

A Fast Wavelet Entropy Decoder for Scalable Video

Hendrik Eeckhaut

PARIS-ELIS, Ghent University
St. Pietersnieuwstraat 41, 9000 Gent, Belgium
Email: Hendrik.Eeckhaut@ELIS.UGent.be

Dirk Stroobandt

PARIS-ELIS, Ghent University
St. Pietersnieuwstraat 41, 9000 Gent, Belgium
Email: Dirk.Stroobandt@ELIS.UGent.be

Abstract—In the RESUME project (Reconfigurable Embedded Systems for Use in Multimedia Environments) we explore the benefits of an implementation of scalable multimedia applications using reconfigurable hardware by building an FPGA implementation of a scalable wavelet-based video decoder. The term ‘scalable’ refers to a design that can easily accommodate changes in quality of service with minimal computational overhead. This is important for portable devices that have different Quality of Service (QoS) requirements and have varying power restrictions.

The scalable video decoder consists of three major blocks: a Wavelet Entropy Decoder (WED), an Inverse Discrete Wavelet Transformer (IDWT) and a Motion Compensator (MC). The WED decodes entropy encoded parts of the video stream into wavelet transformed frames. These frames are decoded bitlayer per bitlayer. The more bitlayers are decoded the higher the image quality (scalability in image quality). Resolution scalability is obtained as an inherent property of the IDWT and frame rate scalability is acquired through hierarchical motion compensation.

In this paper we present the results of our investigation into the hardware implementation of such a scalable video codec. In particular we found that the implementation of the entropy codec is a significant bottleneck. We present an alternative, hardware-friendly algorithm for entropy coding with superior data locality (both temporal and spatial), streaming capabilities, a high degree of parallelism, a small memory footprint and superior compression while maintaining all required scalability properties. We support these claims with synthesis results of an effective hardware implementation on an FPGA. The design was elaborated in VHDL and co-simulated with an extensive System-C testbench. We obtained a real compact implementation which can easily decode the required 50 million symbols per second.

I. INTRODUCTION

“Scalable video” is encoded in such a way that it allows to easily change the Quality of Service (QoS) i.e. the frame rate, resolution, color depth and image quality 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 without having to decode the whole video stream if only a part of it is required.

Such a scalable video codec has advantages for both the server (the provider of the content) and the clients. On the one hand the server scales well since it has to produce only one video stream that can be broadcast to all clients, irrespective of their QoS requirements. On the other hand the client can easily adapt the decoding parameters to its needs. A home cinema system can decode the stream at full quality, while a small portable client can decode the stream at low resolution and frame rate without needing the processing power of the

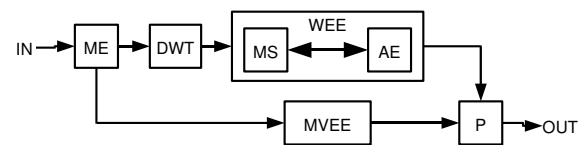


Fig. 1. High-level overview of the video encoder

larger clients. This way the decoder can optimize the use of the display, required processing power, required memory, ...

The internal structure of one implementation of a scalable encoder is shown in Figure 1 and was described in [1]–[5]. It consists of the following parts:

ME: “Motion Estimation” exploits the temporal redundancy in the video stream by looking for similarities between adjacent frames. To obtain temporal scalability (i.e. adjustable frame rate of the video), motion is estimated in a hierarchical way as illustrated in Figure 2. This dyadic temporal decomposition enables decoding of the video stream at different bitrates. The decoder can choose up to which (temporal) level the stream is decoded. Each extra level doubles the frame rate.

An intermediate frame is predicted from its reference frames by dividing it into macroblocks and comparing each macroblock to macroblocks in the reference frames. The relative position of the macroblocks in the reference frames with respect to the intermediate frame is stored as motion vectors. The difference between the predicted and the original frame is called an “error frame”.

MVEE: “Motion Vector Entropy Encoder” is responsible for entropy encoding the motion vectors.

