A SELF-CONTROLLED AND DYNAMICALLY RECONFIGURABLE ARCHITECTURE*

Florian Dittmann, Achim Rettberg
University of Paderborn, Germany
roichen@upb.de, achim@c-lab.de

Abstract: Reconfigurable systems have the potential to combine the performance of ASICs with the flexibility of software. The architecture presented in this paper offers a new concept for reconfiguration by operating self-timed and self-controlling. Data is routed together with its control information in a so-called packet through the operator network to make local decisions concerning the behavior of the network. Therefore, we can realize different paths without a central control unit. In this paper, we describe the architecture from the aspect of reconfiguration. An example shows the architecture in practical operation.

Keywords: High-Level Synthesis, Reconfigurable Architectures, Embedded Systems.

1. INTRODUCTION

Nowadays processors work with clocks running in gigahertz and are programmable to execute all imaginable software programs. The flexibility is bought dearly by high power consumption and goes along with barely influenceable possibilities to use the available parallelism of algorithms. In contrast, ASICs provide high parallelism at low power consumption, yet only for fixed algorithms. Both concepts only partly fit in the requirements of data processing today. E. g. mobile devices demand for low power consumption and real-time data processing. Furthermore, all existing and in future arising standards should be supported. Such devices need the combination of the performance of ASICs with the flexibility of General Purpose Processors (GPP), more precisely their software.

Reconfigurable systems and their concepts address this problem area [2]. In such systems, existing modules are reused for other tasks and dynamically adjusted for current requirements. This procedure is supported by FPGAs. FPGAs are no longer only programmable at the beginning of appropriate applica-

*This work was partly funded by the *Deutsche Forschungsgemeinschaft (SPP 1148)*

tions; they can be partial reconfigured during operation. At the same time alternative architectures, like the PACT XPP (eXtreme Processing Platform) [7] or Quicksilver's ACM (Adaptive Computing Machine) [9] come on the market.

Thus, technical opportunities for dynamic reconfiguration are given, being constantly improved, and optimized. Besides the basic technical aspects, we need additional methods to realize simplified and flexible automated and concrete design cycles. By considering all the given arguments, it is not desirable to use centralized control processes, which represent a complex energy and area consuming control unit.

The MACT (Mauro, Achim, Christophe and Tom) architecture [11–13] developed at the University of Paderborn describes a concept to flexibly decentralize considerable tasks of reconfiguration by self-controlling. Necessary information for reconfiguration exists locally due to the combination of control information and data word. It is possible to adapt the processing of each data packet according to individual requirements. Required operators are requested and released. The clear identification of data allows direct and serial processing of different data.

In this paper, we firstly describe related work including the development of the MACT architecture and all necessary operators. Secondly, we give an overview of the requirement analysis for the data packet of the MACT architecture w. r. t. reconfiguration. Finally we present an example with an application that shows how MACT realizes the adaptation to exogenous effects.

2. RELATED WORK

The PACT XPP, which addresses coarse grain reconfiguration, resembles the MACT architecture. A typical realization of the PACT XPP is similar to an array with processing nodes. The nodes are always alive, yet the functionality is dynamically changeable. The configuration respectively the reconfiguration is implemented by an appropriated flow. Therefore, it is essential to transform the instruction flow of a GPP into a configuration flow. This flow is mapped into a control flow graph representing the alternating configuration of the processing nodes over time. The control flow graph is executed sequentially.

The ACM approach by Quicksilver is based on run-time reconfigurable PLDs (Programmable Logic Device). Technical details concerning the implementation are hardly available. The main goal is to include algorithmic concepts in the architecture. Several ACMs are switched together according to requirements of the algorithms.

The MACT architecture is close to the concept of dataflow computation [14], as it is based on dataflow graphs. Further, concepts of dataflow computation often base on the inter-digitations of control- and dataflow. There, the processed data is combined with tokens to symbolize the status of affiliation.

The operators are triggered by the tokens evaluated from compare elements leading to demand-oriented execution similar to the MACT architecture. Synchronization at not unary operators does not have to be planned by a compiler in advanced, rather it is implicitly contained in the architecture.

A self reconfigurable platform based on FPGAs is described in [1]. In this example, the reconfiguration is executed by a microprocessor. The overhead of reconfiguration w. r. t. to power consumption is characterized in [10]. In this approach, the time of reconfiguration is reduced by pre-fetching. This leads to a more compact schedule.

