Low-Power, High-Performance TTA Processor for 1024-Point Fast Fourier Transform

Teemu Pitkänen, Risto Mäkinen, Jari Heikkinen, Tero Partanen, and Jarmo Takala

Tampere University of Technology, P.O. Box 553, FIN-33101 Tampere, Finland
[teemu.pitkanen, jari.heikkinen, risto.makinen, jarmo.takala]@tut.fi

Abstract. Transport Triggered Architecture (TTA) offers a cost-effective trade-off between the size and performance of ASICs and the programmability of general-purpose processors. This paper presents a study where a high performance, low power TTA processor was customized for a 1024-point complex-valued fast Fourier transform (FFT). The proposed processor consumes only 1.55 μJ of energy for a 1024-point FFT. Compared to other reported FFT implementations with reasonable performance, the proposed design shows a significant improvement in energy-efficiency.

1 Introduction

Fast Fourier transform (FFT) has an important role in many digital signal processing (DSP) systems. E.g., in orthogonal frequency division multiplexing (OFMD) communication systems, FFT and inverse FFT are needed. The OFMD technique has become a widely adopted in several wireless communication standards. When operating in wireless environment the devices are usually battery powered and, therefore, an energy-efficient FFT implementation is needed. In CMOS circuits, power dissipation is proportional to the square of the supply voltage [1]. Therefore, a good energy-efficiency can be achieved by aggressively reducing the supply voltage [2] but unfortunately this results in lower circuit performance. In this paper, a high performance, low power processor is customized for a 1024-point FFT application. Several optimization steps, such as special function units, code compression, manual code generation, are utilized to obtain the high performance with low power dissipation. The performance and power dissipation are compared against commercial and academic processors and ASIC implementations of the 1024-point FFT.

2 Related Work

Digital signal processors offer flexibility and, therefore, low development costs but at the expense of limited performance and typically high power dissipation. Field programmable gate arrays (FPGA) combine the flexibility and the speed of application-specific integrated circuit (ASIC) [3]. However, FPGAs cannot compete with the energy-efficiency of ASIC implementations. For a specific application, the energy-efficiency between these alternatives can differ by multiple orders of magnitude [4]. In general, FFT processor architectures can be divided into five categories: processors are based

on single-port memory, dual-port memory, cached memory, pipeline, or array architecture [5]. In [6], a reconfigurable FFT-processor with single memory based scalable IP core is presented, with radix-2 algorithm. In [7], variable-length FFT processor is designed using pipeline based architecture. It employs radix-2/4/8 single path delay feedback architecture. The proposed processor supports three different transform lengths by bypassing the input to the correct pipeline stage. In [5], cached memory architecture is presented, which uses small cache memories between the processor and the main memory. It offers good energy-efficiency in low voltage mode but with rather low performance. In [8], an energy-efficient architecture is presented, which exploits subtreshold circuits techniques. Again the drawback is the poor performance.

The proposed FFT implementation uses a dual-port memory and the instruction schedule is constructed such that during the execution two memory accesses are performed at each instruction cycle, i.e., the memory bandwidth is fully exploited. The energy-efficiency of the processor matches fixed-function ASICs although the proposed processor is programmable.

3 Radix-4 FFT Algorithm

There are several FFT algorithms and, in this work, a radix-4 approach has been used since it offers lower arithmetic complexity than radix-2 algorithms. The specific algorithm used here is a variation of the in-place radix-4 decimation-in-time (DIT) algorithm and the 4^n-point FFT in matrix form is defined as

$$\begin{aligned} F_{4^n} &= \left[\prod_{s=n-1}^{0} [P_{4^n}^s]^T (I_{4^{n-1}} \otimes F_4) D_{4^n}^s P_{4^n}^s \right] P_{4^n}^{in} \; ; \\ P_{4^n}^s &= I_{4^{(n-s-1)}} \otimes P_{4^{(s+1)},4^s} \; ; \;\; P_{4^n}^{in} = \prod_{k=1}^{n} I_{4^{(n-k)}} \otimes P_{4^k,4} \; ; \\ P_{K,R}(m,n) &= \begin{cases} 1, \text{iff } n = (mR \bmod K) + \lfloor mR/K \rfloor \\ 0, \text{otherwise} \end{cases} \end{aligned} \tag{1}$$

