FPGA Synthesis on the XC6200 using IRIS and Trianus/Hades (or from Heaven to Hell and back again[1])

R Woods*, S Ludwig+[2], J Heron*, D Trainor* and S Gehring+

* Institute of Advanced Microelectronics
The Queen's University of Belfast
Stranmillis Road, Belfast, N Ireland

\+ Institut für Computersysteme
Eidgenössische Technische Hochschule
Zürich, Switzerland

ABSTRACT

The implementation of a number of FIR filter structures in the Xilinx XC6200 technology is presented. The designs have been implemented using a combination of IRIS, an architectural synthesis tool and Trianus/Hades a set of integrated tools for implementing algorithms on Custom Computing Machines. The main attraction of this approach is that it allows algorithms to be compiled quickly allowing performance changes to be made at the architectural level in IRIS rather than at the FPGA layout level.

1 INTRODUCTION

There has been considerable interest in the development of custom computing machines based on re-configurable FPGAs and a number of systems have been presented at the FPGAs for Custom Computing Machines (FCCM) conferences in the past years. These range from the earliest devices such as the NCSU Anyboard [Bou92] from Brown University, to the DEC PeRLe-1 from the DEC Paris Research Laboratory [Ber93], Splash-2 system [Pri93], OneChip [Wit96] from the University of Toronto and the MATRIX [Mir96] from MIT. However, considerable effort is still required to create the hardware programs. This typically involves tedious hand crafting of designs using logic synthesis techniques and conventional FPGA designs.

Work has been presented to allow implementation of C programs on FPGA based systems [Pet96, Waz93]. However, conventional programming languages only allow serial programs to be implemented and it has long been recognised that custom computing machines will only become dominant if they can efficiently implement algorithms which exhibit considerable parallelism. There has been some development in languages suitable for FPGA compilation e.g. Ruby [LSC96]. A number of simple examples have been presented using Ruby but the system still relies on performing synthesis via Synopsys and using conventional FPGA place and route tools to achieve the designs [LSC96].

In this paper, we present a synthesis approach which links up a high level synthesis tool, developed originally for VLSI Architectures, IRIS with a closely coupled, FPGA design framework, Trianus and synthesis back-end Hades. The major advantage of the approach is that it allows you to modify algorithmic descriptions and implement FPGA layout in minutes rather than hours. This quick iteration cycle allows the user to relax placement

[1] IRIS is the messenger between the Greek Gods and earth and Hades, the Greek god of the Underworld
[2] Present address: Digital Systems Research Center, Palo Alto, California

constraints originally deemed necessary to achieve high performance results [Woo96] and to quickly iterate to solutions optimised for both time and area. An example for an FIR filter is examined.

2 ARCHITECTURAL SYNTHESIS

A considerable body of work has been carried out into the synthesis of VLSI architectures for computational complex algorithms in particular DSP algorithms. These techniques include methods based on Recurrence Equations [Bal94], dependence graphs [Kun88] and, algebraic methods [Luk89]. Many of the high level methods have the advantage that they derive a great number of solutions from a single algorithmic representation. Whilst specialist tools can produce architectural representations from high levels of abstraction, they are unable to produce functionally-correct, implementable circuits, as issues such as internal word growth, truncation and data organisation, have not been considered. These issues can have considerable effect on circuit performance such as latency and data timing and can invalidate any derived architecture. It is left to the IC or FPGA designer to modify and refine the initial architectures taking into considerations these design details.

A number of synthesis tools have also been developed such as CATHEDRAL [DeM90], PHIDEO [Lip91], and HYPER [Bro92] which can take algorithmic descriptions and apply scheduling, assignment and hardware mapping techniques to synthesise an architecture. With these systems, hardware mapping, the process that maps a flow graph onto the available hardware blocks, is carried out after the various scheduling and assignment procedures. The assumption is therefore made that hardware mapping does not alter the structure or functionality of the design. For the hardware units generally used in reported design examples synthesised by these tools, this may be a valid assumption. However, it has been demonstrated [Tra95] that if the hardware units are complex pipelined processors, hardware mapping can invalidate the architecture. This needs to be resolved if the circuit is to operate correctly once implemented using the chosen hardware, and has been shown to be a complex task [Tra95].

