AN EFFECTIVE HARDWARE/SOFTWARE SOLUTION FOR FINE GRAINED ARCHITECTURES

Gareth J. Jones
Darren M. Wedgwood

Pilkington Micro-electronics Limited
Gadbrook Business Centre,
Northwich, Cheshire, CW9 7TN, UK

Abstract

This paper describes the development of a new fine grained sea-of-gates FPGA architecture. The architecture has been developed simultaneously with a proprietary set of autolayout tools and has been designed to be sympathetic towards these tools. New structures have been added that aid the autolayout process and the interaction of these structures with the software is explained. In addition the paper describes some of the problems traditionally associated with the fine grained approach and looks at how these have been addressed in this new architecture.

INTRODUCTION

The only way to measure the quality of an architecture is by its performance in real applications and as such the architecture is only as good as the software that was used to implement the design. Although this is fairly obvious it presents a number of problems when developing new FPGA architectures. Effectively a full set of layout tools must exist before the architecture has been defined. In order to take this into the development process, PMeL have developed a suite of architecture independent layout tools. These tools allow the designer to quickly define architectures using a device description language. The device description contains all of the structural details of the array, along with any necessary timing information. A set of device independent algorithms were developed that would work from the device description. These algorithms are fully timing driven and take all of their information from the device description. They contain no device specific code. An outline of the design flow for these tools is given in figure 1.

The aim of the program was to take PMeLs original architecture [1] and refine it using the architecture development tools described. This process would provide an architecture that worked in harmony with the software solution rather than an architecture around which a software solution had to be built. If the tools cannot quickly adapt to architectural changes then

the tendency is to rely on the software to overcome architectural limitations as further modifications to the architecture have a considerable time penalty.

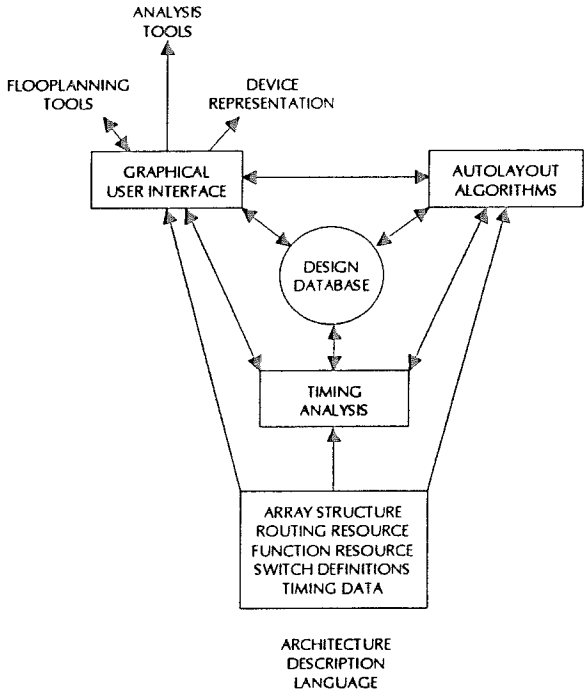

Figure 1. - Architecture Development Tools

With the existence of the development tool suite it was practical to evaluate and benchmark a potential architecture in a reasonable time. For incremental changes, a fully working autolayout system could be generated in less than a day. During the course of the development of the new architecture (TS1) over 20 architectures were fully evaluated. Evaluation was carried out using a combination of in house designs and a standard benchmark suite. All evaluation was carried out on a model of a 10K cell array. Although the focus was on making the hardware more sympathetic to the software, the suite of layout algorithms was continuously enhanced and extended throughout the program. For instance as the performance of the routing paths became more consistent the timing driven algorithms were adapted to take advantage of this by using estimates in place of calculated routing delays and updating these estimates with actual calculated figures on an occasional basis.

PMeL's original array was a flat homogeneous sea-of-gates array of simple cells (NAND and Latch function). Early work with this architecture had shown good performance implementing manually laid out designs. The efficient nature of the fine grained cell allows critical paths to be highly optimised and the compact nature of the cell means that routing distances between closely connected cells are minimal. Manual layout is, however, only practical for regular designs. Manual optimisation of a layout for random logic is not efficient and is not practical for large designs. Attempts to produce a fully automatic layout solution for this architecture revealed a number of inherent difficulties in providing such a solution. It was in an attempt to find a solution to these problems that the TS1 development program was set in motion.

