LOW-POWER PROGRAMMABLE PROCESSOR FOR FAST FOURIER TRANSFORM BASED ON TRANSPORT TRIGGERED ARCHITECTURE

Jakub Žádník and Jarmo Takala

Faculty of Information Technology and Communication Sciences, Tampere University, Finland
{jakub.zadnik, jarmo.takala}@tuni.fi

ABSTRACT

This paper describes a low-power processor tailored for fast Fourier transform computations where transport triggering template is exploited. The processor is software-programmable while retaining an energy-efficiency comparable to existing fixed-function implementations. The power savings are achieved by compressing the computation kernel into one instruction word. The word is stored in an instruction loop buffer, which is more power-efficient than regular instruction memory storage. The processor supports all power-of-two FFT sizes from 64 to 16384 and given 1 mJ of energy, it can compute 20916 transforms of size 1024.

Index Terms— Fast Fourier Transform, Transport Triggered Architecture, Application-Specific Instruction-Set Processor

1. INTRODUCTION

Fast Fourier transform (FFT) is one of the most widely used signal processing algorithms thanks to its ability to represent a time-domain signal in a frequency domain. For example, FFT is used in orthogonal frequency division multiplexing (OFDM) systems, which are employed in wireless communication devices. Due to the popularity of embedded and battery-powered systems, minimizing power consumption is a major objective.

Regarding power consumption, application specific integrated circuit (ASIC) implementations are considered as more efficient compared to reconfigurable hardware (such as field-programmable gate array (FPGA), coarse-grained reconfigurable array (CGRA)) or general purpose processors (GPPs). However, ASIC-based FFT processors are mostly fixed-function and lack programmability. While the reconfigurable fabric offers more silicon reusability than ASIC, their functionality can be only modified by a hardware design process similar to ASIC. The goal of this work is to propose a software-programmable mixed radix-4/2 FFT processor with an energy-efficiency comparable to fixed-function ASIC implementations. Other software implementations of FFT are implemented on either GPP [1] or a graphics processung unit (GPU) [2]. However, both of these approaches aim for the best performance and do not provide sufficiently low-power solutions.

Fixed-function ASIC FFT processors can be divided into two categories - pipelined and memory based. Pipelined architectures ([3], [4], [5]) rely on a cascade of processing elements (PEs) processing the input data stream. The intermediate results are stored in a distributed memory system. Due to a higher number of PEs, pipelined architectures consume more power and occupy larger silicon area than memory based architectures. However, they usually have higher throughput, which can lead to a high energy-efficiency.

Memory based architectures ([6], [7]) typically have one PE and data is processed in a sequential fashion. They typically use only one or two global memory elements, thus a conflict-free memory access has to be maintained. The proposed architecture is memory based, using a single-port data memories as it allows for a convenient software-programmable implementation.

The processor was designed using a transport triggered architecture (TTA) template [8]. It improves a previous FFT processor [9] by further increasing its energy-efficiency. Several optimizations were applied to allow compressing the computation kernel into only one - repeatedly executed - instruction word that can be executed in a more energy-efficient way.

2. FFT ALGORITHM

The proposed processor supports all the power-of-two FFT sizes from 2^6 to 2^{14}. A mixed radix-4/2 algorithm was used, following a decimation-in-time (DIT) approach [10] as it provides better signal-to-noise ratio (SNR) compared to decimation-in-frequency (DIF) approach[11]. Otherwise, they share the same arithmetic complexity. Radix-4 is used in a majority of the stages because is requires less operations per FFT than the radix-2 algorithm [12]. At the same time, radix-4 butterfly operation requires only trivial operations. Higher radices require more complicated operations. However, using only radix-4 would restrict the processor to only power-of-four FFT sizes. Therefore, in the last stage of the computation, radix-2 butterflies are used for FFT sizes which can not be computed using radix-4 algorithm (i.e. for FFT sizes 2^k where k is odd). The computation follows an in-place approach where output samples are written back to the same memory locations from which the operands were read. This allows to utilize only one memory module of the size equal to the computed FFT size.

3. TRANSPORT TRIGGERED ARCHITECTURE

TTA [8] is a processor template, which exposes its internal datapaths to a programmer. Similarly to very long instruction word (VLIW) [13], it utilizes long instruction words and instruction-level parallelism. The difference is that TTA gives a programmer the control over the data flow. It is possible to bypass accesses to register files (RFs) by feeding results from one functional unit (FU) directly to the input of another. Register bypassing reduces the required RF size and hardware complexity leading to significant power savings [14].

