DOWNLOAD PDF DESIGN ABSTRACTION AND CIRCUIT VALIDATION

Chapter 1 : Circuit design - Wikipedia

LAKSHMI NARAYAN COLLEGE OF TECHNOLOGY & SCIENCE. MEDC- VLSI Design - Design Abstraction and Circuit Validation Ayoush Johari Assistant Professor Department Of Electronics and Communication Engineering LNCTS.

The truth of a CTL formula is interpreted with respect to a state, by considering the tree of infinite computations starting from that state. For example, a formula AG f is true in a state s, if on all paths starting from s, formula f is true globally in each state along the path. Similarly EF f is true in a state s, if there exists some path starting from s, such that formula f is eventually true on some state on that path. The nesting of these modalities can express many correctness properties such as safety, liveness, precedence etc. Typically, model checking is used to check the truth of a formula with respect to the initial state of a given finite state design. In this case, since the number of control states in a CDFG design representation is typically small, a static analysis can be performed to identify irrelevant datapath operations. This typically provides better abstraction than a purely syntactic analysis on the next state logic of a standard RTL description. Next the focus is on the controller-datapath separation. Datapath variables that do not directly appear as atomic propositions in the CTL property are candidates for abstraction as pseudo-primary inputs, thereby resulting in a much smaller state space than that of the concrete design. At this stage, the user can also manually give hints about which parts of the design to include in the abstract model, and to carry out appropriate bit-width abstraction. The variables i, j, A, B, C, and F are primary inputs, and all other variables make up the datapath state. The light bordered boxes indicate the datapath operations executed in each control state, while the labels on the edges between control states identify the conditions under which those transitions take place. Note that while the number of control states is small, the total state space is actually large if the full datapath state is modeled. Cone-of-influence analysis is used to determine that state ST3 does not contain any relevant datapath operations. Next, since M is the only datapath variable referred to in the atomic proposition, M and its immediate dependency H are included as state variables. All other data variables are regarded as pseudo-primary inputs. The resulting abstract model is shown in FIG. In this section, a symbolic model checking-based algorithm for CTL properties is described, which is used to compute states of interest in the abstract model. The pseudo-code for this algorithm, called mcâ€”forâ€”sim model checking for simulation , is shown in FIG. Its inputs are an abstract model m, which has a smaller number of state variables but potentially more transitions than the concrete design d, and a CTL formula f in negation normal form, where all negations appear only at the atomic level. As before, if the original property is an A-type formula, all counter-examples are looked for. On the other hand, if the original property is an E-type formula, all witnesses are looked for. For the rest of this discussion, the assumption is that the goal lies in finding witnessesâ€”the same discussion holds, however, for finding counter-examples. The main idea is to use model checking over m to precompute a set of abstract states which are likely to be part of witnesses, and to use this set for guidance during simulation over d, in order to demonstrate a concrete witness. In particular, over-approximate sets of satisfying states are targeted during model checking, so that searching through an over-approximate set of witnesses during simulation can be performed. Note that model checking is performed over the abstract model m, while simulation is performed over the concrete design d. Since atomic level negations can be computed exactly, and all other CTL operators in a negation normal form are monotonic, an over-approximation for the overall formula can be computed by computing over-approximations for the individual subformulas. As shown in FIG. With each subformula, the algorithm associates a set of abstract states called upper, which corresponds to an over-approximate set of concrete states that satisfy the subformula. The standard symbolic model checking method is adequate for handling atomic propositions which are computed exactly and Boolean operators which simply propagate over-approximations in the sets associated with the subformulas. Since m may have many false paths with respect to d, standard model checking over m may result in an under-approximation over d. Therefore, upper is computed by considering the corresponding E-type operator, which is guaranteed to result in an over-approximation. However, this over-approximation is rather coarse. To mitigate this effect, a set of abstract states called negative is also computed, which corresponds to the intersection of set upper with

