# Fast and Parallel AAC Decoder Architecture for a Digital Radio Mondiale 30 Receiver

Mohammed Shaaban Ibraheem, Khalil Hachicha, Olivier Romain

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 21, 2017, accepted July 19, 2017, date of publication July 25, 2017, date of current version August 14, 2017.

*Digital Object Identifier 10.1109/ACCESS.2017.2731902*

# Fast and Parallel AAC Decoder Architecture for a Digital Radio Mondiale 30 Receiver

**MOHAMMED SHAABAN IBRAHEEM[1], KHALIL HACHICHA[1], AND OLIVIER ROMAIN[2]**
[1]LIP6 Laboratory, University Pierre and Marie CURIE, 75005 Paris, France
[2]Information Processing and Systems Laboratory, ETIS Laboratory, Cergy-Pontoise University, 95014 Cergy-Pontoise, France

Corresponding author: Khalil Hachicha (khalil.hachicha@upmc.fr)

This work was supported by Agence nationale de la recherche under Grant ARPEGE-SEGI 17.

**ABSTRACT** An embedded real-time indexing engine for live radio stations requires a high-speed parallel advanced audio coding (AAC) decoder architecture to decode hundreds of compressed audio streams broadcast in the digital radio mondiale band. Several AAC hardware core units must be integrated into a single chip and synchronized with a global controller to achieve such high performance. This paper proposes a new parallel AAC decoder architecture to address this challenge. The proposed architecture includes multiple AAC decoder core units, each of which achieves a speed-up by a factor of 2 compared with the existing AAC decoder core units while using optimal logic resources. The proposed architecture overcomes several challenges faced by existing architectures. The Huffman decoder module decodes one word per clock cycle regardless of word length, and a smaller lookup table size is achieved for the inverse quantization module. The inverse modified discrete cosine transform architecture is fully pipelined, and the resource sharing technique is used to reduce the logic area. An initial prototype is implemented on an FPGA platform.

**INDEX TERMS** DRM, MPEG-4, AAC, FPGA, Huffman decoder, filter banks, IMDCT.

## I. INTRODUCTION

Broadcast radio in Digital Audio Broadcasting (DAB and T-DMB) and Digital Radio Mondiale [1], [2] (DRM30 and DRM+) are promising radios that contain various data, such as music, news, sports, weather services, traffic information, commentary, drama, advertising and an increasing amount of metadata to help identify and organize content.

Broadcasting information will substantially enrich the diversity of available content and also open the way to index the content either via existing metadata tags or using original features extracted from the radio signals themselves. Just as it took web browsers to transform the Internet from a simple computer network into a searchable worldwide database, new types of navigators capable of searching and organizing the multimedia streams now available on broadcast radio will be developed. Unfortunately, the conflicting needs of systems exploiting these streams have slowed the development of new products in this emerging market. Today, software-defined radio techniques based on FPGA devices allow hardware designers to imagine the power of such media monitoring systems on a single chip. This advancement opens the door to a wealth of exciting new consumer electronics applications.

DRM30 is a broadcasting radio system that replaces the classical analog radio system. DRM30 introduces high-quality audio combined with rich information about the broadcasted program. Processing this type of data requires a new generation of radio receivers. Ideally, the DRM30 receiver should be able to demodulate more than 1,000 streams simultaneously to monitor all the broadcasted programs. Therefore, a new radio receiver architecture called SurfOnHertz [3] has been proposed. This architecture is an FPGA-based platform designed to implement a multi-standard multi-tuner (AM, FM, DRM and T-DMB), capturing and indexing in parallel all channels having the same frequency band. This architecture will serve as a bridge between search techniques currently deployed in software, and the new media standards opportunities. The SurfOnHertz radio receiver prototype consists of four modules, as shown in Fig. 1.

The SurfOnHertz radio receiver consists of an antenna, analog front end, demodulator, and indexing engine. The analog front end passes the received frequency modulation (FM) signal to an analog-to-digital converter (ADC). The N streamed digital signals, each of which contains a

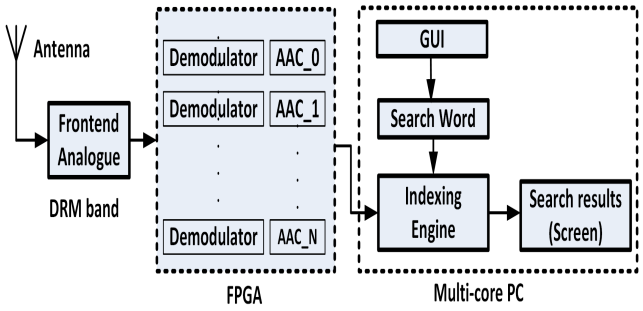


**FIGURE 1.** SurfOnHertz digital radio receiver prototype.

different radio station, are demodulated and uncompressed. The output is passed to the audio indexing module needed for speech processing, music type detection and the identification functions required by the streams' search engine. The input search and its results can be accessed via a graphical user interface (GUI) module.

Broadcasting these streams often creates a transmission bottleneck, which is why the DRM standard introduced data compression to transfer large amounts of data with less bandwidth. The Moving Picture Experts Group, MPEG, AAC [4] is the most widely used standard for audio compression. It has a high audio quality that is suitable for limited-bandwidth channels and was thus adopted in the DRM standard.

The literature offers several designs of AAC decoders. The designed architectures can decode only a limited number of compressed audio streams at the same time. The audio indexing engine in the SurfOnHertz project requires hundreds of streams to be decoded in parallel to satisfy the real-time constraint. This key criterion is mandatory to ensure a highly interactive graphical interface (GUI). This GUI allows the end user to look for a specific content extracted from the radio channels. New challenges emerge when designing a high-speed AAC decoder architecture for parallel audio stream decompression. In this work, a new highly optimized system-level architecture is proposed to address this challenge. The new design allows for the simultaneous and fast decompression of the received DRM's channels. An FPGA solution is adopted to combine the advantages of design flexibility and high performance.

