International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 03 Issue: 10 | Oct -2016 www.irjet.net p-ISSN: 2395-0072

Functional Constraint Extraction at Register Transfer Level for ATPG to Improve Verification in terms of Coverage

A. Venkata Ramana[1], J. Anusha[2]

[1]PG Scholar, Department of ECE, JNTU Anantapur, Andhra Pradesh, India
[2]Lecturer, Department of ECE, JNTU Anantapur, Andhra Pradesh, India

---

Abstract - The gate level techniques are used in simulation to identify ISE's and also these techniques are implemented based on Sequential Automatic Test Pattern Generation (ATPG). But now a day's IC's are very complex so by using above gate level method, it is very difficult to compute and also in this process, illegal states are occur but this causes unwanted behavior and false error detection in the verification process. To overcome this problem a new tool is proposed based on VHDL Parsing Expression Grammar (PEG). Functional Constraint Extraction (FCE) at Register Transfer Level approach is used in this new tool. This Functional Constraint Extraction (FCE) is used in ATPG process to generate pseudo functional scan test patterns which avoids ISE's. The end result of this brief is an automatic tool that performs HDL parsing and analysis of legal state computation and functional constraint generation. This approach is also used to avoid the false error detection during RTL simulation.

Index Terms— Parsing Identification, HALS States, ATPG, Verification

I. INTRODUCTION

As hardware complexity continues to follow Moore's law, verification complexity is also becoming even more challenging, with verification now estimated to be taking 50%–70% of the total time of a project [1]. . Results obtained by applying this methodology have confirmed its potential in terms of verification coverage and time improvements. Only an experienced Verification Consultant can transform an VLSI engineer into a Verification Expert. verification folks who are new to CDV usually ask, which option [1or 2] is good for achieving 100% coverage: [1] Minimum number of seeds and maximum transactions pretest case[2] Minimum number of transactions per test case and many seeds. It's always good to consider various factors like the simulation runtime, DUT features, unusual bugs etc. while defining the test case, rather than just focusing only on reaching coverage goals. Most of the verification engineers start their career as an HVL expert. They realize the importance of Assertion Based Verification, only when they become seasoned verification engineers, especially when they take the complete ownership of RTL sign-off. If you are working in the VLSI domain, especially in the functional verification domain, you should know about the latest verification methodologies and technologies. Most of the engineers run the regressions and spend most of their time on analyzing the coverage reports. They wrongly assume that they are verifying the chips. Actually they are managing the regressions and reporting the bugs to the designers. Functional verification continues to be one of the most expensive and time-consuming components in a typical design process. Practical functional verification relies on extensive simulation of directed and/or guided random tests due to its flexibility and scalability Shown in Fig.1.

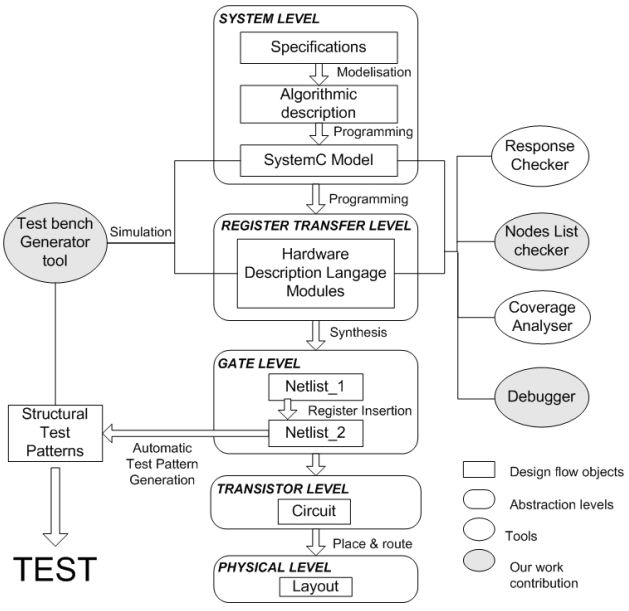

Fig 1: The additional tools of the proposed methodology in a standard implementation flow

Although simulation-based verification can be very effective, its success both in terms of total effort spent and final verification coverage achieved depends heavily on the quality of the tests in use. Effective

tests can achieve higher verification coverage in shorter time, which saves engineering resources and improves condense on the quality of the design under verification (DUV). However, generating effective tests for complex designs has always been a challenging problem. Directed tests are written to cover corner cases and important features of a design. Writing directed tests has been a dominant test generation methodology even with the emergence of constrained random test generation. Directed tests are crucial for verification as in many cases they are the only tests that can reach corner cases.

