Stepwise Refinement of Behavioral VHDL Specifications by Separation of Synchronization and Functionality

Claus Schneider, Wolfgang Ecker

Siemens AG, Corporate Research and Development, ZFE T SE 5
D-81730 Munich
E-Mail: Claus.Schneider | Wolfgang.Ecker@zfe.siemens.de

Abstract

We present a new method of behavioral modeling consisting of separation of synchronization and functionality. In this way incomplete specification and incremental refinement can be performed to reduce the modeling effort in early design stages. Compared to known methods our approach allows early cycle based analysis to select appropriate architectures and to perform parallel/serial trade-off. Due to the separation of synchronization and functionality the datapath can be developed independently of the controller and thus enables concurrent engineering. Another important characteristic is the reuse friendly architecture proposed in this paper.

1 Introduction

Top down design methodology is used world-wide to cope with design complexity. So called specifications document interim results of design stages in a natural language. These specifications increase more and more. Currently they are replaced by executable specifications, to make verification, validation, and analysis of design steps and their results possible. Different approaches to abstraction are used to reduce the model expense of executable specifications:

- Explicitly described freedom like don't care ('-').
- Abstraction in time, value and description style
- Application of super symbols like records, subroutines or classes as known from software design.
- Incompleteness by omitting partial behavior like error cases or initialization by reset.

1.1 Related work

A classification of design levels related to abstraction in time, value and description style was presented in [RAM91]. The design cube [EcHo92] models these abstraction levels independently and associates an axis in a three dimensional design space with each of them. Time abstraction namely propagation delay, clock relation and causality is seen as the most important factor in this model. A comparative study of different description or specification languages according to their abstraction mechanisms can be found in [NaGa93]. An extended VHDL-subset for time abstraction is described in [BeSt91] and a pure VHDL based approach can be found in [EcMa93]. The paper [HuDi95] reports on an industrial application of causality as time abstraction. Its benefits are early functional integration. The disadvantage however is that cycle based analysis, which is important for architecture selection and parallel/serial trade-off, can not be performed in early pure causal specification stages.

Special abstract modeling approaches, such as the use of Petri Nets [AbCo90, FRBC91, MüKr93, Ram93] or stochastic system models [HuTo90] are also used for early system evaluation. These approaches do not allow cycle based analysis, either.

The application of software techniques like structured analysis for early real time system modeling is described in [LSK91] and [SKS91]. An application specific approach for architecture evaluation, considering full functionality and timing for analysis, can be found in [PSL91]. Here timing and functionality are modelled in one run and a lot of modeling effort must be spent.

Another approach to reduce modeling effort is the application of incomplete specification and incremental design as proposed in [Hoh91]. Here again, abstraction approaches as described in [Ram91] are used and thus no investigations based on cycle analysis can be performed. We show in this paper a new approach of incremental design enabling early cycle based analysis: Separation of synchronization, representing clock related timing, and functionality.

1.2 Overview

The paper is organized as follows: First our approach of separation of timing and functionality is described in general. Afterwards different modeling alternatives are discussed in relation to this approach. The presentation and discussion of a special modeling style follows subsequent. A verification method and an application example conclude the paper.

2 General Approach and Design Flow

Our approach consists of separation of synchronization and functionality through all design steps including modeling, verification, validation and analysis. Timing is modelled by synchronous controllers and functionality by synchronous or combinational datapaths. Designs with data inherent synchronization can be modelled by a decoder in the datapath that extracts control from the data flow.

Our approach starts with modeling timing by designing and composing controllers. These controllers are either responsible for synchronization or time related control of data operations. Based on this approach a first cycle based timing analysis can be performed. The network of controllers is then extended by dummy datapath operations and

EURO-DAC '96 with EURO-VHDL '96
0-89791-848-7/96 $4.00

data buffers. This architecture allows additional dataflow analysis. A trade-off between buffer size, processing time and parallelism can now be evaluated. The modeling effort to achieve the timing and dataflow model is relatively small, compared with a full functional model.

