Design, Implementation and Evaluation of two MP3 Hardware Decoder in Different Abstraction Levels using SystemC

Felipe Goldstein
Institute of Computing
UNICAMP - State University of Campinas
Email: felipe.goldstein@students.ic.unicamp.br

Rodolfo Azevedo
Institute of Computing
UNICAMP - State University of Campinas
Email: rodolfo@ic.unicamp.br

Abstract— **The time-to-market requirements are forcing companies and designers to review their tools and methodologies. In this paper we have implemented, from scratch, two MP3 hardware decoders using SystemC, one at the RTL level and the other at the Behavioral level. We have validated them using the ISO reference specification and synthesized using TSMC 0.13um technology. To accomplish that, we took 6 designers for 12 months in the RTL level and 1 designer for 3 months to make 14 design points in the Behavioral level. We have compared the results and showed that the better Behavioral design point is faster then the RTL design and uses almost the same area. The 14 design points were also analyzed in respect to area, power consumption, energy consumption and latency.**

I. Introduction

Nowadays, the growing market of mobile devices stimulates multimedia applications so that everyone can hear its preferred music or see movies in any place. MP3 is the most used compression method for audio and was created in 1993 by the Fraunhofer institute. The algorithm was standardized as MPEG-1 Layer III (ISO 11172-3) [1], [2]. Using the knowledge of the limitations in the human hearing to eliminate information without affecting the sound quality perception, this algorithm obtains a good data compression.

Decoding an MP3 can be done in several different ways: using a general purpose processor, a dedicated Digital Signal Processor (DSP) or by a specific hardware decoder. The first two approaches are very similar, requiring only an off-the-shelf component or IP-core (the processor itself) and an embedded software to decode the music streams. The third approach requires more designers and time, and that is why we are focusing on two different abstraction levels of doing it: RTL and behavioral design. In all the cases, the designed decoder must be fast enough to decode the MP3 stream without affecting the sound quality.

We designed from scratch two hardware MP3 decoders, one using the classical SystemC RTL development and the other using behavioral synthesis. This work compares both approaches in respect to (1) the software reference model adopted, (2) the time required to make the design, and (3) the speed, area and power budgets. Both designs were completed successfully, have been fully validated using the ISO reference MP3 samples and have been tested in a FPGA design board.

As final result, the behavioral model was designed in three months using one designer and created 14 design points, which are different RTL implementations varying in area and latency. The RTL design took 6 designers in 12 months to make only one design point. In the remaining of the paper we show the tradeoffs and methodologies we used in this comparison.

The rest of this paper is organized as follows: Section II gives a brief overview of the MP3 algorithm. Section III shows two of the options we faced while looking for a reference design and the tradeoff between clean software (unoptimized) versus optimized software. Next, we describe both implementations in Section IV, followed by a section devoted to compare both approaches (Section V). Section VI discusses related work and we conclude the paper in Section VII.

II. The MP3 Algorithm

The base unit of coding of MPEG 1 Layer 3, or simply MP3, is a *Frame*. One music is divided in pieces of 1152 PCM samples per channel (2 if it is stereo) and these samples are coded and encapsulated in a *Frame*. The small granularity of a *Frame* allows the distribution of the MP3 data as *stream*, each *Frame* is independent and self-contained. All information necessary to decode is stored in the header and in the section of codified audio data. The header contains basic information about the MP3 bit-stream like Sample-Rate, Bit-Rate, Channel-Mode, and others.

The input of the decoding algorithm is an MP3 bit-stream and the output are PCM audio samples. First, the header is decoded and the frame is unpacked in different parts. Then begins the process of data reconstruction, the audio data stored in the frame must be processed by the Huffman algorithm and re-scaled, i. e. the values are re-scaled to a wider range of numbers. After that, the numbers pass trough an alias reduction filter. Finally the process of audio synthesis will produce the PCM audio samples. The audio synthesis uses two levels of a Fourier transform, the first level is a modified version of the cosine transform and the later is the regular cosine transform.

