HARDWARE / SOFTWARE CO-DESIGN OF A SIMPLE RISC MICROPROCESSOR FOR DIGITAL SET-TOP-BOX APPLICATIONS [*]

Marco Antonio Simon Dal Póz (mdalpoz@lsi.usp.br), José Edinson Aedo Cobo (aedo@lsi.usp.br), Wilhelmus Adrianus Maria Van Noije (noije@lsi.usp.br), Marcelo Knörich Zuffo (mkzuffo@lsi.usp.br)

Laboratório de Sistemas Integráveis (LSI)
Departamento de Engenharia Eletrônica – Escola Politécnica da Universidade de São Paulo (USP)
Av. Prof. Luciano Gualberto, 158, travessa 3, São Paulo - SP

ABSTRACT

We propose the definition and evaluation of an instruction set designed and tuned for multimedia applications on a Digital Set-Top-Box. The proposed instruction set had its performance evaluated in software and hardware to obtain the best cost / benefit relationship referring to performance and silicon chip area. An instruction set was obtained enhancing the performance of iDCT algorithms to achieve the needs of real time MPEG-2 video decompression and to have an extra processing power available for future more complex algorithms (e.g., MPEG-4). A RISC basic core was modeled in VHDL and the defined instruction set was added into this core. In this way, the evaluations were made through out logical simulations by implementing over FPGAs, and the results of the added instructions over the algorithm performance were evaluated using high-level synthesis tools and in-field tests.

1. INTRODUCTION

A Set-Top-Box is the interface between cable TV network and the TV set. The presence of a Set-Top-Box in the subscriber's home becomes necessary for the introduction of new services through the cable TV network, such as high definition television (HDTV), Internet access using a simple TV set, electronic commerce, tele-medicine and others.

One of the main focuses of the Digital Set-Top-Box is the possibility of digital transmission of the TV channels, which implies in higher video and audio quality and less spectral occupancy. With less spectral occupancy, the cable TV provider can offer a higher number of channels and other services. This reduction in spectral occupancy (band) is due to video compression algorithms, such as MPEG-1 [15], MPEG-2 [16] and all derivatives.

The American Standard for digital transmission (ATSC) [1] and the European Standard (DVB) [2] use the same algorithm for video compression (MPEG-2). This algorithm is essentially composed by three parts:

- Discrete cosine transform (DCT) inter/intra frame;
- Quantization of DCT coefficients, according to tabulated rules;
- Statistical data compression (Huffman encoding and Run-Length-Encoding).

The resulting data is packed into an MPEG-2 stream and broadcasted over the cable network. The Set-Top-Box, for enabling the subscribers to watch the digital channels, should run exactly the inverse of the above operations. This operation is called video decompression, which can be hardware, software or hard/software implemented (mixed).

It is known that the most complex of the above operations for video decompression is the inverse DCT (iDCT), which requires special attention in the digital design, because the big amount of arithmetic operations. In this paper, we present a microprocessor core design and implementation that can run efficiently an iDCT software and is able to easily introduce additional instructions for future reconfigurable algorithms and performance improvement.

2. DIGITAL SET-TOP-BOX PERFORMANCE VS UPGRADEABILITY

The most part of the current Digital Set-Top-Box available works with MPEG-2 data streams. Some of them have an internal MPEG-2 chip decoder (hardware ASIC implementation), and others use a fast processor running an MPEG-2 software decoder. The hardware implementation usually has the best performance, which means that the decoder operates with low clock frequencies, low power consumption and small chip area, obtaining a low cost decoder, but has no upgrade capability. In this way, when the cable TV providers adopt the MPEG-4 (or better) video compression standard, all digital Set-Top-Boxes based on MPEG-2 hardware decoder will not still be compatible with the new contents being broadcasted through the network. So, a new hardware investment will be necessary to exchange these digital Set-Top-Boxes from the subscribers, which represents a high cost (for the provider or for the subscribers). The software implementation usually has poor performance, which means that the decoder demands a fast processor (higher power consumption) and larger chip area, obtaining a high cost decoder, but has the advantage of being upgradeable.

