Performance Requirements for Reconfigurable Hardware for a Scalable Wavelet Video Decoder

Harald Devos, Hendrik Eeckhaut , Mark Christiaens, Fabio Verdicchio, Dirk Stroobandt and Peter Schelkens

Ghent University, ELIS
Sint-Pietersnieuwstraat 41
9000 Gent, Belgium
{heeckhau,hdevos,mchristi,dstr}
@elis.ugent.be

Vrije Universiteit Brussel, ETRO
Pleinlaan 2
1050 Brussel, Belgium
{fverdicc,pschelke}
@etro.vub.ac.be

Abstract—**Nowadays multimedia applications emerge on portable devices everywhere. These applications typically have a number of stringent requirements. A first requirement is a high amount of computational power together with real-time performance. A second requirement is that they must be reconfigurable i.e. that the application or the characteristics of the application can be modified at will. The performance requirement often drives the design towards a hardware implementation while the reconfigurability requirements are better served by a software implementation. In this paper we try to reconcile these two requirements by exploring a third implementation option: FPGAs. By using an FPGA to implement the performance critical parts of an application we can achieve both the performance and the flexibility that is needed for multimedia applications. As a proof of concept we are designing a scalable wavelet video decoder. Through analytical means we explore the question of whether an FPGA implementation of this codec can be used to reach the performance goals and what their resource requirements are. We find that modern FPGAs offer enough computational power to obtain real-time performance of the decoder, but that reaching the necessary memory bandwidth will be a challenge during this design.**

Keywords—**Multimedia, reconfigurable hardware, scalable wavelet video, motion compensated temporal filtering, HW resource estimation**

I. Introduction

When designing a new multimedia system one is quickly faced with the question which part of its functionality needs to be implemented in hardware. This is not an easy choice. Usually one would prefer to implement most of the functionality in software running on a general purpose processor since this improves the ease and flexibility of the design process and avoids the high cost of designing an ASIC. Still, often the performance requirements of the multimedia system are such that specific hardware is a necessity.

Another important disadvantage of using ASICs instead of a software implementation is that their functionality is cast in stone. For example an ASIC designed for doing JPEG compression utilizing Huffman encoding will become useless when application changes require arithmetic encoding. This can result in an overdimensioning of the ASIC so that it can accommodate all the required operations of the multimedia system although these operations may only be needed infrequently. In software such overdimensioning has little consequence but in hardware it increases the complexity of the design and the production cost.

In the last decade a new class of hardware devices has emerged which holds the middle between ASICs and general purpose processors: FPGAs (field programmable gate arrays) [3]. These devices contain a large amount of gates whose specific functionality and interconnection can be reprogrammed by updating SRAM memory blocks on the FPGA chip. When doing so one gets on the one hand a device with a performance which leans towards an ASIC design. On the other hand the device can still be reconfigured giving it some of the flexibility of a software implementation.

In the RESUME project (Reconfigurable Embedded Systems for Use in scalable Multimedia Environments [1]) we explore the usefulness of FPGAs when designing multimedia systems. In order to do so we are building (among others) a *scalable* wavelet based video decoder [6], [8]. By "scalable" we mean that it is designed to allow us to easily change the quality of service (QoS) i.e. the framerate, resolution, color depth, ... of the decoded video without having to change the video stream used by the decoder (except for skipping unnecessary blocks of data without decoding) or having to decode the whole video stream if only part of it is required.

Such a scalable video decoder has advantages for both the server (who provides video streams) and the clients. On the one hand the server scales down since it only needs to produce one video stream for all clients. On the other hand the client can easily change the decoding parameters

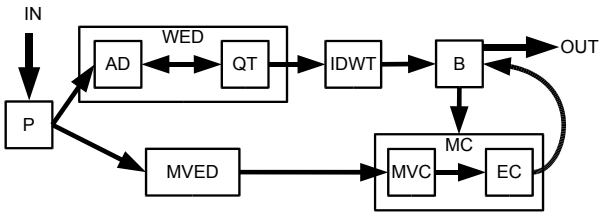

Fig. 1. High-level overview of the video decoder.

to optimize the use of the display, the required processing power, the required memory, ...

A scalable video decoder is a perfect fit for the use of FPGAs. Indeed if we were to make an ASIC implementation then we would have to overdimension the ASIC since it would need to be capable of producing the highest QoS, even if this is not required at all times. A software implementation would also be a suboptimal choice simply because we could not deliver the required performance. An FPGA implementation however can be reconfigured each time the QoS requirements change.