These patterns are generated by an ATPG tool based on a launch-on capture transition fault model. The result is a verification environment that can be seamlessly integrated in the design flow, without requiring circuit modification or remodeling steps. Figure 1 describes a standard design flow and its relationship with verification and test, more specifically the scan-based test[1] and Register Transfer Level (RTL) model verification.

II. EXISTING METHOD

The proposed verification environment encloses RT and gate levels. We assume that a SystemC golden model is available and used as a reference model by the verification system at the system level. Note that such a model is also required to apply any verification methodology. In this paper, the SystemC golden model used has the same level of details as the RTL model verified. It is also possible to use a transaction level SystemC model as a golden model. In fact, in [2] the authors describe how to use RTL test benches for verification of a SystemC model at a higher abstraction level (as transaction level).

A. Test bench Generator Tool

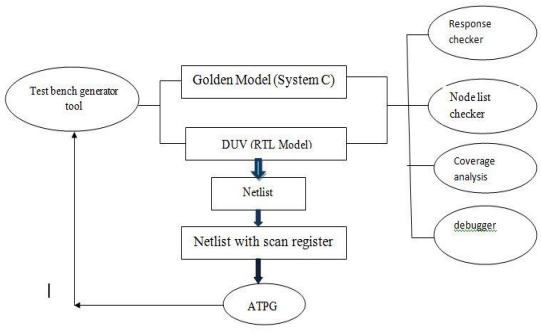

Fig:2 Verification tool

One of the key components of the proposed methodology is its automated aspect. In fact, to effectively generate verification data for functional VHDL descriptions based on structural test patterns, we built up an automatic test bench generator that executes the following algorithm:

B. Nodes list checker:

A list checker module is added to the verification environment. A list of transitions and states to be simulated based on the FSM model is created, and for each simulation the module checks which state/transition had been simulated by the patterns and update its list by removing the one simulated. After the simulation is completed, the set of nodes not covered will include the set of states and transitions remaining in the list. This list can be used for further directed simulations, to identify the target nodes for directed patterns in order to accomplish an even more complete coverage.

III.PARSING EXPRESSION GRAMMER

A PEG is a recognition-based formal foundation for language syntax. It describes the language syntax in terms of a set of rules [6]. A VHDL PEG was defined in [6], but the grammar was not complete, as it did not cover different condition constructs and overlaps as well as design hierarchy. Based on the definition in [6], we developed a more complete VHDL PEG described as follows.

1. ∑ = [keywords, symbols ('<=', '('…), operators]. In Fig. 2, terminal symbols are identified, in upper case.
2) N =[Module, entity,architecture, port, component, signal_type, component_inst, process, if_st, case_st, case_comp, condition, signal_ass, operation, VHDL_type, label, value, with_st, when_st].
3) e_S = Module.
4) The set of parsing rules P is defined in Table I, where terminal

Table: 1Parsing rules

Parsing Rule	E	A
P1	Entity architecture	Module
P2	'ENTITY IS' port 'ENTITY END;'	Entity
P3	'PORT ('(label':'('IN'|'OUT') VHDL_type)+');'	Port
P4	Component *'BEGIN' (component_inst*|Process*| signal_ass*|with_st*|when_st*) 'END ARCHITECTURE;'	Architecture

P5	'COMPONENT IS' port 'END COMPONENT;'	Component
P6	'SIGNAL' label'; ' VHDL_type ';'	Signal type
P7	label; label 'PORT MAP ('label=>label)+');'	component_inst
P8	'BEGIN'(if_st*|case_st*|signal_ass+) 'END PROCESS'	Process
P9	(numerical value| label)(operator operation)?	Operation
P10	'IF('condition')THEN' (case_statement *| if_statement *| signal_assignment+)('ELSE case_statement*| if_statement *| signal_assignment+)?'END IF;'	if_st
P11	'CASE'label 'IS'(case_ component)+ 'END CASE;'	case_st
P12	'WHEN'value=> case_st*|if_st| signal_ass+';'	case_comp
P13	Label('<'|'>'|'<'|'>'|'=') operation	Condition
P14	Label<=(value| operation)';'	Signal_ass
P15	KEY WORDS+	VHDL_type
P16	[a-z]+	Label
P17	[0-9]+	Values
P18	'WITH' label 'SELECT' (signal_ass | 'WHEN'value)	With_statement
P19	Signal_ass |'WHEN' condition ('ELSE'| label| 'WHEN'| condition)*	WHEN_st

Therefore, the analysis performed based on the proposed VHDL PEG, is able to identify the majority of VHDL constructs except user defined types as well as subtypes, files, and loop constructs.