An alternative synthesis framework, called IRIS has been employed at Queen's University [Tra95]. It works on the building block philosophy where the designers can define the processing blocks and perform synthesis on these circuits. Starting with a Signal Flow Graph (SFG), the users selects processing blocks from the library and embeds these in the SFG. These blocks have specific timing and data organisation and simply inserting them into the SFG would invalidate the design. However, by use of re-timing routines, IRIS synthesises an architectural description which implements the original SFG using the blocks specified.

At first glance, this methodology may not appear attractive but hardware designers typically like to mix and match circuits, use different number representation and clever circuit design techniques to achieve efficient FPGA solutions. IRIS achieves this by enabling the extraction of parameterised expressions from complex VLSI processing elements, and using these expressions to achieve functionally-correct solutions for circuits built from these processors. Designers can quickly create and evaluate architectures that utilise existing hardware blocks and realise hardware using commercial synthesis tools.

2.1 IRIS Structure and Operation

The system viewpoint for IRIS is given in Figure 1. The IRIS shell is the central communicating process which processes commands from the user, invokes and monitors several tools within the IRIS framework, and relays the text and graphical information by these tools back to the user. Parameter files

provide the shell with information regarding the display and location of files.

From the shell, the user has control over the invocation and operation of two tools, a Schematic Editor which also functions as an interface for the various synthesis functions and a Process Designer (PD) for the design of the parameterised models required within IRIS. The PD is currently a simple parameter capture tool which allows the user to derive the various parameterised expressions for the particular circuit. In addition, the user can associate relevant speed and area estimates obtained from synthesised circuits.

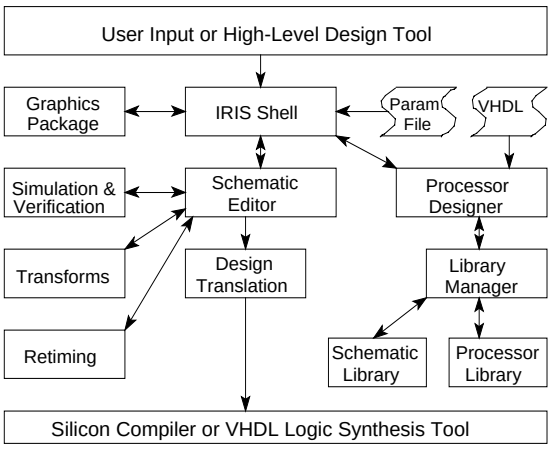

Fig. 1. IRIS System Functionality

The main design entry is via Signal Flow Graph (SFG) which the users creates in the schematic by using instances of various processing blocks connected together by wires (representing signals in the SFG) and terminated using external connectors (signifying external SFG inputs and outputs). At this stage, a number of design functions can be invoked including a symbolic simulator which generates a difference equation corresponding to the SFG and a numerical error analysis tools which currently provides Mean Square Error analysis. One of the core design functions is the retiming routines which ensure that the circuit derived using user defined processors, implements the original signal flow graph. This uses delay scaling, retiming and processor embedding.

Architectural optimisation is available within IRIS in a number of ways. The user can use transformations available within IRIS such as hardware scheduling, lookahead computation and power minimisation techniques. Alternatively, the user can choose different processing blocks (e.g. different multiplier structures), different internal pipeline strategies and vary wordlengths. Combining both of these techniques gives the user considerable architectural freedom, offering an extremely powerful environment for achieving optimal hardware implementation for the SFG.

IRIS can produce a VHDL description of the synthesised architecture using parameterised VHDL descriptions of the processor blocks held in the IRIS processor library allowing designs to be implemented by VHDL-based logic synthesis tools. A Lola [Wir95] interface has now been developed which allows the user to produce complete Lola descriptions for the Trianus/Hades environment. This is discussed later.

It should be pointed out that, though IRIS presently uses a graphical SFG input and was aimed initially at DSP algorithms, internally it works on a weighted graph and can thus accept any algorithm than can be represented in this form.

3 TRIANUS AND HADES

Today's hardware synthesis tools are usually big, bulky and slow. They provide little support for an iterative design cycle and even less for manual intervention by the user. Especially during the implementation of FPGA circuits, the designer's knowledge about the structure and layout of the design is often needed to meet the specifications. However, this knowledge cannot easily enter the design cycle in current tools.

