Path Concepts for a Reconfigurable Bit-Serial Synchronous Architecture *

Florian Dittmann[1], Achim Rettberg[2], and Raphael Weber[2]

[1] University Paderborn/HNI, Fuerstenallee 11, 33102 Paderborn, Germany
[2] University Paderborn/C-LAB, Fuerstenallee 11, 33102 Paderborn, Germany
roichen@upb.de, achim@c-lab.de, syne@upb.de

Abstract. This paper develops path concepts for the execution of different algorithms on a reconfigurable architecture. New architecture concepts demand for permanent evaluation of such extensions, also including validating case studies. The recently patented synchronous bit-serial pipelined architecture, which we investigate in this paper, comprises synchronous and systematic bit-serial processing without a central controlling instance. It targets future high speed applications due to the abdication of long wires. The application specificity of the basic version of the architecture can be overcome by so called routers, achieving a reconfigurable system. This paper focuses on the difficulty to conceptualize these routers and proposes several variations for implementation. The case study, which comprises a combined version of the FDCT/IDCT algorithm, serves as an application example for the reconfigurability of the architecture. The example – implementing both algorithms in one operator network – broadens the application area of the architecture significantly.

1 Introduction

The bit-serial architecture (referred to as MACT – Mauro, Achim, Christophe and Tom), which we examine and extend in this paper, was invented in response to current problems of chip design. MACT combines ideas of asynchronous design and bit-serial processing, and represents a synchronous and pipelined architecture without central controlling instance.

Implementations of the so far presented version of the MACT architecture would be application specific [1, 2]. This limitation is only partly in line with current market conditions of the intended application area: data-stream oriented processing (e. g., image compression or filtering).

In this paper, we investigate how to realize multiple paths, and how to model and implement routers and multiplexors to enable path merging and path selection. We abstract and conceptualize graph variations capable for path selection. Strict categorization helps us to distinguish possible cases and to develop solutions for the high level synthesis of such elements. Thereby, we can fall back on

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

an already realized high level synthesis for the architecture [3, 4], which automatically generates MACT implementations in VHDL out of data flow graphs. In the high level synthesis, we avoid deadlocks and establish the local control mechanism of the MACT architecture.

As an example for the effectiveness of the extended MACT architecture, we implemented parts of the MPEG-2 algorithm with the components of our architecture.

The rest of this paper is organized as follows. First, we resume related work. Next, we shortly explain the MACT architecture, summarizing the main aspects and benefits of the architecture. Then, we systematically introduce concepts for multi paths within the MACT architecture, including detailed description of the extension. Section 5 familiarizes the reader with the example: combining IDCT and FDCT. Section 7 sums up with a conclusion and gives an outlook.

2 Related Work

Adding routers to the MACT architecture means extending MACT towards a reconfigurable architecture. In the literature, we find several concepts, which can be related to the reconfigurable MACT architecture and give some basic information for our router conceptualization.

The concept of wormhole run time reconfiguration [7] relies on a distributed control scheme and therefore avoids central controllers. Wormhole run time reconfiguration is based on the stream concept. The idea is implemented in the Colt Configurable Computing Machine [8]. Our architecture relies on a similar concept. In contrast to wormhole reconfiguration, we transport only the information when and which path to select and not the whole reconfiguration stream.

So-called self-reconfiguration [9] can be seen as one step towards easing the control process, as the part of the controller tracking the processing can become obsolete. Yet, advanced concepts are needed. One idea is to locate the state machine inside the FPGA, either in a soft- or hard-core processor. Considering Xilinx's Virtex FPGAs, both is possible using MicroBlaze or a PowerPC.

In [10], we find basic work for deadlock-free routing. The authors introduce virtual channels to achieve deadlock-freeness. In principle, MACT relies on a similar concept; yet, the deadlock-freeness is realized by the local control mechanism triggered by data packets traversing the architecture. Thus, the architecture itself bears the capability to avoid deadlocks. Deadlock avoidance is a critical issue during the high level synthesis especially if routers are added to the MACT architecture (see below).

