An Energy-Efficient FPGA-Based Packet Processing Framework*

Dániel Horváth[1], Imre Bertalan[1], István Moldován[1], and Tuan Anh Trinh[2]

Budapest University of Technology and Economics
[1] Inter-University Cooperative Research Centre for Telecommunications and Informatics
[2] Department of Telecommunications and Mediainformatics
Magyar Tudósok krt. 2, 1117 Budapest, Hungary
{horvathd,emerybim,moldovan,trinh}@tmit.bme.hu

Abstract. Modern packet processing hardware (e.g. IPv6-supported routers) demands high processing power, while it also should be power-efficient. In this paper we present an architecture for high-speed packet processing with a hierarchical chip-level power management that minimizes the energy consumption of the system. In particular, we present a modeling framework that provides an easy way to create new networking applications on an FPGA based board. The development environment consists of a modeling environment, where the new application is modeled in SystemC. Furthermore, our power management is modeled and tested against different traffic loads through extensive simulation analysis. Our results show that our proposed solution can help to reduce the energy consumption significantly in a wide range of traffic scenarios.

Keywords: energy management, packet processing.

1 Introduction

The future communication systems must meet several, often conflicting requirements. Such requirements are the ever-growing processing power while keeping the energy consumption down. Scalability is required without increasing the complexity. Flexibility of the software at the speed of the hardware is a requirement as well. And of course all of this at low price.

Requirements for the increased complexity of hardware design lead to a way of designing chips, where Composability, Predictability and Dependability need to be taken into account. Composability enables the development of stable software for very large systems by ensuring that parts of the software can be developed and verified separately. Predictability is the ensuring of real-time requirements on a device. Tools that calculate the timing behavior enable the correct design and scaling of systems need to be used to ensure that the real-time conditions are met. Last but not

* The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement n° 100029 and from the Hungarian National Office for Research and Technology (NKTH).

F.A. Aagesen and S.J. Knapskog (Eds.): EUNICE 2010, LNCS 6164, pp. 31–40, 2010.

least, dependability relates to the behavior of the system in case of errors and problems during operation.

FPGAs are used in the prototyping of the devices but as the price of larger FPGAs become accessible, they are considered as a design alternative. For special functions that are not deployed in large quantities in the network, and which require very fast processing such as network monitoring applications, firewalls etc. an FPGA based design provides flexibility and the required performance at reasonable price.

Power consumption is also of major concern in future packet processing architectures. We address this topic with a hierarchical power management that minimizes the energy consumption at chip level.

The aim of our work is to provide a toolchain which accelerates the development process of network applications on FPGA. The toolchain will provide a set of hardware modules which builds up a variety of network devices. Key use-cases are switching, routing devices, a NAT device, a firewall/deep packet inspector, and a traffic loopback device.

For design space exploration and to validate the design, a SystemC [5] based modeling environment is used. The results of the SystemC modeling can be used to construct the final hardware models and the corresponding software. The available hardware components will also be available in generic hardware description language (Verilog/VHDL) making possible the synthesis of the hardware. The toolchain will also provide the top-level hardware model. The modules required for the generic networking applications have been selected by identifying the most important use cases.

This paper is structured as follows. Section 2 we describe the generic architecture and its power efficiency considerations are detailed in Section 3. The simulation and results obtained from it is shown in Section 4. Section 5 concludes the paper.

2 Packet Processing Architecture

The development of the new applications is also assisted by a number of available hardware component models that can be used for generic networking application development. The already available components define a generic packet processing/forwarding mechanism with extensible filtering and processing properties.

Networking applications can be composed from predefined modules and custom modules. Example predefined modules are e.g. MAC-layer input and output, switching fabric, and packet scheduler. The behavior of the network devices can be further customized by adding custom modules that apply application-specific operations at different stages of packet processing. For easy integration well defined interfaces are used.

The packet processing framework provides the background for new application development. Its extensible, modular structure is designed to allow easy integration of application-specific header operations at the ingress and egress. A method for buffering the packet payload is also provided.

