Layout-Driven High Level Synthesis for FPGA Based Architectures

Min Xu[†]
Fadi J. Kurdahi [‡]
[†] Department of Information and Computer Science
[‡]Department of Electrical & Computer Engineering
University of California, Irvine, CA 92697

Abstract

In this paper, we address the problem of layout-driven scheduling-binding as these steps have a direct relevance on the final performance of the design. The importance of effective and efficient accounting of layout effects is well-established in High-Level Synthesis (HLS), since it allows more efficient exploration of the design space and the generation of solutions with predictable metrics. This feature is highly desirable in order to avoid unnecessary iterations through the design process.By producing not only an RTL netlist but also an approximate physical topology of implementation at the chip level, we ensure that the solution will perform at the predicted metric once implemented, thus avoiding unnecessary delays in the design process.

1 Introduction

With recent advances in semiconductor technology, increases in design complexity and drastic reduction in time-to-market, the need for High Level Synthesis (HLS) on higher abstraction levels where functionality and design tradeoff are easier to understand is unavoidable. However, the gap between HLS and physical design severely prevented the acceptance of HLS for practical applications. This is because most of existing CAD systems treat HLS tasks and physical design independently. They suffers from three major drawbacks: (1) it is not known whether the design will meet the constraints or not until both HLS and physical design have been finished; (2) it is difficult for the early design tasks to make the right decision without knowing the information related to layout; (3) when the constraints are not met, it is difficult to identify where the problem comes from and at which level the design should be modified; (4) The time-consuming phase of placement and routing has to be run within each iteration in the design process, thus lengthening that process considerably.

2 Previous Work

3-D [7] presented an approach to the problem of scheduling-binding while simultaneously considering floorplanning. As operations are scheduled and functional modules are allocated, 3-D decides their shape and position on the floorplan concurrently. However, this approach didn't consider the cost and delay of registers, multiplexors, and wiring. There are several efforts that addressed binding with layout information. GBA [8] considered binding with physical information. However, GBA applies only to one dimension bit-slice design. PBITNET [9] and LDB [3] incorporate the interconnection delay when binding is performed. Ewering [10] and ApplaUSE [11] addressed the binding with physical information problem by moving placement earlier before bus and register assignment, but no physical information is taken into account when FU binding is performed. SMB [12] presented an integrated approach for minimizing critical path delay by simultaneously performing FU binding and floorplanning.

On the other hand, our approach estimate the layout information before actually perform scheduling-binding tasks and use the layout information to guide our scheduling-binding tasks.

3 Architectural Model and Problem Definition

In high-level synthesis, an RTL system that consists of FUs, storage units, and interconnections is synthesized from the behavioral description. In order to explore the impact of physical design information in HLS, we need to define a target architecture. We confine our scope to multiplexor-based architectural model in this paper and multi-cycled operations are possible.

Our problem can be defined as follows:

> Given (1) a data flow graph (DFG), (2) maximum allowable clock period and execution time, which are usually part of the system specification, (3) component shape functions, identify whether there is a feasible RTL datapath solution or not. If there is a solution, perform scheduling and binding and generate an RTL netlist and its corresponding floorplan; Otherwise, report it to the user and output the best solution that could be achieved.

The example in Figure 1 illustrates the problem. We assume that the controller is implemented as a Moore FSM with status and control registers and work here concentrates on the datapath area and delay metrics.

4 Our Approach

The flow of our algorithm is shown in Figure 2. The main features of this work are the following: (1) the final result is evaluated without actually going through the time consuming phase of placement and routing, and the fast

