Customizing a VLIW-SIMD Application-Specific Instruction-Set Processor for Hearing Aid Devices

Julian Hartig, Lukas Gerlach, Guillermo Payá-Vayá, and Holger Blume
Cluster of Excellence Hearing4all, Institute of Microelectronic Systems
Leibniz Universität Hannover, Appelstr. 4, 30167 Hannover, Germany
Email: {hartig, gerlach, guipava, blume}@ims.uni-hannover.de

***Abstract*—Hardware architectures for modern hearing aid devices have to provide ultra low power consumption at a small silicon area and moderate computational performance to deal with the continuously growing complexity of hearing aid signal processing. At the same time, they need to remain flexible for future algorithmic changes. These challenging design goals can be achieved by using Application-Specific Instruction-Set Processors (ASIPs), where a baseline architecture is customized to the target class of applications. In this paper, hardware modifications of a generic VLIW-SIMD processor architecture targeting audio processing are described and their influence in area-performance efficiency and power are evaluated. As exemplary hearing aid signal processing application, the evaluated algorithms contain a complex modulated filter bank and a noise reduction algorithm. The proposed architecture requires 2 times less silicon area and a 6 times lower clock frequency than a Tensilica Xtensa LX4 when running the same algorithms under real-time conditions.**

I. Introduction

Over 360 million people worldwide suffer from disabling hearing loss, i.e., a hearing loss greater than $40\,\mathrm{dB}$ in the better hearing ear, which is about $5\,\%$ of the world's population [1]. An even greater number of people are affected by mild or moderate hearing loss. Thus, many people could potentially benefit from more advanced hearing aid devices. In digital state-of-the-art devices, the embedded processing systems have to meet a variety of stringent design goals:

- *Moderate performance* to meet real-time processing, while at the same time being able to perform complex signal processing (e.g., adaptive beamforming, feedback cancellation, noise reduction, or wide dynamic range compression).
- *Flexibility* by means of programmability to take care of short innovation cycles, research progress on audio signal processing, and to adapt the device individually to the hearing characteristic of the user.
- *Low processing delay* (less than 10 ms) between the input and the processed audio signal required for a proper audio perception [2].
- *Ultra low power consumption* for at least 50 hours of run-time on a single battery. As hearing aid devices should be very small for cosmetic reasons, this leaves limited space for silicon area and battery [3].

The high number of people, whose life quality can be improved by better devices, and the broad technical design space, make hearing aid hardware systems a challenging field of research. Dedicated hardware solutions, although having advantages in energy and area efficiency, do not provide that degree of flexibility required by hearing aid systems. Hence, DSP-like solutions are necessary, representing the state-of-the-art. However, it is essential to reduce their power consumption, e.g., by using efficient data parallelism mechanisms such as vectorized execution (SIMD). Approaching from a DSP-like point-of-view, this paper follows the concept of *Application-Specific Instruction-Set Processors* (ASIPs). A baseline RISC or VLIW processor architecture is customized towards a target application by performing efficient custom hardware modifications. On the one hand, these hardware modifications consist of adding new dedicated hardware modules, e.g., functional units (FUs). Hence, computation intensive parts of the application code are optimized and the efficiency is increased. On the other hand, removing unused parts of the processor can be a successful way to reduce unnecessary power consumption. Concerning performance and power, a specialized ASIP solution lies between pure DSP and dedicated hardware solutions in the design space, while remaining programmable.

In the RAPANUI project [4], a generic VLIW-SIMD ASIP for multimedia applications has been presented and architectural design alternatives have been analyzed. The proposed processor architecture presents several generic parallel mechanisms for full customization towards a specific target application in terms of performance, silicon area, and power consumption. Not only inserting new custom operations (e.g., new functional or co-processor units) is possible but also deeper modifications inside the architecture (e.g., modification of pipeline, register file, or forwarding fabric). In [3] and [5], it has been shown, that the concept of an ASIP for hearing aid devices is promising, because it fits the desired constraints of both flexibility and low power consumption at moderate performance. This paper describes and evaluates several hardware mechanisms of this processor in the context of audio processing.

The paper is organized as follows. First, Section II presents processor approaches related to this work. Then, Section III describes the explored VLIW-SIMD ASIP architecture, which is evaluated in a case study in Section IV in the context of hearing aid applications using an exemplary hearing aid signal processing chain. Finally, conclusions are given in Section V.

