Proceedings of the IEEE Workshop on Signal Processing Systems Design and Implementation (SiPS 2006) pp. 277-282, (Banff, AB, Canada), Oct. 2-4, 2006.

Transparent Embedded Compression in Systems-on-Chip

A. K. Riemens, R. J. van der Vleuten, P. van der Wolf, G. Jacob, J. W. van de Waerdt, J. G. Janssen

***Abstract*—Bandwidth to off-chip memory is a scarce resource in complex Systems-on-Chip for embedded media processing. We apply embedded compression for bandwidth-hungry image processing functions in order to alleviate this bandwidth bottleneck. In our solution embedded compression is implemented as part of the System-on-Chip infrastructure, fully transparent for the hardware and software image processing components. Hence it can be applied without requiring changes to these components. We present the compression algorithm and demonstrate that we achieve significant bandwidth reductions (20% - 40%) for image data at acceptable cost (approximately 1 mm^2 in 90 nm CMOS) while preserving high image quality.**

I. Introduction

The ongoing advance of IC technology leads to steadily and rapidly increasing computational power for media processing applications. The increased processing power also requires increased data throughput. This holds in particular for bandwidth-hungry applications, like video processing.

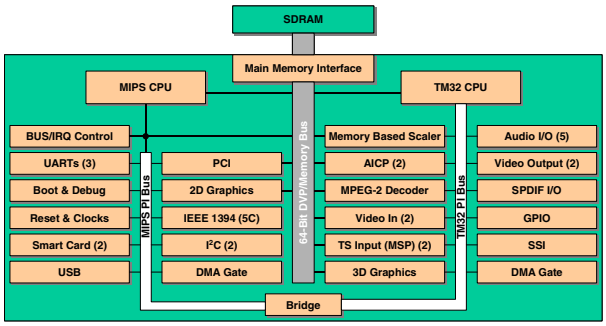

Fig. 1. SoC conceptual top-level architectural view.

This paper addresses the memory bandwidth bottleneck for embedded media processing systems in the domain of consumer electronics. Cost is a key issue in this domain and hence a System-on-Chip (SoC) typically has a single SDRAM with an interface as small as possible to limit the pin count. We describe a method applying embedded compression to alleviate the SDRAM bandwidth bottleneck. So we apply both image compression and image decompression embedded in a single system. The method is applicable to systems with dedicated IP blocks as well as signal processing functions executing on a programmable core (or any combination thereof). Examples of such systems are described by Rathnam and Slavenburg [1] and Dutta *et al*. [2]. A data sheet of a recent product is [3]. Figure 1 illustrates the architecture of such a SoC. Figure 2 illustrates an example of an application that can be mapped on this SoC.

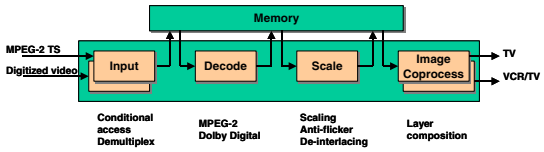

Fig. 2. Example application view.

The class of systems that we target is characterized by:

- Unified memory located externally from the processing chip.
- A substantial part of the available memory bandwidth is consumed by image data.
- Memory-based communication between various (hardware or software) components.

To elaborate further on the system-level communication, consider all images to be stored in the off-chip memory. When e.g. a filter IP block processes an image, its operation is setup by control software executing on a processor, which specifies the source address of the input image and the destination address of the output image. The IP block autonomously traverses the images, possibly using a DMA engine to transfer image data from/to memory. When finished, the IP block typically issues an interrupt to the processor. This way, the software maintains the memory buffers while multiple hardware blocks can perform various processing steps in an application concurrently.

We distinguish three types of components. IP blocks (1) are hardware components dedicated to specific signal processing functions and/or I/O. Signal processing functions however can also be implemented as software modules (2). Finally, control software (3) implements an application by taking care of the setup of the signal processing components, buffer management, data flow, low-rate parameter adjustments, etc.

