CUSTOMIZED TTA PROCESSOR FOR EFFICIENT IMPLEMENTATION OF VARIABLE LENGTH FFT IN SDR SYSTEMS

Tomasz Patyk, David Guevorkian, Teemu Pitkänen, and Jarmo Takala

Department of Computer Systems, Tampere University of Technology,
P.O. Box 527, FI-33101 Tampere, Finland, e-mail: name.surname@tut.fi

ABSTRACT

In this paper, we describe a processor architecture tailored to mixed-radix-4/2/3 FFT algorithm. The proposed design supports FFT sizes $64 - 2048/1536$ needed in different radios such as WLAN, LTE, DVB, *etc*. Thus different radios of a SDR system may use the same FFT implementation. The processor is based on the Transport Triggered Architecture (TTA) processor structure customized with a set of functional units. Those units were designed especially for the application at hand, that is, for radix-2/3/4 FFT butterfly operations. The proposed processor has been synthesized on a 130 nm standard cell ASIC technology. Efficiency analysis illustrates that, while the developed processor is programmable, its efficiency is comparable to that of fixed-function ASIC implementations.

Index Terms— Fast Fourier Transform (FFT), Long-Term Evolution (LTE), Application Specific Integrated Circuit (ASIC), Parallel Architectures, Software-defined Radio (SDR), Transport-Triggered Architecture (TTA),

1. INTRODUCTION

Very high computational demands of digital radio systems require efficient, customized implementations of the algorithms most frequently implemented in those systems. Typically, specialized hardware (HW) architectures have been used to achieve the needed efficiency. A software-defined radio (SDR) assumes flexible, programmable implementations of several radios sharing the same HW resources, which therefore, cannot be optimized for a single algorithm or a particular radio. This brings even higher demand for the computational power. At the same time, the design should be really low-power and low-cost to be useful, since, the main target devices are portable consumer electronics, such as, mobile-(smart-)phones, laptops, *etc.*

In order to support such tight requirements, the design of SDR platforms should consider optimization of both the HW and software (SW) components. In other words, efficient HW/SW co-design approaches may significantly improve performance of SDR platforms. One of the most advanced HW/SW co-design approaches is provided by Transport-Triggered Architecture (TTA) [1] processor template supported by TTA-based Co-design Environment (TCE) toolchain [2]. It is, therefore, of high interest to use the TTA-TCE co-design framework to design application-customized processors for efficient implementation of SDR algorithms.

One of the most often implemented and, at the same time, computationally demanding algorithms in practically every radio system, is the Discrete Fourier Transform (DFT). Interest towards efficient implementation of the DFT started in 1965 from the famous Cooley-Tukey Fast Fourier Transform (FFT) algorithm presented in [3]. Still, after almost half a century, this interest remains very high due to the fundamental, useful properties of the DFT. in particular, the application in the Long Term Evolution (LTE) [13] wireless communication standard. Those technologies require very efficient implementations of the FFT in order to support extremely tight, mutually contradicting constraints such as real-time processing requirements on top of the low-power, low-cost, and flexible HW platforms. However, the main difference, in comparison to FFT implementations in other fields, is related to the necessity to support efficiency of the same platform to implement DFTs of multiple sizes. Those sizes, not necessarily need to be powers of two.

For example, in LTE, computation of a DFT of series of orthogonal frequency-division multiplexing (OFDM) symbols is needed [13] with the speed of $66.67\mu s$ per symbol. Each symbol is a N-length vector of complex numbers, where N may take one of the following values: $N = 128, 256, 512, 1024, 1536,$ or 2048 [13]. In a SDR system, SW implementation of several radios, one of them typically being LTE, should be supported on top of a shared HW platform [14]. Therefore, in SDR, even wider range of FFT sizes need to be supported under even tighter requirements. Thus, there is a great demand for efficient, very high-speed programmable implementation of FFTs of various sizes, including the sizes that are not powers of two.

