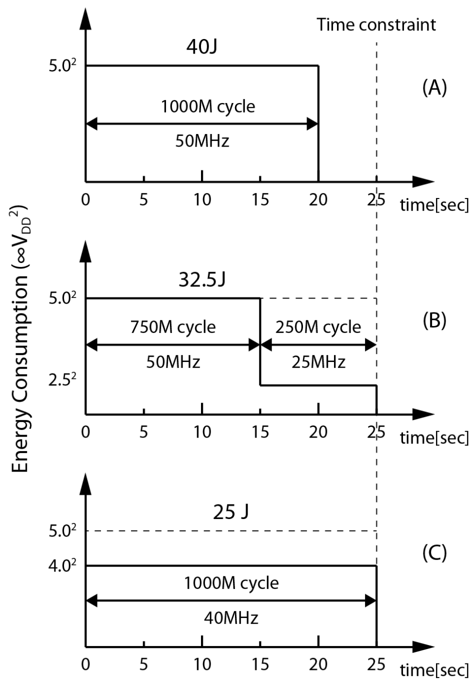

Figure 4. Motivational Example [16].

Table I
LIST OF TESTED FPGAS

FPGA	FPGA	
	Chip	*Board*
Spartan-3	Xilinx XC3S1000	Digilent Starter-Kit
Virtex-4	Xilinx XC4VFX100	PLDA XpressFX
Virtex-5	Xilinx XC5VLX50T	Xilinx ML505

with DIP switches that allow to control a clock divider for the FPGA. This enabled us to measure at different clock speeds without re-synthesizing the FPGA, which is important because a new synthesis would influence the result of the measurements as soon as components were placed and routed differently (cf. IV-F). In addition to that, the system clock had to be switched-off to measure static power dissipation, which was also possible by these DIP switches.

All boards have an external power supply where we could connect our test circuit as shown in Fig. 5. We used for all measurements the same adjustable power supply to avoid inaccuracies, together with two multimeters for voltage and current measurements. Each time we measured with and without clocking to separate dynamic from static power dissipation (cf. Eq. 1) .

B. Static vs. Dynamic Power Dissipation

On a Virtex-4 XC4VFX100 FPGA we could accommodate between 1 and 34 PMMs, consisting of a standard soft processors of type MicroBlaze and 16 KB memory each. The chip is manufactured in 90 nm gate size. At first all

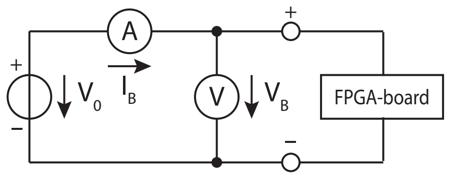

Figure 5. Test circuit for power measurements

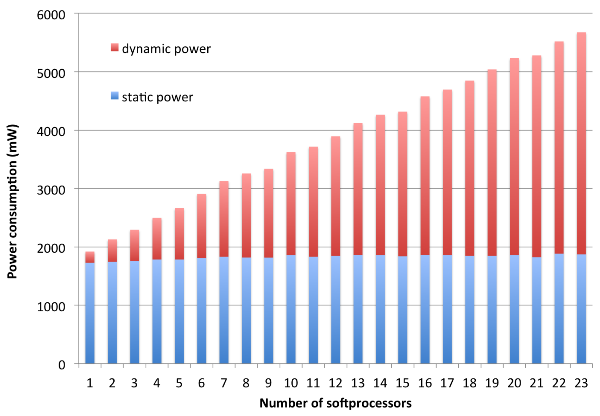

Figure 6. Static vs. dynamic power consumption for various numbers of soft processors, measured on Xilinx Virtex-4 FPGA.

processors were clocked with 100 MHz, while the total power consumption of the FPGA was measured. After this, the system clock was disconnected to measure the static power consumption only. Dynamic power dissipation was obtained as the difference of both measurements. The results of up to 23 soft processors are shown in Fig. 6, where blue columns indicate static power dissipation and red columns indicate dynamic power dissipation.

