DISCRETE WAVELET TRANSFORM ON DYNAMICALLY RECONFIGURABLE HARDWARE

Karel H.G. Walters, Gerard J.M. Smit, and Sabih H. Gerez

University of Twente
Department of Electrical Engineering, Mathematics and Computer Science
P.O. Box 217, 7500 AE Enschede, The Netherlands
Email: k.h.g.walters@utwente.nl

Abstract

The increasing amount of data produced in satellites poses a problem for the limited data rate of its downlink. This discrepancy is solved by introducing more and more processing power on-board to compress data to a satisfiable rate. Currently, this processing power is often provided by custom-off-the-shelf hardware. These architectures introduce problems when they are scaled up. To deal with the ever increasing data rate of the newest sensors and still be within the same power and thermal constraints, new architectures are being examined.

In this paper we will describe one of these new architectures, the Montium tile processor, and we show how a well-known algorithm, the CCSDS 9/7 integer *discrete wavelet transform* (DWT), can be mapped to this architecture. Furthermore, some improvements are discussed that can make the Montium and reconfigurable architectures in general, an even better platform to support algorithms like the DWT.

1. INTRODUCTION

To cope with the increasing complexity of algorithms and the data rates of sensors the satellite construction industry is looking into new architectures [1]. Near future solutions are often built around currently available hardware, such as the high performance platform for GAIA's video processing. One of the downsides of the platform is its power dissipation (33W), which causes thermal difficulties [2]. The space industry is therefore looking into new long term solutions [2].

One of the areas that is being explored for space type applications is the dynamically reconfigurable architecture. The research and industrial communities have been trying to fill the gap between the speed of ASICs and flexibility of FPGAs with dynamically reconfigurable hardware. The Montium tile processor [3–5] is such a dynamically reconfigurable architecture. We show the possibilities using the Montium with the use of the well-known DWT, as described in the CCSDS image compression standard [6]. In Section 2, the architecture is explained followed by the algorithm and mapping in Sections 3 and 4. Some recommendations for the architecture are discussed in Section 6.

2. ARCHITECTURE

The Montium was developed within the University of Twente to provide a low-power high-throughput architecture for streaming media applications, such as MPEG processing on a small mobile battery powered device. Since MPEG is an evolving standard, the architecture needed to support reconfiguration with minimal effort. Although on-board processing is not the Montium's initial application domain some of the issues that arise in streaming mobile applications are similar to those in on-board processing. Especially low-power dissipation, the ability of easy reconfiguration and high throughput are topics of intrest in on-board processing.

Currently the Montium is further developed within the company Recore Systems [7], a spin-off company of the University of Twente. The Montium targets the DSP algorithm domain. The Montium tile processor is parameterisable at design

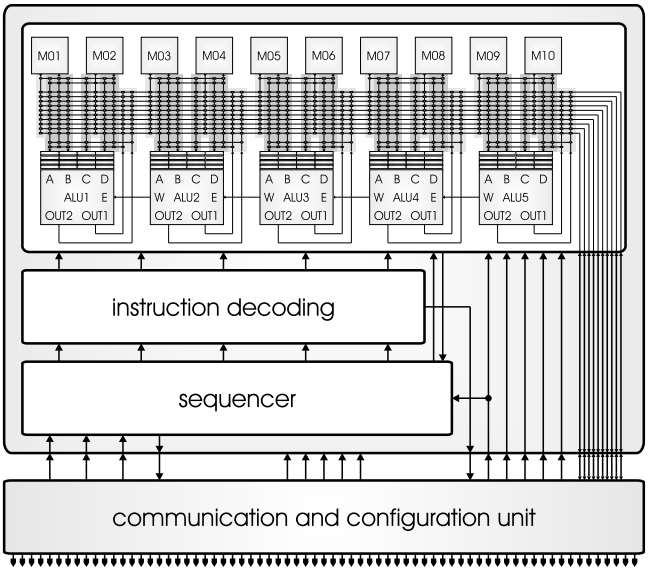

Figure 1. Montium tile processor.

time. So, at design-time the datapath width, number of ALUs, number of memories and the size of the memories can be specified. Figure 1 depicts a Montium tile processor with 10 memories and 5 ALUs. In the remainder of this paper we will consider a datapath width of 16-bit and a memory size of 1024 addresses.