There is a vast amount of different implementations of FFT, e.g., [4]–[12] to mention only few most recent publications related to communication applications. In particular, mixed-radix-4/2 [4]–[9], and mixed-radix-4/2/3 [10] variable length FFT implementations were proposed. In most of the publications, either dedicated (fixed-or-reconfigurable) FFT

$$F_{4^m2^n3^l} = \left[\left(F_3 \otimes I_{\frac{lN}{3}}\right) T^N_{\frac{lN}{3}}\right] I_3 \otimes \left\{\left(F_2 \otimes I_{\frac{N}{2\cdot 3}}\right) T^{\frac{N}{3}}_{\frac{N}{2\cdot 3}} \left[\prod_{i=1}^{m}\left(I_{2^k} \otimes I_{4^{i-1}} \otimes F_4 \otimes I_{4^{i-1}}\right)\left(I_{2^k} \otimes I_{4^{i-1}} \otimes T^{4^{m-i+1}}_{4^{m-i}}\right)\right]\right\} R_{4^m2^n3^l} \quad (1)$$

HW architectures [6]–[10] or SW FFT implementations on existing processor architectures, [11]–[12] are proposed.

Conventionally, HW implementations are thought to provide better performance but poor flexibility while the SW implementations are thought to provide high flexibility but poor performance. However, recently, it has been shown that programmable FFT implementations with performances comparable to that of fixed HW implementations may be achieved by making use of HW/SW co-design methodology provided by TTA-based Co-design Environment (TCE) technology [2]. Using this technology, new customized TTA processor architectures were proposed and Assembly implementations of mixed-radix-4/2 FFTs were developed in [4], [5]. The resulting implementations illustrate execution time and power consumption performance similar to those of fixed FFT HW accelerators. Unfortunately, implementations of [4], [5] consider only FFT of sizes $N = 4^m2^n$ being powers of two.

In this work, we further develop this approach and propose a new, customized Transport Triggered Architecture (TTA) processor for programmable implementation of mixed-radix-4/2/3 FFTs of sizes $N = 4^m2^n3^l$. Thus, in particular, FFTs of all the sizes needed in LTE, $128 - 2048/1536$, are supported. Compared to processors presented in [4], [5], the proposed architecture achieves not only higher flexibility, by supporting FFT sizes being a multiple of 3, but also further improves the performance due to the optimization of the previous designs. In particular, modified functional units for twiddle factor generation and operand address generation, feature shorter critical paths, thus, allowing synthesize of the processor for higher frequencies.

The rest of the paper is organized as follows. Section 2 explains details of the FFT algorithm used in this work. Section 3 presents some features of the TTA processor architecture, while Section 4 specifies the implementation of the mixed-radix-4/2/3 FFT TTA processor. Section 5 discusses results of the timing and area analysis conducted on the design. Finally, Section 6 concludes the paper.

2. FFT ALGORITHM

The DFT of an input vector $\mathbf{x} = [x_0, x_1, ..., x_{N-1}]^T$ is defined as the vector $\mathbf{y} = [y_0, y_1, ..., y_{N-1}]^T$ such that:

$$y_m = \sum_{n=0}^{N-1} W_N^{nm} x_n, \quad (2)$$

or equivalently:

$$\mathbf{y} = F_N \mathbf{x}, \quad (3)$$

where F_N is the $(N \times N)$-matrix of DFT with entries:

$$W_N^{nm} = \exp -\frac{i2\pi nm}{N}. \quad (4)$$

Many Fast Fourier Transform (FFT) algorithms were developed for efficient computation of the DFT. In this paper, we are using the in-place, decimation-in-time (DIT), mixed-radix-4/2/3 algorithm with permuted input, and in-order output. The formula for our FFT algorithm of size $N = 4^m2^n3^l$, where $m \in \mathbb{N}_0$, $n \in \mathbb{N}_0$, and $l = 0, 1$, is given by (1). The following notations were used in this formula:

$$F_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad (5)$$

$$F_3 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & W_N^{\frac{N}{3}} & W_N^{\frac{2N}{3}} \\ 1 & W_N^{\frac{2N}{3}} & W_N^{\frac{N}{3}} \end{pmatrix}, \quad (6)$$

$$F_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}, \quad (7)$$

where $W_N^m = e^{i2\pi\frac{m}{N}}$ and:

