Y-Chart Based System Level Performance Analysis: An M-JPEG Case Study

Todor Stefanov[1] Paul Lieverse[2] Ed Deprettere[1] Pieter van der Wolf[3]

[1] Leiden Institute of Advanced Computer Science, Leiden, The Netherlands
[2] Delft University of Technology, Delft, The Netherlands
[3] Philips Research Laboratories, Eindhoven, The Netherlands

Abstract— **In the Artemis project an architecture workbench is being developed. One of the inputs for defining this workbench is the SPADE methodology. SPADE (System level Performance Analysis and Design space Exploration) follows the Y-chart approach; application and architecture are modeled separately, and the mapping of the application onto the architecture is an explicit design step. As an advantage we can easily modify the application, architecture, or mapping, resulting in a quick turnaround time to explore alternative system implementations.**

In this paper we introduce and evaluate SPADE through an illustrative case study. In this case study we start from a modified M-JPEG application and map this application onto a shared memory multi-processor architecture. The example system is also used in the Artemis project as a driver and case study for the design and evaluation of the workbench. We define the application as a Kahn Process Network. The architecture and mapping are specified using Spade architecture and mapping languages. We present results of simulations for alternative architecture instances and mappings.

Keywords—**system level design, M-JPEG, design space exploration, signal processing, application modeling, architecture modeling**

I. INTRODUCTION

Modern signal processing systems are increasingly becoming multi-functional systems that also have to support multiple standards. For example, digital televisions, set-top boxes, and mobile devices need to offer a variety of functions and must support different standards for transmission and coding of digital contents. In order to provide the required flexibility such systems need to be implemented using programmable components. On the other hand, in order to meet performance requirements and cost constraints, parts of these systems are still required to be implemented in dedicated hardware blocks.

Designing such *heterogeneous* systems, composed programmable and dedicated components and various kinds of communication structures, is not an easy task. Each of the components still can be designed by using current methodologies and tools, such as writing RT level HDL descriptions or using high level synthesis tools. However, these methodologies are no longer sufficient at the system level. The questions a system designer has to deal with are, e.g, which parts of an application are implemented in software and which in hardware, which hardware components are selected, what kind of communication structure is going to be used, what are the critical parts of the system. To find an answer to these questions designers need to be supported by a different kind of design technology than which is used to design the components. Today, several methodologies and tools are being developed to support the designer at the system level, e.g., Polis [1], VCC [2], SystemC [3], and eArchitect [4].

In this paper we present and evaluate a methodology, named SPADE (System level Performance Analysis and Design space Exploration) [5], for *architecture exploration* of heterogeneous signal processing systems. This exploration starts from executable specifications of a set of representative target applications. The result is the definition of a heterogeneous hardware architecture capable of executing these applications within predefined constraints with respect to cost, real-time response, etc.

We evaluate SPADE in the context of a case study in which we are mapping a modified M-JPEG application onto a shared memory multi-processor architecture. This case study is part of the *Artemis* project, and is called *Startemis*. The Artemis project [6] (ARchitectures and meThods for Embedded MedIa Systems) is a research project that aims at reducing the development time of embedded media systems with a high degree of programmability. One of the research challenges that is addressed in Artemis is the development an architecture simulation workbench which provides methods, tools, and libraries for the efficient exploration of heterogeneous embedded system architectures. SPADE will be one of the methodologies used in this workbench; the Sesame (Simulation of Embedded System Architecture for Multilevel Exploration) framework [7] is another methodology being used. The case study will be used as a driver for the development of the Artemis workbench. Therefore, the application used in the case study should be realistic, yet not too complex, and it should be possible to refine it later on. Also, it should be extensible, in order to add complexity when needed. As we are building a workbench for heterogeneous systems, the architecture onto which the application is mapped should be heterogeneous in the components and protocols used. The architecture should also be sufficiently complex in order to allow for refinement of the components. The Startemis case study meets these requirements.

In the next section first a description of the SPADE methodology and tool is given. The Startemis case study is described in Section III, including the modeling of the case study using SPADE. The exploration we did, and the results and conclusions obtained from the exploration are presented in Section IV. Finally, some overall conclusions are given.

II. SPADE

SPADE is a methodology, implemented in a tool, that enables modeling and exploration of heterogeneous signal processing systems. In this section we first describe the basic concepts on which SPADE is based. Then we go into some detail of the application modeling, architecture modeling, and mapping. For more details on SPADE we refer the reader to [5].

A. Concepts

The main concept on which SPADE is based is the *Y-chart* [8], as depicted in Figure 1. The Y-chart represents a general scheme for the design of heterogeneous systems. In the Y-chart a clear distinction is made between *applications* and *architectures*, which are related via an explicit *mapping* step. This concept can be applied at various levels of abstraction; here we focus on the system level, but the Y-chart can be applied at more detailed levels, such as RT level, as well. The Y-chart approach permits multiple target applications to be mapped one after another onto candidate architectures in order to evaluate their performance. The resulting performance numbers may inspire an

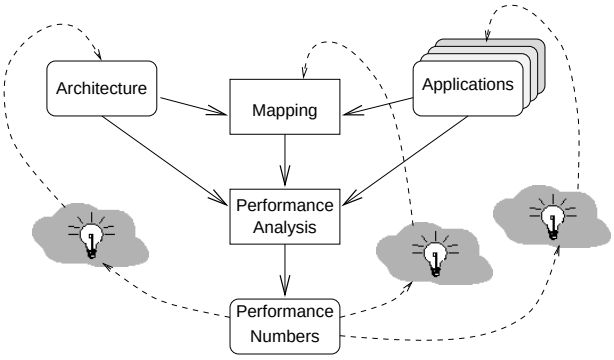

Fig. 1. The Y-chart

architecture designer to improve the architecture. He may also decide to restructure the application(s) or to modify the mapping of the application(s). These options are indicated in Figure 1 by light bulbs.

