

A PARALLEL MPEG-4 ENCODER FOR FPGA BASED MULTIPROCESSOR SOC

Olli Lehtoranta, Erno Salminen, Ari Kulmala, Marko Hännikäinen, and Timo D. Hämäläinen

Tampere University of Technology, Institute of Digital and Computer Systems,
P.O. Box 553, Korkeakoulunkatu 1, FI-33101 Tampere, Finland
email: olli.lehtoranta@tut.fi, timo.d.hämäläinen@tut.fi

ABSTRACT

A parallel MPEG-4 Simple Profile encoder for FPGA based multiprocessor System-on-Chip (SOC) is presented. The goal is a computationally scalable framework independent of platform. The scalability is achieved by spatial parallelization where images are divided to horizontal slices. Slice coding tasks are mapped to the multiprocessor consisting of four soft-cores arranged into master-slave configuration. Also, the shared memory model is adopted where large images are stored in shared external memory while small on-chip buffers are used for processing. The interconnections between memories and processors are realized with our HIBI network. Our main contributions are the scalable encoder framework as well as methods for coping with limited memory of FPGA. The current software only implementation processes 6 QCIF frames/s with three encoding slaves. In practice, speed-ups of 1.7 and 2.3 have been measured with two and three slaves, respectively. FPGA utilization of current implementation is 59% requiring 24 207 logic elements on Altera Stratix EP1S40.

1. INTRODUCTION

Video is becoming an essential part of mobile multimedia terminals. There are, however, many contradicting constraints to be met in video codec implementations. One challenge is the rapid evolution of compression standards with several different algorithms. This requires programmability that is naturally not a problem for processor based platforms. However, achieving the best power, energy, and silicon area efficiency requires custom hardware implementations. On the other hand, HW design cycle is more demanding than SW development, and any modification is very expensive and time consuming. For example, Non Recurring Engineering (NRE) costs, especially fabrication costs, increase rapidly with each technology generation making frequent HW upgrades less favorable. Software implementation solves the flexibility and upgradeability problem but is not an optimal solution from performance versus silicon area point of view.

The work in this paper solves the HW video codec design flexibility and upgradeability problem with a fully programmable Multiprocessor System-on-Chip (MPSOC) approach [1]. The key idea is to use synthesizable soft-core processors and a synthesizable SOC interconnection network, which allows prototyping and implementation on any FPGA platform, or on an ASIC technology with a rapid design cycle. In addition, our implementation framework enables a seamless trading off between performance and area without extra burden in system design by scaling the number of identical processors.

Typical HW/SW encoders combine a RISC processor, HW accelerators connected to as a functional pipeline [2], and shared memories as well as buses. A MPEG-4 Simple Profile (SP) encoder in [3] requires 828k transistors in 0.35 μ m CMOS and achieves 30 CIF frames/s at 40 MHz. In [4], an FPGA based H.263 encoder is demonstrated requiring 400 k gates for HW accelerators while providing 30 QCIF frames/s at 12 MHz. In [5][6], FPGA based HW accelerated H.264 encoders are presented. An interface between a host PC and a FPGA based MPEG-4 encoder is build in [7] enabling fast prototyping and debugging. However, the aforementioned FPGA designs concentrate on single encoder cores while the proposed implementation is one the first utilizing multiple parallel encoders on FPGA.

In this paper, we use Altera FPGA as the target platform [8], Altera Nios processors, and our Heterogeneous IP Block Interconnection (HIBI) [9] as the communication network. No special HW accelerators are currently used, but HIBI provides a very convenient plug-and-play method to add IP blocks independent of the FPGA vendor. As a test case, we use MPEG-4 SP encoder implementation operating on QCIF video format (176x144 pixels). Topics of interest are practical implementation issues, such as utilized FPGA resources and achieved performance, design cycle improvement, as well as encoder specific issues like memory optimization needed due to scarce on-chip memories. The implementation works in practice with an FPGA board attached to a PC that sends source video streams and receives compressed data.

This paper is organized as follows. We first consider video encoding and present our scalability approach that is

based on horizontal spatial data parallelization in Section 2. In Section 3, our MPSOC architecture is described while Section 4 explains the SW implementation in low memory conditions. The results are presented in Section 5. Finally, Section 6 summarizes the paper and discusses future work.

