DESIGN OF APPLICATION SPECIFIC PROCESSORS FOR THE CACHED FFT ALGORITHM

O. Atak, A. Atalar, E. Arikan
Department of Electrical and Electronics Engineering
Bilkent University,
Ankara, Turkey

H.Ishebabi, D. Kammler, G. Ascheid, H. Meyr
Institute for Integrated Signal Processing Systems,
RWTH Aachen University,
Aachen, Germany

M. Nicola, G. Masera
VLSI Lab
Electronic Department
Politecnico di Torino,
Torino, ITALY

ABSTRACT

Orthogonal Frequency Division Multiplexing (OFDM) is a data transmission technique which is used in wired and wireless digital communication systems. In this technique, Fast Fourier Transformation (FFT) and inverse FFT (IFFT) are kernel processing blocks in an OFDM system, and are used for data (de)modulation. OFDM systems are increasingly required to be flexible to accommodate different standards and operation modes, in addition to being energy-efficient. A trade-off between these two conflicting requirements can be achieved by employing Application-Specific Instruction-Set Processors (ASIPs). In this paper, two ASIP design concepts for the Cached FFT algorithm (CFFT) are presented. A reduction in energy dissipation of up to 25% is achieved compared to an ASIP for the widely used Cooley-Tukey FFT algorithm, which was designed by using the same design methodology and technology. Further, a modified CFFT algorithm which enables a better cache utilization is presented. This modification reduces the energy dissipation by up to 10% compared to the original CFFT implementation.

I. INTRODUCTION

In Orthogonal Frequency Division Multiplexing (OFDM) Systems, data bits are sent by using multiple sub-carriers in order to obtain a good performance in highly dispersive channels, and a good spectral efficiency. Because of its robustness against frequency-selective fading and narrowband interference, OFDM is applied in several digital wireless communication systems such as Wireless Local Area Networks and Terrestrial Digital Video Broadcasting (DVB-T). In OFDM systems, Inverse Fast Fourier Transformation (IFFT) and Fast Fourier Transformation (FFT) are used for modulation and demodulation respectively. From a signal processing perspective these are two key blocks in an OFDM system. In particular, in hand-held devices an energy efficient implementation is inevitable. To cover different standards using OFDM (and, in a next step, other schemes like CDMA, TDMA), a programmable solutions has strong advantages. A general purpose processor, however, is far less energy efficient. A trade-off between flexibility, energy-efficiency and real-time requirements can be achieved by employing Application Specific Instruction Set Processors (ASIPs). It was shown in [1], [2] that ASIPs can be used to efficiently implement the Cooley-Tukey FFT (CT-FFT) algorithm with a high degree of flexibility. The energy-efficiency of FFT ASIPs can be significantly increased by taking advantage of newer FFT algorithms such as the Cached FFT algorithm (CFFT). This algorithm efficiently uses data locality to reduce the number of accesses to the main data memory [3]. We therefore present an ASIP design which utilize the advantage offered by the CFFT algorithm. Furthermore, we show that given a suitable processor architecture, the algorithm can be efficiently parallelized at the instruction level. In order to have a conclusive comparison between ASIPs which implement the CT-FFT and the CFFT algorithms, the same design flow and technology are used as in [1]. The design flow is based on the Architecture Description Language (ADL) LISA [4] and on a framework for automated ASIP implementation [5].

For a variable length implementation of the CFFT algorithm, the size of the cache is selected to fit the largest FFT size. This leads to an inefficient cache utilization for lower size FFT. This paper presents a modified CFFT algorithm which allows a better cache utilization for a lower size FFT. This increases the energy-efficiency of the implementation.

The rest of this paper is organized as follows: the modified CFFT algorithm is presented in section II. Two architectures for the Cached FFT algorithm are then presented in section III. A comparison of results is given in section IV. Finally, concluding remarks are provided in section V.

II. THE MODIFIED CFFT ALGORITHM

The notation and the terminology in this section are taken from the description of the CFFT algorithm in [3], which will not be reproduced here for space reasons. The size of the cache of an implementation of the CFFT algorithm is determined by the number of epochs of the implementation, and by the size of the FFT [3]. For a variable length implementation, the size of the cache is determined by the largest possible number of points [6]. Smaller size FFTs use a part of the cache only, so that the latter is only partially utilized. However, it is possible to change the structure of

the algorithm to fully exploit the cache. This is achieved by computing more stages in epoch 0, rather than evenly distributing the stage computation in the two epochs. The parameters of the CFFT algorithm for the 2 epochs are then determined as follows:

1) The number of butterflies in a pass is given by $C/2$, where C is cache size.
2) The number of passes for the two epochs is given by $log_2 C$ and $N - log_2 C$ for epoch 0 and 1 respectively, where N is the number of FFT points.
3) The number of groups is given by N/C.

