C-based design environment for ICs : feasibility study of TSS for an industrial design environment

Westeneng, H.J.

Award date:
2001

TU/e technische universiteit eindhoven

Faculty of Electrical Engineering
Section Design Technology For Electronic Systems (ICS/ES)
ICS-ES 760

Master's Thesis

C-BASED DESIGN ENVIRONMENT FOR ICS

Feasibility study of TSS for an industrial design environment.

H.J. Westeneng

Coach:	ir. H.G.M. Janssen (Philips Semiconductors Nijmegen)
	Ir. A. Stuivenwold (Philips Semiconductors Nijmegen)
Supervisor:	prof.dr.ir. J.L. van Meerbergen
Date:	April 2001

C-based design environment for ICs

Feasibility study of TSS for an industrial design environment

Student: H.J. Westeneng
Eindhoven University of Technology
Department Electrical Engineering
Information- and Communication Systems (ICS)
St. id. no. : 453696
h.j.westeneng@stud.tue.nl

Supervisors: ir. H.G.M. Janssen
Philips Semiconductors Nijmegen
Business Line Video (BL-Video)
Design flow Services

prof. dr. ir. J.L. van Meerbergen
Eindhoven University of Technology
Department Electrical Engineering
Information- and Communication Systems (ICS)
Philips Research Laboratories Eindhoven
Embedded Systems Architectures on Silicon (ESAS)

ir. A. Stuivenwold
Philips Semiconductors Nijmegen
Business Line Infotainment Systems (BL ITS)

Preface

This thesis is the result of the graduation project for fulfilling the requirements for the degree of Master of Science at the Eindhoven University of Technology. This graduation project is carried out at Philips Semiconductors in Nijmegen, business line Video. I got support of Philips Research in Eindhoven where I stayed for one day a week.
I would like to thank my daily supervisor Herman Janssen from Philips Semiconductors in Nijmegen and my graduation professor J. van Meerbergen who was in the last part of my graduation project my supervisor of the university as well. For giving the possibility to do a graduation project at Philips Semiconductors in Nijmegen I would like to thank Armand Stuivenwold. For support in TSS if have to thank Jeffrey Kang at Philips Research in Eindhoven. And I would like to express my gratitude all people in Nijmegen who have spent time for me for

Nijmegen, 2001
Henri Westeneng

Contents

PREFACE **2**

SAMENVATTING **5**

SUMMARY **5**

1 INTRODUCTION **6**

1.1 INDUSTRIAL DESIGN ENVIRONMENT 7
1.2 THE ADOC PROJECT 7
1.3 CURRENT SITUATION 8
1.4 CO-DESIGN AND CO-VERIFICATION 9
1.5 DEFINITION OF ABSTRACTION LEVELS 9
1.6 REPORT OUTLINE 11

2 PROBLEM STATEMENT **12**

3 PROJECT DESCRIPTION **14**

4 TOOL FOR SYSTEM SIMULATION **15**

4.1 INTRODUCTION TO TSS 15
4.2 TIMING 16
4.2.1 Processes *16*
4.2.2 Simulation loop *17*
4.3 VIEWPORTS OF AN INSTANCE 18
4.4 DIFFERENCES BETWEEN A MODEL IN TSS AND VHDL 18
4.5 ABSTRACTION LEVEL OF A COMMON TSS MODEL 19
4.6 PROJECTS AND APPLICATION AREAS 19
4.7 C-BASED SIGN-OFF 20

5 DIGITAL OUTPUT PROCESSOR **21**

5.1 SCOPE 21
5.2 FUNCTIONS OF DOP 22
5.2.1 Beam Current Limiter *22*
5.2.2 Peak White Limiter *24*
5.2.3 Graphics Video blending *25*
5.2.4 Scan Velocity Modulation and Dynamic Contrast Control *25*

6 MODELING DIGITAL OUTPUT PROCESSOR **27**

6.1 INTRODUCTION MODELING SYSTEMS 27
6.2 MODELING DOP 27
6.2.1 Comparison TSS-model and VHDL-RTL description of DOP *29*

7 EXPERIMENTS **30**

7.1 DOP STAND-ALONE 30
7.1.1 Simulation DOP stand-alone *31*
7.1.2 Conclusion DOP stand-alone *35*
7.2 DOP IN ADOC FRAME WORK 38
7.2.1 (co-) Simulations DOP in TSS ADOC-framework *39*
7.2.2 Conclusions DOP in TSS ADOC-framework *39*

8 ARGUMENTS AND ANSWERS ON RESEARCH QUESTIONS **41**

8.1 SIMULATION OF BUS-BASED ARCHITECTURES WITH TSS 41
8.2 ARGUMENTS TO USE TSS 42
8.3 ANSWERS RESEARCH QUESTIONS 44

9 CONCLUSION **51**

10 RECOMMENDATION **52**

GLOSSARY **53**

REFERENCES **54**

Samenvatting

Deze scriptie beschrijft de resultaten van een haalbaarheidsonderzoek van TSS (Tool voor Systeem Simulatie) in een industriële ontwerp omgeving. Dit is een software programma welke binnen Philips ontworpen is en ook binnen Philips gebruikt wordt om systeem simulaties uit te voeren. Daarbij worden modellen van IP blokken gebruikt, welke in de programmeer taal C beschreven zijn.
Met de resultaten van dit onderzoek wordt ondersteunt dat met simulaties in TSS een aanzienlijke simulatie tijd winst is te behalen in vergelijking met simulaties met beschrijvingen in VHDL-RTL.
In VHDL-RTL beschrijvingen van chips in een industriële design omgeving blijken een groot aantal (klok) processen gebruikt te worden. Terwijl in een TSS model één klok proces per klok frequentie is gedefinieerd. Dit blijkt een belangrijke factor te zijn waarom simulaties in TSS aanzienlijk sneller zijn.
TSS modellen kunnen bit true en cycle true gemaakt worden en hebben een iets hogere abstractie niveau dan modellen in VHDL-behavioural.
Door snelle simulaties, minder regels code en het hoger abstractie niveau van TSS is de turn around time kort. In een vroeg ontwerp stadium kan het systeem geverifieerd worden op een hoger abstractie niveau. Waarbij ook de driver software kan worden geverifieerd, zodat in een vroeg stadium integratie fouten gevonden kunnen worden. Daarnaast kunnen modellen kunnen door de hardware designers gemaakt en gebruikt worden voor prototypen en specificatie.

Summary

This thesis describes the results of a feasibility study of TSS (Tool for System Simulation) for an industrial design environment. This is a software program which is developed within Philips and is currently in use within Philips.
TSS is used for system simulations. For these simulations are models used of IP blocks, which are described in the C-language.
Results of this feasibility study show that simulations in TSS achieve a considerable simulation time profit in comparison with simulations with descriptions in VHDL-RTL.
Common VHDL-RTL descriptions of chips in an industrial design environment contains a large amount of (clock) processes. While in TSS one clock process is defined for each clock frequency. This proves to be a significant factor why simulations in TSS are considerable faster.
Common TSS models can be made bit true and cycle true and have a slightly higher abstraction level than VHDL-behavioural models.
Due fast simulations, less lines code and the higher abstraction level of TSS is the turn around time short.
The whole system can be verified in an early design stadium and at a higher abstraction level. In which driver software can be verified, so integration mistakes can be found earlier.
Hardware designers can make an use TSS models for prototyping and specification.

1 Introduction

In general there are the following trends in the chip industry [16]:

1. Product life spans are getting shorter
2. Design complexity increases with 70 % per year.
3. The efficiency of CAD-tools increases with 21 % per year

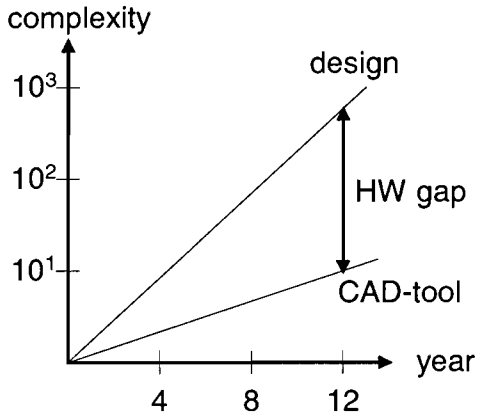

Figure 1-1: Hardware gap

The first trend is a consequence of economical and social processes, including globalisation of markets and increasing consumer demands. The second trend is just another formulation of Moore's Law. It's obvious that trend 2 and 3 will increase the design time of chips when current design methodologies will be used. These trends will cause a hardware gap, which is shown in Figure 1-1.

To meet these objectives higher abstraction levels of systems is needed. Therefor new design methodologies are needed. System simulation seems to be a promising method by using models with higher level than Register Transfer Level (RTL). This means that higher level models have to be available or have to be designed. This method gives the opportunity to verify systems in an early stage of the design process, before many man-years are spent to implement these systems in silicon.

These high level descriptions can be seen as simulatable specifications and these descriptions can be adapted more easily than models at RTL architecture level and will simulate faster.

Top-down methodology means that designers will start with designing a system at a more abstract level, before making an implementation at sub-system level. The different abstraction levels before making an implementation will be more explained in chapter 1.5.

In Philips Semiconductors and all over the world there is an increasing recognition of an enlarging gap between the possibilities of silicon technology and the design productivity. To cope with this problem reuse will become an important issue. It will be much more frequently required from designs of IC-blocks that the accompanying higher-level descriptions will be delivered as well.

Another need for high level models is raised by software designers who want to start as early as possible. With the help of high-level models of hardware, software designers can develop and verify software before these systems are implemented in silicon, so integration problems can be found earlier.

This thesis describes the results of the feasibility study of a tool named TSS (Tool for System Simulation). This tool uses higher level models for simulation of hardware blocks.

1.1 Industrial design environment

This feasibility study is done in an industrial design environment at Philips Semiconductors in Nijmegen by the Digital Television System group (DTS-group) group, which is part of Business Line (BL) Video.
Products designed by BL-Video are meant for the high, mid and low range consumer market.
To be able to compete with other companies Time To Market is a very important item in the consumer market. This has immediate consequences for the design environment. There will be a pressure for decreasing design time of the more and more complex chips.
The consumer market is a high volume market with sales of millions of chips. Therefor the price/die is extra important. With a decreasing price/die ratio, the die size has to be decreased. For this reason dedicated hardware will be designed to satisfy this constraint. This has influence for ReUse. Designers have to make more generic blocks when these blocks will be ReUsed later in derivatives, to satisfy different customers. But generic blocks will contain more overhead and through this overhead die size will increase. Therefor ReUse is currently not used with in the DTS group, without customising the blocks.
Currently BL-Video is developing the ADOC (Analogue Digital One Chip). This is a chip which will be used in a TV for analogue broadcasting and meant for the mid range consumer market. For this chip the DTS group is developing a Digital Output Processor (DOP). Blocks like DOP will be used in stream-based architectures. This means that a stream of data (e.g. video frames) will be processed through blocks which are placed in succession. Output of a certain block will directly be connected to the input of its successor. Control information will be sent with the help of a common bus. It is this stream based architecture that is used for this feasibility study of TSS.

1.2 The ADOC project

Currently Philips Semiconductors is developing the ADOC. This chip is part of the SALSA system, a highly integrated solution for TV video and sound processing and TV control, aimed at the mid range market segment. SALSA is a system for reception of analogue broadcast signals.
The ADOC chip will replace the basic analogue one chip receivers like UOC (Ultimate One Chip) by a new concept that implements all basic functions with digital technology. Only a few functions are implemented analogue in the MPIF IC (Multiple Platform Interface) such as IF functionality. Except from the digitisation of all basic functions the ADOC integrates digital sound processing, and feature processing like e.g. Scan Rate conversion (100 Hz etc.)
The architecture of the ADOC is divided in three parts: video core, audio core and the control Core. The video core contains the functions for video processing. The audio core contains the functions for decoding signals for the sound standards NICAM. A2, BTSC Stereo and Japan stereo. The control core contains the functions for controlling the ADOC set by use of an embedded processor. The Digital Output Processor (DOP) is a block, which is the last block in the Video Core. The DOP is connected to the Cathode Rate Tube via an external video amplifier

1.3 Current situation

Major decisions at system level of the ADOC have been taken mainly based on experience and estimation. The tools, which have been used here, are mainly Excel (spreadsheet), for rough calculations and PowerPoint for building block-diagrams. System simulations have been executed for parts of the system after the system study was done. Of these parts behavioural-VHDL is written.
When starting designing the ADOC, no abstract functional model of the whole ADOC was made. Only of some parts of the VideoCore algorithm models in C were developed, just for functional verification of algorithms, not performance etc..
For designing the ADOC, an amount of legacy blocks where available for the control core. But the video core has to be designed from scratch, no legacy blocks were available.
The designers of the DOP have mainly followed a bottom-up-design approach. First sub-blocks of the DOP are designed by using specifications written in natural language. These sub-blocks are connected together afterwards, to form the DOP. Designers will start dependent on the complexity with writing behaviour of a sub-block. Currently for them it is the fastest methodology to design chips. Because more functionality can be integrated on a chip every year, chips become more and more complex. Due to this complexity there is a need for an early verification of the system. The problem will be that it takes a lot of time before the whole system can be verified when a bottom-up-approach is used.
Behavioural models in VHDL are build for some of the sub-blocks of the DOP. Higher simulation speed of VHDL- behavioural through fewer lines of code make it possible to verify ideas faster.
Currently software designers do mainly hardware/software co-verification when the hardware is implemented in silicon or is available at RTL architecture. Integration problems will appear mainly from that time. The turn around time will be slow on this level.
Simulations of the complete DOP were done at RT-architecture level using the tool NcSim. It took about 6 hours to simulate one single frame. An accelerator system, Cobalt, will be used for verification. Additional breadboarding is used for real time verification of some parts of the ADOC. For these verifications RTL-architecture descriptions of the design have to be available as well.
Common appearing system problems have to do with bus load, execution speed of software and connections between blocks, which currently will only visible after the entire system is finished.
The group DTS of BL-Video has almost no experience with TSS and doesn't use this tool for designing their chips. This thesis will describe the added value of TSS will for DTS.

1.4 Co-design and Co-verification

Co-design and co-verification are consecutive steps in the system level design flow. In the co-design step, a system is partitioned into hardware and software, performance analysis needs to be done and simulation is done in order to verify the system specifications. In this step, hardware is modeled in high-level language descriptions like C/C++. With the design process going on, the abstraction level is going down. By co-verification, the system architecture is fixed and HW/SW partitioning has been performed. The refined implementation of hardware and software is still to be done. Before physical implementation (place and route) the functionality of the software and the hardware, both working together, has to be verified. This is called co-verification.
Crucial to the approach is however that most of the system in TSS is not modeled in HDL. Because of this, higher performance of the simulation runs can be obtained. And software can run on a TSS-model of a processor.
Commercial co-verification tools (Seamless) however, model the major part of the system, including the memories, in HDL. Mostly only the processor is modeled in C TSS is mostly used as co-design tool, while Seamless are mostly used as co-verification tools.

