JPEG2000 image compression in Multi-Processor System on Chip

Delft University of Technology
11/03/2008 - 31/07/2008

Tutor:	BORNAT Yannick	ENSEIRB
Supervisor:	STEFANOV Todor	TU Delft
Author:	AZKARATE-ASKASUA Mikel	Electronique (SE)

(This page has been left blank on purpose.)

Contents

1 Introduction **1**
1.1 Motivation and Goals . . . 2
1.2 Contributions . . . 2
1.3 Related Work . . . 3
1.4 Thesis Organisation . . . 3

2 TU Delft **4**
2.1 Activities . . . 4
2.2 Organisation . . . 5
2.3 Research (Computer Engineering) . . . 6

3 Background **7**
3.1 Daedalus . . . 7
3.1.1 KPNGen: Application Parallelization . . . 8
3.1.2 YML and PNRunner: High Level Simulation . . . 9
3.1.3 ESPAM: Hardware Platform Generation . . . 9
3.2 JPEG2000 . . . 10
3.2.1 Discrete Wavelet Transform . . . 11
3.2.2 T1 Arithmetic Enconding . . . 12

4 Requirements and Constrainst **14**
4.1 Constrainst . . . 14
4.2 Requirements . . . 14
4.3 Tools . . . 15

5 Development Steps **16**
5.1 Reference Code . . . 17
5.1.1 JasPer . . . 17
5.1.2 OpenJPEG . . . 17
5.1.3 Comparison . . . 18
5.2 Code Partitioning . . . 19
5.2.1 Program Organization . . . 19
5.2.2 Removing Global Shared Data . . . 20

5.2.3 Memory Optimization 21
5.2.4 Configurable Parameters 22
5.2.5 KPNGen 22
5.3 Simulation 23
5.4 Example Application 24
5.4.1 Platform 24
5.4.2 Mapping 25
5.5 Design Exploration 26
5.6 MP-SoC 26
5.6.1 Program FPGA version 26
5.6.2 Xilinx Platform Studio 28
5.6.3 Execution 29

6 Experiments and Results 30
6.1 Multiprocessing 30
6.2 Splitting 31
6.3 Optimizing 33
6.3.1 Tiling 33
6.3.2 Division into Components 34
6.3.3 Memory occupation 35
6.3.4 Wavelet in Lossless Compression 36
6.3.5 CrossBar Switch 36
6.3.6 Compiler Options 38
6.4 Best Result 38
6.5 3 T1 Encoders 39

7 Comparison 42
7.1 Comparison with Related Work 42
7.2 Projected Architectures 43
7.3 Worktime of Project Steps 44

8 Conclusion and Future Work 45

List of Figures

2.1 EWI faculty building 5
2.2 An overview of the organizational structure of TU Delft 5

3.1 The Daedalus Tool-Flow 7
3.2 The JPEG2000 algorithm 11
3.3 Wavelet multi-resolution Transform 11
3.4 Tier-1 Encoding process 12

5.1 Workflow description 16
5.2 Code Partition Example 19
5.3 Partition into Processes 20
5.4 Partition into Processes 20
5.5 Tile Structure 20
5.6 Codeblocks Structure 21
5.7 Passes Structure 21
5.8 Packets Structure 21
5.9 Char Optimized Tile Structure 21
5.10 types.h: Program Configurable Parameters 22
5.11 JPEG2000.c: Input file for KPNGen 23
5.12 JPEG2000.pla: Platform Specification 24
5.13 MP-SoC Example Platform 25
5.14 JPEG2000.map: Mapping Specification 25
5.15 Processes Mapped into the Example Platform 26
5.16 Original Raw Image Reading Versions 27
5.17 Original Raw Image Writing Versions 27
5.18 Xilinx Platform Studio 28
5.19 J2K MP-SoC Execution 29

6.1 Clock cycles Vs. Number of processors 31
6.2 Computing time percentage taken by each process 32
6.3 Speed-up with parallel T1-encoders 32
6.4 Speed-up against number of T1 independent processes 33
6.5 Tiling effect in size and time 34

6.6 Final JK2 File and Tile Header Overload 34
6.7 Tile Component Structure 35
6.8 Improved JPEG2000.c with Components 35
6.9 Crossbar Switch 37
6.10 Part to be added in the platform specifications 37
6.11 Best MP-SoC platform 39
6.12 JPEG2000.c File for Best Solution 40

7.1 Memory Occupation Formules 43
7.2 Computation Time Formules 43

List of Tables

4.1 Virtex II features 15

5.1 JasPer Vs OpenJPEG comparison table 18
5.2 Executions times with “time” command 18
5.3 Code lines of the source files 18
5.4 Structures final size for 32x32 pixels tiles 22

6.1 Mapping for the Increasing Processors 30
6.2 Memory Occupation Processes 36
6.3 Crossbar Vs Point to point 38
6.4 Compiler optimization improvement 38
6.5 Platform, Mapping and Section sizes for Best Result 41
6.6 Implementation information 41

7.1 Comparison with Commercial tools 42
7.2 Proposed Architectures 43
7.3 Time periods through the workflow 44

Acknowledgments

Thanks to...

...the TU Delft and Chess B.V. for giving me the chance of doing this thesis and specially to Todor Stefanov for his implication in the project and his unconditional support.

Merci à...

...l'ENSEIRB ainsi comme au programme ERASMUS pour me donner la opportunité de vivre cette extraordinaire expérience Européenne. Je voudrais aussi remercier à Yannick Bornat son travail comme tuteur pendant le projet.

Gracias a...

...mis amigos Holandeses que si bien nacieron más al sur serán mi mejor recuerdo de este periodo, especialmente a mi compañera de despacho por aguantarme todos los días con una sonrisa.

Eta azkenik, eskerrik asko...

...guraso, arreba eta betiko lagunei momentu txar zein onetan alboan izan direlako nahiz eta ehundaka kilometrotara egon.

Chapter 1

Introduction

The complexity of modern embedded multimedia systems, which are increasingly based on heterogeneous MultiProcessor System on Chip (MP-SoC), has led to the emergence of system level design. The use of MP-SoC allows the efficiency of multiprocessor even using parallelism between tasks and the flexibility of using software applications instead of hardware. System level design for MP-SoC based embedded systems however still involves a substantial number of challenging task. For example, applications need to be discomposed into paralell specifications so that they can be mapped onto a MP-SoC architecture. Subsequently, applications need to be partitioned into HW and SW parts since MP-SoC architectures often are heterogeneous (hardware and software) in nature.

To cop with this design complexity, system-level design aims at raising the abstraction level of the design process, for example, with the use of architectural platforms to facilitate re-use of IP components and the notion of high-level modeling and simulation. The latter allows for capturing the behavior of platform components and their interactions at a high level of abstraction. As such, these high-level models minimize the modeling effort and are optimized for execution speed, and can therefore be applied during the very early design stages.

Several Dutch universities work in this System Level Design direction inside the Artemisia project creating the Daedalus framework to address these system-level design challenges.[1]

The main challenges of this MP-SoC design tool are:

1. The partition of the application into concurrent processes.
2. The simulation of this processes in a high level of abstraction.
3. The Multi-Processor platform generation.
4. The mapping of the application processes onto the multiprocessor platform.
5. The fast modification of the application on response to user requirements or bugs.

The Leiden Institute of Advanced Computer Science (LIACS) in the Netherlands deals with the first and the third points, using Kahn Process Networks to schedule parallel tasks and developping an own solution to synthesize custom platforms to run those processes. In the same way, Amsterdam University focuses on simulators which enable to test the application from the beginning and could give the best platform mapping solution after a design space exploration of the application. All these efforts end up in the Daedalus open-source tool-flow that nowadays is being tested in the Delft University of Technology (TU Delft).

The current version of the Daedalus framework make it suitable preferably for multimedia streamings as image or video compression. Multimedia applications provide a pipeline structure where each stage of the algorithm is identified as a task and parallelism of these tasks could be used to achieve time-computation requirements. JPEG and M-JPEG compression examples have been generated during the previous stages of the Daedalus developing process.

1.1 Motivation and Goals

Until the beginning of this project, the Daedalus toolchain has only been used into university frame and there was a big interest from the developers to validate it from a commercial point of view. The potential of the tool resides in the fast prototyping of the MP-SoC. The fact of evaluating multiple configurations in terms of hours make Daedadus attractive for companies which work with multimedia systems and want to evaluate, for example, different performance-cost solutions for a given application.

This is the case of the Dutch SME Chess B.V. in Harlem, company that become part of the Artemisia project and collaborate with the above mentioned universities and which involves the design of an image compression system for very high resolution (in the order of Gigapixels) cameras targeting medical appliances. They propose to take JPEG2000 image compression standard as validation example to satisfy their customers in the medical sector.

Therefore the main goals of the project are:

- Evaluate the commercial potential of the Daedalus flow. This evaluation covers the robustness of the present tools as well as the learning time and prototyping delay for a fresh new user.
- Generate a MP-SoC JPEG2000 application using Daedalus. In addition to developping a new example for Daedalus this JPEG2000 MP-SoC will be compared to present commercial implementations and confirm its validity.

1.2 Contributions

In this thesis we present the JPEG2000 MP-SoC and its developing process using Daedalus as system level design tool. The main contributions are:

- Generation of a flexible JPEG2000 MP-SoC application obtained from free open-source tools.

- Implementation of an optimized MP-SoC, using software optimization and hardware parallelism.
- Comparison of commercial hardware IPs and the generated optimazed solution.

1.3 Related Work

Systematic and automated application-to-architecture mapping has been widely studied in the research community. The closest to our work is the Kosku MP-SoC design flow and the SystemC-based design methodology presented in Friedrich Alexander University (Germany). Koski provides a single infrastructure for modeling of applications, automatic architectural design space exploration and automatic system-level synthesis, programming and prototyping of selected MP-SoCs [6]. The second methodology supports automated design space exploration performance evaluation and automatic parallelization of applications, not design space exploration level at application level [10]. Both tools require applications to be specified by hand in UML and SystemC, respectively. Companies such as Xilinx and Altera provide design tool chains attempting to generate efficient implementations starting from descriptions higher than (but still related to) the register transfer level of abstraction. The required input specifications are still so detailed that designing a single processor system is still error-prone and time consuming.

In the side of the JPEG2000 hardware commercial tools three companies have been found with IPs that fully covers the image compression algorithm: Analog Devices ADV212 [3], Barco Silex [12] and Cast [2]. All of them offer high data compressing speed as well as configuration options that are limited to standard image features.

1.4 Thesis Organisation

The remaining parts of this thesis are organised as follows:

Chapter 2 gives an overview of the Delft University of Technology, room where the project has took place.

In Chapter 3 the main points of the project are shown: a detailed explanation of the Daedalus tool-flow and the bases of the JPEG2000 image compression.

Chapter 4 and Chapter 5 present the requirements and tools to challenge the work and the way the project has been developed.

Chapter 6 as Chapter 7 presents the results of the experiments realised during the project in order to achieve the technical requirements and the best implemented solution is presented is compared with other related work.

In the final Chapter 8 the objectives above are overviewed to give the conclusions of the current work and the bases of the future works in this thesis line.

Chapter 2

TU Delft

Founded in 1842, Delft University of Technology is the oldest, largest, and most comprehensive technical university in the Netherlands. With over 13,000 students and 2,100 scientists (including 200 professors), it is an establishment of both national importance and significant international standing.

Renowned for its high standard of education and research, TU Delft collaborates with other educational establishments and research institutes, both within and outside of the Netherlands. It also enjoys partnerships with governments, trade organizations, numerous consultancies, industry and small and medium sized enterprises.

