FACTA UNIVERSITATIS (NIŠ)
Series: Electronics and Energetics vol. 11, No.3(1998), 1-8

SEMIAUTOMATED IMPLEMENTATION OF ANALOGOUS FUZZY SYSTEMS

Silvio Triebel, J. Kelber and Gerd Scarbata

Abstract. A tool set for the semiautomated design of analog fuzzy hardware for on–chip implementation is presented here. Based on the description of the desired behaviour, established for example by a commercial development tool and available in a standard format (FPL), this tools are able to generate all necessary data to describe a hardware cell providing the specified behaviour. Here analogous circuits espeacially in CMOS are considered.

1. Fuzzy hardware

One of the most actual problems regarding the realization of technical systems is the specification and implementation of non–linear functionality. Here the fuzzy methodology is an interesting solution. It allows to describe problems in terms of human–like language constructs and is able to approximate non–linear functions with only little restrictions. Furthermore, it supplies a methodology to transform such constructs into formal expressions which can be implemented by well–known systems, e.g. standard software.

Today such fuzzy systems are mainly designed using one of the fuzzy development tools available on the market. Mostly these tools are easy to use and offer different specification and simulation capabilities. Using such tools the specified behaviour of a developed system is expressed in terms of well–defined language constructs.

Starting from a behavioural description different ways are possible to implement a fuzzy controller:

Manuscript received September 1, 1998.

The authors are with Technical University Ilmenau, Faculty of Electrical Engineering and Information Technology, Department of Microelectronic Circuits and Systems, PO Box 10 05 65, D-98684 Ilmenau, Germany, E-mail: silvio@inf-technik.tu-ilmenau.de.

(1) software on standard computers,

(2) programmable fuzzy chips and

(3) dedicated fuzzy hardware.

The common way is to implement the fuzzy system on a standard computer using programming languages like "C". Furthermore, there are some special fuzzy chips available which can be adapted to the given problem by programming internal memory [2,5]. The disadvantage of solution (1) is that it runs slowly in comparison to (2) and (3). Solution (2) normally comes with a relatively large overhead because such chips are designed to meet most of the possible requirements.

In some cases it can be attractive to have a dedicated hardware for a fuzzy system designed for a special application. For example, such a solution is interesting in the case of:

1. hard timing restrictions: optimization possibilities for systems with fixed hardware are very limited
2. mobile applications: decrease of power consumption due to the absence of programming overhead
3. high production volume: decrease of chip area

A possible application of dedicated analogous fuzzy hardware can be found in on–chip embedded systems. Such systems normally consist of a processor core running some software and additional components. Interfaces to sensors and actors are an important part. To prevent these components from distruction, normal operation is to be distinguished from malfunction in a very short time. Analyzing this problem, non-linear functions occur which can be modelled by fuzzy formalisms. A software solution for the implementation will normally be much too slow. One possible solution is the inclusion of anlogous fuzzy hardware inside the interfaces themselves.

The design process for an embedded system including fuzzy hardware is illustrated in Fig. 2. The problem, of course, is the dedicated design and production of the hardware which leads to relatively high design costs. To minimize these amounts and to possibly consider more than one potential solution during the design phase, the generation of the hardware–specific design data should be done automatically. For this task the concept of a module generator, known from other applications (e.g. RAM, PLA), seems to be a useful solution strategy. Based on a specified behaviour it should be able to generate all necessary implementation data.

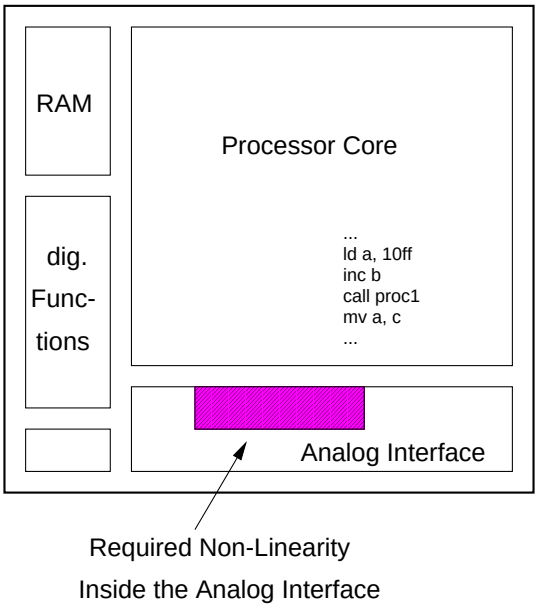

Figure 1. Example of an on–chip embedded system.