3 MACT Architecture

MACT is an architecture, that breaks with classical design paradigms. Its development came in combination with a design paradigm shift to adapt to market requirements. The architecture is based on small and distributed local control

Fig. 1. Example data packet.

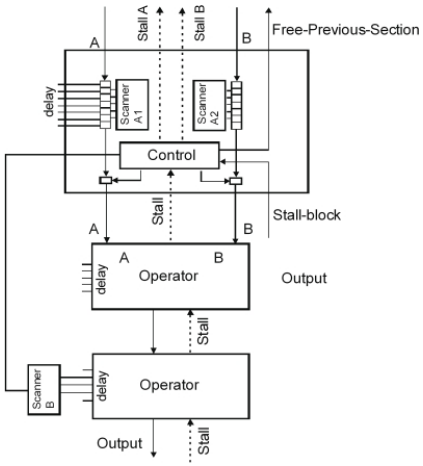

Fig. 2. Synchronisation

units instead of a global control instance. The communication of the components is realized by a special handshake mechanism driven by the local control units. MACT is a synchronous, de-centralized and self-controlling architecture. Data and control information are combined into one packet and are shifted through a network of operators using one single wire only (refer to Figure 1).

The controlling operates locally only based on arriving data. Thus, there exist no long control wires, which would limit the operating speed due to possible wire delays. This is similar to several approaches of asynchronous architectures and enables high operation frequency. Yet, the architecture operates synchronous, thus enabling accurate estimation of latency, etc. a priori.

MACT operates bit-serial. Bit-serial operators are more area efficient than their parallel counterparts. The drawback of bit-serial processing is the increase in latency. Therefore, MACT uses pipelining, i. e., there exist no buffers, operators are placed following each other immediately.

Thus, MACT resembles a systematic bit-serial architecture. It enables the user to benefit from the advantages of bit-serial processing like low area consumption, significant reduction of I/O pins (serial instead of parallel communication), while offering a reliable pipelined system.

Further, processing increases to be more and more bit-serial. Systems nowadays increasingly use serial communication, while still processing in parallel. Using MACT enables integrated bit-serial processing, avoiding discontinuity concerning the bit-width, i. e., no parallel/serial conversion is needed.

Implementations of MACT are based on data flow graphs. The nodes of these graphs are directly connected, similar to a shift register. Synchronization of different path lengths at nodes with multiple input ports is resolved by a stall mechanism, i. e., the shorter paths, whose data arrives earlier, will be stalled until all data is available (refer to Figure 2). The necessary stall signals run in

opposite to the processing direction and are locally limited, in order to avoid a complete stall in the preceding pipeline. The limitation is realized by a so called *block stall* signal, which is locally tapped in a well defined distance.

We consider the flow of data through the operator network as processing in waves, i. e., valid data alternates with gaps. Due to a sophisticated interlock mechanism adapted from asynchronous design, the gap will not fall below an individual lower bound. Thus, the MACT implements a fully interlocked pipeline. In combination with the developed high level synthesis, the MACT guarantees deadlock free processing. The corresponding signal is the so called *free previous section* signal, which is generated by small logic in each synchronizing control block (see Figure 2). These control blocks are found at each multiple input operator and synchronize packets arriving at different instances of time. The architecture is described in more detail in [1, 2].

4 Router Design

Specialized routing components are further extensions of the architecture to allow data-driven reconfiguration. Therefore, we interweave data and configuration information by processing packets including both. So far, the routers only have been conceptualized in [11]. In order to efficiently use routers in the MACT architecture, this section introduces the routing concept abstractly.