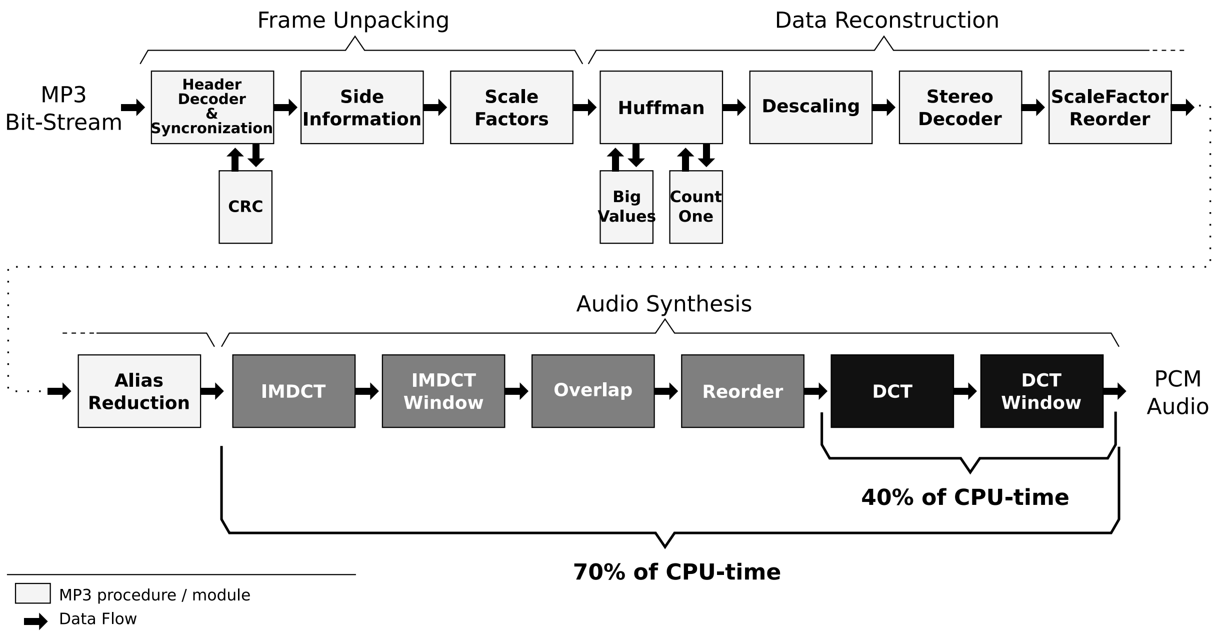

Fig. 1. MP3 pipeline implementation

Detailed information on the decoding and encoding process can be found in [1]–[7].

III. Reference Models

When talking about MP3, there are dozens of software implementations available so that the only problem is to choose a feasible one as the reference model design. We have looked at two different implementations: LibMad, an Open-Source library and the *dist10* ISO Reference Model, a source code released by the MP3 ISO standardization group.

The *dist10* is a software decoder developed to be ISO compliance and serve as a reference model to other MP3 software implementations. Because of its didactic purpose, it is very well documented, easy to understand and does not have any software optimizations like LibMad.

LibMad is a highly optimized MP3 decoding library that uses fixed point math. It is a tempting choice because the apparent easiness of converting the fixed point math code into the hardware description, compared to the difficulties imposed by a floating point implementation. Even though, analyzing more carefully the LibMad, it is easy to understand that all of its optimizations are just implementation decisions targeted to improve the software efficiency and does not represent an asymptotic improvement to the algorithm itself.

When considering those optimizations we found that they were implemented in such way that software can be more efficient, however they are not good for hardware implementation. We noticed that some software optimizations can make the hardware development more difficult and can get a worse hardware performance in respect to the used area and throughput.

The most common example of these optimizations that can be found in the LibMad is the use of a great amount of buffers and pre-calculated tables to diminish the amount of calculations that are executed in real-time. In the hardware, the usage of buffers considerably increases the area. However, the accomplishment of a calculation in the hardware can be cheap (less area) and most of times is easily implemented. Other common optimization is the use of pointers and dynamic memory allocation, which can not be easily implemented in hardware.

Moreover, the optimized software impose difficulties to understand the algorithm, modify and try different strategies of implementation by the designer. The clean source code also has other advantages: commonly it is more modularized and has a well defined data-flow, so it is easier to maintain and also helps the operation scheduling when synthesizing, what affect the latency. So, as a reference model, we choose to use the *dist10* ISO source code. Other reference sources used are master/doctor thesis [8]–[11] and books [3]–[5].

