The Design of a New FPGA Architecture [⋆]

Anthony Stansfield[1] and Ian Page[2]

[1] SRF/PACT 10 Priory Road, Bristol, BS8 1TU.
(Formerly of Oxford Parallel, Parks Road, Oxford OX1 3QD.)
[2] Group Leader, Hardware Compilation Research Group, Oxford University Computing Laboratory, Parks Road, Oxford OX1 3QD.

Abstract. A new Field Programmable Gate Array (FPGA) architecture is described. This architecture includes a number of novel features not found in currently available FPGAs. It is believed to offer a significantly improved logic density in some common applications.

This paper describes the development of the FPGA and looks at the mapping onto it of some interesting application circuit elements. The design is discussed in approximately chronological order which allows us to explain other options which were considered and rejected during the development process.

1 Introduction

In late 1992 Oxford Parallel set out to develop a classification scheme for FPGA[3] architectures, with the ultimate aim of identifying the most appropriate architectures for use with the hardware synthesis tools previously developed in the Oxford University Computing Laboratory [1]. As a result of this work it became apparent that many FPGAs available at the time were broadly similar to each other from the point of view of suitability for use as a synthesis target, and in particular they all had similar weaknesses. Further work produced a way of modifying an FPGA to improve its usability and efficiency in this kind of application, and therefore in mid 1993 work started on designing an FPGA to include these improvements. The design was completed in early 1995, and the first silicon became available in late June 1995.

[⋆] This work was undertaken by Oxford Parallel, which is part of Oxford University Computing Laboratory. The work was initially funded by the Department of Trade and Industry under the Parallel Applications Programme, with the support of Inmos Limited (now part of SGS-Thomson Microelectronics Limited). Since May 1994 it has been supported by ESPRIT project P8202 (CoCo).

[3] Throughout this document the term FPGA is often used to refer to all types of field-programmable logic, both 'Traditional FPGAs' (of the Xilinx/Actel/Altera type) and also those more commonly referred to as PALs, PLAs and CPLDs (Complex Programmable Logic Devices)

2 FPGA classification

Oxford Parallel's primary interest was in FPGAs as a hardware synthesis target, especially for use with the methods of synthesis from high-level programming languages developed within the Computing Laboratory[1]. Manual programming via schematic capture and manual placement or routing was of secondary interest.

The first stage of this work was to identify the kinds of hardware structures generated by the synthesis tools, followed by a consideration of how these structures could be implemented in an FPGA. This resulted in a list of the features that would ideally be present in an FPGA to be used with the synthesis tools. Finally a number of commercially available FPGA designs were compared with this list.

2.1 Hardware Generated by the Synthesis Process

The Computing Laboratory's synthesis approach is based on a direct translation of statements in the source program to hardware elements. The individual elements of the program are converted into groups of logic gates, and then the overall circuit is assembled from these groups in a manner which directly reflects the structure of the original source program. An outline of the mapping between program statements and hardware elements is given in Table 1.

Table 1. Mapping of Program Elements to Hardware Elements

BOOL a:	(variable declaration)	Create a register holding value of ‘a’.
INT [128]b:	(array declaration)	Create memory array.
c := d	(assignment)	Register update
b[3]:	(array reference)	Access memory array
x + y	(addition)	Create a row of full adder cells
	(similar rules for other arithmetic operators).	
c = d	(comparison)	Bitwise XOR; OR gate to merge results.
x >> 2	(right shift)	Reorder the wires within a bus.
SEQ	(sequential composition)	Connect up control circuits in series.
PAR	(parallel composition)	Connect up control circuits in parallel.
WHILE...	(loop)	More complex control, with looping.
IF...	(conditional branch)	Pass control to either substatement.
CASE...		Pass control to particular substatement.