While the IC technology shows an increasing processor-memory performance gap, the application field of mainstream consumer appliances remains bandwidth-hungry. This is illustrated by high-definition TV, which is becoming commodity, compression standards with multiple reference images (e.g. H.264), ever more advanced image enhancement algorithms, etc. Storing image data on-chip is

Manuscript received July 31, 2006.

A. K. Riemens, R. J. van der Vleuten, and P. van der Wolf, are with Philips Research, High Tech Campus 36, 5656AE Eindhoven, Netherlands.

G. Jacob, J. W. van de Waerdt and J. G. Janssen are with Philips Semiconductors, 1140 Ringwood Court M/S 42 J, San Jose, CA 95131, USA.

Contact: e-mail: bram.riemens@philips.com; tel: +31-40-27 43833

1-4244-0383-9/06/$20.00 ©2006 IEEE

not a cost-effective option considering the large amounts of data. In view of these considerations, architectural and algorithmic solutions to deal with the memory bandwidth bottleneck are required. This paper presents a solution, based on embedded compression to reduce the memory bandwidth consumption, that can be seamlessly integrated in a SoC.

II. Problem statement

A. Memory technology

Memory access speed improves at a much slower pace than processing speed. Hennessy and Patterson report a yearly increase of 55% for processing speed, while memory latency improves by only 7% per year [4]. This is known as the "processor-memory performance gap", or "memory wall" [5]. Note that available memory capacity increases very fast: 40% - 60% per year.

For present-day memory devices, memory bandwidth is hampered by the latency associated with setting up a data transfer, that is, the time that passes between offering an address and having the corresponding data word available. However, when this data word is available, neighboring data words are also available and can be transferred during subsequent clock cycles. For this reason, a memory transaction typically consists of a burst of consecutive data words. Hence, data is addressed once; then multiple data words are transferred, belonging to consecutive addresses. In this way memory bandwidth can be used efficiently. In order to remain efficient, the increasing latency (in terms of processor cycles) urges for increasing burst length. Of course, larger bursts can only be effective if the data words in a burst are actually needed by the requesting processing unit. Image processing functions have a good spatial locality, as large sets of neighboring pixels are typically processed together. At present times, a typical burst length is in the range of 64 to 256 bytes. Using such burst transactions, bandwidth utilizations of 70% - 80% are feasible in present-day SoCs.

B. Requirements

In the systems that we target, a significant part of the bandwidth is consumed by image processing functions. We therefore focus on applying embedded compression on image data in order to reduce the amount of image data that needs to be transferred between processing units and off-chip memory. Main emphasis is reduction of this bandwidth consumption. The additional property of embedded compression to reduce the memory footprint of applications is not taken into account.

In the industrial practice, platform based design and reuse of existing IP blocks is vital to achieve acceptable time-to-market at reasonable design cost. Therefore, the embedded compression module shall be fully compatible with legacy IP blocks, without the need to "touch" existing blocks when applied in a new SoC. This also applies to the signal processing software and the control software. The signal processing functions use addresses to traverse the image data. Hence, the conversion of a pixel location to an address is tightly integrated with the signal processing function. The use of the embedded compression shall not lead to changes propagating throughout the system. Hence, the addressing scheme of the image processing units shall not be influenced by the use of embedded compression. These considerations boil down to a transparency requirement: the application of embedded compression shall be transparent to both the signal processing functions and the control software.

The high volume electronics market is very price-sensitive. Hence, the amount of hardware available for the addition of embedded compression in the SoC is limited. We therefore focus on low-complexity algorithms. On the other hand, we aim for solutions that preserve the high image quality as much as possible. Finally, extensive analysis of the bandwidth consumption of numerous application scenarios at hand showed that a compression ratio of at least 1.5 is required to comply with the bandwidth budget. A careful balance between these (conflicting) requirements is essential.

Some use cases require more bandwidth resources than others. To adapt the resource use and to cope with the conflicting requirements mentioned above, an adjustable compression ratio is required.

Finally, the class of systems that we target run multiple applications simultaneously and demands real-time data processing. In such systems, performance predictability is of utmost importance. Therefore, memory latency for non-compressed accesses shall not be affected significantly in order to reduce the impact on system performance. For image data, typically a prefetching scheme can be applied to compensate for memory latency. The compression functionality inevitably adds to this latency, but this can be compensated for by means of prefetching (as long as the latency is predictable).