2. HORIZONTAL SPATIAL PARALLELIZATION

Video encoder parallelization approaches can be categorized to temporal, functional, instruction level, sub-word level, and spatial methods [2]. This work, however, concentrates on the spatial method due to possibilities to control granularity, e.g., macroblock row, macroblock and block level image sub-divisions are possible.

Spatial parallelization can be performed with vertical, horizontal, rectangular, or arbitrary shaped slices. The problem of vertical parallelization, such as on the left side of Fig. 1, is that predictive coding is not considered leading to Motion Vector (MV) and DQUANT (denoting changes in Quantization Parameter (QP)) dependency problems [10].

Horizontal spatial partitioning, however, is natural to macroblock (MB) coding order. The right side of Fig. 1 depicts our previous implementation on a distributed memory DSP using MB row granularity [10]. The reconstructed images are made slightly overlapping in order to allow motion vectors to point over slice boundaries. The overlapping areas are also exchanged between processor after local reconstruction. Prediction dependencies are eliminated by inserting slice headers such as H.263 Group-Of-Block (GOB) or MPEG-4 Video Packet Headers (VPH) in the beginning of a slice. Clearly, this results in some overhead but prediction dependencies are avoided.

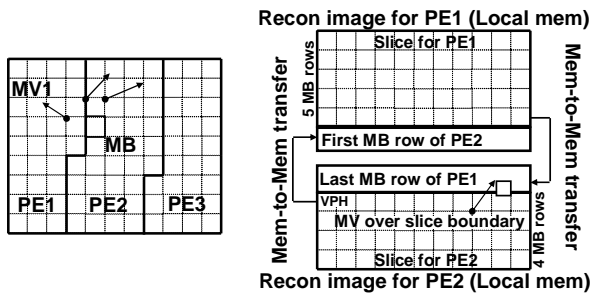


Fig. 1. Motion vector dependency problem in vertical parallelization (left) and horizontal parallelization for distributed memory machines (right).

However, a drawback of [10] is a somewhat coarse granularity leading to unbalanced computational loads, i.e., unequal size of slices. Therefore, the original approach is improved by sub-dividing images using macroblock granularity as in Fig. 2. Inter-processor communication and overlapping are further avoided by exploiting a shared

memory. The new method is highly scalable since the whole image is assignable to a single processor while the largest configuration dedicates a processor for each MB. No inter-processor communication is needed since data can be read directly from the global memory buffer. The shared memory, however, is a potential bottlenecked, and thus HW platform must be carefully designed.

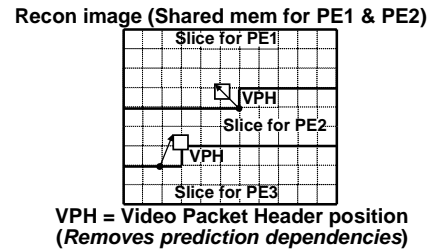


Fig. 2. Horizontal data parallelization for shared memory.

3. MPSOC ARCHITECTURE

From the prototyping point of view, it is desirable to support the most common communication models used in multiprocessor systems, i.e., message passing and shared memory. Therefore, efficient interconnections for HW modules are needed.

Our scalable MPSOC architecture is presented in Fig 3. a). The MPSOC consists of a Nios I and three Nios II (fast core version) processors arranged into a master-slave configuration. Both these 32-bit RISC variants are used as soft-cores and support extensions to Instruction Set Architecture (ISA) as well as customization of peripherals. The processors run at 70 MHz and have been synthesized into Altera Stratix EP1S40 [8]. We use different Nios variants for two reasons. First, Nios I is needed because we use a TCP/IP stack on Ethernet originally developed for Nios I. Secondly, it is demonstrated that our HIBI is well suited for supporting different types of processors and memories, i.e., suitable for heterogenous systems.

In the system, the master is responsible for controlling slaves and transferring data with an external host PC computer using a 100 Mbps Ethernet. The slaves are dedicated to application specific processing. As depicted in Fig 3. b), the identical slaves consist of a CPU core, peripherals, as well as interconnection logic including 1 KB receive (rx)/transmit (tx) buffer and Nios2Hibi Direct Memory Access (N2H DMA).