DWT: The “Discrete Wavelet Transform” takes a reference or error frame and separates the low-pass and high-pass components of the 2D image as illustrated in Figure 3. Each LL-subband is a low resolution version of the original frame. The inverse wavelet transform (IDWT) in the decoder can stop at an arbitrary level, resulting in resolution scalability.

WEE: The “Wavelet Entropy Encoder” is responsible for entropy encoding the wavelet transformed frames. The frames are encoded bitlayer by bitlayer (from

most significant to least significant), yielding progressive accuracy of the wavelet coefficients (Figure 4). The WEE itself consists of two main parts: the “Model Selector” (MS) and the “Arithmetic Encoder” (AE). The MS provides the AE with continuous guidance about what type of data is to be encoded by selecting an appropriate statistical model for the symbol (a bit) that has to be encoded next. It exploits the correlation between neighboring coefficients in different contexts. Finally the AE performs the actual compression of the symbol stream.

P: The “Packetizer” packs all encoded parts of the video together in one bit stream representing the compressed video.

Scalability in color depth is obtained by encoding luminance and chrominance information in three different channels in the YUV 4:2:0 format. Omitting the chrominance channels yields a grayscale version of the sequence, allocating more bits to these channels increases the color depth. Motion estimation is computed from luminance information only, but is also applied to the chrominance channels. In the subsequent parts of the algorithm the channels are processed totally independently.

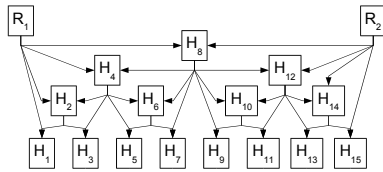


Fig. 2. Frame rate scalability. Motion estimation processes one Group of Pictures (GOP) consisting of 16 consecutive frames. The arrows illustrate which frames are used as a first approximation of the intermediate frames at lower compositional levels. R_1 is the reference frame of this GOP, R_2 is the reference frame of the next GOP and the H_i are the intermediate frames.

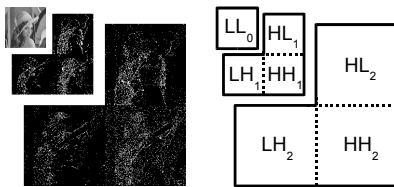


Fig. 3. Resolution scalability. Numbers in subscript indicate the resolution layers.

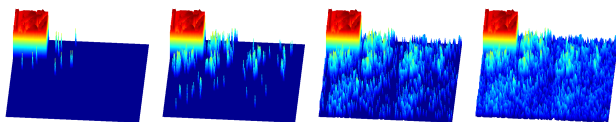


Fig. 4. Quality scalability: decoding more bitlayers gives a more accurate wavelet transformed frame.

By inverting the operations of Figure 1 we obtain a scalable video *decoder* consisting of a Depacketizer (DP), a Motion

Vector Entropy Decoder (MVED), a Wavelet Entropy Decoder (WED), an Inverse Discrete Wavelet Transform (IDWT) and Motion Compensation (MC).

In this paper we focus on a hardware implementation of the Wavelet Entropy Decoding part of the algorithm. Motion Compensation and Inverse Discrete Wavelet Transformation are already comprehensively elaborated in e.g. [6], [7]. The entropy decoding is by far the most computationally intensive part of the algorithm [8].

Conventional video codecs don’t offer quality scalability. Quantization is performed at encoding time and all (DCT) coefficients are encoded as a whole. In this new (scalable) approach all bits of the non-quantized (wavelet) coefficients are processed separately. This implies that a context model has to be calculated per bit instead of per coefficient. So the pressure on the Model Selector is much higher. The new approach also has to decode more bits. We start decoding every bit from the highest bitlayer, in which only the largest coefficients are non-zero. So we have to process the same number of bitlayers for the largest as for the smallest coefficients (which are often zero). This results in a larger total number of bits that the arithmetic decoder has to process for the same quality settings. In conclusion the actual question is whether the bitlayer approach for quality scalability is practically feasible.

