A Novel EDA flow for SoC Designs based on Specification Capture, Block-clustering and Bus-partitioning

Ashwin K. Kumaraswamy[*†], Ramalingam Kannan[*] and Indrajit Atluri[†]

[*]Institute for System Level Integration, Alba Centre, Alba Campus, Livingston, UK.
[†]School of Engineering and Electronics, University of Edinburgh, Edinburgh, UK.

Abstract

The RTL-to-GDSII tool suites are already being used in order to carry out the logical or register transfer level (RTL) design and synthesis, in tandem with floor planning, placement and routing, to expedite the process of ASIC design production. This paper proposes a novel EDA flow for SoC designs based on the block clustering and bus-partitioning techniques which starts at the design specification capture level finally giving the required SoC. Conventionally an UML (Unified Modelling Language) can be used to model reusable components at higher levels of abstraction and System level design languages (SLDL) such as SpecC, SystemC etc., provide a platform for generating "executable specification" that describe the functionality of the system along with performance, cost and other constraints, without including prematurely developed implementation details. Since the UML is not executable and the functionality depicted in the model cannot be verified for accuracy, the authors present a system modeling aspect, which binds these two technologies and provides a platform for interoperability between UML and SLDLs. The correctness of the specification is checked at this stage. Further, the methodology explained in this paper clusters both the macro and micro blocks, based on the communication traffic between each of the blocks, and performs intermediate floor planning to facilitate designers to estimate the area of the design, to reduce the avoidable increase in power consumption of the chip prior to placement and routing by assuring the placement of the clustered macro-blocks near the power grids and to have a timing efficient system.

1 Introduction

In the context of todays increasingly complex SoC's, designs consist of embedded software running on multiple processor cores, connected to memory and peripherals. As complexity grows, an increasing proportion of the software and the hardware peripherals constitute a SoC device. To cope with these ever increasing requirements the design abstraction level needs to be precise and informatic.

Recent advances in semiconductor technology facilitate the integration of many million gates on a single chip and result in the integration of Systems on Chip (SoC). Typical examples of SoC applications can be found in the multimedia domain. In those applications, generally, large amounts of data must be processed in parallel. However, performing such processing by software running on a high-performance processor or only by hardware is not usually efficient except for a few specific cases. An optimal combination of hardware and software and an optimal system partitioning into hardware and software is most desirable.

Studies into SoCs have been made for a long time. The first methodology is the one of Intellectual Properties (IP) re-use [1]. The waterfall and the V models for SoC design delve into interface synthesis which is required in building a system based on the IP. In this methodology, however, there are limitations and few issues to be addressed. A methodology of partitioning a system into hardware and software directly from its specification description called the co-synthesis has been studied as another co-design methodology [2]. In conventional co-synthesis, performance degradation was observed, since the target-architecture was composed of general-purpose processors and a bus. To address the above deficiencies studies of how to achieve global optimization by avoiding ending up only in local optimization at the time of hardware/software partitioning is presently taking place.

Since SoCs are often highly cost-sensitive, optimization of silicon area has the same level of importance as design efficiency to accelerate the SoC development process. Conventional approaches for hardware-software partitioning focused mainly on EDA aspects, such as algorithm and design flow improvements, which are however not sufficient for the development of cost-efficient SoCs. In this paper the authors developed a flexible flow which can be entered at any stage of the design process. This new EDA methodology for SoC design is based on specification capture, system partitioning, communication trafficking based block clusterered synthesis and floorplanning. This methodology envisages a proposal where the specification is captured using UML and based on the specification capture system partitioning is performed using SpecC [3,4] as a System Level Design Language (SLDL). Further, the advantages of the specification capture and system partitioning techniques in tackling problems which arise during SoC design development by employing communication trafficking based block clustered synthesis and floor-planning is explicitly discussed.

2 Background and Related Work

The need for co-designing hardware and software has long been pointed out for the development of SoCs and co-design tools have been developed by many projects. Co-design can be roughly classified into two approaches, software oriented partitioning and hardware oriented partitioning. The difference between the two is whether the functional specification model is written as a software model or hardware model.

