DesignCon 2006

Designing A Real-Time HDTV 1080p Baseline H.264/AVC Encoder Core

Vincenzo Liguori, Ocean Logic Pty Ltd
enzo@ocean-logic.com

Kevin Wong, Ocean Logic Pty Ltd
kevin@ocean-logic.com

Abstract

The hardware implementation of the emerging H.264/AVC video compression standard presents a number of difficult challenges when it comes to real-time encoding at HDTV rates. This paper describes an efficient implementation of a baseline H.264/AVC encoder core capable of encoding a 1920x1080 video stream in real time at 30 frames per second (HDTV 1080p). A very favorable comparison with the JM 8.6 software reference model also will be presented. While the specific target of the design was HDTV 1080p, the small size and low clock frequency required make this core suitable for a variety of applications, from mobile communication devices to HDTV camcorders and video surveillance systems.

Authors Biography

Vincenzo Liguori was born in 1966 and graduated in electrical engineering at the University of Naples, Italy in 1989. He has worked for Marconi Italiana and as a researcher for Canon Information System Research in Australia. In 1996 he obtained his Master of Research in image compression at Sydney University, Australia. That year he founded Ocean Logic Pty Ltd, a provider of encryption and multimedia cores.

Kevin Wong graduated in 1985 with a Bachelor of Engineering (Electrical) degree from the University of Sydney. His engineering work began in the graphics and video processing area for companies like Rank-Cintel and Canon Information Systems Research Australia. Following that, he also worked in Telecommunications for Fujitsu and Cisco Systems. He joined Ocean Logic in 2004.

Introduction

The hardware implementation of the emerging video encoding standard known as H.264/AVC[1] presents a number of difficult challenges. We discuss the issues surrounding the design of a baseline H.264/AVC encoder core capable of processing up to HDTV 1080p (1920x1080) at 30 fps. We start with a brief outline of the baseline H.264 algorithm and discuss the features and capabilities of the core. We then examine some of the challenges in implementing these features. Finally, we show the performances of the core in terms of gate count as well as PSNR comparisons with the JM software reference model.

The Baseline H.264/AVC Algorithm

The Baseline H.264/AVC algorithm is quite complex, and we will briefly outline the process and information flow during encoding in order to understand the implementation issues that face hardware designers.

H.264 Baseline encoding can be essentially divided into a series of processing steps for each 16x16 pixel macroblock in the source video frame. We will focus only on the subset of the H.264 Baseline algorithm features that we have supported. A data flow diagram is shown below.