The source code developed by the ISO group can be found freely in the Internet, and is commonly referred by the name *dist10*. The version used in the project was downloaded from the website [12].

IV. Implementation

The MP3 decoding can be viewed as a pipeline of 14 stages. Each stage is a procedure that do some computation over a fraction of the input data. The sequence of decoding stages is illustrated in the Figure 1. By profiling the software code, we found out that the last two stages, named DCT and DCT Window, are responsible for 40% of the CPU time when running in software and the last seven stages (IMDCT, IMDCT Window, Overlap, Reorder, DCT and DCT Window) are responsible for 70% of the CPU time.

The decoder designed directly in SystemC RTL uses the same architecture illustrated by the Figure 1, where each pipeline stage is an independent SystemC module running in parallel with others as a pipeline. This partition permits the designer to reuse some basic unit modules that are common to others DSP applications, for example the DCT can be reused in other audio synthesis application or even in a JPEG implementation. It also helps the designer to concentrates his efforts in optimizing the slowest modules. On the behavioral model each procedure is executed sequentially like the software reference model. The behavioral synthesis will infer the parallelism on the loops specified by the designer.

The entire MP3 pipeline was designed in SystemC behavioral level. On the RTL version we decided to left out the first modules because they represent less than 30% of CPU time and can be done in software easily. So, on the RTL version we designed the pipeline from the *ScaleFactor Reorder* module to the end. In fact, most of the processing that has to be done in the first modules are related to header unpacking and error detection and are, usually, performed in software.

Even though, when comparing the RTL and the behavioral level version we always compare the area and latency corresponding to the part of the pipeline from the *ScaleFactor Reorder* module to the end. But when comparing the different design points of the behavioral level only, we use the entire pipeline.

We selected SystemC as a design language to have both RTL and behavioral versions implemented in the same language. The behavioral version was synthesized using the Forte Cynthesizer tool set.

A. Design Space Exploration

When making code modifications to optimize the hardware, the designer must concentrate his effort in the critical part of the algorithm. The software profiling can show where is the most CPU time consuming part and in the case of the MP3, the hardware design followed the pattern showed by the software profiling. In both the RTL and the behavioral implementation the bottleneck is the DCT module, that consumes 40% of CPU time as mentioned in previous section. Modifications in this module leads to expressive differences in latency.

The RTL MP3 implementation had a total latency of 300 cycles and the DCT had a latency of more than 250 cycles, the worst latency of the RTL modules. The RTL designers made an effort to implement another DCT with a better latency but it could not be characterized to run at the desired frequency with the used technology. The DCT is the most difficult module the RTL designers faced. The chain of arithmetic operations of the cosine transform impose a crucial tradeoff. If the operations are concatenated in the same clock cycle it can lead to a long critical path and could not fit in the desired clock cycle, or if the designer divides the operations in more cycles it leads to a longer latency. The RTL designers had chosen the second option.

Another difficulty the RTL designers faced was related to the pipeline synchronization. When implementing the pipeline stages the designers had chosen to use a complex handshake between the modules and independent buffers in each module to make the development easily. We believe that the use of a simplified handshake and shared buffers between the modules could improve the pipeline efficiency.

The complexity of designing the RTL MP3 from scratch did not allow us to apply an extensive design space exploration. So, the design space exploration was achieved only on the behavioral model.

On the behavioral synthesis we focused the modifications only in the inner loops of the critical modules, the DCT and IMDCT. We were able to achieve 14 completely different synthesis results. Each design point generated contains a full RTL implementation of the design. This facility helps the designer to select between different tradeoffs in typical designs. Two of the most important design optimizations we did are described bellow:

Loop Unrolling: We have unrolled four small inner loops in critical modules and got a latency reduction of 53% (from 541 cycles per sample to 243) at the cost of an area increase of only 6%. If we go further and also have set the synthesis tool schedule police ASAP (as soon as possible), the latency reduction goes to 67% but the area increase goes to 40%. These are the kind of tradeoffs designers face when using loop unrolling.

