A MODULAR FRAMEWORK FOR VLSI DESIGN MANAGEMENT: ENHANCING PRODUCTIVITY AND EFFICIENCY

AUTHOR 1 : THADURI RAHUL

STUDENT

SIDDHARTHA INSTITUTE OF TECHNOLOGY AND SCIENCES, HYDERABAD, INDIA.

AUTHOR 2 : MADIPALLI SUMALATHA

ASSISTANT PROFESSOR

SIDDHARTHA INSTITUTE OF TECHNOLOGY AND SCIENCES, HYDERABAD, INDIA.

Abstract:

The increasing complexity of Very Large Scale Integration (VLSI) design necessitates the development of effective management frameworks that can enhance productivity and efficiency throughout the design process. This paper proposes a modular framework for VLSI design management, aimed at streamlining workflows, improving collaboration among multidisciplinary teams, and optimizing resource allocation. The framework integrates key components such as design specification, verification, testing, and project management into a cohesive system that supports modular design principles. By leveraging advanced methodologies like agile project management and concurrent engineering, the proposed framework fosters a dynamic environment where teams can adapt to changing requirements and rapidly iterate on designs. A case study of a leading semiconductor company demonstrates the framework's effectiveness in reducing time-to-market and improving overall design quality. Quantitative metrics indicate a significant increase in productivity, with design cycle times reduced by 25% and a 30% enhancement in cross-functional team collaboration.The findings underscore the potential of a modular approach to VLSI design management, providing a scalable solution for addressing the challenges posed by the evolving semiconductor industry. This framework not only serves as a guide for VLSI designers and managers but also contributes to the ongoing discourse on optimizing design processes in complex engineering environments.

Key words: Conceptual model, VLSI design cycle, VLSI design process, design process management, description methodology

INTRODUCTION

The Very Large Scale Integration (VLSI) design process has been applied to many systems to achieve new prospects for high-performance computing, telecommunications and electronic equipment intended for everyday use (e.g., entertainment). Progress in VLSI technology has continually increased the number of fabricated devices in a single integrated circuit (Moore's Law). The increase in complexity, which involves considerable bookkeeping and management tasks, has made it more vital to control the design process to maintain quality, reliability and extensibility. This control involves "definition, execution and control of design methodologies in a flexible and configurable way" (Kalavade and Lee, 1994; Kalavade *et al*., 1995). The VLSI design process is typically described as an iterative process that refines an idea for a manufacturable device through several levels of design abstraction (Kishore and Prabhakar, 2009). It involves a series of procedures, from specification to fabrication, in which the integrated circuit is produced. Starting with abstract requirements, the process involves converting these requirements into a register transfer description, e.g., control flow, registers and arithmetic and logical operations, which is simulated and tested. The design process then shifts to circuit representation, e.g., gates, transistors and interconnections. Simulation is also utilized at this level to verify each component. Finally, the geometric layout of the chip is produced in the form of geometric shapes representing circuit elements and their interconnections. The layout blueprint aims at achieving area compactness and accuracy in routing and timing (Navabi, 1998; Perry, 2002). Such a process requires different design tools (e.g., VLSI CAD) and management apparatus (e.g., database systems). "As a design is processed, it must be passed from tool to tool. For example, the designer may use a schematic capture tool for initial input, then they wish for their design to be minimized and finally simulated" (Hodges and Rounce, 1991). Comprehensive management of the design is very important. Accordingto Hodges and Rounce (1991):

It is all too easy to make changes to a design without documenting them, or without keeping backup versions in case references to these changes need to be made at a later date. In a similar way, it is equally possible to accidentally delete files with no means of retrieval… Problems also arise with building up a hierarchy of blocks, with each level making use of the blocks in the levels below…the various tools often do not support this system of hierarchy directly...

Workflow-based methodologies have been used to coordinate execution of multiple tasks and activities in VLSI design processes (Shepelev and Director, 1996; Marinescu, 2002). Database systems have been incorporated in many VLSI design systems (Chu *et al*., 1983; Hollar *et al*., 1984; Jullien *et al*., 1986; Katz, 1986) to provide an environment for design data management or tool integration.

In the general area of modeling a design process, several basic diagrammatic techniques are available, such as using a weighted (completion time) directed graph to represent the flow of design, where nodes represent tasks and edges depict sequencing of tasks. Markov chains (Johnson *et al*., 1996), signal flow graphs (Eppinger *et al*., 1997) and petri-nets (Magott and Skudlarski, 1989) are other examples of graphing techniques used to analyze design processes. According to Xiu (2008):

System-level design does not involve implementation detail. It is the approach of viewing the chip in a big-picture perspective. It abstracts away the full detail of the design, retaining just enough features to validate that functions embodied by the design can perform the specified design goal and can satisfy the performance criteria… Preferably, the system-level study and modeling should also support the smooth migration to downstream implementation.

