ADEN: AN ENVIRONMENT FOR DIGITAL RECEIVER ASIC DESIGN

Thorsten Grötker, Peter Zepter, Heinrich Meyr

Integrated Systems for Signal Processing
Aachen University of Technology
D-52056 Aachen, Templergraben 55, Germany
e-mail: {groetker,zepter,meyr}@ert.rwth-aachen.de

ABSTRACT

Different levels of abstraction are suited for algorithm design and hardware architecture development. This paper presents a tool (ADEN) that provides a link from system design to VLSI implementation. It generates synchronous timed descriptions of digital hardware from dynamic data-flow system level configurations. It allows to make use of optimized architectures available for a broad range of communication system components. These components are kept in the extensible *ComBox* library which provides means to characterize their data-flow and timing properties. The design methodology together with the tool operation and the library concept will be explained. An actual design example is presented to demonstrate effectiveness of this approach.

1. INTRODUCTION

Different levels of abstraction are suited for algorithm design and hardware architecture development when targeting towards a digital receiver. Using data-flow semantics for algorithm design has a major advantage: it does neither anticipate implementation specific details that do not influence the algorithmic performance nor implementation related signals like clock and reset. Therefore it does not imply unnecessary restrictions on the implementation. Furthermore the simulation efficiency is higher than that of simulators working on the register transfer level ([1], [2]). However a state of the art design process for fully synchronous receiver components will include the use of hardware description languages (HDLs) that offer access to logic synthesis and (event-driven) simulation of the synthesized hardware.

A typical design process does not only involve a one way (top-down) transition from higher to lower levels of abstraction. To obtain efficient systems, algorithm and architecture have to be optimized jointly ([3]). This involves the troublesome transition between different description styles and the use of different tools and libraries.

We developed the ADEN compiler to overcome these difficulties. ADEN takes a block-diagram used in the data-flow simulation system *COSSAP* [1] as primary input and generates a synthesizable VHDL description of a synchronous clocked system. The implementations for each block of the data-flow graph can be selected from the *ComBox* implementation library (see figure 1). The component characterization in the *ComBox* library provides ADEN enough

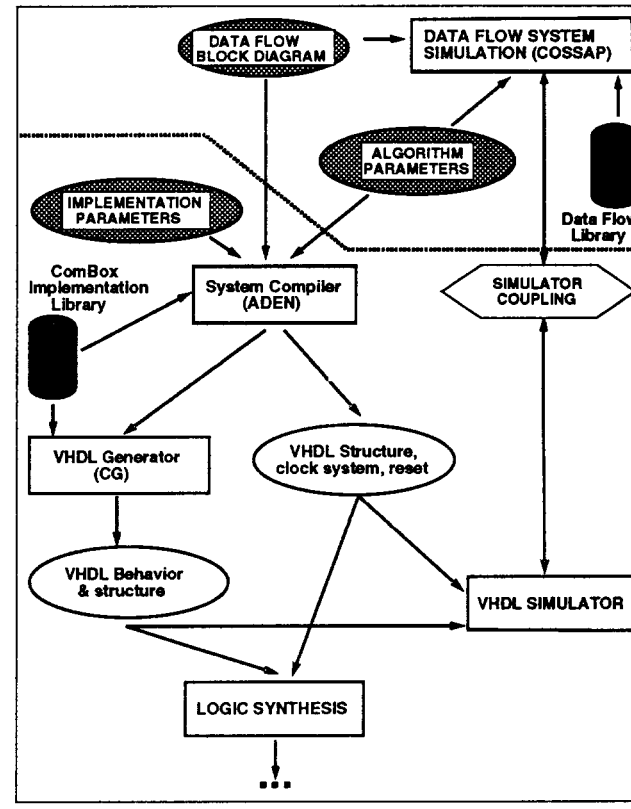

Figure 1: ADEN design system overview

information on data-flow and timing properties to take advantage of complex components like filters, interpolators or synchronizers when generating a multi-rate dynamic data-flow system. Related approaches ([4], [5]) do not provide this functionality. These require the specification of timing behavior and the configuration of timing and control related signals already on the system level and provide a very rigid translation to a hardware description language.

