Session: ANALOG & ELECTRICAL DESIGN

Guidelines for Designing Smart and Reusable Analog IP Cores

P. Nguyen Tuong, M.-M. Louerat, A. Greiner
Pierre et Marie Curie University, Paris, France

Abstract

Every analog designer has to face with the legacy of the traditional analog development, an iterative conception process where a single computation loop can last a week. The tremendous gap between the analog and the digital world's methodology of conception is sharper than ever. The reasons should be found both in the lack of automated tools and in the methodology itself. If a few commercial and academic softwares offer development environments, these environments do not easily allow creating analog generators because of lack of clear methodology. This paper presents a guideline for designing smart and reusable analog generators as an answer to the problem of analog IPs.

Introduction

Motivations

For the past decade, the most challenging activity of the micro-electronics design was to deal with the high speed endless technology evolution. As the digital design has developed dedicated synthesis algorithms and popular automated tools, analog design has to develop its own methodologies to deal with the complexity of electrical systems, intellectual property reuse and technology migration which is a time consuming common task. Thus, we need a methodology to develop component which will be instantiated without running through a complete development cycle from scratch.

Commercial and academic realizations

To face with the lack of efficient analog electrical synthesis algorithms and to promote IP reuse, one can develop precharacterized cells (hard cells) or generators (dynamic generation). Both of these objects rely on the designer's expertise. A few commercial and academic softwares can handle analog design, yet none of them is providing a tool allowing the designer to create generators of analog functions. We can cite the Barcelona Design tool ([3]), the Koan/Anagram tool ([4]), the Neolinear/Neocell tool for Cadence ([1]) and a lot of other approaches described in [1]

Goals: designing analog generators

The challenge is to capture the designer's knowledge, in order to perform design space exploration in a way ensuring reuse and portability (regarding electrical specification and process migration, figure 1). Following the conventional design flow (Choice

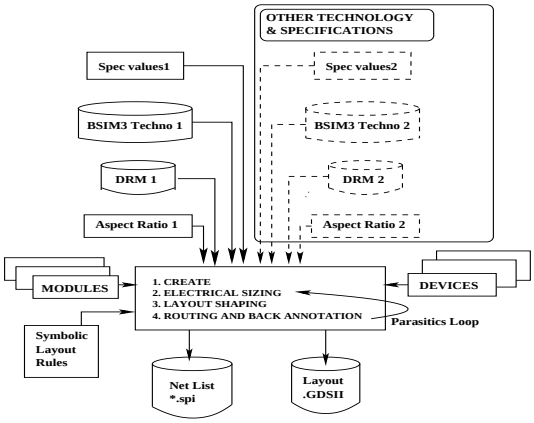

Figure 1: CAIRO+ Generator used for process and specification migration

of the topology, electrical sizing, electrical simulation, layout generation, parasitics extraction) we show where the critical stages are located, more precisely where the designer's knowledge is involved

and should be captured. Then, to capture this knowledge, we propose a generator definition based on several templates: a netlist template, a layout template and a specification template. Upon these informations, we have developed a dedicated language made of C++ macro-functions to ease (guide) the description as well as an original design methodology called CAIRO+. The main idea is the design space exploration achieved by the capability of the generators to be questioned about a parameter regarding a specification and to give back an answer.

Analog design flow

The critical stages of the circuit design

In the analog design flow, when the system architecture is chosen, the next critical stage is the electrical sizing. The goal is to satisfy the set of circuit specification of the resulting layout. Seeing that the parasitics have a dramatic influence on the circuit performances, they have to be included in a circuit sizing loop, allowing the designer to correct their effects (figure 2). The analog designer is in charge of the

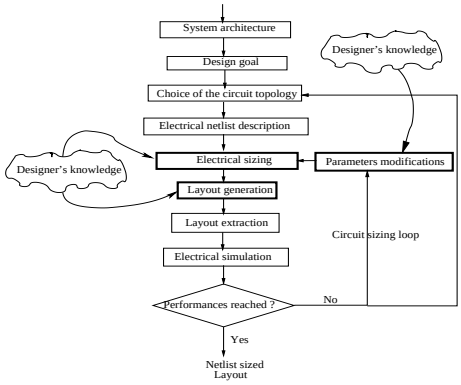

Figure 2: Critical stages of the circuit design

description of the circuit, as well as the electrical sizing. The main engine of the electrical sizing is the designer's experience which drives the entirely design.
The other critical stage of the circuit design lies in the layout design. The performance of the resulting circuit is dependent on the quality of the actual layout ([4]). A lot of previous studies exist on automating the layout, but few studies have tackled the problem of codesign of electrical and physical views [5] [6].