Moreover we want to leave out the block based approach to make the entropy coding part harmonize better with the wavelet transform and to avoid block artifacts. Of course this will increase the memory requirements.

II. A HARDWARE-FRIENDLY SCALABLE WAVELET ENTROPY CODER

In [8] we proposed a new algorithm for entropy coding which fully supports the scalability of the presented video codec. It offers quality scalability by encoding the wavelet image bitlayer by bitlayer and resolution scalability by encoding data from the different resolution layers independently. It is really economical with memory, both in memory footprint and in required bandwidth. Moreover, despite the sequential nature of entropy coding algorithms, it allows a high degree of parallelism on the subband level. The algorithm was designed for simplicity to stimulate an elegant implementation. Finally, it gives state of the art compression results.

The new algorithm is shown in Figure 5. All subbands of the wavelet transformed channel are encoded (and decoded) independently. Therefore it is possible to process all subbands of the wavelet transformed frame in parallel. The subbands are processed bitlayer per bitlayer from top to bottom. The top is the bitlayer that contains the most significant bit of the largest absolute value of all coefficients and the bottom is the bitlayer containing the least significant bits. The bitlayers are processed in scanline order. This greatly benefits the memory accesses, since this is the order in which the data is stored in memory. It also enables us to stream data and use burst mode features of slower memories. All data from one subband is processed sequentially since all bits are now encoded based on information of previously encoded bits.

```

encode_frame:
  for all subbands:
    load_subbandmodel
    if (LL-subband of reference frame)
      encode (mean of frame, data model)
      subtract mean subband
    encode (number top bitlayer, data model)
    for all bitlayers
      encode_bitlayer

encode_bitlayer:
  for all bits //scanline order
    if coefficient was not significant yet
      if starting bitlayer
        encode (bit, toplayer model)
      else
        encode (bit, significance model)
    if bit becomes significant
      encode (sign, sign model)
      update bitmaps
    else //was already significant
      encode (bit, refinement model)

```

Fig. 5. Entropy encoding of one wavelet encoded frame.

As can be seen from the code in Figure 5 symbols are encoded with different models (the second argument of the encode routine) depending on their context. To improve memory accesses this context is kept very small. A model contains information about the expected value of the incoming symbol and is used to encode this symbol as efficiently as possible in the Arithmetic Coder (AC).

The Model Selector (see Figure 6) is responsible for selecting these models. Models are selected based on information regarding previously encoded bits. Model selection is used to exploit statistical characteristics (e.g. the fact that pixels become significant in clusters) by encoding symbols with a similar distribution using the same arithmetic coder. For optimal compression, storing all information about previously processed data would be ideal but since this excludes an efficient hardware implementation only the most relevant information is stored. Our algorithm limits this information to the sign and the significance of each coefficient. This information can easily be organized as two bitmaps with dimensions equal to the subband's. From these bitmaps the number of significant (or negative) neighbors of the current coefficient are counted to determine the model for the arithmetic coder.

There are four types of models: data models are used to encode data such as the number of the starting bitlayer; sign models assist in the prediction of the signs of the wavelet coefficients; significance models are used to predict the most significant bit of each wavelet coefficient. A coefficient is called significant as soon as we encounter its most significant bit. This group of models is subdivided further into two classes: the highest bitlayer models and the remaining bitlayer models; Finally there is a refinement model, used to estimate the value of bits of the coefficient below the significance bit. These bits have the characteristics of noise and are therefore hard to predict. In total, 64 models

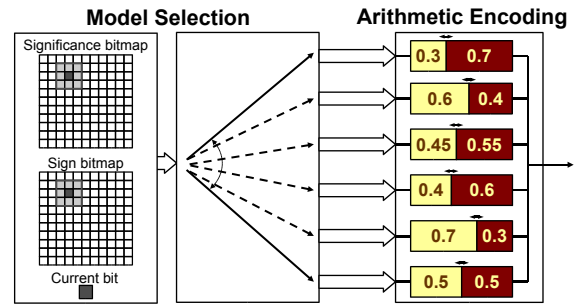


Fig. 6. The model selector selects a model based on information of neighbors, stored in sign- and significance bitmaps. Once the model is determined, the arithmetic encoder encodes the bit with the selected model.

are used per subband.

