A JPEG Encoder in SystemC

Björn Jonsson

Luleå University of Technology
MSc Programmes in Engineering
Department of Computer Science and Electrical Engineering
Division of EISLAB
A JPEG Encoder in SystemC

Master’s thesis

Björn Jonsson
Abstract

This thesis evaluates how SystemC, an open source system level modeling language, may improve system design. A gap exists in current design workflow, which causes an unnatural interrupt between high-level and low-level modeling. A common ground for software and hardware development is also absent in present design methodologies, which decreases the chances for successful co-operation between project members.

As a part of this investigation, a JPEG encoder has been modeled in SystemC. A software prototype of the JPEG encoder has later been implemented on an ARM Integrator platform. Based on these experiences, a proposal for a complete system design flow, incorporating SystemC, has been made. The design flow spans from Algorithmic level down to Register Transfer Level. The essence of the design flow is SystemC in conjunction with Transaction Level Modeling, which not only provides solutions to overcome the limitations of traditional design, but also introduces new possibilities.
Preface

As a part of the one-year long Vulcanus in Japan programme, I have performed my master’s thesis at the Electronic Device Laboratory of Dai Nippon Printing in Tokyo. This thesis work has been a part of the Master of Science in Computer Science and Engineering program at the Embedded Internet Systems Laboratory (EISLAB) at the Department of Computer Science and Electrical Engineering (CSEE) at Luleå University of Technology. For have been given this opportunity, I would like to thank the helpful staff of the EU-Japan Centre for Industrial Cooperation in Brussels and in Tokyo. I would especially like to thank Margherita Rosada and Trinidad Iglesias in Brussels and Keiko Sato-san in Tokyo for their consistent support throughout the whole year.

From Dai Nippon Printing, there are more persons than I could possibly fit on this page that I would like to express my gratitude towards. I would especially like to thank the following persons:

- Ohsumi-san, my supervisor at the Electronic Device Laboratory. Thank you for making this thesis a reality and for your guidance.
- Osawa-san, my mentor and guru. Thank you for your patience to my never-ending flow of questions and for your cheerful attitude.
- Umekai-san, thank you for your C++ expertise and for helping me out in my daily life.
- Seino-san, thank you for your help in hardware design and for always having a smile close at hand.
- Ikebe-san, my sensei and karaoke instructor. Thank you for your explanations of image technology and for your positive spirit.
- Itoh-san, I wish to thank you, Umekai-san, Seino-san and Ikebe-san for all the interesting lunch conversations we enjoyed together over a bowl of hot ramen and a glass of tea at Hong Kong Tei.
- Watanabe-san, my dormitory manager. I wish to thank you, your family and all the members of the Kameido dormitory for being such excellent hosts and for making me feel welcome.

Finally, I would like to thank my examiner Per Lindgren for valuable help on this thesis and my dear friends Anton Persson and Andreas Lundström for proofreading this report.

Oxelösund, 16th March 2005

Björn Jonsson
# Contents

1. **Introduction**
   - 1.1 Background .................................................. 1
   - 1.2 Purpose ....................................................... 1
   - 1.3 Objectives .................................................... 2
   - 1.4 Limitations .................................................... 2
     - 1.4.1 Software environment .................................. 2
     - 1.4.2 Hardware environment .................................. 2
   - 1.5 Thesis Outline ............................................... 3

2. **Background**
   - 2.1 SystemC ....................................................... 4
     - 2.1.1 Architecture ............................................ 4
     - 2.1.2 Design methodologies .................................. 7
     - 2.1.3 Industry support ....................................... 8
     - 2.1.4 Known limitations ..................................... 8
   - 2.2 System Design Flow ......................................... 9
     - 2.2.1 Traditional system design flow ....................... 9
     - 2.2.2 System design flow with SystemC ..................... 10
   - 2.3 RS232 .......................................................... 10
   - 2.4 JPEG Encoding ............................................... 12
     - 2.4.1 Image data layout .................................... 13
     - 2.4.2 Color conversion ...................................... 15
     - 2.4.3 Sub sampling ........................................... 15


2.4.4 2D FDCT ......................................................... 15
2.4.5 Quantization ..................................................... 16
2.4.6 Entropy encoding ............................................... 16

3 Method .................................................. 18
  3.1 System Design Flow with SystemC ......................... 18
    3.1.1 Algorithmic model ........................................ 19
    3.1.2 Functional models ......................................... 19
    3.1.3 Transaction level models ................................. 20
    3.1.4 Register Transfer Level model .......................... 21
  3.2 An UART in SystemC ........................................... 22
    3.2.1 Specification ............................................. 22
    3.2.2 Implementation .......................................... 22
    3.2.3 Verification ............................................ 23
  3.3 A JPEG Encoder in SystemC ................................. 23
    3.3.1 Specification ............................................. 23
    3.3.2 Software-hardware prototype ............................ 24
    3.3.3 Software prototype ..................................... 26
    3.3.4 Verification ............................................ 27
    3.3.5 Simulation ............................................... 27

4 Discussion ...................................... 30
  4.1 SystemC for System Design ................................. 30
  4.2 System Design Flow with SystemC ......................... 30
  4.3 Future Work ................................................. 31
  4.4 Conclusion ............................................... 32

A Optimization of FDCT algorithm .......................... 34
List of Figures

2.1 An overview of the basic architecture of SystemC .......................... 5
2.2 Core language connectivity ............................................. 5
2.3 Survey of SystemC usage ................................................. 9
2.4 Traditional system design flow .......................................... 10
2.5 SystemC design flow ..................................................... 11
2.6 Relationship between DTE and DCE .................................. 11
2.7 Connection of signals .................................................... 12
2.8 RS232 bit sequence ....................................................... 12
2.9 A typical data transmission ............................................. 12
2.10 JPEG encoding flow ..................................................... 13
2.11 Data block and sample layout ....................................... 14
2.12 Comparison of MCU configurations ................................. 14
2.13 Comparison of YUV formats ........................................... 15
2.14 2D FDCT formula ....................................................... 16
2.15 Zigzag scan sequence .................................................. 16

3.1 Overview of development models ..................................... 19
3.2 Abstraction levels ........................................................ 20
3.3 Intended system layout of the software-hardware prototype .......... 24
3.4 Intended component mapping of software-hardware prototype .... 25
3.5 System layout of the software prototype ............................ 26
3.6 Component mapping of software prototype .......................... 27
3.7 Normalized simulation speed .......................................... 28
LIST OF FIGURES

A.1 Input and output 1D FDCT array. ........................................... 34
A.2 Optimized 1D FDCT formula .................................................. 35
List of Tables

3.1 Characteristics of development models .......................... 22
3.2 Overview of image size & ratio. ................................. 28
3.3 Simulation time of JPEG encoder .............................. 28
3.4 Scaled simulation time of JPEG encoder ...................... 29
A.1 Result of FDCT optimization. ................................. 34
Chapter 1

Introduction

For a long time, there has been a missing link between software and hardware development in system design. The software engineers have been using their own methodologies and tools in one end of the project. The result of their work has then been handed over to the hardware engineers, who have used methodology and tools specific for them.