Routers are placed inside the data flow graph and allow for different processing paths. Each data packet carries routing information or information for unique identification, thus enabling path selection. The routers process the header and/or the data section of each data packet and forward the packet to the corresponding path. This feature allows space saving and flexible implementations of data flow graphs, as multiple algorithms or different characteristics of algorithms may be present in the same implementation of MACT. As example, we may select different compression granularity or increase processing accuracy using router based MACT. The re-use of common sections reduces the overall area requirement. In the following, we formalize graphs containing routers abstractly.

4.1 Variants of multiple path folding into one operator network

There exist several variants how routers may be present in graphs, which we will categorize and explain below. In addition to routers, we have to integrate the corresponding counter part: multiplexors. Thus, the two operators, which enable multiple paths within one processing graph are routers (R) and multiplexors (M). Routers act like de-multiplexors.

Tree The simple tree case comprises two alternatives as displayed in Figure 3 and Figure 4. In the first, we only find routers, which distribute incoming packets to the appropriate output, while in the latter, we combine multiple inputs to one output via multiplexors.

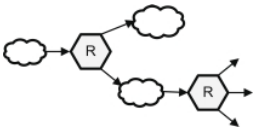

Fig. 3. Tree with one input and multiple output variations.

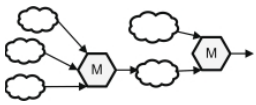

Fig. 4. Tree with variable inputs leading to one output.

Fork – Join In Figure 5 a, we display the possibility of alternative paths. There, data packets arrive at the router and are forwarded to one path according to their routing information. This kind of path option can be reasonable for data stream processing, where only parts are different, e. g., encoding granularity varies.

Join – Fork In contrast to the latter characteristic, Figure 5 b shows how similar sections within a processing algorithm can be utilized using MACT with routers.

Combinations The above mentioned basic characteristics can be combined in several ways. Apart from a random combination, there may exist an ordered structure, e. g., mesh-based. While random based versions require extended generation algorithms, mesh-based structures can base on principles of systolic arrays. These concepts are out of the scope of this paper.

4.2 Path Encoding

We can encode the information for path selection into several parts of the data packet (refer to Figure 1). Thereby, we can indicate different paths using only a few bits due to logarithmic dependency.

Header We can use the header to hold the routing information, i. e., the information for the path selection. This localization is the most appropriate, as headers precede in the data packet and their content is more easily accessible.

Data If we make a path decision based on the content of the data, we introduce if-then-else to the MACT architecture. We can use this style, when specific paths are only needed, after the data has passed a threshold, etc.

Header + Data When we use both section of a data packet, we can process both by the routers. Thus, we can base the path selection not only on the

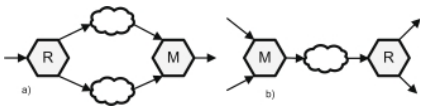

Fig. 5. a) Fork and Join, b) Join and Fork.

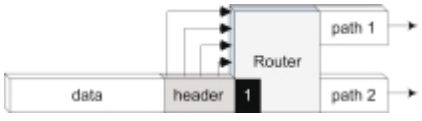

Fig. 6. Router implementation.

header information, but also on the data content itself. Thus, individual routers forward packets depending on either the control information present in the header, the data content present in the data section, or on both.

4.3 Router Implementation

Figure 6 shows an exemplary implementation of a router. Routers tap the information of approaching data packets, when the preceding '1' of every packet arrives at the router. Within one step (clock cycle), the router processes the information and subsequently forwards the data packet to the appropriate path based on predefined table entries. Yet, the paradigms, which lie behind the routers, can follow different principles.

Basic In the basic version, the router only reads the information, selects the appropriate path for the packet and forwards the packet without modifications.

Consuming In a more advanced concept, the router can consume parts of the information of the header, i. e., the corresponding section in the header is removed. Thus, we can decrease the packet length, which can lead into less area needed and shorter wire lengths.

Additive Further, routers may extend packets by additional information. This possibility makes sense, e. g., when there exists a short common section, and packets must be distinguished directly after this section.