Our current goal in the project is to design the scalable video decoder in such a way that we can generate semi-automatically a whole range of different configurations of the FPGA by simply changing the QoS requirements that need to be met. These implementations will form a near Pareto-optimal set of configurations that each have optimal performance for a certain set of requirements and a certain cost-function.

In the remaining of this paper we present an overview of the decoder in Section II and point out its scalability features in Section III. In Section IV we present the results of our efforts to profile the software version of the decoder. In Section V we show some preliminary performance estimates. Finally in Section VI we detail the future evolution of our project and in Section VII we present some conclusions.

II. System Overview

Figure 1 shows a high-level overview of the wavelet video-decoder. It consists of the following parts:

P: The "parser" receives a bit stream representing a compressed video. It analyzes the structure of this stream and is capable of extracting relevant parts and dropping other parts that are not required for the further decoding process.

WED: The "wavelet entropy decoder" decodes entropy encoded parts of the bit stream into wavelet transformed frames. The WED consists of two strongly interconnected parts: the "arithmetic decoder" (AD) and the "quadtree decoder" (QT) [6]. The QT provides the AD with continuous guidance about what type of data is to be decoded next (i.e. are we decoding a sign bit, a significance bit, ...) and the AD provides the QT with the decoded data it requires. This close cooperation improves the compression ratio that is achieved but hinders the independent implementation of these components.

IDWT: The "inverse wavelet transform" takes a wavelet transformed frame and does an inverse wavelet transform producing either a reference image (the first image of a group of pictures) or an error-frame.

MVED: "Motion vector entropy decoding" performs the entropy decoding of the motion vectors.

MC: "Motion compensation" does the actual reconstruction of the final video frames. It is composed of two subparts: the "motion vector compensation" (MVC) and the "error correction" (EC). The MVC takes 2 reference frames and a set of motion vectors. It constructs a first estimate of the resulting video frame by combining translated blocks of the reference frames. The corresponding error frame is added in the EC in order to 'fine-tune' the first estimate and to produce the final image. The motion vectors have an integer and a fractional part. Fractional translation of blocks is obtained through interpolation of the original pixels.

B: The "reorder buffer" puts the decoded frames in the correct order resulting in the final video stream.

III. Exploiting the Scalability

We can distinguish two types of scalability: the scalability of the QoS of the decoder and the scalability of the hardware cost. The former may include the framerate, the resolution of the frames, the accuracy (cfr. PSNR - Peak Signal to Noise Ratio), the availability of colors, ... The latter contains parameters like used chip area, power consumption, used network bandwidth, ... It is clear that all these parameters are not independent of each other.

We will focus here on the parameters that describe the scaling of the QoS of the video and describe their influence on the calculation cost.

Frame rate: Different frame rates are possible through a dyadic temporal decomposition [7] of the video stream. This is illustrated in Figure 2. The decoder can choose to which (temporal) level the stream is decoded. Each extra level doubles the frame rate. Because the frames are grouped in GOPs (Group Of Picture) of 16 frames, the possibilities are 30, 15, 7.5, 3.75 and 1.875 frames per second.

Resolution: In the encoder a 2-D discrete wavelet transform (DWT) in K levels is performed on the error and reference frames. The 0^{th} level has the total frame as input and generates four subbands LL, HL, LH and HH. In the next levels the same action is performed with the LL-

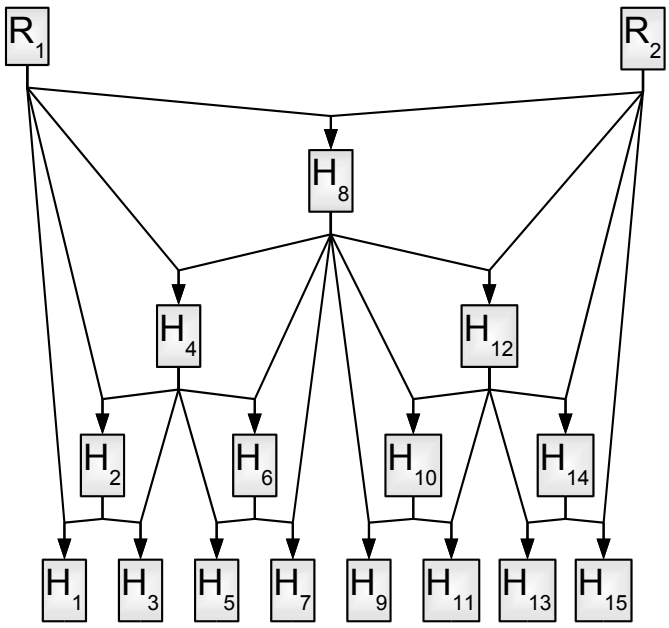

Fig. 2. The temporal decomposition of one GOP. The arrows illustrate which frames are used to reconstruct (via motion compensation) other frames at lower decomposition levels. R_1 and R_2 are reference frames, the H_i are reconstructed (intermediate) frames.

