A GaAs data-path compiler

O. Beaurin, P. Royannez,
J-P.CHAPUT
Laboratoire MASI/CAO-VLSI
Université Pierre et Marie Curie
4, place Jussieu
75252 Paris Cedex 05
FRANCE

A. Amara
Membre du laboratoire MASI
Laboratoire de Microélectronique
Institut Supérieur d'Electronique de Paris
21, Rue d'Assas
75270 Paris cedex 06
FRANCE

Abstract

This paper deals with a GaAs data-path compiler for the public domain ALLIANCE CAD SYSTEM[1]. Based upon macro cells generators, cells libraries, and software tools, it provides a fast and easy way to describe and implement bit-sliced structures for DCFL logic. The use of over cells-routing, power supplies sharing and multiple access terminals coupled to a symbolic approach leads to a high density portable layout.

Introduction

Since GaAs is quite expensive, it is obvious to use a full-custom approach in order to save area. Of course this choice is at the expense of time and manpower. This can be solved by using a set of macro-cells generators and associated software tools. In that context this paper describes a layout compiler for regular structures.

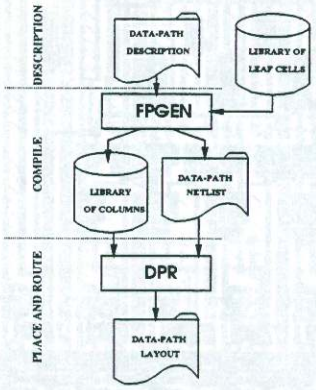

Figure 1: Data-path synthesis

As shown on figure 1 the designer have just to specify its data-path into a C code ASCII file and to run several tools to obtain the layout of the circuits. The first tool compiles the description and generates both the netlist and the physical views of the sub-cells in term of column of data-path. Those columns are optimized macro-cells (adders, shifters, register file, ...) or simple columns of standard-cells (NOR, NAND, MUX, ...). The second tool places the macro-cells and columns and performs over-cells routing.

The first part of this paper enumerates the topological constraints and then presents the set of generators and leaf cells. The next part shows the software aspect, from the specifications of the data-path to the placement and routing. To conclude, it presents some examples of data-path obtained with this approach and gives some conclusions.

Physical constraints

The portable libraries described hereafter have been designed for E/D MESFET processes with at least three metal layers. DCFL logic has been chosen to perform best compromise in term of delay, area and power consumption[2].

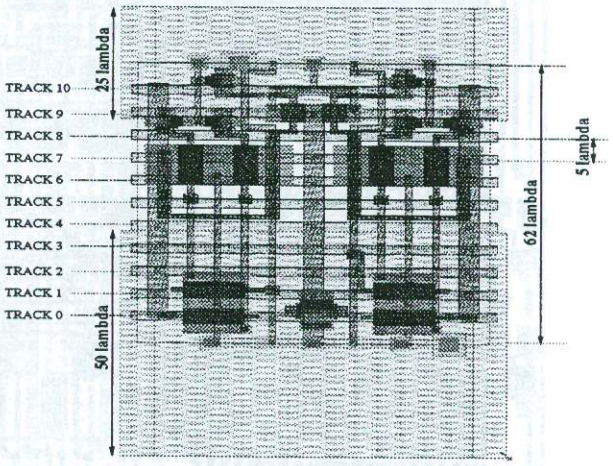

Figure 2: cell's topology

Internal cells wiring is performed with gate metal and METAL1, Over-cell wiring uses METAL2. METAL3 is dedicated to power supply lines. Two hierarchical levels have to be considered. At the cell level, the topological constraints are(see figure 2):

- ◇ Heigh of the slice: 62λ
- ◇ Tracks per slice for over-cells routing: 11
- ◇ Shared power supplies lines: respectively 50λ and 25λ
- ◇ Stretch line: 33λ
- ◇ Half design rule on WEST and EAST faces

At the block level (see figure 3):

- ◇ Data flow: Horizontal
- ◇ Command flow: Vertical
- ◇ Horizontal axis symmetry for odd slice.
- ◇ Two slices on the top of the data-path for control signals buffering.