II. Related Work

In [5], an ASIP implementation of a digital hearing aid system based on the Tensilica/Cadence Xtensa LX4 processor is presented. This processor consists of a 32-bit RISC architecture with configurable instruction/data cache size and number

of instructions executed in parallel (Flexible Length Instruction eXtension - FLIX). Moreover, the Xtensa framework provides the ability to add custom instructions to the instruction-set by using Tensilica Instruction Extensions (TIE). However, the tools have limitations when enhancing the provided LX4 processor. Deep modifications in the pipeline structure (e.g., extreme reduction of pipeline stages) or register file architecture (e.g., modifying the number of ports of the general purpose register file) and enhancements in the forwarding path are not possible. It is shown, that the most efficient LX4 configuration for running a complex modulated filter bank [6] and a noise reduction algorithm [7] requires several TIEs and no FLIX. Its size is $0.623\,\mathrm{mm}^2$ (TSMC40LP) and it requires $13.24\,\mathrm{MHz}$ for real-time computation while consuming about $2\,\mathrm{mW}$.

The authors of [3] present a modified Silicon Hive Pearl 16-bit 3-issue-slot VLIW-ASIP architecture with a 32-bit datapath, 40-bit registers for intermediate results, separated instruction and data memory, fixed-point arithmetic units, and custom instructions. The performed algorithms are beamforming, feedback cancellation, an FIR filter bank, compression, and noise reduction. After voltage and frequency scaling the processor needs only $0.964\,\mathrm{mW}$ power at a silicon area of $0.49\,\mathrm{mm}^2$ (TSMC C65G) and a clock frequency of $11\,\mathrm{MHz}$.

Moreover, in [8] a low power fixed-point DSP is proposed, that contains a 16/32-bit data-path and twelve concurrent 16-bit multipliers. The processor is optimized for FFT computation using up to 128-bit wide-SIMD instructions, optimized instruction scheduling and bit-reversed addressing for enhanced FFT processing. The performed algorithms are an FFT-based filter bank, compression, and feedback cancellation. For these algorithms, the DSP needs less than 40% of the specified real-time processing time at a clock frequency of $8\,\mathrm{MHz}$.

Comparing to the above mentioned approaches, this paper presents a custom VLIW-SIMD ASIP for audio processing, which implements enhanced parallelism mechanisms and up to two co-processors for processing a complex modulated filter bank [6] and a noise reduction algorithm [7].

III. VLIW-SIMD ASIP Processor

In order to fulfill the design goals of ultra low power, moderate performance, and high flexibility, a generic multimedia ASIP architecture [4] has been used and optimized towards an exemplary hearing aid application. Beginning with this baseline architecture, ASIP customization is driven by the two aspects of *parallelization* and *specialization* resulting in an area-performance trade-off, which is essential in signal processing ASIPs.

A. Parallelization Mechanisms

Since multimedia and signal processing applications provide a high degree of inherent parallelism within the algorithms, the baseline architecture extensively implements instruction-level (VLIW) and data-level (SIMD) techniques. It is based on a **dual-issue vector unit**, that can decode and execute two 32-bit instructions in parallel (see Fig. 1). An enhanced instruction scheduler distributes the application assembler code into the different concurrent issue-slots to achieve a high code compaction, i.e., few execution cycles and thus a high execution performance.

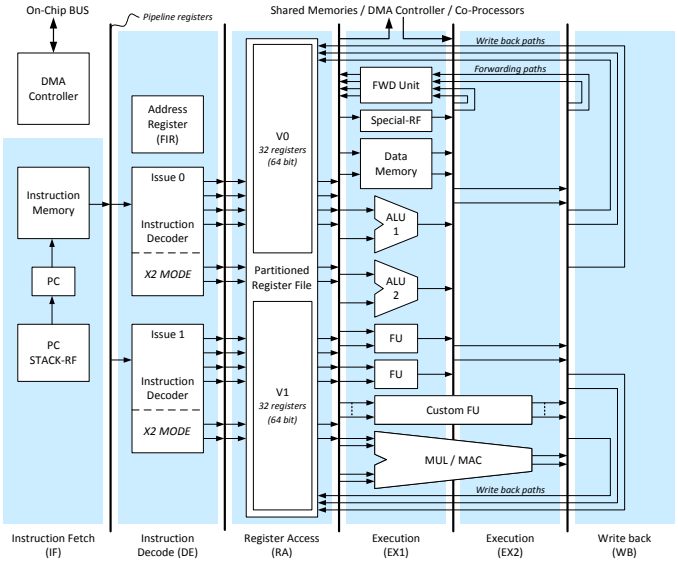

Fig. 1. Structure of a vector unit of the VLIW-SIMD processor [4].

```
1 for (i=0; i<8; i++){
2   if (V0R0[i*8+7:i*8] == 0){
3     V0R3[i*8+7:i*8] = V0R1[i*8+7:i*8] + V0R2[i*8+7:i*8];
4     }
5 }
```

```
1 SMVI V0CONDSEL,#COND_ZERO // choose evaluated condition
2 SUBICS_8 V0R4,V0R0,#0  // store operation flags (CS)
3 ADDCR_8 V0R3,V0R1,V0R2 // cond. execution using flags
4                        // and condition implicitly (CR)
```

Fig. 2. Assembler example of subword parallelism and conditional execution. The addition is only executed on those 8-bit subwords with a valid condition.