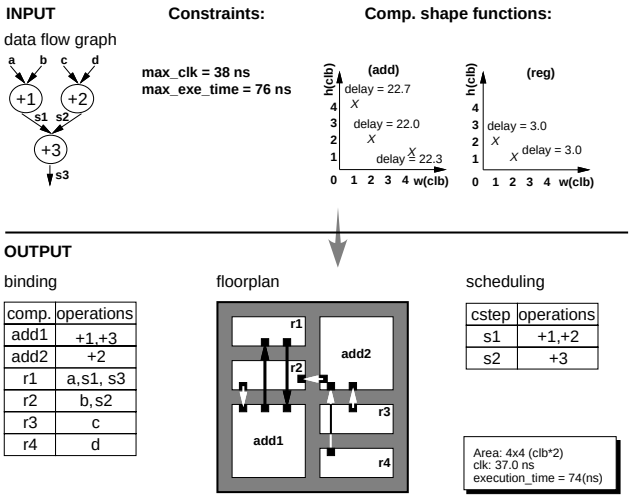

Figure 1: An example illustrating the inputs and outputs of the problem

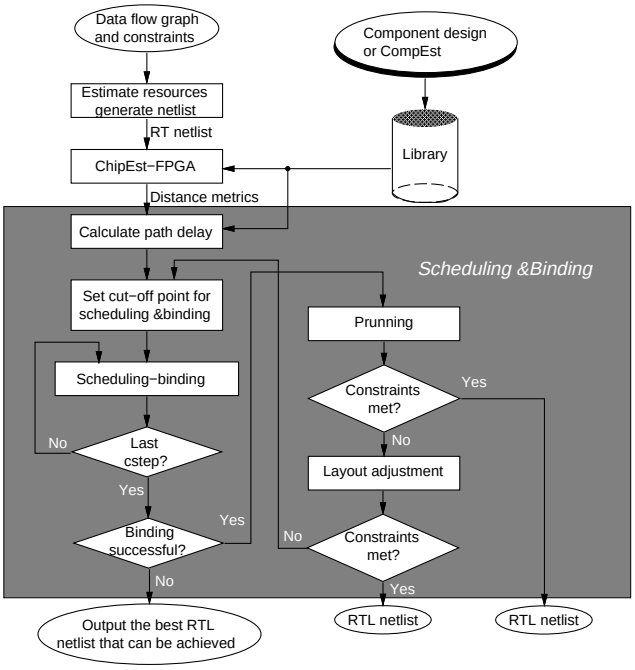

Figure 2: Layout-driven scheduling-binding technique for HLS

layout information feedback is used to guide scheduling-binding tasks; (2) when time constrains are met, the algorithm will output not only a scheduling solution and a structural RTL netlist, but also it's corresponding physical topology which can be carried through silicon implementation in a predictable manner; (3) whenever time constraints are not met, our scheduling and binding techniques provide a means of exploring the design space in a realistic and efficient way, with this exploration, our scheduling and binding techniques will provide feedback to the previous tasks if the constraints can not result in any feasible solution and output the best implementation that can be achieved;

Given a flow graph and time constraints, first, we calculate the number of control steps using maximum execution time divided by the maximum clock period. Then we estimate the RTL minimum clock period using the way proposed in MinClkGen [5]. The RTL clock period and the number of control steps are then used to estimate lower bound of the resources [5]. The underlying concept of the resource estimation is that, if there are n operations that need to be finished within s states, and a component used to perform an operation requires at least c clock cycles to finish the execution before it could be used again to execute another operation, then clearly, the minimum number of components required is equal to $\lceil (n \times c)/s \rceil$. Next, we construct a fully connected netlist in which each FU is connected to every register and each register is connected to every FU and feed it into our our physical level estimation tools ChipEst-FPGA [1] and CompEst-FPGA [2] to obtain an approximate topology of the layout. CompEst-FPGA is a component estimation tool which predicts the area and delay of a given RTL component netlist. Once we have obtained a shape function for each component, ChipEst-FPGA is used to generate an approximate topology of the overall design. At this moment, we can get distance metrics between the different units and this step provides valuable feedback to the scheduling and binding task in HLS as described later.

4.1 Set Cut-Off Point for Scheduling-Binding

Having estimated all the path delays, we can set the cut-off point to decide whether a path can be used for binding or not. *Cut-off point* is a number such that all paths whose delays fall above that number will not be considered as candidates for binding. Let's denote the initial cut-off point for binding as CT_{init} and the cut-off point for the current iteration as $CT_{current}$. Let cr_delay_{prev} be the critical path delay of the previous binding solution and α be the factor of choosing the current cut-off point. The user can decide whether α should equal 1, 10, 100, 1000... so that the trade-off between the time spent on exploration and the number of solutions explored can be made.

The initial cut-off point and current cut-off point can be obtained by the following equation:

$$CT_{init} = \lceil MAX(Delay_{i,j,k} | i,k = 0,1,...r; j = 0,1,...f) \rceil \quad (1)$$

$$CT_{current} = \lfloor (cr\_delay_{prev} * \alpha)/10 \rfloor * 10/\alpha \quad (2)$$

where $Delay_{i,j,k}$ is the delay from the ith register, through the jth FU to kth register, r is the number of registers, and f is the number of FUs.

