A DSP Based H.264 Decoder for a Multi-Format IP Set-Top Box

Fernando Pescador, *Member, IEEE,* César Sanz, *Member, IEEE*, Matías J. Garrido, Eduardo Juárez, *Member, IEEE*, and David Samper

Abstract — *In this paper, the implementation of a digital signal processor (DSP) based H.264 decoder for a multi-format set-top box is described. Baseline and Main profiles are supported. Using several software optimization techniques, the decoder has been fitted into a low-cost DSP. The decoder alone has been tested in simulation, achieving real-time performance with a 600 MHz system clock. Moreover, it has been integrated in a multi-format IP set-top box allowing the implementation of actual environment tests with excellent results. Finally, the decoder has been ported to a latest generation DSP* [1].

Index Terms — IP-Set-Top Box, DSP, H.264, multi-format video decoder.

I. Introduction

In home entertainment networks, set-top boxes (STBs) are becoming key devices. STB usage is two-folded. On one hand, they are employed as digital television (DTV) receivers but, on the other hand, STBs are used as residential gateways to deliver multiple services [1]. To gain in flexibility and modularity in home networks, STBs functionality may be distributed among a main device and several peripherals, all of them interconnected by an Ethernet network [2]. The peripheral devices are IP DTV decoders. Since a user can locate these decoders close to each TV set, they are also called IP-STBs [3]. In addition, these devices can decode audio and video information from the Internet.

The aforementioned distributed functionality model will be successful only if the IP-STBs are inexpensive and versatile enough. Video and audio decoders are key elements. The solutions based on non-programmable decoders [4] cannot cope with the quick evolution of audio and video decoding algorithms. On the other hand, the latest generation of digital signal processors (DSPs) [5]-[8] can support inexpensive and flexible multi-format decoders [9].

Lately, new video coding standards [10] have been adopted allowing more data compression. However, the complexity for both encoders and decoders has increased as well [11]. With the latest generation of Digital Signal Processors (DSPs), very flexible decoders can be implemented at a relative low cost. The complexity of an H.264 decoder may double that of an MPEG-4 SP decoder [11]-[12], which in turn is more than that of an MPEG-2 decoder. Thus, a real-time H.264 standard definition DSP-based decoder is hard to obtain [13]-[16].

In the last years, our work has been focused on the development of a DSP based multi-format IP-STB for different audio and video standards [17]-[19]. In this paper, the implementation of a video H.264 decoder for the aforementioned IP-STB is described.

This paper is organized as follows. In section II the IP-STB architecture is briefly explained for reference. In section III the decoder algorithm is explained. Section IV is devoted to the decoder optimization details. In section V simulation test results for the decoder alone are reported. Section VI explains the tests carried out with the IP-STB in an actual environment. Our work in progress is outlined in section VII. Finally, section VIII is devoted to the conclusion and future work.

II. IP-STB Architecture

In this section, the IP-STB architecture is briefly explained for reference. Details can be found in [17].

A. DSP architecture

The IP-STB has been designed using a fixed point video-oriented DSP [5]. In Fig 1, a simplified block diagram of the DSP internal architecture is shown.

The CPU is a VLIW processor with a performance of up to 4800 MIPS @600 MHz. There are two 16 KB level-1 caches for code (L1P) and data (L1D). Moreover, there is a 256 KB internal SRAM that can be configured as a level-2 cache (L2) and/or as an internal data/program memory.

The external memory is accessed through a dedicated interface, EMIF, using a 64-bit data interface. The other peripherals are a DMA controller, two video ports, an Ethernet port (EMAC), an output audio interface (McASP) and several general-purpose I/O pins (GPIO).

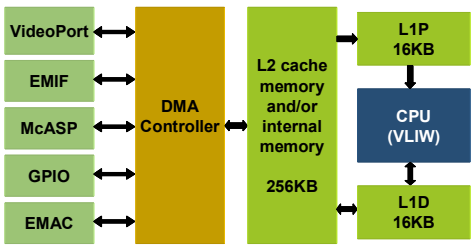


Fig 1. Architecture of the DSP.

[1] This work was supported by the Spanish Ministry of Science and Technology under grant TEC2006-13599-C02-01 and by the Comunidad de Madrid regional government under grant S-0505/TIC/0398.

F. Pescador, C. Sanz, M. J. Garrido, E. Juarez and D. Samper are with the Electronic and Microelectronic Design Group (GDEM) at the *Universidad Politécnica de Madrid*, Spain. (e-mail: {pescador, cesar, matias, ejuarez, dsamper}@sec.upm.es).

Manuscript received January 15, 2008

The DMA controller allows moving data between memory and peripherals. DMA transfers can be requested by L1P, L1D, L2, the user and the peripherals. The user programmable requests can be QDMA (Quick DMA) and EDMA (Enhanced DMA, more flexible but slower).

B. IP-STB Hardware Architecture

A block diagram of the IP-STB hardware architecture can be seen in **Fig. 2**. The DSP interfaces to an Ethernet port, two external memories, a video encoder, an audio digital to analog converter (DAC), an infrared receiver and a JTAG emulator using a minimum amount of glue hardware. The DSP reads from the Ethernet port an MPEG-2 Transport Stream (MP2TS) encapsulated over IP/UDP containing, among others, the program selected by the user. The DSP outputs the video data to the encoder in ITU-R BT.601 format and the PCM audio (Pulse Coded Modulation) to the DAC. The video encoder generates composite video (CVBS) and S-video (Y/C) to interface with a standard TV set. The audio DAC outputs a stereo analog audio signal. An infrared sensor connected to a general purpose DSP input port allows the implementation of a remote control system. Finally, 1 MB of Flash memory and 16 MB of SDRAM are provided as program and/or data memory. The prototype shown in **Fig. 3** implements the IP-STB with a 600 MHz system clock.

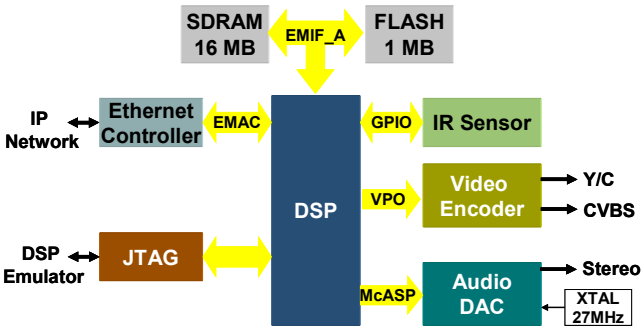


Fig. 2. IP-STB hardware architecture block diagram.