The data transports are defined by a *move* instruction - the only instruction of the TTA's instruction set. *Moving* data into a *trigger* port of a FU triggers the desired operation. FUs can also have *operand* ports for additional data that can be loaded anytime without triggering the operation. Memory access is performed by load-store unit (LSU) in a similar way as any other instruction. A control unit

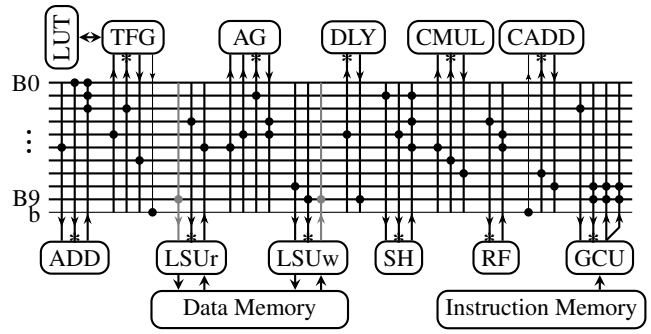

Fig. 1. Architecture of the proposed TTA processor. *TFG*: twiddle factor generator: *AG*: address generator. *DLY*: rotating register as a delay unit. *CMUL*: complex multiplier. *CADD*: complex adder. *ADD*: adder. *LSU*: load-store unit. *SH*: shifter. *RF*: 8x32 register file. *GCU*: general control unit. Gray color denotes unused connections. Trigger ports are marked with '*'.

responsible for instruction fetching, decoding, and executing is also implemented as one of the FUs. Data moves are distributed over an interconnection network consisting of several parallel buses. The number of the parallel buses determines the maximum number of instructions that can be executed in parallel, i.e., the maximum number of simultaneous data moves.

4. PROPOSED PROCESSOR ARCHITECTURE

The proposed architecture is shown in Fig. 1. The architecture consists of ten 32-bit wide buses (B0–B9) and one 1-bit bus (b), represented by horizontal lines. FUs and one RF are connected to the buses. Vertical lines represent sockets, which connect input/output ports of FUs to the interconnection network. The connections are marked as dots.

Two LSUs are connected to a data memory system that behaves like a dual-port memory. In fact, two single-port memories are used and connected to the LSUs via an added logic, which provides a conflict-free memory access. The parallel memory system was chosen due to a lower power consumption of single-port memories compared to multi-port memories [15].

The parity of the address determines which one of the two single-port memories is accessed. In the case when both LSUs are trying to access an address with the same parity (i.e. the same memory module), the processor is temporarily locked and the accesses are resolved sequentially. However, the conflict-free memory access is guaranteed for the FFT addressing scheme.

The streamlined instruction schedule (see Section 6) implies generation of two parallel streams of addresses - read and write. In order to guarantee a different parity for any two parallel addresses (thus conflict-free memory access), a special scheduler module was put between the LSUs and the parallel memory logic described in a previous paragraph. The scheduler internally buffers and reschedules the LSU data in a way that always two parallel read addresses or two parallel write addresses are loaded into the parallel memory logic. Because the address generator preserves parity (see Section 5.1), the scheduler guarantees a conflict-free memory access. The internal buffering is not recognized by a high-level compiler and, therefore, the programming is only possible by low-level assembly. However, it is possible to provide a software-exposed switch in a form of another port of LSU or a special FU that toggles the scheduler on and off, thus preserving a full compiler support for generic applications.

Loop buffer [16] - a critical component of the design - is implemented as a part of the general control unit (GCU). It is a small instruction memory cache used for storing frequently repeated instruction words, e.g., loops. Reading from a loop buffer consumes significantly less power than reading an instruction directly from the instruction memory.

Each single-port data memory is composed of increasingly sized memory blocks (32, 32, 64, 128, ..., 4096 - summing up to total 8192). Based on the access address, only one block is selected at a time while the other do not receive any control signals. This significantly decreases dynamic power consumption when computing smaller FFT sizes.

The processor was designed using TTA-based Co-Design Environment (TCE) toolset developed at Tampere University of Technology (TUT) [17]. TCE provides a comprehensive set of tools for designing TTA processors including a retargetable compiler and a hardware description of the most common FUs. For data and instruction memories, low power Cacti-P models were used [18].

5. SPECIAL FUNCTION UNITS

This section describes special FUs developed specifically for this work. All the other FUs were taken from TCE component libraries.

Complex numbers are represented by two 16-bit fixed point numbers sharing one 32-bit data word. The real part occupies least significant bits (LSBs) of the data word while the imaginary part takes the most significant bits (MSBs).

In order to prevent overflow, each addition is divided by two. When summed up, the complex adder divides the result by four in case of radix-4 and by two in case of radix-2 butterfly. The complex multiplier divides the result by two.