1.5 Definition of abstraction levels

The system level design flow can be seen as a set of transformations from one abstraction level to the next abstraction level. System simulation may be used as a verification step for the transformation of the next level.
Where the levels of the flow are defined as [3, 9]:

1. **A non-executable specification of (parts of) the system**
   This can be in a natural language or for instance a part of a standard.
2. **Functional specification of system level components**
   This specification is an executable description of the system or parts of the system which have to be designed. Most times this specification consists of a big sequential program, for example in C. The creator of this description has no knowledge about the hardware on which the system has to be implemented. Think here of designers that are developing picture improvement algorithms, set-top box functionality, or new compression algorithms for audio.
3. **- Set of Assembled Communicating Processes**
   Now the specification in level 2 is changed such that the original description is divided into a number of parallel processes, with explicit communication between them by using channels. The existence of different kinds of processes for DSP and control tasks requires different computational models. For specifying DSP tasks many times Data Flow Graphs (DFG) are used. These are abstract models, which represent operations and data dependencies. While for specifying control tasks Finite State Machines (FSM) are used.
   - Development of concept architecture
   Concept architecture is an *architecture template* whereby a decision is taken for basic concepts involving memory, communication and synchronisation.
4. **Architectural platform mapping**
   Now the communicating processes are allocated to elements of an architecture. An architecture instance is derived of the concept architecture. For every element in

the architecture there is made a specific decision like a certain processor or memory. The architecture instance contains hardware and software. The behavioural descriptions of the set of assembled communicating processes is translated into higher level behavioural descriptions of the subsystems and a description of the interaction or communication between the subsystems. The hardware of the subsystems is defined in a higher level behavioural description in a language like C/C++. For instance the subsystems can be special hardware ASIC modules, a general purpose processor, a DSP etc..

5. **RTL Behavioural Hardware (HW) and Software (SW) implementation**
   Architectural platform mapping is making high level behavioural descriptions of the hardware of the subsystems in a language like C/C++. Descriptions at this level can be bit and cycle true if needed.
   RTL-behaviour specifies all operations, which could be executed in a certain clock cycle, without describing which resources (adders, multipliers) will be used. No scheduling is made. A RTL-behavioural description can be bit-true and cycle true. It will describe *'what'* has to be done. Time concept will be the clock and data types will be integers or floats etc. For software one can think of Host Code Emulation (HCE) or cross-compiled code on an Instruction Set Model (ISM). For hardware you can think of VHDL-behavioural model.
6. **RTL architecture (Cycle true and bit true implementation)**
   For cross-compiled software code, cycle true and bit true behaviour is a must. RTL-architecture of hardware consists of registers and combinatorial logic. And specifies all operations which have to be executed in a certain clock cycle and describes on which resources these execution has to be done. Therefor a scheduling is made. So such descriptions are cycle and bit true. And will describe *'how'* the *'what'* has to be done. The intention is that RTL-architecture will be synthesisable.
7. **Realisation**
   This is the lowest level of description for the System Level Design Flow. The hardware part of the design is described in terms of gates. For the cross-compiled code the target embedded processor has to be added to the result of the logic synthesis process of the RTL description. Also RAM's, ROM's and other IP cores are to be added to the final gate level description to obtain a detailed implementation of the complete system. For some parts of the system (IP blocks) a gate level description is not available. Examples are RAM, ROM and perhaps processors. For these modules only higher level models and the layout are available.

Figure 1-2 shows a design process and the feedback , which is called the verification. This verification can be done by doing simulation.

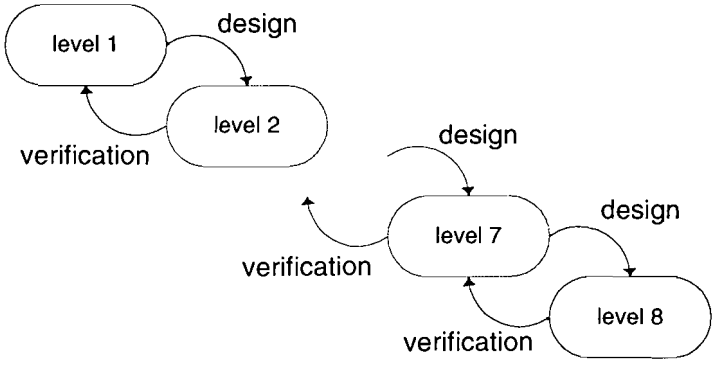

Figure 1-2: Design and verification

1.6 Report Outline

The chapters 2 and 3 explain the problem statement and the project description of this thesis respectively. The tool TSS will be introduced and important properties are explained in chapter 4. The functions of the DOP which are modeled will be explained more in detail in chapter 5. Chapter 6 describes the modeling of the DOP. And chapter 7 shows the results of the experiments. While in chapter 8 answers will be given on research questions by using the arguments which are explained in the same chapter. The conclusion is described in chapter 9 and the recommendations in chapter 10.

2 Problem Statement

Currently there is a division between the algorithm world and the VLSI hardware world in the entire process of making a chip.
The algorithm world uses C-like languages while the hardware world uses VHDL-like languages.
Descriptions at algorithm level will not be cycle true but can be bit true if needed.
Hardware descriptions in VHDL can be bit and cycle true.
Figure 2-1 shows the different abstraction levels of a system and shows the divisions between the two worlds.
Definitions the abstraction levels are explained in chapter 1.5. Algorithm level can be seen as a combination of the levels 3, 4 and 5 of the defined levels of this chapter.

Abstraction level	Language	Accurateness
Algorithm	C	bit-true
RTL behavioural	VHDL	bit-true and/or cycle true
Architectural	VHDL	bit-true and cycle true

Figure 2-1: Abstraction levels of a system

Characteristics:

- Double specification environment is needed, one for the algorithm and one for RTL behavioural and RTL architectural. Consequently there consists a poor link between software and hardware.
- Currently mainly at RTL architecture level the hardware of a IP-block and the total hardware of a system will be verified, after a lot of design effort .
- higher simulation speed at higher abstraction levels.
- lower complexity at higher abstraction levels.
- faster turn-around time at higher abstraction levels.

At this moment there is a new trend that hardware will be described in a C like language because the system and software world use C like languages already.
An embedded system consists of three disciplines: System, SW and Silicon. System and SW uses a common C-like description language. Only Silicon uses VHDL-like languages. The different disciplines will use their own abstraction levels. The system discipline will work at algorithm level, and HW at RT behavioural level and RT architecture level.
C like languages can be used for hardware descriptions, there are a few problems, if it is possible to describe in the language parallelism and bit true types. Further a link to implementation (synthesisability) should be available.

It's possible to make a kind of a road map which shows the trend of C-like languages for HW descriptions. This road map can be divided into 3 phases. Phase 1 is the current situation, where there is a separation between algorithm world and hardware world. Phase 2 shows the situation where the separation line is gone one step down.

This is possible when TSS will be used. Models for TSS are written in C-like languages. Because TSS models are not synthesisable, a synthesisable VHDL description at RT architectural level is still needed.
Probably in the future synthesisable C/C++ descriptions are possible. SystemC can be a candidate.
In Figure 2-2 shows the abstraction levels and the different descriptions which will be possible in the different phases.
It's obvious that the feasibility study will be concentrated on phase 2.

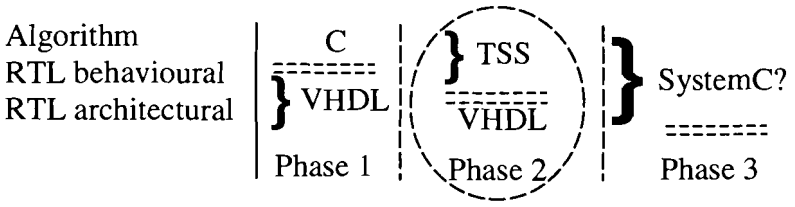

Figure 2-2: Different descriptions for several levels of abstraction

3 Project Description

TSS with its current models appears to have a large added value in the world of IC design. Specially for Compute platforms and bus architectures. With preservation of a reasonable simulation speed the performance of a candidate architecture can be estimated, before this architecture will be implemented. With the help of this tool it is possible to simulate a candidate architecture with different parameters.
This tool will be the subject of this thesis and therefore the central question of this thesis is:

What is the added value of TSS in an IC industrial design environment?

The task is to study TSS, and to determine what is needed for effective use of TSS. This will be done with the help of a module, which is being implemented in silicon at this moment. The model mentioned is the DOP, of which no high-level description at this moment exists. This module will be used in different projects in the future.
The knowledge, which will be acquired by using TSS for this kind of descriptions, is passed to the Digital Television Systems group (DTS-group).

The research questions of this thesis are:

What is the added value of TSS for streaming based platforms like the VideoCore in an ADOC architecture for projects in the future? :

1 What is the impact on verification (time and flow)?
2 What is the development effort for making a TSS model?
3 What is the impact on reuse, is it less difficult to make derivatives?
4 What is the importance of such a model for early software development for a system like the ADOC on a platform as PC?

4 Tool for System Simulation

The purpose of this chapter is to give an overview of TSS (Tool for System Simulation). For more details see [1,2,3,4]. This chapter will explain the TSS simulation system. Important properties of TSS and difference between VHDL and TSS will described as well. An answer will be given which level a common TSS model has.

4.1 Introduction to TSS

TSS is an event based C-simulation framework and is developed at Natlab. This tool allows a higher level description of complex hardware architectures in the C programming language. The models are more abstract than RT architecture level and gate level. TSS models can be bit true and cycle true.

The simulation system, which is shown in Figure 4.1-1, consists of two components:

- The kernel is, in fact, the TSS (operating) system. It provides the clock, it transfers signals originating in a module to other modules as described by the netlist and it takes care that the functionality as described in the various modules is computed whenever certain module inputs change. So the kernel implements the communication and synchronisation of models.
- A number of models, which have to be written in ANSI-C. These modules contains processes which implements the module functionality. Currently a library of modules with an PI-bus PI-bus interface exists for TSS.

The simulator kernel and the system modules are available separately. The user may have developed some user modules, which are compiled to object files by an ANSI-C compiler, and put together in a library. The two parts are linked together by a linker called TSSBUILD to create a single simulator executable. The simulation system executable will take a netlist file as input which describes the system architecture.

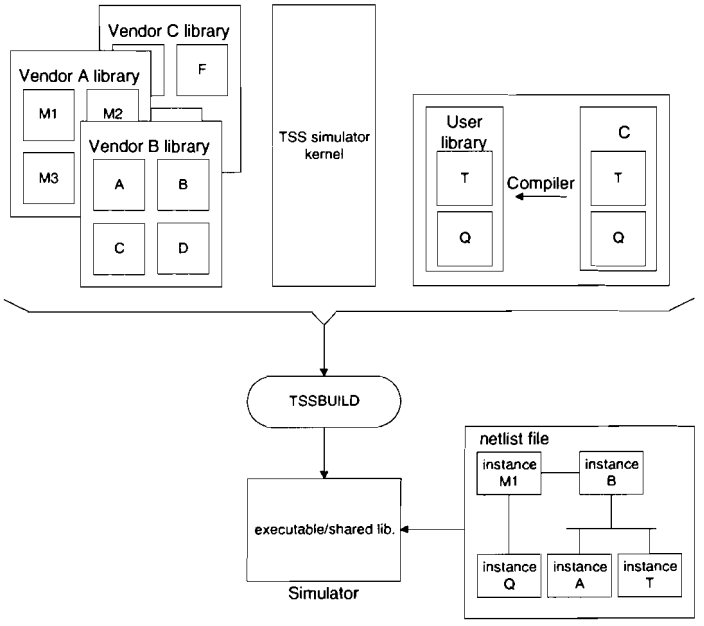

Figure 4.1-1: TSS simulation system

TSS can be used in the early design phase of the system development cycle. System architecture development typically requires extensive performance analysis, preferably with running parts of the application. The speed accuracy of the simulator also allows the development of low level software like device drivers. Furthermore

the C-models can be co-simulated with Speedsim, NcSim, Verilog-XL and Leapfrog to speed up the simulation speed in comparison with simulation of only models in VHDL. This enables multilevel simulation with TSS.
TSS uses Tcl (part of Tcl/Tk) as user interface. The user interface supports variables, procedures, loop constructs and monitoring of viewports. TCL is an interface between the user interface and the C models you work with.
TSS is supported by ED&T in Eindhoven. CTO-RTG in Eindhoven and Bangalore develop TSS models. And currently CTO-DTG in Southhampton is developing a translation flow from RTL-architecture described in VHDL to a TSS-model.

4.2 Timing

TSS descriptions are not sufficiently detailed to decide whether timing specs in therms of nsec are met. For that level of timing, VHDL or an equivalent gate level description is required. TSS can simulate at the clock tic level: under the assumption that what is described within a clock pulse can actually be realised within a clock pulse, TSS will simulate "clock tic accurate".

4.2.1 Processes

Processes model the functionality of a module. A process is defined by means of the function 'tss_define_process'. The scope argument defines the instance the process belongs to. This must be the instance handle of the instance defining the process. Processes are triggered by means of events on the ports in the sensitivity or explicitly through the 'tss_trigger' function.
TSS models can have multiple clock domains. But mainly one clock process for each clock frequency will be used.
Figure 4.2.1-1 shows a module with a number of inputs, a number of outputs and the module's "state": (the contents of) the module's collection of flip-flops and registers.
Two "processes" are depicted in this example: a clock process and an other process, which is sensitive to signal S. All such processes together derive next module states and new values for auxiliary signals from inputs, present module state and auxiliary signals.

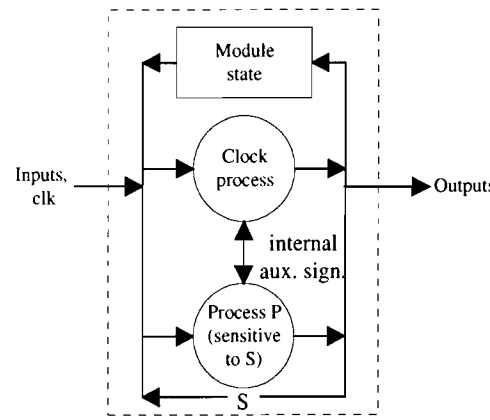

Figure 4.2.1-1: TSS module

It is customary to call those variables that express flip-flops and registers "state variables" and all other variables "auxiliary variables". This reflects state variables to be changed in the clock process. All other variables are than handled in the other process(es). An auxiliary signal can be for example a variable which is defined in the module state, and which can be changed in a certain process by reading an input. This change can have influence on the execution of the clock process.