Section 2 describes the application domain and it's requirements. Section 3 explains the *ComBox* library concept. The ADEN design methodology is described in section 4. Section 5 contains a description of the steps that are performed during system compilation and section 6 presents an application example.

[1] *COSSAP* is a trademark of *Synopsys Inc.*

0-7803-2431-5/95 $4.00

2. APPLICATION DOMAIN

ADEN has been developed for the design of high to medium throughput signal processing systems like digital receivers for communication links. These systems show specific characteristics that have to be considered in the design process.

- Digital receivers often use multiple data rates and dynamic data-flow.
- There is only little global control.
- The systems are modeled using blocks of relatively coarse granularity like filters or decoders. Optimized architectures are well known for a broad range of such components.

The approach presented here is complementary to state of the art circuit design methods including the use of high level synthesis tools[6]. The user can take advantage of existing optimized hardware blocks and the combination of different architectural styles.

3. LIBRARY CONCEPT

The *ComBox* library maintains two abstraction levels - the data-flow level and the implementation level. A single data-flow model (that corresponds to a block in the data-flow simulation) may have several implementations that share the same arithmetic-true behavior yet have different timing properties. The clear separation of the two abstraction levels in the *ComBox* is the basis for the design methodology described in section 4.

3.1. Dataflow Model

Data-flow [7] systems are described as networks of blocks performing the signal processing and signals connecting the ports of those blocks. In static data-flow the blocks consume and produce a fixed number of samples at their ports during each activation. Although a broad range of algorithms can be described with means of static data-flow it imposes obstacles when modeling complete signal processing systems. Therefore ADEN makes use of a more general approach: dynamic data flow[7].

In order to obtain efficient implementations ADEN imposes some restrictions on the blocks' dynamic data-flow behavior:

- The data rates (the number of samples produced or consumed per activation; shown in circles at the blocks' ports in figure 2) have to be specified for each port of a block.
- The data rates consist of two parts; a symbolic rate that is either 0 or 1 depending on a control condition and a static rate. The control condition has to depend solely on the current value of a so-called control port (e.g. `1 if (control is odd) else 0`).
- A control port has to be a static port with rate 1.

This data flow model allows to specify all configurations in the considered application domain without imposing unnecessary restrictions.

3.2. Timing Model

The target architecture is a fully parallel, synchronous circuit with central reset and clock generation subsystems. The blocks exchange data items at fixed time intervals called *iteration interval*. It is sufficient to specify the iteration interval for the port with the highest static rate (this *intrinsic* iteration interval is 1 in the example shown in figure 2), the other iteration intervals can be derived from it considering the static data rates. Furthermore the ports may have different port delays, i.e. the differences in time (measured in multiples of clock cycles) those ports read/write the first data item of an activation compared to the port with the first I/O operation. In figure 2 the port delays are indicated by numbers in rectangles at the blocks' ports. Note that these processing delays have to be fixed, i.e. independent of the current input values and the states of the blocks. Data-dependent computation times can be modeled using dynamic ports.

4. DESIGN METHODOLOGY

The output of the data-flow level design phase is an arithmetic true description of the algorithm without any implementation decisions (the separation of data-flow and implementation level is indicated by a dotted line in figure 1).

The amount of information gathered up to now marks both; the starting point of the implementation phase and the reference model for functional verification of the implementation. The latter one can be achieved using a coupling between the *COSSAP* simulator and a VHDL simulator as shown in figure 1 [8,4].

The user may select implementations from the *ComBox* for some blocks and specify implementation parameters like the number of pipeline stages. ADEN then reads the characterization for each implementation from the *ComBox* to obtain all necessary data-flow and timing information to generate VHDL code for

- block instantiation and interconnection
- hierarchical clock and reset generation subsystems
- control signal generation (which includes the generation of *data-valid* signals for external dynamic ports) and
- the instantiation of registers (*shimming delays*) for timing synchronization and implementation of algorithmic delays.