3. MACT ARCHITECTURE

We describe the MACT architecture from two points of view. On one hand, the architecture is similar to the concept of the Internet on hardware level. On the other hand, it can be interpreted as a systematic approach for bit-serial calculation.

Within the Internet, data is transported without a central control element. All necessary information is transmitted in packets, with decisions done locally by routers or switches. Arriving packets activate nodes. These nodes decide how to proceed according to temporal circumstances. MACT uses a similar concept to transmit on dataflow level. This concept is especially suitable for processing of streaming-data. The architecture consists of an operator network derived from a dataflow graph. Data-words are assembled with suitable meta information and sent into the operator network. Each data-word activates self-controlled the next operator and ensures a minimal distance between the following packets. Valid data-words are alternated with minimal idle times. Thus, the processing of data within the MACT architecture is free of deadlocks and similar to operating in waves. Meta information of the packets is evaluated at specific points in the operator network to select the path.

Bit-serial processing is characterized by small operators, less area usage, low number of I/O pins, but higher latency in opposite to parallel calculation [3]. MACT is a systematic approach to process data bit-serial. Data of the addressed application classes, like control algorithms or signal processing, often is bit-serial. Therefore, conversion is not necessary. Further, we use advanced operators (e. g. [4]) and pipelining to considerably reduce the latency.

Data packets are transmitted synchronously into the operator network. The network can be interpreted as a clocked shift register. Due to the fixed length of data packets, scanners allow a precise evaluation and modification of the information at any time. Typical operators are addition, multiplication, etc. that are cascaded and assembled directly. Therefore, there exists no buffer storage. The local self-control cares for necessary synchronization of not unary operators, provoked by different path length.

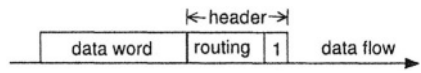

Figure 1. Data packet

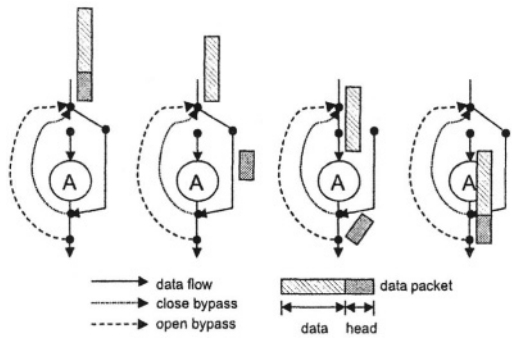

Figure 2. Bypass

3.1 Realization

The functionality of the architecture is exemplary described by means of a data packet (see Fig. 1). Valid data packets consist of a leading flag ('1') in front of the data. The flag is needed to identify arriving packets at operators. Furthermore, we use the flag for local control. Additional information for routing in the operator network is stored between the leading bit and the data word. The bit length of the header information (flag and routing information) and the data word length are fixed at implementation. As mentioned before, such data packets are transported bit-serial on one wire.

When packets arrive at arithmetic operations, header information and data word have to be split, as the arithmetic operation should not process the head information. This is realized by so-called *Bypass* signals. The *Bypass* is implemented parallel to the operator (see Fig. 2). Initially the *Bypass* is active. When the flag of the header information reaches a specific point in the network the *Bypass* is deactivated and the following part of the data packet (data word) is directed to the operator. The *Bypass* is set to the active state when the data packet is outside the operator, again signaled by the leading flag of the header information. Details concerning the implementation of the *Bypass* can be found in [5].

It is necessary to synchronize the dataflow within the operator network at non-unary operators. We do this by *Stall* signals that are directed in opposite direction of the dataflow. These *Stall* signals can stop the dataflow. So-called *Synchronizers* in front of each non-unary operator implement this functionality. Yet, it is only necessary to delay a minimal amount of operators and not

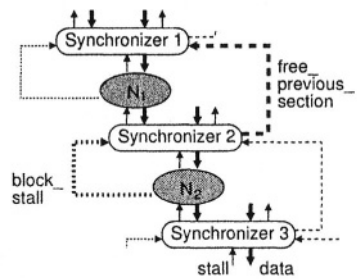

Figure 3. Synchronizer: A packet activates the *Block_Stall*signal at N_2, that is transmitted to Synchronizier 2. This one generates a *Free_Previous _Section*signal and transmitted it to Synchronizier 1. Consequently section N_1 is released.

the complete network. Only the block of operators actually processing the data packet is delayed. We implement the *Block_Stall* therefore. The signal is gripped from the shift register and directed to the corresponding *Synchronizers.*

Synchronizers having received a *Block_Stall* accept new data packets, but will not dispatch them. If necessary a *Stall* is produced. The reactivation of the block that is blocked by a *Block_Stall* is done by a *Free_Previous_Section* signal generated from a *Block_Stall* of the following operator block, see Fig. 3.