VLSI design is analogous to the design tasks involved in building construction Wikibooks, 2011.First, we obtain an architectural drawing based on requirements and specification. Floor plans and constraints are then laid out based on the connectivity/accessibility/size of internal spaces. This is followed by depicting streams of flow (e.g., electrical, water) in the building, including various requirements, synchronization and triggering among different flows (e.g., supplied power, uniformity of flow). There is alsothe issue of maintaining integrity within spaces by specifying partitions and shields.

Similarly, VLSI design evolves from an architectural blueprint drawn on the basis of specifications of the product. Floor plans of the chip are based on the connectivity/accessibility/space of components with constraints on placement of the blocks. Electrical plans are then laid down in a power- grid topology that includes power requirement supplied over the topology for uniform distribution across all parts of the chip and standard-cells. There is also the phase of synchronization that involves timing analysis and regioning to place cells (circuits) that share data near each other to minimize timing and reduce wire lengths.

This study introduces a diagrammatical methodology that produces a purely conceptual framework that specifies flows of artifacts in the VLSI design process. The methodology is a descriptive process characterized by being independent of technical notions and features uniformity in application (e.g., at different levels and of various artifacts). It can provide an environment for managing the design process and supporting high-level control policies.

VLSI design cycle: The VLSI design cycle (Fig. 1) consists of eight interdependent stages: system specification, architecture design, functional design, logic design, circuit design, physical design, fabrication and packaging. The cycle moves from an abstract specification to a more detailed design that can be assessed, tested and implemented.

The first stage is the system specification stage, which is a high-level representation of the system. During this stage, the design specifications are determined and set, taking into consideration certain factors such as performance, functionality, physical dimension, design technique and fabrication technology. Size, speed, functionality and basic architecture specifications of the system result from this stage. The next stage is architecture design, which describes such things as use of Reduced Instruction Set Computer (RISC) and determination of cache size to produce micro architectural specifications of the system.

The third stage is the functional design stage, which defines the main functional units and the interconnection between the units in the system.

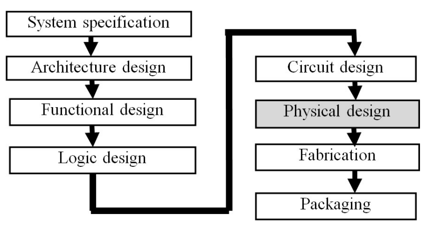

Fig. 1: VLSI design cycle

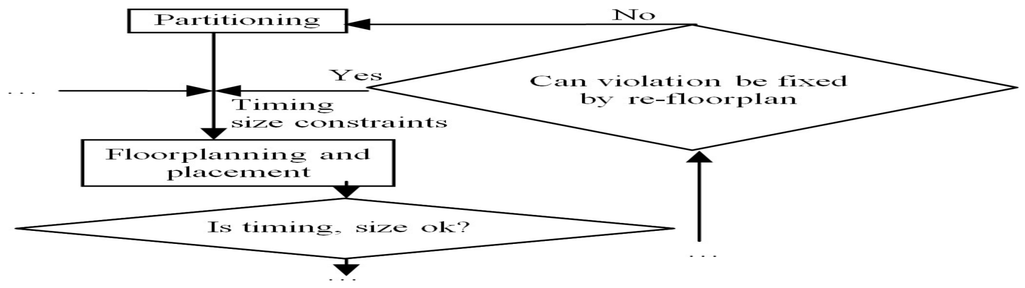

Fig. 2: A sample of the level of description following the general depictions of stages and phases of the VLSI design life cycle

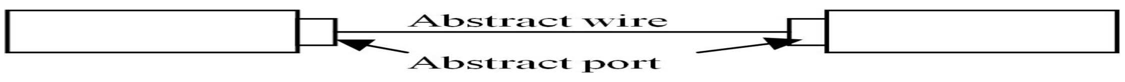

Fig. 3: Abstract wire and abstract port

At this stage, the system's behavior is set in terms of system input and output. In addition, the area, timing, power and other parameters of each unit are calculated. The output at this stage most of the time is a diagram showing relationships and timing between the system's units. Next, logic design is the stage that derives the logic description of the system, including Boolean expressions, data and control paths and register allocation.

The outcome of the logic design stage is a Register Transfer Level (RTL) description represented by use of Hardware Description Language (HDL). The circuit design stage deals with logic gates, transistors, interconnections and so forth. The physical design stage follows the circuit design; it takes the resulting circuit sketch and converts it into a geometric description. This stage will be explained in more detail to showcase our proposed representation. The design is then sent to the fabrication and packaging stages to produce and test the chip to ensure that all system requirements and specifications are satisfied.

