IIP Framework:
A Tool for Reuse-Centric Analog Circuit Design

Benjamin Prautsch*, Uwe Eichler*, Sunil Rao*, Björn Zeugmann*, Ajith Puppala*, Torsten Reich*, Jens Lienig†

*Fraunhofer Institute for Integrated Circuits IIS, Division Engineering of Adaptive Systems EAS, Dresden, Germany
{benjamin.prautsch, uwe.eichler, sunil.rao, bjoern.zeugmann, ajith.puppala, torsten.reich}@eas.iis.fraunhofer.de
†Dresden University of Technology, Dresden, Germany; jens@ieee.org

Abstract—**Current design of analog integrated circuits is still a time-consuming manual process resulting in static analog blocks which can hardly be reused. In order to address this problem, a new framework to ease reuse-centric bottom-up design of analog integrated circuits is introduced. Our *IIP Framework* (IIP: Intelligent Intellectual Property) enables the development of highly technology-independent analog circuit generators applicable in multiple design environments. IIP Generators are parameterizable descriptions of each view of an analog block, i.e., layout, schematic, and symbol. They allow the adaptation of complex layouts within seconds to minutes in order to incorporate hardly estimable parasitics and further considerations into the design flow. Due to the abstract generator description, valid design data is created for very different technologies such as 28 nm and 180 nm bulk CMOS, 28 nm FD-SOI, and others. The design experiment shows that procedural generators can be an effective tool for the efficient design of analog integrated circuits.**

Keywords—Layout; Analog Design Automation; Generator; Technology Independence; Reuse; Efficient Design; FD-SOI

I. INTRODUCTION

Analog circuit design is still a matter of comprehensive manual tasks. It includes the selection of circuit architectures and circuit topologies, a proper definition of the verification environment (testbench), circuit sizing, and finally the iterative design of all related layouts including placement and routing led by many very detailed expert decisions. This enormous amount of very different and separated steps results in long design times [1]. Combined with the lack of analog automation analog designers spend a great amount of manual work even for small problem sizes. Especially analog layout design is very tedious. In contrast, the digital domain benefits from complete synthesis flows. Thus, analog parts of ICs are not only critical regarding ever more important time-to-market, but they are also the main reason for circuit failure [2].

A. State of the Art

Basically two major approaches arose in order to address the analog design problem, particularly *optimization-based* and *procedural generator-based* methods [2, 3]. Both concepts are expected to collaborate in an industrial "bottom-up meets top-down design flow" [1], which is partly comparable to the more abstract *template-based optimization* approaches in academia [4, 5]. Optimization-based approaches are a very general way of addressing the analog design problem. A prerequisite for this methodology is that many constraints are handled properly [6] which, in addition, must be propagated throughout the entire circuit hierarchy [7]. Generators, on the other hand, are procedural "bottom-up" [1] descriptions of analog blocks with a dedicated structure of often comparably low complexity resulting in very high execution speeds. Their procedural nature can be subdivided into two major groups, particularly *parameterized cells* [8, 9, 10], which create blocks temporarily in the computer memory, and *circuit generators* or *IP generators* (IP: intellectual property) which create all views of an analog block as a persistent library cell similar to manual designs [11, 12, 13]. In addition, higher-level *templates* are often used in optimization-based approaches in. They refer either to more detailed parameterized cells [14, 15] or to more abstract layout representations [16, 17, 18]. Thus, in [18] templates are called either *geometric* or *structural/symbolic*, respectively. Furthermore, besides procedural description which is the focus of this paper, layouts can be fully described as an optimization problem as well [19].

Former procedural approaches either do not consider advanced design rules [15, 20, 21] or report related issues [9]. In [22] an abstract placement graph is utilized which is created from the generator code at runtime. It represents the intent of the designer, thus, such generators adopt the template-based principle on a detailed generator level. Additionally, most generator-based approaches are either fixed to a specific design environment or create parameterized cells only (see Fig. 1).