IV.PROPOSED METHODE

It is important to specify that we consider two types of states:

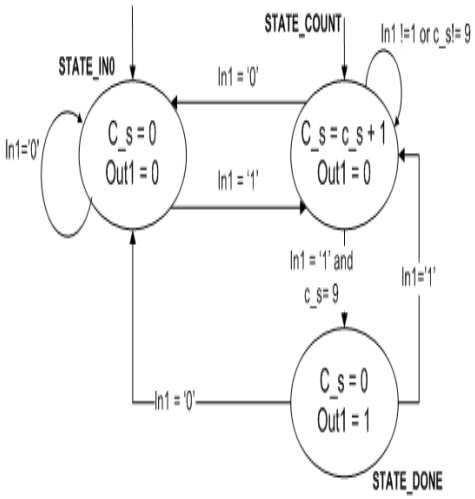

Fig: 3HALS State identification flow

The high abstraction level state (HALS) and the low abstraction level or RTL state (simply called state in the rest of this brief) related to the state signals. We define a high-level state as the set of RTL state signal assignments associated with particular conditions within a process. So Above HALS state represent in the following way that are nothing but RTL States.

Working with HALS instead of RTL states minimizes the computation complexity and time. Instead of computing the corresponding results for each unique state value and each possible combination, which may increase the procedure time and complexity, we compute ranges of state signal values that can occur at the same time, defining an HALS(Fig 3 and 4).

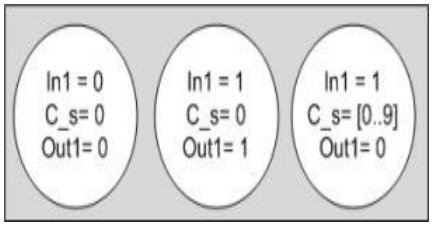

Fig: 4HALS states

Implementing tool:

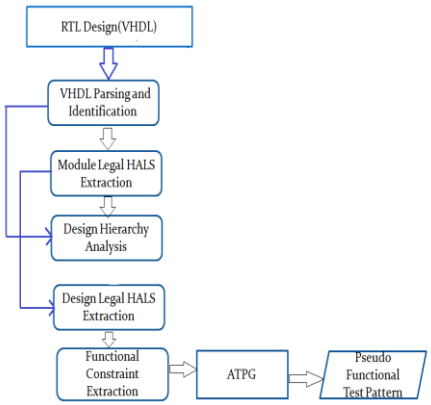

Fig: 5 Implanting Tool Flow chart

A. VHDL Parsing and Identification

Our first step in the methodology is a VHDL parsing and identification, which corresponds to a lexical and syntactic analysis of the VHDL code that helps identify different VHDL statements. The parsing is done line by line, based on the proposed VHDL PEG[6]. The tool reads the HDL line and translates it into a stream of tokens: each token is a sequence of characters representing a symbol, such as an identifier, an operator, and so on. Therefore, based on the VHDL PEG finite set of parsing rules and the sequence of tokens, each statement is identified in its context, and corresponding data and dependencies are extracted and stored in the corresponding statement representation.

