Implementing Non Power-of-Two FFTs on Coarse-Grain Reconfigurable Architectures

Arnaud RIVATON, Jérôme QUÉVREMONT
Thales Communications
Colombes, France
{arnaud.rivaton, jerome.quevremont}@fr.thalesgroup.com

Qiwei ZHANG, Pascal WOLKOTTE, Gerard SMIT
University of Twente
Enschede, The Netherlands
{Q.Zhang, P.T.Wolkotte, G.J.M.Smit}@utwente.nl

***Abstract*—To improve power figures of a dual ARM9 RISC core architecture targeting low-power digital broadcasting applications, the addition of a coarse-grain architecture is considered. This paper introduces two of these structures: PACT's XPP technology and the Montium, developed by the University of Twente, and presents the implementation of a Fast Fourier Transform on 1920 complex samples on both of them. Results in terms of processing time, resource utilization and energy dissipation are described and compared to those we have obtained on the RISC core. Then, as a conclusion, the paper presents the next steps of the development and some development issues.**

I. Introduction

The DRM (Digital Radio Mondiale) standard [1], [2] proposes the digitization of radio broadcasting in frequency bands below 30 MHz. A System on Chip (SoC) called DiMITRI was designed to show the feasibility of a DRM reception solution and to obtain a first receiver prototype [3]. Analyses showed that most computation power is used in the Coded Orthogonal Frequency Division Multiplexing (COFDM) [4] demodulation to compute Fast Fourier Transforms (FFT) and inverse transforms (IFFT) on complex samples. These FFTs have to be computed on non power-of-two numbers of samples, which is very uncommon in the signal processing world. These algorithms already exist in software on a 32-bit ARM9 RISC core and our objective is to implement them on a more optimized structure which would reduce their power dissipation with limited impact on the silicon area.

More and more, digital systems demand the combination of high performance and low power dissipation to implement signal processing algorithms. The usual DSP and RISC implementations are very flexible at the expense of power dissipation (for a given algorithm). On the other hand, hard-wired structures like ASICs lack flexibility and evolution capabilities but display the best results in terms of performance and power dissipation. Reconfigurable structures claim to bridge the gap between programmable processors and hard-wired structures through an average balance between flexibility and efficiency (which we define as the computing performance divided by the power dissipation). In this paper, the implementation of an FFT-1920 on two coarse-grain reconfigurable architectures is presented, as well as the performances obtained in terms of processing time, silicon area and power consumption.

This work is part of the 4S project that has been supported by the Sixth European Framework Programme (IST 001908).

II. Coarse-Grain Reconfigurable Architectures

Reconfigurable architectures may be split into two families: fine-grain architectures which manipulate bits (FPGAs) and coarse-grain architectures based on function units such as multipliers or ALUs which handle words (multi-bit data). Coarse-grain architectures have been developed recently to overcome some of the limitations of fine-grain reconfigurable tiles such as power dissipation, routing complexity, configuration memory and configuration time. Two different coarse-grain structures have been studied and are presented below.

A. PACT XPP Technology

The eXtreme Processing Platform (XPP) is a run-time coarse-grain reconfigurable architecture based on a 2D array of computing elements, internal memories and a circuit switch-like communication network [5]. The XPP64-A1 chip can be used as a standalone processor, or as a coprocessor next to a microcontroller [6]. Its structure is presented in figure 1. The XPP64-A1 is built from an 8x8 array of 24-bit ALU-PAEs (Arithmetic and Logic Unit - Processing Array Elements) and two rows of 512 24-bit words RAM-PAEs on the sides. In each configuration, a PAE performs one dedicated operation. The array is coupled with a Configuration Manager responsible for the run-time management of configurations. ALUs do not have instruction sequencers and caches, since the operations to be performed are statically configured during the lifetime of a configuration.

The PAE objects are integrated within a network-on-chip, providing point-to-point connections with data handshaking. The dataflow structure implies that an operation is performed as soon as all necessary input values are available and the previous output has been consumed by the downstream operation. The XPP is supported by a dedicated development suite. The architecture is programmed using the low level Native Mapping Language (NML). For our trials, we have used a board including an XPP64-A1, a microcontroller, some external memories, etc.