a set which is recursively computed for the negation of the A-type subformula. Note that, by induction, the latter set corresponds to an over-approximate set of concrete states that satisfy the negated subformula. The use of these sets is described later. Though not shown in the pseudo-code in FIG. As a result, at most two recursive calls are made for each subformula of the CTL propertyâ€”one for the subformula itself, and the other for its negation. Thus, its overall complexity is the same as that of standard symbolic model checking. Conclusive Proof Due to Model Checking It is possible that model checking on m itself provides a conclusive result for d in some cases. Pseudo-code for performing this check is shown in FIG. Recall that the set upper corresponds to an over-approximate set of concrete satisfying states. Therefore, if the initial state does not belong to this set, clearly the property is false. Now, assume that the initial state does belong to set upper. Recall also that for an A-type operator, we compute the set negative. If the initial state does not belong to set negative, then there does not exist any path in m starting from the initial state that shows negation of the property. Therefore, it is guaranteed that no such concrete path exists in d, i. In all other cases, the result from model checking is inconclusive. The purpose of precomputing negative sets for A-type subformulas is to avoid a proof by simulation where possible. Note that an abstract state s which belongs to upper, but not to negative, is a very desirable state to target as a witness for the A-type subformula. Again, this is because there does not exist any abstract path in m starting from s for the negation of the subformula, thereby guaranteeing that there is no such concrete path in d. Therefore, the proof of the A-type subformula is complete as soon as state s is reached during simulation, with no further obligation. On the other hand, if a state t belongs to negative also, the task during simulation is to check whether there is a concrete path starting from t which shows the counter-example for the A-type subformula. If such a counter-example is found, state t is not a true witness state for the A-type subformula, and can be eliminated from further consideration. This allows the use of standard symbolic model checking techniques to compute sets upper over-approximations for most subformulas. Furthermore, this computation of the negative sets for the A-type subformulas is similar to computing under-approximations, in the sense that the complement of an over-approximation for the negated subformula can be seen as an under-approximation for the subformula itself. Ultimately, these sets are used to provide guidance during simulation for designs where it may not be possible to perform any symbolic analysis at all. Instead, coarser approximations are usedâ€”using the E-type operators in place of the A-type operators. Indeed, it would be appropriate to use any known technique for obtaining the tightest possible approximations. The additional contribution is also in showing how these sets can be used to demonstrate concrete witnesses in the context of simulation. For atomic propositions and Boolean operators, this is trivially true since there are no paths to consider. In the context of the mcâ€”forâ€”sim algorithm, satisfying sets for subformulas and their negations are over-approximated as sets upper and negative. Since the latter sets are computed bottom-up, the former sets top-down are used as care-sets for constraining solutions. At the topmost level, the set of reachable states is used as the care-set. Note that the special handling of EX-type subformulas requires an extra image computation in order to exploit the care-set. In general, such use of care-sets may result in substantial pruning. Another kind of pruning occurs due to the A-type subformulas. Recall that for a state s that belongs to set upper, but not to set negative, the proof of the A-type subformula holds due to model checking itself. Therefore, there is no need to extend a witness from this state during simulation. Instead, it is necessary to focus on states that belong to set negative, in order to search for a concrete counter-example during simulation. Therefore, recursive calls are made to mark witnesses for the negated subformula, which are then associated as the set negâ€”witness. Returning to the example, for the abstract model of FIG. Iterative Refinement of the Abstract Model The amount of detail that can be allowed in the abstract model is a function of the level of complexity that the model checker or constraint solver can handle in its analysis. However, once pruning is done, the model can be refined, and it may be possible to perform the analysis again. Recall that the initial abstract model was obtained by abstracting away many of the datapath variables as pseudo-primary inputs. Refinement is performed by selectively bringing back some of these datapath variables into the state space of the abstract model. Note that pruning after analysis reduces the size of the model, while refinement increases it. Getting back to the example, suppose it is decided to add datapath variables D and K as state. This results in the model shown in FIG. At this point, it may not be desired to add any more datapath state to the model,

