SAST: An Interconnection Aware High-level Synthesis Tool

Presented by
Chandan Karfa

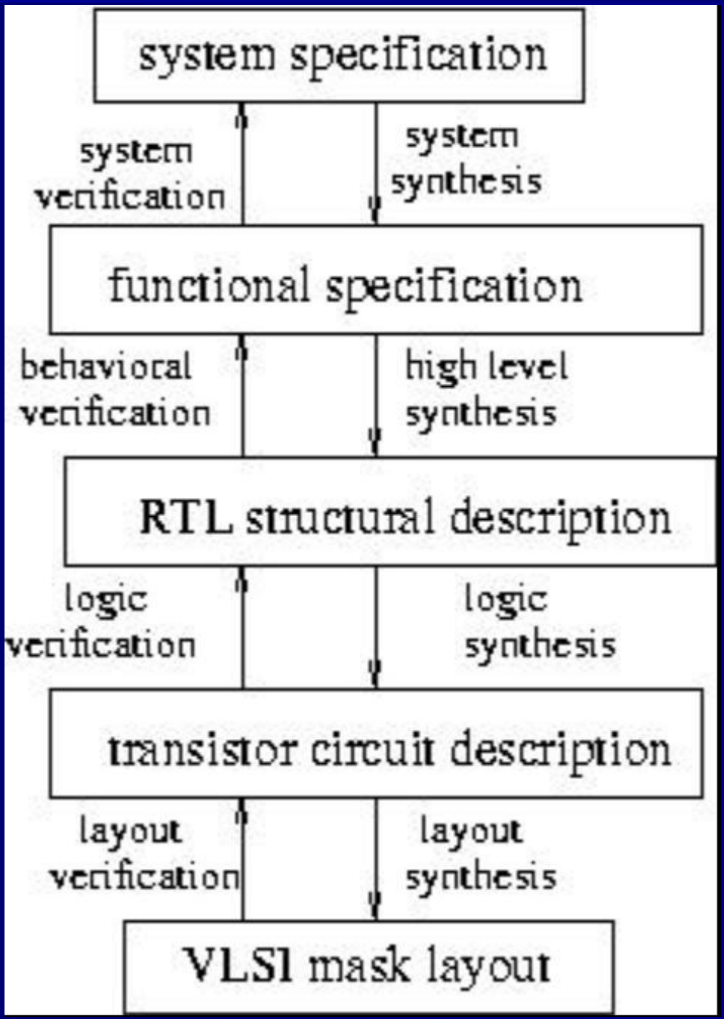
Tool developed by
Chandan Karfa, Jala Srinivas Reddy, Murali Krishna G, Chittaranjan Mandal, Dipankar Sarkar, Santosh Biwsas.

Dept. of Computer Sc. & Engg.,
IIT, Kharagpur.

Outline of the Presentation

- Introduction to High-level Synthesis.
- Structured Architecture
- Contribution of our work
- SAST Synthesis Flow
- Experimental Results
- References

