A Timing-Driven Module-Based Chip Design Flow

Fan Mo
University of California, Berkeley
1-510-642-4641
fanmo@ic.eecs.berkeley.edu

Robert K. Brayton
University of California, Berkeley
1-510-643-9801
brayton@eecs.berkeley.edu

ABSTRACT
A Module-Based design flow for digital ICs with hard and soft modules is presented. Versions of the soft modules are implemented with different area/delay characteristics. The versions represent flexibility that can be used in the physical design to meet timing requirements. The flow aims at minimizing the clock cycle of the chip while providing quicker turn-around time. Unreliable wiring estimation is eliminated and costly iterations are reduced resulting in substantial reductions in run time as well as a significant decrease in the clock periods.

Categories and Subject Descriptors
J.6 [**Computer-Aided Engineering**]

General Terms
Algorithms, Performance, Design

Keywords
Design flow, Physical synthesis, Timing constraints

1. INTRODUCTION
IC design has begun to benefit from the Deep Sub-Micron (DSM) technology, which offers higher density of transistor integration and higher performance. However, DSM technology has introduced many new problems [4, 6], challenging the feasibility and efficiency of current CAD tools. One of the most critical issues is timing closure, which is mostly due to the growing importance of the wiring effects. Various improvements have been made, among which the Physical Synthesis technique is popular. However, the timing closure problem is far from solved, and will get worse

In this paper, a Module-Based design flow is presented. There are two basic elements of this flow. One is to build multiple versions for each soft module with different area/delay tradeoffs, letting the physical design stage choose the appropriate version to meet the timing requirements. The second is a module placement and routing stage, during which the appropriate version choice is made. Although it is not claimed that the timing closure problem is completely solved, compared to Physical Synthesis, it does produce (1) better achievable clock cycles, and (2) shorter run-times.

DAC'04, June 7–11, 2004, San Diego, California, USA

The paper is organized as follows. In Section 2, the formulation of the timing requirements is given. A Physical Synthesis flow is described in Section 3, with which our flow is compared. The Module-Based flow is detailed in Section 4. Experimental results are reported in Section 5 and Section 6 concludes.

2. THE TIMING REQUIREMENTS
Synchronous digital circuits with one single clock source are considered. The goal is to implement the input netlist with a small clock cycle time with the chip area given a lesser priority. A design can be specified as a set of modules that are inter-connected. Hard modules, such as memory blocks or other IP blocks, are both logically and physically fixed. Soft modules contain only initial Register-Transfer-Level (RTL) descriptions and need logic optimization and physical design.

A combinational path, abbreviated as "path", is a connection from one register to another (or the same) register, through logic gates and wires. A path delay consists of three parts, the internal delay in the driver, the wire delay denoted by d_w and the internal delay in the load module. Wires can be buffered to speed up signal propagation. If the driver and load modules are the same, we call the path an "internal path". If all the soft modules are flattened such that the design only contains one large soft module, the number of internal paths can be huge. Denote path delay by $d(path)$ and the number of clock cycles allowed for the signal propagation by $n(path)$. For a given clock cycle K, $d(path)/n(path) \le K$ must be satisfied for all paths. In the following discussion, we will consider only single-cycle paths for simplicity. Usually the desired clock cycle K_0 is given as a starting point. However, when the flow cannot meet this, two options are possible. One is to increase the clock cycle and run the flow again. The other is to modify the input netlist (possibly the behavior of the circuit) and run the flow again with the same clock cycle. The latter is beyond the scope of our discussion. We focus on the former; hence the problem can be stated as follows: Starting from clock cycle K_0, find the minimum working cycle $K_W \ge K_0$ such that all path delays are satisfied, and the chip area is minimized. Run-time is used as a criterion for the efficiency of the design flow; if a flow cannot find a solution for a given clock cycle, it should terminate early so that a new run can be started with an adjusted clock cycle.

It is convenient to represent the timing constraints by "required/arrival times". Required times are defined on pins of the modules and all register inputs. The real times when the signals propagate to those nodes, known as "arrival times", must be no later than the required times. The difference between the required time and arrival time of a pin is called the "slack". Negative slack means a timing violation. Constraints in the form of arrival and required times are the input to a synthesis tool. Hard and soft modules differ in the setting of the timing constraints. Inside a module, hard or soft, is a sequential circuit composed of registers

and combinational logic. The internal implementation of a hard module is fixed; so the output pins carry arrival times accounting for known internal delays, and similarly the input pins carry required times. For soft modules, the register-to-output-pin and input-pin-to-register delays are the result of synthesis from the RTL. A soft module is said to be fast if such delays are short, or slow otherwise. This can be controlled by setting arrival times on the module's input pins and required times on its output pins. Also a set of implicit constraints require that arrival times are zero on register outputs and required times are K on register inputs.