The common way to design circuits relies on slow simulations, and moreover on the knowledge of the analog designer. In this approach, the designer is not required to derive synthesis equations of the circuit. It is a simulation based approach, which can not ensure the reusability of the designed circuit because the electrical sizing and the simulations must be performed each time a new target process is chosen.

Knowledge capture and reuse

In the previous section, the common analog flow has been discussed showing that the knowledge of the designer involved in both the critical stages (electrical and layout) is neither captured nor stored. Yet it seems possible to store a large amount of the efforts made by the designer to have a chance to get benefits of previous work. Capturing the designer's knowledge may imply an equation based approach. The investment is certainly large, but it is an efficient solution to the problem of analog reuse of commonly used blocks. We propose a knowledge based approach relying on explicit equations written by the designer, hardcoded in a generator. Because parasitics are essential and to avoid time consuming extractions/simulations, we propose to estimate the parasitics in the layout generation (figure 3). A parasitics loop is automatically performed, but without a simulation stage. This allows a faster generation and ensures a larger design space exploration. Instead of waiting days between each attempt, the analog designer can change the parameters almost "on the fly". Of course an external simulation is still performed, but at the very end of the design. The goal is that the

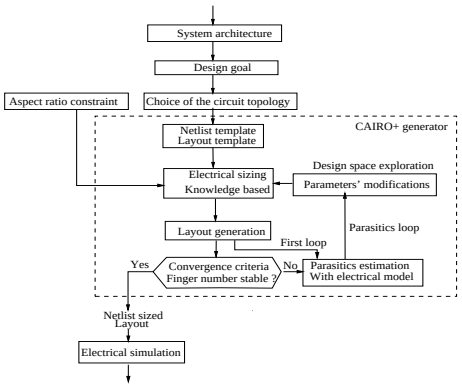

Figure 3: Knowledge capture and reuse approach

final generated netlist is as close as possible as the one extracted from the final generated layout.

Relevant criteria for an analog generator

As an answer to Analog IP creation and reuse, we propose analog function generator dedicated to a specific electrical topology. The generator input are a set of specifications (electrical values and aspect ratio) as well as technological data, and the output are a sized netlist together with the layout (figure 1). Such a generator has to deal with modules, made of interconnected devices which can be electrically and physically co-sized. In the following, we will explain what kind of information is handled by these generators.

Templates

Netlist template: specifying the electrical topology

The netlist template is an unsized hierarchical electrical netlist with logical connectors. Objects are devices (Transistors, differential pairs, resistors...) or modules (Integrators, modulators...). Figure 4 presents an example on the left.

Layout template: specifying the relative placement

To avoid expensive computing time, it is possible to specify a floorplan for the entire circuit. We have chosen a slicing tree structure in order to consider analog constraints such as symmetry. The whole description can be found in [2] (figure 4 on the right).

Specification template: specifying the input and output parameters for electrical sizing

Input parameters of a generator are the variables which can influence the realized analog function. Output parameters are all the informations given back to characterize the circuit. An explicit declaration forces the analog designer to clarify the equations of the system. It is very important to carefully choose the parameters: only the essentials parameters should be declared. The next section details this point.

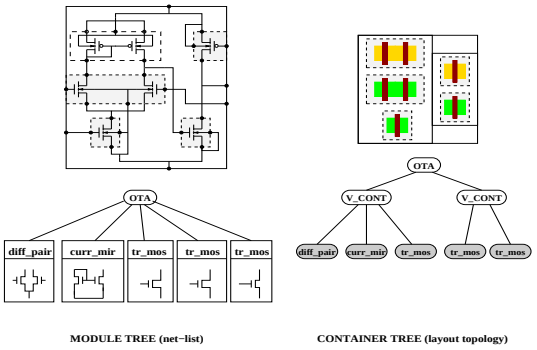

Figure 4: Netlist and Layout Templates

Design Space Exploration

Hierarchical communications - Question and answer mechanism

Generators should be “smart” generators: given a set of input data the generators are able to answer a question (Figure 5). This is a hierarchical mechanism.

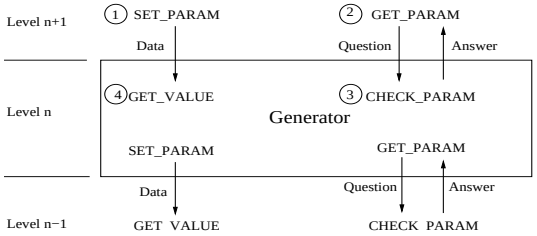

Figure 5: Communication mechanism

① In order to ask a question, the analog designer uses a SET_PARAM function to set all the parameters needed to answer the question (One call for each parameter to be set).