This leaves a gap between the two groups, breaking the natural flow of development. While this might have been sufficient in the simple designs of yesterday, the increasing complexity of today’s system design calls for a new methodology. A new methodology with development in higher abstraction levels, which offers increased simulation speed, increased verification and simplified IP-reuse. It also has to provide increased cooperation between software and hardware designers in an efficient way throughout the entire project development.

1.1 Background

The Electronic Device Laboratory of Dai Nippon Printing (DNP) is currently developing a new kind of image sensor. A sensor that offers a substantial increase of dynamic range compared to equal sensors available today. As a part of this development, there is a desire from DNP to investigate into the possibility of establishing a new system design flow. The primary target of the new design flow would be increased co-operation between software and hardware engineers and increased performance of architecture exploration.

An evaluating system including the new image sensor is under development to demonstrate the image sensor’s capabilities. A combination of a JPEG encoder and a JPEG decoder will handle the streaming of image data throughout the system. While DNP already has produced their own JPEG decoder, a JPEG encoder is left for development.

1.2 Purpose

The purpose of this thesis is first and foremost to evaluate the use of SystemC for system design. This includes the development of a stable design flow incorporating SystemC, beginning from algorithm level and continuing down to RTL level.
The intention is to implement an UART and a JPEG encoder in SystemC as a first test of this design flow. The JPEG encoder might also fill a function in DNP’s prototype system and for future use.

1.3 Objectives

- Evaluate SystemC for system design.
- Establish a System design flow with SystemC.
- Implement an UART in SystemC.
- Implement a general JPEG encoder in SystemC.
- Develop a prototype of the JPEG encoder on an ARM Integrator board.

1.4 Limitations

1.4.1 Software environment

The software development has taken place on an Intel Pentium 4 equipped PC running Microsoft Windows XP SP1.

Microsoft Visual C++ 6.0 SP 5 and SystemC 2.01

Visual C++ has been used for C/C++ software development. Together with SystemC 2.01, the latest available version of SystemC, it has been used for SystemC development and simulation.

Mentor Graphics ModelSim 5.7

ModelSim 5.7 has been used for VHDL RTL modeling and simulation.

Microsoft Excel

Excel has been used to analyze and optimize various algorithms used in the JPEG encoder.

1.4.2 Hardware environment

The hardware prototype environment is based on the ARM RealView Integrator platform.
ARM Integrator/AP baseboard

The Integrator/AP Baseboard provides the system infrastructure with the underlying AMBA bus infrastructure for communicating between core modules and logic modules. It also contains connectors for external communication capabilities.

ARM Integrator/CM7TDMI Core module

The CM7TDMI Core module is equipped with an ARM7 class CPU and contains an SDRAM DIMM socket equipped with 128 MB of on-board system RAM. The Core module provides an environment for embedded software prototyping.

ARM Integrator/LM-EP20K1000E Logic Module

The LM-EP20K1000E Logic Module contains an Altera Apex EP20K1000E FPGA that can be used for hardware prototyping. This board was intended to be used for the hardware part of the JPEG encoder. Unfortunately the only available board proved to be defective and a replacement could not be provided before the end of the project.

1.5 Thesis Outline

Chapter 2 introduces technologies of interest to this thesis.
Chapter 3 describes the method used to fulfill this thesis purpose.
Chapter 4 discusses the results of the project and future improvements.
Chapter 2

Background

2.1 SystemC

As the name might imply, SystemC is a System Design language built upon C++. The purpose behind SystemC was to meet the needs of today’s development process by “supplying a standardized modeling language intended to enable system level design and IP exchange at multiple abstraction levels, for systems containing both hardware and software components”. [6]

The initial version 0.9 was released in 1999. This version supported modeling and simulation at Register Transfer Level(RTL), an abstraction level similar to what the older HDL languages use. SystemC has since then been continuously enhanced. The current version 2.01 of today now also supports development at higher abstraction levels. [1]

Behind SystemC stands the Open SystemC Initiative (OSCI). OSCI is an independent organization consisting of major players in the electronics business, including EDA (Electronic Design Automation) tool vendors and electronics manufacturers. Their task is to develop, standardize and promote the use of SystemC throughout the world.

2.1.1 Architecture

SystemC is not a language in itself, but rather an extension to C++ in form of a class. The standard C++ language thus provides the base of its architecture. On top of that, several building blocks of SystemC extend the modeling capabilities of the design (figure 2.1).

Core Language

One vital concept in SystemC modeling is the separation between communication and functionality. This enables the designer to model different architectural elements at different abstraction level, which for example simplifies architectural exploration.

Communication is done via channels, while processes provide the functionality of the design. These two elements are defined by the Core Language block of SystemC. This block defines the basic structural elements of SystemC. These elements and how they interact are described by figure 2.2.
Figure 2.1: An overview of the basic architecture of SystemC

Figure 2.2: Core language connectivity
Modules contain processes and ports. Ports provide the modules with means of external connectivity. Processes are the computational units of the design and are executed in concurrency with each other. Processes communicate between each other through channels. Processes interact with their channels through the channel’s methods, as defined by its interface. Events are used to trigger specific actions within the processes. [5]

Modules Except for containing ports and processes, modules also contain internal channels and other elements belonging to its processes. Modules can also be placed within other modules, thus providing a solution for creating a well-organized hierarchy. [5]

Processes In contrast to software, where execution is done sequentially, execution is done in parallel on hardware. SystemC uses processes to model this behavior. Processes have sensitivity lists that control their execution. Events trigger these sensitivity lists.

Three kinds of processes exist in SystemC:

- Methods
- Threads
- Clocked Threads

Methods are the simplest form of a process. After being triggered by an event, it executes until it reaches its end. Threads on the other hand have a more advanced nature, which allows them to be temporarily stopped during execution. They can later be restarted from their current position. This makes a more high-level functionality possible, at the cost of decreased simulation speed. Finally, Clocked Threads are a special form of threads that are only being triggered by clock events. [5]

Ports Ports let modules communicate with each other through shared external channels. In this case, the ports and their connected channels share the same interface. The modules processes can then access the external channels methods via the ports.

Ports can also be connected directly to another port. This further helps to design well hierarchical organized models, creating modules within modules. Three kinds of ports exist, in, out and inout. Their names simply implies in which direction they are able to communicate. [4]

Channels Processes can be connected to channels in two different manners:

- Direct connection, for communication between processes of the same module.
- Connection via ports, for communication between processes of different modules.

Channels can be subordinated into two categories: Primitive channels and Hierarchical channels. Primitive channels have a simple functionality and cannot use processes of their own or communicate with other primitive channels by themselves. This basic behavior gives them the advantage of high simulation speed.