But the upgrade capability of the software implementation is limited. Since the complexity of video compression and

[*] This project is funded by Fapesp grant #99/01097-9 and NEC do Brasil / CPDIA Project: Multimedia Applications for Digital Services Networks

decompression algorithms is growing, the performance of the software tends to decrease. This effect usually is reduced by the adoption of a very fast processor, which implies in a high cost digital Set-Top-Box. To improve the performance of the software, the processor should have specific instructions designed for running the new software decoders. But, since a processor is implemented in an ASIC, further modifications are impossible without hardware exchange, which is also too much expensive.

3. RECONFIGURABLE DIGITAL SET-TOP-BOX

The solution for this problem of performance *versus* cost can be obtained with the reconfigurable computing. Reconfigurable computers have part of the hardware implemented in FPGAs (Field Programmable Gate Arrays), which implies that parts of the hardware can be modified after installed in the subscriber's home and while it is working. If the Digital Set-Top-Box uses at least one FPGA in the hardware design, the internal logical functions can be modified *a posteriori*.

The solution being proposed here is to implement a simple RISC core microprocessor in a fast FPGA and, when the cable TV provider decides to change the video compression method, a new core can be sent and written in the FPGA, producing an upgrade in all the Set-Top-Boxes working within that cable TV network. With this reconfigurable capability, new instructions can be implemented (in the same equipment) in the processor, and so the performance of the software will be much better than the pure software upgrade.

The implementation of a processor in an FPGA can become practical if the following requirements are satisfied:

- The FPGA should be capable of working in high clock frequencies. This is necessary because the performance of a digital circuit implemented in an FPGA is usually worse than the one implemented in an ASIC;
- The FPGA should have a large number of gates. This is required to allow several future modifications in the digital design;
- The FPGA should not be very expensive. Of course the FPGA implementation of a processor will be more expensive than an ASIC, because to achieve similar performance, FPGAs will require better microelectronics technology (smaller dimensions).

The problem of the high cost of such an FPGA can be overcame if we consider the high useful lifetime that this kind of implementation may have, supporting many types of future video compression techniques. Another way of cost reduction is the large utilization of FPGAs, which can reduce their price.

4. TECHNIQUES AND TOOLS FOR DEDICATED MICROPROCESSOR DESIGN

Several works have been done in the design field of general-purpose processor. There are many alternatives for ASIP (Application Specific Instructions Processor) design. One methodology consists in the interactive design of the instruction set and the appropriate compiler for the processor. Examples of these tools are “CHESS” and “CHECKER” [14], a configurable compiler for C language and a simulator generator oriented for ASIP design in digital signal processing field, respectively. These tools use a language called nMLx [6] to describe the instruction set semantics and micro-architecture details. The design process begins assuming one data path that can be based on operation types analysis and the most frequent operation sequences. Then the application is mapped onto the data path. After that, statistical informations can be obtained about the application's data path use. With this information, changes are suggested in the data path, for example, new connections between elements of the data path. After the implementation of these changes, the application is mapped again and a new analysis is done. This procedure is repeated until the design objectives are reached.

On the other side, Sato [7] proposes a new methodology in which the instruction set is selected from a “super-set” that is derived from the GCC’s intermediate language (GNU GCC). The instruction set is generated by “super-set” modification based on statistical information obtained from a “profiling” process done by *benchmark* programs. This methodology uses a sequential model of the processor and the design space is limited to the GCC’s instruction set.

Holmer [8] proposes a method to automatically derive the instruction set based on a data path architectural pattern and on a *benchmark* set. The method consists in transforming the *benchmarks* in a state transition set, where each sequence represents shorts sequences of the *benchmark* code. After that, the optimum instruction set is determined for each state transition, and the final instruction set is defined by means of required instructions coverage to run all pairs of *benchmark* state. This methodology uses one parametrizable data path in the synthesis process. The data bus and address bus length, the number and class of registers, the operations that can be done with the data (add, mul, etc.) are pre-determined by the designer. The number of operands referred by the instruction, the placement of this operands and the way to reference this operands (addressing modes) are determined by the tool. Data path parameters such as number of read ports and write ports of the register bank, memory ports, number of functional lines, number of cycles for the memory operations are determined from the data path architectural pattern. Huang [9] had developed a technique of co-synthesis of the instruction set and micro-architecture. The instruction set model, the architectural pattern and the *pipelining* model are previously specified by the designer. By means of another tool called “Piper”, the control part and the data path are projected. This tool generates a RTL description and a rearranging table to be used by the compiler. In these works, the authors have treated the instruction set design problem and the instructions selection like a *scheduling* or modules selection problem. There should be noticed that *scheduling* and modules selection techniques can not generate new logical resources in the solution. Another technique in which the generation of new resources is possible is reported in [10]. In such a case, the compiler extracts the functionality of these resources using hardware description language, which are implemented in an FPGA connected like a co-processor in a main processor, running the application in a cooperative way. One problem in this technique is the overhead introduced between the processor and FPGA communication.

