Area Efficient DSP Datapath Synthesis

Andrew A. Duncan
Department of Engineering
University of Aberdeen
Aberdeen, UK
a.a.duncan@aberdeen.ac.uk

David C. Hendry
Department of Engineering
University of Aberdeen
Aberdeen, UK
d.c.hendry@aberdeen.ac.uk

Abstract

COBRA is a behavioral high level synthesis tool for datapath dominated applications. It uses a regular architecture which has been previously shown to significantly reduce the area of synthesised datapaths and integrates the traditional scheduling, allocation and binding tasks into one global optimisation. Optimisation is performed using simulated annealing in a three dimensional space termed "datapath space". Unlike previous approaches, COBRA uses the actual data flow defined by the behavioural description to imply *a datapath. The combination of the datapath space concept and target architecture gives COBRA a strong relationship between behavioural and physical domains and provides area efficient, globally optimising datapath synthesis.*

1 Introduction

A high level synthesis system creates an RT-level structural netlist from an algorithmic behavioural description. Logic and layout synthesis tools can then be used to assemble the RT-level netlist thus providing a route from algorithmic behaviour to physical chip layout.

The RT-level structure forms the interface between high level synthesis and silicon assembly. High level synthesis systems have therefore traditionally optimised the RT-level structure, however the *real cost* of the synthesised system, will ultimately be measured in the physical domain in terms of chip area. The high level synthesis system therefore optimises the final cost of the solution *indirectly*. The physical cost of an RT-level structure is difficult to estimate, and to compound this, the physical cost of a partial structure *during* high level synthesis is even more difficult to estimate.

The importance of wiring and layout on high level synthesis cannot be over emphasised. It was concluded in [14] that high level synthesis must take physical domain effects into consideration if they are to produce high quality designs. Similar conclusions were drawn in [10] where it was shown that BUD could not produce a cost–performance trade-off curve when physical domain effects were taken into account.

High level synthesis tools also typically perform synthesis in a number of discrete steps. For example, scheduling may be performed first, followed by operator allocation and binding, interconnect allocation, and finally register allocation. The optimisations carried out at each stage are heavily interdependent and do not guarantee to optimise the *final* cost of the RT-level structure.

Partitioning the high level synthesis problem and optimising the RT structure, makes the problem easier to formulate, but the overall consequence is that a *sub-optimal* solution is found for a cost function which only *indirectly optimises* the real cost. There are therefore essentially two problems:

1. The cost of the solution is measured in the structural domain rather than in the physical domain.
2. Subdividing the synthesis problem results in sub-optimal solutions.

This paper presents the COBRA (Column Oriented Butted Regular Architecture) high level synthesis tool for datapath dominated applications. COBRA addresses the two problems defined above by :–

1. Defining a target architecture and layout style which gives a close relationship between structural and physical domains and significantly reduces interconnect area.
2. Performing global optimisation of the RT-structure.

The overall effect is to provide a global optimisation of a cost function which is closely related to the *physical* cost of the solution.

2 Previous Work

There are a number of ways to take physical domain effects into account. Firstly, physical effects can be estimated during synthesis. This approach is adopted in systems such as BUD[9] and the Siemens Synthesis System[18] which use hierarchical clustering to give an estimate of layout and wiring costs. HYPER[16] partitions the synthesised datapath using locality of interconnect as one of the main criteria in the partitioning process and 3D Scheduling[19] schedules and floorplans simultaneously. Pangrle et al[13] perform placement and routing simultaneously with connectivity generation, similar ideas are also employed in Pangrle and Jang's GB system[6].

A second method involves iterative improvement of the RT structure using information fedback from later in the design cycle. The Chippie expert system[1] controls the Slicer/Splicer system using a "knobs and gauges" approach which allows it to meet area, timing and power constraints. The University of Illinois' IBA system uses layout information fedback to its Fasolt[7] tool to improve the synthesised datapath.

A third method is to define a target architecture such that the interconnect and layout style are well defined. This approach has been used by the PARBUS[4], DAGAR[17] and CASS[3] systems which have all reported large area savings when compared to "conventional" synthesis techniques. These systems do not need heuristic physical domain estimation techniques or feedback from layout tools; the architecture and layout style are such that the structural domain cost is closely related to the physical cost. This is the approach used in COBRA.

