# STAYING AHEAD OF THE GAME IN SILICON FOR DIGITAL MOBILE COMMUNICATIONS

Ravi Subramanian[1], Marc Barberis[1], Herbert Dawid[2], Klaus-Jurgen Koch[2]

[1]Synopsys, Inc.
700 E. Middlefield Road
Mountain View, CA 94043, USA
email: ravis@synopsys.com

[2]Synopsys, Inc.
Kaiserstrasse 100
52134 Herzogenrath, Germany
email: koch@synopsys.com

**Abstract**- While the mobile communication electronics industry's appetite grows for ever more functions and ever higher levels of integration, the complexity of these large designs is creating a discontinuity in the method by which these systems are designed. In this paper, we will take a close look at what is causing the design discontinuity, and how new design technologies are being used to design advanced digital communications systems for portable and wireless communication applications. We will examine how system-level design tools closely tied to silicon design implementation and verification technologies are enabling the creation of digital communications ICs in record time. We take several examples of commercially available silicon solutions designed using these methodologies- a G.721 ADPCM speech codec for cordless telephony and a complete variable-rate digital-video broadcast receiver for the DVB-S broadcast standard. Other examples of lower complexity were presented earlier this year.

## I. INTRODUCTION

While the mobile communication electronics industry's appetite grows for ever more functions and ever higher levels of integration, the complexity of these large designs is creating a discontinuity in the method by which these systems are designed. In this paper, we will take a close look at what is causing the design discontinuity, and how new techniques are being used to design advanced digital communications systems for portable and wireless communication applications. We first look at the phenomenon of silicon inversion and how it is changing design. Then, we briefly review behavioral design before summarizing a proven design methodology for the realization of digital communication Ics [3]. We then look at several examples of silicon on the commercial market that were realized using this methodology.

## II. SILICON INVERSION AND DESIGN METHODOLOGY

While silicon complexity continues to grow in excess of 10x (in area or dynamic performance) every six years, the problem of designing a mobile communications digital transceiver has run into the silicon inversion problem. For two decades, silicon has been used to mimic systems, and integration has been a way to reduce cost or push existing products into new markets. This has been especially evident in the mobile communications markets. While design methodologies evolved from polygons, to gates, to HDL (hardware description languages), the design of the semiconductor solutions going forward is evolving back to designing with "polygon-like" blocks- which are now taking the form of cores (DSP and microcontrollers), algorithm accelerator modules, peripherals, and interface modules.

In this new world, the design of the silicon solutions takes on a completely different flavor. Designing a chip involves two different flows- module design (to create the building blocks to realize a funtion) and module reuse and integration (to define and realize a system). This is shown in Fig. 1. More fundamentally, tools to aid in the design of modules, i.e. module synthesis, will emerge as distinct from tools that enable building a system-on-a-chip, i.e. system synthesis, stitching together the modules.

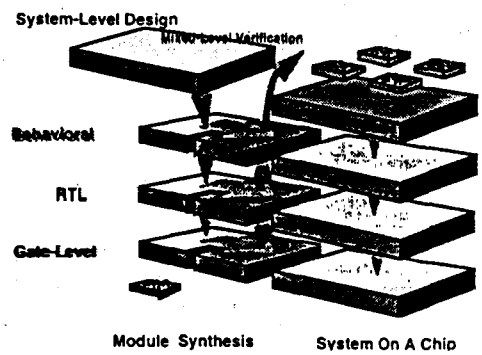


Fig. 1: Designing a Digital Communication System on a Chip

Module design involves system-level design creation and optimization, and module implementation using module synthesis capabilities, exploiting behavioral and RTL synthesis and silicon compiler technologies. System synthesis involves module interface synthesis, module design reuse, and mixed-level verification.

Computer-aided design tools have provided an effective means of designing microelectronic circuits for DSP applications. Today, however, the silicon inversion phenomenon is forcing fundamental change in how system-level design tools work together with module implementation and verification tools.