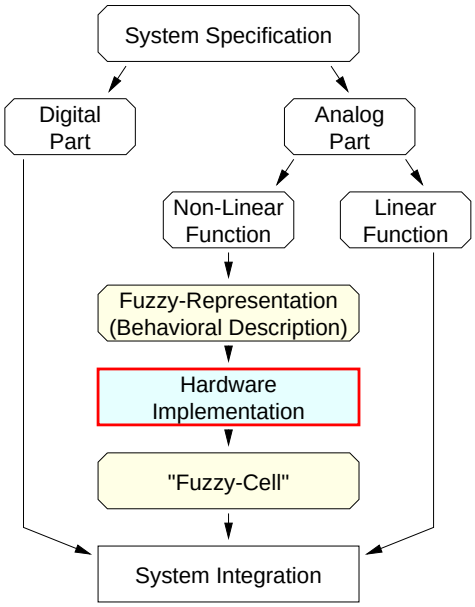

Figure 2. Design process for systems with fuzzy hardware.

The generation result is a cell in a given chip technology, but not only a netlist of gates and transistors. This set of data can be used as an input to common chip CAD–systems for further processing (e.g. design rules check, post layout simulation) or to include it in more complex chips. This paper deals with a module generator for analogous fuzzy hardware, developed at our institute. These investigations are based on the following fundamentals:

- fuzzy Values are represented by electrical currents
- restriction to standard CMOS technologies
- generation of compact and regular layouts
- strictly hierarchical implementation of the generator
- implementation of the software in C++

2. Basic design flow

The basic design flow is divided into two steps comparable to the common design process of other systems, for example digital hardware (see Fig. 3. and 5.). After a description of the problem by means of fuzzy formalisms, which is not the subject of our work, the first step of the hardware implementation is a structural interpretation of the specified behaviour. This is done by a simple assignment of a component to a particular function. For example a dedicated multi-input MAX-operator is provided for each required MAX-function. Instead of this procedure a real synthesis approach will be discussed in the future but is not included in this paper. The result of the structural interpretation is a netlist of fuzzy components denoted in an ADL-file[1]. Each component must have a representation in the library of the module generator tool set. Additionally the designer can specify toplogical constraints to control the layout generaion process.

The ADL-format is a language for the description of functional units. It has been developed at our institute and can include component instantiation, connectivity description and geometrical (topological) information, for example. The language is similar to "C" and contains only a few dedicated constructs. So it is very easy to understand.

In Fig. 4 an example of an ADL–file together with the resulting layout is shown. The whole cell consists of four parts which are stacked vertically. This is described by the "abutment over"–expression. At the bottom there are two linguistic variables with five trapezium fuzzy sets each. In the upper part, the rule base and the defuzzification circuit is located. The internal structure of the four components is described in separate files which are included in the overall ADL–file of Fig. 4.

