XDC Support in Synthesis Tool using YACC

Sandeep Kumar Mittal
Research Scholar
University Institute Of Engineering And Technology
Kurukshetra University

Sanjeev Dhawan, PhD
Assistant Professor
University Institute Of Engineering And Technology
Kurukshetra University

ABSTRACT
With progressive FPGA technology, XILINX required a need for new format to provide needful assistance as a part of their tool set for design constraints. In order to achieve the same, XDC (Xilinx Design Constraints) was introduced. In this paper, we describe a generalized technique to integrate XDC in synthesis tool using YACC. Proposed system not only tokenizes the input XDC commands but syntactically and semantically validates them to generate desired lexeme. This paper shows the parsing mechanism to generate desired lexeme which can be used by several synthesis subsystems for further computations.

General Terms
Synthesis tool, lexeme, XDC

Keywords
Syntax, Semantic, YACC

1. INTRODUCTION
This paper presents an abstraction layer which accepts XDC commands as input, parses it using YACC & TCL and creates the desired lexeme to retrieve the constraint commands be used by synthesis subsystems. This method can convert an annotated context-free input into a deterministic LR output.

XDC commands are checked for syntactic and semantic correctness. Hardware and software requirement for implementation is C++ language on any 32/64 bit operating system along with YACC and TCL to generate desired lexeme [10]. Lexeme is proposed to generate output in a predefined sequential ordering along with sub-commands of a given XDC command.

2. ICFIT
To meet the requirement of this fast paced modern era, smarter algorithms are being used and developed to collect required system information. So, it is easier to have validated information, in a manner to support the existing system more efficiently than before. The proposed algorithm in this paper can be viewed as generalized overview of input stream being computed through set of rules via YACC, pre-analyzed and verified for their

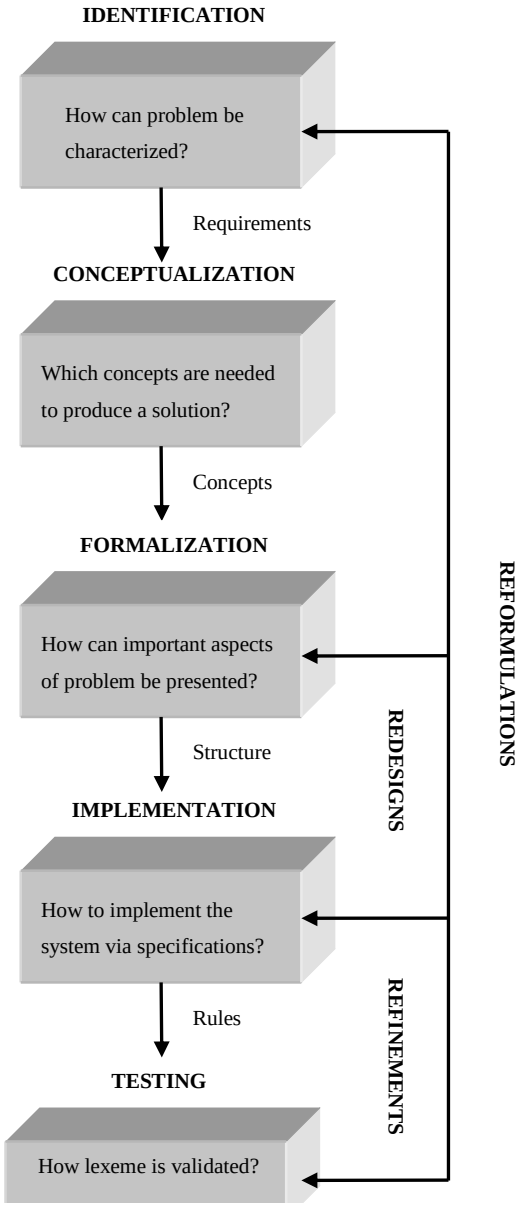

Figure 1: ICFIT stages and roles

syntactic and semantic relationship with XDC constrained taken as fundamental for computation in a synthesis tool. This paper presents the proposed algorithm broadly into five categories which are collectively labeled under ICFIT.