Most methods are focussing on hardware/software partitioning techniques to transform functional specifications into optimal system architecture. Studies of performing optimal hardware/software partitioning by applying proprietary algorithm or methodologies are under way. The method aiming at automatically generating a software model that can be cross-compiled and a hardware model that can be synthesized at a high level by applying the hardware/software partitioning algorithm is called co-synthesis. One of the merits of co-synthesis is that a hardware model and a software model can be partitioned with little user intervention. However this feature reduces visibility to the designer and hence controllability. Some co-synthesis methods take the phased approach, where certain stages are defined in the process to create optimal system architecture from functional specifications, in which designers can refine the functional specification. The merit of this approach is that architectural limitations can be revealed at intermediate stages thus identifying specification problems early that would otherwise come to the surface only in later stages.

As stated above, many methods for system development have been described in the literature. But applying those methods to actual SoC designs is still difficult due to inherent limitations.

One limitation is that the design process is based on target- architecture with general-purpose processors and a bus on which ROM, RAM, hardware accelerators, co-processors, etc. are allocated. It is very hard to estimate execution cycles for the hardware units, because the bus is shared by various data transfers.

It is preferable to use a model that is executable on the target-architecture as specification, because reusability of existing specification models is improved. Further, it is preferable that the specification model is a software model that can be cross-compiled. The reason for that is that refinement of the software model will directly influence the performance of the final system. In conventional co-design methods, a specification model cannot be cross-compiled because languages such as expanded C language or proprietary languages were used such that the specification model can be described in both a software model and hardware model. Under critical conditions, the difference between the evaluation result where hardware/software partitioning of the specification model was done and the one where only an execution model was used is considerable and it is very likely that a time consuming repetition of the partitioning process is required.

Finally, despite of recent remarkable developments of EDA tools, hardware design is still a time-consuming task. That is why it is important that a system simulation is fast and accurate. To that end, not only the verification on the RTL (co-verification) level but also on a higher level model such as C-based (co-simulation) must be done. However, in co-simulation methods, where basic blocks of conventional co-design projects are used, the achievable accuracy is not quite satisfactory.

3 Contemporary Challenges encountered in SoC Designs

As designs enter DSM technology of 0.25μ and below, the design community is confronted with several challenges. These challenges can be broadly be grouped into three categories:

1. Timing closures
2. Circuit capacity
3. Physical properties

(1) Timing closures: Conventional design flows use statistical wire load models to estimate metal interconnects for pre-layout timing analysis. For geometrics of 0.25μ and above, the new approach presented in this paper meets timing goals and constraints at the pre-layout stage, could be implemented to achieve same results after physical design. At DSM technology level, interconnect delays become significant and must be accurately estimated if timing closure is to be achieved [6].

Statistical wire load models are inaccurate because they represent a statistical value based on block size. The distribution of wire load at the mean value can vary greatly so that the interconnects on the tail of the distribution are significantly underestimated.

(2) Capacity: With DSM technology it is feasible to integrate >10million gates onto a single IC using 0.18μ or below technologies which introduces significant capacity challenges to many of the tools in the design flow. To

manage this level of complexities DSM designs adopt the following solutions:
(a) Hierarchical Designs: Hierarchical design flows support multiple levels within the design.
(b) Design Re-use: Design reuse integrates pre-existing blocks with newly authored ones.

(3) Physical Properties: At DSM levels of technology, several physical effects need to be accounted for, within the design flow. The evolution of DSM results in fewer device geometrics, more layer of metal interconnect, lower power supply voltages, millions of devices within a single IC, lower device thresholds and higher clock frequencies. These factors cause signal integrity issues and design integrity issues to be of greater concern than at more relaxed geometrics.

Coupled with technological problems at DSM level, SoC designs have to contend with larger abstraction of integration of hardware and software components.

