Computer Science • 14 (4) 2013 http://dx.doi.org/10.7494/csci.2013.14.4.531

Sławomir Cichoń
Marek Gorgoń

FPGA-BASED DVCPRO HD DECODER IMPLEMENTATION USING IMPULSE C

Abstract *High-level languages (HLL) for defining hardware implementation are important in both academic and commercial research. Impulse C could be an example of such language. This environment provides a programming model and library of functions for parallel applications, targeting FPGA-based platforms with the ability to partition the algorithm between software and hardware. This article's aim is to briefly describe DVCPRO HD, one of the intra-frame video-coding algorithms widely used in consumer equipment. DVCPRO HD is a DCT -based lossy video coding algorithm which uses variable-length coding (VLC) and run-length encoding (RLE) to achieve a 5:1 compression ratio. This paper presents DVCPRO HD video-coding-standard principles as well as decoder implementation working in real-time, compliant with the afore-mentioned standard and implemented in Impulse C. According to the authors' knowledge, the presented solution is the first FPGA implementation of this coding standard which includes all three VLC stages of data re-arrangement. What is more, this is the first DVCPRO HD implementation which utilizes Impulse C.*

Keywords high definition, video decoding, high level languages, pipelined architecture, intra-frame

1. Introduction

Currently, digital images and video samples are present in almost every aspect of life. Semi-professional cameras, with their ability to capture high-quality and high-resolution images, recently have become readily available to most people. The images made by such devices are compressed using GIF, JPEG, JPEG-2000, or other algorithms. On the other hand, todays consumers demand high-definition video capability in their homes and personal equipment. DVD and Blu-Ray discs, along with numerous social portals where everyone can share their own videos, triggered the development of various motion video coding standards, including DV, DVCPRO, H.263, H.264, Dirac, VC-1, and many others. Video coding techniques can be divided into two categories: inter-frame and intra-frame. Inter-frame algorithms typically require more memory space and are more computationally expensive than intra-frame algorithms; this is largely due to dependencies between the current and previous frames. Intra-frame coding is less complex because it uses data only from the current frame. Because of this, achieved compression ratios are higher for inter-frame types of coding. There are multiple platform types used to perform video encoding/decoding tasks; the most-explored and -suitable of those are multicore-CPUs, ASICs, FPGAs, and GPUs. Each combination of video-coding algorithm and platform is continuously being explored. Examples of ASIC solutions are presented in [21, 25, 9]. In the past few years we have observed rapid growth in the number of GPU platforms. Video-coding-related research on GPU are presented in [19, 3, 18]. Because of the reconfigurable feature of FPGA platform, it is commonly and extensively used in commercial products and various disciplines of academic research. Recent reported work on video-coding implementation using a reprogrammable chip are presented in [15, 13, 14, 10, 20, 12]. Hybrid architecture FPGA–GPU–CPU, capable of real-time processing of a stereo video stream, is reported in [8].

Over the past several years, high level languages (HLL) have begun to play an important role in hardware implementation design in both academic and commercial research. Impulse C could be such an example [11]. This environment provides a programming model and library of functions for parallel applications, targeting FPGA-based platforms. FPGA-based video decoding solutions seem to adhere to those capabilities. One of the intra-frame video-coding algorithms widely used in consumer equipment is DVCPRO HD [23]. It is a DCT-based lossy video coding algorithm which uses variable-length coding (VLC) and run-length encoding (RLE) to achieve a 5:1 compression ratio. Similar implementation for standard definition (SD), but in Handel-C, has been reported in [4] and [7]. This paper presents a DVCPRO HD decoder architecture and results achieved using Impulse C. Section 2 briefly describes the DVCPRO HD video coding standard, focusing on three passes of VLC coefficients arrangement and differences between DV and DVCPRO HD. Section 3 contains a brief overview of Impulse C high level language (HLL) as well as the decomposition of a video-decoding algorithm into processing elements (PEs), operating in parallel for implementation purpose. Section 4 presents the achieved results as well

as the performance of our implementation. Conclusions and future work are discussed in section 5.

2. Description of a problem solution

The subject of this work is 1440 x 1080 video signal, compressed using the DVCPRO HD algorithm, compliant with the standard [23]. This video-coding specification defines a lossy intra-frame only, DCT-based coding method with a compression ratio up to 5:1. Video input uses YCrCb color space with 4:2:2 chroma sub-sampling. The standardization document [23] describes in detail each of the functional steps present in the video coding process:

- Shuffling of macro blocks and video segment composition,
- Forward 2D-DCT,
- Weighting of DCT coefficients, with different matrices for luminance and chrominance,
- Quantization to reduce DCT coefficient value,
- Variable Length Coding to limit amount of data,
- Compressed data arrangement at block level, macro block level, and video segment level.