The processor is pipelined into six basic stages. Since in most cases the critical path lies in the execution stage, complex functional units (FUs) can be implemented with two pipeline stages for performance purpose. For more **instruction level parallelism**, either identical FUs and/or the entire vector unit could be replicated if the application is able to utilize this degree of parallelization.

All FUs implement 64-bit SIMD instructions making use of **subword parallelism**, i.e., the 64-bit operands can be divided into two 32-bit, four 16-bit, or eight 8-bit operands to execute the same operation in all these subwords simultaneously. In addition, conditional instruction execution is used to transform control flow dependencies into data dependencies. This leads to higher code compaction as slow conditional branches are avoided (see Fig. 2).

Both issue-slots in one vector unit share a flexible register file (RF). Since in many cases in VLIW architectures the register file ports are the bottleneck prohibiting a dense code compaction, the register file can be partitioned into two small register files with 32 registers each (see in Table I all available register file configurations). The register file configuration also has a large influence on the performance of one mechanism special to this VLIW-architecture, which is the **X2-mode** [4]. It allows identical instructions that share the same opcode to merge and access registers, which only differ in their last address bit (see in Fig. 3 the implementation of the X2-mode register file). Hence, more issue-slots are virtually created since one X2-instruction only needs one issue-slot although it

TABLE I. AVAILABLE REGISTER FILE CONFIGURATIONS. AREA RESULTS FROM TSMC40LP SYNTHESIS OPTIMIZED FOR AREA AND POWER (APPROX. 50 MHZ MAXIMAL CLOCK FREQUENCY).

#	Port config.	Port width	#Parallel X1-accesses	#Parallel X2-accesses	Norm. area	Scheduling flexibility
RF1	4r2w	64	2	1	1.00	normal
RF2	4r2w	128	2	2	1.06	high
RF3	2r1w	128	1	1	0.76	low

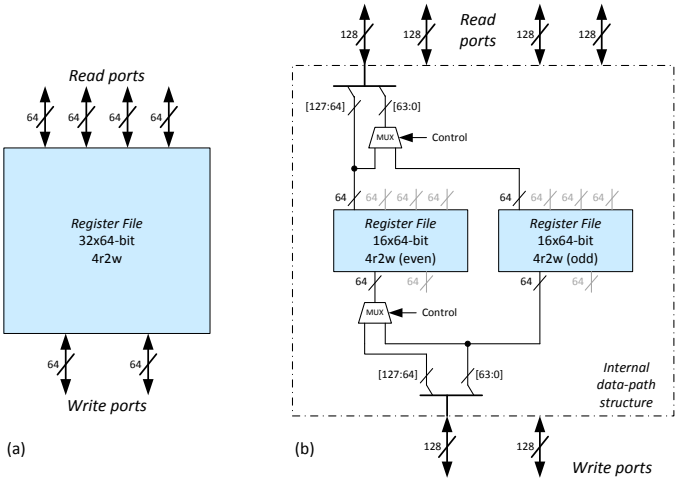

Fig. 3. Examples of RF configurations for architectures that implement X2-mode operations. In (a) a basic RF with 4 read and 2 write ports is shown. In (b), the data read/write port width is doubled to 128-bits. Two internal 16x64-bit RF are used, hence single 64-bit accesses can be performed to all the positions of both RFs by using the 64 LSB of each port. Aligned (even positions) 128-bit accesses can also be performed [9].

performs the instruction twice (assuming physically duplicated FUs). Since still only one instruction is decoded, only slight changes in the decode stage and the register file structure are necessary to enable X2-mode execution with very low overhead (see X2-mode code example in Fig. 4).

B. Specialization Mechanisms

In order to further enhance the processing performance, silicon area, and/or energy efficiency of an ASIP architecture, specialization is mandatory. This is performed by either adding new or by removing unused hardware and functionality for the target applications.

This VLIW-SIMD architecture not only provides the possibility to replicate but also to remove existing FUs within the execution stage or unused SIMD-modes depending on the requirements of the application. Naturally, modifying the instruction-set by adding new **custom FUs** is another proper way of specialization.

For even more complex operations, **co-processors** can be instantiated. By default, the VLIW-SIMD architecture contains a 2x32bit SIMD fixed-point division co-processor unit (DCU), that implements an iterative non-restoring algorithm in dedicated hardware. After storing the divisor operand, the processor starts the division increasing the precision of the result in every cycle, i.e., iteration. Thus, the desired minimal precision can be controlled by the user-defined number of cycles between storing the last operand and loading the result (see Fig. 5).