A. Arithmetic Coder

For the arithmetic coder we opted for a modified version of the CABAC arithmetic entropy encoder used in the AVC codec [9]. This is a low-complexity adaptive, binary arithmetic coder with a probability estimation algorithm that is well suited for an efficient hardware implementation. A few modifications were made to improve the integration into our wavelet entropy encoder. Since all memories on an FPGA are 9 bit wide, we augmented the 7 bit state per model (i.e. the current estimated probability of the model) to 9 bit. This increased the accuracy for probability estimation and as a consequence the compression performance. We also perfected the transition rule table for updating the probability estimation, but this falls outside the scope of this paper. The fact that only a 9 bit state per model needs to be stored, means that we only require 576 bits. The cost for a large number of models is in other words very limited.

III. BITRATE REQUIREMENTS

How fast should the Wavelet Entropy Decoder be if we want to achieve real time video? In Table I we measured the number of decoded symbols per second for the 'Foreman' and 'Mobile' sequence for both CIF and QCIF resolutions at lossless quality. The decoder must be able to at least output 45 million bits per second.

The frames of the sequences of Table I were transformed over 3 wavelet levels, thus have 10 subbands. The largest subband contained 278139 decoded bits. If we would instantiate 10 decoders and require 30 frames per second, the decoder should run at least at 8.4MHz for CIF. But 10 decoders (3 wavelet levels) will probably occupy a lot of area.

TABLE I
NUMBER OF DECODED SYMBOLS PER SECOND.

Sequence	# million decoded bits per second		# bits in largest block	
	QCIF	CIF	QCIF	CIF
Foreman	9.2	38.9	63087	277752
Mobile	10.55	44.0	69584	278139

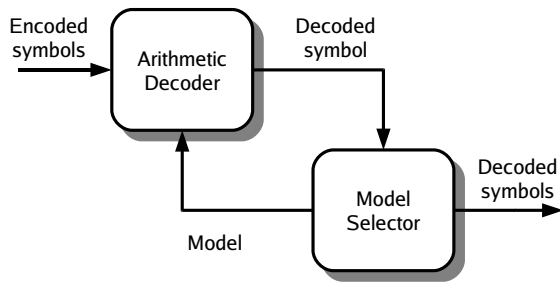


Fig. 7. The critical feedback loop in the entropy decoder: The model selector has to provide the arithmetic decoder with a new model that depends on the output of the arithmetic decoder.

IV. HOW HARDWARE-FRIENDLY IS HARDWARE-FRIENDLY?

Entropy coding is a very sequential process by nature. Its purpose is to exploit as much statistical redundancy as possible. Therefore it has to take all previously coded symbols into account to achieve optimal compression. This results in a critical feedback loop in the algorithm of the arithmetic decoder. The model and result of the currently decoded bit depend on the result of the previously decoded bit. So what is there to gain with a hardware-implementation? If we want to speed up the decoding time we have to squeeze every drop of possible parallelism out of the algorithm. Through advanced pipelining and speculative execution and data fetching, we managed to develop a decoder that manages to decode one symbol per clock cycle.

A. Arithmetic Decoder

The flow diagram for the binary arithmetic decoding process for a single binary value is illustrated in Figure 8. The current internal state for the arithmetic decoding engine is characterized by two quantities: the current interval range (R) and the current value in the current code interval (Figure 9). In the first step the state (corresponds with the probability) of the current model is fetched from the 'State-table' together with the LPS (Least Probable Symbol) range that corresponds with this state. This $range_{LPS}$ is an approximation, a quantized value which is stored in a precalculated table. In a second step the current value is tested to be in the MPS (Most Probable Symbol) range or in the LPS range of the current interval. Depending on whether we are decoding an MPS or an LPS the value, range and state are updated. In the final step the range is renormalized to make sure enough accuracy is maintained.