One reason for this is the use of stochastic algorithms, such as simulated annealing, during

layout synthesis. Because they are stochastic, different runs of the algorithm produce different layouts even for the same input. If several design cycles are needed to achieve a satisfactory result (for instance, because a layout is not routable), this unpredictability of the outcome is unwanted. Instead, the designer will want to gain control over the synthesis process, e.g. by specifying the circuit's placement by hand. Manual placement is often used for building high-performance FCCMs [Ber93, Pri93].

A reason for the inefficiency of synthesis is the lack of interoperability between different design tools. As the user switches from one tool (such as a schematics editor) to the next one (such as a layout editor), a number of design files are typically stored to disk and read again, i.e. the data structures representing the design in the first tool are externalized to disk and subsequently internalized and translated to the data structures needed by the second tool.

3.1 Trianus Framework

To remedy the aforementioned problems, the Trianus framework was developed [GL96, Geh97]. Trianus is an extensible framework of tightly integrated FPGA architecture-independent tools (front-end), which are supplemented with architecture-dependent tools for specific FPGA architectures (back-ends). Together they form complete circuit development systems from initial design entry to bit-stream downloading.

The Trianus front-end comprises an HDL compiler for the Lola language [Wir95], a schematic editor and a framework for the construction of layout editors. Additionally, a circuit checker allows to validate manual circuit modifications.

Trianus can represent designs in three different ways: textually, schematically or using layout editors (Fig. 2). Arrows in the figure denote transitions from one representation to another. The three representations are centered around an architecture-independent, hierarchical data structure which is used system-wide. In the hierarchical data structure, instances of the same type (component) are always represented identically (e.g. the multiply-accumulate structures in the FIR filter example of Fig. 4). The hierarchical information is maintained by all tools and is available at all times.

This approach bears several advantages:

- High speed of operation, as no translation between circuit representations is necessary.
- New tools can be developed in short time, as only one data structure has to be supported.
- A consistent user interface of various tools is guaranteed by the editor framework.
- The hierarchical data structure encourages the use of hierarchical information for design representation.

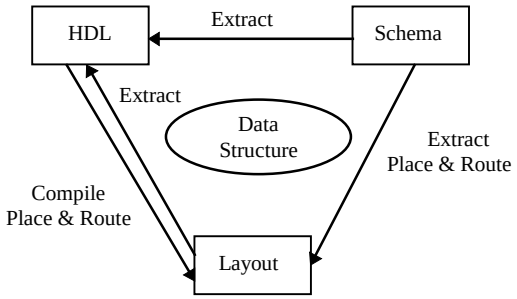

Fig. 2. Trianus Tools

3.2 Hades Synthesis Back-End

The Hades software [GL96, Lud97] implements a Trianus back-end and comprises a technology mapper, a placer, a router, a bitstream generator and a runtime system. The Hades hardware is a reconfigurable coprocessor based on the XC6200 FPGA from Xilinx and is described in [Lud96, Lud97].

The technology mapper is concerned with the packing of Boolean operators and registers into the available cells of the FPGA. Thanks to the simple and regular architecture of the XC6200,

this is a relatively easy task, which can be executed in linear time.

The placement algorithm uses a greedy heuristics and places Boolean expression trees in a constructive way. It first places the root of a tree into a cell and then places the two children to the right and above that cell. This simple strategy ensures the routability of the expression tree. Arrays of trees (as often occur in data paths) are placed either vertically or horizontally. Using and preserving the available hierarchical information, the circuit is placed from the inside out, and every instance of the same type has the same placement. The algorithm is very fast, taking time linear in the number of tree nodes.

An automatic router based on the well-known Lee-type maze-running algorithm is used to route a placed circuit [Lee61]. It, too, makes use of hierarchical information by routing from the inside out. Once a type is routed, the routing information is copied to all instances of that type. By using this copying process, the quadratic runtime of the Lee-algorithm is not such a severe problem. A design using 90% of an XC6216 can be routed in under 1 minute on a Pentium-class PC.

As further tools, a bitstream generator and loader and a runtime system were developed. The latter supports the development of coprocessor applications by representing the hardware part of an FCCM application through an automatically generated software interface to the programmer.

4 FPGA IMPLEMENTAION USING IRIS AND TRIANUS/HADES

Automated synthesis of FPGA layouts without thought to structure can lead to low density designs. In order to avoid difficulties at the place and route level, it is important to either build in enough redundancy (e.g. over specify the performance) to ensure the specification will be met or employ a strict architectural and/or floorplanning strategy. This strategy has been shown to work well [Woo96, Luk96] but is acknowledged as time-consuming [LSC96].