SoC design is principally software development - HDL & embedded code. Without a quality approach to design, we are in serious trouble. Design processes exists to produce systems of consistent quality, reliably deliver systems meeting complex behavioral requirements, predict when systems will be complete, predict cost of development and production, identify milestones during development to facilitate mid-course correction, enable efficient team collaboration. SoC design is principally software development - HDL & embedded code. Without a quality approach to design, we are in serious trouble. Different projects follow different models. But experience has shown that the following are important:

- A well-defined requirements capture process.
- Prototyping & Modelling is needed at each level of abstraction in the design phase.
- Executable models are preferred to textual models.
- Verification should be built in at each stage, and early modelling should drive later phases of test.
- Feedback between stages is needed to refine the design.

A SoC design methodology should contain these features. For all these reasons we propose a new EDA tool flow for SoC methodology.

4 Proposed Design Flows

In case of traditional design flows, meaningful floor planning cannot take place until a gate level net list is available; this requires extremely computationally intensive and timing efficient logic synthesis or physically aware synthesis and in-place optimization.

Recently proposed CAD methodologies have dealt with timing closure problems, by expressing interconnect delay to the logic synthesis tool using wire load models, which estimates a nets capacitive load as a function of the fan-out of the net's driving pin and the gate utilization within the chip area that the wire load model represents. These methodologies deal with nothing but a customized wire load models which in reality is the iterative form of wire load models to attack timing closure problem.

Our present methodology is composed of three stages and a block diagram of the entire SoC flow is shown in Figure (3).

a. UML to SpecC
b. Design Entry level and verification
c. RTL to optimised hardware

4.1 UML-SLDL

We propose a methodology that can transform UML models into a known System Level Design Language (SLDL) (in this case SpecC). In other words, UML model acts as a "wrapper" to the SLDL's methodology. In UML each aspect of the SLDL's methodology can be modelled and refined. This has various advantages. The standardization of UML provides a base to revise the approaches to combine SLDL with object oriented analysis and design techniques (OOAD) techniques. One of the main directions for the joint application of SLDL and UML can be identified as modelling SLDL specifications with UML. This direction serves mainly the idea to make large SLDL specification better understandable and to give additional information (e.g. inheritance hierarchies, dependencies, pattern structures) for documentation purposes or as additional implementation advice. UML is mapped onto the SpecC methodology. Uniqueness, to this new methodology, is that the UML representation of the system is separated from the underlying methodology. This helps in unifying the ways a system can be represented in UML without worrying about the way it will be implemented. The reason behind using this approach is that the UML model can be ported seamlessly to any methodology. Thus we have to first understand how a system can be modelled in UML. Although there can be numerous ways of describing a system in UML, only one of these methods can be chosen. This way the code-generation (transformation) phase will be made easy.

A Hardware/Software co-designed system can be specified through the concepts of behaviours that interact via channels through ports and interfaces. There is a clear separation between computation and communication where behaviours model computation, and communication is modelled by using shared variables and/or channels. Keeping this in mind, the first step is to decide on the modelling of the different aspects of a system, i.e. computation and communication. Computation will consist of behaviours and their definitions. Communication will consist of ports, channels and interfaces. (Interfaces can also be used in the modelling of computation.)

Modelling of Computation

In UML, the behaviours are modelled as classes. The local variables and the functions are also modelled within the class in their respective positions. A composite behaviour will contain instances of other behaviours. These compositions can be modelled using associativity. When breaking down behaviour into sub-behaviours, for structural hierarchy, generalizations can be used.

There can be two types of hierarchy: structural and behavioral. Structurally, behaviours can be broken down into sub-behaviours and these into sub-behaviours, and so on. Designs are specified in a hierarchical manner using top-down functional decomposition (behavioral hierarchy). Both these hierarchies correspond to the concept of generalization and associativity in UML [5].

Modelling of Communication

To model interfaces, UML's interface notation is used. An interface is like an abstract class that consists of a set of method declarations. Interfaces can also be placed in a hierarchical fashion. Behaviours can, optionally, "realize" single or multiple interfaces. The channel or the behaviour that realize the interfaces should supply the definitions for the method declarations.

The stereotype, <<channel>>, is used to represent a class as a channel. Channels are also modelled in the same manner as behaviour.