At first glance the Montium architecture bears a resemblance to a *very long instruction word* (VLIW) processor. However, the control structure of the Montium is optimized to minimize the control overhead which is imperative for energy efficiency. The lower part of Figure 1 shows the *communication and configuration unit* (CCU) and the upper part shows the reconfigurable *tile processor* (TP). The CCU implements the interface for off-tile communication. The current implementation of the CCU provides four output and four input lanes off-tile. The off-tile interface depends on the interconnect technology that is used in the system-on-chip (SoC).

The TP is the computing part that can be configured to implement a particular algorithm. The five identical ALUs (ALU1 . . . ALU5) in a tile can exploit spatial concurrency to enhance performance. The data path of the ALUs has a width of 16-bits and the ALUs support both signed integer and signed fixed-point arithmetic. Each ALU is divided into two levels. Binary operations like SHIFT, AND, OR reside in Level 1, also operations like MIN, MAX ADD and SUB can be found here. Level 2 of the ALU provides the multiplication and the input and output to the other ALUs as well as another adder for accumulation purposes (see Figure 2).

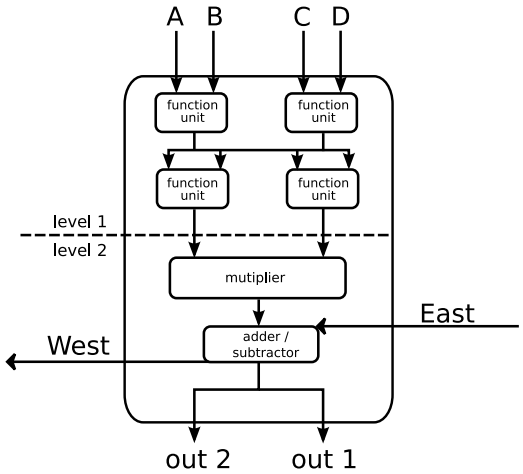

Figure 2. ALU diagram.

The five ALUs demand a very high memory bandwidth, which is obtained by having 10 parallel local memories (M01

...M10)l. The local memories imply a good locality of reference. A relatively simple sequencer controls the entire tile processor. The sequencer selects configurable tile instructions that are stored in the decoders (see Figure 1).

Each local SRAM is 16-bit wide and has a depth of 1024 positions, which adds up to a storage capacity of 16Kbit per local memory. A reconfigurable *address generation unit* (AGU) accompanies each memory. The AGU can generate the most frequently used address patterns, but when needed also an ALU can generate address patterns. It is also possible to use the memory as a lookup table for complicated functions that cannot be calculated using an ALU, such as "sine" or division (with a constant). A memory can be used for both integer and fixed-point lookups.

Each one of four 16-bit inputs to an ALU has a private input register file that can store up to four operands. The input register file cannot be bypassed, i.e. an operand is always read from an input register. Input registers can be written by various sources via a flexible interconnect. Two 16-bit outputs from each ALU are connected to the interconnect. Neighboring ALUs can also communicate directly: the West output of an ALU connects to the East input of the ALU neighboring on the left.

Multiple Montium tiles can be combined on an SoC connected through a network on chip (NoC). An SoC combining multiple Montiums and other IPs exists in the form of the Annabelle chip [4]. This SoC has been realized in 130nm technology. The Annabelle block diagram is depicted in Figure 3. More details can be found in [4].

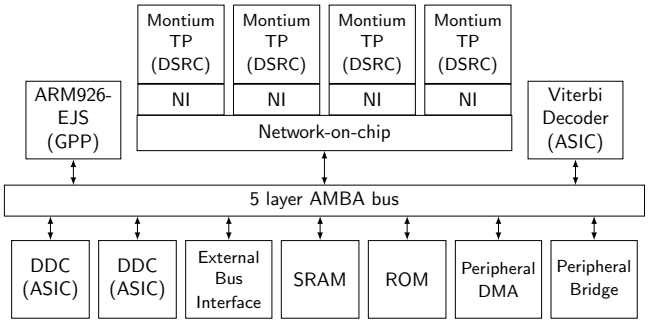

Figure 3. Annabelle SoC block diagram.

