

PACT

XPP Technologies

Video Decoding on XPP-III

White Paper

For further information and questions, please contact support@pactxpp.com.

Revision 1.1.1, July 18, 2006

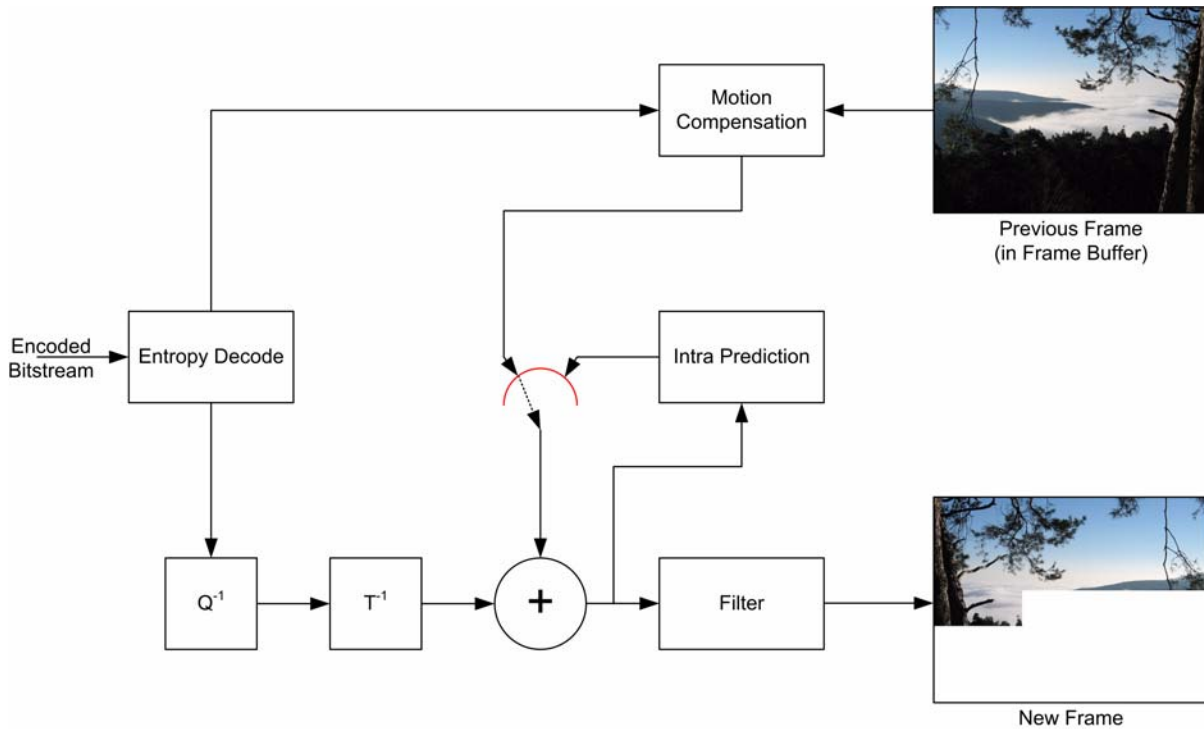
Table of Contents

1. Introduction.....	3
1.1 Design Flow and Algorithm Partitioning	5
1.2 XPP-Array Reconfiguration	6
2. Modules for Video Decoding.....	8
2.1 Entropy Decoding	8
2.2 Rescaling or Inverse Quantization	8
2.3 Inverse Transformation	8
2.4 Motion Compensation	9
2.5 Deblocking and Deringing Filters	12
3. Results	13
3.1 Frame Buffer Size	13
3.2 XPP Frequency and Local Buffer Size.....	13
4. Conclusion	15

1. Introduction

This paper contains details on using PACT XPP-III Architecture for accelerating the decoding of various video sequences, like MPEG2, MPEG4, H.264 and VC-1 (WMV9).

The block diagram below shows the general idea of all video decoders:



The encoded video stream is received and entropy decoded. Usually the entropy codec is a combination of Huffman and run-length coding. The results of entropy decoding are quantized coefficients and decoded parameters like motion vectors.

The quantized coefficients are passed through rescaling (Q^{-1} , also called inverse quantization) and the inverse signal transformation (T^{-1}). The result of inverse transformation is the so-called residual data. The reconstructed data is the sum of this residual data and some prediction computed via motion compensation or via Intra prediction.

