Chapter 11

TOWARDS HIGH-LEVEL ANALOG AND MIXED-SIGNAL SYNTHESIS FROM VHDL-AMS SPECIFICATIONS

A Case Study for a Sigma-Delta Analog-Digital Converter

Hua Tang[1], Hui Zhang[1], Alex Doboli[1]

[1]*Department of Electrical and Computer Engineering, Stony Brook University, USA*

Abstract This paper presents our experience on high-level synthesis of $\Sigma - \Delta$ analog to digital converters (ADC) from VHDL-AMS descriptions. The proposed VHDL-AMS subset for synthesis is discussed. The subset has the composition semantics, so that specifications offer enough insight into the system structure for automated architecture generation and optimization. A case study for the synthesis of a fourth order $\Sigma - \Delta$ ADC is detailed. Compared to similar work, the method is more flexible in tackling new designs, and more tolerant to layout parasitic.

Keywords: VHDL-AMS, high-level analog synthesis, $\Sigma - \Delta$ analog to digital converters, constraint transformation

1. Introduction

Currently, there is a severe shortage of efficient high-level synthesis tools for analog and mixed-signal systems. As a result, analog and mixed-signal design continues to be a lengthy and error prone activity, which requires cumbersome design experience and expertise. In a recent paper [14], Gielen and Rutenbar offer an in-depth discussion about present methodologies and tools for analog and mixed-signal synthesis. They conclude that existing work mostly targets circuit sizing and

layout generation, which are low level design activities. Circuit sizing [16], [15], [23] assumes a known circuit topology, and finds the transistor dimensions that optimize circuit performance, such as gain, bandwidth, slew-rate, and power. Layout generation [3], [7], [17], [18] performs transistor placement and wire routing, while contemplating wire parasitic, cross-talk, and substrate coupling. In contrast, analog system synthesis uses high-level descriptions for producing alternate system architectures, and identifying constraints for the architectural components, so that the required system performance is met [9], [11].

High-level analog system synthesis includes four main tasks [9], [11]: (1) specification, (2) architecture (system net-list) generation, (3) performance model generation, and (4) constraint transformation. There is general consensus regarding the need of developing description languages for high-level analog synthesis. These languages should permit specification of a large variety of systems (like filters, converters, PLL, oscillators etc), express both analog and digital functionality and constraints, and provide sufficient insight for automated generation of alternative architectures [12]. In fact, the last requirement implies identifying the "minimum" amount of structural information that has to be present in a specification, so that system architectures result through a systematic process of mapping language constructs to implementation structures.

This paper presents our experience on automated synthesis of $\Sigma - \Delta$ analog-to-digital converters (ADC) from VHDL-AMS specifications. ADC circuits are critical for many wireless, multimedia and telecommunication applications. The paper summarizes the main characteristics of a proposed VHDL-AMS subset, and then explains its usage for high-level synthesis. VHDL-AMS [1], [6] is a standardized hardware description language that includes constructs for both analog and digital functionality. Considering that extensive knowledge already exists on using VHDL for behavioral digital synthesis, it is explicable to attempt expanding VHDL-AMS to analog synthesis too. The presented VHDL-AMS subset was already used for high-level synthesis of different kinds of analog systems, like signal conditioning applications [8],[9] and filters [11]. This paper shows that functionality descriptions having the *composition semantics* are useful for high-level analog synthesis. For this semantics, the meaning of a system results by composing the meanings of its building elements. Provided that each language construct can be mapped to circuits, correct system architectures are generated by composing the hardware structures for each instruction of the specification [11], [12].

With analog synthesis, the goal is not merely to implement the desired functionality within a given chip area, but also to keep *minimal* the per-

formance degradation due to layout parasitic [7],[17],[18],[21],[22],[23]. Most of the analog system synthesis methodologies follow a top-down design flow, in which layout generation follows system and circuit sizing [2]. However, it is possible that the fixed circuit parameters do not leave enough performance margins to accommodate the layout-induced performance degradations. In this case, the final design would be incorrect. Typical examples include high-frequency filters that incorporate capacitors of the same order of magnitude (tens/hundreds of fF) as the interconnect parasitic [21, 22]. Parasitic capacitors become an integral part of the signal processing performed by the passive elements. Costly re-iterations through circuit sizing and layout generation are needed to produce a constraint satisfying design. The solution is to combine the parameter sizing process with the layout design step to improve design quality and convergence of the CAD algorithms.

The paper presents a constrained transformation method that includes, besides the traditional parameter exploration step, the tasks of block floorplanning, and wire routing. As opposed to other layout-aware synthesis methods [23], the proposed technique is not limited to one application type, as floorplanning and global routing are integrated within the system synthesis method. This allows early contemplation of layout parasitic. Our constraint transformation technique is different from similar work [13] because it includes the additional steps of parameter classification, parameter domain pruning, and identification of parameter dependencies. These tasks improve the convergence of constraint transformation, provided that large parameter domains must be sampled with fine steps. Also, the method is more flexible because it does not require a working design beforehand.

