SystemC-based Modelling, Seamless Refinement, and Synthesis of a JPEG 2000 Decoder

Kim Grüttnert*, Frank Oppenheimer*, Wolfgang Nebel*, Fabien Colas-Bigey†, Anne-Marie Fouilliart†
*OFFIS e.V., Germany  †Thales Communications, France

Abstract

This paper will exemplarily describe and evaluate the OSSS methodology for embedded hardware/software systems and its use in a JPEG 2000 decoder case-study. The OSSS approach defines a design flow starting from an Application Model providing a rich subset of SystemCTM/C++ augmented with specific OSSS language concepts. It can be used to identify the most promising parallel structure by comparing different design alternatives. A clearly defined refinement process leads to the Virtual Target Architecture (VTA) Model. These refinements enable an analysis of the system behaviour at cycle-accurate granularity and support the exploration of different target architectures for the JPEG 2000 decoder. VTA models can be used as direct input for the FOSSY synthesis tool, which performs an automatic transformation into implementation models; that is to generate VHDL code for hardware, C/C++ for software, and platform configuration files for the target technology.

1. Introduction and Motivation

Today, highly integrated embedded systems have a wide range of usage in many innovative European industries, such as telecommunications and automotive. These modern embedded systems consist of several IP blocks plus a few custom components and often use pre-defined technology platforms to implement them. Existing and upcoming technologies offer ever more possibilities to cope with the increasing application requirements. Nevertheless, efficiently designing such systems remains a major challenge since electronic design automation tools and methodologies cannot keep pace.

To remain competitive, manufacturers have to offer innovative products while reducing costs and development time. Accomplishing this strategy therefore requires new specification languages and design flows. A suitable design methodology must address finding the right communication architecture and support easy integration of hardware and software. The development of such embedded hardware/software systems is often a collaborative effort of specialised engineers. However, developing hardware and software elements separately might lead to specification and implementation inconsistencies. The hardware and software design flows therefore need to be more tightly coupled. New specification languages should favour this coupling, resolve integration problems and reduce time to market. Moreover, the automation of time-consuming steps in the design flow and architecture exploration should lead to a more efficient and robust implementation.

There are several design methodologies available that address these demands. SpecC [2] is completely refinement driven and introduces lots of interesting concepts, like hierarchical behaviours and channels. Since it is based on the C language it does not work well with the upcoming object-oriented development of SW dominated SoCs. Moreover it is not well accepted by the European industry which favours SystemCTM[10] instead. The OSCI TLM standard [12, 9] seems well adapted for the modelling of communication in SystemC. There have already been some attempts to use TLM in rapid prototyping [7], which are promising. Commercial C/C++ to hardware tools like [1, 8] allow behaviour level synthesis and gain more and more interest by the industry.

In this paper we briefly describe the SystemC-based OSSS design methodology developed in the ICODES project [6], which provides a refinement concept close to SpecC. For the description of communication it internally makes use of the OSCI TLM standard. Moreover the FOSSY synthesis tool provides a path to communication synthesis as well as high-level hardware synthesis. We illustrate and evaluate selected parts of the proposed approach in an industrial case-study, namely a JPEG 2000 decoder.

In the following section we briefly describe the main components of the JPEG 2000 decoder. The third section introduces the main concepts of the OSSS methodology and illustrates their application in the case-study. The last part of the paper compares quantitatively the OSSS approach with a standard design approach using an industrial C+/VHDL-based implementation of the JPEG 2000 decoder on a Xilinx Virtex-4 FPGA.
2. JPEG 2000 Case-Study

The JPEG 2000 decoder used as case-study in this paper is a recent compression standard targeting different types of still images (bi-level, grey-level, colour, multi-components, etc.). It supports different characteristics (natural images, scientific, medical, remote sensing imagery, text, etc.) allowing different imaging models (client/server, real-time transmission, image library archival, etc.) within a unified system.

As shown in Figure 1 the JPEG 2000 decoding chain is composed of several functional blocks, each of which performs a specific part of the image processing. The core components are the arithmetic decoder and the IDWT (Inverse Discrete Wavelet Transformation). These two steps allow compressing and organising data efficiently. In our case-study the JPEG 2000 decoder supports a lossy (IDWT 97) and lossless (IDWT 53) mode. Most JPEG 2000 images are processed as tiles (small parts of the image), which are more manageable and more adapted to a pipelined computation. For more information about JPEG 2000, please refer to [11].