The way of computing the prediction is selected according to the macroblock type “Intra” or “Inter”. For Intra blocks the prediction is computed from neighboring macroblocks in the same frame (Intra prediction). For Inter blocks the prediction is computed from previously reconstructed frames (motion compensation). The actual data used from previous frames is selected by a motion vector.

Finally, the reconstructed data is passed through a deblocking filter and added to the current frame.

Each frame of the video sequence is assigned one of three encoding types called Intra frame (or I-frame), Predicted frame (or P-frame), and Bi-predicted frame (or B-frame). I-frames are special in allowing only Intra-coded macroblocks, i.e. I-frames can be decoded without any knowledge of previously decoded frames. Obviously, the first frame of each sequence is always an I-frame. P- and B-frames allow mixing Intra-coded macroblocks with motion compensated Inter-coded macroblocks. P- and B-frames can only be decoded, when all referenced previously decoded frame(s) are available in the frame buffer.

Since algorithms like rescaling, inverse transformation and motion compensation are computationally very intensive and consume a significant percentage of CPU time, these algorithms are good candidates for acceleration by specialized hardware or reconfigurable processors like the XPP dataflow array.

When MPEG2 was the only video codec widely used, acceleration by dedicated hardware was the typical approach. However, with the emergence of new video standards, different hardware accelerators are required for each new standard. This increases silicon area and thus cost.

Moreover the existing video standards keep evolving, new extensions being constantly added to the standard. Only a fully programmable solution provides the flexibility required to adapt enhancements like the scalable extension of H.264 without modifying the hardware.

PACT's fully programmable XPP-III allows re-using the same silicon for different standards, saving cost and area, and allows for future adoption of up-coming standards. XPP-III does not comprise any specialized video decoding hardware.

The XPP-III architecture features fully programmable elements for dataflow processing as well as for sequential processing. The *XPP dataflow array (XPP-Array)* comprises ALU-PAEs (PAE = Processing Array Element) and local RAM-PAEs connected by high-bandwidth data and event (control) networks. The XPP-Array is course-grained and runtime reconfigurable. The *Function-PAEs (FNC-PAEs)* are sequential processing cores that are designed to efficiently process highly sequential and highly conditional algorithms.

The scalability of the XPP-III architecture ensures that it delivers just the right performance, tuned for the target application. Consequently, PACT proposes three different sizes of XPP-III for video codecs.

XPP12.6.3¹: A comparatively small 16-bit array, targeting mobile applications and decoding resolutions up to VGA (640x480x30fps).

XPP20.8.4: A medium-sized 16-bit array targeting SD resolution and decoding up to 1280x720x30fps.

XPP40.16.8: Larger 16-bit arrays can handle up to full HD resolution (1920x1080x30fps).

¹ The figures specify the number of ALU-PAEs, RAM-PAEs and Function-PAEs

1.1 Design Flow and Algorithm Partitioning

The design flow for an XPP-III Processor comprising Function-PAEs (FNC-PAE) and an XPP dataflow array (XPP-Array) is very similar to the typical DSP design flow: we start off with C code compiled for the FNC-PAE and profile it on typical input data. From the profile we select the hot-spots and accelerate them. On a DSP this means either selecting an existing fully optimized library module (like DCT or FFT), or manually optimizing the code. The same holds for XPP: either we select an existing fully optimized library module or manually optimize the code.

According to the described design flow, the following dataflow algorithms of the video decoders are selected for acceleration on the XPP-Array:

- Rescaling or inverse quantization (Q^{-1}).
- Inverse signal transformation (T^{-1}).
- Motion compensation and reconstruction (MC).
- Deblocking filter.

These algorithms are analyzed in detail in Section 2 of this document. The following sequential algorithms are running on FNC-PAE:

- Entropy decoding.
- XPP-Array control (reconfiguration and data transfer).
- Purely sequential parts of above mentioned algorithms, for example H.264 Inter Prediction and parts of H.264 deblocking.
- General system/DMA/Address Generators control, house keeping, I/O communication, interrupt handling, booting other FNC-PAE cores etc.

Optimized implementations of the algorithms mentioned above are available from PACT for several video codec standards as a library. The video codecs are partitioned into a number of modules executed fully pipelined on multiple FNC-PAEs and the XPP-Array.