Synthesis Flow



High-level Synthesis

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

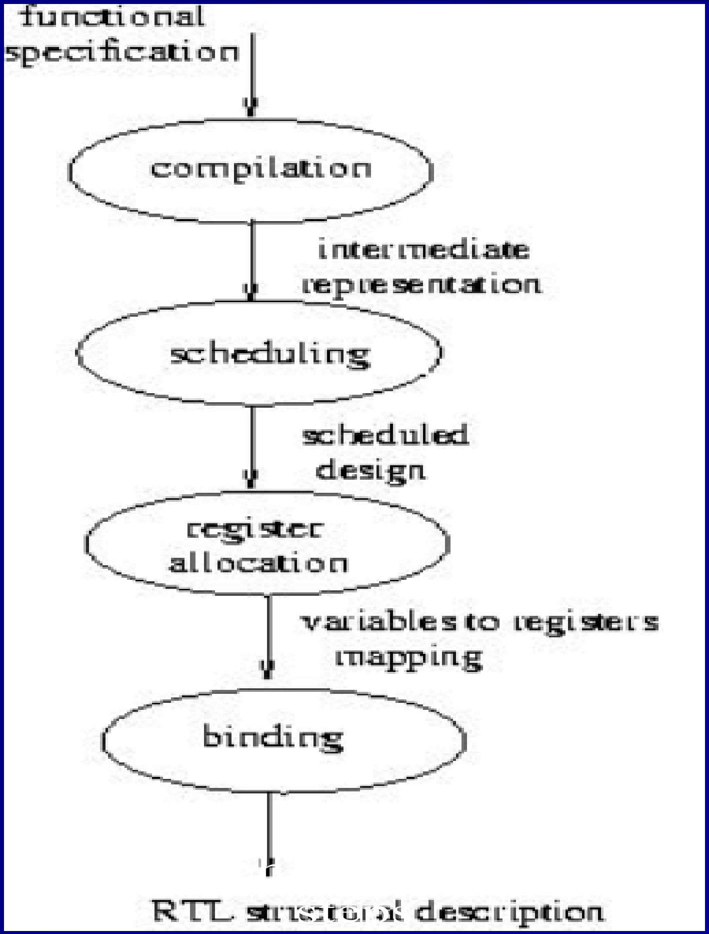


Fig: High-level synthesis flow

Structured Architecture

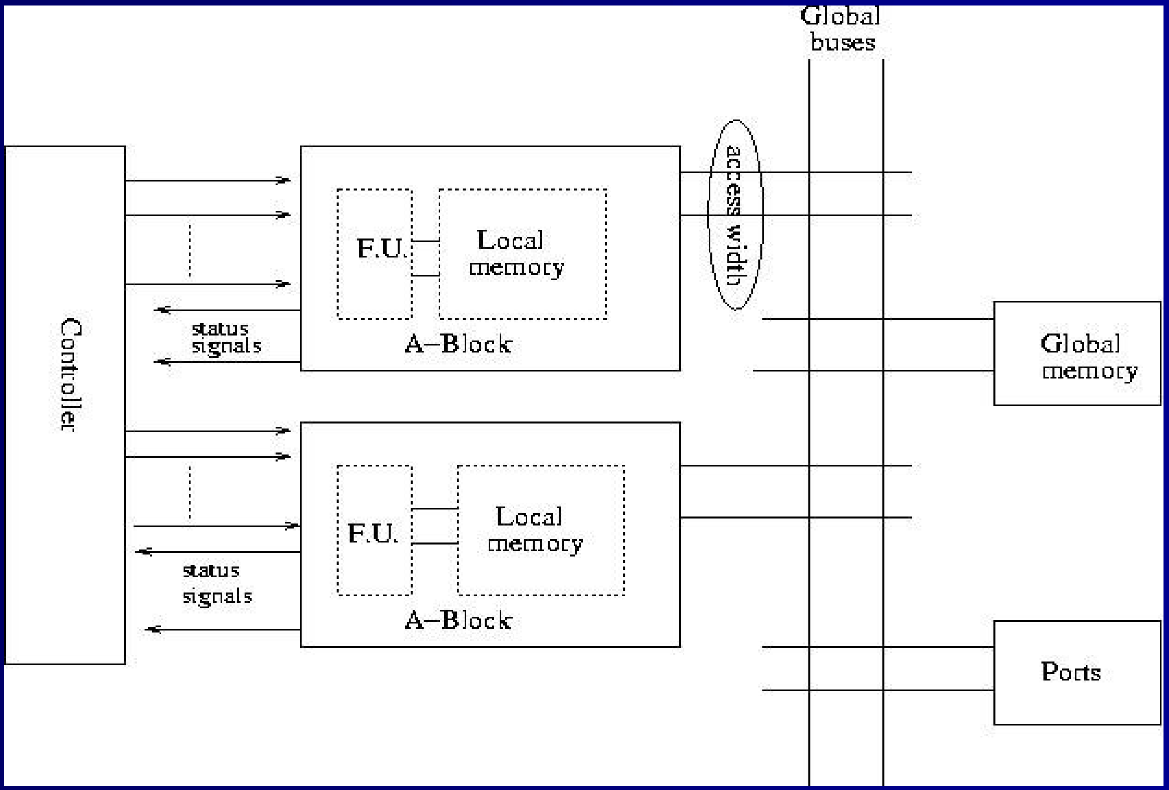


Fig: Structured Architecture

Characterization of SA:

- Number of A-block
- Number of global bus
- Access width
- Maximum number of writes to the storage location in a A-block

Advantages:

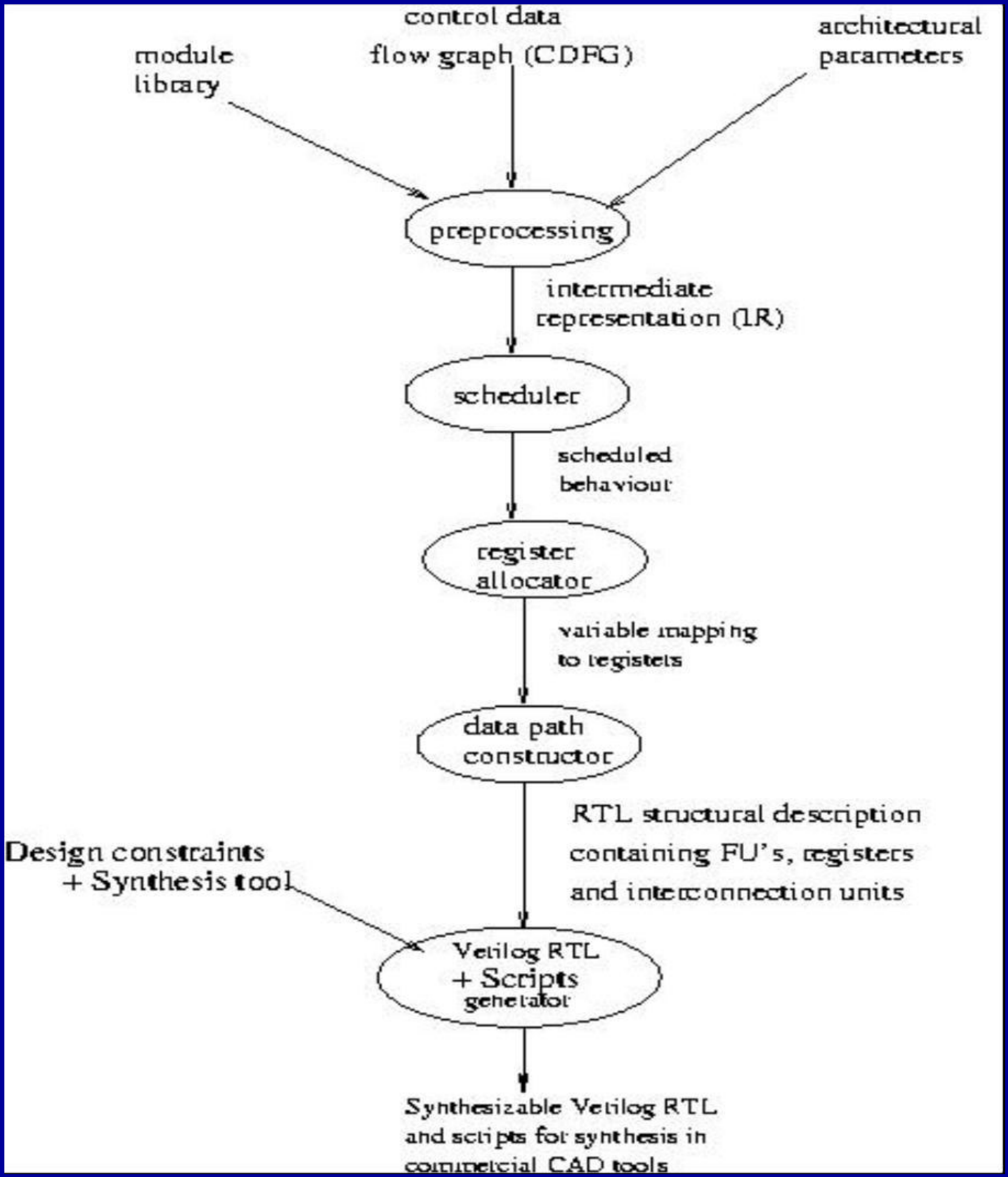
- Avoids random interconnection between data path elements.
- Easy to implement in on programmable device like FPGA.

Contribution of our work

Development of a genetic algorithm based High-level Synthesis tool “SAST”(Structured Architecture Synthesis Tool) that supports the proposed architecture. The tool 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.