subband of the previous level as input. Each LL-subband is a low resolution version of the original frame. The inverse discrete wavelet transform (IDWT) in the decoder performs the inverse transformation. However, the IDWT can stop at an arbitrary level. This results in a lower resolution (proportional to the number of performed levels) of the video. This way the computation time of the IDWT is reduced. In addition the wavelet entropy decoding (WED) of the unused highfrequency subbands can be omitted.

Image quality: The quadtree entropy decoder decodes a bit stream containing the quantised wavelet encoded frames, bit plane per bit plane, starting with the most significant plane. Resources can be economized when some less significant bit planes are omitted. This results in a lower quality (measured by e.g. the PSNR) of the decoded frames.

IV. A ROUGH PROFILE OF THE DECODER

In this section we try to obtain a rough estimate of where the computationally expensive parts of the decoding system are located. To obtain a rough estimate of the execution time of the system we profiled the decoding algorithm based on a preliminary C/C++ implementation (on a standard PC[1]). Note that we are dealing here with prototype code, written for ease of modification, not for optimal speed. Therefore the results in this section are only intended as a rough, first estimate of the computational costs of the different blocks of the decoder.

[1]Pentium IV 2.0 GHz, 512MiB RAM

A. PSNR

Execution time alone is not enough to evaluate our scalable codec. We also need a measure for the quality of decoded sequences: the PSNR of the decoded frames vs. the original frames. The PSNR is defined as follows. If $Y_{i,j}$, $U_{i,j}$ and $V_{i,j}$ and $Y'_{i,j}$, $U'_{i,j}$ and $V'_{i,j}$ are resp. the luminance and the two chrominance channels of a YUV 4:2:0 image and its reconstructed version both with resolution $r \times s$ then the PSNR is defined as follows:

$$10 \log_{10} \frac{255^2 \frac{3}{2} rs}{\sum (Y - Y')^2 + \sum (U - U')^2 + \sum (V - V')^2} \quad (1)$$

The PSNR gives us a number for the quality of one decoded frame; for the quality of the entire video stream we use the mean PSNR. This works fine if we are dealing with sequences with the same frame rate. But since we are dealing with temporal scalability we run into problems when the original and decoded sequence no longer have the same frame rate. In this case, there are two options: reduce the frame rate of the original sequence or inflate the rate of the decoded sequence before the PSNR is calculated. The former is called *'Reduced Frame Rate (RFR) PSNR'* and actually ignores the framerate. It assumes that the lower frame rate was the wish of the user and is not an admission of quality. The latter, *'Full Frame Rate (FFR) PSNR'*, considers a lower framerate as lower quality. Both approaches are used in the following experiments. The inflation of the frame rate is illustrated in Figure 3.

Two different (encoded) sequences were decoded for a number of different bitrates while measuring the execution time and the resulting PSNR. In this article the bitrate is defined as the number of (decoded) bits per pixel. Note that this is different from the number of bits per second. Two streams with the same bitrate but a different frame rate for example will not have the same number of bits per second because the total number of pixels is different.

The considered sequences are reference sequences widely used for video coding research:

- *Foreman* : 288 CIF (352×288 pixels + $2 \times 176 \times 144$ pixels for the color channels) frames (@30Hz$\Rightarrow 9.6$s)
- *Mobile CIF* : 288 CIF frames (@30Hz$\Rightarrow 9.6$s)

The results for the foreman sequence are illustrated in Figures 4, for reduced frame rate, and 5, for full frame rate. The PSNR is naturally the same for both approaches when all frames are decoded (time level 4). When the quality is measured with the 'full frame rate PSNR' the effect of a