Ports can be modelled in two ways. A port can either be a simple variable or another Interface or Class. In order to identify an object as a port, the <<port>> stereotype is used. If the port is declared as a simple variable of type type1, the variable declaration in UML will be as

```
name: type1 <<port>>
```

The <<port>> stereotype helps in identifying certain variables and also associations as ports, rather than local variables or instances respectively.

Modelling of Execution

The main problem in designing a system is the modelling of execution or show parallelism i.e., to represent behaviors that will be executing in sequence, parallel or pipelined. There are two different ways of showing this. It is well known that in UML different views are meant for different activities of modelling. Thus, these considerations have to be mentioned in more than one of the views. In the static view (class diagram) we annotate these using stereotypes. This is very helpful, because the class diagram shows the static structure of the system.

The problem of showing parallelism in the execution model can be solved through composition. Leaf behaviour, by itself will only perform its operations sequentially. If a component has to be modelled to execute in parallel or pipelined mode, then its behaviour can be further reduced into separate classes and its objects will be composed into the main component. These sub-behaviours can then be modelled to run in parallel or pipelined mode by specifying the mode of execution to the composite behaviour (main component). This can be done in the static view of the model. The actual execution of the composite behaviour can be modelled in detail, using Statechart diagrams and/or Sequence diagrams.

It was concluded that the State Machine view and the Activity view of the UML had enough notations specified to describe the internal behaviours of any component. Clocks can also be modelled as behaviours and can be made to generate events. These events can be used in other views to specify the timing characteristics of the system.

Transformation of Static View

Since SpecC is not an Object oriented language, there is no way of representing object hierarchies. Thus generalization is used to model behavioral hierarchy. In other words, behavioral hierarchy is modelled as a composition of multiple behaviours, according to the SpecC methodology. Therefore generalizations are transformed in the same manner as associations. A static view is shown in figure (1).

Transformation of State Machine View

The state machine view describes the dynamic behaviour of objects. Each object is treated as an isolated entity that communicates with the environment by detecting events and responding to them [7]. A state machine is a graph of states and transitions. Usually a state machine is attached to a class and describes the response of an instance of the class to events that it receives.

A State Machine view is used to model the internal behaviour of an object of a class. A state machine contains states that are connected by transitions. Each state is defined as some unit of time in which the object stays and performs certain operations, whereas transitions are instantaneous, i.e. they occur at zero time. When an event occurs, it may cause the firing of a transition that takes the object to a new state. When a transition fires, an action attached to the transition may be executed. Theoretically, this execution period is zero. State machines are shown as a state chart diagram (Figure 2).

The different components of a State diagram are:

1. State
2. Terminal States
3. Transitions

Now we propose a methodology which aims of solving the timing closure problem by extracting information at the highest level of abstraction i.e. during or along with the RTL stage.

4.2 Design Verification

Design verification starts in concurrence to the creation of specification as system specification drive the verification strategy.

In our proposed flow we can check the specification correctness at the UML stage. If the specification laid down seems incorrect we can input the correct UML model. The next stage is system behavior verification whose validation leads to suitable hardware and software architecture. Then the partitioning of software design into hardware and software is performed. Software level verification is token based and is not cycle accurate as yet.

RTL Verification:
RTL verification usually is logic simulation for functionality check. The RTL code is verified for syntax errors.
Software Verification:
Software verification is performed using soft prototype (emulation of the software).
Netlist Verification:
The RTL is synthesized and gate level netlist is generated. Netlist is verified using equivalence checking tool with RTL code as reference and netlist as implementation design.
Timing Verification:
Timing verification can be verified at various stages of the chip plan phase to ensure that the design meets the specifications and requirements.
Physical Verification:
Physical verification includes the DRC, layout versus schematic and SI checks.

Specman Elite generates test automatically by capturing rules from the design specifications. Automation involves generating function tests; this is the widely used verification language.

4.3 RTL and Software Code to Optimized Hardware

The RTL to final optimized hardware transition comprises of the following steps:

4.3.1 RTL: The logic functionality of the system is designed using either of the two HDLs based on requisite specification.
4.3.2 Intermediate Floor Planning: Once the RTL is ready, we observe the RTL simulation and infer the following information:

a. communication traffic between the blocks and sub-blocks
b. switching activity in the blocks
c. Gate count or area information from previous knowledge.