In this case, C is given by $C_{modified} = \sqrt{N_{max}}$, rather than $C_{original} = \sqrt{N}$. Since $C_{modified} > C_{original}$ for lower size FFT, less number of groups are required because of the larger number of butterflies in a pass. Consequently, the number of cache loading and dumping is reduced.

For example, for a 64 point FFT, there are 6 passes (stages), and $C_{original} = 8$. If $N_{max} = 256$, $C_{modified} = 16$, the number of groups is 4, and the number of butterflies in a pass is 8. The resulting flow graph is shown in figure 1. A comparison of the number of passes per epoch, groups per pass and butterflies per group is given in table I.

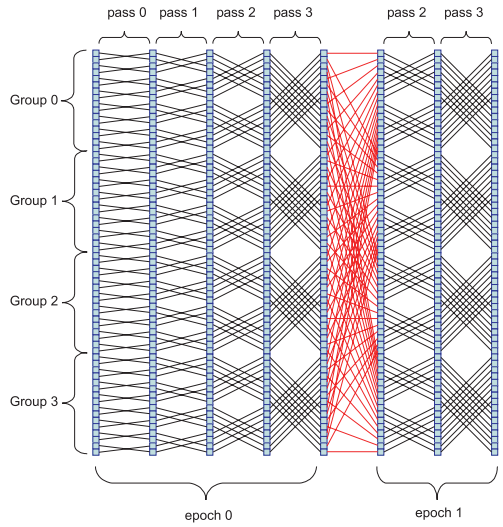

Fig. 1. Modified Cached FFT Algorithm

Table I. Comparison for $N = 64$ and $N_{max} = 256$

	original	modified
Passes (epoch 0/ epoch 1)	3/3	4/2
Groups per pass	8	4
Butterflies per group	4	8

III. ARCHITECTURES FOR THE CFFT ALGORITHM

In this section, a Single Instruction Single Data (SISD) and a Very Large Instruction Word (VLIW) processor architecture for the CFFT algorithm are presented. The basic instruction-set for the two architectures is the same. In both processors, registers are used as caches. Since the size of FFT that can be implemented depends on the size of the cache and the number of epochs, a cache size of 32 and a number of 2 for epochs is selected. Then, the processors can compute up to 1024 points according to the equation $C = N^{\frac{1}{E}}$ where E is the number of epochs [3].

III-A. Instruction-Set Design

Both processors have a special BFLY instruction which calculates the register indexes and the twiddle coefficient address. The address calculations requires: 1) The total number of passes, groups and butterflies. These 3 parameters together with the number of bit reversing are specified in a 16-bit control register (CTR). 2) The group, pass, butterfly and the epoch numbers for a given iteration. The first three parameters are passed with general purpose registers whereas the epoch number is passed as a 2 bit immediate value. Figure 2 shows the structure of the instruction.

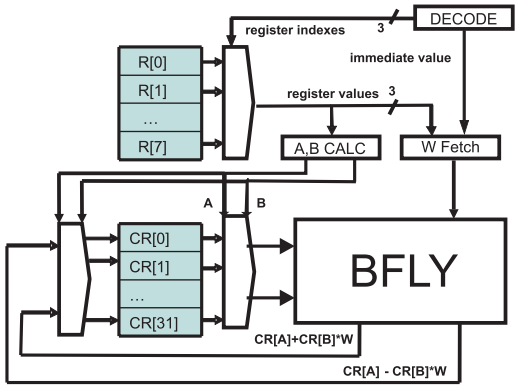

Fig. 2. Structure of the BFLY instruction

For speeding-up the execution of loops, a simple RPT instruction was added, which causes the architecture to repeat the following instruction for a specified number of times.

For loading data from the memory into the cache registers, a READ instruction was added. The instruction uses 2 pointers: Read Pointer (RP) for indexing the memory and Cache Pointer (CP) for indexing the cache registers. The READ instruction loads data from memory addressed by RP into the cache register indexed by CP. It also automatically increments the CP, and takes the number of increment for RP as an immediate operand. The code below shows how 32 data words are loaded into the cache registers and the butterflies computed.

```
RPT #32
READ #1
RPT #16
BFLY R[1],R[2],R[3]++,#0
```

Both RP and CP are dedicated registers, and can be initialized by special instructions.

III-B. The SISD Architecture

A load-store architecture with 6 pipeline stages was implemented. The architecture has 8 general purpose registers in addition to the 32 cache registers and 12 special purpose registers. The latter ones are used for addressing and for flow control. In total, there are 3 memories for program, data and coefficients with the configurations 24x256, 32x1024 and 32x512 respectively. The 16-bit real and imaginary parts of data or coefficients are concatenated to form 32-bit words.