The remainder of this paper is organized as follows. Section II reviews related work. Section III presents an overview of the AAC decoder algorithm. The proposed decoder architecture is detailed in Section IV, and the obtained results and prototype are presented in Section V. A discussion and concluding remarks are presented in Section VI.

## II. RELATED WORK

AAC decoders have been implemented and designed using many approaches in the past decade. Software (SW) decoders are used most often because of their high flexibility and short development time [5], [6]. A major development focus has been to accelerate SW decoders. Reduced instruction set computing (RISC) microprocessors have been widely used for this purpose. Takamizawa *et al*. [7] developed a SW AAC decoder for an embedded RISC, but it lacks hardware utilization and non-optimum scheduling. Ryu *et al.* [8] introduced the digital signal processor (DSP) as an alternative to the RISC approach. Mesarovic *et al*. [9] presented an optimized and cost-effective implementation of the decoder on a dual DSP.

Tsai and Chen [10] used a co-design approach. The bitstream parser was performed by software, and the higher-complexity portion was solved by hardware. With this approach, over 91.26% of the processor-based loading was substituted by hardware IP. Tao *et al.* [11] introduced another co-design solution verified on Xilinx XUP FPGA. The Huffman and inverse modified discrete cosine transform (IMDCT) hardware accelerator modules were developed to accelerate the decoding process.

The application-specific integrated circuit (ASIC) approach is considered the best solution in terms of power consumption and area costs. Zhang*et al*. [12] implemented a low-power AAC decoder on ASIC, with a power consumption of 20 mW. Another ASIC implementation presented by Tsai and Liu [13] consumes considerably less power (2.45 mW). An ASIC MPEG-4 targeting low-power systems was presented by Liu *et al.* [14], and a very-low-power audio decoder compatible with DAB/DAB+ was presented by Wang *et al.*, with a power consumption of only 10.4 mW [15].

The FPGA is considered an alternative hardware approach to ASIC. Although it consumes more power, FPGA-based platforms are more flexible than ASIC-based platforms. Renner and Susin [16] presented a full implementation of an AAC decoder used for Brazilian Digital Television. The designed architecture is capable of decoding a low-complexity audio signal in real time with a clock frequency of 4 MHz. Another FPGA-based implementation was introduced by Sampaio *et al.* [17]. The design was implemented on an Altera Cyclone II FPGA with NIOS II/s as a processor and was able to decode up to six audio channels.

Previously developed designs focused their approaches mainly on *(i)* satisfying the real-time constraint while decoding one compressed audio stream and *(ii)* minimizing the power consumption by performing an ASIC implementation. The specifications and needs of the SurfOnHertz project are different. The embedded audio engine indexes hundreds of radio stations in parallel, which requires the simultaneous decoding of incoming audio streams. The following sections will address this requirement.

## III. AAC DECODER ALGORITHM OVERVIEW

Fig. 2 shows the AAC decoding system. AAC has many advantages and is widely used in many applications [18]. It supports up to 84 audio channels and has a wide range of sampling rates, ranging from 8 kHz to 96 kHz. AAC also has three different profiles: main, low complexity (LC) and scalable sample rate (SSR).

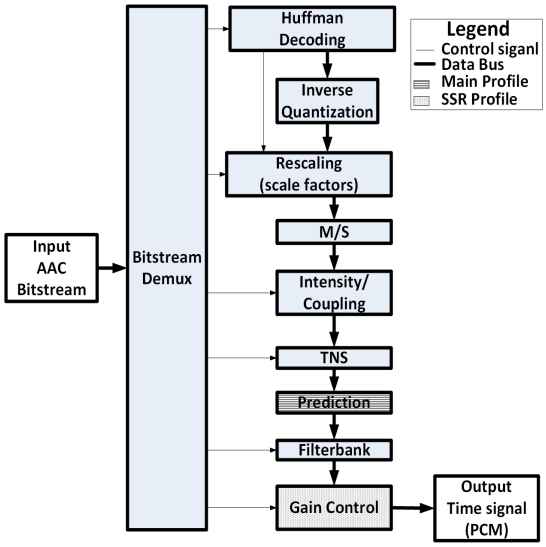


**FIGURE 2.** Schematic of the MPEG-2 AAC decoder.

The audio quality depends on the decoding profile type. The main profile has the highest audio quality, whereas the LC profile delivers a primary profile-like audio quality. The LC profile requires few memory and processing tool resources, making it a more suitable choice for applications requiring short decoding times. The AAC decoding system [4] consists of a set of tools, as shown in Fig. 2. First, the bitstream de-multiplexer tool extracts side information and control signals that are required by the other tools. It also extracts the coded quantized spectral data that represent the audio samples.

The Huffman decoder tool applies noiseless decoding to spectral and scale factor data using 12 Huffman codebooks. Each codebook contains code words, their lengths, and their corresponding spectral coefficients.

The inverse quantization tool takes the quantized spectral data and converts the integer values to non-scaled data, passes them to the rescaling tool and multiplies them by their relevant scale factor data. The Mid-Side (M/S) tool and Coupling/Intensity tool are used for stereo audio streams to improve the coding efficiency. The temporal noise shape (TNS) tool filters the coding noise. The prediction tool inserts redundancies, which are removed during the encoding process.

The filter bank tool converts the scaled inverted spectral data to the time domain by applying the IMDCT. The time samples are in pulse coded modulation (PCM) format.

The proposed AAC decoder architecture includes only the essential required tools to run an LC profile to achieve a smaller area and shorter decoding time. It also supports the audio data transport stream (ADTS) format, whereas decoders in previous work supported the audio data interchange format (ADIF). The main difference between the two formats is that the ADIF stream has only one header, whereas the ADTS stream consists of multiple frames, each with its own header, which is a useful way to simplify radio broadcasting.

## IV. PROPOSED DECODER ARCHITECTURE

This section is divided into two parts. First, the proposed structure of the full AAC decoder, which can process N streams using only N/2 core units, is presented. Next, the architecture of the elementary AAC core units is discussed in detail.