Work has also focused on globally optimising high level synthesis. The various tools from the University of Waterloo, eg. [5], have developed an integer programming model which can perform optimisation on the whole datapath synthesis problem. Devadas and Newton [2] use simulated annealing to perform global datapath optimisation. However, neither of these systems are designed to take physical domain effects into account during synthesis.

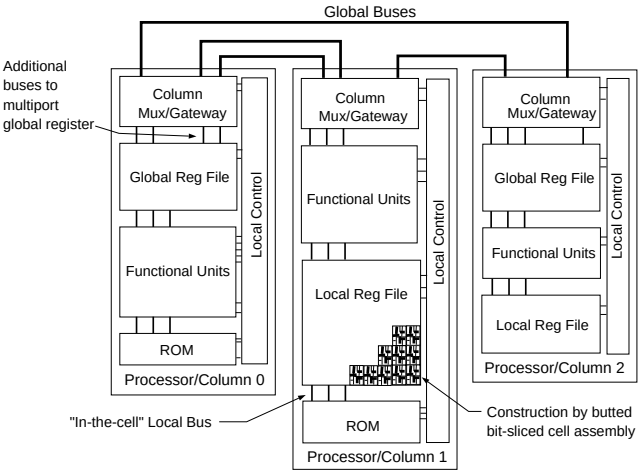

Figure 1: Example of target architecture/layout-style composed of three processors

3 Target Architecture

COBRA uses the CASS target architecture and layout style first defined in [3]. In [3] this architecture and layout style was shown to have the potential to save around 50% area when compared to "conventional" techniques. This was achieved by virtually eliminating global wiring in the final datapath. Estimation of wiring cost is perhaps the biggest problem in determining the physical cost of an RT-structure. Therefore, because of the large reduction in global wiring, the structural cost of CASS datapaths will be a good estimate of their physical cost.

Figure 1 shows the target architecture. The datapath is partitioned into a number of communicating processors, each with functional units and local memory (register file and ROM). Each processor is designed around a three bus local bus consisting of two read buses and one write bus. Processors communicate using a system of global buses. In addition to local memory, some processors may also require a multi-ported "global register file". Sufficient global buses, and global register ports, are allocated such that all global communications can be serviced without having to add extra c-steps to the schedule.

Processors are implemented as separate datapaths and are constructed by butted bit-sliced cell assembly. Cells are butted horizontally to achieve the desired bit-width and stacked vertically to achieve connectivity between datapath elements. The processors therefore form "columns" of cells in the physical domain. Butting the cells to form columns essentially provides *free* local interconnect. Since the local bus structure is fixed, the area cost of each cell will be fixed and real physical cost of each processor is therefore easy to calculate.

Global bus routing may also be provided by butt connection, however, if column lengths are markedly different a more area efficient solution may involve conventional macrocell place and route techniques viewing each processor as a macrocell. Global wiring between processors is required for this method, but since COBRA minimises the number of global buses required by a solution, any required wiring area will also be minimised.

The combination of the architecture and the synthesis tool localises computation, and thus interconnect in columns. Moreover, the combination of the architecture and layout style almost completely eliminates global interconnect in the final datapath, thus significantly reducing area. There is a strong relationship between the structural and physical domains since the structure implies a floorplan.

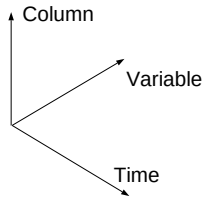

Figure 2: Three dimensional "datapath space"

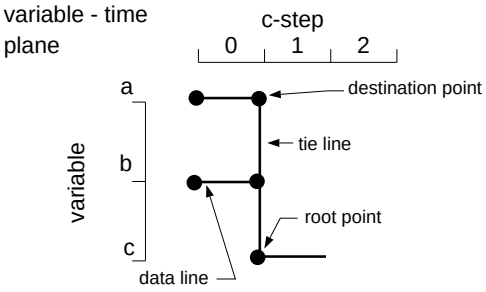

Figure 3: Mapping on variable-time plane for $c = a + b$.

The target architecture therefore addresses problem 1 of section 1.