Therefore, we aim for embedded compression to reduce bandwidth consumption with requirements summarized as:

- Transparency.
- Low complexity algorithms (i.e. low cost).
- No significant impact on picture quality.
- Compression ratio of at least 1.5.
- Adjustable compression ratio.
- Predictable impact on latency for memory accesses with embedded compression; maintaining minimal impact on latency for other memory accesses.

III. Related work

The issue around the widening memory gap has been approached in various ways.

General-purpose programmable processors have steadily increased their cache size and introduced multi-level caching techniques trading off-chip bandwidth for on-chip bandwidth [4]. But, for media processing, caches often do

not reduce the off-chip memory traffic since the image data does not fit in caches or on-chip buffers in a cost-effective manner. This is due to rapidly increasing image sizes and the simultaneous use of a larger number of images.

Media processing ICs have conventionally applied function-specific memory interfaces. E.g. for video scan rate conversion, a static on-chip communication scheme with streaming data access is applied [6]. Several attempts to increase flexibility lead to more complicated communication schemes (e.g. [7]). However, such approaches are becoming less and less attractive with increasing system integration, because the wider range of functions integrated in a single SoC leads to more varying and increasingly data dependent access patterns.

A different approach is to reduce the memory bandwidth by compressing the data before it is stored in the memory. Such embedded compression is conventionally tailored to streaming image data access patterns. In an attempt to optimize the compression efficiency, the embedded compression algorithm is typically fully interwoven with specific functions and data access patterns. I.e. the embedded compression function/algorithm is part of the media processing function/algorithm. Examples of this approach for MPEG de/encoding are e.g. described in [8], [9], [10], and [11]. However, this class of solutions typically even further complicates data dependent access, which can result in a significant reduction (or even completely annihilation) of the memory bandwidth gain obtained by the embedded compression of the data.

Hence, none of these conventional approaches fully satisfies the requirements for the class of systems under consideration.

IV. Solution Approach

A. Basic idea

From the transparency and backward compatibility constraints, we deduced that the embedded compression should be considered part of the communication infrastructure of the chip. This poses an entirely different viewpoint compared to conventional approaches. Traditionally, embedded compression has mainly been viewed as an algorithmic issue. We however, take a system architect's perspective and embed the compression in the heart of the communication system.

As discussed above, memory traffic is executed in transactions, consisting of a burst of data. Note that this applies to the whole communication system, including bus arbitration, bus scheduling and the memory access protocols. This memory access property is strongly related to the technological causes of the processor-memory performance gap; therefore it applies to any high performance memory system.

In view of these observations, we decided to:

1. Apply embedded compression in the memory hierarchy on a single transaction, without any state to be preserved between memory transactions.
2. Only modify the burst size (length) of the transaction.

Compression algorithms typically allow two main approaches. On one hand, one may choose to keep the compression ratio fixed ("fixed bit rate"). That results in variable image quality, depending on the amount of detail in the image. Alternatively, one may choose to vary the compression ratio ("variable bit rate") to keep the image quality constant. Variable bit rate systems typically use a feedback control loop to keep the data rate within acceptable limits around a certain average setpoint. Since our first aim is towards systems with predictable performance, we will describe our solution based on a fixed compression ratio. Note however, that the same principle can be applied with a variable compression ratio, provided that an additional software component takes care of the bit rate control.

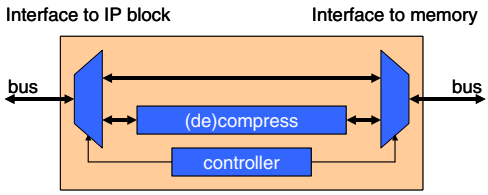

Fig. 3. Embedded compression as memory bus component.

Figure 3 shows the block diagram of the embedded compression module we propose. The module interfaces to an on-chip memory bus on both sides. The memory bus handles data transfers after each other, for many clients, carrying data consisting of e.g. software instructions, execution stack data, application control state, audio and image data. When image data passes over the bus, a controller activates the embedded compression module. The control mechanism is described in detail in the next section. The compression unit reduces the amount of data to be transferred; hence, the transfer length is reduced. The address however is not modified. After processing a data transfer, no state information needs to be preserved.