4.4 High Level Synthesis and Extensions

Both routers and multiplexors represent new parts of the MACT architecture and therefore must be considered by the high level synthesis. Thereby, we can mainly rely on our already developed high level synthesis.

Router: As routers distribute packets to different paths only, and therefore do not affect preceding paths, they can be treated as normal operator elements by the high level synthesis. In detail, they accept new packets, when all possible succeeding paths are empty, i. e., no packet is present within the tapping range of the control signals.

Multiplexer: In contrast, multiplexer re-unit data paths, i. e., they combine more than one line. The arrival of two packets at the same or nearly same time can cause problems due to false free path information.

The deadlock free processing of the MACT architecture relies on gap and data periods (processing in waves). In order to guarantee a minimal lower bound between two data packets, we enhance multiplexors by *block stall* signals (refer to Figure 7). These signals are activated as soon as one data packet reaches the area between the last synchronizer and the multiplexer of the current path. All other similar areas of the other paths preceding the multiplexer receive a block stall signal, and thus deny arriving packets to enter this section. The packet of the one valid path can be processed without interferences.

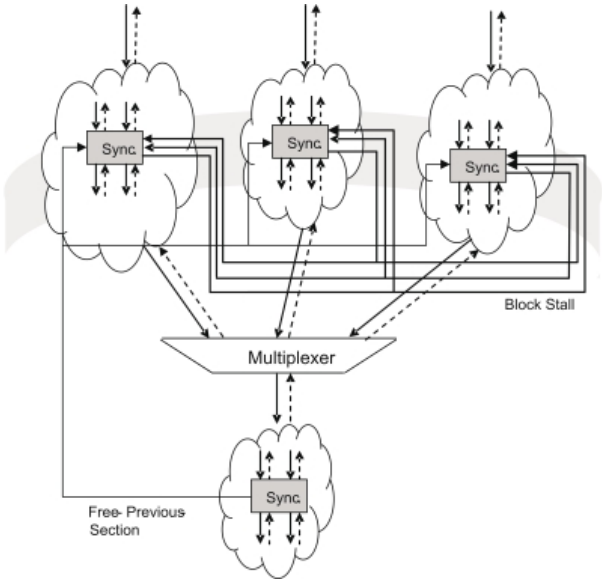

Fig. 7. Multiplexer realization

Yet, all sections must be freed again, which is done by the *free previous section* signal, which is generated by local logic when the data packet has exited the critical section. It is received by all synchronizers preceding the multiplexer and frees all corresponding paths.

Up-to-date technologies of chip design enable to increase the router concept further. So far, all possible paths must be known at implementation time. Routers enable dynamic path selection between predefined processing paths. Reconfigurable devices (e. g. Xillinx Virtex FPGAs) possess the capability of partial run time reconfiguration. Thus, we can add additional paths of the MACT architecture including routers to our system during run-time (using partial reconfiguration), i. e., new path alternatives can be realized.

We thus achieve a specific self-controlled run time reconfiguration. Generally speaking, during run time reconfiguration, tasks are dynamically loaded into the reconfigurable processing unit on demand. The area currently reconfigured does not influence other regions in operation at the same time. In the MACT version, we additionally avoid a usually large and complex central control entity. There is no need to track the data in the operator network by a central control unit, in order to activate reconfiguration. The request for reconfiguration is generated locally.

Therefore, we modify the routers of MACT. Upon arrival of a new packet, the modified router detects the information needed (header and/or data section) and checks for the availability of the required data path. If the needed path is not resident, the router activates the run time reconfiguration.