Problem focus: Current descriptions: According to Sherwani (1999):

It is important to note that design of a complex VLSI chip is a complex human power management project as well. … As a result, design is usually partitioned along functionality and different units are designed by different teams. At any given time, each unit may not be at the same level of design. While one unit may be in logic design phase, another unit may be completing its physical design phase. This imposes a serious problem for chip level design tools, since these tools must work with partial data at the chip level…

The entire design cycle may be viewed as transformations of representations in various steps. In each step, a new representation of the system is created and analyzed. The representation is iteratively improved to meet system specifications. We observe from such a description that a great portion of the complexity involved is related to the management of transformations of representations. This is translated in our conceptualization to control the flow of representations (previously called artifacts), starting with drawing maps of flows of representations (which we later call flowthings) running through the design flow. These flowthings are created, transformed (processed), transferred, released and received by various functions along the design flow at different levels (a hierarchy). The resulting conceptual framework can be used to support designers with computer-aided tools to organize and manage the chains of tasks. Such a global conceptual framework for managing representations is missing from currently known methodologies. After giving such "conceptually simple" sketches of stages and phases shown in Fig. 1, Sherwani (1999) provides a flowchart-like depiction, partially shown in Fig. 2, as an example of a "design style" that "shows the physical design cycle with emphasis on timing." Utilizing such a flowchart-like methodology to represent a complex process has many well-known limitations in the software development cycle. Additionally, the description loses the essence of the process described by Sherwani (1999) as transformations of representations. The object-oriented approach to data management has also been proposed in the context of the VLSI design process (Heiler *et al*., 1987; Weiss *et a1*., 1986). For example, Chung and Kim (1990) propose a "design pattern", used repeatedly, modeled as an identifiable object called an abstract object and represented "internally" as a class, called an abstract class (as in UML). The abstract class includes an "entity type" with four possible values: cell (circuit), port, net (a wire connecting two ports) and constraint. Figure 3 shows a sample of such representation (Chung and Kim, 1990). Clearly, such a modeling is developed for technical purposes and does not help in development of a design environment.

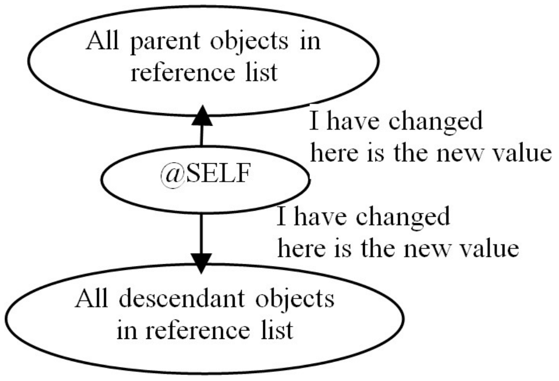
All parent objects in reference list
I have changed here is the new value
@SELF
I have changed here is the new value
All descendant objects in reference list

Fig. 4: Sample representation used in selecting a fabrication technology (Hekmatpour *et al*.,1991)

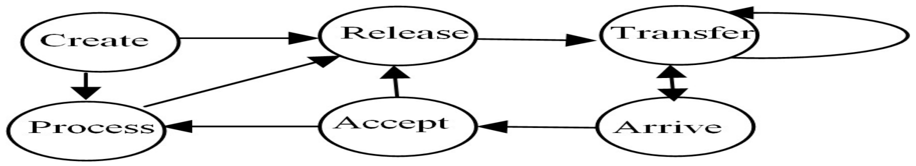

Fig. 5: Flowsystem, assuming that no released flowthing is returned

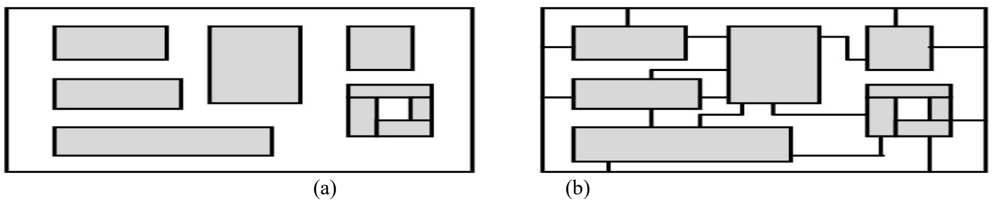

Fig. 6: Two sample descriptions (a): Layout of circuit blocks (b): Detailed routing

Another adoption of the object-oriented approach and at a level that follows a hierarchical representation of design space, is presented by Hekmatpour *et al.* (1991), who developed an application-specific model to customize the design environment:

For example, an important task in VLSI design is selecting a fabrication technology that best matches the constraints of the design space. In this selection procedure, engineers evaluate the current state of design in conjunction with design objectives and constraints…

[In Fig. 4] The generation of the list of objects… is accomplished by creating a referenceList object…

Note that the purpose of this discussion is to show the type of representation used, not to present a complete description of the discussed examples.

These object-oriented–based methods do not provide a suitable tool for managing a design environment. Technical details are usually intermixed with the programming "flow control" and data structures, with results no better than the flowchart-based description.