3. DWT

The discrete wavelet transform implements multi-resolution analysis using sub-band coding [8]. It can be used in many signal processing applications from feature extraction to signal smoothing. However, it is best known for its ability to decorrelate information in natural images. This ability has led to several image compression standards based on the DWT of which the JPEG2000 standard is probably the most well known.

An image is processed several times by the 2D-DWT, each time corresponding to a new *level*. The number of levels in standards varies but most of them process an image 3 to 5 times. A 3-level 2D-DWT image decomposition looks like Figure 4.

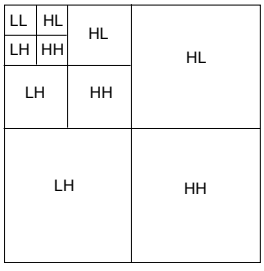

Figure 4. 3-level 2D-DWT decomposition.

Images are first processed row-wise after which they are processed column-wise. The output of a row and column are each divided in two parts. The part that corresponds to the low frequency components of the image and the part that corresponds to the high frequency components. This process is then repeated for the first quarter of the image until the

number of levels that is required. The first quarter of the image is the part that corresponds with the low frequency output of the row processing as well as the low frequency output of the column processing.

The DWT used here is the one described by the CCSDS 122.0 image compression standard [6] and depicted below in Equations 1 and 2. This is the lossless integer-to-integer transform which can be implemented on a fixed-point architecture. The standard also describes a lossy transform, which is not used due to the fact that floating point operations are not supported by the Montium. It is a relatively simple standard, it is well-described and open source implementations are available. These advantages allowed us to put more focus on the mapping possibilities and less on the actual compression itself.

In the equations below the variables C and D denote the output values of the scaling and wavelet equations respectively while X relates to the input and j denotes the index of that input.

$$D_j = X_{2j+1} - \left\lfloor \frac{9}{16}\left(X_{2j} + X_{2j+2}\right) - \frac{1}{16}\left(X_{2j-2} + X_{2j+4}\right) + \frac{1}{2} \right\rfloor \tag{1}$$

$$C_j = X_{2j} - \left\lfloor -\frac{D_{j-1} + D_j}{4} + \frac{1}{2} \right\rfloor \tag{2}$$

Equations 1 and 2 are the lifting implementations of the wavelet transforms described by the standard. They result in an in-place algorithm which makes it easier to implement on devices which have a limited amount of memory.

4. MAPPING

The algorithm is well suited to be mapped onto the Montium architecture. This is mainly because of the fractions ($\frac{1}{16}$, $\frac{1}{4}$), that can be mapped onto integers without loosing precision. The fixed-point representation of the Montium is $\langle 1.15 \rangle$ in which there is one sign bit and 15 fraction bits. The integer representation uses 16 bits in two's complement. For this algorithm, the integer representation has been chosen. By multiplying Equation 1 with 16 a full-integer representation of the equation is created. The fact that everything is now shifted with 4 bits to the left does have a downside. The maximum input value is now limited to 12 bits solely by the fact that any higher input value could cause an overlow. The result has to be shifted to the right by 4 bits to get the correct answer. For Equation 2 a similar action can be performed although the multiplication factor is now 4. The result now has to be shifted back by 2. This can now be mapped onto the datapath as shown in Figure 5.

