SAST: AN INTERCONNECTION AWARE HIGH LEVEL SYNTHESIS TOOL

C. Karfa[1], J.S.Reddy[2], S.Biswas[3], C.R.Mandal[4], D.Sarkar[5]

Abstract

Today's VLSI technology allows us to construct large, complex systems with million transistors on a single chip. Most of the existing high level synthesis systems give more priority to optimization of area, power, resource and time steps compared to interconnection cost, whereas the later becomes predominant with the technology scaling and increase in complexity. Further, field programmable gate arrays (FPGA) are now becoming attractive platform for prototyping. Programmable devices tend to have limited wiring resources between the data path elements. This work is concerned with the development of a CAD tool for HLS named, "Structured Architecture Synthesis Tool (SAST)", which incorporates structured architecture generation with special emphasis on optimization of interconnect area. The too takes a behavioral description written in 3-address form and generates synthesizable RTL codes with scripts for compliance with standard design tools like Synopsys, Magma etc.

Key Words: *High level synthesis*, *Structure Architecture*, *Interconnection*, *Genetic Algorithm*.

1 Introduction

The high level synthesis (HLS) problem consists of translating a behavioral specification into an register transfer level (RTL) structural description containing a data path and a controller so that the data transfers under the control of the controller exhibit the specified behavior. Thus, the HLS problem can be formulated as follows: Given a functional specification in the form of an algorithm, and a set of constraints, synthesize an RTL equivalent of the algorithm comprising a data path composed of modules obtained from a

[1] C Karfa is an MS student of Dept. of Comp. Sc. & Engg, IIT Kharagpur. Email: ckarfa@yahoo.co.in

[2] J.S.Reddy was an M.Tech. student in the Dept. of Comp. Sc. & Engg, IIT Kharagpur. Presently he is with Intel Corp. Bangalore, INDIA. Email : srinivas_reddy_j@yahoo.com.

[3] S.Biswas is a PhD student Dept. of Comp. Sc. & Engg, IIT Kharagpur. Email: santoshbiswas402@yahoo.com

[4] C.R.Mandal is an Associate professor of Dept of Comp Sc.& Engg, IIT, Kharagpur

[5] D.Sarkar is a Professor of Dept of Comp. Sc. & Engg, ,IIT Kharagpur.

module library, and a controller, such that the RTL equivalent exhibits the same behavior as of the functional specification of the input. High-level synthesis is divided into several steps, namely, preprocessing, scheduling, allocation and binding followed by controller design [Gajski & Dutt (‘92)].

Several works in this field are reported in the literature. Force directed scheduling [Paulin & Knigh (’89)] attempts to minimize the cost of hardware operators while trying to find a schedule within a specified number of time steps. The scheduling algorithm called dynamic loop scheduling (DLS) [Rahmouni (’95)] is more suitable for scheduling control-oriented design. [Gupta & Gupta (’02)] described a technique for elimination of dynamic common sub-expression by aggressive speculative code motions.

A number of systems like HAL [Paulin & Knight (’89)], STAR [Tsai & Hsu (’92)], SAM [Cloutier (’90)], GABIND [Mandal & Chakrabarti (‘96)] and SPARK [Gupta & Gupta (’03)] are now available to support the HLS of digital systems. Over the last several years, these systems have evolved from elementary systems producing non-optimized data paths to more sophisticated systems generating data paths optimized with respect to area, time, power and testability. With the advancement of the VLSI circuit technology, a rapid scaling of the feature size has been performed. Device scaling implies that the circuit performance will be increasingly determined by the interconnection performance. For instance, interconnection contributes 50 percent of total delay in 0.35 micron technology whereas it is expected to rise up to 70 percent in 0.25 micron technology. Thus, interconnections are expected to play the most critical role in design of chips in deep sub-micron technologies. The development of FPGA has also taken place around this time. These are now becoming attractive platforms for prototyping designs, simulation acceleration, hardware in loop simulation etc. To the best of our knowledge most HLS tools, however, produce optimized designs in terms of resources, time steps, power, area etc. without much emphasis on reduction of long and random interconnections.

The aim of this work is to design a CAD tool named “SAST ” for facilitating an automated design environment for High-level synthesis. The main contribution of this paper is that of “Structured Architecture” (SA) for HLS that is described in the next section. A genetic algorithm based scheduler developed by the authors supports this structured architecture. The SA enabled the avoidance of random interconnection between the architectural components. It also uses a few numbers of buses for interconnection. So generated designs are easy to implement on programmable device such as FPGA.