2.1 Background

In the literature, we have found similar work dealing with packet processing on FPGA-based systems. Notably, we would mention the work of D. Antos et al. [1] on

the design of lookup machine of a hardware router for IPv6 and IPv4 packet routing with operations are performed by FPGA, since FPGA-based implementations are an attractive option for implementing embedded applications. In this framework, part of the packet switching functionality is moved into the hardware accelerator, step by step. This allows keeping the complete functionality all the time, only increasing the overall speed of the system during the whole development process. D. Teuchert et al. [6] also dealt with FPGA based IPv6 lookup using trie-based method. Another interesting application of FPGA-based design is Gigabit Ethernet applications [2] due to the fact that FPGA-based implementations offer the possibility of changing the functionality of the platform to perform different tasks and high packet-processing rate capabilities. In particular, the authors of [2] proposed a versatile FPGA-based hardware platform for Gigabit Ethernet applications. By introducing controlled degradation to the network traffic, the authors provided an in-depth study on real-application behavior under a wide range of specific network conditions, such as file transfer, Internet telephony (VoIP) and video streaming. However, in those previous works, energy-efficiency of the system is not yet properly dealt with. As energy efficiency has become a key performance metric, techniques that can quickly and accurately obtain the energy performance of these soft processors are needed. In [3], this issue is addressed and methods for rapid energy estimation of computations on FPGA-based soft processors are provided. The authors propose a methodology based on instruction level energy profiling by analyzing the energy dissipation of various instructions with an energy estimator is built using this information. The paper deals with the estimation of energy consumption of computations on FPGA platforms, but does not propose any methods to manage and optimize it. Kennedy et al. [10] provide a frequency scaling based low power lookup method, but they optimize only the lookup function. We would also mention NetFPGA platforms that are largely in use in academic research to test for ideas and implement them on flexible hardware. Given that background, in this paper, we propose a practical energy management module on FPGA platform and validate our proposed solution through simulation analysis by using SystemC.

2.2 Packet Processing Pipeline

The packet processing pipeline is similar to the model recommended by Xilinx [9] and the model used by the Liberouter project [8]. The packet headers are processed in the pipeline while the packet payload needs to be buffered as shown on Fig. 1. In our initial work the buffering of packets is realized using the on-chip RAM referred as Block RAMs or BRAMs available in the FPGA. In the final design the packets will be buffered in an external DDR RAM. Since the memory access can be the bottleneck in our packet processing pipeline, we tried to avoid copying the packet. We have decided to use the shared-memory packet forwarding model, this way avoiding unnecessary copying of data. The model has a further advantage: even multicasting/broadcasting can be done without actually copying the data.

The arriving packets are buffered, while their header is processed in the input block. The input block applies the input filtering rules and sends the header to the forwarding module. The routing/switching module is responsible to decide on which

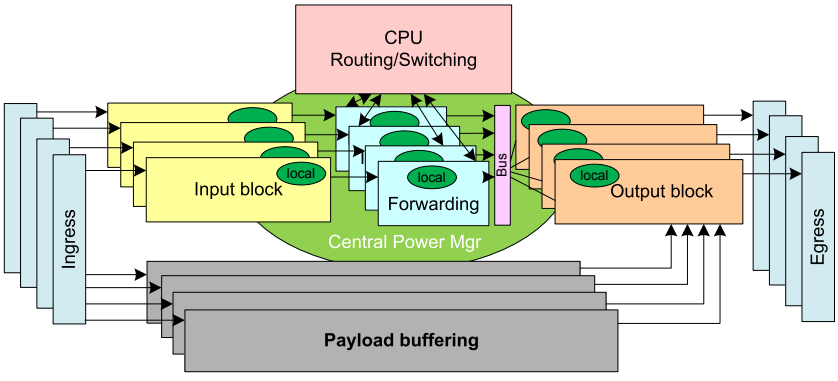

Fig. 1. Generic packet processing framework

interface the packet leave should. For flooding, broadcast and multicast multiple interfaces may be selected. For different application types, different routing/switching modules can be designed. After the decision, the packet can be output filtered, and the header will go to an output header buffer at the egress port. The packet will be assembled during transmission: when scheduled, the header will be sent from the header buffer, while the payload will be read from the payload buffer.

Each block has a local power manager, and together with the central power manager an efficient power management solution is provided.

This packet processing pipeline illustrates an example packet forwarding design. However, the design possibilities for new applications using the framework are endless; based on the existing modules monitoring, firewall, routing and switching applications can be created easily, while designing new modules any hardware-assisted packet processing application can be developed quickly. These models are generic, and planned for easy customization and implementation in other FPGA-based platforms. Furthermore, their scalability is also taken into account. For most models the scalability is ensured by design, providing cycle-accurate model of the hardware. Some of the simulation models however require high processor usage and long programming cycle. Their model is approximate.