Another methodology is Satsuki [11] system. This system uses a general-purpose architecture (*Harvard* type) with a invariant instruction set. The buses length of the processor and of the registers bank can be modified in such a way that these parameters are selected to achieve the user requirements for a specific application. The configurable processor used in this methodology is called ASAP ("Application Specific Adaptable Processor"). The environment is constituted by a tool that generates the compiler and has a high-level synthesis tool. In this way it is possible to compile the applications and synthesize the ASIP in a port level. The limitation introduced by this method is the restriction of the base micro-architecture, because the control part is constituted by a global state machine. This global state machine becomes too difficulty due to the introduction of new instructions.

There should be noticed that some of the mentioned methods do not consider the possible gains introduced by the use of optimized data structures or when some strategies like sub-word level parallelism are used, that is efficiently employed in multimedia applications [12]. In these cases, it is necessary to introduce manual changes. By this reason, a configurable architecture, modeled in a convenient way using a hardware description language, simplifies considerably the changes in the architecture and in the instruction set. In the same way, it allows a fast implementation using high-level synthesis tools, because the modeling is done using synthesizable constructions. Also, the changes impact in clock cycle and area can be measured exactly after the high-level synthesis process.

5. PROCESSOR AND SOFTWARE CO-DESIGN

Since the iDCT operation is the most complex of the video decompression operations, the iDCT algorithm should be optimized as best as possible. This is the main rule for this processor and software design techniques. Three algorithms were analyzed: Feig & Winograd [3], LLM [4] and AAN [5]. The main characteristics of these algorithms are listed in table 1.

Table 1: number of arithmetical operations for each algorithm

Algorithm	Adds per 8 element 1D DCT	Muls per 8 element 1D DCT	Adds per 8x8 elem. 2D DCT	Muls per 8x8 elem. 2D DCT
Feig&Wino	N/A	N/A	454	94
LLM	28	11	448	176
AAN	29	5	464	144

For area reduction, implementations using floating point should be avoided. Using fixed point arithmetic instructions, the main hardware optimization consists in adding an instruction MAC (multiply and accumulate) to the processor design. This processor design is based on the instruction set described in Patterson [13] and was modeled entirely in synthesizable VHDL using generic directives. The control part was created in a modular structure in order to ease the insertion and removal of new instructions. Synopsys^{TM} Design Compiler was used to synthesize the processor code and Synopsys VSS^{TM} was used to simulate the project.

The two-dimensional iDCT algorithm, despite the minor number of multiplications, requires more complex instruction set, which implies in higher area consumption, which is not desired at all. Beside, it also requires a larger constants bank to be stored, which requires more area, too. Therefore, the choice was the AAN algorithm, using 16 bit multiplications and 32 bit accumulations (sums).

To decode a standard definition television (SDTV), let's calculate the maximum time for a 8x8 pixels decoding. The resolution is 640 x 480 pixels, with a frame rate of 30 fps. So, for each frame there are 4800 iDCTs. Assuming that the others operations (Huffman decoding, RLE and data realignment) have the same computational cost as the iDCT (which was confirmed by doing software tests using a standard MPEG-2 with source code available), the maximum time for a complete iDCT results 3.472µs (1.736µs for luminance data and 1.736µs for chrominance data).