The paper is organized as follows. Section 2 gives the introduction of the structured architecture for the HLS. Section 3 describes the implementation of the basic steps used in SAST. Section 4 summarizes the experimental results. Section 5 concludes by underlying the effectiveness of SAST.

2 **Structured Architecture (SA)**

The schematic diagram of the SA is shown in figure 1. The data path is organized as architectural blocks (A-block). Each A-block has a local functional unit (FU), local storage and internal interconnections, as shown in figure 1. A few global buses interconnect the A-blocks. Each A-block is connected to the global buses by means of a specific number of access links. SA also permits the use of memories as architectural components. These are connected to the global

buses like the A-blocks. The structure of the data path is characterized by a set of architectural constraints like the number of A-blocks, the number of global memories, the number of global buses interconnecting the A-blocks, the number of access links or access width which connect an A-block to the global buses and the maximum number of writes per time step to storage locations in an A-block. These structured data paths avoid random interconnects between data path elements. Each A-block has a simple implementation. This makes the generated design easy to implement on programmable devices such as FPGA. Global memories help improve the availability of operands and relieve the storage requirement in individual A-blocks. For example a signal-processing algorithm for which the hardware is being constructed may require arrays. The availability of global memories as an architectural component makes it possible to construct data paths for the hardware implementation of these algorithms easily by storing the array elements in global memory.

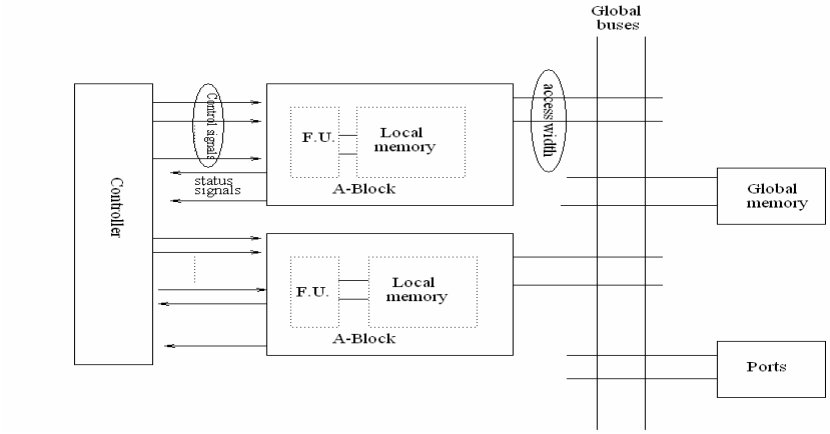

Figure 1: Schematic of Structured Architecture

3 Implementation

The design flow used by "SAST" comprises the following steps:

1. Preprocessing: Translation of the input control data flow graph (CDFG) [5] to an intermediate representation (IR) and calculation of necessary information for scheduling.
2. Schedule of the operations and the transfer of variables in minimum number of control steps for a given architectural specification. The scheduler accomplishes functional unit formation.
3. Allocation and binding of variables to registers.
4. Data path generation from the schedule of operations, bus transfers and the variable mapping to the registers.
5. Generation of synthesizable Verilog code (RTL).
6. Generation of scripts for gate level synthesis using commercial design tools like Synopsys DA, Magma etc.

This section deals with a detailed description of each step.

3.1 Preprocessing

In translating a CDFG to an RTL design, one needs to know the data dependency information in each basic block for scheduling and data path allocation. Also, information of incoming and outgoing variables of each basic block is re-

quired for scheduling. This information is computed in the preprocessing step. Following set of variables are used in computing live incoming and outgoing variable sets for each basic block i,

use_i : set of variables whose values used in i prior to any definition of the variable.

def_i : set of variables defined in i prior to any use of that variable in i.

in_i: set of variables live at the entry point of i.

out_i: set of variables live at the exit point of i.

Procedure described in [2] to compute these values are used in SAST.

3.2 **The GA Based Scheduling Algorithm**

A genetic algorithm (GA) based scheduling is implemented which supports the synthesis of structured data paths. Variables are meant to reside at specific storage locations in specific A-blocks. So, it may require transferring one variable from one A-block to another during scheduling. The scheduling algorithm is versatile enough to handle multi-cycle operations, pipelined operators and multiple implementations of an operation. SAST uses a slow adder when there is slack time available that enables reduction in cost of the FUs and a fast adder otherwise. The scheduler of SAST delivers the following:

- Schedule of operations i.e. assign time step to each operation,
- The A-block in which each operation is scheduled,
- Schedule of all transfers over the global buses satisfying the architectural constraints, and
- Composition of the functional unit (FU), in each A-block, in terms of the specific implementations of the operators from a module database.

