Allocation and Binding in Data Path Synthesis Using a Genetic Algorithm Approach

C. A. Mandal
Dept. of Comp. Sc. & Engg.
Jadavpur University, Calcutta
W. B. 700 032, INDIA

P. P. Chakrabarti S. Ghose
Dept. of Comp. Sc. & Engg.
Indian Institute of Technology, Kharagpur
W. B. 721 302, INDIA

Abstract

A technique for allocation and binding for data path synthesis (DPS) using a Genetic Algorithm (GA) approach has been developed. The proposed genetic algorithm uses a non-conventional crossover mechanism, relying on a novel force directed data path binding completion algorithm. The proposed technique has a number of features such as acceptance of some design parameters from the user, use of a bus based topology, use of multi-port memories and provision for multi-cycling and pipelining, among other features. The results obtained on the standard examples are promising.

Keywords: Data Path Synthesis, Binding, Multi-port Memory, Genetic Algorithm, VLSI.

1 Introduction

Data path synthesis involves first scheduling of operations, and then allocation and binding which consists of several subtasks. These include determining the the mix of functional units, grouping variables and assigning these variable clusters to storage units, port assignment when multi-port memories are used in the design, mapping operations to the functional units and mapping transfers to buses, when buses are used. All are NP-hard problems. Several researchers have worked on DPS problems and many solutions have been proposed [1, 2, 3, 4, 5]. Some of the well known systems for DPS are Facet, HAL [2], Splicer [6], STAR [7], SAM [5] and Vital-NS [8, 9]. Here we present a GA based technique for allocation and binding for DPS. GAs are becoming increasingly popular as a tool for EDA applications and have been used for placement, routing and testing, among others.

In our approach the construction of the data path (DP) is guided by two design parameters, viz. the number of functional units and the total number of buses to be used therein. The former indicates the total number of sites where operations may be performed, while the latter indicates the total number of paths for carrying data transfers, in the context of allocation and binding. Other information like the cost of primitive hardware operators, etc. are also required. The output is the optimized RTL data path. Support for multi-cycle operations, use of memories and pipelined functional units in the data path is available. The interconnection style is bus based. The choice of a bus based scheme is motivated by the fact that in a data path comprising of components requiring high interconnectivity the bus based scheme is expected to require fewer number of active interconnect elements –the switches–, than a point to point scheme. Storage is implemented using *multi-port memories and register files* in addition to individual registers. A memory is a regular structure. By placing several variables in a single unit the number of independent sources and destinations of data is reduced. Presence of a *common source* of multiple data transfers is identified. The multiple transfers originating from this source may be routed through common buses to make use of existing links and switches.

The techniques which have been proposed in this paper have been tested on prevailing examples such as Facet, Differential equation solver and Elliptic wave filter. For the last two examples arithmetic pipelining has also been used. The results obtained have been encouraging. The next section explains the bus based DP structure after which our algorithm is described.

2 Data Path Structure

All the components are connected to one or more buses over which the data transfers take place. Multiple sources driving any part of the circuit need to be switched. The cost of the data path is the sum of the cost of the individual components. The components are broadly categorized in four groups, viz. functional units (f.u.), storage units, interconnection elements and interface components (such as the interface ports). The notion of a f.u. is similar to an arithmetic logic unit. We do not consider the cost of physically routing the wires.

The cost of a f.u. is taken as the sum of the costs of the primitive hardware operators for the operations that it implements. The cost of a memory is computed with the knowledge of the number of ports that it has and the number of memory cells that it houses. In order to bound the access time of the memory, the max-

imum number of cells in a memory is restricted to some predefined "small" number. The cost is computed as $c_m(n,p) = n(\alpha p + \beta) + \gamma p$, where α, β and γ are constants, and c_m is the cost of a memory with n cells and p ports. β is the cost of each cell. α is the cost of the access logic per port per cell. γ is the cost of the driver and other logic per port. A switch is required whenever there are more than one sources driving any input in the circuit. The cost of a switch, which is often implemented using CMOS pass transistors is taken to be a predefined constant.

