



## Scalable Architecture for SoC Video Encoders

TERO KANGAS AND TIMO D. HÄMÄLÄINEN

*Tampere University of Technology, Institute of Digital and Computer Systems,  
P.O. Box 553, FI-33101 Tampere, Finland*

KIMMO KUUSILINNA

*Nokia Research Center, Tampere, Finland*

**Published online:** 27 May 2006

**Abstract.** Evolving video coding standards demand functional flexibility for implementations, not only at design time but also after fabrication. This paper presents a System-on-Chip design approach with a feasible combination of performance, scalability, programmability, area efficiency, and design time effort for a video encoder. The encoder is based on a homogeneous master-slave processor architecture. Each slave encodes a part of the frame in the Single Program Multiple Data (SPMD) data parallel model. Both shared and distributed memory architectures are presented. Design effort is reduced by identical program codes, automated assembly of software and hardware modules independent of the number and type of processors, as well as our flexible on-chip communication network called Heterogeneous IP Block Interconnection (HIBI). A case study implementation with two to ten simple ARM7 processors, 32-bit HIBI bus and non-optimized processor-independent software gives the performance from 6 to 53 fps for QCIF. The whole encoder area ranges from 173 to 770 kgates excluding the memories. The relation scales reasonably well to systems with more powerful processors and optimized code. The optimization of the communication network shows that with more than six slaves even a serial HIBI connection with 100 MHz speed is feasible. HIBI and the parallelization approach allow exploration and optimization of the communication both at the application and architecture layers.

**Keywords:** System-on-Chip, video encoding, architecture exploration

### 1. Introduction

Video encoding calls for high processing power, but maximizing performance and minimizing costs are not the only design problems. The continuous progress in video encoding standardization [1] demands functional flexibility as new and optional features should be included also after fabrication. In addition, the effort in design time and manpower should be decreased at the same time. Altogether this development favors programmable processor-based architectures, but at the moment dedicated implementations have gained most attention in the system-on-chip domain. Focusing on these issues, we present a SoC design that optimizes the

combination of performance, programmability, area, and design time.

Despite other design factors, the processing performance cannot be compromised. Since the operating frequency and voltage have practical and economical limits, execution parallelism has to be utilized. Surveys of the parallelization strategies for both dedicated, programmable and hybrid architectures are given in [2, 3].

The scalability and the programmability of presented video encoder architectures are often limited in parallelization strategy and computation platform. Both the instruction and task level parallelism are inherent in video encoding. Many of the proposed architectures, however, cannot exploit the

task-level parallelism at all. In video encoding, data and task-level parallelism scale very well since the computations can be addressed to multiple data, such as pixels, blocks, macroblocks (MB), or frames, simultaneously.

Best programmability is obtained with general-purpose processors. In order to achieve a sufficient performance, both homogeneous [4] and heterogeneous [5, 6] multiprocessor architectures have been developed. However, these are not suitable for portable systems due to the costly hardware and high energy consumption. This is improved in dedicated encoding architectures [7], which means also limited programmability. Many architectures are neither purely dedicated nor programmable but combine the best of both approaches by utilizing co-processors that accelerate regular and computation-intensive tasks [8–11]. For portable devices, programmability is often sacrificed for other design requirements. Most of those architectures rely on dedicated processing elements that give the highest processing efficiency with respect to the hardware usage and power consumption [12–17]. As a drawback, dedicated architectures are non-flexible as they are adapted to the processing of a specific algorithm.

In our approach, the encoder architecture is based on scalable homogeneous multiprocessing and a SoC communication network called HIBI [18]. The encoder architecture follows *Single Program Multiple Data* (SPMD) model of parallelism and the master-slave processor configuration. In this paper we utilize low-cost RISC cores, but more powerful processors can also be used. It should be noted that we focus issues on SoC architecture, design effort and flexibility in implementation, for which reason the encoder software optimiza-

tion and the highest raw performance does not have a central role.

The paper is organized as follows. Section 2 introduces our HIBI-based, scalable SoC platform for multimedia applications. The H.263-based [19] video encoder algorithm and the task-level parallelization method are examined in Section 3. Section 4 presents the video encoder architecture with an emphasis on the system-level memory architecture choices. In Section 5, the verification environment is briefly described. In Section 6, the encoder performance results are given together with the area and the power estimations. It also analyzes the optimization of the clock frequencies and data widths of the platform. Although software optimization does not have a central role in this design, the overall encoding performance with respect to communication and computation complexity is analyzed. In Section 7, the concluding remarks are given.

## 2. SoC Platform

Figure 1 depicts an overall block diagram of the HIBI based SoC platform. It facilitates the separation of communication from computation both on application and architecture levels. In the architecture, the communication between IP-blocks is realized with the HIBI network [18]. HIBI is well-suited for streaming transfers that is inherent in video encoding because of its high throughput and low latency.

The hierarchical architecture, depicted in Figure 1, is easily scalable and parameterizable for a variety of communication requirements ranging from low-bandwidth arbitrary message and control exchange to simultaneous high-bandwidth transfers. Due to the

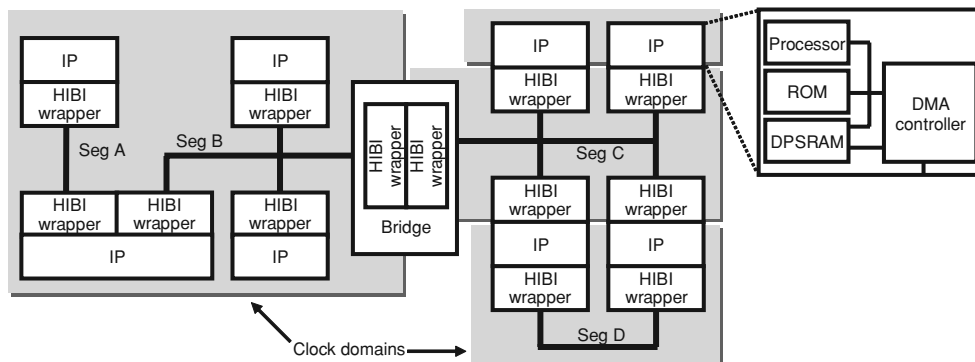


Figure 1. Example of hierarchical HIBI network with multiple clock domains and bus segments.

lack of centralized arbitration mechanism, HIBI can be scaled by duplicating the *wrappers* and connecting them to shared signals. Hierarchical sub-systems (i.e. segments) enable the localization of communication and the performance scaling of each segment separately. The communication on a segment is arbitrated at two levels. First, time division multiple access (TDMA) is applied to divide the time into repetitive slots to guarantee each wrapper access to the communication channel. In the second level, the unassigned or unused time slots are arbitrated by either round-robin or priority based competition.

HIBI wrapper is a parameterizable HW component that is used to construct modular, hierarchical bus structures with the distributed arbitration and multiple clock domains. Configurability takes place both at synthesis time (e.g. data width and buffer sizes) and at runtime (TDMA and competition arbitration parameters). This simplifies design and allows reuse since the same wrapper can always be utilized.

To facilitate both scalability and programmability in the video encoder, the computation is performed in parallel identical processors. HIBI does not limit the type of processors that depends, among other things, on performance requirements, support for specific computation like floating point functions, parallelization granularity of the algorithm, amount of existing reusable code, and economical factors. For the case study we chose RISC processor, since we preferred straightforward compiling of the generic program codes without hand-optimization. Compilers perform usually better on such a simple processor architectures. The processor is ARM7TDMI [20], since it is low-price, small, and provides a simple interface for connecting it to HIBI. The performance is moderate, but the focus is on the design approach and not on the highest performance.