Another aspect is the specialization of the complete processor, e.g., choosing one of the above mentioned register file

```
1 //    Issue-slot #0           ; Issue-slot #1
2 (0x00) ADD_16 V0R0,V0R2,V0R4  ; NOP
3 (0x01) ADD_16 V0R1,V0R2,V0R5  ; NOP
4 (0x02) ADD_16 V0R6,V0R8,V0R10 ; NOP
5 (0x03) ADD_16 V0R7,V0R9,V0R10 ; NOP
```

```
1 (0x00) ADD_16 V0R0,V0R2,V0R4  ; ADD_16 V0R1,V0R2,V0R5
2 (0x01) ADD_16 V0R6,V0R8,V0R10 ; ADD_16 V0R7,V0R9,V0R10
```

```
1 (0x00) ADD_16_X2 V0R0+V0R1,V0R2,V0R4+V0R5  ; NOP
2 (0x01) ADD_16_X2 V0R6+V0R7,V0R8+V0R9,V0R10 ; NOP
```

Fig. 4. Scheduled assembler example using X2-mode. Four additions with just one physical arithmetic unit (top), replication of arithmetic unit (middle) and using X2-mode (bottom). Note the two emerging empty issue-slots (NOPs) when using X2-mode. Those can be used by the scheduler for remaining operations, that do not use the arithmetic unit e.g. LOAD/STORE-operations.

```
1 //store dividend
2 STOREDCU 0x200,V0R0,#0
3 //store divisor / start
4 STOREDCU 0x201,V0R1,#4
5 //delayed load of result
6 LOADDCU  V0R2,0x200
```

(a) Coded assembler operations

```
1 (0x00) STOREDCU 0x200,V0R0,#0
2 (0x01) STOREDCU 0x201,V0R1,#4
3 (0x02) NOP
4 (0x03) NOP
5 (0x04) NOP
6 (0x05) LOADDCU  V0R2,0x200
```

(b) Scheduled assembler operations

Fig. 5. Exemplary use of division co-processor unit (DCU). After storing both operands, the DCU starts operating. The user-defined delay (#4) determining the DCU iterations is automatically considered during instruction scheduling. In this example, one DCU iteration is performed in each clock cycle.

architectures or the best pipeline configuration. The register file ports can be reduced for reducing the silicon area requirements in cases where a higher degree of flexibility is not needed by the instruction scheduler to achieve a high performance (see Table I). When a high clock frequency is necessary, it is possible to increase the number of execution stages to reduce the critical path. In the case of a low power design such as a hearing aid system, the clock frequency is low, so removing pipeline stages and saving the area and power required for pipeline registers is beneficial.

In addition, the processor system can be configured with different peripheries and interfaces. It contains separated instruction and data memory of configurable size and a DMA controller for reading from an audio buffer and communication with an external memory.

C. Customization Trade-off

The customization of the processor with respect to ultra low power and hearing aid systems can be summarized in an area-performance trade-off shown in Fig. 6. Due to continuous audio sampling, the system has to meet a real-time constraint. A higher performance in this context means being able to execute *equal* software tasks in *less time* or to run with a *lower minimal clock frequency* for less dynamic power consumption. The minimal frequency F_{min} is defined as

$$F_{min,Processor} = \frac{\#Cycles_{Audioblock}}{\#Samples_{Audioblock}} \cdot F_{Sampling}. \quad (1)$$

Applying X2-mode and changing the register file or the pipeline influences the baseline architecture slightly, whereas implementing new functional units and co-processors results in large changes in area, execution cycles and thus performance. As an efficiency measure, the *inverse product of area and minimal clock frequency* can be considered. During ASIP customization this product needs to be minimized, which results in an overall reduction of total power consumption.

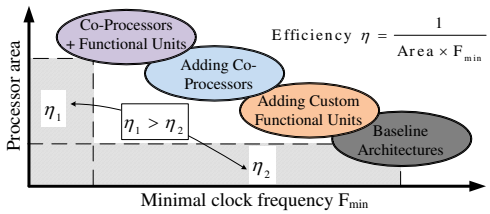

Fig. 6. During ASIP customization, additional area (higher static power) is traded against less execution cycles and less necessary minimal clock frequency for real-time processing (less dynamic power). The product of area and F_{min} needs to be minimized and can be considered as inverse efficiency.

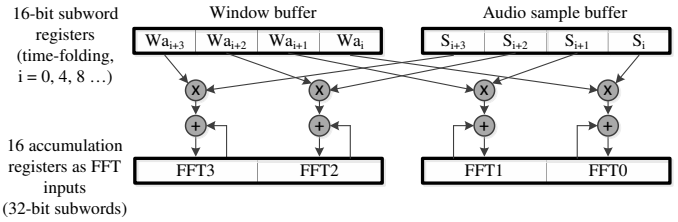

Fig. 7. One single MAC_16 SIMD instruction is used for window multiplication and time-folding in WOLA analysis before FFT. WOLA analysis (synthesis) requires $La/4$ ($Ls/4$) MAC_16 operations.