The following document describes some of the problems encountered when attempting to provide a software solution for the original array and describes the architectural modifications implemented to overcome them. First, let us examine PMeL's reasons for choosing a fine grained approach to FPGAs.

THE FINE GRAINED ARCHITECTURE

There have been a number of papers demonstrating the efficiency of large logic blocks, both in terms of performance and area [2][3][4]. This work generally assumes that the proposed architecture is implemented using a general channelled interconnect structure. The advantages of a small cells are only apparent when using an interconnect structure tailored for such an array. Fine grained arrays obviously require more cells to construct large functions, In order for these architectures to perform it is essential that the critical routing between cells is minimised. Due to the compact nature of the basic cell in a fine grained array and the close proximity of cells in a sea of gates array, the routing overhead for local cell connections is negligible. By providing restricted interconnect for these local connections, a special level of high performance routing can be provided. This interconnect allows multilevel functions to be constructed efficiently from basic cells. The limited number of inputs to a cell also means that a useful number of cells can be covered without excessively loading the line. The local interconnect for TS1 is shown in figure 2 and remains very similar to that of the original architecture.

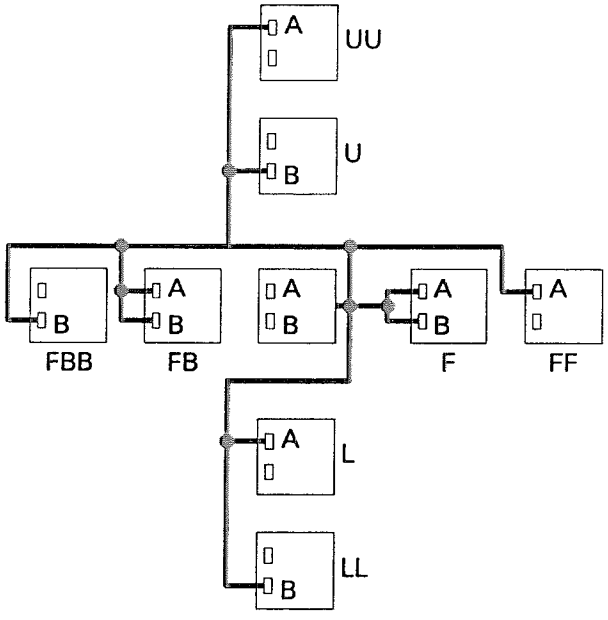

Figure 2.- Local Interconnect

The PMeL array has three levels of interconnect. The high performance short range interconnect allows larger functions to be generated locally, these local functions are then connected together using medium and global interconnect as if they were larger logic block of mixed sizes. By providing a high performance set of interconnect at the local level, larger functions can be constructed efficiently without any routing penalty. As the size and complexity, i.e. number of inputs, of the basic logic block increases such a routing structure becomes less efficient as the scope and loading of the interconnect increases.

When considering the performance of the basic cell it is necessary to take account of the range of functions to be supported. The low level design of the basic cell plays a significant role in determining the overall performance of the array and the techniques used to implement the cell are dependant on the level of functionality to be supported. For example Xilinx use look up tables, Actel use multiplexers and PMeL use fixed functions. These differences in implementation can cloud the theoretical comparison of architectures, but are non the less vital when considering the practical design of an array.

Table 1 shows a comparison of the theoretical minimum delay through several arbitary logic functions. These functions have been implemented using a 4 input look up table and the TS1

logic block, it is assumed that routing delays are zero. The performance of the 4i/p logic block was estimated by comparison with commercially available products [5]. The performance of the TS1 block was simulated for a 0.8μ process. A number of possible logic levels for each n input function were considered.