4 Datapath Space

High level synthesis systems are typically *hardware oriented*, that is, they allocate hardware and connect it such as the given behaviour can be realised. In contrast, COBRA is *data oriented*, it uses the data flow described by the behaviour to *imply* a datapath. COBRA uses a solution space known as *datapath space* (dp-space) in which the lifetimes of variables are globally optimised using simulated annealing. The configuration of the variable lifetimes in datapath space *implies* a datapath and optimising the variable lifetimes in dp-space will optimise the datapath. Dp-space provides a strong relationship between behavioural and structural domains, and therefore a strong relationship between behavioural and physical domains via the architecture and layout style. The dp-space model allows COBRA to perform simultaneous scheduling, allocation and binding.

COBRA currently only synthesises straight line code segments and uses a three dimensional datapath space model as shown in figure 2. The three dimensions of dp-space are time, variable and column. Ordinates in the variable dimension represent variables (both explicit and implicit) in the behavioural specification and the column dimension directly relates to the columns of the target architecture. The data flow described by the behavioural specification is mapped into, and optimised in, datapath space.

Figure 3 is a mapping of $c = a + b$ into datapath space. The operation uses a 1-cycle adder and shows four important entities in dp-space: *data lines*, a *tie line*, *destination points* and a *root point*.

- A data line defines data. A tie line in the time dimension implies storage, and hence register use, for that variable for the length of the line. A data line in the column dimension implies a global bus transfer between columns.

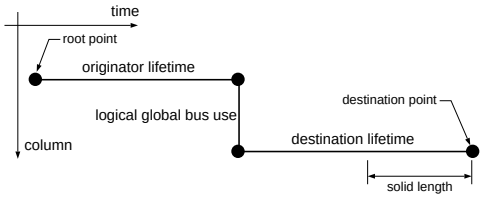

Figure 4: A two pin net

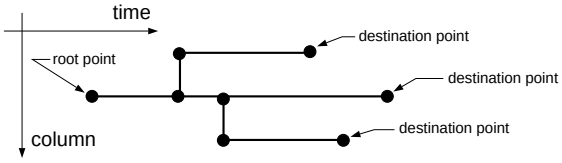

Figure 5: Data line composed of three two pin nets

- The root point is the *data source* for all uses of a variable until it is reassigned.
- Destination points are *data sinks*.
- A tie line "ties" data from one or more destination points together and creates new data at the new root point. The tie line in figure 3 ties a and b together to create the new data c. Therefore, the tie line implies an adder is required in c-step 0.

Data lines are composite entities and are composed of an overlay of one or more *two pin nets*, as shown in figure 4 (cf. two pin net in routing terminology). Every use of data requires a unique two pin net which defines the "route" of the data from its data source to its data sink. In this context, a two pin net defines the *lifetime* of a particular data between its birth at its root point, and its death at its destination point. Figure 4 shows that a two pin net is composed of *originator lifetime*, *destination lifetime* and *logical global bus use* segments:

- The originator lifetime is the time contributed by a net to the variable data line in the source column.
- The logical global bus transfer schedules a *logical global bus* transfer between source and destination columns. Logical global buses provide connectivity between a column and other columns in the datapath. Logical global buses are an upper bound on the number of physical global buses which will be required in the final datapath.
- The destination lifetime segment is the time contributed by a net to the variable data line in the destination column. Part of the destination lifetime is "solid." The solid part cannot be shortened by moving the logical global bus transfer back in time. As such it represents the delay of the operator associated with the destination tie line.

Two pin nets define root and destination points and hence tie lines. The overlay of two pin nets define data lines. Figure 5 shows a data line composed of an overlay of three nets. As can be seen, a data line is essentially a rectilinear Steiner tree (RST) rooted at the root point with leaves at the destination points. The RST will be planar in the column-time plane (since a unique RST exists for each variable) and defines the c-steps in which the variable is alive for each column. Storage and bus use are not directly associated with nets since this could imply multiple resource use for storing/transferring the same data.

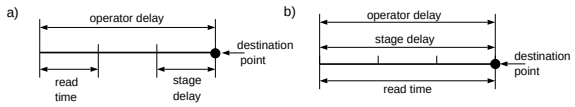

Figure 6: Solid length of destination lifetime.

As mentioned above, the solid part of a net represents the delay of an operator associated with the destination tie line. To be more precise, it defines the time when the data is operated on by the target functional unit, when transactions occur on the local bus and how frequently operations can be started on a pipelined operator. Figure 6 shows the solid part of a destination lifetime in more detail. As can be seen, it is composed of *delay*, *read time* and *stage delay* segments. Delay represents the overall operator delay whereas stage delay refers to the delay of each stage of a pipelined operator. The read time is the number of cycles the register file needs to be read for this operator, and therefore when the local read bus is required. The local write bus will be used during the last "stage delay" of the operator execution.