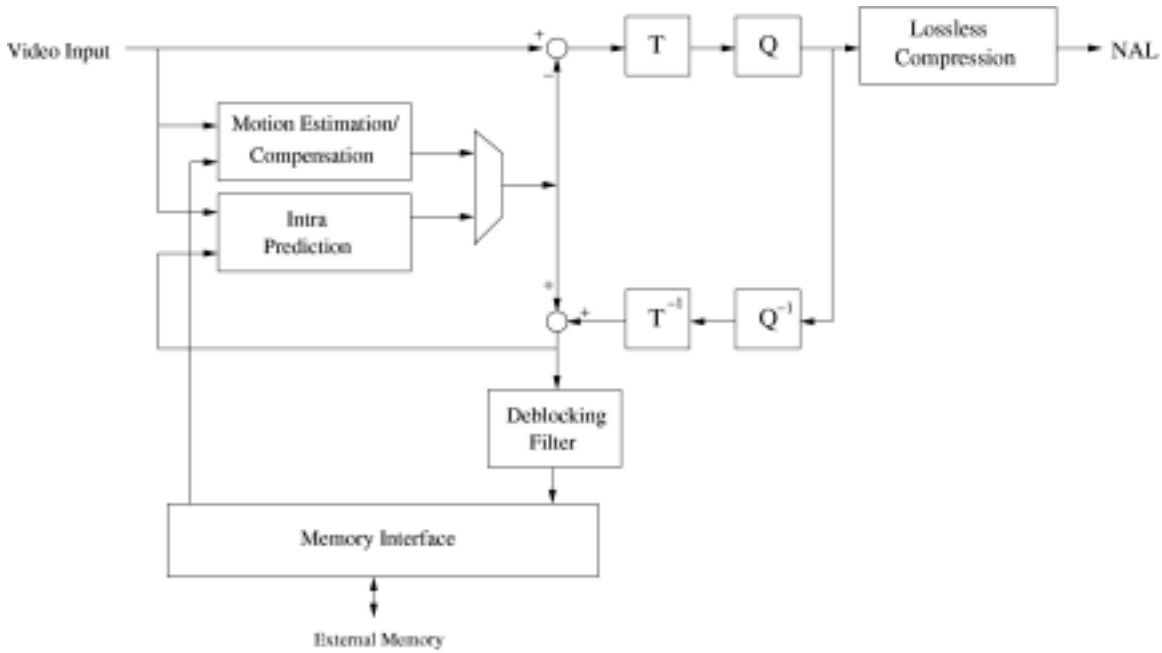

Figure 1. H.264 encoder dataflow

The initial step is the generation of a prediction. The baseline H.264 encoding algorithm uses two kinds of prediction: *intra prediction* (generated from pixels already encoded in the current frame) and *inter prediction* (generated from pixels encoded in the previous frames).

A residual is then calculated by performing the difference between the current block and the prediction. The prediction selected is the one that minimizes the energy of the residual in an optimization process that is quite computationally intensive.

A linear transform is then applied to the residual. Two linear transforms are used: Hadamard and a transform derived from the discrete cosine transform (DCT).

The coefficients resulting from the transformations are then quantized, and subsequently encoded into Network Abstraction Layer (NAL) units. These NALs include context information—such as the type of prediction—that is required to reconstruct the pixel data. The NAL units represent the output of the baseline H.264 encoding process.

Meanwhile, inverse quantization and transform are applied to the quantized coefficients. The result is added to the prediction, and a macroblock is reconstructed. An optional de-blocking filter is applied to the reconstructed macroblocks to reduce compression artifacts in the output. The reconstructed macroblock is stored for use in future intra prediction and inter prediction. Intra prediction is generated from unfiltered reconstructed macroblocks, while inter prediction is generated from reconstructed macroblocks that are filtered or unfiltered.

The next subsections detail several of the steps just described.

Intra Prediction

As mentioned above, intra prediction is formed from pixels that were previously encoded. Two kinds of intra predictions are used: intra16x16 and intra4x4.

In intra16x16, all the pixels already encoded at the boundary with the current block can be used to generate a prediction. These are shown shaded in the figure below. The core can generate the four modes of the intra16x16 prediction. In intra4x4, 16 4x4 blocks of prediction are generated from the pixels at the boundaries of each 4x4 prediction block as shown in Figure 2, Boundary pixels used in intra16x16 and intra4x4 intra prediction modes. For clarity, only some of the pixels in the prediction are shown. The core supports all the intra4x4 modes, with the exception of modes 3 and 7.

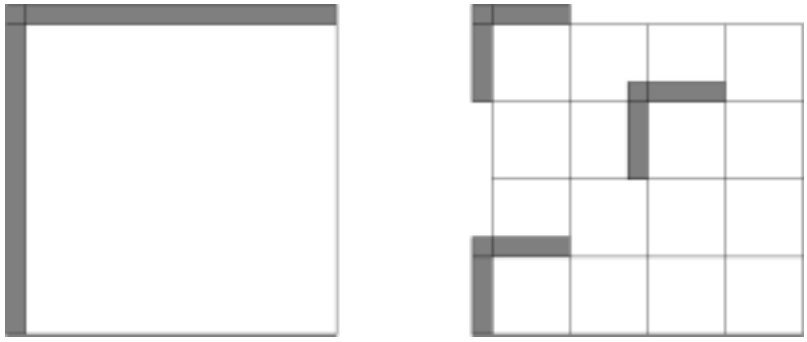

Figure 2. Boundary pixels used in intra16x16 and intra4x4 intra prediction modes

Inter Prediction

The inter prediction is generated from motion estimation. Motion estimation is at the heart of all standard video compression algorithms. This technique is used to exploit the temporal redundancy present in natural video sequences.

Motion estimation is performed by searching for a 16x16 area of pixels in a previously encoded frame so that the energy of the residual (difference) between the current block and the selected area is minimized. The selected area is indicated by a motion vector. Figure 3 illustrates this process.