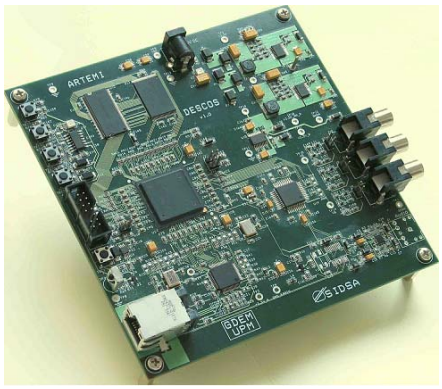


Fig. 3. IP-STB prototype.

C. IP-STB Software Architecture

The IP-STB software has been developed using an RTOS [20] that supports the definition of tasks, several inter-task communication methods and interfaces to the hardware. The IP-STB software architecture is based on RF5 [21].

A block diagram including tasks, algorithms, buffers, SCOMs and SIOs is shown in **Fig. 4** (see [21] for notation details). There are six tasks: the *Transport* task reads the MP2TS from the Ethernet port and splits the audio and video streams in two buffers. The *Video dec* task reads the video stream from one of these buffers, decodes the pictures and stores them also in a buffer. The *Video play* task reads the decoded pictures and writes them to the video port. The *Audio dec* and *Audio play* tasks perform similar operations with the audio stream. Finally, the *Application* task implements the interface with the user and configures the other tasks.

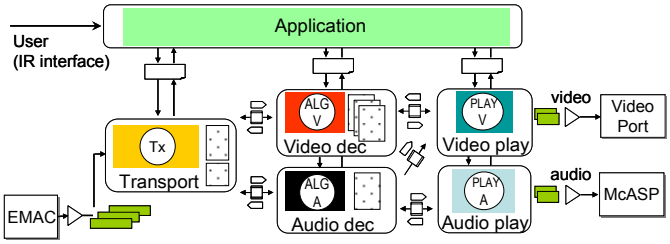


Fig. 4. The IP-STB software architecture based on RF5.

III. H.264 Decoder Algorithm

The decoder implements the Baseline Profile and the Main Profile of H.264 video coding standard [10], [22], [23] at level 3. Interlaced video, Multiple Slice Groups (MSG) and Arbitrary Slice Ordering (ASO) are not currently supported.

In **Fig. 5**, a simplified flow diagram of the decoding process for a Network Adaptation Layer (NAL) unit is shown. The decoder reads the H.264 stream from an input buffer and decodes the NAL units in sequence. After decoding the NAL header, the NAL unit content is identified as a slice header or another syntax element (e.g. an SPS or a PPS, see [10], [22], [23] for details). When the NAL unit contains a slice, the decoder executes a loop for each macroblock (MB).

Fig. 5 also shows the MB loop for INTER coded MBs. Data read from stream is entropy decoded using Context Adaptive Binary Arithmetic Coding (CABAC) or Context Adaptive Variable Length Coding (CAVLC). After decoding the slice header, up to 32 motion vectors may be read for a unique MB (e.g. sixteen 4×4 luma blocks with bidirectional prediction in the Main Profile). Afterwards, the Integer Cosine Transform (ICT) coefficients are read and the Inverse Integer Cosine Transform (IICT) is computed to obtain the residual MB. The different reference blocks are read from previous decoded pictures using the motion vectors and then, the predicted MB is obtained. The residual MB is motion compensated by adding the prediction and the result is filtered (with the deblocking filter) and written to the current decoded picture.

For INTRA coded MBs there are neither motion vectors nor reference data; instead, an INTRA prediction computed from the neighbor MBs is used.

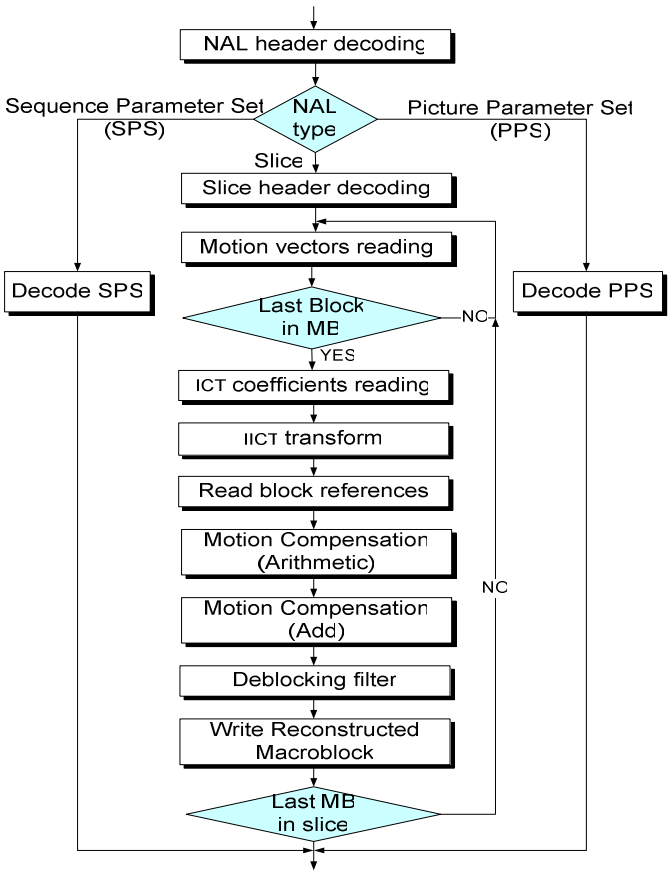


Fig. 5. Simplified flow chart for decoding a NAL unit.

IV. H.264 Optimization Process

The starting point in the implementation of the algorithm outlined in section III was a standard compliant raw-C decoder fully tested first in a PC environment and moved to the DSP environment afterwards. Initially, the code was located entirely in external memory and spent typically about 10^9 clock cycles per frame to decode H.264 MP Level 3 streams (i.e., it was able to decode less than 1 fps).

In this implementation, the CPU executed all the MB loop operations in sequence as it is shown in **Fig. 6**. The CPU was continuously accessing to external memory allocated data. Moreover, the CPU parallelized instructions were not used because the algorithm was executed at a pixel basis.

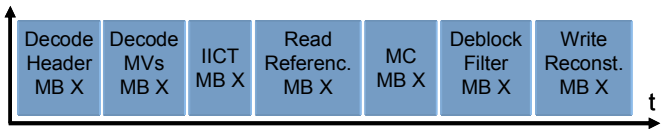


Fig. 6. Scheduling of decoder execution for a MB.

This initial implementation was optimized to increase the execution speed in about two orders of magnitude using, among others, the techniques described in [17] and [24]-[25].