The distinction between applications and architectures can be rephrased as the distinction between *workload* and *resources*. An application imposes a workload onto resources which are defined by an architecture. A workload consists of both *computation workload* and *communication workload*; resources can be *processing resources*, *communication resources*, and *memory resources*. Computation workload requires processing resources; communication workload requires a combination of communication resources and memory resources. The architecture design process is concerned with the specification of resources that can best handle the workloads imposed by target applications.

The explicit mapping defines how an application is mapped onto an architecture, in other words, how the workload of an application is mapped onto the resources of an architecture. Within SPADE, we are using simulation as the means for performance evaluation. So, we need a way to capture this mapping such that we can simulate a system, consisting of both an application and an architecture. For this purpose we extend a technique called *trace-driven simulation*. This is a simulation technique that has been applied extensively for memory system simulation in the field of general-purpose processor design [9]. The workload of an application is captured in one or more *traces*. A trace contains symbols, called *trace entries*, that represent the computation and communication operations performed by an application; data dependent behavior in the application is thus captured by these traces. The resources in an architecture accept these trace entries as the workload to be executed. The traces *drive* computation and communication activities in the architecture.

B. Application modeling

One of the objectives of application modeling in SPADE is to expose task level parallelism and to make communication explicit. We have chosen the Kahn Process Networks [10] model of computation for application modeling. In the Kahn model, parallel *processes* communicate via unbounded FIFO *channels*. There are two main reasons for choosing this Model of Computation. First, the model fits nicely with signal processing applications as it can model *stream processing* and as it guarantees that no data is lost. Second, the execution of a Kahn Process Network is deterministic, meaning that for a given input always the same output is produced and the same workload is generated, irrespective of the execution schedule.

Each process in the network produces a trace to capture the workload of that process. The communication workload is also captured in these process related traces.

Spade uses the YAPI Application Programmers Interface [11] for application modeling. The following three functions are provided[1].

- A *read* function. This function is used to read data from a channel via a process port. Furthermore, the function generates a *trace entry* in the trace of the process by which it is invoked, reporting on the execution of a read operation at the application level.
- A *write* function. This function is used to write data to a channel via a process port. It also generates a trace entry, reporting on the execution of a write operation.
- An *execute* function. This function performs no data processing, but only generates a trace entry, reporting on processing activities at the application level. The execute function takes a *symbolic instruction* as an argument in order to distinguish between different processing activities. For example, such an instruction may correspond to an IDCT operation on an eight by eight matrix.

The trace entries generated by the read and write functions represent the *communication workload* of a process. The trace entries generated by the execute function represent the *computation workload* of a process. The trace entries can be used either to drive the architecture simulation, or, when executing the application *stand-alone*, to analyze the computation and communication workload of an application.

C. Architecture modeling

In order to efficiently explore different architectures, it is required that architecture models can be easily constructed. In SPADE, functional behavior is described at the application level. If this behavior is data dependent, the traces, which drive the operation of the architecture, also depend on the input data. Therefore, we can use architecture models that do not need to model the functional behavior, while maintaining functional correctness. Such architecture models can be constructed from *generic building blocks*. As the building blocks are generic, we can provide a library of such blocks. The generic building blocks need

[1] Note that the YAPI `select` function is not supported by SPADE.

to model the different types of resources in an architecture, such as processing resources, communication resources, and memory resources. Defining an architecture then becomes as easy as instantiating building blocks from a library and interconnecting them.

The processing resources in the architecture model take the traces generated by the application as an input. We have taken a modular approach to allow the construction of a great variety of processing resources from a small number of basic building blocks. A processing resource is built from the following two types of blocks.

- A *trace driven execution unit (TDEU)* which interprets trace entries. The entries are interpreted in the order in which they are put in the trace, thereby retaining the order of execution of the application process. A TDEU has a configurable number of I/O ports. Communication via these I/O ports is based on a generic protocol.
- A number of *interfaces* which connect the I/O ports of a TDEU to a specific communication resource. An interface translates the generic protocol into a communication resource specific protocol, and may also include buffers to model input/output buffering of processing resources. Currently, we have interfaces for point-to-point communication via a bus, and for communication via a bus and shared memory, both buffered and unbuffered. No interfaces are needed for communication via a FIFO or an unbuffered direct link; these communication blocks can be directly connected to the I/O ports of a TDEU.

The current library contains the TDEU and interface blocks described above, a generic bus block, including a first-come-first-served arbiter, a FIFO block, an unbuffered direct link block, and a generic memory block. All blocks are parameterized. For each instantiated TDEU a list of *symbolic instructions* and their *latencies* has to be given. This list specifies which instructions from the traces can be executed by the processing resource and how many cycles each instruction takes when executed on this processing resource. These latencies can be obtained either from a lower level model of a processing resource, from estimation tools, or they can be estimated by an experienced designer. For instances of the FIFO and interface blocks, buffer sizes can be given. For a bus instance, the bus width, setup delay, and transfer delay can be specified.

An architecture is specified by means of a textual description using some dedicated architecture description language. In this description first the processors, buses, and FIFOs are defined. Here the user does not need to define the exact interfaces; these are inserted automatically when the architecture model is constructed. Only the parameters, such as, latencies, buffer sizes, and bus width need to be specified. After the components are defined, the structure of the architecture is defined, by describing for each FIFO and each bus which processor ports are connected to them. An example of such textual architecture description is given in Figure 4.

D. Mapping

Once both an application model and an architecture model have been defined, mapping can be performed. This means that the workload of the application has to be assigned to resources in the architecture as follows.