2.1 Activities

The TU is divided in 8 faculties:

- **3mE**: Mechanical, Maritime and Materials Engineering
- **BK**: Architecture
- **CiTG**: Civil Engineering and Geosciences
- **EWI**: Electrical Engineering, Mathematics and Computer Science (EEMCS)
- **IO**: Industrial Design Engineering
- **LR**: Aerospace Engineering
- **TBM**: Technology, Policy and Management
- **TNW**: Applied Sciences

Figure 2.1: EWI faculty building

The faculty where the project has been developped is the EEMCS (EWI in Dutch), more precisely the Computer Engineering department. The TU Delft Computer Engineering deals with the architecture, design, development, testing, and evaluation of hardware and software for computers and networks.

2.2 Organisation

Figure 2.2 below shows the TU Delft way of organisation:

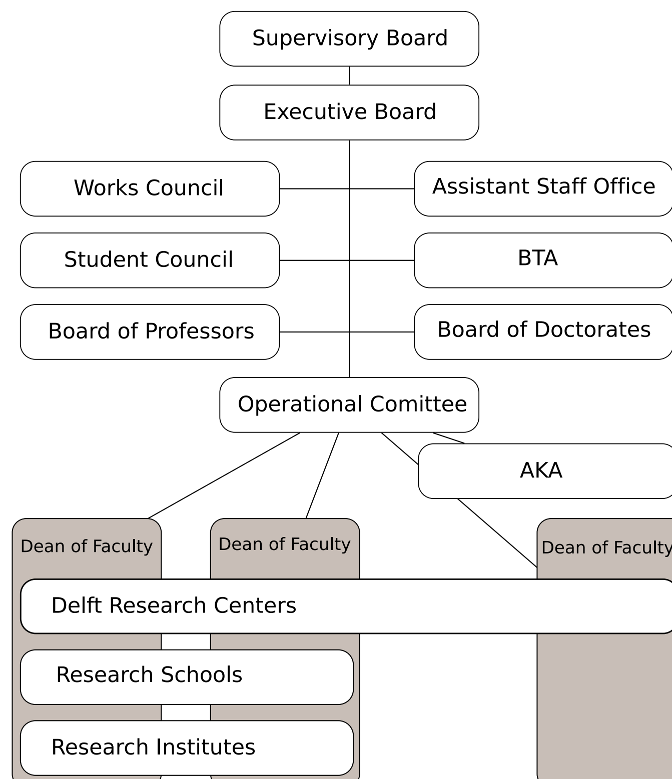


Figure 2.2: An overview of the organizational structure of TU Delft

2.3 Research (Computer Engineering)

The general research topics of the TU Delft computer engineering include computer hardware, software, and networks. More specifically the computer engineering research focuses in the following areas:

- **Hardware:** computer architecture, microarchitectures, digital design, parallel vector and media processors, embedded processors, SoCs, VLSI design, computer arithmetic, low power designs, reconfigurable processors, feed forward neural networks (threshold logic), memory and logic testing, design for testability.
- **Software:** backend compilers, system software, software for automatic synthesis, performance and software tools, hardware software co-design, software simulators, code instrumentation and performance enhancement tolls, design space exploration software for computer architectures and machine organizations, placement and routing algorithms, physical design, binary translators.
- **Networks:** computer architecture for Network processors, interconnection networks, internet and web processing, mixed optical/electronic switches, distributed processing, ubiquitous (i.e., anywhere and anytime) and unobtrusive (i.e., without much user intervention) communication environments.
- **Speculative research:** nano computing, chaotic computational systems, threshold logic processors, non conventional computer architectures, interacting migrating processes.

Chapter 3

Background

In this chapter the main points of the project are presented: on the one hand the high level MP-SoC design tool Daedalus and on the other hand the JPEG2000 compression application.

3.1 Daedalus

The Daedalus framework is the result of the work between the Leiden Institute of Advanced Computer Science and the Amsterdam University. Its main objective is to bridge the implementation gap between the system level description to the RTL implementation for the design of multimedia MP-SoCs.

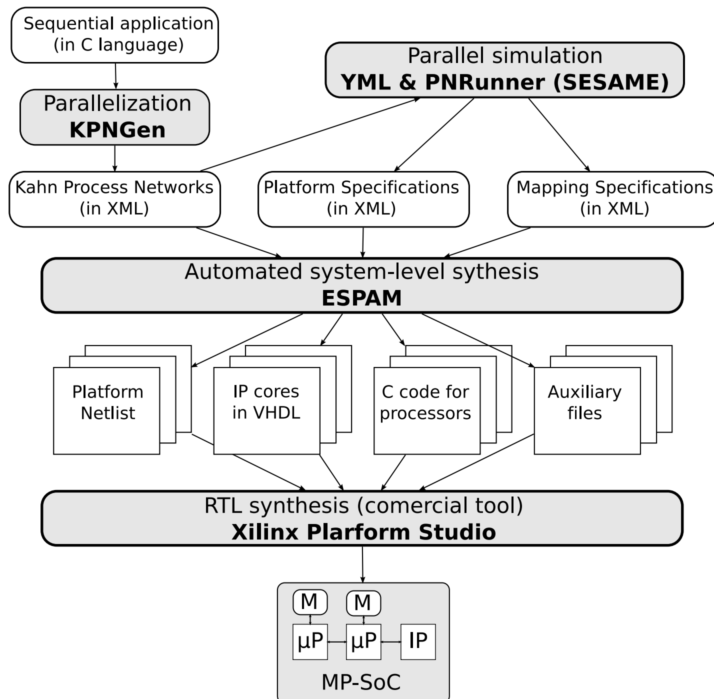


Figure 3.1: The Daedalus Tool-Flow

MP-SoC designs need both software and hardware development. The software must be partitioned in order to enable parallel computation and the hardware must be able to execute these computations. The Daedalus toolchain starts with the software partition and follows with the hardware platform design until arriving to the MP-SoC implementation.

As illustrated in Figure 3.1 above, the Daedalus tool-flow has the application in C language as first input for the Kahn Process Network Generator (KPNGen). This C application should be written in a specific way as will be explained in Section 3.1.1.

Before any other step it is possible to validate the generated processes in a high level of abstraction running YML and PNRunner (See Section 3.1.2).

The validated tasks are mapped into the processors with the Embedded System-level Platform synthesis and Application Mapping (ESPAM) tool following the platform and mapping specifications. Using IP libraries and RTL models ESPAM realises the necessary RTL level files for commercial synthesis tools as Xilinx Platform Studio (XPS). This last tool will be treated in Section 3.1.3.

3.1.1 KPNGen: Application Parallelization

The applications are typically specified in sequential programs using languages as C or C++, what is relatively easy for program developers. But this sequential nature does not reveal parallelism and makes very difficult the mapping of the program in a multiprocessor platform.

By contrast, using parallel model of computation (MoC) the mapping becomes a transparent and systematic process. But parallel MoC is not natural for developers and this difficulty is time consuming. KPNGen fills this gap between the sequential program application to a parallel application specification using different compiler techniques.[11]

Process Networks

KPNGen uses the process network model of computation. This is a simple and streaming oriented model that fits with the multimedia domain of the whole Daedalus tool-flow. These process networks consist of a set of processes or nodes which communicate each other through channels. The processes are auto-scheduled by the communication channels, therefore a process will be blocked if it needs data from a channel that is not available yet. In the same way, it will block if it tries to write in a full channel.

In the specific case of the Kahn Process Networks (KPN) the channels are unbounded FIFOs which blocks in read. To implement this “unbounded” FIFOs, the maximal size of the FIFO must be calculated offline in order to avoid deadlocks when a process wants to write and there is no space. The KPNs are also deterministic, the output is always the same for a given input, and this behaviour assures the well operational behaviour of the application in any case of scheduling.

Static Affine Nested Loop Programs

The KPNGen tool generates the process networks by means of a technically called static affine nested loop programs (SANLP). SANLPs are important in scientific or matrix computation and multimedia or adaptive signal processing applications.

They consist of a set of statements enclosed in loops or/and guarded by conditions, but all lower and upper bounds of the loops as well as all condition expressions must be known at compilation time and these conditions must be affine with means linear mathematical expressions, not power expressions for example.

3.1.2 YML and PNRunner: High Level Simulation

YML and PNRunner are both part of SESAME, a project of the Amsterdam University. The SESAME environment provides modeling, simulation methods and tools for the efficient design space exploration of heterogeneous embedded multimedia systems. The SESAME software project currently consists of an architecture simulator (YMLPearl), an application simulator (PNRunner) and the Y-CHART modeling language (YML) which glues the simulation model descriptions together.

YML makes a C++ object oriented program from the KPNGen output process network specification. This program executes every process independently, as it will be in the FPGA. FIFOs read and write management is also included in the program.

3.1.3 ESPAM: Hardware Platform Generation

ESPAM is developed to allow system designers to specify a system and its related applications at a higher level of abstraction (System Level) to save design effort and time. This tool uses these system-level input specifications, together with RTL versions of the components from the IP library, to automatically generate synthesizable VHDL that implements the candidate MP-SoC platform architecture. In addition, it also generates the C code for those application processes that are mapped onto programmable cores. Using commercial synthesis tools and compilers, this implementation can be readily mapped onto an FPGA for prototyping. Such prototyping also allows for calibrating and validating Sesame's system-level models, and as a consequence, improving the trustworthiness of these models [11].

The ESPAM tool input are three specification files in XML language:

- **Application Specification:** the Kahn Process Network specification generates by the KPNGen. It gives information about the different processes and the communication between them.
- **Platform Specification:** defines the hardware platform (processors, peripheral, etc) of the MP-SoC.
- **Mapping Specification:** maps the processes into the desired processor.

3.2 JPEG2000

JPEG2000 is a wavelet-based image compression standard created by the Joint Photographic Expert Group (JPEG) committee in 2000 with the aim of replacing the original discrete cosine transform-based JPEG.

Only the first of the 13 parts of the standard is free of royalties which has prevented the standard expansion for example in the internet. This part specifies the core and minimal functionality and is known as JPEG2000 codec.[5]

The main advantages of JPEG2000 against the classical JPEG are:

- Superior compression performance: specially at low bitrate (e.g. less that 0.25 bits/pixel).
- Multiple resolution representation.
- Progressive transmission: after a smaller part of the whole file has been received, the viewer can see a lower quality version of the final picture.
- Lossless and lossy compression.
- Random codestream access and processing: different grades of compression could be given to some Regions Of Interest (ROI) of the image.
- Error resilience: small independent block avoid error propagation.

The JPEG2000 algorithm flow shown in Figure 3.2 reveals the first and simple stages: The whole raw image in divided in the three Red-Green-Blue (RGB) components. Each component is divided in equal smaller pieces called tiles, which are coded independently. The RGB components are transformed in YUV model that requires less memory. The last stages: Discrete Wavelet Transform (DWT) and T1 Arithmetic Enconding are explained more carefully in Sections 3.2.1 and 3.2.2.