A design flow usually contains two stages, logic synthesis which optimizes the logic netlist, and physical design which places the components and builds the interconnections. However two major problems acerbate timing closure: (1) Wire delays are part of the path delays. These cannot be measured with much accuracy until routing is done. (2) Even if the wire delay can be safely ignored, how to distribute the path delay between the driver and load modules (both may be soft modules) remains a question. The two design flows to be described in the next two sections provide possible solutions: The **Physical Synthesis** approach, tries to increase the accuracy of pre-synthesis wiring estimation. The **Module-Based** flow tries to reduce as much as possible the need for pre-synthesis wiring prediction. Instead it creates "versions" of the soft modules representing flexibility. Of equal importance, it uses a physical design method that can adapt to the wire delays on-the-fly, thus choosing the appropriate versions.

3. A PHYSICAL SYNTHESIS FLOW

The main idea of the Physical Synthesis flow is to floorplan the locations of the modules and their interconnections before (logic) synthesis starts.

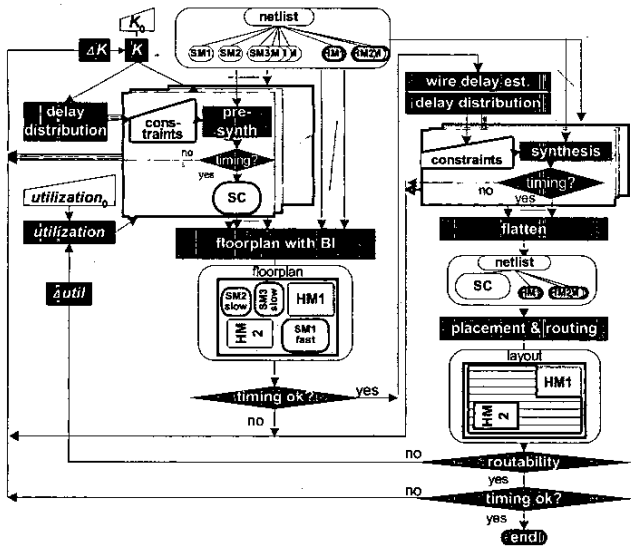

Figure 1: The Physical Synthesis flow.

A typical Physical Synthesis flow is shown in Figure 1. One run of the flow contains three main stages: floorplanning, module synthesis and physical design. Check points are set up in the flow to catch timing and routing violations and trigger new iterations. The modules are floor-planned such that their relative locations on the layout are determined. Buffer locations are also planned. This floorplanning information is used to estimate wire delay and generate timing constraints. However, the floorplanning requires sizes and shapes of the soft modules, which are the results of logic synthesis. To break the loop, each soft module is pre-synthesized (pre-synth) to obtain an estimated size. The real synthesis stage produces the cell netlist for each soft module with the constraints derived through wiring estimation. Retaining the initial partitioning of the synthesizable logic into soft modules is not necessary for this flow, because powerful commercial synthesis tools can handle synthesis of millions of the cells (perhaps these tools internally partition the big netlist into small soft modules). Since the synthesis tool available to us, *SIS* [3,4,1], cannot handle big flattened netlists, we run it for each soft module and then flatten the modules to get one big cell netlist. The last stage is the physical design. The floorplan generated earlier is maintained, but some adjustment might be needed, because the areas of the synthesized soft modules may be different from those used in the floorplanning stage. Also, the soft modules have been flattened.

The Physical Synthesis flow may still suffer from the following problems: (1) Pre-synthesis may produce wrong estimation of the module sizes. (2) The delay distribution problem, mentioned in Section 2, remains. (3) Area utilization for gate placement is hard to determine at the early stages.

4. THE MODULE-BASED DESIGN FLOW

The Module-Based design flow is shown in Figure 2. It includes two stages. The first is the preparation of the versions of the soft modules, turning each soft module into two (or more) versions of hard implementation with different area/delay tradeoffs. Which version is to be used in the final layout is determined at the second stage, the block-level physical design.