IRIS can produce workable architectural descriptions from high level descriptions where wordlength and truncation issues have been considered and the Hades synthesis tools can quickly generate FPGA layouts from Lola descriptions.

IRIS has now been closely integrated with Hades resulting in a powerful system capable of investigating FPGA implementation. It has been achieved by developing a Lola interface from IRIS which allows IRIS to produce Lola code for the algorithm under question. The systems are well matched as there is considerable structure within IRIS which Hades can preserve and quickly translate into layout. This allows a different design strategy to be employed which removes the designer from the low level design flow. For example, if the designer finds the required target performance has not been met at the circuit layout stage, one can go back to IRIS and choose many of the circuit transforms available. The use of adding pipelining delays is a particularly good option for FPGAs as it can sometimes be implemented with no extra cost. The key issue is that circuit optimisation is being performed at the algorithm/architectural level which is less consuming than varying placement and routing at the FPGA level. The design flow for using IRIS with Trianus/Hades is shown in Figure 3.

5 FIR FILTER EXAMPLE

The equation for an n-tap FIR filter is given as:

$$y_n = \sum_{i=0}^{n-1} a_i x_n \qquad (1)$$

where x_n is the input data stream, y_n, the filtered output and a_i represents the filter response. There are a number of possible SFG

representations for this FIR filter one of which is given in Figure 4.

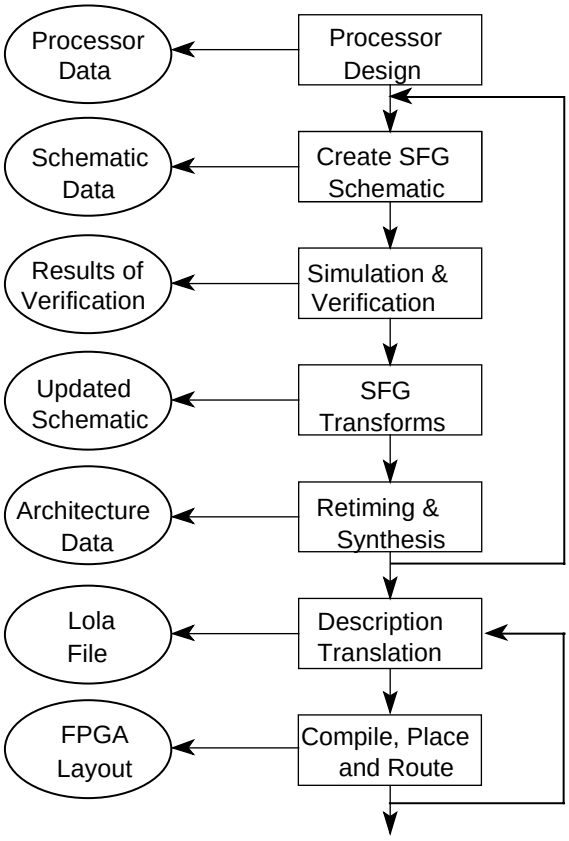

Fig. 3. Unified IRIS/Trianus/Hades Design flow

It has been decided to build a 3 tap filter with a throughput rate of 20MHz. For this example, we will examine the possibility of using the carry-save Mac block.

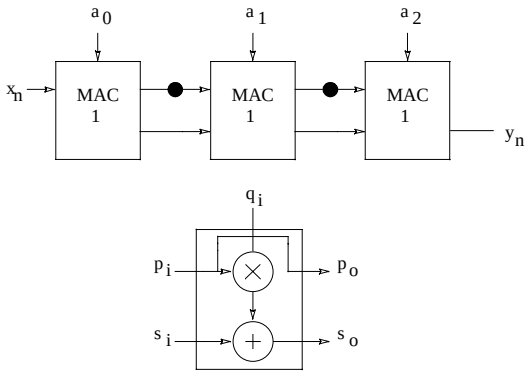

Fig. 4. FIR Filter implementation using MACs

This implementation involves combining the multiplier and adder from a classical SFG description into a single Multiplier Accumulate (Mac) operation which can be implemented using a number of available Mac structures. The Carry-Save structure given in Figure 5 has been used in this instance.