No. of Inputs	No. of Levels	4 I/P lut (ns)	TS1 (ns)
2	1	4.5	1.2
3	2	4.5	2.4
4	2	4.5	2.4
4	3	4.5	3.6
5	3	9	3.6
5	4	9	4.8
6	3	9	3.6
6	5	9	6
7	4	9	4.8
7	6	9	7.2
8	4	9	4.8
8	7	9	8.4
9	5	13.5	6
9	8	13.5	9.6

Table 1. - Low Level Function Performance

Table 1 illustrates some of the advantages and disadvantages of a fine grained architecture. It can be seen that the larger logic block only obtains a comparable performance when the full capacity of the block is utilised. Similarly silicon area efficiency is only achieved when a high proportion of the blocks capacity is being used. In order to achieve results that bear out this in practice, the autolayout tools for the TS1 architecture must take advantage of the available (negligible delay) local connections and use them to implement functions on the critical path. Efficient use of the available resource by the autolayout tools is obviously critical to the performance of the TS1 architecture.

THE OBJECTIVES

From work with the earlier architectures, a number of key objectives for the TS1 development were identified. These represented targets for an integrated hardware/software solution. The original architecture had certain characteristics which provided obstacles to these goals. These had to be addressed in order to provide a competitive product. The solutions that were found to these problems are described in the next section. Here is a list of the key goals during the course of the project.

Efficient Mapping

The mapping problem should be as straightforward as possible. This eliminates the need for a sophisticated intermediate level of software which may introduce inefficiencies and errors. Mapping to a limited set of basic functions also allows the efficient use of existing ASIC synthesis tools.

Reduced Data Size

The small size of the basic cell results in the design being decomposed into a large number of elements. It was necessary to reduce the data set to a size similar to that produced for competitive architectures in order to provide a practical software solution. An excessively large

data set results in long run times and high memory requirements. This is particularly significant when producing software for the PC platform.

Consistency

If the user cannot predict the likely autolayout runtime or if the completion rate is erratic then they will quickly become frustrated. It was considered necessary to provide software that behaved predictably for designs within the claimed capacity of the device. This also requires the architecture to be reasonably tolerant to variations in user design style.

Performance

The layout software had to be fully timing driven, without the need for the user to specify any timing criteria. The software must, therefore, provide an intelligent guess at the designs potential performance. Although Performance was the key criteria, it was necessary to ensure that targeting a reasonable set of performance criteria would not impact the utilisation.

Efficiency

Every attempt had to be made to keep the silicon efficiency as high as possible. Providing an excess of routing resources would obviously simplify the autolayout process, but if carried to extremes would make the array impractical. A compromise was necessary between the level of resource and the cost of the array. The ability to quickly iterate architecture variants was essential in obtaining a good solution for this.

THE IMPLEMENTATION

Following are the fundamental changes made to the PMeL architecture during the TS1 development program. These are only the modifications that effect the basic principals of the architecture. Numerous other changes were made, particularly at the transistor level in order to improve performance, however these are outside the scope of this document.

Mapping

The original PMeL array could perform 3 basic functions, a latch, a NAND gate and by default an inverter (NAND with one input tied high). These functions allowed all possible logic functions to be implemented. There were, however, inefficiencies in the implementation of certain functions. The inverter was wasteful of the cells function as only one input was used and functions such as an OR gate could require up to 4 cells to construct if no optimisation was possible. The latch cells could be combined to form a D-type, but they had to combined in a strict manner. This resulted in a fixed two cell macro, the use of which made the autolayout task more complex.

Other functions such as the XOR required awkward combinations of elements that proved inefficient to layout. It became obvious that a more efficient set of primitive functions was required. These would define a library of basic elements, implementable within a single cell and suitable for synthesis. The required set was defined as follows;

D-TYPE FLIP-FLOP

LATCH

AND

XOR

TRISTATE

INVERTER

Any design can be efficiently mapped into the above set of functions using standard gate array synthesis tools - no sophisticated mapping is required.

The Heterogeneous Array

Having determined a set of target elements, it was necessary to efficiently implement them in silicon. One of the basic advantages of a fine grained array is that there is little redundancy in the cell. If a function is mapped to a cell then it will generally use all of the cells inputs and outputs and most of the cells functional capability. Therefore, as the set of basic functions was being extended it was essential to keep the individual cells as minimal as possible. The method chosen to do this was to distribute the functions required over a group of cells. In order to do this a study was made of the general frequency of occurrence of logic functions and the general arrangement of key functions. The gate array database of a major silicon vendor was analysed to produce the relative frequency statistics shown in table 2.