Hierarchical channels have a more advanced functionality, which comes with a cost of decreased simulation speed. They can contain internal processes and other channels, while at the same time providing a simple, outward interface. This property makes hierarchical channels excellent for modeling complex channels, for example buses. [4]

Interfaces An interface is a collection of definitions of methods. It does not define the functionality of these methods, but merely defines how to call upon them and their expected output format.

An interface is implemented by a channel, which also defines the functionality behind it. Ports also use these interfaces, and can only be connected to channels that implement an identical interface. [4]

Events Events are messages triggered mostly by channels when something special happened. They can be used for synchronizing and controlling the flow of processes. They are simulated to execute in “zero time”, that is they have an immediate effect on their targets.
**CHAPTER 2. BACKGROUND**

Data Types

While the built-in data types of C++ can be used perfectly in SystemC, the Data Types block contains hardware-style data types. These data types all have their own set of specific values and operations. This makes it possible to more accurately model the required precision and behavior of the process. Since SystemC is based on C++, there is of course also possible to define your own data types.

The SystemC data types are:

- **sc_bit** bit with two values (0, 1)
- **sc_logic** bit with four values (0, 1, X, Z)
- **sc_bv** vector of sc_bit with an arbitrary size
- **sc_lv** vector of sc_logic with an arbitrary size
- **sc_int/sc_uint** signed & unsigned integer with a fixed size of 1 to 64 bits
- **sc_bigint/sc_biguint** signed & unsigned integer with an arbitrary size
- **sc_fixed/sc_ufixed** signed & unsigned fixed point data (static constructor)
- **sc_fix/sc_ufix** signed & unsigned fixed point data (variable constructor)

Elementary Channels

SystemC comes with some already implemented primitive channels that might be useful in many situations:

- **sc_mutex/sc_semaphore** synchronizing of objects shared between processes.
- **sc_fifo** simple, synchronized FIFO communication between processes.
- **sc_signal** modeling of hardware data signals.

**sc_mutex** and **sc_semaphore** can be used for securing correct operation of shared objects between processes and preventing race conditions.

**sc_fifo** can be used to model FIFO channels. This is especially helpful for simple synchronized communication between processes in development in higher abstraction layers. Both blocking and non-blocking communication can be utilized.

### 2.1.2 Design methodologies

One key feature of SystemC is its support for high level modeling, which holds advantages in simulation speed and hardware-software management compared to traditional modeling. System design with SystemC, including design methodologies such as Transaction Level Modeling, will be presented in detail in a separate section.
2.1.3 Industry support

The success of SystemC will, to a great degree, depend on the support offered by the EDA industry. If users of a current HDL-based workflow are to develop their design methods and incorporate SystemC, the necessary EDA-tools and training have to be available.

Development environment

Since SystemC is an extension to standard C++, it should in theory work easily with most standard C++ compilers. In reality, SystemC has proved to perform well together with Microsoft Visual C++ and GCC amongst others. Common operation systems for development are Windows 2000/XP, Linux and UNIX.

Available tools

Although SystemC and the included reference simulator work completely on its own, several options that simplify the development of advanced designs are available from major EDA vendors.

The purpose of many of the tools currently available is to combine existing HDL workflow with SystemC. Most SystemC tools are quite new and have not yet reached the same level of maturity as corresponding HDL tools. They can offer features such as:

- Automatic HDL RTL to SystemC translation and vice-versa.
- HDL and SystemC Co-simulation.
- SystemC verification

Usage of SystemC

A company named Doulos Ltd., which offers training courses in SystemC, made a survey among its 25 top European customers in the electronics industry in May 2003. One of the questions asked in the survey was “What are you using SystemC for, now or in the near feature?” (figure 2.3). [7]

Although this survey may be somewhat limited, it is still interesting. From the result of the survey the conclusion can be made that the current usage is concentrated to system optimization, high-level modeling and co-simulation. There also exist an interest for high-level synthesis and real-time software modeling. RTL synthesis seems to have been rejected by the community.

2.1.4 Known limitations

The present version of SystemC has rather poor support for Real-Time Operation System (RTOS) modeling. OSCI implies in [6] that future version of SystemC will likely include support for advanced thread operations and timing constraints, which consequently will increase the usability of SystemC for software modeling.

Today SystemC also lacks support for effective direct synthesis of SystemC code. A workaround to this problem is to either manually or automatically translate the SystemC model to an equivalent
CHAPTER 2. BACKGROUND

Figure 2.3: Survey of SystemC usage

HDL model. From there on established HDL tools can be used for synthesis. The Synthesis Working Group (SWG) of OSCI is currently preparing SystemC for future support for synthesis.

SystemC does not either support continuous time models used for analog modeling. This is also under consideration by OSCI and support for analog mixed signal modeling might be available in upcoming versions of SystemC. [6]

2.2 System Design Flow

2.2.1 Traditional system design flow

In traditional system design, architectural design and hardware design are separated from each other. A gap in the development process exists between functional modeling and Register Transfer Level modeling. This interrupt may introduce errors in the translation from functional models to RTL models, errors that could be avoided if the gap can be overcome (figure 2.4).

This gap leaves software and hardware engineers working in their own environments, separated from each other. These environments have their own set of tools, programming languages, verification processes and methodologies. This makes cooperation between the two groups difficult and non-effective, which complicates system optimization and sharing and re-use of IP, test benches etc.

The continuous advancement of System-on-Chip (SoC) of today put additional pressure on traditional system design. Imminent designs have reached a stage where existing methodologies can not deliver efficient construction. Simulations and performance analysis at a low-level get too expensive in terms of time and required processing power.
2.2.2 System design flow with SystemC

System design with SystemC tries to overcome this gap by inserting Transaction Level Modeling (TLM) as a link between the two different worlds of architectural and hardware design. This creates a natural workflow. By using gradual refinement, taking smaller steps instead of giant leaps, fewer errors are likely to be introduced from each move.

A language spoken by both software and hardware designers creates a common development environment. This makes communication between the two groups easier and more efficient. The use of a shared language also means that developed resources can be shared easier which leads to saved development time and a more practical IP library.

To overcome the obstacle of complex SoC, SystemC offers high level modeling with timing through TLM. High abstraction level modeling increases simulation speed substantially compared to conventional RTL simulation. This makes way for saved time during debugging and allows for more efficient architecture exploration.

2.3 RS232

The initial RS232 specification, later known as EIA232, was developed by the Electronic Industries Association (EIA) in 1962. The purpose was to provide a common interface standard for data communications equipment. Since then, several updates and improvements have been made to the standard. [8]

RS232 directly connects two devices with each other over a serial communicating link, providing full-duplex operation. These two devices are defined as Data Terminal Equipment (DTE) and Data
Circuit-terminating Equipment (DCE). Typically the DTE consist of a PC and the DCE of a modem (figure 2.6).