3 Inputs for Allocation and Binding

The scheduled data flow graph (SDFG), the cost of primitive operators, the design parameters and the basic block decomposition [10] of the operations and transfers are the inputs which are required. The SDFG format consists of two parts. Each part, viz. the operations and the transfers, are listed time step by time step. An operation is identified by its type, expressed as a non-negative integer. The format is <*op_type*> <*mc_flag*>. A transfer is identified by its source and destination. A source or destination may be either an *input/output of an operation*, a *behavioural variable* or an *interface port*. The requirement of multi-cycling for both operations and transfers is expressed through an associated flag. The format of a transfer is <*source*> <*destination*>. A source or destination is formatted as <*sdm_type*> <*index_1*> <*index_2*>. Index_2 is used only to represent the input/output of an operation. Index_1 indicates the particular operation itself. For variables and interface port the identification is also through index_1 by the index of the variable or the port.

A primitive operator, which may be fully combinational or pipelined, is the physical implementation of an operation in the behavioural specification (like adder, etc.). An operation is required to be mapped to one primitive operator. It is assumed that module selection from module library has been done. A pipelined operator also carries the information about the number of single time step pipe stages that it has. This information is necessary to avoid output conflict of operations that may be mapped to the same f.u., in different time steps. An operation and the corresponding operator are identified by means of a unique *operation id*.

4 GA for Allocation and Binding

The steps followed in the GA for solving the problem are as follows:

Initial population generation: A population of initial data paths created by randomly generating feasible bindings. Care is taken to ensure that a solution generated is always a feasible one.

Replacement policy: The basic replacement policy is designed to ensure that all solutions generated stay in the population for at least one iteration. This is done by introducing all the solutions generated through crossover during one iteration of the GA into the population, replacing an equal number of existing solutions. The offsprings are not immediately placed in the main population, but are stored in an adjoint pool. The basic replacement policy is to displace the lowest cost solutions with some additional considerations.

First we have made provision for the retention of a minimum number of solutions having the k^{th} best cost, $k > 1$ for a fixed value of k. The second provision is specifically aimed at maintaining a good diversity of memory configurations in the population of solutions so that some minimum number of distinct memory configurations in the population is eventually built up. Solutions with the best memory configurations are kept track of.

Parent selection: Each offspring is generated from two parent solutions chosen from the population with a slight bias towards coercing some crossovers between solutions which are either genetically close or which are more fit to keep a check on the amount of type II deceptability during crossover, to some extent. To implement crossover between better fit parents a list of solutions whose cost is less than some threshold, determined on the basis of the distribution of the solution costs in the population, is maintained. Solutions can be picked up from this list at random for crossover. Genetic closeness is difficult to determine. Some amount of closeness is determined by grouping solutions having the same memory configuration. Such solutions differ only with respect to the operation and transfer bindings. Normally a solution is selected only once during one generation.

Crossover: An algorithmic crossover has been used. First an inheritance plan is created from which a a tentative partial structure of the data path (TPDP) is first constructed, which is used to evaluate the actual bindings. The TPDP is updated with each actual binding. The heart of the crossover is a *completion algorithm* which takes the TPDP and other inputs to generate a complete data path.

Design representation: A record structure of three fields, one for each class of binding decisions, is used. Each field is a structured array. For each operation and each transfer, the corresponding field indicates the f.u. index or the bus index to which it is bound, respectively. The binding of a variable indicates the memory index and the number of ports that it has.

Stopping criterion: The GA is run for a certain minimum number of iterations. For every improvement it is run for a fixed number of additional iterations till no more improvement takes place.

5 Details of Crossover

The basic steps for the crossover are as follows:

• Determining the specific bindings of each parent solution which should participate in the crossover. The idea is to give preference to bindings which are likely to reduce interconnect overhead. We explain the gradation of transfers, operations and variables are graded in a similar manner. The sources or destinations that are connected to a bus are grouped into two sets, *source* and *destination* which are not disjoint, in general. The *access frequency* of a member in either set is defined as the number of transfers that use this member. A member is considered to be extraneous if *(access frequency of member)* $<$ $(1-\alpha_t)$ *(average access frequency)* $+$ α_t *(minimum access frequency)*, where α_t is a constant. If $\alpha_t \approx 1$ then the minimum access frequency receives more weight and most of the members qualify. If $\alpha \approx 0$ then the condition becomes tighter and few members qualify. The appropriateness of a transfer is measured with respect to either its source or destination or both, in which case the source or the destination or both should be should be non-extraneous.