As an example, the following scenario can be applied, using a compression ratio of a factor of 2. For a write operation, the embedded compression module receives the start address A and 256 bytes of image data. The data will be compressed to 128 bytes of data, which is stored at the address A. The remainder 128 bytes of address space is not used. At a read transaction on address A, the signal-processing unit requests 256 bytes of image data. Only 128 bytes of data are actually read from the memory. The compression module decompresses this to 256 bytes of image data, which is passed to the requesting bus client.

B. SoC architecture considerations

As explained, we propose the use of an embedded compression module that works on a per burst basis without maintaining state. In figure 4 we present a more detailed block diagram.

As shown in figure 4, the start address of the burst passes the module unmodified. The start address is further used by a controller to perform address discrimination against a set of pre-programmed address ranges in a lookup table. The basic idea is that upon allocation of an image buffer in memory, the control software can decide whether image data that will be communicated via this image buffer needs to be compressed. If the image data needs to be compressed, then the control software can program the address range of the image buffer in a register of the embedded compression module. So for each application use case in the embedded media processing system, we allow the application programmer to select which images need to be compressed. Upon a mode switch to a different use case, the embedded compression module can be re-programmed to help satisfy the bandwidth needs of the new use case. The number of registers in the module determines for how many image buffers data compression can be performed and is therefore a design parameter. If the control unit decides that the start address of a burst points into an image buffer for which image data is to be compressed, it instructs a compression unit to (de-)compress the data arriving on the data bus (from)to memory. Also the burst length will be adjusted based on the achieved compression ratio. If no (de-)compression is to be performed for a burst, the data and length are forwarded unmodified.

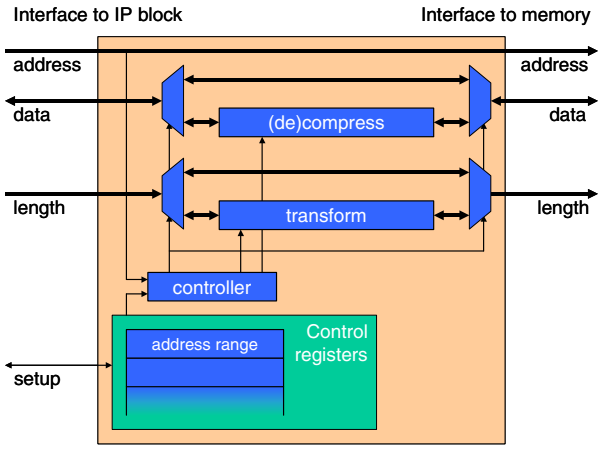

Fig. 4. Embedded compression block diagram.

Our approach requires a slight adaptation of the control software; typically at the stage of application setup when buffers are allocated. In addition to the memory management administration, the address range registers of the embedded compression module are set. Although this compromises the transparency requirement for the control software, it provides a very natural control point in the software system, since only the setup stage is affected, not the execution stage. This solution is preferred over alternatives, like tagging the data with extra control words or extra control wires. Such solutions would violate all transparency requirements since they need to be in effect on each bus transaction.

An important architectural issue is where to fit the embedded compression module in the SoC infrastructure. We present three options (see also figure 1).

1. Attach the embedded compression module to the ports of the IP blocks.
2. Locate the module near the main memory (SDRAM) interface.
3. Closely integrate the module with the main memory (SDRAM) interface.

The first option has the obvious disadvantage that the module needs to be replicated multiple times, at least once for each IP block for which image data potentially is to be (de-)compressed.

For the second option the module is attached as a front-end to the port of the SDRAM interface. If the SDRAM interface has multiple ports, then we may have to replicate the module. However, since multiple IP blocks typically share one SDRAM port, this option is still favorable over the first option.

The third option allows the module to be shared for data transferred via the different ports of the SDRAM interface and therefore avoids replication. However, for this option the SDRAM interface needs to be modified, which is not feasible if it is obtained from an external provider.