4.2 Scheduling-Binding

Our approach then sequentially performs scheduling and binding simultaneously, one control step at a time. Mainly, it has the following basic steps:

- **Obtain the number of available functional units and registers:** Once a cut-off point is set, paths with

delay longer than the cut-off point are dropped from consideration for scheduling-binding. The functional units and registers appeared on the remaining paths are available for scheduling-binding in current control step.

- **Resource constrained execution interval analysis:** Recall that our problem is both time and resource constrained. Once the number of control steps is calculated and the current available resources are estimated, we use execution interval (EI) analysis under resource constraints proposed by Adwin et. al [6] to detect the feasibility of partial scheduled data flow graph. The analysis prunes the search space of scheduling-binding without limiting the solution space and therefore speeds up the scheduling-binding algorithm.
- **Generating ScheduleGroups:** A depth-first, recursive procedure is used to generate different ScheduleGroups under a given set of operations that are already bound in the previous control steps, a set of operations that will be scheduled and bound within current control step, and a set of operations that are ready to be scheduled and bound.
- **ScheduleGroup selection:** Different ScheduleGroups may corresponds to a different scheduling-binding solution. During the search procedure, instead of finding the best solution in each iteration, the algorithm will stop further searching once a proper ScheduleGroup is found. The identification of proper ScheduleGroup is described in Section 4.2.2.

4.2.1 Generating ScheduleGroups

A depth-first, recursive procedure is used to generate the set of ScheduleGroups. The ScheduleGroups are constrained by the data/control constraints between operators and the available scheduling-binding resources based on the layout information.

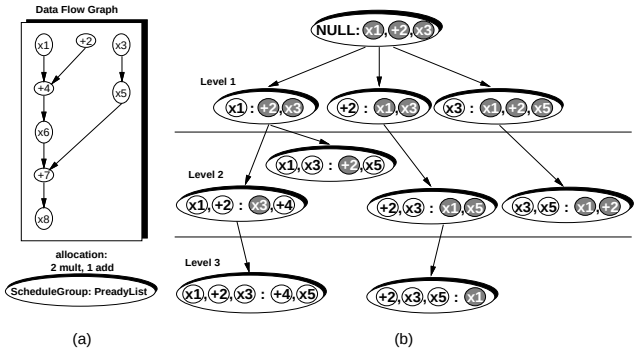

Figure 3: A tree shows all possible ScheduleGroups

Figure 3(b) shows all possible ScheduleGroups for the first control step for the given DFG which is shown in Figure 3(a). The current available resources include two multipliers and one adder. Each node in the tree contains two entities, schedule group ($ScheduleGroup$) and priority ready list ($PreadyList$). The priority ready list, $PreadyList$, initially contains all operations that: (1) have their dependencies scheduled & bound in the previous control steps or with primary inputs; and (2) meet the constraints: $New\_OEI\_start(O_i) \leq current\_cstep \leq New\_OEI\_end(O_i)$.

The $ScheduleGroup$ is initialized to null. The tree is dynamically constructed by depth-first traversal using a recursive call. Whenever a *proper* ScheduleGroup is found, the tree construction, i.e. finding new ScheduleGroups will stop. Since the maximum number of operations that can be scheduled and bound in the current control step should be less than or equal to the number of available functional units, the maximum depth of the tree equals the number of the current available components.

4.2.2 Schedule_Group Selection

To decide whether a schedule-group is proper or not, the algorithm does *bindable check* and *schedulable check*. *Bindable check* checks whether operations in the schedule-group can be bound in the current control step given the estimated resources and the paths. *Schedulable check* checks whether the rest of the unscheduled operations can be scheduled successfully within the rest of the cycles assume that operations in the schedule-group are scheduled in the current cycle. If the current schedule-group is not bindable, all it's child schedule-groups are not bindable. Otherwise, the algorithm will proceed to perform schedulable check. If the schedule-group is both bindable and schedulable, it may be a proper solution, the algorithm will continue its search until no child feasible solution can be found. The youngest child feasible solution is chosen as the proper schedule-group (we explore other solutions later by further lowering the *cut-off point*). If the schedule-group is bindable but not schedulable, it's child or sibling schedule-group will be checked.