RS232 uses six hardware signals to communicate between the two connected devices:

- DTR - Data Terminal Ready
- DSR - Data Set Ready
- RTS - Request To Send
- CTS - Clear To Send
- TXD - Transmit Data
- RXD - Receive Data
DTR and DSR are used to verify that the two devices are correctly connected to each other. RTS and CTS are used for handshaking, while TXD and RXD are the signals where the data is actually transferred (figure 2.7).

![Diagram of DTE and DCE connections](image1)

**Figure 2.7: Connection of signals**

RS232 implements a serial protocol, which means it transfers one bit per clock cycle. Data is transferred one byte at a time in an 11-bit sequence. The first bit is a start bit, followed by the 8 actual data bits, one parity bit to check the validity of the data and finally a stop bit indicating the end of the transmission (figure 2.8).

![Diagram of RS232 bit sequence](image2)

**Figure 2.8: RS232 bit sequence**

A typical data transmission may contain the chain of events seen in figure 2.9.

![Diagram of a typical data transmission](image3)

**Figure 2.9: A typical data transmission**

### 2.4 JPEG Encoding

The current standard for JPEG (Joint Picture Expert Group) compression is a recommendation issued by ITU (International Telecommunication Union) in 1992. JPEG is an image compression
algorithm for continuous-tone images and photographs. Several variants of the JPEG compression method exist. This section briefly describes the baseline sequential process, which is the most common method. [3]

From research made on the human visual system, psycho visual models have been created. These models explain among other things what we are able to see and to what extent. These models, together with the FDCT (Forward Discrete Cosine Transform), form the foundation of JPEG encoding.

When JPEG compression is applied on continuous-tone images, a high quality output can often be achieved with a compression ratio as high as 1:10 - 1:20. At a compression ratio of 1:5, the output is virtually indistinguishable from the original source. The inclusion of a “lossy” algorithm makes these compression ratios possible. “Lossy” compression means that information is disregarded and lost during the compression phase. This is also the major drawback of JPEG compression.

The JPEG encoding process can be divided into five steps:

- Color conversion
- Sub sampling
- Two-dimensional FDCT
- Quantization
- Entropy encoding

![Figure 2.10: JPEG encoding flow](image)

2.4.1 Image data layout

The image consists of one or several color components or color channels. The individual data elements the source image and the output image consist of are referred to as samples and coefficients. Prior to the JPEG compression is started, the input image is divided into square data blocks of eight by eight samples (figure 2.11). Data blocks representing each of the available color channels are grouped into Minimum Coded Units (MCU). A MCU represents the smallest possible subset of the encoded image. The size and arrangement of the MCU varies depending on which color format is used (figure 2.12).
CHAPTER 2. BACKGROUND

Figure 2.11: Data block and sample layout

<table>
<thead>
<tr>
<th>Color format</th>
<th>MCU configuration</th>
<th>Number of data blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>YUV444</td>
<td>Y U V</td>
<td>3</td>
</tr>
<tr>
<td>YUV422</td>
<td>Y Y U V</td>
<td>4</td>
</tr>
<tr>
<td>YUV411</td>
<td>Y Y Y Y U V</td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 2.12: Comparison of MCU configurations
2.4.2 Color conversion

The studies made on the human visual system showed that humans are more sensitive to changes and details in brightness than in color. JPEG uses a color space that takes advantage of this fact, CIELAB or YUV. YUV consists of three components: Luminance \( Y \), Chrominance A \( U \) and Chrominance B \( V \). Luminance represents brightness, while the two chrominance channels together represent color.

The most common color space for full-color images in computer-based systems is RGB. RGB expresses color in a combination of amounts of Red \( R \), Green \( G \), and Blue \( B \). This means that the source image usually have to go through a color conversion process from RGB to YUV before JPEG compression will take place.

2.4.3 Sub sampling

Since humans are not particular sensitive to color detail, compression can be achieved by removing information from the chrominance channels, without necessarily reducing the visual image quality. This reduction of color data is done through sub sampling. JPEG compression supports three YUV color formats: YUV444, YUV422 and YUV411.

All three YUV formats contain full brightness resolution. YUV444 also contains full color resolution, while in YUV422 color resolution is reduced by half and in YUV411 only a fourth of the original color resolution remain. YUV422 thus decrease the original data by 33 percent and YUV411 decrease it by 50 percent (figure 2.13).

![Figure 2.13: Comparison of YUV formats](image)

2.4.4 2D FDCT

By applying a 2D FDCT on the source image, it is transformed from the spatial domain to the frequency domain. This transformation basically separates visually important image detail from detail of less visual importance (figure 2.14).

For each data block of YUV samples from the source image, a corresponding data block of coefficients is created as a result of the 2D FDCT. Together all coefficients render their data block. Each coefficient adds image detail to the data block. The coefficients are ordered according to their significance to image detail, from the top-left corner diagonally downwards to the bottom-right corner.
\[ S_{vu} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} s_{yx} \cos \left( \frac{(2x+1)u\pi}{16} \right) \cos \left( \frac{(2y+1)v\pi}{16} \right) , \]

where \( C_u, C_v = \begin{cases} 1/\sqrt{2} & \text{for } u, v = 0 \\ 1 & \text{otherwise} \end{cases} \)

Figure 2.14: 2D FDCT formula

### 2.4.5 Quantization

Since the 2D FDCT divided image detail information according to their importance, this knowledge makes it possible to further remove redundant data. Quantization vectors are utilized to individually scale each coefficient after their significance and after the sought compression ratio. The ITU recommendation contains suggested quantization tables that are carefully balanced between image quality and compression ratio.

### 2.4.6 Entropy encoding

**Zigzag scan**

After quantization, each data block is zigzag scanned as a preparation for entropy encoding. The zigzag scan reorganizes the coefficients in an order which makes the entropy encoding more efficient (figure 2.15).

![Zigzag scan sequence](image)

Figure 2.15: Zigzag scan sequence

**Differential Pulse Code Modulation**

In each data block, the upper, leftmost coefficient represents an average value for the entire data block. It is referred to as the *DC coefficient* while the other 63 coefficients are referred to as the *AC*
coefficients.

The DC value is encoded with Differential Pulse Code Modulation (DPCM). DPCM enhances the compression performance for continuous-tone images by encoding the difference between adjacent data blocks’ DC coefficients.

Run-Length Encoding

Run-Length Encoding (RLE) is a lossless encoding algorithm that substitutes sequences of the same value with a symbol. In JPEG compression, runs of AC coefficients with value zero are encoded as the amount of adjacent zeros.

Each non-zero AC coefficient is divided into two values: amplitude level and amplitude. The amplitude level signifies the interval the value of the AC coefficient lies within. The amplitude specifies the position of the value in this interval.