B. Module Legal HLS Extraction:

The procedure implementations used for the module legal HLS extraction is described in [9]. A VHDL operation consists of a set of arithmetic or logical operators whose inputs can be signals, variables, or constants. An operation on N different signals x can be modeled as a function f as follows:

$f(a1X1,...,ajXj,...,aNXN)$, with $j \in \{1 \dots N\}$ and a= cte;

1) The function f corresponds to the set of operators;
2) The inputs of the function correspond to the operands;
3) The domain (D) of the function corresponds to the set of ranges of operand values

$$D = \left\{ \bigcup_{j=1}^{N} [x_{j\min}, x_{j\max}] \right\} \quad (1)$$

4) the range R of f is the set of all resulting outputs

$$\text{with } R = [f_{\min}, f_{\max}]. \quad (2)$$

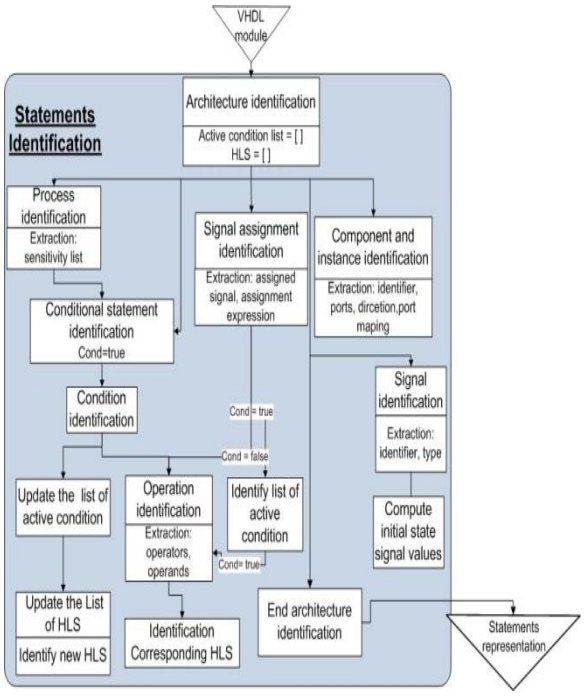

Fig: 6 Statements identification flow chart

Here is an example of an operation: $c\_s <= c\_s + 1$; where the function f is: +; the inputs are: $x0 = c\_s$ and $x_1 = 1$; the domain(D)of the function is: $D = \{[c\_s\min, c\_s\max], [1, 1]\}$; the range $R$of f is: $R(c\_s) = [c\_s\min, c\_s\max+1]$. As for the domain, it is initially set to the initial state signal values (LV0j) computed based on the respective signal types, and is updated with every condition evaluation.

HALS Computation: As mentioned earlier, an HALS is the set of state signal ranges of values that can appear simultaneously in the design under the same conditions. It corresponds to the values extracted from signal assignments, under the same conditions, as well as the actual condition values. Each HALS is, therefore, characterized by its constraints that consist of the set of conditions, and its effects that consist of the signal assignments under these conditions

C. Design Hierarchy Analysis:

During syntactic and lexical analyses, port mapping is examined and connections with other instances are detected. Design hierarchy borders are crossed to take upper-level and lower-level instances into account. A data structure is built, where each instance is defined with its hierarchy level, its input dependencies as well as its output dependencies.

Table:2 Binary counter HALS States

Id_halstate	I		IS		O	
	In1		C_s		Out	
	min	max	min	max	min	max
1	0	0	0	0	0	0
2	1	1	1	9	0	0
3	1	1	0	0	1	1

Mod-5 Counter:

Id_halstate	min	max	min	max	min	Max
1	0	0	0	0	0	0
2	1	1	1	5	0	0
3	1	1	0	0	1	1

D. Design Legal HALS Extraction

To compute the final design set of legal HALS values, we need to combine the set of legal HALS of all the instantiated modules while considering instance connections and dependencies[1]. The combination process is carried out based on the flatten model of the design that shows explicitly the instance dependencies and connections. All combined sets are built based on instance connections while respecting HALS dependencies and avoiding having ISEs built as follows:

For each pair of connected instances (*A* and *B*), we compute the resulting combined set of legal HALS values whose constraint may be narrowed, as compared with the initial sets. Once the whole set *Q*(*A*, *B*) is extracted, it will be combined to HALS sets of other instances that are connected to *A* and/or *B*.

For each HALS_{i}(A) \in Q_{A}
For each HALS_{j}(B) \in Q_{B}
X= Out (A) \cap_{ }{In (B)}
//where out/In(M) is the set of possible values of
//the output/input of the module M
If X = Ø {then}
Next;
Else
HALS(A,B) =

$$\bigcup_{\substack{min\\max}} \{In(A), Is(A), X, Is(B), Out(B)\}$$

Q(A, B) = {Q(A, B) U HALS(A, B)}

End if;
End for;
End for.