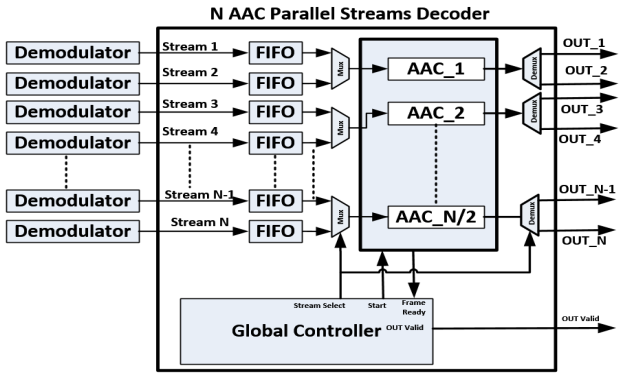


**FIGURE 3.** Proposed architecture of the AAC decoder for parallel N streams.

### A. AAC ARCHITECTURE FOR PARALLEL DECOMPRESSION

The global architecture of the proposed parallel decoder is shown in Fig. 3. It consists of N/2 elementary AAC decoder units, stream buffers, and a global controller. The controller is responsible for multiplexing and de-multiplexing incoming audio streams.

The data flow diagram, shown in Fig. 4, presents a decoding example of 4 parallel streams. The decoder starts by buffering the incoming frames for each stream in a corresponding first-in, first-out (FIFO) scheme. Assuming that the incoming four streams are $Sn = \{S1, S2, S3, S4\}$, each stream has its frames $Sn\_fm = \{Sn\_f0, Sn\_f1, Sn\_f2, Sn\_f3, \ldots\}$. In that case, only two decoders are required to process the four streams. Because of the high-speed of the single AAC core unit, two frames are processed in a time-multiplexed manner.

### B. AAC CORE UNIT ARCHITECTURE

The proposed AAC core unit is divided into four main modules, as shown in Fig. 5: the data de-multiplexer and Huffman (DEMUX_HUFF) module, the inverse quantization/rescaling (IQ_RESC) module, and the filter bank module. The last module is the PCM converter. The finite state machine (FSM) of the local controller is shown in Fig. 6.

### C. DATA DE-MULTIPLEXER AND HUFFMAN MODULE

The *DEMUX_HUFF* block architecture is shown in Fig. 7. It receives the input bitstream and parses it to extract the side information. Then, it decodes the scale factors and

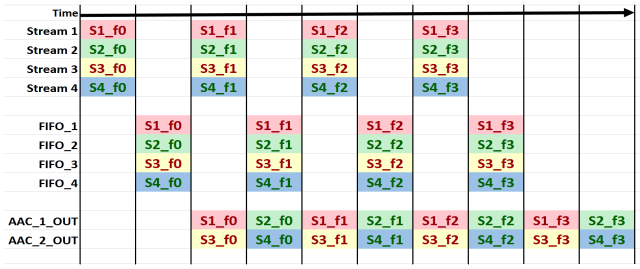


**FIGURE 4.** Example of how the proposed architecture processes parallel streams.

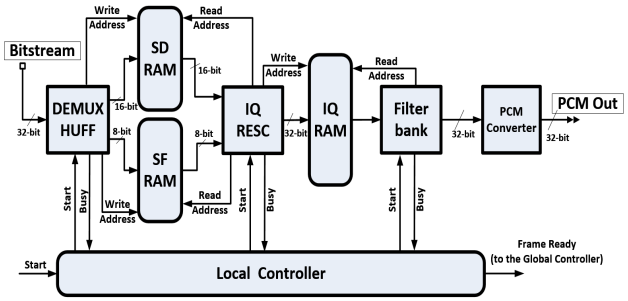


**FIGURE 5.** AAC Core unit top-level architecture.

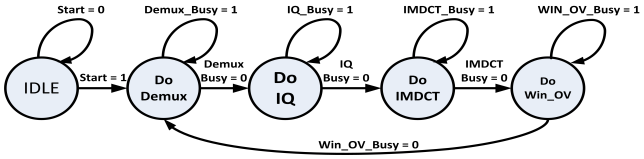


**FIGURE 6.** Central controller FSM.

audio spectral data using the Huffman decoder. The decoded scale factor data are written to the scale factor (*SF-RAM*), whereas the decoded audio samples are written to the spectral data (*SD-RAM*). The key novelty of this design arises from the fact that we combine bitstream parsing and the decoding operations in the same module to save area and optimize the central controller.

### 1) BITSTREAM PARSING OPERATION

This operation is a bit-level processing operation using a barrel shifter that reads 64-bit input data from the two registers (*Reg_0* and *Reg_1*). It shifts its input by a certain number of bits (shift length) indicating the number of processed bits. The length is calculated using add and accumulate circuit. The shift length varies according to the operation type: either parsing or Huffman decoding. The barrel shifter output is passed to the *Demux_Controller* and Huffman decoder module.

### 2) HUFFMAN DECODING OPERATION

Many approaches have been explored to implement the Huffman decoder, such as the binary tree method [8], which is suitable for software implementation, and the lookup table approach [14]. However, both methods take many clock