Huffman encoding

Huffman encoding is another lossless encoding algorithm. It achieves compression by substituting values with codes of various lengths after their rate of recurrence. Frequently occurring values are replaced with codes of short length, while seldom occurring values are replaced with codes of longer length. The codes can either be dynamically generated from the content or be based on previously collected statistics from similar content.

The codes are placed in a Huffman table. This table is used for translating the codes to their real value during the decoding process. In JPEG compression, Huffman encoding is employed to further compress the DPCM encoded DC coefficients and the RLE encoded AC coefficients.
Chapter 3

Method

The project was initiated with a study of SystemC. Except for a purely theoretical study, a practical implementation in SystemC was performed as well, in order to provide more practical experience. An UART was chosen as a suitable implementation for this purpose. The experience gained from the UART implementation also gave useful insight for a first draft of the system design flow.

Simultaneously with the UART development, an intense study of the JPEG encoding procedure was conducted. Algorithms used in JPEG encoding, such as the 2D FDCT, were realized in Microsoft Excel. The purpose of this was to get a better understanding of these algorithms, while at the same time offering an opportunity to do some initial algorithm optimization.

A prototype of a JPEG encoder with very limited functionality was created. This prototype was implemented in C and had the aim to further increase the knowledge of the JPEG encoding process. The lessons learned from the C-prototype were used to plan and layout a complete object-oriented JPEG encoder in C++. This encoder was later implemented as a software-prototype running on the ARM CPU of the Integrator Core Module.

The JPEG encoder was implemented in the different abstraction levels SystemC provides. This was done in a step-down refinement manner, moving from a high abstraction level downwards to lower abstraction levels. This made it possible to experience the positive and negative features each of the design levels possesses. Finally, a subset of the JPEG encoder was modeled at RTL level in VHDL to complete the establishment of a system design flow incorporating SystemC.

During the development in C/C++ and SystemC, the functionality of the JPEG encoder was mostly verified visually. Its output images were compared to reference images in terms of visual quality and file size. The VHDL subset was verified with a test bench generated from the SystemC model.

3.1 System Design Flow with SystemC

The complete system design flow can be broken down into several development models. These models serve to define at which abstraction layer a current implementation is modeled. Several definitions of the models with similar classifications exist. The models of this system design flow are based on the definitions found in [2] and used by [1].

Each model possesses unique capabilities and fulfills different purposes in the development process. Together the models create a smooth transition from a purely algorithmic level all the way down to
a low-level, cycle- and pin-accurate model (figure 3.1).

3.1.1 Algorithmic model

An algorithmic model implemented in mathematical software can be used to further develop an idea for a design. The functionality of the design can be either partly or fully confirmed. Reference output might be created for verification purposes in later system models. Underlying algorithms of the design can be optimized in terms of minimization, calculation accuracy and data precision.

3.1.2 Functional models

Limited prototype

An initial prototype with limited functionality can serve as a deeper study into the design. The experience gained from the construction of the limited prototype might aid the layout of the full design.
Specification model

A Specification Model, sometimes referred to as Executable Specification, is the first implementation of the design with complete functionality. The functionality of the design is roughly divided into one or more independent processes, which allows for concurrent behavior. The processes have synchronized communication between themselves. There is still no information about time involved in neither functionality nor communication.

Important lessons about further layout can be learned from this model. This knowledge can be used to create an initial allocation of the functionality of the design on the available processing elements. It also offers early verification of an estimated version of the final system.

3.1.3 Transaction level models

Transaction Level Modeling consists of several models, each one having its own qualities and specific usefulness. Together these models provide natural step-down refinement, which simplifies the move from the functional level downwards to the register transfer level (figure 3.2).

![Abstraction levels](image)

**Figure 3.2: Abstraction levels**

Component-Assembly

At the Component-Assembly (CA) level, the functionality of the design is refined from the Specification Model and mapped onto the available processing elements of the system. An approximate notification of time is added to the processing elements, while communication is still performed without timing in high-level, self-synchronizing channels.

The information gathered from the simulations in CA can help to create a well-balanced system. The computational tasks of the design can be proportionally distributed over the available components of the system, thus making best use of each component’s properties.
Bus Arbitration

In the Bus Arbitration (BA) model, the communication of the design is refined from high-level channels into abstract bus models with approximate timed transactions. The approximate timing in both computation and communication delivers high-speed simulation, while at the same time presenting a sufficient overall-image of the system.

The abstract nature of the BA model offers straightforward investigations of available bus models, until an appropriate bus for the final system has been determined. This flexibility combined with its speed, makes this abstraction level especially useful for system architecture exploration and performance analysis.

On top of this, the resulting model provides a virtual prototype of the final system to the software and hardware designers. This prototype enables both groups to simultaneously develop their respective assignments and encourages increased cooperation by offering simple co-simulation.

Bus Functional

The Bus Functional (BF) level improves the detail of the abstract bus with cycle-accurate timing and a pin-accurate interface. Although the communication channels are interfaced pin-accurately, the computational units can still access them via abstract methods by the usage of intermediate adapters.

The increase of timing information provides the model with enhanced performance analysis at the cost of decreased simulation speed. This model might be useful for fine-tuning of the system and refinement of the implementation.

Cycle-Accurate Computation

The Cycle-Accurate Computation (CAC) model follows the alternative path from the BA model and adds cycle-accurate execution and pin-accurate access to the functional units. Approximate timing and abstract interfacing is still maintained in communication.

Similar to the BF model, this increase of timing information comes at the cost of slower simulations. The motivation for doing a CAC model also corresponds with the motivation for a BF model: system performance fine-tuning and implementation refinement.

3.1.4 Register Transfer Level model

Register Transfer Level (RTL) modeling is the classical approach to computer aided digital design. In RTL, both the processing elements and the communication channels are implemented with cycle-accurate timing and pin-accurate interfacing.

Today there are no tools available that allows for implementation beyond RTL in SystemC. Therefore the models need to be transferred to a well establish development platform. An implementation of the design in a Hardware Description Language (HDL) will gain access to tools that allow for further development to a realizable system.

Simulation speed of RTL models will generally depend on the engine of the actual simulator, not necessarily if the models are implemented in SystemC or HDL. SystemC will therefore not possess
any actual speed advantage at RTL level. What justifies for modeling of RTL in SystemC, is simplified transfer to HDL environment via step-down implementation refinement.

