RFG, A PORTABLE REGISTER FILE GENERATOR

L. Winckel, F. Wajsbürt, L. Lucas and A. Greiner
Laboratoire MASI/CAO-VLSI Université Pierre et Marie Curie
4, place Jussieu 75252 Paris Cedex 05 FRANCE

Keywords

Portable layout, register file.

Abstract

In this paper we describe a full-custom register file generator based on a tiling approach to achieve a high density integration. The leaf cells are designed using symbolic layout, providing a high degree of technology independence and portability. Co-simulation and formal proof ensures the validity of the tool.

Introduction

Register files are main component of almost ASIC. To obtain compact and efficient register file we have developed, in the framework of the IDPS Esprit2 project, a general purpose register file generator. This tool, named RFG, is now part of the public domain Alliance CAD system [1]. To be useful a register file generator must accept a wide range of parameters : the word length (from 2 to 64 bits), the number of registers (from 2 to 256), the number of read outputs (1 or 2) and some other parameters are provided to the users. Register file are based on static memory point to ensure a high reliability and high read performances.
We present the register file composition in Section Architecture then the layout constraints are describe in Section Layout Strategy. Section Parameters and Generated Views defines available parameters and views. Section Validation shows the validation procedure and Section Performances the generator global performances.

Architecture

A register file is composed of two parts : the decoder part and the registers part. This two parts are generated in a single layout block but are topologicaly separated.
The decoder part converts the write and read addresses into write and read orders for the registers. This conversion consist in the activation of the read or write signal corresponding to the given address. For the write decoder, an optional write enable signal conditions the output signals and a clock signal synchronize the output write orders. Data are written on the high level of the clock signal into the register. As a register file could have one or two read busses, the decoder part could have one or two read decoder. This part is made of full CMOS gate. Figure 1 shows the decoders part synoptic for one read bus.
The registers part contains an array of memory points (Figure 3), where memory point of a column have the same read and write command signals. Figure 2 shows the registers part synoptic for one read bus.
Memory points are based on an reversing device loop. Data are write into a memory point through the input pass-transistor who is only activated during write action. While writing a dynamic electrical conflict occurs between the input signal driver and the weak reversing device of the memory point. The input driver signal is designed to win

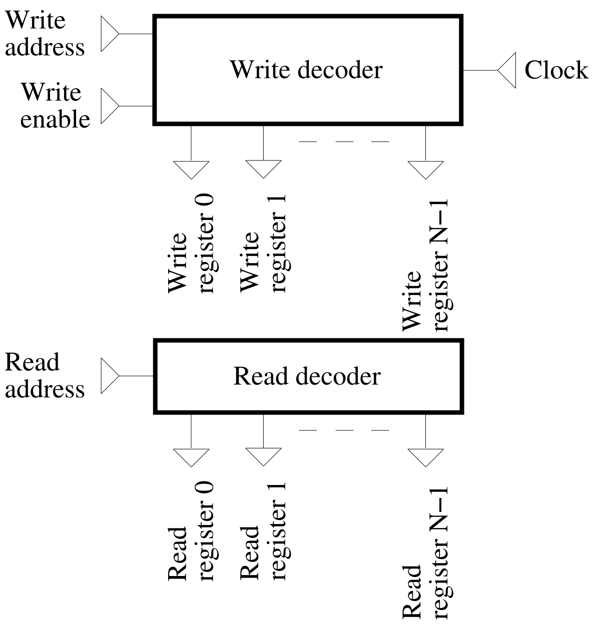

Figure 1: Decoders part.

this conflict. This conflict occurs only while writing a data who is different than the memory point data. Reading a data is done by a distribute pass-transistor multiplexor. Data are emit through an ouput pass-transistor on the output bus. Each memory point is composed of height for one read bus or nine transistors for two read busses. This design provides reliable, fast and compact memory points. Figure 3 presents a single memory point for two read busses.

Layout Strategy

Generated register file must comply with some topological constraints [2]. The leaf cells are designed to fit into a fixed height slice, or a multiple of the slice. Each slice corresponds to one bit information. The prime attributes of each cell are an abutment box, terminals, and respect of data path constraints, like a fixed number of second metal tracks for over the cell routing. The abutment box is used by the tiller to place the cells. This allows to share layout objects as contacts or body ties between neighbor cells. Virtual terminals [3] are used to indicate several equivalent tracks where the primary inputs and outputs can be connected by the router. The cells must offer several possibilities without increasing the size area. Internal terminals for validation and designer information are on the abutment box.
The tiling approach offers the high integration density and timing performances of custom designs. The price to pay is the complexity of the abutment scheme and the resulting important number of cells to satisfy all the configurations

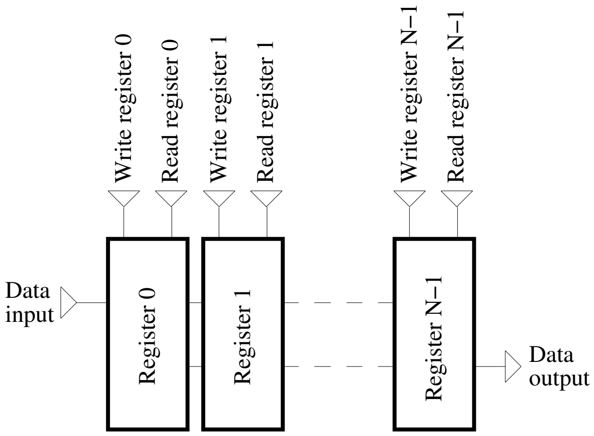

Figure 2: Registers part.