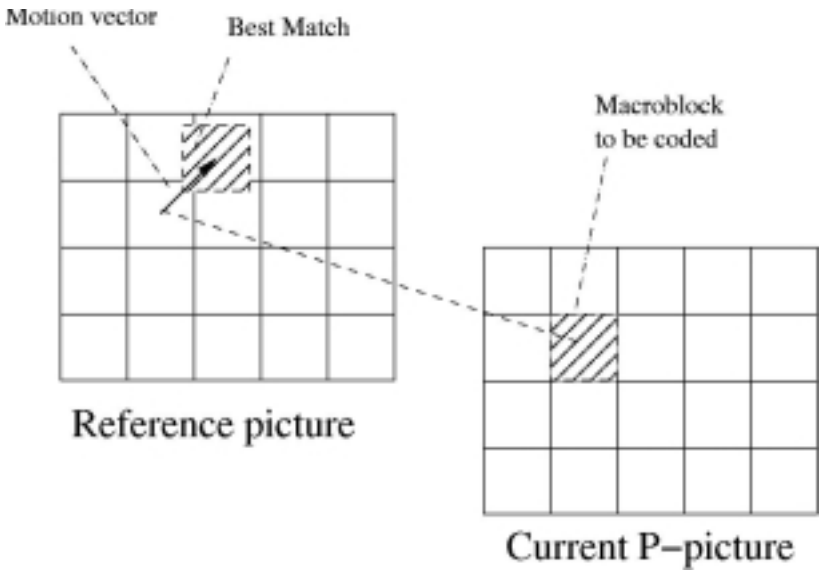

Figure 3. Inter prediction

The core will search an area 32x32 pixels wide, down to ¼ pixel of resolution (-16.00, +15.75 in both X and Y direction). Pixels at ¼ resolution are generated with a complex interpolation filter described in the ITU-T H.264 specification.

Transform and quantization

As mentioned above, two types of transforms are used in the baseline H.264 encoding algorithm: the Hadamard transform and an integer transform derived from the DCT.

The full definitions of these transforms can be found in the ITU-T H.264 standard. The important thing to notice here is that both transforms (and their inverse functions) can be performed by using only additions, subtractions and shift operations.

Both quantization and its inverse are also relatively simple and are implemented with multiplication and shifts.

NAL encoding

The core generates the NAL types shown in the following table.

NAL Unit Type	Description
1	Coded slice of a non-IDR picture.
5	Coded slice of a IDR picture.
7	Sequence Parameter Set.
8	Picture Parameter Set.

Table 1. NAL Unit types

H.264 encoding can be essentially divided into two independent processes: motion estimation and compensation, and variable length encoding. The resultant bitstream is assembled into NAL units and output in byte stream format as specified in Annex B of the ITU-T H.264 specification.

Core features

The objective of our design was an IP core that would satisfy the needs of as many potential customers as possible. The requirements of different customers are often in conflict, and the attempt to reconcile the different requirements results in an interesting constrained optimization problem.

We have focused our design efforts on producing a general-use core that has acceptable performance for a range of products, from mobile devices to high-end video surveillance and “industrial” video quality HDTV camcorders.

The following features of the core are the result of the compromise between encoding performance and resources required to implement such a core in silicon. Most of the features are subsequently discussed.

- Baseline Profile, but encoded bitstream can be decoded by Main Profile decoder (for use with high-resolution devices).
- Supports up to Profile level 4.1, the highest HDTV video resolution (1920x1080 @ 30 fps progressive).
- Very low operational frequency: from ~1.5 MHz for QCIF @ 15 fps to ~250 MHz for 1920x1080 @ 30 fps.
- No CPU required for encoding.
- Variable Bit Rate (VBR) and Constant Bit Rate (CBR).
- Very low latency in VBR (~1.1 ms for VGA @ 30 fps).
- External memory interface tolerant of high latencies and delays, ideal in a system on chip (SoC) or when using a shared bus with a CPU.
- Motion vector up to –16.00/+15.75 pixels (search area is 32x32-pixel wide down to quarter pixel).
- Support for most of the intra4x4 and all intra16x16 modes.
- Support for multiple slices for better error resilience.
- Block skipping logic for lower bitrate.

- A deblocking filter for better quality.
- A simple, fully synchronous design. Implemented as fully functional and synthesizable VHDL or Verilog soft-core suitable for a variety of technologies from low end FPGAs to submicron ASIC.

Very Low Clock Frequency and HDTV Support

HDTV support up to 1080p at 30 fps was one of the main objectives of the design. With the increasing capabilities and the falling cost of high resolution image sensors, encoding support for HDTV resolutions will become more and more important.

In order to achieve this at a reasonable frequency, the core has been designed to process a 16x16 block of pixels every 1024 cycles at an average rate of one pixel every 4 clock cycles. An example of the relationship between clock frequency and some common resolutions and frame rates is shown in the table below.

Resolution	QCIF @ 15 fps	CIF @ 30fps	VGA @ 30fps	1280x720 @ 30fps	1920x1080 @ 30fps
Core freq.	~1.5 MHz	~12.1 MHz	~36.8 MHz	~110.5 MHz	~250.6 MHz

Table 2. Core frequency versus video resolution and frame rate.