In this optimization process, several code and data sections have been allocated in the internal DSP memory. In particular, the data used in the MB loop algorithm were allocated in internal memory and moved from/to external memory using explicit DMA transfers[2]. The loop has been re-scheduled in order to increase the parallelization of DMA transfers and CPU execution. Moreover, the deblocking filter and its related data flow have been heavily optimized. Finally, several code sections have been written directly in assembly language. The details of this optimization process are given in the following subsections.

A. Allocation of the MB loop data in internal memory

In this optimization, the data used in the MB loop are allocated in internal memory to increase the execution speed. As can be seen in **Fig. 7**, several buffers are allocated in internal memory. The reference data pointed by the motion vectors are moved from the reference picture buffers to the REF buffer. Actually, there are three REF buffers: one for luma (REF_Y) and two for chroma (REF_CR and REF_CB, not shown in **Fig. 7** in sake of clarity). The ICT_COEFFS buffer is used to store the ICT coefficients and also to store the residual MB. A ping-pong buffer, REC, is used to store the MB prediction, computed using the data stored in the REF buffers. Then, the residual MB is added to the prediction and stored also in the REC buffer. Finally, this reconstructed MB is filtered and moved to the current picture buffer afterwards.

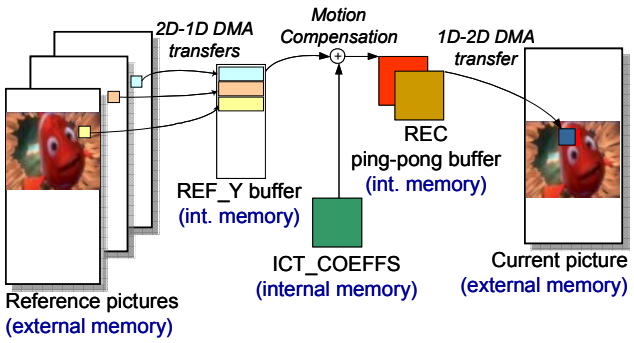
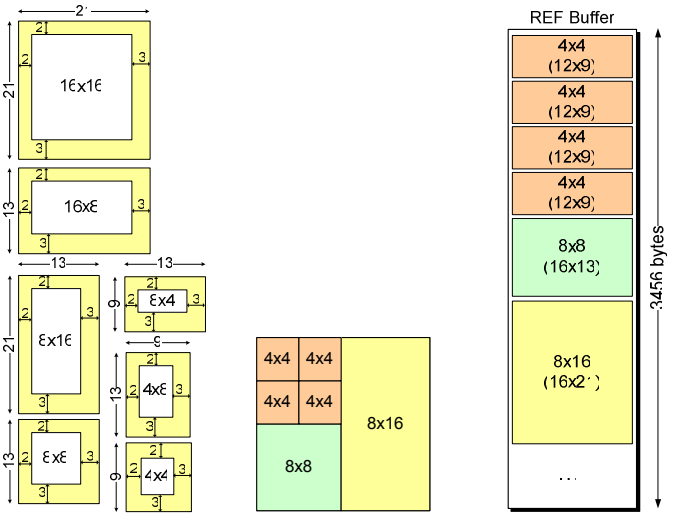


Fig. 7. Transfers between internal and external memory and use of internal buffers to decode one MB.

The implementation of the REF buffers needs a more detailed description. The REF_Y buffer is 3456 bytes length. This room is enough to store all the reference blocks and their borders in the worst case (thirty-two 4×4 blocks). The blocks must be moved with their borders because 1/4 pixel arithmetic may be further applied to them. **Fig. 8**-a shows, only for luminance, all possible block sizes with their borders, and Fig. 8-b is an example of how different block types can be combined in a MB. Finally, Fig. 8-c shows the content of the REF_Y buffer for a reference used to predict a MB like the one presented in **Fig. 8**-b. In **Fig. 8**-c, the memory size (shown in brackets) needed to allocate blocks, is higher than the block size shown in **Fig. 8**-a; this is to allow the 32-bit aligned DMA transfers that will be mentioned in the next paragraph. The REF_CR and REF_CB buffers are implemented similarly.

[2] QDMA transfers have been used in all cases (see section II.A).



a) Block sizes with borders b) An example c) The REF_Y buffer

Fig. 8. Contents of the REF_Y buffer.

To move the reference and reconstructed data from/to external memory to/from internal memory, explicit 32-bit aligned DMA transfers are used. The new MB loop flow chart can be seen in Fig. 9. After each motion vector is read, three DMA requests are started to move the luma and chroma reference data from (one or more) picture buffers in external memory to the REF buffers in internal memory. The data movement of a block is parallelized with the (CAVLC or CABAC) decoding of the next motion vector. The data movement of the last reference blocks is parallelized with the IICT computation. After the motion compensation and the deblocking filter are performed, three DMA requests are started to move luma and chromas from the ping-pong buffer to the current picture buffer. The time scheduling of Fig. 10 shows how the CPU processing is parallelized with the DMA transfers. The use of a ping-pong buffer allows the CPU to write on a buffer while the DMA controller is transferring data from the other one.

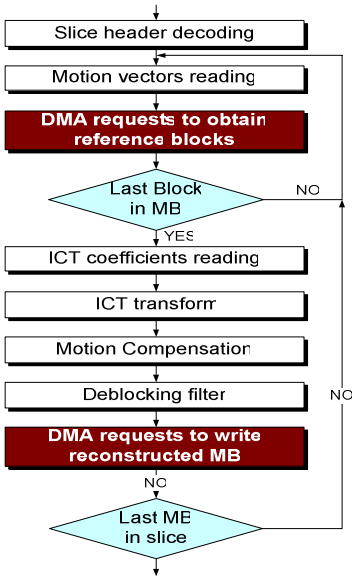


Fig. 9. Flow chart for decoding a MB including DMA transfers.

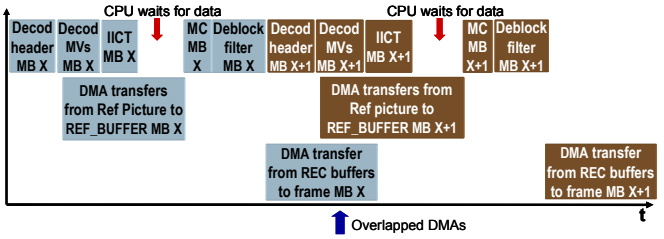


Fig. 10. Scheduling of DMA transfers and CPU processing.