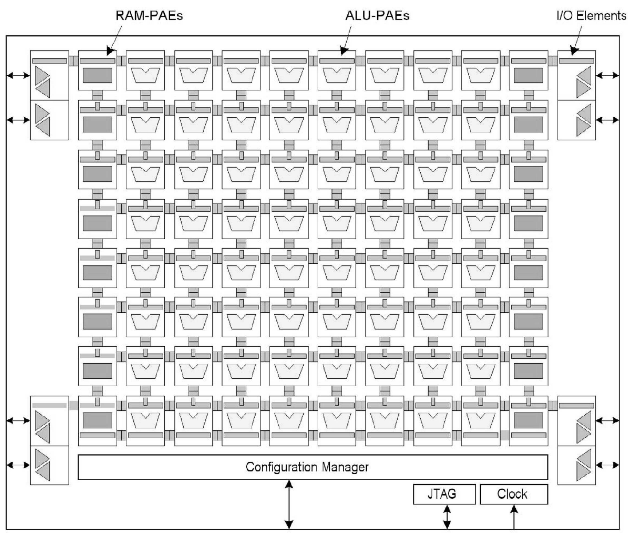

Figure 1. The XPP64-A1 structure made of ALU-PAEs and RAM-PAEs

B. Montium Reconfigurable Tile Processor

The Montium was developed by the University of Twente [7]. It consists of a Communication and Configuration Unit (CCU) and the reconfigurable Tile Processor (TP) which is shown in the upper part of figure 2. The TP bears a resemblance with a Very Long Instruction Words (VLIW) processor, but its control structure is quite different to minimize the energy consumption.

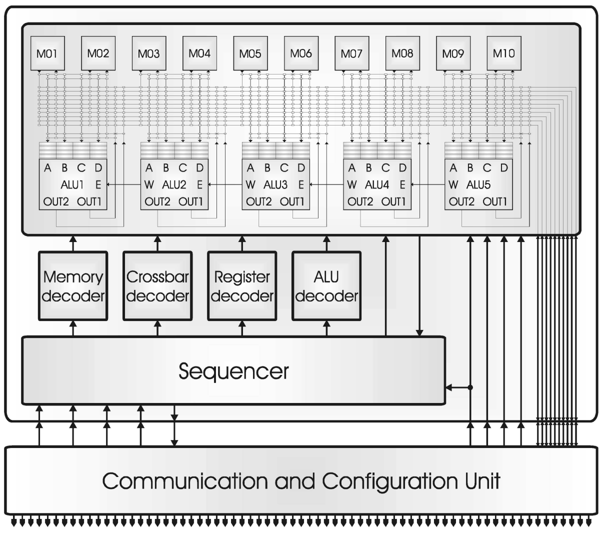

Figure 2. The Montium processing tile with the Montium TP and the CCU

The TP is the computing part that can be configured to implement a particular algorithm. It consists of five 16-bit ALUs which can exploit spatial concurrency and 10 local SRAMs containing 1024 16-bit words each. A reconfigurable Address Generation Unit accompanies each memory. A relatively simple sequencer controls the entire array, by selecting configuration instructions that are stored in the decoders. An ASIC synthesis of the Montium TP was performed in 0.13 μm technology, giving a maximal clock frequency of 40 to 150 MHz (depending on the algorithm).

III. The DRM Waveform

The DRM (Digital Radio Mondiale) standard [1],[2] has been adopted by the ETSI at a European level and by the IEC (International Electrotechnical Committee) at a worldwide level. DRM offers digital radio broadcasting in three frequency bands up to 30 MHz. It brings important improvements compared to existing analogue broadcasting in the above mentioned frequency bands: stereophonic sound, FM-like sound quality, data transmission, etc. A transmitter can cover a region, a country or even reach any point in the world.

Our focus is the receiver side, and more specifically the COFDM demodulation. The market demands will require low-power for battery-powered mobile receivers. The FFTs of the COFDM demodulation have proved to occupy most resources of the ARM9 processor on the DiMITRI chip, which translates into excessive power dissipation. Therefore, new structures are explored to decrease the power consumption. The first idea is naturally to implement a hard-wired module to compute the FFTs. Unfortunately, in our case there are 18 types of FFTs and IFFTs[1], which makes a hard-wired implementation impracticable from a design complexity and silicon area viewpoint. A more flexible structure, like a coarse-grain reconfigurable one, could offer a better balance between flexibility and efficiency for our application domain and would also allow to share the same silicon for all these types of FFTs and IFFTs.