$$W_N^{\frac{N}{3}} = -\frac{1}{2} - i\frac{\sqrt{3}}{2}, \quad (8)$$

$$W_N^{\frac{2N}{3}} = -\frac{1}{2} + i\frac{\sqrt{3}}{2}, \quad (9)$$

$$W_N^{\frac{N}{3}} = \left(W_N^{\frac{2N}{3}}\right)^*. \quad (10)$$

I_N stands for an identity matrix of order N and $\otimes$ denotes a tensor product. T matrices, holding twiddle factors for the corresponding radix-4, radix-2, and radix-3 computation stages, are obtained with:

$$T^{rs}_s = \oplus_{i=0}^{r-1}(D^{rs}_s)^i, \quad (11)$$

$$D^{rs}_s = diag(W^0_s, W^1_s, \ldots, W^{s-1}_s), \quad (12)$$

where $\oplus$ denotes a matrix direct sum. Finally, $R_{4^m2^n3^l}$ is an input permutation matrix based on the stride-by-S permutation matrices P of order N, and is given by the formula:

$$R_{4^m2^n3^l} = \left\{ I_{3^l} \otimes \left[I_{2^n} \otimes \Big(\prod_{i=1}^{m} I_{4^{m-i}} \otimes P_4^{4^m} \Big) \right] \right\} P_{3^l}^N. \quad (13)$$

With the formula (1), a DFT of an input vector of size $N = 4^m 2^n 3^l$ can be computed in $n + m + l$ stages as illustrated in Fig. 1 for the case of $N = 24 = 4^1 2^1 3^1$. Each stage consists of two substages. In the first substage, multiplication of a diagonal matrix to the vector of intermediate results is performed. This, in fact means multiplying so called twiddle factors (powers of $W_N = \exp -\frac{i2\pi}{N}$) with the inputs to the stage. Note that all the twiddle factors of the first stage are equal to unity and, therefore, are not shown in Fig. 1. The second substage is a multiplication of a sparse matrix to the vector of intermediate results obtained after first substage. Each sparse matrix is presented as a Kronecker product and may, by row and column permutations, be transformed to a block-diagonal matrix with N/r blocks being DFTs of size r, where $r = 4$ for the first m stages, $r = 2$ for the next n stages, and $r = 3$ for the last l stages ($l = 0, 1$). Multiplying such matrix to a vector means implementing N/r so called radix-r butterfly operations.

Properties of the employed representation (1) release certain benefits as compared to the other approaches of computing the DFT. We took advantage of them while implementing our processor. In particular, an in-place algorithm ensures that the size of the data memory used, can be limited to the maximum size of the FFT that the processor supports. This is especially important in the embedded applications. The chosen order of the different radix stages, as well as, the use of the in-order output rather than in-order input algorithm, simplified some of the functional units of the processor. The operand address generator, and twiddle factor generators, described in more details in Section 4.2, are build up from fewer gates performing less switching activity compared to [4], [5]. This should lead to the decrease of both static and dynamic power consumption.

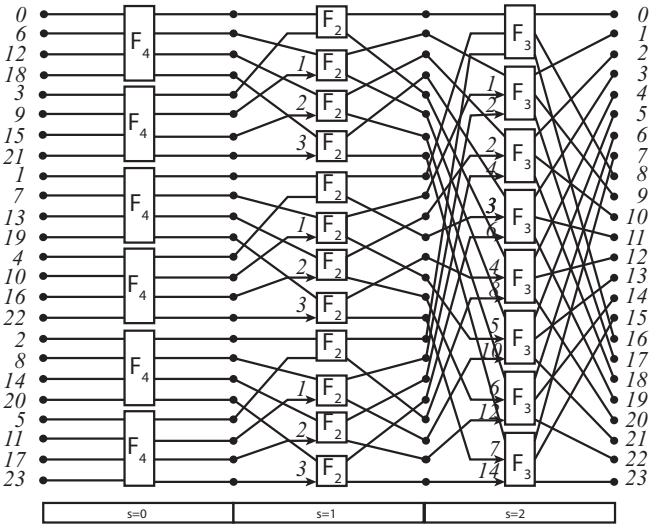

Fig. 1. Signal flow graph of a 24-point mixed-radix FFT. Constant k in the signal flow graph represents the twiddle factor W_N^k, where $N = 8$ for the radix-2 stage, and $N = 24$ for the radix-3 stage.