Loop Pipelining: We selected the same four small inner loops in critical modules to apply the loop pipelining optimization and got a latency reduction of 42% (from 541 cycles per sample to 294) at the cost of an area increase of 34%. If we again apply the ASAP schedule, the latency reduction goes to 65% and the area increases 60%.

All the design points will be shown in Section V.

B. Verification

Both models were tested in a similar way, by following the classic structure (see Figure 2): a set of input files and the respective output golden files. The input files are MP3 encoded files. Some of them came together with the *dist10* source code and others were produced to achieve a better coverage of encoding parameter combinations. The output golden files were produced decoding the MP3 files using the *dist10* implementation.

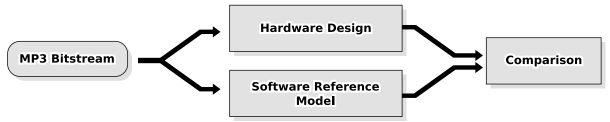

Fig. 2. MP3 verification methodology

The golden outputs are based on the PCM generated by the *dist10* compiled with *g++*. All comparisons between the golden outputs and the hardware decoded outputs are done in a RMS (Root Mean Square) manner, in the way imposed by the ISO standard:

- RMS = $sqrt(\frac{1}{N} \times sum(diff^2))$
- RMS must be $< \frac{2^{-15}}{sqrt(12)}$ relative to scale
- Max absolute value of diff must be 2^{-14} relative to scale

V. Results and Comparison

To compare both approaches, we selected two different groups of designers, group one with six designers to implement the RTL and group two with only one designer to implement the behavioral model. All the designers were undergraduate students with about the same knowledge level in signal processing and hardware design. They also received a set of training courses on SystemC RTL and behavioral design. The group one took 12 months to finish their design from the Scale Factor Reorder module to the end of the pipeline that corresponds to about half of the pipeline. The second group took only 3 months to finish all the design points in the behavioral description with the full pipeline implemented. Both designs were verified as shown in the previous section.

A. RTL Versus Behavioral

The first comparison we made focused only on the RTL design and the behavioral design point with basic configuration. In order to do this comparison we synthesized both designs from the Scale Factor Reorder module to the end of the pipeline (not the full pipeline), because the RTL design has only this part of the pipeline implemented as mentioned before. Using this part of the pipeline the RTL design had a latency of 300 cycles per sample and an area of $171Kum^2$ while the behavioral design had 196 cycles per sample and an area of $176Kum^2$. Table I shows the behavioral basic configuration and the RTL design with the pipeline synthesized from the Scale Factor Reorder module to the end and their corresponding Area (in um^2), Latency (in cycles) and Development time (in months). Therefore, with a shorter development time, using the behavioral synthesis, it is possible to get a better design in terms of latency (35% better) and almost the same area (3% worse). This synthesis was targeted to a TSMC $0.13um$ technology library.

TABLE I
Behavioral and RTL design with half pipeline synthesized

DP	Area	Latency	Development time
Behavioral	176	196	3
RTL	171	300	12

The big difference in the success of the Behavioral design over the RTL can be explained by two main reasons:

- The undergraduate students from both groups had none experience in hardware development prior to this project and the RTL design had to be optimized by the students themselves while the behavioral design had the help of the behavioral synthesis tool in the optimization. So, it can explain why the behavioral design has a better latency.
- The reference model in C is easier to be transformed in a synthesizable behavioral model, while the RTL has to be entirely implemented from scratch. It can explain why the RTL took so long to implement only half of the pipeline.

Therefore when training a workforce to the behavioral development model, the learning curve is shorter and can generate better results.

B. 14 Behavioral Design Points

The second comparison focused only the behavioral synthesis and was between the basic configuration without loop unrolling or pipelining and the different strategies of loop unrolling and pipelining. Following the experiments, we synthesized the complete pipeline (including the first stages) using the behavioral design and generated 14 design points in contrast to only one design point of the RTL design. All the design points were synthesized to run at 100MHz using the same TSMC $0.13um$ technology library. Table II shows all the design points produced by the behavioral synthesis and their corresponding Area (in um^2), Latency (in cycles), Power (in mW) and Energy Consumed to decode an MP3 stereo frame.