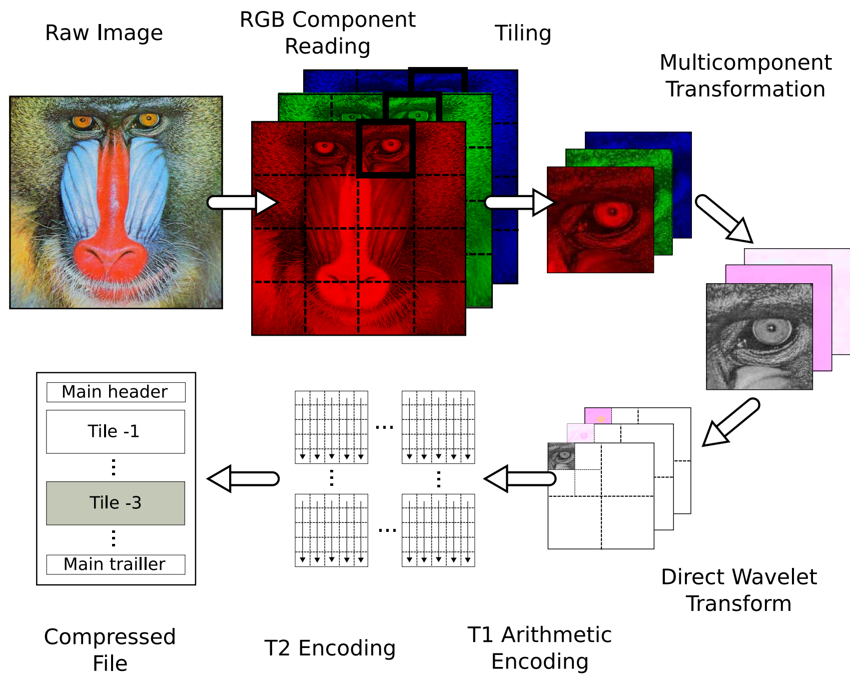


Figure 3.2: The JPEG2000 algorithm

3.2.1 Discrete Wavelet Transform

The Discrete Wavelet Transform is generally understood as tree-structure subband transform with the multi-resolution structure. In contrast to the Discrete Cosines Transform (DCT) used in the classical JPEG and other image compression standards, the image is processed continuously and not in blocks that afterwards improve the compressed image quality for the same level of quantization.

There are two main wavelet types:

- **Irreversible**: the CDF 9/7 wavelet transform, lossless compression with this type of wavelet is impossible because the quantization noise introduced by the precision of the decoder.
- **Reversible**: a rounded version of the bi-orthogonal CDF 5/3 wavelet transform, the one used for lossless compression where only integer coefficients are used and the output does not need rounding (there is not quantization noise).

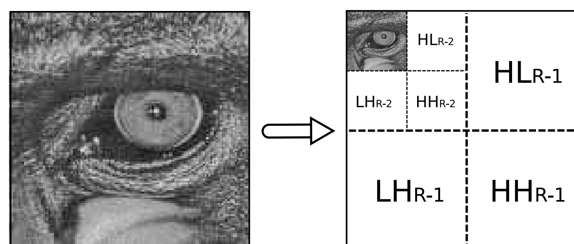


Figure 3.3: Wavelet multi-resolution Transform

Figure 3.3 shows the multi-resolution behaviour of the wavelet transform. Human eye is much more sensitive to slow luminance changes than to the fast ones. Wavelet transform nature divide every image in four bands taking the low frequencies band (LL) as a new interpolated lower resolution image.

3.2.2 T1 Arithmetic Enconding

After DWT is performed, Tier-1 coding takes place. The arithmetic encoding removes redundant data improving the size of the information. The quantizer indices for each subband are partitioned into codeblocks and each of the codeblocks is independently coded. The coding is performed using the bit-plane coder.

For each codeblock, an embedded code is produced, comprised of numerous coding passes. The output of the Tier-1 encoding process is, therefore, a collection of coding passes for the various codeblocks. In case of lossy compression only some of this coding passes are included, in contrast, in lossless compression every pass is included in the final code stream.

There are 3 types of coding passes:

1. **Significance:** Is the first coding pass for each bit plane. The main information of the image is obtained in this pass.

2. **Refinement:** The second coding pass for each bit plane is the refinement pass. This pass signals subsequent bits after the most significant bit for each sample.

3. **Clean Up:** The third (and final) coding pass for each bit plane is the cleanup pass. This pass is used to group the rest of bits that contains less significant information.

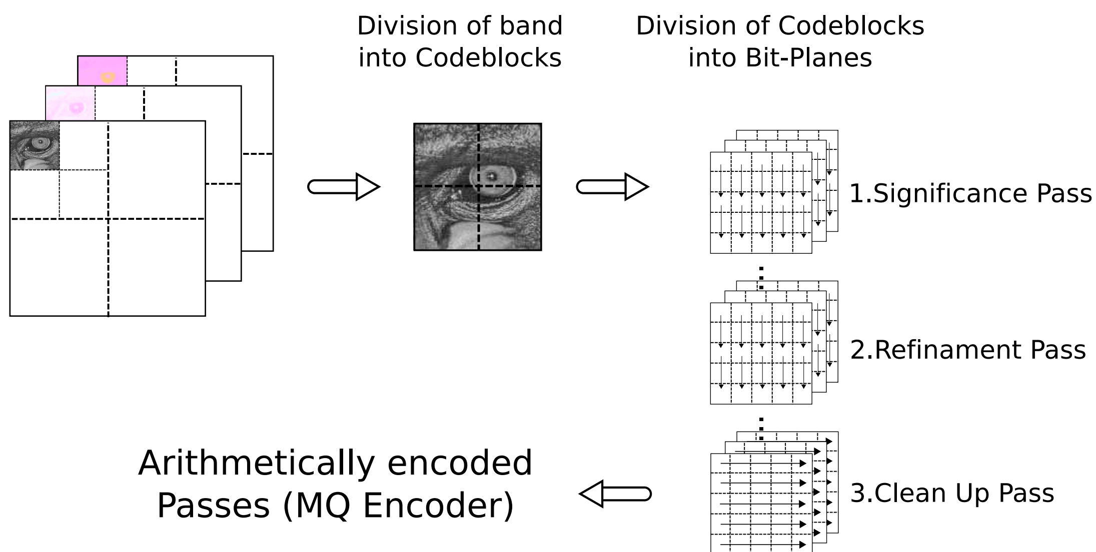


Figure 3.4: Tier-1 Encoding process

Figure 3.4 illustrated the division of each resolution into codeblocks and how in each cases codeblocks are traversed bit per bit in different directions in order to compute specific passes. After the coding passes the data is arithmetically coded using MQ-Encoder.

Chapter 4

Requirements and Constrainst

The technical constrainst are some of them shortly tied to the used Daedalus tools and some others requirements imposed by Chess B.V. company's clients. These features as well as the other tools beyond Daedalus are mentioned in the subsections below.

4.1 Constrainst

The fact of using Daedalus carries the following constrainst:

- The reference code must be written in C and it should be oriented to be divisible in sub-processes.
- Two versions of the reference code must be writen. As mentioned in the previous section there is a high level of abstraction where the code is executed in a PC and an embedded environment (the FPGA) where the way of obtaining the image or writing the result is not the same.

Also using a FPGA to synthesize the hardware platform have the specifications below:

- Independent processes of DWT and T1 arithmetic encoding must be developed for a possible fast implementation of hardware accelerators substituting this software processes by commercial IPs.
- Images have to be computed in parts due to memory limitations of the FPGA card therefore this tiling option must be implemented.

4.2 Requirements

Chess B.V., as commercial partner, has oriented the project to possible medical application that have these requirements:

- Lossless compression for medical certification reasons.
- RGB raw image as input.
- Measure of work time for each task through the workflow.

4.3 Tools

Several software and hardware tools have been used during this project. On the one hand programs which facilitate the development of the code:

- Eclipse IDE 3.2 environment.
- CVS 1.12.13 version repository.
- GDB 6.8 debugger.
- IrfanView 4.10, JPEG2000 image viewer.
- Meld 1.1.5.1 difference viewer.
- Xilinx Platform Studio 9.1.

On the other hand, the Virtex II (xcv6000) FPGA has been the most significant hardware resource, a PC-slot version accessible remotely from the internet. Its main features are:

	Gates	**Slices**	**On-chip BRAM**	**Off-chip RAM**
xcv6000	6M	33792	288kB	6 x 256KB

Table 4.1: Virtex II features

Also in the hardware part, the is a list of used Xilinx IPs in the Daedalus library:

- **Processor**: Microblaze 4.00a.
- **Bus**: lmb_v10 and opb_v20 (connection between processors and off-chip memory controllers).
- **FIFO**: fsl_v20 (shared memory between processes).
- **Memory**: bram_block 1.00a (code and data memories).

Chapter 5

Development Steps

In this chapter the development stages of the project are described. From the reference program selection, through the code partition and simulation to an example application that shows the potential of the tool once the program is made and working.

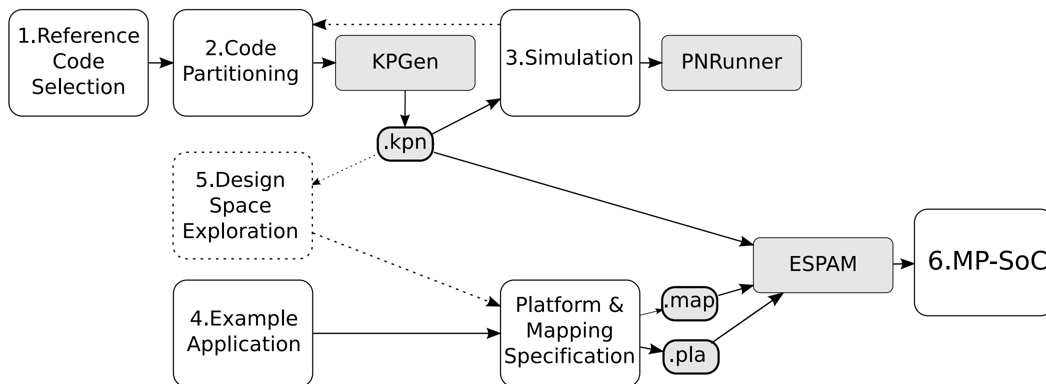


Figure 5.1: Workflow description

In Figure 5.1 the workflow is displayed where six development steps have been defined:

1. **Reference Code Selection:** An open-source JPEG2000 application written in C language is chosen following Daedalus toolchain constraints.

2. **Code Partitioning:** The reference code is divided in modules or processes that allows the automatic generation of process network through KPNGen tool.

3. **Simulation:** The generated process networks are tested in a high level of abstraction, if any modification is necessary at this step, the partitioned code should be rewritten and KPNs must be regenerated.

4. **Example Application:** As piece of example, a simple example configuration is shown, that means the number of processors, how the processes are mapped

onto them and the way of data connections. This example application allows the explanation of the next steps of the workflow. Once the application configuration is chosen, this step illustrates how to write the platform and mapping specification for Daedalus.

5. **Design Space Exploration:** This step shows how to obtain the best configurations for a given application using high level simulation. Mapping and platform specifications could be automatically generated.

6. **MP-SoC:** Finally, it is explained how the application and specifications are sythetized using ESPAM.

Therefore this chapter could be also understood as a tutorial of how to develop a project using the Daedalus toolchain.

5.1 Reference Code

The starting point of the toolflow is the sequential program. As input of KPNGen tool, the application must be written in C language. The other requirement is to be open-source in order to be in line with Daedalus which is also freeware. There has been found two open source JPEG2000 libraries implemented in C language (as required) Jasper and OpenJPEG.

5.1.1 JasPer

The JasPer Project is an open-source initiative to provide a free software-based reference implementation of the codec specified in the JPEG-2000 Part-1 standard (i.e., ISO/IEC 15444-1).

This project was started as a collaborative effort between Image Power, Inc. and the University of British Columbia. Presently, the ongoing maintenance and development of the JasPer software is being coordinated by its principal author, Michael Adams, who is affiliated with the Digital Signal Processing Group (DSPG) in the Department of Electrical and Computer Engineering at the University of Victoria. The code is ready for download in: http://www.ece.uvic.ca/~mdadams/jasper/.

5.1.2 OpenJPEG

The OpenJPEG library is an open-source JPEG2000 codec written in C language. In addition to the basic codec, various other features are under development, among them the JP2 and MJ2 (Motion JPEG2000) file formats, an indexing tool useful for the JPIP protocol, JPWL-tools for error-resilience and a Java-viewer for j2k-images.

The library is developed by the Communications and Remote Sensing Lab in the Université Catholique de Louvain (UCL). The JPWL module is developed and maintained by the Digital Signal Processing Lab (DSPLab) of the University of Perugia, Italy (UNIPG)[9]. The application could be downloaded from: http://www.openjpeg.org/.