3. TRANSPORT TRIGGERED ARCHITECTURE

The proposed processor is based on the Transport Trigger Architecture (TTA) [1] processor template. The TTA falls into the category of statically programmed instruction-level parallelism (ILP) architectures. It belongs to the class of the exposed data path, Very Long Instruction Word (VLIW) processor architectures, where the details of the data path transfers are disclosed to the software designer. This can be benefited twofolds by: allowing unique optimizations in the code; and the customization of the data path interconnection network.

The TTA processor is a modular design including a set of register files (RF), functional units (FU), a control unit, and an interconnection network (IC) between the data path resources. The programming model of the TTA differs when compared to the general purpose processor architectures, e.g., RISC or CISC, where instructions are decoded into control signals which initialize the operations. Instead, TTA instructions specify data moves in the IC. Data are written into the input ports, and read out of the output ports of the processor units. Each unit has a single triggering input port, which triggers the operation of that unit, whenever data is written into it. Therefore, the operations of the processor can be seen as a "side effect" of the data transports. If the inputs of the FU are registered, a set of operations can be performed on the same set of input data, by triggering the unit with different opcodes. Sharing the operands between different operations of the FU reduces data traffic over the IC, and the need for temporal storage in the RFs or data memory.

It can be shown that the modular nature of the TTA architecture can be exploited for reduction of the power consumption, as well as custom optimization for speed or area. Firstly, the processor structure can be tailored specifically to fit the application it is designed for. Removing unnecessary resources, e.g., FUs, and connection sockets from the IC, reduces the static power consumption due to the reduced gate count. Secondly, one can include in the processor design special function units (SFUs) with the user-defined functionality. Those units can not only perform application-specific task more power-efficiently but also reduce the number of necessary data moves over the IC. SFUs also let to reduce the instruction overhead, thus, they reduce the power consumption due to the instruction fetch. In general, dynamic power consumption reduction can be obtained. The speed or area optimization can be obtained by adding/removing resources to the processor. One can add more resources to improve performance by exploiting ILP. Leaving only necessary resources will reduce the silicon area the processor takes to minimum. A trade-off between speed and area can be worked out according, e.g. to requirements. In this work, we have used several

custom-designed units tailored for the FFT application, which guarantee a trade-off between execution time, area, and power consumption of the processor.

4. PROCESSOR ORGANIZATION

The work presented in this paper modifies and extends the mixed-radix-4/2 FFT TTA processor presented in [4] and [5].

4.1. Mixed-radix-4/2 FFT TTA

The processor described in [4] and [5] implements the in-place, DIT, in-order input, mixed-radix-4/2 FFT algorithm. The processor supports FFT sizes in the range of $64 - 16384$. It calculates the FFT with several radix-4 stages, followed by a single radix-2 stage when the FFT size is not a power-of-four. Since the FFT is inherently a complex-valued algorithm, the 32-bit word was adopted to represent the complex numbers, where 16 most significant bits hold the real part, and 16 least significant bits hold the imaginary part. Appropriate, arithmetic special function units were designed to perform the complex multiplication and addition, required by the DIT radix-2 and radix-4 butterflies.

Indexing of the input data for every stage of the FFT computations can be represented by a permutation matrix [15]. However, instead of using generic arithmetic units to calculate the indexes, one can perform index manipulations on the bit-level. This proved to be a low-power solution and was implemented as a special function unit as well. Yet another low-power solution was called a twiddle factor (TF) generator. In principle, the generator exploits the redundancy among TFs. Instead of storing all the required TFs in a look-up table, only $1/8 + 1$ of them is kept in the memory, while the rest is generated by modifying the stored ones.