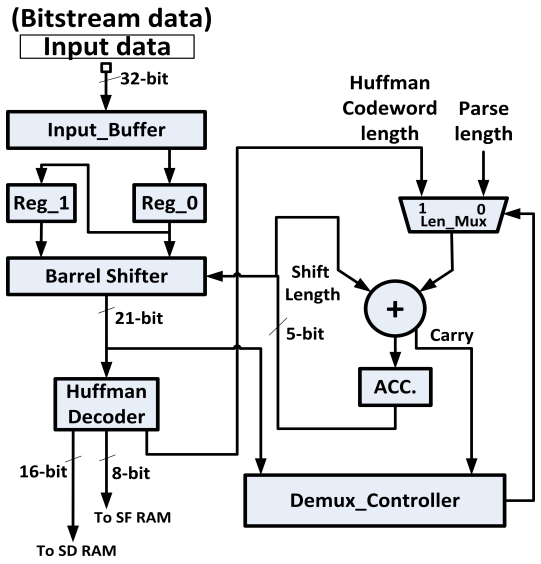


**FIGURE 7.** De-multiplexer and Huffman decoder (bitstream parser).

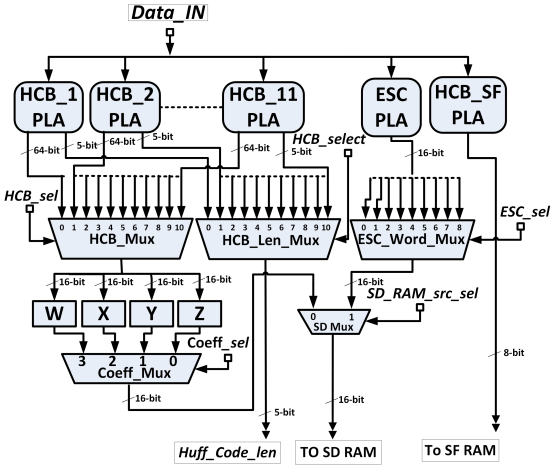


**FIGURE 8.** Huffman decoder architecture.

cycles to decode each code word. An approach based on the parallel programmable logic array (PLA) is used here. This approach is efficient for hardware implementation because of its high throughput property. It takes only one clock cycle to decode one common code word and takes two clock cycles to decode the escape (*ESC*) code words, (*ESC_Word*) and (*ESC_Prefix*) [4].

The proposed architecture of the Huffman decoder module is shown in Fig. 8. The PLA-like method has been used to store the 12 Huffman codebooks (HCBs) and the escape codebook. The barrel shifter contains the Huffman encoded bits. All inputs of the codebooks' PLAs are tied to the barrel shifter output, which makes the search process more efficient and able to perform in parallel. Each HCB PLA has two outputs, the decoded word and its length, which is used to calculate the new shift amount of the barrel shifter. The scale factor data are decoded using the Huffman codebook (*HCB_SF*), and then,

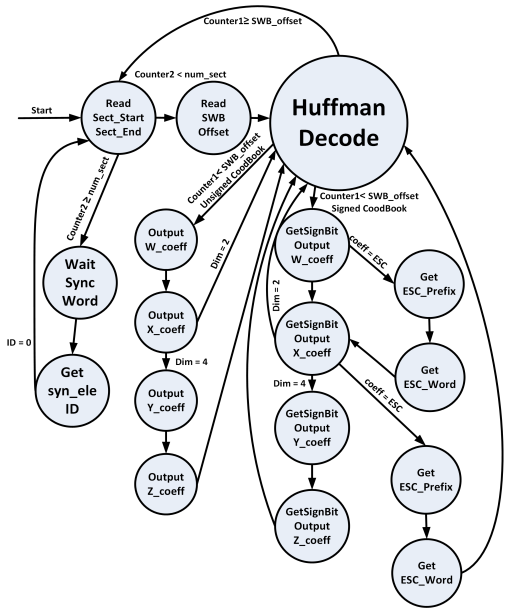


**FIGURE 9.** Huffman decoder controller FSM.

the spectral data are decoded. Depending on the codebook dimension, the resulting decoded word is de-grouped into 2-tuple ($W$ and $X$) or 4-tuple ($W$, $X$, $Y$, and $Z$) coefficients according to the standard [4].

The *Demux_Controller* sends select signals for the MUXs, write enables and addresses for *SD-RAM* and *SF-RAM*. The Huffman decoder controller FSM is shown in Fig. 9. The state "*Huffman Decode*" indicates that the bitstream is decoded in one clock cycle due to the PLA approach. This clock cycle is followed by 2 or 4 clock cycles to de-group the coefficients into *{W, X}* or *{W, X, Y, Z}*, respectively. Finally, 2 or 4 clock cycles are necessary to retrieve the sign bits from the bitstream.

### D. INVERSE QUANTIZATION AND RESCALING MODULE

The inverse quantization function is expressed as

$$X_{inv\_qunt} = sign\left|X_{qunt}\right|^{4/3}, \tag{1}$$

where $X_{qun}$ is the quantized coefficients (spectral data) and $X_{inv\_qunt}$ is the inverse quantized coefficients, which are rescaled by a gain value as described in (2) to obtain $X_{rescale}$, as illustrated in (3).

$$gain = 2^{0.25\times(sf-100)}, \tag{2}$$

where $sf$ is the scale factor.

$$X_{rescale} = X_{inv\_qunt} \times gain. \tag{3}$$

The *IQ_RESC* module reads the spectral data and applies the inverse quantization to them. Then, it rescales the outputs using the rescaling factors and writes the result to the *IQ_RAM*. The *IQ_RESC* architecture is shown in Fig. 10. The operation of this module is divided into two stages:

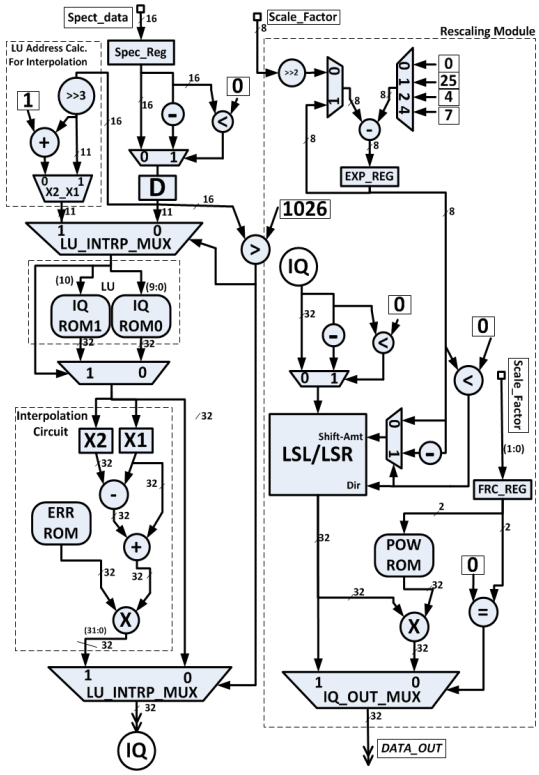


**FIGURE 10.** IQ_RESC module architecture.

#### 1) INVERSE QUANTIZATION STAGE

The algorithm used in FAAD2, an open-source software AAC decoder, is translated into hardware. There are two probabilities for the IQ operation depending on the spectral coefficient value. If the value is less than 1,026, the IQ is calculated using a direct lookup table (LUT); otherwise, the spectral coefficient is rescaled by a factor of 8. The scaling operation is a shift to the right by 3 bits to generate the LUT address. Then, a direct linear interpolation method is applied to the LUT output.