There are two major impediments for a one-cycle execution of arithmetic decoding process: The algorithm needs multiple sequential memory operations and the third step, the renormalization loop, has a variable number of iterations. We will now present our solutions to these problems.

1) *Memory accesses:* In the first step of the algorithm we need to fetch the size of the LPS range for the current model from the $Range_{LPS}$ -table. Therefore we need to first fetch the state of model from the State-table. In the second step we also

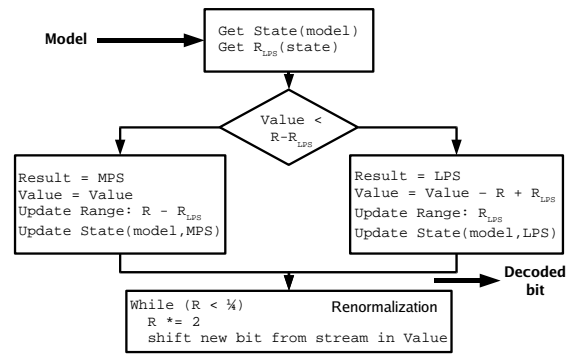


Fig. 8. The flow diagram for the binary arithmetic decoding process including the updating process of probability estimation for a single binary value.

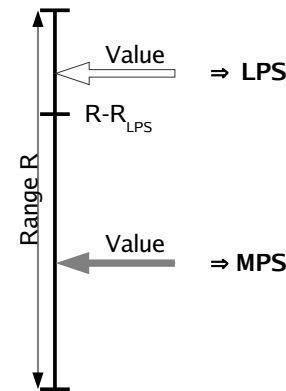


Fig. 9. Illustration of the meaning of range and value. If value is in the LPS range, the decoded bit is LPS. Otherwise the result is MPS.

have to fetch the next state of the current model and write it back to the State-table. If we want to use normal memories to store the different tables, this requires multiple clock cycles.

This problem can be circumvented by using modified data structures and speculatively fetching data from the tables. If we extend the State-table with some extra columns that contain all possibly needed data, we are able to load all necessary data in one cycle. The data ($Range_{LPS}$ and the next state) for the next occurrence of this model can be loaded in the next cycle and the updated extended state entry can be written back in a third cycle. As illustrated in Figure 10, this data fetching can be pipelined. Because we can make use of multiple parallel, on-chip, dual-port memories we are now able to process all memory operations in one clock cycle. Even if we encounter the same model for two (or more) consecutive cycles, all necessary data is always available. The disadvantage is that we have to store a larger table. The extended table is now $2 \times 9 + 4 \times 16 \text{ bit} = 82 \text{ bit}$ wide instead of the 9 bit of the original State-table. But since the total size is only 5248 bit , the new table still fits easily in modern FPGA memories.

2) *Renormalization loop:* The second problem that hinders a one clock execution of the arithmetic decoding algorithm is the renormalization phase. To make sure that the new

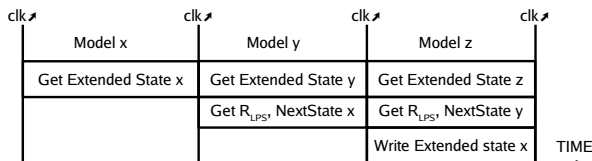


Fig. 10. The pipelining of the data fetching.