A 32-bit HIBI interconnection connects all processors and an external 16 MB SDRAM, providing support for message passing and shared SDRAM. The Nios bus interface is implemented with a Dual Port RAM (DPRAM) based ring buffer and a N2H DMA module. The task of N2H is to move data between HIBI and the ring buffer. For example, a processor performs a HIBI write operation by first copying data to the ring buffer, then configuring N2H

to write the desired number of words, and by finally starting the transfer. N2H reports completed data transfers with interrupts (IRQs), and thus processors can perform computations in parallel with data transfers. A polling mode is also supported, in which the completion of reads/writes are detected by monitoring registers of N2H.

The SDRAM interface is realized by mapping the HIBI commands via 32-bit SDRAM DMA which transfers data between SDRAM and HIBI. The 1 MB external SRAM is reserved for program memory and is connected to processors via the Altera's Avalon bus. In the future, also the program memory will be connected via HIBI. Currently, the program memory is divided to two sections of which the first 512 KB is dedicated to the master, while the rest is reserved for slaves. However, slaves receive the same identical program binary since the slaves operate in the single program multiple data mode. The shared program memory bottleneck is circumvented by using instruction caches.

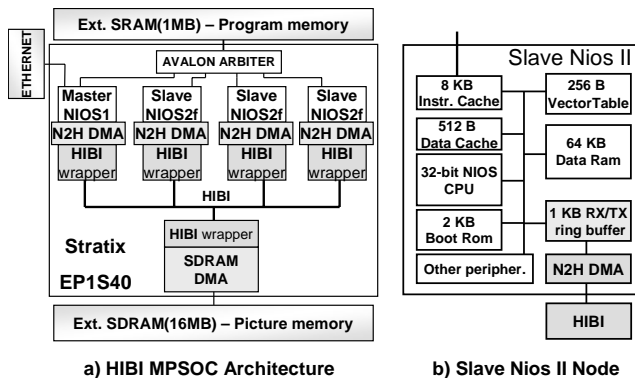


Fig. 3. MPSOC HW architecture

4. SOFTWARE IMPLEMENTATION

Software implementation is divided to three program types that are a host PC user interface, a master Nios I control program, and slave MPEG-4 encoders. The flow graphs and the synchronization of SW are depicted in Fig. 4 while implementation details are explained in the following.

4.1. Host PC Tasks

The host PC implements a user interface for inputting encoding parameters to the master. The user interface enables the selection of a video format (resolution and frame rate), bit rate control mode (constant or variable), Quantization Parameter (QP), as well as the number of slaves used in the encoding. The host PC and the master Nios communicate via a custom UDP/IP based messaging protocol, which supports its own flow control, re-transmissions, and packet structures, fragmentation, and

assembly. Our messaging protocol allows real-time modification of the frame rate, QP and bit rate parameters during encoding.

The tasks of the host PC include the capturing and loading a raw video image, sending raw data to the master Nios, as well as decoding the output. Received bits are stored to the local disk for debugging. In addition, the host PC measures statistics such as the average encoding frame rate and bit rate. At any time, the host PC can issue a re-initialization command causing Nios processors to stop, release dynamically allocated SW resources, e.g., memory, and return to the initial state. For example, this feature enables changing video resolution and number of slaves without re-booting the platform. Also, prototyping and testability friendliness is improved since several video formats can be successively tested just by changing the parameters.

4.2. Master Nios Tasks

The tasks of the master Nios are illustrated in the middle of Fig 4. The tasks include communicating with the host, defining horizontal slices as in Fig. 2, configuring parameters of slaves, synchronization, and merging of encoded slices. All parameterization is performed via shared SDRAM. However, since N2H DMA is not connected to the data and memory buses of CPU, one cannot refer to SDRAM via pointers. For example, the C language statement `sdramVariable = pSdramAddr[0]` is not possible.

Instead, the communication requires four steps depicted in Fig 5. First, the master allocates memory from SDRAM. Second, the master prepares local copies of parameters in local RAM. Third, the master uploads the local copies to the SDRAM, after which slaves download the data to their local memories in the fourth step. Thus, the programmer is responsible for keeping data structures in a coherent state.

