A NEW SCALABLE DSP ARCHITECTURE FOR SYSTEM ON CHIP (SOC) DOMAINS

Matthias H. Weiss, Frank Engel, and Gerhard P. Fettweis

Mobile Communications Systems Chair
Dresden University of Technology, 01062 Dresden, Germany
{weissm,engel,fettweis}@ifn.et.tu-dresden.de

ABSTRACT

The ongoing advances in semiconductor technology are the enabler for complete *System on Chip* (SoC) solutions. In this SoC domain *Digital Signal Processors* (DSPs) are employed to carry out software driven digital signal processing tasks. Although DSPs could still be modified in the SoC domain, they are mainly employed as fixed DSP cores. Possible adaptations to the embedding system cannot be carried out.
Thus, our work is targeted to design expandable DSP architectures. To achieve this expandability, we designed a sliced DSP architecture. Here, the number of slices can be adapted towards system needs. Specific system requirements can be achieved by adding dedicated datapaths to these slices. With this approach one magnitude of order in performance boost can be achieved, which creates new demands for I/O processing. Thus, within our DSP architecture we integrated a dedicated I/O processor.
In this paper we present this new scalable DSP architecture, tools to map algorithms onto this DSP architecture, and the concept of our new I/O controller. These technologies allow to easily adapt our DSP architecture to different system requirements.

1. INTRODUCTION

In *Systems on Chip* (SoC) domains software driven digital signal processing tasks are carried out by *Digital Signal Processors* (DSPs). Examples are sound-, video-, modem- or speech- applications. In this SoC domain, system tasks can be partitioned onto different SoC components such as micro controller, dedicated *Application Specific Integrated Circuits* (ASICs), or DSPs. If the DSP needs to provide more performance but cannot be extended, tasks have to be supported by an ASIC, although DSP flexibility would be needed. Thus, in the SoC domain a DSP architecture is required, which can be adapted towards system needs.

1.1. Requirements

This becomes obvious in applications with high data rates, such as video- or graphic-codecs, speech recognition, or high data rate modems. Here, data rates are in a similar range as the processor clock. Thus, the processing has to be speeded up by exploiting parallelism. This can be achieved by exploiting coarse grain parallelism, e.g. by implementing different algorithm blocks onto different hardware units. The dataflow graph is imitated by the hardware implementation. However, if the application changes the hardware has to be changed also.
A different approach is to exploit fine grain parallelism, e.g. by employing a processor architecture with parallel datapaths. This provides a software programmable solution and can be even more efficient than a hardwired solution [1]. Since here parallelism is exploited on the instruction level, different algorithms need to use one common architecture.
Thus, in the SoC domain a programmable processor architecture is needed, which provides scalability and expandability.

This work has been sponsored in part by the Deutsche Forschungsgemeinschaft (DFG) within the Sonderforschungsbereich (SFB) 358, Siemens Semiconductor, and Asahi Chemical.

1.2. Current DSP Architectures

One way to expand DSP architectures is to duplicate the complete core. An example for such a multi core solution is TI's C80x. This architecture comprises 4 DSP units, 4 memories, and one RISC core. The connection is performed by a crossbar switch. However, this leads to an expensive communication network and requires a complicated programming paradigm. Even TI reduced this architecture down to 2 cores by introducing the C82x architecture. A different approach is taken by media processors. These processors such as TI's C62x or Philip's TriMedia are typically based on a *Very Long Instruction Word* (VLIW) *Instruction Set Architecture* (ISA), employ register files, and provide packed arithmetic. Here, the communication problem is solved via an expensive global register file, which had to be split into 2 parts in TI's C60 case.
Another way to solve this communication problem is to use a matrix memory as suggested by [2] or [3]. However, memory is a critical system issue and often standard memory components are preferable. Thus, MicroUnity employed a wide standard memory, which contains grouped data [4]. Here, several data elements share one common address and are accessed in groups only. The processing is based on the *Single Instruction Multiple Data* (SIMD) principle. However, in MicroUnity's MediaProcessor a special communication unit is employed which accesses the group register file. In case of extending this architecture, by adding more parallelism, or introducing new algorithms, this communication unit has to be redesigned.