• Obtaining a correspondence between the data path structures of the two parent solutions. The representation of a solution is not unique. For example, if the solution uses n_f f.u.'s, n_b buses and n_m memory units then the same solution can be expressed in $n_f!n_b!n_m!$ ways. Therefore, before proceeding with the crossover, a matching between the data path elements of the two solutions is obtained through a matching. Affinity between the DP elements is computed on the basis of the preferred bindings.

• The operations and transfers planned to be inherited from the parent solutions are now determined time step by time step. Only the operations and transfer marked favourable earlier are considered.

• The memories are formed partially by inheriting preferred variable bindings from the parents. After a memory has been partially formed additional variables are also packed in, if possible.

• Finally the offspring is generated using the inheritance plan and the tentative partial structure, time step by time step. At this stage the memory formation is completed. First the multi-cycle operations and transfers are bound to satisfy binding decisions taken in earlier time steps. Then operations and transfers in the inheritance plan which can be feasibly inherited in the current time step are processed. The remaining bindings are made using a *force directed completion algorithm* which we explain in the next section.

6 Force Directed Completion Algorithm

First the operations and then the transfers are bound. In this process the actual data path structure is obtained by augmenting and appropriately modifying the current partial structure. Self and other forces are computed for each candidate binding, the one with minimum force is taken. For operation binding we compute i) forces for mapping operations to f.u.'s and ii) lookahead forces for transfer to bus bindings which may take place as a result of a particular operation to operator binding. We explain below the first category of forces only. The following definitions will be useful in explaining the algorithm.

DEF is used with an actual binding.

LKY is a constant used with a likely binding,

ULK is used with an unlikely binding and

Total unmapped operations (U_o) is the number of operations in the current time step which are yet unmapped.

Total unmapped operations of type y (U_o^y) is the number of unmapped operations of type y.

Number of available f.u.'s (A_f) are those on which no operation has yet been mapped onto these f.u.'s.

Cost of operation type y ($C[y]$) The cost of a f.u. will increase by this amount to be able to implement an operation of type y.

Distribution graph of operations ($D_o[u,y]$) Distribution graph of operations of type y on f.u. u.

$$D_o[u,y] = \begin{cases} \text{if operation type } y \textit{ is present in } f.u.\ u \\ \quad \text{then} \quad -\dfrac{U_o^y}{A_f}.C[y] \\ \quad \text{else} \quad \begin{cases} \text{if likely to be present} \\ \quad \text{then} \quad \dfrac{U_o^y}{A_f}.C[y].\text{LKY} \\ \quad \text{else} \quad \dfrac{U_o^y}{A_f}.C[y].\text{ULK} \end{cases} \end{cases}$$

While considering an operation (in column r of the specification) of type y to map on f.u. u two types of forces are computed. These are the self force and other forces. These force are exerted on f.u. u and other f.u.'s $l, l \neq u$.

The forces on f.u. u are as follows:

$$\text{self force :} \qquad F_{os}^{uy} = \frac{A_f - 1}{A_f}.D_o[u,y].$$

The force due to other operations of type y_o is as follows:

$$\text{other force :} \quad F_{ox}^{uy_o} = \begin{cases} \text{if } y \neq y_o \quad -\dfrac{U_o^{y_o}}{A_f}.D_o[u,y_o] \\ \text{else} \qquad -\dfrac{U_o^{y}-1}{A_f}.D_o[u,y] \end{cases}$$

The forces on f.u. $l, l \neq u$ are as follows:

$$\text{self force :} \qquad F_{os}^{ly} = \left(\frac{-1}{A_f} + \frac{U_o^y}{A_f(A_f-1)}\right).D_o[l,y],$$

$$\text{others :} \qquad F_{ox}^{ly_o} = \frac{U_o^{y_o}}{A_f(A_f-1)}.D_o[l,y_o].$$

After operation binding is through, transfer bindings are processed in a similar manner. Transfer forces are computed to minimize the formation of new links and switches. The computation of these forces is similar to the computation indicated above. Instead of pending operations and available f.u.'s, pending transfers and available buses are considered, respectively.

7 Experimental Results

The techniques described in this paper have been tested on some well known examples. The results have been tabulated along with results of some other well known systems available with us, which include Facet, Diffeq. and elliptic wave filter. In the tables the result of our system have been labeled GABIND. Solutions labeled (1) and (2) use the standard schedule the third result is using another schedule. Name of the system, number of switches for multiplexing, number of links, number of storage cells and memory configuration are given column wise. Column four represents the number of registers used; for our system it represents the number of distinct registers or memory cells, normalized for comparison with other systems, where constants and some some other storage elements are usually not included while counting the registers.

