SynZEN: A Hybrid TTA/VLIW Architecture with a Distributed Register File

Stefan Hauser, Nico Moser, and Ben Juurlink
Embedded Systems Architecture
Technische Universität Berlin

Abstract—**The quest for higher performance within a certain power budget in the fields of embedded computing demands unconventional architectural approaches. To this end, in this paper we present *synZEN* (sZ): a (micro-)architecture that combines features of *very long instruction word* (VLIW) and *transport triggered architectures* (TTAs) to cover the needs of different applications. SynZEN features a distributed *register file* (RF) (i.e., each *functional unit* (FU) has its own RF) and a wide memory connection to exploit spatial data locality. FPGA synthesis results demonstrate that due to the distributed RF the sZ design can be implemented in less area (in terms of FPGA slices) than existing TTA and VLIW designs. Furthermore, using two micro-benchmarks we show that because of the wide memory connection, sZ outperforms both the TTA as well as the VLIW design.**

I. Introduction

We present a processor architecture, called synZEN (sZ), which was initially designed as an application-specific processor. The design goals of this architecture are to utilize inherent parallelism, provide flexibility, and to be scalable. Therefore our architecture combines features of two well known architectural concepts. We combine the powerful VLIW FUs with a flexible *interconnection network* (ICN) based on the TTA concept.

This paper focuses on presenting the micro-architecture and outlines other engineering aspects necessary to understand the architecture. Furthermore, we compare our approach to a VLIW architecture and a TTA implementation. The main contributions of this paper can be summarized as follows:

- We present an architecture which has the potential of extracting high inherent *instruction level parallelism* (ILP) of applications. Due to the very good architecture scalability the extractable ILP scales too.
- sZ features a wide memory interface to the data memory, which allows parallel access to consecutively stored data which supports the parallelism of the architecture.
- With the distributed RF we significantly increase the scalability and decrease the overall costs while avoiding reducing the number of registers each FU has access to.
- We evaluate this hybrid architecture by comparing its performance and cost to similar architectures. It is shown that the combination of architectural features of different architectures result in higher performance as well as in lower resource consumption. We also show that the resource consumption of our local registers increases only linearly with the increasing number of FUs in contrast to the quickly increasing cost of architectures with a central RF.

This paper is organized as follows. In the next section we briefly review work related to our architecture. In Section III we present the details of our hybrid architecture. After that, in Section IV, experimental results are provided, which show the capabilities of this architecture. Finally, Section V summarizes and highlights the contributions of this work and presents our final conclusions.

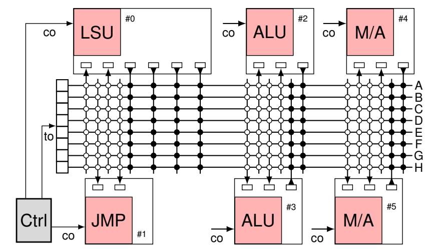
II. Related Work

One major drawback of the VLIW architecture are the hardware costs for the necessary multiport RF. The most common solution to avoid these costs is using clustered RFs [1] where each of these clustered RFs serves more than one FU. Some approaches go even further and distribute the control path also, such as [2]. This leads to a more difficult branch handling. In contrast, in our approach there are no clustered RFs, but the RF is distributed among the different FUs and therefore scalable. Ongoing academic research takes place in the ϱ-VEX project, which is based on the VEX compiler toolchain. After an initial parametrizable soft core implementation [3], further research aims to enhance the architecture. A promising approach by using FPGA specific features to solve the multiport RF challenge is shown in [4]. This work also shows a way to use resource sharing in multiprocessor settings, where control logic supports the use of functional logic in different data paths [5]. Compared to our architecture, the biggest bottlenecks are the restricted memory interface and the central RF.

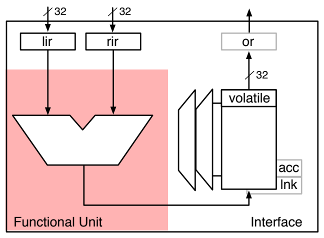
Current work on TTA uses the inherent streaming qualities to realise TTA-based processors for GPU applications [6]. Caused by TTA origin this processor lacks data locality and self-scaling local memory capabilities. In [7] an architecture with distributed RFs is shown where the read access is exclusive to assigned FUs. Compared to sZ this approach has an ICN in common but has upstreamed RFs which causes data duplication.