Based on the inferences from the RTL simulation an intermediatory floor plan is performed.

Block Clustering and Communication Traffic Estimation: The modulus operandi of this floor plan is we create a physical placing of the blocks and their sub-blocks depending of the communication traffic between them. This communication traffic information which comprises of the number of transactions, amount of data communicated, communication delays is collected from the design model or by simulation of functional specification. The higher the communication traffic the closer the blocks are placed and same is looked in the case of the address and data bus. Once the blocks and sub-blocks which are critical are placed closer and mapped we perform power planning by drawing power grids closer to the blocks which are critical with respect to the communication traffic. The power planning is based on the assumption that the components that have greater communication traffic requires more power compared to other components.

Some of the methods for estimation of the communication traffic between blocks are:
(a) Using UML: By inspection of UML we can determine the communication interaction between blocks and the requisite theoretical timing between the blocks.
(b) Switching activity: Through RTL simulation we can estimate the communication interaction between the blocks by which we can estimate the communication traffic between blocks.
(c) Previous Experience: Through observation based on experience with previous designs.

4.3.3 Synthesis: During synthesis we can either perform synthesis of the whole design or perform block level synthesis to garner the information on timing and area of the components.
4.3.4 Final Floor Planning: Floor planning stage performs the complete mapping of the components using the floor plan map created earlier.

The remaining part of the design flow consists of the following steps similar to those in a conventional design methodology.

4.3.5 Clock Tree Generation

4.3.6 Global Place and Route

4.3.7 Delay calculation and SDF back annotation

4.3.8 Final routing

4.3.9 Optimized hardware

4.3.10 Software Synthesis: Software FSM is implemented by mapping it into software structure along with the Real time operating system.

4.3.11 Software FSM: The reactive behaviour is synthesized into two stages.

(a) Implement & optimize the desired behaviour in a high level processor independent representation which is similar to the control or data flow graphs.

(b) Control/data flow graphs are translated to C code and compiled to implement and optimize on a microcontroller.

5 Advantages

The proposed flow offers the flexibility to perform specification capture based partitioning. The timing information can be matched more realistically compared to the traditional EDA flows. Reduction in power consumption could be pre-planned while carrying out the intermediate floor-planning. This methodology is technology independent and one can enter the design process at any given stage of the design flow. The flow can be used as a plug and play EDA tool which makes it a flexible, yet concrete and complete SoC design flow.

6 Results

A methodology which transforms UML models into a known System Level Design Language (SLDL) (in this case SpecC) has been presented and implemented successfully. Further, in the SoC flow, an intermediate floor-planning is presented which facilitates SoC designers to estimate the area of the design and to reduce the avoidable increase in power consumption of the chip prior to placement and routing thereby reducing the problem of timing closure which is crucial for present and future SoC design.

7 Conclusion

A new EDA flow for a SoC design methodology based on specification capture, block-clustering and bus-partitioning is presented. The challenges faced by today's designers especially in the production of SoC designs have been examined. Finally the advantages of using this flow for future and contemporary SoC designs have been emphasized.

References

[1] M. Keating and P. Bricaud, "Reuse Methodology Manual for System-on-Chip designs, 2nd Edition, Kluwer Academic Publishers, Norwell 1999.

[2] F. Balarin *et. al*, "Hardware-Software Co-Design of Embedded Systems, The POLIS approach," Kluwer Academic Publishers, 1997.