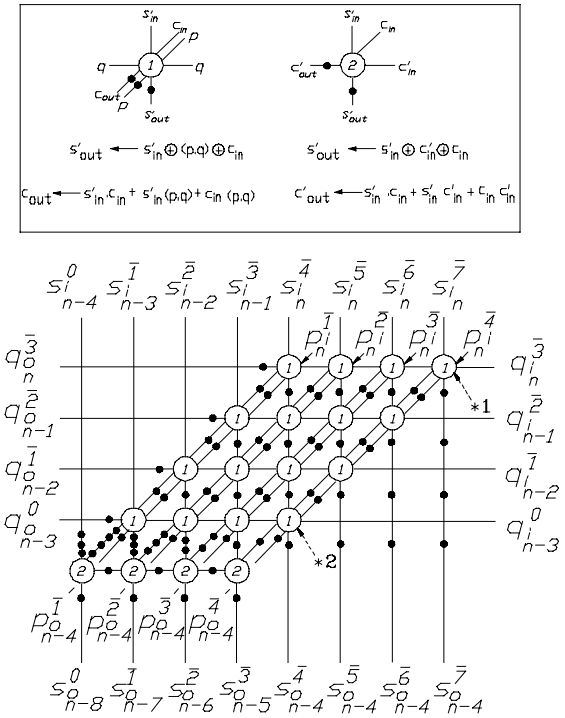

Fig. 5. Structure of Carry-Save MAC Processor

The carry-save is useful in this application as it can be easily parameterised and the level of pipelining can be varied within the block allowing both area and speed to be varied. The inputs p_i and q_i correspond to multiplier and multiplicand respectively and s_i correspond to the additive input (subscripts *i* refers to inputs and *o* to the outputs). An abstract model for this structure which is what IRIS uses is shown in Figure 6. The parameterised expression in the boxes refers to the latency of that particular datapath, whilst the data time shape is shown graphically at each input and output of the model. The values *m* and *n* represent the wordlengths of the datapaths p and s respectively and *x* represents the level of pipelining. The effect of truncating data between processors models is included within the

processor block by the parameters t_q, t_s and t_p referring to each datapath respectively.

Inserting this particular processor into the signal flow graph results in the circuit of Figure 7.

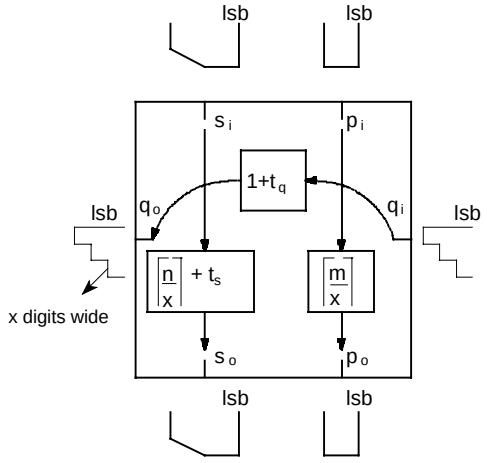

Fig. 6. Carry-Save MAC Model.

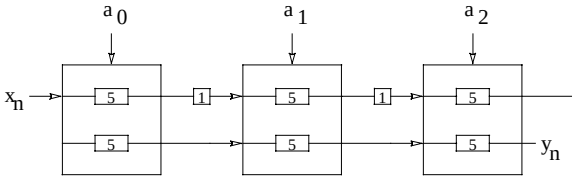

Fig. 7. FIR Filter implementation using MACs

The choice of a wordlength of 5 bits in this example for each of the MACs has resulted in both of the latencies being 5 (we are actually using 4 bits but we have made this 5 to avoid truncation in this example - something which is easily dealt with within IRIS). The resulting circuit is shown in Figure 8.

The total area of the XC6200 used is 520 CLBs with a throughput rate of 50MHz. As the throughput rate exceeds that required (20MHz), we would want to examine alternative solutions that use a smaller area. Hardware sharing and reducing levels of pipelining have been chosen for this example.