2.3 Input and Output Blocks

The incoming frames of data are to be stored in BRAMs available on the FPGA during processing the header information. For receiving the information, we use input modules (Fig. 2.). These per-port modules get the frames from the PHY. The bitstream is deserialized and then it is forwarded both to the input buffer and header filters. The input filters perform the filtering action before reaching switch/router module. Before new frames are received, buffer reservation is checked, whether there are free slots available or not. A module called BRAM manager is responsible for the management of the BRAM buffers associated to an input.

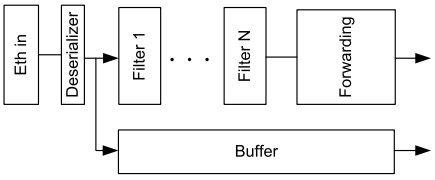

Fig. 2. Input port module

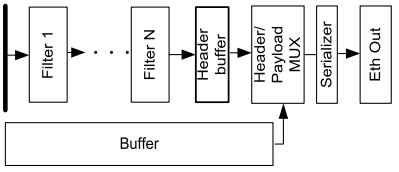

Fig. 3. Output port module

The output module (Fig. 3.) is responsible for sending out the routed/switched frames. To do this, the upper level modules (switches, routers) send the new packet header with payload buffer information. The ingress port and BRAM address is required to fetch the associated payload. The header information is stored in the header buffer, which is processed in a FIFO basis. This buffer plays the role of the output buffer, but since we do not want to copy the payload, only the headers are queued. The header/payload multiplexer takes the headers from the FIFO, starts sending them, and appends the payload data from the buffer indicated. The serializer is preparing the packet in a format required for the PHY.

2.4 MAC or IP Lookup Module

The MAC learning and lookup module as well as the IP lookup module is designed to be processor based. The module receives the destination MAC address to look it up, and the source MAC address to learn (Fig. 4.). The IP lookup module receives the destination IP address to look it up only. After the lookup either module returns the egress port or ports in case of multicast, broadcast or flooding. The simplest models would use a static local forwarding table; more complex models can use centralized lookup tables based on CAMs (or TCAM) and for large routing tables a CPU based lookup method can be used. The arbiter module is responsible for the access to the common bus of the output FIFOs. The switch modules send the headers and additional data to output stores. The arbiter enables them the access to the bus in a round-robin way.

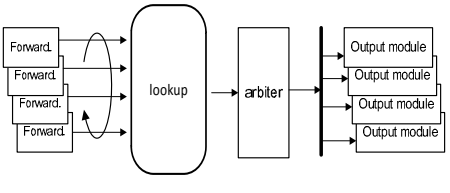

Fig. 4. Switch modules and the lookup engine

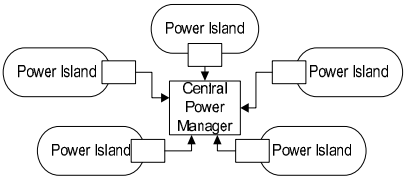

Fig. 5. Hierarchical power management

The fact that these modules are based on a CPU enables us to tune the duration of the lookups. This is done by controlling the frequency. Reducing frequency means power consumption reduction. In the implemented 4-port Ethernet switch, the CPU has 4 different frequency steps, resulting 4 level of performance and 4 level of power consumption.

3 Resource Management

The overall power requirement of a networking device can be reduced in several ways. The most obvious method is to switch off the parts that are not used. For example, switch off the interfaces that are not in use. Further energy can be saved by introducing power levels in the system that can be achieved either by allowing processors (or multiprocessor cores) to enter sleep mode or reducing the clock frequency and core voltage [7]. Another approach is to use energy management at the packet processing hardware level. This method requires that each module in the pipeline has an enable pin, and each module is powered only for the time period its operation is in need.

In a high-speed packet processing device the energy management should not compromise the performance. Performance degradation results in additional delays, which lead to traffic burstiness, jitter and increased end-to-end delays.