The proposed constraint transformation algorithm uses *symbolic tiles* [19] for compact representation of the floorplan. Though this approach has been used before [7], [17], [19], our representation is much simpler and more compact, thus increases the efficiency of the design algorithms. A smaller number of tiles offers the advantage of faster methods for tile swapping and tile moving. Another difference is that our tiles are *soft* (their sizes and aspect ratio can change). This is a consequence of parameter optimization being part of the synthesis loop. For keeping the complexity of tile managing methods low, complete knowledge about left, right, top and bottom neighbors of each tile must be *explicitly* available. We decided to store the neighborhood information as distinct O trees [20] for each of the four directions. Other representations, such as sequence-pairs [3] or B* trees [5], are not efficient for our problem as they do not implicitly offer the neighborhood information.

Section 2 discusses the proposed VHDL-AMS subset for synthesis. Section 3 presents the layout-aware synthesis algorithm. Section 4 shows synthesis results for a $\Sigma - \Delta$ ADC. Section 5 offers conclusions.

2. VHDL-AMS Subset for Synthesis

VHDL-AMS [1, 6] is a hardware description language (HDL) for simulation of mixed-domain systems. Currently, it is one of the few standardized mixed-domain HDL.

VHDL-AMS specifications are sets of differential and algebraic equations (DAE) involving continuous-time electrical quantities, like currents and voltages. The system behavior is obtained by numerically solving the DAE sets at time points decided by the simulation cycle of the language [6]. VHDL-AMS specifications do not have the composition semantics, thus they give limited insight into the system structure [12]. Unrestricted VHDL-AMS programs offer poor support for producing architectures. For example, a continuous-time filter specification might include DAE for poles and zeros, phase margin, and quality factor. However, these performance descriptions do not help automatically creating filter architectures through a systematic process of mapping language constructs to circuits [11], [12]. This section presents restrictions that enforce the composition semantics on VHDL-AMS programs.

Signal-flow graphs (SFG) formulate the system behavior as a composition of signal processing operators and signal flow paths. SFG operators are simple linear functions, like addition, subtraction, integration, and multiplication by a constant. Operators generate their outputs depending only on their inputs. There are no influences between connected operators. In our experience [8], [9], [11], [12], SFG provide sufficient insight for automatically producing alternative architectures for a system. To impose the composition semantics on VHDL-AMS programs, we had to identify language restrictions, so that VHDL-AMS constructs can be represented as SFG structures.

The composition semantics requires that connected blocks do not influence each other, so that block outputs depend only on the block inputs. This assumption is widely used for high-level expression of systems, like ADC, PLL, transmitter and receiver systems, to name a few. Enforcing the composition semantics on circuit and transistor level designs is unrealistic. Sections 3 and 4 explain that the analog synthesis flow considers circuit loadings (input/output impedances) and non-idealities (like finite gain and poles) for evaluating the functionality and performance of an implementation. An implicit goal of synthesis is to minimize

the mismatch between the behavior of the implementation and the behavior of the ideal system having the composition semantics.

In [12], we presented a detailed justification of the VHDL-AMS subset for synthesis. VHDL-AMS programs consist of entity declarations, architecture bodies, package declarations, and package bodies. The proposed VHDL-AMS subset for synthesis defines analog signals as free quantities. The subset includes three language constructs: simple simultaneous statements, simultaneous if/case statements, and procedural statements. Following restrictions define the composition semantics for the constructs.

Simple simultaneous statements (SSS) express DAE sets. DAE have a broader meaning than SFG, in the general case. We imposed following three restrictions on SSS, so that their semantics can be described as a unique SFG: (1) The left side of an SSS contains only one quantity, or one derivative of a quantity. (2) In a DAE set, a quantity occurs only once in the left side of an SSS. Under the two constraints, the behavior of the left-side quantity is equivalent to the behavior of the output of the SFG structure for the right side of the SSS. (3) The right side of an SSS includes only linear operators such as addition, subtraction, integration, and multiplication by a constant. This restriction is explained by the need to realize these operators using the circuits currently present in our circuit library [8]. The restriction can be loosened, if more circuits are added to the library (for example, mixer circuits).

Simultaneous if/case statements (SIS) represent systems with multiple modes of behavior. Digital signals control the activation of the modes. In the context of linear systems, SIS describe variable amplification stages. For example, in [8], the receiver module of the telephone set had a variable gain controlled by a digital signal. We imposed two restrictions for the statements in the SIS branches: (1) All left-side quantities of the SSS in a branch must appear in the other branch, too. Otherwise, the quantity will not have well-defined values for all conditions. Such functionality is incorrect for continuous-time systems. (2) For a quantity present in the left side of two SSS (one SSS for each branch), the operator patterns of the two SSS must be the same. Same operator patterns for two SSS means that their right-side expressions refer to the same quantities, and apply the same operators to the quantities. Different constant values are allowed for the same patterns.