leading to the final Witness Graph as shown in FIG. Furthermore, the associated sets are herein used for marking states in order to prune the abstract model before attempting further refinement. Existing techniques do not perform such model pruning. Finally, since the target herein includes bigger designs than can be handled by any kind of symbolic traversal, the goal of this iterative refinement process is not only to obtain a conclusive result by model checking. Rather, it is to reduce the gap between the abstraction levels of the final Witness Graph and the concrete design to be simulated. Witness Graph as a Coverage Metric Apart from using a Witness Graph for generating a testbench, it can also be used as a coverage metric for evaluating the effectiveness of a given set of simulation vectors. Note that a high coverage still does not guarantee correctness in the designâ€”it only provides a metric to assess the quality of simulation. In contrast, this metric is obtained by analysis of the design with respect to the given property. These techniques can potentially be used to extend this per-property analysis to coverage of overall correctness. The testbench is patterned upon this algorithm, and can be automatically generated for a given property. The pseudo-code for witnessâ€”sim algorithm, as shown in FIGS. The handling of atomic propositions and Boolean operators is fairly obvious. The E-type temporal operators are also handled by using their standard characterizations in terms of image and fixpoint operations, where abs s refers to the abstract state corresponding to a concrete state s. The handling of the A-type operators reflects the above remarks thatâ€”if abs s does not belong to set negative, the proof of the A-type subformula is complete due to model checking itself, and the return can be SUCCESS. Otherwise, a counter-example for f is looked for starting from s. If such a counter-example is found, i. In principle, the witnessâ€”sim algorithm will find a concrete witness if it exists, because the witness sets computed earlier are based on over-approximations of concrete satisfying sets. However, in practice, it is impossible to search through all possible concrete states in the for each loops of the pseudo-code. Such search would be typically limited by available resources, such as space and time. Therefore, the next task is to prioritize the search, in order to provide increased reliability with increased resources.

Chapter 2 : CiteSeerX â€” Design Verification for Sequential Systems at Various Abstraction Levels

Granular validation logic and rules can be abstracted away from the service contract, thereby decreasing constraint granularity and increasing the contract's potential longevity. Application Abstracted validation logic and rules need to be moved to the underlying service logic, a different service, a service agent, or elsewhere.

The method employs automatic abstractions for the test model which reduce its complexity while preserving the class of errors that can be detected by a transition tour. A method for design validation comprising generating a state-based test model of the design, abstracting said test model by retiming and latch removal; and applying validation technique on the abstracted test model. First, the number of internal non-peripheral latches in a design is minimized via retiming using a method of Maximal Peripheral Retiming MPR. According to the MPR method, internal latches are retimed to the periphery of the circuit. Subsequently, all latches that can be retimed to the periphery are automatically abstracted in the test model. The validation technique may comprise of model checking, invariant checking or guided simulation using test sequences generated from the abstracted test model. Field of the Invention [] This invention relates to a method for abstraction of test models for validation of industrial designs using guided simulation, model checking and invariant checking. Specifically, this invention relates to a method for automatic abstractions for the test model which reduce its complexity while preserving the class of errors that can be detected by a reachability test. The invention comprises a method of Maximal Peripheral Retiming MPR , where the number of internal non-peripheral latches is minimized via retiming and some latches at the periphery are removed. The invention is embodied in a method for generating test models that has been shown to be practical by providing a detailed case study, as well as experimental results of applying this abstraction on a set of benchmark circuits. Background of the Invention [] Large industrial designs are commonly validated using test sequences. In a typical methodology, a test model is derived from the design, and test sequences are generated from it by using formal verification techniques. These test sequences, which satisfy certain coverage criteria, are then used for functional simulation. A popular conventional method comprises generating test sequences by performing a transition tour on the state space of the test model, thereby guaranteeing complete transition coverage for it. In practice, 5 such a method has been shown to be effective in uncovering errors that are otherwise difficult to find. A primary conventional methodology for design validation is simulation of functional models of design. There are at least two major problems with using such an approach. The first is for any reasonable coverage of possible behaviors of the design, a substantial amount of computational resources are required. The second is that such an approach lacks any formal measure of this coverage. However, exhaustive simulation is beyond practical limits. In response to this, formal verification is emerging as a no-test-vectors validation methodology. In such a formal verification method, a formal proof of correctness covers the entire space of tests. But, while recent advances in formal verification are promising, they tend to have limited applicability. The use of implicit state traversal techniques has significantly extended these limits. For more details, see J. Madre, Verification of sequential machines using Boolean functional vectors, L. Nonetheless, there continues to be a significant gap between the capabilities of conventional formal verification techniques and the practical requirements imposed by them. These hybrid techniques combine the relative strengths of formal verification on one hand and simulation on the other. A popular hybrid methodology is based on using formal techniques to ensure some form of simulation coverage. It comprises first deriving a formal model of the design, typically a finite state model FSM , hereafter called the test model. Then techniques similar to those used in formal verification e. Finally, this test set is used as stimuli for functional simulation of the entire design, which can be used for either specification validation, or for comparison of an implementation against a given specification. A schematic for the such an application is shown in FIG. These models could includeâ€”for example, coverage of each state, or coverage of each transition, or coverage of each pair-arc. For more details, see Y. Abraham, Automatic extraction of the control flow machine and application to evaluating coverage of verification vectors, [] Proceedings of the IEEE International Conference on Computer Design, pages , October ; H. Abraham, Automatic extraction of the