B. Our Contribution

In comparison to former approaches, our IIP Framework integrates the following advances into a single environment

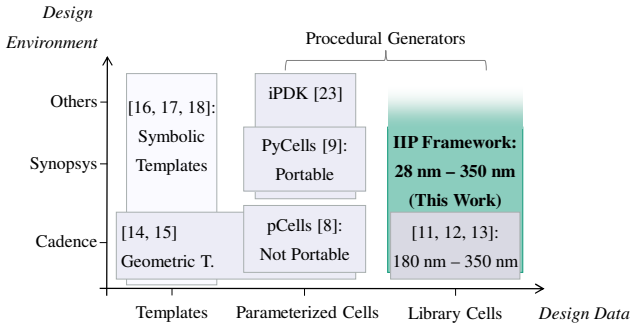

Fig. 1 Our contribution to procedural bottom-up analog design automation. While symbolic templates represent abstract descriptions of layouts, generators (or geometric templates) create cells. Library cells allow much more complex design hierarchies seamlessly contrary to parameterized cells (where additional steps would be required). Our IIP Generators, thus, create library cells and enable reuse over a wide spectrum of technologies as well as over multiple design environments.

Please quote as: B. Prautsch, U. Eichler, S. Rao, B. Zeugmann, A. Puppala, T. Reich, J. Lienig "IIP Framework: A Tool for Reuse-Centric Analog Circuit Design," Proc. of the Int. Conf. on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD 2016), Lisbon, Portugal, 27-30 June 2016.

which can either be used by an optimization-based framework or directly by a designer:

- An abstract programming interface is provided to ease technology independence. Additionally, an abstract placement graph representation is created automatically.
- The generator code programmed using the *IIP Framework* is independent from the design environment.
- Each (parameterizable) generator produces persistent library cells similar to manually designed cells instead of parameterized cells.

This paper is organized as follows. In Section II the *IIP Framework* is presented. A generator-based design example is discussed in Section III and finally, Section IV summarizes and concludes this work.

II. THE IIP FRAMEWORK

Our new *IIP Framework* (IIP: Intelligent Intellectual Property) is focused on the *procedural generator-based* approach. It provides a programming interface, the IIP API, used for the development of generic generators (IIPs) for all views of an analog circuit. The IIP API is based on the object-oriented programming language Python which eases both modularization of the circuit description and the development of generators. The modularization of our IIP Framework targets the design of flexible (reusable) analog circuit generators which are highly technology-independent as well as independent from the design environment (DE). Such highly generic generators are suitable to reduce the design time due to improved reuse both for a particular design task as well as for another design project and other technologies (porting). According to our former work [12] and other approaches [10, 20], we target a complex generator library—however, for a much wider range of technologies.

A. Abstraction of Technologies

For our IIP approach we have implemented a separate command layer called TAL (Technology Abstraction Layer), which separates detailed technology-dependent tasks concerning parameterization and detail placement from the particular procedural generator code. These technology-dependent tasks are executed by a dedicated part of the *IIP Framework* leading to a clear separation between generator code and technological details which results in a high degree of technology independence of our generators. Instead of defining generator code which calculates layout locations, the TAL syntax allows the definition of detailed *relative placement relations* (cardinal directions) and creates an *abstract graph representation* of these relations during run time. Technology data will then be used internally to calculate the final DRC-compliant locations of layout entities depending on the TAL commands (please refer to [22] for more details). The graph is an abstract representation of the designer's intent which can later be used either for visual code checking (GML files are created and can be visualized) or for the application of algorithms which can check and/or modify the design. New technology data and new design rules are implemented in the separated TAL layer which prevents changes at any existing generator code.

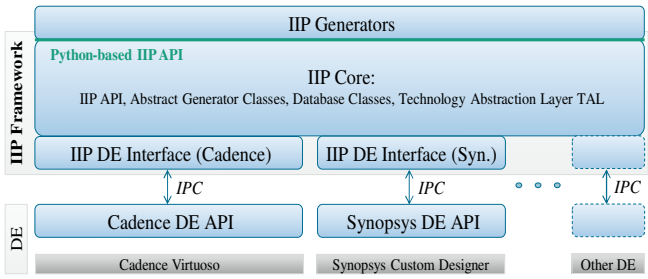

Fig. 2 Layer model of the IIP Framework to access the design database of the corresponding design environment (DE). IIP Generators and IIP core software can be used unchanged for further design environments. Only the DE-specific interfaces need to be developed.