Using this model COBRA can deal with multicycle and pipelined functional units. To illustrate this, figure 6a represents a three cycle pipelined operator with stage delay of 1 c-step. The pipelined operator has internal registers and therefore has a read time of 1 c-step. Figure 6b depicts a 3 cycle non-pipelined operator without internal registers. Since the operator is non-pipelined, the stage delay is the same as the operator delay. Similarly, since the operator has no internal storage, the register file is read for the duration of the operation.

The solid length effectively defines when a functional unit and local bus is in use and hence partially defines local processor operations in the processor to which the operator is assigned.

The configuration of two pin nets in dp-space defines a set of data lines and tie lines. The *data lines imply register and global bus use* whereas the *tie lines in conjunction with the destination solid length imply FU use*. Thus, the configuration of two-pin nets, and hence variables, in dp-space implies a datapath.

The actual connectivity between two pin nets is defined by the specified behaviour but their structure in dp-space can be reconfigured (whilst retaining the connectivity) in order to optimise the implied datapath. It can be seen that reconfiguring the two pin nets, whilst retaining their connectivity, will imply different hardware use, at different times and in different processors, and hence a different datapath. Optimising the variable lifetimes will therefore optimise the datapath required to created the datapath.

COBRA uses a library which associates a delay (in c-steps) and an area estimate with each datapath component type. Module binding is specified to the tool; the user chooses which type of functional unit will be used for each operator.

To summarise, dp-space is a three dimensional space in which the configuration of variables (represented by two pin nets) implies a datapath structure, which in turn implies a floorplan. The physical cost of a dp-space configuration can therefore be easily estimated through the area figures associated with each datapath component.

5 Mapping into Dp-space

COBRA begins by compiling the given input description into a data flow graph (DFG) and performs ASAP and ALAP scheduling as a preprocessing operation. The DFG is then mapped into dp-space. DFG nodes map to tie lines which are *randomly* allocated to columns in dp-space. Input and operator DFG nodes

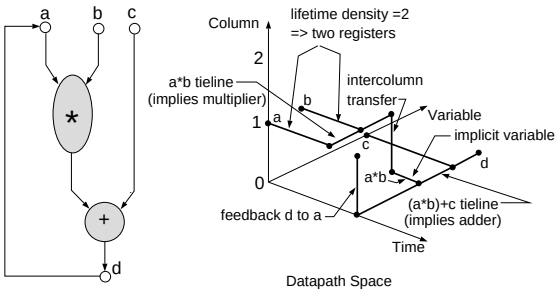

Figure 7: Simple DFG and initial dp-space representation

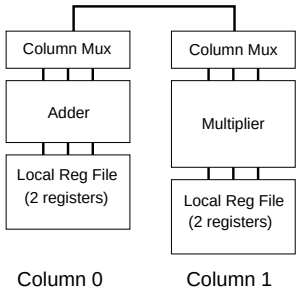

Figure 8: Initial architecture for example

use their ASAP time as their initial time co-ordinate. Output nodes are all scheduled at the time constraint specified in the behavioural description. DFG edges map to two pin nets which are created between root and destination points. The translation of the DFG into dp-space also creates the implicit structural domain representation.

Figure 7 shows the DFG and a random initial mapping for a simple example, the relationship between the graph structure and the mapping should be apparent from the figure. The initial random mapping defines the initial lifetimes and examination of the dp-space configuration will reveal the implied architecture. In this case two columns are required; column 0 requires an adder, which is implied by the $(a * b) + c$ tie line, and column 1 requires a multiplier which is implied by the $a * b$ tie line. Both columns require two registers; dp-space shows that there are two variables alive simultaneously in each column (ie. there is a lifetime density of 2 in each column) therefore two registers will be required. Finally, the intercolumn transfer implies that a global bus is available between columns 1 and 0. Figure 8 shows the implied architecture graphically.

The mapping from DFG to dp-space gives an initial random solution for optimisation by simulated annealing. Dp-space can be used for formulation of both time constrained and resource constrained optimisation problems, however, at the moment only time constrained problems are considered.

Figure 9 shows the internal representation and datapath for the example after optimisation. Naturally, real examples have much more complex dp-space structures than this simple example, however this does serve to illustrate the basic concept.