However, the limited 64 KB local SRAM size of FPGA presents a more demanding challenge considering the fact that a raw QCIF image takes 37.1 KB. Our solution is to allocate large memory buffers, such as currently encoded image, reconstructed images, and output bit buffers, on SDRAM. Two additional 1KB buffers are allocated on local SRAM which are used for processing data of large buffers a small segment at a time.

For example, bit stream merging is implemented as illustrated in Fig. 6. As long as there are slave bits remaining, the master reads a small portion of slave's bit stream into the buffer A, concatenates the data with the tail bits of the global bit buffer into the buffer B, writes result to SDRAM, and re-synchronizes the buffer B to the new tail of the global bit buffer. A similar buffering scheme is also used in send bits phase of the master.

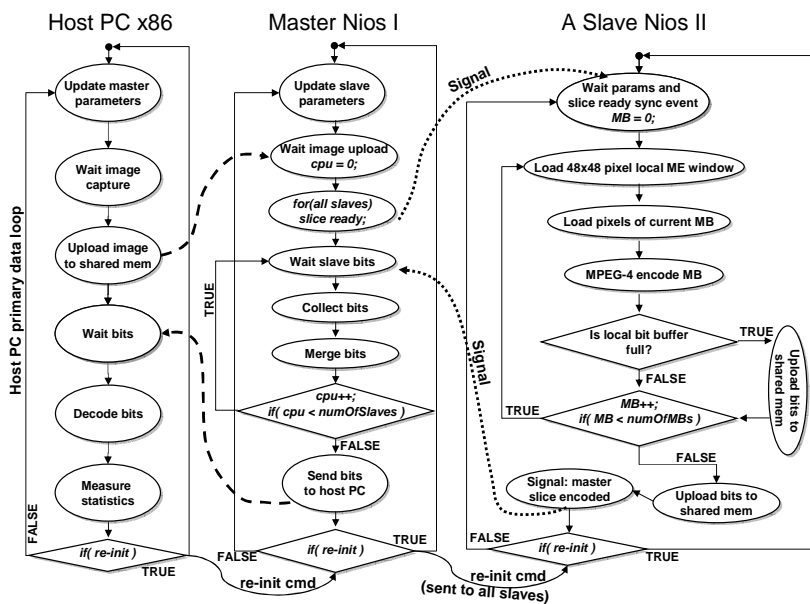


Fig. 4. Software flow graphs and task synchronization.

4.2. Slave Encoder Tasks

The slave processors have been dedicated to parallel encoding. Our MPEG-4 encoder software is identical to [11] except any DSP specific optimizations have been omitted, i.e., platform independent portable ANSI C implementation is being used for Nios II.

Also, the slaves operate in low memory conditions. As Fig. 4 illustrates, the tasks are the loading of source data of Motion Estimation (ME) (reference window and current MB) into local memory, MPEG-4 encoding, and storing bits to SDRAM. The ME process is carried out in a 48x48 sliding window under programmer control as illustrated in Fig. 7 a).

The ME window moves in a raster scan pattern centered on the current MB position. An exception is an image boundary, where the window is clipped inside the image. The ME window loading is optimized by packing four consecutive pixels into a 32-bit word. The data transfers require some CPU intervention since CPU must read data from the N2H ring buffer after transfer complete IRQs. However, IRQs enable overlapping encoding computations with the SDRAM access. The IRQ based communication is used only for the ME window, and other SDRAM reads/writes are performed by polling N2H. Furthermore, overlapping in sub-sequent ME window positions can be used for minimizing accesses to the external memory.

Fig. 7 b) shows two approaches to update the ME window. First, one can load whole window from SDRAM every time the MB position changes, e.g., with DMA. In

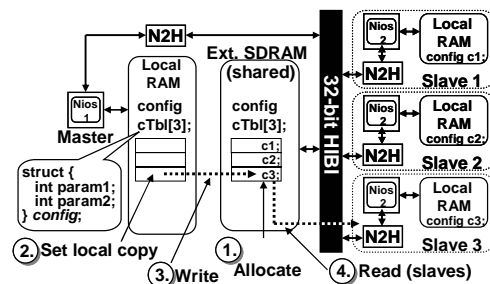


Fig. 5. Slave configuration using SDRAM.