<table>
<thead>
<tr>
<th>Model</th>
<th>Communication</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Scheme</td>
</tr>
<tr>
<td>Specification Model</td>
<td>UT</td>
<td>variable</td>
</tr>
<tr>
<td>Component-Assembly</td>
<td>UT</td>
<td>variable channel</td>
</tr>
<tr>
<td>Bus Arbitration</td>
<td>AT</td>
<td>abstract bus</td>
</tr>
<tr>
<td>Bus Functional</td>
<td>CT</td>
<td>protocol bus</td>
</tr>
<tr>
<td>Cycle-Accurate Computation</td>
<td>AT</td>
<td>abstract bus</td>
</tr>
<tr>
<td>RTL</td>
<td>CT</td>
<td>wired bus</td>
</tr>
</tbody>
</table>

Table 3.1: Characteristics of development models

*UT = UnTimed; AT = Approximate Timed; CT = Cycle Timed*

### 3.2 An UART in SystemC

#### 3.2.1 Specification

This UART (Universal Asynchronous Receiver / Transmitter) is based on the commonly used RS232 standard.

**Full-duplex operation**

This UART offers both bi-directional full-duplex operation, as well as communication in half-duplex mode.

**Protocol implementation**

Although the RS232 specifies signal characteristics and mechanical layout, this UART only implements the RS232 communication protocol.

**System design models**

This UART is modeled as a communicating channel and the design does not incorporate any actual processing element. Therefore some stages of the system design flow will not be modeled.

#### 3.2.2 Implementation

**Specification model**

An initial software prototype was constructed in C++. The main purpose of this phase was to accurately model the communicating protocol. Another purpose was to provide a basic organization of the UART to use for the construction of later models.
The behavior of an UART device was modeled with an UART class, containing two independent sub classes handling transmitting and receiving. Signals were modeled with shared variables. This prototype did not permit concurrent execution and only half-duplex operation.

Component-Assembly model

With the implementation of the Component-Assembly model, the software prototype was transferred to the SystemC environment. The UART device was modeled as a module, containing two internal modules handling transmitting and receiving. Both modules worked and executed independently from each other. Signals were modeled with primitive FIFO channels and full-duplex operation was permitted.

Bus Arbitration model

The Bus Arbitration model transformed the implementation of the UART into a hierarchical channel with approximate timing. Using the structural capabilities of SystemC, the DTR and the DTE were put inside the hierarchical channel. The two UART devices were internally connected with hardware signals. This hid the internal mechanisms for the outside and abstract methods provided a simplified interface to the channel.

Bus Functional model

In the Bus Functional model, clock-triggered processes with cycle-accurate timing enhanced the detail of the approximate timing in the BA model.

3.2.3 Verification

A simple, yet effective, test bench verified the functionality of the models. Using a console window, the test bench provided an “echo”-like functionality. Characters inputted in the console were sent over the UART and printed back in the console upon arrival. This behavior was realized with producer and consumer processes. The test bench remained practically the same during the complete implementation process.

3.3 A JPEG Encoder in SystemC

3.3.1 Specification

This JPEG encoder is based on the baseline sequential process [3], the most common implementation. It does not cover all the details of this definition, but supports the most common applications.

Flexibility

Although the primary purpose of this encoder was to be part of an image sensor system, it is designed with IP reuse in mind. Therefore a flexible design was a priority for simplified adaptation to future
Input

The encoder accepts uncompressed BMP files as input. The supported color model of the source image is 24-bit full color RGB, with 8-bit sample data per color channel. The height and width of the source image has to be a multiple of 16. The encoder can easily be extended to support other file formats and color models, as well as input with unrestricted height and width.

Encoding

The encoder is based on the Discrete Cosine Transform process with sequential encoding. Both interleaved and non-interleaved scans are supported. Huffman encoding with both static and dynamic code generation is supported, with two DC and two AC tables available.

Output

The supported output color formats are YUV444, YUV422, and YUV 411.

3.3.2 Software-hardware prototype

An initial prototype was intended to run on the ARM Integrator platform. This system consisted of one baseboard, providing the underlying system communication through an AMBA bus. On top of that, an ARM processor, mounted on a Core Module, was supposed to run the embedded software part. Finally, a Logic Module equipped with an Altera FPGA was supposed to be the base for the part of the JPEG encoding done in hardware (figure 3.3).

![Figure 3.3: Intended system layout of the software-hardware prototype](image-url)
Component mapping

The components of the JPEG encoder are mapped onto the available resources of the system after their individual characteristics. The assumption was made that the ARM CPU, and its onboard RAM, handle complex and memory-intense algorithms best, while the Altera FPGA is suited to process simple, but computational-heavy algorithms.

Based on this assumption, color conversion, sub sampling, FDCT, quantization and zigzag scan are managed by the FPGA, while the rest of the components are completed by the CPU (figure 3.4).

Figure 3.4: Intended component mapping of software-hardware prototype

Hardware FIFO

This model uses a form of Kahn process network to provide flexibility. Each of the components of the encoder works independently from each other. A simple, yet effective one-way FIFO protocol provides synchronized communication between the components of the FPGA encoder.

This layout makes it possible to individually configure and adapt the components after the available resources of the current system. It also simplifies and encourages architectural exploration, since there is no need to update the communication between the modules when individual internal changes are applied to them.

Fixed-point calculations

The algorithms in the color conversion and the FDCT contain floating-point calculations with many multiplications. Floating-point calculations with multiplications and divisions are very hard for the CPU and/or FPGA to compute. In order to create an implementation that is fast and actually realizable, these calculations need to be performed as fixed-point.
Fixed-point calculations lessen the burden upon the CPU and/or the FPGA by emulating floating-point calculations in integer space, which is generally much easier to compute. Multiplications and divisions can often be substituted with a combination of integer shifts and additions.

Both the color conversion and the FDCT were successfully implemented as complete fixed-point calculations. All multiplications were substituted with combinations of shifts and additions.

**AMBA AHB-Lite**

The high-level models of the JPEG encoder use a FIFO for modeling the communication between the logic module and the core module. In reality, an AMBA AHB-Lite bus interconnects the prototype modules with each other. To complete the software-hardware prototype, the FIFO channel had to be substituted with the corresponding AMBA model.

Unfortunately, due to interface incompatibilities between the JPEG models and the provided AHB-Lite models, the integration of the AHB-Lite could not be completed. The problems could not be resolved before the end of this project. This led to that the FIFO channel was kept as a temporary substitute for the AMBA bus throughout the project.

### 3.3.3 Software prototype

Sadly, the logic module containing the FPGA proved to be defective, which had the consequence that only a software prototype could be created. This prototype was built upon the same prototype environment as the initial prototype, with the exception of the logic module (figure 3.5).

![Figure 3.5: System layout of the software prototype](image)

**Component mapping**

With no FPGA available, not many options for component mapping exists. The complete JPEG encoder is executed on the ARM CPU. The functionality of the prototype is correct, but the per-
formance is limited by the ARM CPU on the prototype board (figure 3.6).

![Component mapping of software prototype](image.png)

Figure 3.6: Component mapping of software prototype

### 3.3.4 Verification

Since JPEG is a compression for images, visual verification can be used for general error checking of the whole system. For more in-depth debugging, a hex editor was used in a combination with the comparison of file size of known reference output. Input and output test vectors generated by the SystemC model was used for verification of individual components in SystemC and VHDL simulation.