- Each process is mapped onto a TDEU. This mapping can be many-to-one, in which case the trace entries of the processes need to be scheduled by the TDEU.
- Each process port is mapped one-to-one onto an I/O port. This mapping also implicitly maps the channels onto a combination of communication resources and memory resources.

If it appears that the functionality of a single process needs to be distributed over more than one processing resource, then the designer first has to rewrite the application such that this process is partitioned into two or more processes. Then these processes can be mapped onto separate TDEUs.

The mapping is also specified by means of a textual description using a dedicated mapping description language. First, for each process in the application it is specified onto which processor in the architecture it is mapped. At the same time, the mapping of process ports onto processor ports is specified. Then some characteristics of the channels are specified, such as the physical size of tokens that are communicated, and whether communication is done via shared memory. Finally, for each processor a scheduler can be defined, including some parameters. An example of such textual mapping description is given in Figure 6.

E. Simulation

Performance analysis in SPADE is done using simulation. We use trace driven simulation to co-simulate an application model with an architecture model. The simulation of the application model is based on the Pamela [12] multi-threading environment, where each Kahn process is executed in a separate thread. The simulation of the architecture model is currently based on TSS (Tool for System Simulation), which is a Philips in-house architecture modeling and simulation framework [13].

F. Performance Metrics

In order to evaluate a system, the SPADE library blocks from which an architecture is built collect several numbers during simulation. From these numbers *performance metrics* can be calculated. These metrics give an indication of the performance of the system, such as, throughput, frame rate, overall latency, and bus utilization. For example, for a video processing system we can collect the times at which a new frame is output; from those times we can calculate the frame rate of the system.

The numbers that are currently collected in the library blocks are the following. For each TDEU we keep track of the number of cycles it was busy with computations, the number of cycles it was doing I/O, split out into reads and writes, the number of cycles it was waiting either for data or for room, and the number of cycles it had nothing to do at all, i.e., was idle. In addition, for each input and output port we get the number of reads or writes that were performed, plus the number of cycles no room or data was available. For each bus we get the number of cycles it was in use and the amount of data transported via this bus. Also, for each interface connected to the bus, we get a histogram of the delays between issuing a bus request and being granted access to the bus.

III. STARTEMIS CASE STUDY

A. Startemis Application

The application we have chosen for the Startemis case study is an M-JPEG encoder. A traditional M-JPEG encoder typically compresses a sequence of video frames, applying a JPEG based compression technique [14][15] to each frame in the video sequence. M-JPEG is used for motion pictures compression like MPEG [16] but without *interframe predictive coding*. This means that M-JPEG does not perform *motion estimation* and *compensation*. We have modified a traditional M-JPEG encoder in order to add data dependent behavior to it. The modified encoder, named M-JPEG*, has three main differences from traditional M-JPEG:

- M-JPEG* supports only lossy encoding, whereas M-JPEG typically supports both lossy encoding and lossless encoding.
- M-JPEG* can operate on video data in both 4:2:2 YUV and RGB formats on a per-frame basis, whereas traditional M-JPEG uses only YUV format.
- M-JPEG* can process the incoming video frames with different quantization and Huffman tables. These tables can differ per video frame, depending on the output bit-rate and the accumulated statistics from previous video frames. Such dynamic change of the tables is not performed by a traditional M-JPEG encoder.

The possibility of M-JPEG* to handle two video data formats and different tables on a per-frame basis means that M-JPEG* is data dependent.

B. M-JPEG* Application Model

We started the application modeling from a C-code specification of a JPEG codec that is publicly available from UC Berkeley. First, we studied this C-code of the JPEG codec in order to fully understand it, and then we modified the code in order to implement our M-JPEG* application. For example, the decoder part was removed and some code was added to implement the Huffman and quantization table adaptation and the RGB to YUV conversion.

Having the M-JPEG* encoder C-code specification we used the SPADE application modeling technique, described in Section II-B, to transform this sequential C-code into a set of parallel communicating processes. Some global data structures were removed in order to be able to parallelize the C-code. The obtained Kahn Process Network which models the M-JPEG* application is shown in Figure 2.

The Video_in process fetches video data (frames) and information about the dimensions and the format, i.e., RGB or YUV, of each incoming frame. Also, Video_in sends all of this data to the DMUX process which distributes it to the different processes within M-JPEG*. The DMUX process checks the format of the incoming video frames and forwards the frames to the DCT process (YUV frames) or to the RGB2YUV process (RGB frames) in a *block-wise fashion*. The RGB2YUV process converts the RGB blocks into YUV blocks. This conversion includes also sub-sampling of the color pixel data in order to obtain 4:2:2 YUV format.

The DCT process receives YUV blocks via two data channels. Via another channel the DCT process receives control tokens

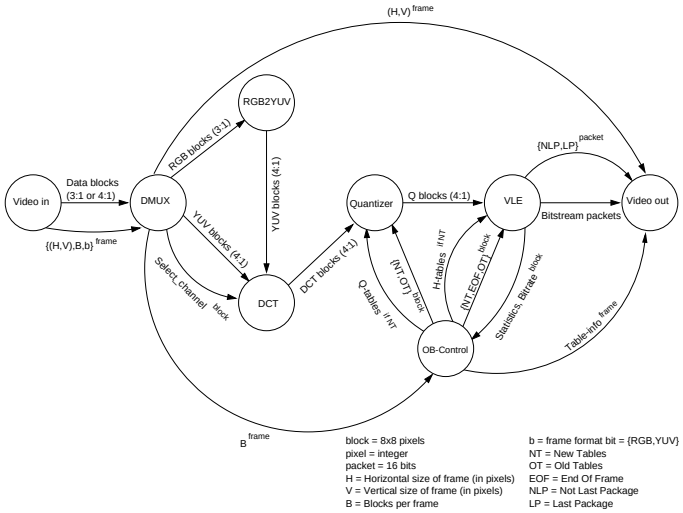

Fig. 2. The M-JPEG* Kahn Process Network.

