Transport Triggered Architecture processor for Mixed-Radix FFT

Teemu Pitkänen and RistoMäkinen and Jari Heikkinen and Tero Partanen and Jarmo Takala.
Tampere University of Technology, P.O. Box 553, FIN-33101 Tampere, Finland
[teemu.pitkanen, tero.partanen, jarmo.takala]@tut.fi

***Abstract*— Transport triggered architecture (TTA) offers a cost-effective trade-off between the energy-efficiency and performance of an ASIC implementation and the flexibility provided by a software implementation on a programmable processor. In this paper, we describe a programmable TTA processor, which is tailored for computing mixed-radix fast Fourier transform (FFT). Several approaches has been exploited to reduce the power consumption of the processor; e.g., special function units for complex-valued arithmetics and address computation, clock gating, instruction compression are utilized. The paper shows FFT implementation supporting power-of-two FFTs with the aid of mixed radix algorithm consisting of radix-4 and radix-2 computations. The developed processor is programmable but shows energy-efficiency comparable to fixed-function ASIC implementations.**

I. 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 from 32-point to 16384 point power of two FFT applications. 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.

II. 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.

III. Mixed Radix-4/2 FFT Algorithm

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

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

where $\otimes$ denotes tensor product, P^{in}_N is an input permutation matrix of order N, F_4 is the 4-point discrete Fourier transform matrix, D^s_N is a diagonal coefficient matrix of order N, P^s_N 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

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

where j denotes the imaginary unit and $\oplus$ denotes matrix direct sum.

When transform size is not equal to the 4^n-point, the last step is performed with radix-2 DIT algorithm, defined as

$$\begin{aligned} X(k) &= F_{1,2}(k) + W_n^k F_{2,2}(k); \\ X(k+N/2) &= F_{1,2}(k) - W_n^k F_{2,2}(k); \\ & \quad k = 0, 1, \ldots, N/2-1 \end{aligned} \tag{3}$$

Finally, the matrix F_4 and F_2 is given as

$$F_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix}. \quad F_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{4}$$

IV. 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 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.

V. TTA Processor for Mixed Radix 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 and the power consumption is reduced.

Due the reductictions in interconnection network, the programmability of processor is decreased. There can be placed sertain connections, which allows the processor to be programmable. This action increase the power dissipation and are thus not used in current proccessor architecture. 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.

A. 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 (1,3) can be isolated to two into 4-input, 4-output blocks described as radix-4 DIT butterfly operation and two radix-2 DIT butterflies as following:

$$(y_0, y_1, y_2, y_3)^T = F_4 (1, W_1, W_2, W_3)^T (x_0, x_1, x_2, x_3)^T \tag{5}$$

$$\begin{aligned} (y_0, y_1)^T &= F_2 (1, W_1)^T (x_0, x_1)^T \\ (y_2, y_3)^T &= F_2 (1, W_2)^T (x_2, x_3)^T \end{aligned} \tag{6}$$

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 (4). In this work, we have defined a four-input, one-output special function unit, CADD, which supports eigth different summations for Radix-4 and Radix-2 DIT butterflies according to each row in F_4 and two times F_2. 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 and F_2, 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 eigth possible complex summations. The CADD consists six 16-bit adders and the latency of the unit is two cycles.

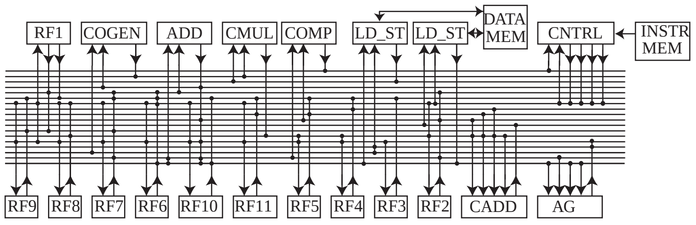

Fig. 1. Architecture of the proposed processor. CADD: Complex adder. CMUL: Complex multiplier. AG: Data address generator. COGEN: Coefficient generator. ADD: Real adder. LD_ST: Load-store unit. COMP: Comparator unit. CNTRL: Control unit. RFx: Register files, containing total of 25 general purpose registers. SH : shifter unit. LOGIC: Logic unit.

B. Address Generation

The N-point FFT algorithm in (1,3) contains 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 2^n input operands are stored into a buffer in-order, the read index to the buffer, i.e., operand for the butterfly operation, is obtained by rotation of two bits to the rigth. However, the length of the bit field to be rotated is dependent on the butterfly column index, s, in (1) and the size of transformation N.