Once a proper schedule-group is found, the algorithm will record its scheduling-binding information and proceed to do the next cycle scheduling-binding. This process will be repeated until all the cycles are processed. If the scheduling-binding succeeds, the algorithm will prune all the unnecessary interconnections, delete all the unnecessary multiplexors and finally, re-size the multiplexors according to the actual interconnection information. The algorithm will then update the area and timing information based on the component information in the library or by invoking CompEst-FPGA [2] and generate an optimized RTL netlist. At this point, if the clock period exceeds the maximum clock period, layout adjustment will be invoked to re-run our ChipEst-FPGA on the pruned RTL netlist

After layout adjustment, if the cycle time still can not satisfy the maximum clock period constraint, we need to reset the cut-off point and redo the scheduling and binding. This iteration will continue until the cut-off point hits the cut-off-point threshold or a feasible solution is found.

5 Layout-Driven Scheduling-Binding For Conditional Branches

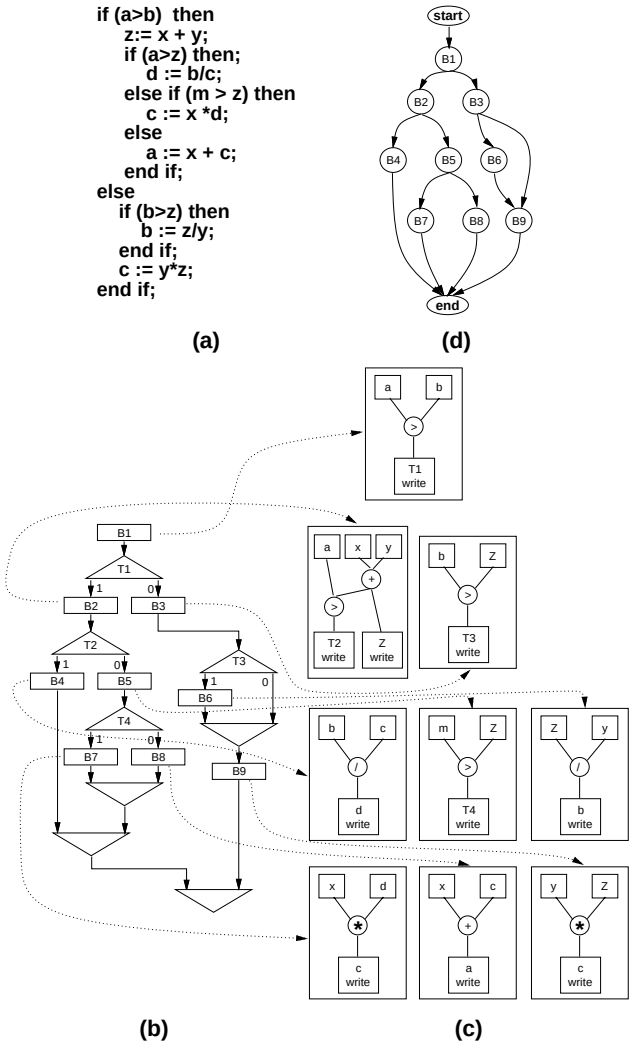

Figure 4: An example: (a) VHDL description (b) Control Flow Graph (CFG) (c) Data Flow Graph (DFG) (d) Dependency Graph for DFGs (DGD)

To handle conditional branch, we use control/data flow graph (CDFG) proposed by Gajski [17] as our representation. Figure 4 shows an example of a VHDL description 4(a), its corresponding control flow graph 4(b) and data flow graph 4(c). The control flow graph captures sequencing, conditional branching and looping constructs in the behavioral description, while data flow graph captures operational activity described by the VHDL assignments statements. Each node of the control flow graph can have a data flow block associated with it that represents the operations performed in that control flow node. The CDFG data flow blocks are similar to basic blocks in structured programming languages. The condition is represented in the control flow graph by fork and joint node. The left path following the fork node is the path the design will be taken when the condition is true while the right path is the path the design will take when the condition is false.

The layout driven scheduling-binding problem now becomes the scheduling and binding for multiple data flow graphs (DFGs). The relationship between these DFGs is represented in the control flow graph.