5.1.3 Comparison

Here the main characteristics of each of selected programs:

	JasPer	**OpenJPEG**
Licence	Open Source (MIT)	Open Source (BSD)
Documentation	PDF Files	Web and Doxygen
Support	Mail-list	Forum
Last Version	2007 (1.900)	21/12/2007 (1.3)

Table 5.1: JasPer Vs OpenJPEG comparison table

As listed in the in Table 5.1, there are not many differences in terms of licence, documentation (going just through the API in both cases) and version support. That is why more detailed comparisons have been made.

During this step, comparisons focus in two main challenges of the work. First, the compression time taken by the programs. The execution times for each program[1], in a Pentium-4 2Ghz for a 22,9MB BMP (4056x1974) picture, are shown in Table 5.2:

	JasPer	**OpenJPEG**
Real Time	17s	40s
Encondig Time	not provided	21s

Table 5.2: Executions times with “time” command

Second, in order to simplify the code partition that will be treated in Section 5.2.1, the lines of the source code have been counted in Table 5.3:

	JasPer	**OpenJPEG**
Sources	40k lines	20k lines

Table 5.3: Code lines of the source files

This last point, which has been followed with a deeper examination of the codes, has been determinant to choose OpenJPEG as the reference code of the project. The examination shows that in OpenJPEG the different stages of the JPEG2000 algorithm are more independent and therefore easier to divide. As well, some memory managements and screen printing could explain the compression time differences against Jasper. From now on this OpenJPEG library will be called “reference code”.

[1]OpenJPEG gives the Encoding Time as output

5.2 Code Partitioning

The input of the KPNGen tool, as mentioned in Section 3.1.1, is a static affine nested loop C language program. This C program, must be organized in independent processes which communicate through FIFOs and should be as small and optimized as possible to fit in the embedded platform.

5.2.1 Program Organization

Firstly, data flow has been identified and unnecessary files have been removed. As mentioned in the previous section, the different processes are clearly shown in the reference code, these processes are exactly the ones of the classical JPEG2000 algorithm: MultiComponent Transform (MCT), Discrete Wavelet Transform (DWT), T1 encoding (T1) and T2 encoding (T2). Moreover, this division allows the future replacing of this specific software processes for hardware IPs.

The first partition just previews an independent .C file for each process and deletes any file that is not necessary for the compression program as listed in Figure 5.2.

```
/* Reference Code */
codec/
|-- Makefile
|-- compat
|   |-- getopt.c
|   `-- getopt.h
|-- convert.c
|-- convert.h
|-- dirent.h
|-- image_to_j2k.c
|-- j2k_to_image
`-- j2k_to_image.c
libopenjpeg/
|-- CMakeLists.txt
|-- bio.c
|-- bio.h
|-- cio.c
|-- cio.h
|-- ... + 60 files more!
```

```
/* Partitioned Code */

jpeg2000/
|-- DWT.c
|-- DWT.h
|-- MCT.c
|-- MCT.h
|-- Makefile
|-- T1.c
|-- T1.h
|-- T2.c
|-- T2.h
|-- VideoIn.c
|-- VideoIn.h
|-- VideoOut.c
|-- VideoOut.h
|-- jpeg2000.c
|-- jpeg2000_func.h
`-- types.h
```

Figure 5.2: Code Partition Example

Two new processes, VideoIn and VideoOut have been created in order to read the original image and write the compressed file (Figure 5.3):

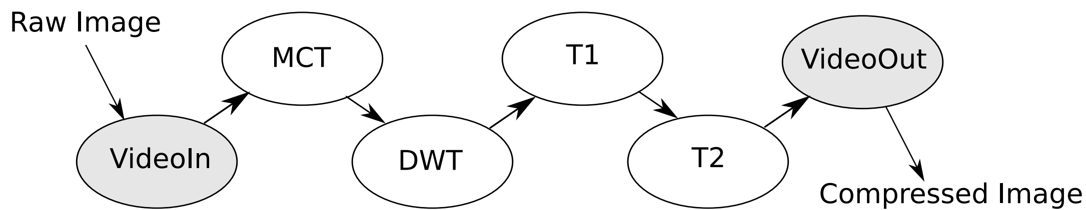


Figure 5.3: Partition into Processes

At this state, only the required functionality (mentioned on Chapter 4) has been left, that means, lossless compression (reversible wavelet), not floating point operations, only support for RGB format images, etc. In this way code size is optimized for the embedded version of the application.

5.2.2 Removing Global Shared Data

When partitioning, every part or process accesses to the same shared data memory. In the case of the reference code this share data is huge, impossible to store in a embedded platform and even more, each process only needs a small part of this data structure.

Therefore, specific data structures have been developped in order to contain only the minimal data to be share between given processes. These structures will become the FIFOs that Kahn Process Network define for interprocess communications 5.4.

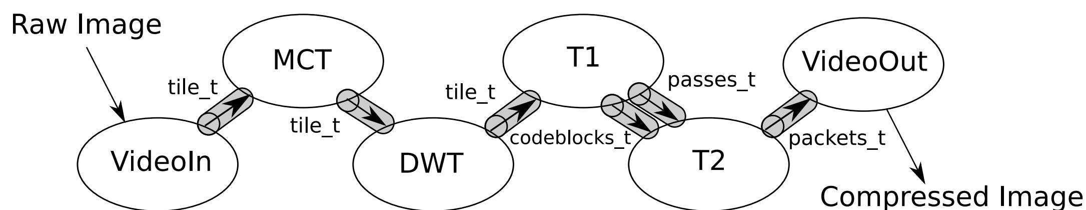


Figure 5.4: Partition into Processes

- **Tile**: This structure stores the 3 components of the original RGB image in separate tables. As all the structures below, it has the number of the tile in computation. In the same way it is the output of the MCT process which gives as well 3 components (YUV) and the result of the computed DWT that being reversible (CDF 5/3) is also made by integer coefficients.

```
typedef struct tile_t {
   int comp[NUM_COMPS][WIDTH_TILE*HEIGHT_TILE];
   int num;
} tile_t;
```

Figure 5.5: Tile Structure

- **Codeblocks**: After the T1 encoding the image information is presented as codeblocks. If the wavelet is computed, the resulting resolutions and bands are stored independently.

```
typedef struct codeblock_t {
   unsigned char data[NUM_RES][NUM_BANDS][NUM_PRECS][NUM_CODEBS][CBLKW*CBLKH];
   int num;
} codeblock_t;
```

Figure 5.6: Codeblocks Structure

- **Passes**: During the T1 arithmetic coding, the information is not only stored in codeblocks, the *passes* structure stores the number of passes for each codeblock in order to make possible the reconstruction of the image.

```
typedef struct passes_t{
   pass_t passes[NUM_RES][NUM_BANDS][NUM_PRECS][NUM_CODEBS][MAX_PASSES];
   int passnum[NUM_RES][NUM_BANDS][NUM_PRECS][NUM_CODEBS];
   int numbps[NUM_RES][NUM_BANDS][NUM_PRECS][NUM_CODEBS];
   double distotile;
} passes_t;
```

Figure 5.7: Passes Structure

- **Packets**: Finally, the T2 encoding presents the packets that complete the compressed file. The length of each packet is also passed.

```
typedef struct packet_t {
   unsigned char data[WIDTH_TILE*HEIGHT_TILE];
   int length;
   int num;
} packet_t;
```

Figure 5.8: Packets Structure

In the original data structure of the reference code, there is also stored internal data that has been reallocated in new internal structures. With these modifications, the Kahn Process Networks are generated automatically and each structure become a FIFO whose size is fixed which are called static FIFOs.

5.2.3 Memory Optimization

The first versions of the partitioned code were too big in terms of the FIFO sizes between processes when the implementation limit of each FIFO is limited to 8KB. Changing integer tiles to char tiles has been one of the taken solutions, reducing the size of the structures 4 times:

```
typedef struct tile_t {
#ifdef INTEGER_TILES
   int comp[NUM_COMPS][WIDTH_TILE*HEIGHT_TILE];
#else
   char comp[NUM_COMPS][WIDTH_TILE*HEIGHT_TILE];
#endif
   int num;
} tile_t;
```

Figure 5.9: Char Optimized Tile Structure

The table below shows the final structures size for a 32x32 pixels char tile:

Structure	Size
tile_t	3076 bytes
codeblocks_t	1032 bytes
passes_t	456 bytes
packets_t	1036 bytes

Table 5.4: Structures final size for 32x32 pixels tiles

5.2.4 Configurable Parameters

After program modifications, the configuration parameters are given in the types.h header file (and not through the command line as in the reference code). In this header file, the size of the original image must be given as well as the number of tiles the image should be partitioned or the number of resolutions (wavelet computations). Figure 5.10 shows the beginning of this file and the multiple configurable options.

```
...
1  //Horizontal tiles
   #define HTILES 4
   //Vertical tiles
   #define VTILES 4
5  // Name of the image
   #define FILE_NAME "baboon_128x128"
   // Size of the image
   #define WIDTH_IMAGE  128
   #define HEIGHT_IMAGE 128
10 // Size of the tile
   #define WIDTH_TILE  WIDTH_IMAGE / HTILES
   #define HEIGHT_TILE HEIGHT_IMAGE / VTILES
   //Tiles with 'int's instead of 'char's
   //#define INTEGER_TILES
15 // Number of components
   #define NUM_COMPS  3
   // Number of resolutions
   #define NUM_RES 1
...
```

Figure 5.10: types.h: Program Configurable Parameters

5.2.5 KPNGen

Once the code is partitioned and optimized, it is possible to pass it through the Kahn Process Network generator. Figure 5.11 is the main program of the static affine nested program and the input to the KPNGen.

```
1  #include "jpeg2000_func.h"

   int main (int argc, char **argv)
   {
5     int i;
      tile_t tile;
      codeblock_t codeblocks;
      passes_t passes;
      packet_t packets;
10
      for (i = 0; i < NUMTILES; i++)
      {
            mainVideoIn(&tile);
            mainMCT(&tile,&tilec);
15          mainDWT(&tilec,&tilec);
            mainT1(&tilec,&codeblocks,&passes);
            mainT2(&codeblocks,&passes,&packets);
            mainVideoOut(&packets);
      }
20
      return 0;
   }
```

Figure 5.11: JPEG2000.c: Input file for KPNGen

In the main function there is shown the previously mentioned (Section 3.1.1) static affine nested loop program. As can be seen, there is a loop of an affine scalar function (the number of tiles is linear, line 11) and the bounds of the loop as well as the structure sizes are static (do not change during the execution) and they are known at compilation time.

The output of the KPNGen tool is a ".kpn" file where all the necessary data to make the process networks run as process function calls or associated communication structures are stored.

5.3 Simulation

Taking the generated Kahn Process Network file, it is possible to simulate the parallel execution conditions. As explained in Section 3.1.2 the SESAME project previews a simulator, PNRunner, for this purpose.

PNRunner runs each process independently and it is possible to detect errors that otherwise will be found after the synthesis of the MP-SoC. At this stage, the simulator executes the network process model in the same way than embedded processor will do in MP-SoC, therefore, it is possible to predict possible deadlocks or simply bugs that work in the sequential model but do not respect the parallel model of computation. The only difference between this simulation and the real FPGA execution is that the hardware constraints as memory size limitations or

memory accesses are not verified.

This early simulation accelerates the prototyping delay, for any modification only the partitioned code must be modified and the new Kahn Process Network file is regenerated in one step.

5.4 Example Application

For a first approach, a simple example is proposed: 3 MicroBlaze processors with two processes mapped in each. The platform (.pla) and the mapping (.map) specifications are given using XML files with specific tags that are explained in the sections below.