1.2 XPP-Array Reconfiguration

The algorithms selected for the dynamically reconfigurable XPP-Array are partitioned into multiple configurations, which are sequentially loaded and executed at run-time. Each *XPP Configuration* describes the operation of the ALUs and their interconnection.

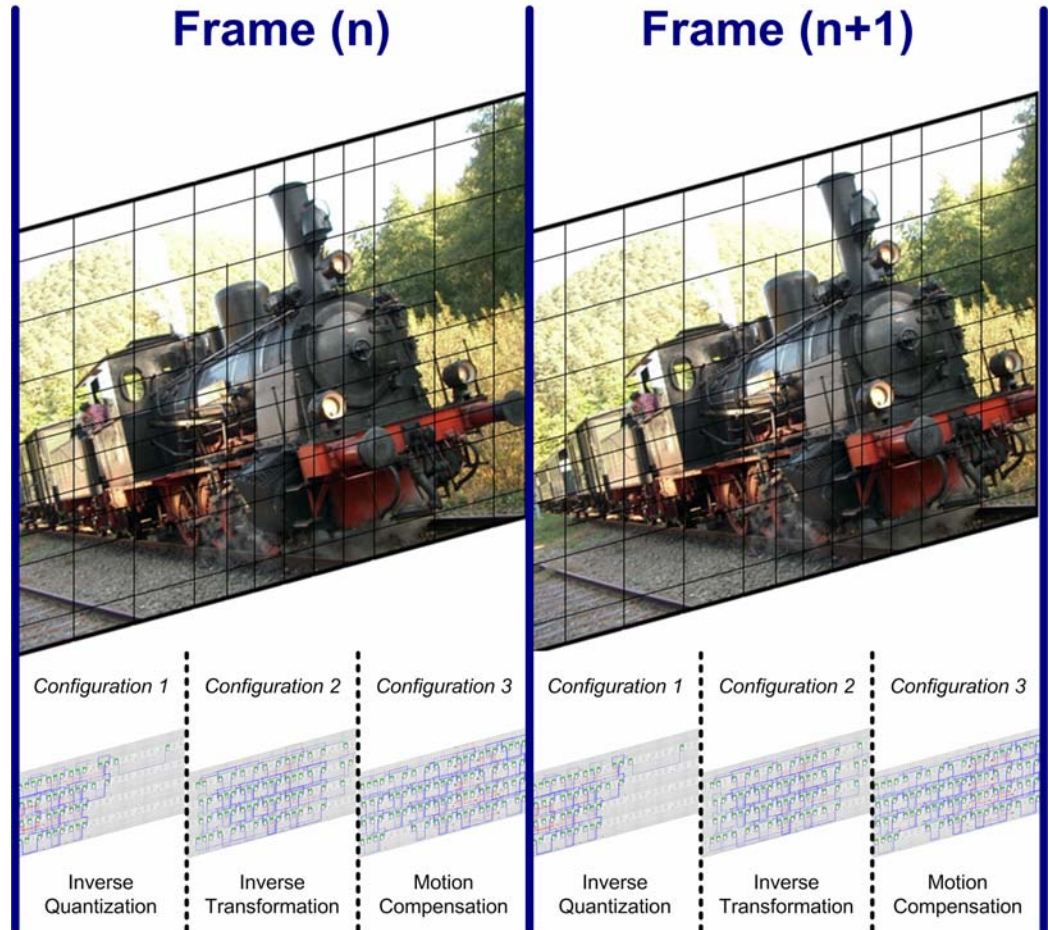
The example below illustrates how an XPP-Array can be reconfigured at runtime. The FNC-PAEs fully control the XPP-Array operation by sending reconfiguration requests and parameters as motion vectors or block offsets.

Assuming inverse quantization, inverse transformation, and motion compensation are partitioned into three configurations C1, C2, and C3, the computations are scheduled as follows:

1. Configure the XPP-Array with C1. The XPP-Array is computing inverse quantization on several blocks of data, and storing intermediate values in a temporary buffer.
2. Reconfigure the XPP-Array with C2. The XPP-Array is reading back several blocks of data from the temporary buffer, computing inverse transformation, and storing intermediate values in a temporary buffer.
3. Reconfigure the XPP-Array with C3. Again, the XPP-Array is reading back data from the temporary buffer, computing motion compensation, and storing the reconstructed blocks in the frame buffer.
4. If more data is going to be processed, start over with C1.

