Addressing Verification Bottlenecks of Fully Synthesized Processor Cores using Equivalence Checkers

Subash Chandar G (g-chandar1@ti.com), Vaideeswaran S (vaidee@ti.com)
DSP Design, Texas Instruments India

Abstract

Formal verification plays an important role in the verification of complex processors. In this paper, we discuss the usage and impact of equivalence checking in the verification of TI's TMS320C27X DSP core. During various phases of the design, we need to ensure the correctness of the design, a significant part of which could be best done with an equivalence checker. (For example, verifying the functionality of the netlist after CTS insertion with the one before CTS insertion). The capabilities and limitations of the commercial equivalence checkers are studied and a set of guidelines for their effective usage during different phases of the design is proposed. Also, a set of RTL coding guidelines to make the design equivalence checker friendly is detailed. Further, we discuss constrained mode equivalence checking which could be used if the implementation design is a super set of reference design. The verification cycle time reduction and the salient features of an automated methodology that was developed specifically for our DSP core are described.

1 INTRODUCTION

As the integration capability of deep submicron technology goes up, the trend is to include more and more functionality in a single chip. This increases the complexity of the design by many folds. The increase in complexity increases verification time significantly. A traditional verification methodology would be based on simulations, which is depicted in figure 1.1. In this method, simulation is used at each stage to verify the correctness of the design. Due to this, simulation forms a significant portion of the design cycle. This demands a shift in verification strategy towards reducing the simulation time to enable quick time-to-market. Formal Verification has the potential to reduce the verification time.

Formal Verification (FV) is a mathematical way of proving the functional correctness of a design. FV comprises of (a) Equivalence Checking (EC) (b) Model Checking (c) Theorem proving. We used Equivalence Checking in the verification of TMS320C27X (C27X) core, which is dealt in great detail in this paper.

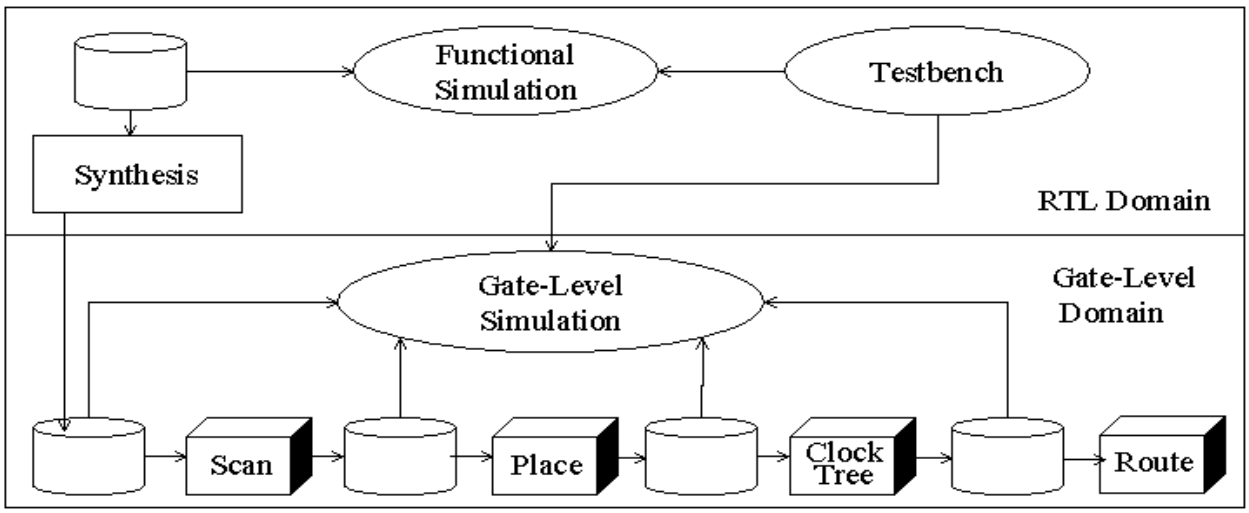

Fig 1.1: Traditional Verification flow

Equivalence Checking is a technique where the functional equivalence between two abstracts or views of a design is proved. EC therefore needs a reference designs against which the design to be verified is checked for equivalence.