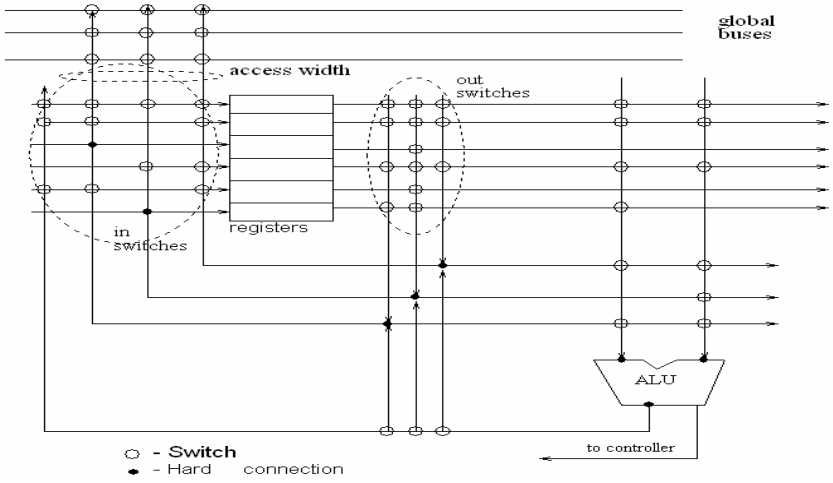

Fig2: Data path of functional, storage and interconnection units in an A-block.

A brief overview of the GA is as follows. In view of the complex nature of the problem, a structured solution representation has been used, as against a simple bit string. An initial population of solutions is generated at random. New solutions are obtained by inheriting values of decision variables from parents solutions, picked up from the population. But the operations may not inde-

pendent of each other. So the resulting solution representation could correspond to an infeasible solution. As a result a completion algorithm has to be used to obtain a feasible solution from the available solution representation obtained through crossover or by setting the attributes at random while generating the initial population of the solutions. A scheduling heuristic has also been used with the completion algorithm and this has been found to improve the performance of the GA. A population control mechanism is used to sustain diversity in the population. At the same time solutions with overall good and partial fitness are retained. The GA is run up to a fixed number of iterations and this serves as the stopping criterion. The last improvement in solution cost (i.e. when the best solution is obtained) usually occurs well before all the iterations are completed.

3.3 **Register Allocation and Binding**

The scheduler binds operations to the functional unit in an A-block and data transfers over the global buses. The remaining tasks in the data path allocation are register optimization and data path construction in each A-block from the scheduled output. Register optimization computes the minimum number of registers required in the data path in each A-block. Register allocation and binding consists of several sub steps explained below.

• Lifetime analysis of variables in an A-block from the schedule of operations and the bus transfers over the global buses,
• Constructing the compatibility graph from the lifetimes of variables in an A-block, and
• computing the minimum number of registers required in each A-block from the compatibility graph.

3.3.3 Clique partitioning

The heuristic given in [Ray('93)] has been employed to find minimal number of cliques from the compatibility graph. So, the total number of registers needed in A_i is the total minimal number of cliques found out from the clique partitioning method. This method also maps the variables to registers by mapping each the cliques to a register.

3.4 **Data path generation**

Interconnection topology is built from the schedule of operations and variable transfers over the global buses. Figure 2 shows the RTL data path in an A-block SAST allows that the data over the global buses can be fed as inputs to the FU and to the set of registers through switches. Also, output from the FU can be transferred over the global buses and to the registers. Switches "in" and "out", as depicted in figure2, correspond to the data transfers over the global buses between the A-blocks and the I/O ports. The "in" switches are enabled for the set of registers, which take values from the global buses. "Out" switches are enabled for the set of registers whose value is transferred over the global buses.

3.5 **Verilog code generations**

Final stage in the SAST is to generate the synthesizable verilog code (RTL) for the Data path and the Controller, which generates control signals for each A-block and interconnection switches. Finally, scripts are generated based on structural design constraints like clock frequency, input-output delay, area

etc. A commercial CAD tool like Synopsys DA, MAGMA, and Cadence PKS etc. for gate level synthesis can directly import it.