where $\otimes$ denotes tensor product, P_N^{in} is an input permutation matrix of order N, F_4 is the 4-point discrete Fourier transform matrix, D_N^s is a diagonal coefficient matrix of order N, P_N^s is a permutation matrix of order N, and I_N is the identity matrix of order N. Matrix $P_{K,R}$ is a stride-by-R permutation marix [9] of order K such that the elements of the matrix. In addition, mod denotes the modulus operation and $\lfloor \cdot \rfloor$ is the floor function. The matrix D_N^s contains N complex-valued twiddle factors, W_N^k, as follows

$$D_N^s = \bigoplus_{k=0}^{N/4-1} \operatorname{diag}\left\{ W_{4^{s+1}}^{i(k \bmod 4^s)} \right\} \; , \; i = 0,1,\ldots,3 \; ; \; W_N^k = e^{-j2\pi k/N} \tag{2}$$

where j denotes the imaginary unit and $\oplus$ denotes matrix direct sum. Finally, the matrix F_4 is given as

$$F_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix}. \tag{3}$$

4 Transport Triggered Architecture

Transport triggered architecture (TTA) is a class of statically programmed instruction-level parallelism (ILP) architectures that reminds very long instruction word (VLIW) architecture. In the TTA programming model, the program specifies only the data transports (moves) to be performed by the interconnection network [10] and operations occur as "side-effect" of data transports. Operands to a function unit are input through ports and one of the ports is dedicated as a trigger. Whenever data is moved to the trigger port, the operation execution is initiated.

When the input ports are registered, the operands for the operation can be stored into the registers in earlier instruction cycles and a transport to the trigger port starts the operation with the operands stored into the registers. Thus the operands can be shared between different operations of a function unit, which reduces the data traffic in the interconnection and the need for temporary storage in register file or data memory.

A TTA processor consists of a set of function units and register files containing general-purpose registers. These structures are connected to an interconnection network, which connects the input and output ports of the resources. The architecture can be tailored by adding or removing resources. Moreover, special function units with user-defined functionality can be easily included.

5 TTA Processor for Radix-4 FFT

An effective means to reduce power consumption without reducing the performance is to exploit special function units for the operations of the algorithm. These units reduce the instruction overhead, thus they reduce the power consumption due to instruction fetch. Here four custom-designed units tailored for FFT application were used.

The interconnection network consumes a considerable amount of power and, therefore, all the connections from ports of function units and register files to the buses, which are not really needed, should be removed. By removing a connection, the capacitive load is reduced, which reduces also the power consumption. Clock gating technique can be used to reduce the power consumption of non active function units. Significant savings can be expected on units with low utilization.

TTA processors remind VLIW architectures in a sense that they use long instruction words, which implies high power consumption on instruction fetch. This overhead can be significantly reduced by exploiting program code compression.

5.1 Arithmetic Units

Since the FFT is inherently an complex-valued algorithm, the architecture should have means to represent complex data. The developed processor uses 32-bit words and the complex data type is represented such that the 16 most significant bits are reserved for the real part and the 16 least significant bits for the imaginary part. Real and imaginary parts use fractional representation, i.e., one bit for sign and 15 bits for fraction. The arithmetic operations in the algorithm in (1) can be isolated into 4-input, 4-output blocks described as radix-4 DIT butterfly operation defined by the following:

$$(y_0, y_1, y_2, y_3)^T = F_4\,(1, W_1, W_2, W_3)^T\,(x_0, x_1, x_2, x_3)^T \tag{4}$$