Very low clock frequency also means very low power, especially at VGA and CIF resolutions.

No CPU required

The core is an independent entity capable of encoding a video stream without the support of an additional CPU. This can be a distinctive advantage as it generally reduces power consumption, gate count and licensing cost.

The core does have registers that need setting, but this can be done by other means and, in any case, this does not require any computational capability to speak of.

Support for both VBR and CBR

The core supports both VBR (Variable Bit Rate) and CBR (Constant Bit Rate) encoding.

In VBR mode, the encoded video quality is determined by a fixed quantisation parameter. While the video quality is fixed, the encoded bit rate will vary depending on the complexity of the incoming video. In CBR mode, the quantisation parameter is varied to maintain a constant average bit rate. This allows the core to be used in a variety of applications where the bandwidth is fixed, such as for wireless devices. Again, by providing both VBR and CBR we are trying to offer maximum flexibility.

In VBR mode the latency of the core is very low. Basically, by the time 16 lines of pixels have been input (just enough data to be able to form the first 16x16 block of pixels), the core can start encoding, outputting bitstream data a few thousand cycles later. For VGA @ 30 fps the time to input 16 lines of pixels is approximately 1.1 ms.

Flexible Memory Interface

The core needs an external memory to store the previous frame and the current frame being reconstructed. Also, chances are that it will be used in a SoC environment where other cores and a CPU might need to share the same resource.

We therefore designed the core with that goal in mind. The characteristics of the memory interface are:

- Very similar to and easy to integrate with the AMBA™ AHB interface
- Tolerant of high latencies and delays typical of shared buses
- SDRAM aware
- Can be clocked at a different clock speed from the main core

The external memory interface can be easily interfaced to the AMBA AHB with a minimal amount of extra logic. This will greatly facilitate core integration in an SoC environment.

The interface is also designed to be tolerant of latencies and delays typical of a shared bus. Basically, as long as a request is satisfied within 800 to 900 cycles of it being made, the core will operate correctly.

The external memory is likely to be, in many cases, a type of SDRAM rather than SRAM. Consequently, we have designed the core to cater to that eventuality.

One of the characteristics of SDRAM is for the memory to behave essentially like a SRAM provided that accesses are confined within a page. Only when crossing a page boundary will the penalty of extra cycles be incurred due to precharge. Therefore the core sorts all its memory accesses in a way that minimizes page boundary crossings, achieving performance closer to one that would be obtained if it was connected to SRAM. However, in order for this to work, the memory controller must be able to postpone precharging as long as accesses are confined to the same page.

The core will make a total of up to 404 read and 192 write accesses to the external memory in the course of 1024 cycles, about 58% of the total number of cycles available.

If the SDRAM controller follows the guidelines mentioned above, the maximum number of precharges required across 1024 cycles is 25 (15 for the read operations and 10 for the write operations). In this case, assuming an overhead of 4 cycles for each read precharge and 5 cycles for each write precharge, the total number of cycles is 596+15*4+10*5=706 cycles, about 69% of the total number of available cycles.

Finally, the external memory interface can be clocked at a different frequency from the main core. This again will assist integration in a variety of situations.

For example, the core could be used in an SoC in association with a CPU that uses a 200 MHz bus and memory system. Assuming that encoding VGA @ 30 fps is needed in such an application, the core can run at just 37 MHz.

If the memory interface was not capable of running at a different clock speed from the rest of the core, the whole core would be forced to run at 200 MHz, resulting in an unnecessarily greater gate count.

Another example is the encoding of two 4-CIF (704x576) video sources at 30 fps in a low-end FPGA. This can be achieved by instantiating two cores and, by clocking the memory interface at a different speed. The external memory can then be shared between the two cores, as shown below.