Procedural statements (PS) are useful for explicitly specifying SFG structures. PS include instructions like assignments, if/case statements and loop statements. Data dependencies among PS instructions define the signal flow between the SFG blocks for the instructions. As opposed to HDL for digital circuits (like VHDL and Verilog), instruction

sequences do not have a meaning for analog synthesis. The left side of an assignment statement defines the output signal for the SFG corresponding to the right side of the assignment. Assignment statements have the same constraints as SSS, if/case statements must meet the same restrictions as SIS. We discussed in [8] the conditions under which for and loop statements can be translated into semantically equivalent SFG structures. Loop instructions are useful for describing multi-channel systems, like the dual tone multiple frequency decoder (DTMF). DTMF includes a bank of eight filters and amplitude detectors to recognize the presence of a certain tone. The abstract specification of the DTMF uses one for instruction that instantiates the generic filter architecture as part of the loop body. Loop instructions are also useful to express structural regularities of a system. Structural regularity helps setting up compact performance models [10] and simplifying architecture generation [11].

3. High-Level Analog Synthesis

Figure 11.1 presents the proposed high-level analog synthesis flow. Rhapsody is the corresponding software environment. Inputs to the flow are VHDL-AMS descriptions of a system, AC and transient domain performance requirements, and physical constraints like area and power consumption. The synthesis output is the sized system architecture (including sized resistors and capacitors, and bounds for op amps/GmC gain, poles, and I/O impedances), the placement of the building blocks, and the global routing of signal and power wires. Given the bounds for the active circuits, op amp/GmC transistors can be sized using state-of-the-art circuit synthesis tools. Following steps form the synthesis flow.

Architecture generation: Architectures are netlists of active circuits, i.e. op amps and GmC, and passive elements, such as resistors and capacitors. Two architecture generation algorithms are currently available. The first approach [8, 9] uses a static library of patterns that relate VHDL-AMS language instructions to analog circuits. A pattern matching algorithm produces alternative architectures by composing together the mapped instructions. The second technique [11] is based on the fact that linear signal processing operators can employ either currents or voltages. This observation makes architecture generation similar to a bi-partitioning problem [11]. A set of conversion rules link processing operators and signal types to corresponding active circuits and devices.

Generation of behavioral performance models: Symbolic models express the system outputs as functions of inputs and design parameters [10]. The used method generates very compact symbolic expressions by using the structural regularities of a system architecture. The derivatives

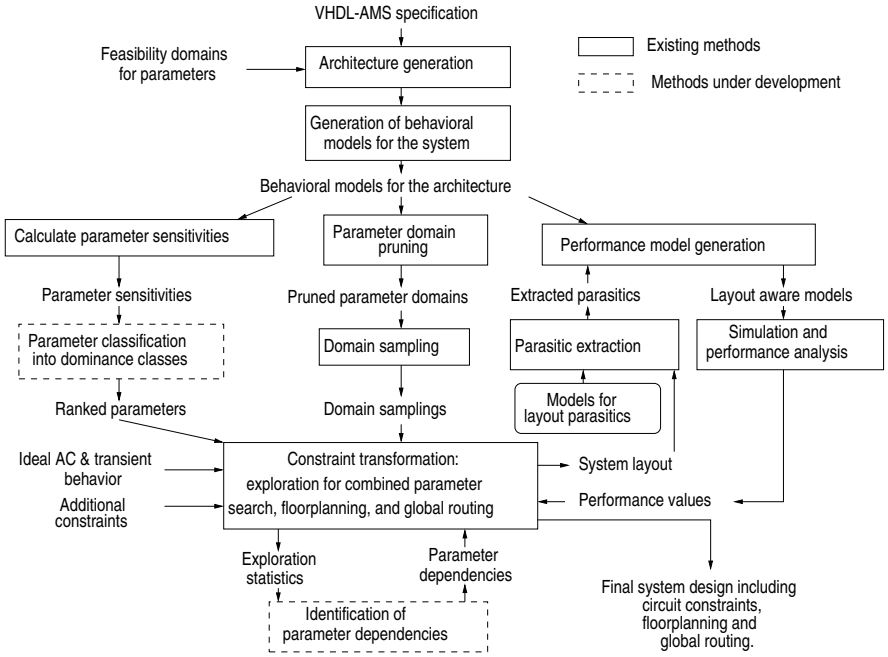

Figure 11.1. Analog synthesis methodology

of state-variables were replaced by their finite differences using integration rules, such as Backward Euler integration. The main reason for behavioral modeling is the long time necessary for SPICE simulations. Section 4 presents several ADC simulations using behavioral models.

Parameter classification: Parameters are classified depending on their effect on the system performance. Then, during exploration, dominant parameters are sampled more often. This improves the quality of synthesis because more samples are analyzed for the important parameters. For example, the parameters of the first integrator of a second-order $\Sigma - \Delta$ ADC have a higher influence on SNR and DR than those of the second integrator [4].

Parameter domain pruning: This step eliminates space regions that are unlikely to result in good solutions. In our experience, the convergence of synthesis is poor because design parameters belong to large domains. These large domains must be sampled with fine steps. The synthesis convergence is significantly better, if less promising domains are eliminated. We use interval calculus for domain pruning [21].