3. Specification in OSSS

The OSSS (Oldenburg System Synthesis Subset) defines a synthesizable system description language based on SystemC. It supports C++ concepts, – like classes, templates, inheritance – and unique OSSS concepts – like Shared Objects and OSSS-Channels [3]. OSSS is provided as an open-source C++ class library, which can be used in arbitrary SystemC models. The following section gives only a brief overview of the design principles in OSSS and explains the motivation for the two OSSS modelling layers. Afterwards the main modelling concepts at each layer are described and their practical use in the JPEG 2000 case-study is shown.

![Figure 1. JPEG 2000 processing overview](image1)

![Figure 2. The OSSS design methodology](image2)

Figure 2 depicts the overall OSSS design methodology of hardware/software SoC design, which defines three types of models. It begins with the Application (Layer) Model. It describes the function, an approximate timing behaviour, and the logical structure of the system but abstracts from all platform-specific details (like processors, memories and communication channels). The Application Layer offers the possibility to express an unbiased functional system model regardless of HW or SW implementations. The aim of this executable model is to identify bottlenecks and to find the most promising parallel structure for the application. Starting from a pure functional model it leads the designer to a HW/SW partitioning decision. A typical design entry is the so-called software centric approach, where the designer starts with a SW only functional description of the entire system. For the refinement on Application Layer this typically means to relocate performance hungry algorithms from software to hardware or compute complex algorithms in pipelined structures.

The next step is to refine the Application Model to a Virtual Target Architecture (VTA) Model. All logical components of the Application Model are now assigned to target architecture resources; software tasks are mapped on processors (N to 1 mapping) and modules on hardware blocks (1 to 1 mapping). Communication links can be mapped on shared buses or point-to-point connections using so-called OSSS Channels (N to 1 mapping). The goal of this model is to assess the effect of different implementation alternatives on the timing behaviour of the design. By mapping multiple communication links on a shared bus or using different bus protocols, the effects on contention in a system can be determined in a cycle-accurate model.

The final VTA Model reflects all relevant details of the resulting implementation model. The implementation model will be generated in the automated synthesis process (see Section 4) which completes the OSSS design flow.
For more information about OSSS please refer to [5].

3.1. Application Layer Model

Every OSSS model at Application Layer is composed of three types of structural blocks: (hardware) modules, Software Tasks, and Shared Objects.

Modules can contain a fixed number of concurrent processes while SW tasks are limited to a single process. In contrast to modules and tasks, Shared Objects are passive which means they cannot initiate an execution on their own. They enable modelling of communication and synchronisation between active system components (modules and SW tasks) at a high level of abstraction. This is achieved by providing method-based interfaces for inter-process communication and flexible scheduling and arbitration mechanisms for the concurrent access to shared resources.

On the Application Layer, communication between modules or tasks via Shared Objects is performed by user-defined method calls with arguments of any valid C++ or SystemC data-type (except pointers & references) through abstract communication links with the following properties:

- **directed** The source (method caller) of a communication link is a port while the destination (method provider) is an interface, i.e. port-to-interface binding.
- **blocking** A method call on a port will not return until its execution has been completed.

A Shared Object can be understood as a hierarchical SystemC channel with properly defined synthesis semantics.

Figure 3 shows the Implementation flow of the JPEG 2000 decoder design. The starting point for our case study was an earlier C/C++ implementation of a JPEG 2000 decoder ①. A profiling run of the JPEG 2000 decoder on a Xilinx MicroBlaze™ processor (because it is the only available target processor considered in this case study) points out the execution times of the algorithmic parts as shown in Figure 1.

In the initial Application Model a Software Task contains the arithmetic decoder, ICT and DC Shift (normalisation) ②. The Shared Object (SO), which here serves merely as a co-processing unit, implements the IQ and IDWT. The software performs first the decoding, calls the IQ, calls the IDWT and finally continues with the ICT and the DC Shift. Since all method-calls to Shared Objects are blocking, the software execution cannot proceed until the SO has finished its computation. The simulation results in Table 1 show a speed-up of about 10/19% (lossless/lossy) compared to ①. This speed-up is higher than the maximum obtainable speed-up because it does not incorporate communication costs. They can be analysed after communication refinement on VTA Layer.