B. Loop reorganization

In the schedule shown in **Fig. 10**, when there are several block references for a MB, a lot of simultaneous DMA transfers are requested so, usually, the CPU must wait for the reference data in order to perform the MB motion compensation. Moreover, while a reconstructed MB is being transferred to external memory several DMA transfers are requested to obtain the references for the next MB. In this case, the DMA controller can collapse and then the DMA transfers may become slower.

To solve these problems, the MB decoding loop has been re-scheduled in such a way that the deblocking filter of the current MB has been delayed until the decoding of the next MB. The new flow diagram is shown in **Fig. 11** and the time diagram of the new schedule can be seen in **Fig. 12**. To minimize CPU waits, the DMA transfers of the reference data for MB #X are overlapped with the deblocking filter of the MB #X-1. In addition, as the DMA transfers of the MB #X-1 filtered data (from the ping-pong buffer to the current picture buffer) are further from the DMA transfers of the reference data for MB #X+1, the amount of simultaneous DMA requests is reduced.

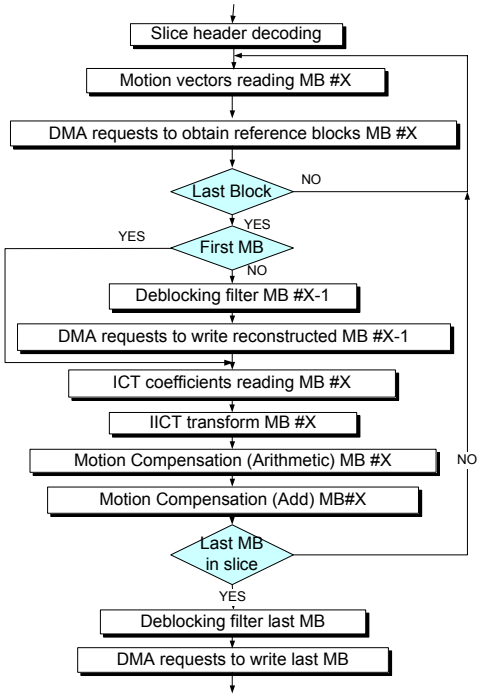


Fig. 11. Loop reorganization to reduce the CPU waits.

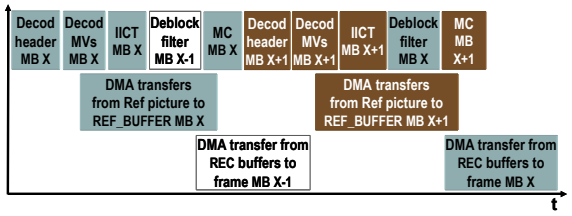


Fig. 12. Improved scheduling of DMA transfers and CPU processing.

C. *The deblocking filter and its related data flow*

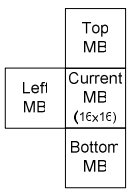
The deblocking filter operation adds more complexity to the decoder data flow. To filter the current MB, the 4 bottom rows from the top MB and the 4 rightmost columns from the left MB (see **Fig. 13**-a) are used. The REC (ping-pong) buffers are dimensioned to provide enough room for these pels (**Fig. 13**-b). After the unfiltered current MB has been moved to the REC buffer, the following operations must be performed prior to filter:

- The 4 rightmost columns of the REC buffer that were used to filter the former MB (MB #X-1 in **Fig. 13**-c) are moved to the 4 leftmost columns of the current REC buffer (the one used to filter the current MB, MB #X).
- The rightmost column in current REC buffer is saved in a small buffer allocated in internal memory (**Fig. 13**-d). These data will be eventually used to compute the next MB prediction, if INTRA.
- The bottom row is also saved in an internal memory buffer (**Fig. 13**-e). This information may be used to compute the MB INTRA prediction of the bottom MB (**Fig. 13**-a) so it must be stored in a line (picture-width) size buffer.
- The 4 bottom rows of the top MB are moved from an internal buffer to the current REC buffer (**Fig. 13**-f).

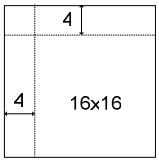
After these steps, the current REC buffer is ready and the current MB can be filtered. After filtering, two operations must be performed:

- The filtered MB must be moved from the current REC buffer to the current picture buffer (**Fig. 13**-g).
- The 4 bottom rows of the current REC MB are not moved to the current picture buffer. Instead, they are moved to an internal memory buffer (**Fig. 13**-h) so as they will be available in the bottom MB filtering process. Actually, the 4 rightmost pels in each row are not saved until the next MB filtering operation.

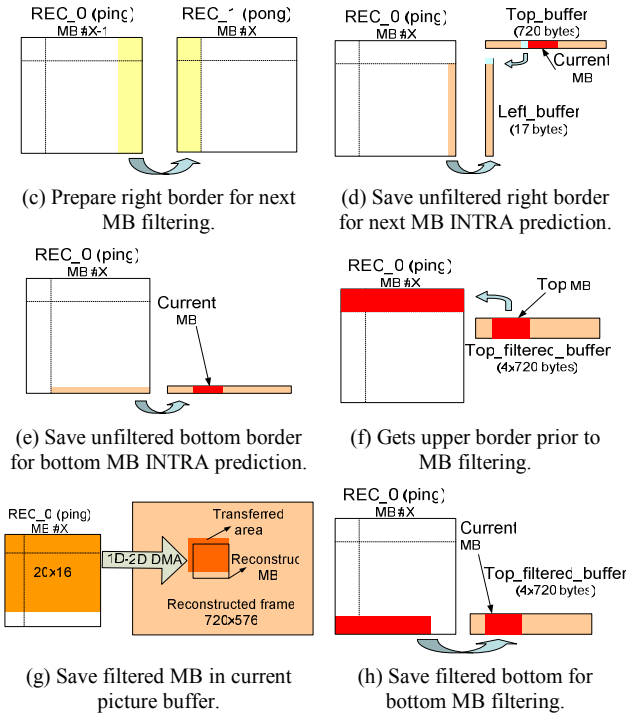
The chroma blocks are processed in a similar way using additional buffers.



(a) Top, left & bottom MBs used in the current MB filtering.



(b) Structure of the REC_0 (ping) and REC_1 (pong) buffers.



(c) Prepare right border for next MB filtering.

(d) Save unfiltered right border for next MB INTRA prediction.

(e) Save unfiltered bottom border for bottom MB INTRA prediction.

(f) Gets upper border prior to MB filtering.

(g) Save filtered MB in current picture buffer.

(h) Save filtered bottom for bottom MB filtering.

Fig. 13. Deblocking filter related data flow.