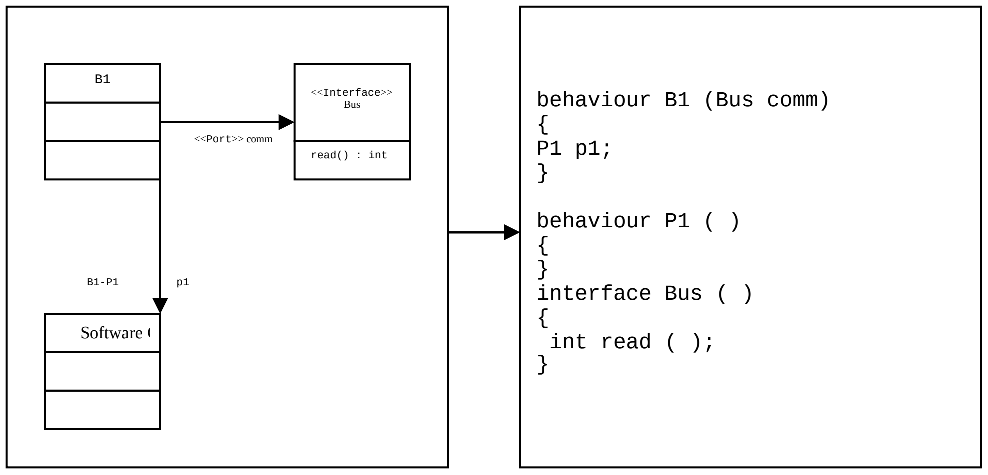
[3] D. Gajski, J.Zhu et al. "SpecC: Specification Language and Design Methodology", *Kluwer Academic Publishers*, 2000.

[4] Rainer Dömer, Daniel D. Gajski, Andreas Gerstlauer, "SpecC Methodology for High-Level Modeling," 9th EDP IEEE/DATC Electronic Design Processes Workshop 2002.

[5] Object Management Group, Omg unified modeling language specification version 1.3, June 1999.

[6] D. E. Lackey, "Applying Placement-based Synthesis for On-time System-on-a-Chip Design", *IEEE Custom Integrated Circuits Conference*, 2000, pp. 121-124.

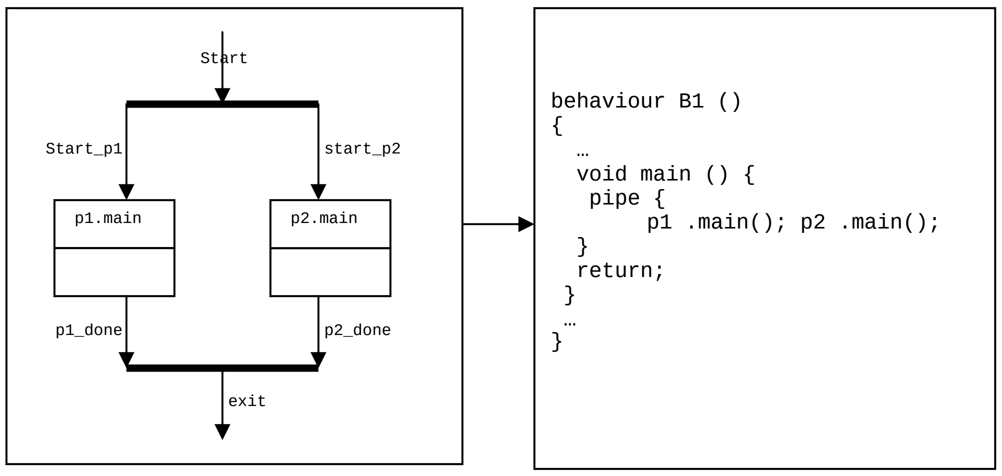
[7] Object Management Group, Omg-xml metadata interchange version 1.2, January 2002.



```
behaviour B1 (Bus comm)
{
P1 p1;
}

behaviour P1 ( )
{
}
interface Bus ( )
{
 int read ( );
}
```

Figure (1): Static View.



```
behaviour B1 ()
{
  …
  void main () {
   pipe {
        p1 .main(); p2 .main();
  }
  return;
 }
 …
}
```

Figure (2): State Machine View.