IV. Case Study

The above presented VLIW-SIMD architecture has already been evaluated in the context of video processing [9]. By using its X2/X4 instruction merging mechanisms (code up to four operations into one instruction) and custom functional units it became possible to process a stereo video algorithm under real-time conditions, resulting in very high computational efficiency. The same baseline architecture is explored in this paper in a case study of a typical hearing aid processing chain, i.e., a filter bank and a noise reduction algorithm using fixed-point arithmetic and a target maximal clock frequency of 10 MHz.

A. Filter Bank

1) Algorithm: To apply separate gain values in different frequency bands, a complex modulated weighted overlap-add (WOLA) filter bank [5-6] performs the split into subbands here. The WOLA algorithm is an extension of the short-time Fourier transform, containing multiplication with window functions and an FFT/IFFT. By performing a time-folding step, the analysis and synthesis filter sizes La and Ls can be adjusted independently to the FFT size N.

2) Implementation: For operating with a low processing delay (6.5 ms), the selected parameters are La(128), Ls(64), and N(32). The filter bank processes blocks of audio samples with size R(8), which results in an oversampling factor of 4. The implementation is performed for 16-bit audio samples and using a target sampling frequency of 16 kHz. Analysis window multiplication and time-folding are implemented with 16-bit SIMD MAC-operations, that store their real-valued accumulation results into registers partitioned into two 32-bit subwords ($[Re|Re]$, see Fig. 7). The circular shift, necessary in WOLA, is done inside the FFT. The FFT continues working on these 32-bit subwords storing the final complex-valued results in a $[Re|Im]$ subword format. The synthesis again performs 16-bit operations, which makes shifts necessary after IFFT and synthesis window multiplication.

For WOLA analysis and synthesis, circular buffers are required to store new 8-sample blocks and iterate over memory. Since checking the end of the buffer and wrapping the address pointer around at the highest buffer address is inefficient when using assembler instructions, the following circular buffer mechanism is executed in hardware whenever a pointer is incremented in software:

$$addr_{new} = (addr_{prev} \wedge mask) \vee (\text{inc}(addr_{prev}) \wedge \overline{mask}) \quad (2)$$

TABLE II. Evaluation of WOLA filter bank implementations.

Processor Configuration	Cycles	Dyn. IPC	F_{min} [MHz]
baseline (maxPipe_RF1_X1_1F)	898	1.57	1.80
perf_optimized (minPipe_RF2_X2_2F)	551	2.58	1.10
baseline + CMU	562	1.64	1.12
perf_optimized + CMU	341	2.53	0.68

A mask splits the address into an upper and a lower part. The new content of the address pointer after incrementation is a concatenation of the upper part (mask=1) of the unincremented address and the lower part (mask=0) of the incremented address. The size of the buffer is determined by the number of bits in the lower part of the mask. The address of the first buffer position has to be aligned accordingly.

To accelerate the FFT butterfly computation, a complex multiplication unit (*CMU*) has been implemented for instruction-set extension. The CMU can perform the following operation in just one cycle due to the low clock frequency:

$$\begin{aligned} C &= [A_{Re} \cdot B_{Re;x} - A_{Im} \cdot B_{Im;x} | A_{Re} \cdot B_{Im;x} + A_{Im} \cdot B_{Re;x}] \\ &= [A_{Re}|A_{Im}] \times [B_{Re;high}|B_{Im;high}|B_{Re;low}|B_{Im;low}] \end{aligned} \quad (3)$$

The complex operand A and the result C both have 32-bit subwords, whereas the FFT twiddle factors B are 16-bit subwords. It can be selected to take either B_{high} or B_{low} into account. This allows loading two complex twiddle factors with a single LOAD-instruction.

Further, the above mentioned processor specialization features are applied systematically to evaluate their influence on the execution performance and the processor area (see below). This includes replication of all FUs (*1F* vs. *2F*) for more parallel resources, removing all pipeline stages up to a last stage RA/EX to ensure synchronous register file access (*maxPipe* vs. *minPipe*), which accelerates the execution of MAC- and branch-operations, and using X2-instructions (*X1* vs. *X2*) with the register file modifications indicated in Table I (*RF1*, *RF2*, *RF3*). Exemplary software modifications are given in Fig. 8 and 9 showing the performance-increasing ability of the X2-mode in the WOLA implementation.

A WOLA performance evaluation for a baseline and performance enhanced processor configuration, both with and without CMU, is summarized in Table II. The number of execution cycles, the dynamic instructions per cycle (IPC) and the necessary minimal clock frequency (F_{min}) for real-time processing are presented. Note, that the use of X2-mode allows to execute up to four instructions in parallel resulting in an IPC above two on the dual-issue VLIW architecture.

```
1 MV      V0R0,(WaAPtr)+
2 MV      V0R2,(SmplAPtr)+
3 MAC_16  V0R4+V0R5,V0R0,V0R2
4
5 MV      V0R1,(WaAPtr)+
6 MV      V0R3,(SmplAPtr)+
7 MAC_16  V0R6+V0R7,V0R1,V0R3
```