### 3.3.5 Simulation

In order to evaluate the speed and efficiency of the SystemC simulator, several tests were performed:

**Encoding at different abstraction levels.** To evaluate the performance of the SystemC simulator, every test run was performed on each abstraction level, beginning from Component-Assembly down to RTL.

**Encoding at different image sizes.** To evaluate the efficiency and scaling of the SystemC simulator, every test run was performed at different image sizes.
CHAPTER 3. METHOD

Setup

- Input: Three differently sized versions of a 24-bit BMP image. See table 3.2 for details.
- Model: FPGA part of Software-Hardware prototype. See figure 3.4.
- Simulator: SystemC reference simulator 2.01.
- Output: YUV422 scaled with the original quantization tables found in [3].
- Each test was performed three times to minimize the amount of error in the speed measurement.

<table>
<thead>
<tr>
<th>Image size</th>
<th>Ratio</th>
<th>Width</th>
<th>Height</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>240x320</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>480x640</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>960x1280</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Overview of image size & ratio.

Results

Measurements that seemed unlikely and did not correspond well to other results of the same test were excluded. The average value of the remaining results from the simulations can be found in table 3.3.

<table>
<thead>
<tr>
<th>Model</th>
<th>Image size</th>
<th>240x320</th>
<th>480x640</th>
<th>960x1280</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component-Assembly</td>
<td></td>
<td>14</td>
<td>54</td>
<td>216</td>
</tr>
<tr>
<td>Bus Arbitration</td>
<td></td>
<td>19</td>
<td>74</td>
<td>297</td>
</tr>
<tr>
<td>Bus Functional</td>
<td></td>
<td>88</td>
<td>349</td>
<td>1389</td>
</tr>
<tr>
<td>Cycle-Accurate Computation</td>
<td></td>
<td>50</td>
<td>199</td>
<td>797</td>
</tr>
<tr>
<td>RTL</td>
<td></td>
<td>104</td>
<td>414</td>
<td>1655</td>
</tr>
</tbody>
</table>

Table 3.3: Simulation time of JPEG encoder. Time in seconds.

By normalizing the results with respect to the Component-assembly level, the speed difference at different abstraction levels of the model can be found in figure 3.7.

<table>
<thead>
<tr>
<th>Level</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>1</td>
</tr>
<tr>
<td>BA</td>
<td>1.38</td>
</tr>
<tr>
<td>BF</td>
<td>6.43</td>
</tr>
<tr>
<td>CAC</td>
<td>3.69</td>
</tr>
<tr>
<td>RTL</td>
<td>7.66</td>
</tr>
</tbody>
</table>

Figure 3.7: Normalized simulation speed at 240x320 pixels image size

To get an understanding of the simulator’s scalability, the results in table 3.3 were scaled according to their size ratio in table 3.2. The results can be found in table 3.4.
Analysis

The results gained from the simulations shows a clear speed advantage of higher abstraction levels over lower abstraction levels. In this particular model, the biggest difference in speed was the use of abstract timing in the communication, instead of using cycle-accurate communication. Where the biggest difference in speed is gained is highly dependant on the model simulated. One can assume a big speed advantage by the utilization of abstraction timing instead of cycle timing, no matter if it is used in computation, communication or both.

The speed difference between abstract timing and cycle timing was not as big as expected, which may have several possible causes:

**Simple communication scheme** This modeled uses different variants of a rather simple FIFO channel at different abstraction levels. A more complex communication implementation, such as a bus, would most likely increase the speed difference considerably.

**System complexity** Another factor that will affect the simulation time is the complexity of the modeled system. A model containing more complex parallel processes and more interacting communication channels than this model, would also increase the speed advantage of abstract timing compared to cycle timing.

The simulator scales well with different input sizes, which may not come as a surprise.

<table>
<thead>
<tr>
<th>Image size</th>
<th>CA</th>
<th>BA</th>
<th>BF</th>
<th>CAC</th>
<th>RTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>240x320</td>
<td>14</td>
<td>19</td>
<td>88</td>
<td>50</td>
<td>104</td>
</tr>
<tr>
<td>480x640</td>
<td>13.5</td>
<td>18.5</td>
<td>87.3</td>
<td>49.8</td>
<td>103.5</td>
</tr>
<tr>
<td>960x1280</td>
<td>13.5</td>
<td>18.6</td>
<td>86.8</td>
<td>49.8</td>
<td>103.5</td>
</tr>
</tbody>
</table>

Table 3.4: Scaled simulation time of JPEG encoder. *Time in seconds.*
Chapter 4

Discussion

4.1 SystemC for System Design

The foundation on C++ makes the transition to SystemC relatively painless for designers familiar with C, C++ or comparable programming languages. Time does not need to be spent on learning a completely new language and development environment. Time can rather be spent on learning the progressive methodologies and new opportunities SystemC offers. For hardware designers with little or no previous experience in high-level programming, the transition can be somewhat more difficult.

The architecture of SystemC offers well-organized system design, with clear structural layout at all levels. This makes designs in SystemC easy to visually imagine. SystemC is prepared for future design advancements through an easily extended architecture. The distinct separation between communication and computation makes high-level design methodologies, such as Transaction-level modeling, a reality.

SystemC provides all data types necessary to accurately model designs at high abstraction levels, as well as at low abstraction levels. Data types for special purposes, such as fixed-point calculations, are available and widen the appliance of SystemC.

Although being a relatively new language, SystemC already attracts strong interest from the industry and is supported by major EDA vendors. The fact that is open-source greatly increases its chances to unify system design and to eliminate the need for vendor-specific tools.

SystemC does not yet fully support synthesis. The tools available for SystemC design have not yet reached the same level of maturity as corresponding HDL tools. Therefore, SystemC is not ready to completely replace existing HDL methodologies at this point. It should rather be used in conjunction with established workflows.

At this time is SystemC less suitable for complex software modeling. Such models, including real-time operation systems, require support for advanced thread operations and timing constraints.

4.2 System Design Flow with SystemC

The proposed system design flow offers a solution to overcome problems present in traditional workflows. Through the use of SystemC and TLM, it provides a common ground for the different designers.
involved. This addresses the need of increased co-operation between the two groups. Developed resources such as test benches and programming code could be reused with less effort, thus reducing the need for multiple implementations.

TLM and implementation through refinement proved to be a pleasing method to overcome the gap, present in traditional design, between functional models and RTL models. The development models all serve their specific purposes and have individual advantages and disadvantages. Together they lay an evenly spaced path from algorithmic level down to RTL.

Out of the four TLM models, perhaps the Bus arbitration model is the most interesting. This model introduces a complete, high-level model of the system, which offers early development and simulation of embedded software. The approximate timing in both communication and computation is a balanced compromise between detail and simulation speed. Simulations of the JPEG encoder clearly showed that high-level modeling in SystemC offers an important speed advantage over RTL simulation. This opens doors to improved architecture exploration and performance optimization.

