# High-Level Synthesis of Structured Data Paths

*C. Mandal, R. M. Zimmer*
*Department of Computer Science, Brunel University*
*Uxbridge, U.K. UB8 3PH*
*Email: {Chittaranjan.Mandal, Robert.Zimmer}@brunel.ac.uk*

**Abstract**

In this paper we present a genetic scheduling algorithm to support the synthesis of structured data paths with the aim of producing designs with a predictable layout structure and conserving on-chip wiring resources. The data path is organized as architectural blocks (A-block) with local functional unit (FU), memory elements and internal interconnections. The A-blocks are interconnected by a few global buses. Our scheduling algorithm delivers the schedule of operations, the A-block in which each operation is scheduled and also the schedule of transfers over the buses, including transfers required to define variables in the basic block which remain live after its execution, all satisfying specified architectural constraints. The make up of the FUs in each A-block in terms of specific implementations of operators from a module database is also provided.

**Keywords**

Scheduling, Allocation, High-Level Synthesis (HLS), Genetic Algorithm (GA).

## 1 INTRODUCTION

Initial work on DPS led to the development of several scheduling and allocation techniques such as force directed scheduling [1], STAR [2], I.L.P. based data path synthesizers [3, 4], GABIND [5] and COBRA [6]. The reported techniques address optimizations for the data path with respect to its performance or the cost of the components used. Most of the above techniques produce designs with random interconnect structure liable to high cost of physical design. The problem of producing data paths which will have low layout cost is well recognized and is still an open one. Our approach is to impose restrictions on the structure of the data path so that it will have a predictable layout structure. Similar approaches have been adopted in STAR, GABIND and COBRA. These techniques use bus based architectures and require data transfers to take place over the buses. Our tool SAST (Structured Architecture Synthesis Tool) which is described here, synthesizes structured data paths but, as compared to other similar tools, permits better control over the architecture and uses a simple A-block. It uses a genetic algorithm to perform the optimization and obtains favorable results.

## 2 APPROACH AND SOLUTION TO THE PROBLEM

The architecture is characterized by a set of *A-blocks* housing a functional unit (FU), local memory elements and local interconnect buses which connect the inputs and output of the FU to the local storage cells and the global buses. A functional unit is a set of one or more, possibly pipelined, hardware operators, such that in any time step only one operation can be initiated and in any time step only one result can be generated. There are a set of *global buses* interconnecting the A-blocks to permit the transfer of data between them. Each A-block is connected to the global buses by means of a specific number of bi-directional *access links*. The layouts of an individual A-block and that of the overall architecture including the global buses are easy to predict. The number of A-blocks, access links and buses are provided as design parameters. The input is a dependency graph of operations. The operations have to be scheduled on FUs of A-blocks, avoiding execution and output conflicts. A schedule of data transfers to provide input operands for the operations and for specific assignments is also required.

A GA has been used for scheduling. The distinguishing features of the GA used are an algorithmic crossover and a diversity sustaining replacement scheme. A certain number of time steps within which the schedule is to be obtained is specified. A schedule requiring extra time steps attracts a penalty. The cost assigned to a solution is $C = (penalty)(extra\ \ time\ \ steps) + (cost\ \ of\ \ FUs)$. With probability $p_{s_i} = \dfrac{C_{max} + \delta - C_i}{N_{sols}(C_{max} + \delta) - \sum_i C_i}$, where $\delta \geq 0$, $C_i$ is the cost of solution $s_i$, $C_{max}$ is the maximum solution cost in the current population and $N_{sols}$ is the number of solutions in the population, a solution $s_i$ is chosen for crossover. Low cost solutions are selected with higher probability.

The times at which operations and transfers of the new solution are to be preferably scheduled are inherited from the two parent solutions. These attributes do not guarantee a feasible solution but are used to guide the completion algorithm, following the inheritance step, for obtaining a feasible solution. This scheme also alleviates some of the problems of the relatively small population size that is used. The completion algorithm is essentially a list scheduling algorithm employing a heuristic. The heuristic is based on the sum of successors computed for each operation $o_i$, defined as $w_i = \sum_{o_j \succ o_i}(w_j + W)$ where $o_j$ is a successor of $o_i$ and $W$ is a fixed positive value. The main data structures used in the completion algorithm are a pair of lists, the *ready* list and the *active* list for operations and another such pair of lists for assignments. Ready operations or transfers are introduced into the appropriate ready list and transferred to the corresponding active list from time to time. The sum of successors heuristic (SOSH) is used for scheduling from the active list. An operation is chosen randomly with a probability proportional to its successor weight. A stochastic choice is made to avoid excessive bias to a particular decision. In a time step it is attempted to schedule an operation on the inherited FU, failing which other FUs are considered to obtain better utilization of FUs. If an operand is not locally present in the A-block where an operation is being attempted to be scheduled then it needs to be transferred in from another A-block where it is available, in the current or a preceding time step over an available global bus and through an available access link of the source and destination A-blocks. Priority is given to the inherited time and the source A-block for transferring an unavailable operand into the A-block. Once a value is transferred into an A-block it continues to be available there. Operations and assignments are transferred into the active list based on their inherited schedule times. Operations may also be transferred to the active list based on SOSH, stochastically.

All solutions generated stay in the population for at least one iteration. The solutions to be replaced are essentially chosen at random. However, a scheme has been used at the same time to retain the best solutions and also maintain a diversity of FU configurations in the population. In order to retain low cost FU configurations a fixed number of *buckets* of a certain capacity are used to retain solutions having the same FU cost, although they may differ in their solution costs. Solutions which are in these buckets do not get replaced by a newly generated solution. A solution generated with a new and better FU configuration will displace solutions from a bucket representing an inferior FU configuration.

Structured architecture synthesis tool (SAST) has been used to schedule the differential equation solver [1], fifth order elliptic wave filter (EWF) [6] and discrete cosine transform (DCT) [6] and satisfactory results within acceptable run times were obtained.

## REFERENCES


[1] P. G. Paulin and J. P. Knight, "Force-directed Scheduling for ASICs," *IEEE Transactions on Computer Aided Design*, June 1989.

[2] F.-S. Tsai and Y.-C. Hsu, "STAR, An Automatic Data Path Allocator," *IEEE Transactions on Computer Aided Design*, September 1992.

[3] B. Landwehr, P. Marwedel, and R. Doemer, "Oscar: Optimum Simultaneous Scheduling, Allocation and Resource Binding Based on Integer Programming," in *Proceedings of European Design Automation Conference*, pp. 90–95, 1994.

[4] M. Balakrishnan and P. Marwedel, "Integrated Scheduling and Binding: A Synthesis Approach for Design Space Exploration," in *Proceedings of the 26th ACM/IEEE DAC*, pp. 68–74, 1989.

[5] C. A. Mandal, P. P. Chakrabarti, and S. Ghose, "Allocation and Binding for Data Path Synthesis Using a Genetic Approach," in *Proceedings of VLSI Design '96*, pp. 122–125, 1996.

[6] A. A. Duncan and D. C. Hendry, "Area Efficient DSP Synthesis," in *Proceedings of the 1995 European Design Automation Conference*, pp. 130–135, September 1995.