SAST synthesis flow

Design flow of SAST


control data flow graph (CDFG)
module library
architectural parameters
preprocessing
intermediate representation (IR)
scheduler
scheduled behaviour
register allocator
variable mapping to registers
data path constructor
RTL structural description containing FU's, registers and interconnection units
Design constraints + Synthesis tool
Verilog RTL + Scripts generator
Synthesizable Verilog RTL and scripts for synthesis in commercial CAD tools

Preprocessing

- Data dependency information between the operations in each A-block. Represented by dependency flow graph.
- Incoming and outgoing variables of each A-block.

Following set of variables used to calculate incoming and outgoing variables of a 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.

Scheduling

Features:

- Genetic Algorithm (GA) based.
- Can handle multi-cycle operations, pipelined operators and multiple implementation of operators.

Output of the scheduler:

- a schedule of operations.
- the A-block in which each operation is scheduled.
- the schedule of all transfers over the global buses.
- satisfying the architectural constraints, the composition of the FU in each A-block, in terms of specific implementations of operators from a module database is also provided.

GA based scheduling algorithm

- A structured solution representation has been used, as against a simple bit string due to the complex nature of the problem. It consists of information about scheduling of operations and variables transfer and composition of FU.
- 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.

Contd.

- Solution generated at random may corresponds to a infeasible solution. Therefore a completion algorithm is required to identifies the feasible 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.

contd.

Cost function :

C = (penalty)(extra time steps) + cost of FUs

Parent selection probability:

$$p_{s_i} = \frac{C_{max} + \delta - C_i}{N_{sols}(C_{max} + \delta) - \sum_i C_i},$$

where C_{max} is the maximum solution cost and N_{sols} is the number of soulutions in current population.

contd.

Cross over :

- Selection of parents solutions.
- Inheritance of attributes from parents including
  - Variable initialization
  - Schedule times
  - Transfer times
  - A-block binding of operations

Generated solution may be infeasible.

Register Allocation & Binding

Sub steps:

- ➢ Lifetime analysis of variables in an A-block from the schedule of operations and the bus transfers of variables over the global buses.
  1. Bus Transfer
  2. Type of operator
- □ Compatibility determination, constructs the compatibility graph from the lifetimes of variables in an A-block
- □ Register optimization step computes minimum number of registers required in the data path of an A-block from the lifetimes of variables and the compatibility graph in an A-block.