Identification of parameter dependencies: This step automatically finds dependencies among parameters. For example, the *gm* and capacitor values of a filter are related to the system performance attributes, like the 3dB point, quality factor, and resonance angular frequency. The solution

space will include many infeasible points, if parameter dependencies are not considered. This obviously affects the synthesis convergence.

Parasitic extraction, simulation and analysis: For each solution visited during exploration, the behavioral models are merged with the extracted layout parasitic to create layout-aware performance models. Models are simulated in the AC and transient domains. After analysis, the obtained attributes guide exploration (as part of the cost function).

Constraint transformation through combined parameter exploration, block floorplanning and global routing: Constraint transformation includes (1) finding the block parameters (i.e. circuit gains, poles, input and output impedances), (2) block floorplanning (including finding the aspect ratios and placements of blocks), and (3) wire routing. The combined system parameter exploration and layout generation allows accurate evaluation of layout parasitic. This results in layout awareness of the synthesis methodology [21, 22]. Simulated annealing or tabu search algorithms can be used for the combined exploration step.

The remaining part of this section details the data structures and methods used for system floorplanning.

3.1 Tile Representation

A tile [19] based representation is adopted for our layout. Figure 11.2 depicts the tile representation. It represents both active blocks and channels. The active part of the tile is the actual component, and the channel part is the portion of the channels surrounding the active region. The widths of the channel part are denoted by Δ_i, i=1,4. A layout is a collection of tiles. As compared to the tile definition in [19], the used definition reduces the number of tiles for a layout as it decreases the number of tiles needed to express empty spaces.

As shown in Figure 11.3, tiles can be of three types: (1) active tiles, (2) empty tiles, and (3) margin tiles. The active tile is a tile, which represents an electrical component, such as a resistor, a capacitor, an op-amp etc. A tile, which does not represent an electrical component, is called an empty tile. The tiles at the four borders of the layout are called margin tiles.

A tile is defined by its four *corner-points*. Figure 11.2 shows corner-points as gray bubbles. A corner-point is the pair $(x, y) \in R^2, 0 < x < w_{max}, 0 < y < h_{max}$. T is the set of all tiles. A corner-point belongs to at least one tile. The set of all corner-points is denoted by CP. A *joint* of a tile is a corner-point, which meets an adjoining tile at a point other than its corner-point. A joint is the pair $(x, y) \in CP, (x, y) \subseteq T$ & $\exists t \in T$ for which (x,y) is not a member but one of its corner-points has its x- or y

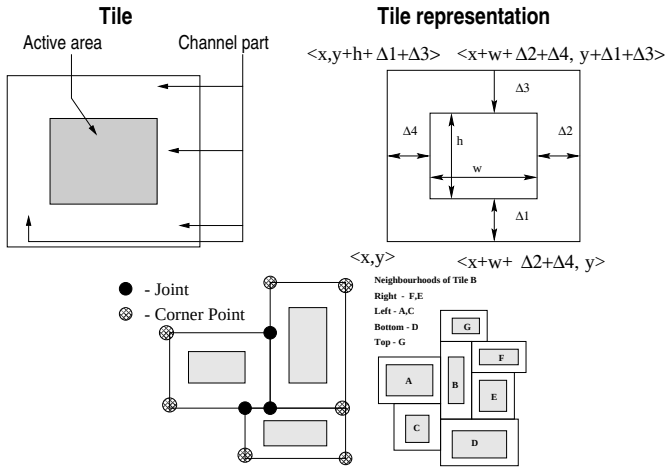

Figure 11.2. Tile definition and tile relations

coordinate equal to x or y. The left part of Figure 11.2 illustrates joints as black bubbles.

The relative positions of the neighboring tiles is defined by neighborhood relationships. Four neighborhood relations exist for each tile, as shown in Figure 11.2. The existence of a left neighborhood relation is identified by the constraints (1) $|y_1, y_1 + h_1 + \Delta_1^1 + \Delta_3^1|$ intersects $|y_2, y_2 + h_2 + \Delta_1^2 + \Delta_3^2|$ and (2) $x_1 + w_1 + \Delta_2^1 + \Delta^1 + 4 = x$. Δ_j^i corresponds to the channel j of tile i. Similarly relationships are defined for the other neighborhoods. The width of the Δ-s is determined during the routing phase by the number of nets which are to be routed through the empty spaces of the channel parts. These relationships are stored as distinct O trees [20] for each of the directions. This helps immediate retrieval of the tile neighbors.

Tile based moves

Domino Move: In the domino effect, to accommodate an expansion in a tile in the layout, its neighbors will have to be moved to avoid overlapping. This movement of tiles, called domino effect, moves towards the edges of the layout stopping either when the move is completely absorbed by the empty tiles, or at the border. The domino move function, which implements this phenomenon, is an important function in floorplanning as all other routines will rely on this function.