Alternatively, we aim at drawing a conceptual framework of "things that flow" (representations) in different streams of the design process to be used as a tracking mechanism for directing traffic during the design process. This is analogous to a real-time system in an imaginary city that tracks movements and *states* of different types of vehicles among streets, factories, stores and intersections. This includes the introduction of new vehicles, their processing (e.g., loading, changing color and shapes), their release and transfer from one place to another, arrival and acceptance when they enter factories, stores and so on. The conceptual model forming the backbone of such a scenario, called the Flowthing Model (FM), has been introduced in many publications and will be reviewed in the next section (Al-Fedaghi, 2010; 2011a-c; Al-Fedaghi and Al-Saleh, 2011; Al-Fedaghi and Fairouz, 2011).

Foundation: The flowthing model: According to Alberts *et al.* (1989), a design is an abstract entity that is gradually given a concrete form in the course of the design process. Especially in the case of VLSI design, it consists of a set of descriptions only. Since a design is represented by its description(s), the obvious way to describe design actions is as transformations between descriptions. We extend this line of thought by viewing a description as an integral element of a process, called a flowsystem, constructed from a generic operation to "handle" descriptions. A description is a thing that flows (a flowthing) in a flowsystem. Figure 5 shows a complete flowsystem. There is no description that is not created at a certain point. It can be processed (changed in form, but without producing a new description). It can be released and transferred, arrive at another place and be accepted (or rejected). In what other ways is a description handled? It can be stored, but storing is not a generic handling operation because created descriptions can be stored, processed descriptions can be stored. Similarly, descriptions can be copied, but copying can be performed on any one of the six generic types shown in Fig. 5. Accordingly, a description is a flowthing. A flowthing is a thing that is handled by these six generic, mutually exclusive operations. Figure 5 represents the flow system (denoted as flowsystem) of the flowthing. The environment of a flowsystem is called its sphere.