② Then the question is asked thanks to the GET_PARAM function, which gives back the answer.

③ Inside the generator, when the GET_PARAM function is called, the CHECK_PARAM function checks the question parameter

④ All the parameters are retrieved from the level n+1. Then the answer is calculated and returned.

The same process is also used between the level n and the level n-1.

Portability: process and specification

An other goal is to ensure the portability of the generator. Considering the lowest levels of the module tree (the devices), the equations are technological dependent. The communication mechanism explained in the previous section is an abstraction layer which hides all the details. In figure 6 the module asks for the “IDS” value to the device level (All the necessary parameters are shown on the figure). The implementation of the GET_PARAM function in the device calls a function belonging to a MOSFET calculator. This calculator contains a switch to call the correct function depending on the transistor model. As stated in figure 6 the answer is returned up to the module level.

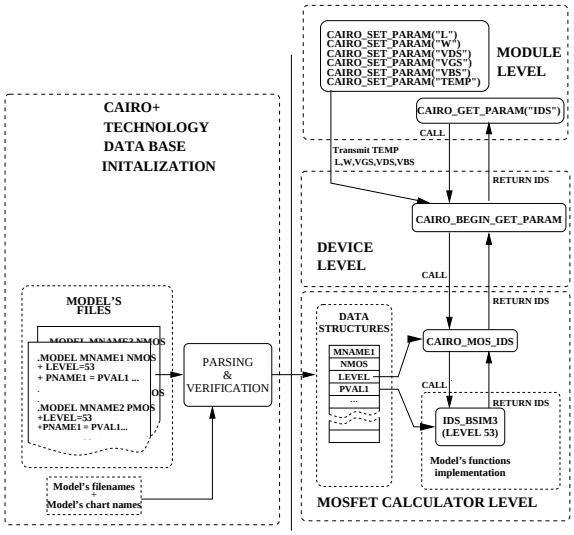

Figure 6: Process Electrical Parameter Management

Layout

In order to automate the parasitic loop in figure 3, it is compulsory to predict layout parasitics early in the design phase. We have seen that the layout template can describe the relative placement of instances inside a circuit as a tree, which leaf cells are devices (figure 4). Device generators have to provide two types of information : layout respecting analog requirements [7], as well as electrical models having an accurate knowledge of the actual device layout shape. Physical matching is included in the devices [6].

Sizing stages with the CAIRO+ language

CAIRO+ is an integrated design environment which implements the guidelines of this paper. The analog designer has to write four functions (figure 7). Each

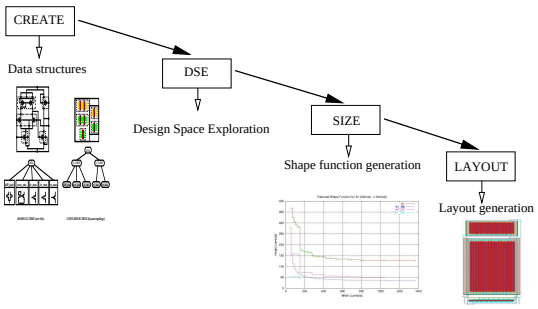

Figure 7: CAIRO+ functions

function is written as a variadic function:

```
CAIRO_BEGIN_XXX(NAME,char *name,parameters)
   // my code
CAIRO_END_XXX(NAME)
```

where XXX stands for CREATE, SIZE, GET_PARAM, LAYOUT (for a module) and GENERATE (for a device). “parameters” are c++ parameters choosen by the designer.