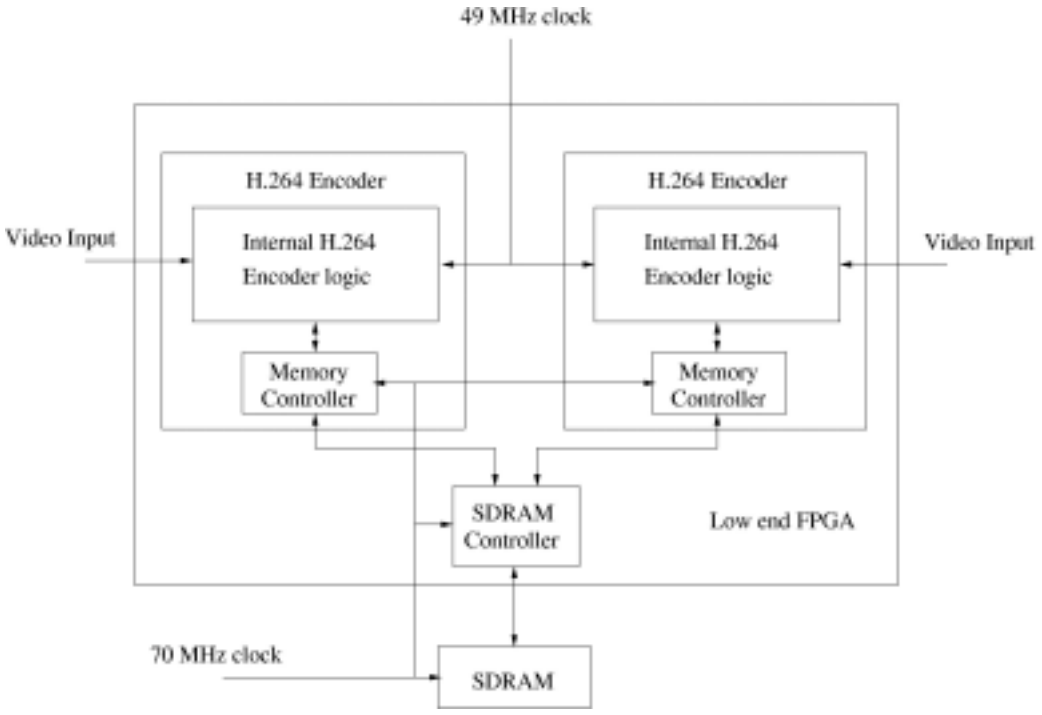

Figure 4. Two cores sharing memory at different clock frequencies

Other Baseline Features

The motion search area is limited to a single frame, +/-16 pixels down to ¼ pixel resolution and with a single 16x16 partition. This represents a good compromise between gate count and required bandwidth. Search in the subpixel area is done using the complex 2D filter specified by the encoding algorithm in order to generate the half pixels.

Intra prediction includes all intra16x16 and all intra4x4 modes with the exception of modes 3 and 7.

Other features include block skipping for lower bit count and multiple slice encoding for error resilience. Finally the deblocking filter is also used in order to improve image quality at low bit rates.

Technology Independence

The core is implemented as VHDL or Verilog RTL. It is also fully synchronous, with no gated clocks and transparent latches. This allows the same code to be ported virtually unchanged between various technologies from FPGAs to ASIC.

Design Challenges

Processing a 16x16 block of pixels in just 1024 cycles represents a considerable challenge. Some features of the H.264/AVC algorithm are particularly unsuitable to simple and fast hardware implementation.

We will now discuss some of the issues relating to the hardware implementation of the features previously described.

Motion Estimation

The motion estimation submodule of the core consists of two stages: integer pixel motion estimation followed by a refining step that searches for matches down to ¼ pixel resolution.

The integer search unit is relatively simple. It utilizes a 4 step search and sums of absolute difference (SAD) process to estimate the motion vector. Its performance is quite close to a full search. Most of the complexity in this unit comes from minimizing the number of accesses to the external memory.

A greater challenge is to generate the half-pixel samples to refine the search within the 1024 cycles allotted for each block. The complexity lays not so much in the computational difficulty (the interpolation filter is quite simple), but rather in minimizing the number of multi-port memories that a straightforward implementation would suggest.

Intra Prediction

Similar to the case of motion estimation, SADs are used to search for the intra prediction mode that best matches the current block of pixels.

While the implementation of all the intra16x16 prediction modes is fairly straightforward, searching the intra4x4 modes requires extra complexity. This is because the various intra4x4 predictions need samples from the neighbouring blocks. Since the neighbouring pixels must be identical to those available for the decoder, it follows that each neighboring block must have gone through transformation, quantization and their inverse. These operations add a considerable burden since they are speculative and, for example, if motion estimation produces a better match, they will have to be performed again.

Transform, Quantization and their Inverse

Given the required speculative execution of the transform, quantization and their inverse required for the intra4x4 modes as well as the general high-speed requirement, it would seem that a pipelined implementation of these elements would be desirable.

Unfortunately a straightforward pipelined implementation is not possible since for intra16x16 mode, the Hadamard transform of all the DC values is also required. All the DC values from the direct transform need to be processed first, preventing a straight passage to an inverse transform through pipeline architecture.