5.4.1 Platform

The platform file describes the number of processors, their type, their program/data memory sizes and their connections as well as other peripherals and links. Each XML tag is linked with the Daedalus IP library which implements the desired module.

```
1  <platform name="j2kPlatform">

   <processor name="MB_1" type="MB" data_memory="8192" program_memory="8192">
      <port name="OPB_1" type="OPBPort"/>
5  </processor>
   <processor name="MB_2" type="MB" data_memory="8192" program_memory="65536">
      <port name="OPB_2" type="OPBPort"/>
   </processor>
   <processor name="MB_3" type="MB" data_memory="16384" program_memory="65536">
10    <port name="OPB_3" type="OPBPort"/>
   </processor>

   <peripheral name="ZBT_CTRL_1" type="ZBTCTRL" size="1000000">
      <port name="IO_1" type="OPBPort"/>
15 </peripheral>
   <peripheral name="ZBT_CTRL_2" type="ZBTCTRL" size="1000000">
      <port name="IO_2" type="OPBPort"/>
   </peripheral>
   <peripheral name="ZBT_CTRL_3" type="ZBTCTRL" size="1000000">
20     <port name="IO_3" type="OPBPort"/>
   </peripheral>

   <link name="mb_opb_1">
      <resource name="MB_1" port="OPB_1"/>
25    <resource name="ZBT_CTRL_1" port="IO_1"/>
   </link>
   <link name="mb_opb_2">
      <resource name="MB_2" port="OPB_2"/>
      <resource name="ZBT_CTRL_2" port="IO_2"/>
30 </link>
   <link name="mb_opb_3">
      <resource name="MB_3" port="OPB_3"/>
      <resource name="ZBT_CTRL_3" port="IO_3"/>
   </link>
35  </platform>
```

Figure 5.12: JPEG2000.pla: Platform Specification

Figure 5.12 shows the platform file of the example configuration where into the “platform” tag the desired number of processor or peripherals could be defined. The ports of these modules are connected via links. The external memory used to store the raw and compressed images for the FPGA is a Zero-Bus Turnaround ram memory (ZBT) and it can be connected through link with the desired processors.

The size of the data and program memories are calculated using some methods explained in Section 5.6.2. The Figure 5.13 below gives a visual idea of the implemented platform:

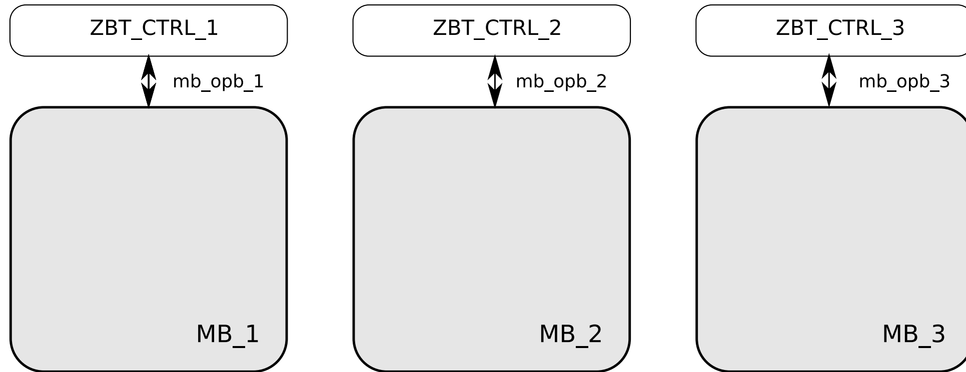


Figure 5.13: MP-SoC Example Platform

5.4.2 Mapping

For a given platform, the mapping file puts each process in the desired processor. For example in Figure 5.14 from line 3 to 6 processes ND_0 and ND_1 are mapped onto processor MB_1.

```
1  <mapping name="j2kMapping">

   <processor name="MB_1">
      <process name="ND_0" />
5     <process name="ND_1" />
   </processor>

   <processor name="MB_2">
      <process name="ND_2" />
10    <process name="ND_3" />
   </processor>

   <processor name="MB_3">
      <process name="ND_4" />
15    <process name="ND_5" />
   </processor>

   </mapping>
```

Figure 5.14: JPEG2000.map: Mapping Specification

As can be seen in Figure 5.15 the mapping specification fills the platform with the processes and makes possible different implementations for a single platform.

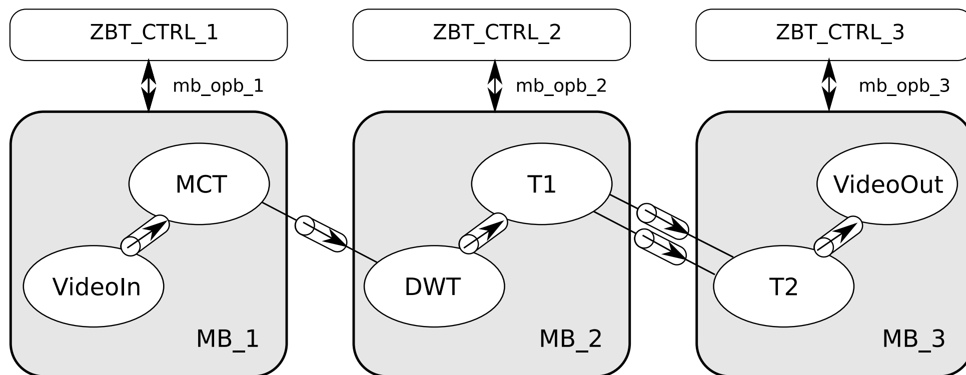


Figure 5.15: Processes Mapped into the Example Platform

5.5 Design Exploration

As mentioned in Chapter 3, with the SESAME tool of the Amsterdam University it is possible through high level simulation to obtain the best configuration of the system. This computation, which takes some hours, gives the best disposition of processes even splitting them for best timing results. Only the results that fit the hardware constraints of the FPGA can be implemented. [8]

Due to the limited time of the project this Desgin Space Exploration step has been done empirically. Through specific experiments that are shown in next chapter, the JPEG2000 application has been characterized. The fact of doing it manually allows to realize step by step of the different peculiarities of the application and apply specific solution to each problem.

5.6 MP-SoC

With the Kahn Process Network, Mapping and Platform specifications the ESPAM tool generates the necessary files to open a new Xilinx Platform Studio. But before final synthesis and test, there are some modifications that should be done in the partitioned program and in the ESPAM generated files. These issues will be treated in the following sections.

5.6.1 Program FPGA version

Now that the program needs to be executed in a FPGA, some functionalities differ from the high level partitioned version. The changes are related to the read and write of the raw and compressed images.

For reading, the high level version previews three files (".R", ".G" and ".B") which are read once in a static table for the rest of the execution. But in the FPGA there is not file management, therefore, the original raw image must be stored somehow in the FPGA memory and read from there as a table. How to store the image is explained in Section 5.6.3 and in Figure 5.16 the switch between the two readings ways is shown. This switch is made using the define "PC" (line 1).

```
/* VideoIn.h */
...
1  #ifdef PC
   static int Rtable[WIDTH_IMAGE * HEIGHT_IMAGE / 4];
   static int Gtable[WIDTH_IMAGE * HEIGHT_IMAGE / 4];
   static int Btable[WIDTH_IMAGE * HEIGHT_IMAGE / 4];
5  #else
   static volatile int *Rtable = ZBT_MEM;
   static volatile int *Gtable = (volatile int *) (ZBT_MEM + (WIDTH_IMAGE * HEIGHT_IMAGE / 4));
   static volatile int *Btable = (volatile int *) (ZBT_MEM + (2 * (WIDTH_IMAGE * HEIGHT_IMAGE / 4)));
   #endif
...
/* VideoIn.c */
...
1  #ifdef PC  // Read the image once
      if (num == 0)
      {
         char name[MAX_NAME];
5        strcpy(name,FILE_NAME);
         FILE *f;
         strcpy(name,FILE_NAME);
         strcat(name,".R");
         f = fopen(name,"rb");
10       fread(Rtable, 4, WIDTH_IMAGE * HEIGHT_IMAGE / 4, f);
         fclose(f);
         /* Made twice more for B and G components */
         ...
    }
   #endif
...
```

Figure 5.16: Original Raw Image Reading Versions

For the writing of the compressed file, a similar solution is taken (Figure 5.17). The high level version used to write a new file with the compressed image, for that, a table is filled until the last tile when the file is generated. In the FPGA version, this table is addressed to the FPGA memory, and the file is generated with external tools as will be explained in Section 5.6.3. The offset variable store the size of the compressed data to enable the file generation.

```
/* VideoOut.h */
...
1  #ifdef PC
   static FILE *f;
   static int outMemory[IMAGE_SIZE];
   #else
5  static volatile int *outMemory=(volatile int *)(ZBT_MEM +1);
   #endif
...
/* VideoOut.c */
...
1     if ((tileno == NUMTILES -1) && (compno == 2))
      {
   #ifdef PC
        f = fopen("test.j2k","wb");

5       for(k = 0; k < offset; k++)
        {
           putc(outMemory[k],f);
        }
        fclose(f);
10
   #else
        *ZBT_MEM = offset;
   #endif
      }
...
```

Figure 5.17: Original Raw Image Writing Versions

Finally, before opening the Xilinx Platform Studio project the ".MHS" (file where the hardware modules are described) must be changed to include the FIFO sizes for the example application.[13]

5.6.2 Xilinx Platform Studio

The Xilinx Platform Studio is the last tool from where the bitstream to configure the FPGA will be obtained. The project is opened through the ".XMP" file and some updates are demanded for the last XPS version.

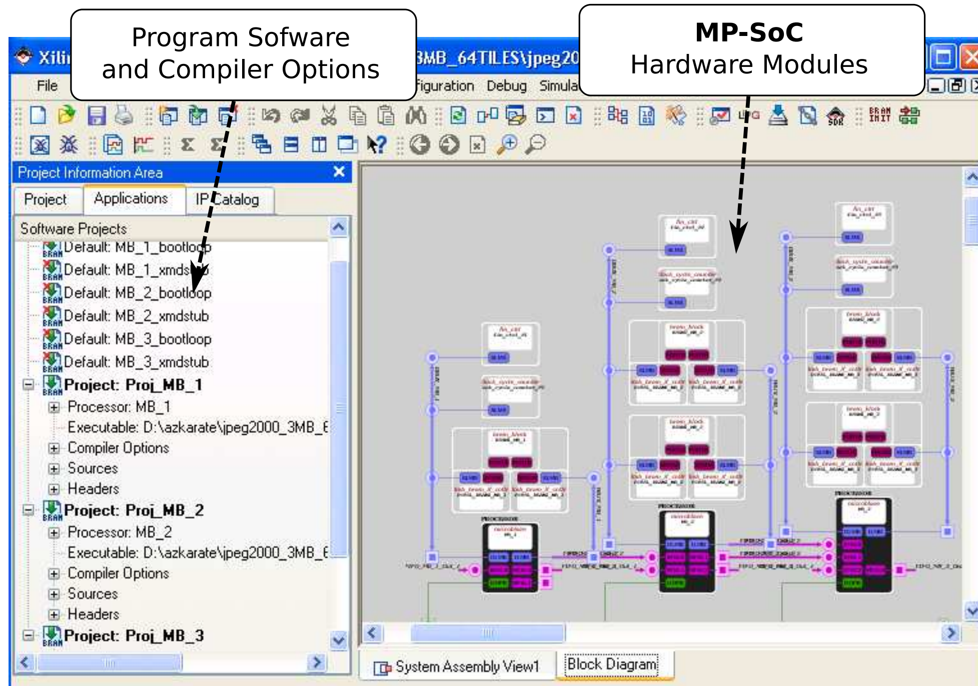


Figure 5.18: Xilinx Platform Studio

The XPS tool is shown in the Figure 5.18 above. On the right side the automatically generated MP-SoC hardware could be modified if necessary, for example, changing a software processor for a hardware IP. On the left side the sources must be attached and after that, the compiler options and the stack size must be set.