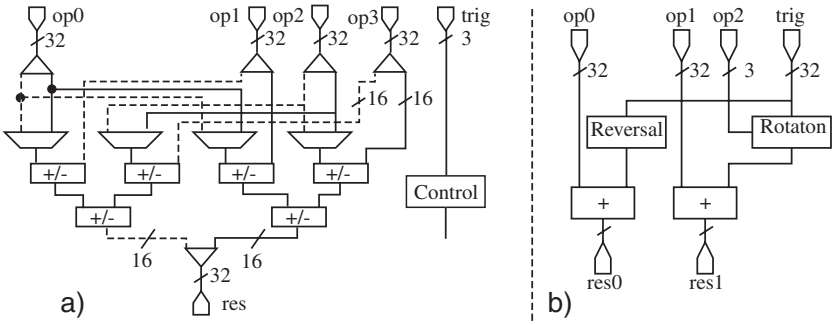

Fig. 1. a) Block diagram of complex adder. Solid lines represent real parts and dotted lines imaginary parts. ports op1-3 are operand ports and trigger port defines the operation. b) Block diagram of data address generator. op0: Base address of input buffer. op1: Base address of output buffer. op2: Butterfly column index. trig: Element index, trigger port. res0: Resulting address after field reversal. res1: Resulting address after index rotation.

where x_i denotes an input operand, W_i is a twiddle factor, and y_i is an output operand. One of the special function units in our design is complex multiplier, CMUL, which is a standard unit containing four 16-bit real multipliers and two 16-bit real adders. When the operand to the CMUL unit is a real one, i.e., multiplication by one, the other operand is directly bypassed to the result register. The CMUL unit is pipelined and the latency is three cycles. The butterfly operation contains complex additions defined by (3). In this work, we have defined a four-input, one-output special function unit, CADD, which supports four different summations according to each row in F_4. The motivation is that, in a TTA, the instruction defines data transports, thus by minimizing the transports, the number of instructions can be minimized. Each of the four results defined by F_4 are dependent on the same four operands, thus once the four operands have been moved into the input registers of the function unit, four results can be computed simply by performing a transport to trigger register, which defines the actual function out of the four possible complex summations. The block diagram of the CADD unit is illustrated in Fig. 1 a).

5.2 Address Generation

The N-point FFT algorithm in (1) contains two type of data permutations: input permutation of length N and variable length permutations between the butterfly columns. In-place computations require manipulation of indices into data buffer. Such manipulations are low-power if performed in bit-level. If the 4^n input operands are stored into a buffer in-order, the read index to the buffer, i.e., operand for the butterfly operation, can be obtained by bit field reversal. This reminds the bit reversal addressing in radix-2 algorithms but, instead of reversing single bits, here 2-bit fields are reversed [11], i.e., a $2n$-bit read index $r = (r_{2n-1} r_{(2n-2} \dots r_0)$ is formed from an element index (a linear counter) $a = (a_{2n-1} a_{2n-2} \dots a_0)$ as

$$r_{2k} = a_{2n-2k-2} \; ; \; r_{2k+1} = a_{2n-2k-1} \; , \; 0 \leq k < n \tag{5}$$

This operation is implemented simply with wiring. In a similar fashion, the permutations between the butterfly columns can be realized in bit-level simply by rotation of two bits to the right. However, the length of the bit field to be rotated is dependent on the butterfly column index, s, in (1). The $2n$-bit read index $p = (p_{2n-1}p_{(2n-2}\dots p_0)$ is formed from the element index a as [11]

$$\begin{cases} p_{2k} = a_{(2k+2s) \bmod 2(s+1)}, 0 \le k \le s \\ p_{2k+1} = a_{(2k+1+2s) \bmod 2(s+1)}, 0 \le k \le s \\ p_{2k} = a_{2k}, s < k < n \\ p_{2k+1} = a_{2k+1}, s < k < n \end{cases}. \quad (6)$$

Such an operation can be easily implemented with the aid of multiplexers. When the generated index is added to the base address of the memory buffer, the final address to the memory is obtained. The block diagram of the developed AG unit is shown in Fig 1 b). The input ports of the AG units are registered, thus the base addresses of input and output buffers need to be store only once into operand ports op0 and op1, respectively. The butterfly column index is stored into operand port op2 and the address computation is initiated by moving an index to trigger port. Two results are produced: output port res0 contains the address according to input permutation and port res1 according to bit field rotation.