Optimisation by reconfiguration of lifetimes in dp-space provides a global perspective on the synthesis problem and addresses problem 2 of section 1.

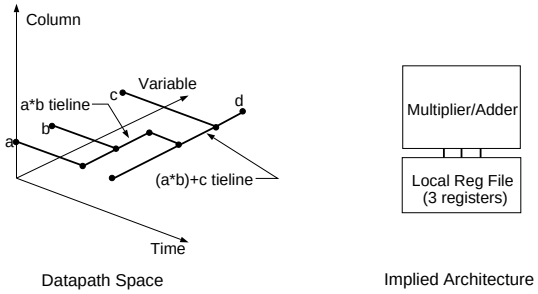

Figure 9: Dp-space representation and synthesised architecture after optimisation

6 Optimisation by Simulated Annealing

Every dp-space configuration which retains the data-flow specified by the behavioural description is a solution to the datapath synthesis problem. Simulated annealing is used to optimise the dp-space configuration, and thus optimise the synthesised datapath. Dp-space forms the annealing state space with the cost of a solution measured from the implied structural description. The number of functional units, registers and global buses can be evaluated from the implicit structure. The cost of a solution is calculated as:

$$Cost = \sum_{\text{Columns}} ColumnCost + \sum_{\text{Buses}} GlobalBusCost + \sum_{\text{Transfers}} TransferCost$$

where $ColumnCost$ is calculated as follows:

$$ColumnCost = FuncUnitArea \times ConcurrencyPenalty + RegisterArea + MuxArea$$

A number of points should be noted. The dp-space configuration may imply an *illegal* solution, which arises because of the fixed local bus structure. The configuration may imply concurrent use of local bus resources by more than one functional unit, which is, of course, illegal. It is easy for COBRA to identify illegal solutions and it adjusts the cost of these solutions using the $ConcurrencyPenalty$ term in the cost function. Illegal solutions will therefore be expensive and annealing will tend to move away from them. At the moment, the cost function does not incorporate any mechanism for avoiding columns of different height, however, it is envisaged that such a feature will be added in the future.

The global bus term in the cost function takes into account the number of logical global buses allocated. This influences potential global wiring (if required), the complexity of the column multiplexers and also the number of ports on the global register files.

The $\sum_{Transfers} TransferCost$ term is introduced in an attempt to minimise the use of the allocated logical global buses. This does not directly affect the area of the solution during annealing, however, it may allow more efficient use of buses when logical buses are mapped to physical global buses during post-processing.

During annealing, one of three operations can be performed:

- Move tie line in time dimension – this is analogous to *scheduling operations*, it also affects register use and potentially logical global bus use.

- Move tie line in column dimension – this affects the binding of operators to columns and thus the allocation of FUs to columns. It also affects register use and logical global bus use.

- Move logical global bus transfer in time dimension – this effectively schedules intercolumn communications. It affects global bus use and register use.

Depending on the current state of the solution, annealing is biased to either perform an operator scheduling move, or biased to perform one of the other two moves. If the current solution has a "bad" schedule[1], a scheduling move is likely to be made in an attempt to improve the schedule. Similarly, if the current solution is deemed to have a "good" schedule one of the other two moves will probably be made. The biasing mechanism has been found empirically and is quite effective.

The annealing cooling schedule has also been empirically derived and typically produces good results in very acceptable run times.

7 Postprocessing Operations

When annealing terminates the final structure of the datapath has been optimised. COBRA then performs a number of postprocessing steps to complete datapath synthesis.

The first post processing step attempts to reduce the number of physical global buses. This is performed using both Prim's spanning tree algorithm[11] and Dijkstra's shortest path algorithm[11]. This has the potential of reducing the number of physical buses by routing logical bus communications over other physical global buses.

A second step determines which variables must be mapped to the global register files and evaluates the number of ports required on each global register file and column multiplexer. The final post processing step then assigns constants to ROM and performs explicit mapping of variables to registers using the well known left-edge algorithm. Note that annealing only determines *how many* registers will be required; the mapping step will always require the number of registers determined during annealing.

After all postprocessing is complete COBRA creates a VHDL netlist to describe the final datapath.

8 Results

COBRA is implemented in C and runs on Sun workstations. A number of results for the elliptical wavefilter benchmark are given in table 1 and comparisons are given where possible. The `cosine` results are for the fast discrete cosine transform example from [12].