We organise this paper in the following way, section 2 discusses a verification methodology that uses EC. The EC flows developed for verification of our designs and the applications of EC as applied across and within our designs are discussed in section 3. Section 4 discusses a set of guidelines for effective use of EC & we conclude with section 5.

2 VERIFICATION METHODOLOGY– HYBRID SOLUTION

Equivalence checkers find unparalleled advantages in the verification of different abstracts of the same design. In a synthesis based design it has been observed that a two pass VHDL coding is a preferred coding strategy - the first pass to get the functional correctness with a reasonable speed and a second pass to modify the RTL model to tackle the critical paths. Most often the processors are re-spinned just for higher speed where RTL modifications are primarily for better timing with no change in functionality. Also, to get the best out a new technology, many devices go for a re-spin where the current design is migrated to a faster technology. Equivalence checkers are the best fit in these scenarios.

A simulation based verification methodology described in Fig1.1 is time consuming. This is because each view of the design is verified using simulation, which is costly. The time taken for RTL simulations might be tolerable but netlist simulation times are prohibitively large. Verifying several abstractions (*which is common in a synthesis-based methodology*) would result in large amount of simulation time thereby impacting the design cycle time. So complex designs are forced to look for an alternate solution with acceptable verification time. A Verification methodology that uses equivalence checkers is the present-day solution.

Use of EC, not only cuts down the verification time, but also ensures that no new bug gets introduced across various abstracts. Simulation based verification does not guarantee the same, if the bug that gets introduced is transparent to the existing test vectors.

Equivalence Checking cannot replace simulation altogether because it needs a reference model to start with. An efficient solution could be a hybrid solution that uses simulation to verify the reference model & Equivalence Checkers to verify various views. This hybrid methodology is shown in figure 2.1. Equivalence Checkers (functional verification) along with Static Timing Analysis (timing check) is a complete solution that can replace timing simulations.