B. Abstraction of Design Environments (DE)

The idea of DE-independence is already followed on a rather low (device) level to create interoperable PDKs, so-called iPDKs [23]. The *IIP Framework* is implemented in a way which allows simple addition of interfaces to further design tools and databases (also interaction with OpenAccess databases would be possible). Fig. 2 shows the concept of this modularity. The API used by IIP Generators allows a rather high level and object-oriented, thus compact, generator description which, therefore, results in a comparably large number of script commands in the design environment (= high IIP code efficiency). If a new design environment is used, only the related low-level interfaces must be developed. Currently, interfaces for both Cadence Virtuoso® and Synopsys Custom Designer® exist.

This flexibility is an advantage when providing or porting analog IP for different teams or projects and is not possible with DE-specific soft IP solutions such as in [8, 11, 24].

C. Generator Structure

Each generator is programmed in the same structural way using class inheritance. The following methods are always utilized. First, in *param_spec()* parameters and their constraints are defined which either are automatically shown in a parameter mask (used by the designer) or hierarchically utilized by higher-level generators. Second, in *param_check()* cross-dependencies of parameters are defined which are considered automatically. Third, the generator parameters are prepared using method *prepare()* to maintain coherent data for all views of a generated cell. This includes that identical parameters for both schematic and layout (e.g. transistor width) are stored only once to improve LVS-compliance and code compactness. Finally, separate methods are implemented to describe the circuit representations (views) for schematic, layout, and symbol (as well as optionally a testbench).

D. Generator-Based Design Flow for Improved Reuse

Instead of fixed, sized designs, parameterizable IIP Generators describe all required views in a flexible way. They may provide initial sizing values for each technology based on formulas which utilize access to process parameters from the technology interface through TAL (see Section II.A), as e.g. mobility factors. Of course, these initial values must be subsequently refined either by a designer and/or optimizer.

The particular development flow is as follows. First, a schematic is designed manually. Then, using our *Schematic*

Importer, the existing schematic is converted into an initial generator code template which exactly replicates the input schematic design (using the input data as default parameters, which can be adapted). Moreover, TAL is used to convert particular technology-dependent parameters into their generic representation (which may not be a bijection, thus, the initial generator might require few changes to correct ambiguities). The code template is the basis to integrate additional functionality into the generator. Moreover, the layout is programmed in this step. Once the procedural generator code is finished, it can be instantiated together with other generators or PDK devices into a new analog block. Using the *Schematic Importer* again, this block can be imported into another generator template followed by the aforementioned procedure. This way, hierarchical IIP Generators are built efficiently.

III. Design Experiment

We have designed a 12 bit current-steering digital-to-analog converter (CS-DAC) in the STM 28 nm FD-SOI process as part of a test chip. Its current mirror stage is large in size but has a very regular structure. Contrary to the *general* reuse concept presented in [12], we have decided to develop a *dedicated* generator for this particular task. Even such (structurally fixed) dedicated generators are likely to be reused, since the chosen DAC topology is frequently utilized in many designs. Moreover, the regular structure of the CS-DAC can be implemented efficiently using iterative generator code. Therefore, the overall design effort including the next test chip and the final chip is reduced. In addition, our flexible IIP can be reused in future designs for other specifications and other technologies.

A. Topology of the DAC Output Stage

A segmented current method is adopted in our implementation, wherein eight MSBs (most significant bits) are realized by 255 thermometer code units with a weight of 16 each. The remaining four LSBs (least significant bits) are binary weighted (1, 2, 4, and 8). This segmentation targets the trade-off between minimized silicon area while still achieving the required non-linearity specification. A total of 4095 cascode current mirror stages (plus one dummy) are required and must be matched. All cascode unit cells are placed using the so-called Q² random walk switching scheme [25] to achieve high intrinsic matching. This scheme is instantiated 16 times in the DAC matrix. Additionally, we used the back-gate contact available in the 28 nm FD-SOI process to lower the threshold voltage of the cascode transistors, which decreases by about 85 mV/V [26]. In contrast, the body connection in bulk technologies is typically tied to a static voltage. The consideration of this *structural* technological difference is important in order to achieve robust and reusable generator descriptions since it cannot be mapped through TAL.