4.2. Mixed-radix-4/2/3 FFT TTA

Considering FFT sizes required by the LTE standard, the processor presented in Section 4.1 supports them all, with the exception of 1536-point FFT. The proposed design supports also this case. Out of several different mixed-radix decomposition possibilities, we chose the mixed-radix-4/2/3 DIT, in-place algorithm with in-order output. There were two reasons behind this choice: 1) the maximum possible utilization of the existing mixed-radix-4/2 processor; and 2) simple implementation of radix-3 SFUs. Still, the latter one was not an obvious choice. In general, radix-3 stage operand index and TF calculations can be done in a simple manner if the ternary number system is used. In the binary system, on the other hand, bit-level operations are not possible. However, if we decompose a 1536-point FFT into three 512-point FFTs followed by a single radix-3 stage, the operand address and twiddle factor generators for radix-3 stage can be implemented with bit-level operations. The three 512-point FFTs can be computed by the modified, mixed-radix-4/2 processor , introduced in Sec. 4.1. Following paragraphs list SFUs that our processor consists of, as well as, the general organization of the design.

A. Mixed-radix-4/2 Operand Access Generator

The Mixed-radix-4/2 Operand Access Generator is a modified design of the Address Generator (AG) presented in [4]. The main principle, of rotating the address of the operand to be fetched from the data memory, still applies. However, since the FFT algorithm has changed from in-order input to in-order output, the unit had to be updated to the new order of operand access.

B. Mixed-radix-4/2 Twiddle Factor Generator

The Mixed-radix-4/2 Twiddle Factor Generator is yet another SFU which has been adopted from the [4] design. It rests on the foundation shown in [16], that proves, that the twiddle factors for mixed-radix-4/2 can be generated from $N/8 + 1$ of the all coefficients simply by manipulating their real and imaginary parts. However, the change of the FFT algorithm, from in-order input to in-order output, resulted in apparent simplification of the permutation logic. This logic, generates the index of the base coefficient, which is fetched from the ROM memory, and latter modified by six different combinations of the exchange, addition, or subtraction operations. The simplified logic of the unit shorten the critical path for the whole processor. It was one of the optimizations which allowed to synthesize the new processor at significantly higher operating frequency, namely $400MHz$ compared to $250MHz$ in [4].

C. Mixed-radix-4/2 Butterfly Unit and Complex Adder

Mixed-radix-4/2 Butterfly Unit and Complex Adder SFUs are exactly the same like in [4], [5]. They implement the arithmetics of the complex numbers, addition and multiplication respectively.

D. Radix-3 Operand Access Generator

As explained in Section 4.2, the chosen FFT algorithm allowed to simplify the implementation of the processor . The radix-3 stage is executed at the very end of the computations, where all the samples are already in-order. This makes the implementation of the Radix-3 Operand Access Generator straightforward. For the $1536 - point$ FFT, indexes of the input samples to each radix-3 butterfly, are always separated by 512, e.g., 0, 512, 1024; 1, 513, 1025 etc. If we generate the index for the first sample with a linear up-counter, the remaining two indexes can be calculated by bitwise *OR* operation with 512 and 1024, for $2nd$ and $3rd$ input sample

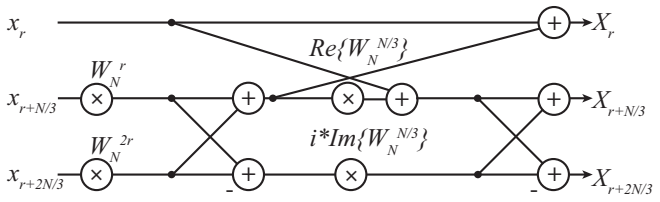

Fig. 2. Signal flow graph of the modified radix-3 butterfly [17].

respectively.

E. Radix-3 Twiddle Factor Generator

The implementation of the Radix-3 Twiddle Factor Generator follows the same principles applied to the Mixed-radix-4/2 Twiddle Factor Generator discussed earlier. The biggest difference occurs in the input logic, which generates the index of the coefficient to be fetched from the ROM memory. Each radix-3 butterfly requires two TFs. For the in-order output FFT, the index of the $1st$ TF can be generated with the linear up-counter, while the $2nd$ index of the same butterfly is the $1st$ one multiplied by 2. On the bit-level, the multiplication by 2 can be substituted by a left shift . The unit requires also a number of 193 coefficients stored to the ROM memory.