At every clock event, all clock sensitive processes in all modules are executed exactly once. Think of the clocking of state variables such as flip-flops and registers, using the current state, module inputs and the current value of auxiliary internal signals. This may change internal auxiliary signals of the module as well as output signals of the module, but such changes are not effectuated until all clock processes have been executed.
Any process can be made "sensitive" for one or more input signals. After all clock processes have been executed, all module outputs are conveyed to their destinations (inputs of other modules), according to the netlist. If any inputs for which processes are sensitive turn out to have changed or rather: been operated, than these processes are activated in an order which can not be further influenced.

4.2.2 Simulation loop

One simulation cycle consists of 2 phases:

- In the first phase the clock processes of the scheduled clocks will be activated. Clock processes are those processes having a port in their sensitivity list which is connected to a clock channel.
- In the second phase all values assigned to output ports will be propagated to the connected input ports (of other modules) and the processes which have these input ports in their sensitivity list will be activated. This second phase is divided into zero or more delta cycles. Each delta cycle consists of the propagation the values of the output ports to the connected input ports and the activation of the processes with the input ports in their sensitivity list. As long as there are values assigned to output ports, a new delta cycles is started.

After the second phase, time is advanced to the next time tick at which a clock event will occur.
Simulating fully synchronous blocks doesn't use delta steps, so cycle based simulation will be done. This improves the speed of simulation.

There are two types of processes:

- Immediate and delayed processes. Immediate processes are executed once per delta cycle for every event on the input ports in their sensitivity list.
- Delayed processes are executed at most once per delta-cycle, if there is at least one event on any of the input ports of their sensitivity list.

4.3 Viewports of an instance

Viewports give you TSS ***runtime external*** access to ***possibly complicated*** items within your modules. The bold print shows two reasons why handling viewports is more complicated than handling ports, which 'only' provide inter module traffic of rather simple variables. Viewports are the *external* inputs and outputs of an instance. Note the word *instance*: view ports are specific to an instance, not a module, since different instances have the same viewports, but they lead to different items. They are not *shared.* Typical viewports objects are registers in an instance, which can be used for example to do debugging (PC in processor model) or gather statistics about instances for performance analysis. The viewport interface is not meant for communication between modules.

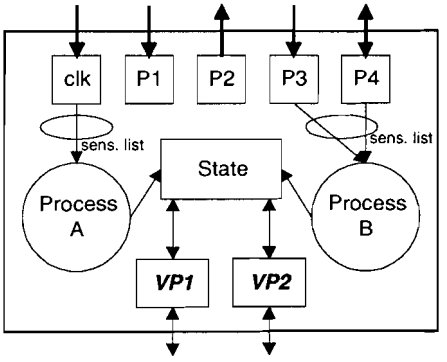

Figure 4.3-1: Viewports of an instance

Viewports are also used to send module commands to module instances. Module commands can be used for example to load a memory, view module registers, change registers or set break points. Figure 4.3-1 shows an instance with two processes A and B, 3 input ports (CLK, P1 and P3), 1 output port (P2) and 1 inout port (P4). The sensitivity list of process A contains only the CLK port, which most likely will be connected to a clock channel. The sensitivity list of process B contains two ports P3 and P4. Both processes share the same state and port structure. This means that both processes can assign values to the ports P2 and P4 and read the values from P1, P3 and P4. Reading a value from the CLK port doesn't make any sense. The above instance also shows two viewports, VP1 and VP2. These viewports provide a controlled view to the outside world of some objects in the instance.

4.4 Differences between a model in TSS and VHDL

A signal assignment changes the value of a signal. Signals are associated with time. When a value assigned a signal, the assignment will not take effect immediately, instead will be scheduled to a future simulation. Only assignments to ports in TSS is comparable with signal assignments like in VHDL. In other cases the variable will directly change by an assignment of an other value.

With C no parallelism is possible within a process. Sequences of statements in TSS are more important than in VHDL. For example when shifting values in a filter, the sequence in VHDL doesn't matter. Because at the same time all the values will change after all the individual assignments. In TSS the values will change after each individual assignment.

Because TSS doesn't know about signal assignments like in VHDL, a TSS model has for each clock frequency one clock process. This to avoid problems by changing the state and reading and writing to input and output ports. Because of this the simulation speed will increase in comparison with VHDL. And less overhead is needed for, because less processes are defined in TSS. This has influence for an TSS model on the amount of lines of code.

By making one clock process in TSS of a amount of clock processes in VHDL, it's obvious that the sequence of functionality of the individual clock processes in VHDL is important.
TSS models are not synthesisable, this in comparison with VHDL descriptions at RTL architecture level. The bit sizes of TSS data types are limited, these are 8,16, 32 and 64 bit.. TSS is a Philips proprietary tool, while VHDL is a IEEE standard.
In Table 4.4-1 the advantages and disadvantages are described of models in different descriptions.

Models in:	Advantages	Disadvantages
C	Very fast simulation Reals Can be bit true	No Concurrent processes No syntheses No Cycle true No signal assignments No bit size
TSS	Fast simulation Bit true and/or cycle true Viewports Reals	No syntheses Limited amount of bit size No signal assignments
VHDL-behavioural	Bit true and/or cycle true Faster simulation than RTL	No syntheses
VHDL-RTL	Syntheses Bit true and cycle true	Low simulation speed

Table 4.4-1: Comparison between models in different descriptions

4.5 Abstraction level of a common TSS model

Time aspect of all TSS models is the clock. And most available TSS models are bit true and cycle true as well.. Possible data types in TSS are: integers, floats, pointers to an array, and Boolean. Functionality can be described at a behavioural manner in processes.
RT behavioural level can be seen as the abstraction level of a common TSS model. (See chapter 1.5 for definition of the abstraction levels.) The level of such a TSS model will be slightly higher than a VHDL-behavioural description. Because TSS doesn't know about signal assignments (except for writing to ports) like in VHDL and has 1 clock process for each clock frequency.

4.6 Projects and application areas

Several groups within Philips Semiconductors develop IP's, which can be used as part of a product. Some of these groups deliver TSS models as part of their IP deliverable. Most of these groups have put out the TSS model development by behavioural modeling in Bangalore. TTI (Trimedia Technologies Inc) delivers TSS trimedia models in heir development kit.
TSS is used in the following different application areas

1. architectural exploration
   Which is mainly done within research. In different projects TSS is used for study the feasibility of certain architectures.
   Examples of these projects are:

- Storage home infotainment systems
- 3rd generation mobiles

Projects within Semiconductors are for example:

- Next generation TriMedia
- Universal Display Processor (UDP)

2. system design.
   With system design is meant the creation and verification of IP on system level. At this stage models will be made in C/C++. Afterwards these models will be used for HW/SW partitioning, high-level design trade-offs, design exploration etc. Especially PS-Hamburg is using TSS for system design. System lab in Eindhoven is using TSS for system design as well.
3. Simulation engine for performance estimations.
   Tools like SPADE, C-Heap and YAPI uses TSS as simulation engine.
   In application areas:
   - Digital still Camera
   - MPEG encoding / decoding
   - Audio encoding / decoding
   - Imaging
   - 100 Hz TV
   - Speech technology

4.7 C-based sign-off

TSS is made such, that the major part of the system is modelled in C, including the CPU (ISM), the bus and the memories. Because of the wide possibilities of TSS in the C domain, a C-based sign-off design process has been established within Philips [SLH and TriMedia see 3] . In this approach the complete system is described in TSS. During the conversion (mostly by hand) from C behaviour to synthesisable HDL, only a small part of the C description of the system is replaced with synthesisable HDL, and its functionality is checked though the co-simulation interface. The largest part of the system stays in the C domain (ISM or TSS models). In this way the complete system is refined to synthesisable HDL, module by module. In the end verification entire system in VHDL has still to be done on e.g. CoBalt.
All this is in contrast with a design flow where gradually all the modules are transferred from the C domain (if available) into the HDL domain. (See figure 4.7-1) As the design process advances the simulation get slower and slower because the HDL part gets larger and larger.

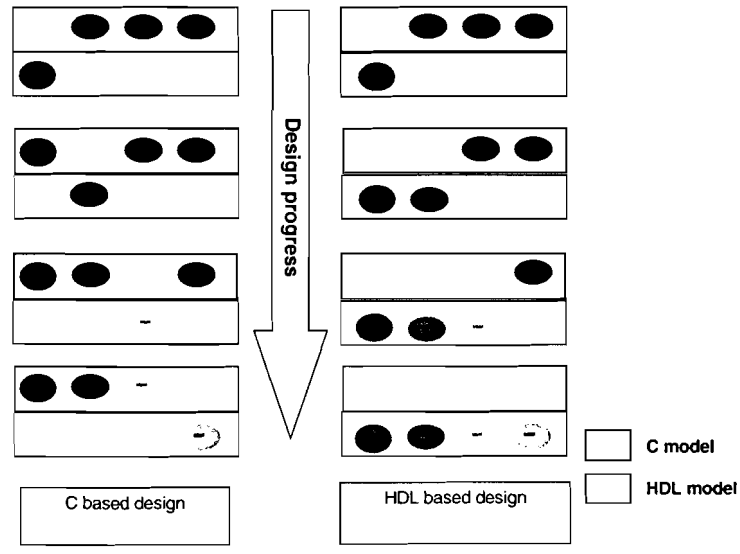

Figure 4.7-1: C-based sign-off

5 Digital Output Processor

The first part of this chapter will give the scope where in the Digital Output Processor (DOP) can be placed. So the Salsa system will be introduced. In the second part of this chapter explains the functions of the DOP which are modeled in TSS.

5.1 Scope

The ADOC IC is part of the Salsa system, a highly integrated solution for TV video, and sound processing and TV control, aimed to the mid range market segment. A global view of the Salsa system is depicted in Figure 5.1-1 There by a division is made between a few functions witch will be implemented analogue in the MPIF IC and the rest of the functionality which will be implemented digital in the ADOC-IC.

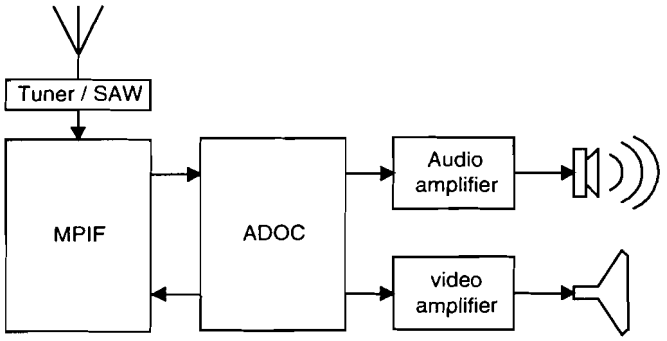

Figure 5.1-1: SALSA System

The ADOC-chip is divided in three parts (see Figure 5.1-2). The VideoCore, ControlCore and SoundCore.

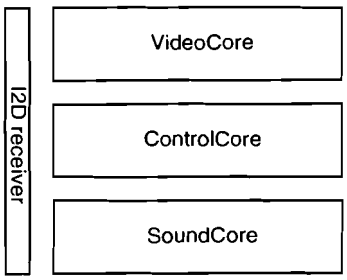

Figure 5.1-2: ADOC

VideoCore contains the functions for video processing (see Figure 5.1-3). It includes:

- 2 Video decoders (Viddec), decoding CVBS signal into YUV format.
- Feature box consists of (with some features):
  - Front End Features (FEF), noise measurement, black bar detection, histogram processing.
  - Memory Based Features (MBF), Picture in Picture (PIP), Double Window.
  - Back End Features (BEF), Panorama, frame generation and colour space correction
- Digital Output Processor (DOP): Beam Current limiter, Peak White limeter, Graphics / Video blending, Scan Velocity Modulation, deflection processing, Low-Power start-up, Geometry correction, Cathode calibration

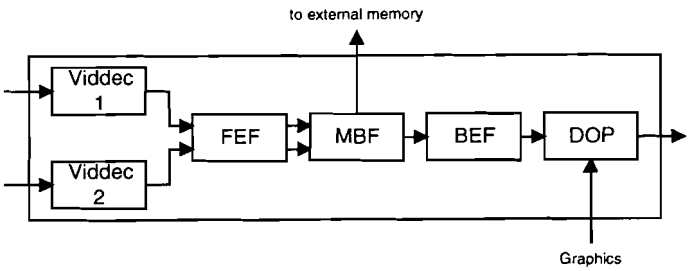

Figure 5.1-3: VideoCore

The ControlCore contains the functions for controlling the TV set. The SoundCore contains the functions for decoding signals for the sound standards NICAM, AS, BTSC Stereo and Japan stereo, further more it contains audio processing functions for supporting features like incredible stereo expander, Dolby Prologic, bass management, loudness and 5 band Equaliser.

5.2 Functions of DOP

This chapter explains functions which will be modeled in a TSS model of the DOP. The functions are Beam Current Limiter (BCL), Peak White Limiter (PWL), Graphics Video blending and Scan Velocity Modulation (SVM).
An abstract scheme is shown in Figure 5.2-1 of the signal processing part of the DOP. A TSS model is made of the grey coloured parts.

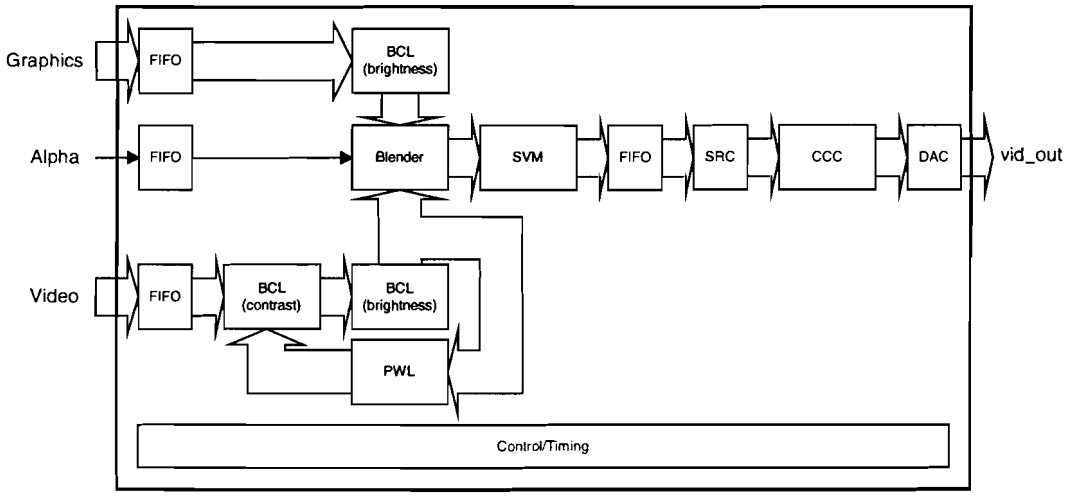

Figure 5.2-1: Global view of signal processing part of DOP

The FIFO's at the input of the DOP are used for synchronisation between the Video and Graphics stream.
The FIFO and the SRC behind the SVM are used for the establishment of the link between the orthogonal pixel domain on a crystal clock running at 54 MHz, and the display clock domain, which runs at 81 MHz, also crystal clock based however now the pixel position is closely related to time. Modelled part is in the 54 Mhz clock domain.
Continuous Cathode Calibration (CCC) determines correct biasing levels for the picture tube cathodes such as the black level.