The diagram on the next page depicts the reconfiguration of the XPP-Array. The three XPP Configurations represent C1, C2 and C3 respectively. For decoding one frame, each of the configurations is loaded into the XPP-Array one after another. After the last configuration has completed, the whole process starts over for decoding the next frame.

For simplicity, the diagram shows all configurations working on a complete frame before reconfiguring XPP-Array, requiring a large buffer for storing intermediate results. In order to reduce the buffer size, we can iterate through all configurations multiple times per frame, according to the description above. See Section 3.2 for actual buffer sizes and XPP operating frequencies.



The trade-off between reconfiguration overhead and buffer size is typical for XPP applications. For more details on XPP-Array reconfiguration, please refer to the PACT White Paper *Reconfiguration on XPP-III Processors* available from <http://www.pactxpp.com/>.

2. Modules for Video Decoding

This chapter discusses the implementation of key components of video decoders on an XPP-III Processor.

2.1 Entropy Decoding

For entropy coding, all video codec standards considered in this paper use a combination of Huffman and run-length coding. Only H.264 optionally features adaptive arithmetic coding.

All entropy codecs have in common that the decoding algorithms are inherently sequential and cannot benefit from parallelism and pipelining on the XPP-Array. The FNC-PAEs are high-performance, VLIW-like sequential processors, designed to handle algorithms like entropy codecs highly efficiently.

The computationally most demanding entropy codec is the Context Adaptive Binary Arithmetic Codec (CABAC) defined in the H.264 standard. One FNC-PAE can decode video streams up to 30 Mbits/s. By combining two FNC-PAEs, streams up to 50 Mbits/s can be decoded.

FNC-PAE is the only fully programmable core supporting CABAC for HD-level streams as defined in the BlueRay-Disk and HD-DVD standards.

2.2 Rescaling or Inverse Quantization

The rescaling or inverse quantization is the first decoding step implemented on the XPP-Array. From the quantized values received from the entropy decoder, the original values are reconstructed and then passed to inverse transformation.

It is due to quantization that the video codecs are called “lossy”, because the result after rescaling is not the exact original value, but the closest value from a set defined by the quantization algorithm.

Rescaling typically comprises one multiplication with a scaling parameter stored in XPP internal RAM (RAM-PAE), followed by normalization and rounding. This consumes very few resources on XPP, so the rescaling is typically not run as an individual XPP Configuration, but it is combined with inverse transformation. This means that rescaling is computed “on-the-fly” at almost no cost, only adding minor latency to inverse transformation.

2.3 Inverse Transformation

All codec standards considered in this paper perform an IDCT-like inverse transformation on the values received from rescaling. The transformation is generally performed on parts of a 16x16 macroblock, typically on 8x8 or 4x4 blocks. The details of IDCT approximation and the transformation block size vary between the standards, but the general structure is always the same.

The two-dimensional inverse transformation is sub-divided into two stages, applying a one-dimensional transformation horizontally and vertically. An XPP12.6.3 provides enough resources for configuring one stage at a time, computing the inverse transformation with two separate XPP Configurations. On XPP20.8.4, both stages run fully pipelined in a single XPP Configuration. On XPP40.16.8, two transformations can be computed in parallel, providing twice the throughput of XPP20.8.4.

Since the configurations for rescaling are quite small, they are integrated with the inverse transformation into a single configuration, computing the rescaled coefficients on-the-fly. The table below summarizes the performance.

Note that the throughput is identical for all standards. For VC-1 the latency is significantly higher, due to additional XPP reconfiguration for different transformation block sizes (8x8, 8x4, 4x8, and 4x4).

	XPP Size	Cycles/ Macroblock	Latency ¹ (Cycles)
MPEG 2	XPP12.6.3	768	1300
	XPP20.8.4	384	1150
	XPP40.16.8	192	2200
H.264	XPP12.6.3	768	1000
	XPP20.8.4	384	850
	XPP40.16.8	192	1700
VC-1	XPP12.6.3	768	3600
	XPP20.8.4	384	3000
	XPP40.16.8	192	6400

2.4 Motion Compensation

Motion compensation is the key feature for compressing video sequences with high quality at low bitrates. From the encoding point of view, the general idea is to find in a previously encoded frame an area that is very similar to the currently encoded block (reference area) and then transmit only the differences between reference area and current block. Such kind of similar areas are very common in video sequences, as consecutive frames often show the same content, only with parts of it moved to a slightly different position.