### 2.1. Interfacing a Processor to the Communication Architecture

Both communication architectures and processors have specific interfaces that are often incompatible with each other. The simplest version of HIBI has a FIFO-based wrapper for connecting IP-blocks [18]. The HIBI interface supports also more complex interfaces such as the VCI [21] or OCP [22]. If the interfaces do not match, the wrapper and IP block are connected with an *adapter*. Our adapter for ARM7TDMI connects to the processor memory bus. The adapter also includes a DMA and interrupt controller that [23] enables complex transfer

schemes controlled through a set of memory mapped registers. The processor can initiate single and block data write and read requests.

Multiple data transfers can be active simultaneously. Transfers are carried out in the background while the processor is processing buffered data. The DMA controller also enables prioritized transfers that utilize the parallel FIFO buffers of the HIBI wrapper. Since the DMA controller is connected to the processor data memory bus, it is not dedicated only for ARM7TDMI but can be used with other processors as well, with minor modifications to accommodate the processor interrupts.

The ARM7TDMI, local program and data memories, and the DMA controller comprise a processing unit that can be connected to a HIBI wrapper as shown in Figure 1. The SoC platform for the multimedia system is formed by instantiating these processing units and connecting them to the HIBI system.

## 3. Parallelization of a Video Encoder Algorithm

In the case study we implemented a video encoder similar to MPEG-4 simple profile on the HIBI platform. Table 1 lists the key properties of the algorithm. The execution of the encoding algorithm is parallelized utilizing the task-level approach presented in [4], in which the same algorithm was implemented on a quad-DSP platform.

The basic idea is to divide the image into slices comprising at least one group-of-block (GOB) and to distribute the encoding for processing elements as depicted in Figure 2. As the standard specifies, a GOB

Table 1. Properties of the implemented video encoding algorithm.

Property	Value
Algorithm	H.263 version 1
Coding modes	All but the optional modes (such as OBMC modes and PB-frames)
Motion vector range	[−15.5, 15.5]
Images sizes	sub-QCIF, QCIF, CIF, 4CIF, 16CIF
Motion estimation options	New-diamond search, logarithmic search, sub-sampled full search, hexagonal search
Motion estimation accuracy	1/2-pixel
Test sequence	foreman (QCIF)
Color sampling	4:2:0

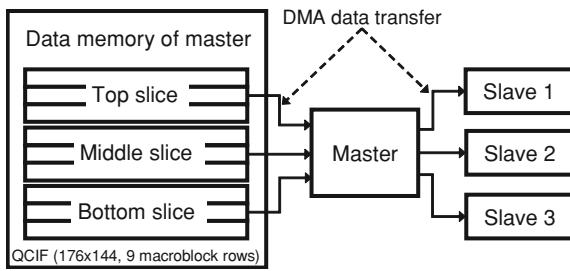


Figure 2. Parallelization of the execution of video encoding.

Table 2. Number of GOBs in different picture formats.

Picture format	Frame height	GOB height	Number of GOBs
sub-QCIF	96	16	6
QCIF	144	16	9
CIF	288	16	18
4CIF	576	32	18
16CIF	1152	64	18

comprises  $k \cdot 16$  lines of pixels where  $k$  depends on the picture formats. The size and the number of GOB for each picture formats is tabulated in Table 2. Each slave processor carries out the complete encoding of a given slice.

### 3.1. Encoding Flow

The video encoding, including DCT, motion estimation, and bitstream coding, is carried out by the slave processing units. The master controls the encoding, dis-

tributes the macroblock rows to slaves, collects and merges the bit streams, and handles the global rate control. The I/O module takes care of raw video input and compressed video output streams. Figure 3 depicts the logical communication links between modules. The symbols denote the amount of the data transferred over the link and are examined later in the following Section.

The master module initializes the encoding parameters of a slave module during system initialization. These parameters include, for instance, information about the slice coordinates, neighboring slaves, image formats, and the quantization factor. Each of the slave processors has an identical program following the SPMD model of parallelism. Despite the fact that this may result in small overhead in code size, it still eases the software development considerably. If a shared program memory with small local program caches is used, a single binary, in fact, reduces the total code size. The number of slaves is defined with a single constant in a header file for the master processor code.

In addition to the advantages in software development, the homogeneous structure of the video encoder enables easy performance scaling by changing the amount of slave processors. In the utilized parallelization strategy, the number of slave processors can be at most the number of GOBs in a frame that depends on the picture format as shown in Table 2.

### 3.2. Software Development

The video encoder algorithms were described in standard ANSI C code and the utilized algorithms were not optimized for any certain processor although floating point calculations were totally avoided. The generic C

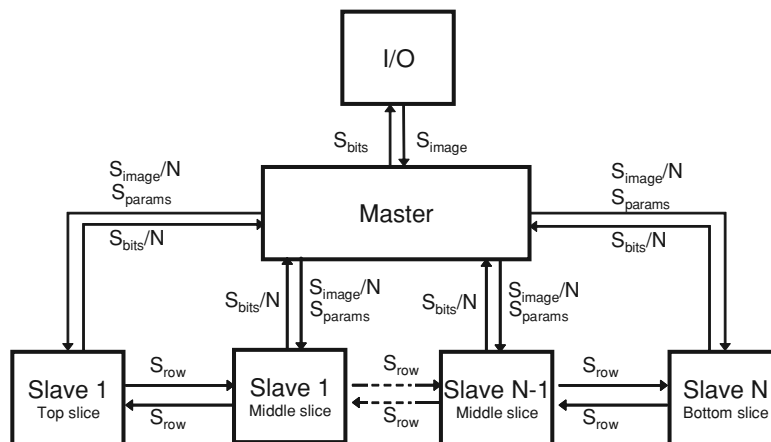


Figure 3. Logical communication links.

code enables fast software development and portability of the algorithms to other processor architectures.

The encoding software was first developed so that the SPMD parallelism could be simulated with a single processor. The encoding process of each slave is carried out sequentially and the communication between the slaves is implemented with shared variables. To implement the video encoder in the target SoC platform, the communication functions had to be modified to use the communication scheme provided by the DMA controller.

Only two distinct software images are created from the C source code. The first includes the object codes for algorithms needed by the master processor and the other is for the slave processor. Each slave has an identical copy that does not depend on the number of slaves. The information on the number of slaves is included in the master's program.

Two encoding software versions were created to evaluate both the *local* and the *shared* data memory implementations. The differences and the criteria for selecting the suitable memory approach for a video encoder application is analyzed in more detail in the following sections. Most of the additions to shared memory version are related to updating the motion estimation window. Instead of accessing the reference image from the local memory, the slave encoder has to read the required macroblock from the shared memory. Moreover, the slave processors do not communicate directly but through the shared memory.

#### 4. Video Encoder Architecture

The structure of the video encoder is depicted in Figure 4. It consists of the instantiated processor modules

for video encoding and an I/O module for connecting the SoC to external systems such as the frame buffer of a camera device. In the simulations, the master sends the merged bitstream to the I/O module that stores the stream into a file. For a mere video encoding system, most of the master processor resources are wasted due to the light processing load. Utilization of a separate control is justified assuming that the video encoder is a part of a more complex system such as an entire base-band processor. Mastering the video encoding would then only be one of the tasks on the control processor.

##### 4.1. Local and Shared Memory Approaches

In a parallel multiprocessor system, it is a trade-off between several factors whether the data and program of a processor should be stored in a local memories of each processor or to a global shared memory. In the local memory approach, the program and/or data are buffered in a memory that is directly connected to the memory bus of the processor interface. The latency of the memory accesses can be kept low and deterministic as there is no congestion between the other processors. In the shared memory approach, the processors use a global shared memory for buffering most or all of the program and/or data. As the shared memory is located further from the processor, the drawbacks include longer latencies and non-deterministic access times.