Our platform is designed to provide line-rate throughput on all interfaces. This means, that although there is buffering for the payload and header, it is not necessary to make copies of the payload, only the headers. Of course buffering may occur at the output interfaces caused by traffic aggregated from different interfaces, but the routing/switching must be designed to handle line-rate traffic from all input interfaces. In other words, while a whole packet is received on an interface, the routing/switching process should find the egress port to avoid delays on the output port, even if there are packets at other interfaces that need to be serviced with priority. In worst case scenario, 3 packets in other queues must be serviced before the round-robin scheduler services the new packet. Thus, the processor speed should be always selected according to the number of packets actually received.

The optimal energy management solution has the following requirements:

a) power up hardware components only for the time required
b) provide maximum energy efficiency by selecting the processing power according to the traffic conditions
c) do not compromise the line-speed forwarding at any time

On the other hand, selecting the speed of a processor or starting/halting cores is not practical and may not be feasible on a packet arrival basis (e.g. ~700ns on a 1Gbps link vs. ~10μs change of the state of an StrongARM CPU [11]). Therefore we must predict the traffic load for longer time period and make energy management decisions on a larger time scale.

In our framework, we propose a hierarchical power management with two levels: a central power manager that is connected to several local power managers. The local power managers are responsible for the power management in their own domain, called “power island” (see Fig. 5).

The central power manager provides user-manageable functions like disabling the input/output port module and makes global decisions, while the local power managers optimize the power dissipation micro-managing the units in the power island.

3.1 Lower Level – Packet Processing Line

Each module has a chip enable pin which is connected to a local power manager. The different ports and port handling modules are connected to this local power manager,

forming several power islands. The enable signal of the local power managers is collected in levels of hierarchy, and the upper level power manager is responsible of handling the enable signal of the power islands.

Within a power island the local manager is responsible of enabling the modules of the packet processing pipeline. Each module is powered only for its job, and it is powered down after completion, keeping consumption at an optimal level. The PHY layer is handled by a chip and is always on. These analogue and digital circuits are responsible for the line coding and physical requirements of the different physical layer standards. At a packet arrival they immediately inform the local managers that packet processing starts, following modules in the pipeline need to be powered up. Each module in the pipeline should be operating until the packet leaves the module and the end-of-packet signal notifies the local power manager to switch it off. The operation of the manager is optimal, as they micro-manage the pipeline to achieve maximum performance and maximum power saving.

3.2 Higher Level – Administrative and Power Control

At a higher level, two main functionalities are performed: power management and load-sensitive frequency scaling of lookup processors. The central manager presents a power management interface for the management. Power management decisions such as disabling interfaces are performed at this level. When the operator decides to disable an interface or even a subsystem, the central power manager commands all power islands related to the disabled subsystem to enter low power state.

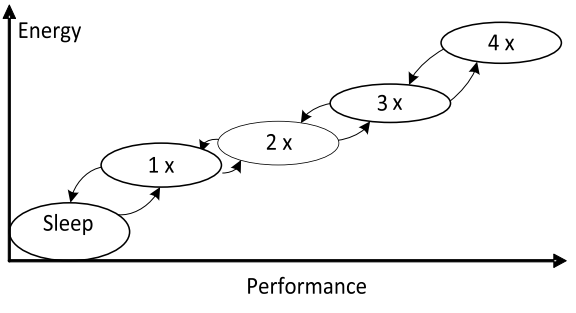

Fig. 6. Processor power management

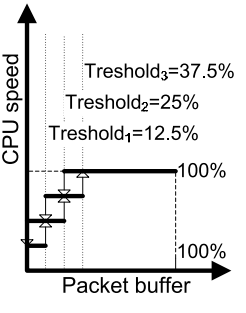

Fig. 7. Buffer Thresholds

Besides the resource management interface the central manager is also optimizing the power management by managing the power of lookup/routing CPU cores. Setting processing speed or waking up/shutting down CPU cores is performed according to the offered load. In our model the CPU speed is modeled in 5 power states: the sleep mode and 1..4× frequency mode (see Fig. 6.).

The maximum performance can be achieved running at 4× speed but of course it also means higher dissipated energy. The network load is estimated according to the input buffer utilization. We have defined 3 thresholds (Fig. 7.) for buffer utilization and the CPU speed is increased or decreased whenever a threshold is crossed.

The two power management modes together provide a nearly optimal usage of available resources while keeping the system performance nearly unaffected.