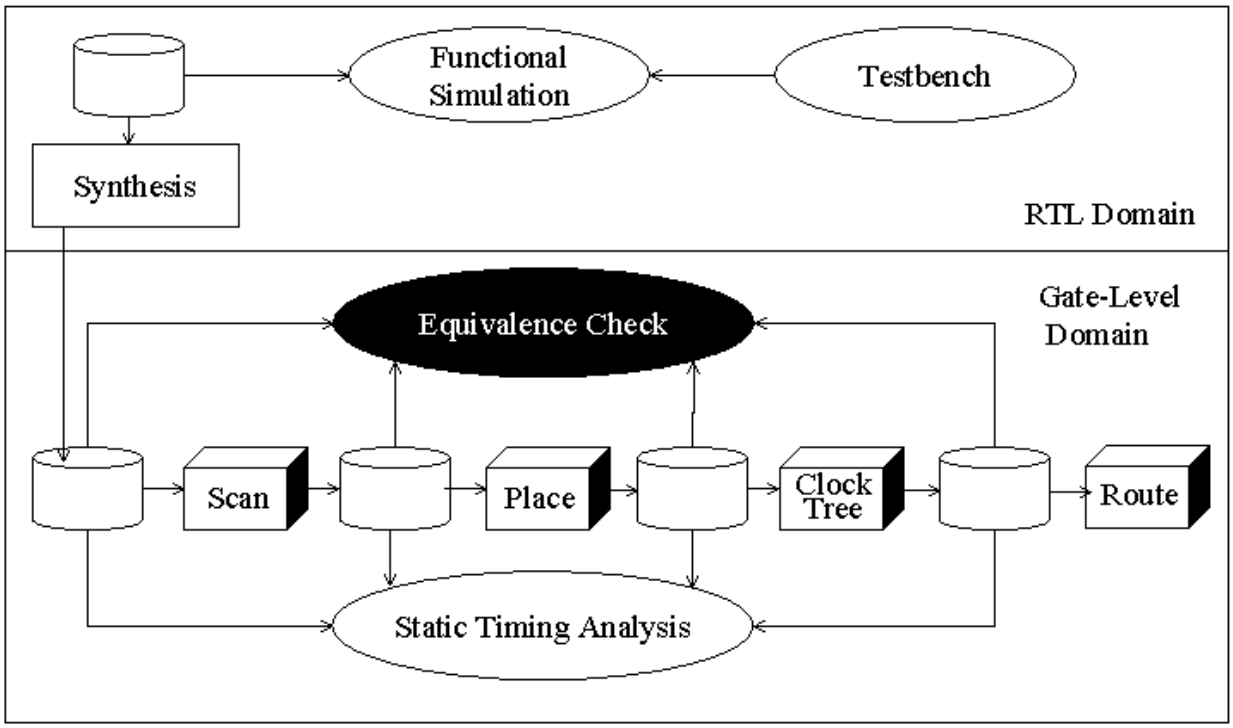

Fig 2.1: Verification Methodology based on a Hybrid solution

The commonly used commercial Equivalence Checking tools are Design VERIFYer and Formality. Their merits and de-merits are discussed in the following section.

3 EC FLOW

Efficient usage of present-day Equivalence Checking tools needs quite a bit of designer's intervention, making complete automation a challenge. This is because tools vary in their strengths in verifying different designs, and appropriate tool to be used can be best judged only with the knowledge of the internals of the design. In addition, providing correct options and guidance form key to the efficient equivalence checking. Actual challenge is to provide right options and guidance, most of which are design specific.

Though coming up with generic automated efficient flow is difficult, it is possible to optimize different phases with the pre-knowledge of the abstracts on which EC is performed. E.g. CTS inserted netlist can be ECed with pre-CTS netlist efficiently with predefined set of mapping information for clock network.

The study of EC tools and their application to our designs are discussed in the following sub-sections 3.1 and 3.2.