F. Radix-3 Butterfly Unit

The Radix-3 Butterfly Unit implements the algorithm for FFT radix-3 butterfly computations as in [17]. Fig. 2 presents the signal flow graph of the computations. Compared to the classic radix-3 butterfly implementation the number of arithmetic operations has been reduced from 6 complex multiplications to 2 complex and 2 real multiplications by a constant. The number of complex additions equals 6 in both cases.

G. General Organization of the Processor

The general organization of the proposed TTA processor for mixed-radix-4/2/3 FFT computations is shown in Fig. 4. The processor is composed of 12 separate FUs and a total of 16 RFs containing 25, 32-bit general-purpose registers and 6 Boolean registers. The FUs and RFs are connected by an IC consisting of 25 buses, compared to 23 that the mixed-radix-4/2 design has. The number of connecting sockets has been hand-optimized down to 111. In addition, the processor has a control unit, instruction memory, and 32-bit, dual-ported data memory. The size of the data memory is limited by the size of the maximum FFT to be supported by the processor, in our case, $2048 + 1$. The last word holds the size of the FFT to be computed.

The in-order FFT algorithm applied in our design implies

```
main() {
  initialization(); /* 10 to 35 instructions */
  if (fftSize == 1536) {
    for (iter = 0; iter < 3; iter++) {
      radix42_prologue();       /* 14 instr. */
      for (idx = 0; idx < 512; idx++)
        radix42_kernel();       /* 16 instr. */
      radix42_epilog();         /* 15 instr. */
    }
    radix3_prologue();          /* 12 instr. */
    for (idx = 0; idx < 1536; idx++)
      radix3_kernel();          /* 12 instr. */
    radix3_epilogue;            /* 12 instr. */
  } else {
    radix42_prologue();         /* 14 instr. */
    for (idx = 0; idx < (Nlog_4N)/16 - 1; idx++)
      radix42_kernel();         /* 16 instr. */
    radix42_epilog();           /* 15 instr. */
  }
}
```

Fig. 3. Pseudo code illustrating structure and control flow of the program.

the need for a permutation of the input samples, so that, the correct results are produced. In case of the power-of-two FFTs only permutation is required. For the $1536 - point$ FFT initial decomposition, into 3 sets, comprising every $3rd$ sample, is needed. Those operations should be taken into account when integrating our design into the system with a source of samples, and the sink for results. Integration details remain beyond the scope of this paper.

H. Instruction Schedule

Traditionally, the kernel of the FFT algorithm is implemented with three nested loops. We use a different approach, where the Twiddle Factor Generators, as well as, the Operand Access Generators use a single up-counter and a FFT size stored in a register, to fetch and calculate correct operands. This results in a single *for* loop implementation for power-of-two FFTs, as shown in the "else" branch of the pseudo code in Fig. 3. Calculating $1536 - point$ FFT requires two consecutive *for* loops, as illustrated in the "if" branch of the pseudo code in Fig. 3. The first one computes 3 times a $512 - point$ mixed-radix-4/2 FFT on 3 sets of data, decomposed as explained in the previous paragraph. Subsequently, a radix-3 FFT is calculated on all 1536 samples producing a final result. The software implementation is a hand-optimized TTA-assembly code, which exploits both, the instruction level parallelism, and the software pipelining.

5. PERFORMANCE ANALYSIS

In order to evaluate our processor, we implemented all SFUs from Section 4.2 in hand-written VHDL. The structural description of the processor core was obtained with *PRODE*,