The create function contains the module/device instantiations, the electrical port declaration and the parameters declarations. Please note the three templates:

```
CAIRO_BEGIN_CREATE(OTA,char *name,char type,
                   bool ring,bool dummy)

  // Saving options
  CAIRO_SET_LOCAL_VARIABLE("ring",ring) ;
  CAIRO_SET_LOCAL_VARIABLE("type",type) ;

  // ********* NETLIST TEMPLATE

  // --------- Instance creation
  // Current mirror
  CAIRO_CREATE("libMOS","CM_MOS","CM","MP3","MP4",
               TRANSP,ring,dummy) ;

  // Differential pair
  CAIRO_CREATE("libMOS","DP_MOS","DP","MN1","MN2",
               TRANSN,ring,dummy) ;

  // Simple transistor
  CAIRO_CREATE("libMOS","TR_MOS","BIAS","MN5",
               TRANSN,ring,dummy) ;
```

```
  // ----------- connectors
  CAIRO_LOGICAL_IO("VE1",CP_WEST        ) ;
  ...
  CAIRO_LOGICAL_IO("VS" ,CP_EAST        ) ;
  CAIRO_LOGICAL_IO("VSS",CP_WEST,CP_EAST) ;

  // ----------- connections
  CAIRO_CONNECT("CM","SIG1","VS","VDD","VDD");
  CAIRO_CONNECT("DP","SIG1","VS","VE1","VE2",
                "SIG2","VSS") ;
  CAIRO_CONNECT("BIAS","SIG2","VP1","VSS","VSS");

  // ********** LAYOUT TEMPLATE

  CAIRO_VERTICAL_CONTAINER(name,
      "BIAS",SYM_Y,SD_NTIE,SD_NTIE,PITCH*2,PITCH*2,
      "DP",NOSYM,SD_NTIE,2*PITCH,SD_NTIE,SD_NTIE,
      "CM",SYM_Y,2*PITCH,3*PITCH,SD_NWELL_PTIE,
                                  SD_NWELL_PTIE) ;

  // ********** PARAMETERS TEMPLATE

  CAIRO_DECLARE_PARAM("TEMP"        ,CP_IN) ;
  CAIRO_DECLARE_PARAM("VDD"         ,CP_IN) ;
  CAIRO_DECLARE_PARAM("VSS"         ,CP_IN) ;
  CAIRO_DECLARE_PARAM("L_INIT"      ,CP_IN) ;
  ...
  CAIRO_DECLARE_PARAM("VP1", CP_OUT) ;
  CAIRO_DECLARE_PARAM("GBW",  CP_INOUT) ;

CAIRO_END_CREATE(OTA)
```

The GET_PARAM function is the design space exploration function. It contains the knowledge of the analog designer:

```
CAIRO_BEGIN_GET_PARAM(OTA,char *name)

    double temp      = 0.0 ;
    ...
    double capa_load = 0.0 ;

    // temp
    CAIRO_TRY_GET_VALUE("TEMP",temp)
      cout <<"--ampli : TEMP = " << temp << endl ;
    IF_NO_VALUE
      FATAL_ERROR_PARAM("TEMP","parameter not set",
                        LOCATION);
    ENDIF_NO_VALUE

    ...

    // ************** GBW parameter
    CAIRO_CHECK_PARAM("GBW")

      // ****************** BIAS point
      CAIRO_BEGIN_PROCEDURE("IBIAS_OTA")
          ...
          CAIRO_SET_PARAM("libMOS","DP_MOS","DP",
                          "VBS",vbs_dp) ;
          // Ask W=f(L,IBIAS,VGS)
          TRY
              CAIRO_GET_PARAM("libMOS","DP_MOS",
                "DP","W","W(L,IBIAS,VGS)",w_dp);
          IF_ERROR_PARAM
            cout << "Param : " <<VALUE<< endl <<
            "Reason : "<< WHAT<< endl ;
            exit(1) ;
          ENDIF_ERROR_PARAM
          // Ask for small signal parameters
          ...
          c2 = cgd_cm + cdb_cm + cgd_dp + cdb_dp
                                 + capa_load ;
          ...
          gbw = gm/c2 ;
          ...
          CAIRO_SET_LOCAL_VARIABLE("L_BIAS"...) ;

          CAIRO_RETURN_PARAM(gbw);

      END_PROCEDURE

      // ****************** GBW
      CAIRO_BEGIN_PROCEDURE("GBW_OTA")

      for(i = 1 ; i < 5 ; i++)
        {
          // Compute ibias current
          ids_bias = -ibias_ota_new ;
          ...
          CAIRO_SET_PARAM("libMOS","DP_MOS",...) ;

          TRY
           CAIRO_GET_PARAM("libMOS","DP_MOS",...);
          IF_ERROR_PARAM
            ERROR_PARAM("W","Required W is ...);
          ENDIF_ERROR_PARAM
          ...
          gbw_cal = ...
        }

       CAIRO_RETURN_PARAM(gbw_cal) ;

       END_PROCEDURE

       CAIRO_DEFAULT_PROCEDURE
          cout << "Unknown...module " << endl ;
          ERROR_PARAM("...");
       END_DEFAULT_PROCEDURE

  END_CHECK_PARAM
```

The SIZE functions calls the SIZE function of its children:

```
CAIRO_BEGIN_SIZE(OTA,char *name)

    CAIRO_SIZE("libMOS","DP_MOS","DP",w_dp,l_dp) ;
    CAIRO_SIZE("libMOS","CM_MOS","CM",w_cm,l_cm) ;
    CAIRO_SIZE("libMOS","TR_MOS","BIAS",
                                  w_bias,l_bias) ;
    ...
    CAIRO_COMPOSE_RSF() ;

CAIRO_END_SIZE(OTA)
```