control flow machine and application to evaluating coverage of verification vectors, Proceedings of the IEEE International Conference on Computer Design, pages , October Several conventional variations on aspects of the hybrid theme have also been proposed. In one such variation, the coverage criteria used in the hybrid theme is used to evaluate the quality of a given test set or to drive the search for additional test sequences that fill in the gaps. Instead of building a separate test model, such measures are also used directly during simulation to provide partial coverage of the state space for checking invariants. For details, see J. Other related techniques use property-specific models, instruction templates, and HDL descriptions for generation of the test sets. Ur, A methodology for processor implementation verification, [] Proceedings of the Conf. While most prior art methodologies based on simulation coverage have been shown useful for detection of design errors, there is no teaching on how coverage measures on the test model translate to coverage of design errors. Furthermore, the derivation of the test model remains largely ad-hoc. Importantly, very few formal guidelines have been provided for such a derivation. For example, for processor validation, the datapath in the design is typically abstracted out and only the controller portion is retained in the test model. For details, see D. Wolsfthal, Coverage-directed test generation using symbolic techniques, In [] Proceedings of the Conf. Springer-Verlag, New York, June There has been some work on state space abstraction based on equivalence of output control signals as seen by the datapath. For details, see R. It is important to note that the success of abstraction in the area of formal verification is predicated on a clear formulation of the correctness criteria. Once such criteria exist, it is possible to reason about their preservation by use of appropriate abstractions. Despite the above advances, prior art has failed to provide a method for deriving and abstracting of test models suitable for validation of large industrial designs using guided simulation. Correctness of the abstraction implies completeness of the transition tour error coverage of the abstract test model with respect to the original design. Such an abstraction is correct if it preserves coverage of those errors that can be captured by a transition tour. In other words, the test sequences generated from a transition tour on the abstract test model should cover the complete set of those design errors that are covered by test sequences generated from any transition tour on the original design. This notion can be potentially extended to other modes and applications as well. An error is detected in model checking or invariant checking by demonstrating the reachability of a particular state which causes incorrect behavior. These states may cause incorrect behavior by virtue of their presence in undesirable loops or because specific logic expressions to have incorrect values. A property of the retiming based abstraction technique is that reachability of states is preserved. Consequently, an error that was detected as a result of performing model checking on the original model will also be detected by model checking on the abstract model. An invariant is a property a logic expression or a model complex relationship between variables that should hold at all times. Checking an invariant also involves ensuring that states that can cause the property to be violated are never reached. The method of the present invention also works for reachability in general. In general terms, an MPR is a retiming where as many state elements as possible are moved to the periphery of the circuit. Consequently, there are as few internal state elements as possible. Intuitively, a subcircuit consisting only of peripheral latches can be safely removed if it contains no errors, because it does not affect the detection of errors in rest of the circuit. Under this condition, it is shown that the abstraction is correct, i. The practical importance of this abstraction is in the significant reduction in the number of latches, and thereby the complexity of validation. Empirical evidence of its practical efficacy will be demonstrated using a set of benchmark circuits. This is followed by the formulation of the general MPR problem and its solution, as defined by an embodiment of the present invention. Next, a detailed case study of a DLX processor, followed by experimental results of applying MPR to a set of benchmark circuits will be presented. Retiming for Abstraction [] Retiming is the process of repositioning latches across the combinational logic in a sequential circuit so as to minimize the clock period, the number of latches, or to meet a given clock period while minimizing the number of latches. Polynomial time algorithms for these were introduced by Leiserson and Saxe. For details on these algorithms, see Charles E. Leiserson and James B. Saxe, Retiming Synchronous Circuitry, [] Algorithmica, 6 1: Definitions [] For the purpose of describing the preferred embodiments of the present invention, the following definitions will be used: Intuitively, input peripheral latches are located at the input periphery of a circuit, such that no fanin signal to these latches is