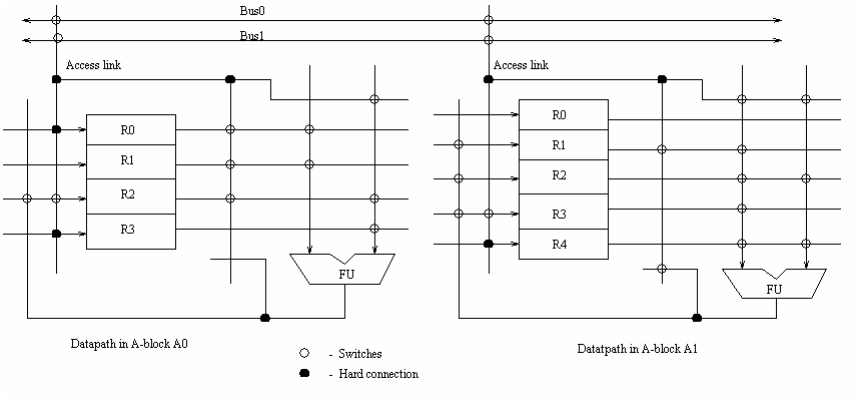

Figure 3: Data path in the A-block A_0 and A_1.

4. **Experimentation**

In this paper, two benchmark problems Differential Equation Solver (DIFFEQ) and 5th order Elliptic wave filter (EWF) have been solved using the proposed approach. Details of scheduling, register allocation and binding have been tabulated for DIFFEQ. The architectural parameters for the DIFFEQ are: 3 A-blocks, 2 global buses, 1 access link (access width) and 2 memory ports per A-block. The scheduled output with the control steps for each basic block is shown in table 1. Variable mapping to registers in the A-blocks is depicted in table 2.

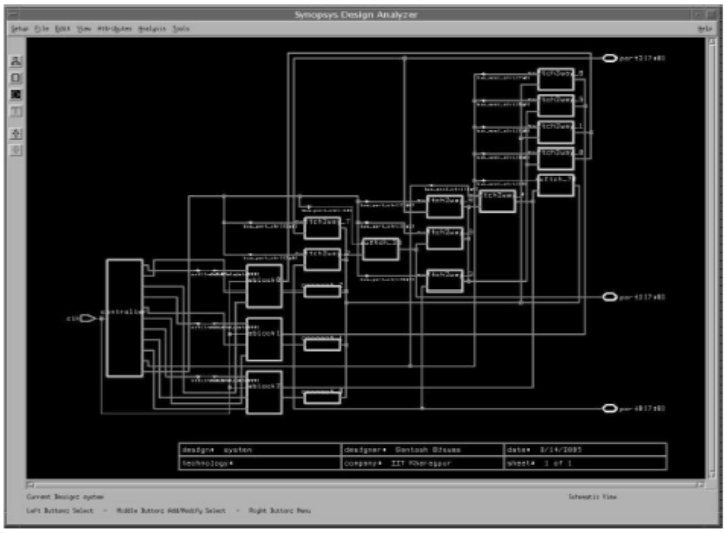

Figure 4: Synopsys DA output for EWF.

The data paths from the scheduled basic blocks and the register allocation phases are given in figure 3 for A-blocks A0 and A1. Data path for A-block A2 is not shown here for brevity. In control step 5 of the basic block B1, v2 is

output from the FU in A-block A_1 and it is transferred over the global bus in that control step to A_2. This is shown in table 1. Further, v2 has no usage in A_1 and it is not considered for register allocation in A_1. Here multiplication operations take 2 cycles to execute. In the table 1, $=_r$ p, $=_w$ p indicates read from port and write to port respectively. In table 2, [R] are [V] are order.