Figure 11.3 shows an example of a domino move. Tile 7 is to be expanded towards the left. When this tile expands, it overlaps with tiles 4 and 5 which are on its left, as shown in Figure 11.3(c). To avoid the

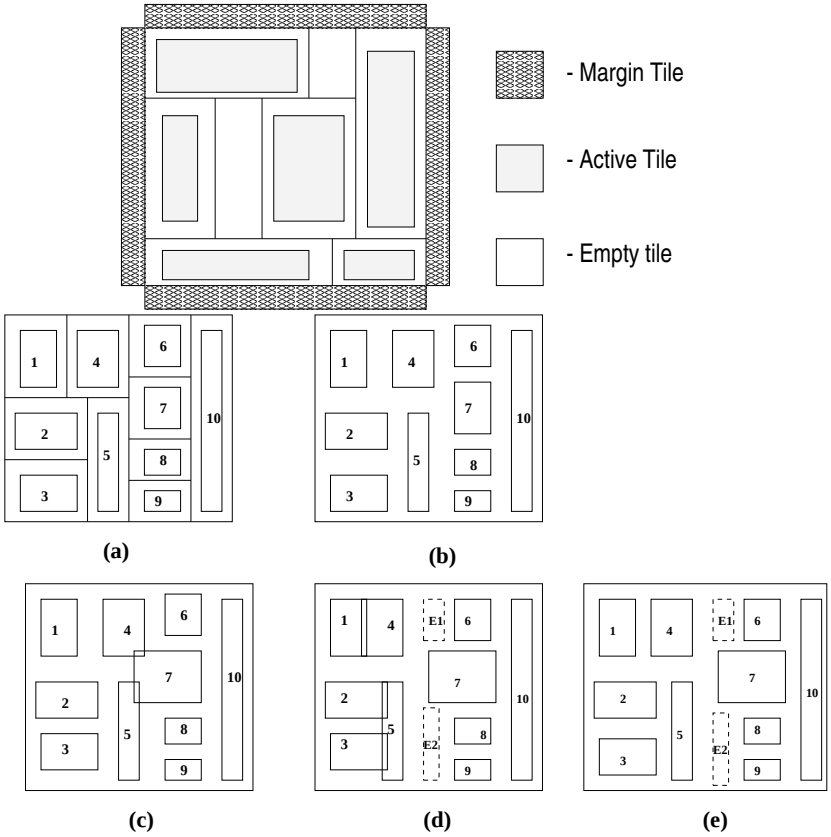

Figure 11.3. Layout and domino move

situation of overlap in the layout, the tiles 4 and 5 are moved to the left by an amount equal to that by which the tile 7 was expanded. These tiles overlap with tiles 1, 2 and 3 (see Figure 11.3(d)). These tiles are moved to the left to avoid overlaps. As the tiles are on the edge of the layout, the chip size is increased so that the tiles are not in overlap with the margin tiles (Figure 11.3(e)). The domino move has a worst case complexity linear with the total number of tiles. Hence, having a reduced number of tiles in the representation helps preserving a low execution time for the domino move step.

Swap_tile function: Given two tiles as an input, this function swaps the two tiles. The dimensions of the tiles are changed to those of each other (the domino move function will be called here). The neighbor lists of the tiles are then interchanged. The coordinates of the tiles are now changed to the coordinates of each other. The swapped tiles are then

resized to their original dimensions. A similar method is used for the **move_tile** step.

Resize_tile Function: This function moves a tile in the desired direction with the specified amount to be moved. This involves a change in the coordinates of the tile. This function is important for searching the circuit parameters of a design, such as the values for resistors and capacitors of a filter.

Compaction: As the tiles are moved around and their dimensions varied, there is a danger that the size of the chip may exceed the specified dimensions. To avoid this, compaction of the circuit is done routinely during the iterative design process. The compaction module involves deleting as many unnecessary empty tiles as possible. This procedure would ensure that the circuit dimensions are within limits.

4. Case Study

The case study presents a synthesis experiment for a fourth order $\Sigma - \Delta$ ADC [4]. The goal was to maximize the signal to noise ratio (SNR) and dynamic range (DR) of the ADC.

Figure 11.4 presents the VHDL-AMS specification of the ADC modulator. The architecture body describes the ADC signal flow. It instantiates four times the same stage, as shown by instructions *i1-i4* in the code. This similarity in the specification helped performance model generation, because the symbolic code for a stage was reused four times. Each ADC stage has the same signal processing, but involves different symbolic constants $g1$, $g1\_prime$, $g2$, $g2\_prime$, $g3$, $g3\_prime$, $g4$, and $g4\_prime$. The numeric values for the symbolic constants were found during constraint transformation. The SFG specification of a stage (starting from statement *"ARCHITECTURE sfg of stage IS"*) includes SSS. The quantizer block is described as a process statement sensitive to $ABOVE$ events on the continuous quantity $s5$.