Stack, Code and Data Memory

The platform file must preview the necessary code and data memory for each processor. Cross compiling the code with the microblaze compiler ".text" (code) and ".data" section size could be obtained. But this information is not enough to set the processor memory because it is impossible to get the maximal stack size without executing the program.

Therefore, the processor minimal memory will be set adding the empirically obtained maximal stack plus the code and data memory of the program. Another solution could be to allocate the stack into the 1,5Mb off-chip memory, but this memory is connected though a ZBT bus that does not fill the time constraints of the stack instructions.

5.6.3 Execution

For the MP-SoC execution a PC connected Virtex II FPGA has been used. The main challenges to make the MP-SoC work are: the initialisation of the FPGA, the load of the raw original image into the off-chip ZBT memory, the execution of the hardware and the recuperation of the compressed image from the same memory. These problems are solved with a C++ program using Visual Studio and the FPGA DLLs for the PC use of the Virtex II board. The main function of this program is summarized in Figure 5.19:

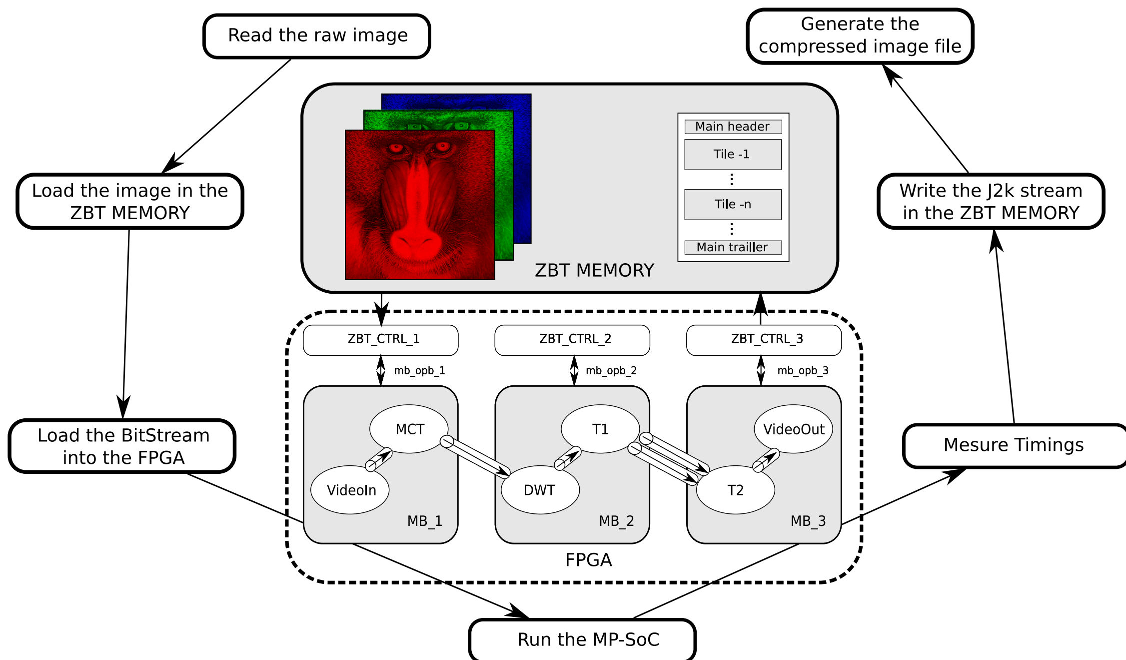


Figure 5.19: J2K MP-SoC Execution

- **Read the raw image:** the three RGB raw image files are read.
- **Load the image in the ZBT memory:** the image is copied to the off-chip memory of the FPGA board.
- **Load the BitStream:** the FPGA is configured with the synthesized hardware and compiled software.
- **Run the MP-SoC:** the execution is started as well as the counters to measure the execution time.
- **Measure and display timings:** once the execution is finished, the measure are read from the FPGA registers and from the ZBT memory if implemented.
- **Read the stream and generate the file:** the JPEG2000 compressed image is read from the ZBT memory and is copied into a file.

Chapter 6

Experiments and Results

In this chapter the experiments that have been done during the project are presented as well as their results. The variations between the experiments are the number and size of processors, the different mappings for a given platform and the number of tiles in which the image is divided. The main limitation for the experiments has been the 144 BRAM blocks memory (288KB) of the Virtex II FPGA. All the configurations have been tested using the same methodology explained in the previous chapter.

All the mentioned experiments are ready to be tested in the CVS repository (restricted server) of the Leiden Institute of Advanced Computer Science (LIACS) in : `~/docs/students/MikelAzkarate/Experiments/`.

6.1 Multiprocessing

The first experience illustrates the computational time improvement depending on the number of Microblaze processors in the MP-SoC.

For that, 4 platforms have been developped which have from 1 to 4 embedded processors, 64 tiles (16x16 pixels) and the mappings described in Table 6.1.

	1xMB	**2xMB**	**3xMB**	**4xMB**
VideoIn	MB_1	MB_1	MB_1	MB_1
MCT	MB_1	MB_1	MB_1	MB_1
DWT	MB_1	MB_1	MB_2	MB_1
T1	MB_1	MB_1	MB_2	MB_2
T2	MB_1	MB_2	MB_3	MB_3
VideoOut	MB_1	MB_2	MB_3	MB_4

Table 6.1: Mapping for the Increasing Processors

As can be seen in Figure 6.1 once at least two processors are defined in the platform there is not significant improvement in computational time and additional measures should be taken to improve the compression time. It is clear that processes

have enough processor resources after the second MicroBlaze. At this point, the are two new ways to accelerate the application, one of them is to increase the processor clock frequency but this one is limited by the FPGA technology. And the other one is to profit the fact of using Kahn Process Networks and exploit parallelism. Using KPNs means that each process is independent from others and auto scheduled. As a result most time consuming processes can be divided in multiple processes that execute tiles in parallel. This principle is applied in the next section.

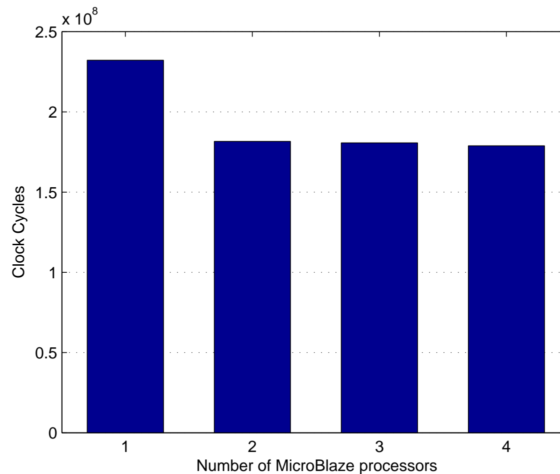
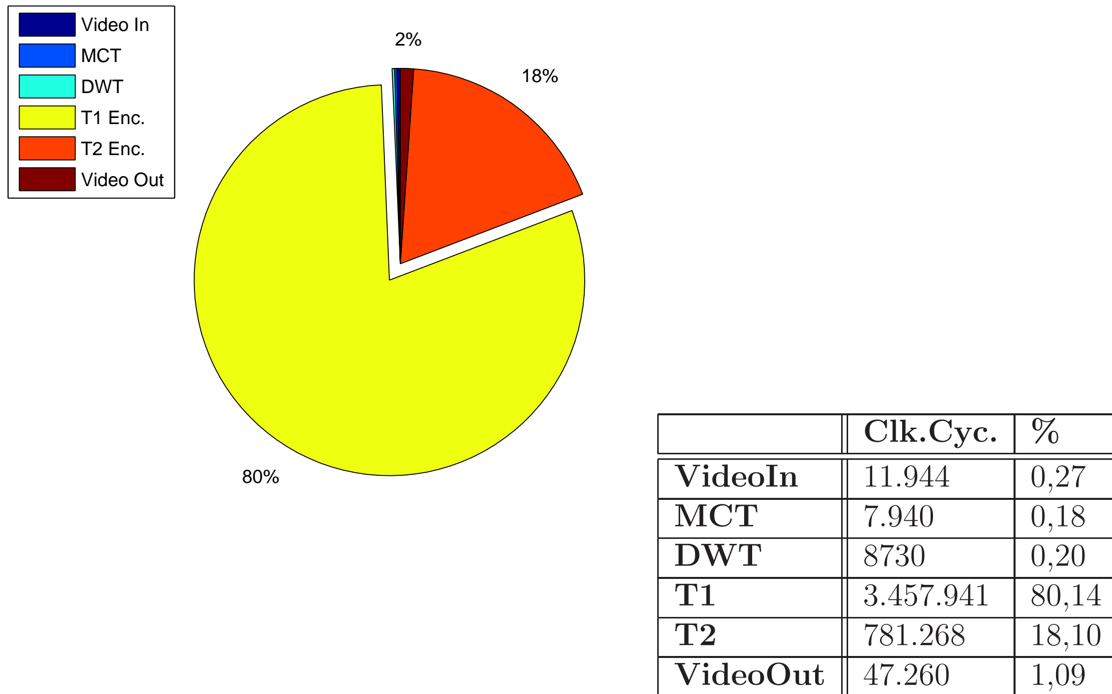


Figure 6.1: Clock cycles Vs. Number of processors

6.2 Splitting

In order to identify the most time consuming processes, the computing times of each process has been measured using a hardware counter in the FPGA. This is the same principle applied in a manufacturing company: to improve the developing time of a product, the critical path of the production tasks are measured and additional resources are given to the longest stages.

For 3 MicroBlaze configuration with 16x16 pixels tile the result has been the following (Figure 6.2):



	Clk.Cyc.	%
VideoIn	11.944	0,27
MCT	7.940	0,18
DWT	8730	0,20
T1	3.457.941	80,14
T2	781.268	18,10
VideoOut	47.260	1,09

Figure 6.2: Computing time percentage taken by each process

Therefore, the experiment clearly points the main computational charge in the Tier-1 encoder process. This fact has been also identified in bibliography [7]. The solution for this problem is to split this process and parallelize it mapping it twice.

Two examples have been done using this philosophy, the first one with a 2 processor platform and the other one with 4 processors where the T1 process has been mapped twice. In the first case the T1 processes are mapped sharing space with other processes, in the second case, T1 processes have an independent MicroBlaze (Figure 6.3).

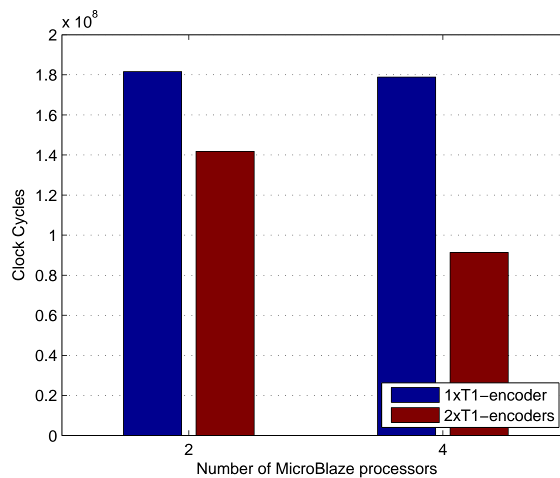


Figure 6.3: Speed-up with parallel T1-encoders

The Figure 6.3 above shows how the improvement is almost double faster in the case where T1 processes are mapped independently and not so significant if they are sharing processors with the rest of functions.

Knowing that, a 5 Microblaze platform with 3 independently mapped T1 processes has been developped. As the Figure 6.4 denote, this technique accelerates the processing time almost as many times as number of independently mapped T1 processes in the system.