Basic block	Time	Bus transfer	Schedule of operations		
			A0	A1	A2
I	1	p1→dx, p3→a	(0, =rp)	(4, =rp)	
	2	p2→ x, p1 →y	(1, =rp)	(2, =rp)	
	3	p2→u, x(0)→1			(3, =rp)
B1	1	3 (0)→1		(1, *)	
	2	dx (0)→1, 2		| *|	(0, *)
	3	y (1→0)	(4, *)	(2, +)	| *|
	4	v0 (2→1)	| *|	(3, *)	(7, *)
	5	v2(1→2)	(6, *)	| *|	| *|
	6	v6 (2→1)	| *|	(9, +)	(5, -)
	7	v5(0→2)			(8, -)
C1	1			(0, <)	
B2	1	x(1→p1)		(0, = wp)	
	2	u(2→p1) y(1→p2)		(1, =wp)	(2, = wp)

Table 1: Schedule for differential equation solver

A-block	#Registers	Variables	Register
A_0	3	[dx], [x, v3, v5], [y]	[R0], [R2], [R3]
A_1	5	[a], [y], [x], [3, v1, v6], [dx, v0]	[R0], [R1], [R2], [R3], [R4]
A_2	3	[u, v4], [dx, v2, v6], [v0]	[R0], [R1], [R2]

Table 2: Variable mapping to registers in the A-blocks of DIFFEQ.

System	Time steps	# +	# *	Bus, Ablk, Accl	# Reg.
Elliptic wave filter scheduled in 18 steps using multi-cycle multipliers					
SAST	18	3	2	1, 3, 1	14
COBRA	18	3	2	3, 3, -	12
CASS	18	3	2	5, 4, -	16
HAL	18	3	2	—	12
PSGA SYN	18	3	2	—	10
Elliptic wave filter using pipelined multipliers					
SAST	18	2	1	2, 3, 2	13
COBRA	18	2	1	3, 3, -	13
HAL	18	3	1	—	12
PSGA SYN	18	3	1	—	10
SAM	19	2	1	—	12
STAR	19	2	1	—	11
PARBUS	19	2	1	—	12

Table 3: Comparison of results with a few other synthesis tools.

Comparison of results of the current method has been done with ones reported in the literature for EWF, namely SAM, STAR, HAL, COBRA, etc. and is illustrated in Table3. It can be observed that the solution given by the SAST is comparable to other tools in terms of time steps, resources, A-Blocks, registers but has reduced the number of global buses.
Further, RTL generated by SAST has been synthesized using Synopsys DA with 0.18 Micron CMOS9 library of National Semiconductor Corp., USA. Result has been illustrated for EWF in Figure4. The structured nature of the interconnection can be noted from the figure4. The ratio of the interconnection overhead to that of the cell area is 1.1 percent (reported by Synopsys DA) for EWF.

5 Conclusions

This work is concerned with the development of a CAD tool SAST, for synthesizing structured architectures with a simple and predictable layout structure and generates synthesizable RTL codes. It uses GA for scheduling. SAST is able to handle multiple implementations of operations varying in speed, including multi cycle and pipelined implementations. In all cases the FU cost of designs synthesized by SAST are comparable with those of other systems. Important feature of this work is that random long-distance interconnects between data path elements in the synthesized design are avoided by considering the structured architecture for synthesis. SAST also uses a very few number of global busses. Thus it reduces total interconnection area.

References

[1] Aho,A.V. and Sethi R and Ullman, J. D. (1987) Compliers: Principles, Techniques and Tools, Addison Wesley publishers, June.
[2] Mandal, C. A. and Chakrabarti, P. P. and Ghose, S (1996), Allocation and binding for data path synthesis using a genetic approach, in Proceedings of VLSI design'96, pp.122 –125.
[3] Gajski, D. D. and Dutt, N. D. and Wu, Allen C-H and Steve Y-L Lin (1992), High Level Synthesis: Introduction to chip and System Design, Kluwer Academic Publishers.
[4] Tsai, F. S. and Hsu, Y. C. (1992), Star: An automatic data path allocator, IEEE Trans. on CAD, September.
[5] Ray, Jay. (1993), Parallel Algorithms for High level Synthesis, Ph. D dissertation, University of Cincinnati, February.
[6] Paulin, P. G. and Knight, J. P. (1989), Force Directed Scheduling for ASIC's, IEEE Transactions CAD, vol 8, No 6, June.
[7] Cloutier R. J. and Thomas, D. E. (1990), The Combination of scheduling, allocation and Mapping in a single algorithm, 27th Design Automation Conference, October.
[8] Rahmouni. M and Jerraya. A. A. (1995), Formulation and evaluation of scheduling techniques for control flow graphs. In proceedings of European Design Automation Conference'95, Brighton.
[9] Gupta, S. Dutt, N. Gupta, R. and Nicolau, A. (2003), SPARK: A High-Level Synthesis Framework For Applying Parallelizing Compiler Transformation, Proceedings of the 16th International Conference on VLSI Design.