In this paper, the local memory approach means that all the required data for encoding a slice of a frame is stored in the slave processor. In the shared memory approach, only the most frequently used program or data, such as current macroblock and motion estimation window, are stored in the local buffer of the slave module. For buffering all the other instructions or data, the slaves utilize a shared memory that is external to

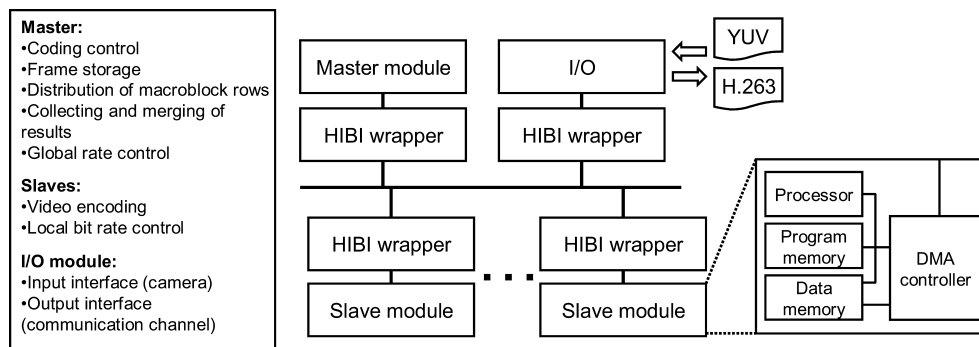


Figure 4. Video encoder architecture and the tasks of each processor. Both master and slave modules have the same internal structure.

Table 3. Memory requirements for local data memory approach.

	Master [bytes]	Slaves [bytes]	
Bitstream	$2*S_{bit}$	$N*S_{bit}$	
Raw image	$2*S_{image}$	$3*S_{image}$	
Overlapping MB rows	0	$(N-1)*4*S_{image}/(\text{MB rows})$	
Temporary storage	6000	$N*6000$	
Program code	10000	$N*25000$	
Picture format	Bitstream $S_{bit}$ [bytes]	Raw image $S_{image}$ [bytes]	Number of MB rows
sub-QCIF	3000	18432	6
QCIF	5000	38016	9
CIF	10000	152064	18
4CIF	20000	608256	36

the slave modules. The shared memory for the slaves is assumed to be implemented with the local memory of the master processor since it already buffers the frames coming from the I/O module and the utilization of master processors is low enough to assign it more tasks.

#### 4.2. Shared vs. Local Data Memory Approach

Table 3 depicts the memory requirements of the video encoder for different picture sizes. The symbols used in Table 3a are explained in Table 3b. The master process requires space for incoming bitstream from slaves and combined bitstream ( $2*S_{bit}$ ).  $S_{bit}$  is estimated for each picture format according to the resulting bitstream size (with a substantial marginal) of the most common video sequences. In each slave, storage for three slices of raw image is reserved for one current and two reconstructed frames ( $3*S_{image}$ ). Each slave stores also the overlapping macroblock rows between neighboring slaves for

both reconstructed frames being  $(N-1)*4$  rows in total. The temporary storage refers to the other global variables and stack. The required storage is dominated by the amount of data rather than the program, particularly with higher numbers of slaves.

In the presented architecture, processors have a sizeable amount of common data that is transferred between local memories during encoding as depicted in Table 4. The transfers correspond to the logical links shown in Figure 3. The majority of the overall communication is due to the image transfers in the I/O-master and the master-slave links. The portion of macroblock row transfers increases with the number of slave processors.

The processor idle times due to the waiting for data are minimized by double buffering. For instance, the master processes a frame from one buffer at the same time as a new frame is stored in another buffer with a DMA transfer. Similarly, slave processors buffer macroblock rows that are transferred between local memories of the master and the slaves. Double buffering increases memory size but is essential for pipelined parallel processing. Design issues associated with memory architectures of multimedia processing have been studied in more detail in [24, 25].

Storing the data locally to processor units involves a relatively high area cost. With large internal memories, slaves can store up to three slices the sizes of which depend on the frame size and the number of slaves.

Overall memory consumption can be decreased by utilizing a large common memory and small local buffers. This would increase the communication considerably since processors have to store most of the temporary data in a shared memory. Moreover, the original logical direct links between modules would now circulate through the shared memory. Due to the small local memories, the block size of a transfer would be small as well. Implementing this kind of transfers

Table 4. Data transfer per frame for INTER QCIF using local memory approach.

Data type [bytes]	Number of slave processors								
	1	2	3	4	5	6	7	8	9
Parameters ( $S_{params}$ )	16	32	48	64	80	96	112	128	144
Images ( $S_{image}$ )	76032	76032	76032	76032	76032	76032	76032	76032	76032
MB rows ( $S_{row}$ )	0	8448	16896	25344	33792	42240	50688	59136	67584
Bitstream ( $S_{bits}$ )	500	500	500	500	500	500	500	500	500
Messages	60	84	108	132	156	180	204	228	252
<b>Total</b>	<b>76608</b>	<b>85096</b>	<b>93584</b>	<b>102072</b>	<b>110560</b>	<b>119048</b>	<b>127536</b>	<b>136024</b>	<b>144512</b>

Table 5. Data transfer per frame for INTER QCIF using shared memory approach.

Data type [bytes]	Number of slave processors								
	1	2	3	4	5	6	7	8	9
Parameters ( $S_{\text{params}}$ )	16	32	48	64	80	96	112	128	144
Images ( $S_{\text{image}}$ )	76032	76032	76032	76032	76032	76032	76032	76032	76032
ME windows	190080	190080	190080	190080	190080	190080	190080	190080	190080
Bitstream ( $S_{\text{bits}}$ )	500	500	500	500	500	500	500	500	500
Messages	60	84	108	132	156	180	204	228	252
<b>Total</b>	<b>266688</b>	<b>266728</b>	<b>266768</b>	<b>266808</b>	<b>266848</b>	<b>266888</b>	<b>266928</b>	<b>266968</b>	<b>267008</b>

requires a very low latency communication architecture or double buffering to hide the latency. Table 5 shows the data transfers incurred by the shared data memory approach. The total amount of data transfers is nearly independent of the number of slave processors since the direct communication between them is negligible.

Most of the video encoding tasks are performed for macroblocks, which has to be taken into account in defining a reasonable minimum size for the local memory. In motion estimation, local memory should be reserved at least for one macroblock from the current frame and for an estimation window including a number of macroblocks from the reference frame. With a search range of 15 pixels, the estimation window comprises nine macroblocks. Moreover, space is needed for encoding parameters, bitstream buffers, and storage of other temporary data as tabulated in Table 6.

Figure 5 compares the total memory requirements between shared data memory and pure local memory approaches. As mentioned, the slave processor utilizes the local memory of the master processor as their shared memory. The difference between the memory

Table 6. Memory requirements for shared data memory approach.

	Master [bytes]	Slaves [bytes]
Bitstream	$2 * S_{\text{bit}}$	$N * S_{\text{bit}}$
Raw Image	$3 * S_{\text{image}}$	0
ME window	0	$N * (9 * 384)$
Current & recon MB	0	$N * (2 * 384)$
Temporary storage	6000	$N * 6000$
Program code	10000	$N * 25000$

approaches becomes significant when the number of slaves increases. In both approaches, the overall data memory size increase linearly. Due to the increase of overlapping data, the scaling factor is considerably higher with the local memory approach.