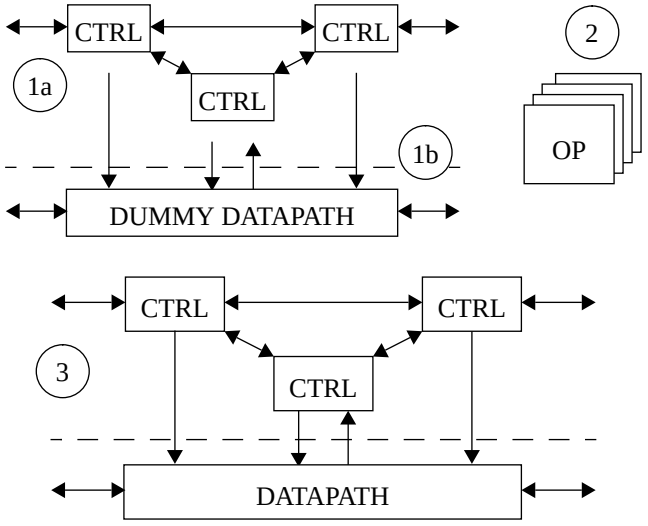

Figure 1: Design Flow

In a second step, functionality is modelled timing independent i.e. all computations are performed in zero time. This second step can be done completely independent of the timing and dataflow modeling and thus enables concurrent engineering.

Finally timing and functionality are integrated. Now a detailed and exhaustive verification considering also data dependent processing time can be performed.

3 Modeling Alternatives

Different modeling approaches were candidates for our approach. We used and extended the architecture taxonomy (SISD, SIMD, MIMD) presented in [GVNG94] for evaluation of the best modeling style.

We distinguished the number of datapaths and controllers, which are used to model one unit. Using controllers is only useful if synchronization but no data processing is required. The number of controllers thereby is determined by the nature of the synchronization problem. The other extreme, using datapaths only can be depicted to pure software, and shall not be further discussed in this paper.

An entity consisting of one controller and one datapath only relates to a typical von Neumann architecture (SISD). This architecture can also be described in one VHDL unit using multiple wait statements for the specification of the control steps. The structured programming approach inherent in this description is in most cases not suitable to express all required transactions in the control flow. Another objection to this description style is, that a new model must be generated for each of our design steps due to the fact that VHDL does not support different subroutine bodies. We detected moreover, that a separated single controller is in most cases not suitable due to the fact that control and synchronization problems are in most cases not sequential. Thus single controller single datapath and single controller multiple datapath (SIMD) modeling is not useful for our approach.

Multiple controller multiple datapath architectures can be partitioned into two classes: A net of single controller single datapath architectures as known from multiple processor computers (MIMD) or a glue of controllers and datapaths. Shared controllers are useful but a set of datapaths is seen as an overhead, which only reduces simulation performance.

For our approach we selected a multiple controller single datapath (MISD) model. This partition makes incremental extensions, using VHDL structural and configuration description capabilities, very easy to model. The concurrent control flow can be modelled in a very natural way and a single datapath ensures high simulation speed.

4 Modeling Issues

Before we present our modeling approach in detail, some general remarks about behavioral modeling shall be made:

- **Correctness** and **Readability** are the most important issues in behavioral modeling. An error in the specification due to an incorrect behavioral model, that is detected in the RTL model for the first time, causes a long iteration loop back to the specification which increases the development time. Besides, behavioral models are part of the specification and because of that they should be easy to read.
- **Simulation Performance** is due to the huge amount of testcases important and therefore should be considered. But it has only second priority to correctness and readability.
- The **Level of Abstraction** is a trade-off between development time and meaningfulness. On the one hand the behavioral model must abstract from the later implementation to reduce modeling effort and simulation time, to get a stable specification as soon as possible. On the other hand the model must be precise enough to get clear statements about the system behavior.

