An Embedded Implementation of Discrete Zolotarev Transform Using Hardware-Software Codesign

Jan KUBAK, Jakub STASTNY, Pavel SOVKA

Dept. of Circuit Theory, Faculty of Electrical Engineering, Czech Technical University in Prague

jankubak@gmail.com, stastnj1@seznam.cz, sovka@fel.cvut.cz

Submitted October 14, 2020 / Accepted March 19, 2021

Abstract. *The Discrete Zolotarev Transform (DZT) brings an improvement in the field of spectral analysis of non-stationary signals. However, the transformation algorithm called Approximated Discrete Zolotarev Transform (ADZT) suffers from high computational complexity. The Short Time ADZT (STADZT) requires high segment length, 512 samples, and more, while high segment overlap to prevent information loss, 75% at least. The STADZT requirements along with the ADZT algorithm computational complexity result in a rather high computational load.*

The algorithm computational complexity, behavior, and quantization error impacts are analyzed. We present a solution which deals with high computational load employing co-design methods targeting Field Programmable Gate Array (FPGA). The system is able to compute one-shot DZT spectrum 2 048 samples long in ≈ 22 ms. Real-time STADZT spectrum of a mono audio signal of 16 kHz sampling frequency can be computed with overlap of 91%.

Keywords

Discrete Zolotarev Transform (DZT), Approximated Discrete Zolotarev Transform (ADZT), embedded hardware, hardware-software co-design, Field Programmable Gate Array (FPGA), VHDL

1. Introduction

The Discrete Zolotarev Transform (DZT) [1], [2] shows promising results in field of non-stationary signal analysis. The transform employs Zolotarev polynomials [3] in its basis. To evaluation the DZT spectrum Approximated DZT (ADZT) along with Short Time ADZT (STADZT) were developed [1, 2, 4, 5]. The STADZT provides interesting results analyzing different kinds of non-stationary signals, for more details see [2, 5–7]; it has been shown recently that the STADZT performs well detecting faults in rotating machinery, bearings, in comparison with other methods [8]. We wanted to be able to demonstrate STADZT capabilities in real-time and have a powerful tool for application experiments. However, the ADZT algorithm leads to relatively high computational load compared to other spectral analysis methods, e.g. FFT algorithm.

The STADZT improvement becomes apparent for high segment lengths; 512 samples and higher. In general, longer the segment the better [6]. It is recommended to use segment overlap at least 75% segment length, 90% ideally, to prevent information loss [6]. Large segment length, large segment overlap, and structure of the ADZT algorithm leads to significantly high computation load. Therefore, software implementation runtime of the algorithm is relatively long. We decided to design an embedded solution, which both satisfies our needs and enables application in an embedded system. We kept in mind main goals of embedded system: low area requirement and low power consumption, and approached to the system design with co-design methods. The algorithm is implemented on hardware (HW) device with encapsulated processor; main computational stress is loaded on HW units allowing processor to be simple, small and low power. We chose Field Programmable Gate Array (FPGA) as HW platform: the FPGA is widespread nowadays and can be easily integrated into a large System on Chip (SoC). Furthermore, the FPGA power consumption is low especially with non-volatile devices.

This paper presents the first embedded implementation of the ADZT transform. The system requirements are to compute real-time STADZT spectrum of following parameters:

- the input signal is mono audio signal with sampling rate of $f_\mathrm{s} = 16\,\mathrm{kHz}$,
- maximal segment length is $N_\mathrm{DFT} = 2\,048$ samples,
- segment overlap of 80% at least, 90%, if possible.

This paper is organized as follows. Section 2.1 introduces the ADZT algorithm in brief description. Section 2.2 shows the algorithm time complexity analysis followed by the algorithm computational model in Sec. 2.3. Section 2.4 describes the system level design and Section 2.5 the design verification. Section 3 presents the parameters of implemented system and Section 4 concludes the article.

2. Methods

2.1 The ADZT Algorithm

The ADZT algorithm [2], [5] can be divided into three steps, see Fig. 1, of which descriptions follows.

Step 1 The first step is DFT spectrum computation using FFT algorithm of length N_{DFT}. Since the algorithm works only with real signal, it is sufficient to work with one-sided spectrum of length $N = \frac{N_{\mathrm{DFT}}}{2}$.

Step 2 In the second step the ADZT basis selectivity refinement takes place. The algorithm works in spectral domain for each ℓth spectral line, $\ell = \{1, \ldots, N\}$. The DZT spectrum $Z(\ell)$ can be decomposed [1] in stationary part and non-stationary (N-S) part. The stationary part $S(\ell)$ is ℓth DFT spectral line. The N-S part

$$N(\ell, m) = \sum_{\ell'=1}^{m} (-1)^{\ell'} S(\ell - \ell') \tag{1}$$

is parametrized by bandwidth[1] m of approximated polynomial. The transformation selective feature is comprised in the N-S part. The transformation selectivity refinement is administered by varying the non-stationary part using the bandwidth. Optimal bandwidth is estimated according to

$$m_{\mathrm{opt}}(\ell) = \arg\min_{m'} \left\{ \frac{1}{m'} \, N(\ell, m') \, \mathrm{sgn}(S(\ell)) \right\}, \qquad m' = \{1, 2, \ldots, N-1\} \tag{2}$$

where m' is tested bandwidth. The optimal N-S part $N_{\mathrm{opt}}(\ell)$ is acquired from (1) using estimated optimal bandwidth (2).