The n-bit read index $p = (p_{max\_n-1} p_{max\_n-2} \dots p_0)$ is formed from the element index a as follows:

$$\begin{cases} p_i = \ a_i \ , \ i \ >= \ n-2s \\ p_i = \ a_j \ , \ i \ < \ n-2s \\ j = \ i+2 \qquad\qquad , \ i \ < \ n-2-2s \\ j = \ i-(n-2-2s) \ , \ 1 \ >= \ n-2-2s \end{cases} . \tag{7}$$

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 input ports of the address generation unit(AG) are registered, thus the base address and size of FFT needs to be stored only once into first and second operand port. The butterfly column index is stored into third operand port and the address computation is initiated by moving an index to trigger port. Resulting the bit field rotation. The latency of address generator is one.

C. 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 [11] with the aid of simple manipulation of the real and the imaginary parts of the coefficients. In similar fashion the reduction of twiddle factors can be refined to the mixed radix algorithm, by adding permutation network for butterfly element index. Same principle can be applied to the radix-2 FFT. Both radix-4 and radix-2 consist same set of twiddle factors, i.e. only $N/8+1$ coefficients are needed. 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 three input operands: butterfly column index ($s = 0, 1, \dots, ceil(log_4(N)) - 1$), element index ($a = 0, 1, \dots, N-1$)) and the size of transformation (N). 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.

D. General Organization

The general organization of the proposed TTA processor tailored for mixed radix FFT processor is presented in Fig. 1. The processor is composed of 11 separate function units and a total of 11 register files containing 25 general-purpose registers and 3 boolean registers. The function units and register files are connected by an interconnection network (IC) consisting of 17 buses and 65 sockets. In addition, the proposed processor contains a control unit, instruction memory, and dual-ported data memory. The size of the data memory is 16400 words of 32 bits implying that 32-bit data buses are used. The data memory is divided to seven blocks. There is one 1-bit bus, which is used for transporting the Boolean values.

E. Instruction Schedule

In principle, mixed radix-4/2 FFT algorithm in (**??**) 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 N-point FFT, a total of $(N * stages * 2)$ memory accesses are needed. If a single-port data memory is used, the lower bound for the number of instruction cycles for a, i.e., 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 mulitple-point FFT code follows the principal code in Fig. 2.

In initialization, pointers and loop counters, i.e., butterfly and element indices, are set up. The size of transformation is fetched from data memory buffer. The input data is stored in order into data memory buffer. The intermediate and final results will be stored is same buffer replacing previous data in buffer, i.e. for 16384 point transform 16385 buffer slot's, one slot for size of transformation, are used. The AG output is used to access the buffer in correct place, i.e. no extra code is needed for input or intermediate permutation.

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

```
main() {
   initialization(); /* 8 to 42 instructions */
   for(stage=0; stage<[log4 N]; stage++) {
      prologue(); /* 18 instr. */
      for(k=0; k<(N-14)/12; k++)
         kernel(); /* 12 instr. */
      if(r2flag == 1)
    oddepilogue; /* 18 instr. */
   else
    evenEpilogue; /* 14 instr. */
   }
}
```

Fig. 2. Pseudocode illustrating structure and control flow of program code.

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 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 transports 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 mixed radix FFT contains a total of 107 instructions and the instruction length was 167 bits. The execution of 1024-point FFT takes 5264 instruction cycles, thus the overhead to the theoretical lower bound with dual-port data memory (5120 cycles) is only 3% (144 cycles). Maximum overhead is 208 and minumum overhead is 78 cycles.

Overheads in calculation of larger FFT is negligible compared to overheads seen in typical software implementations.

F. 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 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 [12]. 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.

In order to reduce the power consumption of the 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 167 bits to 7 bits. The decompression, i.e., the dictionary access was supplemented to the control unit without additional pipeline stage. The actual dictionary was implemented using standard cells.

TABLE I
CHARACTERISTICS OF MIXED RADIX FFT PROCESSOR ON 130 NM ASIC TECHNOLOGY WITH 1.5V SUPPLY VOLTAGE.

Clock Cycles	**174 to 114891**	Execution Time	**676 ns to 489,56 μs**
Power	**75,8 to 104 mW**	Clock Frequency	**250 MHz**
Area	**890 kgates**	Energy	**52,76 nJ to 47,79 μJ**

VI. Performance Analysis

In order to analyse the characteristics of the proposed processor, the structures of the previous special function units were described manually in VHDL. The structural description of the proposed core was obtained with the aid of the TCE [13], which generated the VHDL description and provided fast and illustrating simulator for design of the proposed processor.

Then the proposed processor 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 I. It should be noted that the memories take from 40 to 52% of the total power consumption of 75,8 to 104 mW with 1.5V supply voltage. If the supply voltage is reduced to 1.1V, the total power consumption will drop down to about 40 to 55 mW. However, this will reduce the maximum clock frequency.