Without these additional instructions, the processor should get the 8x8 pixel data from memory, run the calculus and write the results into the memory in 1.736µs. Since the number of arithmetic operations is 608 and 64 words should be read and written, a 16-bit processor would require a maximum period of 2.855ns, or a 350MHz clock. An FPGA capable of running 350MHz and with enough area to implement an entire processor has actually a prohibitive cost (if it exists). Of course more intelligent approaches can be designed. Adding the MAC instruction and a 32-bit processor, the number of arithmetic operations is 608 and 32 words should be read and written. But the iDCT data have 16 bits of data, which implies that the MAC instruction, using 32 bit data bus, can handle 2 multiplications and sums at a time, reducing the number of arithmetic operations to 304. But the MAC instruction does 2 multiplications and 2 sums, which can handle 144 multiplications and 144 sums in 72 instructions. The others sums can be made with 320 standard sum operations, resulting in 392 processor instructions. The processor would require a maximum period of 4.429ns, or a 226MHz clock. An FPGA capable of running 230MHz still have a prohibitive cost. To reduce the need for high speed processor, more specific instructions should be implemented. Since the MAC instruction operates with 16 bits of data, two data at the same time, one natural instruction required to enhance the performance is the HADD instruction, which sums simultaneously 2 words of 16 bits. This reduces the amount of ADD instructions to 160 HADD instructions, resulting in 232 processor instructions. The processor would require a maximum period of 7.483ns, or a 134MHz clock. An FPGA capable of running 150MHz with enough area has actually a reasonable cost and is suitable for commercial applications. Of course, more intelligent instructions can be added, and the performance results will be discussed later.

The RISC processor implemented has 32 registers (R0 to R31) of general use. To simplify the software design, a constants bank (16 constants, K0 to K15) was added to the data path. This reduces the data transferred from the memory to the registers with a small increase in area occupation. These constants contain the iDCT coefficients that are used in the multiplications, cossines multiplied by 2^{15} (in this case, by the MAC operations). Four kinds of MAC instructions were created:

- MAC Ri, Rj, Rk : this instruction (multiply and accumulate) multiplies the high word (16 bits) of Rj with the high word of Rk, sums the result with the high word of Ri and stores the result in the high word of Ri; simultaneously multiplies the low word of Rj with the low word of Rk, sums the result with the low word of Ri and stores the result with the low word of Ri
- MACL Ri, Rj, Rk : this instruction (multiply and accumulate, previously cleaning Ri) multiplies the high word of Rj with the high word of Rk and stores the result in the high word of Ri; simultaneously multiplies the low word of Rj with the low word of Rk and stores the result in the low word of Ri
- MACK Ri, Rj, Kk : this instruction (multiply and accumulate using constant register) multiplies the high word of Rj with the high word of constant Kk, sums the result with the high word of Ri and stores the result in the high word of Ri; simultaneously multiplies the low word of Rj with the low word of constant Kk, sums the result with the low word of Ri and stores the result with the low word of Ri
- MACKL Ri, Rj, Kk : this instruction (multiply and accumulate using constant register, previously cleaning Ri) multiplies the high word of Rj with the high word of constant Kk and stores the result in the high word of Ri; simultaneously multiplies the low word of Rj with the low word of constant Kk and stores the result in the low word of Ri

As shown in design requirements (minimum speed for iDCT), instructions for 2 parallel 16 bits sums were created [12]:

- HADD Ri, Rj, Rk : this instruction (half add) adds the high word of Rj with the high word of Rk and stores the result in the high word of Ri; simultaneously adds the low word of Rj with the low word of Rk and stores the result in the high word of Ri
- HADDAC Ri, Rj, Rk : this instruction (half add and accumulate)adds the high word of Rj with the high word of Rk, sums the result with the high word of Ri and stores the result in the high word of Ri; simultaneously adds the low word of Rj with the low word of Rk, sums the result with the low word of Ri and stores the result in the low word of Ri
- HADDF Ri, Rj, Rk : this instruction (half add followed by a full adder) adds the high word of Rj with the high word of Rk, simultaneously sums the low word of Rj with the low word of Rk, and sums both results and stores the final result in Ri (32 bits datum)
- HADDFAC Ri, Rj, Rk : this instruction (half add and accumulate, followed by a full adder) adds the high word of Rj with the high word of Rk, simultaneously sums the low word of Rj with the low word of Rk, and sums both results with the double-word of Ri (32 bits datum) and stores the final result in Ri.

The iDCT software has to be made in Assembly, because it is a critical operation and should be optimized at maximum. The results of the MAC operations showed that the MAC instruction lasts 68.4ns for 32 bit data bus (2 multiplications simultaneously and 2 sums). Again, we adopted the technique of pipelining to divide the MAC instruction into 8 stages. In principle, 7 stages would be enough, but to allow an additional clock increase, 8 stages were adopted. Since the RISC core has already a 4-stage pipeline, the MAC instructions have a latency of 11 clock cycles. The software design should take care of this MAC latency and use the registers efficiently, avoiding pipe stalls. The software design resumes to create an adequate sequence of the Load, Store and Arithmetic operations that does not create direct dependence of adjacent instructions.