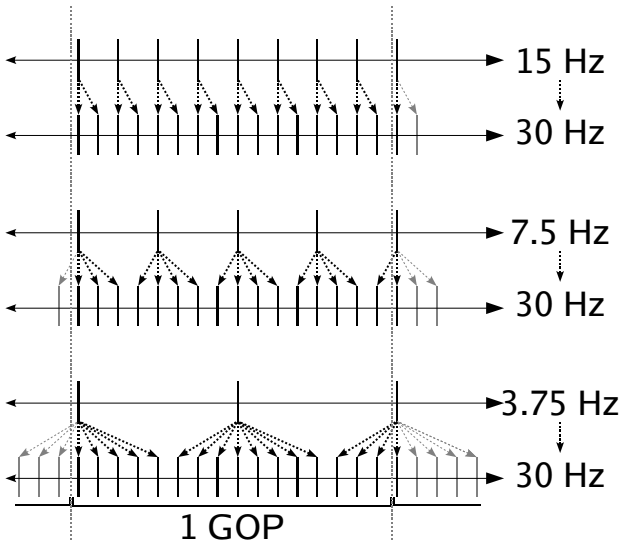

Fig. 3. Inflation of the framerate. Frames are duplicated to obtain 30 frames per second.

lower frame rate is clearly visible. For 'reduced frame rate PSNR' on the contrary, we observe no significant impact on the PSNR. The results for the mobile sequence, illustrated in Figures 6 and 7, present the same characteristics. But since the mobile sequence is more complex, the PSNR increases slower for increasing bitrate.

In Figure 8 we plotted the execution time of the three major time consuming blocks of the algorithm versus the bitrate. Only the Wavelet Entropy Decoding (WED) depends on the bitrate. For higher bitrates this is the most time consuming block of the decoding algorithm. Motion Compensation (MC) is independent of the framerate but varies for different sequences. It varies from frame to frame dependent of the actual values of the motion vectors. The share of the Inverse Discrete Wavelet Transform (IDWT) is rather limited and independent of both bitrate and sequence; each frame needs exactly the same number of computations.
Here again we see that the 'mobile' sequence is more complex than the 'foreman' sequence.

B. Experiment

In Figure 4 and following we could see that we are far from a real time (software) implementation. Therefore we experimented with different configurations of the decoding algorithm. Each configuration had three parameters:

Interpolation: Interpolation is used to calculate subpixel shifts of reference blocks during motion vector compensation. This is a computationally expensive part of the

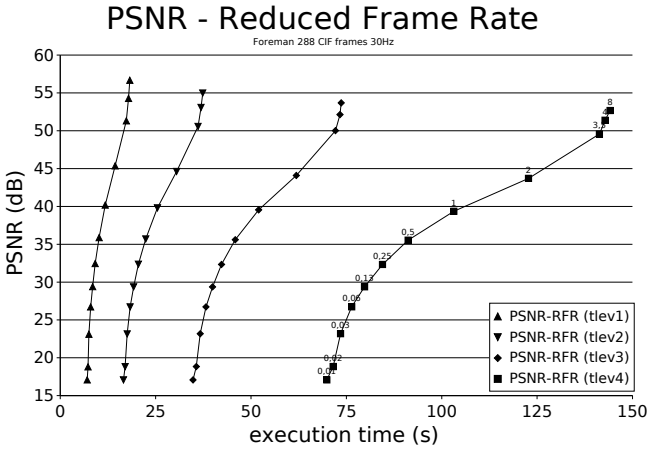

Fig. 4. The Reduced Frame Rate PSNR for the foreman sequence. The bitrates (bpp) are only labeled for the curve of sequences decoded at maximum frame rate (time level 4); the values for the other curves are the same. It obviously takes more time to decode more frames and the higher the bitrate the better the quality.

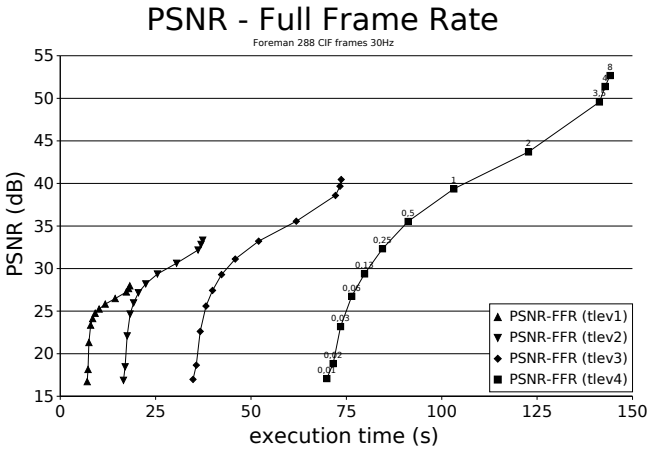

Fig. 5. The Full Frame Rate PSNR for the foreman sequence. The effect of a lower frame rate (less time levels) is now clearly visible in the attainable PSNR.

algorithm that can be skipped, at the expense of a decoder reconstruction drift.