2.1 Identification

This stage [8] helps in identifying the XDC commands set and its subcommands as stock of list. This way we are able to find the overall complexity and size of commands set, on which input is being required to be computed. This is the first and most important requirement required for conceptualization stage.

2.2 Conceptualization

It relates [6] the concepts and relation among the main XDC commands and subcommands. Here we prepare the flow plan for the grammar rules to be used for implement a XDC command. The flow takes into account the specific sub-commands as specified in XDC standard for each command along with the specific values incorporated by it during run time.

2.3 Figures Formalization

Stage [4] expresses the key concepts and relations in formal way i.e. values based on setup and hold time. It helps in modulation of data models and methods in digital system and converts it into respective logical structural or programmable model. In our discussion we limit our self's till "-to", "-from" and "-through" sub-commands. This help to detect any clock period miss-match.

Here we finally implement the proposed discussion into a system where grammar rules are written as per conceptualization phase. Major emphasis is laid in to validate the commands before being further computed.

2.4 Implementation

This step determines the actual development of a system based on features and algorithm as analyzed at Identification and conceptualization stage. Here the grammar rules are implemented and user-friendly error messages and variables are used.

2.5 Testing

This stage [5] is responsible for analyzing and searching any ambiguity among the grammar rules written for parsing the input stream of tokens. Ambiguity can be shift reduce conflicts or reduce-reduce conflicts. Shift reduce conflicts occur when a non terminal symbol can either shift to new rule or can be reduced to a terminal symbol. Reduce – reduce conflict occurs when a token can be reduced to two terminal symbols at any given instant of parsing in the same flow.

Shift-reduce conflicts can be removed by using LR parser (Bottom Up) where as reduce-reduce conflicts can be removed by using LL parser (Top Down). Basic over view of two parsing types are:

a) LR parser => list: item
| list item ;

b) LL parser => list: item
| item list ;

3. LOGICAL CONVERSION

XDC commands are written specifically for synthesis tool by Xilinx [1]. So to parse the command input, we need to first convert the electronic circuit into the logical equivalent XDC command form. These commands are then read by using LL (1) logic, where each command is separated from its respective sub-commands in form of tokens by TCL. Token are the matched first for the command and then for its respective sub-commands by using LR (1) logic. Here Yacc is used to parse the inputs as it helps in making BNF rules fit for processing and validating the lexical and semantic nature of the constraints commands [3].

Parser maintains its own stack for each new XDC command encountered till parsed successfully or a user-friendly message is reported. Successfully parsed tokens are then dumped based the manner decided at the figure formulization of ICFIT process.

For the sake of simplicity and generalization of the logical conversation, combinational circuit is taken into account without any link to sequential circuit. Hence the paper tries the present the broad overview of the representation of electronic circuit into set of rules.

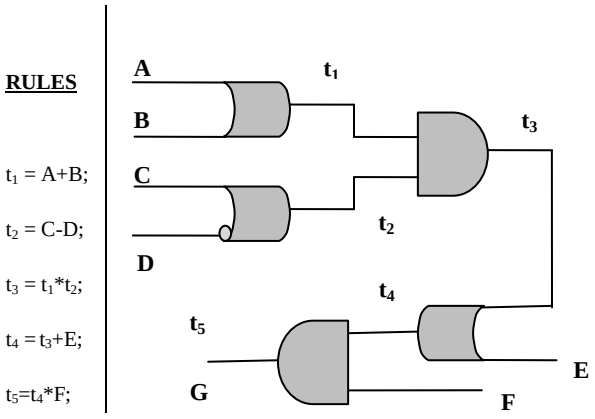

Figure 2: Generalized Logical Conversion

So, as per figure 2 the final expression obtained considering G as output is G = (A + B) * (C – D) + E * F. The input is read via LL (1) logic, thus obtain a SDT as in figure 3.