A simulation of the processor can be found in fig. 1, where it is shown the control signals and some instructions of the basic core being tested (time scale in ns, two-phase non-overlaping clock).

These instructions with different number of pipeline stages created an extra complexity in the control part of the processor, because there are possible combinations of subsequent instructions that try to write simultaneously in the register bank. A controller that can handle this situations was modeled, and new simulations were made. The result has shown that the critical path had became the controller, which lasts 15.6ns. This fact is a critical limitation in processor performance, and the adopted solution is to use the simplest version of the controller, that can't handle this situation. This possibility should be avoided by the compiler or by the programmer.

6. PERFORMANCE ANALYSIS

The basic RISC core (without multiply instructions), implemented in VHDL, was compiled and simulated using Synopsys™ Design Compiler and Synopsys™ VSS with Altera™ libraries for Flex10k FPGA. The FPGA actually available for this work is FLEX10K70RC240-4. The simulations have shown that the critical path for the entire processor is the ADD instruction, which lasts 30.2ns. To achieve the maximum period of 11.408ns, the instruction was subdivided into 3 stages in a pipeline scheme. This implies that, if subsequent instructions depends on the previous one, 2 clock cycles will be lost. Since iDCT calculations can be arranged to avoid this situation, this restriction is not going to be a problem.

Others simulations involving the HADD instruction have shown that it lasts 23.3ns, which is less than the ADD instruction, but it also had to be divided into 3 stages in the same pipeline scheme. The same restrictions of the ADD apply to HADD.

The same procedure was used to evaluate the latency of the HADDAC instruction, which does the double of the sums of the HADD instruction. It lasts 49.7ns, and was divided into 5 stages in the same pipeline.

The iDCT software has to be made in Assembly, because it is a critical operation and should be optimized at maximum. The results of the MAC operations showed that the MAC instruction lasts 68.4ns for 32 bit data bus (2 multiplications simultaneously