5.2.1 Beam Current Limiter

Beam Current Limiter (BCL) monitors the beam current as flowing through the Line Output Transformer (LOT) and protect LOT for saturation by reducing contrast and brightness when to high beam current is measured.
The LOT is a special transformer which in conjunction with the horizontal output Line Transistor/deflection circuits boosts the voltage of the low voltage power supply to the 20 to 30 KV for the CRT. When LOT gets saturated, primary current will grow too large causing damage to Line Transistor. For this reason Beam Current Limiter control may adjust contrast and brightness during field scan hence causing a different contrast/brightness in the first few video lines of a field than during the last lines of a field.
Figure 5.2.1-1 displays the block diagram of the Beam Current Limiter. The idea is that the video signal path will be affected firstly by reducing contrast and if that does not help also brightness will be reduced in case of too large beam currents. As in the

graphics signal path no contrast control is intended, Beam Current Limiting can only affect brightness here.
Beam Current Info is combined with relevant threshold values and gains in order to make a separate controlling signal for contrast and another one for controlling brightness. As most of the functionality used is identical here, one sub-block will be defined which will be instantiated twice. This sub-block whose block diagram is depicted in Figure 5.2.1-2 introduces a threshold above which bcl info will certainly not affect brightness or contrast and a gain value which determines the proportional influence of bcl info on brightness or contrast. Furthermore a 'shadow' function shall intervene whenever threshold and or gain values set possibly might damage the TV-set. The reader might be confused by the polarity in the block diagram Figure 5.2.1-2, but bcl voltage decreases when beam current increases.

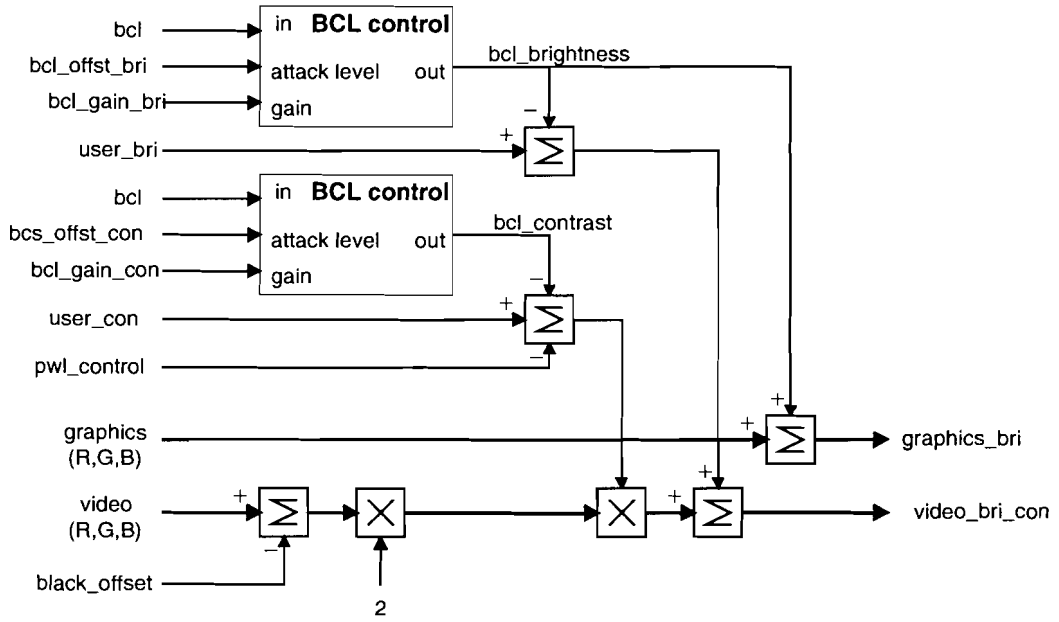

Figure 5.2.1-1:Beam Current Limiter block diagram

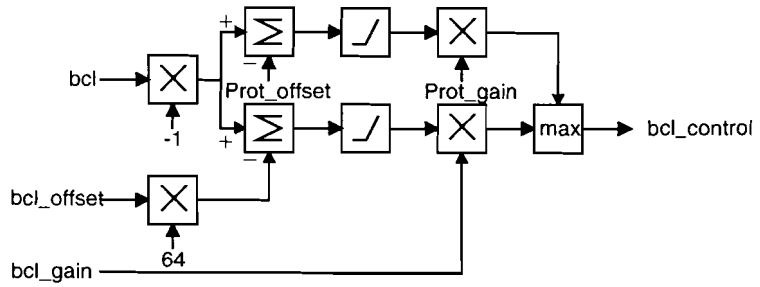

Figure 5.2.1-2: 'BCL control' for converting bcl (information) to control signal

For DOP input signals are supposed to be superposed on a black level of 64_{dec}. In ADOC, the block preceding DOP, "Back End Features" (BEF), which will not satisfy this DOP requirement. For that reason a programmable black level subtractor is to implemented. Input signals will be in a range 288_{dec} (black) to 726_{dec} (peak white). Implementing an 9 bit subtraction ($0..511_{dec}$) should be satisfactory for realising a

black level of 32_{dec} (peak white will be 470_{dec}). So black offset has to be 256_{dec} .Then the signal will be multiplied by two in order to create a range of 64_{dec} (black) to 940_{dec} (peak white) leaving some headroom (10% should be enough) for peaking.

5.2.2 Peak White Limiter

Peak White Limiter (PWL) detects relative small areas in a scene to be displayed with very high luminance and reduce contrast in order to prevent deterioration in resolution due to spot blooming. Figure 5.2.2-1 displays the block diagram of the PWL.

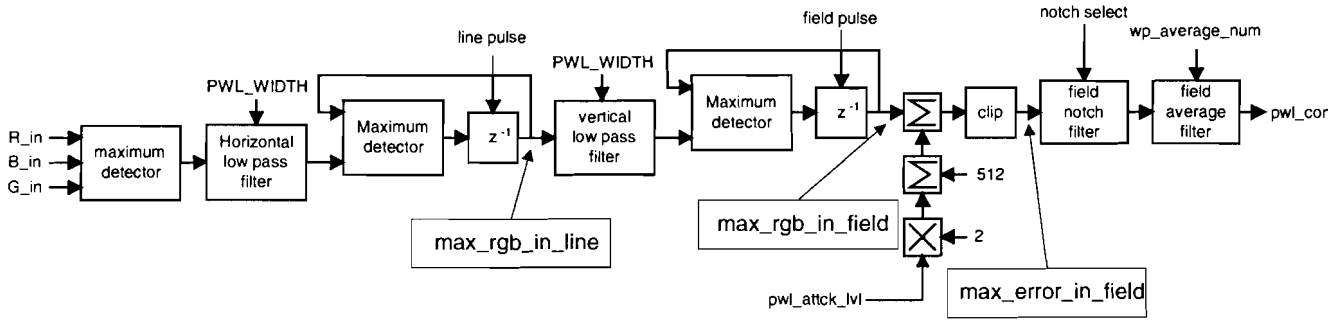

Figure 5.2.2-1: Peak White Limit algorithm in DOP

The maximum value of R, G and B will be fed to a low pass filter in order to filter for spikes and noisy variations. With pwl_width the time constant can be selected. In practice this parameter determines the horizontal size of the area that just should trigger the Peak White Limit system.
The maximum value found within each video line will be determined and at the end of each line transferred to the vertical low pass filter. With pwl_width in this filter the vertical size of the area to be detected by the Peak White Limit system can be adjusted.
The maximum error found within each field will be determined and will then be applied to a subtractor that determines the size of the error with respect to the desired Peak White level with the value of pwl_attck_lvl.
After clipping the result of this subtraction than will be filtered again on field basis in a FIR filter with a selectable notch. The SECAM colour standard can under certain circumstances yield 12.5Hz flickering in peak luminance. In case the inter field averaging (to be discussed later on) are set to say 2 fields averaging, then this 12.5Hz flicker will be transferred to contrast control, thus giving a very annoying 'pumping' picture. To overcome this effect, a lowpass filter will be introduced which has notches at multiples of the field frequency. In that way signals varying with a multiple of the field frequency will not be transferred to the contrast control stage.
The output of this field notch filter than will be applied to a field based IIR filter. The number of fields the error shall be averaged over can be set by the value of wp_average_num. The output of this filter then will be available as signal pwl_control.

5.2.3 Graphics Video blending

This blends video and graphics in 16 steps (internal 64 steps possible). When alpha is is 0000 then 100% video and if alpha is 1111 then 100% graphics.

5.2.4 Scan Velocity Modulation and Dynamic Contrast Control

In a conventional Television set, the light output as perceives by the viewer is mainly determined by the beam current . When studying Scan Velocity Modulation (SVM), one should be aware that this light output can be modified by varying the scan speed. With a given constant beam current, the amount of electrons which hit the phosphor screen per unit area depends on the scan speed, hence light output depends on the scan speed. So, the lower the scan speed the more light output from the picture tube, still with a constant beam current!
The opportunity SVM offers is to introduce the same affect as peaking without increasing the beam current. This is quite an attractive opportunity as the spatial bandwidth decreases rapidly with increasing beam current. This is an important advantage of SVM: conventional peaking increases the beam current, hence reduces the spatial bandwidth.
Figure 5.2.4-1 shows the effect of SVM. The upper traces show a luminance input signal. The traces below al show the first derivative of the input signal (solid) and the second derivative of the input signal (dashed). The lower trace then displays the light output as perceived with SVM. In fact the luminance as perceived has been calculated

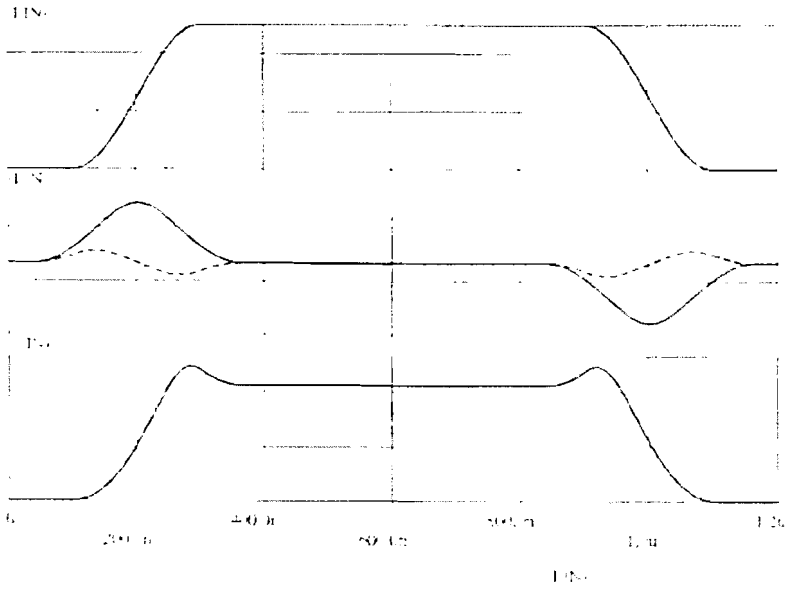

Figure 5.2.4-1: Luminance signals

by dividing the input luminance signal by the scan speed. The scan speed has been calculated by $V_{scan} = V_{nom} (1+d^2Y/dt^2)$ (in which Y represent the luminance input signal as depicted in the upper trace in Figure 5.2.4-1.

Functional diagram is depicted in Figure 5.2.4-2, which shows the basic idea of the set-up.

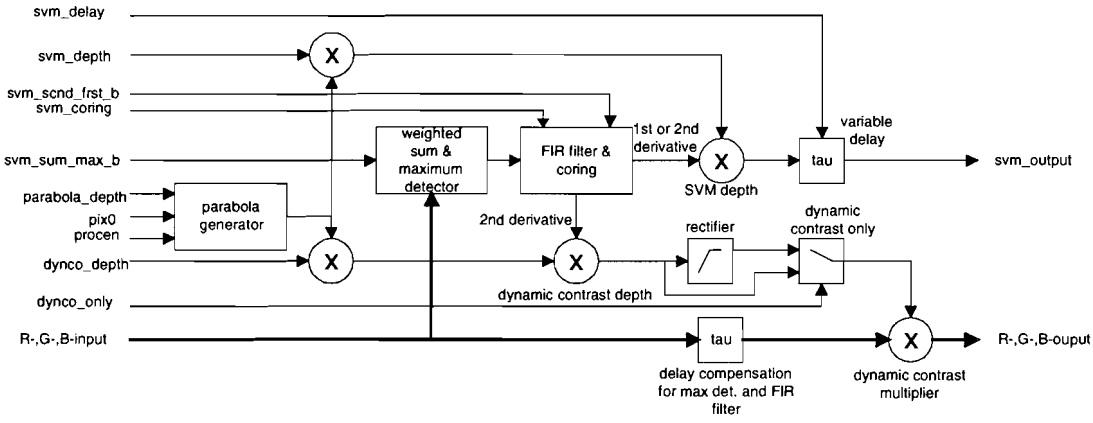

Figure 5.2.4-2: Functional block diagram of SVM and dynamic contrast

The amount of gain on dynamic contrast and SVM can be made place dependent by setting parabola_depth a value different from zero. Further the first derivative of the maximum or weighted sum of R, G and B will be attenuated and delayed such that the voltage to current amplifier which drives the SVM-coil can be connected to the D to A converter directly. In case the SVM coil is driven by a voltage to voltage driver, the SVM system must output the second derivative of the maximum or weighted sum of R, G and B. SVM-Coil controls the speed of writing of the beam, whereby the speed is proportional with voltage over the coil. So the second derivative of svm_ouput is proportional with the speed of writing.
In case no SVM coil is to be connected, then still dynamic contrast can be used in order to avoid spot blooming at high beam current levels. In this case however, the dynamic contrast should reduce contrast only. For this reason a rectifier has been foreseen.
A coring stage to be implemented at the output of the FIR filter that generates the first derivative of the maximum or weighted sum of R, G, and B will reduce influence of noise on the SVM performance.

6 Modeling Digital Output Processor

This chapter describes the modeling approach followed for modeling a TSS the DOP. Thereby a comparison will be made between the VHDL model and the TSS model of the DOP.