The stages EX2 to EX4 are used by the BFLY instruction only. The execution of this instruction proceeds as follows: In EX1 stage, the 4 operands of the BFLY instruction are used to calculate cache register indexes and the twiddle address. The twiddle is then fetched from the dedicated twiddle memory. In the following two stages, a complex multiplication between the second input sample and the twiddle factor is performed. Four parallel multiplications are computed in one stage, followed by two parallel additions in the other. In the final pipeline stage, the results calculated in the previous stage are added to and subtracted from the first input sample to calculate the 2 outputs of the butterfly, and the results are then saved to the respective cache registers.

III-C. The VLIW Architecture

The second implemented architecture is a load-store VLIW ASIP with 4 slots, each of which can execute a BFLY instruction. The BFLY instruction is similar to the one in the SISD architecture with the difference that the execution occurs in 3 rather than 4 stages. In this case, the last 2 operations in the BFLY are done in one stage. The reason is that, in the former case, the BFLY instruction takes its inputs in the 3^{rd} pipeline stage and outputs the result in the 6^{th}. For the SISD architecture, the BFLY instructions are executed sequentially, and there is no data hazard between the passes. However, for the VLIW case, 4 BFLY instructions are executed in parallel and there are data hazards between the passes due to pipelining. In order to reduce this problem, the last two pipeline stages are combined so that the results of the BFLY instructions are available earlier. As it will be shown in section IV, the resulting increase in the critical path is compensated by the higher degree of parallelism. An alternative solution of using a forwarding mechanism is not considered for this architecture. Such a mechanism would result in a significant area increase, since each of the 8 outputs could potentially go into any of the 8 inputs.

Combining the last two stages does not completely resolve data dependencies. Because of significant area increase, interlocking was not implemented. Instead, since the code size is rather small, the problem is resolved in the assembly source by inserting NOP instructions between the passes. The trade-off is acceptable in this case because there is a maximum of 7% increase in total number of executed instructions for $N = 256$.

Since 4 butterfly instructions execute in parallel, and since they need 4 different twiddle factors for some passes, the twiddle coefficient memory is divided into 4 physically separate memories, each with the configuration 32×128. Each slot of the VLIW architecture has access to every of the 4 twiddle memories. Expensive interleaving is not necessary because access conflicts can be completely avoided in software. Table II shows the twiddle addressing scheme for the VLIW architecture. The most significant 2 bits are used to select the twiddle memory. The 2 bits are in turn determined by the group, pass, epoch and the butterfly number. In any iteration, the former 3 numbers are the same. Therefore, selecting butterfly numbers having a difference of 4 guarantees that there is no conflict for all passes. The code below shows how this is achieved for N=64 for epoch 0 (for modified CFFT).

```
BFLY R[0],R[1],R[2]++,#0 || \
BFLY R[0],R[1],R[3]++,#0 || \
BFLY R[0],R[1],R[4]++,#0 || \
BFLY R[0],R[1],R[5]++,#0
```

The registers R[2],R[3],R[4] and R[5] which contain the butterfly numbers are initialized to 0, 4, 8 and 12 respectively prior to each group iteration.

Table II. Twiddle Addressing Scheme for the VLIW ASIP

E	P	Twiddle Coefficient Address								
0	0	0	0	0	0	0	0	0	0	0
	1	B_0	0	0	0	0	0	0	0	0
	2	B_1	B_0	0	0	0	0	0	0	0
	3	B_2	B_1	B_0	0	0	0	0	0	0
	4	B_3	B_2	B_1	B_0	0	0	0	0	0
1	0	G_4	G_3	G_2	G_1	G_0	0	0	0	0
	1	B_0	G_4	G_3	G_2	G_1	G_0	0	0	0
	2	B_1	B_0	G_4	G_3	G_2	G_1	G_0	0	0
	3	B_2	B_1	B_0	G_4	G_3	G_2	G_1	G_0	0
	4	B_3	B_2	B_1	B_0	G_4	G_3	G_2	G_1	G_0
		Sel. the memory		Sel. the cell within the twiddle memory						

IV. RESULTS AND DISCUSSION

The ASIPs were targeted for a 130nm technology library. Table III shows a comparison of 3 ASIPs. CFFT-S is the SISD ASIP that was described in Section III-B. For a comparison with the Cooley-Tukey algorithm, the ASIP which is described in [1] is considered. In the publication, two ASIPs with an optimized data-path and with an optimized control-path respectively are described. The former is selected for comparison because this ASIP is comparable to CFFT-S: both contain a butterfly instruction which fetch the operands and update the addresses with a minimum in overhead. However, the data path optimized ASIP, which is here called CT-D, does not have instructions for ZOL as is the case with CFFT-S. Since this comparison aims at determining the reduction in energy dissipation as a consequence of less number of accesses to the main memory, a direct comparison

with CFFT-S would be inconclusive. Therefore, an ASIP CT-Z which is similar to CT-D, but which supports nested ZOLs was implemented. The selected technique for nested ZOLs is the same as in [1]. The bypass mechanism of CT-D was not re-implemented in CT-Z. This is because the CT-FFT algorithm can be similarly implemented on CT-Z without any need for bypassing.