Function	Frequency
Register	1
XOR	1
AND,etc.	5

Table 2. - Function Frequency

Based on these figures a tile of four cells was chosen. It was important that the array did not become targeted to this ideal design so these figures were only used as a basis indicating where savings may be made. The functions were distributed in the following manner. All cells were given the basic AND function as a primary function. This allows all cells to be used for routing and gives greater positional flexibility to the most common function. In addition each cell was given a subsidiary function. These functions are the XOR, D-type/Latch and Tristate.

The arrangement of subsidiary functions was determined by looking at commonly used functions which may be implemented as high performance hard macros (i.e. counters, adders) and providing a suitable arrangement to aid implementation of such layouts. At this stage the ability to link selected cells to produce a fast carry chain was included. This special chain was designed to fit in with the macro structures previously defined. The result of this work is a 4 cell tile as shown in figure 3, which is then repeated throughout an array in much the same way as the single cell of the original architecture. The 4 cell tile allows the construction of stackable macros for a variety of 1bit counters and adders.

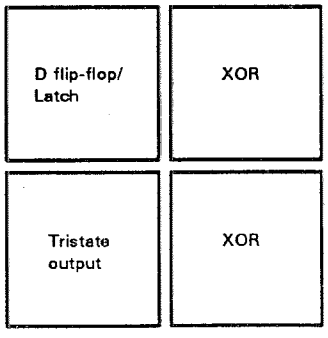

Figure 3. - The Four Cell Tile

The use of whole cells to produce inversions in a signal had proven very inefficient in the past as the layout tools had to track the sense of a net and add or remove inversion cells as

required. The solution was to provide programmable inversions on the inputs of all cells. The sense of a net could then be determined without modifying the structure of the layout. This implementation of inversions provides significant performance advantages over using additional cells, as well as the obvious utilisation advantages and software simplification. The AND function can still be used to route signals if required.

The Partitioned Array

One of the major problems in writing layout software for fine grained arrays is the size of the data structures, due to the design being broken down into its minimal components. The constrained nature of the available routing resources tends to preclude a traditional place and route approach to the problem. For a fine grained (particularly sea of gates) architecture, if a section of layout is completed, this will not only constrain the placement of other elements but will also constrain the routing of global signals through the completed area. If allowance is made for this and routing space is left this inevitably leads to inefficient utilisation. Similarly, if the global routing is completed first then this will constrain the placement of elements and the local routing. Making allowances for this will again lead to inefficiencies. Therefore the only efficient method is to solve the entire placement and routing problem for the whole array simultaneously, this is a very big problem. What was required was an architecture that allowed a partial solution without compromising the efficiency of the array.

The solution to this problem was to take the uniform array and apply a partitioning layer. Separating the array into two distinct levels of hierarchy. The partitioning is performed by port cells, which separate the global interconnect from the medium and local interconnect. This partitioning layer is applied directly over the original flat array to produce a partitioned array, see figure 4.