6.1 Introduction modeling systems

Complex systems can be generally categorised into three classes: transformational, interactive, and reactive systems. Transformational systems operate on inputs available at the beginning and stop after delivering outputs at the end. Most traditional computing programs fall into this category. In contrast, interactive and reactive systems typically do not terminate (unless they fail). These systems maintain an ongoing series of interactions with their environment. Interactive systems constantly operate on inputs when the systems are ready, and deliver outputs when the systems are willing to. Examples of interactive systems include operating systems and multimedia network applications. Unlike interactive systems that interact at their own speed, reactive systems follow a pace dictated by the environment. The environment determines when the systems must react and provides inputs. The systems respond to the inputs by possibly sending outputs to the environment. The ADOC is an example of a reactive system. The received TV-signal has to be processed fast enough without loosing frames.
Because systems become more and more complex, higher abstraction levels are needed to control the complexity. At these levels it is possible to keep an better overview over the complexity. As well design decisions made at these higher levels have a big impact on the implementation and design space is bigger as well.
General goals for modeling are:

- Modeling for specifying a system, which can be used by adaptations for new designs.
- Higher simulation speed with higher level models, which decrease the turn around time by designing a system at an early design stage.
- better possibilities for verification.

6.2 Modeling DOP

In the project description in chapter 3 is written that there will be made a TSS model of the DOP. Because each TSS model has a clock process, so each TSS model can be made cycle true and if necessary bit-true. The level of a common TSS model will be RTL-behavioural (see chapter 4.5).
The partitioning of the sub-blocks modeled in TSS is about the same as the in the final implementation in VHDL at RT architecture level.

Of the parts which are modeled is made a deterministic description, by using a Event Processing (EP) / Signal Processing (SP) scheme which is shown in Figure 6.2-1.

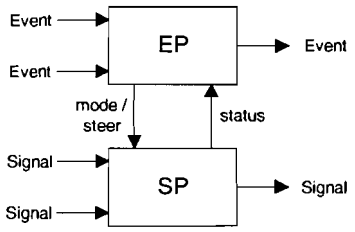

Figure 6.2-1:EP/SP scheme

Definition event:
An irregular and unpredictable (time, value) pair which can be represented as bit vectors.

Definition digital signal:
Signal which is created by sampling an analogue signal followed by quantisation and binary encoding what results in a periodic stream values in the form of bit vectors.

In Figure 6.2-1 is made a difference between signal processing and event processing.

- In event processing there will be time aspects, because inputs are time dependent. It depends on events from the outside world at the input of EP or on the status of SP, if synchronous or asynchronous mode/steer information will be generated for SP or events will be generated to the output. EV processing will be throughput insensitive, while the latency is very important. Events are, of course, not predictable.

- Digital signals are derived from analogous signals and represent a measurable quantity. The derivation consists of sampling in time, following by quatisation and a binary encoding. It follows that a digital signal represents a periodic stream of values. SP will generate a new signal or it will deliver status information from EP. Block SP consists of a functional behaviour of the signal processing part of the system. Signal processing will be latency insensitive and throughput is very important.

The parts which are modeled in TSS have only SP functionality, see Figure 5.2-1 and Figure 6.2-2. The event processing part is not modeled.
A behavioural description is made of the BCL, PWL, Blender and the SVM. The best way is to define a procedure for each partition, which is located in a separate file. These procedures will than be called in one (TSS) clock process and will share the state information of the TSS model.

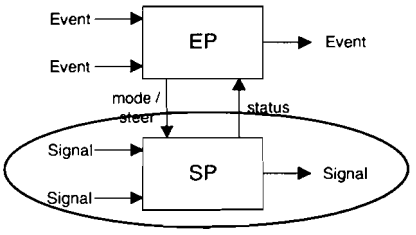

Figure 6.2-2:EP/SP scheme

Writing behavioural descriptions from scheme's like in chapter 5, of each sub-block is quite straight forward.
Each behavioural description of a certain partition is in fact a sequential C-program. There can be made a separation between constants, digital signals and mode/steer. Values of mode/steer can be stored in the state of the description and data signals can be read from the ports or from auxiliary variables of the state. Constants can be defined in the procedure of a certain partition.
When no filters or feedback's are used, than such a description can be delay-free by a non-cycle true description.
Each procedure of a certain partition is called in one clock process.

TSS has limited bit sizes for data types (8, 16, 32, 64 bits), so bit true behaviour can be imitate by using clippers. In the model are functions made of operations frequently used, this saves lines of code.
When the implementation in VHDL RT architecture level is ready than the TSS model can be made cycle true by adding delays in the behavioural description.
Each TSS model contains a session function and a instance function.
The session function of a module describes everything that needs to be done to make the module available to the simulation kernel. This includes definitions the parameters and/or user procedures which may be useful to access the viewports of instances of a module.
The instance function contains all the functionality that is related to an instance of a module. Typically an instance function creates the data structures, the ports, the processes and defines the viewports for each instance.

6.2.1 Comparison TSS-model and VHDL-RTL description of DOP

The resulted TSS model of the DOP is more abstract than the VHDL equivalent.
Abstractions of the TSS model of the DOP in comparison with the VHDL equivalent are:

- Use of one TSS clock process.
- No signal assignments in TSS with the same behaviour as in VHDL, except from assignments to output ports in TSS.
- Behavioural descriptions of multipliers, this in contrast with the implementation whereby shifted values are added.
- Behavioural descriptions for determination of minimum and maximum values

parts	TSS	VHDL	(factor)
Testbench	475	400	0,85
BCL, PWL	515	1880	3,6
Blender	15	560	40
SVM	240	860	3,6

Figure 6.2.1-1: Difference in lines of code between TSS and VHDL

Due abstractions of the TSS model of the DOP the number of lines code will be less than the equivalent description in VHDL.
Table 6.2.1-1 shows the difference in lines of code between TSS model and the description in VHDL. It's shows that when more functionality will be added the difference will become bigger. For the testbench in TSS the lines of code needed for the Session function and the Instance function are counted as well.
In VHDL the lines of code of the entity-, configuration- and architecture files are counted for each part.
SVM in VHDL is mainly written at a RT behavioural level. But one part of SVM is 'RTL-ished behavioural'.

7 Experiments

This chapter will explain the experiments which are done with the TSS model of the DOP. There is a comparison made between simulation time of TSS and VHDL by simulating equivalent sub-blocks of the DOP.
At Nat.Lab is made a TSS frame-work of the ADOC. Which include a part of the VideoCore. An experiment will be done whereby the DOP will be included in this frame work. In the first time the TSS model of the DOP will be included, the second time the VHDL equivalent of the DOP will be included.
Tools are used by these experiments are TSS 3.3, NcSim and Leapfrog (Cadence_ldv 3.1.lsr).
The command 'time' within Unix is used for measuring the simulation speed between VHDL and TSS simulations. There by is '**user time**' used for comparison.
Simulations are done on compute server cicc27 which is a HP 9000/785 compute server.

7.1 DOP stand-alone

This chapter shows the result of experiments done with some parts of the DOP for making a comparison between simulation speed of TSS models and VHDL models. Three simulation set-ups are made. Each next simulation set-up contains more functionality. Several simulations will be done with the use of a certain simulation set-up. The use of disk I/O will be the difference between these simulations. When input disk I/O is used than an image of 720*576 pixels in RGB format (2 fields) will be loaded. Otherwise such a image will be generated.
Results shown in tables in this chapter are the averaged values of 5 simulations. All the measured values are within 2% of the averaged value.

7.1.1 Simulation DOP stand-alone

Simulation set-up 1:

Figure 7.1.1-1 shows the sub-blocks which are used in simulation set-up 1. The blocks used by the simulations are BCL (brightness and contrast) and PWL.

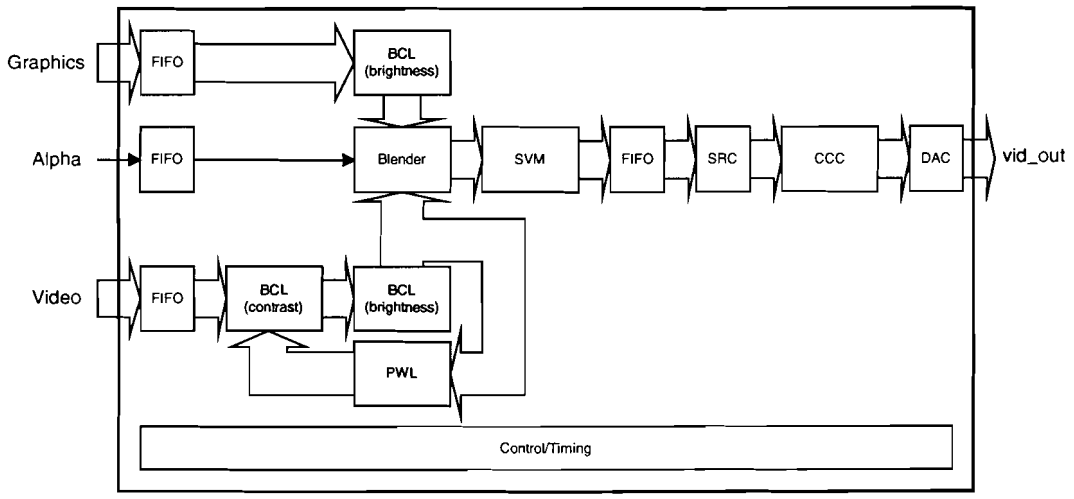

Figure 7.1.1-1: Global view of signal processing part of DOP

Compilation and elaboration times are shown in Table 7.1.1-1 in TSS, NcSim and Leapfrog.

Disk I/O	Compilation (sec)	Elaboration (sec)
TSS	1.9	0.48
NcSim (VHDL)	0.8	0.29
Leapfrog (VHDL)	1.4	0.29

Table 7.1.1-1: Compilation and elaboration times of simulation set-up 1.

Lines of code for TSS model are 990 lines and for VHDL-RTL 2280 lines.
VHDL-RTL has 49 clock processes and 16 asynchronous processes.

Disk I/O	TSS (sec)	VHDL NcSim (sec)	VHDL Leapfrog (sec)	TSS (cycl./sec)	VHDL NcSim (cycl./sec)	VHDL Leapfrog (cycl./sec)
Input: yes, Output: yes	4,7	62,7	223,1	88239	6614	1859
Input: yes, Output: no	3.8	58,8	209,8	109137	7053	1977
Input: no, Output: yes	2.8	57,9	201,0	148115	7163	2063
Input: no, Output: no	1.9	52,8	194,9	218274	7855	2128

Table 7.1.1-2: Simulation results of simulation set-up 1.

Disk I/O	TSS (sec)	VHDL NcSim (sec)	VHDL Leapfrog (sec)	TSS (cyc./sec)	VHDL NcSim (cycl./sec)	VHI Leapf (cycl./
Input: yes, Output: yes	1	13,3	47,5	47,5	3,6	1
Input: yes, Output: no	1	15,5	55,2	55,2	3,5	1
Input: no, Output: yes	1	20,7	71,8	71,8	3,5	1
Input: no, Output: no	1	27,8	101,6	101,6	3,7	1

Table 7.1.1-3: Normalised values of simulation results of simulation set-up 1.

Simulation set-up 2:

Figure 7.1.1-2 shows the sub-blocks which are used in simulation set-up 2. These blocks are BCL (brightness and contrast), PWL and Blender. The speed difference between the tools NcSim and Leapfrog is quite constant for the Simulation set-up 1 by the different disk I/O simulations. NcSim is 3,6 (averaged value) times faster than Leapfrog. For this reason no simulations with Leapfrog are done more in next experiments.

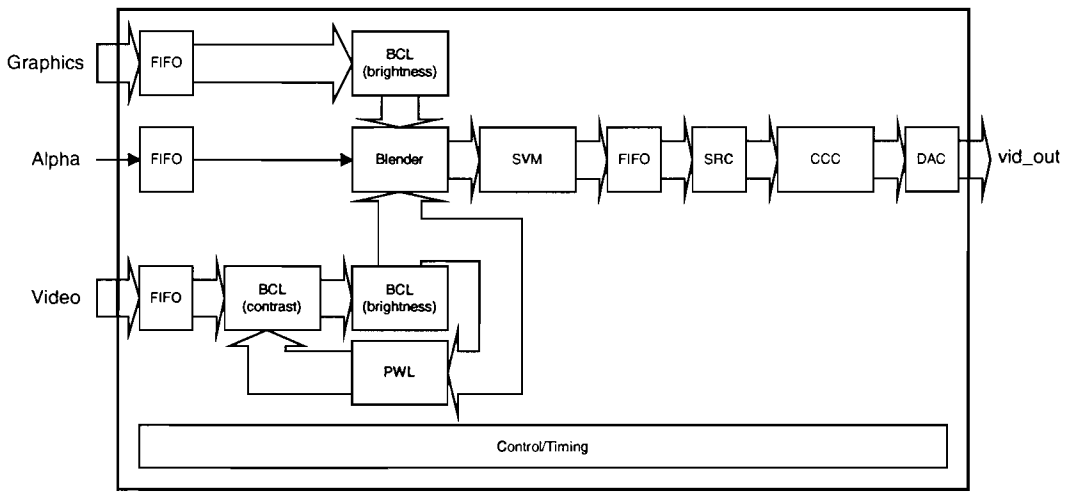

Figure 7.1.1-2: Global view of signal processing part of DOP

Compilation and elaboration times are shown in Table 7.1.1-4 in TSS and NcSim NcSim.

Disk I/O	Compilation (sec)	Elaboration (sec)
TSS	1.9	0.48
NcSim (VHDL)	1.0	0.5

Table 7.1.1-4: Compilation and elaboration times of simulation set-up 2.

Lines of code for TSS model are 1005 lines and for VHDL-RTL 2840 lines.
VHDL-RTL has 54 clock processes and 17 asynchronous processes

Disk I/O	TSS (sec)	VHDL NcSim (sec)	TSS (cycl./sec)	VHDL NcSim (cycl./sec)
Input: yes, Output: yes	4,8	80,4	86400	5158
Input: yes, Output: no	3,8	78,0	109137	5317
Input: no, Output: yes	2,9	75,8	143007	5471
Input: no, Output: no	2.0	71,1	207361	5833

Table 7.1.1-5: Simulation results of simulation set-up 2.

Disk I/O	TSS (sec)	VHDL NcSim (sec)	TSS (cycl./sec)	VHDL NcSim (cycl./sec)
Input: yes, Output: yes	1	16,8	16,8	1
Input: yes, Output: no	1	20,5	20,5	1
Input: no, Output: yes	1	26,1	26,1	1
Input: no, Output: no	1	35,6	35,6	1

Table 7.1.1-6: Normalized values of simulation results of simulation set-up 2.

Simulation set-up 3:

Figure 7.1.1-3 shows the sub-blocks which are used in simulation set-up 3. These blocks are BCL (brightness and contrast), PWL, Blender and SVM.