1.3. The New DSP Architecture

In this paper we present a scalable DSP architecture, which is based on group memory. By slicing this architecture, exploiting SIMD properties, and employing a modular VLIW ISA, this architecture can be adapted towards the application's needs. This property results from applying the method of orthogonalization on all design levels (Section 2). At the system level the DSP contains

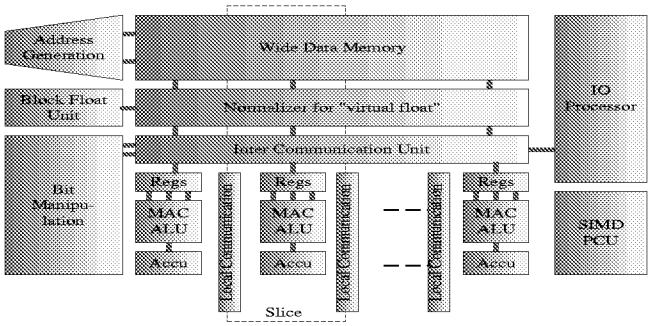

Fig. 1: *Sliced Architecture*

a separated I/O and data processing part (Section 4), at the architecture level it contains a separated register file and datapath units (Section 2.4), and at the algorithms level we separated index/data movement functions from the special arithmetic (Section 3). Finally, in Section 5 for an implementation example named the M^3-DSP benchmarks are presented.

2. ORTHOGONALIZATION OF THE ARCHITECTURE

To satisfy the needs of the target application the DSP architecture must be easily adaptable. Thus, the basic architecture must be expandable by further datapaths or function units. To provide this property, we applied the method of orthogonalisation. Both algorithms and architecture are separated into a *data transfer* (Section 2.1) and *data manipulation* part [5]. While *data transfer* includes all memory-register, register-register, or memory-memory transfers, *data manipulation* includes datapath functionality such as fused vs. divided Multiply/Accumulation, Galois vs. integer arithmetic [6], or the *Add Compare Select* (ACS) functionality to support the Viterbi algorithm. This approach allows to classify the algorithm's *data transfer* behaviour independent from *data manipulation*. Thus, we can concentrate on *data transfer* to find a common memory architecture which can be finally tailorized towards the target application's needs by specific datapaths.

2.1. Data Transfer and Manipulation

One of the major differences between different DSP architectures is the way to get data from memory to the manipulation units and vice versa. The *Multiply/Accumulate* (MAC) unit for instance requires three data elements at the input and produces one output. If this MAC unit is duplicated, six inputs and two outputs must be covered. In general, to avoid memory accesses for all data elements, register files are introduced. However, a register file is a crucial design issue for achieving a power and area efficient implementation. TI split their register file into two parts while in the HiPAR-DSP the generality of the register file was reduced [2].
In contrary to this general register file approach, our register file comprises only connections targeted towards DSP applications. These connections support the following *data transfer* classes:

- *Vector Data Transfer* (VDT),
- *Sliding Window Data Transfer* (SWDT), and
- *Shuffle Data Transfer* (SDT).

Each of these classes has different requirements for the connectivity of our register file. To provide a scalable architecture, the register file has to support these classes and still has to be scalable (Section 2.4). In contrast, data manipulation may differ between algorithms. Thus, within our scalable architecture dedicated datapaths can be included into the generic data transfer framework architecture to support the algorithm's special arithmetic.

2.2. Group Principle

To allow for a simple Address Generation Unit (AGU) while also providing high throughput we apply the group principle. Here, several elements are combined to a group and are accessed in groups only. Thus, in our architecture one memory read provides one element for each slice. Since these groups contain elements which are not always in the desired order, we added several communication features to our register file.

2.3. Scalable DSP Architecture

In general, a DSP architecture comprises function units for both tasks: *data processing* supported by datapaths (*Arithmetic Logical Units* (ALUs), *Multiply Accumulate Unit* (MACs)) or memories units, and for *program processing* such as a *Program Control Unit* (PCU), a *Loop Unit* (LPU), or an *Address Generation Unit* (AGU). To provide a scalable DSP architecture, *data processing units* must be parameterized to allow adaption to system needs, while *program processing units* must be widely independent from architectural changes to avoid expensive redesigns between two DSP architectures [7]. We solved the first issue via slicing the architecture, while the latter was achieved by a new modular *Instruction Set Architecture* (ISA) called *Tagged Very Long Instruction Word* [8].
To provide scalability, we sliced the architecture as depicted in Fig.1. Each slice contains one element of the group memory, several local registers, and a datapath, e.g. a MAC and a scale unit.These slices can be added into one parallel architecture as required by the system needs.