C. Influence of the Number of PMMs

Fig. 6 shows that static power dissipation remains constant while dynamic power dissipation increases linearly with the number of PMMs. Deviations from a straight line are caused by the place-and-route function of the synthesis tool, which was Xilinx XST [17]. Fig. 6 indicates also that for more than 12 PMMs dynamic power dissipation dominates. Other FPGAs show the same principal behavior as it can be seen from 7.

The absolute numbers we have measured in Fig. 7 are: a Spartan-3 PMM consumes 268 mW of dynamic power, a Virtex-4 PMM consumes 120 mW, and a Virtex-5 PMM consumes 53 mW.

D. Influence of the Processor Clock Rate

Another measurement series was conducted to get the dynamic power dissipation versus the processor clock rate. Various processor clock rates were investigated by using a

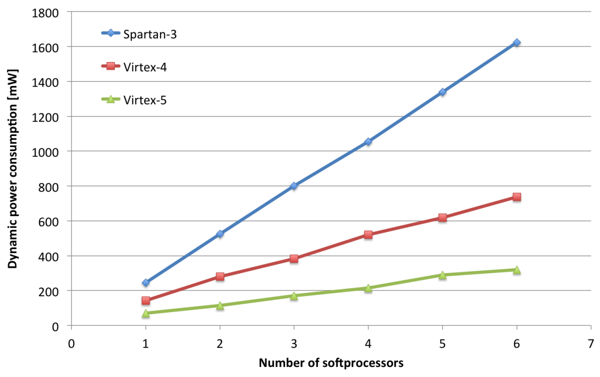

Figure 7. Dynamic power consumption for various numbers of soft processors, measured on Xilinx Spartan-3, Virtex-4, and Virtex-5 FPGAs.

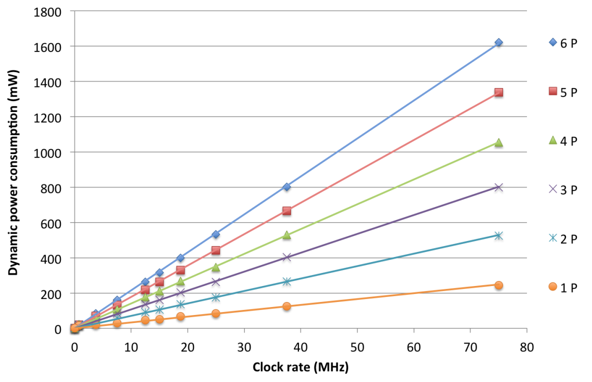

Figure 8. Dynamic power consumption vs. processor clock rate measured on Spartan-3 FPGA. The number of processors is parameterized.

clock divider while the number of PMMs was kept constant. In Fig. 8, the results of MPSoCs with 1 to 6 processors in a Spartan-3 FPGA are shown. In Fig. 9, the results of MPSoCs with 1 to 8 processors in a Virtex-5 FPGA are shown. All results comply with Eq. 2. Both FPGAs show a linear curve from which one can derive also the average dynamic power dissipation in absolute numbers according to (Eq. 3 and Eq. 4).
as:
Spartan-3 FPGA:

$$P_{dyn} = 3.64\frac{mW}{MHz} \tag{3}$$

Virtex-5 FPGA:

$$P_{dyn} = 0.48\frac{mW}{MHz} \tag{4}$$

E. Influence of the Manufacturing Technology

In all measurements of Fig. 7, the same soft processor architecture plus the same size of local memory were used. Nevertheless, the dynamic power dissipation varies with different FPGAs. As one can see, the decrease of dynamic power consumption from one FPGA generation to the next is caused by smaller transistor sizes (65 nm instead of 90 nm) and other technological improvements.