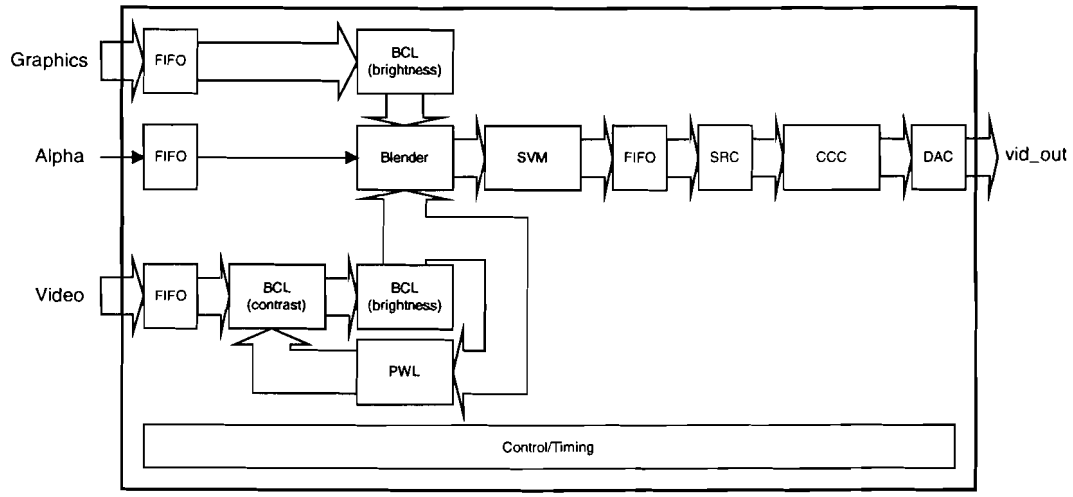

Figure 7.1.1-3: Global view of signal processing part of DOP

Compilation and elaboration times are shown in Table 7.1.1-7 for TSS and NcSim.

Disk I/O	Compilation (sec)	Elaboration (sec)
TSS	1.95	0.48
NcSim (VHDL)	1.41	0.8

Table 7.1.1-7: Compilation and elaboration times of simulation set-up 3.

Lines of code for TSS model are 1245 lines and for VHDL-RTL 3700 lines.
VHDL-RTL has 80 clock processes and 17 asynchronous processes

Disk I/O	TSS (sec)	VHDL NcSim (sec)	TSS (cycl./sec)	VHDL NcSim (cycl./sec)
Input: yes, Output: yes	5,1	133,9	81318	3097
Input: yes, Output: no	4,3	128,1	96447	3238
Input: no, Output: yes	3,1	124,6	133781	3328
Input: no, Output: no	2,3	118,2	180314	3509

Table 7.1.1-8: Simulation results of simulation set-up 3.

Disk I/O	TSS (sec)	VHDL NcSim (sec)	TSS (cycl./sec)	VHDL NcSim (cycl./sec)
Input: yes, Output: yes	1	26,3	26,3	1
Input: yes, Output: no	1	29,8	29,8	1
Input: no, Output: yes	1	40,2	40,2	1
Input: no, Output: no	1	51,4	51,4	1

Table 7.1.1-9: Normalised values of simulation results of simulation set-up 3.

7.1.2 Conclusion DOP stand-alone

Simulation set-ups 1 till 3 show that the simulation speed difference between simulations in TSS and VHDL is the biggest when no disk I/O is used in both cases. Therefor file I/O not cause of difference. The relation between the 3 simulation set-ups and the difference in simulation time between TSS and VHDL is shown in Figure 7.1.2-1. The 4 lines denotes the simulations with or without disk I/O.
Figure 7.1.2-1 and Figure 7.1.2-2 show that the difference in simulation time between TSS and VHDL is much higher and faster increasing than the difference in lines of code. The difference in lines of codes is increased by 7 % by experiment 3 in comparing with simulation 2, while the difference in simulation time is increased with 44 %.
Table 1 shows that the amount of *(clock)* processes in VHDL have a strong correlation with the difference in simulation time. (It is obvious that the difference of the amount clock processes between the TSS and VHDL is the same as the amount of processes of VHDL, because in TSS is one clock process defined.). So the difference in lines of code doesn't have so much influence on the difference in simulation time in comparing with the difference of clock processes. Graphical this can be seen in figures 7.1.2-1, 7.1.2-2 and 7.1.2-3.
Figure 7.1.2-4 shows the relation between amount of clock processes and difference in speed between TSS and VHDL.

Observation:

> **The quotient of the *clock processes* will be proportional with the quotient of the simulation times of two models with the *same functionality* but with a difference amount of *clock processes*.**

In these experiments disk I/O has big influence on the simulation time of TSS. When more functionality is added in the TSS model, than this impact will become less. By using no disk I/O in simulation set-up 1 the simulation time will be 2.5 times faster in comparison when disk I/O is used. For simulation set-up 2 the difference will be 2.4 and for simulation set-up 3 the difference will be 2.2. So the impact of disk I/O will become less when more functionality is added.

Exp.	Processes TSS	Processes in VHDL	TSS Lines code	VHDL lines code	Diff. lines code	Sim. time TSS	Sim. time NcSim	Diff. sim. time
1	1	**48** clk + 16 async.	990	2280	**2,3**	1,9 sec	52,8 sec	**27,8**
2	1	**54** clk + 17 async.	1005 (+1.5%)	2840 (+25%)	**2,8** (+22%)	2.0 sec (+5%)	71,1 sec (35%)	**35,6** (+28%)
3	1	**80** clk + 17 async.	1245 (+24%)	3700 (+30%)	**3,0** (+7%)	2,3 sec (+12%)	118,2 sec (+66%)	**51,4** (+44%)

Table 7.1.2-1: Results simulation when no disk I/O is used.

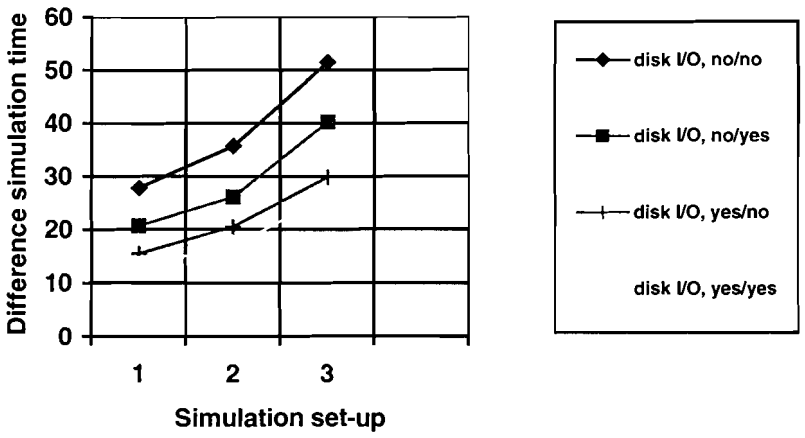

Figure 7.1.2-1: Relation between simulation set-up and difference simulation time between TSS and VHDL model

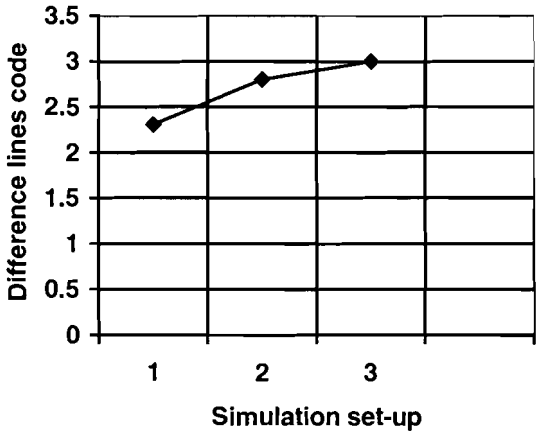

Figure 7.1.2-2: Relation between simulation set-up and difference in lines code between TSS and VHDL model

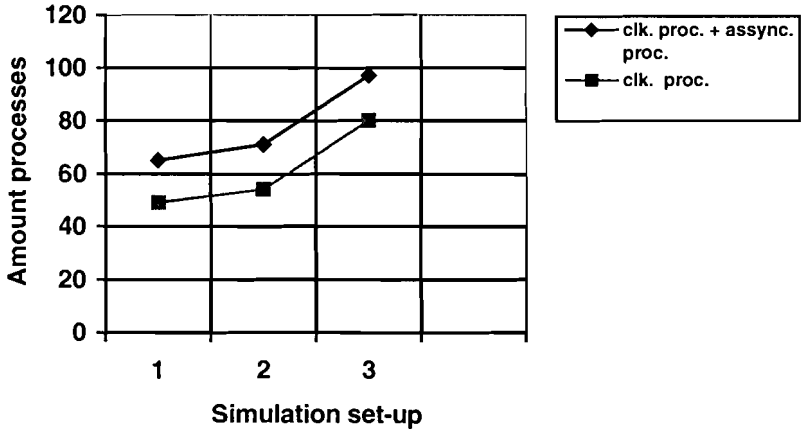

Figure7.1.2-3: Relation simulation set-up and amount processes in VHDL

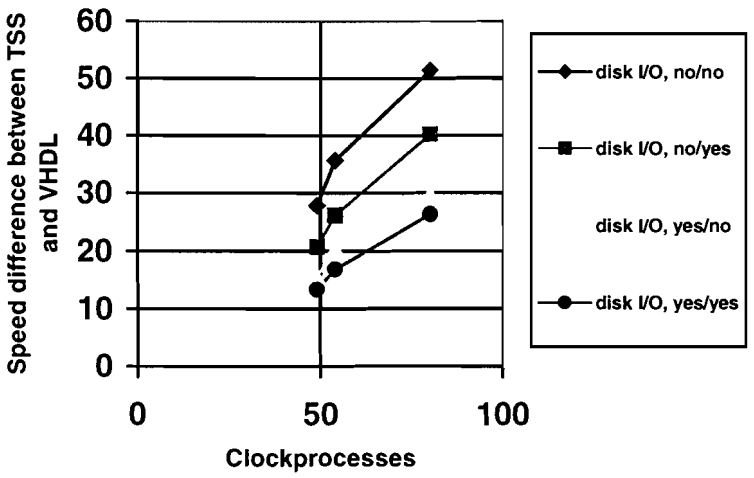

Figure 7.1.2-1: Relation between clock processes and difference in speed between TSS and VHDL

7.2 DOP in ADOC frame work

In the TSS ADOC-frame work is modeled a part of the VideoCode and a part of the ControlCore which is shown in Figure 7.2-1.
Of the VideoCore is modeled the feature box, which consists of FEF, MBF and BEF. Each of these parts is modeled in a sequential C-program (no TSS model). Process interfaces (Process IF) are used for connection these programs with the TSS simulation process. This block translates reads/writes from the UNIX process side into PI-bus requests at the TSS side and is extended with the DTL protocol for point-point communication. The C-programs have no notion of time and runs infinitely fast (for TSS), and only the communication is performed cycle-accurate on the TSS side. However, it is possible to annotate delay statements in order to estimate the latency of the modeling hardware. An additional advantage of this modeling technique is that the effect of moving functionality into software into software can be evaluated very fast. The same functional code is simply compiled for the target processor.
To demonstrate the interaction between the VideoCore and the MIPS, a small interrupt service routine has been implemented, which reads the average field luminance values from the FEF. Moreover, the MIPS can send start signals to the feature box, as well as writing control parameters into registers inside the process interfaces. The simulation produces two files, one is the YUV video file (written by the BEF), and the other is a dump file with the RGB values produced by the DOP, which can than be send to generate an image file of the individual frames.
For a extensive explanation of the TSS ADOC-frame work see [6].

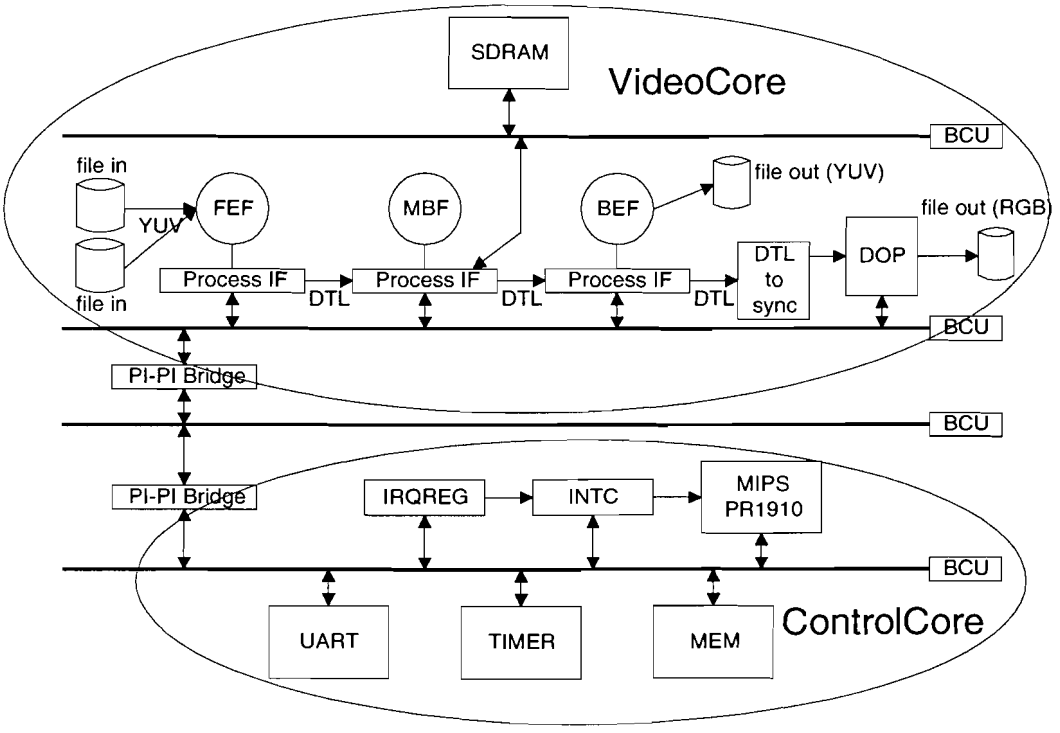

Figure 7.2-1: TSS ADOC-frame work

7.2.1 (co-) Simulations DOP in TSS ADOC-framework

Table 7.2.1-1 shows the results of the simulation with the DOP in the TSS ADOC-framework.
This framework is simulated in different situations. In the first situation is the TSS model of the DOP used and in the second situation is the VHDL-RTL model used of the DOP.
In each situation there are simulations done by writing an output file by the DOP and without writing an output file by the DOP.
The TSS ADOC-frame work consists of 18 clock processes and 100 asynchrony processes. The TSS model of the DOP consists of 1 clock process, while the VDHL model of the DOP consists 98 clock processes and 117 asynchrony processes.

DISK I/O (DOP)	Amount processes ADOC Framework (TSS) + DOP(TSS)	Amount processes ADOC Framework (TSS) + DOP(VHDL)	Sim. time ADOC Framework (TSS) + DOP(TSS)	Sim. Time ADOC Framework (TSS) + DOP(VHDL)	Diff. sim. Time
Output: yes	19 clk.+ 100 async.	98 clk.+ 112 async.	7,8 min	217 min*	**27,8**
Output: no	19 clk.+ 100 async.	98 clk.+ 112 async.	7,4 min	213 min	**28,8**

Table 7.2.1-1: Results simulation

*Extrapolation from results of the simulation of a single field to one frame.