Most system-level design tools allow designers to create, optimize, and validate algorithms at both the floating and fixed-point levels. The fundamental difference between these tools lies in the modeling paradigm- dataflow vs. time-wheel driven. These tools provide validation of an algorithmic model.

The first task in implementation is to partition the design into a hierarchical structure which represents the physical partitioning

of modules. Each of the algorithmic modules is then translated by hand into schematics or into standard HDLs.
Logic synthesis and silicon datapath compiler tools [4,6] are then used to synthesize the gate-level description of the circuits. If the design does not meet any of the area, timing, or other dynamic performance objectives, the algorithms need to be translated into a new architecture, assuming a different datapath, a different controller, and a different memory, all of which needs to be done manually.

## III. BEHAVIORAL SYNTHESIS

Behavioral Synthesis [4] is the process of refining the algorithmic specification of the behavior of a system to a register-transfer level (RTL) structure that implements that behavior, as shown in Fig. 2. This is in stark contrast to RTL or logic synthesis, where the designer mentally conceives a certain architecture through his RTL coding of the algorithm, and tradeoffs and optimizations are performed manually.

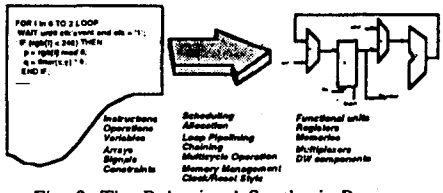

Fig. 2: The Behavioral Synthesis Process

The inputs to the behavioral synthesis procedure are
- •Behavioral description of algorithm(s)
- •Design Constraints
- •High-Level Components Library
- •Technology Library

The behavioral description consists of behavioral HDL code which allows modeling of I/O specifications, operations and dataflow, control flow, and specification of storage elements. Behavioral HDL is an architecture-neutral representation of an algorithm.The design constraints at the behavioral level include specifications on clock period, latency, throughput, and the number and type of hardware resources. The high-level component library describes the basic hardware resources available to build the architecture. It consists of a set of hardware implementations of that component, as well as the protocol for using the components. The technology library is necessary to build an architecture that meets the required constraints, since area and timing information for each operation in the algorithm must be correct.

The output of behavioral synthesis is a constrained RTL architecture that consists of a datapath architecture, a memory architecture, a Finite State Machine, and glue logic. Fig. 3 shows the architecture generated by Behavioral Synthesis.

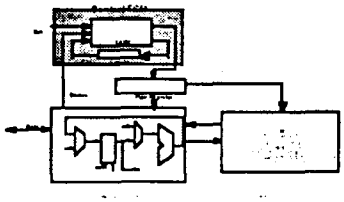

Fig. 3: Architecture Generated By Behavioral Synthesis

## IV. A NEW DESIGN METHODOLOGY

The design flow we proposed in [3] and demonstrate here allows the engineer to rapidly explore the design space of algorithms, architectures, and implementation complexity. Typically, because of the wide range of phenomenon to be captured, there exists no one tool able to handle this design task. Instead, several tools providing the appropriate level of modeling are used.

This new design flow is shown in Fig. 4 [See 3]. This flow is realized in the Synopsys solution via Behavioral and RTL HDL code generation from the COSSAP™ DSP System Design tool, and then using Synopsys' Behavioral Compiler™ and Design Compiler™.

The first task from algorithm to implementation is to partition the design into a hierarchical structure which represents the physical partitioning of processing elements. Once a group of blocks to be synthesized has been defined, there are two paths available- behavioral and RTL synthesis. For those blocks where operations corresponding to high-cost resources (e.g., adders, multipliers) can be shared, the next step is to perform behavioral synthesis in an attempt to select the best architecture that will improve performance or reduce gate count.