#### 4.3. Shared Instruction Memory Approach

The sizes of the instruction memories for the master and the slave are 10 and 25 kilobytes independent of

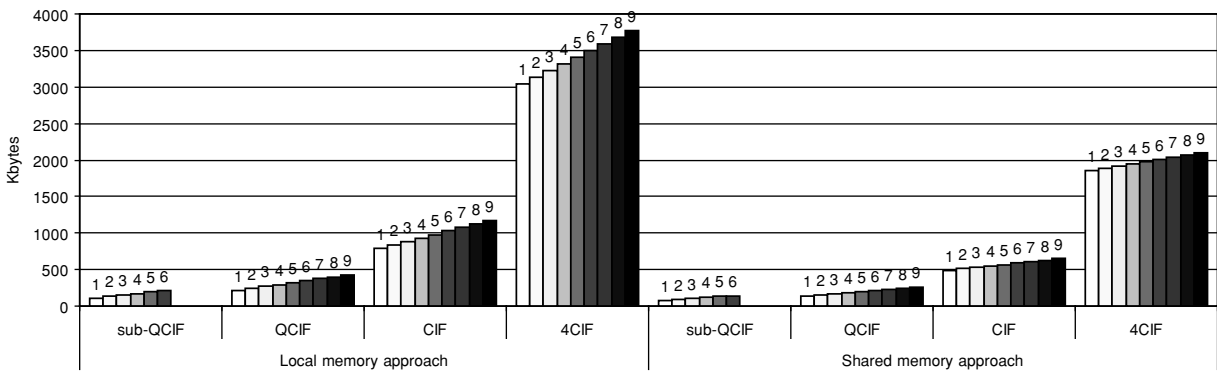


Figure 5. Data memory requirements for both local memory and shared memory approaches. The digit above each bar denotes the number of slave processors.

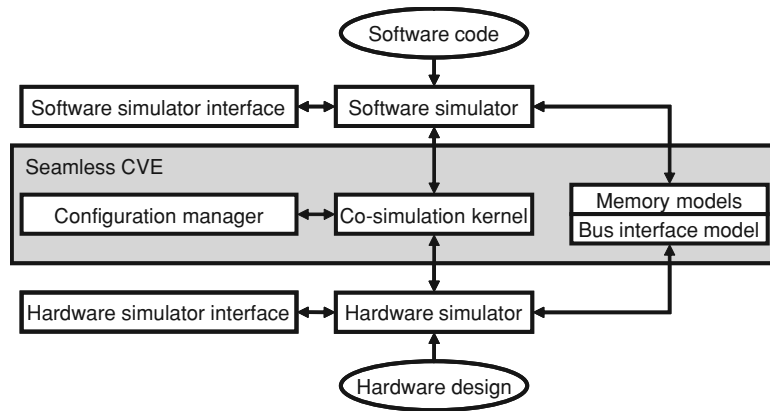


Figure 6. Co-simulation environment.

the number of slave processors. Since the slaves have identical programs it would be attractive to apply the shared memory approach also to instruction memories. The efficient use of a shared instruction memory requires, however, the use of local instruction caches that abates the advantages of this method. Moreover, the total amount of instruction memory is relatively small compared to the data memory (14-35% with QCIF frame size) and, therefore, with this video encoder implementation a shared instruction memory is not necessary.

## 5. Verification

The video encoder verification is carried out with functional simulations in both host (x86) and target (ARM7) platforms as well as RT-level timing simulations by hardware/software co-simulations. To verify the functionality of the video encoder algorithms, the code was first compiled for a simulation platform (x86) and executed in a *direct mode*. The direct mode implies that the parallel processing and the communication between the master and slaves are not simulated. For the better timing accuracy, the code was then profiled and verified in a target processor with an ARM instruction set simulator (AXD) [26]. The timing results of the target platform simulation were later utilized in estimating the entire system performance of different memory approaches.

### 5.1. HW/SW Co-Simulation

Seamless CVE co-simulation tool [27] was used for simulating the communication architecture and proces-

sor bus interface models together with the application program at cycle-accurate level. The simulation architecture of Seamless CVE is depicted in Figure 6. The co-simulation kernel connects and synchronizes the instruction set level software simulation with the memory and processor bus interface models. The pure hardware implementations of the system, such as HIBI network and DMA controller, are executed solely in the hardware simulation. With co-simulations, factors caused by the communication architecture, direct memory accesses, and processor interrupts were taken into account in performance evaluation better than with an instruction set simulator. An obvious drawback with more accurate models is the notable decrease in simulation speed. Co-simulation, however, provides considerably better means to analyze communication performance in large simulations. For example, processor computation stalls due to the congestion in a shared communication resource can easily be detected.

It is tedious to manually build the co-simulation sessions for a number of video encoder instances. In the presented video encoder simulation, building the environment means the creation of directory structures, definition of environment variables, modification of generic parameters in software and hardware description, addition and removal of processor instances in top-level hardware description code, compilation of the codes, and the launch of the simulation. To facilitate rapid evaluation of numerous functional and architectural parameters, scripts were developed to automate the building and running of the simulation. Fast modifications to both the algorithm and the architecture of the video encoder can be made by editing only one configuration file. For instance, the motion

Table 7. Simulation times for encoding one QCIF INTER frame with 900 MHz Sun-fire-880 workstation. The local memory approach of video encoding was utilized in the simulation.

Number of slaves	Simulation cycles	Simulation time [min]	Simulation time per cycles [ms]	Processor cycles per one simulation cycle
1	15363712	517	2.02	1.82E + 06
2	8770271	537	3.67	3.31E + 00
3	5836703	539	5.54	4.99E + 00
4	4982621	1061	12.78	1.15E + 01
5	3718511	1095	17.67	1.59E + 01
6	3688124	1366	22.22	2.00E + 01
7	3688420	1265	20.58	1.85E + 01
8	3272319	1771	32.47	2.92E + 01
9	2010501	908	27.10	2.44E + 01

estimation algorithm can be chosen from a set of C code implementations depending on the encoding quality and performance requirements. All the other files, which the modifications affected, are automatically modified by running the scripts from a command-line user interface.

Running the video encoder HW/SW co-simulation is extremely slow. In order to get reliable results, several frames (both INTRA and INTER) should be encoded. Table 7 depicts the simulation times for encoding one INTER frame. The simulation time is not a linear function of number of slave processes but depends also on the simulation cycles of one frame as well as the load on the simulation workstation. For instance, simulation time for ten processors is smaller than for nine processors because of the fewer encoding cycles per frame. Simulating one cycle of video encoding takes approximately six to seven orders of magnitude more cycles on workstation, just to illustrate the simulation complexity. According to the simulation profiling, most of the complexity is due to the hardware model of the processors. Processor instruction set simulator itself is not a bottleneck although inter-process communication between it and the hardware model may affect simulation speed to some extent.

There are two approaches to accelerate the simulation. We could either use more abstract simulation models or harness more processing resources for simulation. Abstract simulation models are suitable and preferable for early design phases where the accurate timing is not critical. Greater processing resources are required if loss in accuracy is not acceptable. One solution is to distribute the simulation over multiple computers as reported in [28]. In our case, the gain of us-

ing distributed simulation would be approximately the same as the number of simulation computers as long as the number is smaller or equal than the number of processors in the video encoder.

## 6. Results

The video encoder results, including the performance, area, and power, are examined for both the local and the shared data memory approaches. In addition the effects of different frequencies and bus data widths are evaluated to analyze communication and computation complexity with different architectural choices. In the case of HIBI and the DMA controller, the area and power estimations are based on logic synthesis results. The area and the power of ARM7TDMI are 0.53 mm<sup>2</sup> and 0.25 mW/MHz based on [29]. To convert the memory bytes into the gate count, the memory cell sizes, tabulated in Table 8, were applied. The memory power estimations are derived from the memory sizes and the dynamic cell power consumption values.

### 6.1. Analytical Model for Performance Estimation

As Table 7 illustrated, the cycle-accurate hardware and software co-simulation is a time consuming task. To avoid the massive simulations after each architectural or algorithmic modification, an analytical model for performance estimation was created. Instead of co-simulating the system in Seamless CVE, the performance of modified video encoder was estimated executing the parallel implementation sequentially in ARM instruction set simulator (AXD). Obtained cycle counts were then divided by the number of encoding

Table 8. Memory cell size assumptions with 0.18 um CMOS technology.