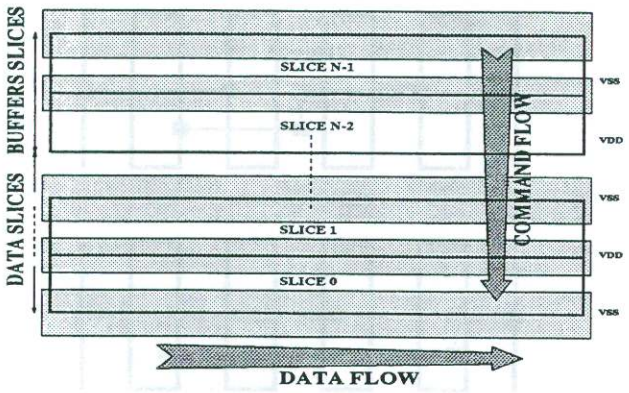

Figure 3: data-path topology

Cells and macro-cells library

This library includes the most commonly used cells and macro-cells namely adder, shifter, register file, multiplier and a large set of standard-cells for "glue logic". Each generator comprises the common features required by designers (see descriptions below). Those generators have been designed using a **tiler/leaf cells** semi-custom approach. Consequently, for each module, a dedicated library of leaf cells have been designed and a software has been carried out. Concerning those software, they have been written using C language with an additive set of layout dedicated functions called **genlib**[3]. Those functions work on a fixed symbolic grid and use pitch matching and abutment-based connections. The validation procedure of each module includes electrical verification, functional abstraction, formal proof, VHDL simulation and design rule checking (see figure 4).

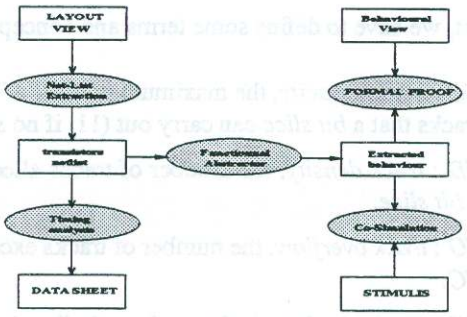

Figure 4: Validation procedure

CLARA: *Fast adders generator*

The generated adders implement the binary carry lookahead algorithm and can be used as a subtracter or a controllable adder/subtracter.

Parameters	Values
Operand size	From 4 to 64 by step of one
carry in	With or without
carry out	With or without
Over flow flag	With or without
Adder/Subtracter	With or without

RAGA: *Register file generator[4]*

The generated blocks use differential latches and pseudo-precharged busses and have one write and two read ports.

Parameters	Values
Size of words	From 4 to 32 by step of two
Number of words	From 4 to 128 by step of one
Number of reading ports	One or two
Decoder	With or without

SARA: *Shifter generator*

The circuit is based upon 3:1 multiplexors and performs logic and arithmetic shift.

Parameters	Values
Operand size	From 4 to 32 by step of two
Arithmetic shift	With or without

BAT: *Multiplier generator*

The generated block implements the modified Booth algorithm with an array structure.

Parameters	Values
Multiplicand size	From 4 to 32 by step of two
Multiplier size	From 4 to 32 by step of two

BUSY: *Buffer generators*

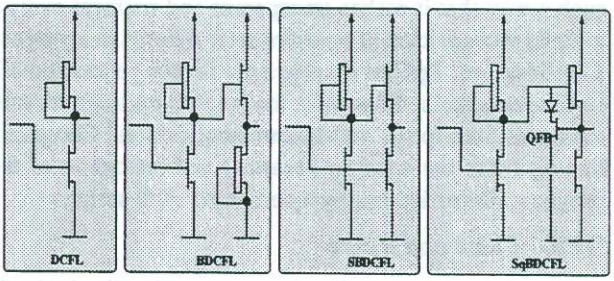

Figure 5: Different types of buffer

Buffering a DCFL circuit is a critical problem[5]. Unlike CMOS, Fan-Out is equivalent to a diode plus a capacitance. The good buffer should take into account both of them[6]. To face the huge number of electrical context that are likely to happen when designing a circuit, a buffer generator has been performed. From a given electrical context in term of gate width and wire capacitance, this generator chooses a single buffer or a buffer chain, computes the transistor width and finally generates the layout. Buffer type could be DCFL, BDCL, Super buffer(SBDCFL) and Squirt buffers(SqBDCFL)[7] (see figure 5).

Parameters	Values
Loading capacitance	From 0 to 500FF
Loading Fanout	From 1 to 64

Standard-cells library

This library contains a complete set of common single output gates. Boolean cells have several power capabilities. Those cells are planed to be stacked in order to compose a data-path column (see 1).

Table 1: Available standard cells

Boolean	inverter, nor2, nor3, or2, or3, nand2, and2, and3, xor2, xnor2
Mask	nand, nor, xor
Mux	mux 2:1
Misc	Zero detector, Constant generator, tri-state
Flip-Flop	With or without write enable, scan and reset

Software aspect

FpGen : *netlist capture*

The first step in the design of a *data-path* is the *netlist* capture. In order to make this design phases easier we have developed a data-path description macro-language, called **FpGen**, based upon the C language. The description of a *data-path* will take the form of a C program calling some particular functions dedicated to the *netlist* generation. Using C give us three advantages, first it reduces the learning time to a minimum, second we can take benefit of all the evolved features of the C language, and third it ensures us a wide portability. Two sets of operators are available. The first one is built upon a standard cell library. Table 1 summarizes its features. The second is based upon the macro-cell library : fast adder (**clara**), register file (**raga**), shifter (**sara**) and multiplier (**bat**). The **FpGen** language has been designed in such a way that it makes transparent and uniform every call to those various kind of operator sets. Always in a purpose of simplification, **FpGen** manages the layout generation of the various operators. Of course, the language allows vectorized terminals description and hierarchical design. Compilation, linking (with the specific libraries) and execution of the C program are carried out by a dedicated shell script.