III. SynZEN Architecture

Our approach combines the advantages of VLIW architectures, TTAs, and a distributed RF. Therefore we equip each FU with a local small RF, and we call the combination a *processing unit* (PU). This extends our approach to more application domains, by overcoming some limitations of existing solutions. Namely we intend more complex functionality for our PUs compared to TTA FUs and we utilize a distributed



(a) The sZ-ϱ instance, with LSU, ALU, and M/A



(b) The sZ PU including functional unit and interface

Fig. 1. Architectural details of (a) a sZ instance and (b) a PU

RF. Additionally, we propose a proper memory connection to provide sufficient data bandwidth and to utilize as many of our PUs as possible. In this section we describe the sZ architecture in detail.

A. Hybrid

Our sZ architecture contains multiple PUs. These units are connected to each other by an ICN. So far this follows the classical TTA concept, as it was proposed in [8]. In a conventional TTA, however, the number of operations that can be performed by each unit is very limited. The operation to be executed is defined by a part of the address, since a TTA supports only transport operations. Therefore each unit has several addresses, and depending on the address the unit performs a different operation. This concept, called virtual addresses, is similar to submitting an operation code to the designated unit itself over the network. Consequently, it requires additional hardware in terms of wiring fabric. In our design, which is exemplarily depicted in Fig. 1a, we separate the operations for the unit from the network operations. Accordingly, we have two kinds of operations: (1) transport operations (TOs) to route the data flow (2) control operations (COs) to select the instructions to be performed by the PUs and address the register file within the PUs. Besides the reduction of hardware complexity, we also increase the instruction code density.

The number of PUs as well as their functionalities can be configured according to the requirements of the target application. The network is also freely configurable, so it can be adapted to the communication structure of the application. As shown in [9], a reduced network will also decrease the hardware costs significantly. Nevertheless, it is essential to access a sufficient amount of data to achieve an (almost) full utilization of the PUs. To avoid communication over the ICN several mechanisms are presented in [9]. These mechanisms are based on program analysis. For example, if an operand is used several times consecutively by the same unit, it can be kept at the unit in an input register. This avoids again transportation over the network. Furthermore, a PU can behave like a 1-address machine, which means that only one operand has to be delivered over the ICN. The second operand is stored in an accumulator register, which is part of the local RF. In addition, if an operand from another unit is required very often, a link between both units can be established, thereby again avoiding communication over the ICN. These modes can be applied to both input operands. Hence, an unit can also perform without data from the ICN. These modes distinguish our approach from the conventional TTA.

B. Distributed Register File

It is well-known that the RF cost increases dramatically with the number of ports [7]. One solution to overcome this is to use a clustered architecture [10]. Even in one cluster instance of clustered VLIW architectures, such as the ϱ-VEX architecture, a non-negligible number of ports is required. Especially for FPGA-specific designs a huge number of ports is difficult to implement, as FPGAs provide only dual-ported *block RAM*s (BRAMs). A larger number of ports can be realized by using additional logic cells.

For sZ we propose a distributed RF. Each PU has a local RF of 16 registers. Hence, the number of registers scales with the number of units. Each output of a PU is connected to one local dual-ported RF, as depicted in Fig. 1b. The output port of the RF is connected to the ICN. Parts of the control operation specify which register is written and which register content is forwarded to the ICN.