Step 3 The DZT spectrum is composed from the stationary and the N-S part for each ℓth spectral line, $\ell = \{1, \ldots, N\}$. At first the signs of each part are compared with each other. Based on the result of the comparison one of the following three cases is taken: the first case is to suppress spectral line $Z(\ell) = 0$; the second one is to preserve DFT spectral line $Z(\ell) = S(\ell)$; the third one is to determine a new spectral line value according to

$$Z(\ell) = norm(\ell) \left\{ S(\ell) + k(m_{\mathrm{opt}}(\ell)) N(\ell) \right\} \tag{3}$$

where

$$norm(\ell) = \frac{1}{\sqrt{1 + m_{\mathrm{opt}}(\ell) k^2(m_{\mathrm{opt}}(\ell))}} \tag{4}$$

is the normalization coefficient

$$k(m) = \sqrt{\frac{1 - \sigma^2}{m \sigma^2}} \tag{5}$$

is the non-stationarity index (N-S index) and σ is the degree of the N-S part suppression. The parameter σ has a value in interval $\sigma \subset (0, 1)$; it is chosen according to user needs.

The algorithm works separately for real and imaginary spectrum part; after each part is processed the complex DZT spectrum is recomposed.

[1]The term bandwidth is different from frequency bandwidth.

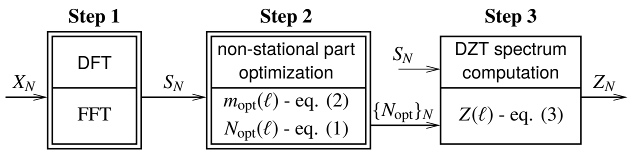

Fig. 1. The ADZT algorithm diagram; blocks with double line border are implemented in HW and with single line in SW.

2.2 Asymptotic Complexity of the ADZT

At first we did an estimation of the computational power necessary to execute ADZT transformation with parameters given by the system requirements. The analysis of the ADZT algorithm asymptotic complexity for each step follows.

Step 1 Computation of the N_{DFT}-point DFT spectrum $S(\ell)$. The DFT spectrum is computed using the FFT algorithm. N_{DFT} is constrained to be the power of 2. Thus the asymptotic complexity of this step is $O_1 = O(N_{\mathrm{DFT}} \log_2 N_{\mathrm{DFT}})$.

Step 2 Estimation of the optimal bandwidth $m_{\mathrm{opt}}(\ell)$ by minimization (2) of the N-S part (1) for one-sided DFT spectrum of length N. The asymptotic complexity of this step is $O_2 = O(N^2)$.

Step 3 The DZT spectrum composition according to (3); this step has constant time complexity of $O_3 = N$.

Hence the asymptotic complexity of the ADZT algorithm is $O_{\mathrm{ADZT}} = O(N^2)$. Step 2 is the most time consuming one. Compare this with FFT algorithm asymptotic complexity of $O_{\mathrm{FFT}} = O(N \log_2 N)$.

Based on the structure of the algorithm we also did a more precise estimation of necessary system-level computation power in MIPS (million instructions per second). The estimation takes into account the system requirements specifications in Sec. 1. We assume a standard Harvard DSP CPU architecture with three buses (instruction, data bus X, and data bus Y), divider with DIV operation, and Multiply And Accumulate (MAC) unit; such that MAC operation unit is able to load two operands simultaneously. Computational powers needed for the algorithm steps are stated in Tab. 1.

Based on the above we can say that real-time STADZT of $N_{\mathrm{DFT}} = 2\,048$ samples with 80% overlap needs a CPU with performance at least $\approx$ 210 MIPS, and perfomance of $\approx$ 420 MIPS for 90% overlap.

Approximate processor requirements are as follows. A 32 bit fixed-point DSP processor with MAC requires clocking frequency of 300 MHz or 600 MHz, for 80% or 90% overlap, respectively. A general purpose processor without a MAC unit requires clocking frequency of 400 MHz or 800 MHz, for 80% or 90% overlap, respectively; the lack of a MAC unit raises power requirements by $\approx$ 30%.

Algorithm step	Operation	Inst. per iteration	Time complexity	Comp. power [MIPS]	
				Overlap (OV): 80 %	OV: 90 %
1(FFT)	complex MULT	4 [2]	$N_{\text{DFT}}/2 \times \log_2(N_{\text{DFT}})$	≈ 2	≈ 4
	complex ADD	2	$N_{\text{DFT}} \times \log_2(N_{\text{DFT}})$	≈ 2	≈ 4
2 (N-S opt)	(1), (2)	5 [3]	N^2	≈ 206	≈ 413
3 (DZT spec.)	(5), (4), (3)	19 + 18 + 3 [4]	N	≈ 2	≈ 3
			Total:	≈ 210	≈ 420

[2] four MAC operations

[3] (1): one MAC operation, (2): one DIV and two MULT operations, arg min search in one operation

[4] (5): three multiplications, one subtraction, one division, and fifteen operations for 32 bit square root using Newton's method; (4): three multiplications, one subtraction, one division, and fifteen operations for 32 bit square root using Newton's method; (3): one addition and two multiplication operations

Tab. 1. The estimation of computational power in MIPS needed for the STADZT real-time computation with segment length of $N_{\text{DFT}} = 2\,048$ samples and sampling frequency of $f_s = 16$ kHz for segment overlaps of 80% and 90%.