System	num. mux. swt.	num. link	num. cell	mem. config.; $< x, y, z >$ $\Rightarrow$ y x-port memories with a total of z cells.
Facet 3 f.u.'s 4 time steps.				
Facet	11	—	8	—
Splicer	8	—	7	—
HAL	6	13	5	—
Vital-NS	6	12	5	—
GABIND(1)	8	10	5	$< 2,3,3 >, < 1,1,2 >$
GABIND(2)	7	11	6	$< 2,3,4 >, < 1,2,2 >$
GABIND	5	11	6	$< 2,3,4 >, < 1,2,2 >$
Diffeq. using 2-cycle multipliers and 5 f.u.'s, $(2\star, +, -, <)$, 4 time steps.				
Splicer	11	—	6	—
HAL	10	25	5	—
Vital-NS	12	22	5	—
GABIND	8	18	5	$< 2,5,7 >, < 1,1,1 >$
Diffeq. $(1 \star (pipe), 1\ alu.)$, 8 time steps.				
HAL	13	19	5	—
Vital-NS	13	17	5	—
GABIND	7	13	5	$< 2,2,5 >, < 1,2,1 >$
Elliptic wave filter 17 time steps, $(1 \star (pipe), 2+)$ for STAR and GABIND, others $(2 \star (pipe), 3+)$, as available.				
HAL	31	—	12	—
SAM	31	50	12	—
Vital-NS	32	50	11	—
STAR	26	—	11	—
GABIND	28	29	14	$< 2,5,14 >, < 1,1,1 >$
Elliptic wave filter $(1 \star (pipe), 3+)$, 18 time steps.				
HAL	34	—	12	—
SAM	30	40	12	—
Vital-NS	33	40	10	—
GABIND	31	35	11	$< 2,6,11 >, < 1,1,1 >$
Elliptic wave filter $(1 \star (pipe), 3+)$, 19 time steps.				
HAL	26	—	12	—
SAM	21	40	12	—
Vital-NS	29	40	11	—
STAR	28	—	—	—
GABIND	27	33	14	$< 2,4,12 >, < 1,2,3 >$

8 Conclusions

We have developed a technique for allocation and binding for data path synthesis using a GA approach. We believe there is some scope of improvement in the way the way GA has been used to solve the problem of allocation and binding. The developed system has a number of features such as specification of some design parameters, use of a bus based topology, use of multi-port memories and provision for multi-cycling and pipelining, among other features. The results obtained on the standard examples have been promising.

References

[1] A. C. Parker, "Automated synthesis of digital systems," *IEEE Design and Test*, November 1984.

[2] P. G. Paulin and J. P. Knight, "Algorithms for high-level synthesis," *IEEE Design & Test*, December 1989.

[3] F. Brewer and D. D. Gajski, "Chippe: A system for constraint driven behavioural synthesis," *IEEE Trans. on CAD*, pp. 681–695, July 1990.

[4] S. Devadas and A. R. Newton, "Algorithms for hardware allocation in data path synthesis," *IEEE Transactions on Computer Aided Design*, vol. 8, July 1989.

[5] R. J. Cloutier and D. E. Thomas, "Thc combination of scheduling, allocation and mapping in a single algorithm," in *Proceedings of the 27th ACM/IEEE DAC*, pp. 71–76, June 1990.

[6] B. M. Pangrle, "Splicer: A heuristic approach to connectivity binding," *Proceedings of the 25th ACM/IEEE Design Automation Conference*, 1988.

[7] F.-S. Tsai and Y.-C. Hsu, "Star, an automatic data path allocator," *IEEE Trans. on CAD*, September 1992.

[8] A. Kumar, *A Versatile Data Path Synthesis Approach Based on Heuristic Search.* PhD thesis, I.I.T. Delhi, January 1993.

[9] A. Kumar, A. Kumar, and M. Balakrishnan, "Heuristic search based approach to scheduling, allocation and binding in data path synthesis," in *Proceedings of VLSI Design '95*, pp. 75–80, 1995.

[10] A. V. Aho, R. Sethi, and J. D. Ullman, *COMPILERS Principles, Techniques and Tools.* Addison-Wesley Publishing Company, June 1987.