5.1. Address Generator

The address generator (AG) is responsible for computing the memory addresses for butterfly operands. It is generated from a linear counter by a bit pair permutation following the same pattern as the reference implementation [19]. An example of an address generation for a 128-point and 256-point FFT is illustrated in Fig. 2. Each b_i represents an i-th bit of a linear counter. The 'index' bits are sufficient to represent the index within one stage while 'stage' determines the current stage of the computation. The position of the LSB bit pair is determined by the 'stage' part of the linear counter.

The address generator preserves the parity of the linear counter. Thus, any two consecutive addresses have a different parity and if fed in parallel into the parallel memory logic (described in 4), a conflict-free memory access is guaranteed.

5.2. Twiddle Factor Generator

The generation of twiddle factors is based on a lookup table (LUT) implemented as a single-port synchronous read-only memory (ROM) of pre-computed values. It follows the same approach as the one described in [20]. The address for the LUT ROM is computed from the linear index by a bit permutation and scaling based on the current FFT size. Only $N/8 + 1$ complex coefficients need to be stored in the LUT [20]. All the remaining coefficients can be reconstructed by a trivial manipulation (negating and swapping the real and imaginary parts) of the stored coefficients. Therefore, in order to support the maximum 16384-point FFT, the LUT has

stage		index						
b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	b_1	b_0	b_6	b_5	b_4	b_3	b_2
0	1	b_6	b_5	b_1	b_0	b_4	b_3	b_2
1	0	b_6	b_5	b_4	b_3	b_1	b_0	b_2
1	1	b_6	b_5	b_4	b_3	b_2	b_1	b_0

b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	b_1	b_0	b_7	b_6	b_5	b_4	b_3	b_2
0	1	b_7	b_6	b_1	b_0	b_5	b_4	b_3	b_2
1	0	b_7	b_6	b_5	b_4	b_1	b_0	b_3	b_2
1	1	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

Fig. 2. Address generation from a linear counter for 128-point (above) and 256-point (below) FFT.

to contain 2049 coefficients. A side function of the twiddle factor generator (TFG) FU is determining whether the current stage is radix-4 or radix-2. This information is then used by the complex adder (CADD).

5.3. Complex Adder

The CADD performs a butterfly operation on four inputs. Based on its 'rx2' input, it performs either one radix-4 or two radix-2 butterflies.

Traditionally, the CADD would be implemented as a four-input FU with the four inputs buffered in register files before feeding them in parallel into the CADD's ports. However, due to the single-instruction kernel requirement, the register file buffering is not possible since the data can be moved only to a single location. Therefore, the proposed CADD FU has one serial data input port and performs the buffering internally. This makes the FU unusable for high-level programming since this mode of operation can not be recognized by a high-level compiler.

Figure 3 shows the CADD's results based on its 'rx2' input. The 'cnt' column is an internal counter that increments each time a data

rx2	cnt	result
0	00	`a + b + c + d`
0	01	`a - i*b + c + i*d`
0	10	`a - b + c - d`
0	11	`a + i*b - c - i*d`
1	00	`a + b`
1	01	`a - b`
1	10	`c + d`
1	11	`c - d`

Fig. 3. An operation performed by a complex adder based on the value of its 'rx2' input and an internal counter ('cnt'). Four operands (`a`, `b`, `c`, `d`) and rx2 are constant until the next reset of the counter. `i` denotes an imaginary unit.

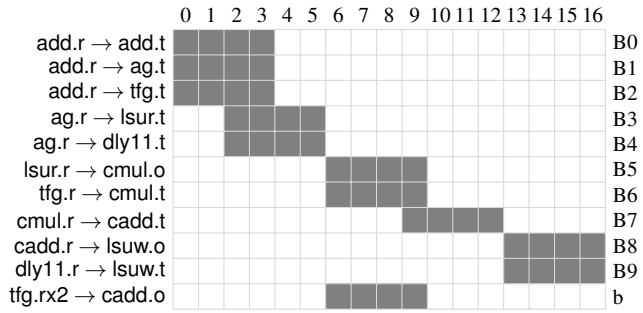

Fig. 4. Bus reservation table of computing one radix-4 butterfly

sample is loaded into the FU's *trigger* port. Both signals form an opcode selecting the operation of the complex adder.

5.4. Complex Multiplier

The complex multiplier performs generic complex multiplication of two operands. The proposed implementation requires four multipliers and two adders.

5.5. Rotating Register

Rotating register is used to delay the address of a butterfly's input sample for the in-place computation. After the butterfly operation is complete, the output of the rotating register is used as an address for the results to store them back to the memory.