We compare two ϱ-VEX RF configurations with our sZ distributed configuration in Fig. 2. The figure depicts the hardware costs in terms of FPGA slices (scaled logarithmically), for a Xilinx Virtex 5 LXT 110 FPGA, for an increasing number of FUs respectively PUs. Increasing the number of units increases the number of registers and only for the ϱ-VEX the number of ports. The unit itself is not taken into account. To achieve comparable results we block the utilization

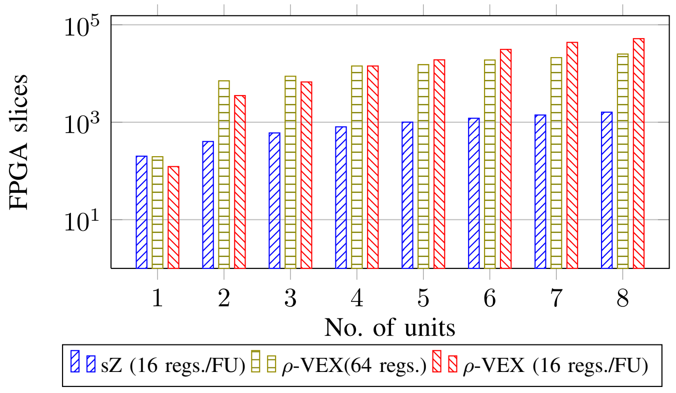


Fig. 2. Resource utilization using FPGA slices (Xilinx Virtex 5 LXT 110 FPGA) of a) the sZ RF, b) the ϱ-VEX RF for different issues with 64 registers, and c) the ϱ-VEX-RF for different number of units and 16 registers per unit

of BRAMs. However, this only affects the case of one unit, because of the low number of ports. We evaluate three different configurations: the first bar represents our distributed sZ RF, the second bar represents the ϱ-VEX RF with a fixed number of 64 registers, as it was specified by the original design. The resource consumption of a centralized RF, where the number of registers scales with the number of FUs, is visualized by the third bar.

The figure shows that the resource consumption of both ϱ-VEX RF configurations (second and third bar in Fig. 2) increases drastically with the number of ports, because per unit one write and two read ports are required. Especially the gap between one and two units is immense, because of the limitations of FPGAs. Furthermore slice consumption for the ϱ-VEX version with 16 registers per FU increases significantly if we apply more than four units. It is important, however, to increase the number of registers with the number of FUs. By doing so register allocation becomes more efficient, simpler, and spill code can be avoided.

With our sZ architecture hardware cost is significantly reduced with the same number of registers, as shown by the first bar in Fig. 2. Of course, there is less connectivity, because not every unit is able to write directly into each register but sophisticated scheduling mechanisms, such as [7], are able to solve this problem.

C. Memory Connection

Another challenge of such an architecture is to provide sufficient memory bandwidth to keep all PUs busy. The *TTA-based Co-design Environment* (TCE) approach [11] allows connecting several *load store unit*s (LSUs), but the system is limited to two LSUs and each LSU requires new hardware resources and a memory connection, which results in more ports. Our approach uses only one LSU, which can be connected to a DDR RAM. This unit caches the data from the main memory, and it provides one complete cache line to the ICN. Thus several data words can be accessed at once. This approach is inspired by the spatial locality principle, which states that data required for consecutive operations are stored in close proximity in memory. Providing a whole cache line at once especially suits applications that exhibit *data level parallelism* (DLP).

IV. Results

Though our architecture is application-specific, the focus is on one specific instance for the scope of this paper. Therefore we created an instance, depicted in Fig. 1a, which is comparable to the existing, publicly available implementation of ϱ-VEX [3]. This instance, called sZ-ϱ, contains four PUs. All PUs support arithmetic and logical operations. Two of them also support multiplications, indicated by M/A. An LSU has been added to load data from an external memory. In sZ normally this unit behaves as a cache. To make a fair comparison and since this feature is not supported by the other considered architectures, we simulate the cache behavior with no delay. The LSU nevertheless reads one cache line, in this case four times 32 bits, at once, as described in Section III-C.

Name	#RF	#reg	#FU	#LSU
sZ-ϱ	4	64	4	1
ϱVEX	1	64	4	1
TCE	4	32	4	1/2

Fig. 3. Characteristics of the architectures

For the comparison to a TTA implementation we employed the TCE [11]. This framework allows to configure an instance by defining the number of FUs and the functionality of each FU. We applied a design similar to our sZ-ϱ instance. Four FUs with similar functionality plus four centralized RFs with 32 registers in total are included. More registers are not required, because the existing ones are not fully utilized. Furthermore, the framework does not allow synthesizing a larger RF. Two other realizations are analyzed by increasing the number of LSUs. This allows a better comparison to the sZ implementation, since the latter employs a wider memory connection, as has been described above.