2.3 Computational Model

To be able to analyze behaviour of the algorithm and to verify HW implementation effectively and reliably we developed models on several levels of abstraction; each model more detailed, closer to physical implementation than previous one. The most abstract is the first, behavioral, model of the algorithm written in Matlab, which was already available at [9]. The behavioral model in Matlab uses 64 bit wide floating-point arithmetics, which HW implementation is rather inefficient in terms of resources and power consumption. For an efficient HW implementation the fixed-point representation is more suitable. We implemented the algorithm functional model using $N_b = 32$ bit wide fixed-point variables and arithmetics written in C language; the model proper functionality was verified against the original behavioral model. The functional model also allows us to simulate behaviour of the algorithm in fixed-point arithmetics with relatively large set of input stimuli. Such simulation would not be possible in the latter design phase, Register Transfer Level (RTL): it would be too time consuming to run it. The functional algorithm model is able to process signal segment of maximal length $N_{\text{DFT}} = 2\,048$ samples. The descriptions of computation methods used in the functional algorithm model in each algorithm step in Fig. 1 follow.

Step 1 We used Fast Fourier Transform (FFT) decimated in frequency (DIF) as DFT computation algorithm. The FFT uses radix-2 with 32 bit fixed-point arithmetics. As precaution measure to prevent overflows each stage output is divided by 2, see [10]. The FFT has $N_{\text{stages}} = \log_2(N_{\text{DFT}}) = 11$ stages in case of maximal signal segment length; the output is 11 times shifted right. The overflow does not necessarily occur in every stage. We carried out a simulation[5] showing that the average count of overflows N_{OF} is 4 and 7 in the worst case. Thus the 32 bit FFT result has 25 Less Significant Bits (LSB) non-zero ($N_b - N_{\text{stages}} + N_{\text{OF}} = 32 - 11 + 4 = 25$ bits) in average and 28 LSB non-zero in the worst case ($N_b - N_{\text{stages}} + N_{\text{OF}} = 32 - 11 + 7 = 28$ bits).

Step 2–1 The N-S part is computed according to (1). Equation (1) can be rewritten into recurrent form

$$N(\ell, m) = N(\ell, m-1) + (-1)^m\, S(\ell - m) \tag{6}$$

which is more suitable for computation. An overflow can occur $\log_2(m)$ times in the worst case where $m = N - 1$. The DFT coefficients have 6 Most Significant Bits (MSB) in zero on average case. The saturation must be applied after each addition. Non saturated overflown result could lead to false minimal N-S part detection in the next step; this would result in a distorted DZT spectral line value. If a result is saturated, it will be the largest one in tested set of bandwidths. The result will not be, most likely, selected as minimal and its value will be discarded. In a nutshell, saturated N-S part does virtually no harm. Moreover, no overflow had occurred in N-S part computation during conducted simulation using 32 bit wide variables and segment length of $N_{\text{DFT}} = 2\,048$.

Step 2–2 Estimation of optimal bandwidth according to (2); the N-S part, computed in previous step, is weighted by weighting coefficient $\frac{1}{m'}$ and $\text{sgn}(S(\ell))$ for each m'. The minimal value is being selected simultaneously. The weighting coefficient $\frac{1}{m'}$ in (2) for each m' is precomputed in memory in order to multiply by $\frac{1}{m'}$ rather than divide by m'. The weights $\frac{1}{m'}$ are ≤ 1 for each m'; after division by 2 overflow is not an issue here.

Step 3 The DZT spectrum composition according to (3) can be rewritten using (4) and (5) to

$$Z(\ell) = \sigma S(\ell) + K(m_{\text{opt}}(\ell))\, N(\ell) \tag{7}$$

where $K(m) = \sigma\, k(m)$. There are two reasons why the rewritten form is more suitable for fixed-point arithmetic computation. Firstly, by pre-computing coefficient $K(m)$ for each bandwidth the N-S is actually multiplied only ones; thus, reducing round-off error. Secondly, the range of $K(\ell)$ is $(1, 0)$, which allows us to avoid normalization; thus there is no signal to quantization noise ratio drop.

[5]The stimuli of the simulation were composed of few musical tracks of different genre, several minutes of spoken language, and several segments of Gaussian noise.

2.4 System Design

Today's embedded systems are composed of both SW (software) and HW (hardware) components, each one has its pros and cons:

- Fully SW product solution, for instance [11], [12], is easy to develop as well as debug. Risk of a costly bug in the final products is low. Later modifications can be applied easily; software is very flexible. On the other hand, the software runtime is often unnecessarily long and requires considerable resources. Also the power consumption is often high.

- Fully HW solution is dedicated to perform specific task effectively with respect to time, area resources, and power consumption (e.g., [13–15]). However, to implement and verify such a solution is time consuming. There is a considerable risk of a bug in the final product, and even a minor modification can result in a time consuming architecture change. Hardware solution is rather optimized but not very flexible.