The Power was measured using the Synopsys Power Compiler tool with the RTL synthesized version of each design point and decoding an MP3 stream. Considering that an stereo frame two channels of 1152 PCM samples each the Energy consumed to decode an stereo frame can be calculated as: $Energy = 2 * 1152 * Latency * Power/Frequency$

The first design point (DP 1) was synthesized with the basic configuration without loop unrolling or pipelining. It has the best area ($243um^2$) and one of the worsts latencies (511 cycles). It has also the one of the smallest power consumption but, since it has a very big latency compared to the others, the amount of energy required to decode a sample is very high ($1863 \times 10^{-6}J$).

TABLE II
The behavioral 14 design points at 100MHz (full pipeline)

DP	Area	Latency	Power (mW)	Energy ($10^{-6}J$)
1	243	511	158	1863
2	325	381	158	1391
3	284	361	161	1345
4	254	290	161	1080
5	268	242	162	905
6	260	259	161	966
7	411	170	164	643
8	379	169	162	632
9	325	294	163	1105
10	331	473	158	1727
11	390	178	162	664
12	330	184	161	686
13	293	801	156	2879
14	292	1433	155	5117

We have plotted some graphs to show this data. In all graphs, the design points 13 and 14 are not shown because they are too distant from the others and could confuse the visualization of the other points. Figure 3 shows the Area X Latency tradeoff. In this graph, the best designs are the ones near the margins, having a good Area or a good Latency. Particularly the design points 4, 5 and 6 are the nearest to the origin, representing the better points to choose in this case. If the user wants a better Latency, say 170 cycles, he/she can choose the DP 7 at the cost of an area of $411um^2$.

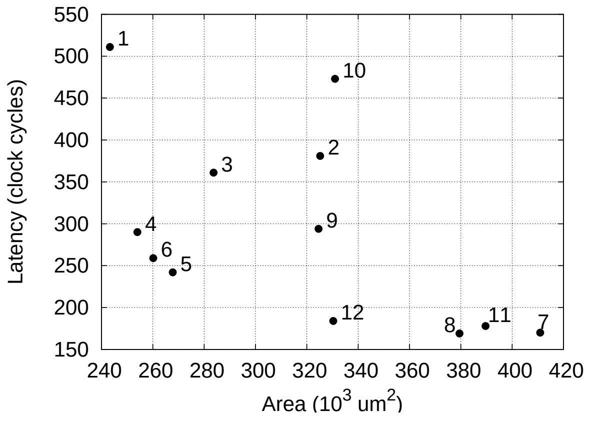

Fig. 3. Area vs Latency evaluation

Figure 4 shows the tradeoff between area and power but the user should also be aware of the amount of energy required to decode an MP3 frame by the decoder, as shown in Figure 5. Indeed, the Power axis in Figure 4 has a very small difference between the better and the worse but the Energy axis in Figure 5 is more coarse. The DP 1 can be selected if the designer look only Figure 4 as the design with best power consumption but this data is incomplete because DP 1 is exactly the one with the worse latency as shown before.

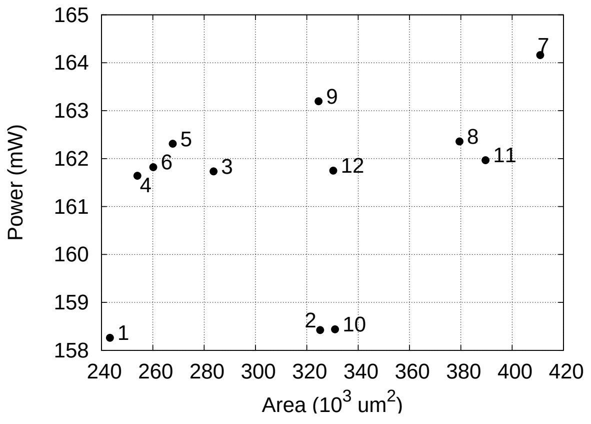

Fig. 4. Area vs Power evaluation

The other set of experiments we did is related to the minimum required frequency of operation for each design point. Since they differ in latency, the required frequency could be reduced in a different way for each one. To calculate the minimum frequency, we choose the frequency by which the decoder is able to decode the double of the maximum output data rate of the MP3. We doubled the number as a safety margin since there will be buffers in the outside margin of the decoder and we do not want the decoder to stall waiting for them. To get a CD-quality audio we must have a data rate of