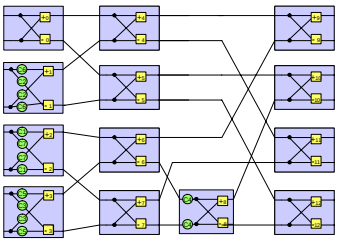

Fig. 8. Inverse Discrete Cosine Transformation

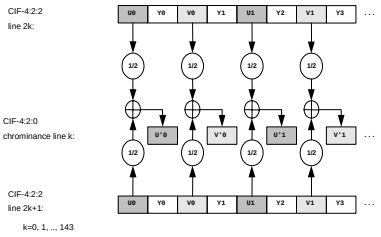

Fig. 9. CIF 4:2:2 to 4:2:0 conversion

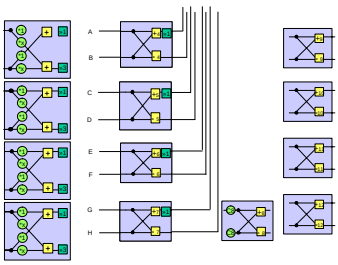

Fig. 10. CIF 4:2:2 to 4:2:0 conversion

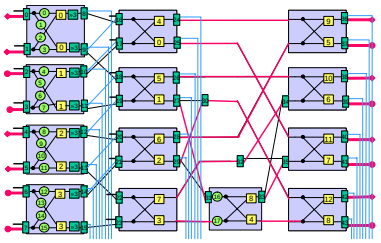

Fig. 11. Mapped Network

5 Case Study

We use as a case study some tasks of MPEG-2. These are the FDCT/IDCT and some video conversion formats. The FDCT/IDCT algorithm is implemented according to the Chen/Wang [5,6] approach. The task of the FDCT is to transform the luminance and chrominance data in a block-wise manner. The task of the IDCT is to re-transform the inverse quantized luminance and chrominance coefficients block-wise into the spatial domain. The input coefficients are provided as 12bit values by the control processor.

Obviously the IDCT depicted in Figure 8 is the reversed graph of the FDCT. Both consist of the same amount of operators, and both implementations use a similar operator network. Therefore, we map both graphs onto one common operator network. As a result, we get a re-configurable operator network. That means, we can use the operator network for encoding (FDCT) and decoding (IDCT) of a video image.

Lee [12] introduces an algorithm to split the N point FDCT/IDCT into two (N/2) point FDCTs/IDCTs. Therefore, to realize a 2D-FDCT/2D-IDCT we need eight instances of the FDCT/IDCT operator network. The throughput remains unchanged for the 2D-FDCT/2D-IDCT implementation.

Another algorithm is the video conversion CIF 4:2:2 to 4:2:0 (refer to Figure 9). The conversion CIF 4:2:2 to 4:2:0 is calculated by using the average value of the chrominance values. That means, previously, there are eight and after the conversion there are four values.

If we investigate the FDCT/IDCT network, part of the structure is similar to the CIF algorithm. We can use the second row to calculate the CIF conversion. Figure 10 shows the FDCT/IDCT network with the wires for the CIF algorithm. We place multiplexors before the operators, and routers at the output of the four relevant lines after the operators. Thus, we fold the CIF algorithm into the FDCT/IDCT network and receive the complete mapped network as depicted in Figure 11.

6 Results

Current implementations of the MACT architecture run on Xilinx's Virtex 400E FPGA. We have implemented all necessary basic components of the architecture to realize the described example.

Logic Utilization	used	total	perc.
No. of Slice FF	1897	9600	19%
Total no. 4-inp LUTS	1865	9600	19%
No. used as logic	1865	2393	78%
No. used as route-thru	528	2393	22%
Number of bonded IOBs	69	158	43%
IOB Flip Flops	21	69	30%
Number of GCLKs	1	4	25%
Number of GCLKIOBs	1	4	25%

Table 1. Logic utilization of the example for a Virtex 400E

Logic Distribution	used	total	perc.
No. of occupied Slices	1999	4800	41%
No. of Slices containing only related logic	1999	1999	100%
No. of Slices containing unrelated logic	0	2166	0%

Table 2. Logic distribution of the example for a Virtex 400E

We are analyzing the behavior of the system based on our own library. Additionally, we simulate the system. The high level synthesis produces appropriate code. The critical path of our example design needs 87 clock cycles. One cycle is 40 ns long. Therefore, the clock frequency is 25 MHz.