VHDL descriptions of the implementations are obtained by invoking a VHDL generator program (CG) that interfaces

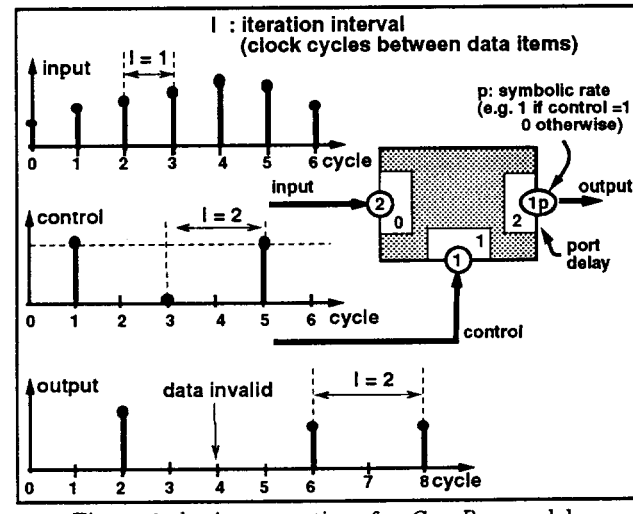

Figure 2: basic properties of a *ComBox* model

with the *ComBox*. The following logic synthesis process may unveil some design bottlenecks in the first iterations. The consequences may be choosing different implementations for a specific block from the *ComBox*, using different ADEN compile options (e.g. concerning input times, register distribution or reset generation), selecting a different algorithm for a subsystem or re-designing a specific implementation.

ADEN is able to generate multirate dynamic data-flow systems that are *correct by construction* for timing related issues using blocks with complex timing behavior. This automated system compilation makes is possible to test a broader range of implementation alternatives compared to a handcrafted system assembly. The users are encouraged to develop a consistent library of re-usable implementations. It is advantageous that a system simulation remains possible throughout all steps of the design process. The clear mapping of data-flow blocks to hardware keeps the natural partitioning of the algorithm and simplifies backtracking of design bottlenecks.

5. SYSTEM COMPILATION

The system compilation process is divided in two steps: data-flow analysis and timing.

5.1. Data-Flow Analysis

The analysis of the data-flow properties of the entire system is necessary to check whether it can be implemented and to obtain information necessary for timing. Because limited buffers sizes for an infinite execution of the data-flow graph with an arbitrary input sequence must be achieved, the graph is required to be *strongly consistent*. To check methods similar to those in [7] are applied [9]. If the following additional requirements are fulfilled it is possible to check for the non-terminatingness of the graph by trying to execute a complete system iteration with all symbolic rates set to 1:

- Two control conditions may not be mutually dependent, i.e. the generation of a control signal may not depend on the value of another control signal and vice versa.
- A single control item has to be sufficient to execute a complete iteration of a dynamic subgraph that depends on this control signal.

5.2. Timing

The timing is divided into two steps; first the static timing that yields the proper (system) iteration interval and the number of shimming delays on the edges. During static timing the dynamic ports are considered static. The second phase is dynamic timing, where gated clocks are introduced for dynamic subgraphs containing algorithmic states.

The major equation for the static timing describes the output time $t_{out}(e)$ of an edge

$$t_{out}(e) = t_{in}(e) + d_s(e) - s(e) * I(e)$$

where $d_s(e)$ is the number of shimming delays on that edge, $s(e)$ is the number of initial values on that edge and $I(e) = I_r(e) * I_s$ is the iteration interval (see figure 3). Initial values on an edge result from an algorithmic delay operation or *separator*. The number of initial values (*separator multiplicity*) is indicated by the number inside the diamond in figure 3. The relative iteration interval $I_r(e)$ is a minimum value computed from the maximum number of data items that have to be transmitted within one period of the system execution. I_s is a system-wide multiplicand.