4.1 Overall Architecture

To achieve both correctness and readability, behavioral models should be described in a natural (problem specific) way. This means the modeling style (sequential - concurrent) should be as close as possible to the problem.

A client-server handshake for example has a concurrent and a matrix-multiplication a sequential character. Each of these problems should be described in the given way. In addition to that the client-server synchronization belongs to the controller and the matrix-multiplication to the datapath. In fact we detected this concurrent character of the controller and the sequential (algorithmic) character of datapath operations in a lot of our applications.

The datapath is completely separated from the controller to allow a stepwise refinement of the model as described in the previous chapter and to separate the mainly concurrent controller domain from the mainly sequential datapath domain. Therefore the controller consists of several concurrent processes, but for the datapath a single process model is worth striving for.

4.1.1 Controller: concurrent vs. sequential

The controller model is a trade-off between the number of

concurrent control processes on the one hand and the complexity (number of states and transitions i.e. lines of VHDL code) of each process on the other hand. The level of concurrence that should be chosen for the behavioral model depends heavily on the character of the control problem.

A serialized one process model of a concurrent control problem for example minimizes the communication effort (number of control signals), but it drastically increases the complexity, because the number of states of the resulting controller is the cross-product of all states of the concurrent controllers. Moreover the use of variables for communication between concurrent controllers inside a single process, would lead to the need to re-model the delayed signal assignment and update characteristics for the communication variables.

On the other hand it makes no sense to partition an already sequential control problem into several concurrent controllers, because the one process sequential solution is closer to the (trained) human thinking and thus easier to understand. The sequential solution also increases the simulation performance.

Therefore the controller should be as concurrent and as sequential respectively, as the control problem itself. This approach leads to a higher model quality, because no error-prone transformations, from concurrent to sequential and vice versa, are needed. In addition the readability of the code is improved, because the problem specific modeling style in many cases is the most compact one.

4.1.2 Single Process Datapath

Due to separation of control and data, the datapath has the task to store and process data. Data processing reaches from simple data transfers between two memory locations to complex operations like a matrix-multiplication. To facilitate parallel work of complex operations, queuing memories are often inserted between operations. Compared with RTL at the behavioral level complex operations are not divided into single steps. They are executed within one processing cycle. Therefore a queuing memory must supply all data for the complete operation in parallel instead of single data words in a sequential order. The higher the level of abstraction, the wider the memory interfaces must be. In case of the matrix-multiplication, at RTL the operation is performed one matrix element after another and therefore the memory interface must only deliver single matrix elements. In comparison to that, at behavioral level the complete operation is done in one step and the queuing memory must supply a complete matrix in parallel. In addition to the wide interfaces, for system simulation a huge amount of data must be processed and therefore a high simulation performance is needed.

Because of these facts the datapath should be modelled in one process whenever possible. The higher simulation performance is achieved by using variables instead of signals inside the datapath process and sequential statements to model the operations. Only the interface to the controller must be modelled using signals even if this decreases the simulation performance. Another benefit of the single datapath process is the reduced modeling effort, because inside the datapath process each operation has random access to all data memories without any modeling overhead for data busses. Finally, interfacing with variables instead of signals demands the algorithmic thinking.

4.2 Datapath Modeling

The single process datapath is divided in memories (MEM) for data storage and operations (OP) for data processing as shown in Figure 2. Complex operations are encapsulated using subprograms to structure the code and to facilitate separate verification. In comparison to the operations the memories (modelled as variables) are structured by the datatype and not by a new level of hierarchy (dashed boxes).