As MACT is a deeply pipelined design, the latency can be understood as setup time, which delays the system start only once. The latency of the example is 3480 ns (87 cycles multiplied with 40 ns). Further, we measure the throughput as average time between two output signals to determine the system speed. In the example, the throughput is equal to the clock frequency 40 ns. Taking into account that we run with 25 MHz we can achieve 25 MBit/s per wire.

The logic utilization and distribution are depicted in Tables 1 and 2. We can see that only 41 % of slices are used on the Virtex 400E.

7 Conclusion and Outlook

In this paper, we have presented concepts to realize routers for the MACT architecture. The concepts include variants of data encoding and router characteris-

tics. We have extended the existing high level synthesis for the MACT architecture to be able to cope with routers and multiplexors. Routers within the MACT architecture enable reconfiguration. They facilitate to use one implementation of MACT for different application areas. Thereby, the bit-serial MACT architecture provides configurable functionality on the level of arithmetic operations, high throughput rates, cost effective bit-serial operator design and short configuration cycles. The example of the IDCT/FDCT algorithm from Chen/Wang [5,6] demonstrates this effectiveness.

Further, we have propose an approach for reconfiguration, which decentralizes the control mechanism and thus results in short wire length. Thus, we will focus on future high-speed applications, where wire delay times affect the maximum processing clock.

References

1. A. Rettberg, T. Lehmann, M. Zanella, and C. Bobda, "Selbststeuernde rekonfigurierbare bit-serielle Pipelinearchitektur," Deutsches Patent- und Markenamt, Dec. 2004, patent-No. 10308510.
2. A. Rettberg, M. C. Zanella, C. Bobda, and T. Lehmann, "A Fully Self-Timed Bit-Serial Pipeline Architecture for Embedded Systems," in *Proceedings of DATE*, Munich, Germany, 3 - 7 Mar. 2003.
3. 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.
4. A. Rettberg, F. Dittmann, M. C. Zanella, and T. Lehmann, "Towards a High-Level Synthesis of Reconfigurable Bit-Serial Architectures," in *Proceedings of SBCCI*, Sao Paulo, Brazil, 8 - 11 Sept. 2003.
5. Z. Wang, "Fast algorithms for the discrete W transform and for the discrete Fourier transform," *IEEE Trans. Acoust., Speech, Signal Proc.*, vol. 32, Aug. 1984.
6. W. Chen, C. Smith, and S. Fralick, "A fast computational algorithm for the discrete cosine transform," *IEEE Transaction Commun.*, vol. COM-25, 1977.
7. R. A. Bittner and P. M. Athanas, "Wormhole Run-time Reconfiguration," in *Proceedings of the 1997 ACM/SIGDA FPGA*, Monterey, CA, USA, Feb. 1997.
8. R. A. Bittner, P. M. Athanas, and M. D. Musgrove, "Colt: An Experiment in Wormhole Run-Time Reconfiguration," in *Photonics East, Conference on High-Speed Computing, Digital Signal Processing, and Filtering Using FPGAs*, Boston, MA, USA, Nov. 1996.
9. B. Blodget, P. James-Roxby, E. Keller, S. McMillian, and P. Sundararajan, "A Self-reconfiguring Platform," in *Proceedings of FPL*, Lisbon, Portugal, Sept. 2003.
10. W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. 36, no. 5, pp. 547–553, May 1987.
11. 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 RSP Workshop*, San Diego, CA, USA, 9 - 11 June 2003.
12. B. G. Lee, "A New Algorithm to Compute the Discrete Cosine Transform," in *IEEE Trans. ASSP-32*, Dec. 1984.