Similar steps are also present in the 61834-2 [1] and DV standard [22], which are predecessors of the DVCPRO HD. The main differences between DVCPRO HD and DV are associated with chroma sub-sampling, DCT modes, and quantization steps. Instead of 8-8-DCT and 2-4-8-DCT as found in DV coding, DVCPRO HD defines two modes: 8-8-frame-DCT, which is identical to 8-8-DCT, and 8-8-field-DCT mode, which is different from 2-4-8-DCT mode. In the 8-8-field-DCT mode, pixels from two vertical adjacent DCT blocks need to be rearranged to form two DCT blocks with pixels from the same fields, as shown in Figure 1.

Different DV standards apply the same DCT mode under different terms. For example, 8-8-DCT from [1] is mathematically the same as 8-8-frame-DCT defined in [23]. This is typical 2D DCT transformation on 8x8 blocks of data (either luminance or chrominance). 2-4-8-DCT is present only in [1] and is calculated on blocks with a major difference between half-frames (pixels from odd- and even-line numbers). It is calculated using a different formula than 8-8-DCT mode. Unlike DV, the same DCT mode is applied to the whole macro block in DVCPRO HD. Quantizer matrices are defined separately for luminance and chrominance. This causes different luminance values reduction than chrominance, which implies a better visual perception. The only lossy operation is the quantization of AC coefficients prior to VLC coding. During the video-decoding process, inverse operations of all of the above steps need to be performed. The most computationally-complex operations are 2D-IDCT and VLC decoding. Hardware implementations for 2D-IDCT have been widely reported in the past [16, 6, 2]. The most computationally-complex part is related to compressed video segment data re-arrangement, which needs to be done in three passes (each at the

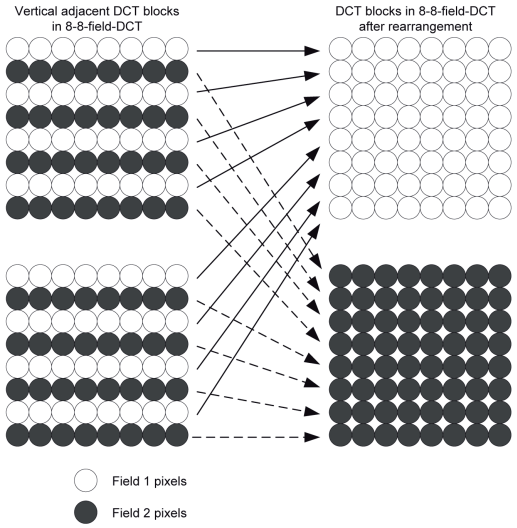

Figure 1. Pixels rearrangement in the 8-8-field DCT.

different data sets). Moreover, the second and third passes use the remaining data from the previous pass. During coding, DCT coefficients are arranged with a zig-zag order and, after quantization, one or more successive AC coefficients are coded using predefined tables defined by the standard [23] into code words with length between 3 and 16 bits. Finally, Run Length Encoding (RLE) is applied. As the last step, compressed data is placed in the compressed data stream. However, the size of the compressed video segment (five macro blocks taken from different places within the frame during a process called 'shuffling') and the single macro block are fixed. These macro blocks are chosen in the shuffling process. Its purpose is to average the video-data stream and minimize loss of data during data arrangement after VLC coding. Fixed bit rate reduction applied without shuffling would generate different distortions, depending on the picture details. Typically less-detailed areas are located on the frame borders, while more-detailed parts are located at the center of the frame. Shuffling will average the amount of data to be coded as well as help to achieve a goal of not exceeding the defined 385 bytes after coding. Data arrangement in a compressed macro block dedicated area is shown in the picture below (Fig. 2).

STAtus carries information about the status of the macro block indicating error and concealment information, ignored in the presented solution. QNO is the quantization number applied to the macro block data. DC consists of the DC coefficient (mean value of pixels within the DCT block) and class of the DCT block. AC is the area where compressed DCT coefficients with non-zero frequency are arranged. Naturally, data from some macro blocks does not occupy the entire dedicated area; on the other hand, there is some data that requires more space. To resolve this, the compressed video segment data is arranged in three passes:

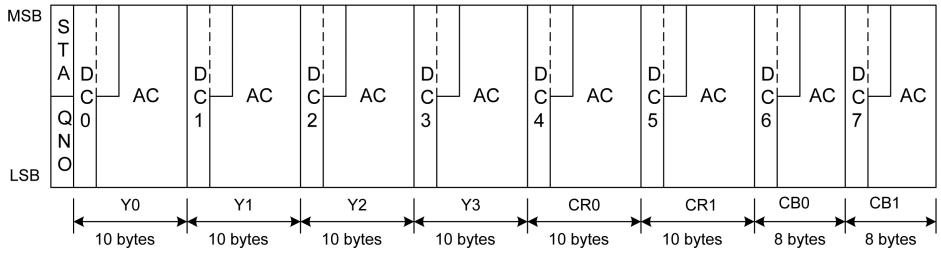

Figure 2. Data arrangement in the compressed macro block.