C. *Algorithmic considerations*

Images are generally stored with horizontally neighboring pixel values (e.g. a line in the image) on consecutive addresses. This is caused by (legacy) line based transmission standards that transmit horizontally consecutive image samples (e.g. [12]). Further, also display interfaces and numerous video-processing ICs require line-based image transfer. Finally, also graphics systems tend to use a similar image or texture storage format (e.g. openGL [13]). In practice, interoperability constraints within SoC platforms cause also signal processing IP blocks to use such format. Hence, this image layout convention is widely respected. We therefore base the compression algorithm on the de-facto standard that the compression algorithm receives horizontally adjacent image data.

Note that this is not a fundamental issue in our solution. Alternative compression algorithms can be developed in case a single transfer of consecutive memory addresses contains a rectangular block of image data, for example.

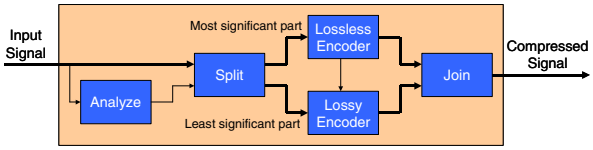

Fig. 5. Block diagram of embedded compression algorithm.

Since the compression function is embedded in the SoC, we have design freedom to choose both the compression algorithm and the data format of the compressed data. The compression algorithm that is applied is derived from the method presented in [14] and [15]. The block diagram is

shown in figure 5.

The input signal is split into two parts: an MSB part, which is losslessly compressed and an LSB part which is lossy. The lossless compression used for the MSBs is "DPCM+VLC". This means that the difference between subsequent MSB samples is computed and that this difference is compactly transmitted using a variable-length code (VLC). The VLC that we use is a modified "Rice" code, as explained in [15]. The LSB part is compressed by a scalable encoder, i.e. in such a way that its description can be truncated at any desired point and all available bits (i.e. the bits left after lossless compression of the MSB part) can be used.

To achieve a given target compression ratio (specified by the application), the number of bits required to compress the MSB part is counted and, if this is too high, the split between the MSBs and LSBs is changed, i.e. the number of losslessly compressed MSBs is reduced and the number of LSBs is increased. The optimal split point (as much MSBs as possible) can be determined either sequentially or in parallel, allowing for a trade-off between IC area and compression latency.

The compression algorithm has several important features that are advantages compared to traditional DPCM. One feature is that the quantization (by means of splitting off bit planes) has been taken out of the prediction loop. This enables the prediction loop to be fully unrolled so, when desired, all computations can be done in parallel to reduce the compression latency. Another feature is that the algorithm can be designed such that repeatedly compressing already compressed (and decompressed) data produces the exact same result as that of the first compression step. Thus, the video quality does not degrade when the compression is used inside a recursive loop (e.g. for temporal noise reduction).

V. Implementation and Results

We have implemented the proposed embedded compression module for use in a video-processing chip for high-end HDTV television sets. The compression factor applied for this chip is 24/16=1.5 for 8-bit video data and 30/16=1.875 for 10-bit video data, in both cases yielding compression that is visually lossless. For other applications (e.g. mobile devices with smaller screens), where the trade-off between bandwidth consumption and image quality may be chosen differently, a compression factor of 2 is feasible on 8 bit video data. For reference, table 1 lists the PSNR on the "lena" image.

Table 1. PSNR with various compression factors on "lena" image.

Bit depth	Compression Factor	PSNR [dB]
8	1.5	54.81
10	1.875	54.60
8	2.0	45.63

Extensive "torture tests" have been applied to the compression algorithm. This included testing the compression performance on very challenging data, such as special test patterns or noisy data. For noisy data, a special noisy test sequence was created from a noise free input. Then, the algorithm was tested in a recursive loop (100 repeated compress-decompress cycles) to ensure that the output did not drift away from the (known noise-free) input data. Furthermore, video enhancement, such as sharpness enhancement boosting the high frequencies, was performed as post processing on the compressed data to make sure that compression errors do not become visible by subsequent processing in the video chain. This way we showed that the picture quality can clearly match the required quality standards.

