Formal Verification of Integer Multiplier Circuits

Jonathan Wang
Price College of Engineering
University of Utah
Salt Lake City, Utah
u1306458@utah.edu

Henry Silverman
Price College of Engineering
University of Utah
Salt Lake City, Utah
henry.silverman@utah.edu

Garrett Slack
Price College of Engineering
University of Utah
Salt Lake City, Utah
u1315263@utah.edu

Dmitry Panin
Price College of Engineering
University of Utah
Salt Lake City, Utah
dmitry.panin@utah.edu

***Abstract*—This paper employs advanced mathematical techniques, specifically polynomials, and ideals, to rigorously verify the correctness of an integer multiplier circuit. By leveraging algebraic methods, this approach will provide a deeper understanding of the circuit's behavior and enable a more robust verification process.**

***Index Terms*—polynomials, ideals, multiplier circuit**

I. Introduction

In the field of digital circuit design, integer multiplier circuits stand as fundamental components with critical applications in various computing systems, ranging from embedded devices to high-performance computing architectures. The accurate and efficient operation of these multiplier circuits is crucial for ensuring the reliability and correctness of arithmetic computations. As technology advances and design complexities escalate, the need for rigorous verification methodologies has become increasingly pronounced to guarantee the integrity of digital circuits. Our project tests and verifies an these integer multiplier circuits by employing the various methods studied in class on smaller circuits and circuits over fields.

II. Approach

Ideal membership testing is a great method to test and verify of a 2-bit, 3-bit, 16-bit, and 32-bit integer multiplier circuit. We manually designed the 2-bit and 3-bit multiplier from scratch and generated the larger circuits utilizing the ABC synthesis tool. Next we convert the blif files to sing files using our parsing python script. Then derived the RTTO (Reverse Topological Term Order) from the circuit, and used it to represent the minimal Gröbner Basis. The circuit is verified if the Gröbner Basis of J + J_0 equals 0.

III. Process

IV. Algorithms and Techniques

Techniques for ideal membership testing:

1) Setup the verification formulation over the polynomial ring R.
2) Declare a specification polynomial from the circuit f_{spec}.
3) Derive the polynomials from the gates of the circuit $\{f_1$, ..., $f_s\}$.
4) Set ideal J = f_1, ...,f_s.
5) Create the ideal of vanishing polynomials for each variable J0 =
6) Take the Gröbner Basis: G = GB(J + J_0).
7) The circuit implements $f_{\text{spec}} \iff f_{\text{spec}} \in (J + J_0)$ if and only if f_{spec} is divisible by G

The steps to run Singular, ABC, and the Python script are in the README.

V. Software implementations

We implemented a Python function with assistance from ChatGPT to convert blif files to sing files. The code defines a gate mapping dictionary that maps gate types from the blif file to their corresponding names in Singular. It iterates through and searches for lines starting with '.gate' and extracts the gate type and its inputs, parsing and separating them accordingly into the Singular file. Additionally, we added code for error handling because the Singular file was empty when experimenting.

Algorithm 1 Convert BLIF to Singular

```
if content EXISTS then
    map gates
    for each line in blif file do
        if line STARTS WITH ".gate" then
            gate_type, gate_inputs ← PARSE line
            if gate_type EXISTS IN gate_map then
                Generate Singular code based on gate_type and
                gate_inputs
            end if
        end if
    end for
    if singular_code is NOT EMPTY then
        WRITE singular_code TO output_file_path
        PRINT "Singular code written to output_file_path"
    else
        PRINT "No valid Singular code generated. Check the
        BLIF file content."
    end if
else
    PRINT "File 'file_path' not found."
end if
END FUNCTION
```

VI. Concepts learned

- Mathematical Ideals - Gröbner Basis - .blif to .sing conversion - Ideal Membership testing - Miters

VII. Labour Division

- Jonathan: Generated the blif files and converted it to Singular files and wrote the report
- Henry: Debugged Python Code
- Garrett: Help write the report
- Dmitri: Provided Singular code for 2-bit and 3-bit multipliers and visuals

References

[1] D. Ritirc, A. Biere and M. Kauers, "Column-wise verification of multipliers using computer algebra," 2017 Formal Methods in Computer Aided Design (FMCAD), Vienna, Austria, 2017, pp. 23-30, doi: 10.23919/FMCAD.2017.8102237.

[2] T. Pruss, P. Kalla and F. Enescu, "Efficient Symbolic Computation for Word-Level Abstraction From Combinational Circuits for Verification Over Finite Fields," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, no. 7, pp. 1206-1218, July 2016, doi: 10.1109/TCAD.2015.2501301.

[3] Jerry R. Burch. 1991. Using BDDs to verify multipliers. In Proceedings of the 28th ACM/IEEE Design Automation Conference (DAC '91). Association for Computing Machinery, New York, NY, USA, 408–412. https://doi.org/10.1145/127601.127703