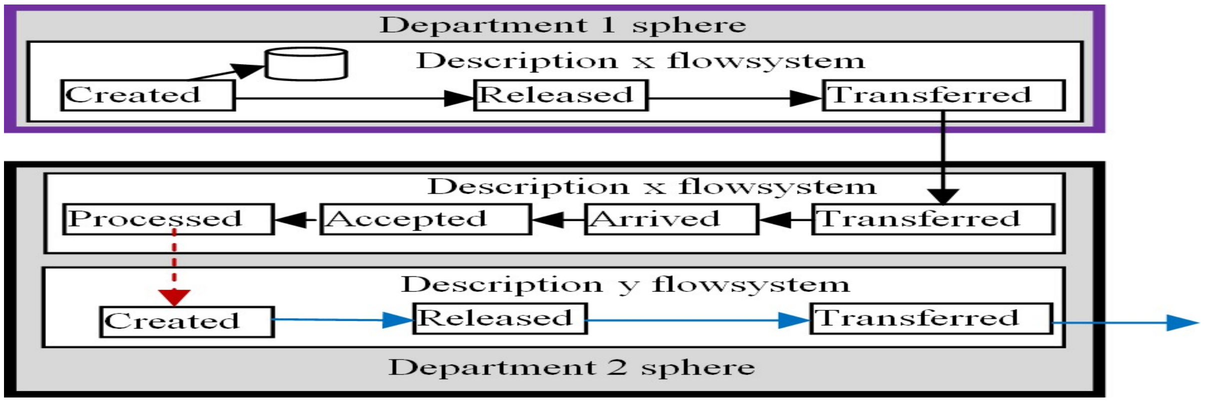

Fig. 7: FM-based flows of descriptions x and y

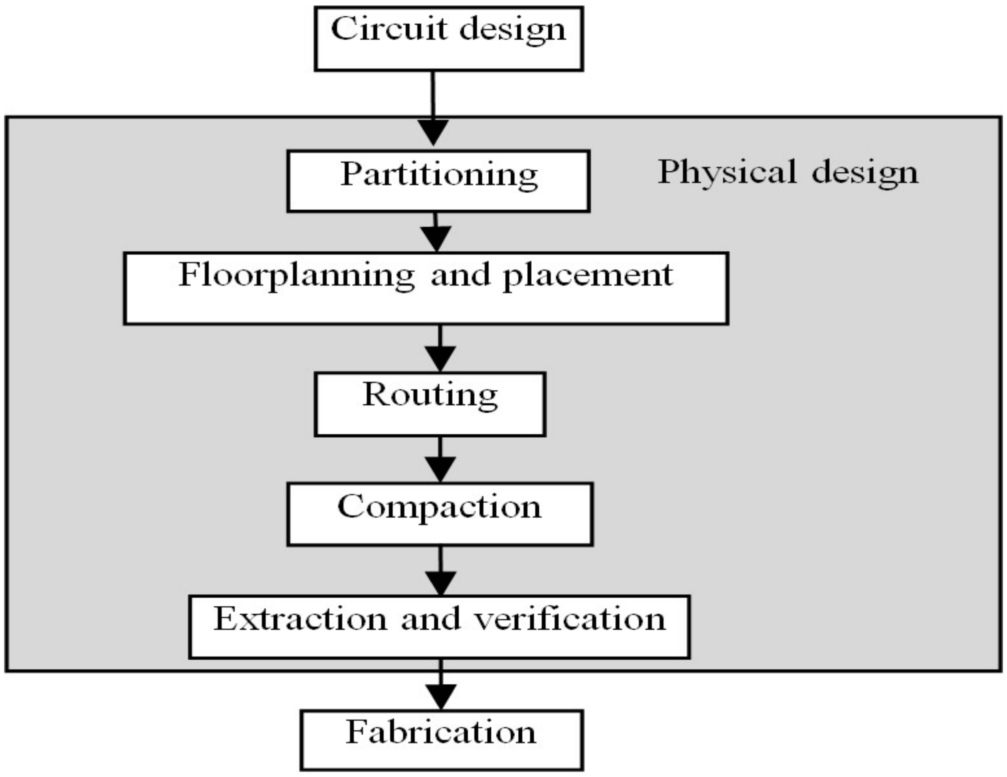

Fig. 8: VLSI physical design stage

Example: Suppose we have two descriptions, x and y, as shown in Fig. 6. Suppose description x is created by department 1 and sent to department 2, where it is processed to create description y. Figure 7 shows the FM-based description of such a process. In department 1's sphere, description x is created (and a copy is stored), released and transferred to department 2's sphere. Note that transfer represents the interface unit of the sphere with the outside. It is possible that a description is released but not transferred (e.g., waiting to fix a broken communication channel).

At department 2, description x enters, arrives and isaccepted in its flowsystem in the sphere. It is possible that it is rejected and thus sent back. Assuming that description x is accepted, it is then processed to *trigger* creation of description y. Triggering is represented by a *dashed arrow*. Description y is then released and transferred to somewhere outside department 2. Such a process includes two flows, one of description x and one of description y. The flow of description x passes through two flowsystems, forming a flowstream. There are two spheres in Fig. 7: that of department 1 and that of department 2. It is possible to have subspheres. For the sake of simplicity, when it is appropriate, we will sometimes merge arrival and acceptance as one stage called receive. In general, this Flowthing Model describes systems of flowthings such as, for example, information, data, money, signals and actions, in addition to artifacts such as descriptions (Al- Fedaghi, 2010; 2011a-c; Al-Fedaghi and Al-Saleh, 2011; Al-Fedaghi and Fairouz, 2011). Flowthings can exist in only six states: being created, released, transferred, arrived, accepted, or processed, with transformations among these states. When a description, as a flowthing, changes from one state to another, it has undergone a change of stage.

The physical design stage of the VLSI design cycle: Without loss of generality, in this section we describe the FM-based representation of the physical stage of the VLSI design cycle. This stage is divided into multiple phases: partitioning, floorplanning and placement, routing, compaction and extraction and verification, as shown in Fig. 8. The following description of the phases is summarized from Sherwani (1999). The first phase entails partitioning a circuit into subcircuits. This is a necessary step either because of the large number of logic gates that cannot be placed in a single chip or because of the limitations of I/O pins. Partitioning is a process that can be performed hierarchically until the size of each subcircuit fits in a single chip board. An important goal of the partitioning process is to minimize the interconnections (external wires) between the partitions (chip boards).

The partitioning process is followed by the floorplanning and placement phase, where the exact position of the circuit components in each partition is determined. In this phase, the components with their I/O pins and the interconnections between the components in each partition must be taken into consideration. The components connected to other components in other partitions must be placed at the edges of the partitions. Moreover, the components that belong to the same partition and are heavily connected must be placed close to each other to reduce wiring cost. The objective of the floorplanning and placement phase is to arrange the components in such a way that leads to minimization of the area arrangement layout and the interconnection area without violating any performance constraints. Determining the minimum area is done iteratively until minimum area layout is achieved. The result is then passed to the routing phase, where the interconnections between the partitions and their components are completed. The routing phase is defined as the process that finds the proper routes in the routing area used to connect the partitions and the components with minimum wiring space.