specifying which data channel is active at the moment. The DCT process transforms YUV blocks into blocks of DCT coefficients using Forward Discrete Cosine Transform and sends the transformed blocks to the Quantizer process. After quantization, the Quantizer process forwards them to the VLE (Variable Length Encoder) process. The VLE process receives quantized DCT coefficients, applies run length encoding and Huffman encoding to the data and transmits the resulting data in 16-bits packets to the Video_out process. A special token indicates whether or not a packet is the last one for the current compressed frame. At the end of the M-JPEG* data flow, the Video_out process makes a standard JPEG header of the compressed image. The Video_out process prepends the header to the compressed image bit stream.

The Quantizer and VLE processes use tables produced by an output bit-rate control process, named OB_Control. The OB_Control process is responsible for generating and distributing these tables to the quantizer and the variable length encoder. The synchronization of the table transmission is performed by a special token which can assume three values: NewTable (NT), OldTable (OT) and EndOfFrame (EOF). At the end of each frame the control process makes a decision whether or not the tables for the next video frame have to be changed. To make this decision it uses the output bit-rate of the current video frame. If the tables have to be changed the control process first computes the new tables using image statistics of the current frame and then sends these tables to the Quantizer and VLE processes. The OB_Control process receives image statistics and output bit-rate from the VLE process. For each frame, the OB_Control process sends the Huffman and quantization tables which are used in the process of compression to the Video_out process.

C. Startemis Architecture

The initial Startemis architecture shown in Figure 3 consists of five processing components connected to a bus and communicating with each other via shared memory. We have chosen such shared memory multi-processor architecture because it is sufficiently complex in order to allow refinement of its components. Also, by changing the implementation of the components, we can easily obtain different heterogeneous architectures, i.e., ar-

chitectures consisting of programmable, reconfigurable and/or dedicated components.

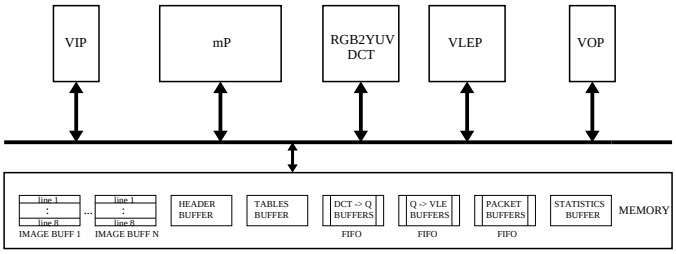

Fig. 3. Abstract view of the Startemis architecture.

Below we give a brief description of the components in the Startemis architecture, depicted in Figure 3. Note that this figure is an abstract view of the architecture; some details, such as the synchronization channels between the processors, are not shown.

VIP (Video In Processor) The *VIP* scans video frames in a *line-wise fashion* and writes the video lines into an *image buffer* in shared memory. There are at least two image buffers, which are used in a round-robin fashion. This allows the *VIP* processor to fill a buffer while the other buffer(s) are consumed. Each image buffer consists of 8 video lines. The size of one line is 1024 pixels. Each pixel in a line is represented by a 32-bit value, containing either the R, G, and B components of the pixel, or the Y, U, and V components. The *VIP* also writes specific information such as the size of the frame, and the frame format, RGB or YUV, for each frame into a *header buffer*.

RGB2YUV/DCT This processor reads the *image buffers* in a *block-wise fashion* and performs a DCT transform onto the fetched blocks. Also, it reads the format of the current frame from the *header buffer*. If the format is RGB an RGB to YUV conversion including 4:2:2 sub-sampling is performed before the DCT transform. The *RGB2YUV/DCT* processor writes the transformed blocks into shared memory in a FIFO buffer called `DCT->Q`. This buffer is used for intercommunication between the *RGB2YUV/DCT* processor and the microprocessor.

mP (microProcessor) The microprocessor has three main functions. First, it reads DCT blocks from the `DCT->Q` buffer, makes a quantization, and writes the quantized blocks into the `Q->VLE` FIFO buffer. Second, it handles the synchronization of the data exchanges between the components in the architecture. Third, the microprocessor handles the adaptation of the quantization and Huffman tables. It uses the information stored in the *statistics buffer* to decide how to change the tables. It updates the *tables buffer* if the tables have been changed. This buffer contains the Huffman and quantization tables for the current video frame.

VLEP (Variable Length and huffman Encoding Processor) The *VLEP* fetches the quantized data blocks from the `Q->VLE` buffer, applies run length encoding and Huffman encoding to them, and writes the resulting bit-stream packets into the *packet* FIFO buffer. Also, the *VLEP* derives statistics from the current block and stores it in the *statistics buffer*.

VOP (Video Out Processor) The *VOP* uses the information stored in the *header buffer* and *tables buffer* in order to make the header of the compressed image. It appends to this header the bit-stream stored in the *packet* FIFO buffer.

D. Startemis Architecture Model

The architecture model was made in accordance with the proposed Startemis architecture shown in Figure 3. We constructed the model using SPADE's library of generic building blocks, described in Section II-C. These blocks are parameterized which means that an important part of the modeling process is to assign realistic numbers to the parameters in order to obtain some realistic results from simulation and exploration.