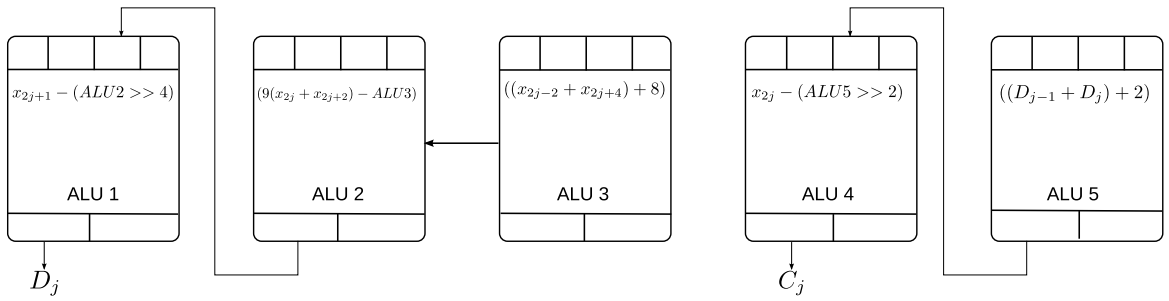

Figure 5. Mapping DWT

An *integrated development environment* (IDE) called *Sensation* [7], based on the Eclipse framework, is available for the Montium. It maps a specification given in the C-like language *MontiumC* to either executable code or a Montium binary file. The executable is mainly for debugging and testing purposes while the Montium binary only runs on the Montium processor. The compiler is still in development but the complete datapath of the Montium is supported. Equation 1 mapped on ALU [1 . . . 3] on the Montium looks like this:

```
word nine = from_int(9);
word eight = from_int(8);
word four = from_int(4);

word alu2 = p2o0(ssub_acc (imul (add(ra(X), rb(X_2)),rd(nine))),west( add(add(ra(X_2),rb(X_4)),rc(eight))));
word D = p1o0(sub(ra(X_1), asr(rc(alu2),rd(four))));
```

The variables X_* denote the input variables. Note that the expression on ALU2 uses input from ALU3 via the `west()` function. Similarly, `add()` denotes an addition and `r*()` denotes a register source.

Our implementation computes the complete 2D-DWT on a single processing tile. An image and in particular an earth observation image can be very large and would not be able to fit in the memory of the Montium. Therefore between each level of the DWT and even more important between each row and column iteration the entire image moves off and on chip. This is inefficient and should be avoided. Some implementations address this problem by first calculating the result of a couple of rows after which a single column output can be determined [9]. For small blocks of input samples this can also be done on the Montium but this would mean a break up of the image into smaller tiles. A general purpose processor or alike would need to do this which results in a large administration overhead.

The basic problem behind this is the transpose operation of rows to columns and vice versa. Employing multiple Montiums solves this problem. As can be seen in Figure 3, this is not a problem since multiple Montiums are available on our platform. A second Montium is used to transpose rows into columns and a third to do the column processing. Still the data needs to be moved to off-chip memory between processing of the different levels but this is something difficult to avoid without processing in a block-like fashion and thus employing a general purpose processor (GPP).

5. RESULTS AND RELATED WORK

The Montium computes both the scaling and wavelet coefficient of the CCSDS DWT in a single clock cycle. The operations are pipelined therefore it takes three lead cycles before the first output is computed and it takes four tail cycles until the last output of a row or column is output.

Mapping algorithms to the Montium is not that difficult. This algorithm took us only a few days to implement, mainly due to the fact that we are familiar with the architecture. Nevertheless people without any experience, students, were able to map algorithms like the Hilbert transformation and matrix vector multiplication to the Montium within a few weeks. This includes becoming familiar with the architecture, the development tools as well as the algorithm.

The Montium consumes 0.5mW/Mhz in 130nm technology [4]. Executing our implementation of the DWT on the Annabelle ASIC would be able to process 100 mega samples or pixels a second at 50 MHz while consuming 25mW

Comparing this result with other currently available implementations is troublesome for a couple of reasons:

- First of all, most accelerators or compression IPs are complete architecture designs. Dedicated architectures are most often more efficient than more general purpose DSP architectures like the Montium. Most of the research is done in the field of dedicated architectures and not in mapping these algorithms onto existing parallel architectures.
- Secondly, there is no pre-existing defined benchmark. This is probably because most standards do dictate how the DWT should be calculated but not with what accuracy or data width. The direct result is that most implementations differ from each other in one way or another, which makes a fair comparison difficult.
- Finally, most standards are optimized towards reducing the number of ALU operations. With current developments in efficient parallel architectures the number of operations itself is not the most important anymore. It is, however, important that data dependencies are brought down to a minimum. This makes it possible to stream large amounts of data through these parallel architectures. For this reason, it might be possible that an algorithm maps well in terms of ALU operations but data dependencies make it almost impossible to fully utilize the parallelism.