SW parts of the system are suited to implement decision-like tasks where data throughput is not an issue often in higher levels of system hierarchy (e.g., control communication protocol, control blocks governing data path blocks). On the other hand HW system parts are best in tasks where many operations with many data are required as fast as possible; data throughput is a key issue (e.g., data path blocks, MAC blocks). We used co-design methods [16], [17] to achieve trade-off between advantages and disadvantages of both SW and HW solutions. Our design goals were speed, simplicity, and possibility of future minor changes.

The first step in the system level design was to draw the boundary between HW and SW part of the system. We started the system design as fully SW based on the functional algorithm model using general-purpose processor; we exploited SW versatility and were able to debug system quickly and effectively. Then we confirmed the computation bottlenecks, which were revealed by the algorithm asymptotic complexity analysis in Sec. 2.2. The first algorithm step, the FFT algorithm, we implemented as HW solution. The FFT algorithm is a well covered topic, see [12, 14, 18]. The second algorithm step is the most time complex one. The step is comprised of addition, multiplication, and minimum search in (2). Thus the step can be easily implemented by HW: addition and multiplication are common operations, and minimum search requires only one decision. The third algorithm step has linear time complexity; this is not an issue from the computational complexity standpoint. Furthermore, the step comprises of several decisions. Hence the third step is implemented by SW. Figure 1 depicts the algorithm steps implemented by HW with double line border and by SW with single line border.

The architecture of the system is outlined in Fig. 2. It is composed of Plasma processor [19], external address decoder, FFT coprocessor, Non-Stationary (N-S) part coprocessor, and interface. The interface unit stands for bridge between the bus and external world. The address decoder selects an external unit and connects its buses to the processor as well as handles simple Direct Memory Access (DMA). We favored 32 bit RISC processor Plasma [19] with MIPS I^{TM} instruction set [20]. The processor has free licence terms, free C language compiler, and it is written in VHDL. The Plasma processor has 8 kB internal memory for program and data. The SW program, which implements the algorithm Step 3 and schedules the execution of both coprocessors, size is $\approx$ 2.8 kB. The data memory is mainly occupied with precomputed values of $K(m)$ for each m in (7): utilizing 4 kB. Thus the processor internal memory size is sufficient for our needs.

The system is separated into two clock domains. The N-S part coprocessor, part of the FFT coprocessor, and the interface are clocked at 100 MHz. The Plasma processor and the radix-2 butterflies within the FFT coprocessor are clocked at 50 MHz. The address decoder serves as a bridge between the two clock domains. The main reason to introduce two clock domains is to speed up the N-S part coprocessor; thus to reduce the Step 2 runtime. The coprocessor accesses the FFT coprocessor memory via DMA during the execution. This requires the address decoder and the FFT coprocessor to be in the same clock domain as the N-S part coprocessor is. However, the butterflies would require additional pipelining to reach the higher clocking frequency. This is not necessary since the FFT computation speed up would not yield a significant runtime reduction of the whole system. The solution is to keep the butterflies in the slower clock region. The resynchronization between the domains is relatively simple. The faster clock domain has exactly twice the slower clock domain frequency. The two clock signals have aligned rising edges. Sufficient adaptation is to lengthen control signals leading from the faster to the slower domain to two clock cycles.

All of the system units has been written in VHDL language.

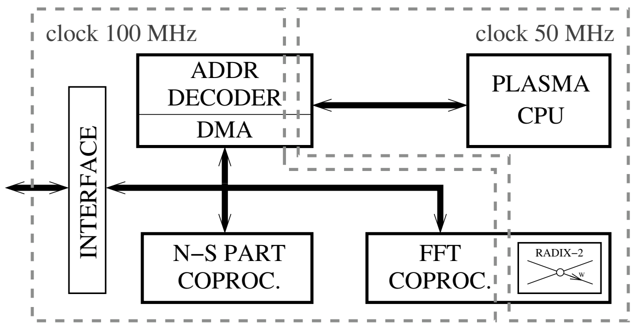

Fig. 2. System top level architecture. Dashed lines depicts distinct clock domains.

2.4.1 FFT Coprocessor

The hardware design methods to implement an FFT coprocessor have been described previously, see [12, 14, 18]. The FFT coprocessor is composed of a pair of radix-2 butterflies, a pair of double port RAM for each butterfly, and dedicated address generator. Twiddle factors are stored in a pair of $2\,048 \times 32$ bit ROM; its size is 16 kB. Each butterfly uses 32×32 bit wide complex multiplier with two stage pipelining. The unit holds complex input samples and output DFT complex spectrum in the memory pair. Real Imaginary Alternate (RIA) format [21] is used. The memory pair has total depth of $2 \times 2\,048 = 4\,096$ depth (2 represents real and imag. part, $N_{\mathrm{DFT}} = 2\,048$); the memory size is 16 kB. The memory is accessible through the system bus, and it is used as data memory for the whole system. The unit total memory requirement is 32 kB. The unit computes one-shot DFT spectrum of length $N_{\mathrm{DFT}} = 2\,048$ in $\approx 5\,700$ clock cycles of the slower clock domain. The FFT coprocessor has been designed as part of master's thesis [22].