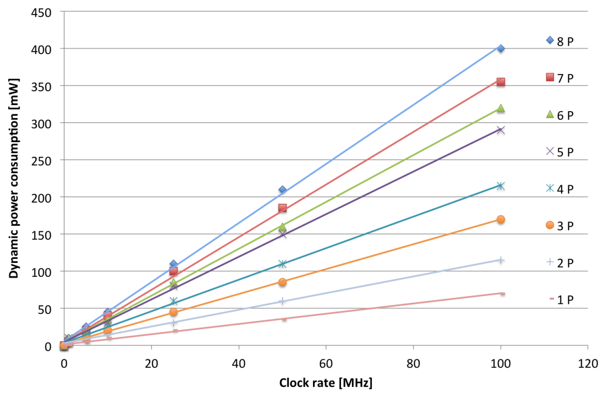

Figure 9. Dynamic power consumption vs. processor clock rate measured on Virtex-5 FPGA. The number of processors is parameterized.

F. Influence of the Place-and-Route Tool

The place-and-route tool is responsible to find space on the chip for all components and to connect them. The length of FPGA-internal connections and their switching capacitance vary from layout to layout. Thus place-and-route tools influence the dynamic power dissipation of FPGA-based systems. An analysis of this influence was made for example by Coxon in [18]. He showed that the dynamic power dissipation can be reduced up to 14% in Spartan-3, up to 11% in Virtex-4 and up to 12% in Virtex-5 FPGAs by optimizing the place-and-route process via user intervention that was made by synthesis directives. In Fig. 9, it can be seen that the distance between neighbor curves is not a constant for all curves. For example, the MPSoC with 5 processors consumes a little bit more power than expected. We explain this by the influence of the place-and-route tool.

G. Influence of the Processor Structure

We investigated the influence of the processor structure on the dynamic power dissipation by means of a Spartan-3 FPGA that was operated a clock rate of 50 MHz. The internal structure of a MicroBlaze can be configured in the Xilinx EDK toolset. The result is shown in Table II. A MicroBlaze that has a simple structure with few internal components only dissipates 182 mW dynamic power. Additionally 107 mW are used for a 5-stage pipeline and 119 mW for a floating point unit, for example.

Fig. 10 shows the dynamic power dissipation of a fully-equipped MicroBlaze. The percentage every component contributes in that processor to the total power is listed in Table II. From this table it can be seen that it pays out a lot to remove unused processor components, what in space-sharing depends of the application software. This result is

Table II
DYNAMIC POWER CONSUMPTION OF SOFT PROCESSOR COMPONENTS MEASURED ON SPARTAN-3 FPGA

Processor Setup	Dynamic Power Dissipation
basic	182 mW
+ 5-stage pipeline	+ 107 mW
+ barrel shifter	+ 13 mW
+ 32-bit multiplier	+ 11 mW
+ integer divider	+ 15 mW
+ floating point unit	+ 119 mW

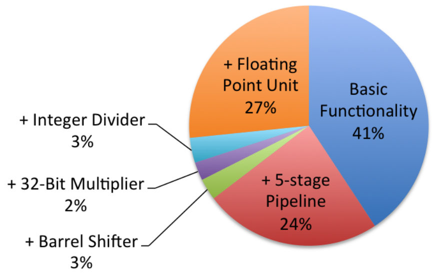

Figure 10. Percentage of dynamic power consumption of processor components in a fully equipped MicroBlaze soft processor as measured on Spartan-3 FPGA.

fully compliant with the software-first design methodology that was mentioned before.

H. Influence of the Local Memory

In order to investigate the influence of the local memory on dynamic power dissipation we measured several MPSoCs with different local memory sizes on a Virtex-4 FPGA at a clock rate of 100 MHz. The number of processors was parameterized. Each PMM executed the same test software that accessed local memory so that it was used by switching bits. Fig. 11 shows the result.

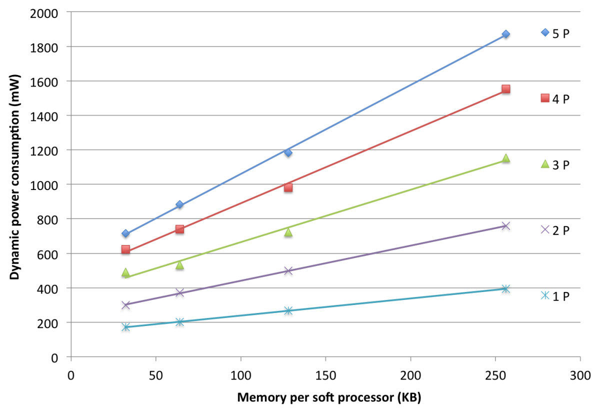

Figure 11. Dynamic power consumption vs. size of local memory measured on Virtex-4 FPGA. The number of processors is parameterized.