ErrorFrameCorrection: After motion vector compensation, an error frame is added to the reconstructed frame to correct deviations. For each errorframe a wavelet entropy decoding and inverse wavelet transform is necessary. Omitting error frame correction completely (i.e. switching off the quality scalability) results in a much faster execution but of course also in less quality.

Bitrate: This is number of decoded bits per pixel (same

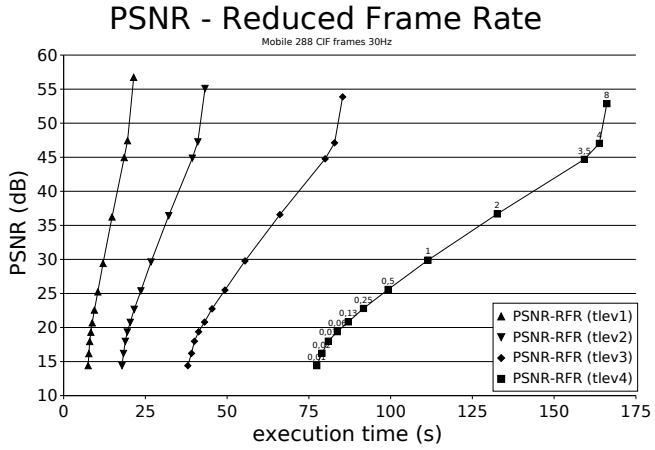

Fig. 6. The Reduced Frame Rate PSNR for the mobile sequence.

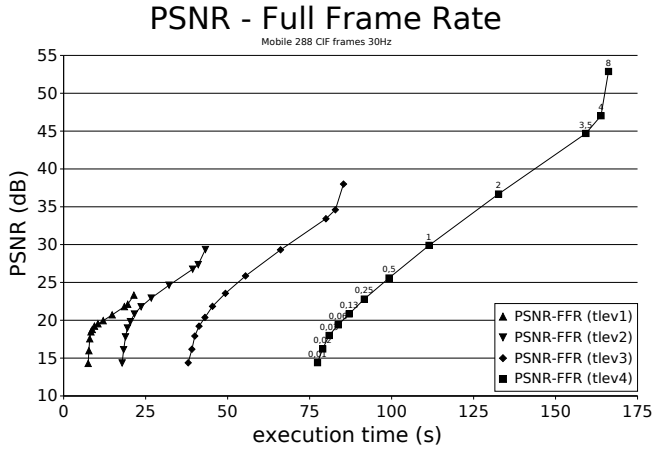

Fig. 7. The Full Frame Rate PSNR for the mobile sequence.

for reference and compensated frames).

In Figure 9 we can see the results for the 'foreman CIF' sequence. The configurations are ordered along increasing mean PSNR. We are decoding at 30Hz (time level 4); in consequence the FFR PSNR is the same as the RFR PSNR. Note also that real-time decoding (in less than 33ms/frame) is already possible for a few configurations. The more complex 'mobile' sequence (Figure 10) needs more time for a similar PSNR, but the trend is similar.
In our implementation we used the same bitrate for reference and error frames. This is certainly not the best choice. From Figures 9 and 10 we can conclude that another distribution of the bitrate between reference frames and error frames is desirable [2]. If we look for example at the PSNR of I_E_0.5 (16 frames at 0.5 bits per pixel) and

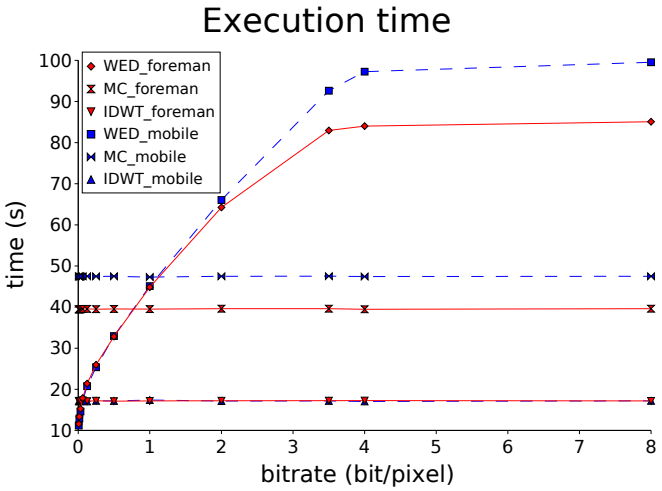

Fig. 8. The execution time for Wavelet Entropy Decoding (WED), Inverse Discrete Wavelet Transform (IDWT) and Motion Compensation (MC) for both the 'foreman' and the 'mobile' sequence.