For the integration of the module in the SoC infrastructure we have selected the second option from the list of options presented in section IV.B.

Shorter burst lengths reduce the efficiency of the SDRAM data transfer. Therefore, straightforward reduction of the burst length does reduce the amount of data to be transferred, but at the same time, it reduces the bandwidth efficiency, thereby sacrificing part of the benefits. In order to avoid this, we use a rather large transfer size of 256, such that even after compression, the burst length is still in a range that allows effective transfers.

All together, the effective bandwidth saving is approximately 20% to 25% for the 1.5 compression ratio described above. For other applications, a compression factor of 2 can be achieved, resulting in an effective bandwidth saving of 40%.

The embedded compression unit has been implemented in 90nm CMOS technology using a synthesizable design style. The whole unit, including encoder, decoder and control, occupies approximately 1 mm^2 and operates at a clock frequency of 350MHz. Latencies are 58/106 cycles for decompression and 80/144 cycles for compression at transfer sizes of 128/256 respectively. As noted in section II, prefetching can be applied to hide this additional latency. Other trade-offs between latency and area are possible.

Figure 6 shows an example of the data stored in memory for both an uncompressed image and its compressed version (luminance only). In this example, each horizontal image line segment (corresponding to a memory access burst) is compressed by a factor of 2. In figure 6, the compressed data is shown as luminance data, such that it is clearly visible that there is "empty space" at the end of each image segment (shown black). It can also be observed that the target compression ratio is always achieved, since we chose for a fixed compression ratio. In the implementation we actually specify the target transfer size and the compression ratio is derived from this. Since the image width is not an integer multiple of the segment size, the last segment contains less data and needs to be compressed less than two times, while still achieving the target transfer size. Therefore the compressed data for the last segment of each image line looks different from the other segments.

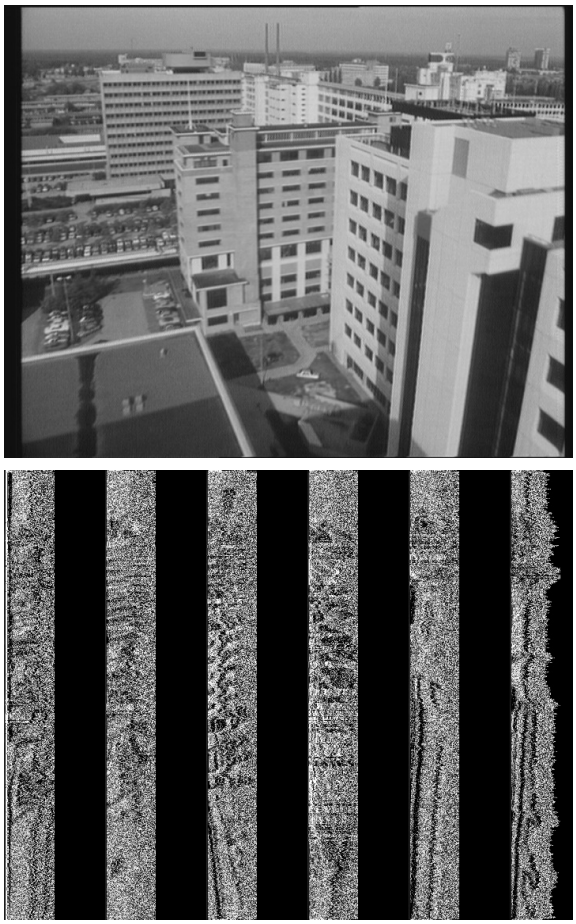

Fig. 6. Memory layout of uncompressed and compressed image data.

We note that the proposed compression module helps to de-risk complex SoC designs. De-risking deals with uncertainties in the actual applications of the system, or even in the specifications during the SoC design phase. When bandwidth problems occur for certain use cases (after the SoC is finished), image buffers can be compressed selectively with appropriate compression ratios to make the bandwidth requirements fit the available bandwidth to SDRAM.