2.2 Implementation on an FPGA

The circuits referred to above can be implemented on an FPGA as follows:

– Variables map directly to registers.
– Arrays can be made using dense memory elements within the FPGA rather than by using large numbers of registers.
– The arithmetic and logical operators (add, subtract, bitwise AND, OR etc.) require 2- or 3-input gates. Note that individual gates can become functionally complex - the sum and carry paths of a full adder cell for example.
– Shifts by constant amounts can be handled in the routing, they need no logic gates.
– Comparisons (equal, not equal) are a two stage process - firstly an exclusive OR of the individual bits in a word with the pattern, and then an AND or OR gate to combine the individual results. The number of inputs to this second stage gate can be quite large (up to the length of a word: 8, 16, 32... bits)
– The program constructors (SEQ, PAR, WHILE, etc.) determine the flow of control within the circuit. They typically use one-hot, decentralised, encoding to minimise routing requirements.
– Conditional branches (IF, CASE) can be implemented as a series of independent comparisons, one for each branch of the IF or CASE. Alternatively, CASEs (and also nested IFs with closely related conditions) can be implemented with an n-input, 1-of-2^n decoder.

From this, the following list of useful features for an FPGA can be developed:

1. An efficient implementation of small, complex, logic gates. ie: up to around 3 or 4 inputs, functions such as Exclusive OR, Sum, and Carry.
2. An efficient implementation of large, simple logic gates. ie: more than 10 inputs, but mainly AND and OR functions.
3. Registers, to be used for the registers to hold variables, and a good clock routing scheme, so that the registers can be run at high speed.
4. Space-efficient (ie: dense) memory within the FPGA.
5. Flexible routing structures, for wiring up the control and data paths, and able to handle shifts/rotates etc. Ideally, there should be sufficient routing resources to give a high utilisation of the logic resources.

Since the main interest was in FPGAs coupled with logic synthesis and automatic place & route, a further item was also added to the list:

6 The FPGA should have a regular structure, with as few special cases as possible.

This idea is borrowed from RISC microprocessor and compiler projects, which have concluded that having a very regular instruction set makes it easier to write and maintain the compilers and optimisers. There is no reason to suppose that the same is not true for FPGAs.

2.3 Comparison with Existing FPGAs

Table 2 is a summary of how a number of FPGAs available in 1992/3 compare with this list of features[4].

Table 2. Suitability of FPGAs as a compiler target.

	Xilinx 3000	Xilinx 4000	Actel 1280	Algotronix	Concurrent Logic	PAL PLA	CPLD (multi-PLA)
Small gates	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\times$	$\times$
Large gates	$\times$	(1)	$\times$	$\times$	$\times$	$\surd$	$\surd$
Registers and Clocks	(2)	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
Memory	$\times$	(3)	$\times$	$\times$	(4)	$\times$	$\times$
Interconnect	$\surd$	$\surd$	$\surd$	$\times$	$\surd$	$\times$	$\surd$
Regular architecture	$\surd$	(5)	$\surd$	$\surd$	$\times$	(6)	(7)

1. Up to 9-input AND/OR gate in 1 cell (ignoring special edge decoders).
2. Not an enormous number of registers available.
3. LUTs can be used as small RAMs (5 bit address). Bigger RAMs still hard.
4. Has a large number of registers, so RAMs could perhaps be emulated.
5. Regular arrangement of cells, but the large cell has many (non-orthogonal) options, which could complicate software support.
6. Locally regular. PLAs large enough for complete systems not available.
7. Individual blocks are very regular, but there would be a discontinuity in the implementation of a circuit as it grew to be just larger than one block.

A number of conclusions followed from the construction of Table 2:

- None of the available devices met all the requirements (although the Xilinx 4000 family came close).
- The distribution of ticks and crosses in the table seems to indicate that most of the devices have similar good and bad points. In particular:
  - They all have provision for registers and good clock distribution.

[4] Note that a number of the companies listed in the table now have different names. For example, Algotronix is now part of Xilinx and Concurrent Logic is part of Atmel.

- None (with the possible exception of the Xilinx 4000) has good support for embedded memories.
- All are good at either small or large gates. None is good at both. Traditional FPGAs are good for small gates, while the PLA-based devices are good for large gates but not the smaller ones, especially for more complex functions. The Xilinx 4000 series has limited support for PLA-style logic in special peripheral units.

3 The New FPGA design

After the foregoing analysis, our assessment was that available FPGAs had weaknesses in the areas of on-chip memory and the ability to combine support for large and small gates. We then tackled the question of whether there was an easy way to improve performance in these areas.