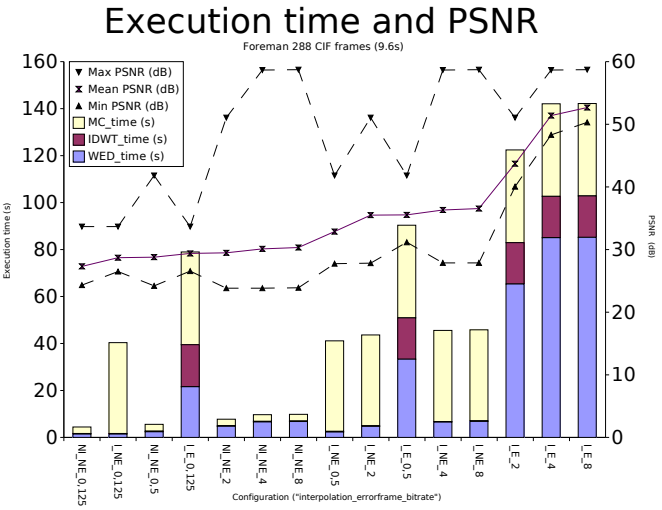

Fig. 9. Execution times of the Wavelet Entropy Decoding (WED), Inverse Discrete Wavelet Transform (IDWT) and Motion Compensation (MC) for decoding 288 (9.6 s) CIF YUV frames of the 'foreman-sequence' for a number of different configurations. Each configuration has three parameters: ErrorFrameCorrection (E) or No ErrorFrameCorrection (NE); Interpolation (I) or No Interpolation (NI) during Motion Compensation and the bitrate. The number of decoded levels was fixed (4).

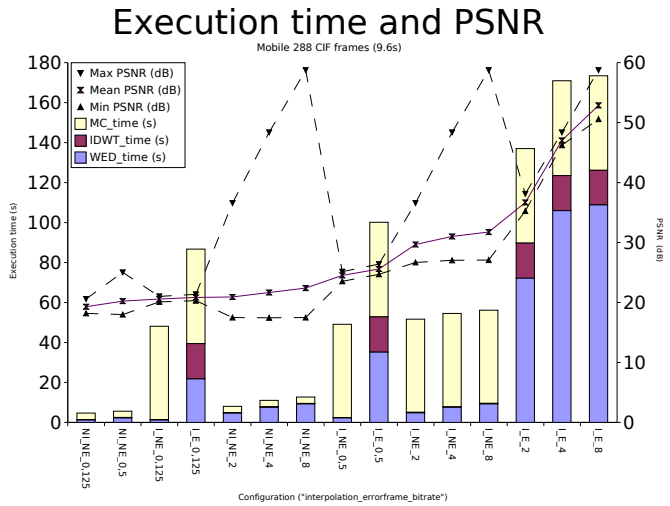

Fig. 10. Same experiment as Figure 9, now for the 'Mobile' sequence .

I_NE_4 (1 frame at 4 bits per pixel and 15 frames at 0 bits per pixel), we see that although the total bitrate is half as small, the PSNR is slightly better. The reference frames do contain more information so they should be alloted more bits.

Even though this algorithm could be accelerated significantly with an implementation aimed at maximal speed, it would still be impossible to decode these sequences in real time at full frame rate and impeccable quality. It now takes about 140 seconds (without IO) to decode a CIF sequence of 288 frames. At 30 frames per second, this means we have to perform the decoding algorithm some 15 to 20 times faster. This will not be possible on a portable device with solely a more optimal software implementation.

V. Estimations

To be able to choose a hardware platform, the requirements of the application need to be known. This includes the amount of necessary memory, the bandwidth to the memory and some measures for the computing power. As the exact quantities of these are only known after implementation, estimations have to be made.