D. *The deblocking filter optimization*

The deblocking filter can be applied to any 4×4 adjacent blocks in both directions, horizontal and vertical. In **Fig. 14**, the flow chart of the filter algorithm for luma horizontal edges is shown. This filter operation concerns to the current and top blocks as shown in **Fig. 15**. The target samples are the 3 lower rows of the top block and the 3 upper rows of the current block. Up to 24 FIR filtering operations may be needed to carry out the filtering in the vertical direction (horizontal edges).

Three parameters must be obtained that determine the characteristics of the filter. The Boundary Strength parameter (BS) is derived from a complex set of conditions (see [10] for details) and determines the strength of the filters for each 4×4 block. The two threshold parameters, α and β, are derived from the quantification parameters used for the current and top blocks coding and essentially determine which samples are filtered. The value of α and β remains constant for the entire MB.

The algorithm shown in **Fig. 14** is repeated four times per block, once for each column. The parameters BS, α and β are used to evaluate if a sample is to be filtered and, if so, to compute the six conditions described in **Fig. 14** (cond1,…, cond6). When these conditions are true or false, different algorithms may be applied to compute a sample. For example, the sample in position p0X (X=0…3) can be computed using four different algorithms (denoted as $P0X_0$,…, $P0X_3$ in **Fig. 14**): $P0X_0$ when cond1 is false, $P0X_1$ when cond1 is true and cond4 is false, $P0X_2$ when cond1 and cond4 are true and cond5 is false, and $P0X_3$ when cond1, cond4 and cond5 are true.

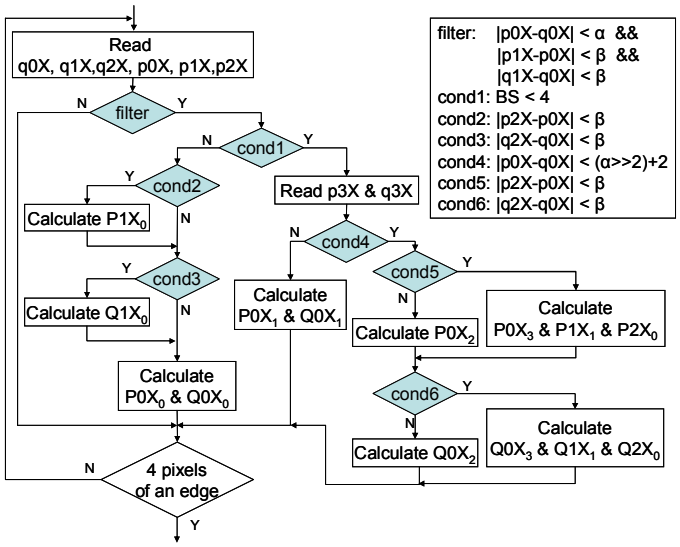


Fig. 14. Horizontal edge filter for column X of a luma 4×4 block (X=0...3).

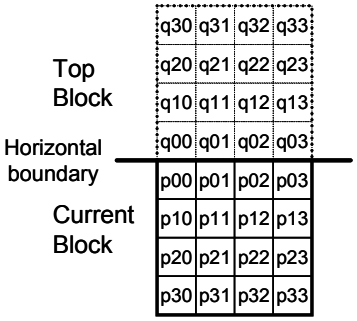


Fig. 15. Pixels used to filter a 4×4 block horizontal edge.

For the vertical edge and the filtering of the chroma samples the algorithm is analogous. Since there are many complex operations and conditional branches, these algorithms are difficult to parallelize efficiently.

The following optimization quits the branches and increases the parallelism. The new algorithm consists of four steps:

- First, all samples needed to filter the entire 4×4 block are read using word-aligned instructions. Eight variables are defined to store the 32 pixels needed to filter the block.
- Second, the six conditions used in the algorithm are computed for all the filtered pixels. Four-byte instructions for comparison and subtraction (subabs4 and cmpgtu4) are used for this purpose Finally 1-bit conditions are converted to 8-bit masks using extension instructions (xpnd4), in a similar way as it was reported in [25].
- Third, all possible filter outputs for each pixel are calculated (e.g. $P0X_0$, $P0X_1$, $P0X_2$ and $P0X_3$ are always computed for p0X sample), in spite of only one of them will be used. This is not inefficient because these results are calculated in parallel by different functional units. As the results of several intermediate operations are 16-bit wide, unpacked instructions (unpklu4 and unpkhu4) for pixel variables are used. This way, two 16-bit pixels are computed in parallel. After obtaining all the results, four pixels are packed again in one 32-bit variable (spacku4).
- Fourth, the filter outputs are combined with the masks calculated in the second step to get a single output for each sample (see [25] for details). Finally, six 4-bytes store instructions are used to save the modified pixels.

Fig. 16 summarizes the optimized algorithm. No branches are used and all pixels of a 4×4 block are calculated in parallel. Moreover the number of CPU cycles used to execute the algorithm is constant because of the lack of conditional branches. The same methodology has been used to optimize the luma vertical edges filter and the chroma filters. The described optimization improves the execution speed in about one order of magnitude regarding to the non-optimized version of the filters.

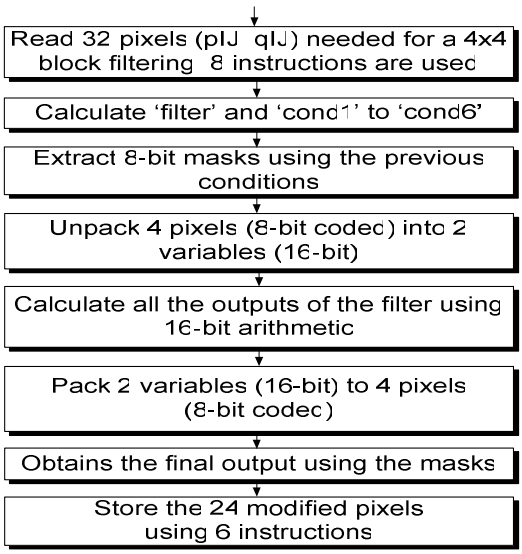


Fig. 16. Optimized algorithm for luma horizontal edge filter.

E. Assembly language.

To improve the execution in speed, several critical modules have been written in assembly language:

- The CABAC core has been optimized, encoded in assembly language and parallelized by hand.
- The IICT and the CAVLC functions have been coded using intrinsic (pseudo-assembler) instructions. In addition, frequent arithmetic operations for Motion Compensation (MC) have also been coded using intrinsic instructions with the same techniques that were described in [17].