2.4.2 Non-stationary Part Coprocessor

The N-S part coprocessor top level schema is in Fig. 3. The unit realizes computation of the N-S part according to (6) and searches for an optimum according to (2) for the current ℓth spectral line. The actual computation is implemented in the N-S core block. The address generator generates addresses of DFT coefficients stored in the FFT coprocessor as

$$S\_\mathrm{addr}_\ell(m') = (\ell - m') \times 2 + \mathit{is\_imag}, \qquad m' = \{0, 1, \ldots, N-1\}\,. \quad (8)$$

Since the algorithm works separately for real and imaginary part, the address is multiplied by 2 in order to select real or imaginary part only (according to used RIA format). The real or imaginary part is computed, while the variable *is_imag* is set to 0 or 1, respectively. The address is routed through the system bus to the FFT coprocessor, which output supplies the N-S core unit with stationary part values. The unit is mounted to the system bus as its master via DMA during the unit execution. Input current spectral ℓth line and variable *is_imag* are stored in the register bank. The register bank also holds results supplied by the N-S core block, optimal bandwidth, and N-S part. Controller block controls access to the register bank and the system bus as well as schedules execution of the other blocks.

Principal schema of the N-S core block architecture is in Fig. 4. The MEM blocks stand for a simple synchronous memory composed of a multiplexor and a synchronous register. SIGN MUX 1, adder, and ALT SUM MEM implement (6); both SIGN MUX and ALT SUM MEM form a MAC unit with multiplication coefficient of $(-1)^{m'}$. The weighs $\frac{1}{m'}$ in (2) are stored in ROM (32 bit wide, N depth: the 4 kB size) and are multiplied with N-S part by $32{\times}32$ bit multiplier. The result of the multiplication is rounded to 32 bit wide signal. SIGN MUX 2 stands for $\mathrm{sgn}(S(\ell))$ in (2). If a provisional minimal value is found, MIN value is updated; the current $m'(\ell)$ and $N(\ell, m')$ are stored in the register bank. Since the unit core works in five pipeline stages, it has its own Finite State Machine (FSM) controller. The unit runtime for $N = 1\,024$ for arbitrary ℓth spectral line is $\approx 1\,038$ clock cycles.

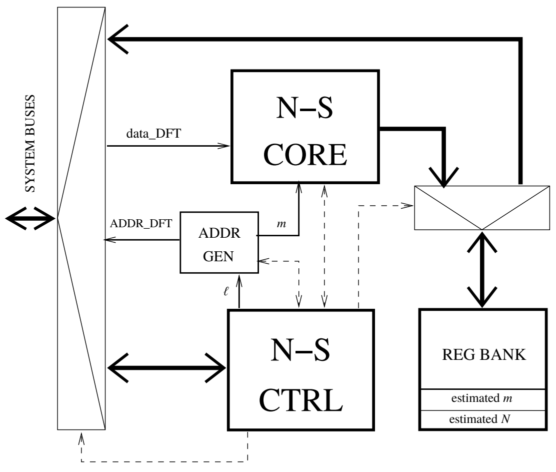

Fig. 3. Non-stationary part coprocessor top level architecture; dashed lines denote control signals, wide lines denote buses. All buses are 32 bit wide. Clock and global asynchronous reset signals are omitted in the schema.

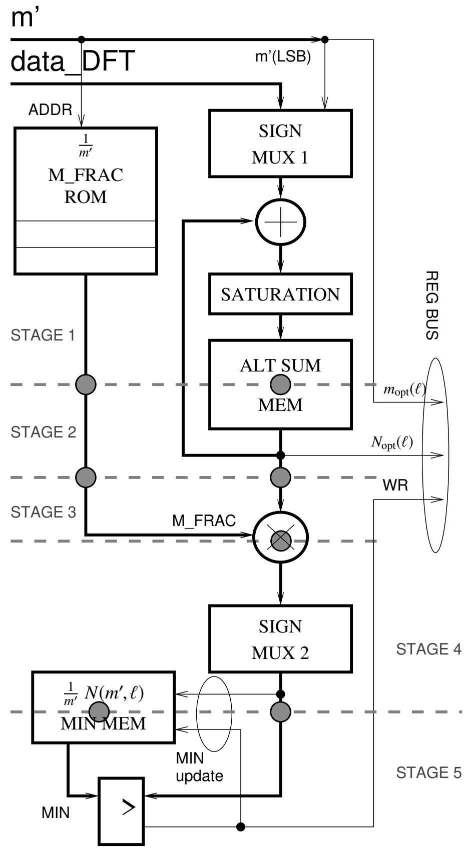

Fig. 4. Non-stationary coprocessor CORE architecture. MEM clock and global asynchronous reset signals are omitted in the schema. The gray dashed lines denote pipeline stages and the gray dots pipeline registers.

Algorithm step	System unit	Runtime [clock cycles]
1	FFT coproc.	≈ 5.7 k
2	N-S coproc.	≈ 539 k
3	Plasma proc.	≈ 318 k
2 & 3 simultaneously	N-S + Plasma + DMA	≈ 570 k

Tab. 2. Algorithm cumulative runtime (the algorithm steps 2 and 3 are executed N times for arbitrary ℓ) for segment length $N_{\mathrm{DFT}} = 2\,048$; clock cycles values are given with respect to the 50 MHz clock domain.