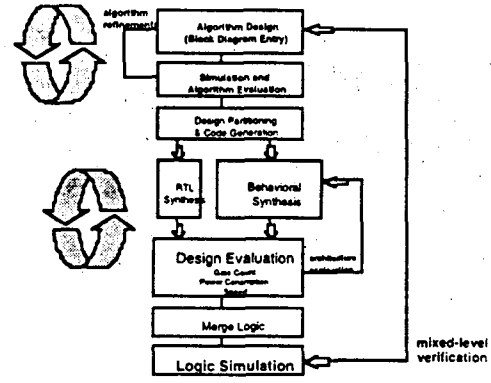


Fig. 4: Proposed DSP System Design Methodology

The designer must, at a minimum, specify the desired clock speed, the latency and the number of clocks per input sample. For faster throughput, Behavioral Compiler™ will implement greater parallelism. It schedules resources to achieve each set of design constraints. Examples of constraints on loop latency, pipelining, and scheduling are shown below:

```
set_cycles  -from_beginning
                    main/reset_loop/main_loop \
            -to_end
                    main/reset_loop/main_loop
            @cycles@

pipeline_loop  main/reset_loop.main_loop
            -i @init@ -l @cycles@

schedule  -area_based  -io_mode  superstate_fixed
```

In certain situations, the designer may want to re-partition the system and try different groupings of functions. This is typically done, for example, in the early stages of defining a chipset. In some designs it makes sense to build a pre-specified architecture at the RTL level if, for example, the operations are very simple (e.g., PN sequence generators, threshold comparators). Once each module has been synthesized, the remaining effort is to merge the pieces together. In the synthesis process, there is the potential

for mismatch due to variations in delay. The final merging process requires that any sample delay mismatch be compensated to insure all the pieces work properly together. The final step is to simulate the synthesized logic. The same stimulus which was used to perform the system level simulations can be used to verify the logic performance. The system level tool thus serves as a system testbench for the complete ASIC.

## V. DESIGN EXAMPLES

### A. *G.721 ADPCM Speech Codec*

The example presented in this section is the commercial implementation of a VLSI ADPCM codec core. The codec has to be G.721-compliant and satisfy the tests a specified by ITU G.726/32 standard input and output test sequences. Such cores are extensively used in system silicon solutions for digital cordless telephony all over the world.

The constraints for the implementation of this design were the extremely tight schedule (under 6 months) and the extremely low power requirement due to its use in a portable consumer product. An area target had beeen specified as well, but this constraint can usually be traded against better power peformance.

The main characteristics of the design are as follows:

- a low ADPCM coding rate of 32 kbits/s
- intensive DSP operations (* and +)
- numerous bit-exact operations
- numerous feed-back paths in the structure

The first two characteristics suggest the use of Behavioral Compiler as there is room for architectural exploration. The numerous feed-back paths call for a systematic and rigorous testing scheme, as e.g. *different initial values in a feed-back path* may lead to inconsistent results.

In a first step, the complete codec is modeled in COSSAP at the C level to serve as a reference at any later stage of the implementation. This is shown in Fig. 5.

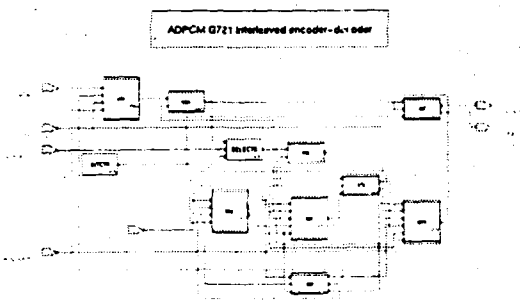

Fig 5: G.721 ADPCM Model In COSSAP

In order to minimize the probability of having errors in the HDL code, two measures were taken: (1) the COSSAP code generator was used to automatically generate a synthesizable VHDL behavioral description for the complete system and (2) the design was partitioned into sub-units which are to be tested individually before generating the VHDL code for the complete design

As a consequence, some sub-systems were defined that were not directly available off-the-shelf from the COSSAP libraries. In these cases, a behavioral VHDL description is written and checked against the reference C code using the COSSAP co-simulation capability with the VHDL System Simulator (VSS) from Synopsys. This means that individual modules can conveniently be checked by taking out a C module or feeding the particular VHDL module with the same data as the original C model and comparing the outputs of the VHDL and the C models. This modular approach proved to yield significant benefits in terms of time-to-market, as the design testing was both drastically simplified and considered very early in the process.