Property	Equation	Value	Ref.
Technology feature size(F)		0.18 um	
Equivalent gate size ( $G_s$ )	$320 \cdot F^2$	$10.4 \text{ um}^2/\text{gate}$	[30]
SRAM cell size ( $S_s$ )	$140 \cdot F^2/0.7$	$6.5 \text{ um}^2/\text{bit}$	[30]
SRAM gates per bit (avg)	$S_s/G_s$	0.63 gates/bit	
SRAM cell dynamic power		$1.0\text{E-}06 \text{ (mW/MHz)/bit}$	[30]
SRAM cell static power		$4.0\text{E-}07 \text{ mW/bit}$	[30]
ROM cell size ( $R_s$ )	$5 \cdot F^2/0.8$	$0.20 \text{ um}^2/\text{bit}$	[30]
ROM gates per bit (avg)	$R_s/G_s$	0.02 gates/bit	
ROM cell dynamic power		$1.0\text{E-}07 \text{ (mW/MHz)/bit}$	[30]
ROM cell static power		$1.0\text{E-}06 \text{ mW/bit}$	[30]
Area scaling factor of N-port SRAM Compared to single-port SRAM	$1 + 0.6 \cdot (N-1)$		[31]

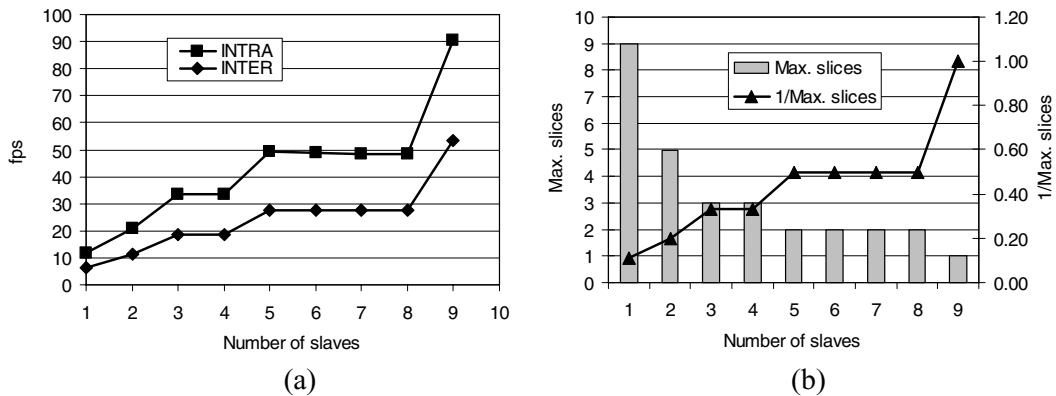


Figure 7. Video encoder performance (a) and the effect of load balancing on the performance (b) with the QCIF picture format.

processors to get a rough approximate of the performance. For more accurate estimation, the effects of the load balancing (Figure 7b) and the communication channel were analyzed comparing the performance results of co-simulation and cycle-accurate instruction set simulation of the local memory approach. As a result, the following estimation function was used to model the cycle count behavior:

$$C_e = C_s * \frac{S_{\max}}{S_{\text{total}}} + C_{\text{bus}} * D_{\text{bus}}$$

$C_e$	Estimated cycle count
$C_s$	Cycle count from sequential simulation
$S_{\max}$	Max. number of slices per slave
$S_{\text{total}}$	Total number of slices
$C_{\text{bus}}$	Bus transfer cycles
$D_{\text{bus}}$	Delay factor for a transfer

Compared to the cycle-accurate hardware software co-simulation (see Table 7), the estimated performance values for one to nine slave processors deviate 5 percent on average. The estimation function assumes bus-based communication architecture but is otherwise platform-independent. Even the sequential simulations with the instruction set simulator can be omitted if an estimation method, described in [32], were applied.

## 6.2. Local Data Memory Approach

Figure 7a depicts the performance of the video encoding with respect to the number of slave processors at 100 MHz processor frequency. The non-linear shape of the curve is due to the parallelization strategy. As mentioned, a frame is divided into slices that are then equally distributed to encoding processors. In some cases, an equal amount of slices cannot be distributed

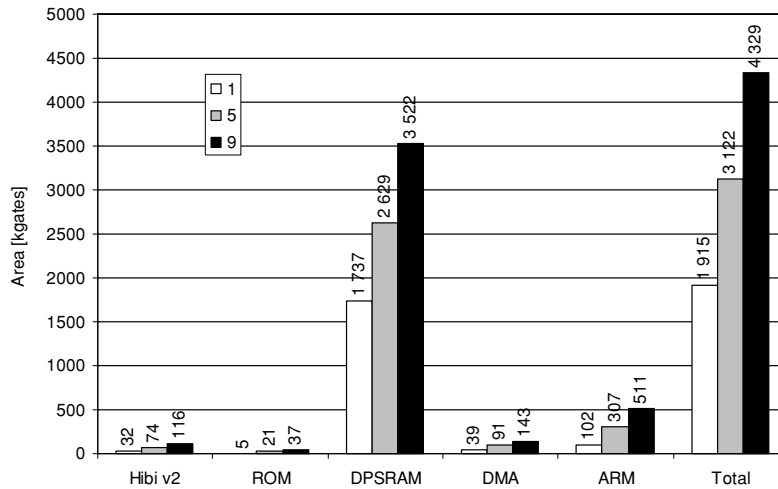


Figure 8. Area of the video encoder for configuration of 1, 5, and 9 slaves with the QCIF picture format.

for all the processors. The processor with the most slices is then the bottleneck of the system.

In Figure 7b, the bars depict the maximum number of slices per processor and the triangles connected by a line represent inverse values. The effect of the number of slices per processor on the encoding performance can be observed from the similarity of the curves. In the cases of a well balanced load (1, 3, and 9 slaves), the performance scales linearly with the number of slaves, with the scaling factor of 0.9. Scaling factor less than 1 is due to the communication overhead caused by inter-slave data exchange as described in Section 2. The performance can be scaled conveniently with the number of encoding processors without modifying the algorithm at all. For performance optimization, the reasonable number of processors depends on the load balancing and the overhead caused by the extra communication.

In the same way as the performance with well balanced load, the area of the encoder scales linearly with the number of slave processors (Figure 8). The total

area can mainly be attributed to memories, particularly data memory (DPSRAM). The relative size of the data memory is smaller with more slave processors being approximately 91 percent with one slave and 81 percent with nine slaves. This is because the number of slices per one processor decreases at the same time. The growth in data memory size is due to the temporary data and the reconstruction rows needed by each slave processor.

In Table 9, the overall data transfers and the bus utilization during the encoding of one frame are tabulated. Bus utilization is a portion of the bus cycles from the total encoding cycle count. The bus and processor frequencies are 100 MHz and a 32-bit HIBI bus was utilized. A noteworthy detail is that the bus utilization is very low even for high number of processors which indicates that the bus is oversized. However, the bus utilization is not a very descriptive quantity of the performance. In fact, the latency is the most limiting factor of the bus performance when dealing with memory accesses through the bus.

Table 9. Communication vs. computation in the case of local data memory approach.