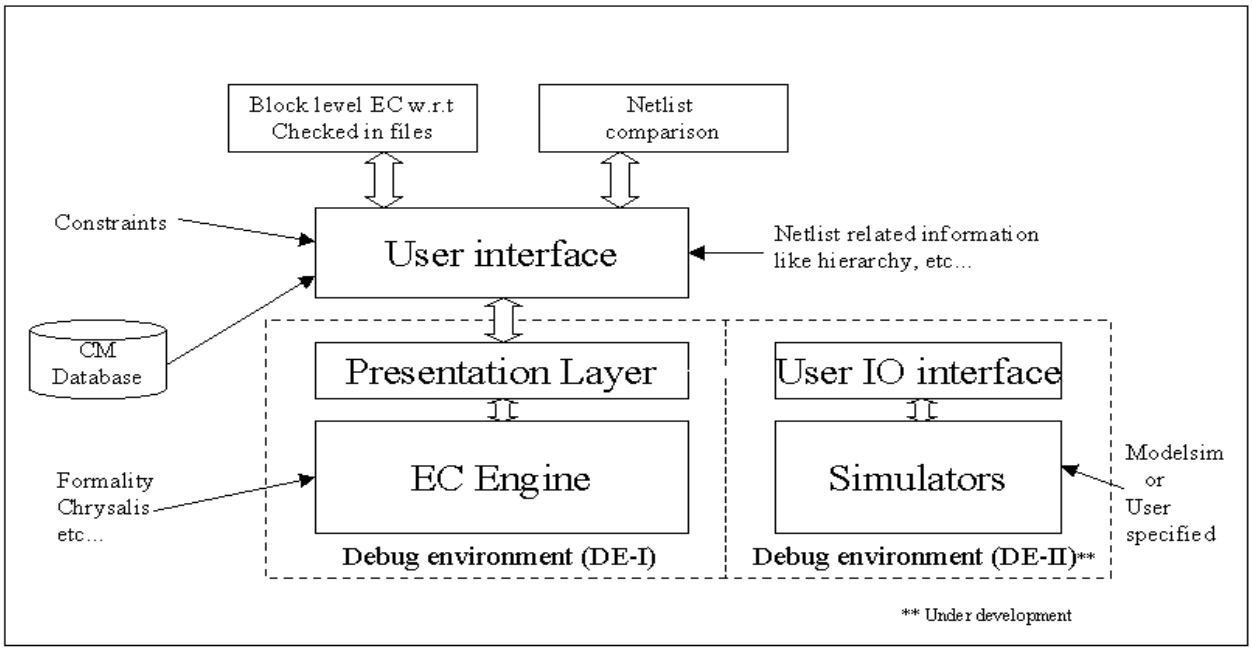

Fig 3.1 EC flow

3.1 Ground work

A complete knowledge of the capabilities of EC tools and their applicability to the design is key to decide on the EC engine for EC flow. The conclusion of our comparative study of Design VERIFYer (ver2.3) and Formality (1999.05-FM1.0) is to go for a combined solution. The relative merits and de-merits of these tools as applied to our designs are listed below.

- ✓ Formality - Merits
  - ⇒ Capacity - ability to handle big designs with acceptable run times
  - ⇒ Fast and efficient in verifying math intensive logic
  - ⇒ Good and extremely fast in verifying two gate level netlist
  - ⇒ Accepts Synopsys db - avoids the language specific parsing
  - ⇒ HDL interpreter is good - strength of Synopsys.
  - ⇒ Easy to use
  - ⇒ Run times in general are better compared to Design VERIFYer
- × Formality - De-merits
  - ⇒ Inefficient/Incapable of understanding signal properties. e.g. applying constraints like one-hot encoding to some inputs
  - ⇒ Inefficient with mapping specified - We have observed repeated crashing of the tool while the redundant ports were specified to be equivalent.

- ✓ Design VERIFYer – Merits
  - ⇒ Efficient in understanding the signal properties
  - ⇒ Efficient in understanding the constraints, like redundant ports, rule based mapping, etc..

⇒ Decent debug capabilities
⇒ Easy to use.

× Design VERIFYer - De-merits
⇒ Bad VHDL interpreter. One of our blocks was never read into Design VERIFYer in the original form. However formality had no problems.
⇒ Not so efficient in verifying math intensive blocks
⇒ Run times are in general more compared to formality.

Based on the experiences with the EC tools we believe that no one tool in the market really serves all the designer's needs. A combination of Design VERIFYer and Formality seems to an acceptable solution for a designer. The EC flow therefore supports both the engines with a simplified user interface. The EC flow is shown in figure 3.1.

EC engines supported are Formality and Design VERIFYer. Presentation layer and EC engine form the basic Debug environment (DE-I). Advanced Debug environment (DE-II) which is under development has the capability to invoke designer's favorite simulator with the testbench generated by presentation layer. This flow is closely integrated with CM databases to provide the capability of verifying designs from different releases.

3.2 Application of EC

3.2.1 EC usage in C27X

We have a complete proven EC solutions in the verification of C27X cores, this is depicted in figure 3.2.
Our EC experiences across and within various C27X cores is described below:

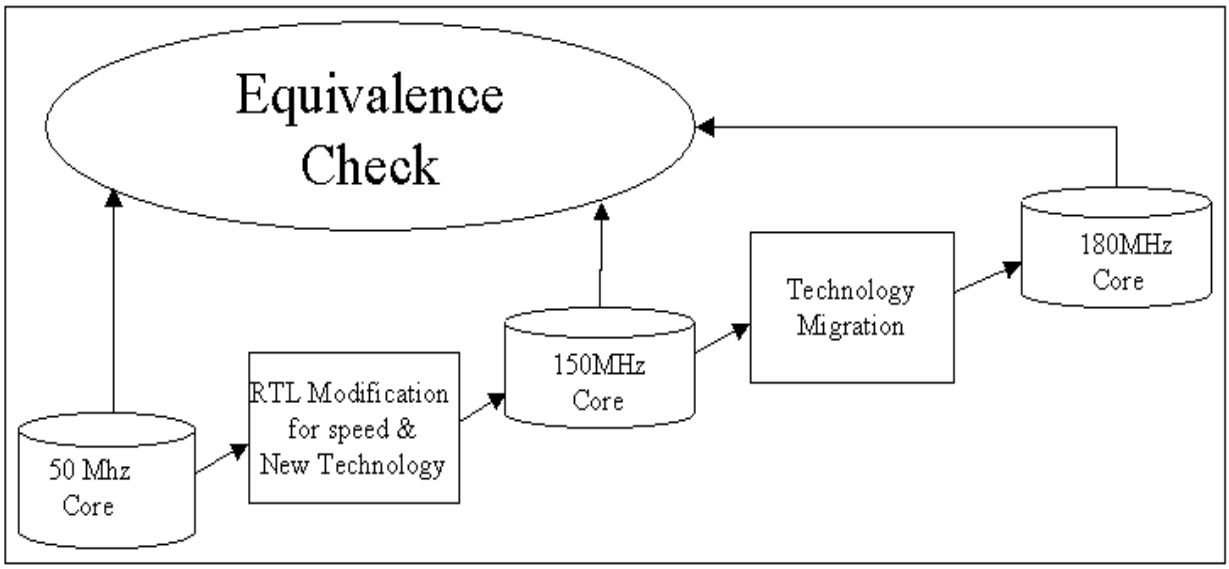

Fig 3.2 EC Usage across C27X cores

The EC flow comprises of User Interface, Presentation Layer, EC Engines, and Debug Environments. User Interface hides the tool specific details (like different format for mapping rules), reads the input constraints and passes it to presentation layer. Presentation layer has the intelligence of our design related information (exact hierarchy, default mapping for CTS, scan chain related information) which are appropriately applied. Presentation layer also decides on appropriate EC engine based on the knowledge of our design and tools capabilities. E.g. for a gate level validation, start with formality to get the benefit of the fast runtime. In case of tool failure in verifying the design, the flow switches over to Design VERIFYer and performs an equivalence check. The presentation layer develops the testbench based on failing patterns dumped out by EC engine.

➢ **C1 (50Mhz) to C3 (150Mhz) design:**

The prime objective of C3 was to achieve high performance. C1 to C3 design modifications involved RTL changes for speed and a new target technology node. Most of the block interface was retained, making it an ideal case for equivalence checkers. The various speed related modifications in C3 are discussed in detail in [3]. The data path blocks are coded in Module compiler Language (MCL) for achieving higher speed goals. Equivalence checkers were used in

1. RTL-to-RTL verification of C3 sub designs versus C1 sub designs
2. RTL (VHDL)-to-Netlist(MCL) verification for the blocks coded in MCL.

3. Netlist-to-Netlist verification on C3 design was used to verify various views like

- Scan-inserted netlist
- Repeater and spare gates inserted netlist (netlist was hacked to insert repeater and spare gates)
- CTS-inserted netlist
- Post-layout netlist

Issues faced and solutions:
(i) MCL does not support hierarchy: The sub block 'alu' netlist generated from MCL netlist is therefore flat. The tools failed to complete equivalence check with RTL/VHDL Netlist. To make EC feasible, each of sub blocks of 'alu' was coded as MCL functions. Each of this function could be first verified. A script to edit MCL code to create a black-box view of specified functions was written, the resulting netlist (from Module Compiler) could then be ECed.