For each processor a set of *symbolic instructions* and associated *latency values* had to be defined. In order to determine realistic latencies we assumed that for the initial Startemis architecture model the *VIP* and *VOP* are implemented using ASICs, the microprocessor is a MIPS, and the *RGB2YUV/DCT* processor and *VLEP* are DSPs. For the high-level symbolic instructions associated with the microprocessor, the *VLEP*, and the *RGB2YUV/DCT* processor, low-level instruction models were constructed. These models are simple assembler programs. Then we used the data books of the MIPS microprocessor [17][18] and the Analog Devices DSP – ADSP-21160 [19][20] to determine the latencies of the assembler instructions used in the low-level models. From these latencies and the low-level instruction models we calculated the latencies of the symbolic instructions. For the symbolic instructions associated with the *VIP* and the *VOP*, we defined ranges of latency values which have to be explored.

The SPADE architecture description language was used to specify the architecture. A fragment of the architecture description is shown in Figure 4. In the architecture model we defined 1 simulation cycle to be 10ns. We relate all times to this uniform time unit, even though different components may run at different clock speeds. All sizes in the architecture and the mapping description are expressed in units of 8 bits, i.e., 1 byte. For example, a buffer size of 64 means a size of 512 bits, i.e., 64 bytes. The architecture description consists of three main parts. In the first part the processor resources are described. For example, the *VIP* has two output ports `o1` and `o2` , no input ports, and one symbolic instruction `op_ElaborateFrame` with a latency of 20 simulation cycles. In the second part the communication resources are specified. Figure 4 shows that two types of communication resources are used: a bus and FIFO buffers. For each of the buffers a *number* of buffer places and the *size* of each place is specified, e.g., FIFO `F6` has 4 places and the size of each place is 64 bytes. Bus `B1` is specified by three parameters: the *width* of the bus, the *setup* time for a transaction, and the *transfer* time per transfered item with the size of the bus width. The last part of the description specifies the structure of the architecture. In this part the connections among the processor resources and communication resources are described. For example, the description in Figure 4 specifies that output port `o6` of the *mP* is connected to input port `i1` of the *RGB2YUV/DCT* processor via FIFO `F3`. Output port `o1` of the *VIP* is connected to bus `B1` via a buffer with a total size of 7 bytes. The architecture model described in Figure 4 is depicted in the lower part of Figure 5.

```
Architecture MJPEG_Arch;
// 1 unit of size = 1 byte = 8 bits
// 1 cycle = 10 ns

// Processor resources
Processor VIP {
    InPorts { }
    OutPorts {o1; o2;}
    Instructions {op_ElaborateFrame = 20;}
}

Processor RGB2YUVDCT {
    InPorts {i1; i2; i3; i4;}
    OutPorts {o1;o2;}
    Instructions {op_DCT = 1024; op_RGB2YUV = 192;}
}
        :
        :

// Communication resources
Fifo F3 {number = 4; size = 1;}
Fifo F6 {number = 4; size = 64;}
Fifo F8 {number = 1; size = 64;}
Fifo F9 {number = 1; size = 64;}
        :
        :
Bus  B1 {width = 8; setup = 1; transfer = 2;}

// Connections
Structure
{
    Fifo F3 {mP.o6  -> RGB2YUVDCT.i1;}
    Fifo F6 {RGB2YUVDCT.o2 -> RGB2YUVDCT.i4;}
    Fifo F8 {mP.out3 -> RGB2YUVDCT.in2;}
    Fifo F9 {mP.out4 -> RGB2YUVDCT.in3;}
            :
            :
    Bus  B1 {
         VIP.o1 {number = 1; size = 7;};
         VIP.o2 {number = 1; size = 64;};
         RGB2YUVDCT.o1 {number = 1; size = 128;};
                 :
                 :
    }
}
```

Fig. 4. Fragment of the description of the architecture model.

E. Startemis Mapping

The final step of the modeling process is the mapping of the M-JPEG* application model onto the Startemis architecture model. Figure 5 shows this mapping in detail. Both the application and the architecture are depicted in terms of the SPADE specific modeling technique. For the mapping we used the SPADE mapping mechanism, described in Section II-D. Within Startemis, the Video_in and Video_out processes are mapped onto the *VIP* and the *VOP*, respectively. The VLE process is mapped onto the *VLEP*. The two processes RGB2YUV and DCT are mapped onto the *RGB2YUV/DCT* processor. The remaining processes are mapped onto the microprocessor. The mapping of the M-JPEG* channels onto the communication structures of the architecture was determined by mapping the processes' ports onto the processors' ports.

For the purpose of mapping, SPADE offers a mapping description language. A fragment of the mapping description is shown in Figure 6.

The first part of the mapping description in Figure 6 specifies the mapping of the processes and their ports onto the processor components and their ports. For instance, process Video_in is mapped onto the *VIP*. The ports `out_HeaderInfo` and `out_BlockData` of the Video_in process are mapped onto the ports `o1` and `o2` of the *VIP*, respectively. Next, for each application channel a *tokensize* is specified. If the channel is mapped onto a bus and the communication should take place via shared memory, then the number of places and the size of each place