5.3 Coefficient Generation

A coefficient generator (COGEN) unit was developed for generating the twiddle factors, which reduces power consumption compared to the standard method of storing the coefficients as tables into data memory. In an radix-4 FFT, there are $Nlog_4(N)$ twiddle factors as defined by (2) but there is redundancy. It has been be shown that all the twiddle factors can be generated from $N/8+1$ coefficients [12] with the aid of simple manipulation of the real and the imaginary parts of the coefficients. The COGEN unit is based on a table where the $N/8+1$ are stored. This table is implemented as hard wired logic for reducing the power consumption. The unit contains an internal address generator, which creates the index to the coefficient table based on two input operands: butterfly column index ($s = 0,1,\dots,n-1$) and element index ($a = 0,1,\dots,4^n-1$). The obtained index is used to access the table and the real and imaginary parts of the fetched complex number are modified by six different combinations of exchange, add, or subtract operations depending on the state of input operands. The resulting complex number is placed in the output register as the final twiddle factor.

5.4 General Organization

The general organization of the proposed TTA processor tailored for FFT (FFTTA) processor is presented in Fig. 2. The processor is composed of eight separate function units and a total of 11 register files containing 23 general-purpose registers. The function units and register files are connected by an interconnection network (IC) consisting of 18 buses and 61 sockets. In addition, the FFTTA processor contains a control unit, instruction memory, and dual-ported data memory. The size of the data memory is 2048 words of 32 bits implying that 32-bit data buses are used. There is one 1-bit bus, which is used for transporting the Boolean values.

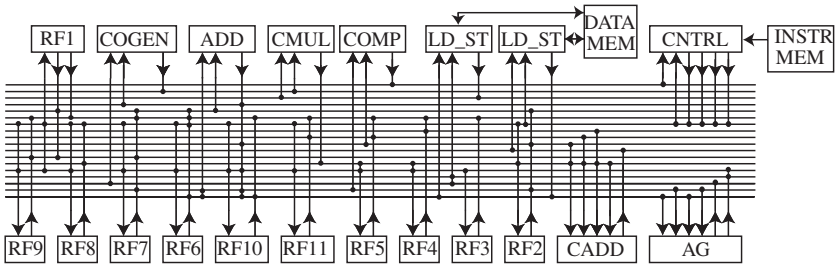

Fig. 2. Architecture of the proposed FFTTA processor. CADD: Complex adder. CMUL: Complex multiplier. AG: Data address generator. COGEN: Coefficient generator. ADD: Real adder. LD_ST: Load-store unit. COMP: Comparator unit. CNTRL: Control unit. RFx: Register files, containing total of 23 general purpose registers.

5.5 Instruction Schedule

In principle, an 4^n-point radix-4 FFT algorithm in (1) contains two nested loops: an inner loop where the butterfly operation is computed $4^{(n-1)}$ times and an outer loop where the inner loop is iterated n times. Each butterfly operation requires four operands and produces four results. Therefore, in a 1024-point FFT, a total of 10240 memory accesses are needed. If a single-port data memory is used, the lower bound for the number of instruction cycles for a 1024-FFT is 10240. If a dual-port memory is used, the lower bound is 5120 cycles.

In order to maximize the performance, the inner loop kernel needs to be carefully optimized. Since the butterfly operations are independent, software pipelining can be applied. In our implementation, the butterfly operations are implemented in a pipelined fashion and several butterflies at different phases of computation are performed in parallel. The developed 1024-point FFT code follows the principal code in Fig. 3.

In initialization, pointers and loop counters, i.e., butterfly and element indices, are set up. The input data is stored in order into data memory buffer. Another 1024-word buffer is reserved for intermediate storage and the final result. There is no separate code performing the input permutation but the address generation unit is used to access the input buffer in correct order with an address obtained from port res0 of AG in Fig.1b). The results of the first butterfly column are stored into the intermediate buffer with an address obtained from port res1 of AG. All the accesses to the intermediate buffer are done by using addresses from port res1 of AG.