	Number of slave processors								
	1	2	3	4	5	6	7	8	9
Total bus cycles	21280	23638	25996	28353	30711	33069	35427	37784	40142
Computation cycles	15770181	8793859	5305245	5312937	3567775	3572212	3581064	3587690	1837584
Cycles per frame	15791461	8817497	5331241	5341290	3598486	3605281	3616491	3625474	1877727
Bus utilization	0.1 %	0.3 %	0.5 %	0.5 %	0.9 %	0.9 %	1.0 %	1.0 %	2.1 %

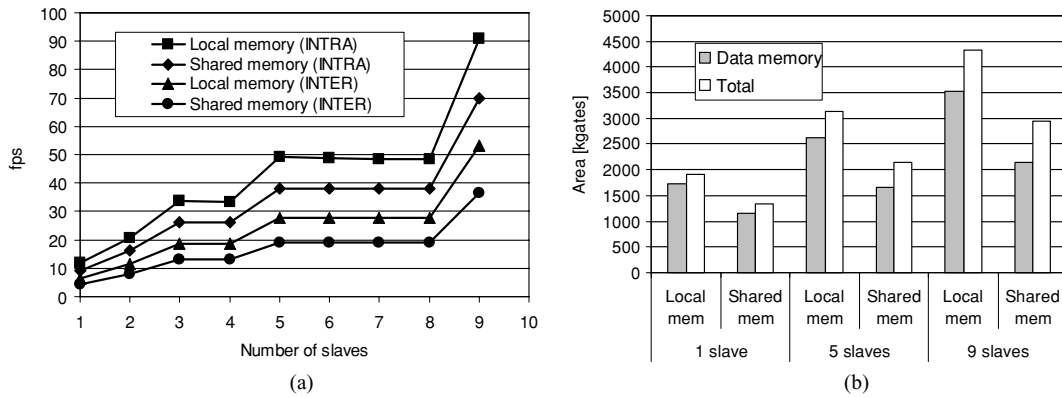


Figure 9. Effect of selected memory approach on performance (a) and area (b).

### 6.3. Shared Memory Approach

Figure 9a represents the effect of the memory approach on the encoding performance. The cycle count obtained from AXD includes already the delay for accessing the shared memory but does not take into account the delay caused by other simultaneous accesses. Since only one access is served at a time, the estimated performance for the shared memory is slightly too optimistic. For the shared memory approach, the performance is about 30 percent less regardless of the number of slaves.

In Figure 9b, the approaches are compared with the data memory and overall area estimations. The area difference of the approaches increases with the number slaves. With nine slaves, the shared memory version is 32 percent smaller than the local memory based encoder. Judging from the area consumption, the shared memory approach is preferable especially with large number of slave processors.

The communication and computation balance is examined in Table 10. Although the data transfers increased significantly compared to the local memory approach (see Table 4 and Table 5), the bus is still very lightly utilized. The same video encoding performance

could be achieved also with smaller bus performance, for instance, by decreasing the bus width. In the next Section, the optimization of communication and computation parameters are analyzed in more detail.

In Table 11, the performance, the area, and the power of the proposed video encoder architecture are compared to other reported implementations. Four different implementations of the presented encoder architecture are tabulated. It is hard to carry out the comparison of implementations since there are no commonly accepted ways to present figures for area and frame rate. For instance, usually authors do not explicitly report which components are included in the gate count figures. According to the table, it can be concluded that the presented video encoder architecture, especially with the local memory approach, provides a comparable performance and area with other video encoders despite the fact the low-level optimization of software and hardware was not carried out. The scalability of the referred encoders is, however, limited mostly to the functionality within a control processor. Only in [14] it is possible to scale the architecture for better performance by adding modular processors to the system. All of the referred encoders utilize hardware accelerators to

Table 10. Communication vs. computation in the case of shared data memory approach.

	Number of slave processors								
	1	2	3	4	5	6	7	8	9
Total bus cycles	74080	74091	74102	74113	74124	74136	74147	74158	74169
Computation cycles	22636747	12647242	7650430	7650986	5150644	5150865	5151306	5151494	2650403
Cycles per frame	22710827	12721333	7724532	7725099	5224768	5225001	5225452	5225652	2724572
Bus utilization	0.3 %	0.6 %	1.0 %	1.0 %	1.4 %	1.4 %	1.4 %	1.4 %	2.7 %

Table 11. Comparison of video encoder implementations.

Architecture	Application	Gate count <sup>1</sup> [gates]	Power [mW/MHz]	QCIF frame rate [fps]	Clock frequency [MHz]	Embedded RAM [Kbytes]
HIBI(3 slaves) <sup>5</sup>	H.263	322400	8	19	100	266
HIBI(9 slaves) <sup>5</sup>	H.263	770500	19	53	100	430
HIBI(3 slaves) <sup>6</sup>	H.263	322400	8	13	100	172
HIBI(9 slaves) <sup>6</sup>	H.263	770500	19	37	100	261
[15]	H.263/G.723	142300 <sup>2</sup>	—	15	54	13
[16]	MPEG-4 (SP@L1)	—	4	15	60	2048
[14]	H.261/H.263	325000 <sup>3</sup>	11	10	66	15
[13]	H.324	420898	15	10 <sup>4</sup>	15	—
[12]	H.263	900000	—	15	40	17

<sup>1</sup> Memories not included

<sup>2</sup> Does not include FIFO buffers (22000 bits)

<sup>3</sup> 1300000 transistors reported, here assuming 4 transistors per gate

<sup>4</sup> Not clear whether the 10 fps is achieved with 15 MHz

<sup>5</sup> HIBI video encoder with the local memory approach

<sup>6</sup> HIBI video encoder with the shared memory approach

implement the entire encoder [13] or for the computation intensive operations [12, 14–16].

#### 6.4. Architecture Optimization

Previous sections analyzed the architectural choices to implement the given functionality. After defining architectural outlines, there are a lot of parameters to be determined such as the number of processors, processor frequencies, bus frequency, and bus width. The goal is to optimize the parameters in such a way that area, power, and performance requirements can still be met. Table 12 presents examples of parameter combinations with a QCIF target frame rate of 20 fps. Area and power requirements are not considered in these examples. The variable parameters are bus frequency, processor frequency, data bus width, and the number of slave processors. Table presents four cases in which one or two of these parameters are kept constant while the others are optimized in such a way that the required performance can be achieved.

In the first case, both bus and processor frequencies were kept constant and different bus widths (from 1 to 32) were evaluated. Originally the bus and processor frequencies were 100 MHz and a 32-bit bus was utilized. Since the bus utilization with those parameters was low (Table 9), either the bus width or the bus frequency can be decreased significantly. With these fixed

parameters (100 MHz frequencies), the target frame rate can be achieved with five slaves. With more than four slaves, the target frame rate is achieved with very low theoretical bus width (1 bit). However, harnessing more processors would not be reasonable if the total area of the video encoder is considered.

Also in the second and third case, the target frame rate is achieved with five slave processors. As all the first three cases show, neither the bus frequency nor width influence the frame rate substantially. This is due to the low utilization of the bus. Moreover, the low latency of HIBI ensures that the bus will not be the bottleneck of the system. With less than five processors, the required performance cannot be achieved with any combination of bus parameters if the processor frequency is smaller than 100 MHz as shown in Table 12.

The processor frequency has a greater effect on the performance as can be seen in the fourth case of parameter combinations (bus width fixed to 16 bits). Now, the target frame rate of 20 fps can be obtained with only one slave processor if its frequency is set to 320 MHz. For technological reasons and greater power consumption, the high frequency is often not the best choice. Instead, by instantiating another slave processor and decreasing the processor frequency to 180 MHz and bus frequency to 50 MHz, the same frame rate is achieved.

Figure 10 illustrates the effect of the processor frequency and the number of slaves on the frame rate. The

Table 12. Examples of parameter combinations with QCIF target frame rate of 20 fps. The fixed values are in bold face.