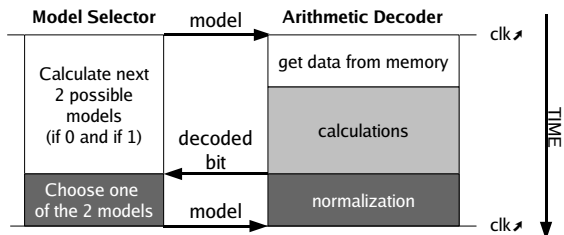


Fig. 12. The time course of decoding one bit. While the arithmetic decoder is normalizing, the ModelSelector selects one of the two possible models depending on the decoded bit.

interval range R stays within its legal range, a technique based on an idea of [10] was used. The loop condition, $R < 1/4$ (of the interval), corresponds actually with the number of zeroes in front of the newly calculated range R . So instead of shifting R and value over multiple iterations, we can simply count the number of leading zeroes of the new range R and shift the new Range this number of bits. In case of LPS, range R is equal to RangeLPS . In case of MPS we can anticipate the number of leading zeroes of $\text{Range} - \text{RangeLPS}$. Leading Zero Anticipation (LZA) [11] can perform this count in parallel with the actual subtraction, which further reduces the time needed for normalization. The core of the architecture is illustrated in Figure 11.

We now shift a varying number (between 0 and $\text{NumberOfRangeBits}-1$) of new (encoded) symbols into value every cycle. This implies a dedicated input buffer ('SymbolBuffer') that is able to deliver these symbols in time.

B. Model Selection

In the previous section it was shown how to decode one symbol per clock cycle. But since no symbol can be decoded unless there is a model available, how is it possible to calculate the next model, that depends on the result of the currently decoded symbol, fast enough? We can exploit two advantageous properties. First, we note that the modeling mechanism of our newly developed algorithm for model selection is relatively simple and uses only limited context information. Secondly, we know in advance that the result of arithmetic decoder is either 0 or 1. Therefore we can speculatively calculate the two possible next models and postpone the selection of one of them until the decoded bit is effectively known. The course of time is illustrated in Figure 12.

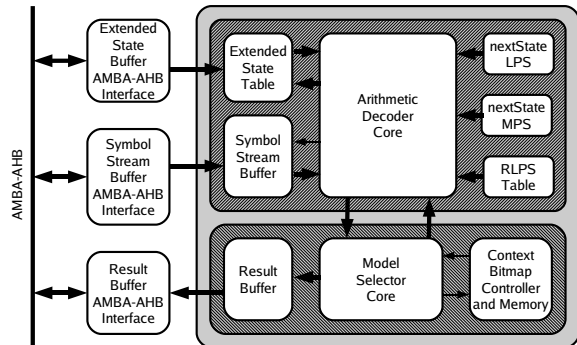


Fig. 13. Architecture of the Wavelet Entropy Decoder.

V. IMPLEMENTATION RESULTS

The entire architecture of the complete Wavelet Entropy Decoder described in Section IV is illustrated in Figure 13. This design was implemented in VHDL, co-simulated in a System-C testbench and synthesized for an Altera Stratix S25 (speed grade 5). The results are summarized in Table II. The one DSP multiplier block is used for calculating the size of the sign and significance bitmaps ($\text{rows} \times \text{cols}$) for the ModelSelector and could easily be avoided.

The entire design works for clock rates up to 52 MHz. The required bitrate for a single decoder decoding lossless CIF (see III) is clearly reached. The implementation is also very compact, less than 10% of the available resources are used. Moreover only 5% of the available memory is used, of which about 70% is needed to store the two bitmaps which support subband sizes of maximum 32768 bits. This is large enough to process the largest subbands of CIF frames which have $176 \times 144 = 25344$ coefficients. The targeted FPGA is clearly large enough to contain multiple instantiations of our decoder.

The same design was re-synthesized for different devices of other families. The results are summarized in Table III. On the newest device, the Stratix II, the design uses only 1481 logic cells (with new LUT structure) and can be clocked at 73MHz.

TABLE III

SYNTHESIS RESULTS FOR DIFFERENT FAMILIES. THE LAST NUMBER IN THE DEVICE NAME IS THE SPEED GRADE.