Finally the LAYOUT function is in charge of the connectors placement and the routing:

```
CAIRO_BEGIN_LAYOUT(OTA,char *name)

  CAIRO_PHCON(ALU1,SW_ALU1,"VDD",NORTH,...) ;
  ...
  // VDD
  CAIRO_WIRING1("VDD",ALU1,SW_ALU1,...) ;
  ...

CAIRO_END_LAYOUT(OTA)
```

Design Examples

The fully differential current-mode integrator used in a third order current-mode continuous-time Sigma-

Delta modulator [8] is taken as a design example.

Figure 8 presents the methodology used for electrical sizing. Specifications coming from higher level are used together with simplified model of the transistors to choose the circuit biasing. Then using the BSIM3v3 built-in functions, a complete net-list of the integrator can be generated, with layout annotations.

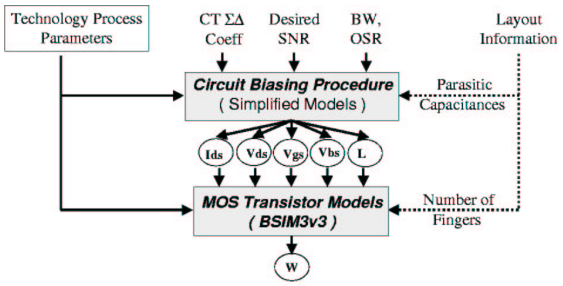

Figure 8: CAIRO+ Sizing Procedure for the ΣΔ Integrator

Figure 9 presents the layout of two current-mode integrators generated by CAIRO+ with the same set of specifications and two CMOS processes, illustrating the ability of CAIRO+ to ensure technology migration.

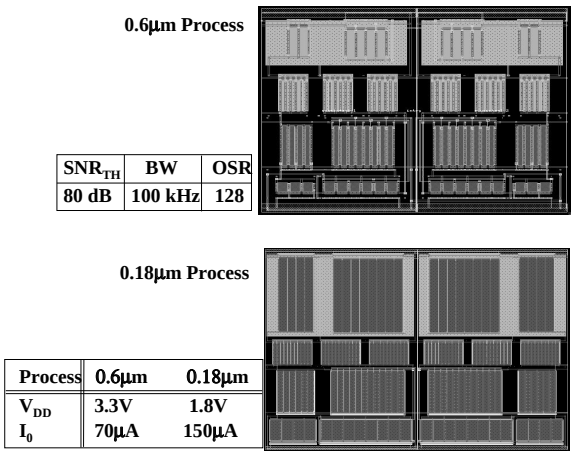

Figure 9: Layout of two current-mode integrators generated by CAIRO+ with the same set of specifications and two CMOS processes.

Conclusion

We have presented a guideline to design analog generators. Using a knowledge based approach and a new methodology without electrical simulations in the optimization loop, a demonstration tool called CAIRO+ was developed. It provides an integrated development environment to design reusable analog generators. Today the designer has to describe the equations of the electrical sizing, but we also intend to help him by plugging an optimization engine in our tool.

References

[1] Gielen and Rutenbar, "Computer aided design of analog and mixed-signal integrated circuits," *Proc IEEE*, vol. 88, pp. 1825–1852, December 2000.

[2] Pierre Nguyen Tuong, Marie-Minerve Louerat and Alain Greiner, "Managing the Shape Function of Analog Devices in a Slicing Tree Floorplan" *MIXDES 2004*, 23-26 june 2004, pp. 226-229

[3] Hershenson, "Efficient description of the design space of analog circuits," *DAC 2003*, pp. 970–973, June 2003.

[4] J.M.Cohn, R.A.Rutenbar, L.R.Carley, "Koan/Anagram II: New Tools for Device-Level Analog Placement and Routing," *IEEE JSSC*, pp. 330–342, March 1991.

[5] Vancoreland, Van de Plas, Steyaert, Gielen, and Sansen, "A layout-aware synthesis methodology for RF circuits," *ICCAD*, pp. 358–362, November 2001.

[6] M. Dessouky and M.-M. Louerat, "A layout approach for electrical and physical design integration of high-performance analog circuits," pp. 291–298, ISQED, 2000.

[7] Hastings, *The Art of Analog Layout*. Prentice Hall, 2001.

[8] H.Aboushady, L.de Lamarre, N.Beilleau and M.M.Louerat, "Automatic Synthesis ans Simulation of Continuous-Time ΣΔ Modulators", *DATE'04*