Syntactic and semantic analysis through YACC is done for each command passed as input via LR (1) logic, thus obtain their respective SDT [9] as in figure 4.

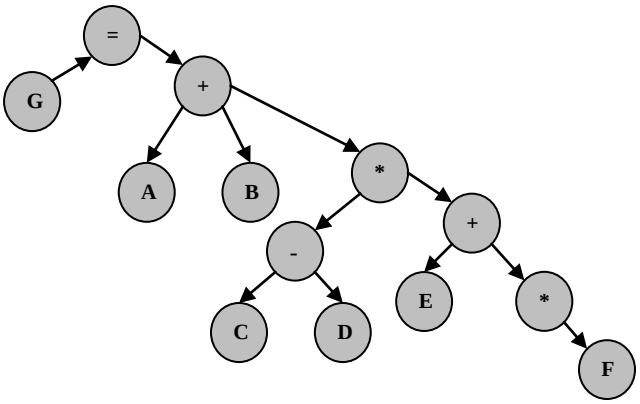

Figure 3: Parsing the expression via LL (1) logic

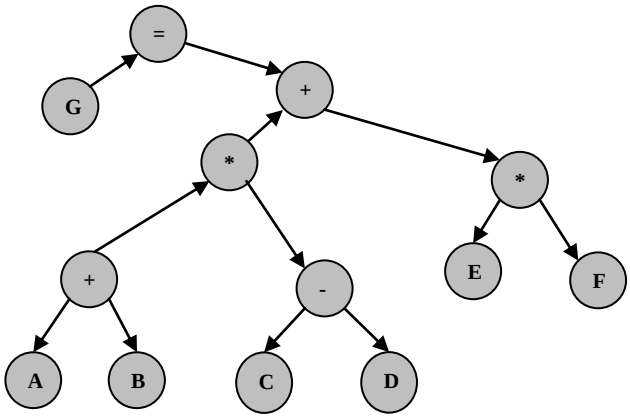

Figure 4: Parsing the expression via LR (1) logic

4. PROPOSED ALGORITHM

a. Allow the system to iterate on stream on input using stream reader. Through this any developer or user can send the collection of commands so to perform a particular job.
b. Input stream is tokenized into small string fragments which are groped as per their priorities in YACC.
c. After one full command string is identified it is abstracted via program logic so to interface with the synthesis tool. If ambiguous or error prone strings are encountered, support algorithm triggers the error code which is the controlled by error handler. [7]
d. If scanned and parsed successfully, output is displayed else respective error should be displayed.

5. PROPOSED DESIGN

System flow proposed through this paper is an overview flow of tokens [1, 2] as in figure 5. Input stream is first being tokenized by Tcl and XDC commands are matched for their existing in the reserve keyword set. If command exists, respective grammar rules are being applied to validate lexeme. The lexeme is then used by constraint handler to further interact with the sub-system of synthesis process.