The COBRA results were synthesised on a Sun SparcCenter 1000 running Solaris 2.3 and the ellip examples typically require around 8 minutes of CPU time. The synthesised architecture for the 19 cycle example with pipelined multiplier is shown in figure 10.

COBRA uses the same target architecture and layout style as the CASS system discussed in [3]. It was shown in [3] that the architecture and layout style had the potential to save up-to 50% area when compared to macrocell layouts synthesised by a conventional tool.

Examination of the CASS results in table 1 shows that, in terms of structural resources, they are inferior to those of other systems. It was initially thought that there was a structural overhead inherent in the architecture which would always require more registers,

[1]This is determined by examining the current number of allocated operators and the implicit operator concurrency.

System	Example	*	+	−	>	Regs	G-Bus/ Cols
non-pipelined multiplier							
COBRA	ellip18	2	3	–	–	12	3/3
	ellip19	2	3	–	–	13	2/3
	ellip21	1	2	–	–	12	2/3
	cosine20	3	3	2	–	14	3/3
CASS	ellip18	2	3	–	–	16	5/4
	ellip19	2	2	–	–	17	4/4
	ellip21	1	2	–	–	16	2/3
	cosine20	4	2	2	–	23	8/6
HAL[15]	ellip19	2	2	–	–	12	–
	ellip21	1	2	–	–	12	–
ELF[8]	ellip19	2	2	–	–	11	–
pipelined multiplier							
COBRA	ellip18p	1	2		–	13	3/3
	ellip19p	1	2	–	–	11	3/3
	cosine20p	2	2	2	–	12	3/3
PARBUS[4]	ellip19p	1	2	–	–	12	–
HAL[15]	ellip19p	1	2	–	–	12	–
ELF[8]	ellip19p	1	2	–	–	11	–

Table 1: Structural comparison of COBRA synthesis results with other systems

however, further examination of table 1 reveals that COBRA solutions are structurally comparable with those of other systems. This indicates that there is no significant structural overhead in using the presented architecture and that the COBRA results are close to optimum.

Even although the CASS results use more structural resources, [3] showed that they were physically superior to conventional techniques. Since COBRA uses the same target architecture and layout style as CASS, COBRA solutions will be physically superior to CASS solutions because they use less resources. Therefore by implication, COBRA has the potential to save even more area when compared to conventional systems. Further work is required to produce actual layout of COBRA datapaths to demonstrate conclusively the area saving, but these initial results are very encouraging.

It is interesting to note that the COBRA solution for the 19 cycle non-pipelined example uses one more adder than other systems. However it should be noted that COBRA uses 5 less registers and three less global buses than the CASS solution. The addition of an extra adder results in a *lower global cost* for the same target architecture, which illustrates that addition of redundant functional units can decrease the overall datapath cost.

9 Conclusions and Present Work

COBRA has been shown to produce solutions which, from a purely structural viewpoint, are as good as other systems, but which will be *physically* superior to that of conventional synthesis tools. Moreover, the global optimisation performed results in better solutions when compared to another system (CASS) synthesising onto the same architecture.

COBRA has proven the dp-space concept and has shown that the actual data flow defined by the given behaviour can be used as the core of a solution. In combination with the target architecture and layout style, dp-space optimisation has begun to address the two problems discussed in section 1. The overall effect is that the *global optimisation* in dp-space is performed on a cost function which will is closely related to the *physical* cost of the solution.

Present work is investigating the extension of the dp-space model to deal with conditional branches, loops, and pipeline synthesis as well as completing resource constrained synthesis. Architectural issues, particularly that of the global bus structure, are also being investigated with a view to introducing hierarchy into

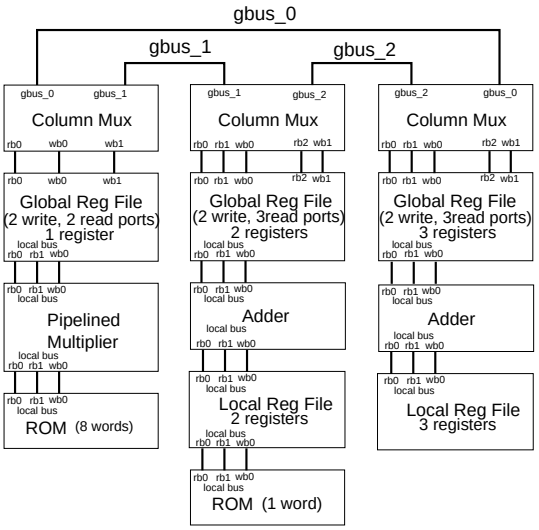

Figure 10: Synthesised architecture for the 19 cycle wavefilter with pipelined multiplier

the global communication layer. This may allow allocated global interconnect to be more efficiently utilised.