The process continues until all instances of the design have been checked. The resulting set of the whole process models the legal HALS values of the entire design, the detailed procedure is described in [10] and [11].

E. Functional Constraint Extraction Algorithm:

Our objective is to construct one large functional constraint that models the design. To respect the ATPG tools requirements, the functional constraint should be a Boolean formula over signal nets, which is a list of Boolean operations involving several literals, with a literal being either a variable involved in the function or its negation. This is true for most commercial ATPG tools as well. Otherwise, if the ATPG expects a different syntax, our tool can easily be adapted to produce the corresponding output. After extracting the legal HLS set of the design, we proceed to constraint extraction. We first extract, from each legal HALS value, a constraint called the HLS constraint, which is a conjunction of individual constraints whose literals are the state signal bits.

The following is description of a pseudo code for extracting functional constraints:

1) k = 1;//number of HALS;
2) While (k = S){// individual constraints;
3) List(*c*ok,…*c*nk) =Create_individual_ constraints(S);
4) *C*k= *c*0kand ... and ... c nk;
//conjunction of individual
constraints;
5) k = k + 1;}//end while;
6) //the design constraint is the disjunction of HLS constraints;
7) *C* = *C*0 or ... or *C*.

V. PROPOSED VERIFICATION ENVIRONMENT

As mentioned before, we proposed in [1] an automatic verification environment based on the use of structural test patterns as simulation patterns; more specifically, the use of launch-on-capture transition test vectors with emulated scan registers in RTL simulation[2]. To obtain these automatically generated test patterns, regular steps of the design flow must be performed in a preliminary way, namely synthesis, scan insertion, and ATPG.

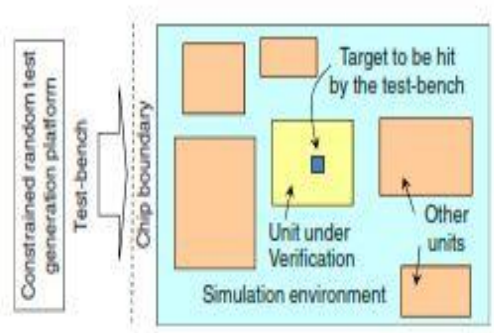

Fig: 7Verification at Simulation level

Functional verification continues to be one of the most expensive and time-consuming components in a typical design process. Practical functional verification relies on extensive simulation of directed and/or guided random tests due to its flexibility and scalability. Although simulation-based verification can be very effective, its success both in terms of total effort spent and final verification coverage achieved depends heavily on the quality of the tests in use. Effective tests can achieve higher verification coverage in shorter time, which saves engineering resources and improves confidence on the quality of the design under verification (DUV).

This may seem counterintuitive as those steps are generally accomplished once the verification of the RTL model is considered satisfactory. By preliminary, we mean without any particular constraints (e.g., frequency, area, and coverage), as these steps have to be redone once the verification of the RTL model is fully completed. Consequently, this environment does not aim at verifying if the scan insertion is properly performed. Once the test patterns are generated (at the gate level), the proposed approach emulates the presence of scan register chains during RTL simulation-based verification, by associating them to state signals forcing, and resulting in controllability improvements. In addition, the use of launch-on-capture transition as fault model helps simulate and exercise the most efficiently the design functionality [4]. In addition to the ATPG is used in full scan mode. The methodology proposed

However, as underlined earlier, a drawback of this methodology is the potential presence of ISEs; when the generated test patterns used in the simulation contains an ISE signal value, with the DUV and the reference model having different coding styles, the designs may behave differently, and inducing false error detection. To overcome this obstacle, we introduced the constraint extractor tool proposed in this brief in the verification environment.

Fig. 8 shows the complete verification environment. We assume that a reference model is available. Note that such a model is also required to apply most common verification methodologies, such as the constrained based and pseudorandom methodologies. In this brief, the golden model used has the same level of detail as the RTL model verified.