2.4. Scalable Register File

Obviously, this sliced architecture is advantageous for vector processing. Here, each slice computes one element of a vector. However, not all digital signal processing algorithms can be efficiently computed by vector processing. Examples are filters or transforms. These algorithms require communication between data elements. Therefore, two communication units are added to the architecture. The *Inter Communication Unit* (ICU) allows to permute a number of elements in a group, which is required to support *Shuffle Data Transfer*. To support communication between neighboring slices as required for complex number arithmetic, a local communication unit is introduced. This unit also supports *Sliding Window Data Transfer* (e.g. found in filters or pattern matching algorithms). By splitting the communication unit into these two units - ICU and LCU - it becomes independent from the number of attached slices. Thus, the ICU, the LCU, and the local registers build our scalable register file.

3. TOOL SUPPORT FOR ALGORITHM MAPPING

To map algorithms onto this scalable architecture tools are required to allow both: extracting the number of parallel slices and mapping algorithms onto this scaled architecture. Thus, our tools

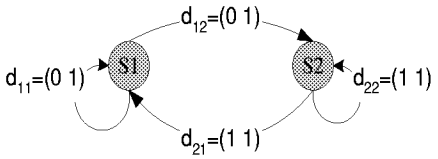

Fig. 2: Reduced Dependency Graph *(RDG) of the Lattice FIR*

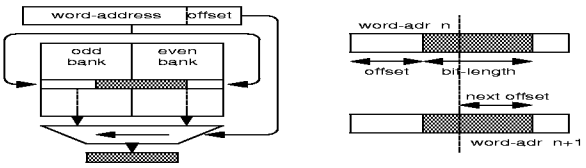

Fig. 3: *Bit extraction over memory boundaries*

are not dedicated to one specific architecture but are inherently scalable.

In [9] we presented methods to expand and program a standard DSP architecture. Using the same methods to program the sliced architecture also, we can provide software compatibility. Here, loop transformations can be applied to all algorithms, which have affine recurrence index functions. Thus, they determine *data transfer* but are independent from *data manipulation*. An example of this algorithm class is the lattice FIR algorithm. It is defined as follows:

$$\begin{aligned}&\textbf{for} \quad i_1 = 0 \textbf{ to } I_1, \textbf{for} \quad i_2 = 0 \textbf{ to } I_2\\ &\quad S_1: \; y_1[i_1, i_2+1] = F_1(y_1[i_1, i_2], x_1[i_2+1], y_2[i_1-1, i_2])\\ &\quad S_2: \; y_2[i_1, i_2+1] = F_1(y_2[i_1-1, i_2], x_1[i_2+1], y_1[i_1, i_2])\\ &\textbf{end for, end for}\end{aligned}$$

where $F_1(a,b,c) = add(a, mult(b,c))$. For this algorithm a *Reduced Dependency Graph* with dependencies $d_{S_1,S_2} = (i_1, i_2)$ can be derived as depicted in Fig. 2.

If we assume an architecture with M slices, each memory access provides M elements. Since state S_1 has no dependencies in i_2 direction, we can employ M independent slices to compute state S_1. Thus, at iteration $i_2 = 0$ groups

$$Y_1^{\text{Gr-1.1}} = y_1[m\varepsilon(0,..,M-1), i_2] \text{ and}$$
$$Y_2^{\text{Gr-1.1}} = y_2[m\varepsilon(-1,..,M-2), i_2]$$

can be read. These groups are used in state S_1 to compute the new group

$$Y_1^{\text{Gr-1.2}} = y_1[m\varepsilon(0,..,M-1), i_2+1],$$

while in state S_2 the new group

$$Y_2^{\text{Gr-1.2}} = y_2[m\varepsilon(0,..,M-1), i_2+1]$$

is computed. In the next iteration $i_2 = 1$ groups

$$Y_1^{\text{Gr-2.1}} = y_1[m\varepsilon(0,..,M-1), i_2+1] \text{ and}$$
$$Y_2^{\text{Gr-2.1}} = y_2[m\varepsilon(-1,..,M-2), i_2+1]$$

are required. Since group $Y_1^{\text{Gr-2.1}} = Y_1^{\text{Gr-1.2}}$ data can be read in *Vector Mode Data Transfer*. In contrary, group $Y_2^{\text{Gr-2.1}}$ and $Y_2^{\text{Gr-1.2}}$ are differently aligned. Here, group $Y_2^{\text{Gr-1.2}}$ has to be shifted by one element to the right. This can be done via a *Sliding Window Data Transfer*. In the RDG this can be seen by the dependencies $d_{S_1,S_2} = (1,1)$. Thus, within the loop we have to read only two groups to keep all datapaths busy. This allows for high data throughput.