[1] **A**butment-oriented Layout **D**escription **L**anguage

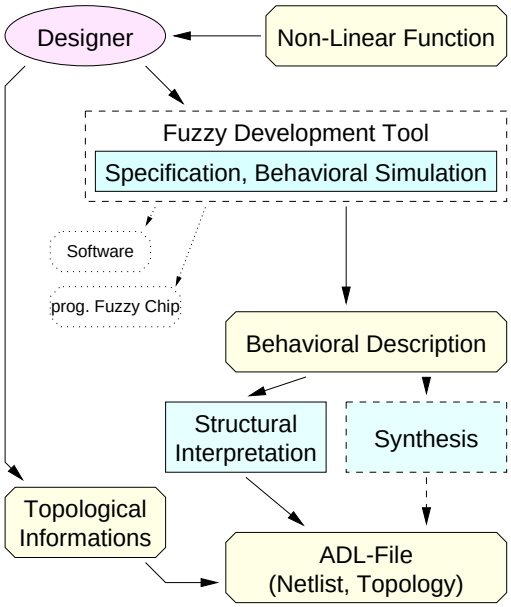

Figure 3. Step 1: Derivation of a structural description.

```
include "defuzzy.adl"
include "rulebase002.adl"
include "lingvar.adl"

controller()
 {
  terminal EK    : RIGHT, INPUT;
  terminal DEK   : RIGHT, INPUT;
  terminal UK    : RIGHT, OUTPUT;
  terminal U1    : RIGHT, INPUT;
  terminal U5    : RIGHT, INPUT;

  abutment over;
   call COG         UK  (0 0.17 0.5 0.83 1);
   call RuleBase002 RB  ();
   call LingVar     DEK (0 0.375 0.375 0.44
                  0.56 0.625 0.625 1 prec=1);
   call LingVar     EK  (0 0.25 0.25 0.5 0.5
                  0.75 0.75 1 prec=1);

  orientation RB MY;

  i = 1;
  while (i<=5) {
   connect RB.uk[i] UK.i[i];
   connect RB.ek[i] EK.out[i];
   connect RB.dek[i] DEK.out[i];
   i = i + 1;
  }

  connect EK.in EK;
  connect DEK.in DEK;
  connect UK.v1 U1;
  connect UK.v5 U5;
  connect UK.out UK;

  connect *.vdd vdd LEFT;
  connect *.gnd gnd LEFT;
  netattrib width gnd 3*size("Metal2");
  netattrib width vdd 3*size("Metal2");
 }
```

Figure 4. Example of an ADL–file and the resulting layout.

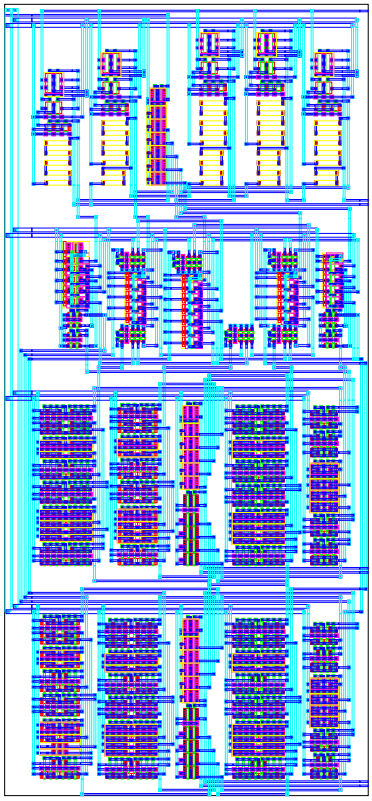

Figure 4. continue

The ADL-descriptions are the input to the second step of the design flow. Here mainly the physical layout and the corresponding transistor netlist are generated (Fig. 5).

3. Layout generation

During the layout generation a placement is executed first (Fig. 5). This is done by simulated annealing. The designer can define topological constraints explicitly. This is strongly recommended because the cost function of the annealing algorithm does actually not take all necessary information into account.

After the placement the generation of the physical layout is executed. The final sizes of the leaf cells are calculated by iterative calls of the built–in generators which come with some useful functions to support this process.

To generate the layout of the complete cell three strategies are used. The first is a pure abutment of subcells. It is applied to the regular parts of the circuit. The characteristic property of these parts is the possibility to divide them into some similar or identical components realized by subcells. In this case some restrictions have to be applied to the layout of the subcells. For example, the terminal positions have to be fixed. The advantages are the absence of an explicit routing between the subcells and a layout with nearly optimal area consumption. Examples for abutment layout are current mirrors for membership functions and multi–input MIN/MAX operators (Fig. 6).

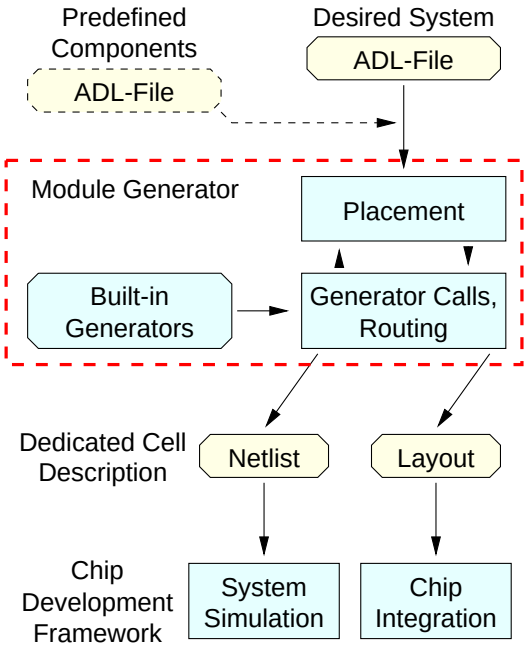

Figure 5. step 2: layout generation.

The second layout strategy is a slicing layout, known from previous work at different institutions. This strategy is very suitable for analog layouts. It allows different components to be considered. Nevertheless matching and other constraints can be taken into account. In this strategy routing channels are introduced between the subcells. Third total irregular layouts can be included but must be designed by hand.