Data path generation

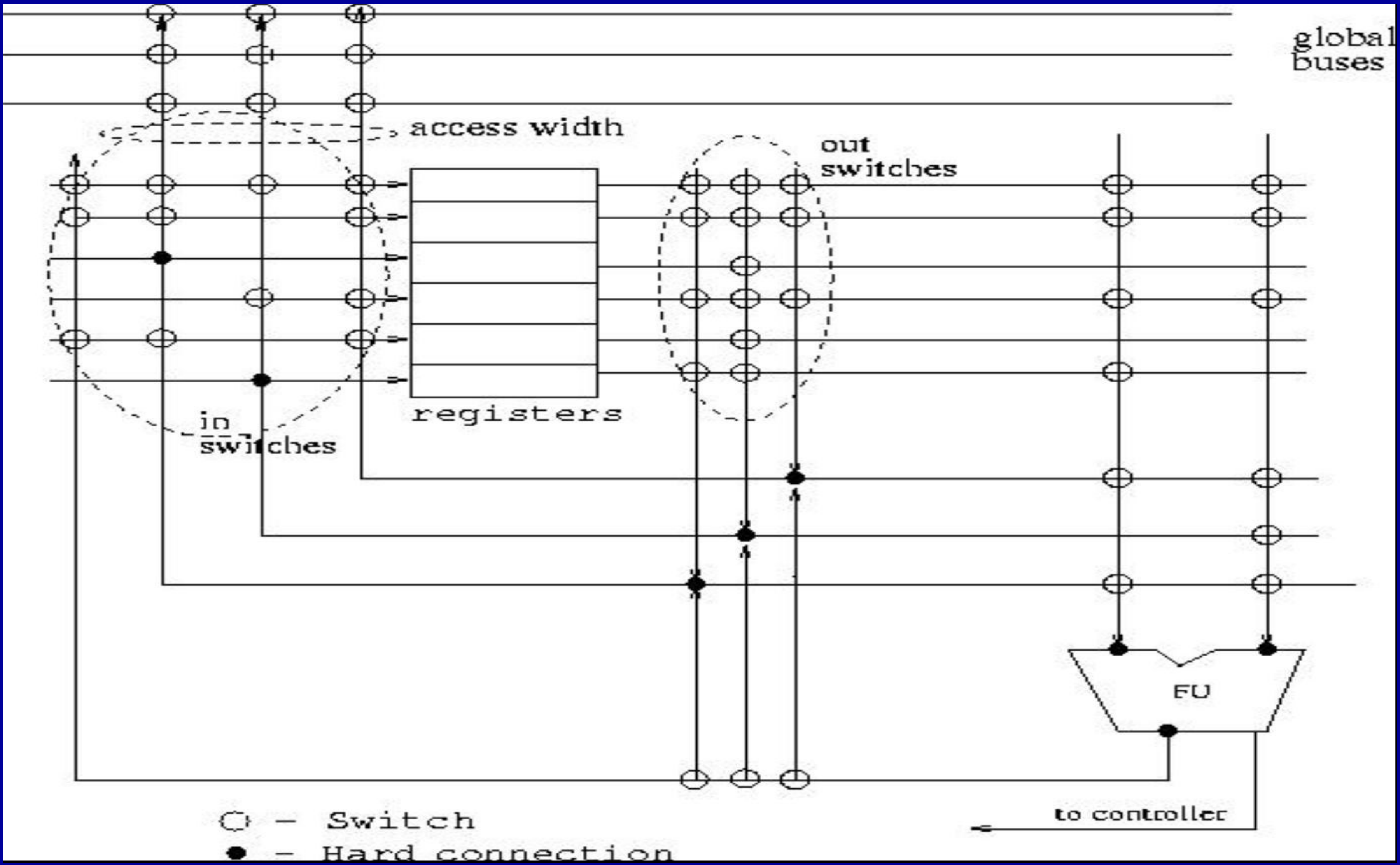


Fig : data path of an A-block

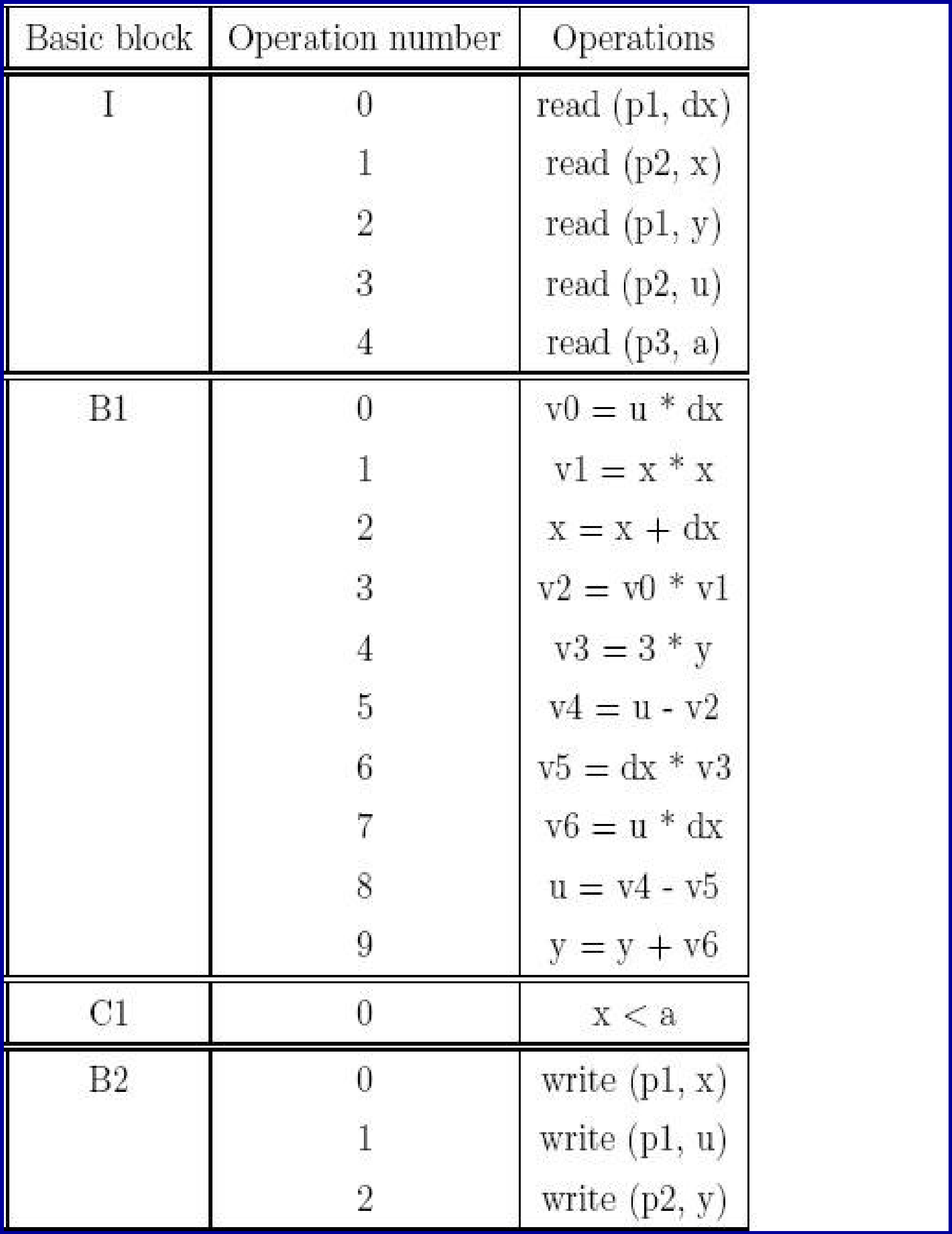
Verilog code generation

- Data path ： Consist of description of each A-block and the connections between them.
- Controller :

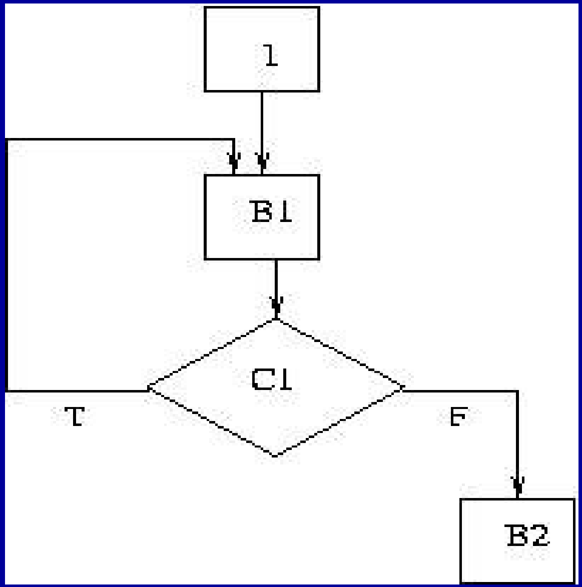
  1.Sequence generator.