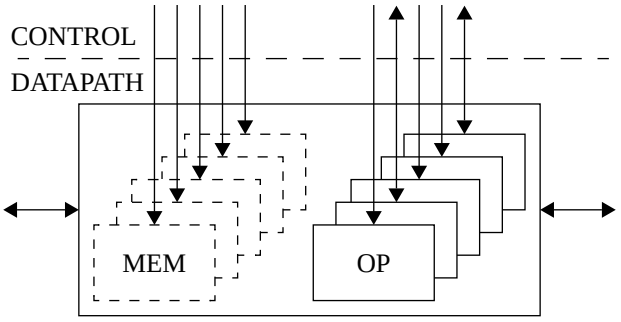

Figure 2: Datapath Model

Both, memories and operations, are controlled by the separated controller part. The memories (e.g. queuing memories) are controlled by READ and WRITE addresses. The operations for example receive a START signal and, if the processing time is not constant but data dependent, they return the required processing time back to the controller. If the data flow includes control information a special decoder operation is modelled to generate control signals for the controller part.

All operations are described in a sequential order and executed in zero simulation time. To facilitate a quasi-concurrent execution of all operations no wait statements are allowed inside the single datapath process and inside the subprograms called from there. The process sensitivity list includes all asynchronous signals (e.g. reset, (macro)-clocks, START-signals) but no data signals. The order of the operations inside the single datapath process is determined by the data flow. Only feedback loops inside the datapath require special handling (e.g. signals instead of variables for the communication). For operations that communicate through a memory, the controller guarantees that a memory cell has been written before it is read.

4.3 Reuse

The separation of control and data and the encapsulation of complex datapath operations eases reuse. For example, since VHDL supports no generic datatypes it is very difficult or even impossible to model a queue unit consisting of controller and memory for performance and for reuse (variable datatype). This is no problem by separating the controller from the datapath as shown below.

The same as to the controller applies to the datapath as well. Reuse is made easier by separating the functional model of a datapath operation from the scheduling controller and by encapsulating the model using subprograms, too. In addition to that the usage of subprograms to encapsulate complex datapath operations (algorithms) is very close to software. Therefore the conversion of algorithms from software to VHDL subprograms is a straight forward task.

In the following chapters two useful controllers (reuse candidates) are presented: A queue model and an operation scheduler.

4.3.1 Queue Model

The queue model is divided into two parts: The queue controller (QC) and the queue memory (QM). The only interfaces between both parts are the read and the write address from the controller. At control level the queue controller uses data strobe and data request signals to interface with other control blocks, Figure 3.

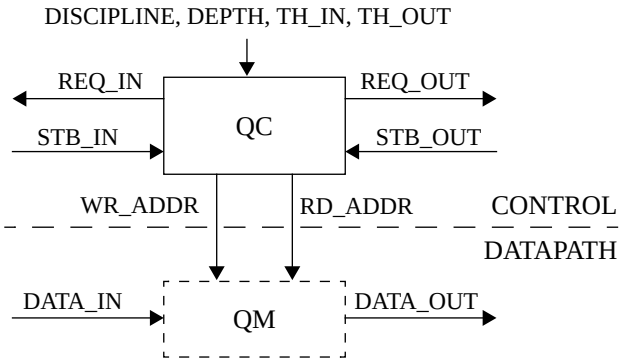

Figure 3: Queue Model

At datapath level the queue memory, implemented as a one dimensional array of any datatype, is accessed by using addresses from the controller as an index. Because of this separation the controller is completely datatype independent and therefore can be reused easily. To cover a wide range of applications the queue discipline (FIFO, LIFO), the queue depth and thresholds for input and output data request are parameterizable. The depth value is also used to define the size of the queue memory in the datapath. A VHDL code example for a reusable queue controller is shown in Figure 4.