More technically, the encoder identifies a *reference area* in one of the previously encoded frames and computes a *motion vector* pointing from the *current block* to the reference area (motion estimation). Given the motion vector, a *predicted block* is computed. The difference between the predicted block and the current block is called

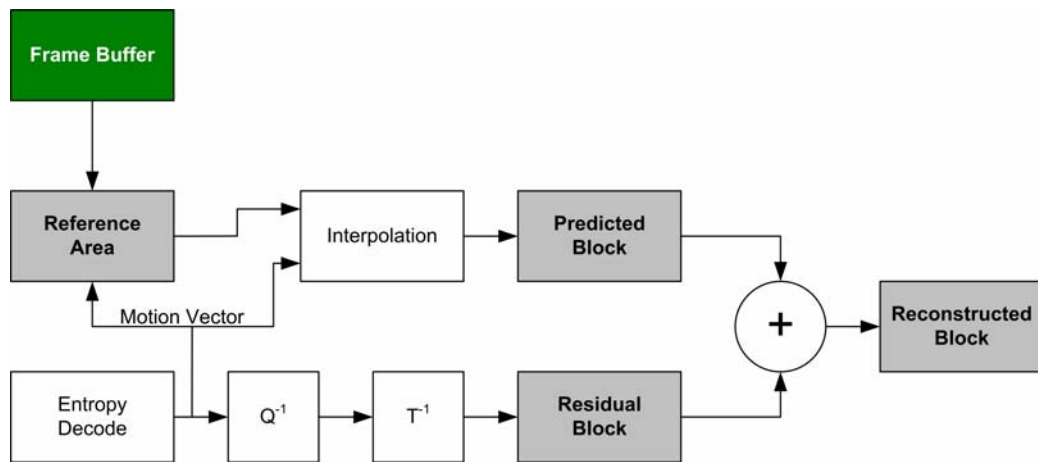
¹ Latency includes time for XPP reconfiguration and for filling the pipeline, i.e. after the Latency startup cycles a new macroblock will be transformed every N cycles (N given in column "Cycles/Macroblock").

residual block. This residual block is then passed to forward transformation and quantization for encoding.

The decoder has to carry out exactly the inverse operation. Given the same motion vector (encoded in the stream), it computes exactly the same predicted block as used in the encoder. The reconstructed block is computed by adding the residual block received from inverse transformation to the predicted block.

A motion vector is not restricted to integer values, as all standards allow half-pixel resolution, some even quarter-pixel resolution. When computing the predicted block, the pixel values at half- and quarter-pixel positions are interpolated from neighboring pixels at integer positions read from the reference area in the frame buffer.

The following figure depicts the reconstruction process.



Though the general concept described above applies to all video codecs considered in this paper, the details of interpolating the predicted block are largely different among the standards. This leads to very different computational effort and I/O bandwidth requirements.

The most obvious difference is that MPEG2 and MPEG4 Simple Profile (MPEG4/SP) allow motion vectors with half-pixel resolution, while MPEG4 Advanced Simple Profile (MPEG4/ASP), H.264, and VC-1 allow quarter-pixel resolution.

Another important difference is the kind of interpolation used. MPEG2, MPEG 4/SP, use simple averaging for interpolation, VC-1 uses either bilinear or bicubic interpolation, and MPEG 4/ASP and H.264 use a small FIR filter for interpolating half-pixel positions followed by averaging for quarter-pixel positions.

The table below shows the number of pixels processed for interpolating an $N \times M$ block together with the smallest allowed block size.

	Input Pixels per $N \times M$ Block	Smallest Block Size
MPEG 2	$(N+1) \times (M+1)$	8x8
H.264	$(N+5) \times (M+5)$	4x4
VC-1 bilinear	$(N+1) \times (M+1)$	4x4
VC-1 bicubic	$(N+3) \times (M+3)$	4x4

The block size and the numbers of pixels processed per block have a huge impact on the performance; see the table on the next page.

The values in the next table apply to P-frames with one reference area per reconstructed block. In B-frames, any block might be reconstructed from multiple reference areas. For VC-1, we assume bicubic interpolation.