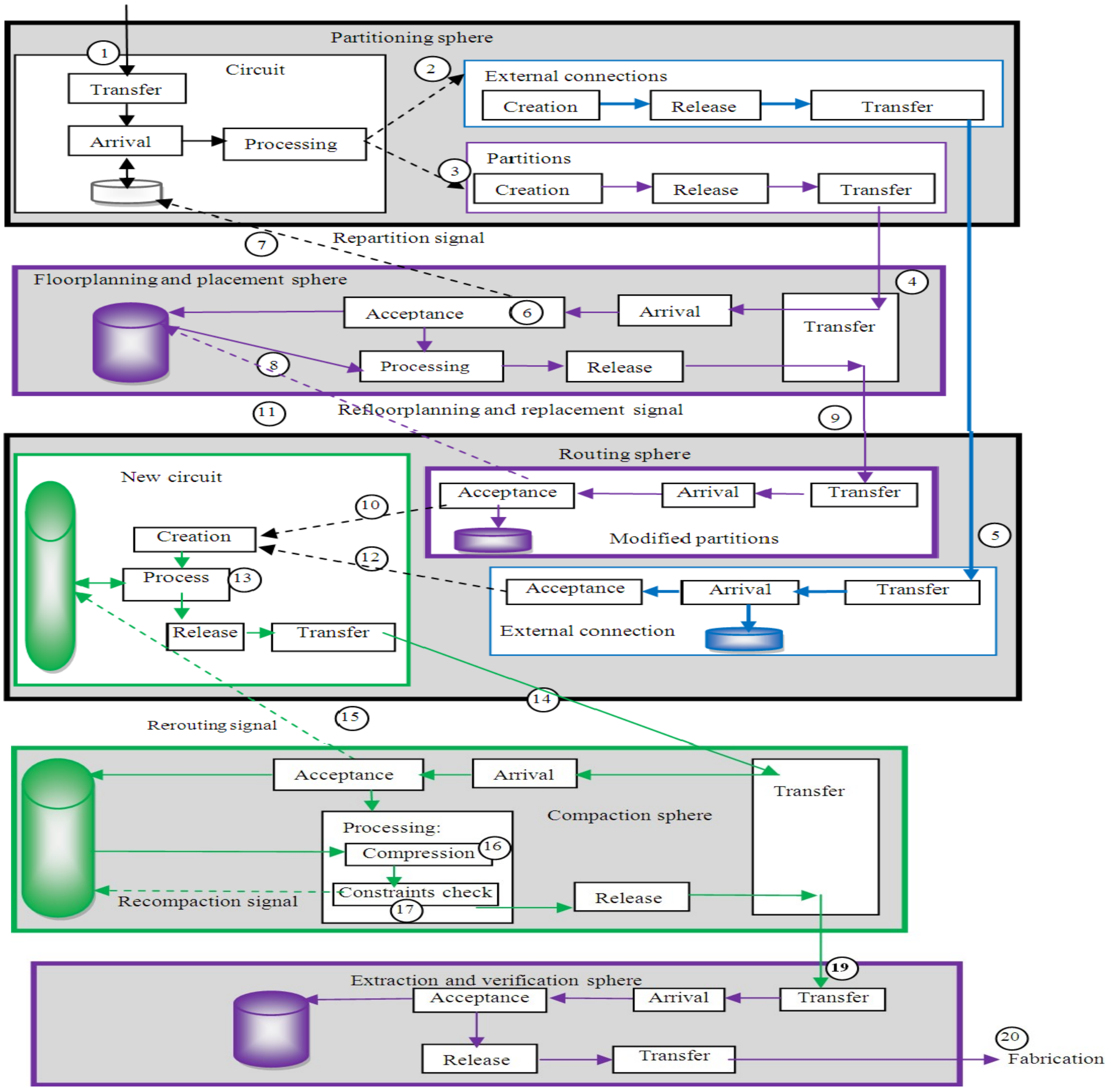

Fig. 9: FM-Based physical VLSI design stage

Routing consists of two steps: global routing and detailed routing. In global routing, the approximate interconnections between the components and partitions are determined while ignoring any geometric details. The resulting figure produces all possible routes in the area through which the wires should go. All this information is used in detailed routing to produce the geometric layout of the wires connecting thecomponents and partitions. The physical design is then passed on to the compaction phase, where the entire design is compressed to minimize the total area of the chip while maintaining all its performance and design constraints. The last phase is extraction and verification. All the geometric patterns—such as thewiring separation rule—are verified using DRC, theDesign Rule Checking process, to check that they meet the design rules set by the fabrication stage and that anydesign rule violations are removed. Many verification processes are available, such as Layout Versus Schematics (LVS) verificationn and Performance Verification and Reliability Verification. In LVS, layout (design) functionality is checked using a Circuit Extraction process, where a circuit representation is generated and extracted from the layout; the accuracy of the extracted description is then verified by comparing it with the circuit description. The design's geometric information is extracted and used in Performance Verification to determine resistance and capacitance to accurately compute the timing of each component with the interconnections. Moreover, thisinformation is also used in the Reliability Verification process to verify the reliability of the layout and ensure that electro-migration, self-heat and other effects will not lead the layout to fail. The final physical design is then carried to the next stage in the VLSI design process, the Fabrication stage.

FM-based description: FM can provide a complete picture of various artifact flows and their transformations in the physical design stage. Figure 9 shows the resulting FM conceptualization in terms of five spheres: partitioning, floorplanning and placement, routing, compaction and extraction and verification. Here, a sphere denotes a new task corresponding to a department, a designer, or a different role in the same organizational unit.