A rich set of attributes states designer knowledge based constraints. These constraints helped pruning some infeasible solution regions during constraint transformation. According to [4], to keep the SNR loss negligible, the ADC stage bandwidth must be higher than the sampling frequency f_s, for this example $f_s = 160$kHz. The bandwidth of a signal is defined using tool-specific annotations. Another constraint states that the input signal should be within the range $[-0.45 \times delta, 0.45 \times delta]$, where $delta$ = VDD-GND. This condition prevents modulator overloading [4]. In our example, the range domain of [-0.2V, 0.2V] for quantity $s5$ corresponds to this need. The next constraint requires that the gain of a stage is smaller than the gain of the next stage, as the distortions in-

```
-- ATTRIBUTE bandwidth: real;
-- bandwidth = FREQUENCY.((QUANT'voltage -
-- QUANT'voltage(DC) < 3dB) at 1);
ENTITY stage IS
  GENERIC (g1, g2: real;)
  PORT (
    QUANTITY vin, vo: IN real;
    QUANTITY o: OUT real;)
END ENTITY;
ARCHITECTURE sfg OF stage IS
QUANTITY s: real;
BEGIN
  s == g1 * vin - g2 * vo;
  o == s'integ;
  ASSERT (s'bandwidth > 160kHz)
    REPORT "CONSTRAINT" SEVERITY WARNING;
END ARCHITECTURE;
ENTITY adc IS
  TYPE lim_out IS range GND TO VDD;
  GENERIC (g1,g2,g1_prime,g2_prime,
           g3,g4,g3_prime,g4_prime:real;)
   PORT (
     QUANTITY vin: IN real;
      -- IS voltage
     QUANTITY vout: OUT lim_out;
      -- IS voltage)
END ENTITY;
ARCHITECTURE sfg OF adc IS
COMPONENT stage IS
  GENERIC (g1, g2: real;)
  PORT (
    QUANTITY vin, vo: IN real;
    QUANTITY o: OUT lim_out;)
END COMPONENT;
QUANTITY s3: real range -0.2V TO 0.2V;
QUANTITY s4, s5, s6: real;
SIGNAL c: bit;
VARIABLE delta: real := VDD - GND;
BEGIN
PROCESS (s5'ABOVE(VDD/2)) IS
BEGIN
  IF (s5'ABOVE(VDD/2) = TRUE) THEN
    c <= '1';
    ELSE
      c <= '0';
    END IF;
  END PROCESS;
   i1: stage GENERIC MAP(g1 => g1, g2 => g1_prime)
       PORT MAP (vin => vin, vo => vout, o => s3);
  i2: stage GENERIC MAP(g1 => g2, g2 => g2_prime)
       PORT MAP (vin => s3, vo => vout, o => s4);
   i3: stage GENERIC MAP(g1 => g3, g2 => g3_prime)
       PORT MAP (vin => s4, vo => vout, o => s6);
   i4: stage GENERIC MAP(g1 => g4, g2 => g4_prime)
       PORT MAP (vin => s6, vo => vout, o => s5);
  IF (c == '1') USE
    vout == VDD;
  ELSE
    vout == GND;
  END USE;
  ASSERT (g1 < g2 AND g2 < g3 AND g3 < g4)
    REPORT "CONSTRAINT" SEVERITY WARNING;
  ASSERT (s5'deriv > 176e3 * delta)
    REPORT "CONSTRAINT" SEVERITY WARNING;
  ASSERT (s3'deriv > 176e3 * delta)
    REPORT "CONSTRAINT" SEVERITY WARNING;
  ASSERT (s4'deriv > 176e3 * delta)
    REPORT "CONSTRAINT" SEVERITY WARNING;
  ASSERT (s6'deriv > 176e3 * delta)
    REPORT "CONSTRAINT" SEVERITY WARNING;
  ASSERT (vin'min>-0.45*delta AND vin'min<0.45*delta)
    REPORT "CONSTRAINT" SEVERITY WARNING;
  ASSERT (vin'max>-0.45*delta AND vin'max<0.45*delta)
    REPORT "CONSTRAINT" SEVERITY WARNING;
END ARCHITECTURE;
CONFIGURATION sd_adc OF adc IS
FOR sfg
  FOR i1: stage USE CONFIGURATION
    WORK.stage;
  END FOR;
  FOR i2: stage USE CONFIGURATION
    WORK.stage;
  END FOR;
END FOR;
END sd_adc;
```

Figure 11.4. VHDL-AMS specification for a fourth-order $\Sigma - \Delta$ ADC

troduced by early stages are more important than those of latter stages. The next four constraints (the assert statements involving quantities $s3'deriv$, $s4'deriv$, $s6'deriv$, and $s5'deriv$) refer to the slew-rate of the integrators, $SR > 1.1\ \frac{delta}{T_s}$ [4]. Finally, the limited output swing of the first integrator imposes a range constraint for signal $s3$.

Note that the specification does not include any circuit-level details, such as opamp finite gain, finite bandwidth, output swings, and mismatches. These constraints are important for the implementation of the modulator [4]. However, they represent physical details, and thus, should not be present in the high-level specification.