Other work is also required to tune the annealing schedule and investigate other optimisation methods in dp-space (eg. genetic algorithms). In addition, work is also required on interfacing with other CAD tools for back-end processing and solution verification.

References

[1] F. Brewer and D. Gajski. Chippie: A system for constraint driven behavioral synthesis. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, CAD – 9(7):681 – 695, July 1990.

[2] S. Devedas and A. R. Newton. Algorithms for hardware allocation in datapath synthesis. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, CAD – 8(7):768 – 781, July 1989.

[3] Andrew A. Duncan and David C. Hendry. DSP datapath synthesis eliminating global interconnect. In *Proceedings of Euro-DAC'93 European Design Automation Conference with Euro-VHDL'93*, pages 46 – 51, Hamburg, September 1993.

[4] Christian Ewering. Automatic high level synthesis of partitioned busses. In *Proceedings of 1990 IEEE International Conference on Computer Aided Design (ICCAD'90)*, pages 304 – 307, 1990.

[5] Catherine H. Gebotys and Mohamed I Elmasry. Global optimization approach for architectural synthesis. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, CAD –12(9):1266 –1278, September 1993.

[6] Hyuk-Jae Jang and Barry M. Pangrle. A grid-based approach for connectivity binding with geometric costs. In *Proceedings of the 1993 IEEE International Conference on Computer Aided Design (ICCAD'93)*, pages 94 – 99, 1993.

[7] David W. Knapp. Fasolt: A program for feedback driven data-path optimization. *IEEE Tansactions on Computer Aided Design of Integrated Circuits and Systems*, CAD – 11(6):677 – 695, June 1992.

[8] T. A. Ly, W. L. Elwood, and E. F. Girczyc. A generalized interconnect model for data path synthesis. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 168 – 173, 1990.

[9] Michael C. McFarland. Using bottom-up techniques in the synthesis of digital hardware from abstract behavioral descriptions. In *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, pages 474 – 479, June 1986.

[10] Michael C. McFarland. Reevaluating the design space for register-transfer hardware synthesis. In *Proceedings of 1987 IEEE International Conference on Computer Aided Design (ICCAD'87)*, pages 262 – 265, 1987.

[11] B. M. E. Moret and H. D. Shapiro. *Algorithms from P to NP*, volume Volume 1: Design and Efficiency. The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1991.

[12] J. P. Neil and P. B. Denyer. Simulated annealing based synthesis of fast discrete cosine transform blocks. In *Algorithmic and Knowledge Based CAD for VLSI*, chapter 4, pages 75 – 93. Institution of Electrical Engineers, 1992.

[13] B. Pangrle, F. Brewer, D. Lobo, and A. Seawright. Relevant issues in high–level connectivity synthesis. In *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pages 607 – 610, June 1991.

[14] A. Parker, P. Gupta, and A. Hussain. The effects of physical design characteristics on the area–performance tradeoff curve. In *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pages 530 – 534, June 1991.

[15] Pierre G. Paulin and John P. Knight. Scheduling and binding algorithms for high-level synthesis. In *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pages 1 – 6, 1989.

[16] J. M. Rabaey, C. Chu, P. Hoang, and M. Potkonjak. Fast prototyping of datapath–intensive architectures. *IEEE Design and Test of Computers*, pages 40 –51, June 1991.

[17] V.K. Raj and C.S. Patwardhan. Automated datapath synthesis to avoid global interconnects. In *VLSI DESIGN '91 Digest of Papers 4th CSI/IEEE International Symposium on VLSI Design*, pages 11 – 16, 1991.

[18] Josef Scheichenzuber, Werner Grass, Ulrich Lauther, and Sabine März. Global hardware synthesis from behavioral dataflow descriptions. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 456 – 461, June 1990.

[19] Jen-Pin Weng and Alice C. Parker. 3D scheduling: High level synthesis with floorplanning. In *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pages 668 – 673, June 1991.