IV. Implementation of an FFT-1920

The Discrete Fourier Transform (DFT) transforms a signal from the time domain to the frequency domain. It is defined by the following relation between N complex inputs $x(n)$ and N complex outputs $X(k)$ [8]:

$$X(k) = \sum_{n=0}^{N-1} x(n).W_N^{nk}, \quad k = 0, 1, ..., N-1 \qquad (1)$$

where $W_N^{nk} = e^{-j\frac{2\pi.nk}{N}}$ are primitive roots of the unit circle also called "twiddle factors". Directly evaluating this formula requires $O(N^2)$ operations.

The Fast Fourier Transform (FFT) is a set of algorithms that improve the efficiency of the DFT. As mentioned in the introduction, our objective is to implement an FFT on N=1920 16-bit complex samples on the XPP64-A1 and the Montium. This particular case of FFT is linked with the characteristics of some processing which are performed on DRM frames. This FFT was chosen because it is the biggest non power-of-two FFT used in our application. If we manage to implement it efficiently, we expect that we will also be able to implement all the other FFTs.

[1] based on a breakdown by five different prime factors instead of the usual breakdown by the 2 prime radix (for power-of-two FFTs).

Two different algorithms have been used to split up the FFT: the "divide and conquer approach" [9], and in particular radix-2 and radix-4 algorithms [10], and the Prime Factor Algorithm (PFA) [11]. The PFA turns the original transform into sets of small DFTs, the lengths of which have to be co-prime. It makes use of Good's mapping to convert the 1D $N=N_1 \cdot N_2$ DFT into a 2D DFT in a row-column fashion. In our case of N=1920, we have chosen N_1=128 and N_2=15 and split up the FFT-1920 into 15 FFT-128 followed by 128 FFT-15. As 128 is a power of two, the FFT-128 can be performed using radix-2 and/or radix-4 algorithms that require $O(N \log_2 N)$ operations.

A. *Implementation of the FFT-1920 on the XPP*

The real and the imaginary parts of the 1920 input values are separated and sent to the XPP array by stream transfers through two input ports. The intermediate values are stored in two external RAMs in normal order and are read in correct order using the addresses stored in pre-initialized FIFOs. Figure 3 outlines the implementation of the FFT-1920 using the PFA. The 128 FFT-15 are also computed using the PFA (five FFT-3 followed by three FFT-5) which implies the use of very efficient algorithms for computing both the FFT-3 and the FFT-5 [12]. The same flow as in figure 3 is used, but with an FFT-3 and an FFT-5. The implementation of the FFT-128 is decomposed into the computation of two FFT-64 (with a radix-4 algorithm) followed by 64 FFT-2.

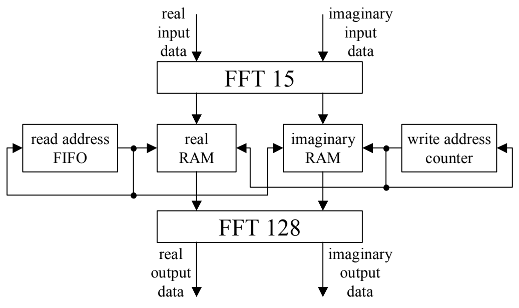

Figure 3. General view of the FFT-1920 using the PFA

First trials to implement the FFT on the XPP were done using the beta version of the C compiler provided by PACT. Unfortunately this C compiler was not efficient enough: the FFT-1920 occupied more than 100% of the tiles and would have had to be mapped in several consecutive configurations of the array. Therefore, the implementation went on in NML.

B. *Implementation of the FFT-1920 on the Montium*

Like for the XPP, the implementation of the FFT-1920 on the Montium uses the PFA.