V. SIMULATION TESTBENCH

A set of simulation tests has been carried out to verify the decoder and to measure its performance. Actual DVD movies like "Star Wars: episode I" and "Finding Nemo" and a football sequence from a digital TV channel have been used to

generate both BP and MP H.264 test streams[3]. The testbench is shown in **Fig. 17**. First, a test stream is read from a file on a picture basis and written into a stream buffer allocated in external memory. Then, the decoder reads the stream from this memory, decodes it on a picture basis and writes the decoded picture into a buffer. The picture is also written into a file.

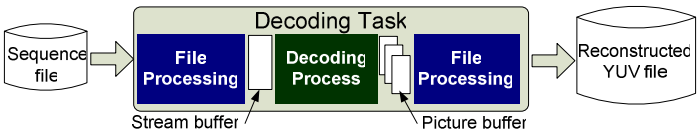


Fig. 17. Testbench used to profile the H-264 decoder in simulation.

Table I contains the profiling results, in average clock cycles per frame, for the decoder and its main parts: CABAC (MP), CAVLC (BP), IICT+MC, deblocking filter and others. The last two rows of Table I show the percentage of CPU load when working at 720 MHz and 600 MHz system clock respectively.

TABLE I
H.264 DECODER PERFORMANCE IN SIMULATION

# cycles×10^6	Nemo		Star Wars		Football		Nemo1M	
	BP	MP	BP	MP	BP	MP	BP	MP
decoder	16.6	22.1	16.0	21.7	16,2	21.5	13,2	20.4
CAVLC/CABAC	4.9	8.2	4.7	7.8	4,9	7.6	3.9	7.0
IICT+MC	5.1	6.7	5.1	6.8	5.1	6.7	4.3	6.0
Deblocking filter	4.2	4.4	3.9	4.2	3,9	4.2	3.2	4.6
others	2.4	2.8	2.3	2.9	2,3	2.9	1,8	2.8
CPU% @720Mz	57.6	76.7	55.6	75.3	56.3	74.7	45.8	70.8
CPU% @600Mz	69.2	92.1	66.6	90.4	67.5	89.6	55.0	85.0

VI. INTEGRATION OF THE H.264 DECODER IN THE IP-STB

The decoder has been integrated into the IP-STB. The BP has been tested using the board shown in Fig. 3, based on the DSP @600 MHz. The testbench can be seen in Fig. 18. A commercial encoder [26] generates the test sequences[4] encapsulated in MP2TS over IP. The board decodes and presents the audio and video information on a TV set. The MP has also been tested with the testbench in Fig. 18, but using a commercial board [27] based on the DSP @720MHz instead of the board shown in Fig. 3.

In Table II, the percentage of CPU load spent by the decoder and by the overall system is given. These data have been measured using an internal DSP timer instead of the profiler. The decoder performance is worst than in simulation because of the interaction with other tasks and the operating system scheduling. This loss is estimated in about 8%.

[3] Length: 100 pictures. Format: 720×576 pels @ 25 fps. Average bit rate: 2Mbps ("Nemo1M" has 1 Mbps). BP: 5% I, 95% P. MP: 4% I, 48% P, 48% B. [online] http://www.sec.upm.es/gdem/en/test_sequences.php.

[4] The video streams have the same average bit-rate and IPB distribution as those used in simulation. The audio streams have been encoded with MPEG-2 layer II. [online] http://www.sec.upm.es/gdem/en/test_sequences.php.

Fig. 18. Testbench used in real-time tests.

TABLE II
H.264 DECODER PERFORMANCE WHEN INTEGRATED IN THE IP-STB

CPU%	BP with DSP @600MHz				MP with DSP @720MHz			
	Nemo	Star Wars	Football	Nemo 1M	Nemo	Star Wars	Football	Nemo 1M
Decoder	79.9	77.5	78.4	65.8	85.7	84.3	83.7	79.8
Total	94.0	91.6	92.5	79.9	99.8	98.4	97.8	93.9

VII. WORK IN PROGRESS

Currently, we are evaluating a new generation DSP [8]. For the sake of simplicity, this new generation DSP is named as DSP-B while the DSP used in previous sections is named as DSP-A. The H.264 decoder has been ported with minimum changes to DSP-B. A testbench has been developed to compare DSP-B with DSP-A in a fair way. Afterwards, several basic optimizations have been made to adapt the decoder to the DSP-B architecture and a testbench have been developed to evaluate them. In the next subsections DSP-B architecture will be outlined for reference, and more details about our work in progress are given.

A. DSP-B Architecture.

DSP-B includes several improvements that may be used to optimize the decoder: new internal memory architecture, new SIMD instructions and a new enhanced DMA controller. Actually, this processor consists of a DSP, a general purpose processor (GPP), several video-oriented processors and several peripherals (see Fig. 19). The DSP is a fixed point VLIW core with 32 KB level-1 memory (L1P) for code and 80 KB level-1 memory (L1D) for data. Both can be configured as cache memories (up to 32 KB) or program/data memories. A 64 KB internal SRAM memory is available also. It can be configured as a level-2 cache and/or an internal data/program memory [28]. The GPP is an ARM9 RISC with 16 KB instruction cache, 8 KB data cache, 16 KB of internal RAM memory and 8 KB of internal ROM memory. The video-oriented processors allow several usual display functionalities as On Screen Display (OSD), image resize, etc. Finally, the set of peripherals includes a DMA processor, video and audio ports. Currently, we are using only the DSP and the peripherals.

B. *Porting the decoder to DSP-B.*

The H.264 decoder has been ported to the DSP-B with a minimum set of changes, needed to adapt the code to the new DMA architecture. Actually, DSP-B has two DMA controllers: an internal DMA (IDMA) and an Enhanced DMA (EDMA). The IDMA transfers data between the internal memories while the EDMA may transfer data between internal and external memory. To increase the efficiency of DMA transfers, we have used the IDMA to program the EDMA registers. Moreover, the new DMA controller allows to link transfers, this means that when a transfer finishes it starts automatically a new preconfigured transfer. This allows, for example, configuring all the transfers (luma and chromas) for a reconstructed MB and start only the first DMA transfer.

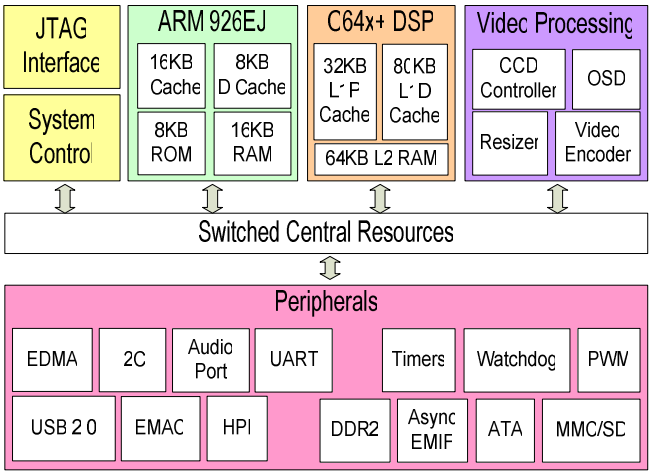


Fig. 19. DSP-B internal architecture.