1. Pass 1: Compressed DCT block data distribution within dedicated block area - DCT block level.
2. Pass 2: Distribution of excessive data which remains after Pass 1 in the free space in different blocks within the same macro block - macro block level.
3. Pass 3: Distribution of excessive data which remains after Pass 2 in the free space in different blocks within different macro blocks within the current segment – video segment level.
4. Data that could not be arranged in any of above steps is discarded.

This idea is presented on the Figure 3.

The same idea has been applied in other DV coding algorithms in the family, like 61384-2 and DV. Additional research has been done on standard definition DV video sequence to compare the influence of the number of passes in compressed data arrangement on decoding quality. Peak Signal to Noise Ratio (PSNR) has been chosen as quality measure calculated with the formula below:

$$RMSE = \sqrt{\frac{1}{3MN}\sum_{i=0}^{M}\sum_{j=0}^{N}\left[\left(R_{ij}-R_{ij}^{*}\right)^{2}+\left(G_{ij}-G_{ij}^{*}\right)^{2}+\left(B_{ij}-B_{ij}^{*}\right)^{2}\right]} \quad (1)$$

where:

R_{ij}, G_{ij}, B_{ij} – color values of the examined frame,

R_{ij}^{*}, G_{ij}^{*}, B_{ij}^{*} – color values decoded using reference decoder.

$$PSNR = 20log_{10}\frac{255}{RMSE} \quad (2)$$

DV software codec libdv [24] has been used as a software decoder. It was modified to allow disabling of each VLC pass. As a reference decoder, MainConcept Reference Demo version has been used [17]. PSNR values have been calculated for each number of passes used during decoding. Four experiments have been conducted for a chosen video sequence which is about 900 frames long. In the first experiment, MainConcept was used to decode a compressed-video sequence. In the second, third, and fourth experiment, the libdv decoder was used with the first pass only, first and second pass only, and all three passes respectively.

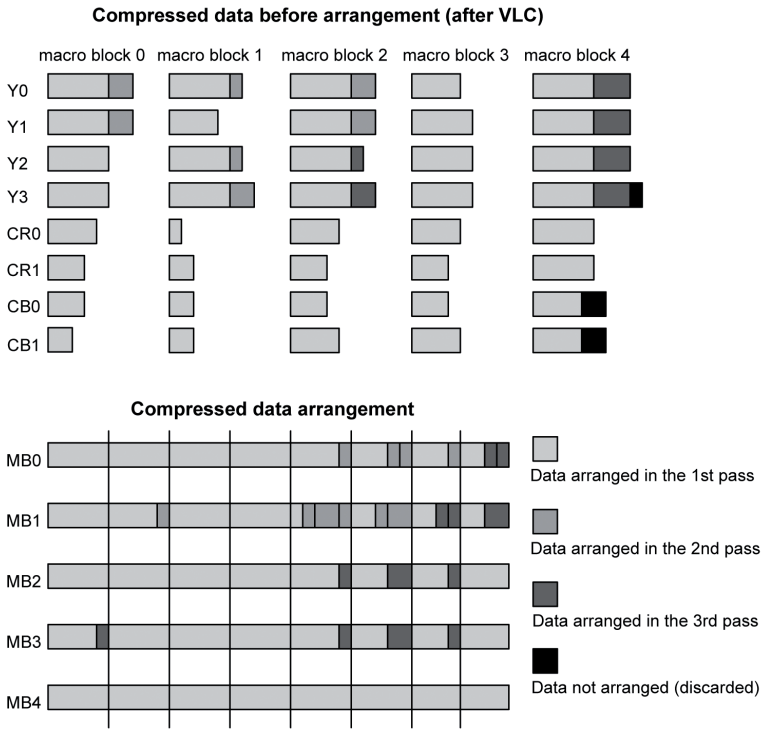

Figure 3. Data arrangement in the compressed macro block.

Results show that in dynamic scene where there are lot of details, all three passes can gain up to 5dB when compared to a single pass. The above results show that, to achieve higher decoding quality in terms of PSNR, implementing all three passes of VLC decoding can be beneficial.

Motion video decoding algorithms must meet at least one requirement – they must operate in real-time. Implementation of the algorithm in real-time is possible with a number of platforms mentioned in the 1 section. The use of FPGA allows us to design an energy-efficient decoder which can be integrated in different classes of devices. It is possible to use it in embedded systems, commercial products, or in high-quality broadcast equipment. Using reprogramming and changing the configuration of the FPGA, the standard of the applied decoder can be easily changed by fitting the reconfigurable logic to the type of image signal received. In this way, one can avoid the necessity of using separate decoders for each standard. A reconfigurable solution with high probability can also be adapted to the new, not yet established standards, allowing the user to avoid the necessity of replacing the unit each time a new standardization is introduced. To achieve the real-time processing goal, an FPGA-based platform has been chosen, which allows the entire algorithm to be broken

down into smaller parts. Each part has been implemented in a dedicated processing element (PE). Processing elements are organized in the pipeline architecture for the best performance. The Impulse C language has been chosen as the design language. Finally, the problem to solve can be defined as: implementation of the FPGA-based DVCPRO HD video decoder using Impulse C, which meets real-time requirement. Since [23] defines a 25-fps frame rate, this directly determines maximal processing time of a single frame to 40 ms. The total number of compressed video segments in the frame is equal to 1215. Upper limit of processing time and number of clock cycles, in which decoding of each video segment needs to be performed, can now be calculated as:

$$\lceil T_{VSdecode} \rceil = \frac{1}{frame\_rate \cdot number\_of\_VS} = \frac{1}{25 \cdot 1215} \approx 33\mu s \tag{3}$$