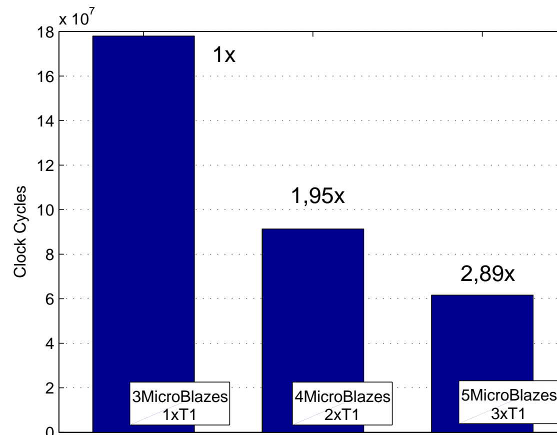


Figure 6.4: Speed-up against number of T1 independent processes

6.3 Optimizing

Once the two main techniques (multiprocessing and T1 encoder splitting) have been analysed, other optimizations are explored in this section. First, the effect of image partitioning in tiles and in components is compared. Then, memory occupation of each process is calculated and the validity of the wavelet transform process is evaluated. And finally, the processors connection type and the compiler optimizations are took into consideration.

6.3.1 Tiling

During the project, experience and bibliography show that for a different number of tiles the execution time varies. In the same 3 MicroBlaze platform, three tile options have been tested: 16 tiles (32x32 pixels), 64 tiles(16x16 pixels) and 128 tiles (8x8 pixels) for a 128x128 pixels 48KB original raw image.

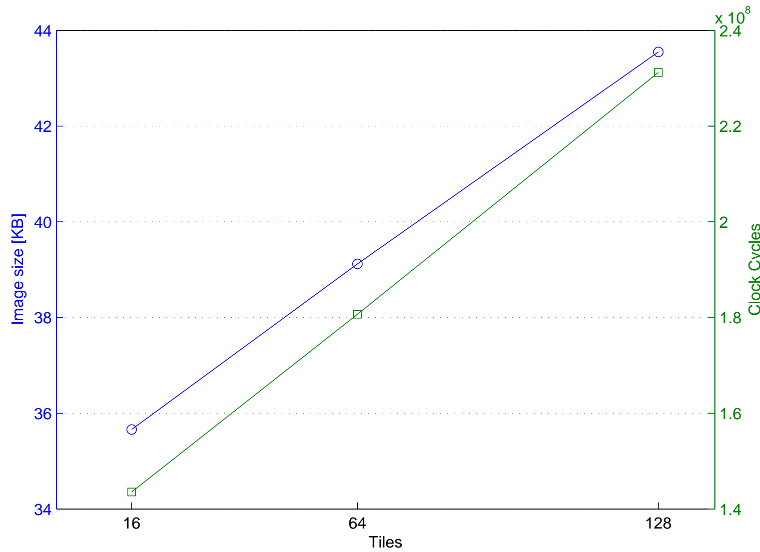


Figure 6.5: Tiling effect in size and time

On the right part of Figure 6.5, it can be seen that the computing time grows proportionally with the number of computed tiles. On the left part another observation shows that the compressed image size also increases with the number of tiles, that is because for each tile a new header is included in the compressed image file overloading it (Figure 6.6).

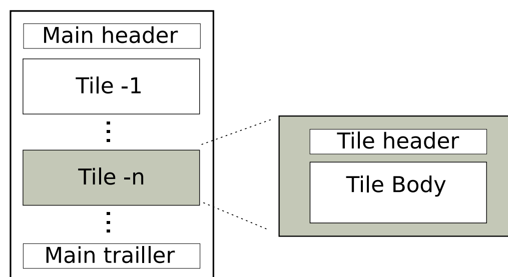


Figure 6.6: Final JK2 File and Tile Header Overload

Therefore, the reasons to use tiles are that dividing the image the shared memory between the processes is smaller and makes the platform fit in the FPGA's limitations and also the fact of using parallelism is only possible if the image is partitioned.

6.3.2 Division into Components

As seen in the previous section tiling is necessary but inefficient. That could be solved by partitioning the computation in another more efficient way. Analysing the reference code there is denoted that each component of the image is computed separately even if they are send together from process to process. Dividing the shared

memory into components the FIFO size will be decreased in as many component as the image have, in this case 3. The structure remains as shown in Figure 6.7.

```
typedef struct tile_c_t {
    int comp[WIDTH_TILE*HEIGHT_TILE];
    int num;
} tile_c_t;
```

Figure 6.7: Tile Component Structure

However, this change needs also the rewriting the code of the KPNGen input file where another loop (Figure 6.8), this time for components, is attached (line 16).

```
1  #include "jpeg2000_func.h"

   int main (int argc, char **argv)
   {
5     int i,j;
      tile_t tile;
      tile_c_tile;
      codeblock_t codeblocks;
      passes_t passes;
10    packet_t packets;

      for (i = 0; i < NUMTILES; i++)
      {
          mainVideoIn(&tile);
15
            for(j = 0; j< NUM_COMPS; j++)

               mainMCT(&tile,&tilec);
               mainDWT(&tilec,&tilec);
20             mainT1(&tilec,&codeblocks,&passes);
               mainT2(&codeblocks,&passes,&packets);
               mainVideoOut(&packets);

      }
25
      return 0;
   }
```

Figure 6.8: Improved JPEG2000.c with Components

6.3.3 Memory occupation

As commented in previous sections, the memory size of the platform processor must be calculated cross compiling the FPGA version code with the MicroBlaze compiler

and adding the needed stack obtained empirically.

In Table 6.2 the code and data sizes in bytes for the processes in the first column as well as the minimal stack for a 16x16 pixels tiles version are listed. The code and data includes also the necessary code to read, write and manage FIFOs, which is fixed for processor. That is why putting two processes together the amount of memory is less that the theoretical addition. Medium optimization level has been used as compiler requirements.

	Code	**Data**	**Stack**
VideoIn + MCT	2.268	60	3.000
VideoIn + MCT + DWT	3.548	92	5.000
T1	34.934	8.960	3.250
T2	36.002	872	5.000
VideoOut	9.440	52	3.000
T2 + VideoOut	39.122	876	5.000

Table 6.2: Memory Occupation Processes

Focussing in the T1 process, it can be conclude that a new processor of around 48KB of memory is needed for each additional parallel T1 encoder.

6.3.4 Wavelet in Lossless Compression

When the JPEG2000 compression is lossless, wavelet is not usefull because it does not use quantization. The only use of wavelet in this case, is giving several resolutions of the image. For optimization wavelet could be removed from the JPEG2000 lossless algorithm, removing the size of the code that is not longer use and the memory occupation of the additional FIFO for the DWT stage.

6.3.5 CrossBar Switch

In the previous experiments, no connection type for communications between processors was specified. That is because by default point to point connection is applied. Point to point is the best connection type in terms of time efficiency but when implemented is not optimal from the memory occupation point of view. For every new communication channel a new FIFO is implemented and this FIFOs has a fixed size that must be power of 2. When the real size of the channel is far from the power of 2 size of the implemented FIFO memory is wasted.

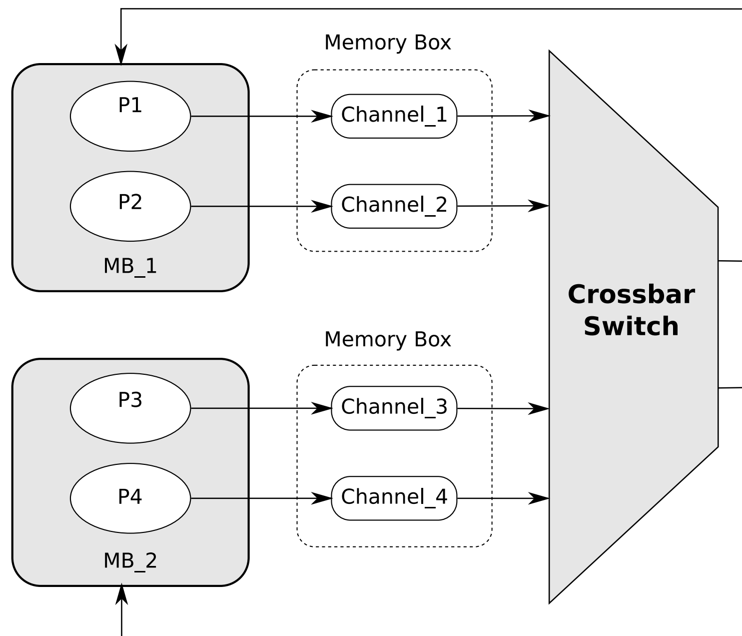


Figure 6.9: Crossbar Switch

The crossbar switch connects every processor to each other through multiplexers that schedule the different accesses. In this case all the channels of a single processor share the memory module which optimize the memory occupation. On the other hand, the scheduling for memory accesses makes this option slower but not significantly.

/* jpeg2000.pla */

```
...
1  <network name="CS" type="CrossbarSwitch">
    <port name="IO_1" />
<port name="IO_2" />
   </network>
5
   <link name="BUS1">
    <resource name="MB_1" port="IO_1" />
    <resource name="CS"   port="IO_1" />
   </link>
10
   <link name="BUS2">
    <resource name="MB_2" port="IO_1" />
    <resource name="CS"   port="IO_2" />
   </link>
...
```

Figure 6.10: Part to be added in the platform specifications

To include the crossbar into the MP-SoC the platfrom specifications must be changed including the crossbar and the connections to each processor (Figure 6.10).

Table 6.3 below shows the small difference in executing time (clock cycles) for a 3 T1 encoders and 16x16 tile, with crossbar and point to point connections:

	Clock Cycles
Crossbar	61.720.707
Point to point	61.613.191

Table 6.3: Crossbar Vs Point to point

At the end, the previously mentioned memory organisation (Figure 6.9) has been found as not optimal for the studied case. As explained, single memory boxes are used by several FIFOs and this will improve the memory utilisation only in the cases where big memory boxes are used to store single small FIFOs. But in this case, the addition of FIFOs forces to uses bigger memory boxes and carries similar memory amount to the ones using in point to point connections.

6.3.6 Compiler Options

Microblaze GGC compiler options have been left in medium in order to have a compromise between program code size and execution time. In this experience, optimization level have been grown to high, increasing the code size but improving the timing of the execution. This table show the improvements, for the same implementation conditions than for the crossbar in the previous section (16x16 pixels tiles, 3 T1 encoders):

	Clock Cycles
Medium optimization	61.613.191
High optimization	55.998.573

Table 6.4: Compiler optimization improvement

As listed in Table 6.4 the number of necessary clock cycles for the 48KB image compression decrease almost a 10%. But this optimization increases the code size of the T1 encoder process from 34KB to 44KB while the data and the necessary minimal stack does not change but adding their 8KB and 3KB makes in total 55KB which does not allow more T1 encoders in the actual FPGA memory.

6.4 Best Result

The best result is the 3 T1 encoders implementation with 16x16 pixels tiles with high code optimization (Figure 6.11). A better implementation with 4 T1 encoders was also possible with medium optimization if 48KB were enough for each T1 process holding processor. But when two memories are used (32KB and 16KB to obtain

48KB) instead of a single one of 64KB, there is a "one memory limitation" in the Xilinx "data2mem" loader where the second contiguous memory is not filled with the executable code.

As will be demonstrated in the following chapter, better implementation are possible beyond the technological limitations.