B. Abstract and Concrete DAC Layout

Using our IIP API, we defined the DAC matrix layout in an abstract and hierarchical way which is highly independent from the technology (see layouts in Fig. 3). This means that the cascode current mirror output itself and its hierarchical

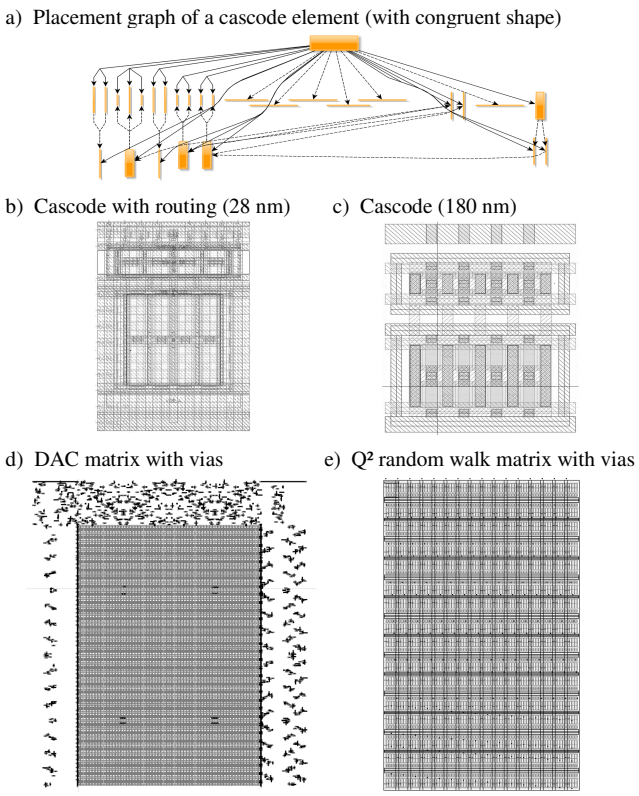

Fig. 3 Illustration of the design experiment. In (a) the abstract placement graph representation of one folded MOS transistor which is a part of the cascode is illustrated. Solid edges represent placement relations while dashed edges represent hierarchical relations (more details can be found in [22]). Figures (b) and (c) depict examples of the cascode in 28 nm (with routing mesh) and 180 nm, respectively. Figures (d) and (e) show the complete DAC matrix and the Q² random walk matrix, respectively; both in 28 nm with vias marked black.

instantiation within the matrix structure is defined in a parameterizable manner. The procedural generator code is transformed into an abstract graph representation during the generator run (see the graph for one element of a cascode in Fig. 3a; the overall graph is not shown due to its complexity).

The implementation of the generator is subdivided into two steps. First, the matrix arrangement is defined in a dedicated class implementation storing the Q² random walk scheme with instance rotation. In the second step, this representation is used to place each cascode cell. These cells contain a predefined and unconnected routing mesh in order to connect their output net. For each placement of a particular cell it is calculated which routing channels of this mesh (horizontal or vertical) are to be connected, meaning that vias are created on a higher hierarchy level in a way comparable to switch boxes. Once each cascode cell is placed and connected, the overall "random" (but regular) array is finished (see Figures 3d, e).

C. Design Results

The DAC matrix generator was executed for multiple parameter sets. Especially width and spacing of the routing tracks were varied. This strongly affects the parasitic resistance of metal lines (and parasitic capacitances in-between) in the 28 nm process in a way, which can hardly be estimated, since sheet parasitics vary greatly depending on the absolute size of layout shapes. Table 1 summarizes the estimated manual design effort, the IIP Generator

development effort, the number of IIP code lines, the number of generated DE commands (here: SKILL [8], cf. Fig. 2), the quotient of DE commands divided by IIP code lines (IIP code efficiency), minimal required reuse for amortization (cumulative IIP Generator development time over cumulative manual design time), and the runtimes in both 180 nm and 28 nm. The reason for the difference of the runtimes is that in the more advanced process node more complex design rules are to be considered. The table shows that numerous DE commands (they are approximately proportional to the manual effort) were executed which results in high IIP code efficiency especially for regular structures. The initial IIP development effort pays out with the first reuse (minimal required reuse is 1.8). Although the generator development is less efficient in lower hierarchy levels (more reuse is required), it is very efficient in the higher ones.