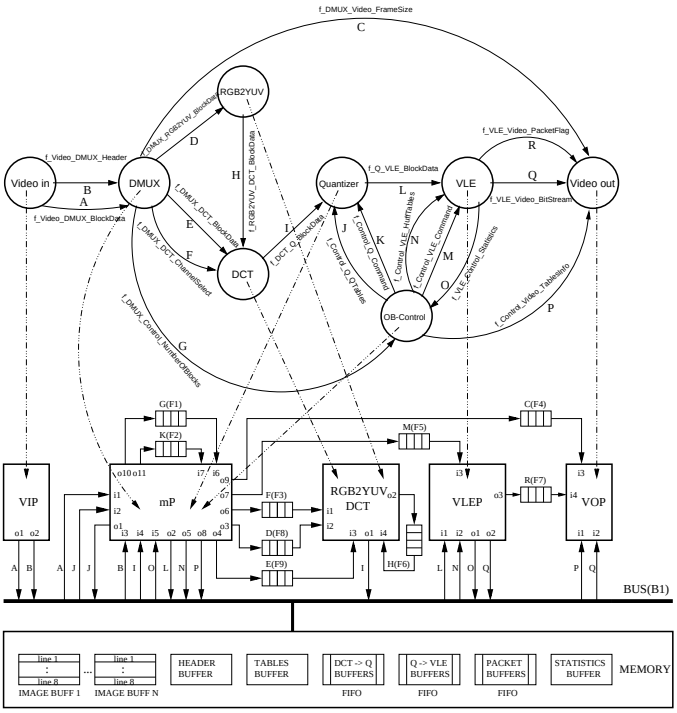

Fig. 5. The application model, the architecture model, and the mapping.

```
Mapping MJPEG_Map (MJPEG_Appl, MJPEG_Arch);

pr_Video_in : VIP {
    out_HeaderInfo : o1;
    out_BlockData  : o2;
}

pr_DCT : RGB2YUVDCT {
    in_BlockData1 : i4;
    in_BlockData2 : i3;
    in_BlockType  : i1;
    out_BlockData : o1;
}

pr_RGB2YUV : RGB2YUVDCT {
    in_BlockData  : i2;
    out_BlockData : o2;
}
       :
       :

Channels {
    f_Video_DMUX_Header {
        tokensize = 7;
        numbermembufs = 1; membufsize = 7;
    };
    f_RGB2YUV_DCT_BlockData {tokensize = 64;};
    f_DCT_Q_BlockData {
        tokensize = 128;
        numbermembufs = 4;membufsize = 128;
    };
            :
            :
}

Schedulers {
    mP           : default { };
    RGB2YUVDCT   : default { };
}
```

Fig. 6. Fragment of the description of the mapping shown in Figure 5

of the buffers in shared memory are specified. For example, the tokens which are transfered via channel `f_DCT_Q_BlockData` have a size of 128 bytes; 4 buffers of 128 bytes are allocated in shared memory for this channel. The last part of the mapping description shown in Figure 6 specifies the type of the schedulers. For the microprocessor and the *RGB2YUV/DCT* processor

the default scheduler is selected.

IV. Experiments and Results

In this section we present some of the experiments we have done using the SPADE simulation framework in order to quantify the proposed Startemis architecture. Also, we present and analyze the results we have obtained from the simulations. We evaluated whether the performance metrics provided by the SPADE tool can be used effectively for exploration. We were interested in finding what kind of useful information about the architecture performance we could obtain from the performance numbers, having in mind that the exploration was made at a high level of abstraction.

In our experiments we were interested in the *maximum frame rate at the output*, measured in *frames per second*, to which we refer as *RATE*. The *RATE* depends on the parameters of the architecture template, the size of the incoming frames, and the format, YUV or RGB, of the frames. For the purpose of the experiments, the architecture parameters that we looked into have been: the number of processor components, the latencies of processor components, the speed of the bus, and the speed and size of shared memory. In order to simplify the simulation process and analysis we initially kept the frame size and the frame format constant. For our experiments the frame size is 128×128 pixels (8 bits per pixel) and the frame format is RGB. For larger frames the *RATE* will decrease; if we increase the horizontal and vertical sizes of the frames by a factor x and a factor y, respectively, then the *RATE* of the architecture will decrease roughly by a factor of $x \times y$. For YUV frames, the *RATE* will be as good as, or better than, the *RATE* for RGB frames, because the only difference is that the RGB to YUV conversion does not need to be performed.

The initial speed and width of the bus have been set to 100MHz and 64 bits, respectively. These values are assumed to be at the upper limit of the range in which they could have been chosen. A similar decision was made for the shared memory for which we selected an SRAM-type memory of size 64KB with write and read cycles of 10ns each.

We started our experiments with a simulation of the given M-JPEG* application-architecture pair using the SPADE simulation environment. The application, the architecture and their specifications (modeling) are described in Section III-B and Section III-D, respectively. From the SPADE simulation we obtained numbers, relevant to the evaluation of the performance of the implementation. Some of them are given in Table I. We used the SPADE performance numbers related to the times at which a new frame appears at the output in order to calculate the *RATE* of the initial architecture which turned out to be *167 frames per second*.

After this first step, we can have three possible scenarios to proceed:

- **Scenario 1:** The *RATE* of the architecture satisfies the requirements and we stop the exploration process.
- **Scenario 2:** The *RATE* of the architecture satisfies the requirements and we start exploring (part of) the parameter space of the architecture for modifications that can improve the current *performance–cost* ratio. This process includes finding and removing some resource redundancy and excess speed without a significant change of the *RATE* and the flexibility of the architecture. We use the SPADE performance numbers as a guidance in this process. As the performance indicated by the *RATE* should not be changed, we can only improve the performance–cost ratio by a reduction in terms of cost, e.g., silicon area or power consumption.
- **Scenario 3:** The *RATE* of the architecture does not satisfy the requirements and we start exploring (part of) the parameter space of the architecture for modifications that can improve the performance (*RATE*). Again, we use the SPADE performance numbers as a guidance in the process.

We looked into Scenario 2 and Scenario 3 independently. The results of these experiments are presented in Section IV-A and Section IV-B, respectively.

TABLE I
PERFORMANCE NUMBERS OF THE INITIAL STARTEMIS ARCHITECTURE.