A detailed discussion of the VHDL code would exceed the paper. Thus we highlight some modeling specials only:

- The queue model is parameterizable by the discipline, depth and thresholds for the request signals and therefore can be used in a wide range of applications. For architectural trade-off the depth is an important parameter, because it has a big influence on system performance. In some cases a queue has to be disabled completely to connect two operations directly. This can be achieved easily by setting the depth value to zero. Therefore in behavioral models these unusual (from an RTL point of view) border values require special attention.
- Another important issue of behavioral models are built-in checks as for example the over-/underflow check, which is modelled by an assertion statement.
- The concurrent procedure call mechanism is a method to encapsulate a process for reuse. The queue controller procedure has to be called concurrent (as a concurrent statement) and thus is a process. In comparison to a sequential procedure call the concurrent procedure is never left (LOOP) and therefore the variables inside the procedure retain their values.
- The above VHDL model has a synchronous reset. An asynchronous reset can easily be obtained by adding the reset signal to the sensitivity list of the wait statement.

```
PROCEDURE queue_ctrl (
 CONSTANT Depth, ThIn, ThOut: IN natural;
 CONSTANT Discipline: IN queue_discipline;
 SIGNAL Clock, Reset, StbIn, StbOut: IN bit;
 SIGNAL AddrIn, AddrOut: OUT integer;
 SIGNAL ReqIn, ReqOut: OUT bit   ) IS
 VARIABLE FillCnt_v: integer:= 0;
 VARIABLE AddrIn_v,AddrOut_v: integer RANGE 0 TO Depth-1;
 VARIABLE AddrEnable: boolean:= (Depth>1) AND (Disci-
pline=FIFO);
BEGIN
 IF Depth = 0 THEN
  LOOP
   ReqIn <= StbOut; ReqOut <= StbIn;
   WAIT ON StbIn, StbOut;
  END LOOP;
 END IF;
 LOOP
  IF Reset = '1' THEN
   AddrIn_v:=0; AddrOut_v:=0; FillCnt_v:=0;
  ELSE
   IF StbIn = '1' THEN
    IF AddrEnable THEN
     AddrIn_v:= (AddrIn_v+1) MOD Depth; END IF;
    FillCnt_v:= FillCnt_v+1; END IF;
   IF StbOut = '1' THEN
    IF AddrEnable THEN
     AddrOut_v:= (AddrOut_v+1) MOD Depth; END IF;
    FillCnt_v:= FillCnt_v-1; END IF;
  END IF;
  ASSERT (FillCnt_v<=Depth) AND (FillCnt_v>=0)
  REPORT "Queue over-/underflow" SEVERITY ERROR;
  FillCnt <= FillCnt_v;
  IF Discipline = FIFO THEN
   AddrIn <= AddrIn_v; AddrOut <= AddrOut_v;
  ELSE -- LIFO
   AddrIn <= FillCnt_v;
   IF FillCnt_v > 0 THEN AddrOut <= FillCnt_v-1;
                    ELSE AddrOut <= FillCnt_v; END IF;
  END IF;
  IF FillCnt_v < ThIn  THEN ReqIn  <= '1';
                       ELSE ReqIn  <= '0'; END IF;
  IF FillCnt_v > ThOut THEN ReqOut <= '1';
                       ELSE ReqOut <= '0'; END IF;
  WAIT ON Clock UNTIL Clock = '1';
 END LOOP;
END queue_ctrl;
```

Figure 4: VHDL listing of the queue controller

4.3.2 Operation Scheduler

Like the queue controller, the operation scheduler is another useful controller of our approach to model the timing behavior of a datapath operation. Let's take a complex datapath operation like a matrix-multiplication, that is performed in two steps. In the first step a source matrix is read, processed and stored in an intermediate memory. In the second step the intermediate matrix is processed and the results are written to the output, as shown in Figure 5.