  2. Control signal generator

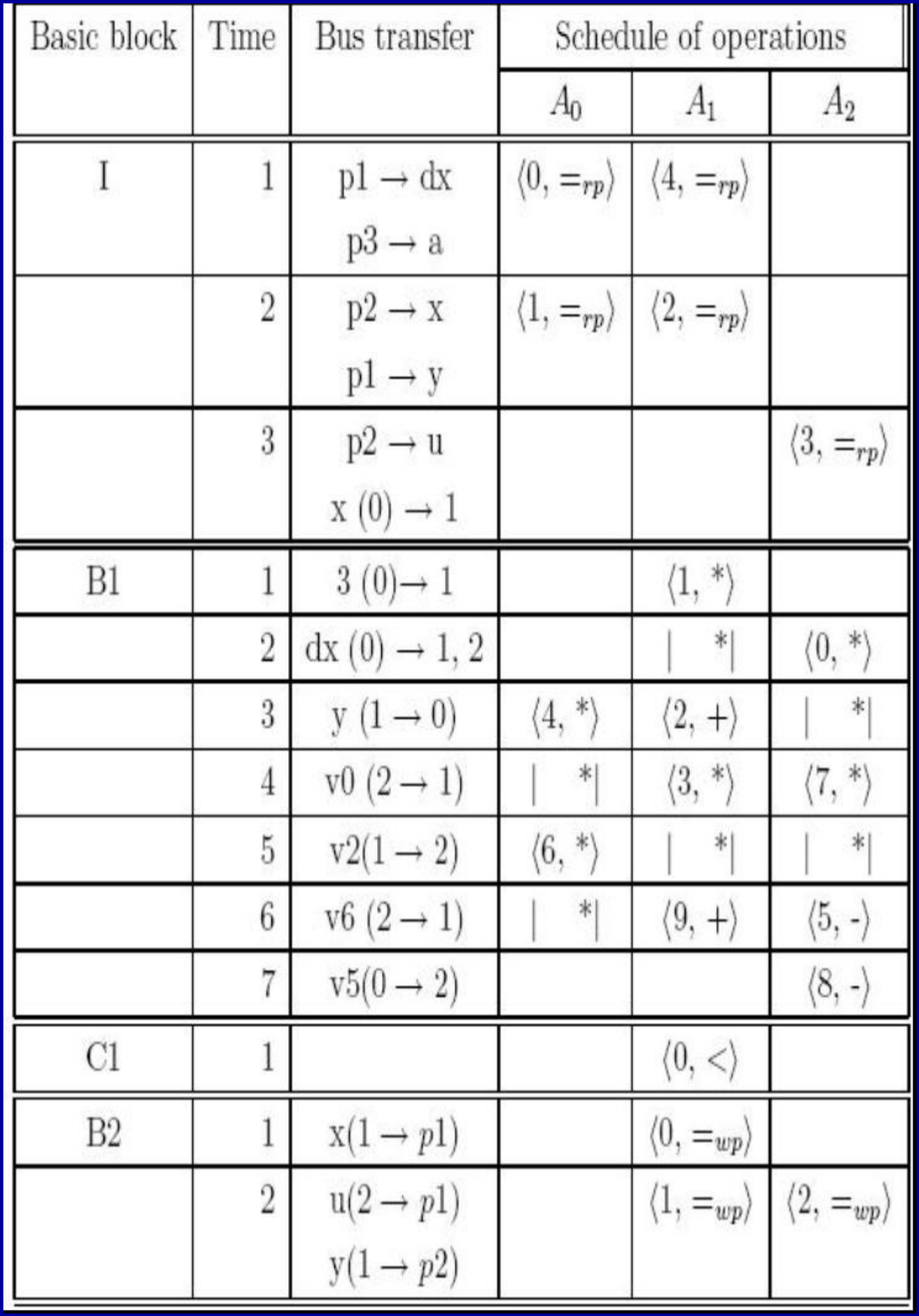
Experimental Results



Basic block	Operation number	Operations
I	0	read (p1, dx)
	1	read (p2, x)
	2	read (p1, y)
	3	read (p2, u)
	4	read (p3, a)
B1	0	v0 = u * dx
	1	v1 = x * x
	2	x = x + dx
	3	v2 = v0 * v1
	4	v3 = 3 * y
	5	v4 = u - v2
	6	v5 = dx * v3
	7	v6 = u * dx
	8	u = v4 - v5
	9	y = y + v6
C1	0	x < a
B2	0	write (p1, x)
	1	write (p1, u)
	2	write (p2, y)