directly used by rest of the logic. In other words, the application of primary inputs is merely delayed by these latches. Output peripheral latches are located at the output periphery of a circuit, such that no fanout signal from these latches is directly used by rest of the logic. Again, these latches merely delay the availability of primary outputs. Clearly, these latches determine the reachability of the state space associated with a circuit. As an example, in the circuit shown in FIG. Design Errors and Transition Tours [] A design implementation is considered to have an error with respect to its specification if the implementation produces an incorrect output for some input sequence. In general, a transition tour cannot capture all errors, since it covers all single transitions only, and not all transition sequences. This illustrates the basic limitation of using transition toursâ€”an error may get detected several transitions after it is excited, and only along a specific path in the state transition graph. If this path is not selected in the transition tour, the error will not be covered. Furthermore, depending on what particular paths are selected, different transition tours may cover different sets of errors. Correctness of Abstraction [] It should be noted that the present invention is not restricted to a particular type of transition tours. In order to not tie the analysis to a particular choice of a transition tour, for the rest of this specification, the focus is on those errors that can be covered by any transition tour of a given design implementation. Thus, it can be shown that reachability of states is preserved by the abstraction used in the present invention. The analysis is also not restricted to systems with special properties that allow all errors to be covered by a transition tour. Toward formalizing a validation methodology using simulation coverage. The following definitions are uses as criteria for correctness of an abstraction in this context: The formal statement of the problem, and an outline for its solution are described below. Once MPR has been performed on the given design, the following two kinds of abstraction are considered for obtaining the test model:

Chapter 3 : .net - C# Custom Object Validation Design - Stack Overflow

Design Validation of RTL Circuits using Binary Particle Swarm Optimization and Symbolic Execution Prateek Puri (ABSTRACT) Over the last two decades, chip design has been conducted at the register transfer (RT) Level.

Kulwant Nagi Roll No. The number of applications of integrated circuits in high-performance computing, telecommunications, and consumer electronics has been rising steadily, and at a very fast pace. Typically, the required computational power or, in other words, the intelligence of these applications is the driving force for the fast development of this field. The current leading- edge technologies such as low bit-rate video and cellular communications already provide the end- users a certain amount of processing power and portability. This trend is expected to continue, with very important implications on VLSI and systems design. One of the most important characteristics of information services is their increasing need for very high processing power and bandwidth in order to handle real-time video, for example. It starts with a given set of requirements. Initial design is developed and tested against the requirements. When requirements are not met, the design has to be improved. If such improvement is either not possible or too costly, then the revision of requirements and its impact analysis must be considered. The Y-chart shown in Fig illustrates a design flow for most logic chips, using design activities on three different axes domains which resemble the letter Y. The Y-chart consists of three major domains, namely: The design flow starts from the algorithm that describes the behavior of the target chip. The corresponding architecture of the processor is first defined. It is mapped onto the chip surface by floorplanning. The next design evolution in the behavioral domain defines finite state machines FSMs which are structurally implemented with functional modules such as registers and arithmetic logic units ALUs. These modules are then geometrically placed onto the chip surface using CAD tools for automatic module placement followed by routing, with a goal of minimizing the interconnects area and signal delays. The third evolution starts with a behavioral module description. Individual modules are then implemented with leaf cells. The last evolution involves a detailed Boolean description of leaf cells followed by a transistor level implementation of leaf cells and mask generation. In standard-cell based design, leaf cells are already pre-designed and stored in a library for logic design use. This Figure provides a more simplified view of the VLSI design flow, taking into account the various representations, or abstractions of design - behavioral, logic, circuit and mask layout. Note that the verification of design plays a very important role in every step during this process. The failure to properly verify a design in its early phases typically causes significant and expensive re- design at a later stage, which ultimately increases the time-to-market. Although the design process has been described in linear fashion for simplicity, in reality there are many iterations back and forth, especially between any two neighboring steps, and occasionally even remotely separated pairs. Although top-down design flow provides an excellent design process control, in reality, there is no truly unidirectional top-down design flow. Both top-down and bottom- up approaches have to be combined. For instance, if a chip designer defined an architecture without close estimation of the corresponding chip area, then it is very likely that the resulting chip layout exceeds the area limit of the available technology. In such a case, in order to fit the architecture into the allowable chip area, some functions may have to be removed and the design process must be repeated. Such changes may require significant modification of the original requirements. Thus, it is very important to feed forward low-level information to higher levels bottom up as early as possible. In the following, we will examine design methodologies and structured approaches which have been developed over the years to deal with both complex hardware and software projects. Regardless of the actual size of the project, the basic principles of structured design will improve the prospects of success. Some of the classical techniques for reducing the complexity of IC design are: Hierarchy, regularity, modularity and locality. Design automation is the order of the day. With the rapid technological developments in the last two decades, the status of VLSI technology is characterized by the following [Wai-kai, Gopalan]: The above developments have resulted in a proliferation of approaches to VLSI design. We briefly describe the procedure of automated design flow [Rabaey, Smith MJ]. An abstraction based model is the basis of the automated design. Abstraction Model The model divides the whole design cycle into various domains. With such an abstraction through a