2.4.3 HW&SW Scheduling

This feature can accelerate the algorithm runtime. The runtime for each of algorithm step pertaining to particular system unit is stated in Tab. 2. Runtime of the FFT coprocessor (Step 1) is negligible compared with the other ones. Runtime of the N-S coprocessor and the algorithm SW part in processor (Steps 2 and 3) are comparable. Moreover, both algorithm steps are executed N-times for an arbitrary ℓ. Therefore, it is suitable to exploit HW&SW parallelism in Steps 2 and 3. Scheduling scheme is to execute the N-S coprocessor to pre-compute $(l+1)$th spectral line during the SW part computes lth one. Using this scheduling the total runtime depends virtually on HW N-S coprocessor runtime.

2.5 Verification of The Design

To be certain of proper functionality of the design has been achieved the verification of the design is mandatory. The verification plan goal on the system level is to validate the system output DZT spectral lines series against the computational model output. The verification uses coverage-driven constrained random-based approach [23]. The functional coverage requirement of the plan is to coverage all three cases in Step 3 of the ADZT algorithm, see Sec. 2.1.

Since the DZT is indented for spectral analysis of non-stationary signals, the test data must have non-stationary character. The test data were 40 segments of length N_{DFT} samples consisting of music of different genders (pop music, metal, classical music), vocal recordings, and few segments of Gaussian noise. The test data were selected as a subset of the computational model simulation test data set in Sec. 2.3. The system level verification environment is depicted in Fig. 5. Under Test (DUT) is encapsulated in the verification environment written in behavioral VHDL code. The environment generates system clock, initializes global reset, and stimulates the DUT with test data. The test data are loaded in Matlab and introduced to the DUT by file I/O interface. The computational model serves as behavioral model, and it is executed in Matlab. The DUT output is compared in Matlab. The system was validated on both RTL and gate levels. The unit level verification was performed for each coprocessor using similar scheme as system level verification. Each coprocessor was verified using its own verification environment and behavioral model on both RTL and gate levels. Real system HW realization was validated on development board [24] using parallel interface [25] as system interface.

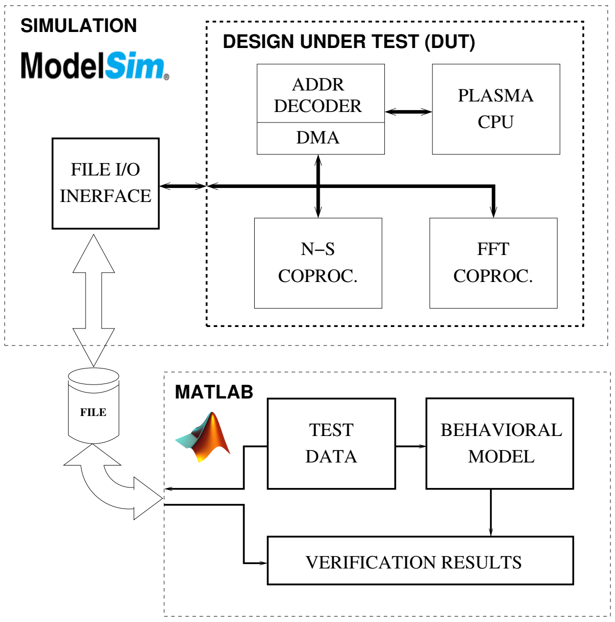

Fig. 5. The system level verification schema of the system.

3. Results

3.1 Physical and Functional Parameters

The system error performance has been measured on the computational model against the original floating-point algorithm version. The measurement was performed as a part of the simulation in Sec. 2.3 with the same test data set. The performance is quantified using Signal to Noise Ratio (SNR)

$$SNR_{\mathrm{dB}}(n) = 10\ \log_{10}\left(\frac{X(n)^2}{\{X(n) - X_{\mathrm{DUT}}(n)\}^2}\right) \tag{9}$$

and mean SNR

$$SNR_{\mathrm{mean}} = 10\ \log_{10}\left(\frac{1}{N}\sum_{n=1}^{N}\frac{X(n)^2}{\{X(n) - X_{\mathrm{DUT}}(n)\}^2}\right) \tag{10}$$

where $X(n)$ is the floating-point model output value, and $X_{\mathrm{DUT}}(n)$ is tested system output value computed using 32 bit fix point arithmetics. The fixed-point version of the algorithm adds quantization noise equivalent to $SNR_{\mathrm{mean}} = 160\,\mathrm{dB}$ and $\min(SNR) = 58\,\mathrm{dB}$. None of computed bandwidths was selected incorrectly, which means that the fixed-point algorithm always selected correct bandwidth and non-stationary part according to (2).

The system runtime for one-shot spectrum computation has been already measured in Sec. 2.4.3 to support HW&SW scheduling. The total runtime for segment length of $N_{\mathrm{DFT}} = 2\,048$ is 570 k clock cycles of the 50 MHz clock domain, see Tab. 2; thus, approximately 11.4 ms, see Sec. 3.2. The system is capable of computing the STADZT results in real-time with $\approx$ 91% overlap considering given segment length and signal sampling frequency, see Sec. 1. Note that fully SW solutions running on the Plasma processor has runtime of $\approx$ 400 ms, 20^6 clock cycles. The required computational power for fully SW solution for given STADZT parameters estimation is 470 MIPS. Employed Plasma processor computational power is 50 MIPS of general purpose instruction set: maximum clocking frequency is 50 MHz [19]. The difference in computational power of $\approx$ 420 MIPS is overcome by HW parallelism.