4 Simulations

The modeling framework has been implemented in SystemC 2.2.0. The simulated hardware is a 4 port Ethernet switch model. The packet processing pipeline consists of a filter for own MAC address, header processing and switch module. The switch module uses a CPU based lookup where the CPU has 4 levels of speed. The switch has low buffering capabilities: each port can buffer only 4 packets. This is a realistic scenario for a Xilinx Spartan 3 FPGA without external RAM. The processing pipeline has been designed with line-speed forwarding in mind, with bottlenecks only at the egress ports, where traffic from different directions is mixed. Therefore the input queuing permits 2 headers only. The simulations were performed using traffic generators and sinks. To each port a traffic generator & sink is connected. Each traffic generator generates traffic to all other sinks, with equal probability.

In the simulation we used 100Mbit Ethernet ports. The simulated traffic was characterized by exponentially distributed inter-arrival times with expected values from 2μs to 1ms. In each simulation the packet size was fixed 64, 128, 256, 512, 1024 or 1500 bytes. During the simulations we estimated the power saving and delay for each module. As expected, the switch is capable of line speed processing, on all ports, without blocking. Header processing is performed while the packet payload is still received, and egress lookup should also be fast enough not to hold the processing pipe.

4.1 Results

To assess the power savings realized by locally managing the enable pins of different subsystems, we performed simulations with increasing traffic. To observe the effect of packet sizes, we repeated the simulations for different typical packet sizes. The obtained results are expressed as relative values compared to the “always on” scenario which is considered 100% uptime. Fig. 8. shows the uptime of the forwarding module in percentage compared to the “always on” scenario. This module needs to be “on” the longest time in the pipe as it must wait the result of the lookup. As expected, the gain decreases with the load on the network, but it is considerable even for high loads. In an extreme case the active link without traffic can save 99% on this module because it is powered off most of the time. Another example, the uptime of the destination MAC address filter is shown on Fig. 9. This module performs a simple operation; it can be “off” most of the time as it works on the headers only. The higher the packet size the higher the energy gain. At least 60% can be saved on this module.

Then we simulated the gain of managing the power levels of the switching processor. In order to keep the delay at minimum, we should only reduce the speed to a level where it is not delaying packets. The processor is entering lower power mode when the overall buffer utilization drops below a threshold. Scaling down the CPU speed comes at a price: the CPU speed cannot be changed as often as the load changes. Thus, the first packet(s) of a burst will be served at a lower speed, introducing delays for the upcoming packets.

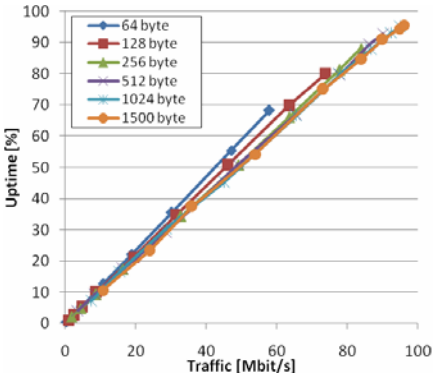

Fig. 8. Uptime of a characteristic module in the pipeline depending on packet size