Processor	*executing*	*busy with I/O*	*waiting*
RGB2YUV/DCT	95%	1%	4%
mP	35%	20%	45%
VLEP	26%	5%	69%
VIP	1%	3%	96%
VOP	1%	1%	98%

Bus utilization	40%

A. Scenario 2

The performance numbers in Table I suggest that the given architecture has a poor load balance because the *mP* and the *VLEP* are not utilized very well. Also, the DSP which we have chosen for the *VLEP* is too powerful for the kind of operations needed by the VLE process. That is why the *VLEP* is waiting most of the time (69%). On the other hand, the *mP* is executing and busy with I/O only 55% of the time. Taking these observations into account we conclude that we might not need a separate processor component for the run length encoding and Huffman encoding. We decided to remove the *VLEP* and to map the VLE process onto the *mP*, expecting that the *RATE* would not decrease significantly. We simulated this modified architecture in order to see how the performance was changed. Table II presents the new values of the performance numbers given before in Table I.

TABLE II
PERFORMANCE NUMBERS AFTER REMOVING THE *VLEP*.

Processor	*executing*	*busy with I/O*	*waiting*
RGB2YUV/DCT	95%	1%	4%
mP	63%	9%	28%
VIP	1%	2%	97%
VOP	1%	1%	98%

Bus utilization	12%

Again we used the SPADE performance numbers to calculate the *RATE* and we saw that it was unchanged at 167 frames per second. This fact shows that we have removed a redundant pro-

cessor component. This results in a reduction of the cost in terms of silicon area. Currently, cost measures are indirectly derived quantities. SPADE does not yet provide metrics related to the silicon area and power dissipation.

Next, we analyzed the new simulation results and we saw that the bus is utilized only 12% of the time which means that the speed of the architecture will not be very *sensitive* to a decrease of the speed of the bus and the shared memory. We observed by SPADE simulation that if we decrease the speed of the bus and the memory five times, then the *RATE* of the architecture is *162 frames per second*. The decrease of the *RATE* is only 3%, which means that we do not have a significant change of the *RATE*. This fact shows that we have removed some excess speed in the architecture. Moreover, the fact that we decreased the speed of the memory five times means that we can use DRAM instead of SRAM which leads again to a reduction of the silicon area, because the silicon area of the DRAM is significantly smaller than the silicon area of the SRAM.

This experiment demonstrates that the SPADE performance metrics provide useful measures to detect resource redundancy and excess speed in the architecture. We showed above that finding and removing this redundancy can lead to a reduction of the silicon area cost of the architecture.

B. Scenario 3

We now follow a different scenario in which we assume that the initial *RATE* does not satisfy the requirements. As we mentioned before, after the first step of our exploration process, we found that the *RATE* of the architecture is 167 frames per second for video frames of size 128×128 pixels. If we assume that for real-time applications we need at least 25 frames per second, then the architecture can process these frames in real time. But the *RATE* of the proposed architecture depends on the frame size. For example, the *RATE* of the architecture for video frames with size 512×512 pixels is 10 frames per second. This means that if we want to achieve a real-time speed for large video frames we have to make some changes in the architecture parameters. The main parameters we can change in order to increase the *RATE* are the latencies of the architecture components. The possible values of the latency parameters should however stay within reasonable bounds.

From the SPADE simulation numbers presented in Table I we see that the *RGB2YUV/DCT* processor executes 95% of the time while the other components execute less than 36% of the time. This result suggests that the *RATE* of the architecture is very *sensitive* to the latency of the *RGB2YUV/DCT* processor. We decreased the latency of this component five times. The *RATE* of the architecture became *256 frames per second* for images of size 128×128 pixels. Using this *RATE* number, we can easily estimate that the *RATE* for images of size 512×512 pixels is about *16 frames per second* which is still not high enough. Simulation results are shown in Table III.

The results shown in Table III show that the *RGB2YUV/DCT* processor is no longer the bottleneck for the performance. If we go on decreasing the latency of the *RGB2YUV/DCT* processor we will unbalance the architecture and we will not achieve a significant increase of the *RATE*. The results in Table III show that we can expect a significant increase of the *RATE* if we decrease

TABLE III
PERFORMANCE NUMBERS AFTER DECREASING THE LATENCY OF THE *RGB2YUV/DCT* PROCESSOR.

Processor	*executing*	*busy with I/O*	*waiting*
RGB2YUV/DCT	36%	1%	63%
mP	55%	39%	6%
VLEP	42%	23%	35%
VIP	1%	8%	91%
VOP	1%	4%	95%

Bus utilization	62%

the latency of the microprocessor. We decreased the latency of the *mP* two times. In this case, the SPADE simulation results show that the *RATE* of the architecture is *354 frames per second*. Estimated *RATE* for video frames of size 512×512 is *22 frames per second*. The SPADE performance numbers for each of the components are given in Table IV.

TABLE IV
PERFORMANCE NUMBERS AFTER DECREASING THE LATENCY OF THE MICROPROCESSOR.

Processor	*executing*	*busy with I/O*	*waiting*
RGB2YUV/DCT	46%	1%	53%
mP	40%	59%	1%
VLEP	54%	33%	13%
VIP	1%	18%	81%
VOP	1%	5%	94%

Bus utilization	81%

Analyzing the results presented above we saw that the decrease of the latency of the processor components increases the *RATE* of the architecture. But still the *RATE* does not satisfy the real-time requirement for large images. If we continue to decrease the latencies of the processor components, they will get outside the feasible range of these latencies. Also, we will not achieve a significant speedup of the architecture, because Table III and Table IV show a significant increase of the time the processor components spend on performing I/O operations. The latter means that the communication structure of the architecture becomes a bottleneck. If we increase the speed of the bus we will not change the *RATE* of the architecture, because the speed of the shared memory is unchanged. We cannot increase the speed of the memory because it will lead to unrealizable memory.