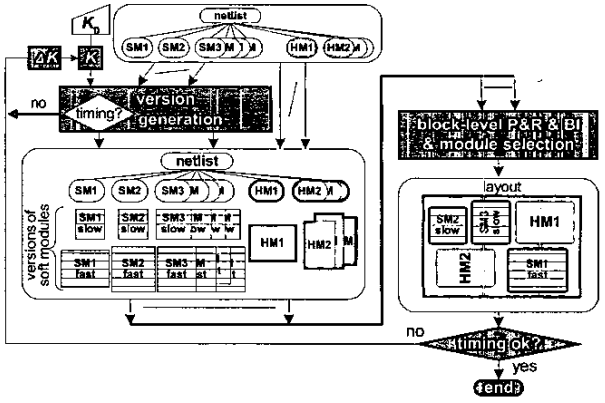

Figure 2: The Module-Based design flow.

4.1. The versions of soft modules

A version of a soft module is associated with a set of timing constraints, i.e., arrival/required times. We experimented with a soft module having two versions: "fast" and "slow". Usually the fast version has a larger area than the slow version. The implementation of a version involves both logic synthesis and gate-level physical design. After version generation, all modules become hard but with the distinction that the previously soft modules have several versions. Of course, the initial hard modules could have been given in several versions as well, but this was not considered.

To generate the "fast" versions, all output pins are given the tightest constraints, that is, $t_{REQ}(opin)$=0, and the input pins are given $t_{ARR}(ipin)$=K. Obviously the set of constraints can never be satisfied, but they push the logic synthesis to make the module "as fast as possible". The result is fast, in the sense that the inter-module paths from/to this module receive as large delay margins as possible for the wires and other modules. The fast version is normally large in area. The generation of the fast version does not involve any wiring estimation. An initially hard module has only one version, which we assume to be a fast version. All soft modules have their fast versions implemented both logically and physically before the generation of the slow versions begins.

The slow version is synthesized with looser constraints, but arbitrarily loose constraints - infinite required times on output pins for instance - may cause the slow version to be useless in the following physical integration of the modules. When the slow version of a soft module is being made, all modules that this module connects to are assumed to be fast. The arrival time $t_{ARR}(ipin)$ on the input pin (load side) is derived assuming the driver side module uses the faster version,

$$t_{ARR}(ipin) = d_f + d_W,$$

in which d_f is the known internal delay of the (fast) driver module and d_W is the wire delay (to be clarified later). An output pin may have multiple fanouts (use fanout=2 as an example). Now the load side modules use fast versions with internal delays d_{f1} and d_{f2}. The wire delays of the pin-pin connections are d_{w1} and d_{w2}. Therefore, the required time on the output pin of the slow soft module is

$$t_{REQ}(opin) = K - \max\left(d_{f1} + d_{w1}, d_{f2} + d_{w2}\right).$$

The synthesis of a slow version may not meet its timing requirements, in which case the slow version is discarded.

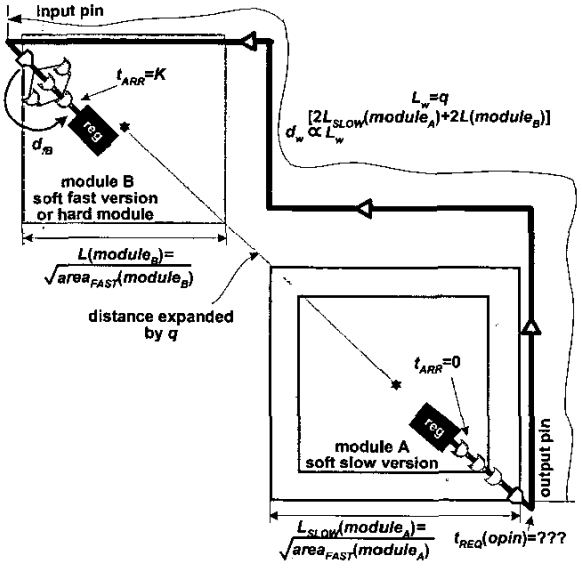

Figure 3: Deriving the constraints for the soft slow version.