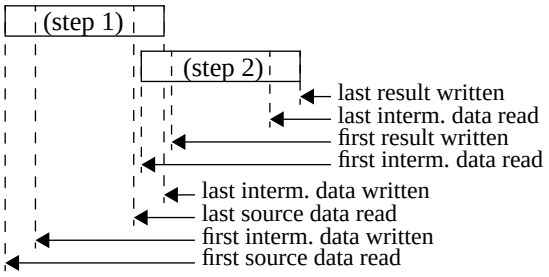

Figure 5: Schedule of a two step operation

The datapath operation model abstracts from this schedule (timing behavior) and performs the complete operation in one subprogram call. It is now the task of the operation scheduler to model the timing behavior in the controller part. In Figure 6 the operation scheduler (OS) is embedded between two queue controllers (QC).

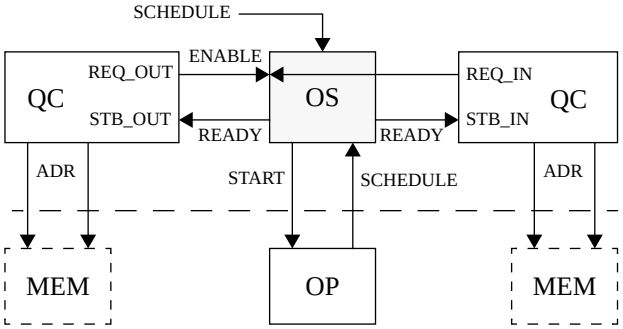

Figure 6: Operation Scheduler

The operation can only be executed if both - the source and the result queue - signal a strobe request to the operation scheduler. After the scheduler has received an ENABLE, which is the logical AND of both strobe requests the operation is executed (START). If the processing time is not constant but data dependent the operation returns a schedule back to the controller. Otherwise a constant schedule, which is a parameter of the controller, is taken to generate READY-signals. In our example one READY-signal would strobe the source queue (source matrix completely read) after the first processing step and another READY-signal would strobe the result queue (result matrix completely written) after the second step.

5 Verification

Like the modeling process, the verification is performed in three main steps i.e. the verification environment (testbench) is growing together with the model.

5.1 Verification Steps

In the first step only the controllers and the handshake between them are modelled and verified. In many cases the standard controllers (queue, scheduler) of the behavioral model can be used in the testbench as well. In case of data independent processing time, no datapath model is needed for performance analysis and therefore the controller outputs leading to the datapath can be left open. But in preparation for the final step and for debugging purposes a dummy datapath with all memories (variables) and dummy operations, that only pass data from the input to the output without processing, is added to the controller. Together with stimuli generators, that generate special test data (e.g. serial numbers), the dataflow can be analyzed and debugged easily.

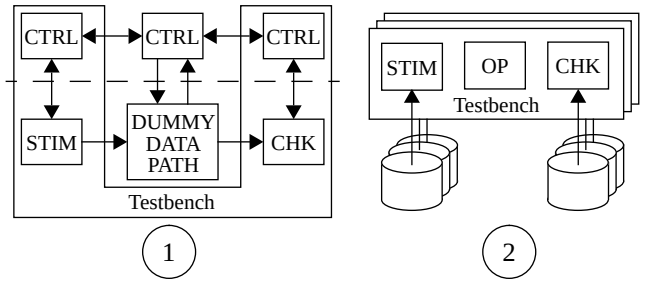

Figure 7: Verification Steps 1 and 2

In the second step, complex datapath operations are modelled and encapsulated by subprograms. The subprogram bodies are not placed inside the single datapath process but in a package to facilitate separate verification and reuse. The stimuli generators and the data checkers are encapsulated in subprograms inside a package, as well. For each complex operation a testbench consisting of operation, data generator and data checker is assembled and a separate functional verification is performed.

The second step can be performed in parallel to the first one and therefore this approach enables concurrent engineering, not only for the design, but also for the verification. As an example, the operations can be derived from a software model by the systems team, having the functional requirement specification in mind. The controllers can be composed by members of the ASIC team, considering scheduling and time budgeting.

In the final step the parts of the previous steps are integrated. For that, in the dummy datapath of the first step, the dummy operations are replaced by the corresponding subprogram calls to build the final model. In the testbench the dummy data generators and checkers are replaced with the ones from the separate functional testbenches of the second step.