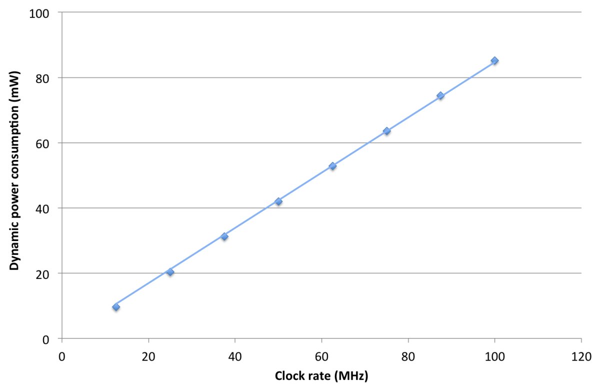

Figure 12. Dynamic power consumption of the MPSoC interconnection network vs. its clock rate measured on Virtex-4 FPGA.

One can also see in Fig. 11 by extrapolating the measurement curve to zero is that a PMM that has no memory consumes about 100 mW. Furthermore, the slope of the curve is about 1 mW per KB of memory. By combining the results from section IV-D with Fig. 11 we can establish to following empiric equation for the Virtex-4 FPGA:

$$P_{dyn} = 1\frac{mW}{MHz} + 0.01\frac{mW}{MHz \cdot KB} \tag{5}$$

I. Influence of the Interconnection Network

In Fig. 12, the influence of the MPSoC network clock rate on the dynamic power dissipation is presented for a Virtex-4 FPGA. The diagram shows that the network's dissipation increases linearly with a slope of about 0.85 mW per MHz. Furthermore, we found that most of the dynamic dissipation mainly arises from the FIFO buffers that decouple processors at the network interfaces for asynchronous interprocessor communication. The network itself is very power efficient because its principle of circuit switching, i.e., signal transfer without buffering. In consequence, the less the FIFO buffer depth is the less dynamic power is consumed. However, the FIFO depth can not be chosen arbitrarily small because it depends on the number of messages that have to be temporarily stored, their size and how big the difference between processor clock rate and network clock rate is. A big speed difference requires a deep FIFO to balance-out message sending and transferring, at least for a while.

J. Power Consumption Constants

Overall, from the accomplished measurements we got the following numeric constants for dynamic power dissipation of a MPSoC that was implemented on a Virtex-4:

$$P_{processor} \approx 1\frac{mW}{MHz} \tag{6}$$

$$P_{memory} \approx 0.01\frac{mW}{KB \cdot MHz} \tag{7}$$

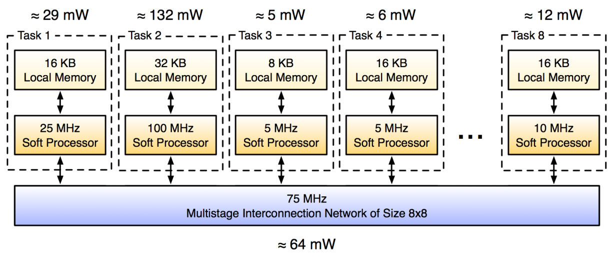

Figure 13. Dynamic power consumption of a MPSoC2 on Virtex-4 FPGA.