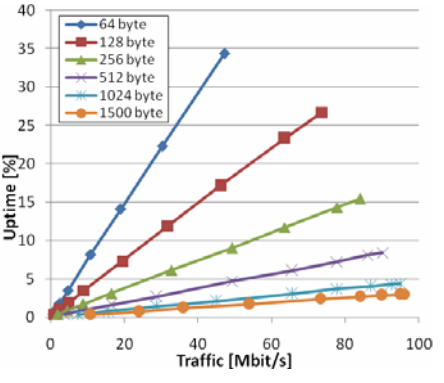

Fig. 9. Uptime of the destination MAC address filter depending on packet size

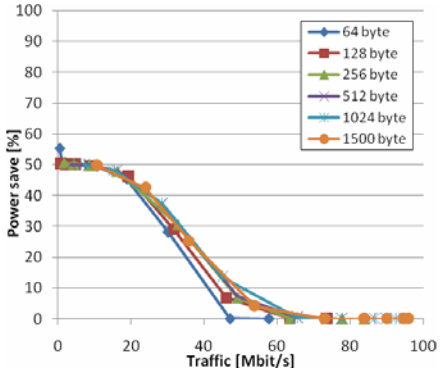

Fig. 10. Power saved with frequency tuning on CPU depending on packet size

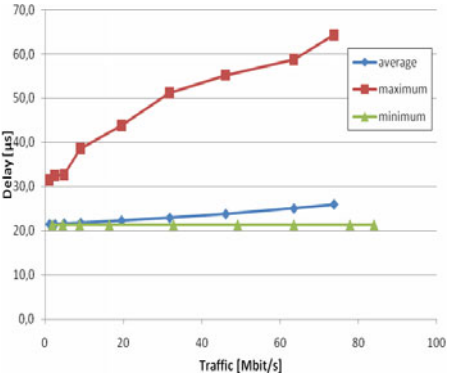

Fig. 11. Latency values for packets of 128 bytes

Again, simulations are repeated for different load levels and packet sizes. Besides the power saving, we also show the mean and maximum delays introduced.

Fig. 10. shows the power saving achieved at the lookup processor as a percentage of the always at highest performance scenario. The gain increases as the load decreases, as expected. Also, the gain increases as the packet size decreases, providing maximal gain when the overall utilization is the lowest. The gain can be as high as 50%. An example for latency at 128 byte packet-size is shown on Fig. 11, although the maximum delay is doubled, the average latency is close to the minimum, meaning that only a few packets get delayed. As the load gets higher, the CPU is running on higher performance step most of the time, handling small burst with higher speed, and delay is still not increased. As the traffic further increases, the buffers saturate, and some frames have to wait more, increasing the maximum delays.

5 Conclusions

In this paper we have presented a generic packet processing framework that can be a background for different packet processing applications like switching, routing filtering, monitoring etc. The framework is modular, which provides a scalable and composable system. For design verification a simulation environment is used, which provides cycle-accurate models for the components.

Using the simulation environment, we have designed an example switch application, with modular input and output filtering pipeline. The energy efficient operation is ensured by a hierarchical energy management. Results show that the hardware is capable of packet processing at the line rate, still achieving up to 60% gain in power for the simple modules, and in idle periods we can achieve a power saving of up to 99%, depending on traffic. For more complex modules like the forwarding module there is small gain at high loads, but again very high gain at low loads. We have also shown that with CPU scaling we can get considerable gain at a price of small increase in the traffic jitter.

Even further energy savings are possible by using dedicated hardware accelerators for specific lookup and other CPU-intensive operations.

References

1. Antos, D., Rehak, V., Korenek, J.: Hardware Router's Lookup Machine and its Formal Verification. In: ICN 2004 Conference Proceedings (2004)
2. Ciobotaru, M., Ivanovici, M., Beuran, R., Stancu, S.: Versatile FPGA-based Hardware Platform for Gigabit Ethernet Applications. In: 6th Annual Postgraduate Symposium, Liverpool, UK, June 27-28 (2005)
3. Ou, J., Prasanna, V.K.: Rapid Energy Estimation of Computations on FPGA-based Soft Processors. IEEE System-on-Chip Conference (2004)
4. Werner, M., Richling, J., Milanovic, N., Stantchev, V.: Composability Concept for Dependable Embedded Systems. In: Proceedings of the International Workshop on Dependable Embedded Systems at the 22nd Symposium on Reliable Distributed Systems (SRDS 2003), Florence, Italy (2003)
5. OSCI SystemC 2.2.0 Documentation: User's Guide, Functional Specifications, Language Reference Manual, `http://www.systemc.org/`
6. Teuchert, D., Hauger, S.: A Pipelined IP Address Lookup Module for 100 Gbps Line Rates and beyond. The Internet of Future, 148–157 (2009) ISBN 978-3-642-03699-6
7. Intel White Paper: Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor (March 2004), `ftp://download.intel.com/design/network/papers/30117401.pdf`
8. Liberouter project homepage, `http://www.liberouter.org/`
9. Possley, N.: Traffic Management in Xilinx FPGAs, White Paper, April 10 (2006)
10. Kennedy, A., et al.: Low Power Architecture for High Speed Packet Classification. In: ANCS 2008, San Jose, CA, USA, November 6-7 (2008)
11. Iranli, A., Pedram, M.: System-level power management: An overview. In: Chen, W.-K. (ed.) The VLSI Handbook, 2nd edn., December, Taylor and Francis, Abington (December 2006)