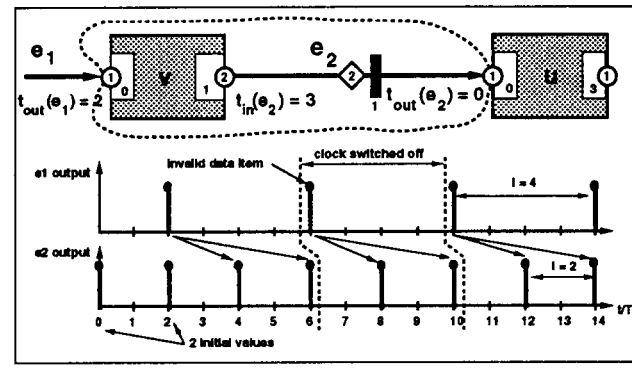

Figure 3: timing of a dynamic subgraph

The timing problem for static data rates can be solved by applying a procedure from [10].

ADEN has to prevent register loading during phases where no valid input is available for those blocks and edges within dynamic subgraphs that contain algorithmic states. Two facts complicate this task:

- We cannot directly access internal registers of a *ComBox* block but have to disable register loading for the complete block.
- Some separator bearing arcs may not contain enough registers to keep all initial values (since the appropriate timing delay of $s(e) * I(e)$ clock cycles may be partially realized by pipelined architectures in the same branch; see figure 3).

A set of sufficient conditions and a clustering scheme could be developed, which allows to find subgraphs where all register operations can be disabled jointly (indicated by the dotted line in figure 3). The implementation is based either on gated clocks or load-enable registers. If no proper clustering can be found the user must either specify the blocks using finer granularity or use an implementation with different timing properties. For dynamic subsystems where either all data items are read from system inputs or written to system outputs the clustering does not become necessary.

After performing the timing, ADEN checks that all blocks are working with their intrinsic iteration interval (or at least with a multiple of it in the case of blocks that do not maintain an algorithmic state). Finally ADEN generates the VHDL code (see figure 1).

6. APPLICATION EXAMPLE

ADEN has been used for the design of a DMSK transceiver for packet-based mobile communication networks shown in figure 4. It consists of the following principle processing steps [11]: after a digital baseband conversion followed by a rectangular to phase conversion and the computation of the differential phase, the timing and frequency offsets are estimated jointly. The frequency correction is followed by an interpolation of the phase samples to provide 8 values

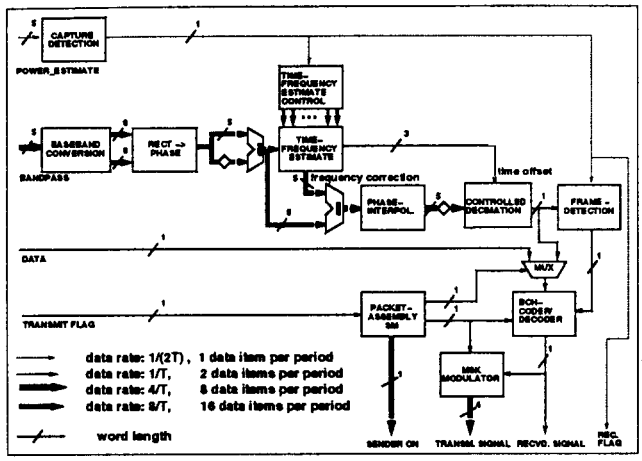

Figure 4: DMSK transceiver designed with ADEN

per symbol, one of which is selected based on the estimated time offset. The output is fed into the frame detector and the BCH codec. The blockdiagram shows that the system uses multiple data rates; the necessary control flow can be modeled using static and dynamic data flow. The required data rate of 2.5MBit/s together with the eightfold oversampling at the input leads to a minimum clock frequency of 20MHz. The resulting processing power is about 1.6 GigaOps/Second.