To perform scheduling-binding, we first build the Dependency Graph for DFGs (DGD), A DGD graph consists of one start node, one end node, one or more DGD nodes, and edges. It can be derived from the control flow graph. As show in Figure 4(d), Each basic block in the control flow graph will generate a DGD node. Two DGD nodes are connected by an edge if there is a path in the control flow graph, which goes through no other basic blocks. Each DGD node has one pointer to its corresponding DFG. Also, it contains one DGD node flag which will not be set unless all the operations in its DFG have been scheduled and bound. An DGD example is shown in Figure 4(d). DGD nodes B_i are derived from the corresponding basic blocks B_i in the control flow graph. Each DGD node has its own DFG.

Once we have the DGD, we build a corresponding DFG for each DGD node. After that, we sequentially schedule and bind the DGD and DFGs simultaneously, one control step at a time. Within each control step, we first find out all the ready DGD nodes. Then for each of these DGD nodes, the layout driven scheduling-binding procedure is invoked. Every pair of DGD nodes either have dependency or belong to different branches. That means operations belonging to different DGDs are mutually exclusive. Such operations (belonging to different DGDs) can share the same resources.

6 Experimental Results

We have implemented our layout-driven RTL binding techniques for HLS in C on the Sun SPARC workstation. The first set of experiments included behavioral description of basic blocks are: (1) *the 2nd order differential equation solver.* (2) *Discrete Cosine Transformer (DCT).* (3) *the 5th order elliptic wave filter (EWF).* (4) *18 tap FIR filter (FIF)* [13] used in a 155 Mbps ATM transceiver. The bit-width of all the examples is 4.

Examples	Execu. time constraints (ns)	Number of control steps	Clock period constraints (ns)	Our approach		Traditional approach	
				Area (clbxclb)	Clock period (ns)	Area (clbxclb)	Clock period (ns)
HAL	250	8	31.3	12x12	29.9	10x10	42.5
		10	25	10x10	21.4	10x10	42.2
		12	20.7	10x10	20.7	10x10	41.5
DCT	400	12	33.2	24x24	32.8	18x18	63.5
		13	30.7	22x22	27.6	18x18	57.5
		14	28.5	22x22	26.5	18x18	57.0
EWF	1000	28	35.7	16x16	35.0	12x12	53.9
		29	34.4	16x16	32.7	10x10	51.2
		30	33.3	14x14	30.5	10x10	48.4
		31	32.2	16x16	27.3	10x10	48.4
FIF	400	12	33.2	20x20	31.5	16x16	60.8
		13	30.7	20x20	30.7	16x16	55.3
		14	28.5	20x20	27.6	16x16	54.6

Figure 5: Experimental results for basic blocks

Figure 5 shows the experimental results. The execution time constraints specify the maximum allowable execution

time. For different number of control steps, the clock period constraints can be calculated as the execution time divided by the number of control steps. Both area and clock period using our approach and traditional approach are listed in the Figure 5. For the traditional approach, we use lower bound based scheduling algorithm [4] which produces schedule (which is optimal in resources count, but does not consider layout effects), then we invoke our layout-driven binding algorithm [3]. We make the following observations: (1) in all cases, using our approach, we can find a result that satisfies the constraints while the time constraints are violated using traditional approach; (2)traditional approach generates a smaller RTL design which does not satisfy timing constraints; (3) the clock period constraints violation is significant, up to 100%. Thus, we can see, taking layout information into account while performing high level synthesis tasks is necessary since the violation can not be solved by fine tuning the HLS algorithms themselves. Our approach is efficient since each solution takes less than 2 minutes of CPU time.

The second set of experiments are used to test our scheduling-binding algorithm for descriptions with conditional branches. They are: (1) Kim's example [14]. (2) Wakabayashi's example [15]. (3) One process taken from MPEG, Cordic [16]. The examples are 8 bits datapaths.

Examples	RT Level constraints (ns)	Number of control steps	Resources	Area (CLB x CLB)	CLock period (ns)	Run time (sec)
Kim	45.6	11	2A, 1M	12x12	70.2	50
	70.4	10	2A, 1M	12x12	87.6	62
	88.2	8	3A, 1M	14x14	103.2	81
Cordic	45.6	5	1A, 2M	18x18	70.7	71
	70.4	4	1A, 2M	18x18	94.5	73
	88.2	4	2A, 2M	20x20	110.1	89
Waka	45.6	9	2A	10x10	66.6	28
	70.4	7	3A	12x12	85.9	33