Assuming FPGA clock frequency 200 MHz

$$\lceil N_{VSdecode} \rceil = \frac{freq_{CLK}}{frame\_rate \cdot number\_of\_VS} = \frac{200 \cdot 10^6}{25 \cdot 1215} \approx 6584 \tag{4}$$

Based on the number of clock cycles needed for each individual processing element to complete and number of video segments within the single video frame, theoretical frame rate achieved by the implemented decoder can be specified using the equation:

$$frame\_rate_{MAX} = \frac{freq_{CLK}}{\lceil N_{VSdecode} \rceil \cdot number\_of\_VS} \tag{5}$$

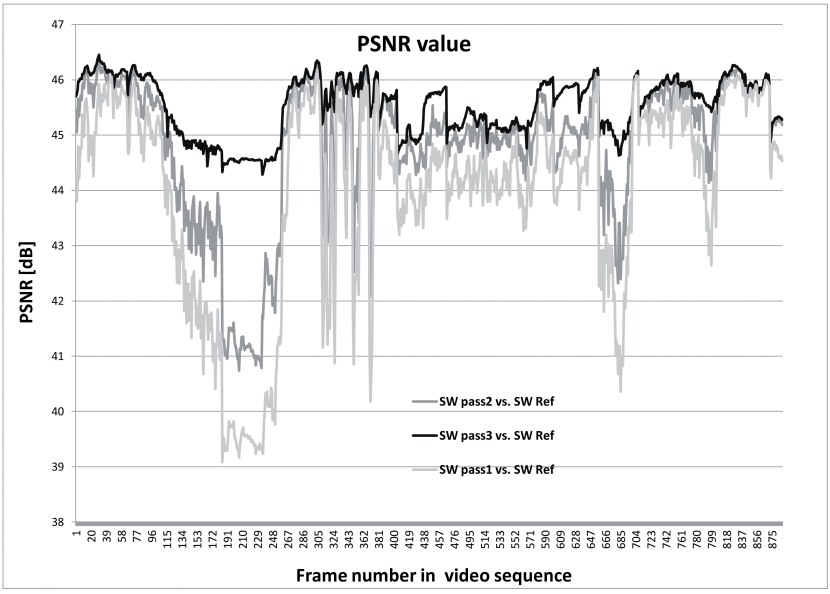

Figure 4. PSNR value for the video sequence with different number of data re-arrangement passes utilized.

$\lceil N_{VSdecode} \rceil$ defines maximal number of clock cycles of each processing element within the pipeline architecture to fulfill the real-time requirement. It is very important to calculate this formula prior to implementation, because Impulse C is an untimed, high-level language. The number of clock cycles used for processing cannot simply be obtained from the C code. One of the methods to gather such information during hardware simulation is presented in section 4.

3. Applied algorithms and methods. Software and hardware used

Impulse C is an ANSI C based language. It provides a library of functions and multiple bits data types. It also provides a programming model for FPGA-based platforms. Impulse C has been designed to simplify the expression, verification, and compilation of complex applications consisting of multiple parallel processes. Impulse C allows us to design mixed software/hardware targets. CoDeveloper includes the Impulse C libraries and associated software tools such as Application Monitor, Stage Master Explorer, and Stage Master Debugger. These allow us to simulate and debug the solution at the C language level as well as the generated HDL level. Both VHDL and Verilog code can be generated from Impulse C project. Impulse C defines three main communication mechanisms between software and hardware processes:

- Stream *co_stream*, in forms of synchronous FIFOs, which can be used to synchronize processes, as well as data exchange.
- Signal *co_signal*, typically are used to synchronize processes. They can also carry 32-bit value.
- Shared memory *co_memory*, for data exchange between processes.

CoBuilder is the hardware generation tool that converts the Impulse C application onto programmable hardware for acceleration. CoBuilder analyzes the application code, extracts those processes that have been specified for implementation in hardware, and creates optimized (HDL format) hardware descriptions ready for synthesis into an FPGA device. The hardware platform is supported by a set of optional libraries, platform-specific libraries, examples, and documentation called Platform Support Package (PSP). An initial study by the authors discovered stream ineffectiveness within the implementation in regards to data throughput. Because of this, Impulse C signals and memories are used within the presented implementation.