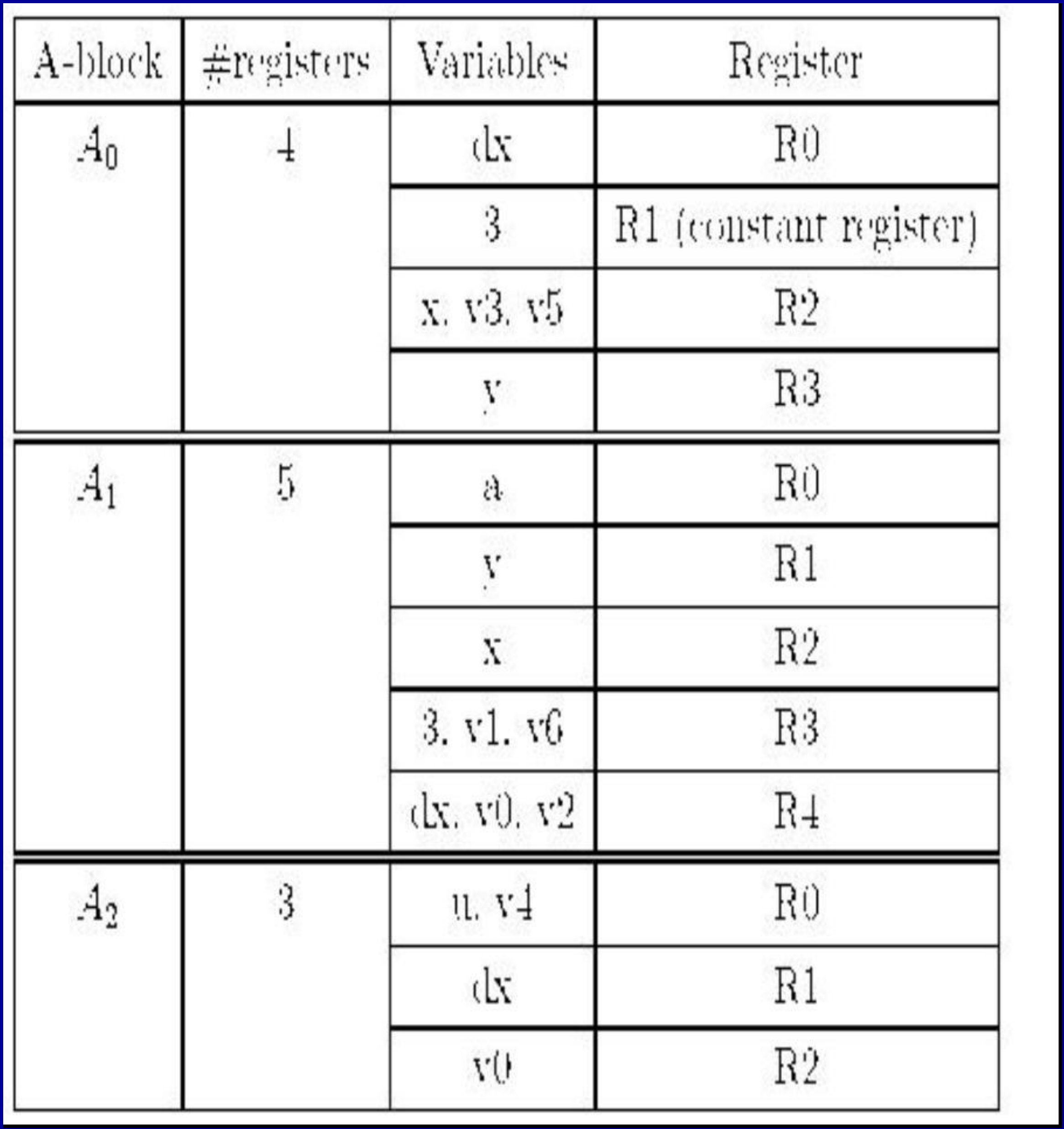


CDFG of ***Differential Equation Solver (DIFFEQ)***



Basic block	Time	Bus transfer	Schedule of operations		
			A_0	A_1	A_2
I	1	$p1 \rightarrow dx$ $p3 \rightarrow a$	$\langle 0, =_{rp}\rangle$	$\langle 4, =_{rp}\rangle$	
	2	$p2 \rightarrow x$ $p1 \rightarrow y$	$\langle 1, =_{rp}\rangle$	$\langle 2, =_{rp}\rangle$	
	3	$p2 \rightarrow u$ $x\ (0) \rightarrow 1$			$\langle 3, =_{rp}\rangle$
B1	1	$3\ (0) \rightarrow 1$		$\langle 1, *\rangle$	
	2	$dx\ (0) \rightarrow 1, 2$		$\vert\ \ *\vert$	$\langle 0, *\rangle$
	3	$y\ (1 \rightarrow 0)$	$\langle 4, *\rangle$	$\langle 2, +\rangle$	$\vert\ \ *\vert$
	4	$v0\ (2 \rightarrow 1)$	$\vert\ \ *\vert$	$\langle 3, *\rangle$	$\langle 7, *\rangle$
	5	$v2(1 \rightarrow 2)$	$\langle 6, *\rangle$	$\vert\ \ *\vert$	$\vert\ \ *\vert$
	6	$v6\ (2 \rightarrow 1)$	$\vert\ \ *\vert$	$\langle 9, +\rangle$	$\langle 5, -\rangle$
	7	$v5(0 \rightarrow 2)$			$\langle 8, -\rangle$
C1	1			$\langle 0, <\rangle$	
B2	1	$x(1 \rightarrow p1)$		$\langle 0, =_{wp}\rangle$	
	2	$u(2 \rightarrow p1)$ $y(1 \rightarrow p2)$		$\langle 1, =_{wp}\rangle$	$\langle 2, =_{wp}\rangle$

Table : Schedule for differential equation solver



A-block	#registers	Variables	Register
A_0	4	dx	R0
		3	R1 (constant register)
		x, v3, v5	R2
		y	R3
A_1	5	a	R0
		y	R1
		x	R2
		3, v1, v6	R3
		dx, v0, v2	R4
A_2	3	u, v4	R0
		dx	R1
		v0	R2

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

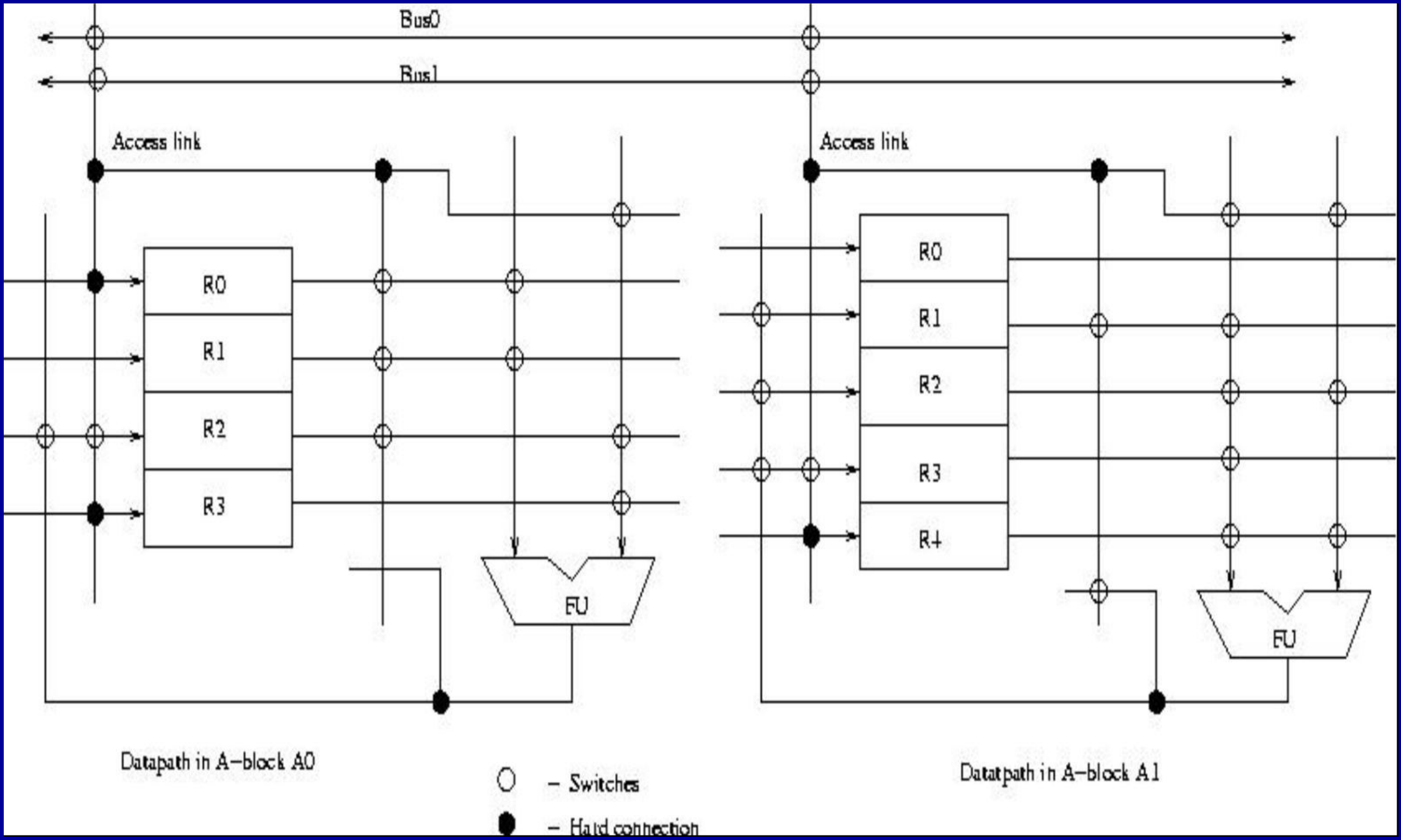


Figure : Data path in the A-block A0 and A1.