In the prologue, the butterfly iterations are started one after each other and, in the actual inner loop kernel, four iterations of butterfly kernels are performed in parallel in pipelined fashion. The loop kernel evaluates also the loop counter. In the epilogue, the last butterfly iterations are completed and the loop counter of the outer loop is evaluated. The kernel contains the functionality of butterfly operations, which requires four triggers for memory reads and memory writes and corresponding address computations, four triggers for complex multiplier and four triggers for CADD unit. Since the branch latency is three cycles, the kernel can actually be implemented with four instructions. However, this approach results in a need for moving variables from an register

```
main() {
  initialization(); /* 9 instr. */
  for(stage=0; stage<5; stage++) {
    prologue(); /* 16 instr. */
    for(k=0; k<84; k++)
      kernel(); /* 12 instr. */
    epilogue; /* 21 instr. */
  }
}
```

Fig. 3. Pseudocode illustrating structure and control flow of program code.

to another. The reason is that parallel butterfly iterations need more than four intermediate results, which need to be stored into register files. Since there is no mechanism to dynamically index the register accesses, the only way is to use the register files as first-in-first-out buffers. Such register copies introduce additional power consumption, in particular, since the moves require additional buses and increase the register activity.

The final implementation of the kernel was 12 instructions and by that way, it was possible to keep the intermediate results in a dedicated register without need to copy the values. This resulted significant savings in power consumption at the expense of lengthening the program code by eight instructions. The parallel code for 1024-point FFT contains a total of 58 instructions and the instruction length was 162 bits. The program spends 96% of the execution time in the kernel. The execution of 1024-point FFT takes 5234 instruction cycles, thus the overhead to the theoretical lower bound with dual-port data memory (5120 cycles) is only 2% (114 cycles). This overhead is negligible compared to overheads seen in typical software implementations.

5.6 Code Compression

TTA suffers from poor code density, which is mostly due to minimal instruction encoding that is used to simplify decoding. Minimal instruction encoding leads to long instruction words. The long instruction word consists of dedicated fields, denoted as move slots. Each move slot specifies a data transport on a bus. Each move slot consists of three fields: guard, destination ID, and source ID. The guard provides means for conditional execution. The destination ID specifies the address of a socket that is reading data from a bus. The source ID specifies the address of a socket that is writing data on a bus. In addition to move slots, instruction words may contain dedicated long immediate fields to define large constant values, e.g., for jump addresses.

The poor code density can be improved by compression. Compression also results in reduced power consumption as fewer bits need to be fetched from the program memory. Dictionary-based compression is one of the simplest compression approaches to improve the code density [13]. Dictionary-based program compression stores all unique bit patterns into a dictionary and replaces them in the program code with code words to the dictionary. Given a program with N unique instructions, the length of the code word is $\lceil log_2|N| \rceil$ bits. During execution, the code word, fetched from the program memory is used to obtain the original instruction from the dictionary for decoding.

Table 1. Characteristics of 1024-point FFT on FFTTA processor on 130 nm ASIC technology with 1.5V supply voltage.

Clock Cycles	**5234**	Execution Time	**20.94 μs**	Power	**74 mW**
Clock Frequency	**250 MHz**	Area	**140 kgates**	Energy	**1.55 μJ**

In order to reduce the power consumption of the FFTTA processor and improve the code density, dictionary-based program compression was applied. All the unique instructions of the program code were stored into a dictionary and replaced with indices pointing to the dictionary. This resulted in decrease in the width of the program memory from 162 bits to 6 bits. The decompression, i.e., the dictionary access was supplemented to the control unit without additional pipeline stage. The actual dictionary(8586 bits) was implemented using standard cells.

6 Performance Analysis

In order to analyse the characteristics of the FFTTA processor, the structures of the previous special function units were described manually in VHDL. The structural description of the FFTTA core was obtained with the aid of the hardware subsystem of the MOVE Framework [14], which generated the VHDL description.

Then the FFTTA was synthesized to a 130nm CMOS standard cell ASIC technology with Synopsys Design Compiler. This was followed by a gate level simulation at 250 MHz. Synopsys Power Compiler was used for the power analysis. The obtained results are listed in Table 1. It should be noted that the instruction and data memories take 40% of the total power consumption of 74mW with 1.5V supply voltage. If the supply voltage is reduced to 1.1V, the total power consumption will drop down to about 40 mW. However, this will reduce the maximum clock frequency.