NAL Encoding

Each NAL unit contains context information about the type of prediction, motion vectors, Quantisation Parameter delta, and the Context Adaptive Variable Length Coded (CAVLC) luma and chroma coefficients. To maintain synchronization with the rest of the processing in the core, the NAL encoding logic is also constrained to encode this information for each macroblock in 1024 clock cycles. The other constraint is that the width of the bytestream-format encoded output be no greater than 32 bits to avoid problems connecting to external memories or buses. As the core outputs bytestream format, it also has to insert the emulation prevention byte. This possibility further restricts the bus width into the bytestream formatter to 16 bits.

In the majority of cases, most of the encoded bits in each macroblock are devoted to the CAVLC coefficients. CAVLC coding operates on 4x4 blocks and scans the coefficients in zig-zag order. Each 4x4 block comprises the following elements:

- the number of non-zero coefficients
- number of trailing ones (up to 3)
- sign of each trailing one (up to 3)
- the level code of each non-zero coefficient
- the zero run code preceding each non-zero coefficient

In the worst case, 16 non-zero coefficient levels have to be coded followed by 16 runs of zero. With each level code being a maximum of 27 bits for Baseline profile, it will require 2 clock cycles to be coded due to the 16-bit output constraint. This adds up to a maximum of 768 clock cycles just for the level codes of the 16x16 luma, and two 8x8 chroma blocks. With 256 cycles left to encode the zero-run codes and context information, it was necessary to devise an efficient way of coding this information.

External Memory interface

As mentioned before, the external memory interface of the core is designed to be tolerant of the unpredictable latencies and delays typical of a shared bus. This is achieved by designing the core in such a way that any memory access is known almost 1024 cycles before the data is actually needed.

This in turns allows posting requests to the external memory long in advance. Another consequence of this is that all the memory operations are fairly decoupled from the main core, and can easily operate at the different clock speed. In other words, the minimal interaction between the main core and its memory operations simplifies the asynchronous operations.

Performance

In this section we show the results of some comparisons between the core and the JM reference software[2]. We also present the results of the implementing the core in various technologies.

Comparison to the JM Software Model

A comparison was made between the C-model of the core and the JM reference software, for encoding the sequences listed in the table below. All the sequences are in YUV 4:2:0 format.

Sequence name	Frame size	Num of frames	File size (bytes)
Mobile	352 x 288	300	45,619,200
Tempete	352 x 288	260	39,536,640
Tennis	720 x 480	300	155,520,000

Table 3. The sequences used in the test

All the frames of the sequences listed are encoded using both encoders. Only the first frame is encoded as an "I" frame. The file size reported includes the complete encoded sequence, including any headers and the first "I" frame. A single reference frame is used.

During the tests, in the JM 8.6 software, all the inter-search partitions are activated (16x16, 16x8, 8x16, 8x8, 8x4, 4x8 and 4x4), rd optimization is off and the normal (non Hadamard) SAD is used.

Tests are conducted with various quantization values, and the deblocking filter enabled or disabled. We only present here the results for the deblocking filters disabled.

The resulting bitstreams are decoded using the JM reference software decoder. The PSNR and file size is recorded and reported.

The test results are listed in the tables and graphs below. File sizes are in bytes and PSNR in dB. The results are also shown as PSNR/size graphs.

Qp	PSNR Y	PSNR U	PSNR V	File Size
15	45.062	45.591	45.623	10,125,736
20	40.451	41.461	41.45	6,228,176
25	36.227	37.807	37.712	3,614,870
30	31.63	34.826	34.634	1,670,053
35	27.516	32.323	32.016	657,534

Table 4. JM 8.6 figures for the Mobile sequence

Qp	PSNR Y	PSNR U	PSNR V	File Size
15	44.991	45.586	45.619	10209836
20	40.393	41.443	41.434	6439912
25	36.135	37.767	37.684	3792762
30	31.553	34.797	34.599	1826485
35	27.46	32.313	31.984	766506