C. *Comparison between DSP-A and DSP-B*

The testbench shown in **Fig. 17** has also been used to compare both DSPs in simulation. The same video sequences have been used and the same profiles have been obtained for both DSPs. To carry out a fair result comparison, each DSP is configured as follows. First, L1D and L1P cache memory sizes are set to 16 KB. Secondly, L2 cache memory size is set to 64 KB and lastly, program functions are stored in external memory. In contrast, intermediate buffers of DSP-B are stored in L1 SRAM while those of DSP-A are put in L2 SRAM.

Table III contains the profiling results for the DSP-A[5]. Results are given in average clock cycles per frame for the full decoder and its main functional blocks (CABAC, CAVLC, IICT+MC, deblocking filter and others). Table IV presents the same information for DSP-B. These results show that DSP-B improves DSP-A in terms of speed in more than 20%.

[5] Data presented in Table I and Table III are quite different because in simulation tests (Table I) the most CPU-demanding functions are allocated in internal memory while in real-time tests (Table III) all the functions are allocated in external memory.

TABLE III
DECODER PERFORMANCE FOR DSP-A.

# cycles×10^6	Nemo		Star Wars		Football	
	BP	MP	BP	MP	BP	MP
decoder	41.4	56.9	39.7	58.8	40.6	58.9
CAVLC/CABAC	13.6	21.7	13.2	22.4	13.7	21.7
IICT+MC	13.2	17.2	13.1	18.1	13.2	18.7
Deblocking filter	10.6	12.7	9.3	12.8	9.6	12.9
others	3.8	5.3	4.1	5.4	4.1	5.6

TABLE IV
DECODER PERFORMANCE FOR DSP-B.

# cycles×10^6	Nemo		Star Wars		Football	
	BP	MP	BP	MP	BP	MP
decoder	31.8	45.5	30.3	45.2	30.4	45.0
CAVLC/CABAC	11.3	16.8	10.8	16.4	11.0	16.1
IICT+MC	7.8	11.1	7.6	11.4	7.5	11.4
Deblocking filter	9.5	11.7	8.5	11.5	8.5	11.3
others	3.2	5.9	3.4	5.9	3.4	6.2

D. *Basic optimizations for DSP-B.*

Basic optimizations have been performed for DSP-B in order to increment the execution speed. Specifically several code and data sections have been moved to the internal DSP memory, L1P cache is configured to 16 KB and both, L1D and L2 caches, are configured to 32 KB. With these changes, the DSP performance has been re-evaluated with the results shown in Table V. These results can be compared with the ones given in Table I. In spite of the results given in subsection C, DSP-B performs slower than DSP-A. The lower amount of L2 memory seems to be the reason of this performance degradation.

TABLE V
PERFORMANCE FOR DSP-B INCLUDING BASIC OPTIMIZATIONS.

# cycles×10^6	Nemo		Star Wars		Football	
	BP	MP	BP	MP	BP	MP
decoder	23.0	33.1	22.0	33.0	22.1	31.8
CAVLC/CABAC	8.4	14.1	8.1	13.8	8.2	13.4
IICT+MC	5.5	8.0	5.4	8.1	5.3	8.0
Deblocking filter	6.8	7.7	6.2	7.7	6.2	7.2
others	2.3	3.3	2.3	3.4	2.4	3.2

VIII. CONCLUSION AND FUTURE WORK

In this paper, the implementation of an H.264 decoder on a low-cost DSP [5] and its integration on a multi-format IP-STB have been shown. Tests in a real environment show that real-time can be achieved for BP@L3 with a 600 MHz system clock. Real-time performance for MP@L3 requires a 720 MHz system clock. Currently, we are working on the evaluation of a new DSP [8]. We have ported the decoder to the DSP and we have developed testbenches to compare both DSPs. Our future work will be focused on the improvement of the optimization process for the new DSP.

ACKNOWLEDGMENT

The authors would like to thank Gonzalo Maturana from E.U.I.T.T.-UPM and Rafael Antoniello from SIDSA for their contributions to this work.

REFERENCES

[1] F.T.H. den Hartog *et al*. "Convergence of Residential Gateway Technology: Analysis of Evolutionary Paths", *IEEE Consumer Communications and Networking Conference*, pp.1-6, Jan. 2004.

[2] W. T. Ng and H. A. Chan. "Streaming multimedia content over home network with an intelligent controller". *The 29th Annual Conf. of the IEEE Industrial Electronics Society*, Vol. 2, pp. 1802-1807, Nov. 2003.

[3] C. Luo *et al*. " Design and implementation of multiplexing rate control in broadband access network TV transmission system", *IEEE Trans. on Consumer Electronics*, Vol. 50, Issue 3, pp. 849-855, Aug. 2004.

[4] Q. Peng and J. Jing. "System-on-chip design for TV-centric home networks". *IEEE Consumer Communications and Networking Conference*, pp. 501-506, Jan. 2004.

[5] Texas Instruments. TMS320DM642 Video/Imaging Fixed-Point Digital Signal Processor. Available online at: http://focus.ti.com/docs/prod/folders/print/tms320dm642.html.

[6] Philips Semiconductors. Nexperia Media Processors. Available online at http://www.nxp.com/products/nexperia/home/products/media_processors/index.html

[7] Analog Devices. Blackfin processors. Available online at: http://www.analog.com/processors/blackfin/index.html.

[8] Texas Instruments. TMS320DM6446 DaVinci Digital Media System-on-Chip. Available online at: http://focus.ti.com/docs/prod/folders/print/tms320dm6446.html.

[9] Y.-S. Tung *et al*. "DSP-Based Multi-Format Video Decoding Engine for Media Adapter Applications". *IEEE Transactions on Consumer Electronics*, Vol. 51, Issue 1, pp. 273-280, Feb. 2005.

[10] ISO114496-10. Information technology. Coding of audio-visual objects. Part 10: Advanced Video Coding (Dec. 2005).

[11] J. Ostermann *et al*. "Video coding with H.264/AVC: tools, performance, and complexity". *IEEE Circuits and Systems Magazine,* Vol. 4, Issue 1, pp. 7-28, 2004.