Using the algorithm discussed in [9], four different architectures were contemplated for each ADC stage. Figure 11.5(a) shows the selected implementation. The system-level performance evaluation module identified a sequence of four series configurations for the stages. Symbolic models for series configurations were selected from the library to re-

late the input voltages and currents to the currents and voltages at the quantizer input [10]. The time-domain model for $Vout(t) = f(Vin)$ was completed by using the symbolic expressions that relate the voltages and currents at the ports of one stage. Based on the small signal models for transconductor and opamp circuits, each stage was modeled as a six terminal block, as shown in Figure 11.5(b). Time-domain symbolic relationships between Iin, Vin, $Iout$, $Vout$, $I1$, $V1$, $I2$, $V2$ were calculated by solving Kirchhoff's current and voltage laws, and replacing the derivatives of the voltage drops on the capacitors with finite differences (according to Backward Euler Integration rule).

Constraint transformation and component synthesis explored about 30,000 solution points in three days. For a sin wave signal of 625kHz at the input of the $\Sigma - \Delta$ modulator, the maximum DR and the output spectrum are plotted at the bottom of Figure 11.5. The maximum SNR is 64db, and DR is about 70db. The design quality is similar to that of the modulator reported in [13]. However, the proposed method is more flexible than that in [13], because it does not assume a working design beforehand. Also, we were successful in synthesizing a higher-order converter, which is more challenging to design. The figure also shows the importance of using detailed circuit models for synthesis, such as models that include poles and zeros rather than ideal models. The two plots with dotted lines correspond to simulations, which used circuit macromodels with one or two poles. In the first case, the circuit still worked as an ADC, but the SNR went down by about 13dB and the DR by about 12dB respectively due to the poles. In the last case, the poles prevented the ADC from a correct functioning.

5. Conclusion

This paper presents our experience on high-level synthesis of $\Sigma - \Delta$ ADC from VHDL-AMS descriptions. We showed that functional descriptions having the composition semantics offer sufficient insight into the system structure for automatically creating alternative architectures. Then, different mappings of VHDL-AMS descriptions to library circuits can be contemplated during synthesis, as the best architecture depends on the targeted performance. The paper also discusses the high-level analog synthesis methodology, and details the layout-aware constraint transformation step. Constraint transformation executes combined parameter exploration, block floorplanning, and global wire routing. The technique was successfully applied to high-level synthesis of a fourth-order continuous-time $\Sigma - \Delta$ ADC. Compared to similar work, the

methodology is more flexible (due to its capability of accepting VHDL-AMS specifications), and more tolerant in tackling layout parasitic.

Acknowledgement

This work was supported by Defense Advanced Research Projects Agency (DARPA) and managed by the Sensor Directorate of the Air Force Research Laboratory USAF, Wright-Patterson AFB, OH 45433-6543.

References

[1] "IEEE Standard VHDL Language Reference Manual", IEEE Std.1076.1

[2] H. Chang *et al*, "Top-Down Constraint Driven Methodology for Analog Integrated Circuits", *Kluwer*, 1997.

[3] F. Balasa *et al*, "Module Placement for Analog Layout Using the Sequence-Pair Representation", *Proc. of Design Automation Conference*, pp. 274-279, 1999.

[4] J. Cherry, W. M. Snelgrove, "Continuous-Time Delta-Sigma Modulators for High-Speed A/D Conversion", *Kluwer*, 2000.

[5] Y. Chang *et al*, "B* Trees: - A New Representation for Non-Slicing Floorplans", *Proc. of Design Automation Conference*, 2000.

[6] E. Christen, K. Bakalar, "VHDL-AMS - A Hardware Description Language for Analog and Mixed-Signal Applications", *IEEE Trans. Circuits & Systems - II*, Vol. 46, No. 10, 1999.

[7] J. Cohn *et al*, "Analog Device-Level Layout Automation", *Kluwer*, 1994.

[8] A. Doboli, R. Vemuri, "A VHDL-AMS Compiler and Architecture Generator for Behavioral Synthesis of Analog Systems", *Proc. of DATE*, 1999, pp. 338-345.

[9] A.Doboli, *et al*, "Behavioral Synthesis of Analog Systems using Two-Layered Design Space Exploration", *Proc. Design Automation Conference*, 1999.

[10] A. Doboli, R. Vemuri, "A Regularity Based Hierarchical Symbolic Analysis Method for Large scale Analog Networks", *IEEE Trans. Circuits & Systems - II*, Vol. 48, No. 11, 2001.

[11] A. Doboli, R. Vemuri, "Exploration-Based High-Level Synthesis of Linear Analog Systems Operating at Low/Medium Frequencies", *IEEE Trans. CADICS*, Vol. 22, No. 11, 2003.

[12] A. Doboli, R. Vemuri, "Behavioral Modeling for High-Level Synthesis of Analog and Mixed-Signal Systems from VHDL-AMS", *IEEE Transactions on CADICS*, Vol. 22, No. 11, 2003.

[13] K. Franken *et al*, "DAISY: A Simulation-Based High-level Synthesis Tool for Sigma-delta Modulators", *Proc. ICCAD*, 2002.