(a) Loading data in normal mode

```
1 MV_X2   V0R0+V0R1,(WaAPtr)++
2 MV_X2   V0R2+V0R3,(SmplAPtr)++
3
4 MAC_16  V0R4+V0R5,V0R0,V0R2
5 MAC_16  V0R6+V0R7,V0R1,V0R3
6
7
```

(b) Loading data in X2-mode

Fig. 8. Loading data using indirect memory access (address pointers) and X2-mode. By using MV_X2 two 64-bit registers are loaded in one instruction. MAC- instructions have two destination registers by default.

```
1 ADD_32    V1R0,V0R0,V0R2 // [FFT1+FFT17 | FFT0+FFT16]
2 MIXRI_32  V0R4,V1R0,#0   // [FFT0+FFT16 | 0]
3 MIXLI_32  V0R5,V1R0,#0   // [FFT1+FFT17 | 0]
4
5 SUB_32    V1R2,V0R0,V0R2 // [FFT1-FFT17 | FFT0-FFT16]
6 MIXRI_32  V0R6,V1R2,#0   // [FFT0-FFT16 | 0]
7 MIXLI_32  V0R7,V1R2,#0   // [FFT1-FFT17 | 0]
8
9 ADD_32    V1R1,V0R1,V0R3 // [FFT3+FFT19 | FFT2+FFT18]
10 MIXRI_32 V0R8,V1R1,#0   // [FFT2+FFT18 | 0]
11 MIXLI_32 V0R9,V1R1,#0   // [FFT3+FFT19 | 0]
12
13 SUB_32   V1R3,V0R1,V0R3 // [FFT3-FFT19 | FFT2-FFT18]
14 MIXRI_32 V0R10,V1R3,#0  // [FFT2-FFT18 | 0]
15 MIXLI_32 V0R11,V1R3,#0  // [FFT3-FFT19 | 0]
```

```
1 ADD_32_X2     V1R0+V1R1,V0R0+V0R1,V0R2+V0R3
2 MIXRLI_32_X2  V0R4+V0R5,V1R0,#0
3 MIXRLI_32_X2  V0R8+V0R9,V1R1,#0
4
5 SUB_32_X2     V1R2+V1R3,V0R0+V0R1,V0R2+V0R3
6 MIXRLI_32_X2  V0R6+V0R7,V1R2,#0
7 MIXRLI_32_X2  V0R10+V0R11,V1R3,#0
```

Fig. 9. Due to SIMD operations, two stage-1 FFT butterflies of the 32-point FFT can be compactly processed *2-in-1* using 32-bit subwords (top). With X2-mode, even a dense *4-in-1* execution becomes possible when choosing the right register allocation (bottom). MIX-operations are typical in SIMD processors to reorder subwords of two input registers. The MIXRL_X2 can merge a right- and a left-oriented MIX, named MIXR and MIXL respectively.

B. Noise Reduction

1) Algorithm: In this case study, the Adaptive Gain Equalizer (AGE) is used for noise reduction [7]. More precisely, the AGE performs speech enhancement by estimating signal activity in different subbands k and attenuating those bands, when there is no activity at the current time instant.

The attenuation gain factors for every subband k are calculated as:

$$g_k(n) = \min\left(1, \left(\frac{A_k(n)}{L \cdot \underline{A}_k(n)}\right)\right) \tag{4}$$

$$A_k(n) = (1-\alpha_k)\cdot A_k(n-1) + \alpha_k \cdot |x_k(n)| \tag{5}$$

$$\underline{A}_k(n) = \begin{cases} (1+\beta_k)\cdot \underline{A}_k(n-1) & \text{if } A_k(n) > \underline{A}_k(n-1) \\ A_k(n) & \text{if } A_k(n) \leq \underline{A}_k(n-1) \end{cases} \tag{6}$$

$A_k(n)$ is a short term average, $\underline{A}_k(n)$ is an estimation of the slowly changing noise floor level and L a normalization factor. By normalization, the maximal gain is limited to 1 (i.e., $g_k(n) \leq 1$). Note, that $A_k(n) \geq \underline{A}_k(n)$ and $\alpha_k >> \beta_k$, so fixed-point formats need to be selected carefully. Further, a division and a square root for the absolute value of complex subband signal $|x_k(n)|$ are necessary in this algorithm.

2) Implementation: Due to symmetry of the spectrum, only $N/2 + 1 = 17$ subband gains need to be computed. The division is performed by the above mentioned co-processor

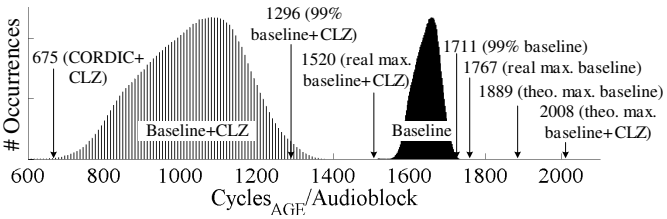

Fig. 10. Distribution of AGE execution cycles per 8-sample audio block after processing the NOIZEUS audio data base [11].

with customized fixed-point format adjustments. For computing the absolute of a complex value, the square of the 32-bit complex input is stored as intermediate result, which is then passed to an integer square root software routine. Since an iterative algorithm, as e.g., Newton's method, in general does not terminate after a predetermined number of iterations, which is unwanted in a real-time system, this implementation uses an algorithm with predictable run-time according to [10], whose maximal number of iterations depends on the size of the input.