The interaction of the signals leads to local consistency of data and forces minimal distances to consecutive data packets. Therefore, the data processing is conflict free and operates in waves. The entire control is based only on local signals and not on long control wires from a central controller. This concept of locally based control elements realizes a deadlock free pipeline processing. The MACT architecture is the second approach implementing a deadlock free pipeline architecture (the interlocking problem) after that one from [6].

3.2 Router

We integrated routing nodes called *Routers* in order to be able to process similar to concepts of the Internet. The *Routers* evaluate the routing information in data packets and decide which paths are selected. We use such path decisions to assemble or to reload suitable operator networks. Further requirements for routers are compactness by high flexibility and minimal latency (decision delay). A first variant of router implementations are multiplexers. In this case, the routing information of data packets is used to set the parameters of operators. This leads to operators with integrated routing structures. If data packets achieve e. g. a constant multiplication with different hard implemented constants, it is possible to select the constant from the routing information grabbed by *Scanners.* This implementation style is useful for low numbers of constants in opposite to include constants within the data packets.

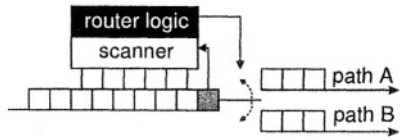

Figure 4. Router example

We achieve more freedom concerning the path selection by implementing the routers as independent elements. Fig. 4 shows a router with two paths. A *Scanner* detects after the arrival of the flag (leading '1', see grey shaded register) the head information of the data packet. Thus, the logic of the routers interprets the head information. Depending on the result, the corresponding paths are triggered.

The router concept tends to result in two levels of reconfiguration. The first level selects between available paths of the operator network with the existing head information. A higher-level reconfiguration is based on this path selection mechanism. At this level e. g. the realization on an FPGA leads to reloading specific parts of a dataflow graph. At this point, the router has to be equipped with an intelligent replacement strategy similar to caching methods to minimize the reconfiguration of the FPGA. Both concepts have enough potential to operate locally and individually for each data packet. In this case, a central control element to generate signals and to track the data is not necessary. Therefore, it is practicable to process sequentially different applications with different requirements in the same operator network.

We integrate *Scanners* on the shift register before routers. These *Scanners* track the head information independently from the following elements (operators, synchronizers, or delay elements). The leading flag of each data packet is used to control the grabbing time. Routers represent an additional element in the shift register. This further delay is used for the decision of the router logic. Thus, we achieve a short latency of one clock cycle and a small area usage.

4. RECONFIGURATION

In the previous section we described the routers and showed how a simple reconfiguration of the MACT architecture is achieved. In this section, we discuss the procedure in detail and examine the problem when using a surrounding system. Thereby, the MACT architecture offers a variety of possibilities for the implementation. Furthermore, we examine the possibilities towards practicability.

The goal to be reached for reconfigurable architectures is the realization of intelligent systems. Such systems have the capability to independently adjust the behavior and structure due to exogenous (environmental influences, user

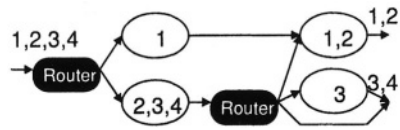

Figure 5. Network with two routers and four paths

interaction) and endogenous effects (ageing, component failure, altered target parameters, etc.). Systems have to be adapted permanently and efficiently to changing requirements without additional control overhead. Comparing this adaptation with a GPP, it is a context switch, which should be as short as possible, because most reconfigurable systems are used under real-time constraints. Reconfigurable architectures allow a context switch on hardware respectively wiring level. The dataflow is manipulated from outside by a control system. Therefore, an external and complex control element is necessary to track the data and to provide appropriate networks and follow-on operations.

MACT offers the possibility to reduce the overhead of a context switch by local presence of the control and routing information. Not a control element, but the data itself controls the way to the operators. Therefore, data is assembled with explicit identification to distinguish between each other. Thus, it is possible to have data from different applications in the same operator network. This offers the freedom to use paths and operators without a central control element. Furthermore, we realize a dynamical extension of algorithms simply according to given facts. This is achieved by assembling data and suitable head information before entering the network. The peripheral preprocessed logic is only responsible for the attachment and generation of the data packet. Further control tasks are decentralized and operate independently in the local control elements (synchronizers, routers).