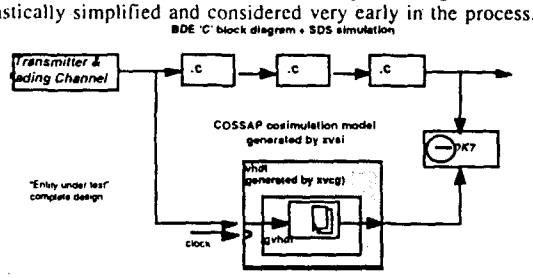


Fig. 6: Module Verification Using Co-Simulation

Once all modules in the COSSAP environment have a valid behavioral VHDL description the COSSAP Code Generator can be used to generate the synthesizable behavioral description for the complete codec. This code is then given straight to Behavioral Compiler for architectural exploration.

The design goal is to achieve a low power design- while minimizing the area. The data rate is relatively low (32 kbit/s) and allows to potentially use a clock up to 60 times faster or more. Thus, the main parameter for architectural exploration with Behavioral Compiler is the number of cycles per sample (NCS), which can be seen as the ratio between the data rate and the clock rate when no pipelining technique is used. The following chart shows the area results for NCS ranging from 2 to 64.

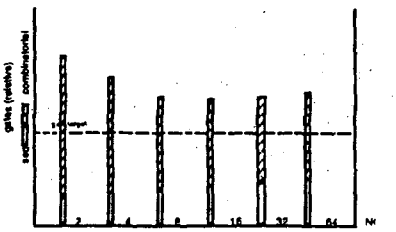

*Fig. 7: Results of Design Space Exploration for ADPCM Core*

It must be noticed that an increase in the NCS beyond 8 does not yield any area gain, as the benefit of any further resource sharing is outweighted by the cost of the sequential logic needed to hold values for more cycles.

As mentioned earlier, the main constraint on this design is to minimize the power consumption. First, it is crucial to be able to get an accurate power estimate, in order to compare it with the power target. Once again, COSSAP is used in combination with VSS and Design Power from Synopsys: the gate-level netlist of one synthesized architecture is simulated using COSSAP/VSS co-simulation. This allows to feed the codec with real-world data, that is data with the right statistics. During the VSS simulation, a toggle file is created, which provides accurate information on the toggling of each single node in the netlist. This toggle file is then passed on to Design Power, which computes an estimate of the power consumption of that particular implementation.

The power estimation results proved that the power targert could be met with an NCS=32. The slight area excess was deemed acceptable as the power consumption turned out to be lower than the target power level. Behavioral Compiler could achieve a very low design power due to its ability to -on request- stall the input of DesignWare components, thus preventing any parasitic toggling within the DesignWare part when it is not being used.

B. *Digital Video Broadcast (DVB-S) Receiver*

As an illustration of the flow presented in the previous sections, we examine the HDM8511, a single-chip all-digital variable rate (5-40Mbps) burst-QPSK receiver with a concatenated code decoder (Viterbi inner-code and Reed-Solomon outer-code). This DVB receiver uses advanced DSP techniques to realize an all-digital demodulator clocked at 60MHz [2].

There were several design constraints in order to achieve the performance and cost targets for the silicon: (a) rapid acquisition time, which requires the use of burst-type synchronization algorithms, (b) all-digital implementation for cost-efficieny in design, manufacture, and operation, as well as further chipset integration in later versions of the product, (c) minimum area, and (d) extremely tight design schedule. In addition, a reliable verification strategy was viewed as essential in order to ensure -as much as possible- *first-time silicon success.*

To achieve these objectives and constraints, the design of the communication system on a chip was broken down into three tasks: system-level module design, module creation, and module integration/verification. COSSAP was first used for modeling and optimization of the complete digital communication system. Modules were defined within the communication subsystem, and Verilog HDL code was automatically generated from COSSAP for these modules. COSSAP was also used for Verilog module validation through mixed-level verification.