All decoding steps have been implemented in Impulse C, while some of decompression stages have been decomposed into smaller parts due to complexity or dependencies between data during decoding. The Impulse C environment allows us to define software/hardware co-design. The designer can declare any number of processes, which are processing element equivalent, to be run as software processes or to be synthesized and run on target FPGA. The designer can choose which process will be run on CPU and which on FPGA. Impulse C defines standard generic functions to communicate between processes, and PSP implements those interfaces for specific

hardware platform. Figure 5 describes hardware processes in the proposed implementation.

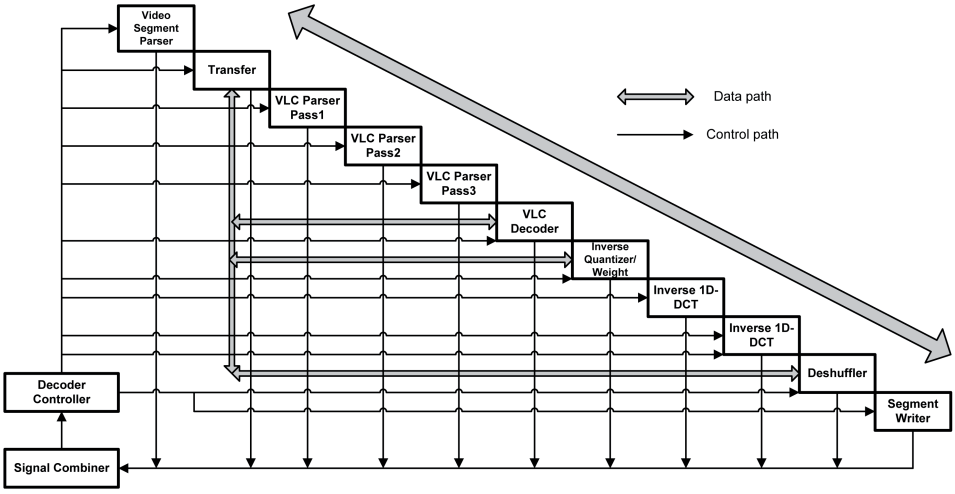

Figure 5. DVCPRO HD decoder functional block diagram.

Some of processes defined can be directly mapped to decoding phases, as described in coding standard [23]. Those are: VLC Decoder, Inverse Quantizer/Weighting, and 2D-IDCT. Inverse DCT has been split into two one-dimensional stages according to the most popular Loeffler fast DCT proposal [16]. There are additional hardware processes:

- Video Segment Parser: decodes from the compressed data stream common macro block and block parameters, that are used later in the decoding process, e.g. DCT mode, quantization class, DC coefficients.
- Transfer: writes coded AC coefficients into dedicated buffers.
- VLC Parser Pass1: parses VLC codeword till End Of Block (EOB) symbol or end of dedicated DCT block area is reached. Then it copies the data to another buffer for the particular DCT block. This buffer will be appended with data parsed during Pass2 and Pass3.
- VLC Parser Pass2: parses the data left after pass 1 for each macro block, and places the data in to DCT buffer.
- VLC Parser Pass3: similar as in VLC Parser Pass2, but it parses the data at the current video segment level.
- Deshuffler: places pixels in the appropriate places in the buffer.
- SegmentWriter: writes decoded macro blocks into specific areas of the decompressed frame memory buffer in external SDRAM, depending on XY coordinates of the processed macro block.

- SignalCombiner: terminates all output processing completion signals from PEs and passes only one signal to DecoderController after receiving all signal confirming the completion of processing for each PE.
- DecoderController: receives signal from SignalCombiner of processing completion in the current iteration, triggers next iteration processing along with additional input data.

Shared memory banks between software and hardware processes are:

- CompressedFrame: memory bank shared between Producer software process and hardware process called Transfer on FPGA. Used to store the compressed frame data read from the file. Size of the buffer: 486000 Bytes.
- DecompressedFrame: memory bank shared between FPGA, SegmentWriter process, and Consumer. Used to store decoded frame. Size of the buffer is: 6220800 Bytes.

Each of the hardware processes involved in the decoding process has defined one input signal and one output signal for synchronization purpose. All output signals from processing elements are connected to Signal Combiner. Its role is to communicate current video segment decoding completion upon reception of a completion signal from all decoding processes. Processing of the next video segment can not be started if all PEs have not completed the processing from the previous segment. Data between hardware processes are exchanged using BRAMs. Signals are used for synchronization purposes between PEs, and they also carry input data specific to the currently processed video segment. An example of two hardware processes and their inter-communication applied in the reported solution is presented in Figure 6.