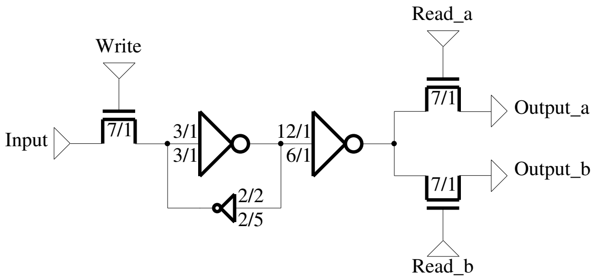

Figure 3: Memory point.

proposed by the generator. The register file library contains 57 leaf cells for 12 different logic functions. Figure 4 shows the cell layout with the abutment box represented by a dashed line.

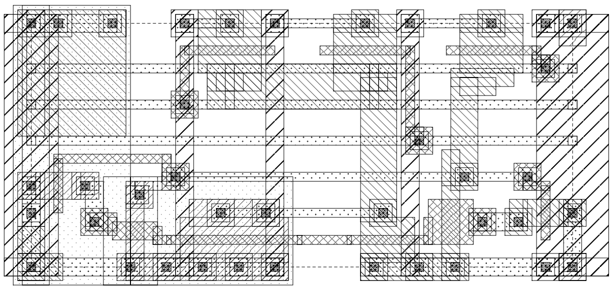

Figure 4: Memory point cell layout.

Layout is built using a symbolic layout method [4], making possible to have a same final layout that matches technologies ranging from 2μ to 0.6μ for a generic two metal and one poly CMOS process. This symbolic layout approach has been successfully used by BULL and UPMC/MASI during the last six years.

Parameters and Generated Views

The generator software development is done using the procedural layout language GENLIB [1]. It is a set of dedicated C functions, compiled in a UNIX library. The generator program is seen as a UNIX-like command. The output, when no parameters are given is the following :

```
Usage : rfg bits words busses ro|rs|ds|ba
        [-ff] [-id] [-lp] [-wel|-weh] [-stuck]
        [-virtual] [-msb0]
        [-layout] [-icon] [-vhdl] [-patterns]
        [-datasheet]
        [-physical_box] [-logical_box]
        [-o name_prefix]
```

The register file generator accept eleven functional parameters :

bits : Defines the number of bits of each register. Its value range from 2 to 64, but only even value are allowed.

words : Defines the number of registers. Its value range from 2 to 256, and as for bits, only even value are allowed. The even restriction on bits and words reduce the generator complexity and keep the leaf cell library to a reasonable amount of cells, without reducing the range of applications.

busses : Defines the number of read busses. A register file can have one or two separate read busses allowing the simultaneous reading of two different registers.

ro, rs, ds and ba : As a register file is composed of two part, a decoder part and a registers part, a parameter indicates which parts are to be include in the resulting block. The possibility are :

- **ro :** Width optimized register part with compressed command signal spacing.
- **rs :** Register part with standard command signal spacing.
- **ds :** Decoders part with standard command signal spacing.
- **ba :** Complete register file.

ff : The register file normally operates as a set level sensitive latches, but with this option it operates as edge triggered flip-flops.

id : With this option, the generated register file has complemented outputs.

lp : This option reduces the power consumption of register files who have more than 16 registers, but increases the latency of read operations.

wel and weh : The write decoder can have a write enable signal. The wel option adds a low level write enable signal, while the weh option adds a high level one.

stuck : For special applications, the register zero has to be set to the constant value of zero. This could be done using this option.

virtual : Allows the use of virtual terminals in the layout view, else only fixed terminals are used.

msb0 : Alters the vectorized signal numbering order. When used the most significant bit has index zero.

Seven different views may be generated for each set of functional parameters :

layout : One level hierarchical symbolic layout. Several output format are supported : cp (COMPASS©), mg (MODGEN [5]) and ap (ALLIANCE [1]).

icon : Symbolic representation for graphical net-list display.

vhdl : Behavioral description using a subset of data flow VHDL.

patterns : Stimulus and expected results for VHDL description.

datasheet : Miscellaneous information are displayed on standard output. This information includes the functionality, the size, the number of transistors, the input capacitance and the timing performances for a 1.0μ CMOS process.

physical_box : Interface layout representation needed by router. It contains the abutment box of the block, its fixed and virtual terminal location and feed-through tracks, as well.

logical_box : Interface logic representation needed by router.