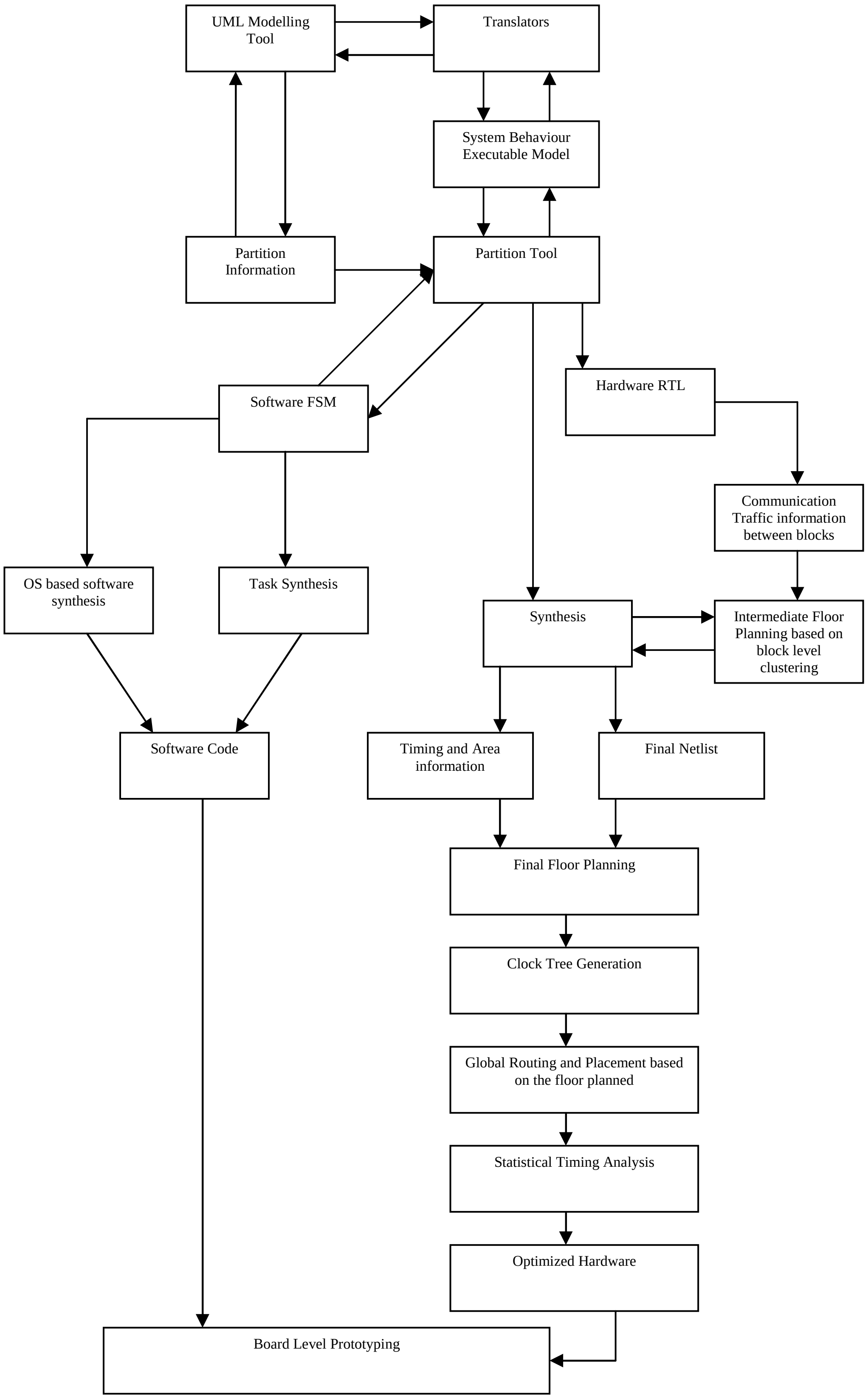

UML Modelling Tool
Translators
System Behaviour Executable Model
Partition Information
Partition Tool
Software FSM
Hardware RTL
Communication Traffic information between blocks
OS based software synthesis
Task Synthesis
Synthesis
Intermediate Floor Planning based on block level clustering
Software Code
Timing and Area information
Final Netlist
Final Floor Planning
Clock Tree Generation
Global Routing and Placement based on the floor planned
Statistical Timing Analysis
Optimized Hardware
Board Level Prototyping

Figure (3): The proposed SoC Design Flow.