DPR : *Place and Route*

The major drawback for data-path routing lies in the fixed number of available horizontal routing tracks (11, in our case). In order to optimize the use of those tracks, let us introduce the concept of *virtual terminal*. Moreover, in the case where a data-path will be intrinsically non-routable with only eleven tracks, a mechanism of stretching is available. This mechanism aims to add more tracks by *stretching* the *bit slices*. We also present the principle of the placement optimizer used in **DPR**. Detailed informations upon **DPR** can be found in [8].

Virtual terminals

The principle of a *virtual terminal* is to associate a set of physical terminals to a single logical one. Each *virtual terminal* corresponds to a place where the cell can be connected to, it gives the authorization to the router of putting a *VIA M1-M2* at this point. In the best case, eleven *virtual terminals* are associated to one logical terminal, that is, one per horizontal track. From the router point of view, all *virtual terminals* associated to the same logical terminal are considered as internally wired.

It can be pointed out that the addition of *virtual terminals* to cells has been done without increasing their area. This fact is due to the reuse of still existent material or the exploitation of spaces that otherwise remain empty.

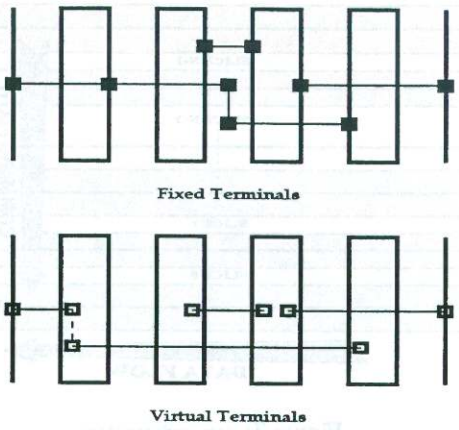

Figure 6: Fixed terminals and virtual terminals

To conclude, **DPR** will decide which *virtual terminal* to use to access the cell. This choice is done so as to favor wiring that require exactly one track (by choosing the appropriate *virtual terminals*), this approach suppresses or strongly reduces the size of the vertical routing channels. It is also easy to show that this mechanism aims to decrease the number of used horizontal tracks.

The mechanism of *virtual terminal* should be understood as a constraint carrying up from the cell level to the router level. In other words, this is no longer the cell that fixes the position of its terminals, but the router in agreement with the routing context.

Placement optimization

First, we have to define some terms and concepts :

- *TC* : *track capacity*, the maximum number of horizontal tracks that a *bit slice* can carry out (11, if no *stretch*).
- *TD* : *track density*, the number of tracks allocated over a *bit slice*.
- *TO* : *track overflow*, the number of tracks exceeding the *TC*.
- *WL* : *total wire length*, the total symbolic wire length.
- *OV* : *overflow*, a boolean value telling us if there is a track overflow.

Most of the columns are blocks made of stacked leaf cells, with a stacking order fixed by the *tiler*. So, the optimization problem is reduced to the choice of a column linear ordering. The placement algorithm behave on two different ways according to the value of *TD* with regard to *TC*. If *TD* is lower or equal to *TC*, the algorithm will exchange the two columns that minimize *WL*. At the opposite, if *TD* is greater than *TC*, the exchange will be done between the two columns that minimize *TO*. A formal description of the algorithm is given figure 7.

ObjF	Value of the objective function : *ObjF* := not(*OV*) × *TD* + *OV* × *TO*
M	Number of columns of the *data-path*.

Algorithm *OptiPlace*
begin
  Optimize := **TRUE**
  while (*Optimize* = **TRUE**) **do**
    Optimize := **FALSE**
    for (*jl* := 1 **to** *M*) **do**
      for (*jr* := *jl* + 1 **to** *M*) **do**
        compute(*ObjF*)
        if (*ObjF* < *MinObjF*) **then**
          MinObjF := *ObjF*
          exchange(*jl*, *jr*)
          Optimize := **TRUE**
        end if
      end for
    end for
  end while

Figure 7: Placement algorithm

Results

Several data-paths have been easily obtained with a high transistor density. Just to give a hint, let's consider an example.