However, correct timing information is required to take full advantage of TLM. A design without specific timing requirements is problematic to accurately model in TLM. A source for this information could be the synthesis result of an at least partial implementation in HDL. The timing information could then be fed back to higher level models, where system optimization could take place.

When SystemC is first employed, all new designs have to be completely implemented throughout the entire development flow. This increases the time needed for actual design, which may leave less time for architecture exploration and performance modeling. If each element of the design is carefully implemented according to the development models, they can be reused without much effort in future projects. This is especially true for communication elements, which probably will be reused more frequently than its computational counterparts. Once an extensive IP library has been constructed, the focus can shift from design to architecture exploration and performance modeling.

### 4.3 Future Work

Organizations interested in using SystemC should consider investigating in an appropriate IP standardization, at least on a local level. Shared interface access methods, especially for communicating elements, simplify IP reuse to a great extent. If strong IP standardizations are used, automation of high-level tasks could become reality.

Continuous evolution of SystemC could extend it with new, important features, such as high-level synthesis and software modeling. At that point, a re-evaluation of the design flow could be made to incorporate these new ideas.

Once a replacement for the damaged logic module has been provided, a complete implementation of the software-hardware prototype would provide additional valuable insight into the design flow.

The JPEG encoder could easily be extended with more accepted input image formats as well as color formats. It would also require little effort to extend it with capabilities for handling freely sized images.
4.4 Conclusion

The primary purpose of this thesis was to investigate how SystemC could change and improve system design. Based on the studies and the implementations made in this thesis, the following conclusions could be made:

**SystemC have been evaluated.** An UART and a JPEG encoder were successfully implemented as a part of this evaluation. SystemC overcomes issues present in current design methodologies and offers high-level modeling. This makes it suitable for high-performance system modeling. The current incarnation of SystemC does not replace HDLs, it completes them.

**A complete system design flow with SystemC has been established.** Transaction level modeling is the key to the improvements SystemC offers. Approximate timing and a clear separation between communication and computation are some important features of TLM. With standardized IP interface, the advantages of IP reuse will increase over time.

**A JPEG encoder prototype was created.** A software prototype of the JPEG encoder could successfully be created. The functionality of the prototype was correct, but the performance was limited by the CPU. An intended software-hardware prototype could unfortunately not be completed due to defective hardware.
Bibliography


http://www.ics.uci.edu/~gajski/presentation/Transaction.ppt
{Acc. 2005-01-07}.

http://www.w3.org/Graphics/JPEG/itu-t81.pdf
{Acc. 2005-01-07}.

http://www.systemc.org/projects/sitedocs/document/v201_Func_Spec/en/1
{Acc. 2005-01-07}.

{Acc. 2005-01-07}.

{Acc. 2005-01-07}.

http://www.doulos.com/systemc_report/
{Acc. 2005-01-07}.

{Acc. 2005-01-07}. 
Appendix A

Optimization of FDCT algorithm

The original 2D FDCT algorithm (see figure 2.14) contains a large amount of multiplications and additions and is therefore quite computational heavy. Luckily, this algorithm can be optimized using common algebraic minimization.

The 2D FDCT algorithm in JPEG is applied on an 8x8 sample block. The 2D FDCT can be divided into 16 1D FDCT calculations over eight 8x1 sample arrays and eight 1x8 coefficient arrays. The result of the optimized 1D FDCT algorithm (figure A.2) can be seen in table A.1.

![Input and output 1D FDCT array.](image)

![Input and output 1D FDCT array.](image)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Operations</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Multiplications</td>
<td>Additions</td>
</tr>
<tr>
<td>Original 1D FDCT</td>
<td>64</td>
<td>56</td>
</tr>
<tr>
<td>Optimized 1D FDCT</td>
<td>22</td>
<td>28</td>
</tr>
<tr>
<td>Original 2D FDCT</td>
<td>1024</td>
<td>896</td>
</tr>
<tr>
<td>Optimized 2D FDCT</td>
<td>352</td>
<td>448</td>
</tr>
</tbody>
</table>

Table A.1: Result of FDCT optimization.
$$\text{Sum}_x = \begin{cases} s_x - s_{7-x} & \text{for } 0 \geq x < 4 \\ s_x + s_{7-x} & \text{for } 4 \geq x < 8 \end{cases}$$

$$\text{Sum}_8 = \text{Sum}_4 + \text{Sum}_7$$

$$\text{Sum}_9 = \text{Sum}_5 + \text{Sum}_6$$

$$\text{Sum}_{10} = \text{Sum}_4 - \text{Sum}_7$$

$$\text{Sum}_{11} = \text{Sum}_5 - \text{Sum}_6$$

$$C_0 = \frac{1}{\sqrt{8}} \times (\text{Sum}_8 + \text{Sum}_9)$$

$$C_1 = 0.5 \times \left( \cos \frac{\pi}{16} \times \text{Sum}_0 + \cos \frac{3\pi}{16} \times \text{Sum}_1 + \cos \frac{5\pi}{16} \times \text{Sum}_2 + \cos \frac{7\pi}{16} \times \text{Sum}_3 \right)$$

$$C_2 = 0.5 \times \left( \cos \frac{2\pi}{16} \times \text{Sum}_{10} + \cos \frac{6\pi}{16} \times \text{Sum}_{11} \right)$$

$$C_3 = 0.5 \times \left( -\cos \frac{\pi}{16} \times \text{Sum}_2 + \cos \frac{3\pi}{16} \times \text{Sum}_0 - \cos \frac{5\pi}{16} \times \text{Sum}_3 - \cos \frac{7\pi}{16} \times \text{Sum}_1 \right)$$

$$C_4 = \frac{1}{\sqrt{8}} \times (\text{Sum}_8 - \text{Sum}_9)$$

$$C_5 = 0.5 \times \left( -\cos \frac{\pi}{16} \times \text{Sum}_1 + \cos \frac{3\pi}{16} \times \text{Sum}_3 + \cos \frac{5\pi}{16} \times \text{Sum}_0 + \cos \frac{7\pi}{16} \times \text{Sum}_2 \right)$$

$$C_6 = 0.5 \times \left( -\cos \frac{2\pi}{16} \times \text{Sum}_{11} + \cos \frac{6\pi}{16} \times \text{Sum}_{10} \right)$$

$$C_7 = 0.5 \times \left( -\cos \frac{\pi}{16} \times \text{Sum}_3 + \cos \frac{3\pi}{16} \times \text{Sum}_2 - \cos \frac{5\pi}{16} \times \text{Sum}_1 + 0.5 \times \cos \frac{7\pi}{16} \times \text{Sum}_0 \right)$$

Figure A.2: Optimized 1D FDCT formula