Number of slaves	1	2	3	4	5	6	7	8	9
<b>Case 1:</b>									
Bus width [bit]	32	32	32	32	1	1	1	1	1
Bus freq [MHz]	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>
Processor freq [MHz]	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>
Bus utilisation	0.1 %	0.3 %	0.5 %	0.5 %	21.6 %	22.8 %	24.0 %	25.2 %	41.1 %
Frame rate [fps]	6.3	11.3	18.8	18.7	22.0	21.6	21.2	20.8	32.0
<b>Case 2:</b>									
Bus width [bit]	<b>16</b>	<b>16</b>	<b>16</b>	<b>16</b>	<b>16</b>	<b>16</b>	<b>16</b>	<b>16</b>	<b>16</b>
Bus freq [MHz]	200	200	200	200	5	5	6	6	3
Processor freq [MHz]	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>
Bus utilisation	0.1 %	0.3 %	0.5 %	0.5 %	25.6 %	27.0 %	24.8 %	26.0 %	59.3 %
Frame rate [fps]	6.3	11.3	18.8	18.7	20.9	20.4	21.0	20.6	22.2
<b>Case 3:</b>									
Bus width [bit]	8	8	8	8	8	8	8	4	2
Bus freq [MHz]	100	100	100	100	10	10	10	18	22
Processor freq [MHz]	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>
Bus utilisation	0.5 %	1.1 %	1.9 %	2.1 %	25.6 %	27.0 %	28.3 %	31.9 %	61.4 %
Frame rate [fps]	6.3	11.3	18.5	18.4	20.9	20.4	20.0	19.0	21.0
<b>Case 4:</b>									
Bus width [bit]	<b>16</b>	<b>16</b>	<b>16</b>	<b>16</b>	<b>16</b>	<b>16</b>	<b>16</b>	<b>16</b>	<b>16</b>
Bus freq [MHz]	100	50	50	30	25	20	20	20	10
Processor freq [MHz]	320	180	120	115	80	80	80	80	48
Bus utilisation	0.9 %	1.9 %	2.3 %	3.9 %	5.2 %	6.9 %	7.3 %	7.8 %	17.3 %
Frame rate [fps]	20.1	20.1	22.1	20.8	21.3	20.9	20.7	20.6	21.6

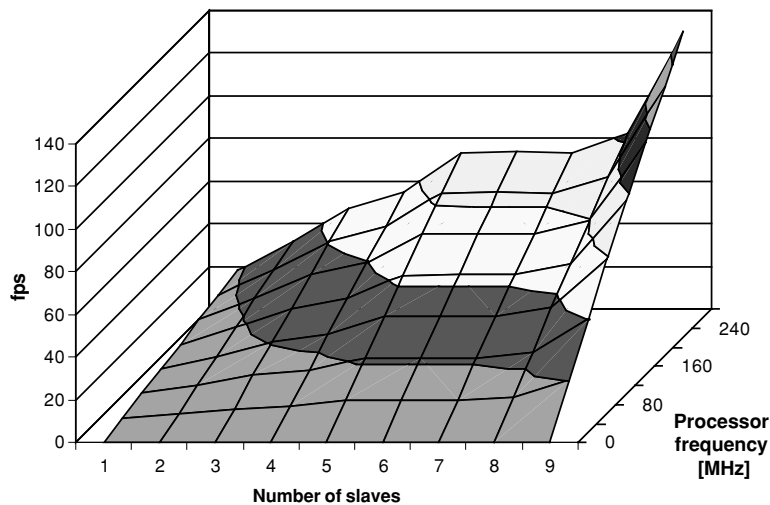


Figure 10. Effect of the processor frequency and the number of slaves on the QCIF frame rate.

frame rate increases linearly with processor frequency. With a small number of slaves the scaling factor is 1 but decreases when the number of slaves increases. This is due to the increased communication overhead with a high number of slaves.

As this analysis shows, the architectural space to explore is vast already with a few optimizable parameters. In addition to data width, the depths of the FIFO buffers and arbitration parameters can be optimized in the HIBI scheme. Efficient optimization even for a small subset of possible architectures, however, requires an automated way to explore the parameter combinations as explained in [33]. The presented video encoder is able to perform 20 fps QCIF video encoding with several different architecture configurations. With higher frame resolution (CIF, 4CIF etc.), the performance requirements for processors and network would be different. The performance of the presented architecture can be scaled up by adding more slave processor (max 18) as described in Table 2, or utilizing more powerful processors. HIBI network proved to have capacity for more active communication as its utilization was low in most of the previous cases.

## 7. Conclusions

This paper presented a scalable architecture for video encoding. The main focus in the design was on architectural strategies and communication design. The architecture is based on homogeneous SPMD architecture comprising identical ARM7TDMI-based processor modules. Area estimations showed that in video encoding the increase of hardware due to the programmability and SPMD architecture compared to the custom hardware implementation is negligible. This is because most of the area is due to the storage needed for the data stream.

Presented architecture can be scaled easily by duplicating the slave processor units. For QCIF-sized frames and with non-optimized software, the encoding performance ranges from 6 to 53 fps with 1 to 9 slave processors, respectively. For larger frames than QCIF, more powerful processing units or thoroughly optimized algorithms are required. Homogeneous architecture and identical program for the slave processors facilitates both the hardware and the software development.

The selection between the shared and local memory architecture is a trade-off between area and performance. Using shared memory minimizes the hardware costs particularly with several slave processors

but decreases the performance. With nine slaves the performance of the shared memory version is about 30 percent worse and the chip 32 percent smaller than the local memory implementation.

The utilized communication architecture, HIBI, provided essential properties for parallelization with respect to scalability and performance. In addition, HIBI could accommodate further extensions of the video codec such as larger frames and advanced coding modes. Currently, the system is being extended with processors and accelerators that implement the MAC protocol and radio interface for wireless communication.

Analyses illustrated that the design space of a highly parameterizable architecture is so large that an automated exploration method is required for full utilization. For HIBI, a communication-centric design method has been developed [34]. It includes the methods and tools for application profiling, architecture exploration, and component-level optimization and synthesis.

The success of parallelization depends on several equally important factors. Starting from the application description, the parallel structures with required level of granularity should be explicitly described. At instruction-level, there are many powerful tools for parsing and compiling the application description for a parallel execution platform. The same methods for determining control and data dependencies can be used in task-level parallelization but automating the process requires well defined interfaces for task descriptions.

Selecting a parallelization strategy, such as choosing between task or data parallelism, depends highly on the nature of application and can have a major effect on the implementation. Whether the application description can be implemented with selected parallelization strategy or not, depends on the support from the computation and communication platform. Using general purpose processors gives flexibility but lags behind dedicated processing units in performance. Parallelization sets lots of challenges in communication architecture. It should have support for heterogeneous processing units, low-latency communication, sophisticated communication mechanisms and protocols, and scalability.