7.2.2 Conclusions DOP in TSS ADOC-framework

By using the simulation time of ADOC and DOP simulated in TSS as reference, one would expect that this simulation time would be around 5 (98 clk. Proc./ 19 clk. Proc.) times faster when the DOP will be simulated in NcSim and the ADOC framework in TSS.
But such estimation can only be done when both tools have comparable simulation performance or if all models are simulated in one tool.
Depended of writing an output file or not writing an output file by the DOP, the difference in simulation time is 27,8 and 28,8 times respectively.
Therefor can be concluded that NcSim increase simulation time, due lower simulation performance.
TSS channels allow a link to be made between a signal in the HDL simulator to a channel in the TSS simulator. Any activity in the TSS side will also occur in the HDL simulator, which slowing down simulation speed. Any non-PI-bus signals that are required to connect an HDL block to TSS, have to be made using this mechanism. So data for the DOP is directly coming from the DTL-to-sync conversion. Therefor tsschannels has to be used for making the connection between data signals of the TSS environment and the VHDL environment which is simulated in NcSim.
Simulation performance will be better when a PI-VHDL bridge is used [20]. The PI-VHDL bridge is used in TSS-Leapfrog or TSS-NcSim co-simulation to connect a PI -

Bus component in VHDL to components in TSS. The advantage of using this component rather than TSS channels is that it can be configured so that there is only activity in the HDL simulator when the PI-BUS components are being accessed as opposed to when there is activity on the bus. This improves overall simulation performance. but this can only used when all the communication is done with the PI-bus.
C-based sign-of design approach is possible. And simulation at VideoCore level is possible as well, this in contrast with the implementation at RT architecture level which was during the time of simulations in TSS not finished.
The simulation with the ADOC frame work shows that multi level simulations are possible. The simulation with the TSS ADOC frame work shows that it is possible to do simulations in a common TSS environment with TSS models, C-programs at algorithm level and VHDL models at RT architecture level.

8 Arguments and answers on research questions

In the first part of this chapter the difference between bus-based architectures and stream-based architectures will be explained. In the second part several arguments for using TSS for stream based architectures will be shown. Answers will be given on research on the research questions asked in chapter 3 in the last part of this chapter.

8.1 Simulation of bus-based architectures with TSS

The figures used in this chapter have some communities with the architecture of the video core. These figures are only used for an indication of bus-based and stream-based architectures. An example of a scheme of a bus-based architecture is shown in Figure 8.1-1. Data and control information will be sending over a common bus and all blocks are connected to this bus. So communication between blocks is only done by bus. TSS has proven to have great added value for bus-based architectures. The performance of a candidate architecture can be estimated before the architecture will be implemented. Performance questions can be intensity of bus traffic, performance of a processor etc.

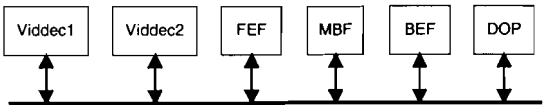

Figure 8.1-1: Bus-based architecture

Streaming based platforms like the VideoCore consists of several blocks of whom the data outputs of a block are directly connected to the data inputs of the following block, see Figure 8.1-2

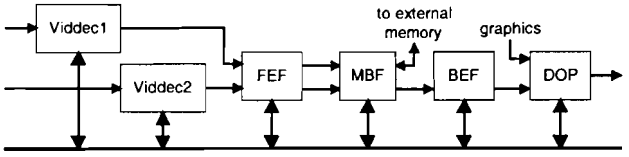

Figure 8.1-2: Stream-based architecture

These kind of platforms are functional static. Data transfers will have one direction between the streaming blocks (Viddec's, FEF, MEF, BEF and DOP). All cores are connected with a bus, which is only used for control by using a processor. The bus is mainly used for transferring status information and control information from the MIPS to the internal control registers in the cores and reading status information from the cores. And the connections between the blocks are used for data communication. The VideoCore is connected with the ControlCore with the help of a common bus.
It's obvious that performance analysis of the bus by bus-based architectures have higher priority than for stream based architectures, because the bus will than be used for data communication and for control information by bus-based architectures.
In this chapter arguments and proofs will be given what the added values is of TSS for a stream-based architecture as the VideoCore.

8.2 Arguments to use TSS

Arguments of the usefulness of a cycle true simulation of TSS models for the stream-based architecture of the VideoCore:

- Higher abstraction, higher simulation speed, shorter turn around time
- Uniform C environment.
- Cycle true / bit true performance estimation
- Satisfy correctness of specification in early design stage.
- Design space exploration, Multi-level simulations
- Library of ReUse modules
- Early software/hardware co-verification

Higher abstraction, higher simulation speed, shorter turn around time

Simulation speed of TSS enables simulations at VideoCore level with reasonable simulation times of 7,5 min.frame.
The results of the experiments in chapter 7 show a large difference in simulation speed between the hardware simulator NcSim (currently used in Nijmegen) and TSS. For that specific experiment 3 is shown that TSS simulates 51,7 times faster than NcSim. Because TSS models are usually modeled at behavioural abstraction level, so TSS models have less lines of code (see chapter 6) and are less complex than VHDL-RTL architecture models. All this results in a shorter turn around time.

Uniform C environment

Models for TSS are written in the 'C' language. Advantage of this is a uniform environment, because everybody has a C-compiler.
System functionality on SoC designs consists of approximately 50-90 percent software. Software engineers use C/C++. System architects use C/C++ (TSS within Philips). Most hardware designers are familiar with C. This makes C the natural language for system-on-chip design. With Moore's Law driving transistor counts ever higher, it is essential to begin design at a significantly higher level of abstraction than hardware description languages (VHDL or Verilog). The approach of describing all system functions in C enables architects to more easily migrate functions between hardware and software implementation [see 12]. Figure 8.2-1 shows that a system architect is the intermediair between the software, hardware and the system. Therefor is it recommended to use a common language.

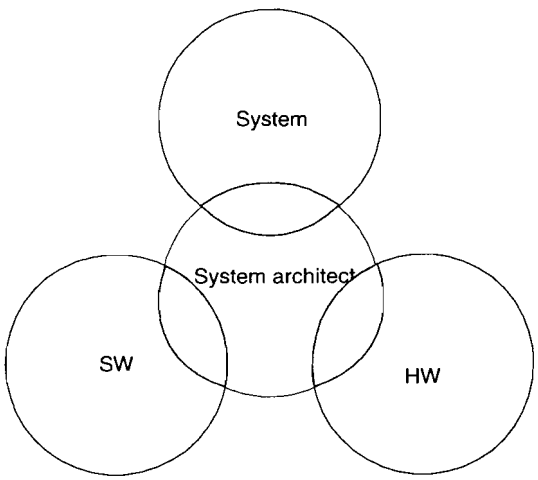

Figure 8.2-1: Elements of an embedded system

Cycle / bit true performance estimation

For the VideoCore are some performance questions defined, which can be answered when TSS-models are available of the VideoCore and ControlCore:

- PI-bus load sufficient if histogram is full in running.
- Verify memory accesses in SDRAM of Video Core and MIPS.

A problem came up during implementing a block of the feature box (MBF) at RT-level. The question came if the memory bandwidth of the SDRAM will be enough by bank switching of the MBF and the MIPS of the ADOC.

Satisfy correctness of specification in early design stage

Specification is normally written in a natural language form. TSS can be used for showing by simulation that a certain system set-up will satisfies its specification. Models have to be written of all hardware blocks. Models can be TSS-models, VHDL-models, and C-models. It's obvious that co-simulation has to be done when other models are used than TSS-models. So legacy blocks in VHDL can still be used. But co-simulation with VHDL-models will reduce simulation speed. (Chapter 7 Experiments). The resulted system set-up will contains the software / hardware partitioning, whereby the hardware mainly modeled is in TSS. Interaction between software and hardware can be verified. Software can be developed before all hardware blocks are implemented in VHDL at RTL-level. So in an early design stage resulted system set-up can be simulated and can be seen as a simulatable specification for hardware and software designers.

Design space exploration, Multi level simulations

TSS can be used for Design Space Exploration whereby one can already evaluate performance impact of several design choices in a system (such as processor type, memory configuration). So ReUse is important. In an early design stage is it possible that not exactly is known what in hardware and what in software has to be implemented. There consists in TSS a process interface block [19] which can connect a sequential C program to the TSS simulation process. This allows simulation of high-level C-programs with cycle accurate TSS models. At the same time it's possible to do co-simulation with VHDL RTL-architecture descriptions by using simulators as NcSim, Speedsim or Leapfrog. This co-simulation makes the so called *C-based sign-off design* methodology possible, which is explained in chapter 4.6.2.

ReUse

TSS has a library with PRISC TSS models, whereof the most models are bit and cycle true. See chapter 8.3 question 3 for more about Reuse.

Early software/hardware co-verification

See chapter 8.3 question 4.

8.3 Answers research questions

What is the added value of TSS for streaming based platforms like the VideoCore in an ADOC architecture for projects in the future? :

1. What is the impact by verification (time and flow)?

In an early stage of design concept verification is possible and performance estimation and evaluation can be done. This is possible with high simulation speed in TSS. A TSS framework of the ADOC was made at Nat.Lab. which is detailed described in [6]. In this frame work the VideoCore is modeled and the ControlCore. Simulations are possible within acceptable times (7.5 minutes / frame) which is shown in this thesis.
So simulation time in TSS is very short in comparison with current hardware simulators as NcSim. (See simulation results in Chapter 7). But its important to know that there is a difference in abstraction level between TSS models (RTL behavioural) and hardware what will be verified at RT-architecture level.
TSS descriptions are not sufficiently detailed to decide whether signals can be or can not be realized within a certain amount of nanoseconds from state variables and inputs. For that level of timing, VHDL or an equivalent gate level description is required. TSS can simulate at the clock tic level: under the assumption that what is described within a clock pulse can actually be realized within a clock pulse, TSS will simulate "clock tic accurate". And TSS will simulate it fastest when all models written in C at a behavioral level. It's possible to write this at an RTL-architecture level but this will increase simulation time.
Co-verification of software with hardware is possible because the availability of TSS models of processors. Cross-compiles code can be executed on these processors. So integration errors of software and hardware can be found earlier.
With the design process going on multi level simulations are possible and useful, due co-simulation of TSS with NcSim, Speedsim or Leapfrog. This co-simulation makes the so called *C-based sign-off design* methodology possible, which is explained in chapter 4.6.2.

Design and verification flow for VideoCore by using Top-down approach:

- Development by the *hardware designers* of a TSS RTL-behavioural models of all blocks where for no TSS models are available. Because at this moment no implementation is available, a bit true and a not very cycle accurate model can be made. Designer will be more thinking what they will implement than how they will implement certain functionality's by making a TSS RTL-behavioural model. Verification has to be done for each individual block. Several hardware designers are needed for making an implementation at RTL architecture level for a certain block. It's recommended that the same hardware designers make a TSS model of that specific block. Than the knowledge of making a TSS model can be used for making the implementation. (This is an added value of TSS.) It's obvious that the same hardware designer will make that part of the TSS model which he will implement later. By making one TSS model of a certain block by several hardware designers it can be expected that less integration mistakes will be found when the sub-blocks will be connected later in the implementation into one block.

This because an agreement is made how the sub-blocks are connected with each other, and can be verified by simulation of the TSS model.

- The developed models can be used for verification at VideoCore level in TSS. It's possible to include the ControlCore as well because the important TSS models for this are available. Software designers have can develop the software for the VideoCore Than co-verification of software with TSS-models of the hardware will be possible. IC-architects can do performance analysis doing system simulation. Verification of the behaviour of the whole system is possible at an early design stage with high simulation speed.
- Each individual hardware designer has to design the RTL architecture implementation of that part whereof he had made a part of the TSS model. At this stage it's clear what each individual designer has to design. When a block is ready, than the equivalent block modeled in TSS can be changed by the implementation of that certain block at RT-behavioural level for simulation. For this simulation a co-simulation of TSS with a hardware simulator has to be done. (See Figure C-based sign-off). Only one block will be changed at a certain moment. With this approach an RTL-architectural implementation of a block can be verified in the system environment. The outputs of the TSS model can be used as reference for the outputs of the implementation at RTL architecture. Because implementations at RTL- architectural of individual blocks will become ready, the TSS RTL-behavioural models can be made more cycle accurate. This valuable for ReUse of TSS models and if needed for more accurate simulations by using the C-based sign flow.

By following such a verification flow and design flow, functionality mistakes, integration problems of software and hardware will be found earlier and earlier system verification is possible.
Turnaround time will be shorter by using TSS for simulations at RT-behavioural level, than the turnaround time at RT-architecture level, this because less rows of code and a significant higher simulation speed of TSS.
It will be expected that by using a top-down design and verification flow the total flow time will be decreased in comparison with the current mainly bottom-up design and verification flow. The effort for making higher level models will be paid back in the verification time of the implementation at RT-architecture level of the whole VideoCore or ADOC [10]. Because most mistakes will be found earlier during simulation with the higher level models. And the hardware designer knows exactly what has to be implemented at RT-architecture level when he has first (made) a TSS model.