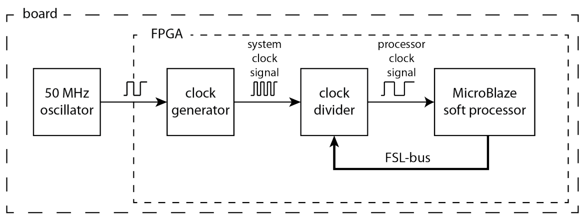

Figure 14. Set-up of the clock rate controller.

$$P_{network} \approx 0.85 \frac{mW}{MHz} \tag{8}$$

V. MPSoC2 WITH SELECTIVE AND ADAPTIVE CLOCKING AND SOFTWARE-FIRST DESIGN

Space-sharing allows selective and adaptive clocking of every PMM by means of a clock rate controller for every PMM. Furthermore, the software-first design method prescribes to configure each PMM such that it matches exactly the requirements of the task it has to execute. To test the influence of both methods, we defined MPSoC2, which has eight PMMs that execute eight different example tasks in their memories. Memory sizes and clock frequencies were chosen as needed by the tasks. In Fig. 13, the resulting MPSoC2 is depicted.

A. Clock Rate Controller

Fig. 14 shows the block diagram of the clock rate controllers of MPSoC2. Every controller derives its input from a system clock generator, which is global for the FPGA and and generates its output by a clock divider. The divider can be set either to a constant rate or, as depicted in Fig. 14, can be controlled by the application software during program execution. In second case the clock divider is coupled to the soft processor via Xilinx FSL-bus. Thereby the clock rate can be set just by sending the new value, which is done within two clock cycles at least.

B. Task Segmentation

To benefit from individual clocking, the application programmer must set the clock frequency for every task explicitly. If the task has several phases with different time constraints then he can set clock rates that are adapted to each phase. To accomplish this, it is required to segment the task into time intervals in which the same rate holds as it is shown in Fig. 15. After that task segmentation, the programmer can define the clock rates by two ways: either he uses a procedure call, which is executed during runtime, or he uses a compiler directive, which is evaluated during compile time. Both methods augment the original code. Other examples of code augmentation by time constraints that can be found in literature are given in [13].

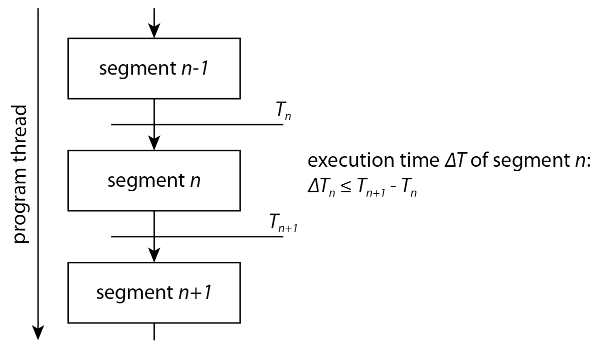
program thread
segment n-1
T_n
segment n
execution time ΔT of segment n:
$\Delta T_n \le T_{n+1} - T_n$
T_{n+1}
segment n+1

Figure 15. Clock rate adaptation by task segmenting.

Fig. 16 shows an example code where a library procedure *set_clock_rate* is responsible for clock rate setting. The application itself *check_new_data* periodically checks a sensor for the arrival of new data. This may be accomplished at a slow sampling rate of 2 MHz, for example. If new data are present, the application *calc_value* is executed at a higher rate of 40 MHz, for example, in order to process the incoming data quickly. By this method, the processor clock varies dynamically during program execution. Thus the dynamic power dissipation rises and falls with the computing requirements.

The disadvantage of this method is that the programmer must identify clock change points in his code and must specify their value manually. Moreover, the clock rates depend on the used PMM because a superscalar processor, which is more powerful would execute more instructions in the same time. This means that clock rate settings have to take into account the concrete PMM on which the task is executed, what makes applications difficult to port between PMMs.