Partitioning: The partitioning sphere includes flows of circuits, external connections and partitions (sub-circuits). It consists of three flowsystems: a circuitsflowsystem, an external connections flowsystem and a partitions flowsystem.

Example: A circuit can be represented by a hypergraph, where the vertex set denotes the set ofpartitions and the hyperedge (a subset-called the pins-of the set of vertices) denotes the set of nets (external connections).

In Fig. 9, the circuit flowsystem (circle 1) receives the incoming circuit sketch from the previous stage (circuit design) and stores a copy for future use. The circuit is then processed (partitioned) based on the number of circuit components and design constraints. The distribution process of the components is performed using optimization partitioning algorithms (e.g., Kernighan-Lin algorithm) in order to minimize the number of interconnections between partitions. Thisprocess triggers the appearance of new flowsystemsthat have emerged as a result of creating new flowthings: an external connections flowsystem (circle 2) and a partitions flowsystem (circle 3).

The created partitions are released and transferred to the floorplanning and placement sphere (circle 4). The created external connections are also transferred to the sphere of the design flow system (circle 5), where they wait for the transformed partitions to again form a united chip.

It is not difficult to see that this description ofartifacts and their transformation can be used as a base for developing a manual or automated system to manage the tracking of various items in the system.

Floorplanning and placement: Partitions are transferred to this sphere. Note that it is possible that a partition that is transferred may not arrive (e.g., lost, destroyed); thus this FM-based model facilities such events, if required. Upon arrival (say, formal acknowledgment of receipt by another designer), the partition may be accepted (circle 6) after verification of design. Or, if not accepted, a repartition signal (request)is triggered to again partition the original circuit sketch (circle 7). Each partition is then stored in the databaseof the floorplanning and placement sphere (circle 8) andprocessed (e.g., the exact location of each component isdetermined). Eventually the redesigned partitions are transferred to the routing sphere (circle 9).

Routing: The resultant redesigned partitions flow to therouting sphere to be verified; if not accepted, they trigger a request for reprocessing in the floorplanning and placement sphere. The external connections are by now waiting in their flowsystem in the routing sphere.If accepted, these two artifacts are viewed as two portions that trigger the creation of a new circuit description (circles 10 and 12). That is:

- The partitions flowsystem in *the routing sphere* receives all partitions from the floorplanning and placement sphere
- The external connections flowsystem in *the routing sphere* receives external connections from the partitioning sphere

- Both events in (1) and (2) trigger creation of the new circuit, which is processed to join its two portions and perform global and detailed routing (circle 13). The design is then released and transferred to the compaction sphere (circle 14)

Compaction and extraction sphere: The next phase isthe compaction phase, where the modified design is compressed. As shown in Fig. 9, if the design is rejected, a rerouting signal is sent to the previous phase (circle 15). The compaction sphere contains twodifferent processes. In the first type of process (circle 16), the design is compressed and in the second process(circle 17), the constraints are checked. In case of any problem, the second process triggers a signal to the database to indicate that a violation has appeared (circle18).

Extraction and Verification sphere: This is the final phase, where the accurate functionality of the design

layout is verified in the acceptance process (circle 19). When the verification process is completed, the final version of the physical design is transferred tothe fabrication stage (circle 20).

CONCLUSION

This study proposes a high-level detailed view that assists in the management of the VLSI design process. Theresulting conceptual framework represents flows andtransformations of various descriptions (e.g., circuits, technical sketches) and is used as a tracking apparatus fordirecting traffic during the VLSI design process. It is applied in describing the physical design stage that includes partitioning, floorplanning and placement, routing, compaction and extraction and vertification. The flow-based description demonstrates the viability of a methodology that can be adopted for all stages and substages of the VLSI design process. The conceptual description can act as a foundation as well as provide a simpler or more detailed specification of the system. Many diagram-based conceptual descriptions become cumbersome because there is no restriction on the basic conceptual apparatus. Considera logical operator such as AND, OR and other flow- control mechanisms such as synchronization that can be erected when flows join together and when timing needs coordination. In the flowthing model, such notions apply, if needed, in the second level of specification. These tools are “unnatural” controls from the flow point of view. Imagine a civil engineer who draws a map of a territory that includes a river system. First, the engineer draws a topographic model, including streams, directions, joins (without, for example, worrying about the type of joining, e.g., relative speed of currents) and branches (e.g., relative division of water; one branch may have a dead end). The engineer then inspects the model and decides about damming, channelization, diversion, bridge construction and sand or gravel mining (Al-Fedaghi and Al-Saleh, 2011). For example, a simpler diagram of Fig. 9 is shown in Fig. 10 that can be used for nontechnical presentation. It is based on the same FM concepts, but the focus is on streams of flows.

