the stream written in the source language to the grammar rules and create a parse tree. In this work, for developing these modules the tool ANTLR was used.

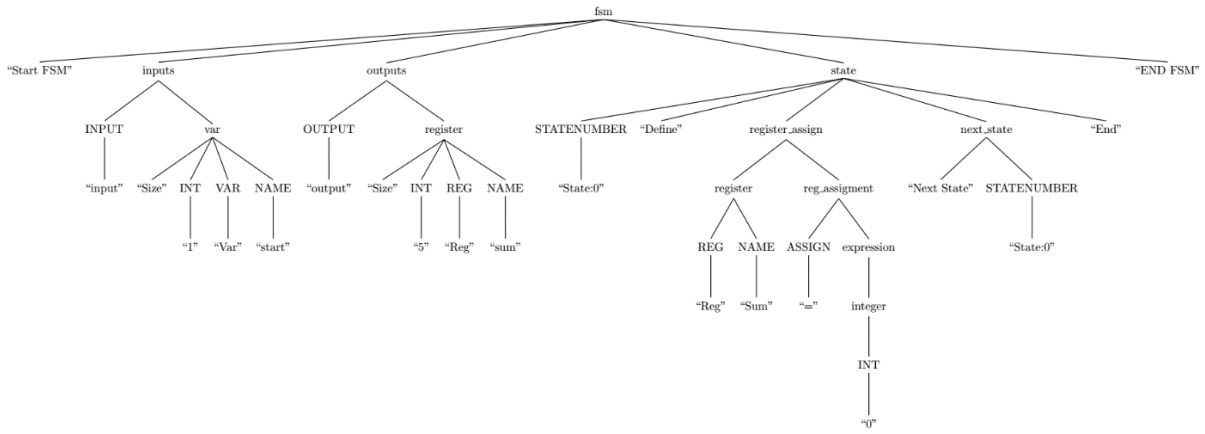

Figure 6. An Automatically Generated Parse Tree

ANTLR:

"ANTLR (Another Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. It's widely used to build languages, tools, and frameworks. From a grammar, ANTLR generates a parser that can build and walk parse trees" [4]. ANTLR creates programming code that provides an API used to parse input data.

ANTLR provided the perfect support for developing the compiler's front-end (Lexical Analyzer or lexer and Syntactic Analyzer or parser).

Once the complete FSMD Grammar was precisely defined, ANTLR was used to generate the lexer and parser modules. Given the grammar and an input FSMD specification, the compiler can now accept or deny it, and when syntactically correct, generate its parse tree.

Code Generation

In the final stage, the compiler needs to generate the code in the target language. The input of the *Code Generator* module is the parser instance, and the output is the RTL design in the target language - in this case Verilog.

ANTLR's Parse Tree Traversal:

ANTLR also provides the necessary API used to traverse the parse tree for an accepted input. ANTLR provides a *walker*, which is used to traverse the parse tree that is generated by the parser instance. The walker first enters the root node and then traverses it depth first. ANTLR provides the ability to override *listener* methods. There are two types of methods for each node, *enter methods* and *exit methods*. These methods allow for arbitrary code execution when a walker reaches a certain node.

This infrastructure allowed us to develop a walker module that traverses the parse tree to extract and store all the necessary information about the input FSMD. Once the walker finishes its traversal, it uses this information for synthesizing the design.

Parse tree's operation nodes are analyzed and used for resource allocation and binding to functional units, its variable nodes are analyzed for register allocation and binding and the design's complete data-path is constructed using this information. The steering logic is added by analyzing the data-flow to complete the generation of the data-path. Parse tree's state nodes are analyzed and the information is used for controller generation.

The data path and controller of the design are created and connected by the tree walker and the Verilog code is created using data-path and controller templates. The complete RTL design is generated using the information collected during the tree traversal. Figure 8 shows the block diagram of F2VGen.