Closest to our implementation is probably the work of Fry and Hauck [10]. They implemented a compression standard for hyperspectral image compression on an FPGA. One of the stages is a DWT which is similar to the one used in the CCSDS standard. The result they obtained for the DWT implemented on a FPGA (Virtex 2000E) was 4 pixels per clock cycle for a 1D-DWT transform. They process 4 rows at the same time and they also have a 16-bit fixed-point data representation.

Another design that comes close to ours is that of Li and Dou [9]. Their work mainly focuses on efficient split up of images and creating an efficient architecture. The actual calculation they do is the same as in our implementation, 2 pixels per clock cycle. The novelty of their work lies in the minimal amount of memory needed. With the NoC and multiple Montiums this could also be performed on our platform.

6. CONCLUSION AND FURTHER WORK

In this paper we have shown that dynamic reconfigurable hardware and in particular the Montium provides a platform which can execute the DWT with high throughput and low energy consumption. Since the Montium comes with a development environment that supports a C-like language, it is relatively easy to map and update the algorithm on the Montium.

A downside of the current implementation is the fact that large amounts of data have to move off chip. To limit this we hope to employ special memory tiles, which are currently under development. These tiles support the transpose operation that is needed to switch from row to column processing.

The CCSDS standard does not prescribe a number of bits that need to be supported. Therefore, our implementation is fully compliant. The standard does supply a set of test images which include some with a 16-bit range, which we can currently not handle. As the Montium is parameterisable, a 24 or even a 32 bit HDL description of the Montium can easily be generated.

In order to support the floating-point part of the standard, a floating-point core would need to be added. Such a modification of the Montium would require a considerable amount of work. Most current algorithms can be implemented in a fixed-point architecture.

A possible solution for floating-point precision is a technique also used in the PACT XPP architecture [11]. In this architecture a simplified IEEE 754 is implemented as block floating-point, in which a floating-point format is created by using two integer packets: a signed mantissa and an exponent, which is relatively simple to implement and causes a limited amount of additional hardware.

A final recommendation would be to emphasize that data dependencies should be kept to a minimum in new standards. This eases the task of mapping onto parallel hardware and optimize for throughput rather than for computing resources.

REFERENCES

[1] J. Portell, X. Luri, and E. Garca-Berro. High-performance payload data handling system for GAIA. *IEEE transactions on Aerospace and Electronic Systems*, 42:421–435, 2006.

[2] Next generation processor for on-board payload data processing application synthesis. Technical report, ESA ESTEC, 2007.

[3] G. J. M. Smit, A. B. J. Kokkeler, P. T. Wolkotte, P. K. F. Hölzenspies, M. D. van de Burgwal, and P. M. Heysters. The chameleon architecture for streaming DSP applications. *EURASIP Journal on Embedded Systems*, 2007:78082, 2007.

[4] G. J. M. Smit, A. B. J. Kokkeler, P. T. Wolkotte, and M. D. van de Burgwal. Multicore architectures and streaming applications. *Proceedings of System-level Interconnect Prediction SLIP' 08*, 2008.

[5] G. K. Rauwerda, P. M. Heysters, and G. J. M. Smit. Towards software defined radios using coarse-grained reconfigurable hardware. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(1):3–13, January 2008.

[6] CCSDS 122.0-B-1. Image data compression blue book. Technical report, CCSDS, 2006.

[7] Recore Systems. `www.recoresystems.com`.

[8] P. S. Addison. *The Illustrated Wavelet Transform Handbook*. Institute of Physics Publishing, 2002.

[9] B. Li and Y. Dou. Fidp: A novel architecture for lifting-based 2D DWT in JPEG2000. *LNCS Advances in Multimedia Modeling*, 4352:373–382, 2007.

[10] T. W. Fry and S. Hauck. Hyperspectral image compression on reconfigurable platforms. *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2002.

[11] M. A. Syed and E. Scheuler. Reconfigurable parallel computing architecture for on-board data processing. *Proceedings of Adaptive Hardware and Systems*, 2006.