3.1 A Heterogeneous FPGA

One way would be to construct a hybrid system. Given that PLAs are good for large gates (and complex inter-related functions), while traditional FPGAs are better for small, complex gates, a device containing both PLA-like and FPGA-like elements might provide the advantages of both. Unfortunately, this approach has a major disadvantage, which can be summarised in the following questions:

1. What should be the ratio of PLA-like to FPGA-like elements?
2. Is it best to have just a few, large blocks of each type, or alternatively more, smaller, blocks?

It is easy to produce cases to justify answers to (1) ranging from 90%:10% one way to 10%:90% the other, and also to justify both the large and small blocks options. There appears to be no single, universally good, pair of answers[5]. Because of this problem, the hybrid approach was not followed any further.

3.2 CAM-based FPGA

Having failed to find a good basis for a heterogeneous array, we became convinced that only a homogeneous solution would meet all necessary criteria. Knowing that PLA-style logic simply had to be supported we began looking at dense array styles. The key insight was enabled because of the background of one of the authors[6] in the design of CAMs for cache memory applications.

Our concept of a CAM-based FPGA was based on the observation that a Content-Addressable Memory (CAM) has a similar structure to the AND-plane

[5] This is hardly surprising, as the extremes represent the pure PLA and pure FPGA cases, and there is sufficient commercial demand to justify the existence of both types of device.

[6] Anthony Stansfield

of a PLA. It is also compatible with Static RAM (SRAM) technology used to make the LookUp Tables (LUTs) in FPGAs such as the Xilinx 3000/4000 and Altera FLEX devices. The basic idea is to replace the LUT SRAM in such a cell with a CAM, in order to create an FPGA cell that can be used either as a LUT or as a PLA element.

A basic 10 transistor, 1-bit, CAM cell is shown in Fig. 1 and has 4 vertical and 5 horizontal bus lines. It can be seen that this is comparable in complexity to a static RAM cell, but the additional i/o lines allow the content addressability.

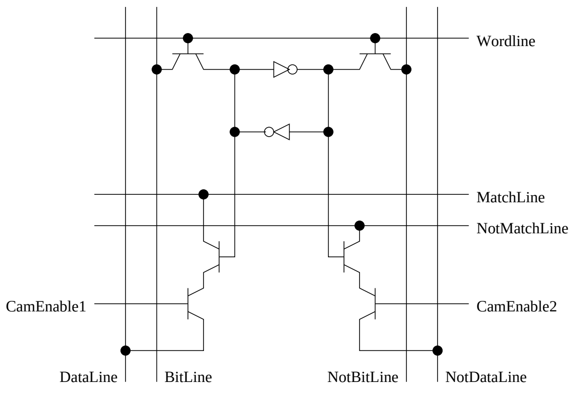


Fig. 1. The CAM cell

After some investigation, we determined that the best size of CAM array appears to be one with 16 bits, organised as a 4×4 array. This organisation is strongly indicated as preferable by a number of different factors. With this shape decided, we determined the following organisation:

- Used as a LUT, the array requires a 4-bit address, allowing it to generate any boolean function of up to 4 inputs[7].
- Used as a CAM, the array has 4 inputs and 4 outputs.
- Since both LUT and CAM (or PLA) operation use the same number of inputs they can share the same set of input buffers. The input options for the cell are therefore independent of its use as a LUT or a PLA.
- Since CAM operation has the same number of inputs and outputs it is easy to arrange to connect the outputs of one cell to the inputs of another in

[7] A number of studies([2][3]) have concluded that the optimal number of inputs for a LUT cell is around 3 or 4. These results might be slightly undermined since they assume that large gates are built from multiple LUTs. This is not the case with a CAM-based cell, where large AND/OR gates are made using the PLA-like features of the cell.

order to make the kind of dual-plane structures used in the classic AND-plane/OR-plane PLA.

Figure 2 is a simplified block diagram of the cell. It shows the following additional features of the cell:

– LUT and CAM operation share the same output buffers. Therefore the LUT can generate up to 4 outputs. This is used to tap some intermediate signals in the LUT column decoder, so that it can generate either 1 function of 4 inputs or 2 functions of the same 3 inputs[8].
– The CAM inputs and outputs are orthogonal - if the inputs are vertical then the outputs will be horizontal, and vice versa.
– Circuits are provided to link the matchlines (and/or datalines) in adjacent cells together. This makes it possible to create PLA-like arrays with more than 4 inputs or outputs.
– The links between adjacent cells are also available to be used as a fast local interconnect when the cell is used as a LUT (for carry paths etc.)
– The cells are also connected to a network of longer wires which link groups of cells in the same row or column.