division process the design is carried out in different layers. The designer at one layer can function without bothering about the layers above or below. The thick horizontal lines separating the layers in the figure signify the compartmentalization. As an example, let us consider design at the gate level. The circuit to be designed would be described in terms of truth tables and state tables. With these as available inputs, he has to express them as Boolean logic equations and realize them in terms of gates and flip-flops. In turn, these form the inputs to the layer immediately below. Compartmentalization of the approach to design in the manner described here is the essence of abstraction; it is the basis for development and use of CAD tools in VLSI design at various levels. The design methods at different levels use the respective aids such as Boolean equations, truth tables, state transition table, etc. But the aids play only a small role in the process. To complete a design, one may have to switch from one tool to another, raising the issues of tool compatibility and learning new environments. The first step in the process is to expand the idea in terms of behavior of the target circuit. Through stages of programming, the same is fully developed into a design description â€" in terms of well defined standard constructs and conventions. Design Description The design is carried out in stages. The process of transforming the idea into a detailed circuit description in terms of the elementary circuit components constitutes design description. The final circuit of such an IC can have up to a billion such components; it is arrived at in a step-by-step manner. The first step in evolving the design description is to describe the circuit in terms of its behavior. The description looks like a program in a high level language like C. Once the behavioral level design description is ready, it is tested extensively with the help of a simulation tool; it checks and confirms that all the expected functions are carried out satisfactorily. If necessary, this behavioral level routine is edited, modified, and rerun â€" all done manually. Finally, one has a design for the expected system â€" described at the behavioral level. The behavioral design forms the input to the synthesis tools, for circuit synthesis. The behavioral constructs not supported by the synthesis tools are replaced by data flow and gate level constructs. To surmise, the designer has to develop synthesizable codes for his design. Optimization The circuit at the gate level â€" in terms of the gates and flip-flops â€" can be redundant in nature. The same can be minimized with the help of minimization tools. The step is not shown separately in the figure. The minimized logical design is converted to a circuit in terms of the switch level cells from standard libraries provided by the foundries. The cell based design generated by the tool is the last step in the logical design process; it forms the input to the first level of physical design. Simulation The design descriptions are tested for their functionality at every level â€" behavioral, data flow, and gate. One has to check here whether all the functions are carried out as expected and rectify them. All such activities are carried out by the simulation tool. The tool also has an editor to carry out any corrections to the source code. Simulation involves testing the design for all its functions, functional sequences, timing constraints, and specifications. Synthesis With the availability of design at the gate switch level, the logical design is complete. The corresponding circuit hardware realization is carried out by a synthesis tool. Two common approaches are as follows: The gate level design description is the starting point for the synthesis here. The FPGA vendors provide an interface to the synthesis tool. Through the interface the gate level design is realized as a final circuit. With many synthesis tools, one can directly use the design description at the data flow level itself to realize the final circuit through an FPGA. The FPGA route is attractive for limited volume production or a fast development cycle. A typical ASIC vendor will have his own library of basic components like elementary gates and flip-flops. Eventually the circuit is to be realized by selecting such components and interconnecting them conforming to the required design. This constitutes the physical design. Being an elaborate and costly process, a physical design may call for an intermediate functional verification through the FPGA route. The circuit realized through the FPGA is tested as a prototype. The step-by-step activities in the process are described briefly as follows: The design is partitioned into convenient compartments or functional blocks. Often it would have been done at an earlier stage itself and the software design prepared in terms of such blocks. Interconnection of the blocks is part of the partition process. The positions of the partitioned blocks are planned and the blocks are arranged accordingly. The procedure is analogous to the planning and arrangement of domestic furniture in a residence. Partitioning and floor planning may have to be carried out and refined iteratively to yield best results. The components placed as described above are to be interconnected to the rest

of the block: It is done with each of the blocks by suitably routing the interconnects. Once the routing is complete, the physical design cam is taken as complete. The final mask for the design can be made at this stage and the ASIC manufactured in the foundry.

Chapter 4 : USA1 - Method for design validation using retiming - Google Patents

VLSI Design Flow System Specification Functional/Architecture Design Logic Design/Synthesis (Translation, Mapping and Placement & Routing) Circuit Design Physical/Layout Design Fabrication Packaging The design process, at various levels, is usually evolutionary in nature.