Note that some of the adders in the interpolation filters exceed the 16-bit word size of XPP. In that case, we build 32-bit adders using two XPP ALUs. The carry is propagated using the XPP event network.

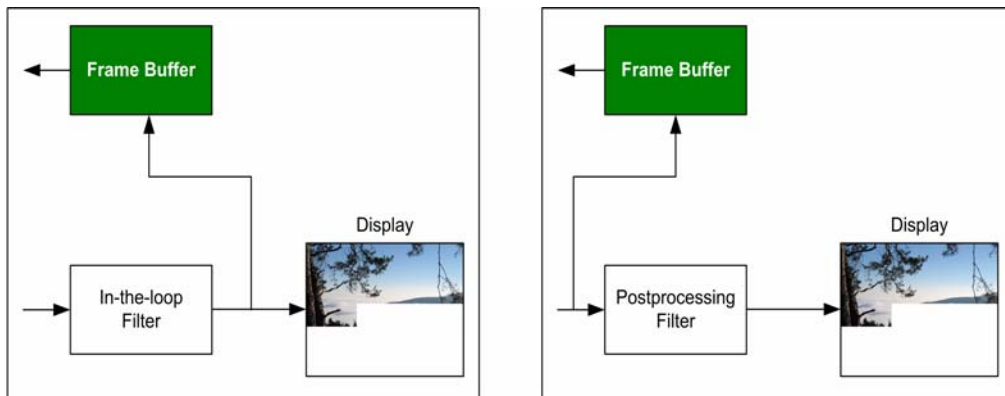
	XPP Size	Cycles/ Macroblock	Latency¹ (Cycles)
MPEG 2	XPP12.6.3	270	627
	XPP20.8.4	162	936
	XPP40.16.8	162	1003
H.264 4x4	XPP12.6.3	2592	650
	XPP20.8.4	1296	1192
	XPP40.16.8	720	2300
H.264 8x8	XPP12.6.3	1352	650
	XPP20.8.4	676	1477
	XPP40.16.8	364	2000
VC-1	XPP12.6.3	1176	750
	XPP20.8.4	672	1450
	XPP40.16.8	336	2700

¹ Latency includes time for XPP reconfiguration and for filling the pipeline, i.e. after the Latency startup cycles a new macroblock will be reconstructed every N cycles (N given in column "Cycles/Macroblock").

2.5 Deblocking and Deringing Filters

Due to the intrinsic quantization, all advanced video codecs are “lossy”. The reconstructed video sequence is “close to” the original sequence, but they are not identical. At lower bit-rates, the coarser quantization makes the boundaries of the transformation blocks visible as artifacts. For reducing these well-known artifacts, special deblocking and deringing filters have been designed.

Depending on the standard, these filters are applied as “in-the-loop” filters or as postprocessing filters. The difference is that with “in-the-loop” filters the filtered frame is used for future reference, while with postprocessing filters, the unfiltered frame is used for reference. Consequently, in-the-loop filters are a mandatory part of the respective standard (e.g. H.264, VC-1), while postprocessing filters are always optional (e.g. MPEG2/4). The following diagram visualizes this difference:



Postprocessing vs. "in-the-loop" Filter

The mandatory in-the-loop filters are always deblocking filters, while the optional postprocessing filters allow choosing deblocking, deringing or both.

The filtering is partitioned between the XPP-Array and the FNC-PAEs. The main filtering algorithm is mapped onto the XPP-Array, while the FNC-PAEs compute parameters and execute some rare special cases.

	XPP Size	Cycles/ Macroblock	Latency ¹ (Cycles)
H.264	XPP12.6.3	1376	1300
	XPP20.8.4	1376	1400
	XPP40.16.8	688	2400
VC-1	XPP12.6.3	1376	1400
	XPP20.8.4	1376	1600
	XPP40.16.8	688	2600

¹ Latency includes time for XPP reconfiguration and for filling the pipeline, i.e. after the Latency startup cycles a new macroblock will be filtered every N cycles (N given in column “Cycles/Macroblock”).

3. Results

This chapter summarizes results for video decoding on an XPP-III Processor, focusing on frequency and memory requirements.

3.1 Frame Buffer Size

All advanced video codecs refer to previously decoded frames (“reference frames”) for prediction and reconstruction. The size of one frame in memory is computed as the number of pixels times 1.5 (for the color components) times 8 bits. For example frames with full HD resolution require $1920 \times 1080 \times 1.5 \times 8 \text{ bits} = 23.73 \text{ Mbits}$ of memory.