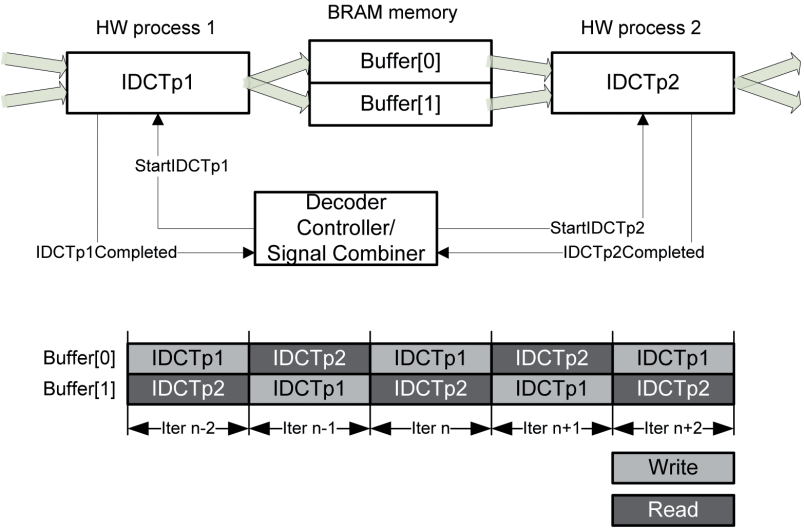

Figure 6. Inter-process communication and synchronization.

The DRC AC2020 (Fig. 7) has been chosen as a hardware platform. It is a high-performance computing system which communicates with the outside world over HyperTransport bus. The module is equipped with Virtex-5 LX220 FPGA as well as two DDR memory banks with 2 GB each. Details on this high-performance computing-hardware platform family can be found on the manufacturer website [5]. The module is placed in the second processor socket (instead of the CPU) on the motherboard of the server, which allows the AC2020 to work as a coprocessor in PC-like architecture. This solution allows us to access the AC2020 memory from both FPGA and CPU.

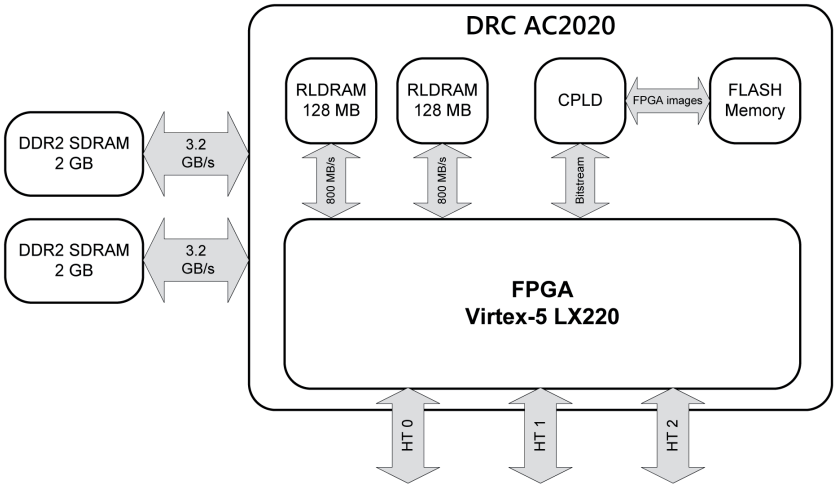

Figure 7. DRC AC2020 reconfigurable processing unit.

Impulse C allows us to define not only hardware, but also software processes. To implement compressed and decompressed frame data transfer from the memory shared between FPGA and CPU, two software processes have been defined (typically in Impulse C) called Producer and Consumer. Producer, in general, transfers data using streams directly to FPGA; however, since the hardware platform is equipped with memory which can be shared between FPGA and CPU, it has been utilized instead of the stream. Upon compressed data transfer, Producer triggers FPGA by posting a signal to kick off the decoding process. Results of the decoding process are stored in the second memory bank, which is also shared with the CPU. Upon decoding completion, FPGA triggers another software process called Consumer; this transfers the decompressed data and stores it in an output file which can be used for verification or used to visualize the result. The sequence of decoding a single frame is shown in Figure 8. It describes a synchronization method between software and hardware processes.

To achieve real-time performance defined by eq. (4), a number of implementation iterations have been simulated. To measure Impulse C code performance, the following additional functionality has been added:

- Additional hardware process ClockCounter, in forms of infinite loop that increments the clock cycle counter and stores its value in the global variable, which can be accessible by other PEs.
- Each hardware process has been instrumented to check the number of clock cycles at their entry and exit points. These values are subtracted to calculate the duration of the current iteration in each PE.

Data collected during hardware simulation, either using Stage Master Debugger or HDL simulation, was essential in bottleneck identification. It was also the input for confirmation of real-time processing of the designed decoder.

4. Results

The DVCPRO HD video decoding algorithm has been implemented and verified in both software and hardware simulations in the Impulse C environment. It has been also verified in terms of functionality as well as real-time requirements on the hardware platform DRC AC2020. It has been proven that the real-time performance condition was fulfilled; i.e., the frame rate shall not be less than 25 fps at an FPGA clock frequency of 200 MHz. Thirteen hardware processes have been designed and organized in the pipeline architecture for best performance of the decoder. Data between processing elements is exchanged by BRAM and signals. Detailed results obtained are presented in the Table 1.