Analysing the Application Model, it becomes apparent that this structure is not optimal as it does not take advantage of executing hardware and software in parallel. Therefore, we changed the initial Application Model ending up with a parallel version of the JPEG 2000 decoder ③. The processing has been parallelised between hardware and software and a pipeline structure operates in parallel on several tiles of the picture. Regrettably, this effort only has a small impact on the overall speed-up (cp. Table 1).

Since the arithmetic decoder takes up to 88 % of the overall processing time, it needs to be parallelised to obtain a more significant speedup. The structure and complexity of the decoder did not allow an affordable hardware implementation. Instead we have chosen to implement it by four independent Software Tasks (cf. ④ & ⑤) performing the arithmetic decoding of disjoint parts of the image in parallel. The amount of parallel HW was not altered since the working load of the IQ+IDWT was rather low.

The Application Layer models ③ & ⑤ contain two Shared Objects. The first Shared Object handles the communication and synchronisation of one ③ to four ⑤ software tasks and three parallel hardware blocks. Moreover, it contains a data structure to transfer large objects, such as...
parts of the images (called tiles) and the IQ algorithm. The ability not only to store and transfer data (here tiles) but also to perform computations (here IQ) within the object was considered to be very useful. The IDWT params Shared Object is responsible for exchanging sequences of parameters between the control part (IDWT2D) and the lossless (IDWT53), and lossy (IDWT97) part of the IDWT. It is used not only for parameter storage and transfers, but also as arbitration unit between the three concurrent IDWT components.

After analysis, exploration and behaviour refinement on the Application Layer we ended up with a design delivering an acceptable speedup by a factor of 4.5/5 compared to the software-only implementation. Version 5 suffers from the increased working load and the corresponding arbitration overhead of the HW/SW SO with seven clients. Hence 5 is slightly slower than 4. In the next step the mapping and communication refinement process is performed to end up with a cycle-accurate Virtual Target Architecture Model.

3.2. Architecture Layer Model

The refinement step from the Application Model to the Virtual Target Architecture Model involves the mapping of modules, tasks, and Shared Objects to appropriate architectural building blocks. Hence, the OSSS methodology provides a set of building blocks that are stored in a hierarchical and user-extensible library. Communication on the VTA is described by so-called OSSS Channels. These channels, which are part of the Virtual Target Architecture communication library, enable the designer to model the communication protocol independently from the behaviour of a design.

A key goal of the OSSS methodology is to enable the seamless refinement and automatic synthesis of the communication architecture.

In order to achieve this, all OSSS Channels use the so-called RMI (Remote Method Invocation) concept [4] to encapsulate the physical layer of the communication. This concept provides a uniform abstraction layer for arbitrary physical communication channels like buses or point-to-point connections. The RMI mechanism decouples the method-based communication at Application Layer from the specific characteristics of the physical channels. Consequently, it enables changing the physical communication layer without modifying the behavioural code and in particular the method calls in the design components.

In essence, the following steps have been performed to transfer the JPEG2000 Application Model to a cycle-accurate Virtual Target Architecture Model (e.g. \( \text{VTA\rightarrow App} \))

**Software Task \( \rightarrow \) Software Processor:** The software task is mapped onto a Software Processor of the OSSS architecture class library. This mapping is done easily by instantiating a Xilinx MicroBlaze class and mapping the software task by calling the `add_sw_task` method on the processor object.

Following the OSSS methodology, the timing behaviour can already be specified at Application Layer where it might be rather coarse. At Virtual Target Architecture Layer it should be refined to match the fine grained cycle-accurate timing of the hardware and the communication model.

We have performed timing profiling on the chosen target processor running the entire JPEG 2000 decoder in software. The timing profiles have been back-annotated to the Application and the Virtual Target Architecture Models. Assuming the arithmetic decoder takes approximately 180 ms for a single tile, the following code snippet shows how to perform software timing annotations. Using the SW profiling information it makes most sense to use the so-called EET (Estimated Execution Time) blocks to annotate the execution time per function.

```c
{ OSSS_EET( sc_time(180,SC_MS) )
  data_send = decode_tile(Image,i+1); }
```

**Shared Object \( \rightarrow \) Object Socket:** All Shared Objects are wrapped by so called Object Sockets. This enables the connection to arbitrary OSSS Channels.

**Module \( \rightarrow \) Hardware Block:** All modules are replaced by Hardware Blocks, which enable the connection to the global clock and reset signals and to arbitrary OSSS Channels.