The DAC matrix IIP Generator was initially developed with focus on a 28 nm bulk technology. With an effort of only one day, including related changes on TAL and the back-gate, we made the generator compatible with the 28 nm FD-SOI technology in which we finally taped out. In addition, our IIP Generator was tested in a 180 nm bulk technology. Currently, six different technologies ranging from 350 nm down to 28 nm are available through TAL (22 nm is planned). Other approaches with such high reusability are not known to us.

TABLE 1 COMPARISON OF THE EFFORT OF MANUAL AND GENERATOR-BASED LAYOUT DESIGN INCLUDING RUNTIMES

	Two cascode elements (one IIP)	**Cascode**	**Q^2 random walk matrix**	**Overall DAC matrix**
Est. manual time *block (cumulative)*	*0.25 Days (0.5 Days)*	*0.5 Days (1.0 Day)*	*1 Week (1.2 Weeks)*	*1 Week (2.2 Weeks)*
IIP Generator development time *block (cumulative)*	*1 Week (1 Week)*	*1 Week (2 Weeks)*	*1.5 Weeks (3.5 Weeks)*	*0.5 Weeks (4 Weeks)*
IIP code lines *layout (schematic)*	*700 (80)*	*1300 (40)*	*750 (70)*	*175 (550)*
DE commands *block (all blocks)*	*1314, 1356 (2470)*	*359 (3029)*	*3954 (6983)*	*17151 (24134)*
DE commands/ IIP code	*1.9*	*0.3*	*5.3*	*98*
Min. required reuse	*10*	*6.7*	*2.9*	*1.8*
CPU runtime IIP @ 180 nm	*1.9 s*	*4.3 s*	*21.6 s*	*110 s*
CPU runtime IIP @ 28 nm	*2.2 s*	*5.1 s*	*30.0 s*	*117.4 s*

IV. SUMMARY AND CONCLUSION

In this work we have presented a new tool to ease analog integrated circuit design and design reuse. Our *IIP Framework* (IIP: Intelligent Intellectual Property) enables the development of highly technology-independent, parameterizable, and hierarchical procedural circuit generators which can be executed in multiple design environments. Utilizing our reuse-centric method, we have designed the complex current mirror stage of a 12 bit current steering DAC as parameterizable generator in about one month. This generator was tested in three very different technologies and was utilized in an out-taped DAC design in 28 nm FD-SOI.

In our opinion, fast and robust generators are essential in advanced processes due to the high amount of hardly estimable parasitics and complex design rules. Although generators are structurally static, especially regular layouts can be realized very efficiently. Moreover, abstract generator descriptions allow a high degree of parameter flexibility and technology independence. We believe that advanced generators are key elements in order to address the bottom-up part of future automation in leading-edge analog integrated circuit design.

ACKNOWLEDGEMENTS

We would like to thank Andreas Krinke for his valuable suggestions and comments on this paper. The presented work was partly supported by the European Union and the Free State of Saxony within the project THINGS2DO (Ref. No. 16ES0240).

REFERENCES

[1] J. Scheible and J. Lienig, "Automation of Analog IC Layout – Challenges and Solutions," *Proc. Int. Symp. on Physical Design,* pp. 33–40, 2015.

[2] G. G. E. Gielen and R. A. Rutenbar, "Computer-Aided Design of Analog and Mixed-Signal Integrated Circuits," *Proc. IEEE 88.12,* pp. 1825–1854, December 2000.

[3] R. A. Rutenbar, "Analog Synthesis (and Verification) Revisited: Whats's Missing?," *Int. Conf. on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design, SMACD,* Sep. 2012, http://rutenbar.cs.illinois.edu/publication/. [Accessed May 2016]

[4] H. Graeb, et. al., "Analog Layout Synthesis - Recent Advances in Topological Approaches," *Proc. Conf. on Design, Automation and Test in Europe,* 2009.

[5] R. Martins, N. Lourenco, S. Rodrigues, J. Guilherme and N. Horta, "AIDA: Automated Analog IC Design Flow from Circuit Level to Layout," *Proc. Int. Conf. on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2012.

[6] G. Jerke and J. Lienig, "Constraint-driven Design — The Next Step Towards Analog Design Automation," *Proc. 2009 Int. Symp. on Physical Design,* pp. 75–82, 2009.

[7] A. Krinke, G. Jerke and J. Lienig, "Constraint Propagation Methods for Robust IC Design," *Proc. ZuE 2015; 8. GMM/ITG/GI-Symp. Reliability by Design,* pp. 1–8, 2015.