6.5 3 T1 Encoders

Figure 6.11 below is the architecture that with the mentioned limitations, better fits the FPGA:

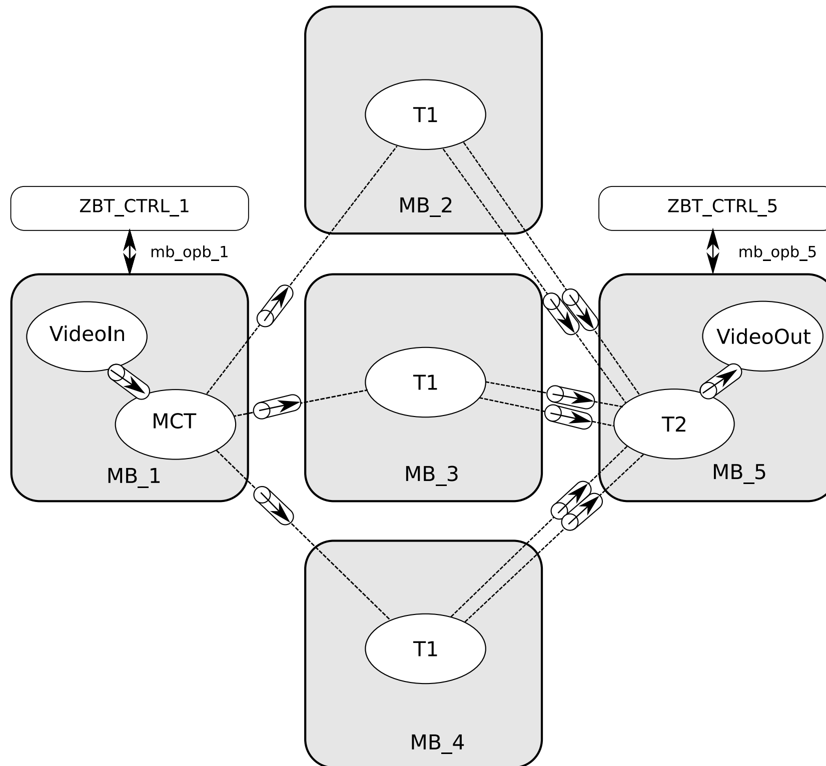


Figure 6.11: Best MP-SoC platform

For every process that want to be splitted the main file, that is the input of the KPNGen tool, must be changed in order to provide to the system the necessary connections and computation loops. For this best solution the file is shown in Figure 6.12. In lines 20-22, the T1 process splitting is shown in a explicit way: three processes are processing each YUV component of the tile.

```
1  #include "jpeg2000_func.h"

   int main (int argc, char **argv)
   {
5     int i,j;
      tile_t tile;
      tile_c_tile;
      codeblock_t codeblocks;
      passes_t passes;
10    packet_t packets;

      for (i = 0; i < NUMTILES; i++)
      {
          mainVideoIn(&tile);
15
            for(j = 0; j< NUM_COMPS; j++)

               mainMCT(&tile,&tilec);
               mainDWT(&tilec,&tilec);
20             if (j%3 == 0) mainT1(&tilec,&codeblocks,&passes);
               if (j%3 == 1) mainT1(&tilec,&codeblocks,&passes);
               if (j%3 == 2) mainT1(&tilec,&codeblocks,&passes);
               mainT2(&codeblocks,&passes,&packets);
               mainVideoOut(&packets);
25
      }

      return 0;
   }
```

Figure 6.12: JPEG2000.c File for Best Solution

In this case the FPGA is not completely used, because 5 processor of 64KB are used instead of the enhanced 48KB, but the extra memory is used to increase the compiler optimization. The Table 6.5 shows the platform size, mapping and executable size of each processor. In the last column the unused memory size is listed. As mentioned before, it will be possible to used this memory for additional T1 coder processes if the Xilinx "one memory limitation" is solved.

	Mapping	**Size**	**Executable**	**Lost**
MB_1	VideoIn + MCT	8KB	5KB	3K
MB_2	T1	64KB	58KB	6KB
MB_3	T1	64KB	58KB	6K
MB_4	T1	64KB	58KB	6K
MB_5	T2 + VideoOut	64KB	50KB	14KB

Table 6.5: Platform, Mapping and Section sizes for Best Result

Table 6.6 list the features of the Virtex II FPGA use. As can be seen the BRAM utilisation is the critical one in order to improve possible configurations.

	Implementation
Slices	3895 out of 33792 (11%)
LUTs	7789 out of 67584 (11%)
BRAMs	143 out of 144 **(99%)**
Clock Cycles	55.998.573 (128x128 pixels)
Clock Frequency	74.683MHz

Table 6.6: Implementation information

.

Chapter 7

Comparison

This chapter makes a comparison between hardware IPs mentioned in Section 1.3 with the obtained best result also with hypothetical configurations that could be implementable in the future with more available memory. Project stage worktimes are also displayed as figure of the fast prototyping capabilities of the Daedalus framework.

7.1 Comparison with Related Work

In Table 7.1 below, some features of commercial hardware IP are compared to the Daedalus proposed architecture, speed is measured pixels per second.[4]

	Analog Dev.	**Barco Silex**	**Cast**	**Daedalus**
Technology	XC2V3000-6	XC2V6000-6	Altera/Xilinx	**XCV6000**
T1 Coders	3	8	configurable	**unlimited**
Max Tile	2048x4096	32x32	4096x4096	**unlimited**
Max Cbkl	64x64	32x32	64x64	**same as tile's**
Memory	not provided	167 KB	not provided	**288 KB**
Speed	250-500Mbps	98-200Mbps	60Mbps	**28,5Kbps**

Table 7.1: Comparison with Commercial tools

As can be seen, the obtained best Daedalus architecture can not achieve the timing and memory levels of commercial IPs. However, the two main advantages of Daedalus architecture are:

- **The price**: These hardware IPs are very expensive and the Daedalus toolchain is obtained through open-source tools free of costs. For instance the JPEG2000 IP for Xilinx or Altera of Cast company costs 100.000 euros per design.
- **The configurability**: With no memory limitation, tile size and the number of T1 entropy coder could be increase arriving to very acceptable compression times.

7.2 Projected Architectures

As previously seen, the limitation of the architectures is the on-chip RAM memory of the FPGA. But with the informantion obtained into the experiences it is possible to estimate the theoretical performance of more sized implementations. In this section the tile number and the number of T1 branches will be variables of the equation in order to get high performace timing solutions.

From Figure 6.5 can be obtained that for every new tile in which the image is divided the number of clock cycles increases in 775.000. On the other hand, we know that for every double independent encoder the speed up is of x1,95. In this case we must specify that the T2 process should be also splitted for every 3 T1 encoders because the time consummation of this process begins to be critical. For any T1 or T2 processor 48KB or memory will be reserved and the original image will have 48KB (that will be taken into consideration for memory constraints). The other memory resources which are not processes or FIFOs will not be taken into account.

The memory approximate occupation formule will be:

$$Mem = \mu P_1 + \mu P_{Additional} \times (T1 + T2) + \frac{Img.Size}{TileNb} * (1 + (\frac{FIFONb-1}{3})$$

$$Mem = 8KB + 48KB \times (T1 + T2) + \frac{48KB}{TileNb} * (\frac{2 \times T1 + T2}{3})$$

Figure 7.1: Memory Occupation Formules

The formule reveal that only critical memory amounts are computed, hardware memory requirements as controller's registers or software contrainst as stack are not included. For computational calculation here the formule:

$$Cycles = 3xT1_1Tile_Cycles/(1,95 \times (\frac{T1}{3} - 1)) + TileNb \times Cycles_Tile$$

$$Cycles = 50M/(1,95 \times (\frac{T1}{3} - 1)) + TileNb \times 775.000$$

Figure 7.2: Computation Time Formules

	T1 Enc.	T2 Enc.	Tiles	Memory	Cycles
Best Result	x3	x1	64	264KB	56M
5 Processors	x3	x1	1	528KB	50M
9 Processors	x6	x2	2	438KB	27M
17 Processors	x12	x4	4	888KB	16M

Table 7.2: Proposed Architectures

For a piece of example, this 17 processors implementation (Table 7.2) for a 128x128 pixels image encoded in 16M clock cycles in a standard speed of 100MHz clock, means 96Kbps processing speed.

7.3 Worktime of Project Steps

Finally, the time of each task has been measured in working days in order to illustrate the time effort of each task. Table 7.3 makes clear that the main effort resides in developing the partitioned code and once this is made and the tools are known, it is possible to develop MP-SoC prototypes in less than one hour.

	Time
Library Selection	2 days
Code Partition	33 days
Simulation	7 days
First Demo Example and Synthesis	5 days
Next Specifications and Synthesis	1 hour

Table 7.3: Time periods through the workflow

This fast prototyping is very interesting for companies that once they have the partitioned program could present different implementations to their clients in terms of days in order to fit their performance/cost requirements.

Chapter 8

Conclusion and Future Work

The main conclusion of the work is that the Daedalus toolflow is ready for commercial use in terms of worktime improvement and satisfactory results. Commercial requirements have been achieved and an optimized version of the solution have been obtained within 5 months of work. Daedalus potential resides in the easy and fast prototyping capability once the partitioned program version is ready.

Using Daedalus, a fully functional lossless JPEG2000 application has been developped written in a static way. This application is divided into modules and connected via fixed sized structures. The main file of the program is the input for the Daedalus system, the Kahn Process Network Generator in first term.

The obtained Kahn Process Networks have been mapped into different test platform through automatized procedures in order to modelate the behaviour of the system. Two conclusions have been obtained from this modelation: 1) Image tiling is inefficient in compression size and time but necessary for computation memory and 2) The T1 encoding takes around the 80% of the whole encoding time.

In the last part of this thesis several improvement have been applied to the system in order to improve the compression timing features. A final best result have been obtained with the Virtex II FPGA memory limitations and some new architectures have been proposed with no memory restrictions.

The results are far from commercial IP hardwares in computational time. That is because this commercial tools use hardware implementations of T1 encoders which is determinant to improve speed significantly, therefore, some hardware accelerators for this T1 coder should be integrated to be closer of this results. But the Daedalus open-source obtained solutions are more configurable and cheaper than commercial products that cost around hundreds thousands of euros.

For future work, 4 main points should be taken into consideration:

1. **Program optimizations:** to the mentioned T1 hardware accelerators many other optimization could be added, as putting the stack in the off-chip memory or sharing single coding tables for the multiple T1 processes, improving the use of memory.

2. **Design Space Exploration:** Amsterdam University's SESAME tool can give

the best configuration of processes in order to achieve the best memory/timing performances. This results could be interesting for future implementations.

3. **New JPEG2000 features:** for medical sector requirement only lossless compression have been left from the reference OpenJPEG library. In the future, lossy compression as well as the whole decoder could be implemented to have a complete JPEG2000 application.

4. **Automization:** Daedalus is already a highly automatized tool, but still some improvement could be done. Between them, FIFOs size could be automatically obtained as well as program size. Code files files could be also automatically attached to the project. The main difficulty for the whole automatization is the stack calculation, maybe some tools must be explored to obtain this parameter off-line.

Bibliography

[1] Artemis. http://ce.et.tudelft.nl/artemis/, 2008.

[2] Cast. http://www.cast-inc.com/, 2008.

[3] Analog Devices. http://www.analog.com/en/audiovideo-products/, 2008.

[4] IEEE. A flexible, hardware jpeg2000 decoder for digital cinema, 2004.

[5] JPEG. The jpeg-2000 still image compression standard, 2005.

[6] T. Kangas. Acm trans. on embedded computing systems, 2006.

[7] Ashwani Kumar. Eficient synthesis of application specific mp architectures for process networks (case study-jpeg-2000), 2005.

[8] University of Amsterdam. Towards efficient design space exploration of heterogeneous embedded media systems, 2001.

[9] OpenJPEG. http://www.openjpeg.org, 2008.

[10] Friedrich Alexander University. A systemc-based design methodology for digital signal processing systems, 2007.

[11] LIACS UvA and Chess BV. Daedalus: Toward composable multimedia mp-soc design, 2008.

[12] Xilinx and Barco Silex. Jpeg2000 encoder and decoder, 2007.

[13] Wei Zhong. Embedded system-level platform synthesis and application mapping for heterogeneous and hierarchical multiprocessor systems, 2004.