#### 2) RESCALING STAGE

The output from the previous stage is passed to the rescaling module, as shown on the right side in Fig. 10. It calculates the exponential part of the rescaling gain and then multiplies the IQ data by the gain. The multiplication is performed by a logical shift right/left (*LSL/LSR*) operation. The exponent part of the gain (*EXP*) register content is shifted by the IQ data value. If the fraction part of the scale factor is not equal to zero, it will be multiplied by one of those values $\{0, 2^{0.25}, 2^{0.5}, 2^{0.75}\}$, which are pre-calculated and stored in the power ROM (*POW ROM*).

### E. FILTER BANK MODULE

The filter bank reads the *IQ_RESC* output data from *IQ RAM*. The filter bank has the longest processing time in the decoding chain because it includes the IMDCT block. Therefore, the modules are fully pipelined to decrease the computation time.

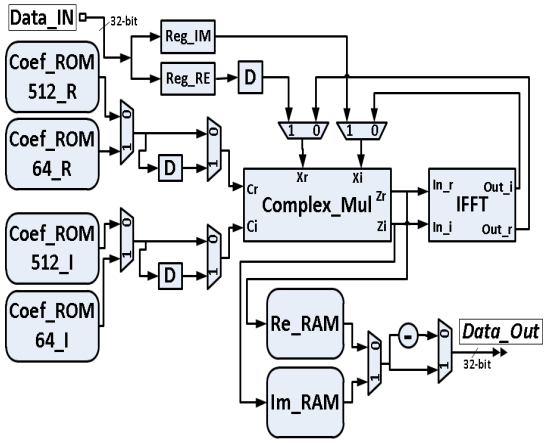


**FIGURE 11.** IMDCT module architecture.

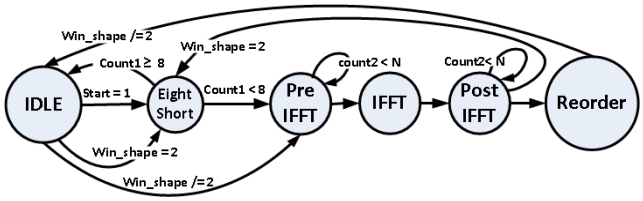


**FIGURE 12.** IMDCT controller FSM.

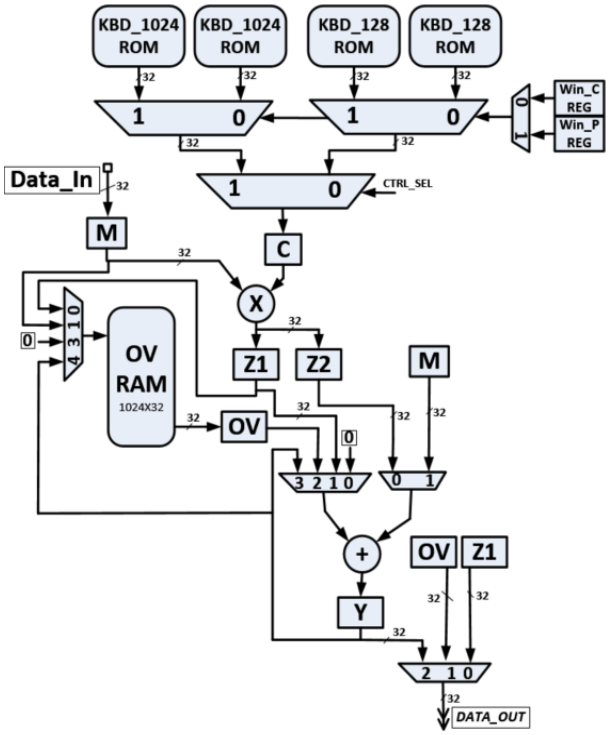


**FIGURE 13.** Windowing and overlapping architecture.

### 1) IMDCT ARCHITECTURE

This architecture is based on the FFT [19]. The IMDCT module is shown in Fig. 11. It consists of four stages: *pre-IFFT*, *IFFT*, *post-IFFT* and reordering. *Pre-IFFT* and *post-IFFT* are complex number multiplication operations. The delay registers (*D*) in Fig. 11 are introduced to synchronize the data flow to handle the pipelining timing. The IFFT block is implemented using a third-party FFT IP core. The IMDCT Controller FSM is shown in Fig. 12. According to the standard [2], if the window shape (*win_shape*) is short, the first three operations are repeated eight times followed by a reordering process.

### 2) PRE-IFFT OPERATION

The controller reads the inputs of the complex multiplier from the *IQ RAM*. Then, it latches them into real data and imaginary data registers (*Reg_RE*, *Reg_IM*). It performs the multiplication when the data are valid in those registers. At the same time, it fetches new data for the next multiplication operation and feeds the multiplication outputs to the IFFT module. This process is repeated 512 or 64 times according to the window sequence (window shape).

### 3) IFFT OPERATION

The IMDCT controller sends a start signal to the IFFT IP, and then, it enters a wait state until the IFFT starts to stream out the output data.

### 4) POST-IFFT OPERATION

The controller exits from the wait state and starts the Post-IFFT operation as soon as the IFFT starts streaming out the data. There is no time wasted buffering the result. The controller writes the Post-IFFT results to the real (*Re_RAM*) and imaginary (*Im_RAM)* RAM.

### 5) REORDERING OPERATION

The reordering process reads the data from the real and imaginary RAMs in a specific order according to the IMDCT algorithm [19].

### 6) WIN_OV ARCHITECTURE

The windowing operation consists of multiplying the IMDCT result by constant coefficients. To construct the final filter bank output, the first half values of the windowed results are overlapped by adding the first half of the samples of the current frame to the second half of the samples of the previous frame. At the same time, the second half of the samples of the current frame are stored into the overlap (*OV RAM*) to be overlapped with the next incoming frame.