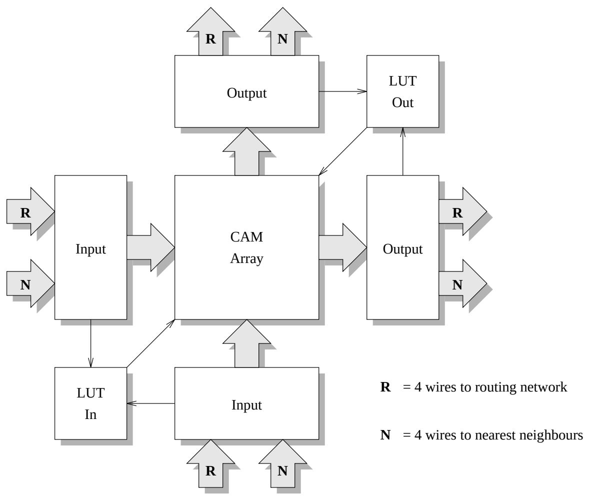

R
N
Output
LUT
Out
R
N
Input
CAM
Array
Output
R
N
LUT
In
Input
R
N
R = 4 wires to routing network
N = 4 wires to nearest neighbours

Fig. 2. Simplified block diagram of the new FPGA cell

This CAM-based FPGA cell appears to offer the advantages of the hybrid FPGA discussed earlier, but because it has only one type of cell it is not necessary

[8] for example, Sum and Carry, so that a full adder fits into one FPGA cell

to decide in advance on an appropriate mix of the two styles. It also has a number of other advantages:

- The CAM operation of the cell provides a fast, flexible, way of making connections between its horizontal and vertical inputs and outputs, and so of linking the horizontal and vertical routing wires. In other words, the 'logic' cell can also be used as a routing switch. This means that an appropriate mix of logic and routing can be chosen for each application, rather than having to be fixed in the architecture.
- The cell can easily be modified so that the CAM array be used as a small SRAM. This allows embedded memories be added to an application. Cells can easily be linked to create larger memory arrays. The row decoder needed for such an array is created using other FPGA cells operating in PLA mode.

The net result of all the above considerations is a single basic cell which can be used in any of the following modes (each of which are explored in more detail in Section 5:

1. a lookup table with up to 4 inputs,
2. a 4-input, 4-output expandable PLA element,
3. 16 bits of expandable SRAM,
4. 16 bits of expandable CAM,
5. a routing switch.

This kind of cell design seemed to be much better matched to the list of features required for our high-level synthesis approach than previous FPGAs and the other architectures considered. Consequently, in 1993 work began on a detailed design for an FPGA based on such a cell.

4 Lessons from the FPGA Hardware Design Process

4.1 Design Methodology

The major features of the design methodology which was followed for the FPGA chip were :

- Design entry using schematic capture, and all layout done manually (ie: no layout synthesis). All layout verified against the schematics.
- Analogue simulation models automatically derived from the schematics and used to verify the behaviour of the FPGA cell.
- A VHDL model (automatically derived from the schematics) used for simulation of both the individual FPGA cell and arrays of cells.
- An independent model of the cell written using a programming language, and used to generate test patterns for the VHDL and analogue simulations. The design only to be regarded as complete when all 3 models give the same results for the same set of inputs.
- The cell to use a mixture of CMOS and NMOS design styles, for compactness, but to be fully functional with a 3V supply.

Having two independent models of the cell was intended to provide a check on the interpretation of the cell specification. Also, writing a software model proved to be useful experience for the software development process.

4.2 Hardware/Software Development

In parallel with the hardware design another group began work on the support software (synthesis, optimisation, placement and routing). Before they started there was a worry that the cell was too flexible, and that it would be impossible to write software to fully exploit it[9]. This fear turned out to be unfounded, as the software group was able to find a viable solution to the problem. A detailed description of the software is outside the scope of this paper, but there are a number of places in which the hardware and software design processes have influenced each other:

- As mentioned above, the LUT column decoder is tapped in several places to give 4 possible output combinations. These were chosen based on the functions likely to be generated by the algorithm used for mapping onto LUT cells.
- All cell outputs are fully buffered (there are no pass transistors connected to the routing wires). This makes it easy to estimate delays, so that the software can easily identify (and hopefully fix) speed problems.