Table II 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 [14] employs custom circuits, clock gating, and reduced supply voltage. The Stratix [15] is an FPGA solution with dedicated embedded FFT logic usign Altera Megacore function. The TI C6416 [16] is a digital signal processor and the Imagine [17] 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

TABLE II

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
proposed	130	1.5	250	21.06	505	max 1K FFT	130	1.5	250	21.06	641
Imagine	150	1.5	232	16.0	16		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
						Intel P4	130	1.2	3000	23.9	0.8

parts) [18]. However, in-place computations cannot be used and the processor has eight memory ports while the proposed processor uses only two. The Intel Pentium-4 [19] 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] FFT-processor employs radix-2/4/8 single-path delay algorithm. MIT FFT uses subtreshold circuit techniques [8].

Only the MIT FFT outperforms the design. However, due to its long execution time, the MIT FFT is not usable in high performance designs. The performance of the proposed 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.

VII. CONCLUSIONS

In this paper, a low-power application-specific processor for FFT computation with different trasform sizes 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, 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] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*. Reading, MA: Addison-Wesley, 1985.

[2] A. Chandrakasan, S. Sheng, and R. Brodersen, “Low-power CMOS digital design,” *IEEE Journal of Solid State Circuits*, vol. 27, no. 4, pp. 473–483, Apr. 1992.

[3] K. Reeves, K. Sienski, and C. Field, “Reconfigurable hardware accelerator for embedded DSP,” in *Proc. SPIE High-Speed Comp. Dig. Sig. Proc. Filtering Using Reconf. Logic*, J. Schewel, P. M. Athanas, V. M. Bove, and J. Watson, Eds., vol. 2914, Boston, MA, Nov. 20–21 1996, pp. 332–340.

[4] A. Chang and W. Dally, “Explaining the gap between ASIC and custom power: A custom perspective,” in *Proc. IEEE DAC*, Anaheim, CA, June 13–17 2005, pp. 281–284.

[5] B. M. Baas, “A low-power, high-performance, 1024-point FFT processor,” *IEEE Solid State Circuits*, vol. 43, no. 3, pp. 380–387, March 1999.

[6] Y. Zhao, A. Erdogan, and T. Arslan, “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, Apr. 4–8 2005.

[7] Y.-T. Lin, P.-Y. Tsai, and T.-D. Chiueh, “Low-power variable-length fast fourier transform processor,” *Proc. IEE Computers and Digital Techniques*, vol. 152, no. 34, pp. 499–506, July 8 2005.

[8] A. Wang and A. Chandrakasan, “A 180-mV subthreshold FFT processor using a minimum energy design methodology,” *IEEE J. Solid State Circuits*, vol. 40, no. 1, pp. 310–319, Jan. 2005.

[9] J. Granata, M. Conner, and R. Tolimieri, “Recursive fast algorithms and the role of the tensor product,” *IEEE Trans. Signal Processing*, vol. 40, no. 12, pp. 2921–2930, Dec. 1992.

[10] H. Corporaal, *Microprocessor Architectures: From VLIW to TTA*. Chichester, UK: John Wiley & Sons, 1997.

[11] L. Wanhammar, *DSP Integrated Circuits*. San Diego, CA: Academic Press, 1999.

[12] C. Lefurgy and T. Mudge, “Code compression for DSP,” EECS Department, University of Michigan, Technical Report CSE-TR-380-98, Nov. 1998.

[13] P. Jääskelainen, “Instruction set simulator for transport triggered architectures,” Master’s thesis, Tampere Univ. Tech., Tampere, Finland, Apr. 2005.

[14] *StrongARM SA-110 Microprocessor for Portable Applications Brief Datasheet*, Intel, 1999.

[15] S. Lim and A. Crosland, “Implementing FFT in an FPGA co-processor,” in *The International Embedded Solutions Event (GPSx)*, Santa Clara, CA, Sept. 2004, pp. 230–233.

[16] S. Agarwala, and group, “A 600 MHz VLIW DSP,” *IEEE J. Solid State Circuits*, vol. 37, no. 11, pp. 1532–1544, Nov. 2002.

[17] S. Rixner, W. Dally, and group. , “A bandwidth-efficient architecture for media processing,” in *Proc. Annual ACM/IEEE Int. Symp. Microarchitecture*, Dallas, TX, Nov. 30 – Dec. 2 1998, pp. 3–13.

[18] *TMS320C64x DSP Library Programmer’s Reference*, Texas Instruments, Inc., Dallas, TX, Oct. 2003.

[19] M. Deleganes, J. Douglas, B. Kommandur, and M. Patyra, “Designing a 3 GHz, 130 nm, Intel® Pentium ®4 processor,” in *Digest of Technical Papers Symp. VLSI Circuits*, Honolulu, HI, June 13–15 2002, pp. 230–233.