6. INSTRUCTION SCHEDULE

The computation of one radix-4 butterfly can be visualized with the aid of a reservation table in Fig. 4. Each column represents one clock cycle. Buses are represented by rows and their names (on the right) correspond to the ones shown in Fig. 1. Gray square denotes that an instruction, i.e., data transfer, is executed on the bus during the clock cycle. The instruction (data *move*) transferred on each bus is shown on the left. The syntax respects the following pattern: source.port → destination.port. Source and destination are FUs. Port can be t (trigger), o (operand), r (result) and rx2 (output port of TFG signalizing whether the butterfly is radix-4 or radix-2).

Full FFT is computed by repeating the above pattern multiple times every four clock cycles. At 13th clock cycle, the bus utilization reaches 100% and the instruction word becomes constant until no new samples need to be computed. Thus, the execution can be separated into three stages: prologue (first 13 cycles), kernel (length depends on FFT size) and epilogue (last 13 cycles). The size of the prologue and epilogue is constant for all FFT sizes. Because the kernel consists of only one repeated instruction word, it can be loaded into the loop buffer from where it can be fetched consuming minimal power.

Apart from the prologue, kernel and epilogue, a setup code consisting of 6 instructions is present to distribute static parameters between FUs. Thus, the size of the complete code is 33 (6+13+1+13) instructions. The architecture uses 51-bit wide instruction words.

7. EVALUATION

The processor was synthesized using Synopsys Design Compiler and two IC technologies were used - a 28 nm FDSOI low-power technol-

	type	tech. (nm)	volt. (V)	freq (MHz)	WL (bits)	t (μs)	power (mW)	FFT/mJ	FFT/mJ norm.	programmable
[4]	pipelined	65	1.10	50	16	21.5	17.60	2641	3196	no
proposed	memory based	28	0.60	450	16	11.4	4.19	20916	3243	yes
[21]	memory based	65	1.20	500	16	2.6	170.0	2287	3292	no
[9]	memory based	130	1.50	250	16	20.6	60.40	802	3609	yes
[6]	memory based	600	3.30	173	20	30.0	845.0	39	6058	no
proposed	memory based	65	1.00	450	16	11.4	12.21	7171	7171	yes
[7]	memory based	90	1.00	160	16	2.6	29.00	13360	18498	no
[5]	pipelined	55	0.90	18	16	1.5	8.88	77131	52865	no

Table 1. Comparison of various FFT processor architectures (1024-point FFT).

ogy and another 65 nm technology.

In order to be able to compare different technologies, the energy was normalized according to the following formula [6], [22]:

$$E_n = E\frac{L_{ref}U_{ref}^2\left(\frac{1}{3}W_{ref}^2 + \frac{2}{3}W_{ref}\right)}{LU^2\left(\frac{1}{3}W^2 + \frac{2}{3}W\right)}, \quad (1)$$

where E_n is the normalized energy; E, L, U and W are parameters of the proposed architecture (energy, technology size, voltage and word length, respectively); the $_{ref}$ suffix marks the reference technology (65 nm, 1.0 V, 16 bits).

Table 1 compares the proposed architecture with selected state-of-the-art solutions and traditional architectures. The chosen focus point is a 1024-point FFT as a mid-point between the smallest and largest supported FFT sizes. The frequency 450 MHz is close to the maximum achievable frequency (500 MHz for 28nm/0.60V and 550 for 65nm/1.00V). The maximum achievable frequency is 1150 MHz with 28nm/1.10V technology.

8. CONCLUSION

In this paper, a low-power software-programmable FFT processor was proposed, which is is based on a TTA template. The key contribution is reducing the computation kernel into only one repeated instruction word and executing it from a loop buffer instead of fetching from an instruction memory every clock cycle. This reduces the power consumption of an instruction memory to a negligible value. In order to achieve the instruction word compression, an internal buffering was introduced in case of a complex adder and a memory access, which renders them unusable by a high level language compiler. However, it is possible to provide a software-accessible switch to disable the memory buffering for more generic applications. Additional functionality can be introduced by adding other functional units. Synthesis power evaluation performed at two different ASIC technologies (28 nm and 65 nm) shows that the processor can provide an energy efficiency comparable with fixed-function ASIC processors.

9. ACKNOWLEDGMENTS

The authors thank the following sources of financial support: Tampere University of Technology Graduate School, Business Finland (FiDiPro Program funding decision 40142/14), and ECSEL JU project FitOptiVis (project number 783162).