4.3 Registers

During the design process, a way of merging the input and output buffers of the cell was found. Furthermore, it was possible to arrange the compound circuit to contain a loop. This loop was used as the basis of a flexible register circuit, which makes it possible to register any output of any FPGA cell.

4.4 Problems

There was one area of the cell which caused major problems, both in its electrical behaviour and in the VHDL simulation. This was an attempt to create a bidirectional, analogue bus running through the middle of the cell, in the belief that it would be smaller than separate unidirectional buses. It took several weeks to get this to simulate reliably in VHDL, and the first identified hardware bug was due to this circuit. In the next revision it will be replaced with unidirectional busses.

[9] For example, with the same cell for both logic and routing, placement and routing are aspects of the same task, and so a simple ‘place, then route’ approach is inefficient

5 Applications

Here we examine some simple examples of kernel application circuits that can be mapped onto the CAM-based FPGA. Each of these applications is important in the context of high-level synthesis as determined by our initial analysis. In advance of the chip and its software tools being available we have only placed and routed relatively simple circuits by labourious manual methods. However these are some of the circuits that we wish to support well and it is crucial that they map densely onto the FPGA.

The cell uses an array of 4×4 one-bit CAM circuits, directly abutted to each other, to build the core of a CAM cell as shown in Figure 3. With the addition of the peripheral circuitry outlined in Figure 2 this forms the repeat unit of the CAM-based FPGA.

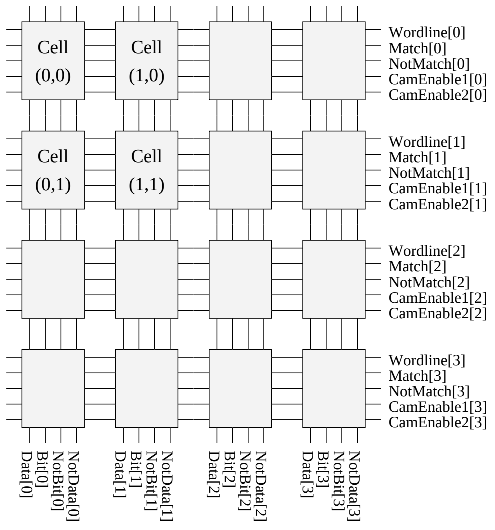


Fig. 3. The 4×4 CAM Cell Core

The fact that there are four output wires from each cell contrasts markedly with the single output of a conventional LUT. It allows much greater density of user circuitry in the array for those fragments of the circuit which can exploit it. These fragments are precisely the ones which are poorly supported by LUT-based architectures. In addition, the choice of a 4×4 cell gives a natural and highly useful input/output symmetry which results in simpler hardware which can be packed more densely and can be supported by simpler software.

5.1 The FPGA Block in LUT/RAM Modes.

Each cell can operate as a fully functional 16-bit LUT. The LUT mode is similar to conventional FPGA architectures except that it can be used to generate one function of four variables, or (the expected) two functions of three variables, four functions of two variables etc. Since the cell is designed to support four local links in each direction, the routing resources to support each of these different organisations is already in place.

The CAM-based FPGA also supports the construction of dense RAMs of arbitrary size in a way which we believe is superior to any competitive architecture. Fig. 4 shows how the CAM-based FPGA cells, operating in RAM mode, can be grouped to form a larger RAM structure. Again it is the dense local interconnection network that makes this organisation possible.