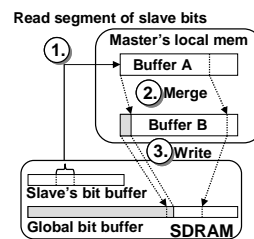


Fig. 6. Master's bit stream merge task.

the second approach, only the rightmost column is loaded from SDRAM while the remaining pixels are copied inside on-chip memory.

The drawback of the first approach is high SDRAM bandwidth requirement, e.g., ME of a 4CIF video (704x576) at 30 frames/s demands 104 MB/s. In comparison, the second approach requires 34 MB/s but the drawback is that 136 Million Operations Per Second (MOPS) are consumed by load/store operations needed for copying. Considering that current SDRAM are fast, e.g., 133 MHz 64-bit SDRAM yields maximum of 8 bytes * 133 M = 1015 MB/s, the first approach is still a viable option due to DMA [10]. Hence, the second approach is well suited for systems with slow SDRAM, while fast SDRAM and DMA can be used to reduce CPU utilization. In this work, we support the second approach for minimizing access to the shared SDRAM.

When ME has been carried out, the rest of the encoding is performed similar to a non-parallel encoder expect that each slave works on a different slice. The slave bit output uses local/global buffering scheme comparable to Fig. 6.

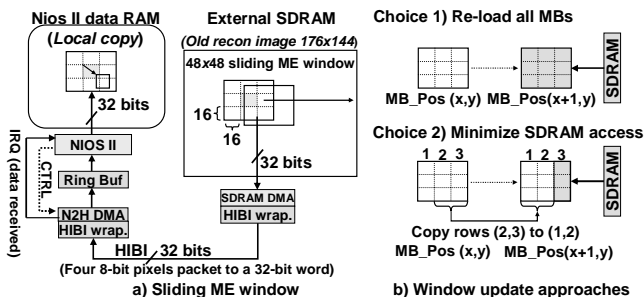


Fig. 7. a) 48x48 sliding window, b) ME window updating.

5. RESULTS

The performance is evaluated by measuring the FPGA utilization, encoding frame rate in a function of number of slaves, and complexities of encoding tasks. All timing procedures for measurements are implemented with HW timers running at CPU clock frequency, i.e., at 70 MHz.

The measurements are carried out with two standard QCIF sequences *news* and *carphone*. The encoder is configured to use IPPP... frame structure, where only the first frame is Intra (I) coded while the rest are Inter (P) coded frames, i.e., motion compensated. All tests are done in Variable Bit Rate (VBR) mode where different bit rates are realized by changing QP. Video sequence relative input/output frame rates are fixed to 30 frames/s in all test cases.

During a benchmarking run, 120 frames of the selected test sequence are encoded. The average encoding frame rate is computed as f_{cpu} / C_{frame} , where f_{cpu} is the CPU frequency and C_{frame} denotes average encoding cycles per QCIF frame. Due to the presence of data/instruction caches, variations in UDP/IP transmissions from a host PC, and IRQ processing, three benchmarking runs are made and their average is reported as the result. In addition, scalability is evaluated by computing the speed-up as FPS_m / FPS_{1_cpu} , where FPS_m is the average frame rate of a multi-slave configuration and FPS_{1_cpu} is the average frame rate with a single slave.

5.1. FPGA Utilization

Table 1 shows the FPGA utilization of MPSOC HW modules where the area is reported in terms of Logic Elements (LE) each consisting of a Look-Up-Table (LUT) and a flip-flop. The statistics have been obtained by synthesizing MPSOC with Quartus II 4.2 into Stratix I. In total, most of LEs are required by Nios processors (26%) while HIBI wrappers consume 24% of the total area. Considering future HW accelerator development, 41% of LE resources are still available. However, 75% of the total on-chip memory is required by the processors, of which 55.5% is due to slaves. Therefore, the memory utilization restricts the system and not the logic utilization. Currently the maximum frequency is dictated by the connection to the external SRAM (see Fig. 3 a).