5. IMPLEMENTATION OF THE SYSTEM

The implementation of the system should make profitable use of the characteristics of the MACT architecture. Thus, we distinguish between two tasks.

Firstly, we assume a hard implemented operator network with routers at suitable locations. These routers offer the possibility of multiple usages of areas of the graph (see Fig. 5). This kind of reconfiguration may be used for different coding standards like TDMA and GSM or refinements of compression algorithms. We code the path into the header of the data package. Concerning the example, we need 2 bits for the four possibilities. In general, the amount of n path possibilities can be realized with b_1 digits referring to $2^{b_1} = n$. The logarithmic dependency realizes many different paths to be coded by few bits.

Further, if data from different applications is intended to use the same path, the header must be extended with additional identification. The same formula

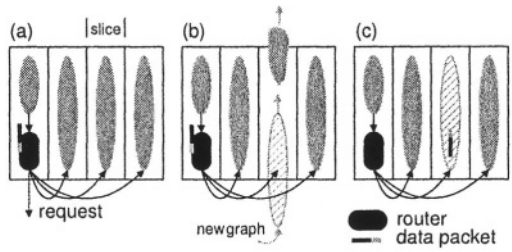

Figure 6. Reconfiguration: In situation (a) a packet reaches a router on an existing path stored in a slice. The router sends a *Request*-signal so that the slice is configured with a new operator network (b). Then the packet is routed to the network (c).

leads to an amount of $m = 2^{b_2}$ possibilities for identification with b_2 bits. Here, we can combine both the path and identification information, as the first is needed inside the network, the latter after exiting the network. Each individual case demands for special care taken concerning possible double usage of bits. E. g. if there are less than 2^{b_1}, but more tan 2^{b_1-1} paths, one of those b_1 bits may be used for the additional identification of the different packets.

If path decisions are located early in the data flow network, we realize a further way of optimization. Parts of the header can be removed after the path decision is done, leading to shorter data packets for the ongoing processing.

We modify the above explained system in order to be able to adapt to new requirements. These new requirements are new versions of processing standard or complete new algorithms for data processing. In most cases, we will have to modify and add paths or exchange operator nodes. We can think of 3G and 4G mobile communication problems.

In order to be able to reconfigure hardware, there has to be the technical condition. We assume an FPGA, which contains the whole MACT architecture. We have multiple possibilities to reconfigure the network. Firstly, the FPGA may be programmed completely. All calculations have to be stopped for this task. The duration may take up to several milliseconds and the package generation part must be reconfigured. Now, we try to avoid these disadvantages.

FPGAs are dynamically partial programmable. It is possible to exchange parts of the circuit during operation. Still, this takes some time. We try to reduce the requirements for the reconfiguration, in order to speed up the reconfiguration phase. Therefore, we adapt the MACT architecture to these requirements. Basically, possible branches i. e. new paths can be found directly after routers. Thus, we extend routers with the possbility to request new paths. The header of the data packet is extended by one additional bit, which signals the need to establish a new path or not.

In order to be able to omit a central controller, the router itself will generate a data package that requests the new path. This data package consists

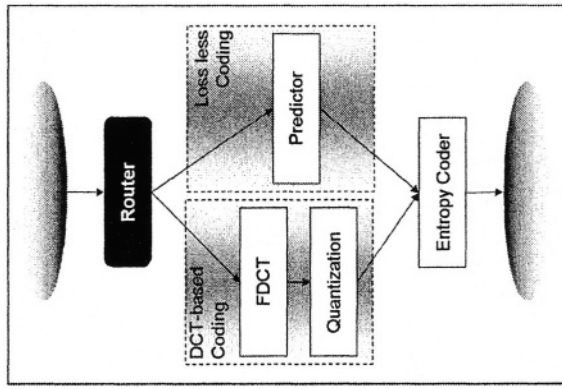

Figure 7. JPEG coding

of a location address (a number of slices of the FPGA) for the new operators. Thus, the reconfiguration data can be retrieved. The data packet generation unit buffered this reconfiguration information. To sum up, the process of reconfiguration is not controlled by a central controller, but organized locally and demand-oriented. Refer to Fig. 6 for a graphical example.