(ii) Top-down EC: The post-layout netlist was flattened, where as the pre-layout was hierarchical. The Top-down verification approach had failed. Hence, we enforced rule-based hierarchy extraction from the post layout netlist and we could then verify the hierarchy-extracted netlist using bottom-up approach. Bottom-up approach involves verifying individual sub blocks first and verifying the next level after block-boxing them, this cycle continues till the top block is verified.

(iii) Redundant ports: One of our sub designs had redundant ports. Both of the redundant ports were present in C1 netlist. In C3, synthesis optimization resulted in using only one port leaving the other ports unconnected. Due to this EC failed. After providing the redundant ports mapping information to the EC tool, the block passed EC.

(iv) Layout scan optimisation: Layout tools introduced lot of extra ports at sub-design boundaries during scan optimization. This resulted in mapping failure. The EC reports were to be manually analyzed to check that rest of the ports was equivalent.

(v) Multiple Clocks port mappings: CTS created lots of clock ports for each sub-block with arbitrary names. This posed mapping problems. We enforced a common prefix for all new ports due to CTS, thereby making rule based mapping feasible for EC.

(vi) Tristate Shifter was replaced with AND-OR shifters. These two were equivalent only on a 'one-hot' condition on their shift controls. It is much easier to specify such constraints in Design VERIFYer compared to Formality. Design VERIFYer was used with 'one-hot' property forced on shift controls, to EC the shifter block successfully.

- **C3 (150Mhz) to C4 (180MHz) design:**

C3 to C4 was mere technology migration. EC was used to verify Netlist-to-Netlist at all the stages similar to C3 design.

Issues faced and solutions:
(i) Due to boundary optimizations, bottom-up approach could not be applied as block boundaries got optimized. Most of the boundary optimizations are due to removal of multiple inverters across block boundaries. Synthesis tool renamed such ports with the suffix 'BAR', which indicates the inversion in the polarity. We could generate mapping automatically using this suffix information. This could be successfully used in ECing the sub blocks of the netlist.

The EC of VHDL verses MCL of datapath blocks of C27X, replaced simulation completely in the verification of MCL model. EC found one bug in auxiliary register file and seven bugs in 'alu' MCL code.

3.2.2 EC usage in our Next Generation core (C28X)

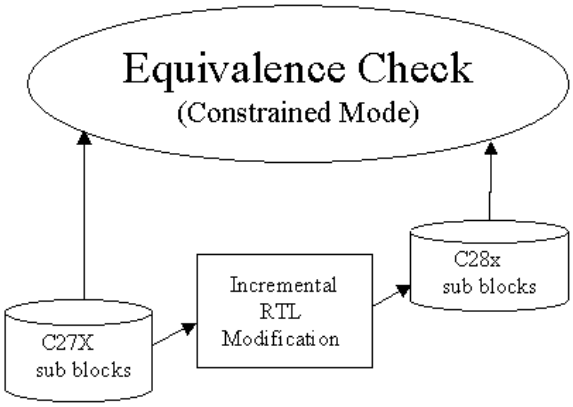

Fig 3.3: EC usage in C28X RTL verification

C28X core is object-code compatible to C27X. The changes in most of the sub designs are incremental. All C28X features are made effective only with new ports. This guideline we followed helped us in doing EC of C28X sub designs against corresponding sub designs of C27X, with the newly added ports forced to inactive state. Here, we use the EC tool's capability of doing EC of two designs under certain conditions. In this case, EC ensures the functional correctness of C28X RTL in C27X mode. This is illustrated in figure 3.3. Therefore the unit-level simulation can primarily focus only on the incremental changes due to C28X. This is a good example of an area where Equivalence Checker cuts down the verification time of a new design, which is a super-set of existing design.