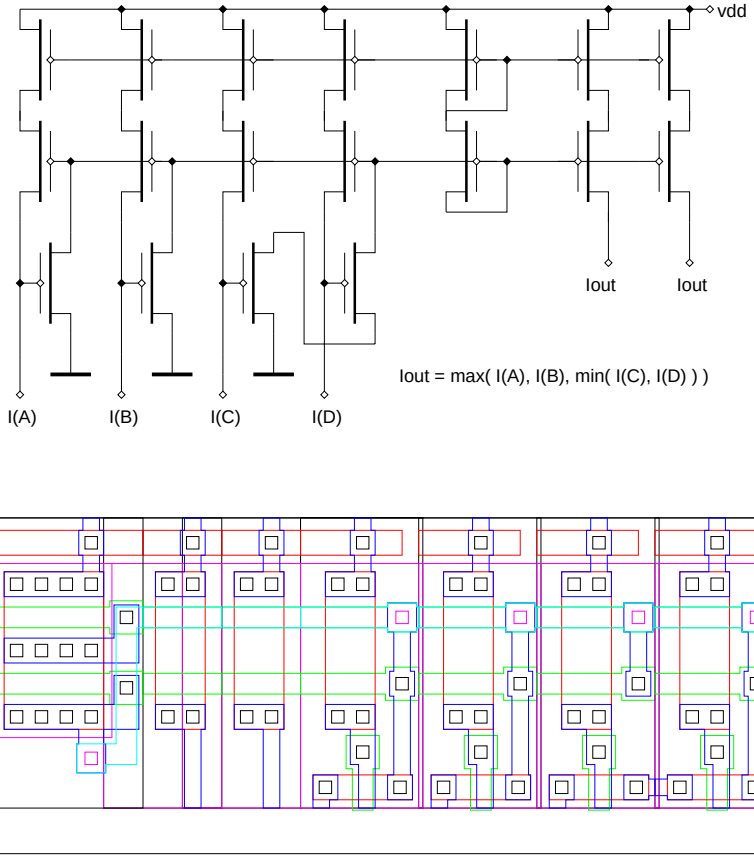

Figure 6. schematic and layout of a multi–input MIN/MAX-operator.

The focus of our work was on the development of the generator library and the analog router. The generator library consists of two parts: built–in generators and ADL-macros. The built–in generators are written in C++ using an own library called "WOOD". Actually the library include the following parts:

- MIN- and MAX- gates
- combined MIN/MAX- gates (Fig. 6)
- cascaded current mirror in- and outputs
- several transistors,
- differential transistor pairs,
- resistors and
- capacitors.

For the implementation of the bulit–in generators a quasi–symbolic layout description was used to abstract from the large amount of data occuring due to the consideration of polygons and to become more technology independent. A special class library with transistors and other relevant objects was created. All objects provide methods for absolute and relative placement, compaction and abutment. The creation of the layout is based on a generic technology to which the used production technology must be bound. Among other things, comparable methods were already published in [4]. Fig. 7 shows the layout generation of a current mirror input, for example.

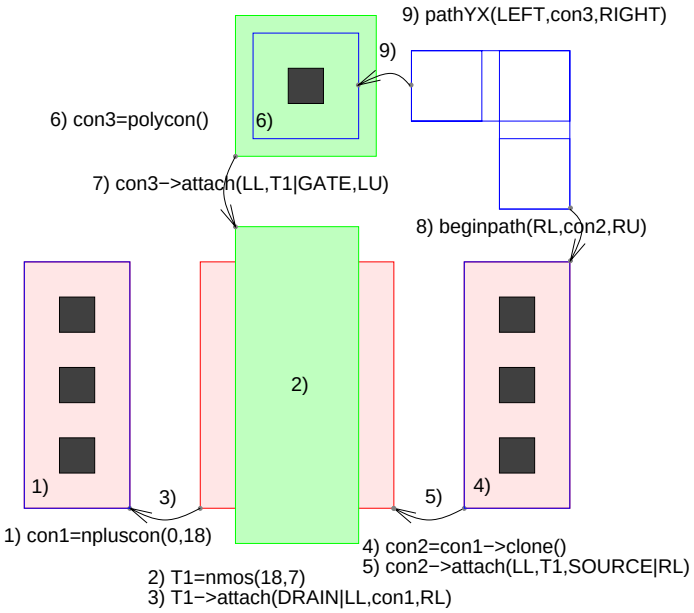

Figure 7. example for the quasi–symbolic layout description.