Using this order for the reconfiguration, we are able to reduce the control flow to a minimum. There is no need for a central controller, which would have to track the state of every data within the network. Again, different data packets of different applications can be calculated within one implementation of the MACT architecture. In order to reduce the busy waiting for new operators during the reconfiguration, we have planned to re-locate the scanners for the routers. Placing them earlier will mean additional time until the data will actually need the new path, thus hiding reconfiguration.

6. EXAMPLE

As an example we present a part of the JPEG algorithm [8]. JPEG compresses images according to individual quality requirements. Fig. 7 shows a relevant part of the JPEG algorithm. Following a preprocessing, it is possible to either choose a lossy DCT (Discrete Cosine Transformation) or a lossless coding, before there is the entropy coding or compressing. We find different paths within the JPEG algorithm, initialized by a router. According to the meta information in the header, each data packet gets assigned the correct paths by the routers. Thus, data of different images with different requirements can be found in the same network.

The example JPEG consists of multiple possibilities to realize reconfiguration with the MACT architecture. This is especially true for an easy realization of dynamically adoptions of the data flow graph to new requirements.

7. CONCLUSION

In this paper we have shown how a new bit-serial self-controlled architecture can be used to easily realize the reconfiguration of signal processing systems. This architecture, the MACT architecture, operates with data packets which carry all relevant information and thus offer the possibility of local controlling. Especially path decisions changing the data flow of the signal processing can be realized by referring to the meta information. In doing so, the functionality of the circuit is adapted dynamically, or even reconfigured completely. Further work will consider the effectiveness of the described system.

REFERENCES

[1] B. Blodget, P. James-Roxby, E. Keller, S. McMillian, and P. Sundararajan. A Self-reconfiguring Platform. In *Proceedings of the International Conference on Field Programmable Logic,* Lisbon, Portugal, Sept. 2003.

[2] K. Compton and S. Hauck. Reconfigurable Computing: A Survey of Systems and Software. *ACM Computing Surveys,* 34(2): 171–210, June 2002.

[3] P. Denyer and D. Renshaw. *VLSI Signal Processing: A Bit-Serial Approach.* Addison-Wesley Publishing Company, 1985.

[4] F. Dittmann, B. Kleinjohann, and A. Rettberg. Efficient Bit-Serial Constant Multiplication for FPGAs. In *Proceedings of the 11th NASA Symposium VLSI Design,* May 2003.

[5] F. Dittmann, A. Rettberg, T. Lehmann, and M. C. Zanella. Invariants for Distributed Local Control Elements of a New Synchronous Bit-Serial Architecture. In *Proceedings of the Delta,* Perth, Australia, 28 - 30 Jan. 2004.

[6] H. M. Jacobson, P. N. Kudva, P. Bose, P. W. Cook, S. E. Schuster, E. G. Mercer, and C. J. Myers. Synchronous Interlocked Pipelines. In *8th International Symposium on Asynchronous Circuits and Systems (ASYNC 02),* Apr. 2002.

[7] PACT. The XPP White Paper. PACT Informationstechnologie GmbH, 2002.

[8] W. B. Pennebaker and J. L. Mitchell. *JPEG: Still Image Data Compression Standard.* van Nostrand Reinhold, New York, 1993.

[9] Quicksilver. Technology Backgrounder. Quicksilver Technology, 2000.

[10] J. Resano, D. Mozos, D. Verkest, S. Vernalde, and F. Catthoor. Run-Time Minimization of Reconfiguration Overhead in Dynamically Reconfigurable Systems. In *Proceedings of the International Conference on Field Programmable Logic,* Lisbon, Portugal, Sept. 2003.

[11] A. Rettberg, M. C. Zanella, C. Bobda, and T. Lehmann. A Fully Self-Timed Bit-Serial Pipeline Architecture for Embedded Systems. In *Proceedings of the Design Automation and Test Conference (DATE),* Munich, Germany, Mar. 2003.

[12] A. Rettberg, M. C. Zanella, T. Lehmann, and C. Bobda. A New Approach of a Self-Timed Bit-Serial Synchronous Pipeline Architecture. In *Proceedings of the Rapid System Prototyping Workshop,* San Diego, CA, USA, June 2003.

[13] A. Rettberg, M. C. Zanella, T. Lehmann, U. Dierkes, and C. Rustemeier. Control Development for Mechatronic Systems with a Fully Reconfig. Pipeline Architecture. In *Proc. of the 16th Symposium on Integrated Circuits and System Design,* Sao Paulo, Brazil, 2003.

[14] T. Ungerer. *Datenflußrechner*. Teubner Verlag, Stuttgart, 1993.