Product design specification The process of circuit design begins with the specification , which states the functionality that the finished design must provide, but does not indicate how it is to be achieved. The specification can and normally does also set some of the physical parameters that the design must meet, such as size, weight, moisture resistance , temperature range, thermal output, vibration tolerance and acceleration tolerance. This can involve tightening specifications that the customer has supplied, and adding tests that the circuit must pass in order to be accepted. These additional specifications will often be used in the verification of a design. It can be defined in terms of its results; "at one extreme is a circuit with more functionality than necessary, and at the other is a circuit having an incorrect functionality". A block diagram of 4-bit ALU The design process involves moving from the specification at the start, to a plan that contains all the information needed to be physically constructed at the end, this normally happens by passing through a number of stages, although in very simple circuit it may be done in a single step. This approach allows the possibly very complicated task to be broken into smaller tasks which may either by tackled in sequence or divided amongst members of a design team. Each block is then considered in more detail, still at an abstract stage, but with a lot more focus on the details of the electrical functions to be provided. At this or later stages it is common to require a large amount of research or mathematical modeling into what is and is not feasible to achieve. At this point it is also common to start considering both how to demonstrate that the design does meet the specifications, and how it is to be tested which can include self diagnostic tools. This stage is typically extremely time consuming because of the vast array of choices available. A practical constraint on the design at this stage is that of standardization, while a certain value of component may be calculated for use in some location in a circuit, if that value cannot be purchased from a supplier, then the problem has still not been solved. One area of rapid technology development is in the field of nanoelectronic circuit design. Balance is the key concept here; just as many delays and pitfalls can come from ill-considered cost cutting as with cost overruns. Good accounting tools and a design culture that fosters their use is imperative for a successful project. Verification and testing[edit] Once a circuit has been designed, it must be both verified and tested. Verification is the process of going through each stage of a design and ensuring that it will do what the specification requires it to do. This is frequently a highly mathematical process and can involve large-scale computer simulations of the design. In any complicated design it is very likely that problems will be found at this stage and may involve a large amount of the design work be redone in order to fix them. Testing is the real-world counterpart to verification, testing involves physically building at least a prototype of the design and then in combination with the test procedures in the specification or added to it checking the circuit really does do what it was designed to. Prototyping[edit] Prototyping is a means of exploring ideas before an investment is made in them. Depending on the scope of the prototype and the level of detail required, prototypes can be built at any time during the project. Later in the cycle packaging mock-ups are used to explore appearance and usability, and occasionally a circuit will need to be modified to take these factors into account. Results[edit] As circuit design is the process of working out the physical form that an electronic circuit will take, the result of the circuit design process is the instructions on how to construct the physical electronic circuit. This will normally take the form of blueprints describing the size, shape, connectors, etc. Documentation[edit] Any commercial design will normally also include an element of documentation, the precise nature of this documentation will vary according to the size and complexity of the circuit as well as the country in which it is to be used. As a bare minimum the documentation will normally include at least the specification and testing procedures for the design and a statement of compliance with current regulations. In the EU this last item will normally take the form of a CE Declaration listing the European directives complied with and naming an individual responsible for compliance.

DOWNLOAD PDF DESIGN ABSTRACTION AND CIRCUIT VALIDATION

Chapter 5 : SOA Patterns | Design Patterns | Validation Abstraction

abstraction procedure consists of 1) partitioning the design into a module call graph -a collection of modules as a list, 2) classifying the variables as control, data, or mixed 3).

Chapter 6 : Design Validation of RTL Circuits using Binary Particle Swarm Optimization and Symbolic Exec

A Design and Validation System for Asynchronous Circuits Peter Vanbekbergen, Albert Wang and Kurt Keutzer Synopsys, Inc. Mountain View, CA,

Chapter 7 : Electronic system-level design and verification - Wikipedia

Combining Simulation and Formal Verification for Integrated Circuit Design Validation Lun Li, Stephen A. Szygenda, Mitchell A. Thornton Dept. of Computer Science and Engineering.

Chapter 8 : VLSI DESIGN FLOW | kulwant nagi - blog.quintoapp.com

The hybridized method performs loop abstraction and is able to traverse narrow design paths without performing costly circuit analysis or explicit loop unrolling. Also structural and functional unreachable branches are identified during the process of test generation.