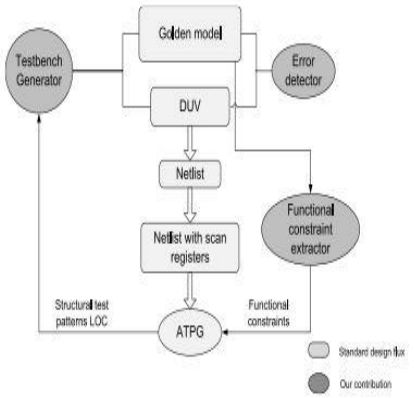

Fig: 8Verification tool

Table:3 Verification Circuit Characteristics:

Circuits	VHDL lines	PI/PO	FF	Gates
Buffer	15	4/1	4	5
Counter	20	4/1	4	15
B02	25	3/1	4	25
B03	141	11/8	30	149
B05	332	1/6	34	935

The proposed environment is fully automated and incorporates three complementary tools: 1) functional constraint extractor described in this brief; 2) test bench generator that adapt and apply structural test patterns in the RTL simulation; and 3) error detector [1] that monitors the DUV and the reference model responses for possible errors.

Table:4 FSM Coverage Comparison

	FSM State(%)		FSM transition(%)	
	PRA	PA	PRA	PA
Buffer	100	100	100	100
Counter	75	100	66.6	100
B02	71.4	100	66.6	100
B03	100	100	100	100
B05	79	95	61	94

We implemented the proposed environment on a Pentium, Dual-Core CPU, 2.2 GHz processor machine, with 1.99 G of RAM. Experiments were run on some ITC'99 benchmark circuits[8]. Experimental results are presented under two different angles:

1) the effectiveness of the overall verification environment in terms of coverage, when compared with other verification methodologies

2)and also find effective statements in the following program

Table:5 Fault Coverage Comparison

	No. of injected errors	PRA(%)	PA(%)
Buffer	20	100	100
Counter	20	64	80
B02	20	34	54
B03	20	40	65
B05	20	45	72

VI.CONCLUSION:

We presented a new simulation-based verification methodology based on the automated application of structural ATPG test patterns in the verification process. In this brief, we have presented a new methodology for legal HALS values extraction and functional constraints built to avoid ISEs in generated test patterns used for verification. The proposed methodology aims to reduce complexity and avoid heavy computation as compared with the techniques presented in the literature. We have to implement these methodology in verilog Programs with better results than VHDL program. And also find time taken to verify our program and no of FSM statements are executed during this process all are results also we have to proved.

REFERENCES

[1].C. Hobeika, C. Thibeault, and J. F. Boland, "Functional Constraint Extraction From Register Transfer Level for ATPG," ieee transactions on very large scale integration (vlsi) systems, vol. 23, no. 2, february 2015

[2]. C. Hobeika, C. Thibeault, and J. F. Boland, "Automatic verification methodology based on structural test patterns," in *Proc. Joint IEEE NEWCAS Taisa Conf.*, Jul. 2009, pp. 292–295.

[3] "Functional verification study," in *Industry Study (Conjunction with Mentor Graphics)*. San Francisco, CA, USA: FarWest, 2007.

[4] C. Hobeika, C. Thibeault, and J. F. Boland, "Use of structural tests in RTL verification," in *Proc. 1st Microsyst. Nanoelectron. Res. Conf.*, 2008, pp. 133–136.

[5]B. Ford, "Parsing expression grammars: A recognition-based syntactic foundation," *ACM SIGPLAN Notices*, vol. 39, pp. 111–122, Jan. 2004.

[6]. C. Hobeika, C. Thibeault, and J. F. Boland, "Illegal state extraction from register transfer level," in *Proc. 8th IEEE Int. NEWCAS Conf.*, Jun. 2010, pp. 245–248.

[7] S. Davidson. (1999). *Characteristics of the ITC'99 Benchmark Circuits*.[Online]. Available: http://www.cerc.utexas.edu/itc99-benchmarks

[8]W. K. Lam, "Hardware design verification, simulation and formal method based approaches," in *Prentice-Hall Modern Semiconductor Design Series*. Upper Saddle River, NJ, USA: Person Education Inc.,2005.

[9]C. Hobeika, C. Thibeault, and J. F. Boland. (2013, Oct.). *TechnicalReport: FunctionalConstraint Extraction from RTL for ATPG* [Online]. Available: http://arxiv.org, arXiv:1310.01001

BIOGRAPHIES:

Venkata Ramana A M.Tech in Digital Systems and Computer Engineering at JNTU,Anantapur. I am interested in VLSI Design flow and verification of ASICS at logic level.

Anusha.J did M.Tech in JNTU,Anantapur.she is interest in digital electronics and ASIC Design implementation. I did research in computer networks.