3.2 Implementation Results

We chose Xilinx Spartan 6 FPGA [26], since we had several development boards [24] with it at our disposal. The design implemented for maximum segment length of $N_{\mathrm{DFT}} = 2\,048$ on targeted device XC6SLX45-3 utilizes 1 164 ($\approx$2%) of the device slice registers and 5 492 (20%) of the device LUTs.

The FFT coprocessor memory utilizes 8 blocks of $1\,024 \times 32$ bit dual-port block RAMs: twiddle factors ROM is implemented as a block RAM. The Plasma registers and RAM are utilized as distributed memory, and both are comprised in the device LUTs utilization. The N-S coprocessor memory is implemented as one $1\,024 \times 32$ bit block RAM. Hence the design utilizes 22 ($\approx$18%) of the device RAMB16BWER block RAMs. The design total memory utilization is 44 kB, of which 4 kB pertain to the N-S part coprocessor.

DSP48A1 blocks are used as 32×32 bit multipliers in both coprocessors. 32×32 bit multiplier is assembled from four DSP48A1 blocks: each block has one 18×18 bit multiplier [27]. The FFT coprocessor employs 24 DSP48A1 blocks: two radix-2 butterflies, where each one has one complex multiplier composed of three real multipliers [27]. The N-S coprocessor has only one real multiplier. Hence the total number of used DSP48A1s blocks rises up to 28 (48% the device utilization).

The design maximal clocking frequency is constrained by its critical path. In our case the critical path occurs in the Plasma processor taking 19.059 ns; this allows us to set clocking frequency up to $f_{\mathrm{CLK}} = 50$ MHz. The critical path leads from processor controller through bus multiplexor to RAM.

4. Conclusion

In this article we present the first embedded implementation of novel selective transform. The ADZT algorithm asymptotic time complexity of $O(N^2)$, which is relatively high compared with the FFT algorithm, was revealed by its analysis. We dealt with such high time complexity using the co-design methods. The analysis of the transformation algorithm supports the partitioning of SW and HW solutions. Dedicated HW parts bear main computational stress, while the SW part carries the less demanding calculations and algorithm decisions. The solution is powerful enough to compute STADZT spectrum of mono audio signal in real-time assuming sampling frequency of 16 kHz, segment length of 2 048 samples and overlap of 91%. Computed real-time STADZT spectrum does not suffer from excessive information loss. Light and low-power processor with 50 MIPS of computational power is employed as a heart of the system; the whole solution computational power is equivalent to 240 MIPS processor with Harvard DSP CPU architecture and MAC unit. The system memory utilization is 44 kB from which the N-S part coprocessor utilizes 4 kB. The platform can be further adapted to support a simple application exploiting ADZT spectrum; the application can be implemented by SW on the processor with relatively low additional design efforts. The design targets FPGA device; therefore, it can be relatively easily integrated into a larger system on chip.

Acknowledgments

This work is supported by the grant GACR 102-11-1795: *Novel selective transforms for non-stationary signal processing* and also by the Student Grant Contest (SGS) grant Advanced Algorithms of Digital Signal Processing and their Applications, grant number SGS14/191/OHK3/3T/13.

References

[1] JANIK, J., TURON, V., SOVKA, P., et al. A way to a new multi-spectral transform. In *Recent Advances in Signal Processing, Computational Geometry and Systems Theory (ISCGAV'11, ISTASC'11)*. Florence (Italy), 2011, p. 177–182. ISBN: 9781618040268

[2] TURON, V., JANIK, J., SPETIK, R., et al. Study of ADZT properties for spectral analysis. In *Recent Advances in Signal Processing, Computational Geometry and Systems Theory (ISCGAV'11, ISTASC'11)*. Florence (Italy), 2011, p. 171–176. ISBN: 9781618040268

[3] VLCEK, M., UNBEHAUEN, R. Zolotarev polynomials and optimal FIR filters. *IEEE Transactions on Signal Processing*, 1999, vol. 47, no. 3, p. 717–730. DOI: 10.1109/78.747778

[4] MASA, P., SOVKA, P., VLCEK, M., et al. Using ADZT for a signal reconstruction. In *Proceedings of the European Conference on Circuit Theory and Design (ECCTD)*. Dresden (Germany), 2013, p. 1–4. DOI: 10.1109/ECCTD.2013.6662335

[5] TURON, V. *Description of Spectral Analysis Based on Zolotarev Polynomials* (in Czech). Ph.D. dissertation, Czech Technical University in Prague, Faculty of Electrical Engineering, 2016.