Validation

Two kinds of errors can occur in a generated block. Symbolic design rule violations are the first kind and behavior errors are the second. The validation must check both kinds of errors. For the first one, we used a symbolic design rule checker from the public domain Alliance CAD system on layout files. For the others, we made comparison between an extracted net-list from the layout view and the VHDL description.
The flexibility of the register file generator allows a great number of parameter combinations. Several millions of different register files can be generated. Thereby an exhaustive validation is impossible. To eliminate the greatest number of errors, the validation procedure contains several stages.
The first stage concerns the validity of the leaf cells. For each leaf cell two files are manually designed. The first is a layout file and the second is a schematic file that represents the transistor net-list of the cell. The layout view of leaf cells are checked against design rule violations. A transistor net-list is then extracted from the layout and compared to the schematic view to verify the layout connectivity. This comparison is done by a net-list comparison tool from the COMPASS© CAD system.
The second stage verifies leaf cells layout compatibility. Blocks composed of a few number of cells has been defined. These blocks contain all possible abutments between neighbor leaf cells. They are checked from design rule violations to insure the respect of abutment constraints.
The third stage of validation checks generated blocks. After a set of representative register files has been selected, their layout, VHDL and pattern views are generated. The layout views are checked against design rule violations. Then, from the layout we obtain a transistor level net-list. This net-list is converted to logical equations in the VHDL format, by a functional abstractor tool [6] then a simulation with the generated patterns is done with the generated and extracted VHDL behavior. For the register files which have 2^n registers, a formal proof is done between both VHDL behavior.
All the validations are done under UNIX shell using makefile utilities and the public domain Alliance CAD system [1].

Performances

Table 1 shows run times on a SPARCstation 2 for the layout generation, the layout size, transistor number and read access time for some register files. The dimensions are given in microns for an 1.0μ CMOS process.

Bits	Words	Busses	Run Time (seconds)	Size (microns)	Transistor count	read access (ns)
8	8	1	0.8	240 x 840	784	3.86
16	16	1	1.1	334 x 1260	2640	4.75
32	32	1	3.2	835 x 2100	9852	5.88
64	64	1	19.4	1636 x 3780	36888	7.24
64	128	1	66.9	3239 x 3780	73264	-
64	256	1	245.6	6458 x 3780	146080	-
32	32	2	4.7	1003 x 2205	11656	7.27
64	256	2	301.7	7631 x 3885	169984	-

Table 1: Register file features

For current size of register files the run time is about some seconds and even for large register files the run time is not prohibitive. The density of generated blocks range from 3.9 to 6.3 Kt/mm^2.

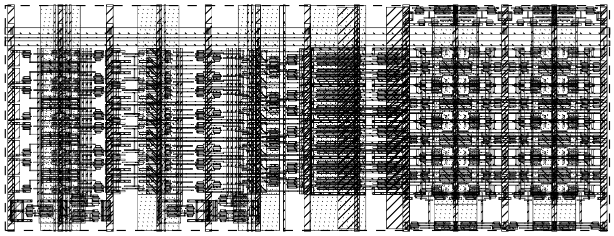

Figure 5: 8 registers of 4 bits with one read bus.

Acknowledgments

The authors would like to thank all the CAO-VLSI team at the MASI laboratory, especially, L. Ben Ammar and F. Pétrot.

References

[1] A. Greiner and F. Pêcheux, 'ALLIANCE: A Complete Set of CAD Tools for Teaching VLSI Design', in procedings of the 3rd Eurochip Workshop on VLSI Design Training, Grenoble, pp 230-237, Sept. 1992.

[2] L. Ben Ammar and A. Greiner, 'Automatic Placement for Bit-Sliced Datapaths', in procedings of the ICM'92, pp. 4.1.1.1-4.1.1.4, Dec. 1992.

[3] L. Ben Ammar and A. Greiner, 'A High Density Datapath Compiler Mixing Random Logic with Optimized Blocks', in procedings of the EDAC'93, pp. 194-198, Feb. 1993.

[4] A. Greiner and J. P. Leroy, 'A symbolic Layout View In Edif For Process Independant Design', in procedings of the Fourth European Edif Forum, Daresbury, Cheschire, Oct. 1990.

[5] H. Janssen, Modgen and graphMG User Manual, Philips Research Laboratories, Eindhoven, The Netherlands, 1991. Modgen has been chosen by the IDPS ESPRIT project as generator exchange language.

[6] A. Greiner, M. Laurentin, R. Marbot, 'DESB, a Functional Abstractor for CMOS VLSI Circuits', in procedings of the EuroDAC, Hamburg, Sept. 1992.