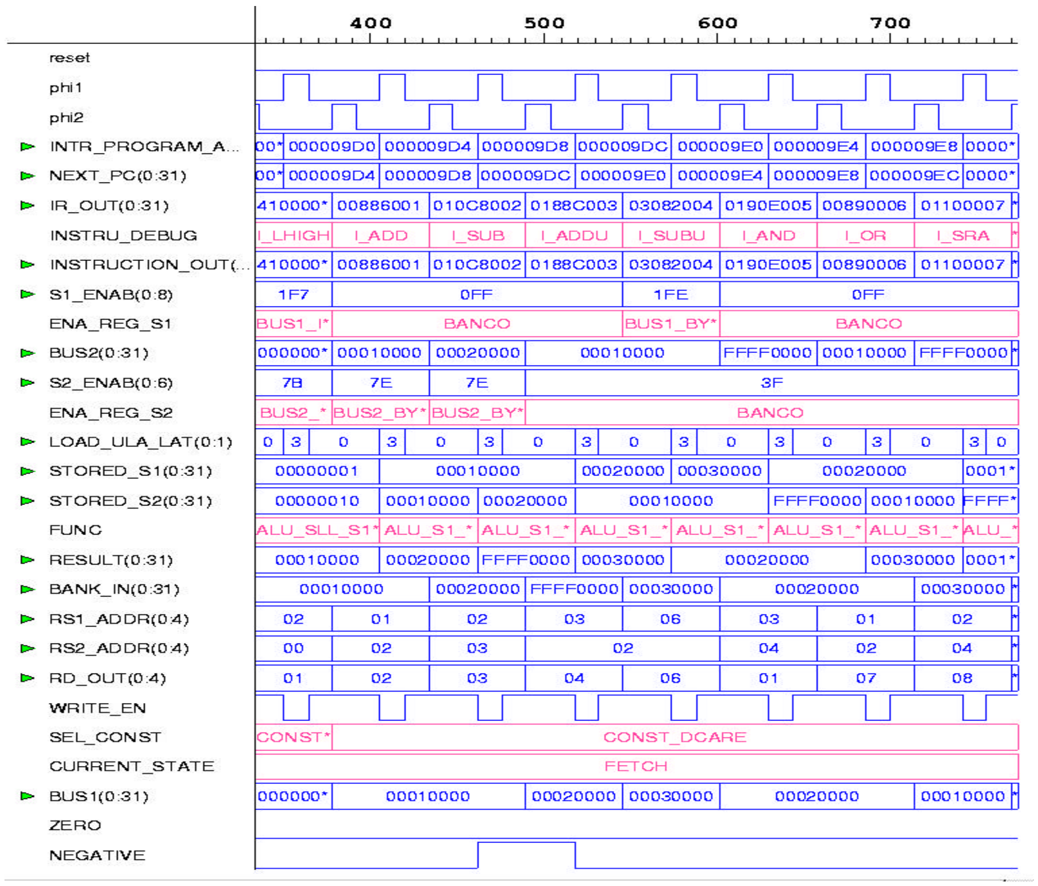

Fig. 1 – Simulation of the RISC core modeled in VHDL (vhdldbx)

and 2 sums). Again, we adopted the technique of pipelining to divide the MAC instruction into 8 stages. In principle, 7 stages would be enough, but to allow an additional clock increase, 8 stages were adopted. Since the RISC core has already a 4-stage pipeline, the MAC instructions have a latency of 11 clock cycles. The software design should take care of this MAC latency and use the registers efficiently, avoiding pipe stalls. The software design resumes to create an adequate sequence of the Load, Store and Arithmetic operations that does not create direct dependence of adjacent instructions.

A simulation of the processor can be found in fig. 1, where it is shown the control signals and some instructions of the basic core being tested (time scale in ns, two-phase non-overlaping clock).

These instructions with different number of pipeline stages created an extra complexity in the control part of the processor, because there are possible combinations of subsequent instructions that try to write simultaneously in the register bank. A controller that can handle this situations was modeled, and new simulations were made. The result has shown that the critical path had became the controller, which lasts 15.6ns. This fact is a critical limitation in processor performance, and the adopted solution is to use the simplest version of the controller, that can't handle this situation. This possibility should be avoided by the compiler or by the programmer.

7. HIGH DEFINITION TELEVISION REQUIREMENTS

As the HDTV resolution (1920 x 1080 pixels) is 6.75 times larger than the standard definition TV (640 x 480 pixels) [16], the iDCT time must be 6.75 times smaller, which implies that the developed processor should be 6.75 times faster, which is impossible using actual FPGAs. Then, the approach used in this Hardware/Software Co-Design cannot be applied to HDTV MPEG-2 decoding. As in the SDTV iDCT the maximum time is 1.736us, we have that for HDTV the maximum is 257.2ns. Using the FPGA available for this work (Altera FLEX10KRC240-4), such performance is impossible. So, the Co-Design of a system capable of HDTV decoding was made based only on simulation. Another approach was used: instead of implementing the iDCT basic operations (MAC) into specific instructions, the entire unidimensional 8-point iDCT core was modeled in VHDL (using 24 bit fixed point arithmetic). This implementation occupies a lot of chip area, and some simulations were made using as target device Xilinx XCV1000E-680C-8 (from Virtex Enhanced Family of FPGAs). Due to the current unavailability of specific libraries for Synopsys™ Design Compiler, Xilinx Foundation Express 2.1i was used.

The IDCT instruction was added to the RISC core, and was designed within the following methodologies: the AAN algorithm was expanded into a big set of sums and shifts; the shift operations become simply displaced inputs to the adders (which was no computational cost, instead of in the software implementation); constant multiplications are expanded into customized adders (with shifted inputs in accordance with the constant values); and flip-flops are inserted in strategic positions of the iDCT core to provide multiple stages division (pipelining).

This IDCT instruction uses 4 registers (eight 16-bit inputs), and writes the results to the same registers (but the internal calculations of the core are made using 24 bits). This implementation avoided the use of specific registers because it would be necessary to use more instructions to move data to these registers, and more processing time would be spent just to move data. Again, control systems to avoid out-of-order instruction execution were not created, leaving this task to the compiler or to the programmer.

The results shown an FPGA occupation of 731 slices, which means more than 93000 gates, which demands the use of extremely large FPGAs; the achived time is 51.18ns for the unidimensional iDCT. Then, to complete the 8x8 iDCT it is necessary to run 16 IDCT instructions in sequence, resulting in a time of 818.9ns. As the required time for the iDCT is 252.7ns, the division of the iDCT core in 6 stages (to achieve a 100MHz FPGA clock) was enough. Multiple iDCTs cores implementation was discarded due to the huge area occupation, enabling the use of smaller Virtex FPGAs.