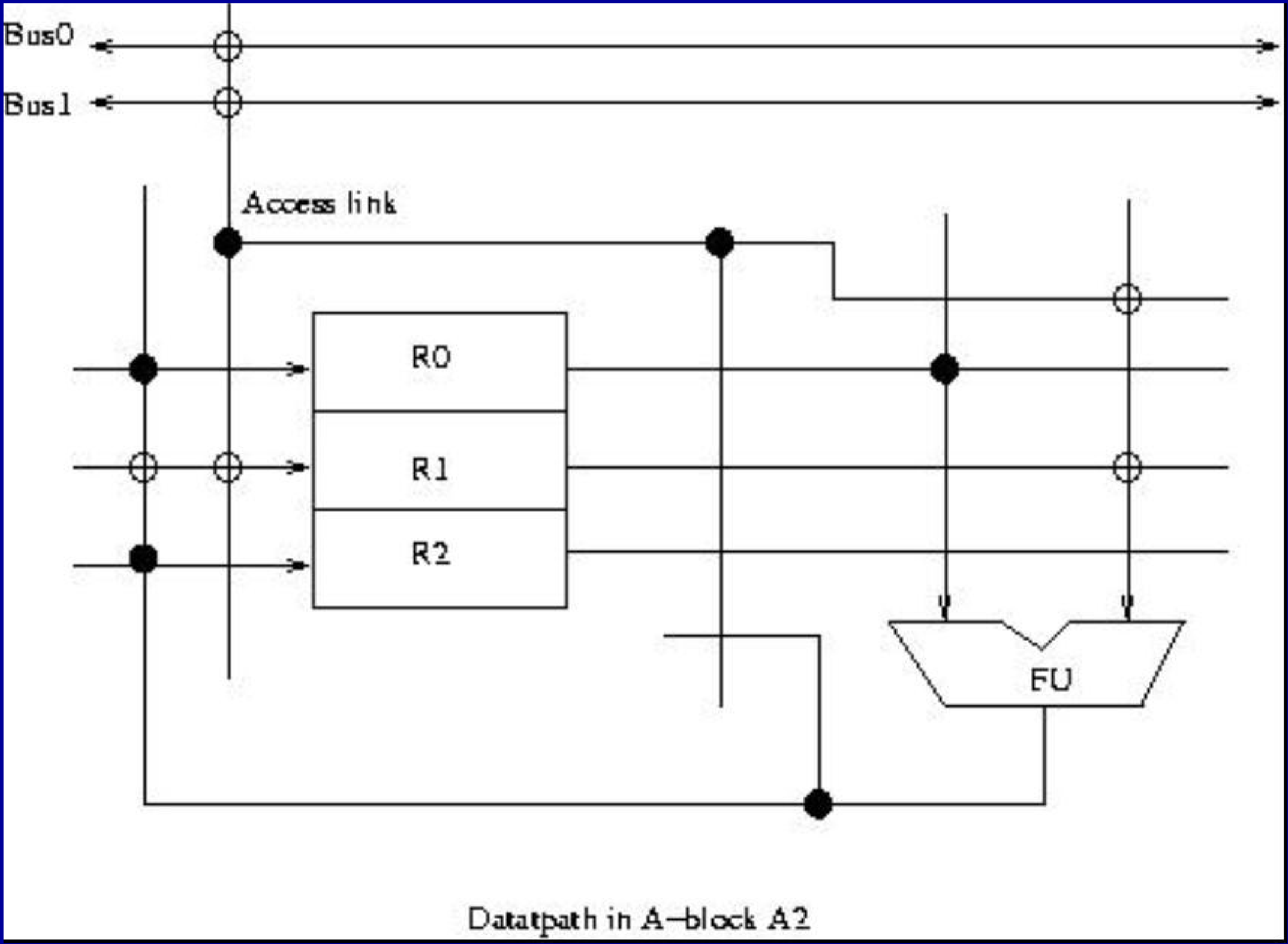
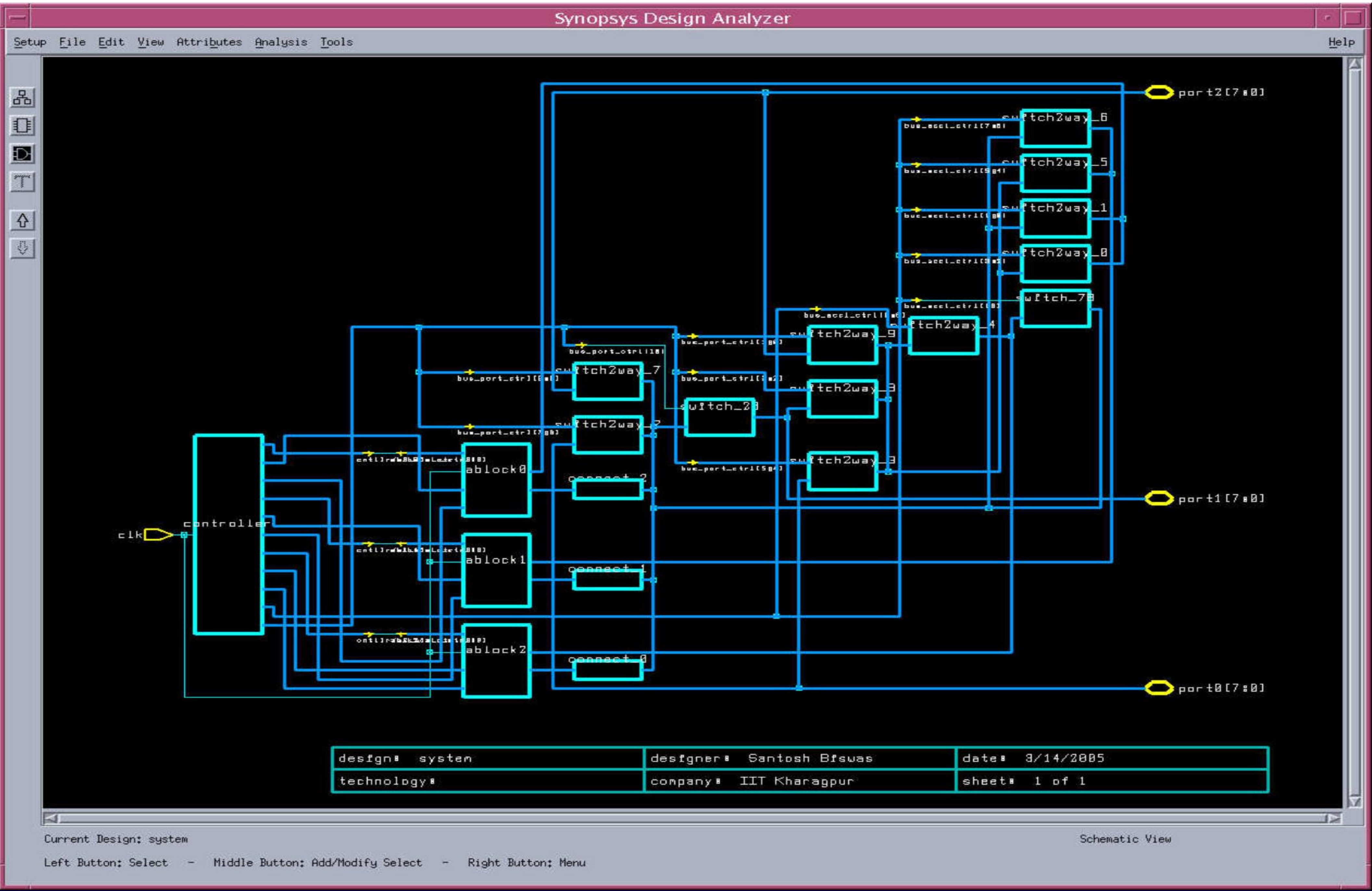