Furthermore, we assured that our sZ architecture provides the same functionality as the ϱ-VEX architecture. Therefore a fully connected ICN is used. Otherwise, not every unit could access all registers, as is needed for the ϱ-VEX architecture. Fig. 3 shows that the different architectures are configured in a comparable way.

A. Resource Consumption

We synthesized our design, the ϱ-VEX implementation, and the two TTA-configurations for a Xilinx Virtex 5 LXT 110 FPGA. All designs are able to operate at a clock frequency of up to 80 MHz. The occupied slices after place and route for memory and logic are depicted in Fig. 4.

It can be seen that the TTA and our sZ-ϱ require significantly fewer resources than the ϱ-VEX. This is due to a smaller control path and simpler FUs. The simplified control logic results mainly from the lack of finite state machines, which are required for the multicycle implementation of the ϱ-VEX. The impression that our design requires less memory is not correct, since the dual-ported memories of our design can be mapped to existing blocks. So the memory consumption of the sZ is comparable to the TTA realization. Due to our

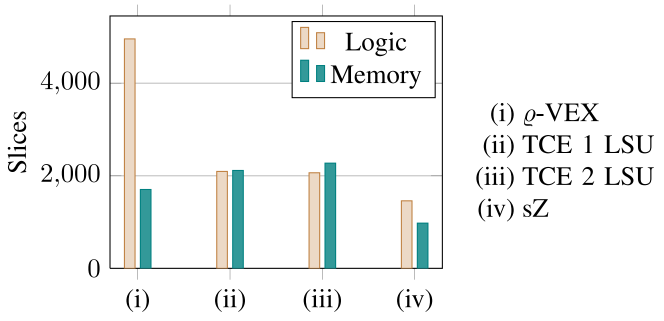
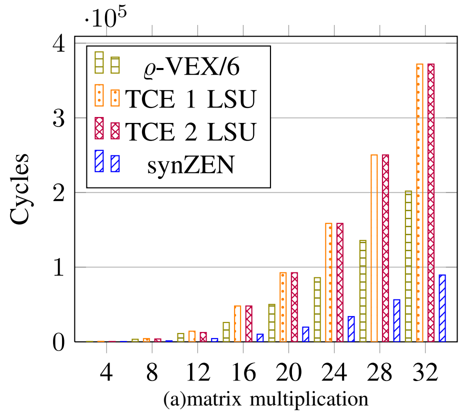


Fig. 4. Slice consumption



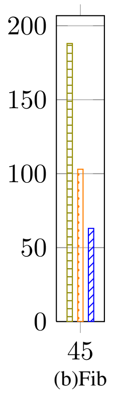


Fig. 5. Performance evaluation

distributed design, however, definitely less resources for logic are required.

B. Performance

To compare sZ-ϱ to the other two architectures, applications which run on all architectures are needed. Furthermore, due to the lack of a suitable compiler for our architecture, we are currently limited to simple applications, and therefore we employ Fibonacci numbers and matrix multiplication as test-kernels.

For the TTAs the tcecc-compiler of the TCE [11] is utilized. The ϱ-VEX test code was first created by VEX compiler toolchain [12], but it was discovered that the performance results were poor. Hence, a hand-optimized assembler version is used. For the sZ-ϱ architecture, hand-optimizations were necessary, too.

Since the available ϱ-VEX architecture is a multicycle implementation, it is difficult to perform a fair comparison. But ϱ-VEX can also be implemented in a pipelined way. To overcome this we assume an ideal pipeline, which means that there are no stall cycles at all and that the pipeline is always completely filled. To approximate this we divided the number of cycles for the ϱ-VEX by 6, as in the multicycle implementation each instruction requires 6 cycles. The results for matrix-multiplication are depicted in Fig. 5a. The performance gap between our architecture and the TTA is due to the distributed RF and the wide memory connection. For non-streaming applications the centralized RF in the TTA approach has significant disadvantages. By adding a second LSU a wider memory connection can also be realized for the TTA architecture. This reduces the number of cycles for $n < 16$, but for $n \geq 16$ the compiler cannot utilize this additional LSU in a noticeable fashion.