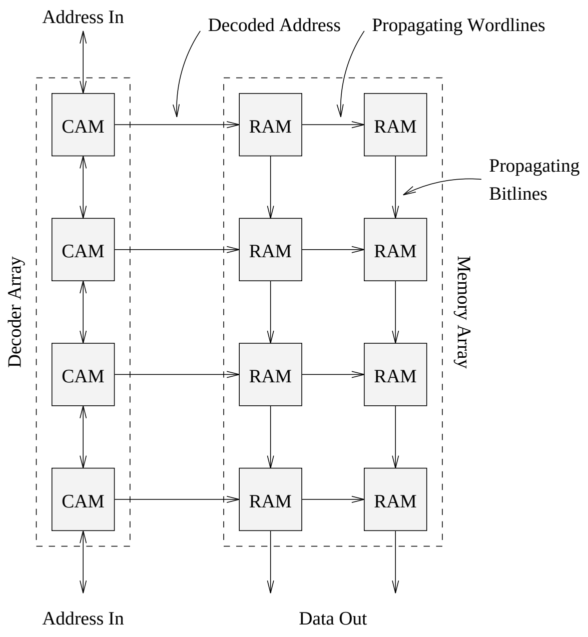
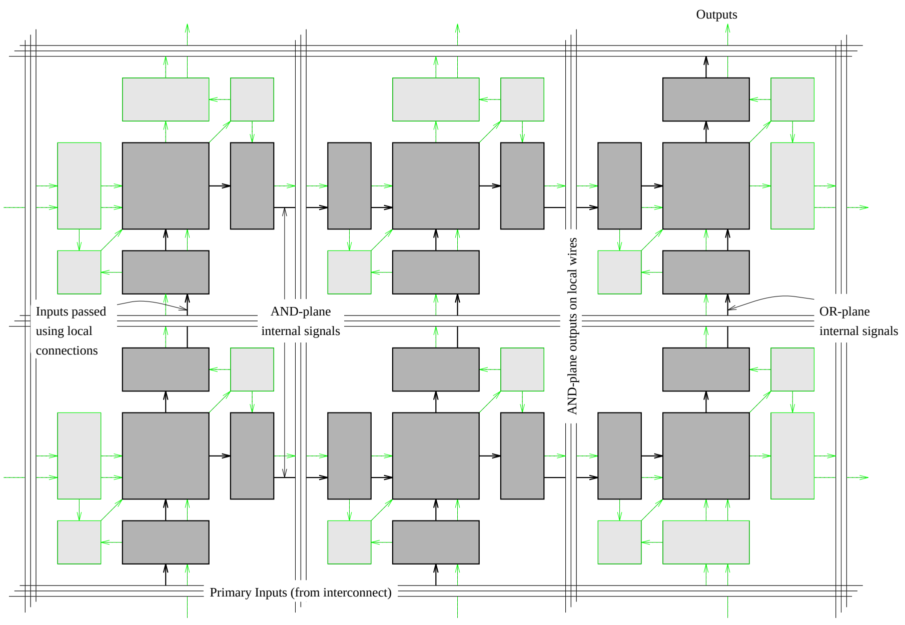


Fig. 4. A RAM structure in CAM FPGA

As can be seen in this figure, the CAM-based FPGA solves, rather neatly, the twin problems of building a dense RAM array and building the necessary decoder logic for the array. The decoder cells simply use the PLA mode with four horizontal outputs per cell, and the RAM cells use the RAM mode. Again there are no wasted cells in the core or periphery of this larger RAM. For larger RAM arrays the user can choose to use exhaustive decoding which results in the same rectangular structure, or can use a decoder tree which will release a few of the cells within the overall bounding box.

5.2 Using Cells in PLA Mode.



Outputs
Inputs passed using local connections
AND-plane internal signals
AND-plane outputs on local wires
OR-plane internal signals
Primary Inputs (from interconnect)

Fig. 5. A Larger PLA built from CAM-Mode cells

By design, the CAM cell is an ideal unit for building AND-OR planes. These cells, operating in PLA mode, can be grouped to form arbitrarily large PLA structures as shown in Fig. 5. When forming part of either the AND plane or the OR plane, each cell contributes sixteen programmable pulldown transistors at the intersection of the four input and four output wires.

For the AND plane, the four bit input pattern is propagated vertically (say) through the cell, with each bit in either normal or inverted form. The horizontal matchline will be high for any row which has a perfect match between the stored pattern and the input pattern.

Note that there are no wasted cells in the core of the PLA or around its periphery. Of course it is possible to apply standard PLA folding techniques to optimise the circuit, which might result in a non-rectangular PLA.

5.3 Using Cells to Build Larger CAMs.

The cells can also be used efficiently to build CAM arrays of arbitrary size. Although this was not an initial design requirement, it is nevertheless a natural consequence of using a CAM for the repeat cell. Having thought about the consequences of this, we are becoming convinced that CAM structures will themselves become an increasingly important building block in future applications.