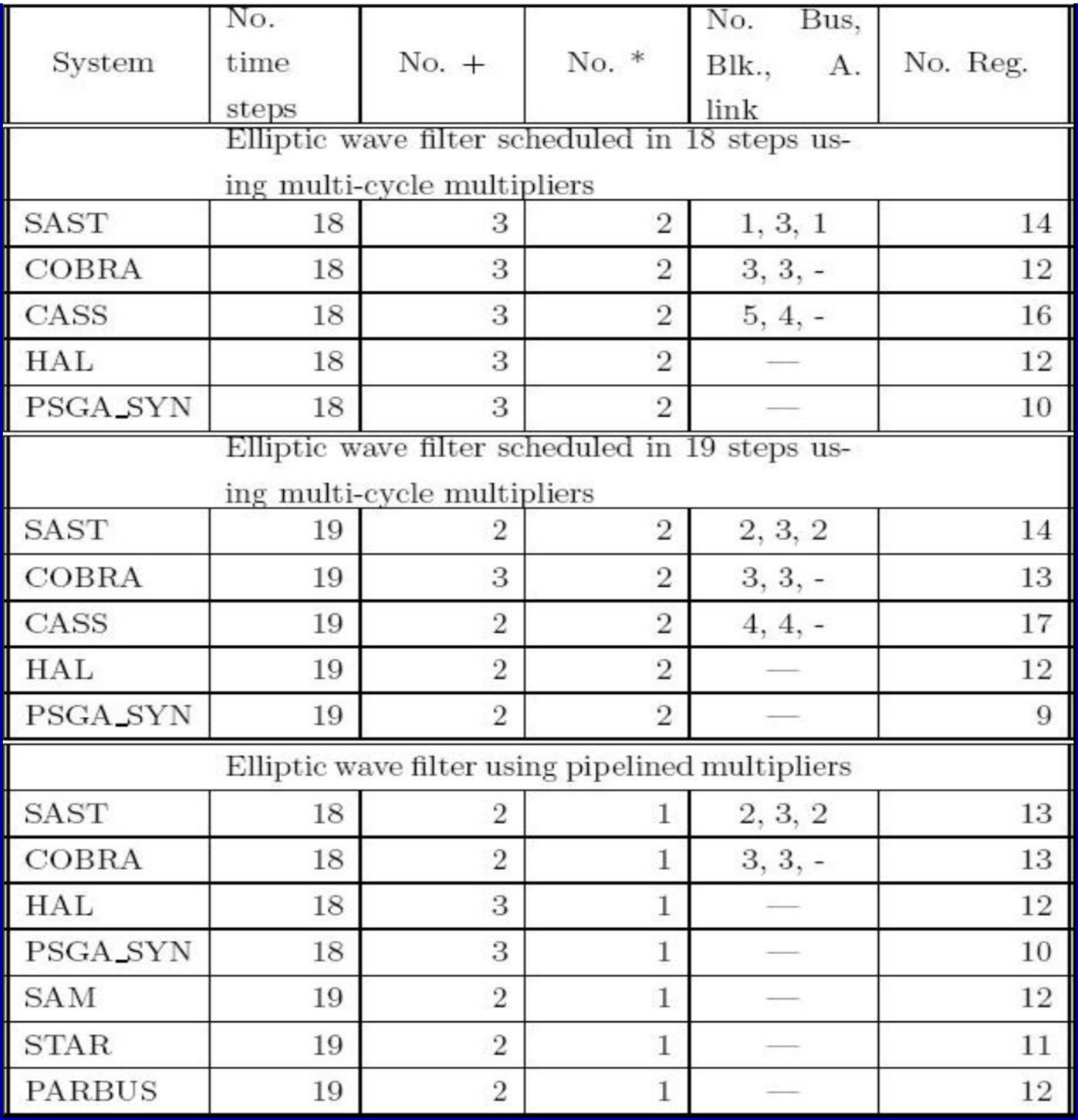
Figure : Data path in the A-block A2

RTL generated by SAST in verilog has been synthesized using Synopsys DA with 0.18 Micron CMOS9 library of National Semiconductor Corp., USA.



design: system	designer: Santosh Biswas	date: 3/14/2005
technology:	company: IIT Kharagpur	sheet: 1 of 1

Figure: Synopsys DA output for EWF.



System	No. time steps	No. +	No. *	No. Bus, Blk., A. link	No. 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 scheduled in 19 steps using multi-cycle multipliers				
SAST	19	2	2	2, 3, 2	14
COBRA	19	3	2	3, 3, -	13
CASS	19	2	2	4, 4, -	17
HAL	19	2	2	—	12
PSGA_SYN	19	2	2	—	9
	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 : Comparison of results for EWF with a few other synthesis tools

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 Trans-actions 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 tech-niques 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

Thank You