The windowing and overlapping operations are pipelined as shown in Fig. 13. First, the controller fetches the values (*Data_In*) from the IMDCT RAM and the coefficients from one of *Kaiser-Bessel derived (KBD)* ROM. Then, the controller writes the input and the selected coefficient to the *M* and *C* registers. The contents of both registers are multiplied, and the result is written to the *Z1* and *Z2* registers. Next, it is overlapped with the overlap (*OV RAM*) contents, which is latched first to the *OV* register.

*KBD* ROM contains the *KBD* window coefficients [4], which are selected based on the window shape of the current and previous frames stored in the registers ((*Win_C REG)* and (*Win_P REG*), respectively).

### F. PCM CONVERTER MODULE

The PCM converter module is the final stage used to make the data ready for the audio codec. It is a pure combinational circuit that converts the filter bank output samples to audio PCM samples. It compares the output samples coming from the filter bank and determines whether they are larger than a 16-bit value to cast them in the adequate word length.

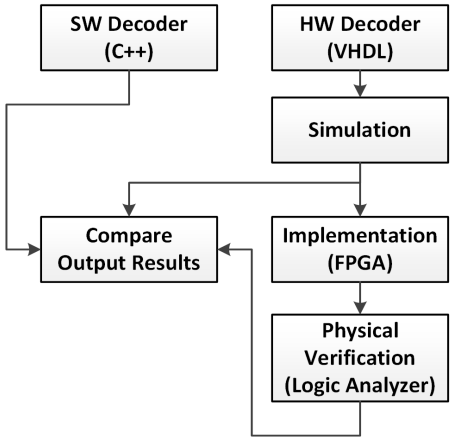


**FIGURE 14.** Design methodology validation process.

## V. RESULTS

### A. AAC DECODER VALIDATION PROCESS

The design methodology and validation process are shown in Fig. 14. First, a single AAC core unit was simulated with EDA tools. The simulation results were compared to the bitstream decoded by a golden software. Next, a single AAC core unit was implemented on FPGA and tested by loading a test ADTS bitstream into ROM. A trigger signal was generated to start the acquisition of the decoded audio samples. The samples were stored in a text file saved on the local memory of a logical analyzer. The text file was compared to the one generated by the golden software. The final step of the verification process was to check the full parallel decoder architecture, including the global controller and stream buffers. An initial validation was performed by comparing the decompressed audio streams and the outputs of the golden software. Next, an FPGA-based platform implementation was carried out. In the same manner, a logic analyzer was used to acquire and store the multiple decoded streams.

### B. SIMULATION RESULTS

The proposed architecture was simulated and compared to the reference model FAAD2 output. FAAD2 is a freeware open-source software AAC decoder, which has been explored, analyzed and then adopted to run in a fixed-point format. Fig. 15 shows a one-frame output from both the VHDL simulation of an AAC single core unit and the software reference model. The maximum absolute error is 0.00002, and the root mean square error (RMSE) is $5.61 \times 10^{-6}$.

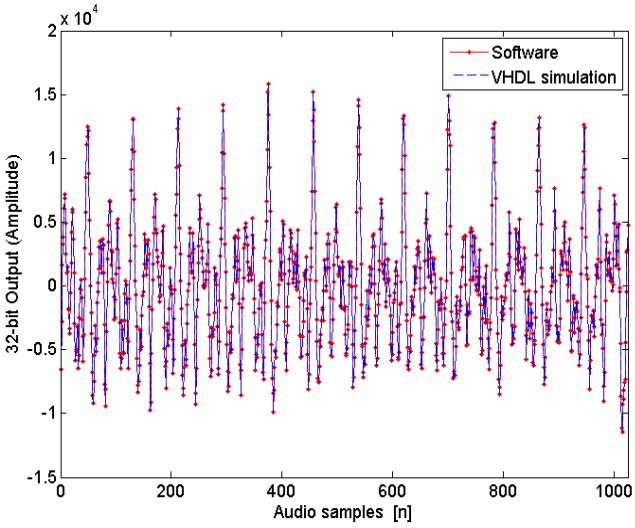


**FIGURE 15.** Comparison of the software output and VHDL simulation.

The consumed time in clock cycles for each block is shown in Table 1. These values are calculated for one frame, which takes 12,620 cycles to be decoded. The filter bank comprises 51.35% of the total computation time.

**TABLE 1.** Number of operation clock cycles in each block.

| Block | Clock Cycles |
|---|---|
| DEMUX_HUFF | 1,883 |
| IQ_RESC | 4,257 |
| IMDCT | 4,416 |
| WIN_OV | 2,064 |
| Total | 12,620 |

The filter bank [IMDCT + WIN_OV] has the longest execution time.

It is difficult to compare the simulation results with the related work because the number of implemented modules in each one is different. Thus, a comparison is performed for the individual blocks.

The proposed Huffman decoder module requires only one clock cycle to decode one word, whereas the decoder of Zhang *et al.* [12] requires 23-35 clock cycles. Due to the fully pipelined filter bank in the proposed architecture, it requires 6,480 clock cycles to process one frame, whereas the approach of Tsai and Liu [13] needs 15,362 clock cycles. The proposed design is faster by a factor of two and outperforms the implementation of Du *et al.* [20] as well.

The proposed architecture of a single AAC core unit requires only 12,620 clock cycles to perform full decoding, whereas the Tsai and Liu design [13] requires 30,081 clock cycles.

### C. FPGA IMPLEMENTATION RESULTS

The single AAC core unit was tested by loading a test ADTS bitstream into ROM and has been successfully decoded. Physical verification has been performed by measuring the

output signals from the FPGA using the logic analyzer. The captured signals correspond well with the simulation results. The full design of an AAC core unit is compared to previous similar work in Table 2. It decodes one frame faster compared to the best related work and thus improves the decoding speed by a factor of two.