Now the problem is to determine d_W when creating the set of constraints for the slow version. Instead of using techniques like floorplanning to estimate d_W, we conjecture that a module, if using a slow version, is likely to be placed close to the modules it is connected to, reasoning that short wire lengths are the reason for it being able to be slow. This "closeness" is the basis for the determination of d_W. An example of deriving the required time for an output pin of a slow soft module (module A) is shown in Figure 3. The load side, the fast version of module B, contributes a known internal delay d_{fB} to the path delay. The size of the slow-version of module A is not known yet. But the fast version has been synthesized, laid out, so its area $area_{FAST}(module_A)$ is known. Using the area of the fast version to approximate that of the slow version is safe, because the slow version is usually smaller than its fast version. The slow version is assumed to be a square with area $area_{FAST}(module_A)$. Module B (fast) has been physically designed, so not only its area $area_{FAST}(module_B)$ but also its X and Y sizes are known. We simply use a square with the same area to approximate module B, because otherwise the orientation of the module needs to be guessed if its X and Y sizes are dissimilar. To make such approximation safe, we set an aspect ratio constraint when doing the gate-level physical design for the soft modules. The two square modules are assumed to be placed corner to corner. "Closeness" is measured by the center-center distance of the two modules with an expansion factor of q, as shown in Figure 3. The Manhattan wire length can now be obtained:

$$L_W = q\left[2L(module_A) + 2L_{SLOW}(module_B)\right]$$
$$= 2q\left[\sqrt{area_{FAST}(module_A)} + \sqrt{area_{FAST}(module_B)}\right].$$

The two pins under consideration are assumed (conservatively) to be located on the two extreme corners of the modules, and the Manhattan connection with good buffering leads to $d_w \propto L_w$. If the net is multi-fanout, we consider each output-input pin pair with the above method and choose the tightest one to get the required time on the output pin. The arrival times of the input pins of the slow version can be derived similarly. **Note**: the derivation of the arrival/required times does not rely on floorplanning. Experimental results show that 1.5~5.0 are feasible values for q. A single q is used for all soft modules. We used q=3.0 to run the experiments and report our results.

The process of version generation starts with K=K_0, where K_0 is the user given desired cycle. All the fast standard-cell (SC) versions are generated first. Any timing violation in this step causes an update of the clock cycle, K=K+ΔK, and all previously built fast versions are discarded. When all the fast SC versions are generated, an "obvious timing violation" test is performed. This checks, for all inter-module paths with both driver and load sides using fast SC versions, if the timing constraints are met (assuming small wire delays, computed also based on the "closeness" described by q). If the test is not passed, there is no way to find a solution for the current clock cycle. If the test is passed, the process enters the next step, the generation of the slow SC versions. For these, the constraints on the pins are created based on the results of the fast versions and parameter q. If the generation of a slow module fails, the version is simply discarded rather than launching a new iteration with a modified clock cycle. The main purpose is to save run time. All modules can be synthesized with circuit structures other than standard-cell. A regular PLA-based structure called Checkerboard [9] was also used in our flow.

4.2. The physical design stage

The task of this stage, is to place and route the modules, insert buffers on wire, and select versions such that all timing constraints are satisfied and the chip area is minimized. In some sense, the Module-Based design flow postpones the area/delay tradeoffs to the physical design stage. The *Fishbone* scheme with

buffer insertion [5,10] is used as the physical design algorithm of the Module-Based design flow. The only modification is that now a former soft module can have several versions. Since the *Fishbone* algorithm is based on simulated-annealing, randomly making version selection is an easy adaptation.

5. EXPERIMENTAL RESULTS

Since benchmark examples, for such large designs of interest, are not available to us, a "diagram-based" circuit generator was created. A circuit diagram describes the modular structure of a circuit system and the interconnections of the modules. However, the detailed netlist of each module is typically unknown. Circuit diagrams are easier to obtain than the real circuits. With a diagram, the hard modules such as memory blocks are sized appropriately for the widths of their data and address buses. To emulate the soft modules, which implement random logic like control structures, we randomly associated these with existing synthesis benchmarks. Pin matching was performed during the association. Five circuit diagrams were used, covering applications like complex processors, wireless computing chips and real time image processing chips. A pool of 77 circuits from MCNC synthesis benchmark suites [8] was used for the creation of the soft modules. The starting clock cycle of a design was taken as the cycle of the slowest hard module in the design. The characteristics of the examples are given in Table 1.