The requirements depend on and scale with the QoS of the video. Only estimations for the maximal QoS (30 frames/s, QCIF or CIF-format) are listed here because these will result in the requirements for the hardware. Estimations for other QoS are interesting to illustrate the tradeoffs between different quality and hardware cost parameters. One of the goals of the project is the validation of these estimation techniques and the development of new estimation techniques.

The decoding algorithm was provided as Matlab/C++ code that was written for ease of developing and not for minimal execution time. This code contains a lot of unnecessary calculations and operations. The following estimations are based on the underlying algorithms, so these redundant operations are not taken into account.

Some of the blocks in the system have a very deterministic behavior (e.g. the inverse wavelet transform). They repeatedly perform the same actions on constant amounts of data. Others are extremely data dependent (e.g. quadtree); the data arrives at varying rates (compressed) and needs varying amounts of operations. For these blocks only the worst case requirements are calculated.

A. Data types

In the C code, the most frequently used data types are floating point and integer. To make a good use of the hardware, fixed point numbers and integers with limited range should be used. Therefore a study of the relation between performance and range or accuracy of the variables was made. It was shown that each fixed point number can be stored in 2 bytes or less (floating point = 4 bytes). Larger word lengths are only necessary for intermediate results and calculations on the FPGA.

B. Memory requirements

Arrays and variables in the software source are allocated for a longer time than really needed. To calculate the real memory requirements one should limit the lifetime of each array and variable to the minimum so that memory can be reused. Also the size of the data types differs from the software implementation as described in the previous paragraph.

If we assume that all components of the decoder will need to be working in parallel (e.g. in a pipelined fashion) the total memory requirement is the sum of the requirements of the different components. Table I gives an overview.

C. Computational requirements

Can a modern FPGA (e.g. Xilinx Virtex-II or Altera Stratix) cope with the amount of computations that needs to be performed for decoding the video stream in real-time? Let us look at the different components of the decoder:

- Previous implementations [9] of arithmetic decoders have shown that this component fits in a very small part (a few %) of a modern FPGA.
- The quadtree decoder has a negligible computational cost but will be I/O bound since it has a large number of accesses to a large data structure with bad locality.

• The inverse wavelet transform has a good temporal and spatial locality so we believe it will be computation bound. For a wavelet transform of degree 4 using the Bior 5.5 filter (the largest filter) we need 2.5×10^6 multiplications and 4.5×10^6 additions per CIF-frame or 754×10^6 multiplications/s and 135×10^6 additions/s.
• The motion compensation (without interpolation) performs 6.08×10^6 additions/s and 3.04×10^6 shifts/s. As shifts can be hard-wired they can be ignored.
• In the worst case, the interpolator performs 231×10^6 multiplications/s and 220×10^6 additions/s.
A Virtex-II 2000 speed grade 5 has 56 $18 - bit \times 18 - bit$ multipliers that can run at 105 MHz. This results in 5.88×10^9 multiplications/s which is more than sufficient. Additions are substantially less expensive and should not be a problem either.

D. Bandwidth requirements

An overview of the bandwidth required between components is shown in table II (the interpolator is integrated in the motion compensation). This table does not show the bandwidth between a component and its (local) memory, required for the internal operation of that component.

Here the quadtree decoder might prove to be a problem. It requires a fairly large work memory that is accessed very frequently in irregular patterns. This type of behavior excludes the possibility of efficient caching. Further experimentation has to show if a real time implementation of the quadtree decoder is feasible. We expect that the typical behavior of this component is much better than our worst case (426 MiB/s internal bandwidth) assumptions.

The IDWT may also prove difficult due to the fact that it needs to traverse the frame multiple times. In a naïve implementation (not block-based) 46.22 MiB/s is necessary.

E. Conclusion

The major conclusions are that the necessary internal bandwidth is the most restrictive bottleneck, that the targeted wavelet entropy decoder has a high compression performance but its sequential nature and memory access pattern hampers an elegant hardware implementation and that memory requirements are quite modest. Luckily, earlier work has shown that embedded entropy decoders can be implemented efficiently [4]. Modern FPGAs should offer enough calculation power to implement the video codec.

VI. Future Work

As stated in the introduction it will be possible to configure the FPGA design of the scalable video. This will effectively yield hundreds of different implementations of the video decoder that each are nearly "optimal". It is up to the user of the FPGA design to define what "optimal" actually means, i.e., low power consumption, small FPGA requirements, low memory requirements, ... given that a certain QoS must be achieved.