Table 5. Ocean Logic core figures for the Mobile sequence

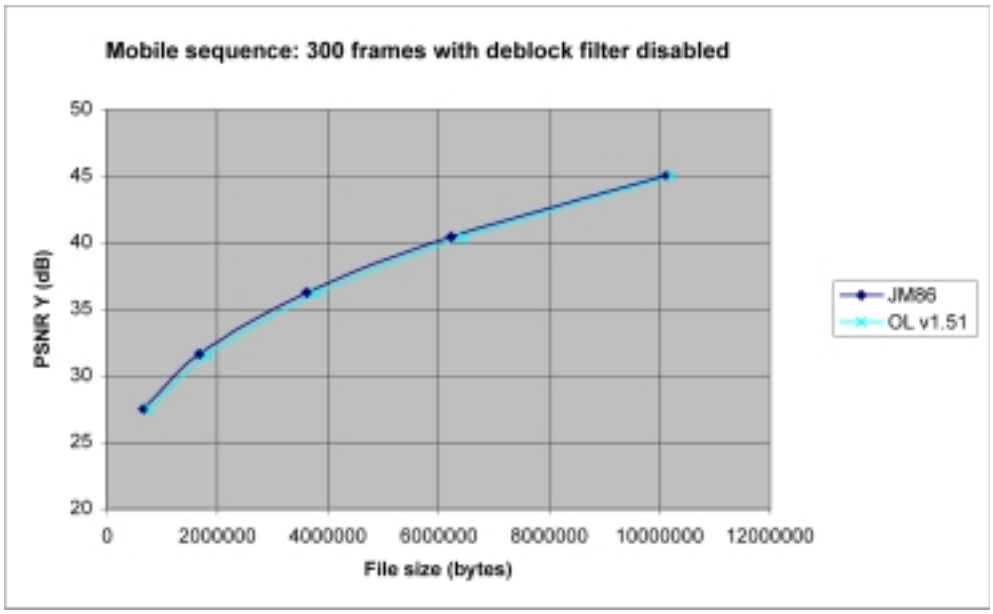

Figure 5. PSNR comparison for the Mobile sequence

Qp	PSNR Y	PSNR U	PSNR V	File Size
15	45.327	45.897	46.112	7,536,704
20	40.947	42.013	42.679	4,350,291
25	36.937	38.757	39.988	2,405,596
30	32.628	36.292	37.978	1,052,044
35	28.89	34.299	36.304	419,850

Table 6. JM 8.6 figures for the Tempete sequence

Qp	PSNR Y	PSNR U	PSNR V	File Size
15	45.265	45.951	46.193	7477296
20	40.909	42.058	42.78	4510059
25	36.868	38.772	40.049	2549262
30	32.549	36.273	38.026	1183653
35	28.737	34.265	36.332	503588

Table 7. Ocean Logic core figures for the Tempete sequence

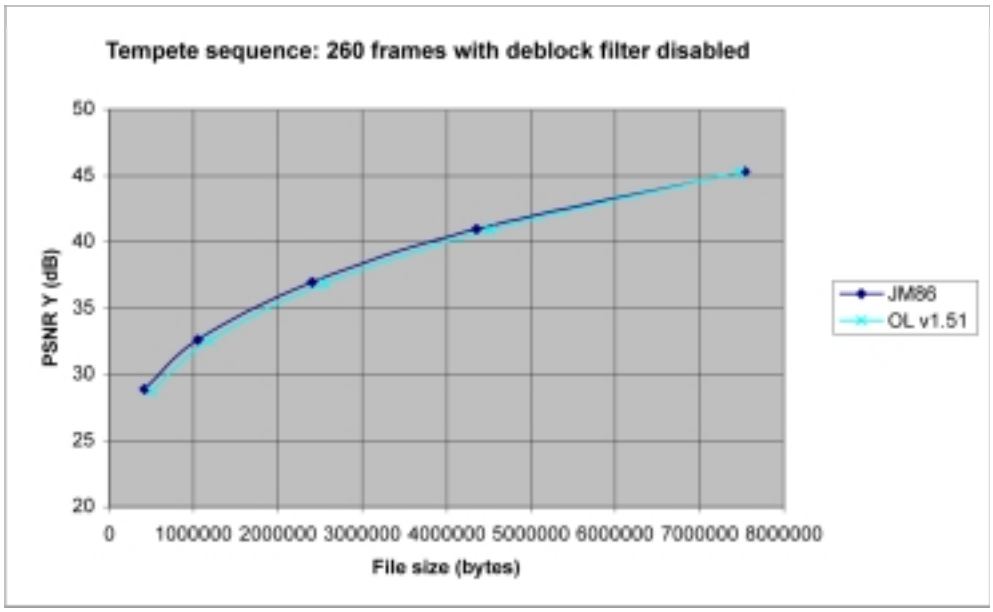

Figure 6. PSNR comparison for the Tempete sequence

Qp	PSNR Y	PSNR U	PSNR V	File Size
15	45.213	46.405	46.992	26,855,138
20	40.804	43.759	44.338	12,717,310
25	37.115	41.576	41.605	4,683,891
30	33.745	39.715	39.367	1,786,854
35	30.712	38.062	37.399	731,041