[6] TURON, V. A study of parameters setting of the STADZT. *Acta Polytechnica*, 2012, vol. 52, no. 5, p. 106–111. DOI: 10.14311/1654

[7] TURON, V., JANIK, J., SPETIK, R., et al. Comparison of two spectral methods for acoustic signal analysis (in Czech). *Akustické listy*, 2011, vol. 17, no. 4, p. 26–30. ISSN 1212-4702

[8] BAJPEYEE, B., SHARMA, S. Detection of bearing faults in induction motors using short time approximate discrete Zolotarev transform. In *Proceedings of the International Conference on Signal Processing (ICSP)*. Chengdu (China), 2016, p. 1–7. DOI: 10.1049/cp.2016.1467

[9] *Novel Selective Transforms For Non-Stationary Signal Processing*. Available at: http://amber.feld.cvut.cz/selectivetransforms

[10] KUO, S. M., LEE, B. H., TIAN, W. *Real-Time Digital Signal Processing*. John Wiley & Sons, Ltd., 2001. ISBN: 9781118414323

[11] CUPAIUOLO, T., LO IACONO, D. A flexible and fast software implementation of the FFT on the BPE platform. In *Design, Automation Test in Europe Conference Exhibition (DATE)*. Dresden (Germany), 2012, p. 1467–1470. DOI: 10.1109/DATE.2012.6176598

[12] SHETTI, K., KOH, C., AUNG, M., et al. Development and code partitioning in a software configurable processor. In *IEEE Region 10 Conference (TENCON)*. Singapore, 2009, p. 1–5. DOI: 10.1109/TENCON.2009.5396149

[13] BAAS, B. A low-power, high-performance, 1024-point FFT processor. *IEEE Journal of Solid-State Circuits*, 1999, vol. 34, no. 3, p. 380–387. DOI: 10.1109/4.748190

[14] CHENG, C., PARHI, K. High-throughput VLSI architecture for FFT computation. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2007, vol. 54, no. 10, p. 863–867. DOI: 10.1109/TCSII.2007.901635

[15] HE, S., TORKELSON, M. Design and implementation of a 1024-point pipeline FFT processor. In *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC)*. Santa Clara (USA), 1998, p. 131–134. DOI: 10.1109/CICC.1998.694922

[16] TEICH, J. Hardware/software codesign: The past, the present, and predicting the future. *Proceedings of the IEEE*, 2012, vol. 100, no. Special Centennial Issue, p. 1411–1430. DOI: 10.1109/JPROC.2011.2182009

[17] TSAI, T.-H., YANG, Y.-C., LIU, C.-N. A hardware/software codesign of MP3 audio decoder. *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, 2005, vol. 41, no. 1, p. 111–127. DOI: 10.1007/s11265-005-6254-2

[18] GENTLEMAN, W. M., SANDE, G. Fast Fourier transforms: For fun and profit. In *Proceedings of the AFIPS Fall Joint Computer Conference*. New York (NY, USA), 1966, p. 563–578. DOI: 10.1145/1464291.1464352

[19] RHOADS, S. *Plasma - Most MIPS I(TM) Opcodes*. 2016. Available at: http://opencores.org/project,plasma,overview

[20] MIPS TECHNOLOGIES. *MIPS32 Architecture For Programmers Volume II: The MIPS32 Instruction Set*. 2001. Available at: http://www.mips.com/

[21] CHANG, Y.-N., PARHI, K. High-performance digit-serial complex-number multiplier-accumulator. In *Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors (ICCD)*. Austin (TX, USA), 1998, p. 211–213. DOI: 10.1109/ICCD.1998.727050

[22] PALASCAK, J. *FFT Core Implementation* (in Czech). Master's thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, 2010.

[23] BERGERON, J. *Writing Testbenches: Functional Verification of HDL Models.* 2nd ed. Norwell (MA, USA): Kluwer Academic Publishers, 2003. ISBN: 1402074018

[24] DIGILENT. *AtlysTM Board Reference Manual*. 2013. Available at: https://www.xilinx.com/support/documentation/university/XUP%20Boards/XUPAtlys/documentation/Atlys_rm.pdf

[25] DIGILENT. *Digilent Parallel Interface Model Reference Manual*. 2004. Available at: https://reference.digilentinc.com/_media/reference/software/adept/adept-2/dpimref_programmers_manual.pdf

[26] XILINX. *Spartan-6 Family Overview*. 2011. Available at: http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf

[27] XILINX. *Spartan-6 FPGA DSP48A1 Slice*. 2014. Available at: www.xilinx.com/support/documentation/user_guides/ug389.pdf

About the Authors . . .

Jan KUBÁK received the M.S. degree in Electrical Engineering from the Faculty of Electrical Engineering of the Czech Technical University (FEE CTU), Prague, in 2013. He is currently studying doctoral degree at the Department of Circuit Theory, FEE CTU. He is engaged in research on selective spectral transforms. He is FPGA digital circuit designer. His interests include radar signal processing research and implementation.

Jakub ŠŤASTNÝ was born in Prague, the Czech Republic in 1978. He received M.Sc. degree in Electrical Engineering from the Faculty of Electrical Engineering of the Czech Technical University, Prague in 2002; and received Ph.D. degree in 2006. His current research interests include digital design architectures for signal processing.

Pavel SOVKA received the M.S. and Ph.D. degrees in Electrical Engineering from the Faculty of Electrical Engineering of the Czech Technical University (FEE CTU), Prague, in 1981 and 1986, respectively. From 1985 to 1991 he worked in the Institute of Radioengineering and Electronics of the Czech Academy of Sciences, Prague. In 1991 he joined the Department of Circuit Theory, FEE CTU. He worked on the application of adaptive systems to noise and echo cancellation, speech analysis, changepoint detection, and signal separation. Presently he is engaged in research on biomedical signal processing and selective spectral transforms.