Figure 6: Experimental result conditional branches

Figure 6 shows the experimental results. RT Level constraints only include FU and register delays (the delay of adder-subtractor is $42.6ns$ and delay of multiplier is $67.4ns$). The examples shown contain multi-cycle, unicycle and chaining depending on different RT Level constraints. **Resources** shows the lower bound on functional units needed for the examples. The results are listed in the last three columns. Based on these results, we can make the following observations: (1) The interconnection delay contributes up to 55% on the performance. This is slightly bigger compared to scheduling-binding for straight line code (up to 50%) because in the conditional branch case, the mux delay is bigger because of resource sharing. (2) The CPU run time for chaining is longer because there are more functional units that can perform the chained operations. (3) The smaller the clock period, the shorter the execution time, this is true for these examples and for these RT Level constraints. Partly this is because we use CDFG as our data representation and different DFGs are forced to be separated and scheduled into different control steps when a fork or a joint nodes occurred. This prevented the algorithm fully exploring the parallelism of the design.

7 Conclusion

We presented a scheduling-binding approach which simultaneously execute scheduling and binding with layout information, and also uses an accurate layout estimator to simultaneously produce an RTL solution and a corresponding floorplan. Future work will incorporate the controller effects into scheduling-binding using the approach proposed in [3].

8 References

[1] M. Xu and F.J. Kurdahi "ChipEst-FPGA: A Tool for Chip Level Area and Timing Estimation of Lookup Table Based FPGAs for High Level Applications," *Proc. of ASPDAC*, Jan. 1997.

[2] M. Xu and F.J. Kurdahi "Area and Timing Estimation for Lookup Table Based FPGAs," *Proc. of ED&TC*, March 1996.

[3] M. Xu and F.J. Kurdahi "RTL Synthesis with Physical and Controller Information," *Proc. of ED&TC*, March. 1997.

[4] S.Y. Ohm , C.S. Jhon and F. J. Kurdahi, "An Optimal Scheduling Approach Using Lower Bound in High-Level Synthesis," *IEICE Trans. on Information and Systems*, March 1995.

[5] H.P. Juan, D.D. Gajski and S. Bakshi, "Clock Optimization for High-Performance Pipelined Design," *Proc. EDAC* 1996

[6] Adwin H. Timmer and Jochen A. G. Jess, "Execution Interval Analysis under Resource Constraints," *Proc. DAC* 1993

[7] Jen-Pin Weng and Alice C. Parker, "3D Scheduling: High-Level Synthesis with Floorplanning," *Proc. DAC*, pp. 668-673, 1991.

[8] Hyuk-Jae Jang and Barry M. Pangrle, "A Grid-Based Approach for Connectivity Binding with Geometric Costs," *Proc. ICCAD*, pp. 94-99, 1993.

[9] Ashutosh Mujumdar, Rajiv Jain, and Kewal Saluja, "Incorporating Performance and Testability Constraints During Binding in High-Level Synthesis," *Transactions on Computer-Aided Design*, pp. 1212-1226, Oct. 1996.

[10] C.Ewering, "Automatic High-Level Synthesis of Partitioned Busses," *Proc. EURODAC*, pp. 304-307, 1990.

[11] Elof Frank, Thomas Lengauer "APPlaUSE: Area and Performance Optimization in a Unified Placement and Synthesis Environment," *Proc. ICCAD*, pp. 662-667, 1995.

[12] Y.M. Fang and D.F. Wong, "Simultaneous Functional-Unit Binding and Floorplanning," *Proc. ICCAD*, pp. 317-312, 1994.

[13] F. Etemadi et al, "155 Mbps ATM Transceiver," *Technical Report*, ECE Dept. UCI, March, 1997

[14] T. Kim, J.W.S. Liu and C.L. Liu, "A Scheduling Algorithm For Conditional Resource Sharing," *Proc. ICCAD*, 1991.

[15] K. Wakabayashi and T. Yoshimura, "Global Scheduling Independent of Control Dependencies Based on Condition Vectors," *Proc. DAC*, 1992.

[16] D.L. Gall, "MPEG: A Video Compression Standard for Multi-media Applications," *Communications of ACM*, Vol. 34, 1994.

[17] D.D. Gajski, N. Dutt, A. Wu and S. Lin, "High-Level Synthesis: Introduction to Chip and System Design," *Kluwer Academic Publishers*, 1992.