Table 8. JM 8.6 figures for the Tennis sequence

Qp	PSNR Y	PSNR U	PSNR V	File Size
15	45.199	46.641	47.478	25581040
20	40.846	43.847	44.493	13565541
25	37.086	41.624	41.697	5318030
30	33.876	39.784	39.452	2247988
35	30.971	38.206	37.565	1043761

Table 9. Ocean Logic core figures for the Tennis sequence

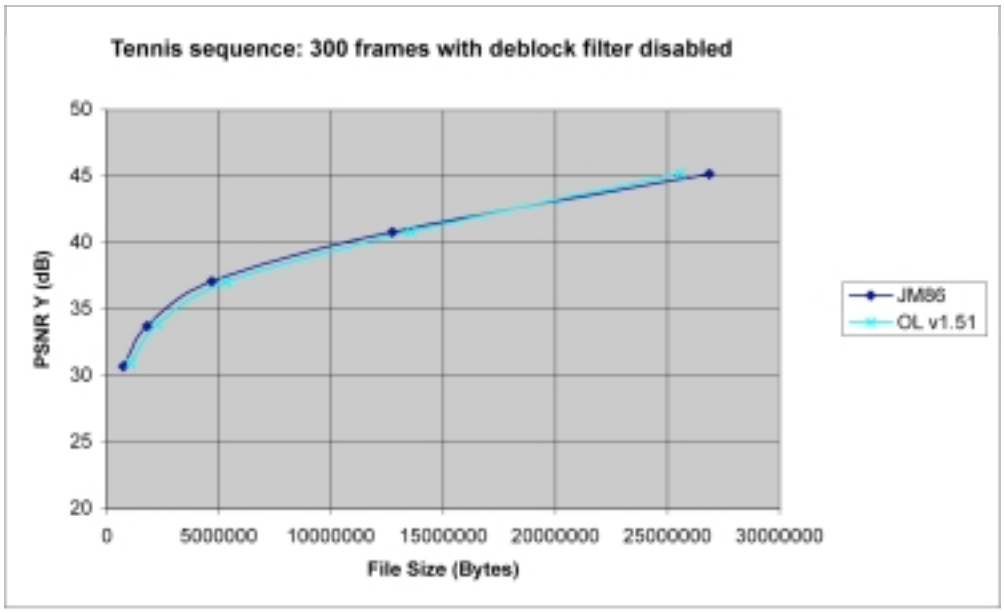

Figure 8. PSNR comparison for the Tennis sequence

ASIC and FPGA Synthesis Results

Synthesis results in various technologies demonstrate that the core is suitable for a wide range of applications from mobile phones to "industrial" video quality HDTV and high-resolution video surveillance.

Different technologies cover different resolutions and frame rates:

- 4CIF (704x576) at 30 fps with low end FPGAs (Xilinx™ Spartan3 and Altera™ CycloneII)
- 720p (1280x720) at 30 fps with high end FPGAs (Xilinx™ Virtex4 and Altera™ StratixII)
- 1080p (1920x1080) at 30 fps with 0.13um ASIC

Representative results for the design without CBR and the deblocking filter and for various technologies are in the table below.

Technology	Approx Area	Speed	Video Throughput
0.13 u LV 0.9V, 125 C	178 Kgates + 106 Kbits RAM Optimized for speed	~ 250 MHz	1920x1080 (1080p) @ 30 fps
0.18 u slow process	129 Kgates + 106 Kbits RAM Optimized for area	~50 MHz	4 CIF (704x576) @ 30 fps
StratixIIC3	17511 ALUTs + 5 M512 + 51 M4K + 3 DSPs	~118 MHz	1280x720 (720p) @ 32 fps
CycloneIIC6	18,510 M4K + 5 M512 + 51 M4K + 3 DSPs	~65 MHz	4 CIF (704x576) @ 40 fps
Virtex4-12	10,500 slices + 3 multipliers + 33 RAM blocks	~110 MHz	1280x720 (720p) @ 30 fps
Spartan3-4	10,500 slices + 3 multipliers + 33 RAM blocks	~50 MHz	4 CIF (704x576) @ 30 fps

Table 10. Implementation results for the H.264 core

Conclusion

The authors have implemented a very fast and small real-time baseline H.264/AVC core suitable for a variety of applications. The core's support for the highest HDTV resolution, 1920x1080 @ 30 fps progressive opens a whole new range of applications from high-end video camcorders to high-resolution video surveillance at very low cost.

References

[1] ITU-T H.264 specification, available for purchase from the ITU-T or by download at: http://www.itu.int/ITU-T/publications/recs.html.

[2] The JM reference software can be downloaded at http://iphome.hhi.de/suehring/tml/index.htm