Table 2 presents how many 1024-point FFT transforms can be performed with energy of 1 mJ. The results are presented for ten different implementations of the 1024-point FFT. For some implementations there are different operating voltage or clock frequency points listed. Spiffee processor [5] employs a high performance architecture and low supply voltages and it's dedicated for the FFT. The StrongArm SA-1100 processor [15] employs custom circuits, clock gating, and reduced supply voltage. The Stratix [16] is an FPGA solution with dedicated embedded FFT logic usign Altera Megacore function. The TI C6416 [17] is a digital signal processor and the Imagine [18] is a media processor. They were both created using pseudo-custom data path tiling. In addition, the TI C6416 employs pass-gate multiplexer circuits. The 1024-point FFT with radix-4 algorithm can be computed in 6002 cycles in TI C6416 when using 32-bit complex words (16 bits for real and imaginary parts) [19]. However, in-place computations cannot be used and the processor has eight memory ports while the FFTTA uses only two. The Intel Pentium-4 [20] is a standard general-purpose microprocessor. Rest of the processors are dedicated for the FFT. The custom scalable IP core Zhao [6], employs single memory architecture with clock gating. The custom variable-length Lin [7]

Table 2. The number of 1024-point FFTs performed with a unit of energy.

Design	Tech. [nm]	Oper. voltage [V]	Clock freq. [MHz]	Exec. time [μs]	FFT/mJ	Design	Tech. [nm]	Oper. voltage [V]	Clock Freq. [MHz]	Exec. time [μs]	FFT/mJ
FFTTA	130	1.5	250	20.9	645		130	1.2	720	8.34	100
	600	1.1	16	330	319	TI C6416	130	1.2	600	10.0	167
Spiffee	600	2.5	128	41	67		130	1.2	300	21.7	250
	600	3.3	173	40	39	MIT FFT	180	0.35	0.01	250000	6452
SA-110	350	2	74	425.7	60		180	0.9	6	430.6	1428
	130	1.3	275	4.7	241	Lin	350	3.3	45.45	22.5	93
Stratix	130	1.3	133	9.7	173		350	2.3	17.86	57	133
	130	1.3	100	12.9	149	Zhao	180	-	20	281.6	43
Imagine	150	1.5	232	16.0	16	Intel P4	130	1.2	3000	23.9	0.8

FFT-processor employs radix-2/4/8 single-path delay algorithm. MIT FFT uses subtreshold circuit techniques [8].

Compared to other FFT designs the proposed FFTTA processor shows significant energy-efficiency. Only the MIT FFT outperforms the FFTTA. However, due to its long execution time, the MIT FFT is not usable in high performance designs. The performance of the FFTTA processor is still quite feasible although it does not provide the best performance. However, the performance can be scaled, i.e., the execution time can be halved by doubling the resources and memory ports. The memory size remains constant and it can be estimated that the energy-efficiency remains the same in terms of FFTs per energy unit.

7 Conclusions

In this paper, a low-power application-specific processor for FFT computation has been described. The resources of the processor have been tailored according to the needs of the application consisting of eight function units and 11 register files. Several methods for reducing the power consumption of the processor were utilized: clock gating, special function units, and code compression. The processor was synthesized on a 130 nm ASIC technology and power analysis showed that the proposed processor has both high energy-efficiency and high performance.

The described processor has limited programmability but the purpose of this experiment was to prove the feasibility and potential of the proposed approach. However, the programmability can be improved by introducing additional function units and loosening the code compression. In addition, different transform sizes can be supported by modifying the address generators and twiddle factor unit. This modifications are mainly addition of multiplexers, thuse significant increase in power consumption is not expected. In addition, the performance of the processor can be improved by adding computational resources implying need for higher data memory bandwidth

Acknowledgement

This work has been supported by the Academy of Finland under project 205743 and the National Technology Agency of Finland under research funding decision 40153/05.