[14] G. Gielen, R. Rutenbar, "Computer Aided Design of Analog and Mixed-signal Integrated Circuits", *Proc. of IEEE*, Vol. 88, No 12, Dec 2000, pp. 1825-1852.

[15] M. Hershenson, S. Boyd and T. Lee, "Optimal design of a CMOS op-amp via Geometric Programming", *IEEE Trans. CADICS*, Vol.20, NO.1, Jan 2001, pp. 1-21.

[16] M. Krasnicki et al, "MAELSTROM: Efficient Simulation-Based Synthesis for Custom Analog Cells", *Proc. Design Automation Conference*, 1999, pp. 945-950.

[17] K. Lampaert, G. Gielen, W. Sansen, "Analog Layout Generation for Performance and Manufacturability", *Kluwer*, 1999.

[18] E. Malavasi *et al*, "Automation of IC Layout with Analog Constraints", *IEEE Transactions CAD*, Vol. CAD-15, no. 8, pp. 923-942, August 1996.

[19] J. Osterhout, "Corner Stitching: A Data-Structuring Technique for VLSI Layout Tools", *IEEE Trans. on CAD*, Vol. CAD-3, No. 1, pp. 87-100, January 1984.

[20] Y.Pang *et al*, "Block Placement with Symmetry Constraints based on the O-tree Non Slicing Representation", *Proc. of Design Automation Conference*, pp. 464-467, 2000.

[21] H. Tang, H. Zhang, A. Doboli, "Layout-Aware Analog System Synthesis Based on Symbolic Layout Description and Combined Block Parameter Exploration, Placement and Global Routing", *Annual Symposium on VLSI (ISVLSI)*, 2003.

[22] H. Tang, A. Doboli, "Employing Layout Templates for Synthesis of Analog Systems", *Midwest Symposium on Circuits and Systems*, 2002.

[23] P. Vancorenland *et al*, "A Layout-Aware Synthesis Methodology for RF Circuits", *Proc. of ICCAD*, 2001, pp. 358-362.

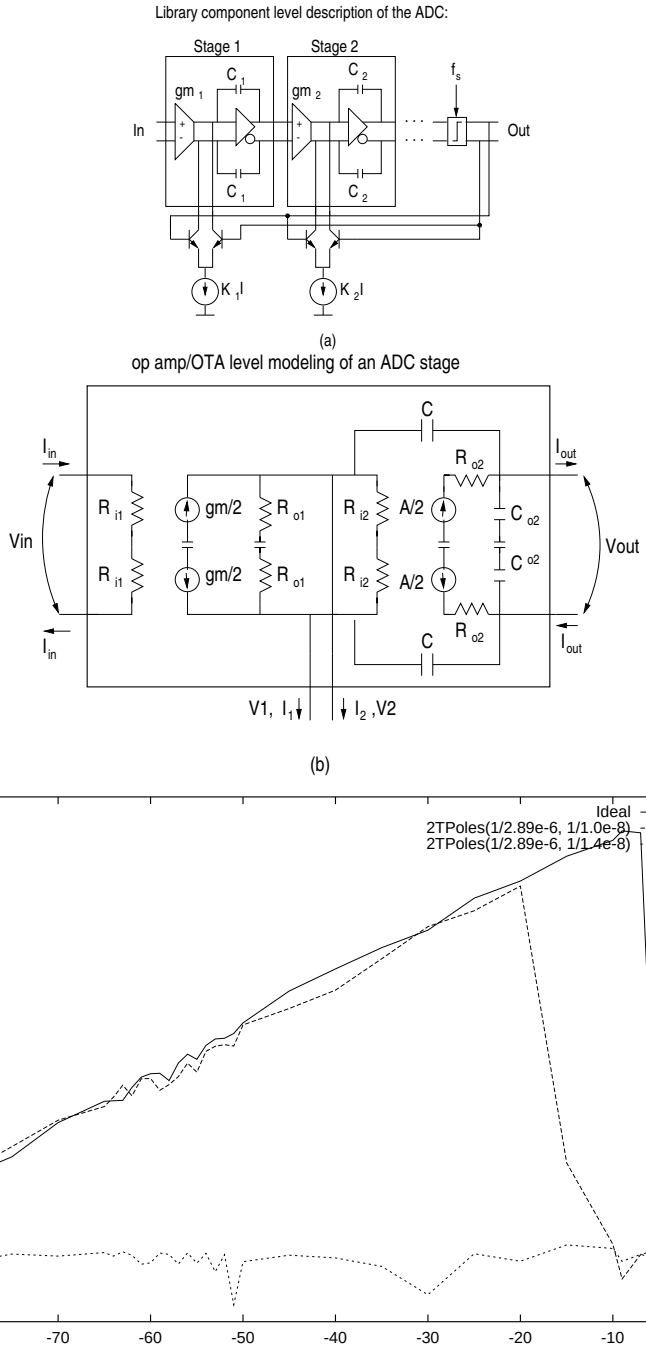

Figure 11.5. ADC architecture, stage model, and SNR and DR plots