[12] M. Horowitz *et al*. "H.264/AVC baseline profile decoder complexity analysis*". IEEE Transaction on Circuits and Systems for Video Technology*. Vol 13. No 7. Jul. 2003.

[13] V. Ramadurai *et al*. "Implementation of H.264 decoder on Sandblaster DSP". *IEEE International Conference on Multimedia and Expo*, 2005. ICME 2005, Jul. 2005.

[14] Yi-Shin *et al*. "DSP-based multi-format video decoding engine for media adapter applications". *IEEE Trans on Consumer Electronics,* pp. 139-140, Jan. 2005.

[15] S-W Wang *et al*. "The optimization of H.264/AVC baseline decoder on low-cost TriMedia DSP processor". *Applications of Digital Image Processing XXVII*. Vol 5558, pp. 524-535.

[16] Y. Moshe and N. Peleg. "Implementations of H.264/AVC baseline decoder on different digital signal processors". ELMAR, 2005. 47th International Symposium. Issue, 8-10 June 2005, pp. 37-40.

[17] F. Pescador *et al*. "A DSP based IP set-top box for home entertainment". *IEEE Trans on Consumer Electronics* Volume 52. Issue 1. Feb. 2006, pp. 254-262.

[18] F. Pescador *et al*. "MPEG-4 SP/ASP decoder for a DSP based Multi Format IP Set-Top Box". *Annual Conference of the IEEE Industrial Electronics Society (IECON06),* Paris, pp. 3397-3402, Nov. 2006.

[19] F. Pescador *et al*. "An MPEG2 TS Parser for a DSP based Multi-format IP Set-top Box". *XXII Conference of Design of Circuits and Integrated Systems,* pp. 88-93, Nov. 2007.

[20] Texas Instruments. "TMS320 DSP-BIOS User's guide" (SPRU303B – May. 2000). http://focus.ti.com/lit/ug/spru303b/spru303b.pdf.

[21] Texas Instruments. "Reference Frameworks for eXpressDSP Software: RF5, Extensive High-Density System" (SPRA795A –April 2003). http://focus.ti.com/lit/an/spra795a/spra795a.pdf

[22] I.E.G. Richardson. "H.264 and MPEG-4 video compression". 1st ed. Chichester (West Sussex): Wiley. 2003.

[23] T. Wiegand *et al*. "Overview of the H.264/AVC Video Coding Standard." *IEEE Transaction on Circuits and Systems for Video Technology*. Vol. 13. no. 7. Jul. 2003.

[24] N. Kehtarnavaz and M. Gamadia. "Real-Time Image and Video Processing: From Research to Reality" Ed. Morgan & Claypool Publishers. 2006, pp. 55-71.

[25] Z. Yang *et al*. "Deeply pipelined DSP solution to deblocking filter for H. 264/AVC". *IEEE Transactions on Consumer Electronics* 2006, pp. 1267-1274.

[26] ATEME. "AMK 430 AVC encoder". http://www.ateme.com.

[27] DM642 Evaluation Module with TVP Video Decoders, 720 MHz. http://www.spectrumdigital.com/product_info.php?&products_id=121.

[28] Texas Instruments. TMS320C64x to TMS320C64x+ CPU Migration Guide. Available online at: http://www.ti.com/litv/pdf/spraa84a.

Fernando Pescador (M'07) received the Ingeniero Técnico de Telecomunicación degree in 1992 and the Ingeniero de Telecomunicación degree in 2001, both from the Universidad Politécnica de Madrid (UPM), Spain. He is currently a Ph.D. candidate in the Electronic Engineering Department at the same university. He is Associate Lecturer at the Department of Electronic and Control Systems at E.U.I.T. de Telecomunicación of the UPM since 1995 and researcher of the Electronic and Microelectronic Design Group (GDEM) since 1999. His research interests are real time video coding and digital video broadcasting.

César Sanz (S'87. M'88) received the Ingeniero de Telecomunicación degree with honours in 1989 and the Doctor Ingeniero de Telecomunicación degree with summa cum laude in 1998 both from the Universidad Politécnica de Madrid (UPM). Since 1984 he has been a member of the faculty of the E.U.I.T. de Telecomunicación of the UPM and since 1999 has been Associate Professor at the Department of Electronic and Control Systems. In addition, he leads the Electronic and Microelectronic Design Group (GDEM) involved in R&D projects. His areas of interest are microelectronic design applied to image coding, digital TV and digital video broadcasting.

Matías J. Garrido received the Ingeniero Técnico de Telecomunicación degree in 1986, the Ingeniero de Telecomunicación degree in 1996 and the Doctor Ingeniero de Telecomunicación degree with summa cum laude in 2004, all from the Universidad Politécnica de Madrid (UPM). Since 1986 he has been a member of the faculty of the E.U.I.T. de Telecomunicación of the UPM and since 1987 he has been Associate Professor at the Department of Electronic and Control Systems. He is a founder member (1996) of the Electronic and Microelectronic Design Group (GDEM) involved in R&D projects. His areas of interest are electronic digital design, video coding and digital video broadcasting.

Eduardo Juárez (M'96) received the Ingeniero de Telecomunicación degree from the Universidad Politécnica de Madrid (UPM), Madrid, Spain, in 1993 and the Docteur ès Sciences Techniques degree from the École Polytechnique Fédéral de Lausanne (EPFL), Lausanne, Switzerland, in 2003. In 1994, he joined the Digital Architecture Group (GAD) of the UPM as a researcher. In 1998, he joined the Integrated Systems Laboratory (LSI) of the EPFL as an Assistant. In 2000, he joined Transwitch Corp., Switzerland, as Senior System Engineer. In 2004, he joined the Electronic and Microelectronic Design Group (GDEM) as a post-doctoral researcher. In 2007, he joined the faculty of the E.U.I.T. de Telecomunicación of the UPM. His current interests are in the design of low-power video and audio decoders for mobile applications.

David Samper received the Ingeniero Técnico de Telecomunicación degree in 2005 from the Universidad Politécnica de Madrid (UPM), Spain. Currently he is studying Ingeniero de Telecomunicación degree at the same university. He joined the Electronic and Microelectronic Design Group (GDEM) as a scholar in July 2004 until September 2007, when he was taken on as researcher. In December 2005 he was awarded the "Liberalización de las Telecomunicaciones 2005" prize from the COITT, as well as the Best Diploma Project and Best Statewide Student's Record of the 2004-2005 School Year in the Electronic Systems. His research interests are real time video coding and digital video broadcasting.