C. Compiler directive for Clock Rate Adaptation

A better method for adaptive clocking is adding a time ruler to the code according to Fig. 15. Each tick in the time ruler T_i defines a point in time until a program phase

```
set_clock_rate(2);                          | => clock rate = 2 MHz
while(new_data = 0){                        |
    new_data = check_new_data(&data);       |
};                                          |
set_clock_rate(40);                         | => clock rate = 40 MHz
value = calc_value(&data)                   |
set_clock_rate(2);                          | => clock rate = 2 MHz
```

Figure 16. System call for setting processor clock rate.

must be completed. The adding of time values as ticks is accomplished manually by a compiler directive. The directives are formatted as a comment with a following $ symbol, for example, to avoid confusion with language extensions or with system calls but they are not treated as comments or extensions. Instead, only a compiler pre-processor reads all meaningful comments and evaluates them. Furthermore, the compiler knows best the PMM type it has to generate code for. This allows for an automatic calculation of the desired clock rate for every program phase. The calculation is performed by the pre-processor in accordance to the respective execution speed of the PMM. The pre-processor counts the number of clock cycles needed to execute every phase due to the time ruler. With that information, the pre-processor can then set the clock rate for every phase, and the programmer does not have to care about the concrete PMMs computation speed, as long as it is fast enough.

```
/*$ initiate */                             |
while(new_data = 0){                        |
    new_data = check_new_data(&data);       |
};                                          |
/*$ event ≤ initiate + 20 */                | => 40 clock cycles / 20µs = 2 MHz
value = calc_value(&data)                   |
/*$ terminate ≤ event + 30 */               | => 1200 instructions / 30µs = 40 MHz
```

Figure 17. Compiler directive for setting processor clock rate.

Fig. 17 shows the example of Fig. 16 with the additional time ruler.

VI. Comparison of MPSoC1 with MPSoC2 under the same User Application

In the following, we compare the dynamic power consumption of MPSoC1 with that of MPSoC2 in order to explore the effect of adaptive and selective clocking, together with the effect of the software-first design methodology. For a fair comparison, MPSoC1 and MPSoC2 got the same eight tasks to execute, and we measured the total dynamic power dissipation during their execution. MPSoC1 is based on a Virtex-4 FPGA with a fixed clock rate of 100 MHz for all system components and a static memory size of 128 KB for all eight PMMs. MPSoC2 is based on the same FPGA but dependent of its task requirements with clock rates of 25, 100, 5, 5 and 10 MHz and with memory sizes of 16, 32, 8, 16, 16 KB. Our measurements at MPSoC1 showed that the total dynamic power consumption is:

$$P_{dyn} \approx 1909mW \tag{9}$$

comprising of

$$P_{processor} \approx 800mW \tag{10}$$

$$P_{memory} \approx 1024mW \tag{11}$$

$$P_{network} \approx 85mW \tag{12}$$

This means that on average every PMM has a dynamic power dissipation of 228 mW. In contrast to that is the example of MPSoC2 (cf. Fig. 13) that produced the same results within the same time limits as MPSoC1. However, it consumed only a total of 184 mW, resulting in an average of 23 mW per PMM. Many other examples can be found that show the same trend. According to that it can be stated that adaptive and selective clocking together with software-first design are efficient methods to reduce the system's energy consumption in space-sharing.

VII. Conclusion and Outlook to Future Work

In this paper, the methods of space-sharing, adaptive and selective clocking and software-first design were proposed and demonstrated in two example multiprocessor systems that resided on a single FPGA. The combination of these methods into a methodology allows to get at the same time high computing power and low electric power dissipation. This new result is valid for the class of embedded real-time systems because smaller tasks have to be executed there. Nevertheless, complex feed forward and feed back control systems can be established that way, comprising hundreds of tasks with interprocess communication. All tasks are executed in parallel on by processor-memory modules (PMMs) that are connected via a special multistage interconnection network that is real-time capable. Interprocess communication is either possible by passing messages through the network or by shared variables that reside in a global (external) memory.

Space-sharing means that exactly so many PMMs are synthesized on one FPGA as there are user tasks, including all interrupt service routines and device drivers that are needed by the user application. Adaptive and selective clocking means that every PMM is clocked individually and with a rate that matches the needs of the task it executes, even if these needs vary from task phase to task phase. Software-first design means that the PMM is configured in its architecture and in its memory size such that it meets exactly the task requirements, thus avoiding waste of chip and energy resources.