To accelerate the square root computation in a first approach, a custom instruction is used to determine the range of the input value by counting the leading zeros (*CLZ*) in hardware compared to a while-based software implementation using only the baseline instruction-set. Since the actual number of iterations varies significantly with these leading zeros, the distribution of execution cycles while processing plenty of blocks of 8 audio samples (17 AGE subband gains in each block) has been investigated using a baseline processor with and without the CLZ-instruction (see Fig. 10). The theoretical maximal number of cycles is never required, which corresponds to maximal input values in all subbands all the time. In order to save power consumption and still meet real-time, the processor can be clocked with a much lower minimal frequency than the theoretical maximum would indicate, e.g., according to the real maximal number of cycles. This is a conservative estimate, since a further reduction of the clock frequency (e.g., according to the *99%-cycles*) is possible. In this case, a hardware mechanism needs to ensure reasonable results before filter bank synthesis in the remaining 1% of the cases, when the real-time condition is not met. This might have a negligible effect on the audio result and warrants further investigation.

Next to baseline architecture with and without CLZ-instructions, a third approach is evaluated here, which is square root calculation using a radix-4 CORDIC co-processor [12]. Between pre- and post-processing for adjusting the input into convergence (also making use of CLZ-instructions) and the output fixed-point format, the CORDIC provides greater precision within fewer cycles than the iterative algorithm. The same delay concept as already presented in Fig. 5 can be used here for trading faster execution against reduced result precision. Since CORDIC run-time is determined by this delayed loading, the required cycles are data-independent.

The resulting AGE performance evaluation is summarized in Table III using three alternatives for square root (baseline, with CLZ-instruction and with CORDIC co-processor). For conservative minimal frequencies, the real maximal numbers of cycles from Fig. 10 at baseline processor configurations for AGE execution are assumed. The use of X2-mode does not provide any performance enhancement in AGE, because the algorithm is heterogeneous.

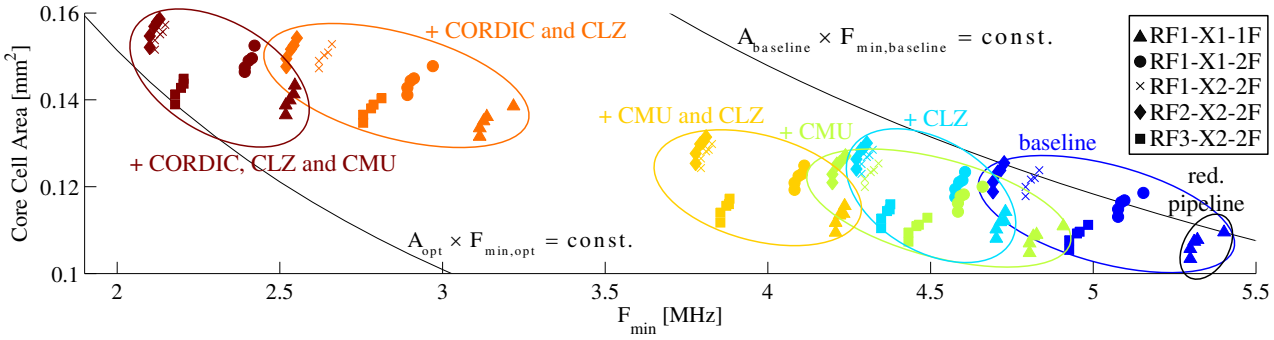

Fig. 11. Area-performance design space of VLIW-SIMD ASIP processor configurations showing lines of constant efficiency for baseline and optimal configuration (40nm TSMC LP, for configuration nomenclature see Section IV).

TABLE III. EVALUATION OF AGE IMPLEMENTATIONS USING THE ACTUAL MAXIMALLY REQUIRED CYCLES (99%-CYCLES IN BRACKETS)

Processor Configuration	**Cycles**	**Dyn. IPC**	F_{min} **[MHz]**
baseline (maxPipe_RF1_X1_1F)	1767 (1711)	1.05 (0.99)	3.53 (3.42)
baseline + CLZ	1520 (1296)	1.25 (1.27)	3.04 (2.59)
CORDIC + CLZ (for pre-processing)	675	1.34	1.35

C. Area and Power Analysis

A 40 nm TSMC low power technology has been used for synthesis (10 MHz clock frequency, optimization for minimal silicon area and dynamic power). The resulting maximal clock frequency is approx. 50 MHz. Core cell areas of different processor configurations at their minimal clock frequencies to process WOLA, AGE and gain multiplication (about additional 0.06 MHz) in real-time are shown in Fig. 11.