Table 1
DVCPRO HD decoder synthesis report.

FPGA device	Frequency	SLICE occupied	BRAM used
Virtex-5	200 MHz	20 390 out of 34 560	112 out of 192
5vlx220ff1760-2		58%	58%

Impulse C code has been additionally instrumented to gather the number of clock cycles used for each PE in every iteration (each video segment decoding) during the single-frame decoding process. The data has been collected during hardware simulation and has been used to determine bottleneck and worst-performing processes.

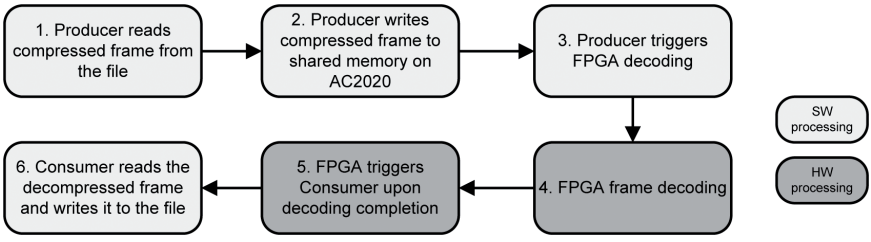

Figure 8. Single frame decoding sequence.

Iteratively, the performance of the pipeline has been improved to finally exceed the desired frame rate. All hardware processes, with the exception of those related to VLC decoding, require a constant number of clock cycles to be completed. Each video segment consists of a different number of VLC code words, which implies that these processes will have different duration for each of data sets. The Table 2 presents the detailed number of processing cycles for every PE. For VLC related processes, average values are provided. The worst-performing processing element is Deshuffle; however, it can be optimized. Having said that, it is not a crucial part of the decoding algorithm, and a non-optimal solution was accepted since the goal for the entire decoding process has been met. All values in the table are lower than those calculated in the eq. (4). Processes like Transfer and VSParser have relatively short durations compared to the longest one. Processing times of all other processes are within the same order of magnitude.

Table 2
Duration of hardware processes (number of clock cycles).

HW process	**VSParser**	**Transfer**	**VLCParserPass1**
Num. of CLK cycles	386	117	avg. 1556/ max. 2584
HW process	**VLCParserPass2**	**VLCParserPass3**	**InverseVLC**
Num. of CLK cycles	avg. 2147/ max. 2726	avg. 2602/max. 3685	avg. 3060/ max. 3564
HW process	**InverseQNOWght**	**IDCTStage1**	**IDCTStage2**
Num. of CLK cycles	3186	3094	3094
HW process	**Deshuffle**	**WriteSegment**	
Num. of CLK cycles	3764	3072	

The above results have confirmed real-time performance of the presented DVCPRO HD implementation. Based on the number of clock cycles of the slowest processing element, an achieved theoretical frame rate value can be calculated using below formula:

$$frame\_rate_{MAX} = \frac{freq_{CLK}}{\lceil N_{VSdecode} \rceil \cdot number\_of\_VS} = \\ = \frac{200 \cdot 10^6}{3764 \cdot 1215} = 43fps \tag{6}$$

This value far exceeds the requirement for real-time processing of the decoder.

5. Conclusions and future work

The DVCPRO HD decoding algorithm solution has been designed and implemented in Impulse C. The solution has been verified on the DRC AC2020 hardware platform. According to the authors' knowledge, the presented solution is the first FPGA implementation of this coding standard with all three VLC stages of data re-arrangement.

What is more, this is the first DVCPRO HD implementation using synthesizeable high-level language. All goals which were set at the start of the project, including real-time processing, have been achieved. The Impulse C language, as well as the DRC platform, proved to be useful for the implementation of such complex algorithms like video decoding, and met demanding performance requirements. Future work can include live demo design, with on-the-fly displaying decoded video on a computer screen. Further improvements and optimizations can also be sought to lower the maximum necessary clock frequency, in order to achieve real-time processing and to prove the applicability of HLL-designed implementations in low-power-demanding systems. Future work may also include a comparison of other decoding quality indicators, to evaluate and better understand the influence of the number of VLC data arrangement passes on the decoding quality.

Acknowledgements

Authors wish to thank the ACC Cyfronet and Department of Electronics, for providing a hardware platform. The work presented in this paper was supported by AGH UST grant 11.11.120.612.

References

[1] CENELEC: *Recording — Helical-scan digital video cassette recording system using 6,35 mm magnetic tape for consumer use (525-60, 625-50, 1125-60 and 1250-50 systems). Part 2: SD format for 525-60 and 625-50 systems (IEC 61834-2:1998)*, 1998.

[2] Chen W. H., Smith C. H., Fralick S. C.: A fast computational algorithm for the discrete cosine transform. *IEEE Transactions on Communications*, vol. 25, pp. 1004–1009, 1977.