Device	Family	Logic cells	Frequency (MHz)
ep1s25f1020c5	Stratix	2025	50.71
ep2s60f1020c5	Stratix II	1481	52.54
ep2s60f1020c3	Stratix II	1481	73.81
ep1c20f324c6	Cyclone	2171	46.48
ep2c20f256c6	Cyclone II	1965	51.93

VI. CONCLUSIONS

In this paper we demonstrated that the bitlayer approach to offer quality scalability is indeed feasible. We presented a small and fast implementation of a scalable Wavelet Entropy Decoder, which is able to produce more than 50 million

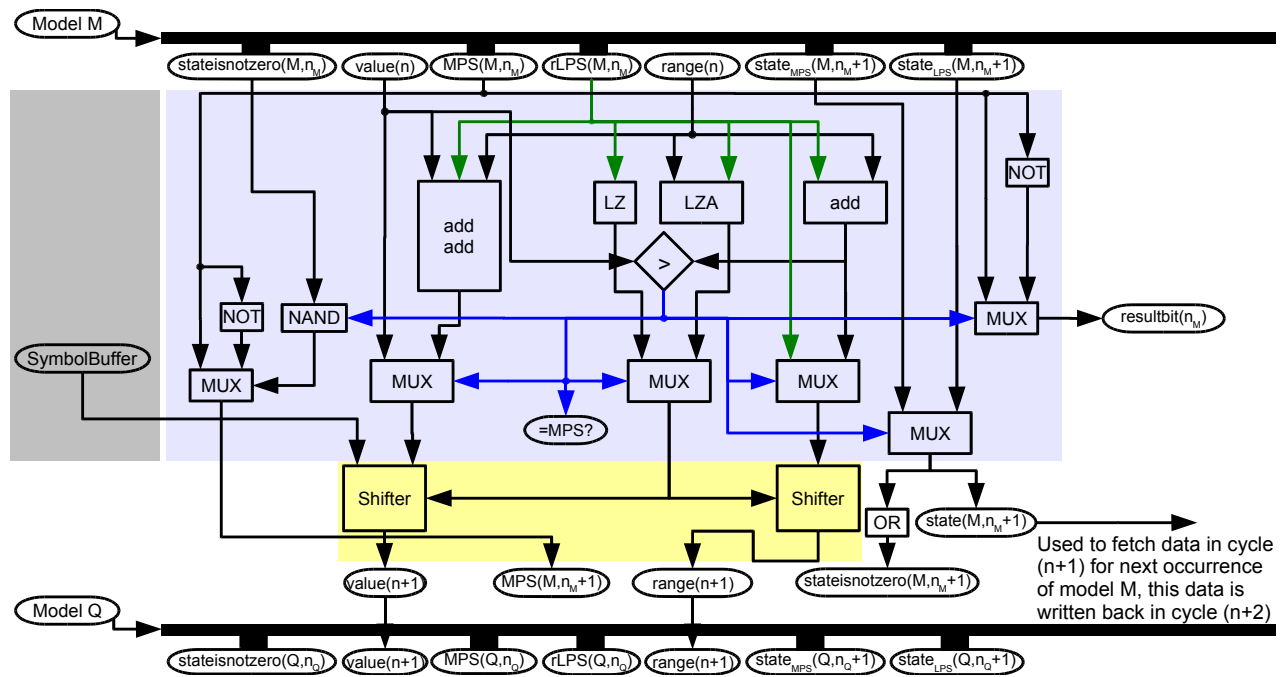


Fig. 11. Schematic of the ArithmeticDecoderCore. The schematic is simplified for clarity. LZ stands for Leading Zero (detection), LZA for Leading Zero Anticipation and MUX for Multiplexer

TABLE II
SYNTHESIS RESULTS FOR AN ALTERA STRATIX S25.