**Data serialisation:** The serialisation cuts large user-defined data structures into manageable chunks of data to be transferred efficiently via OSSS Channels. The OSSS-modelling library provides pre-defined methods, which can be re-used via inheritance to implement the serialisation method.

**Explicit memory insertion:** Some data members - especially large arrays - in HW/SW Shared Objects should be mapped into explicit memory (in the following listing: Xilinx Block RAM with 32 bit data and 16 bit address width). In the VTA Model it is important to assess the effects of data locality in order to reach the best area/performance trade-off for the implementation. If data is not stored explicitly into such memories, it would be synthesized as fast registers and dramatically increase the amount of occupied FPGA slices.

```c
oss array<short,2+N5>
   m_array; //on Application Layer
xilinx_blockram<oss array<short,2+N5>, 32, 16>
   m_array; //on VTA Layer
```

**Communication Link \( \rightarrow \) OSSS Channel:** One of the most challenging parts in the refinement is the mapping of communication links from the Application Layer Model onto physical communication resources (OSSS Channels) of the Virtual Target Architecture Model. In this refinement process multiple communication links can either be
bundled in a physical shared bus or each communication link can be mapped to a dedicated point-to-point connection. In model ⑧/⑨ all communication links to the HW/SW SO are mapped to an OSSS Channel implementing an IBM OPB (On-Chip Peripheral Bus). Communication links to the IDWT Params SO are mapped to dedicated point-to-point channels. All method calls that have been performed through communication links in the Application Layer Model are now performed through OPB or point-to-point channels using the OSSS RMI protocol.

The OSSS Channels based on the RMI abstraction provides a simple mechanism to map the method-based communication onto different physical communication channels. This enables the exploration of alternative mappings leading to more efficient solutions. For example, model ⑧/⑨ uses an alternative mapping where only point-to-point channels implement the communication of the IDWT hardware blocks with the Shared Object.

The lower part of Table 1 shows the impact of the VTA refinements on the simulation results. Now we compare the pure Application Layer Models with the corresponding VTA Layer mappings:

③ → ⑬/⑭: The IDWT time is increased significantly (up to a factor of 8). This increase in time results directly from the channel refinement and the explicit memory insertion in the VTA model. But anyway, this version is dominated by the SW part and therefore the overall decoding time is not affected significantly.

④ → ⑬/⑮: In ⑬ the IDWT time is increased even more than in ⑬ since three more processors are competing for access to the single shared bus. The IDWT times of ⑬ and ⑮ are equal since in both VTA models the same P2P connections are used and the HW/SW SO decouples the bus accesses initiated by the SW tasks.

With all refinements we still observe a speed-up by a factor of 12/16 for the IDWT in HW ⑬/⑮ compared to the SW only execution in ③. The exploration on the VTA Layer has confirmed our assumptions obtained from the Application Layer experiments. It shows that the overall processing time is dominated by the SW arithmetic decoder. Even after SW parallelisation ⑬ is an affordable implementation of the JPEG 2000 decoder while ⑮ does better scale with increasing parallelism.

### 4. Implementation Models

Figure 4 shows the OSSS synthesis flow for the JPEG 2000 decoder design. For synthesis we incorporate the behaviour of the system as defined by the Application Layer Model, the allocated HW resources of the VTA Layer and the mapping information between them. Since our prototypic synthesis flow interfaces the Xilinx Embedded Development Kit (EDK) we are generating vendor-specific architecture definition files. These are the MSS (Microprocessor Software Specification) and MHS (Microprocessor Hardware Specification) files that are used for the creation of an EDK project. For the implementation of the JPEG 2000 case-study we are using the MicroBlaze, the OPB and an OPB multi-channel DDR-RAM controller from the Xilinx EDK IP core library. The SW tasks are cross-compiled and linked against a specific OSSS embedded library that enables the communication with the HW/SW Shared Object. The entire HW subsystem is transformed from SystemC/OSSS to synthesisable VHDL code by our high-level synthesis tool called FOSSY (Functional Oldenburg System Synthesiser). The resulting VHDL code is inserted into the generated EDK project and further processed by the Xilinx Synthesis Tool (XST) or other third-party RTL synthesis tools.