Optimality is clearly defined by a cost function that takes the aforementioned factors (among others) into account. It is necessary to devise techniques to find the optimum amongst all available configurations of the video decoder. This is not an easy task since it requires in theory generating all possible configurations of the video decoder for all relevant FPGA platforms and then evaluating their cost. A future goal of our project is therefore to devise techniques to evaluate the performance/cost of a configuration of the FPGA without actually having to instantiate that design. These may be based on SLIP (System Level Interconnect Prediction) techniques [5].

Another avenue of research we will investigate is how to simultaneously run multiple multimedia applications on the same FPGA or on multiple FPGAs. This will require infrastructure to allow FPGAs to be reconfigured on-the-fly. What is more, when an FPGA needs to be reconfigured we will need to be able to stop the task that is currently running on the FPGA and transfer its state. Then, once the FPGA is reconfigured we will need to transfer this state back to the FPGA and restart it.

TABLE I
Memory (B) requirements of decoder

	QCIF	CIF
Reorder buffer	228096	912384
Motion compensation	44541	172845
IDWT	76032	304128
Quadtree decoder	120384	481636
Arithmetic decoder	86888	342144
Total	555941	2213037

TABLE II
Bandwidth (MiB/s) between components

	QCIF		CIF	
	In	Out	In	Out
Reorder buffer	1.09	11.35	4.35	45.41
Motion compensation	11.84	1.09	47.36	4.35
IDWT	2.18	1.22	8.7	4.89
Quadtree decoder	< 2.18	2.18	< 8.7	8.7
Arithmetic decoder	< 2.18	< 2.18	< 8.7	< 8.7
Total	19.46	21.29	77.81	68.79

VII. Conclusions

There is an ever increasing demand for scalable multimedia applications. Software may not be able to cope with the high performance requirements. Reconfigurable hardware could be a satisfying solution; it combines flexibility with high performance.

This paper considers a scalable wavelet video decoder. Although the software implementation can be further optimized, this will not be sufficient to decode the video streams in real time. Modern FPGAs on the contrary should offer enough computational power. As shown in this paper, only the necessary bandwidth, the sequential nature and the memory access pattern of the wavelet entropy decoder may hamper the implementation.

This preparatory work is a crucial step in the exploration of the usefulness of FPGAs in multimedia systems. With the proposed evaluation of the performance/cost of FPGA implementations and the study of the reconfiguration possibilities, we will be able to fully grab the performance requirements for reconfigurable hardware to be used for scalable video decoder applications.

VIII. Acknowledgments

Harald Devos and Peter Schelkens are supported by the fund for scientific research Flanders (F.W.O.). Hendrik Eeckhaut, Mark Christiaens and Fabio Verdicchio are supported by I.W.T. grant 020174.

References

[1] The RESUME project: Reconfigurable Embedded Systems for Use in scalable Multimedia Environments. http://www.elis.UGent.be/resume.

[2] I. Andreopoulos, A. Munteanu, P. Schelkens, M. van der Schaar, and J. Cornelis. Control of the distortion variation in motion compensated temporal filtering. In *ISO/IEC JTC1/SC29/WG11, MPEG Report M9253*, Awaji island, Japan, December 7-12 2002.

[3] A. DeHon. The density advantage of configurable computing. *IEEE Computer*, 33(4):41–49, April 2000.

[4] P. Schelkens. *Multidimensional Wavelet Coding - Algorithms and Implementations*. PhD thesis, Department of Electronics and Information Processing (ETRO), Vrije Universiteit Brussel, Brussel, 2001.

[5] D. Stroobandt. *A Priori Wire Length Estimates for Digital Design*. Kluwer Academic Publishers, Boston / Dordrecht / London, April 2001.

[6] D. Taubman and A. Zakhor. Multirate 3-D subband coding of video. *IEEE Transactions on Image Processing*, 3(5):572–588, September 1994.

[7] D. Turaga and M. van der Schaar. Unconstrained motion compensated temporal filtering. In *ISO/IEC JTC1/SC29/WG11, m8388, MPEG 60th meeting*, May 2002.

[8] J. W. Woods and G. Lilienfield. A resolution and frame-rate scalable subband/wavelet video coder. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(9):1035–1044, September 2001.

[9] Y. Xie, W. Wolf, and H. Lekatsas. A code decompression architecture for VLIW processors. In *34th Annual International Symposium on Microarchitecture (MICRO'01)*, pages 66–75, December 2001.