## References

1. ITU-T Recommendation H.264, Advanced Video Coding for Generic Audiovisual Services, May 2003.

2. P. Pirsch and H.-J. Stolberg, "VLSI Implementations of Image and Video Multimedia Processing Systems," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 7, 1998, pp. 878–891.
3. A. Dasu and S. Panchanathan, "A Survey of Media Processing Approaches," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 8, 2002, pp. 633–645.
4. O. Lehtoranta, T. Hämäläinen, V. Lappalainen, and J. Mustonen, "Parallel Implementation of Video Encoder on Quad DSP System," *Microprocessors and Microsystems*, vol. 26, no. 1, 2002, pp. 1–15.
5. E. Iwata, K. Seno, M. Aikawa, M. Ohki, H. Yoshikawa, Y. Fukuzawa, H. Hanaki, K. Nishibori, Y. Kondo, H. Takamuki, T. Nagai, K. Hasegawa, H. Okuda, I. Kumata, M. Soneda, S. Iwase, and T. Yamazaki, "A 2.2 GOPS Video DSP with 2-RISC MIMD, 6-PE SIMD Architecture for Real-Time MPEG2 Video Coding/Decoding," *Digest of Technical Papers of IEEE International Solid-State Circuits Conference*, 1997, pp. 258–259, 469.
6. S. Ishiwata, T. Yamakage, Y. Tsuboi, T. Shimazawa, T. Kitazawa, S. Michinaka, K. Yahagi, H. Takeda, A. Oue, T. Kodama, N. Matsumoto, T. Kamei, M. Saito, T. Miyamori, G. Ootomo, and M. Matsui, "A Single-Chip MPEG-2 Codec Based on Customizable Media Embedded Processor," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 3, 2003, pp. 530–540.
7. S. Ramachandran and S. Srinivasan, "A Fast, FPGA-based MPEG-2 Video Encoder with a Novel Automatic Quality Control Scheme," *Microprocessors and Microsystems*, vol. 25, no. 9-10, 2002, pp. 449–457.
8. M. Berekovic, S. Flagel, H.-J. Stolberg, L. Friebe, S. Moch, M.B. Kulaczewski, and P. Pirsch, "HiBRID-SoC: a Multi-Core Architecture for Image and Video Applications," *Proceedings of the International Conference on Image Processing*, vol. 3, 2003, pp. 101–104.
9. H.-J. Stolberg, M. Berekovic, P. Pirsch, H. Runge, H. Moller, and J. Kneip, "The M-PIRE MPEG-4 Codec DSP and Its Macroblock Engine," *Proceedings of the International Symposium on Circuits and Systems*, vol. 2, 2000, pp. 192–195.
10. M. Harrand, J. Sanches, A. Bellon, J. Bulone, A. Tournier, O. Deygas, J.-C. Herluison, D. Doise, and E. Berrebi, "A Single-Chip CIF 30-Hz, H261, H263, and H263+ Video Encoder/Decoder with Embedded Display Controller," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 11, 1999, pp. 1627–1633.
11. N. Minegishi, N. Motoyama, M. Takagi, F. Ogawa, K. Shibata, N. Goda, K. Akiyoshi, T. Kamemaru, and K. Asano, "A Single Chip H.32X Multimedia Communication Processor with CIF 30f/s MPEG4/H.26X Bi-directional Codec," *Proceedings of the European Solid-State Circuits Conference*, 2001.
12. S.H. Lee, M. Kim, and K.-B. Kim, "Modular and Efficient Architecture for H.263 Video Codec VLSI," *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 5, 2002, pp. V-125–V-128.
13. T. Onoye, G. Fujita, H. Okuhata, M.H. Miki, and I. Shirakawa, "Low-Power Implementation of H.324 Audiovisual Codec Dedicated to Mobile Computing," *Proceedings of the Asia and South Pacific Design Automation Conference*, 1998, pp. 589–594.
14. J. Hilgenstock, K. Herrmann, J. Otterstedt, D. Niggemeyer, and P. Pirsch, "A Video Signal Processor for MIMD Multiprocessing," *Proceedings of the Design Automation Conference*, 1998, pp. 50–55.
15. S. Park, S. Kim, K. Byeon, J. Cha, and H. Cho, "An Area Efficient Video/Audio Codec for Portable Multimedia Application," *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 1, 2000, pp. 595–598.
16. M. Takahashi, T. Nishikawa, H. Arakida, N. Machida, H. Yamamoto, T. Fujiyoshi, Y. Matsumoto, O. Yamagishi, T. Samata, A. Asano, T. Terazawa, K. Ohmori, J. Shirakura, Y. Watanabe, H. Nakamura, S. Minami, and T. Furuyama, "A Scalable MPEG-4 Video Codec Architecture for IMT-2000 Multimedia Applications," *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 2, 2000, pp. 188–191.
17. J. Chaoui, K. Cyr, S. de Gregorio, J.-P. Giacalone, J. Webb, J., and Y. Masse, "Enabling Video Processing in Wireless Terminals with a New Open Multimedia Application Platform," *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, 2001, pp. 1009–1012.
18. E. Salminen, V. Lahtinen, T. Kangas, J. Riihimäki, K. Kuusilinna, and T. Hämäläinen, "HIBI Communication Network for Systems-on-Chip," to appear in *Journal of VLSI Signal Processing-Systems for Signal, Image, and Video Technology*, Springer.
19. ITU-T Recommendation H.263, Video Coding for Low Bitrate Communication, January 1998.
20. ARM Limited, *ARM Architecture Reference Manual*, ARM DDI 0100E, 2000.
21. VSI Alliance, *Virtual Component Interface Specification 2*, Version 1.0, 1999.
22. Open Core Protocol International Partnership, <http://www.ocpip.org>.
23. E. Salminen, T. Hämäläinen, T. Kangas, K. Kuusilinna, and J. Saarinen, "Interfacing Multiple Processors in a System-on-Chip Video Codec," *Proceedings of the IEEE International Symposium on Circuits and Systems Conference*, vol. 4, 2001, pp. 478–481.
24. W.-T. Shiue and C. Chakrabarti, "Memory Design and Exploration for Low Power, Embedded Systems," *Journal of VLSI Signal Processing*, Kluwer Academic Publishers, vol. 29, no. 3, pp. 167–178, 2001.
25. S. Dutta, W. Wolf, and A. Wolfe, "A Methodology to Evaluate Memory Architecture Design Tradeoffs for Video Signal Processors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 1, 1998, pp. 36–53.
26. ARM Limited, *AXD and armsd Debuggers Guide*, ARM DUI 0066D, ARM Developer Suite, version 1.2.
27. Mentor Graphics, *Seamless CVE User's and Reference Manual*, Software Version 5.0, 2003.
28. J. Riihimäki, V. Helminen, K. Kuusilinna, and T. Hämäläinen, "Parallelizing SoC Simulations over a Network of Computers," *Proceedings of the Euromicro symposium on Digital System Design*, 2003, pp. 447–450.
29. ARM Limited, *ARM7 Thumb Family Flyer*, DOI 0035-3/02.02, 2002.
30. Sematech, *International Technology Roadmap for Semiconductors: System Drivers*, Report, 2003.
31. H.J. Mattausch, "Hierarchical N-Port Memory Architecture based on 1-Port Memory Cells," *Proceedings of the 23rd*

*European Solid-State Circuits Conference*, 1997, pp. 348–351.

32. H.-J. Stolberg, M. Berekovic, and P. Pirsch, “A Platform-Independent Methodology for Performance Estimation of Streaming Media Applications,” *Proceedings of the IEEE International Conference on Multimedia and EXPO*, 2002.
33. J. Riihimäki, E. Salminen, K. Kuusilinna, and T. Hämäläinen, “Parameter Optimization Tool for Enhancing On-chip Network Performance,” *Proceedings of the IEEE International Symposium of Circuits and Systems*, 2002, pp. 61–64.
34. T. Kangas, P. Kukkala, H. Orsila, E. Salminen, M. Hännikäinen, T. Hämäläinen, J. Riihimäki, and K. Kuusilinna, “UML-based Multi-Processor SoC Design Framework,” to appear in *Transactions on Embedded Computing Systems*, ACM, 2006.



**Tero Kangas**, MSc '01, Tampere University of Technology (TUT). Since 1999 he has been working as a research scientist in the Institute of Digital and Computer Systems (DCS) at TUT. Currently he is working towards his PhD degree and his main research topics are system architectures and SoC design methodologies in multimedia applications.  
tero.kangas@tut.fi



**Kimmo Kuusilinna**, PhD '01, TUT. His main research interests include system-level design and verification, interconnection networks, and parallel memories. Currently he is working as a senior research engineer at the Nokia Research Center.



**Timo D. Hämäläinen**, MSc '93, PhD '97, TUT. He acted as a senior research scientist and project manager at TUT in 1997–2001. He was nominated to full professor at TUT/Institute of Digital and Computer Systems in 2001. He heads the DACI research group that focuses on three main lines: wireless local area networking and wireless sensor networks, high-performance DSP/HW based video encoding, and interconnection networks with design flow tools for heterogeneous SoC platforms.