According to the results of the experiment we presented above we can conclude that although our architecture consists of five fast processor components working in parallel, we cannot achieve real-time speed for large video frames. The main reason is that the architecture cannot exploit the maximum parallelism of the application, because the communication structure – a common bus and shared memory – obstructs the parallelism.

Starting from a given application and a given architecture template, we have demonstrated how we can use the SPADE environment and tools to obtain performance numbers that can

help evaluating the *performance–cost* ratio at an abstract level early in a design process. In the experiments presented in this paper, we have looked at the impact of some of the parameters of the architecture template. The three experiments conducted can be seen as part of an exploration trajectory around the initial Startemis architecture. The results of this session suggest that the next step might be to change the overall architecture template and to go into another exploration step using this new template.

V. Conclusions

We have presented a case study of an M-JPEG encoder application mapped onto a shared memory multi-processor architecture, which is used in the Artemis project as a driver for the development of the Artemis workbench. We did some explorations of the initial application and architecture at an abstract level using SPADE. By doing these experiments we evaluated the use of SPADE early in the design process of heterogeneous signal processing architectures. It appears that SPADE can give a designer useful feedback on the performance of a system, which may help a designer in improving the system. This was illustrated by the scenarios in Section IV. The simulations done in these scenarios typically took a few minutes for an input sequence of several frames. Also, changes to the architecture and mapping could be easily made to the textual descriptions. SPADE currently lacks feedback on other metrics than performance metrics, such as, silicon area and power dissipation. Also, the representation of data is still something that can be improved. These issues will be topic of further research, of which part will be covered in the Artemis project.

References

[1] F. Balarin, E. Sentovich, M Chiodo, P. Giusto, H. Hsieh, B Tabbara, A. Jurecska, L. Lavagno, C. Passerone, K. Suzuki, and A. Sangiovanni-Vincentelli, *Hardware-Software Co-design of Embedded Systems – The POLIS approach*, Kluwer Academic Publishers, 1997.

[2] Grant Martin and Bill Salefski, "Methodology and technology for design of communications and multimedia products via system-level IP integration," in *Proc. DATE'98 Designers' Forum*, 1998, pp. 11–18.

[3] Stan Liao, Steve Tjiang, and Rajesh Gupta, "An efficient implementation of reactivity for modeling hardware in the Scenic design environment," in *Proc. DAC'97*, Anaheim, CA, June 9-13 1997, pp. 70–75.

[4] http://www.innoveda.com/product/earchitect, Innoveda, Inc.

[5] Paul Lieverse, Pieter van der Wolf, Ed Deprettere, and Kees Vissers, "A methodology for architecture exploration of heterogeneous signal processing systems," in *Proc. 1999 Workshop on Signal Processing Systems (SiPS'99)*, Taipei, Taiwan, Oct. 20-22 1999, pp. 181–190.

[6] Andy Pimentel, Pieter van der Wolf, Bob Hertzberger, Ed Deprettere, Jos T.J. van Eijndhoven, and Stamatis Vassiliadis, "The Artemis architecture workbench," in *Progress Workshop 2000*, Utrecht, The Netherlands, Oct. 13 2000.

[7] A.D. Pimentel, A.W. van Halderen, and L.O. Hertzberger, "Sesame: Simulation of embedded system architectures for multi-level exploration," Tech. Rep., University of Amsterdam, 2000, Artemis internal report.

[8] B. Kienhuis, E. Deprettere, K. Vissers, and P. van der Wolf, "An approach for quantitative analysis of application-specific dataflow architectures," in *Proc. ASAP'97*, July 14-16 1997.

[9] Richard A. Uhlig and Trevor N. Mudge, "Trace-driven memory simulation: A survey," *ACM Computing Surveys*, vol. 29, no. 2, pp. 128–170, June 1997.

[10] Gilles Kahn, "The semantics of a simple language for parallel programming," in *Proc. of the IFIP Congress 74*. 1974, North-Holland Publishing Co.

[11] E.A. de Kock, G. Essink, W.J.M. Smits, P. van der Wolf, J.-Y. Brunel, W.M. Kruijtzer, P. Lieverse, and K.A. Vissers, "YAPI: Application modeling for signal processing systems," in *Proc. 37th Design Automation Conference (DAC'2000)*, Los Angeles, CA, June 5-9 2000, pp. 402–405.

[12] A.J.C. van Gemund, "Performance prediction of parallel processing systems: The PAMELA methodology," in *Proc. 7th ACM Int. Conference on Supercomputing*, Tokyo, July 1993, pp. 318–327.

[13] Wido Kruijtzer, "TSS: Tool for System Simulation," *IST Newsletter*, vol. 17, pp. 5–7, Mar. 1997, Philips Internal Publication.

[14] Vasudev Bhaskaran and Konstantinos Konstantinides, *Image and Video Compression Standards; Algorithms and Architectures*, Kluwer Academic Publishers, 1995.

[15] W.B. Pennebacker and J.L. Mitchel, *JPEG Still Image Data Compression Standard*, Van Nostrand Reinhold, New York, 1993.

[16] W.B. Pennebacker, J.L. Mitchel, C.E. Fogg, and D.J. LeGall, *MPEG Video Compression Standard*, Chapman and Hall, 1996.

[17] Joe Heinrich, *MIPS 4000 Microprocessor - Users's Manual*, MIPS Technologies Inc., 1994.

[18] http://www.mips.com/products, MIPS Technologies Inc.

[19] Analog Devices Inc., *ADSP-21160 SHARC DSP; Instruction Set Reference*, Nov. 1999.

[20] Analog Devices Inc., *ADSP-21160 Preliminary Data Sheet*, Jan. 2000.