The implementation phase started with blockdiagram consisting of more than 600 primitive (i.e. non-hierarchical) blocks. A first prototype of the implementation could be generated by using ADEN to assemble a synthesizable VHDL description of the complete transceiver. Cost intensive parts could be easily identified. Apart from selecting different implementations for primitive blocks (e.g. different variants for rectangular to phase conversion) and using different ADEN compile options the quality of the design could be strongly improved by

- exploiting algorithmic trade-offs (e.g. smaller wordlengthes or integration intervals), which made new simulations on the data-flow level (*COSSAP*) necessary to obtain measures of the impact on bit- and packet-error rates and
- replacing implementations of hierarchical blocks (e.g. the frame detector and the BCH codec) with optimized architectures, which were then inserted into the *ComBox* for future reuse. This procedure reduced the number of blocks in the final implementation to 419 while the data-flow graph remained unchanged.

The possibility to perform such optimizations within a single design environment turned out to be crucial for obtaining high-quality results in time.

The design of the complete transceiver (starting from a fully specified data-flow block-diagram) consumed 3 person-months, which includes the exploration of algorithmic trade-offs as mentioned above. The automated VHDL generation, which allowed to reuse all of the information contained in the blockdiagram (the final implementation still contained more than 400 primitive blocks) and freed the designer from error-prone tasks like controller generation, led to a remarkable speedup in the design process. Logic synthesis using a 1μ CMOS library [12] resulted in a circuit with $3,795mm^2$ core area (16.000 equivalent gates).

7. SUMMARY

We presented a tool for generating synchronous timed descriptions of digital receivers from dynamic data-flow system level configurations. It allows to make use of optimized architectures available for a broad range of communication system components. These architectures will be kept in the *ComBox* library which provides means to characterize their data-flow and timing properties. To provide maximum flexibility the approach is based on a strict separation between data-flow and implementation level. ADEN has proved effectiveness during the design of a DMSK transceiver for mobile communication networks.

8. REFERENCES

[1] G. Jennings, "A case against event driven simulation of digital system design," in *The 24th Annual Simulation Symposium* (A. H. Rutan, ed.), (Los Alamitos, California), pp. 170–176, IEEE Computer Society Press, April 1991.

[2] P. Zepter and K. ten Hagen, "Using VHDL with stream driven simulators for digital signal processing applications," in *EURO-VHDL'91 Proceedings*, (Stockholm, Sweden), pp. 196–203, September 8-11 1991.

[3] O. J. Joeressen, G. Schneider, and H. Meyr, "Systematic Design Optimization of a Competitive Soft-Concatenated Decoding System," in *VLSI Signal Processing VI* (L. D. J. Eggermont, P. Dewilde, E. Deprettere, and J. van Meerbergen, eds.), pp. 105–113, IEEE, 1993.

[4] Synopsys, Inc., 700 E. Middlefield Rd., Mountain View, CA 94043, USA, *COSSAP User's Manual: System Architecture.*

[5] P. Scheidt, "The DSP design link with Comdisco," *Synopsys Methodology Notes*, vol. 2, pp. 245–264, February 1992.

[6] Synopsys Inc., 700 E. Middlefield Rd., Mountain View, CA 94043, USA, *Behavioral Compiler User Guide.*

[7] E. A. Lee, "Consistency in dataflow graphs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, pp. 223–235, April 1991.

[8] P. Zepter, "Kopplung eines VHDL Simulators an einen Simulator für Signalverarbeitungsalgorithmen," in *GME Fachberichte 11 Mikroelektronik* (D. Seitzer, ed.), pp. 127–132, VDE Verlag, March 1993. in german.

[9] P. Zepter and T. Grötker, "Generating synchronous timed descriptions of digital receivers from dynamic data flow system level configurations," in *Proc. of European Design And Test Conference*, February 1994.

[10] H. V. Jagadisch and T. Kailath, "Obtaining schedules for digital systems," *IEEE Trans. on Signal Processing*, vol. 39, pp. 2296–2316, Oct. 1991.

[11] U. Lambrette and H. Meyr, "A Digital Feedforward DMSK Receiver for Packet–Based Mobile Radio," in *Proceedings of the IEEE International Conference on Vehicular Technology*, 1994.

[12] ES2 European Silicon Structures, "ES2 Asic Design Guidelines," 1992.