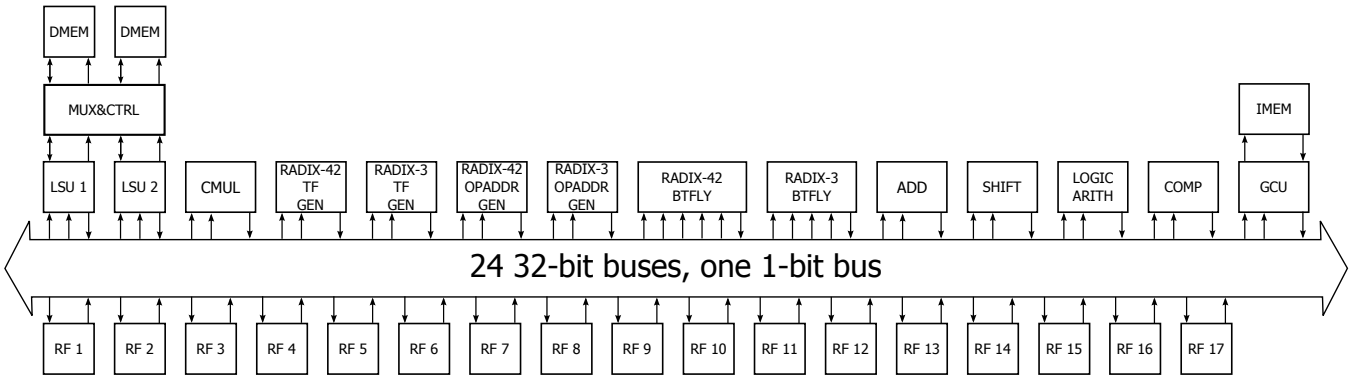

Fig. 4. Block diagram of the proposed FFT processor. DMEM: data memory. IMEM: instruction memory. LSU: load-store unit. TF GEN: twiddle factor generator. OPADDR GEN: operand address generator. CMUL: complex multiplier. RADIX-42 BTFLY: complex adder. RADIX-3 BTFLY: radix-3 butterfly calculation unit. ADD: adder. LOGIC ARITH: logical unit. SHIFT: shifter. CMP: compare unit. GCU: control unit. RF: register file.

Table 1. Characteristics of the proposed processor synthesized on 130 nm ASIC technology (mixed-radix-4/2/3) versus design presented in [5] (mixed-radix-4/2).

	mixed-radix-4/2	mixed-radix-4/2/3
supported FFT sizes	$64-16384$	$128-2048/1536$
cycle count	$207-114722$	$544-12335$
execution time	$828ns-459\mu$s	$1.3-30.8\mu$s
	$@250MHz$	$@400MHz$
max. clock freq.	$250MHz$	$400MHz$
Area [kgates]		
Core	38	46
Imem	2	3
Dmem	240	60
Total	280	109
1024-point FFT		
cycle count	5160	5180
1536-point FFT		
cycle count	N/A	9324
2048-point FFT		
cycle count	12332	12345

a VHDL description generator from the TCE toolchain [2]. From the same toolchain, *PROXIM*, an instruction accurate simulator, was used to measure the number of clock cycles the program execution takes. Complete VHDL description of the processor design was then synthesized to a 130 nm standard cell CMOS technology with Synopsys Design Compiler. Gate level simulation was performed at 400 MHz. The obtained results are compared in Table 1 to design in [5]. As can be seen, the proposed design is more than twice smaller than the previous design , while it supports FFT of size 1536 in addition to FFTs of sizes being powers-of-two. At the same time, the proposed design may be clocked at approximately 1.6 times higher maximum frequency and it uses approximately the same number of clock cycles as the previous design for implementation of FFTs of equal sizes. Thus, faster FFT implementations on a smaller device is achieved. The processor operates in a pipelined fashion, i.e. different FUs process different samples from the same computational set, in the particular clock cycle. This is possible due to the TTA architecture, which exposes data transfers on the processor buses to the software designer. The processor does not have a fixed pipeline, typical to other architectures, rather the programmer (or compiler) can create a pipeline by scheduling data transfers in a particular manner. Hand-optimized assembly code guarantees throughput of 1 sample/clock cycle (s/cc), after latency of several cycles, needed to fill the pipeline in. The latency varies from one FFT size to another but does not exceed 57 cycles. For instance, the theoretical throughput of 1 s/cc for mixed-radix-4/2 $2048-point$ FFT, with five radix-4 stages, and 1 radix-2 stage is $2048*6=12288$ cc. Our processor requires only 57 cycles more, which is the maximum latency for FFT computations with this design.

6. CONCLUSIONS

In this paper, a programmable HW/SW co-design of a FFT processor is proposed. The design, based on a TTA processor template, has been customized for the FFT computations, having in mind especially FFT sizes required by the LTE applications. A set of hand-optimized special function units was designed . The processor supports power-of-two FFT sizes in the range $128-2048$ and the 1536-point FFT. The latter one is obtain with the added support for radix-3 FFT computations.