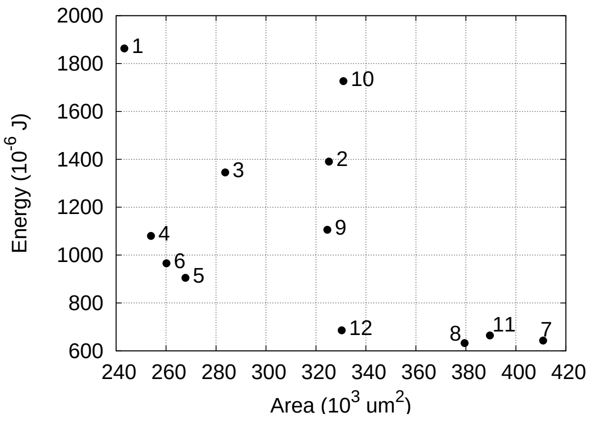

Fig. 5. Area vs Energy evaluation

44100 samples per second. So the new Frequency is calculated as: $NewFreq. = Latency * 44100 * 2$

The results are in Table III. The column Freq. has the calculated frequency and the new Power column now shows the big difference between the first and the other design points. This can be shown in the Figure 6, in contrast with the previous version (Figure 4) and similar to the Energy graph (Figure 5).

TABLE III
NEW FREQUENCIES FOR THE DESIGN POINTS

DP	Area	Latency	Freq. (Mhz)	Power (mW)	Energy ($10^{-6}J$)
1	243	511	45	71	1863
2	325	381	34	53	1391
3	284	361	32	51	1345
4	254	290	26	41	1080
5	268	242	21	35	905
6	260	259	23	37	966
7	411	170	15	25	643
8	379	169	15	24	632
9	325	294	26	42	1105
10	331	473	42	66	1727
11	390	178	16	25	664
12	330	184	16	26	686
13	293	801	71	110	2879
14	292	1433	126	195	5117

VI. Related Work

Diniz, So and Hall [13], developed a compilation system that maps high-level algorithms written in C to application-specific designs for FPGA and loop transformations to achieve design exploration. They evaluated five small applications such as FIR filter and Matrix Multiply.

Chtourou and Hammami [14] presented a methodology for SystemC behavioral synthesis and design space exploration using the Synopsys tool CoCentric SystemC Compiler. They evaluated 3 small applications: FIR, FFT and FIFO.

The work presented in this paper differs from previous efforts mainly in the size of the application evaluated. When a big application is being designed, the effect of the software reference model chosen has an important role on the success

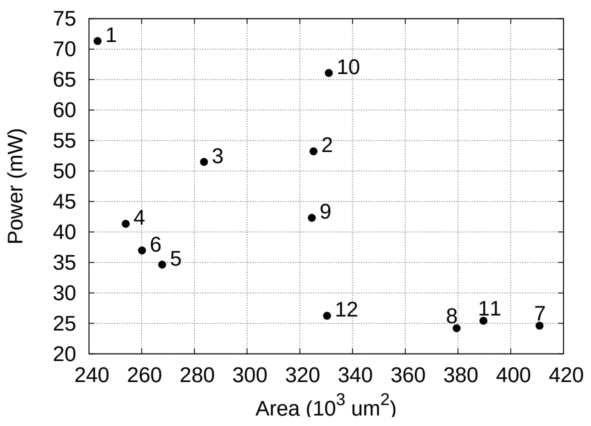

Fig. 6. Area vs Power evaluation with the recalculated frequency

of the design space exploration. This effect is not considered in the related papers. Furthermore, they do not compare the behavioral level development and its design space exploration to the classical RTL development.

VII. Conclusions

This paper presented a hardware implementation, design exploration and evaluation of an MP3 decoder fully designed in both behavioral and RTL SystemC. The MP3 decoder was validated in both abstraction levels on an FPGA by fully decoding ISO standard MP3 streams.