2. What is the development effort for making a TSS model?

Time aspect of all TSS models is the clock. And most available TSS models are bit true and cycle true as well, for example the library of PRISC models. Possible data types in TSS are integers, floats, pointers to an arrays and Boolean. Functionality is commonly described at a behavioural manner in processes. RT behavioural level can be seen as the abstraction level of a common TSS model. The level of such a TSS model will be slightly higher than a VHDL-behavioural description. This because the signal assignments of TSS don't have the same behaviour as in VHDL and has 1 clock process for each clock frequency.
Development effort for a TSS model at a RT-behavioural level will be comparable with the development effort needed for a common VHDL-behavioural model. This because these two descriptions have comparable abstraction level.
Figure 8.3-1 shows a function in TSS and figures 8.3-2 and 8.3-2 show a comparable function in VHDL-behavioural.

```
Ouput_FIR  FIR_filtering_coring(tss_int32T input)
{
      Ouput_FIR Result;
      tss_int8T  i;
      tss_int32T deriv_one;
      tss_int32T deriv_two;
      tss_int32T coring_out;
      tss_int32T coeff[8]={4,8,16,22,28,28,24,13};
      static  tss_int32T sig_del[17];
      static coring_dela, coring_delb;
      static int iFirst;

      if (iFirst==0){
           iFirst=1;
           for (i=0;i<17; i++) {
                sig_del[i]= 0;
            }
      }
      for (i=16;i>0; i--) {
           sig_del[i]= sig_del[i-1];
      }
      sig_del[0]= input;
       deriv_one =(((sig_del[0]- sig_del[16]) * coeff[0]) +
              ((sig_del[1] - sig_del[15]) * coeff[1]) +
              ((sig_del[2] - sig_del[14]) * coeff[2]) +
              ((sig_del[3] - sig_del[13]) * coeff[3]) +
              ((sig_del[4] - sig_del[12]) * coeff[4]) +
              ((sig_del[5] - sig_del[11]) * coeff[5]) +
              ((sig_del[6] - sig_del[10]) * coeff[6]) +
              ((sig_del[7] - sig_del[9])  * coeff[7]))/128;
     if (deriv_one > 0) {
           if ((deriv_one - coring_lvl)< 0) {
                coring_out=0;
           } else {
                 coring_out=deriv_one-coring_lvl;
           }
     } else if (deriv_one < 0) {
           if ((deriv_one + coring_lvl) > 0) {
                coring_out=0;
           } else {
                 coring_out=deriv_one + coring_lvl;
           }
     } else {
           coring_out=0;
     }
    deriv_two       =  coring_out-coring_delb;
     coring_delb    =      coring_dela;
     coring_dela    =      coring_out;
     if (svm_scnd_first_b==0) {
           if (coring_out/2 >= 255) {
                Result.svm_coil= 255;
           } else if (coring_out/2 <= -256) {
                      Result.svm_coil= -256;
                } else {
                      Result.svm_coil= coring_out/2;
                }
     } else if (deriv_two >=255) {
                      Result.svm_coil= 255;
                     } else if (deriv_two <= -256) {
                          Result.svm_coil= -256;
                      } else {
                          Result.svm_coil= deriv_two;
                      }
     if (deriv_two >=255){
           Result.dyn_contr= 255;
     } else if (deriv_two <= -256) {
                Result.dyn_contr= -256;
           } else {
                Result.dyn_contr= deriv_two;
           }

     return Result;
}
```

Figure 8.3-1: Function in TSS RTL-behaviour.

```
ARCHITECTURE behaviour OF csn_svm_rgb_fir IS

TYPE coefdelay_line IS ARRAY (0 TO 7)     OF integer;
TYPE delay_line     IS ARRAY (16 DOWNTO 0) OF integer;

CONSTANT coeff         : coefdelay_line := (4,8,16,22,28,28,24,13);

SIGNAL  sig_del        : delay_line := (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
SIGNAL  procen_int     : std_logic  := '0';
SIGNAL  deriv_one      : integer    := 0;
SIGNAL  coring_out     : integer    := 0;
SIGNAL  coring_dela    : integer    := 0;
SIGNAL  coring_delb    : integer    := 0;
SIGNAL  deriv_two      : integer    := 0;

BEGIN
 delay_procen_proc:PROCESS(clk)
 BEGIN
 if (clk'event AND clk = '1')
 than procen_int <= procen  ;
 end if;
 END PROCESS delay_procen_proc;

 p_lum_delay_proc: PROCESS(clk)
 BEGIN
   if (clk'event AND clk = '1')
   than if procen_int = '1'
            than  FOR i IN 16 DOWNTO 1 loop
                       sig_del(i) <= sig_del(i-1);
                  end loop;
                  sig_del(0) <= to_integer(unsigned(sig_in));
            end if;
   end if;
 END PROCESS p_lum_delay_proc;

 coef_times_sample_proc: PROCESS(clk)
 BEGIN
   if (clk'event AND clk = '1')
   than  if procen_int = '1'
            than  deriv_one <= ((((sig_del(0)-sig_del(16)) * coeff(0))  +
                                          ((sig_del(1)-sig_del(15)) * coeff(1))  +
                                          ((sig_del(2)-sig_del(14)) * coeff(2))  +
                                          ((sig_del(3)-sig_del(13)) * coeff(3))  +
                                          ((sig_del(4)-sig_del(12)) * coeff(4))  +
                                          ((sig_del(5)-sig_del(11)) * coeff(5))  +
                                          ((sig_del(6)-sig_del(10)) * coeff(6))  +
                                          ((sig_del(7)-sig_del(9))  * coeff(7))) /128);
            end if;
   end if;
 END PROCESS coef_times_sample_proc;

 coring_proc: PROCESS(clk)
 BEGIN
   if (clk'event AND clk = '1')
   than  if procen_int = '1'
            than  if deriv_one > 0
                than if (deriv_one - to_integer(unsigned(coring_lvl))) <0
                          than coring_out <= 0;
                               else coring_out <= (deriv_one - to_integer(unsigned(coring_lvl)));
                          end if;
                  elsif deriv_one < 0
                  than if (deriv_one + to_integer(unsigned(coring_lvl))) > 0
                     than coring_out <= 0;
                     else coring_out <= (deriv_one + to_integer(unsigned(coring_lvl)));
                     end if;
                  else coring_out <=0;
                  end if;
            end if;
   end if;
 END PROCESS coring_proc;
```

Figure 8.3-2 : Nearly equivalent part in VHDL-behavioural (part a).

```
second_deriv_proc: PROCESS(clk)
 BEGIN
   if (clk'event AND clk = '1')
   than  if procen_int = '1'
            than  coring_dela <= coring_out ;
                coring_delb <= coring_dela ;
                     deriv_two   <= coring_out-coring_delb;
            end if;
   end if;
 END PROCESS second_deriv_proc;

 format_outputs_proc: PROCESS(clk)
 BEGIN
   if (clk'event AND clk = '1')
   than  if procen_int = '1'
            than if scnd_frst_b = '0'
               than if (coring_out/2) >= 255
                        than frst_deriv <= std_logic_vector(to_signed(255,9));
                        elsif (coring_out/2) <= -256
                        than frst_deriv <= std_logic_vector(to_signed(-256,9));
                        else frst_deriv <= std_logic_vector(to_signed((coring_out/2),9));
                        end if;
                     else if deriv_two >= 255
                        than frst_deriv <= std_logic_vector(to_signed(255,9));
                        elsif deriv_two <= -256
                        than frst_deriv <= std_logic_vector(to_signed(-256,9));
                        else frst_deriv <= std_logic_vector(to_signed(deriv_two,9));
                        end if;
                     end if;
                     if deriv_two >= 255
                     than scnd_deriv <= std_logic_vector(to_signed(255,9));
                     elsif deriv_two <= -256
                     than scnd_deriv <= std_logic_vector(to_signed(-256,9));
                     else scnd_deriv <= std_logic_vector(to_signed(deriv_two,9));
                     end if;
            end if;
   end if;
 END PROCESS format_outputs_proc;
END behaviour;
```

Figure 8.3-3 : Nearly equivalent part in VHDL-behavioural (part b).

3. What is the impact on reuse, is it less difficult to make derivatives?

Currently a lot of reusable TSS models consists, so it's possible to make in a short time a system in TSS of reusable models by only making a netlist and building a TSS-executable which consists the TSS models. When the system contains a processor, software has to be developed and the system can be executed.
TSS-models are written at RT-behavioural level. Because of this level the number of lines will be less than descriptions at RT-architecture level. The TSS model described in experiment 3 of this thesis has 3 times less lines of code than the equivalent description in VHDL-architecture. It's expected that the difference in lines of code will increase when more functionality is added in both cases. Because the 3 is the average value and the difference of the individual blocks are higher than 3.
The testbench was in TSS bigger, because special functionality for TSS was included. So TSS models at RT-behavioural level are less complex than VHDL at RT-architecture level. And simulation speed of TSS is much higher. Due these properties it's less difficult to make a derivative of a TSS model than to make a derivative of a model in VHDL at RTL-architecture level.

4. What is the importance of such a model for early software development for a system like the ADOC on a platform as PC?

An example is taken of the design of the CPA at Nat.Lab. The lack of proper abstraction levels and their associated simulation environments have hampered the design of the CPA driver software and the integration of the driver software with the hardware [10]. There for it is recommended that more effort is spent on these aspects in future design projects.
With TSS-models of the whole VideoCore, software designers have a framework where they can work with for developing software before an implementation is made at RT-architecture level. This because TSS-models can be bit true and simulate clock tic accurate. A TSS model can be used by the designers of the device drivers, since all necessary details, such as the MMIO register addresses are known [10].
And verification of device drivers by using a system modelled in TSS will be much faster than when the system is modeled at RT-architecture level. (See simulation speed differences in chapter 7.)

9 Conclusion

It can be concluded that Phase 2 is a useful situation (see Figure 9-1). In this situation a synthesisable RTL-architecture description in VHDL or Verilog has still to be designed. TSS descriptions are not synthesisable, therefor a TSS description at RT architecture level has not much sense. TSS models are meant for RT behavioural level, which can describe a cycle true and bit true behaviour of a hardware block. But cycle true is not always necessary and possible. It is difficult to make a cycle true model in an early stage of design. This because no implementation is available yet. But such a description can still be used as prototyping and as specification for the

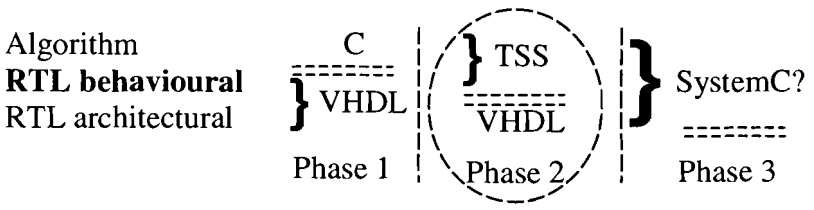

Figure 9-1: Different descriptions for several levels of abstraction

hardware designers. A more experience designer can make a nearly cycle true model before such a description is available.
High speed simulation is possible with TSS due the abstraction level of a TSS RTL behavioural model. For instance, simulation of the VideoCore with 7,5 min/frame (see Table 7.2.1-1). The TSS-description of a part of the DOP is over 50 times faster compared with the equivalent in VHDL at RT-architecture level.
Experiments in chapter 7 show a strong correlation between the amount of clock processes in VHDL and the speed difference with the equivalent TSS model, which has 1 clock process. The difference in lines of code in VHDL and in TSS is negligible for the difference in simulation speed, when there is a significant difference in the amount of clock processes. So to achieve faster simulations, hardware designers have to use a minimum of clock processes.
It's recommended that the hardware designers also develop the TSS model of a block which they will implement. Only than the knowledge of making such a model can be used for making the implementation. The effort for development of a bit true and cycle true TSS RTL behavioural model will be comparable with a VHDL-behavioural model. Such a TSS model can be used as a link to system in a TSS environment. And such a system can contain other bit true and/or cycle true models of library of reusable TSS models. These models can be co-simulated with sequential C-programs at algorithm level without simulation speed decreasing for TSS. Because these kind of programs are faster than models in TSS [19]. Co-simulation is possible with hardware simulators like NcSim, Speedsim and Leapfrog for simulation models written at RT architecture level. Multi level simulation is supported by TSS.
When designers make first a TSS model before making the implementation then it is possible to do system verification in an early stage of design. And driver software can be developed and verified with the hardware in a TSS environment. So integration faults can be found earlier.

10 Recommendation

More experiments needs to be done for proving the strong correlation between the amount of (clock) processes and the simulation time. So it will be expected that the simulation time will be an amount of times faster when the amount of clock processes will decrease with the same amount by merging clock processes.

Glossary

ADL	Assertion Definition Language
ADOC	Analogue Digital One Chip
API	Application Programmer's Interface
BL-Video	Business Line Video
BCL	Beam Current Limiter
CMOS	Complementary Metal Oxide Semiconductor
DFG	Data Flow Graphs
DOP	Digital Output Processor
DTS	Digital Television Systems
DTL	Device Transaction Level
EP	Event Processing
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GDB	Gdb is a an interactive Source-Level Debugger which can be used to debug C programs
HW	Hardware
HDL	Hardware Description Language
IIR	Infinite Impulse Response
IPP	Interconnectivity Processor Peripherals
ISM	Instruction Set Models
NcSim	native compiled simulator
PRISC	Philips Reduced Instruction Set
PWL	Peak White Limiter
RTL	Register Transfer Level
SP	Signal Processing
SPM	Standard Processor Model
SVM	Scan Velocity Modulation
SW	Software
TSS	Tool for System Simulation
Tcl	Tool command language
VHDL	VHSIC (Very High Speed Integrated Circuit) Hardware Description Language

References

[1] Philips ED&T/Synthesis, *TSS User manual, version 3.2, September 2000*

[2] Philips ED&T/Synthesis, *TSS Programmers manual, version 3.2, September 2000*

[3] Frans Theeuwen, Sue XU, *Positioning of TSS in a System Level Design Flow for a SoC*

[4] F. Zandveld, *TSS Cookbook, Na.Lab.*

[6] I.C. Kang, *High-level Modeling of the ADOC*, Nat.Lab. Technical Note 2000/442

[7] Vishal Choudhary, *Cycle-based Verification Methodology for System-on-Chip,* Nat.Lab. Technical Note 2000/476

[5] Paul Stravers and Jan Hoogerbrugge, *The case for single-chip homogeneous multiprocessing. A project preview,* Nat.Lab. Technical Note 2000/106,March 24, *2000*

[6] Rik Jan Zwartenkot, *Recommendation for High-Level design modeling and verification approach for the ADOC project.*

[7] Presentations TSS User Group Meeting, October 11th 2000

[8] E.J.F. Kamps *TSS-SPW co-simulation (Integrating TSS models in Cadence SPW)-* ED&T Num. Nat.Lab/2000/080

[9] J.L.van Meerbergen, An introduction to register transfer level and architectural level synthesis, February 10, 1995

[10] Gerben Essink, CPA Driver Software, Nat.Lab. Technical Note 126/99, September 22, 1999

[11] H.T.J. Zwartenkot, IC Requirement Specification ADOC

[12] Paul Stravers and Jan Hoogerbrugge, The case for single-chip homogeneous multiprocessing A project preview

[13] Sirisha Voruganti, Positioning of NAPA-TSS-VCC in the System Level Design Flow for a SoC (tss_positioning), June 1, 2000

[14] TSS models homepage, IPP Bangalore, http://www.blr.sc.philips.com/groups/ipp/tss/tss.html

[15] SAA6752 EMPRESS Modeling Status - September 1999
Systems Laboratory Hamburg – SLH Dirk Hoppe

[16] Prof. Dr. ir. J. van Meerbergen, Embedded Multi Media Systems in Silicium, Eindhoven University of Technology, Academy year 1999-2000

[17] Maarten van Dommelen, Design Reports for BCL, PWL and SCM

[18] Albers van der Linden, J.P., Kamps, E.J.F, Application code debugging in a TSS system: Using GDB on a MIPS processor model, Nat.Lab. Technical Note 2000/127.

[19] P.E.R.Lippens, A.K. Nieuwland, Co-simulation of Abstract Models with Cycle-accurate Models Using TSS, Nat. Lab. Technical Note 076/98, February 1998.

[20] Philips Semiconductors Southampton DTC, TSS User Guide, December 23, 1999