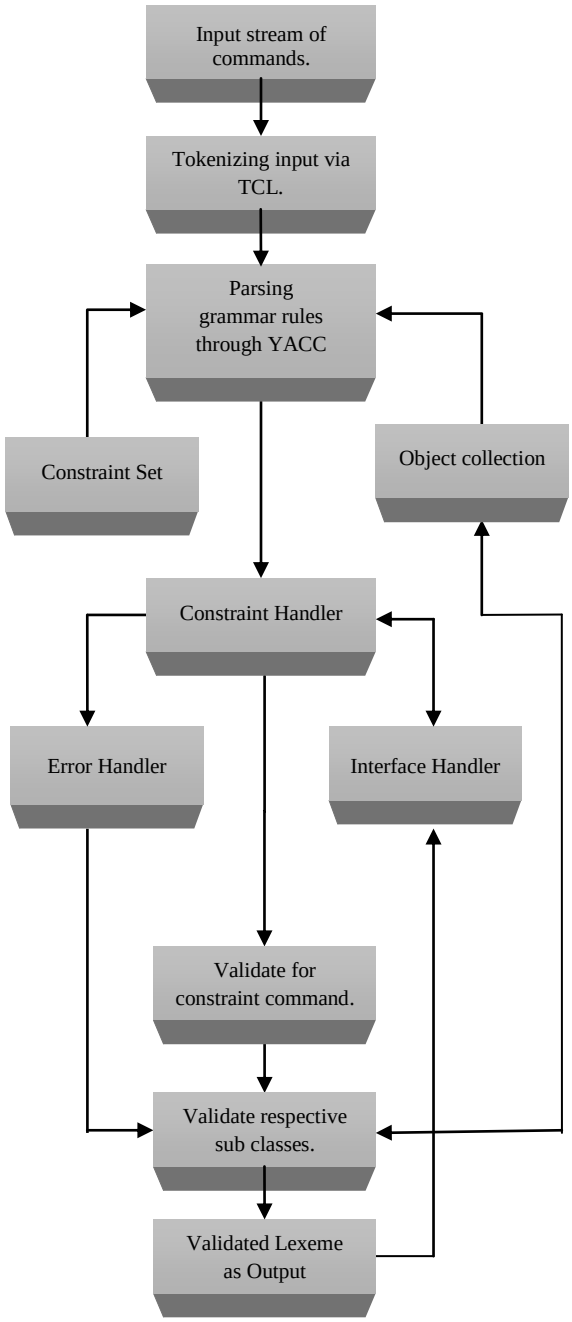

Figure 5: Overall System Flow

6. CONCLUSION

Through this paper, it is concluded that by semantic analysis and type checking even before start of the command execution of timing engine eventually boosts up the performance as lexeme is being validated. Through syntactic and semantic analysis it can not only trace the errors beforehand but also reduce development and shipment time.

7. REFERENCES

[1] XILINX, "Plan Ahead Tcl Command Plan Ahead Tcl Command", XILINX, 2012.

[2] M. Milford and J. McAllister, "Valved dataflow for FPGA memory hierarchy synthesis", In Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing (*ICASSP' 2012*), on 25-30 March 2012 at Kyoto, pp. 1645 – 1648.

[3] Stallman, Charles Donnelly and Richard, book on Bison 2012.

[4] Tarun Kumar Jain, D. S. Kushwaha, A. K. Misra, "Optimization of the Quine-McCluskey Method for the Minimization of the Boolean Expressions", In Proc. of 4th International Conference on Autonomic and Autonomous Systems (*ICAS' 2008*), on 16-21 March 2008 at Gosier, pp. 165 – 168.

[5] Raimundo Barreto, Lucas Cordeiro and Bernd Fischer, "Verifying Embedded C Software with Timing Constraints using an Untimed Bounded Model Checker 2011 Brazilian Symposium on Computing System Engineering", In Proc. of Brazilian Symposium on Computing System Engineering (*SBESC' 2011*), on 7-11 Nov. 2011 at Florianopolis, pp. 46 – 52.

[6] Srinivas Devadas, Hi-Keung Tony Ma and A. Richard Newton, "On the Verification of Sequential Machines at Differing Levels of Abstraction", in *Design Automation*, In Proc. of 24th Conference on Design Automation, ACM, 28-1 June 1987, pp. 271 – 276.

[7] Minying Sun, Hua Wang, "The Memetic Algorithm for The Minimum Spanning Tree Problem with Degree and Delay Constraints", In Proc. of 15th International Conference on Advanced Communication Technology (*ICACT' 2013*), on 27-30 Jan. 2013 at Pyeong Chang, pp. 78 – 82.

[8] Ka Boon Kevin Ng, Chiu Wo Choi and Martin Henz, "A Software Engineering approach to Constraint Programming Systems", In Proc. of 9th Asia-Pacific Software Engineering Conference-2002, *ISBN: 0-7695-1850-8*, pp. 167 – 175.

[9] Julie Zelenski, "Syntax Directed Translation", Maggie Johnson, 2008.

[10] "http://www.mentor.com/", [Online] MENTOR GRAPHICS.