By using a commercial behavioral synthesis tool, targeted to a $0.13um$ technology, and a single designer working during a period of three months, we were able to make a wide design space exploration producing 14 different design points with latencies ranging from 170 to 1433 cycles, and areas from $243um^2$ to $411um^2$. One of the best design points we found used $268um^2$ of area and 242 cycles of latency. The same MP3 hardware decoder application, when designed directly in SystemC RTL, required six designers over one year to produce a single design point. Comparing the RTL design to a behavioral design point with same area, the behavioral design is 33% faster than the RTL. Furthermore, we point out that when training a workforce to the behavioral development model, the learning curve is shorter and can generate better results.

As part of our design exploration methodology, we evaluated two different C source-code reference designs: LibMad, a highly optimized MP3 decoder software library, and the ISO reference design, a non-optimized, clean and easy to understand code. While transforming these two reference designs into behavioral code, we found that calculating data on-the-fly as described in the non-optimized software version, produced much better hardware (less latency and less memory usage) than the highly optimized software implementation which uses pointers and pre-calculated data tables. Therefore it leads to the conclusion that the use of an optimized software as direct reference for the hardware project does not imply an optimized hardware. The opposite can occur: there is a great risk of getting a much less optimized hardware from an optimized software. Also, the choice of reference model is crucial to the success of design space exploration.

Another conclusion is that, when looking for a variety set of configurations, like the 14 design points generated by the behavioral synthesis tool, the designer needs to take care about the meaning of the results he/she has. We have showed in some graphs that one design point that appears to be the better in area and power requirements, in fact, has the worse energy consumption because of its high latency.

Finally, we conclude that the Behavioral synthesis tool can be a good alternative to create several design points, allow designers to make better decision in the latter phases of the design and also allows an improvement in the productivity (in our case, 4 times faster with 6 times less designer).

Acknowledgment

This project has received partial support from FAPESP and CNPq. We would like to thank the Brazil-IP students who worked on the RTL model: Andre Costa, Yang Ju, Edwin Cordeiro, Samuel Goto, Thiago Nasciutti and Valdiney Pimenta.

References

[1] ISO, *MP3 Standard*. Information technology - International Organization for Standardization (ISO/IEC), 1993.

[2] ——, "Coding of moving pictures and associated audio for digital storage media at up to about 1,5 mbit/s - part 3: Audio," Information technology - International Organization for Standardization (ISO/IEC), Tech. Rep. 11172-3, 1993.

[3] M. Ruckert, *Understanding MP3 - Syntax, Semantics, Mathematics, and Algorithms*. Vieweg, Junho 2005.

[4] D. Pan, *A Tutorial on MPEG/Audio Compression*, ser. issue 2. IEEE Multimedia, 1995, vol. 2, pp. 60–74.

[5] J. Watkinson, *MPEG-2*. Focal Press, 1999, vol. 1, pp. 124–159.

[6] K. Brandenburg and H. Popp, "An introduction to mpeg layer-3," *Fraunhofer Institute, EBU Technical Review*, 2000.

[7] S. B. Marovich, "Faster mpeg-1 layer iii audio decoding," HP Laboratories Palo Alto, Tech. Rep. HPL-2000-66, June 2000.

[8] A. Ehliar and J. Eilert, "A hardware mp3 decoder with low precision floating point intermediate storage," Master's thesis, Linkopings universitet, 2001.

[9] K. Lagerstrm, "Design and implementation of an mpeg-1 layer iii audio decoder," Master's thesis, Chalmers University of Technology, May 2001, computer Science and Engineering Program - Department of Computer Engineering.

[10] S. Gadd and T. Lenart, "A hardware accelerated mp3 decoder with bluetooth streaming capabilities," Master's thesis, November 2001, in cooperation with C Technologies AB.

[11] K. Salomonsen, "Design and implementation of an mpeg/audio layer iii bitstream processor," Master's thesis, Aalborg University, Denmark, 1997.

[12] "http://www.mp3-tech.org/."

[13] B. So, P. C. Diniz, and M. W. Hall, "Using estimates from behavioral synthesis tools in compiler-directed design space exploration," in *DAC '03: Proceedings of the 40th conference on Design automation*. New York, NY, USA: ACM Press, 2003, pp. 514–519.

[14] S. Chtourou and O. Hammami, "Systemc space exploration of behavioral synthesis options on area, performance and power consumption," in *ICM 2005: Proceedings of the 17th International Conference on Microelectronics*, 12 2005, pp. 5 pp.–.