Figure 8 shows the top of a 16 bits ALU. One can point out several details. The data-path assembly is based upon column abutment. Horizontal wiring is performed over the cells but vertical wiring needs channel routing. Each bit-slice shares power supply and odd ones are flipped. Power supply are strengthen with vertical lines. Control terminals are located on the NORTH face of the data-path. They are first buffered in the two dedicated slices at the top of the structure and are dispatched vertically throughout the data-path (see figure3 and 8).

Figure 9 presents the data-path part of a 32-bits RISC microprocessor. It includes glue logic and several optimized blocks such as register file, fast adder and shifter.

Conclusion

In this paper a data-path compiler has been presented. It is based upon a cells library, macro-cells generators and a set of tools to make easier the design task. We have pointed out some original feature such as virtual connectors, stretching capability, parameterized generators directly inferred from the compiler, and so on. This package has been successfully used to implement a RISC processor and other GaAs ICs.

Acknowledgments

Thanks to the whole CAO-VLSI team at MASI laboratory, and especially to P. Remy, A. Hajjard, F. Pétrot and F. Wajburt.

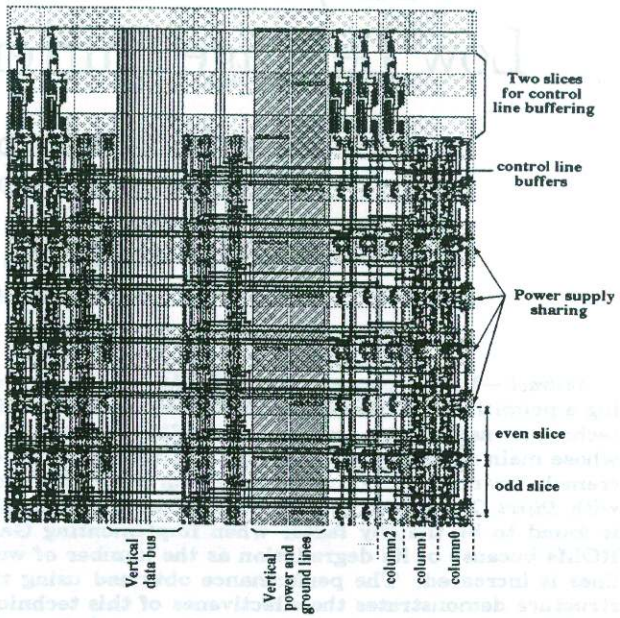

Figure 8: Zoom window on a 16-bits ALU

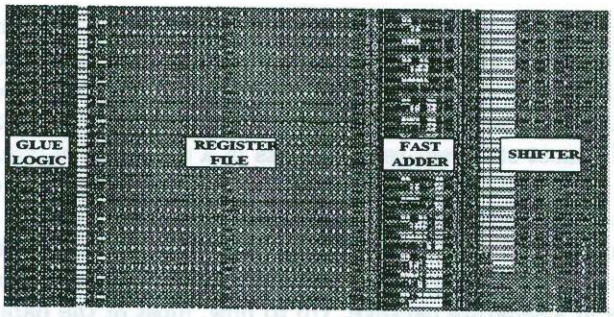

Figure 9: Zoom window on a 32-bits RISC microprocessor ALU

References

[1] A. Greiner, F. Pêcheux, "ALLIANCE: A Complete Set of CAD Tools for Teaching VLSI Design", *3rd Eurochip Workshop on VLSI Design Training, pp. 230-237, Grenoble, Sept. 1992, ftp.ibp.fr available in /ibp/softs/masi/alliance, 1993.*

[2] S. I. Long, S. E. Butner, *Gallium Arsenide Digital Integrated Circuit Design*, McGraw-Hill, 1990.

[3] Alain Greiner and Frédéric Pétrot, "Using C to Write Portable CMOS VLSI Module Generators", *The European Design Automation Conference EDAC'94*pp. 676-681, Sep. 1994.

[4] G. Hofmann, P. Royannez, A. Amara "A register file generator for GaAs data-path compiler" *Sixth International Symposium on IC technology, Systems & Applications, Singapore, September 1995(To be published).*

[5] S. I. Long, M. Sundaram, "Noise-Margin limitation on Gallium-Arsenide VLSI", *IEEE Journal of solid-state circuits, Vol. 23, No 4, August 1988.*

[6] S. J. Harrold, *GaAs IC Design*, Prentice Hall, 1993.

[7] *Foundry design manual-Version 6.0*, Vitesse Semiconductor Corporation.

[8] Lotfi BEN AMMAR and Alain GREINER, "FITPATH : A Process-Independant Data-Path Compiler Providing High Density Layout". IFIP Transactions, Synthesis for Control Dominated Circuits, G. SAUCIER and G. TRILHE Editors, P. 133-151, IFIP North-Holland, 1993.