Step 1. System-Level Design in COSSAP- The COSSAP tool was used to completely model the DVB encoded video signal, channel coder, modulator, channel, demodulator, and source decoder using the standard communications and fixed-point libraries. A complete model of the DVB transceiver was built and the receiver was described using fixed-point (bittrue) models. At that stage, one of the most important operations for the success of an implementation had been conducted: fixed-point optimization. The flexible iteration optimization capability allowed for area optimization in terms of the size of the operators, while minimizing the implementation loss. This means that a virtual prototype of the ASIC was readily available and its performance in terms of BER and implementation loss could be investigated before the ASIC was synthesized or even available in HDL language. A COSSAP block of the DVB receiver is presented in Figure 8.

Step 2. Implementation Specification- COSSAP Verilog Code Generation was used to create the RTL implementation specification. A key benefit of the process was the ability to automatically generate Verilog code from the COSSAP block diagram, hence ensuring functional correctness right from the start. As the ratio between the clock rate and the data rate is very low, thus leaving little room for resource sharing and architectural exploration, an RTL approach was used.

Step 3. Verification- The co-simulation capability of COSSAP with the Cadence Verilog-XL Verilog simulator was extensively used during the whole design cycle to verify and validate the functionality of the HDL-based receiver. This includes both the receiver front-end for which RTL Verilog code was generated from COSSAP as the channel decoding section (Viterbi decoder) which

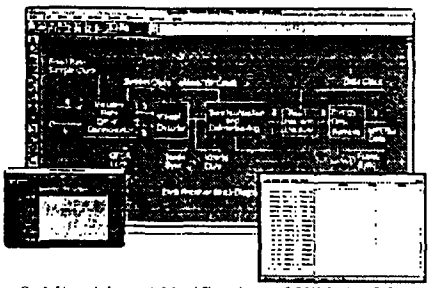

Fig. 8: Mixed-Level Verification of HDL in COSSAP

has been written by hand as a Verilog module. The Verilog code was integrated into the COSSAP environment: as a result, the confidence in the implementation was very high, as was further confirmed by a first-time silicon success for the receiver front-end.

Design Summary-An evaluation board was built with the HDM8511 pre-production silicon in order to perform a last test before production. The results of the pre-production board showed a large correlation with the results from the COSSAP simulation, thus substantiating the claim to be able to characterize a virtual protoype of the chip at the COSSAP level early on in the process-before any VHDL code has been written [2].

*C. Other Examples*

Finally, other examples using this methodology have been presented in [3] and in references contained therein, including FLEX paging receivers, spread-spectrum receivers, front-end filters for DSL modems, and Viterbi detectors for trellis coding.

## VI. SUMMARY

We have demonstrated how modules for digital communication transceiver chips are being designed in record time using new design technologies that allow rapid algorithm optimization, architectural exploration, and module synthesis. These design solutions allow designers to meet stringent cost and power targets through an efficient combination of design technologies that define a solution. We have demonstrated that with a very short turn-around time, the problem of creating modules for digital communications functions can be tackled effectively, repeatably, and systematically. For designers making decisions about digital communication system implementation, this capability enables quick exploration of the hardware architecture space and selection of the type of architecture that leads to an efficient module implementation, as well as a powerful design reuse capability.

## REFERENCES


[1] H. Meyr and R. Subramanian, "Advanced Digital Receiver Principles and Technologies for PCS," IEEE Communications Magazine, January 1995, pp.68-78.

[2] M. Paff, "Design and Implementation of a Variable-Rate Digital Video Broadcast Receiver IC Using COSSAP," Proc. ICSPAT 96, Boston, MA, October 1996.

[3] R. Subramanian et al., "Design and Implementation of All-Digital Receivers for Mobile Communications," Proc. IEEE VTC '96, May 1996.

[4] R. Camposano and W. Wolf, "High-Level VLSI Synthesis," Kluwer Academic Press, Boston, 1991.

[5] G. Fettweis et al., "On the Interaction Between DSP Algorithms and VLSI Architecture," ESA Workshop, ESTEC, The Netherlands, Nov. 910, 1988.

[6] G. de Micheli, "Synthesis and Optimization of Digital Integrated Circuits," McGraw-Hill, New York, 1994.