The effectiveness of computing the butterflies from the cache registers can be observed from table III, where energy dissipation is reduced by 25% and 22% for 256 and 1024 points respectively. However, this implementation of the CFFT algorithm is slower by 21% (run-time) for 256 points. This is attributed to low cache utilization for lower points FFT. Table IV shows the results for the modified CFFT algorithm. The advantage of our modifications is that the number of groups is decreased, so that the number of cache dumps and loads is also decreased. This reduces the increase in run-time to 8% for 256 points, with a corresponding further reduction in energy dissipation of 10%. Significantly better results can be obtained by unrolling the groups and pass loops for the modified algorithm as shown in the table.

Table V shows the results for the VLIW implementation of the modified algorithm. A speed-up of 186% and 39% for 256 and 1024 points are achieved. But, this comes at a cost of more than double the gate count. The area increase is primarily caused by the duplication of the data path. No area overhead is incurred for resolving data dependencies. These are completely resolved by the techniques which are described in the previous section. Even though the CFFT algorithm could be efficiently parallelized at the instruction level with respect to execution time, this approach increases the energy consumption considerably by 17% and 48%.

Table III. Comparison of CFFT and CT-FFT ASIPs

		CFFT-S	CT-Z	CT-D
Cycles	for N= 256	2410	1729	1997
	for N=1024	8906	7757	9351
Energy cons. (μJ)	for N= 256	0.268	0.359	0.384
	for N=1024	1.270	1.635	1.872
Area (KGates)		66.6	76.7	106.0
Clock period (ns)		4.32	4.94	5.00

Table IV. Different CFFT Implementations

		Original	Mod.	Unrolled
Cycles	for N= 256	2410	1811	1322
	for N=1024	8906	8555	6345
Energy cons. (μJ)	for N= 256	0.268	0.241	0.214
	for N=1024	1.270	1.255	1.122

Table V. Comparison of SISD and VLIW Implementations

		SISD	VLIW
Cycles	for N= 256	1811	819
	for N=1024	8555	5636
Energy consumption (μJ)	for N= 256	0.241	0.292
	for N=1024	1.255	2.409
Area (KGates)		66.6	156.7
Clock period (ns)		4.32	4.88

V. CONCLUSION

In this work, two ASIP design concepts for the Cached FFT algorithm have been presented. The design exploration was conducted with the ADL LISA. It has been shown that the Cached FFT algorithm can lead to a significant lower energy dissipation over the Cooley-Tukey algorithm. Moreover, a modified CFFT algorithm with a better cache utilization has been presented. This increases the efficiency of the CFFT algorithm, both with respect to execution time and energy dissipation. Further, it has been shown that the CFFT algorithm can be efficiently parallelized at the instruction level for higher performance without an overhead in area. In our future work, further FFT algorithms are going to be explored.

VI. ACKNOWLEDGMENT

This work was supported by the European Union FP6 NoE NEWCOM under contract FP6-507325

VII. REFERENCES

[1] M. Nicola, G. Masera, M. Zamboni, H. Ishebabi, D. Kammler, G. Ascheid, and H. Meyr, "FFT processor: a case study in ASIP development," in *IST Mobile Summit*, Dresden, Germany, June 2005.

[2] K. Heo, S. Cho, J. Lee, and M. Sunwoo, "Application-specific DSP architecture for fast fourier transform," in *IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, June 2003, pp. 369–377.

[3] B. M. Baas, "A low-power, high-performance, 1024-point FFT processor," in *IEEE Journal of Solid-State Circuits*, vol. 34, no. 3, 1999, pp. 380–387.

[4] Hoffmann, A. and Schliebusch, O. and Nohl, A. and Braun, G. and Meyr, H., "A Methodology for the Design of Application Specific Instruction Set Processors (ASIP) Using the Machine Description Language LISA," in *Proceedings of the International Conference on Computer Aided Design (ICCAD)*. San Jose, USA: IEEE/ACM, Nov. 2001.

[5] Schliebusch, O. and Chattopadhyay, A. and Kammler, D. and Ascheid, D. and Leupers, R. and Meyr, H. and Kogel, T., "A Framework for Automated and Optimized ASIP Implementation Supporting Multiple Hardware Description Languages," in *ASP-DAC*, Shanghai, China, Jan 2005.

[6] J. Kuo, C. Wen, C. Lin, and A. Wu, "VLSI design of a variable length FFT/IFFT processor for OFDM-based communication systems," in *EORASIP Journal on Applied Signal Processing*, vol. 13. Hindawi Publishing Corporation, 2003, pp. 1306–1316.