The resource utilization for each block is listed in Table 3. The targeted device is an Altera Stratix V GS designed with a 28 nm process technology. The majority of the DSP elements are dedicated to the FFT in the filter bank. The FFT also occupies the largest area and RAM space. The proposed decoder requires 50.125 kb of memory bits, which is more than that required by the approach of Tsai and Liu [13]. This larger space requirement is due to the additional RAM needed in the filter bank to achieve pipelining. Finally, we notice that the power consumption is 98.82 mW at 0.85V core voltage.

**TABLE 2.** Comparison of proposed architecture with recent work.

| | [19] | [14] | [13] | [16] | This Article |
|---|---|---|---|---|---|
| Profile | LC | LC | LC | LC | LC |
| Approach | DSP | ASIC | ASIC | FPGA | FPGA |
| Frequency (MHz) | 44 | 5 | 1.3 | 4 | 50.54 |
| Clock Cycles (K) | 997 | 90 | 32 | N/A | 13 |
| Memory Bits (kb) | N/A | 17.7 | 27.75 | 30.36 | 50.125 |
| Power (mW) | N/A | N/A | 2.45 | N/A | 98.82 |

Compared to other designs presented in recent work, the proposed design has the shortest computation time in terms of required clock cycles, as shown in Table 2.

**TABLE 3.** Resource utilization of each entity for a single AAC core.

| Module | Logic Elements | Memory (bits) | DSP Blocks |
|---|---|---|---|
| DEMUX_HUFF | 2102 | 2560 | 0 |
| IQ_RESC | 868 | 0 | 4 |
| IMDCT | 5,454 | 134904 | 60 |
| WIN_OV | 510 | 159744 | 2 |
| PCM Converter | 50 | 0 | 0 |
| Main Controller | 8 | 0 | 0 |
| SF_RAM | 0 | 512 | 0 |
| SD_RAM | 0 | 16384 | 0 |
| IMDCT_RAM | 0 | 63488 | 0 |
| IQ_RAM | 0 | 32768 | 0 |

The majority of the block memory bits are dedicated to the IMDCT to achieve the pipelining. The KBD ROMs inside the WIN_OV are synthesized as memory blocks to reduce the logic area.

The majority of the block memory bits are dedicated to the IMDCT to achieve the pipelining. The KBD ROMs inside the WIN_OV are synthesized as memory blocks to reduce the logic area.

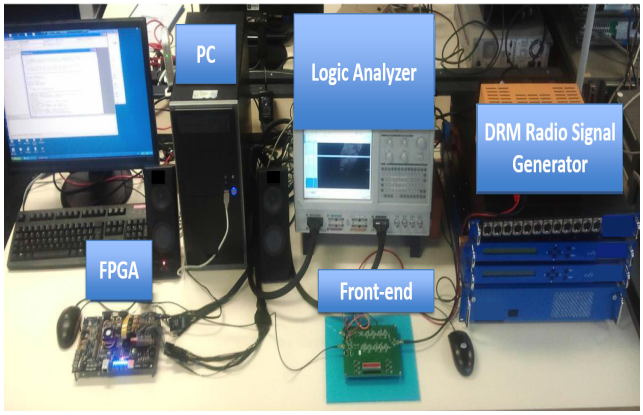


**FIGURE 16.** SurfOnHertz prototype.

**TABLE 4.** Resource utilization of the full parallel AAC decoder.

| | |
|---|---|
| Logic Elements | 153,675 / 262,400 (59%) |
| DSP Blocks | 1,650 / 1,963 (84%) |
| Block Memory bits | 10,273,850 / 52,572,160 (20%) |
| Maximum frequency | 50.54 MHz |

The results above are for N=25 AAC core units to process 50 parallel streams.

### *D. PARALLEL DECODER PROTOTYPE FOR THE SurfOnHertz RECEIVER*

Fig. 16 shows the experimental setup of the designed DRM receiver. The prototype receives (0 – 30) MHz DRM band signals from the signal generator via the front-end module and passes them to the channelizer. The outputs of the demodulators are decompressed using the proposed architecture. Next, the audio streams are indexed on the PC feeding a GUI, which allows users to search for the desired content extracted from the radio channels. In this first prototype, no extra cooling is added to the FPGA chip. However, it's advisable to install a heat sink on the final product since we use extensive physical resources of the FPGA.

Table 4 shows the resource utilization results of the proposed decoder implemented on an Altera Stratix V GS FPGA platform. We used various settings in the Quartus software to optimize resources utilization. The purpose is to be able to put as much AAC core units as possible. The decoder contains 25 AAC core units, which can decompress 50 parallel audio streams. The proposed decoder is scalable to a higher number of parallel streams. The full proposed decoder consumes 84% of the DSP blocks and 59% of the logic area due to the computation-intensive IFFT modules inside the filter bank. It requires a small amount of memory (20% of the available block memory). DSP blocks are considered the bottleneck to integrate more AAC decoder core units. The new FPGA platform generations contain more logic resources, more on-chip memory and more DSP blocks, thus meeting the scalability requirements to decode more audio streams.

## VI. CONCLUSION

This article presented a full-fixed-point AAC decoder architecture that allows for the rapid decompression of parallel audio streams and thus enables the indexing of DRM radio channels in real time. The global architecture consists of multi-AAC decoder core units, stream buffers, and a global controller. The core unit architecture includes the required tools to decode an ADTC bitstream with LC profile. It also supports a variety of sampling rate frequencies and operates in real time.

Speed and area factors are considered in the design of the AAC core unit. An approach based on a PLA-like method has been implemented, making the search process for codebooks possible in parallel. The IMDCT architecture is fully pipelined and operates for two types of window size: short and long. The proposed AAC decoder core unit can decode two frames faster than the best related work presented in the literature. Therefore, it enables a reduction in the number of required core units to decode N parallel streams by half and paves the way for the decoding of hundreds of compressed streams on a single chip. An initial prototype has been implemented on an Altera Stratix V GS FPGA platform. This prototype supports decompressing 50 parallel audio streams with an operating frequency of up to 50.54 MHz.