Concerning parallelization, duplicating the processor FUs and enabling the X2-mode increases the area on the one hand, but on the other hand algorithm execution is accelerated and the minimal clock frequency can be reduced by a higher factor. Thus, these processor modifications achieve a smaller area-performance product, i.e., a higher efficiency. Concerning specialization, reducing the pipeline decreases the area (less flip-flops) and it also lowers the minimal clock frequency because branch- and MAC-instructions are executed faster. Further, different RF configurations, custom FUs and the CORDIC co-processor increase the area but always accompanied by a large reduction in execution cycles and clock frequency, thus those configurations also have a higher efficiency.

That configuration combining CORDIC, CMU, X2, RF3 and a minimal pipeline has the best efficiency, i.e., minimal product of core area ($0.14\,\mathrm{mm}^2$) and frequency (2.18 MHz). As this product is much smaller than for the baseline configuration ($0.11\,\mathrm{mm}^2$ at 5.40 MHz), an overall gain in power can be achieved. The synthesis tool estimates a total core power consumption of the baseline processor of 0.24 mW and 0.18 mW for the configuration with highest efficiency. So the reduced number of execution cycles and the drop in frequency (lower dynamic power) compensates the increase in area for the additional hardware (higher static power). The power results are estimations of the synthesis tool based on statistic estimation of switching activity.

Comparing the optimal configuration from Fig. 11 with a customized Xtensa LX4 [5] for equal algorithms and including memory, our ASIP is smaller by factor 2 in total cell area and the minimal required frequency is about 6 times lower.

V. CONCLUSION

This paper evaluates a generic VLIW-SIMD ASIP architecture and several subsequent hardware architecture enhancements for processing hearing aid algorithms. It is worth mentioning, that by minimizing the area-performance product, the total power consumption can be reduced, because utilizing additional hardware is accompanied by an even greater decrease in processing cycles and minimal clock frequency for real-time execution. In addition, the presented architecture is 2 times smaller at a 6 times lower clock frequency than a Tensilica Xtensa LX4 processing the same algorithms.

REFERENCES

[1] World Health Organization. (2014) Deafness and Hearing Loss. [Online]. Available: http://www.who.int

[2] H. Puder, "Hearing Aids: an Overview of the State-of-the-Art, Challenges, and Future Trends of an Interesting Audio Signal Processing Application," in *Image and Signal Processing and Analysis (ISPA), Proceedings of 6th Int. Symposium on*. IEEE, 2009, pp. 1–6.

[3] P. Qiao, H. Corporaal, and M. Lindwer, "A 0.964 mW Digital Hearing Aid System," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2011, pp. 1–4.

[4] G. Payá-Vayá, "Design and Analysis of a Generic VLIW Processor for Multimedia Applications," Ph.D. dissertation, Institute of Microelectronic Systems, Leibniz Universität Hannover, 2011.

[5] N. Werner, G. Payá-Vayá, and H. Blume, "Case Study: Using the Xtensa LX4 Configurable Processor for Hearing Aid Applications," in *Proceedings of the ICT.OPEN*, 2013.

[6] ON Semiconductor. (2009) WOLA Filterbank Coprocessor: Introductory Concepts and Techniques. [Online]. Available: http://www.onsemi.com/pub/Collateral/AND8382-D.PDF

[7] N. Westerlund, M. Dahl, and I. Claesson, "Speech Enhancement for Personal Communication Using an Adaptive Gain Equalizer," *Signal Processing*, vol. 85, no. 6, pp. 1089–1101, 2005.

[8] Y. Ku *et al.*, "A High Performance Hearing Aid System with Fully Programmable Ultra Low Power DSP," in *Consumer Electronics (ICCE), Int. Conf. on*. IEEE, 2013, pp. 352–353.

[9] G. Payá-Vayá *et al.*, "VLIW Architecture Optimization for an Efficient Computation of Stereoscopic Video Applications," in *Green Circuits and Systems (ICGCS), Int. Conf. on*. IEEE, 2010, pp. 457–462.

[10] J. W. Crenshaw, "Integer Square Roots," *Embedded Systems Programming*, no. 2, p. 130, 1998.

[11] Y. Hu and P. C. Loizou, "Subjective Comparison and Evaluation of Speech Enhancement Algorithms," *Speech communication*, vol. 49, no. 7, pp. 588–601, 2007.

[12] S. Nolting, G. Payá-Vayá, I. Schmädecke, and H. Blume, "Evaluation of a Generic Radix-4 CORDIC Coprocessor Tightly Coupled with a Generic VLIW-SIMD ASIP Architecture," in *Proceedings of the ICT.OPEN*, 2012.