VI. Conclusions

It shall be clear that the solution, described in this paper, is perfectly suited for a platform approach, where IP blocks can be designed independently of the SoC communication infrastructure and instantiated in several different SoCs. Since the application of embedded compression is considered (an optional) part of the communication infrastructure, the IP design is not affected by it. Furthermore, this allows legacy IP blocks (either hw or sw) to be used in a system that utilizes these novel compression techniques. Therefore, the main requirement of transparency is achieved.

We have shown that significant bandwidth savings on image data are possible. In an application targeting high image quality video processing, a compression factor of at least 1.5 was achieved. For other applications, a compression factor of 2 is feasible. This results in effective bandwidth savings of 20% to 40% respectively.

The cost of the embedded compression unit in terms of Si area is low: approximately 1 mm^2 in 90nm CMOS technology. These cost/performance features pose an attractive solution for system architects, involved in the design of complex Systems-on-Silicon.

Summarizing these results, we conclude that we meet all our requirements.

References

[1] S. Rathnam and G. Slavenburg; "An architectural overview of the programmable multimedia processor TM1"; *Proc. Compcon, IEEE CS, Press.* 1996.

[2] Santanu Dutta, Rune Jensen and Alf Rieckmann; "Viper: A multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems"; *IEEE Design and Test of Computers, Vol. 18 Issue 5, Sept. 2001.*

[3] Nexperia PNX8500 product description http://www.semiconductors.philips.com/acrobat/other/nexperia/pnx8500-1000.pdf.

[4] John L. Hennessy and David A. Patterson; "Computer architecture, A quantitative approach"; *Morgan Kaufmann Publishers, 3rd edition, 2003, ISBN 1-55860-596-7.*

[5] M. Wilkes; "The memory gap (keynote)"; *Solving the Memory Wall Problem Workshop, 2000.* http://www.ece.neu.edu/conf/wall2k/wilkes1.pdf.

[6] G. de Haan, P.W.A.C. Biezen and O.A Ojo; "An evolutionary architecture for motion-compensated 100 Hz television"; *IEEE Trans. on Circuits and Systems for Video Technology,* Volume 5, Issue 3, June 1995, pages:207 – 217.

[7] E.G.T. Jaspers, P.H.N. de With and G.W.M. Janssen; "A flexible heterogeneous video processor system for television applications"; *IEEE Trans. on Consumer Electronics*, Volume 45, Issue 1, Feb. 1999, pages:1 – 12.

[8] P.H.N. de With, P.H. Frencken and M. van der Schaar-Mitrea; "An MPEG decoder with embedded compression for memory reduction"; *IEEE trans. on Consumer Electronics*, Volume 44, issue 3, Aug. 1998, pp. 545 - 555.

[9] E.G.T. Jaspers, P.H.N. de With; "Compression for reduction of off-chip video bandwidth"; *Proceedings of the SPIE The International Society for Optical Engineering (USA)*, vol. 4674, pp.110-20, 2001.

[10] R. P. Kleihorst and R. J. van der Vleuten, "DCT-domain embedded memory compression for hybrid video coders," *J. VLSI Signal Processing*, vol. 24, pp. 31–41, Feb. 2000.

[11] R. Peset Llopis et al., "A low-cost and low-power multi-standard video encoder"; *First IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pp.97-102, Oct. 1-3, 2003.

[12] Recommendation ITU-R BT.601-5: "Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios" and Recommendation ITU-R BT.656-3: "Interfaces for digital component video signals in 525-line and 625-line television systems operating at the 4:2:2 level of Recommendation ITU-R BT.601 (Part A)"

[13] Dave Shreiner, Mason Woo, Jackie Neider, Tom Davis; "OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2 (5th edition)"; *Addison Wesley Professional,* 2005.

[14] R. J. van der Vleuten, "Low-complexity lossless and fine-granularity scalable near-lossless compression of color images"; *Data Compression Conference (DCC 2002), (Snowbird, UT)*, p. 477, Apr. 2-4, 2002.

[15] R. J. van der Vleuten and S. Egner, "Lossless and fine-granularity scalable near-lossless color image compression" *25th Symp. Inform. Theory in the Benelux, (Kerkrade, The Netherlands)*, pp. 209-216, June 2-4, 2004.