[3] Cheung N. M., Fan X., O. C. A., Kung M. C.: Video Coding on Multicore Graphics Processors. *IEEE Signal Processing Magazine*, vol. 27, pp. 79–89, 2010.

[4] Cichoń S., Gorgoń M., Pac M.: Handel-C design enhancement for FPGA-based DV decoder. *Reconfigurable Computing Architectures and Applications, Lecture Notes in Computer Science, LNCS*, pp. 128–133, 2006.

[5] DRC Computing: *Accelium Coprocessors Product Datasheet webpage*, 2013. `http://www.drccomputer.com/pdfs/DRC_Accelium_Coprocessors.pdf`.

[6] Eijndhoven van J., Sijstermans F.: *Data Processing Device and method of Computing the Cosine Transform of a Matrix*, Patent WO 9948025, 1999.

[7] Gorgoń M.: *Architektury rekonfigurowalne do przetwarzania i analizy obrazu oraz dekodowania cyfrowego sygnału wideo.* UWND AGH, Kraków, 2007.

[8] Greisen P., Heinzle S., Gross M., Burg A P.: An FPGA-based processing pipeline for high definition stereo video. *EURASIP Journal on Image and Video Processing*, 2011.

[9] Hsiao Y. M., Chang F. P., Chu Y. S.: High speed multimedia network ASIC design for H.264/AVC. In: *The 5th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2010.

[10] Huang H. J., Fang C. H., Fan C. P.: Very-large-scale integration design of a low-power and cost-effective context-based adaptive variable length coding decoder for H.264/AVC portable applications. *IET Image Processing*, vol. 6, pp. 104–114, 2012.

[11] Impulse Accelerated: *CoDeveloper User Guide*, 2013. `http://www.impulseaccelerated.com/ReleaseFiles/Help/iAppMan.pdf`.

[12] Kalali E., Adibelli Y., Hamzaoglu I.: A high performance and low energy intra prediction hardware for HEVC video decoding. In: *Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2012.

[13] Kim S., Kim H., Chung T., Kim J. G.: Design of H.264 video encoder with C to RTL design tool. In: *International SoC Design Conference (ISOCC)*, 2012.

[14] Kinsman A. B., Nicolici N.: A VLSI Architecture and the FPGA Prototype for MPEG-2 Audio/Video Decoding. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, pp. 499–503, 2011.

[15] Lee C., Yang S.: Design of an H.264 decoder with variable pipeline and smart bus arbiter. In: *International SoC Design Conference (ISOCC)*, 2010.

[16] Loeffler C., Ligtenberg A., Moschytz G.: *Practical Fast 1-DCT Algorithms with 11 Multiplications*. In: *Proc. of the International Conference on Acoustics, Speech, and Signal Processing*, pp. 988–991, 1989.

[17] MainConcept: *TotalCode Studio webpage*, 2013. `http://www.mainconcept.com/products/apps-plug-ins/transcoding/reference.html`.

[18] Pieters B., De Cock J., Hollemeersch C., Wielandt J., Lambert P., Van de Walle R.: Ultra High Definition video decoding with Motion JPEG XR using the GPU. In: *18th IEEE International Conference on Image Processing (ICIP)*, 2011.

[19] Rodriguez R., Martinez J. L., Fernandez-Escribano G., Claver J. M., Sanchez J. L.: Accelerating H.264 inter prediction in a GPU by using CUDA. In: *International Conference on Consumer Electronics (ICCE)*, 2010.

[20] Shan J., Chen C., and Yang E.: *High performance 2-D IDCT for Image/Video Decoding based on FPGA*. In: *International Conference on Audio, Language and Image Processing (ICALIP)*, 2012.

[21] Shou-Gen X., Ming-Jiang W., Shi-Kai Z.: A new hardware architecture for H.264 intra prediction frame processing. In: *IEEE 5th International Conference on Internet Multimedia Systems Architecture and Application (IMSAA)*, 2011.

[22] SMPTE: *SMPTE 314M: Data Structure for DV-based Audio, Data and Compressed Video 25 and 50 Mb/s*, 1999.

[23] SMPTE: *SMPTE 370M: Data Structure for DV-Based Audio, Data and Compressed Video at 100 Mb/s 1080/60i, 1080/50i, 720/60p, 720/50p*, 2006.

[24] SourceForge: *Libdv webpage*, 2006. `http://sourceforge.net/projects/libdv/`.

[25] Staworko M., Modrzyk D.: A high-performance VLSI architecture of 2D DWT processor for JPEG2000 encoder. In: *Proceedings of the 18th International Conference Mixed Design of Integrated Circuits and Systems (MIXDES)*, 2011.

Affiliations

Sławomir Cichoń
AGH University of Science and Technology, Krakow, Poland, slawcich@agh.edu.pl

Marek Gorgoń
AGH University of Science and Technology, Krakow, Poland, mago@agh.edu.pl

Received: 4.09.2013
Revised: 30.09.2013
Accepted: 01.10.2013