10. REFERENCES

[1] M. Khelifi, D. Massicotte, and Y. Savaria, "Parallel independent FFT implementation on intel processors and Xeon phi for LTE and OFDM systems," in *IEEE Nordic Circ. Syst. Conf. & Intl. Symp. SoC*, 2015, pp. 1–4.

[2] X. Lyu, J. Zuo, and H. Xie, "Non-equispaced FFT computation with CUDA and GPU," in *Intl. Conf. on Virtual Reality and Visualization (ICVRV)*, 2016, pp. 227–234.

[3] T. H. Tran, S. Kanagawa, D. P. Nguyen, and Y. Nakashima, "ASIC design of MUL-RED radix-2 pipeline FFT circuit for 802.11ah system," in *Proc. IEEE Low-Power and High-Speed Chips Symp.*, 2016, pp. 1–3.

[4] M. Garrido, R. Andersson, F. Qureshi, and O. Gustafsson, "Multiplierless unity-gain SDF FFTs," *IEEE T. Very Large Scale Integration Syst.*, vol. 24, no. 9, pp. 3003–3007, 2016.

[5] M. Garrido, S. Huang, and S. Chen, "Feedforward FFT hardware architectures based on rotator allocation," *IEEE T. Circ. Syst. I: Regular Papers*, vol. 65, no. 2, pp. 581–592, 2018.

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

[7] S. Huang and S. Chen, "A high-parallelism memory-based FFT processor with high SQNR and novel addressing scheme," in *Proc. IEEE ISCAS*, 2016, pp. 2671–2674.

[8] H. Corporaal, *Microprocessor Architectures: From VLIW to TTA*, John Wiley & Sons, Inc., 1997.

[9] T. Pitkänen and J. Takala, "Low-power application-specific processor for FFT computations," *J. Signal Process. Syst.*, vol. 63, no. 1, pp. 165–176, 2011.

[10] A. Saidi, "Decimation-in-time-frequency FFT algorithm," in *Proc. IEEE ICASSP*, 1994, vol. 3, pp. III–453.

[11] W. H. Chang and T. Q. Nguyen, "On the fixed-point accuracy analysis of FFT algorithms," *IEEE T. Signal Processing*, vol. 56, no. 10, pp. 4673–4682, 2008.

[12] T. Pitkänen, *Fast Fourier Transforms on Energy-Efficient Application-Specific Processors*, Ph.D. thesis, Tampere University of Technology, Finland, 2014.

[13] J. A. Fisher, "Very long instruction word architectures and the ELI-512," in *Proc. of the 10th Annual Intl. Symp. on Computer Arch.*, Stockholm, Sweden, 1983, pp. 140–150.

[14] P. Jääskeläinen, H. Kultala, T. Viitanen, and J. Takala, "Code density and energy efficiency of exposed datapath architectures," *J. Signal Process. Syst.*, vol. 80, no. 1, pp. 49–64, 2015.

[15] T. Pitkänen, J. K. Tanskanen, R. Mäkinen, and J. Takala, "Parallel memory architecture for application-specific instruction-set processors," *J. Signal Process. Syst.*, vol. 57, no. 1, pp. 21–32, 2009.

[16] V. Guzma, T. Pitkänen, and J. Takala, "Reducing instruction memory energy consumption by using instruction buffer and after scheduling analysis," in *Intl. Symp. on SoC Proc.*, 2010, pp. 99–102.

[17] P. Jääskeläinen, T. Viitanen, J. Takala, and H. Berg, "HW/SW co-design toolset for customization of exposed datapath processors," in *Computing Platforms for Software-Defined Radio*, Waqar Hussain, Jari Nurmi, Jouni Isoaho, and Fabio Garzia, Eds., pp. 147–164. Springer, 2017.

[18] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques," in *Proc. Intl. Conf. on Computer-Aided Design*, 2011, pp. 694–701.

[19] T. Pitkänen, R. Mäkinen, J. Heikkinen, T. Partanen, and J. Takala, "Low-power, high-performance TTA processor for 1024-point fast Fourier transform," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, Berlin, Heidelberg, 2006, pp. 227–236, Springer.

[20] T. Pitkänen, T. Partanen, and J. Takala, "Low-power twiddle factor unit for FFT computation," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, Berlin, Heidelberg, 2007, pp. 65–74, Springer.

[21] M. A. Shami, M. A. Tajammul, and A. Hemani, "Configurable FFT processor using dynamically reconfigurable resource arrays," *J. Signal Process. Syst.*, 2018.

[22] Y. Chen, Y. W. Lin, Y. C. Tsao, and C. Y. Lee, "A 2.4-gsample/s DVFS FFT processor for MIMO OFDM communication systems," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 5, pp. 1260–1273, 2008.