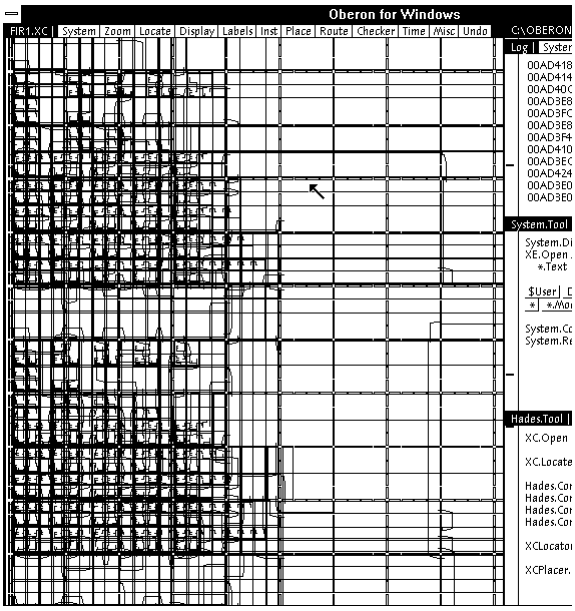

Fig.8. FIR Filter implementation
(Trianus/Hades Screen View)

5.1 Hardware sharing

One option is to multiplex the hardware. This is achieved by working out the schedule for the computation. Figure 9 gives the current schedule.

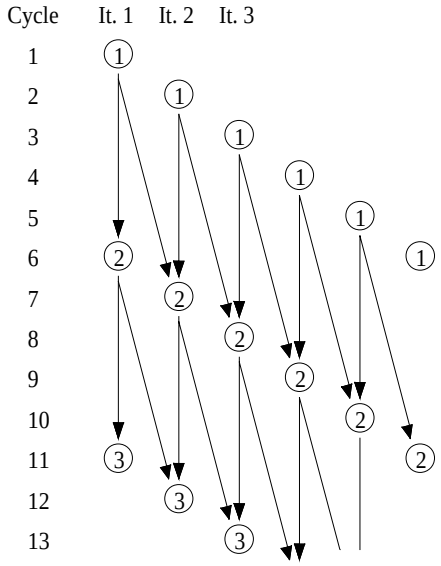

Fig. 9. Schedule for FIR Filter of Fig. 7

The scheduling process is similar to that employed in many synthesis systems but has been modified to cope with several overlapped

iterations of the algorithms due to the latencies within the block. The cycles are listed on the y-axis and the progression of the iterations of the algorithm is given in the x-axis. The lines indicate the dependencies which must always be preserved. It can be seen that all three iterations overlap so it is necessary to schedule the operations to occur at different times. This is done by reducing the computation rate to once every two cycles. This gives the schedule of Figure 10. It can be seen that from this, we can either multiplex operations 1 and 3, 1 and 2 or 2 and 3.

This scheduling process is contained within IRIS and the user simply chooses to merge two operators, for this case 2 and 3. The resulting circuit is given in Figure 11.

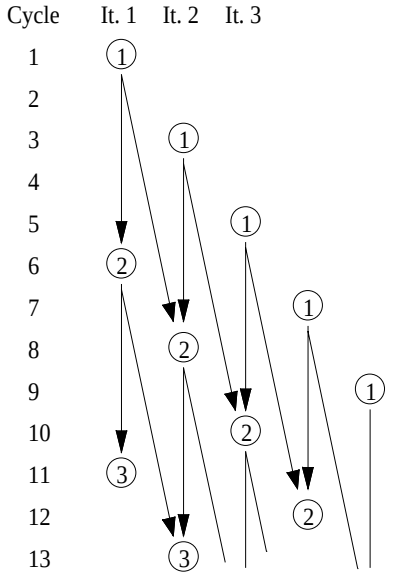

Fig. 10. Revised Schedule to allow multiplexing

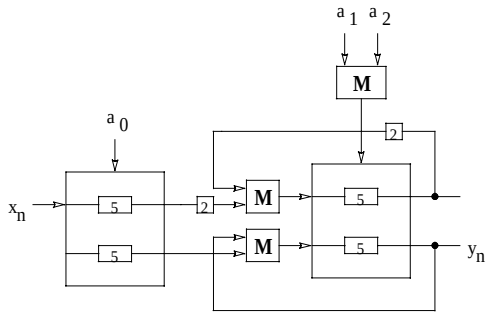

Fig. 11. FIR Filter with hardware sharing

IRIS has added multiplexers (signified by the M blocks) and additional timing delays necessary to preserve the timing. This results in a design with a throughput rate of 20MHz (half the clock rate) and an area of 380 CLBs. This meets the specification required. For demonstration, we have also shown the results of hardware sharing all three operations as given by Figure 12.