[8] Cadence, "Virtuoso Parameterized Cell Reference Product Version 6.1.6," 2015.

[9] Synopsys, "PyCell Studio," [Online]. Available: https://www.synopsys.com/Tools/Implementation/Customimplementation/Pages/pycell-studio.aspx. [Accessed May 2016].

[10] J. Crossley, et. al., "BAG: A Designer-Oriented Integrated Framework for the Development of AMS Circuit Generators," *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD),* pp. 74–81, 2013.

[11] IPGen 1Stone Developer, [Online]. Available: http://www.ipgenme.de/eda-and-ip-products/1stone-developer.html. [Accessed May 2016].

[12] T. Reich, U. Eichler, K.-H. Rooch and R. Buhl, "Design of a 12-bit Cyclic RSD ADC Sensor Interface IC Using the Intelligent Analog IP Library," *Proc. ANALOG 2013 – Entwicklung von Analogschaltungen mit CAE-Methoden,* March 2013.

[13] T. Reich, H. D. B. Prautsch, U. Eichler and R. Buhl, "Silicon Proof of the Intelligent Analog IP Design Flow for Flexible Automotive Components," *Proc. Design, Automation & Test in Europe Conf. & Exhibition,* pp. 403–404, 2015.

[14] R. Castro-López, O. Guerra, E. Roca and F. V. Fernández, "An Integrated Layout-Synthesis Approach for Analog ICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 27, no. 7, pp. 1179–1189, Jul 2008.

[15] R. Castro-López, F. V. Fernández, F. Medeiro and A. Rodriguez-Vazquez, "Generation of Technology-Independent Retargetable Analog Blocks," *Proc. Analog Integrated Circuits and Signal Processing,* vol. 33, no. 2, pp. 157–170, 2002.

[16] A. Unutulmaz, G. Dündar and F. V. Fernández, "A Template Router," *Proc. 20th European Conf. on Circuit Theory and Design (ECCTD),* pp. 334–337, 2011.

[17] R. Martins, N. Lourenco and N. Horta, "LAYGEN II—Automatic Layout Generation of Analog Integrated Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* pp. 1641–1654, 2013.

[18] N. Jangkrajarng, S. Bhattacharya, R. Hartono and C.-J. R. Shi, "IPRAIL—Intellectual Property Reuse-Based Analog IC Layout Automation," *Integration, the VLSI Journal,* vol. 36, no. 4, pp. 237–262, 2003.

[19] H. Habal and H. Graeb, "Constraint-Based Layout-Driven Sizing of Analog Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 30, no. 8, pp. 1089–1102, 2011.

[20] X. Jingnan, J. Vital and N. Horta, "A SKILL-based Library for Retargetable Embedded Analog Cores," *Proc. Conf. on Design, Automation and Test in Europe,* pp. 768–769, 2001.

[21] K. Lampaert, G. Gielen and W. Sansen, "Module Generation," *Analog Layout Generation for Performance and Manufacturability*, Boston, ISBN 0-7923-8479-2, 1999, pp. 53–69.

[22] B. Prautsch, U. Eichler, T. Reich, A. Puppala and J. Lienig, "Abstract Technology Handling for Generator-Based Analog Circuit Design," *Proc. ZuE 2015; 8. GMM/ITG/GI-Symp. Reliability by Design,* pp. 1–6, 2015.

[23] IPL (Interoperable PDK Libraries) Alliance, "IPLnow," [Online]. Available: https://www.iplnow.com/. [Accessed May 2016].

[24] A. Graupner, R. Jancke and R. Wittmann, "Generator Based Approach for Analog Circuit and Layout Design and Optimization," *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, IEEE, 2011, pp. 1–6.

[25] G. A. Van der Plas, J. Vandenbussche, W. Sansen, M. S. Steyaert and G. G. Gielen, "A 14-Bit Intrinsic Accuracy Q^2 Random Walk CMOS DAC," *IEEE Journal of Solid-State Circuits,* pp. 1708–1718, 1999.

[26] P. Flatresse, "UTBB-FDSOI Design & Migration Methodology," [Online]. Available: http://cmp.imag.fr/IMG/pdf/utbb-fdsoidesign_migration_methodology_.pdf. [Accessed May 2016].