As a result, space-sharing eliminates the need to find a proper schedule for a set of tasks that must be executed until a given time interval or at a given time point. It also eliminates the analysis of worst-case execution time (WCET) in embedded controllers, which can be very complex. Additionally, space-sharing allows for a better electrical power management and for memory protection because of the spatial isolation of tasks. Finally, dynamic power dissipation can be reduced in space-sharing by means of adaptive and selective clocking and by a software-first design to the absolute minimum. This is applicable, e.g., in car electronics, where an increasing amount of real-time tasks has to be proceed at a minimum of energy use.

Future work will create a tool set that automatically synthesizes a FPGA-based MPSoC from a XML description of tasks and from a second description of the tasks' memory and time constraints. The resulting embedded controller will be easily synthesize-able because it consists only of small and individual processor-memory-modules. It will though exhibit high computing power and low energy dissipation at the same time.

REFERENCES

[1] S. Aust and H. Richter, *Space Division of Processing Power For Feed Forward and Feed Back Control in Complex Production and Packaging Machinery*, Proc. World Automation Congress (WAC 2010), Kobe, Japan, Sept. 2010, pp. 1-6.

[2] S. Aust and H. Richter, *Energy-Aware MPSoC with Space-Sharing for Real-Time Applications*, The 5th International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2011), Lisbon, Portugal, Nov. 2011, pp. 54-59.

[3] P. Marwedel, *Embedded System Design*, 2nd edition, Dordrecht; Heidelberg: Springer, 2011.

[4] G. C. Buttazzo, *Hard Real-Time Computing Systems. Predictable Scheduling, Algorithms and Applications*, Boston; Dordrecht; London: Kluwer Academic Publishers, 1997.

[5] Rapita Systems Ltd., www.rapitasystems.com (last checked: 11-06-20).

[6] Symtavision GmbH, www.symtavision.com (last checked: 11-06-20).

[7] Xilinx Inc., *MicroBlaze Processor Reference Guide*, October 2009.

[8] H. Richter, *MULTITOP - Ein Multiprozessor mit dynamisch variabler Topologie*, Dissertation, Fakultaet fuer Elektrotechnik und Informationstechnik der TU Muenchen, 1988, (in German).

[9] H. Richter, *MULTITOP - A multiprocessor with dynamic variable topology (English Summary)*, IPP Technical Report R/35, Max-Planck-Institut fuer Plasmaphysik, 1988.

[10] H. Richter, *Interconnecting Network*, US-Patent Nr. 5,175,539, 1992.

[11] S. Aust and H. Richter, *Real-time Processor Interconnection Network for FPGA-based Multiprocessor System-on-Chip (MPSoC)*, The 4th International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2010), Florence, Italy, Oct. 2010, pp. 47-52.

[12] S. Aust and H. Richter, *Skalierbare Rechensysteme fuer Echtzeitanwendungen*, in: W. A. Halang (editor): Herausforderungen durch Echtzeitbetrieb, Springer, 2011, pp. 111-120, (in German).

[13] A. Leung, K. V. Palem, and A. Pnueli, *TimeC: A Time Constraint Language for ILP Processor Compilation*, Constraints, vol. 7, no. 2, 2002, pp. 75-115, doi: 10.1023/a:1015131814255.

[14] N. S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, K. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, *Leakage Current: Moore's Law Meets Static Power*, IEEE Computer, vol. 36, issue 12, Dec 2003, pp. 68-75.

[15] L. Shang, A. S. Kaviani, and K. Bathala, *Dynamic Power Consumption in VirtexTM-II FPGA Family*, Proc. of the 2002 ACM/SIGDA tenth international symposium on Field- programmable gate arrays (FPGA '02), Monterey, CA, Feb. 2002, pp. 157-164.

[16] H. Yasuura, T. Ishihara, and M. Muroyama, *Energy Management Techniques for SoC Design*, Essential Issues in SoC Design, Springer, 2006, pp. 177-223.

[17] Xilinx Inc., *XST User Guide for Virtex-4, Virtex-5, Spartan-3, and Newer CPLD Devices*, Xilinx document no.UG627, December 2010.

[18] A. Coxon, *FPGAs auf Low Power trimmen*, elektronik industrie, Huethig, issue 1/2, 2009, (in German).