The generators offer a couple of useful properties. So a fast algorithm for the calculation of the sizing parameters of each cell in a slicing floorplan was implemented. Furthermore, the generators distinguish virtual from real interface terminals. For global routing all possible terminal positions are available. The global router decides, which terminals are really to be generated. So the optimization space is not restricted additionally and the existence of unused wires inside the reusable modules will be avoided.

The implemented channel router is based on the results in [1]. This approach provides the possibility of different wire width and distances. The channel boundaries can be irregular. The algorithms were improved to

handle terminals in different layers and to allow the description of "wire–clusters" for example pairs of wires with symmetrical geometry (Fig. 8) or bundles of wires which could be kept together to provide common shielding structures.

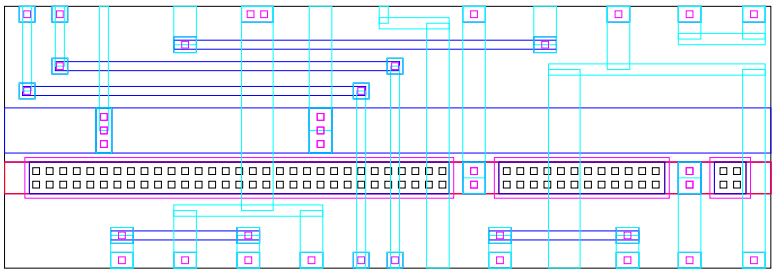

Figure 8. Example of a channel.

For example, the generated layout of a fuzzy controller is shown in Fig. 9. It was implemented in a 2.4μm CMOS technology (MIETEC). The core size of the implemented chip is about 1.5mm^2.

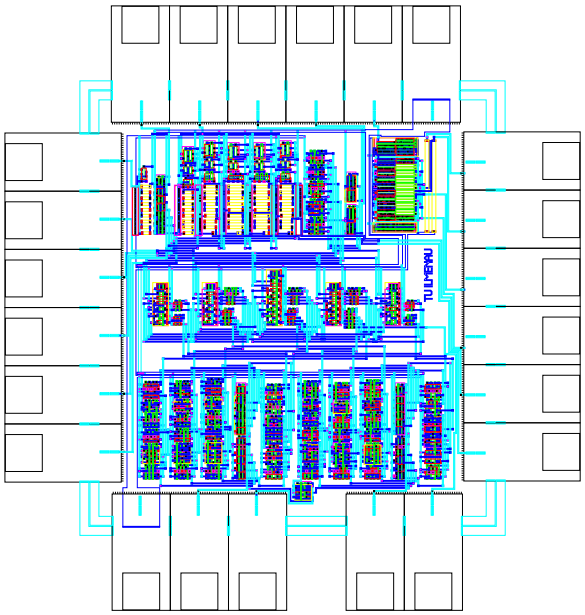

Figure 9. Layout of a fuzzy controller chip.

4. Conclusions

Discussed here was a realization of a module generator to design analogous hardware for fuzzy–controller automatically. Such a generator seems to make a good link between the world of fuzzy development tools on one side and chip development systems on the other side. Of course there are many problems left to be solved. Examples are the possible implementation of a true synthesis process and the development of additional interface modules to overcome the restriction of the application of only currents to represent the interesting values.

Acknowledgement

The authors would like to thank the "Deutsche Forschungsgemeinschaft" supporting this work within the "Sonderforschungsbereich 358".

REFERENCES

1. R.S. Gyurcsik, J.-C. Jeen: *A generalized approach to routing mixed analog and digital signal nets in a channel.* IEEE J. Solid–State Circuits 24 (2): 436–442, April 1989.
2. H. Eichfeld, T. Künemund, M. Menke: *Architecture of a General–Purpose 12 Bit Fuzzy Coprocessor.* Proc. 3. EUFIT, Aachen, Sept. 1995, pp. 1815–1819.
3. J. Kelber, S. Triebel, G. Scarbata: *Automatic Generation of Analogous Fuzzy Controller Hardware Using a Module Generator Concept.* Proc. 2. EUFIT, Aachen, Sept. 1994, pp. 1562–1569.
4. B.R. Owen, R. Duncan, S. Jantzi: *Balistic: An Analog Layout Language.* IEEE Custom Integrated Circuits Conference, 1995, pp 3.5.1–3.5.4.
5. H. Watanabe, W. Dettloff, K. Yount: *A VLSI fuzzy logic controller with reconfigurable, cascadable architecture.* IEEE journal of solid–state circuits, Vol. 25, April 1990, 2, pp. 376.