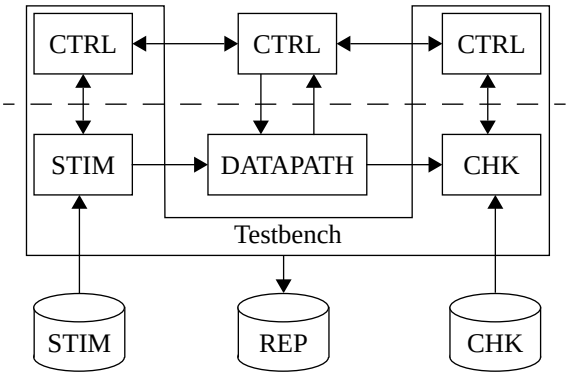

Figure 8: Verification: Final Step

5.2 Benefits

The benefits of this modeling and verification approach can be summarized as follows:

- Reduced verification effort and easy debugging at each step due to lower complexity (divide and conquer).
- The encapsulation of complex operations by subprograms inside a package increases the readability of the single datapath process.
- The integration of controller and datapath and the verification of the complete system model is much easier, because all parts are already verified separately.
- Due to separation of control and datapath the granularity (complexity) of building blocks is lower, the modeling of flexible reusable blocks (e.g. queue) is easier and the reuse frequency is higher.

6 Application

Our approach is demonstrated with an industrial application. A system consisting of a DMA controller and several DMA devices is chosen to show the usage of a queue controller not only for the behavioral model, but in particular to

model data source and sink in the testbench. For performance analysis only the controllers are modelled.

The DMA controller has the task to transfer data between host and device bus. At device bus side the controller (DC) selects one DMA channel for the transfer according to the request signals (DREQ) from the DMA devices and the internal priority mechanism by activating the corresponding acknowledge signal (DACK). Data is transferred with strobe and ready signals (DADS, DRDY). A similar control mechanism applies to the host side (HC). The DMA channels are modelled using the queue controller (QC) configured as a FIFO, Figure 9.

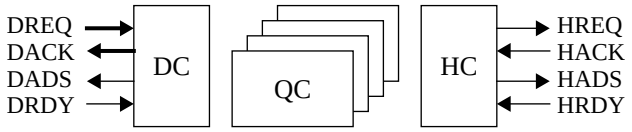

Figure 9: Structure of the DMA controller

In the testbench the DMA devices stimulate the DMA controller acting as data source or sink. The model of a source is shown in Figure 10. A sink model has the same structure, but '_IN' and '_OUT' signals are swapped.

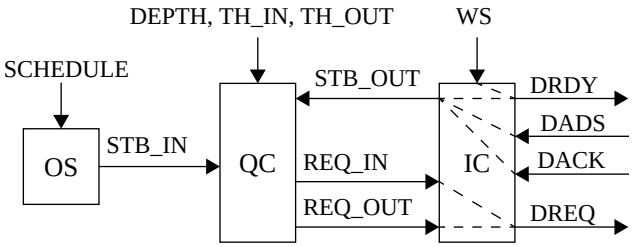

Figure 10: DMA device model (data source)

A DMA device is characterized by a buffer DEPTH and thresholds that control the request for a transfer. These characteristics are modelled by a queue controller configured as a FIFO (QC). An interface controller (IC) generates the request (DREQ), the strobe (STB_OUT) and the ready signal (DRDY) according to the wait-states (WS) of the device. In performance analysis another characteristic is the average transfer rate of the device, which is modelled by an operation scheduler (OS). The schedule for the OS (period between two strobe pulses of STB_IN) can be calculated using the following formula: SP = 1 / (CP * DR * DW)

The schedule period SP is the reciprocal value of the product of clock period CP (in seconds), average data rate DR (in Bytes/sec) and data bus width (in Bytes).