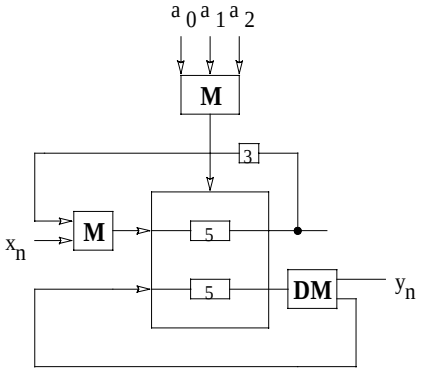

Fig. 12. Hardware sharing onto a single MAC

5.2 Levels of Pipelining

Another method of reducing area is to reduce the levels of pipelining in the circuit. This involves taking ever other row of latches from the circuit of Figure 5. Fortunately, this is one of the parameters within the IRIS model (see Figure 6) and just involves changing a parameter. In this example, we have reduced the pipelining to once every other row. This reduces the delay in the MAC block from 5 to 3. The circuit is shown in Figure 13.

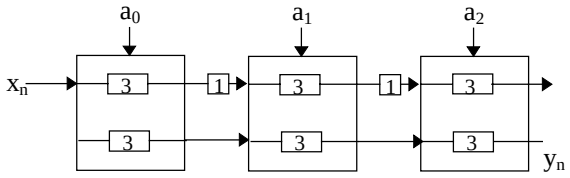

Fig. 13. FIR Filter implementation using MACs

6 DISCUSSION

The results for each of these four examples (Figures 7, 11, 12 and 13) are given in Table 1. The table gives the figures for the actual CLBs used to implement logic, the bounding area (i.e.

rectangle) of the design and the throughput rate. Table 2 gives the CLB utilisation which is computed by dividing the CLBs used by the bounding box and the area relative to the largest design, i.e. Figure 7.

It can be seen from Table 2, that the pipelined designs give the highest throughput rate. The hardware sharing results in proportional reduction in hardware and speed much as expected. In this case, there is no area penalty for the additional routing needed to route the feedback lines of Figure 11. There is available routing due to the 60% utilisation.

Circuit	No of CLBs	Bounding Box (h*v)	Data rate (MHz)
Fig. 7	520	15 x 56	47
Fig. 11	380	16 x 40	20
Fig. 12	225	18 x 16	11
Fig. 13	508	15 x 52	25

Tab. 1. FIR Filter comparison

It can be seen that reducing pipelining gives a drop in performance with no real area gain. This is due to the XC6200 feature that allows a flipflop and combinational logic to be implemented in one CLB. Thus, to a certain extent, pipelining can be viewed as a "free" algorithmic transform which increases performance with little area penalty.

Circuit	Utilisation (% CLBs used)	Relative area	Data rate (MHz)
Fig. 7	62	1	47
Fig. 11	59	0.76	20
Fig. 12	78	0.35	11
Fig. 13	65	0.93	25

Tab. 2. FIR Filter comparison

7 CONCLUSIONS

In this paper, we have presented a fast efficient synthesis route for implementing FPGA hardware using IRIS and Hades, the back-end of the Trianus system. A simple example for a FIR was presented which was synthesised into a FPGA layout, then optimised using transforms within IRIS and then synthesised again.

The advantage of this approach is that the systems are well matched and is fast compared to existing approaches and thereby allows you to evaluate circuit output without the need for considerable work at the circuit level. It gives the user more time to explore different solutions at the algorithmic and architectural level which will give greater performance improvement than attempting to optimise at the circuit level. Concepts such as blocks utilising distributed arithmetic and different number representations can be included within IRIS and used in the design flow. Thus, it is possible to develop specific circuits and transforms suitable for a particular class of FPGA families.

8 ACKNOWLEDGEMENTS

The authors would like to acknowledge Xilinx, Scotland for their continued support over the years, to the Engineering and Physical Sciences Research Council who supported the IRIS work and to the Department of Northern Ireland for their financial support.

9 REFERENCES

[Bal94] D G Baltus, "Efficient Exploration of Affine Space-Time Transformations for Optimal Systolic Array Synthesis", PhD Thesis, MIT, Feb 1994.