Further, BIST (Built In Self Test) flow hacks the netlist directly to rip-apart the scan-chain and re-stitches the flip-

flops forming number of stump channels. We need to ensure the correctness of the functionality of the netlist after this hacking. In order to improve the coverage, the BIST flow inserts multiplexers to gain controllability of internal points. Further more, to increase the observability the state of intermediate points of a logic cone are captured in a flip-flop, which is added for this purpose. All these additions happen directly on the final netlist. EC is the best fit to verify the BIST inserted netlist for functional correctness.

Our experiences with EC show that a typical time to verify a new netlist view was about a couple of hours for C27X core. The functional simulation of the same netlist takes about 10 days to run (around 350 assembly tests) assuming regression to be running on one ultra machine. Hardware accelerators can be used to speed up the simulation, however it is an expensive solution.

4 GUIDELINES FOR EASY EC

1. Always EC between two successive stages of abstraction. E.g. EC the pre-scan Vs post-scans Vs Repeater + Spare gate inserted Vs CTS-inserted Vs post-layout. Don't verify the PG netlist with the initial netlist.
2. EC of Full chip RTL Vs gate-level flat netlist is very inefficient. Maintain the hierarchy in the netlist to enable hierarchical verification (bottom-up strategy yields best results). Flat netlist could then be verified against hierarchical netlist.
3. Pick the appropriate tool based on the merits and de-merits listed above. E.g. Pick Design VERIFYer to force if you were to force signal properties before doing EC.
4. Use appropriate options based on the characteristics of the design to the help the EC tool. E.g. Design VERIFYer treats 'X' and 'Dont_Care' differently. So, if your VHDL code contains 'X' assignments in case statements then use the "Enable_Dont_Care_Default_Mapping" attribute or use "General_X_to_D_Conversion" as appropriate.
5. When comparing two gate-level netlist that are structurally very different but with the same function (E.g. same netlist obtained using Design Compiler and Module Compiler), the best approach is to verify each gate-level netlist against the original RTL.
6. Constrain the CTS flow to add a common prefix for all the new ports that is created at sub block boundaries. This will help in creating rule based mapping for multiple clock ports.
7. Maintain the boundaries of the sub designs as much as possible. This would help in bottom-up verification.
8. Do not combine the Boundary optimizations with compile. Do Boundary optimization separately after the compile. The reason behind this is to have minimal changes when the sub design boundary is disturbed, so that top-down EC can be applied. With major changes, it is easier to do EC if the block boundaries are retained
9. Extract the hierarchy from the post layout netlist before ECing it. This reduces the EC time.
10. While writing a MCL code, partition the code into functions, each function mapping on to a sub design. The functions individually can be verified against the VHDL model of the sub designs. This also allows an easy way of black-boxing the sub design for the top-level verification.
11. Avoid redundant ports in the RTL. A set of redundant ports might be unconnected after synthesis optimization. This would need additional mapping information for successful EC.
12. Incremental changes to a design should be done as much as possible with a new set of ports. This would help in ECing the new model with new ports forced to inactive state, to ensure the functional correctness of old mode of operation.

5 CONCLUSIONS

We presented our experiences with the equivalence checking as applied successfully in the verification of commercial DSP cores. We highlighted the key areas where EC is best fit. We described the hybrid flow developed for verification of our designs. We shared our experiences with the Issues faced with EC and solutions we deployed. We concluded with a set of guidelines that would help in easing the equivalence-checking job, thereby making it more applicable to a design. Our future work would be to add a simulation hook up to the EC flow, for the extra flexibility of debugging. Our experiences with EC reveal that a considerable amount of verification time can be cut down reducing the total design cycle time.

REFERENCES

[1] Design VERIFYer(2.3) users guide & reference manual

[2] Formality(1999.05-FM1.0) users guide & reference manual

[3] A Framework for Cost Vs Performance Tradeoffs in the design of Digital Signal Processor Cores by Karthikeyan Madathil, Subash Chander, et al - In the proceedings of the conference on VLSI design 2000.