8. FUTURE WORK

There are several improvements that can be done. The first one is to implement and test in faster FPGAs (like Xilinx Virtex and Altera Apex). This will allow us to run higher clock frequencies. The second is to expand the bus to 64-bit and add more registers. This can easily be done, because the VHDL processor code was made using extensively the generic directives, which determines the entire bus size. The third is to buy FPGAs with a great number of pins, which will allow the physical implementation of a 64-bit RISC core.

One of the goals of this project is to achieve data rates necessary to decode High Definition Television (HDTV), which is possible with adoption of an special instruction that demands much more area than all the remaining portion of the processor: the IDCT instruction, which is a severe limitation to the possible FPGAs to be used in this design. To reduce this limitation, we intend to work on a super-scalar version of the microprocessor core, and create more variants of the MAC instruction, to enable the processor to run several multiplications and accumulations in only one clock cycle.

Other related works can be the implementation of a MPEG-4 software decoder using this RISC core processor and maybe the addition of new instructions designed to accelerate MPEG-4 decoding.

9. CONCLUSIONS

MPEG video can also be one field of application of reconfigurable computing by using Hardware/Software Co-Design. In this work, we are exploring the reconfigurable computing resources in the digital video field. The achieved results comply with the needs, and can easily be implemented to work with Digital SDTV. To operate with HDTV, the technique presented in this paper will be improved, using faster FPGAs and larger data bus. The main contribution of this research is to allow the construction of very long lifetime digital Set-Top-Box, which can represent a reasonable economy for cable TV providers and subscribers. New researches are going to be done to implement and test the designed RISC core in huge and fast FPGAs capable of handle with HDTV data in real time.

10. REFERENCES

[1] Advanced Television Systems Committee (ATSC) standards (http://www.atsc.org)

[2] Digital Video Broadcasting (DVB) standards (http://www.dvb.org)

[3] E. Feig and S. Winograd. “Fast Algorithms for the Discrete Cosine Transform”. *IEEE Transactions on Signal Processing, vol. 40, no. 9,* pages 2174-2193, Sep. 1992

[4] C. Loeffer, A. Ligtenberg, and G. S. Moschytz “Practical fast 1D DCT algorithms with 11 multiplications”. *Proceedings ICASSP 1989*, pages 988-991, 1989.

[5] Y. Arai, T. Agui, and M. Nakajima “A Fast DCT-SQ scheme for images”. *Transactions on IEICE, vol. E-71, no.11,* pages 1095-1097, Nov1988.

[6] A. Fauth, J. Van Praet and M. Freericks. “Describing Instruction Set Processor using nMLx”. *Proceedings European Design and Test Conference,* Paris, Mar 1995.

[7] J. Sato et al “An Integrated Design Environment for Application Specific Instruction Processor”. *Proceedings ICCD*, pages 414-417. 1991.

[8] B. Holmer “Automatic Design of Computer Instruction Sets” *University of California at Berkeley,* Ph.D. dissertation, 1993.

[9] I. Huang, A. M. Despain “Synthesis of Application Instruction Sets”. *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol.14, no.6, pages 663-675, June 1995.

[10] Peter M. Athanas and Harvey F. Silverman “Processor reconfiguration through instruction-set methamorphosis”. *IEEE Computer*. vol. 26, no.3, pages 11-18, Mar 1993.

[11] B. Schackleford, M. Ysuda, E. Okushi and H. Koizumi “The Integrated Processor Synthesis and Compiler Generations Systems”. *SASIMI.* pages 135-142, Nov 1996.

[12] R. Lee “Subword parallelism with MAX-2”. *IEEE Micro,* vol.16, no, pages 51-59, 1996.

[13] J. N. Hennessy and D. A. Patterson “Computer Architecture: a Quantitative Approach”. *Morgan Kaufmann Publishers,* second edition, 1996.

[14] D. Lanner, J. Van Praet, A. Kiflt et al. “CHESS - Retargetable Code Generator for Embedded DSP Processor”. *Kluwer Academic Publisher,* pages 85-103, 1995.

[15] ISO/IEC JTC1/SC29/WG11 MPEG, International Standard ISO 11172, coding of moving pictures and associated audio for digital storage media up to 1.5 Mbits/s, 1992

[16] J. Watkinson, MPEG-2, Focal Press 1999