## REFERENCES

[1] F. Hofmann, C. Hansen, and W. Schafer, "Digital radio mondiale (DRM) digital sound broadcasting in the AM bands," *IEEE Trans. Broadcast.*, vol. 49, no. 3, pp. 319–328, Sep. 2003.
[2] *ETSI, Digital Radio Mondiale (DRM); System Specification*, ETSI Standard ETSI ES 201 980 and V3.1.1, 2009-08. 8, 2009.
[3] B. H. Tietche, O. Romain, and B. Denby, "Sparse channelizer for FPGA-based simultaneous multichannel DRM30 receiver," *IEEE Trans. Consum. Electron.*, vol. 61, no. 2, pp. 151–159, May 2015.
[4] *Information Technology—Generic Coding of Moving Pictures and Associated Audio Information, Part 7: Advanced Audio Coding*, document ISO/IEC 13818-7, 2004.
[5] V. Q. Do, N. Binh, S.-T. Chung, and S. Cho, "Design and implementation of an embedded multimedia live streaming decoder system," in *Proc. Int. Conf. Adv. Technol. Commun. (ATC)*, Hanoi, Vietnam, Oct. 2014, pp. 377–382, doi: 10.1109/ATC.2014.7043415.
[6] S. V. Pande and P. D. Bhirange, "Simulink based low power MPEG-4 AAC Audio encoder and decoder," in *Proc. Int. Conf. Commun. Signal Process. (ICCSP)*, Melmaruvathur, India, Apr. 2015, pp. 0418–0424, doi: 10.1109/ICCSP.2015.7322922.
[7] Y. Takamizawa, K. Nadehara, M. Boegli, M. Ikekawa, and I. Kuroda, "MPEG-2 AAC 5.1-channel decoder software for a low-power embedded RISC microprocessor," *J. VLSI Signal Process.-Syst. Signal, Image, Video Technol.*, vol. 29, no. 3, pp. 247–254, Nov. 2001.
[8] C. W. Ryu, D. H. Lee, H. J. Chi, K. S. Kwan, T. H. Kim, and J. S. Park, "Design of digital audio DSP core," in *Proc. 1st Int. Forum Strategic Technol.*, Ulsan, South Korea, Oct. 2006, pp. 59–62.
[9] V. Mesarovic, N. D. Hemkumr, and M. Dokic, "MPEG-4 AAC audio decoding on a 24-bit fixed-point dual-DSP architecture," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 3. Geneva, Switzerland, May 2000, pp. 706–709.
[10] T.-H. Tsai and D.-M. Chen, "A hardware/software co-design of high efficiency AAC audio decoder," *J. Signal Process. Syst.*, vol. 88, no. 3, pp. 345–356, 2017.
[11] Z. Tao, G. Buning, Q. Haojun, and Y. Fengping, "MP3/AAC audio decoder implementation based on hardware and software co-design," in *Proc. 3rd Int. Congr. Image Signal Process.*, vol. 8. Yantai, China, Oct. 2010, pp. 3695–3698.
[12] H. Zhang, M. Lu, and G. Wang, "An ASIC implementation of MPEG audio decoders," in *Proc. 7th Int. Conf. ASIC*, Guilin, China, Oct. 2007, pp. 754–757.
[13] T. H. Tsai and C. N. Liu, "Low-power system design for MPEG-2/4 AAC audio decoder using pure ASIC approach," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 1, pp. 144–155, Jan. 2009.
[14] P. Liu *et al.*, "VLSI implementation for portable application oriented MPEG-4 audio codec," in *Proc. IEEE Int. Symp. Circuits Syst.*, New Orleans, LA, USA, May 2007, pp. 777–780.
[15] G. Wang, H. Zhang, M. Lu, C. Zhang, T. Jiang, and G. Guo, "Low-cost low-power ASIC solution for both DAB+ and DAB audio decoding," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 4, pp. 913–921, Apr. 2014.
[16] A. Renner and A. A. Susin, "An MPEG-4 AAC decoder FPGA implementation for the Brazilian digital television," in *Proc. Southern Conf. Program. Logic*, Bento Goncalves, Brazil, Mar. 2012, pp. 1–6.
[17] R. C. Sampaio, P. de Azevedo Berger, and R. P. Jacobi, "Hardware and software co-design for the AAC audio decoder," in *Proc. 25th Symp. Integr. Circuits Syst. Design (SBCCI)*, Brasília, 2012, pp. 1–6, doi: 10.1109/SBCCI.2012.6344447.
[18] M. A. Watson and P. Buettner, "Design and implementation of AAC decoders," *IEEE Trans. Consum. Electron.*, vol. 46, no. 3, pp. 819–824, Aug. 2000.
[19] P. Duhamel, Y. Mahieux, and J. P. Petit, "A fast algorithm for the implementation of filter banks based on 'time domain aliasing cancellation,'" in *Proc. Int. Conf. Acoust., Speech, Signal Process.*, vol. 3. Toronto, ON, Canada, pp. 2209–2212, Apr. 1991.
[20] F. Du, Y. Song, D. Zhang, and M. Gao, "An implementation of filterbank for MPEG-2 AAC on FPGA," in *Proc. 2nd Int. Conf. Anti-Counterfeiting, Security Identificat.*, Guiyang, China, Aug. 2008, pp. 391–394.

**MOHAMMED SHAABAN IBRAHEEM** received the B.S. degree in computers and systems engineering from Minia University, Egypt, the M.S. degree in electronic design from Lund University, Sweden, and the Ph.D. degree from the UPMC Sorbonne University, Paris, France. His research interests include hardware architectures for signal processing algorithms.

**KHALIL HACHICHA** received the M.S. and Ph.D. degrees in electronics engineering from UPMC Sorbonne University in 2001 and 2005, respectively. He is currently an Associate Professor with UPMC Sorbonne University. His current research involves audio and video compression algorithms, embedded system design, and the power consumption of connected devices.

**OLIVIER ROMAIN** received the Engineering degree from ENS Cachan, the M.S. degree from Louis Pasteur University, and the Ph.D. degree from UPMC Sorbonne University, Paris, all in electronics. Since 2012, he has been the Head of the Architecture Department, ETIS Laboratory. He is currently a Full Professor of electrical engineering with Cergy-Pontoise University, France. His research interests include system-on-chip for broadcast and biomedical applications.