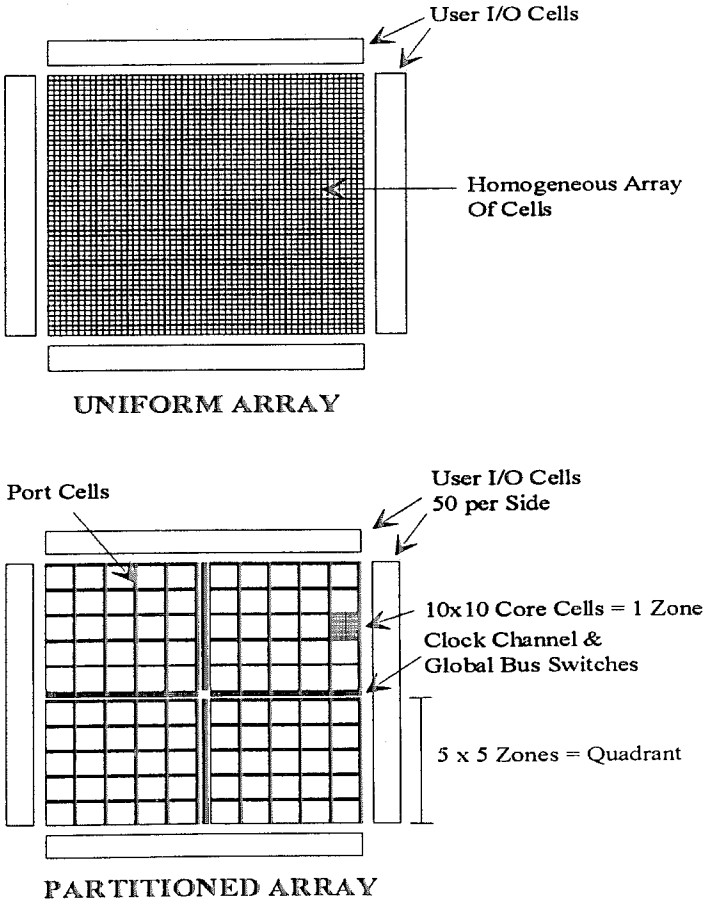

Figure 4 - The Partitioned Array

The array cells now only connect to local and medium range interconnect. Connections to the global interconnect network can only be made through the port cells. The partitioned array is now built up from a repeated pattern of 'zones'. Each zone consists of an array of ten by ten cells, with port cells at the edges to interface to the global interconnect network. This produces a hierarchy of routing resources as shown in figure 5.

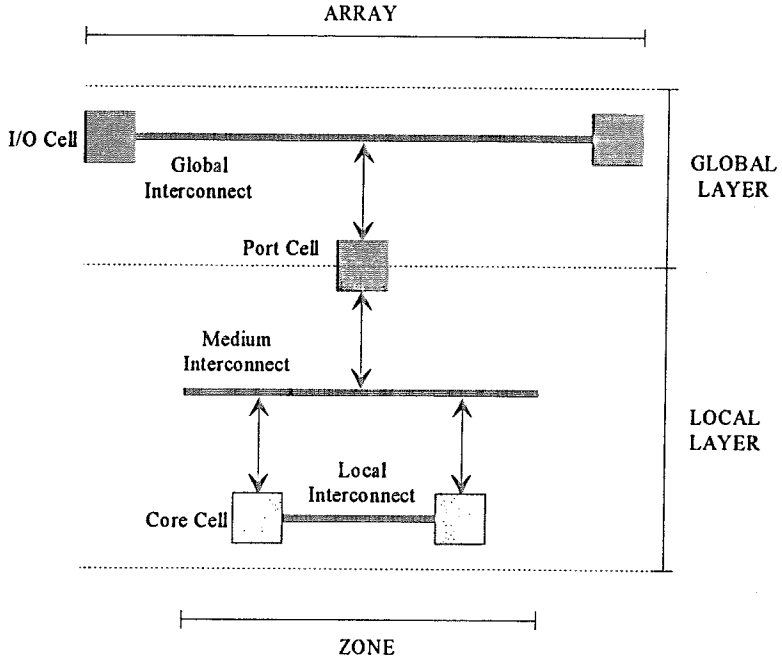

Figure 5. - Resource Hierarchy

The software solution can now be divided into two distinct areas, global placement and routing and local placement and routing. The introduction of a fixed interface between global and local levels in the hardware not only helps to simplify the software development task, but also improves the device performance.

Because the array cells no longer connect directly to the global interconnect network, it has been possible to extend the available routing resource at the local and medium levels (which connect directly to the array cells) without significantly increasing the cell dimensions. This increased wealth of routing resource at the local level simplifies the autolayout task at this level, leading to improved utilisation and performance (less cells are used as routing resources). Similarly, because the port cells only connect to medium and global interconnect, It is possible to extend the global interconnect without over complicating the port cell. Once again this simplifies the software task. As global buses now only connect to every tenth cell, the loading on these long lines is much reduced over the previous architecture, providing further performance improvements.

Once a partitioning structure exists the layout problem can be broken down into sub problems. The initial design is partitioned into the various zones. The enhanced global routing structures make the inter zone delay penalty highly predictable allowing accurate timing driven partitioning. Once partitioned the global routing can be completed without the need to carry out any of the lower level layout. Each of the zones can then be laid out independently. This results in a problem size of up to 60 elements per zone. In computational terms it is preferably to use a divide and conquer approach where there are a 100 of these sub problems to solve rather than a single problem of 5-6K elements. Some of the algorithms within the tool suite have a complexity order of N^3, therefore the benefits of this approach are obvious.