REFERENCES

1. Azadeh, A. and F. Ghaderi, 2006. A framework fordesign of intelligent simulation environment. J. Comput. Sci., 2: 363-369. DOI:10.3844/jcssp.2006.363.369
2. Chu, K.C., J.P. Fishburn, P. Honeyman and Y.E. Lien, 1983. Vdd-A VLSI design database system. Proceedings of the Engineering Design Applications,(EDA' 83), KEG, Tsinghua, pp: 25-37.
3. Chung, M.J. and S. Kim, 1990. An object-oriented VHDL design environment. Proceedings of the 27th ACM/IEEE Design Automation Conference, (DAC' 90), ACM, New York, pp: 431-436. DOI: 10.1145/123186.123328
4. Eppinger, S.D., M.V. Nukala and D.E. Whitney, 1997. Generalised models of design interation using signal flow graphs. Res. Eng. Design, 9: 112-123. DOI: 10.1007/BF01596486
5. Haddad, J.S., 2009. Basic theoretical backgrounds ofthe engineering design procedure of a drying plant. Am. J. Eng. Applied Sci., 2: 466-470. DOI: 10.3844/ajeassp.2009.466.470
6. Heiler, S., U. Dayal, J. Orenstein and S. Radke-Sproull, 1987. An object-oriented approach to datamanagement: Why design databases need it. Proceedings of the ACM/IEEE 24th Conference onDesign Automation, June 28-Jul. 1, IEEE Xplore Press, Miami Beach, Florida, USA., pp: 335-340. DOI: 10.1145/37888.37939
7. Hekmatpour, A., A. Orailoglu and P. Chau, 1991. Hierarchical modeling of the VLSI design process. IEEE Expert, 6: 56-70. DOI: 10.1109/64.79710
8. Hodges, S. and P. Rounce, 1991. A VLSI design management environment. Design Manage.Environ. CAD, IEE Colloquium.
9. Dr. Naveen Prasadula (2023) Review of Literature of A Modular Framework for VLSI Design Management: Enhancing Productivity and Efficiency
10. Hollar, L., B. Nelson, T. Carter and R.A. Lone, 1984. The structure and operation of a relational databasesystem in a cell-oriented integrated circuit design system. Proceedings of the 21st Conference on Design Automation, Jun. 25-27, IEEE Xplore Press, pp: 117-125. DOI:10.1109/DAC.1984.1585784
11. Ismail, A., R. Aminzadeh, A. Aram and I. Arshad, 2010. Value engineering application in highwayprojects. Am. J. Eng. Applied Sci., 3: 699-703.DOI: 10.3844/ajeassp.2010.699.703
12. Johnson, E.W., L.A. Castillo and J.B. Brockman, 1996. Application of a Markov model to the measurement, simulation and diagnosis of aniterative design process. Proceedings of the Design Automation Conference, Jun. 3-7, IEEE Xplore Press, Las Vegas, NV, USA., pp: 185-188. DOI: 10.1109/DAC.1996.545569
13. Alberts, L.K., C. Huijs, N.J.I. Mars and L. Spaanenburg, 1989. A knowledge-based approach to VLSI-Design in an open CAD-environment. Microprocess. Microprogramm., 27: 77-84. DOI: 10.1016/0165-6074(89)90024-0
14. Al-Fedaghi, S., 2010. System-based approach to software vulnerability. Proceedings of the IEEE 2nd International Conference on Social Computing, Aug. 20-22, IEEE Xplore Press, Minneapolis, USA., pp: 1072-1079. DOI: 10.1109/SocialCom.2010.159
15. Al-Fedaghi, S.S., 2011a. Conceptual foundation for specifying processes. Int. J. Adv. Comput. Tech.,3: 265-278.
16. Al-Fedaghi, S., 2011b. Pure conceptualization of computer programming instructions. Int. J. Adv.Comput. Technol., 3: 302-313. Al-Fedaghi, S., 2011c. Developing web applications. Int. J. Software Eng. Appli., 5: 57-68.
17. Al-Fedaghi, S. and L. Al-Saleh, 2011. Specifying processes: Application to electrical power distribution. J. Comput. Sci., 7: 1729-1740. DOI: 10.3844/jcssp.2011.1729.1740
18. Al-Fedaghi, S.S. and A. Fairouz, 2011. Alternative flowdiagram for supply chain: Application to risks. AISS: Adv. Inform. Sci. Service Sci., 3: 111-121.
19. Jullien, C., A. Leblond and J. Lecourvoisier, 1986. A database interface for an integrated CAD system. Proceedings of the 23rd Design Automation Conference, June 29-Jul. 2, IEEE Xplore Press, pp:760-767. DOI: 10.1109/DAC.1986.1586175
20. Kalavade, A. and E.A. Lee, 1994. Manifestations of heterogeneity in hardware/software co-design. Proceedings of the 31st Design Automation Conference, (DAC' 94), ACM, New York, pp: 437-438. DOI: 10.1145/196244.196457
21. Sherwani, N.A., 1999. Algorithms for VLSI Physical Design Automation, 3rd Edn., Kluwer Academic Publishers, Boston, ISBN: 0792383931, pp: 572.