Block	Logic Elements		Memory Bits		DSP	Maximum
	25660 in total		1944576 in total		Blocks	Frequency (MHz)
<i>Arithmetic Decoder</i>						
ArithmeticDecoderCore	693	(3%)	0	(0%)	0	60.18
SymbolStreamBuffer	391	(2%)	0	(0%)	0	218.72
LPSROM	0	(0%)	1536	(< 1%)	0	300
MPSROM	0	(0%)	1536	(< 1%)	0	300
RLPSROM	2	(< 1%)	14848	(< 1%)	0	300
<i>ModelSelector</i>	857	(3%)	68608	(4%)	1	89.38
BitmapController	175	(< 1%)	66560	(4%)	1	166.67
ResultBuffer	141	(< 1%)	2048	(< 1%)	0	214.18
<i>WED</i>	2025	(8%)	101248	(5%)	1	52.70

decoded bits per second. This is sufficient for decoding lossless CIF sequences. Larger resolutions are possible since the core is small enough to allow multiple parallel instantiations.

ACKNOWLEDGMENT

This research is supported by I.W.T. grant 020174, F.W.O. grant G.0021.03 and by GOA project 12.51B.02 of Ghent University. Harald Devos is supported by the fund for scientific research Flanders (F.W.O.).

REFERENCES

- [1] H. Devos, H. Eeckhaut, M. Christiaens, F. Verdicchio, D. Stroobandt, and P. Schelkens, "Performance requirements for reconfigurable hardware for a scalable wavelet video decoder," in *CD-ROM Proceedings of the ProRISC / IEEE Benelux Workshop on Circuits, Systems and Signal Processing*. STW, Utrecht, November 2003.
- [2] A. Munteanu, "Wavelet image coding and multiscale edge detection - algorithms and applications," Ph.D. dissertation, Vrije Universiteit Brussel, 2003.
- [3] A. Munteanu, J. Cornelis, G. Van der Auwera, and P. Cristea, "Wavelet image compression - the quadtree coding approach," *IEEE Transactions on Information Technology in Biomedicine*, vol. 3, no. 3, pp. 176–185, September 1999.
- [4] P. Schelkens, A. Munteanu, J. Barbarien, M. Galca, X. Giro-Nieto, and J. Cornelis, "Wavelet coding of volumetric medical datasets," *IEEE Transactions on Medical Imaging, Special Issue on "Wavelets in Medical Imaging"*, vol. 22, no. 3, pp. 441–458, March 2003.
- [5] D. Stroobandt, H. Eeckhaut, H. Devos, M. Christiaens, F. Verdicchio, and P. Schelkens, "Reconfigurable hardware for a scalable wavelet video decoder and its performance requirements," *Computer Systems: Architectures, Modeling, and Simulation*, vol. 3133, pp. 203–212, July 2004.
- [6] P. M. Kuhn, *Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation*. Kluwer Academic Publishers: Kluwer Academic Publishers, 1999.
- [7] N. D. Zervas, G. P. Anagnostopoulos, V. Spiliotopoulos, Y. Andreopoulos, and C. E. Goutis, "Evaluation of design alternatives for the 2D-discrete wavelet transform," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 12, December 2001.
- [8] H. Eeckhaut, H. Devos, B. Schrauwen, M. Christiaens, and D. Stroobandt, "A hardware-friendly wavelet entropy codec for scalable video," in *DATE 2005 Designers' Forum Proceedings*, March 2005, pp.

14–19.

- [9] D. Marpe, H. Schwarz, G. Blättermann, G. Heising, and T. Wiegand, "Context-based adaptive binary arithmetic coding in JVT/H.26L," *Proc. IEEE International Conference on Image Processing (ICIP'02)*, vol. 2, pp. 513–516, September 2002.
- [10] R. R. Osorio and J. D. Bruguera, "Arithmetic coding architecture for H.264/AVC CABAC compression system," in *Euromicro Symposium on Digital System Design*. Rennes, France: IEEE, August 31 - September 03 2004, pp. 62–69, dSD'04.
- [11] M. S. Schmookler and K. J. Nowka, "Leading zero anticipation and detection—a comparison of methods," in *15th IEEE Symposium on Computer Arithmetic (Arith-15 2001)*. Vail, Colorado: IEEE Computer Society, June 2001.