Timing Issues

The entire software suite, from the initial partitioning algorithms and global routing, down to low level placement and routing algorithms, is all timing driven. The user is able to exercise some level of control over the performance of his circuit (or even just critical parts thereof) even when using the full autolayout capabilities. If the user does not specify a required performance then the software will attempt to predict the optimal target performance.

The initial partitioning algorithm is timing driven and is able to give a good estimate of the likely performance of the completed layout. This can save the user the frustration of waiting until the entire layout has been completed before discovering that the autolayout has failed to meet the given timing constraints. The accuracy of the initial timing estimate is due to the partitioned nature of the architecture. Once the partitioning breaks the design into zone sized groups an initial timing analysis can be performed to provide target delays. Global routing can then be performed using these targets as a guide. Once the global routing has been completed accurate timing constraints for each zone can be calculated. At this point the global routing will not change so as long as the layout within each zone meets its constraints the initial target will be met. It has been found that the performance of layouts over this area (10 x 10 cells) is highly predictable. The constraints on each zone are continuously monitored to ensure that they are feasible. These practical constraints along with the small size of the problem at the zone level ensures that the layout tools are able to implement the low level critical functions with zero or minimal routing delay.

Skew, particularly on high fanout and clock nets is also a common problem when designing with FPGAs. To reduce this problem an 8-bit wide global clock bus has been provided on chip. This clock bus may be driven directly from I/O pads or from internally generated signals and is guaranteed to have minimum skew (<1ns) to any point in the array. This network can also be used to distribute normal high fanout nets if they are critical. In addition to this network the global routing algorithm is able to construct balanced high speed and low skew routing structures from the architecture description using standard interconnect and allocate these to user specified nets.

DESIGN METHODOLOGY

Because the functional cells in a fine grain array are so simple, e.g. two input NAND, there are few specific design methodologies to follow in order to obtain optimum results from the device. The designer is free to design using low level functions or to use a high level design language such as VHDL. The primitive functions available on the device are straightforward to synthesise to. Synthesis tools have freedom to minimise the function on critical paths without the need to worry about under utilising the cells. As previously mentioned, when using a coarse grained device it is often necessary for the designer to bear in mind the available functionality within each cell and to design his circuit in order to ensure that each cell is maximally utilised.

As each cell has a primary and secondary function, the array can be treated as 100% combinatorial or 25% register 75% combinatorial. This means that the arrays performance remains consistent for purely combinatorial or heavily register based designs. This is helped by the extensive hidden clocking network which removes the burden of routing additional clock/reset signals.

CONCLUSION

The problems traditionally associated with fine grained architectures do not necessarily stem from the architecture of the hardware, more significant has been the lack of an effective software solution. This paper has shown that by developing a fine grained sea of gates FPGA architecture simultaneously with the supporting software tools, and by designing into the architecture features which make the hardware sympathetic towards the autolayout tools an effective hardware/software solution can be achieved.

REFERENCES

[1] Plessey Semiconductors, "ERA60100 Electrically Reconfigurable Array - ERA." Data Sheet August 1990.

[2] Jack L. Kouloheris *et al.*, "FPGA Performance versus Cell Granularity," *Proceedings of the 1991 Custom Integrated Circuits Conference.*

[3[Jonathan Rose *et al.*, "The Effect of Logic Block Functionality on Area of Programmable Gate Arrays," *IEEE Journal of Solid-State Circuits*, Vol. 25, No. 5, October 1990.

[4] Satwant Singh *et al.*, "Optimisation of Field-Programmable Gate Array Logic Block Architecture for Speed," *Proceedings of the 1991 Custom Integrated Circuits Conference.*

[5] Xilinx, Inc., Programmable Gate Array Data Book, 1993.

[6] Stephen Beavis, "Design Flow for the Development of New FPGA Architecture," *IEE Colloquium on Field Programmable Gate Arrays*, London, February 1993.

[7] Malkit Jhitta, "Introduction of a New FPGA Architecture," *Proceedings of then 1993 FPGA Workshop*, Oxford, September 1993.

[8] Fred Zlotnick *et al.*, "A High Performance Fine-Grained Approach to SRAM Based FPGAs," *Westcon*, S an Francisco, September 1993.

[9] Actel Corp., ACT Family Field Programmable Gate Array Databook April 1992.