4. I/O PROCESSING

I/O-processing is typically employed to provide data to the processing units and to fetch the results. Especially, if parallel processing is applied, the amount of cycles spent for I/O transfer can become dominating, since this transfer is time dependent, e.g. serial data arrive at fixed sample rates. A comparison of cycles needed for signal- versus I/O-processing shows a significant deterioration of the possible signal processing performance. Therefore all I/O related tasks should be swaped out into a special unit.

4.1. I/O Requirements

Often, in different types of data streams (e.g. JPEG, ADSL, ATM) data is grouped in frames where each bit position has a special semantic (e.g. the first 8 bit provide control information, the second 3 bits determine the frame length, etc.). Here, two main tasks can be distinguished:

1. Tasks for *Data Manipulation*:
   - detecting/creating of control information,
   - mapping data sets to the requirements of the DSP core (8,16,32 bit wide data).
2. Tasks for *Data Transfer*
   - handling of communication protocols (e.g. INIT sequence for transmission),
   - handling of parallel tasks (e.g. switching between 2 different data streams),
   - controlling resource utilization (e.g. prioritizing memory accesses).

Typically, these tasks are handled by embedded micro controllers. However, they are not suited for all data widths. Currently, common data sizes are 8, 16, or 32 bit, but in many data streams these sizes are not fixed at all. They vary like in ADSL's *Tone Ordering* procedure, where tones (or carriers) are ordered according to the number of data bits assigned to each of them. Here, data width starts with 1 bit followed by 2 bits, 3 bits, etc.. The actual mapping-scheme is sent as a table during the connection setup. However, for signal processing the tones are needed in ascending order [10]. A reordering is necessary. In *Advanced RISC Machines'* ARM7 processor the instruction contains a shift field for data alignment before the actual operation is performed. However, this method cannot be applied, if a bit sequence resides between the data boundary. Here, an expensive combination from two different sources is necessary. Thus, a new kind of I/O processors is required for this parsing and low-end protocol task.

4.2. Bit wise Addressing

To overcome this problem we introduce a new concept of bit wise addressing. Since a pure bit by bit addressing is too expensive, we work on byte wise addressing also. Here, we interpret the lower 4 address bits as an offset within 2 consecutive data words. The alignment is performed by a circular shifter (Fig. 3). In case of *Tone Ordering* this principle strongly simplifies the procedure. To access a bit sequence the address is split into 2 fields. The lower field contains the bit position in a word while the upper field determines the word itself. To calculate the address of the next element only the bit length to the previous offset is necessary. In case the data boundary is exceeded an overflow occurs which automatically increments the word address.

Table 1: Benchmarks for the M^3-DSP vs. TI's C6x, HiPAR, and Butterfly DSP

Algorithms	M^3 DSP @100 Mhz	TI's C60 @ 200 Mhz	HiPAR @ 100 Mhz	Butterfly DSP @ 50 Mhz
1024 complex point FFT (Radix-2)	2200 cycles/22 μs	20815 cylces/104 μs	42 μs	54 μs
complex FIR, 32 coeff.,100 samples	1204 cycles/12 μs	6410 cycles/32 μs	N.A.	N.A.
Lattice FIR, 8 coeff.,128 samples	268 cycles/2.7 μs	1546 cycles/8 μs	N.A.	N.A.
BCH code(216,124,25)	244 cycles/2.4 μs	N.A.	N.A.	N.A.

4.3. Context Switching

Furthermore, the ability to switch fast between tasks is required. The introduction of several register banks is one method but limits the number of parallel tasks to the number of parallel banks - or a costly register saving has to be executed. Thus, we decided to employ a register file where each register can be used as *program counter* (PC). To select the current PC a circular table is implemented. Here, each entry points to this register, which contains the current PC. Using these principles we provide an I/O controller based on a load/store architecture with a data width of 16 bits as an integral part of our DSP architecture.