The implementation of the FFT-15 differs from the implementation on the XPP. It could have been computed more efficiently by applying the PFA, but the data reorganization required by each FFT-15 would have consumed a large amount of resources of the Montium. Therefore, we have decided to implement it by optimizing the general DFT formula (1) rewritten with N = 15. The real and imaginary parts of the twiddle factors for N odd have the following properties:

$$\begin{aligned} \Re(W_{15}^{nk}) &= \Re(W_{15}^{(15-n)k}) = \Re(W_{15}^{n(15-k)}) ; \\ \Im(W_{15}^{nk}) &= -\Im(W_{15}^{(15-n)k}) = -\Im(W_{15}^{n(15-k)}) \end{aligned} \tag{2}$$

Then (1) can be written (setting N = 15):

$$\begin{aligned} X(0) &= \sum_{n=0}^{14} x(n) \\ X(k) &= x(0) + \sum_{n=1}^{7} (x(n) + x(15-n)) \cdot \Re(W_{15}^{nk}) \\ &\quad + \sum_{n=1}^{7} (x(n) - x(15-n)) \cdot \Im(W_{15}^{nk}), \quad k = 1, 2, ..., 7 \\ X(15-k) &= x(0) + \sum_{n=1}^{7} (x(n) + x(15-n)) \cdot \Re(W_{15}^{nk}) \\ &\quad - \sum_{n=1}^{7} (x(n) - x(15-n)) \cdot \Im(W_{15}^{nk}), \quad k = 1, 2, ..., 7 \end{aligned} \tag{3}$$

The butterfly structure of the Montium can be used to calculate $X(k)$ and $X(15–k)$ concurrently. They are computed in pairs, using four ALUs. One such pair requires 7 clock cycles and thus 49 clock cycles are needed for all seven pairs. $X(0)$ is calculated in parallel (on the fifth ALU) by adding all the inputs together. In this way, the total number of multiplications can be reduced by a factor 4 compared to the normal DFT-15. After the computations of the FFT-15, all the results are stored back in the memory in an order that facilitates the FFT-128 computations. A total of 7045 clock cycles is needed for all 128 FFT-15 calculations.

Afterwards, the 15 FFT-128 are executed on 15 blocks of data in the memory. Each FFT-128 is computed with a radix-2 algorithm, which also differs from the mixed-radix implementation used on the XPP. The details of the radix-2 FFT mapping to the Montium are shown in [7]. The results of the FFT-128 are stored back in the memory waiting to be read by the CCU. In contrast to the XPP implementation no external memory is needed. The size of the configuration file for the FFT-1920 is 2.6 kbytes. When the configuration is performed at 100 MHz, it can be loaded in 13 µs.

V. RESULTS

Table I shows the implementation results of an FFT-1920 on 16-bit complex data on the XPP. The algorithm is executed on the array in one single configuration. The implementation was verified by comparison with the results of FFT-1920 computations made in Scilab on identical input data. The output values are 24-bit values. The estimated power consumption for one 24-bit PAE is approx. 0.09 mW/MHz when it is heavily computing. Table I also gives the results we have obtained for an optimised[2] implementation of the same FFT on a 32-bit ARM9 RISC processor. Its power consumption, in 0.13 µm technology, is 0.25 mW/MHz [13].

[2] Critical parts have been coded in assembly language.

The use of the XPP architecture decreases the calculation time in cycles by a factor 36 and the energy consumed by 6. This architecture, originally built for intensive and regular operations (e.g. DCT, video processing) is flexible enough to compute a non-regular FFT such as the FFT-1920. The main drawback is a very large silicon area[3] which is not affordable for integration as an IP into the SoC we target[4].

TABLE I. COMPARISON OF RESOURCES AND PERFORMANCE FOR THE IMPLEMENTATION OF ONE FFT-1920

	Computing Structures		
	ARM9	***XPP64-A1***	***Montium***
CMOS Process (µm)	0.13	0.13	0.13
Architecture (# bits)	32	24	16
Clock frequency (MHz)	96	64	100
Processing time (# cycles)	476 000	13 248	14 033
Processing time (µs)	4 958	207	140
Resource utilization (mm²)	4.7	35.1	2.0
Power for one FFT[5] (µJ)	119.0	19.2	8.2

In [7], the power consumption of the Montium, in 0.13 µm technology, is estimated at 0.577 mW/MHz. The results of the FFT-1920 implementation on the Montium are also listed in Table I. These results show a saving of a factor 35 in terms of processing time, and 14 in terms of power consumption compared to the RISC implementation, and a smaller area. Although its datapath is only 16-bit wide, the Montium architecture seems to be the most promising to decrease the power dissipation and speed up the computations of the COFDM demodulation. These results may be explained by the fact that the micro-sequenced structure of the Montium is more suitable to algorithms that require lots of local sequencing (e.g. read and write address generators for accessing the RAMs).