Table 1: The testing circuits

circuit	application	#HM	#SM	#I/O	#global net	target cycle(ps)
A1	CISC	11	8	114	668	2000
A2	CISC	12	8	174	964	2000
A3	unknown	10	10	166	1160	2000
A4	wireless computing	17	41	403	2832	2222
A5	real time image processing	10	15	158	1008	5000

In this experiment, a 0.18-micron technology was used[1]. Five metal layers are available, the three lower layers for short or local connections and the two upper layers for long or global connections. Logic synthesis is done using *SIS*. The physical design part of the Physical Synthesis flow uses the *Cadence Silicon Ensemble* version 5.3 (*QPlace* plus *Warp Router*) coupled with a high-quality floorplanner that we wrote based on simulated-annealing, Steiner Trees, and buffer insertion. This floorplanner was used in previous work where it was shown that it can offer slightly better area/delay than the block-level placer used in our Module-Based flow, if these two are treated as standard-alone algorithms [5]. When iterations occur, the increment of the clock cycle is chosen as:

$$\max\{-MostNegativeSlack + 20\text{ps}, 100\text{ps}\}.$$

The achievable clock cycle, area and run time are compared in Figure 4. On average, the Module-Based flow gives 13% smaller cycle times than the Physical Synthesis flow. Chip areas are not directly comparable, since the chips produced by the two flows may work under different clock periods, and the minimization of the clock cycle had higher priority than the area in the experiments. The Module-Based flow, on average, takes 77% less run time than the Physical Synthesis flow.

[1] For smaller geometries, problems associated with wires will be more severe. This should make our method even more effective.

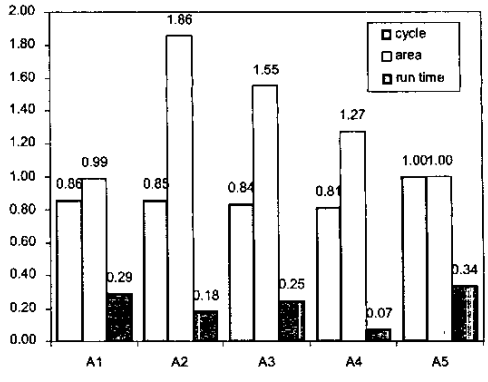

Figure 4. Comparison of the results (%) Module-Based flow versus Physical Synthesis flow

6. CONCLUSION

A Module-Based design flow was presented, which contains a version generation stage and a physical design stage. Key elements are the creation of several versions of the soft modules and the use of a physical design stage that does simultaneous place and route using accurate wire delays. Soft modules were implemented in fast and slow versions. Version generation does not rely on wiring estimation. Versions are used at the physical design stage to meet real wiring situations. Experimental results are good; the Module-Based design flow achieves shorter clock periods than the Physical Synthesis approach and requires significantly less run time.

7. REFERENCES

[1] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton and A. Sangiovanni-Vincentelli, *SIS: A system for sequential circuit synthesis*, Tech. Rep., UCB/ERL M92/41, Electronics Research Lab, University of California, Berkeley, May 1992.

[2] R. Brayton, G. Hachtel, C. McMullen and A. Sangiovanni-Vincentelli, *Logic minimization algorithms for VLSI synthesis*, Kluwer Academic Publishers, 1984.

[3] R. Brayton, G. Hachtel and A. Sangiovanni-Vincentelli, "Multi-level logic synthesis", *Proceedings of the IEEE*, vol. 78, Feb 1990.

[4] R. Bryant, K-T. Cheng, A. Kahng, K. Keutzer, W. Maly, R. Newton, L. Pileggi, J. Rabaey, A. Sangiovanni-Vincentelli, "Limitations and challenges of computer-aided design technology for CMOS VLSI", *Proceedings of the IEEE*, vol. 89, issue 3, Mar 2001, pages 341-365.

[5] F. Mo and R.K. Brayton, "Fishbone: A Block-Level Placement and Routing Scheme", *International Symposium on Physical Design*, Apr 2003, pages 204-209.

[6] J. Cong, "An Interconnect-Centric Design Flow for Nanometer Technologies", *Proceedings of the IEEE*, vol. 89, no. 4, Apr 2001, pages 505-528.

[7] The ISPD 98 circuit benchmark suite, http://vlsicad.cs.ucla.edu/~cheese/ispd98.html

[8] MCNC92 benchmark, http://www.cbl.ncsu.edu/CBL_Docs/lys92.html

[9] F. Mo and R.K. Brayton, "Checkerboard: A Regular Structure and its Synthesis", *International Workshop on Logic and Synthesis*, May 2003.

[10] L.P.P.P. van Ginneken, "Buffer Placement in Distributed RC-Tree Networks for Minimal Elmore Delay", *International Symposium on Circuits and Systems*, 1990, pages 865-868.