5.2. Measured Encoding Speed and System Scalability

Fig. 8 and Fig 9 present average encoding frame rates in a function of QP and number of slaves for *news* and *carphone*, respectively. The bit rates are reported relative to the fixed 30 frames/s sequence output rate. The straight lines depict an optimistic speed-up, which is obtained by multiplying the frame rate of a single slave with the total

number of slaves. The frame rates are measured from the master's main encoding loop.

The measurements indicate that the encoder is not sensitive to changes in bit rate although QP is greatly varied. One reason is that the frame rate remains low, since maximum of 6 QCIF frames/s is achieved with three slaves. In practice, this translates into low bit processing rate, e.g., *news* with QP 12 and three slaves results in 6 frames/s * 1 / 30 frames/s * 56 kbps = 11.2 kbps. Average speed-ups for *news* are 1.71 and 2.23 when using two and three slaves, respectively. For *carphone*, the same figures are 1.73 and 2.27. The parallelization efficiencies, i.e., the ratios of encoding frame rates to optimistic projections are 86% and 74% for *news*, and 87% as well as 76% for *carphone*. In comparison, [10] provides efficiencies of 93% and 89% for two and three slave configurations, suggesting that our current Nios SW implementation may have bottlenecks.

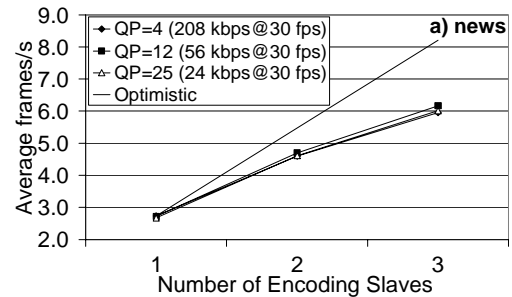


Fig. 8. Measured frame rates for *news.qcif* (176x144).

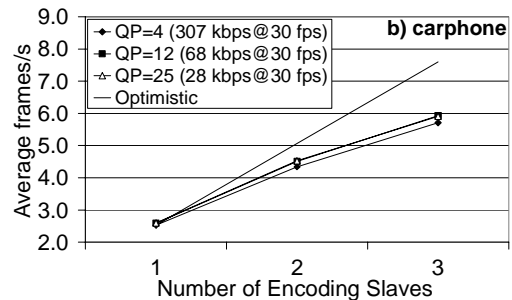


Fig. 9. Measured frame rates for *carphone.qcif* (176x144).

5.3. Relative Complexity of Encoding Tasks

Fig. 10 presents reconstruction of relative complexities of encoding task in a master and a slave with respect to the frame encoding time. The results have been obtained with *carphone* by using QP of 12, three encoding slaves, and collecting statistics from the slave holding the middle slice (See Fig. 2 depicting image sub-division.).

Table 1. FPGA utilization statistics.

HW Module	Module count	Total Mem [KB]	% of Total Mem	Module Area [LE]	Total Area [LE]	% of LE
Master Nios I	1	81.8	19.6	2 897	2 897	7.0
Slave Nios II	3	232.5	55.5	2 580	7 739	18.8
N2H DMA	4	0	0	708	2 832	6.9
HIBI wrapper	5	0	0	1 988	9 940	24.1
SDRAM DMA	1	0	0	799	799	1.9
Utilization		314.3	75.1		24 207	58.7

In the master, waiting image receive (*img rx*) to complete consumes 13% of total time which is directly reflected to slaves due to synchronization. However, 70% of the master's time is spent while waiting for encoding to complete. The bit stream processing is insignificant due to the low bit rate. The section others (*oth*) includes master's IRQ processing, UDP/IP packet processing, and some control code not profiled causing additional delay worth of 17% of the total frame encoding time.

In a slave, 30% (13%+17%) of time is spent waiting for the master. The other significant tasks are ME (22%), accessing data from SDRAM (*com*, 15%), DCT and quantization (6%), and half-pixel interpolations (5%). Also, slave's others (*oth*, 14.5%) consist of IRQ processing and control code not profiled.