The ϱ-VEX architecture performs better than the TTA, since this architecture is more suited to this kind of applications. The reason for this is, that the second matrix cannot be loaded consecutively. Therefore a lot of results must be stored. For a TTA this means additionally transportations: First to the RF and second later to the next unit. The performance is lower, however than our sZ approach. The reason for this is the limited memory bandwidth. In this architecture only one load operation per cycle is possible.

The 45th Fibonacci number is computed by the second benchmark. Fig. 5b shows the results. On this benchmark the TCE design performs better than the ϱ-VEX architecture, because the application is more suited to data flow architectures. However, the central RF is the limiting factor; therefore the sZ architecture achieves the best results.

V. Conclusions

In this paper we have presented a hybrid architecture, which combines VLIW and TTA features. In addition, we have applied a distributed RF and a wide memory connection. Furthermore, we have evaluated our architecture in terms of resource consumption and performance.

We have demonstrated that our approach is competitive. Due to adaptable PUs and simple control logic, our design consumes fewer resources than the other considered approaches. Even increasing the number of registers, with the number of units, is possible. Furthermore the down streamed RF allows to interrupt the dataflow with less overhead. So results can be stored directly in the output RF of the computing unit and later they can be transported further. This makes our design suitable for several application domains in contrast to a classical TTA approach. Beyond that, it simplifies the instruction scheduling and helps to avoid spill code.

Furthermore, we achieve a remarkable performance through the different features of our architecture described in this paper. Especially if we consider the low resource requirements, which makes it feasible to create an instance with more units or to combine several instances within one FPGA.

In future work we intend to evaluate the power consumption of the sZ architecture. Furthermore, we aim at integrating sZ as an ILP accelerator in a heterogeneous multicore architecture. In addition, we are currently working on improving the toolchain and especially the compiler, as well as evaluating the performance of sZ on complete applications.

References

[1] R. Simar et. al., "How TI adopted VLIW in digital signal processors," *Solid-State Circuits Magazine, IEEE*, vol. 1, no. 3, pp. 10–14, 2009.
[2] H. Zhong et. al., "A distributed control path architecture for VLIW processors," in *Parallel Architectures and Compilation Techniques, PACT.*, Sept. 2005, pp. 197–206.
[3] S. Wong et. al., "rho-VEX: A reconfigurable and extensible softcore VLIW processor," in *ICECE Technology. FPT.*, Dec 2008, pp. 369–372.
[4] F. Anjam et. al., "A multiported register file with register renaming for configurable softcore VLIW processors," in *Field-Programmable Technology (FPT)*, Dec 2010, pp. 403–408.
[5] F. Anjam et. al., "A shared reconfigurable VLIW multiprocessor system," in *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, Apr 2010, pp. 1–8.
[6] C. S. de La Lama et. al., "Programmable and Scalable Architecture for Graphics Processing Units," *Transactions on HiPEAC*, vol. 5, 2010.
[7] P. Mattson et. al., "Communication Scheduling," *SIGARCH Comput. Archit. News*, vol. 28, no. 5, pp. 82–92, Nov 2000.
[8] H. Corporaal, *Microprocessor Architectures: From VLIW to TTA*. John Wiley & Sons, 1997.
[9] N. Moser et. al., "A Hybrid Transport/Control Operation Triggered Architecture," in *Workshop on Parallel Systems and Algorithms ARCS*, Feb 2010, pp. 121–125.
[10] J. A. Fisher et. al., *Embedded Computing. A VLIW Approach to Architecture, Compilers and Tools*. Morgan Kaufmann, 2005.
[11] TCE, "TTA-based Co-design environment. http://tce.cs.tut.fi/" 2012.
[12] HP Lab, "VEX Toolchain. http://www.hpl.hp.com/downloads/vex/" 2012