5. APPLICATION AND RESULTS

As an implementation example for our sliced DSP architecture we chose 16 parallel slices, each consisting of one MAC unit, 4 input registers, one accumulator, and a scaling unit, which allows to apply block floating mechanisms [11]. This is the first realization of our processor architecture called, *Mobile Multimedia Modem-* or M^3-DSP [12], which must be able to provide up to 3000M MAC/s running at 100 Mhz. It allows for a complete software implementation of a new OFDM-based 25Mbit/s Hiperlan wireless ATM modem. With the help of our new architecture the M^3-DSP clearly outperforms current DSP solutions as TI's C60x, high end mediaprocessors as the HiPar DSP, and achieves similar performance to customized high-end ASICs as the Butterfly DSP (Table 1). Due to the ability to tailorize the DSP architecture to specific application needs, this concept opens the road for software based solutions of problems which are currently considered *ASIC-only* domains.

6. CONCLUSIONS

In this paper we presented a new scalable DSP architecture for System on Chip applications. By employing a group memory and group registers, the DSP architecture can be split into slices. Thus, according to system's needs the number of slices can be adapted, which allows this DSP architecture to suit different system requirements. Furthermore, we showed methods for algorithm mapping. Due to the high data throughput new concepts for I/O processing were designed by defining a dedicated I/O processor, which is an inherent part of our DSP architecture.
In our future work we will develop an integrated design environment, which supports the DSP design at system level. This should enable system designers to explore the DSP's design space already in an early phase of system specification.

7. ACKNOWLEDGEMENT

We like to thank our colleagues for their support, in particular Wolfram Drescher, Dirk Fimmel, Shiro Kobajashi, Menno Menenga, Thomas Richter, Attila Roemer, Paul Schwann, and Ulrich Walther.

8. REFERENCES

[1] K. Kim, R. Karri, and M. Potkonjak, "Synthesis of application specific programmable processors," in *Proc. of DAC '97*, pp. 353–358, 97.

[2] J. Wittenburg *et al.*, "HiPAR-DSP: A parallel VLIW RISC processor for real time image processing applications," in *Proc. of ICA3PP '97*, pp. 155–162, 97.

[3] M. Trenas, J. Lopez, and E. L-Zapata, "A memory system supporting the efficient SIMD computation of the two dimensional DWT," in *Proc. of ICASSP '98*, 98.

[4] C. Hansen, "MicroUnity's mediaprocessor architecture," *IEEE Micro*, vol. 16, pp. 34–40, Aug 96.

[5] G. Fettweis, "Design methodology for digital signal processing," in *Proc. of ASAP '97*, (Zurich, Switzerland), 97.

[6] W. Drescher, M. Mennenga, and G. Fettweis, "An architectural study of a digital signal processor for block codes," in *Proc. of ICASSP '98*, vol. 5, (Seattle, WA, USA), pp. 3129–3133, May 98.

[7] G. P. Fettweis, "DSP cores for mobile communications: Where are we going ?," in *Proc.of ICASSP 97*, vol. 1, pp. 279–/283, 97.

[8] M. H. Weiss, U. Walther, and G. P. Fettweis, "A structural approach for designing performance enhanced DSPs: 1-MIPS GSM fullrate vocoder case-study," in *Proc. of ICASSP 97*, vol. 5, pp. 4085–4088, IEEE, Apr 97.

[9] M. Weiss *et al.*, "Using loop transformations to optimize memory accesses and register allocation in ASDSPs," in *Proc. of SDA98*, pp. 9–17, Feb 98.

[10] Alcatel, *Standards project for interfaces relating to carrier to customer connection of ADSL-equipment*, Sept. 97.

[11] S. Kobajashi and G. P. Fettweis, "A block-floating-point system for multiple datapath DSP," in *Proc. of SiPS '98*, (Boston, MA, USA), Oct. 98.

[12] G. Fettweis, M.Weiss, W.Drescher, U.Walther, F.Engel, and S.Kobayashi, "Breaking new grounds over 3000 MOPS: A broadband mobile multimedia modem DSP," in *Proc. of ICSPAT98*, (Toronto, Canada), Sept. 98.