Based on measurements, it is deduced that the waiting in slaves is currently limiting scalability. However, *img rx* could be eliminated by scheduling this task in parallel with encoding, e.g., by using double buffering [10]. By eliminating *img rx*, the encoding frame rate would improve from 5.9 frames/s to $70M / (11.77M - 0.13 * 11.77M) = 6.8$ frames/s, i.e., the total number of clock cycles per second is divided by the average cycles per QCIF frame minus the clock cycles required by the *img rx* phase.

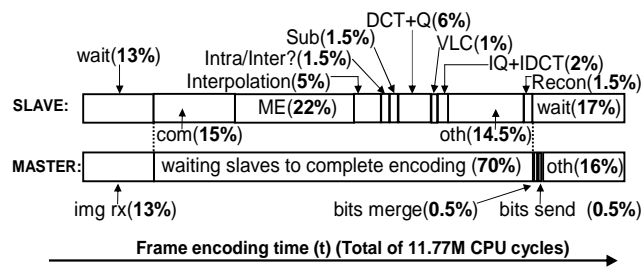


Fig. 10. Relative complexities of encoding tasks.

6. CONCLUSIONS

The spatial parallelization enables a scalable system where the number of encoder modules is selectable according to computational/area requirements. We presented a flexible encoder framework for MPSOC, whose debugging features enable rapid HW/SW co-development while constantly verifying operation with real video. It was shown that FPGA memory limitations can be circumvented by allocating small on-chip buffers which are used to access

shared memory a segment at a time. In the future, we will develop HW accelerator integration to the SW framework.

7. REFERENCES

- [1] E. Salminen, *et al.*, "HIBI-based Multiprocessor SoC on FPGA," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS2005)*, May 2005, pp. 3351-3354.
- [2] I. Ahmad, *et al.*, "Video Compression with Parallel Processing," *Elsevier Parallel Computing*, vol. 28, issue 7-8, pp. 1039-1078, 2002.
- [3] W.-M. Chao, *et al.*, "Platform Architecture Design for MPEG-4 Video Coding," in *Proc. International Conference on Image Processing (ICIP2003)*, vol. 3, Sept. 2003, pp. 93-96.
- [4] M. Garrido, *et al.*, "An FPGA Implementation of a Flexible Architecture for H.263 Video Coding," *IEEE Transactions on Consumer Electronics*, vol. 48, no. 4, pp. 1056-1066, Nov. 2002.
- [5] R. Kordasiewicz, *et al.*, "Hardware Implementation of the Optimized Transform and Quantization Blocks of H.264," in *Proc. Canadian Conference on Electrical and Computer Engineering (CCECE2004)*, vol. 2, May 2004, pp. 943-946.
- [6] Q. Peng, *et al.*, "H.264 Codec System-On-Chip Design and Verification," in *Proc. 5th International Conference on ASIC*, vol. 2, Oct. 2003, pp. 922-925.
- [7] Y.-Li Lin, *et al.*, "Versatile PC/FPGA Based Verification/Fast Prototyping Platform with Multimedia Applications," in *Proc. 21st IEEE Instrumentation and Measurement Technology Conference (IMTC2004)*, vol. 2, May 2004, pp. 1490-1495.
- [8] Altera, "Nios Development Board: Reference Manual, Stratix Professional Edition," Available via http://www.altera.com/literature/manual/mnl_nios_board_stratix_1s40.pdf, document version July 1.1, (2003) (Site visited on 22.03.2005)
- [9] E. Salminen, *et al.*, "HIBI v.2 Communication Network for System-on-Chip," In Pimentel, A.D., Vassiliadis, S. (eds.): *LNCS 3133 Computer Systems: Architectures, Modeling, and Simulation*, Springer-Verlag, Berlin, pp. 412-422, 2004.
- [10] O. Lehtoranta, *et al.*, "Parallel Implementation of Video Encoder on Quad DSP System," *Elsevier Microprocessors and Microsystems*, vol. 26, issue 1, pp. 1-15, 2002.
- [11] O. Lehtoranta and T. D. Hämäläinen, "Feasibility Study of a Real-Time Operating System for a Multichannel MPEG-4 Encoder," In Creutzburg, R., Takala J.H. (eds.): *Proceedings of SPIE: Multimedia on Mobile Devices*, vol. 5684, pp. 292-299, 2005.