The authors have found no documented ASIC implementation of non-power-of-two FFTs. [15] presents a high-speed FFT-1872 implemented on an FPGA[6].

VI. CONCLUSION

The evaluation of the coarse-grain reconfigurable architectures has taught us that, like for the programmable processors, the choice of a coarse-grain reconfigurable structure must be adapted to the targeted application to get the best performance at the lowest cost (in terms of power consumption and silicon utilization). In the case of the XPP processor, it is well adapted to intensive processing on large sets of data such as DCT computation, MPEG4 decompression but a micro-sequenced structure like the Montium looks more promising for processing that are somehow less intensive but more complex to control. This argument was confirmed by the porting of the FFT-1920, which can be considered as a complex computation but does not require the full computation power provided by the XPP architecture.

Within the 4S project [14], our next steps will be the integration of the DRM application on a platform which comprises Montium processors to handle COFDM processing, an ARM9 core and some hard-wired signal processing accelerators.

The development time has not been taken into account in our experiments. However, the effort to port algorithms on coarse-grain reconfigurable structures is considerable when using the ad-hoc low-level languages (NML, pseudo-assembler, etc.). The availability and the efficiency of compilers to quickly port algorithms described in C (or some other high level language) will be a key issue for the adoption of these structures in industry.

ACKNOWLEDGMENT

The authors are grateful to PACT for their support on the XPP board, the development tools and the results analysis.

REFERENCES

[1] "Digital Radio Mondiale (DRM); System Specification", ETSI, ES 201 980, V2.1.1, Nov. 2003.

[2] http://www.drm.org

[3] J. QUÉVREMONT, M. SARLOTTE, B. CANDAELE, "Development process of a DRM digital broadcast SoC receiver platform", Annales des Télécommunications, Sept-Oct 2004.

[4] J. H. STOTT, "Explaining some of the magic of COFDM", 20th International Television Symposium, Montreux (Switzerland), June 13-17, 1997.

[5] "The XPP white paper", Release 2.1, PACT XPP Technologies AG, March 2002.

[6] "XPP64-A1 Reconfigurable processor – Datasheet", Rev.1.1, PACT XPP Technologies AG.

[7] P. M. HEYSTERS, "Coarse-grained reconfigurable processors", CTIT Ph.D.-thesis series No. 04-66, 2004.

[8] P. DUHAMEL, M. VETTERLI, "Fast Fourier Transforms: a tutorial review and a state of the art", Signal Processing, Vol. 19, 1990, pp. 259-299.

[9] J. W. COOLEY, J. W. TUKEY, "An algorithm for the machine calculation of complex Fourier series", Math. Comput., Vol. 19, April 1965, pp. 297-301.

[10] D. F. ELLIOT, "Handbook of digital signal processing (Engineering Applications)", Academic Press Inc., London, England, 1987.

[11] I. J. GOOD, "The interaction algorithm and practical Fourier analysis", J. Royal Statist. Soc., ser. B, Vol. 20, 1958, pp. 361-372, with corrections in "Addendum", vol. 22, 1960, pp. 372-375.

[12] H. F. SILVERMAN, "An introduction to programming the Winograd Fourier transform algorithm", IEEE Trans. Acoust. Speech Signal Process., Vol. ASSP-25, No. 2, April 1977, pp 152-165, with corrections in Vol. ASSP-26, No. 3, June 1978, p. 268, and in Vol. ASSP-26, No. 5, Oct. 1978, p. 482.

[13] http://www.arm.com

[14] http://www.smart-chips.net

[15] "Mixed-Radix 'Dual Speed' FFT Product Specification", RF Engines Ltd, April 2004.

[3] Many ALU PAEs are actually used for local micro-sequencers required by the FFT-1920 algorithm.

[4] Future versions of the XPP structure are, however, planned to improve the power and silicon utilization figures.

[5] Power figures on ARM9 and XPP do not include the external RAMs.

[6] This implementation favors computation speed at the expense of silicon occupation. Further comparisons are difficult since we deliberately favored flexible solutions able to compute 18 types of FFT on a common hardware.