The processor was synthesized on a 130 nm ASIC technology . Timing, and area analysis showed that the proposed design features higher performance. The proposed design is more than twice smaller the previous design, it supports $1536-point$ FFT, and can be clocked at approximately 1.6 times higher maximum frequency. The hand-optimized assembly code allows to execute computations in pipelined fashion with throughput of 1 sample/clock cycle after a small number of clock cycles required to fill the pipeline in.

Future work will include adding the radix-5 support, as well as, expanding the range of computable FFT-sizes to other than 1536, sizes divisible by 3.The power-efficiency analysis and optimization shall also be done.

7. REFERENCES

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

[2] "TTA-based co-design environment (TCE)," http://tce.cs.tut.fi/.

[3] J.W. Cooley and J.W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, pp. 297–301, April 1965.

[4] T. Pitkänen and J. Takala, "Low-power application-specific processor for FFT computations," in *Proceedings of IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP-2009), Taipei, Taiwan*, April 2009, pp. 593 – 596.

[5] T. Pitkänen, R. Mäkinen, J. Heikkinen, T. Partunen, and J. Takala, "Transport triggered architecture processor for mixed-radix FFT," in *Conf. Record Asilomar Conf. Signals, Syst. Comput., Pacific Grove, CA*, Oct. 2006, pp. 84–88.

[6] Ch.-H.Yang, T.-H. Yu, and D. Marković, "Power and area minimization of redonfigurable FFT processors: a 3GPP-LTE example," *IEEE Journal od Solid-State Circuits*, vol. 47, no. 3, pp. 757–768, March 2011.

[7] G. Yang and Y. Jung, "Scalable FFT processor for MIMO-OFDM based SDR systems," in *Proceedings of 5th Int. Symp. on Wireless Pervasive Computing (ISWPC), Modena, Italy*, May 2010, pp. 517–521.

[8] A. Karachalios, K. Nakos, D. Reisis, and H. Alnuweiri, "A new FFT architecture for 4×4 MIMO-OFDMA systems with variable symbol length," in *Proceedings of the 6th Int. Conf. on Innovations in information technology IIT'09,*, Dec. 2009, pp. 1–5.

[9] R. Pandey and M.L. Bushnell, "Architecture for variable-length combined FFT, DCT, and MWT transform hardware for a multi-modewireless system," in *Proceedings of IEEE Int. Conf. on VLSI Design held jointly with 6th Int. Conf. on Embedded Systems, Bangalore, India*, January 2007, pp. 121–126.

[10] S.-Y. Peng, K.T. Shr, Ch.-M. Chen, and Huang Y.-H., "Energy efficientt $128 \sim 2048/1536-$ point FFT processor with resource block mapping for 3GPP-LTE system," in *Proceedings of IEEE Int. Conf. on Green Circuits and Systems (ICGCS)*, June 2010, pp. 14–17.

[11] N.K. Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, and J. Manferdelli, "High performance discrete Fourier transforms on graphics processors," in *Proceedings of IEEE Int. Conf. on High Performance Computing, Networking, Storage and Analysis (SC-2008), Austin, Texas, USA*, Nov. 2008, pp. 1–12.

[12] V. Volkov and B. Kazian, "Fitting FFT onto the G80 architecture," in *CS 258 final project report,*. 2008, University of California, Berkeley.

[13] "The 3rd generation partnership project (3GPP)," http://www.3gpp.org/lte/.

[14] A. Ahtiainen, H. Berg, U. Lcking, A. Pärssinen, and J. Westmeijer, "Architecting software radio," in *Proceedings of the SDR 07 Technical Conference and product Exposition*, 2007.

[15] H. V. Henderson and S. R. Searle, "The vec-permutation matrix, the vec operator and kronecker products: A review," *Linear and Multilinear Algebra*, vol. 9, no. 4, pp. 271–288, 1981.

[16] L. Wanhammer, *DSP integrated circuits*, Academic Press series in engineering. Academic Press, San Diego, CA, 1999.

[17] T. Mateer, *Fast Fourier Transform algorithms with applications*, ProQuest, UMI Dissertation Publishing, 2011.