In the following we are going to have a closer look on

---

**Table 1. Simulation results**

<table>
<thead>
<tr>
<th>Version of JPEG Decoder Model</th>
<th>Decoding Time* [ms]</th>
<th>IDWT Time* [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>lossless</td>
<td>lossy</td>
</tr>
<tr>
<td>SW only on MicroBlaze</td>
<td>3188.79</td>
<td>2662.76</td>
</tr>
<tr>
<td>HW/SW not parallel</td>
<td>2847.03</td>
<td>2144.96</td>
</tr>
<tr>
<td>HW/SW parallel (3 IDWT modules)</td>
<td>2845.08</td>
<td>2138.21</td>
</tr>
<tr>
<td>SW parallel (cp. ③)</td>
<td>712.11</td>
<td>537.62</td>
</tr>
<tr>
<td>SW &amp; HW/SW parallel (cp. ⑤)</td>
<td>721.08</td>
<td>550.22</td>
</tr>
</tbody>
</table>

*Time needed to decode 16 tiles with 3 components each @ 100 MHz

---

**Figure 4. Synthesis flow**
Table 2. RTL Synthesis results of the IDWT

<table>
<thead>
<tr>
<th></th>
<th>lossless (IDWT53)</th>
<th>lossy (IDWT97)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOSSY</td>
<td>reference</td>
<td>reference</td>
</tr>
<tr>
<td>Number of Slice Flip Flops</td>
<td>298</td>
<td>293</td>
</tr>
<tr>
<td>Number of 4 input LUTs</td>
<td>682</td>
<td>619</td>
</tr>
<tr>
<td>Number of occupied Slices</td>
<td>436</td>
<td>356</td>
</tr>
<tr>
<td>Total equivalent gate count</td>
<td>7882</td>
<td>7146</td>
</tr>
<tr>
<td>Estimated frequency (MHz)</td>
<td>212.1</td>
<td>212.4</td>
</tr>
</tbody>
</table>

The FOSSY generated VHDL code of the IDWT modules.

**IDWT Reference Models:** A wavelet transform improves encoding efficiency by exploiting pixel correlation. It is composed of a set of filters that extract parts of the original signal: low-pass filters preserve a blurred representation of the original signal, while high-pass filters preserve transitions and textures of the image. On the decoder side, the Inverse Discrete Wavelet Transform reconstructs the image as shown in Figure 1. In the lossless mode (IDWT53) the used filter-bank is a bi-orthogonal (5,3) with integer taps, and in the lossy mode (IDWT97) it is a Daubechies (9,7). The hand-crafted reference model is written in synthesisable VHDL and consists out of 404 (IDWT53) and 948 (IDWT97) lines of code.

**FOSSY Generated Models:** The synthesisable SystemC IDWT models consist out of 356 (IDWT53) and 903 (IDWT97) lines of code. The overall structure of the SystemC and the reference VHDL model is very similar. Both use explicit state machines and functions and procedures to separate the more complex filter algorithms from the control dominated part. The FOSSY generated VHDL models consist out of 2231 (IDWT53) and 4225 (IDWT97) lines of code where all functions and procedures have been in-lined into a single explicit state machine. Since all identifiers are preserved during synthesis the resulting VHDL code remains human readable.

Table 2 shows the RTL synthesis results for a Xilinx Virtex-4 FPGA. For the IDWT53 the results are very similar. The area overhead induced by FOSSY is about 10%. Concerning the IDWT97 the FOSSY generated design is 15% smaller but 28% slower than the reference. Since the JPEG 2000 decoder is a HW/SW design where both the MicroBlaze and the OPB run with a frequency of 100 MHz the synthesis results perfectly match the timing requirements.

**5. Conclusion**

In this paper we presented the modelling and design of a JPEG 2000 decoder using the OSSS design methodology. The initial Application Model specifies the logical structure of the application, which identifies parallel components and their communication relations. The Application Model enables early functional and performance analysis in a rather abstract thus fast simulation model. Moreover, the designer can easily restructure the model, e.g. the hardware/software partitioning and assess the effect on the behaviour and estimated performance of the design.

The refinement process leading to the Virtual Target Architecture Model adds implementation details such as communication protocols and memories. This model exhibits cycle-accurate timing behaviour and is the input for the automatic synthesis process leading to the implementation on a specific target platform.

The comparison of the synthesis results using the FOSSY synthesis tool or a standard VHDL design process shows no significant difference in the efficiency in terms of area or timing.

The OSSS methodology supports many useful high-level design concepts such as method-based communication, hardware/software co-design and Shared Objects while maintaining a direct path to the implementation as a HW/SW SoC.

**References**