References

1. Weste, N., Eshraghian, K.: Principles of CMOS VLSI Design: A Systems Perspective. Addison-Wesley, Reading, MA (1985)
2. Chandrakasan, A., Sheng, S., Brodersen, R.: Low-power CMOS digital design. IEEE Journal of Solid State Circuits **27**(4) (1992) 473–483
3. Reeves, K., Sienski, K., Field, C.: Reconfigurable hardware accelerator for embedded DSP. In Schewel, J., Athanas, P.M., Bove, V.M., Watson, J., eds.: Proc. SPIE High-Speed Comp. Dig. Sig. Proc. Filtering Using Reconf. Logic. Volume 2914., Boston, MA (1996) 332–340
4. Chang, A., Dally, W.: Explaining the gap between ASIC and custom power: A custom perspective. In: Proc. IEEE DAC, Anaheim, CA (2005) 281–284
5. Baas, B.M.: A low-power, high-performance, 1024-point FFT processor. IEEE Solid State Circuits **43**(3) (1999) 380–387
6. Zhao, Y., Erdogan, A., Arslan, T.: A low-power and domain-specific reconfigurable fft fabric for system-on-chip applications. In: Proc. 19th IEEE Parallel and Distrubuted Prosessing Symp. Reconf. Logic, Denver, CO (2005)
7. Lin, Y.T., Tsai, P.Y., Chiueh, T.D.: Low-power variable-length fast fourier transform processor. Proc. IEE Computers and Digital Techniques **152**(34) (2005) 499–506
8. Wang, A., Chandrakasan, A.: A 180-mV subthreshold FFT processor using a minimum energy design methodology. IEEE J. Solid State Circuits **40**(1) (2005) 310–319
9. Granata, J., Conner, M., Tolimieri, R.: Recursive fast algorithms and the role of the tensor product. IEEE Trans. Signal Processing **40**(12) (1992) 2921–2930
10. Corporaal, H.: Microprocessor Architectures: From VLIW to TTA. John Wiley & Sons, Chichester, UK (1997)
11. Mäkinen, R.: Fast Fourier transform on transport triggered architectures. Master's thesis, Tampere Univ. Tech., Tampere, Finland (2005)
12. Wanhammar, L.: DSP Integrated Circuits. Academic Press, San Diego, CA (1999)
13. Lefurgy, C., Mudge, T.: Code compression for DSP. Technical Report CSE-TR-380-98, EECS Department, University of Michigan (1998)
14. Corporaal, H., Arnold, M.: Using transport triggered architectures for embedded processor design. Integrated Computer-Aided Eng. **5**(1) (1998) 19–38
15. Intel: StrongARM SA-110 Microprocessor for Portable Applications Brief Datasheet. (1999)
16. Lim, S., Crosland, A.: Implementing FFT in an FPGA co-processor. In: The International Embedded Solutions Event (GPSx), Santa Clara, CA (2004) 230–233
17. Agarwala, S., Anderson, T., Hill, A., Ales, M., Damodaran, R., Wiley, P., Mullinnix, S., Leach, J., Lell, A., Gill, M., Rajagopal, A., Chachad, A., Agarwala, M., Apostol, J., Krishnan, M., Duc-Bui, Quang-An, Nagaraj, N., Wolf, T., Elappuparackal, T.: A 600 MHz VLIW DSP. IEEE J. Solid State Circuits **37**(11) (2002) 1532–1544
18. Rixner, S., Dally, W., Kapasi, U., Khailany, B., Lopez-Lagunas, A., Mattson, P., Owens, J.: A bandwidth-efficient architecture for media processing. In: Proc. Annual ACM/IEEE Int. Symp. Microarchitecture, Dallas, TX (1998) 3–13
19. Texas Instruments, Inc. Dallas, TX: TMS320C64x DSP Library Programmer's Reference. (2003)
20. Deleganes, M., Douglas, J., Kommandur, B., Patyra, M.: Designing a 3 GHz, 130 nm, Intel® Pentium ®4 processor. In: Digest of Technical Papers Symp. VLSI Circuits, Honolulu, HI (2002) 230–233