7 Conclusion

We presented a new behavioral modeling approach consisting of separation of synchronization and functionality based on a multiple controller single datapath architecture. The proposed problem specific modeling style of the controller doesn't need error-prone transformations (concurrent/sequential) between thinking and model, in many cases is the most compact one and therefore improves correctness and readability. On the datapath side the readability is increased by structuring the memories by the datatype and by encapsulation of operations using subprograms. Simulation performance is achieved using variables instead of signals inside the single process datapath. Another benefit of the separation of controller and datapath is the good trade-off for the level of abstraction: One the one hand the model is abstract like software (datapath) and on the other hand its synchronization (controller) is close to a cycle accurate RTL model. In addition, the productivity can be increased be deriving datapath operations from an existing software model and by reusing standard controllers (e.g. queue, scheduler). The proposed architecture facilitates not only stepwise refinement of specifications, but also incremental verification. Due to the separation of controller and datapath, concurrent engineering for both the modeling and the verification process is made possible.

8 Bibliography

[AbCo90] Aboulhamid, M.; Cordeau, M.: *System Level Modeling in VHDL using Timed Petri* Nets. Proceedings of the EURO-VHDL'90.

[BeSt91] Benders, L.; Stevens, M.: *Petri Net Modeling of Task Level Behavioral VHDL for VLSI*. Proceedings of the EURO-VHDL'91.

[EcHo92] Ecker, W.; Hofmeister, M.: *The Design Cube - A Model for VHDL Designflow Representation*. Proceedings of the EURO-VHDL'92.

[EcMä93] Ecker, W.; März, S.: *System level design using VHDL: A case study.* Proceedings of the CHDL'93.

[FRBC91] Fermy, A.; Rossignol, B.; Bakowski, P.; Calvez, J.: *Tools to Design at a Functional Scheme Level using VHDL*. Proceedings of the EURO-VHDL'91.

[GVNG94] Gajski, D.; Vahid, F.; Narayan, S.; Gong, J.: *Specification and Design of Embedded Systems*.Prentice Hall, 1994.

[HaBr95] Hashmi, K.; Bruce, A.: *Design and Use of a System-Level Specification and Verification methodology.* Proceedings of the EURO-VHDL'95.

[Hoh91] Hohl, A.: *Incremental Design - Application of a Software-Based Method for High-Level Hardware Design with VHDL*. Proceedings of the EURO-VHDL'91.

[HuDi95] van den Hurk, J.; Dilling, E.: *System Level design, a VHDL Based Approach*. Proceedings of the EURO-VHDL'95.

[HuTo90] Hubbard, P.; Torres, J.: *Using VHDL for High-Level and stochastic System Modeling.* Proceedings of the EURO-VHDL'90.

[LSK91] Lahti, J.; Sipola, M.; Kivelä, J.: *Behavioral System Modeling with structured Analysis and VHDL*. Proceedings of the EURO-VHDL'91.

[MüKr93] Müller, J.; Krämer, H.: *Analysis of Multi-Process VHDL Specifications with a Petri net Model*. Proceedings of the EURO-VHDL'93.

[NaGa93] Narayan, S.; Gajski, D.: *Features Supporting System-Level Specification in HDLs.* Proceedings of the EURO-VHDL'93.

[PSL91] PitKänen, P.; Skyttä, J.; Laakso, T.: *Comparison of Digital Filter Architectures* Using VHDL. Proceedings of the EURO-VHDL'91.

[RAM91] Rammig, F.: Approaching System Level Design. Proceedings of the EURO-VHDL'91. Proceedings of the EURO-VHDL'91.

[Ram93] Rammig, F.: Modeling Aspects of System Level Design. Proceedings of the EURO-VHDL'93.

[SKS91] Sipols, M.; Kivalä, J. Soininen, J.: System Real Time Analysis with VHDL from Graphical SA_VHDL. Proceedings of the EURO-VHDL'91.