As an example, we have recently implemented the Single Cube Containment (SCC) algorithm for combinational hardware optimisation using an actual CAM chip. The use of the CAM dramatically improves the performance of the algorithm and allows it to run in linear time. We have looked at a number of actual implementations of this algorithm in available CAD packages and they have all used a quadratic algorithm.

We believe that CAMs will find other uses in applications where previously they had not been considered as part of the solution space and we plan to continue studying this topic.

5.4 Using Cells as Routing Switches.

There is little to be said about this mode as the cell simply acts as a fully populated 16×16 crossbar switch. This is in fact not a new mode for the cell but rather a special case of its use as a PLA. This means that there is in fact more functionality available than a straightforward crossbar switch and some simple gates can be incorporated into the routing. These gates can be used to combine signals passing through a cell in the same way that signals are combined within a PLA. The major advantage of this functionality is that multiplexors can sometimes be implemented without using additional cells.

Naturally, these routing switches can be expanded to make larger routing structures merely by concatenating adjacent cells.

6 Combining Processor and FPGA on the Same Chip

All the evidence of the last four year's work at Oxford has indicated that there is a significant role to be played by FPGAs used as flexible co-processors interfaced to conventional microprocessors. We have implemented a number of applications on a microprocessor plus FPGA pair[4]. These have ranged from real-time video object tracking to spell checking and we have achieved speed-up factors of 6 to 30 times over a microprocessor alone. The novelty of our approach is that we take a purely programming approach to the construction of hardware/software systems. There was *no* user interaction with the design tools except via the high-level program. All optimisation, mapping, and place & route stages were completely automatic making the whole flow a ‘single button-press’ operation.

If our predictions about the relevance of FPGA co-processors prove correct, there will naturally follow a commodity market for an FPGA and processor on the same chip. This will obviously yield a reduction in the number of parts in a typical system implementation, and it will also speed up communication between the two processors, and may also simplify the communication and clocking arrangements. One of the authors (Ian Page) is actively engaged in designing FPGA co-processor systems and building the necessary software infra-structure so that the resulting hardware/software systems can be specified by a single, largely behavioural, specification. We expect that the CAM-based FPGA will have a significant role to play in these developments since it was designed from scratch with the needs of this style of high-level synthesis firmly in mind.

7 Conclusions

We have presented a novel FPGA architecture which we believe offers a significant advance on current FPGA implementations with regard to its ability to support the full range of sub-circuit organisations found in discrete logic applications. This is particularly true of those designs which are implemented by high-level synthesis, which we believe will become increasingly prevalent.

We have shown that FPGA co-processor combinations can usefully support applications and that entire hardware/software applications can be produced in actual practice via a purely programming approach. The development of the software to support this implementation route, the refinement of FPGA architectures, and the integration of conventional microprocessor cores and FPGAs are three of the most interesting areas for the future development of FPGAs.

8 Acknowledgements.

The authors would like to thank Jonathan Saul for his comments on this paper and for the implementation of the SCC problem referred to. In addition to the agencies acknowledged in the introduction to this paper, the work which underlies this paper has been supported over the last few years at Oxford by (in alphabetical order) Advanced Risc Machines Limited, Atmel Corporation, EPSRC (U.K. Engineering and Physical Sciences Research Council), European Union (Esprit/OMI programme), Hewlett Packard, Music Semiconductor, Oxford University Computing Laboratory, Sharp European Laboratories, and Xilinx Development Corporation, and others who wish to remain unacknowledged.

References

1. Ian Page and Wayne Luk. "Compiling occam into Field-Programmable Gate Arrays" in *"FPGAs" W.R. Moore, W. Luk (eds), Abingdon EE&CS Books, 1991*
2. Brown, Francis, Rose and Vranesic. *"Field Programmable Gate Arrays", Kluwer Academic Publishers*
3. Singh, Rose, Chow and Lewis. "The effects of Logic Block Architecture on FPGA Performance" in *IEEE Journal of Solid State Circuits, Vol.27 No.3 (March 1992)*
4. Lawrence et. al.. "Using Reconfigurable Hardware To Speed Up Product Development And Performance" in *this volume*

This article was processed using the LaTeX macro package with LLNCS style