[Bou92] D Van den Bout et al., "Anyboard: an FPGA-Based Reconfigurable System", IEEE Design and Test of Computers, Sep 1992.

[Ber93] P Bertin, D Roncin, J Vuillemin, "Programmable Active Memories: A Performance Assessment", Research on Integrated Systems: Proc. 1993 Symposium, MIT Press, 1993.

[Bro92] “Anatomy of a Silicon Compiler”, eds. R W Broderson, Kluwer Academic Publishers, Boston, MA, 1992.

[DeM90] H DeMan et al., “Architecture-Driven Synthesis Techniques for VLSI Implementation of DSP Algorithms”, Proc. IEEE, Vol 78, No.2, Feb 1990.

[GL96] S Gehring, S Ludwig. “The Trianus System and its Application to Custom Computing”, Proc. 6th Intl. Workshop on Field-Programmable Logic and Applications. LNCS 1142, Springer, 1996.

[Geh97] S Gehring. “An Integrated Framework for Structured Circuit Design with Field-Programmable Gate Arrays”, Dissertation, ETH Zürich, 1997.

[Kun88] S Y Kung, “VLSI Array Processors”, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[Lee61] C Y Lee. “An Algorithm for Path Connections and its Applications”, IRE Trans. Electronic Computer, Vol. EC-10. Sep 1961.

[Lip91] P Lippens et al., “PHIDEO: a silicon compiler for high speed algorithms”, Proc. European Conference on Design Automation, 1991.

[Lud96] S Ludwig. “The Design of a Coprocessor Board Using Xilinx's XC6200 FPGA - An Experience Report”, Proc. 6th Intl. Workshop on Field-Programmable Logic and Applications. LNCS 1142, Springer, 1996.

[Lud97] S Ludwig. “Hades - Fast Hardware Synthesis Tools and a Reconfigurable Coprocessor”, Dissertation, ETH Zürich, 1997.

[Luk89] W Luk, G Jones, M Sheeran, “Computer Based Tools for Regular Array Design”, in Systolic Array Processors, eds. J V McCanny et al., Prentice-Hall Ltd., 1989.

[Luk96] W Luk, S Guo, N Shirazi, N Zhaung, “A Framework for Developing Parameterised FPGA Libraries”, Proc. 6th Intl. Workshop on Field-Programmable Logic and Applications. LNCS 1142, Springer, 1996.

[LSC96] W Luk, N Shirazi, P Cheung, “Modelling and optimising run-time reconfigurable systems”, Proc. IEEE Symposium on FPGAs for Custom Computing Machines, IEEE Computer Society Press, 1996.

[Mir96] E Mirsky, A DeHon, “MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources”, Proc. IEEE Symposium on FPGAs for Custom Computing Machines, IEEE Computer Society Press, 1996.

[Pet96] J B Petersen, R B O’Connor, P M Athanas, “Scheduling and partitioning ANSI-C programs onto Multi-FPGA CCM Architectures”, Proc. IEEE Symposium on FPGAs for Custom Computing Machines, IEEE Computer Society Press, 1996.

[Pri93] D Prior, M Thistle, N Shirazi, “Text Searching on Splash 2”, Proc. IEEE Workshop on FPGAs for Custom Computing Machines, 1993.

[Tra95] D Trainor, R F Woods, J V McCanny, “The architectural synthesis of an image processing algorithm using IRIS”, IEEE workshop on VLSI Signal Processing, Sakai, Japan, Oct 1995.

[Waz93] M Wazlowski, L Agarwal, T Lee. A Smith, E Lam, P Athanas, H. Silverman, S. Gosh, “PRISM-II Compiler and architecture”, Proc. IEEE Workshop on FPGAs for Custom Computing Machines, 1993.

[Wir95] N Wirth. “Digital Circuit Design. An Introductory Textbook”, Springer, 1995.

[Wit96] R Wittig, P Chow, “OneChip: An FPGA Processor with Reconfigurable Logic”, Proc. IEEE Symposium on FPGAs for Custom Computing Machines, IEEE Computer Society Press, 1996.

[Woo96] R F Woods, A Cassidy, J Gray. “VLSI Architectures for Field Programmable Gate Arrays: A Case Study”. Proc. IEEE Symposium on FPGAs for Custom Computing Machines, IEEE Computer Society Press, 1996.