The number of reference frames varies between the different codec standards. MPEG2/4 and VC-1 use two reference frames, while H.264 allows up to 15 reference frames.

# Frames	Frame Buffer Memory Size		
	640 x 480	720 x 480	1920 x 1080
3	10.6 Mbit	11.9 Mbit	71.2 Mbit
5	17.6 Mbit	19.8 Mbit	118.7 Mbit
10	35.2 Mbit	39.6 Mbit	237.3 Mbit
16	56.3 Mbit	63.3 Mbit	379.7 Mbit

3.2 XPP Frequency and Local Buffer Size

Based on the performance numbers for the XPP Configurations shown in Section 2, the required XPP frequency for decoding video streams with various resolutions and codec standards can be computed. The tables on the next pages show results for CIF, VGA, SD/D1, and some HD resolutions. For each resolution several standards and XPP sizes are listed.

Please note that the required frequency depends also on the size of the local buffer used for storing intermediate results. The smaller the local buffer, the higher is the overhead for XPP reconfiguration, and hence the higher is the required frequency. Therefore each table on the following pages lists the respective size of the local buffer.

The trade-off between local buffer size and frequency is illustrated for VGA resolution, providing two tables with different local buffer sizes. For more details on the trade-offs offered by XPP reconfiguration, please refer to the PACT White Paper *Reconfiguration on XPP-III Processors* available from <http://www.pactxpp.com/>.

VGA (640x480), 30 fps Local Buffer: 240 Kbit			
	XPP12.6.3	XPP20.8.4	XPP40.16.8
MPEG 2	45 MHz	25 MHz	19 MHz
H.264	237 MHz	148 MHz	80 MHz
VC-1	149 MHz	106 MHz	59 MHz

VGA (640x480), 30 fps Local Buffer: 60 Kbit			
	XPP12.6.3	XPP20.8.4	XPP40.16.8
MPEG 2	50 MHz	31 MHz	28 MHz
H.264	242 MHz	156 MHz	92 MHz
VC-1	161 MHz	118 MHz	84 MHz

SD/D1 (720x480), 30 fps Local Buffer: 270 Kbit			
	XPP12.6.3	XPP20.8.4	XPP40.16.8
MPEG 2	50 MHz	28 MHz	21 MHz
H.264	266 MHz	166 MHz	90 MHz
VC-1	167 MHz	119 MHz	66 MHz

HD (1280x720), 30 fps Local Buffer: 480 Kbit			
	XPP12.6.3	XPP20.8.4	XPP40.16.8
MPEG 2	132 MHz	72 MHz	53 MHz
H.264	n/a	325 MHz	174 MHz
VC-1	n/a	312 MHz	165 MHz

HD (1920x1080), 30 fps Local Buffer: 720 Kbit			
	XPP12.6.3	XPP20.8.4	XPP40.16.8
MPEG 2	300 MHz	162 MHz	117 MHz
H.264	n/a	n/a	390 MHz
VC-1	n/a	n/a	366 MHz

4. Conclusion

Though video coding schemes are based on similar fundamental principles, their specific implementation is rather different. The coarse-grained reconfigurable XPP-III Processor provides a new programmable approach to combine the flexibility that is required for multi-standard video codecs with high computational performance.

XPP-III is the only fully programmable solution for decoding high-quality HD video streams like BlueRay-Disk or HD-DVD.

This Video Decoding White Paper described the methods and results of an implementation on an XPP-III Processor consisting of the reconfigurable XPP dataflow array and sequential Function-PAEs. The provided tables may advise system designers conceiving a system architecture, which is driven by the requirements of the intended top-level target application such as mobile video or HDTV only. The important parameters are the size of the XPP-Array as well as the RAM and bandwidth requirements.

XPP designs adapt through software to a wide application space, while conventional state-of-the-art hardwired solutions are tailored to a few predefined applications. Thus XPP's high performance and fast reconfiguration enable product differentiation, e.g. improved video postprocessing within the given application space can be designed and adapted during the lifetime of a product. Though we discussed only the decoding part of video streams, it is a small step to extend the software also towards the more demanding encoding part, e.g. for video phones or HDTV camcorders.