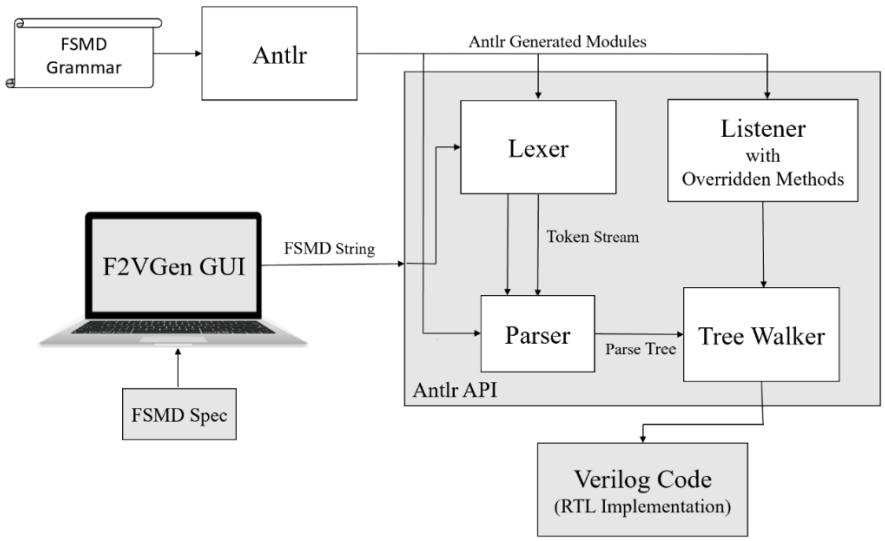

Figure 7. F2VGen Synthesis Tool

Experimental Results

In this work, several Verilog models of designs of small to medium scale have been successfully generated from their high-level FSM specification as shown in Table 1. The Verilog design implementations were successfully synthesized and mapped for download into a ZedBoard FPGA.

Table 1. F2VGen Generated Designs

Design	Inputs	Outputs	Variables	States	Synthesized	LUTs	Registers
Fibonacci	2	2	6	4	Yes	34	24
Factorial	2	2	5	4	Yes	52	16
Hannoi	2	2	5	5	Yes	21	16
Exponent	3	2	7	5	Yes	56	24
Polynomail1*	2	2	7	7	Yes	63	26
Polynomial2**	2	2	7	11	Yes	53	51
Polynomial3***	2	2	7	13	Yes	55	51
Is Even	2	2	5	5	Yes	10	8
Is Prime	2	2	8	6	Yes	142	32
Square	2	2	5	4	Yes	9	11

*$4x^2 + 6x + 7$
**$8x^4 + 2x^3 + 9x^2 + 5x + 2$
***$7x^5 + 4x^4 + 23x^3 + 8x^2 + 9x + 11$

Conclusion

This paper has presented F2VGen, a lightweight higher-level synthesis solution for auto generation of RTL designs (in Verilog HDL) from FSMD behavioral specifications. This tool successfully abstracts the behavioral modeling. This allows for designs to be rapidly prototyped and quickly synthesized to hardware devices such as FPGAs. The tool's GUI provides creative widgets that allow for incremental construction of the FSMD specification. Antlr provided the necessary modules to accept textual representations of FSDM and create a code generator based on that representation. While F2VGen's GUI is an academic proof of concept tool that is currently used for entering small to medium size FSMD examples, future work will aim to expand testing larger and more complex design specifications.

References

[1] D. K. D. a. R. Sanyal, ""Semi-automatic generation of UML models from natural language requirements"," in *ISEC '11: Proceedings of the 4th India Software Engineering Conference*, February 2011. https://core.ac.uk/download/pdf/59347307.pdf

[2] B. T. a. R. K. Hammond Pearce, ""DAVE: Deriving Automatically Verilog from English"," in *MLCAD '20: Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*, November 2020. https://arxiv.org/pdf/2009.01026.pdf

[3] C. B. H. a. I. G. Harris, ""GLAsT: Learning formal grammars to translate natural language specifications into hardware assertions"," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016. https://past.date-conference.com/proceedings-archive/2016/pdf/0334.pdf

[4] T. Parr, "The Definitive ANTLR 4 Reference", Pragmatic Bookshelf, 2013.