For Loop Support and Abstract Syntax Tree Optimization in Odin II

Bo Yan / Kenneth B. Kent
University of New Brunswick
Faculty of Computer Science
b.yan@unb.ca ken@unb.ca

Introduction

A for loop is an important statement in Verilog. As a Verilog HDL complier, Odin II should support for loop statements. Moreover, in order to optimize the circuit and make it more efficient, Odin II should prune the abstract syntax tree (AST) before creating the netlist.

The project involves modifying the AST of for loops, to support netlist creation. And some optimization methods are applied to the process. Optimizing the AST is to prune the branches that can be reduced, in order to simplify the architecture of the circuit in the end.

Background

The Verilog-to-Routing (VTR) project provides a complete and open-source framework for conducting FPGA architecture and CAD research and development. Odin II is one of the core tools of VTR. It complies a Verilog source file in the following process: interpret the Verilog syntax and generate an abstract syntax tree (AST), create a netlist from the AST, and save the result by using "Berkeley Logic Interchange Format (BLIF)".

The parser of Odin II translates the syntax of a Verilog file and generates an AST. If the Verilog includes a for loop, the AST will be created according to the statement, but it cannot be transformed into a netlist. Although the second core tool of the work flow-- ABC will do some optimization, it's reasonable to optimize some parts in Odin II if it is possible. And the most optimal method is to reduce the AST at the beginning.

Methodology

1. For loop support

In order to recognize an AST that includes for loops, the structure of branches representing the syntax of for loops should be modified. The basic idea is to copy the branches showing the for loop's body as many times as the for loop expects. Then replace the for node by the copied branches. If there are some intermediate variables, reduce them, otherwise; Odin II cannot resolve the timing between statements. Figure 1 shows the original AST of a for loop.

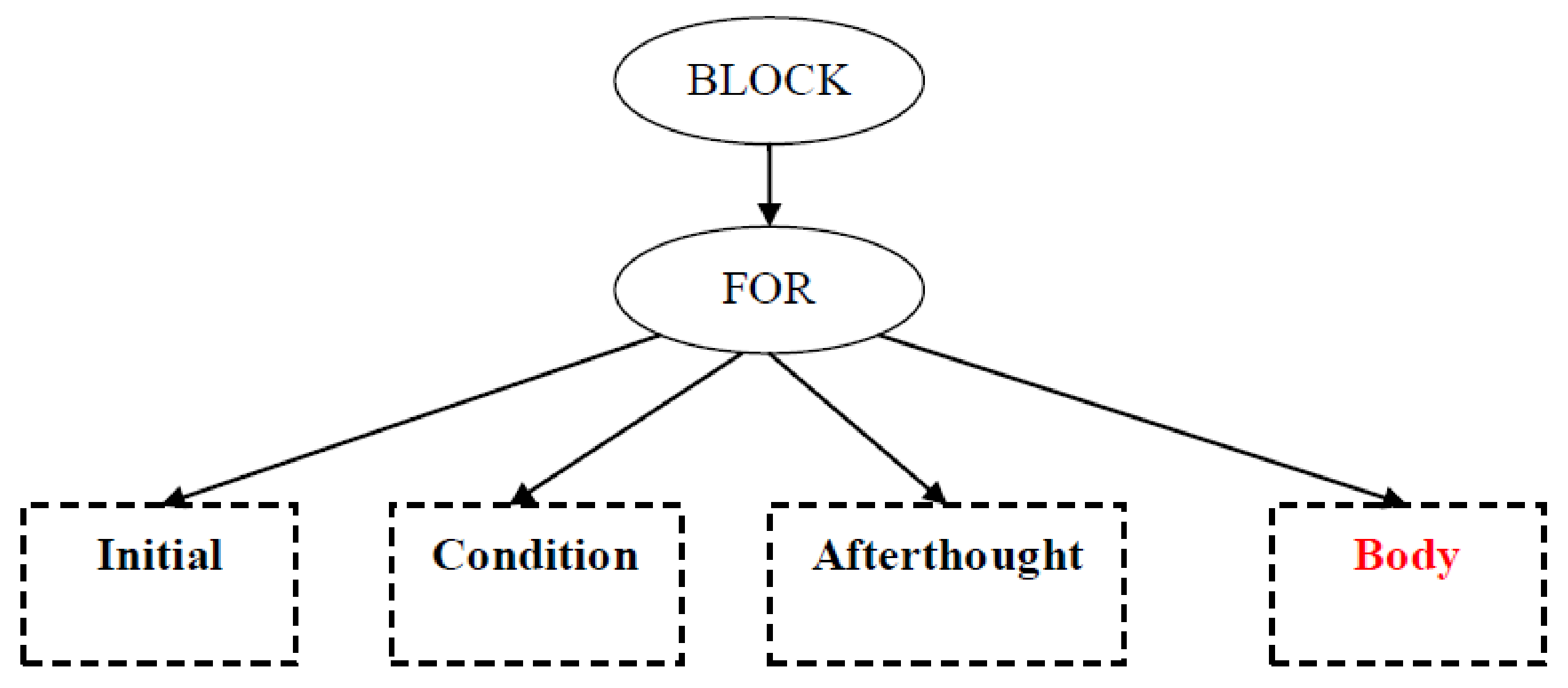

Figure1. Original AST of For Loop

According to initial, condition, and afterthought, the program can calculate the number of times to copy the body. If there are some intermediate variables, they will be reduced and the final variable will be kept. Figure 2 shows the structure of AST after being modified.

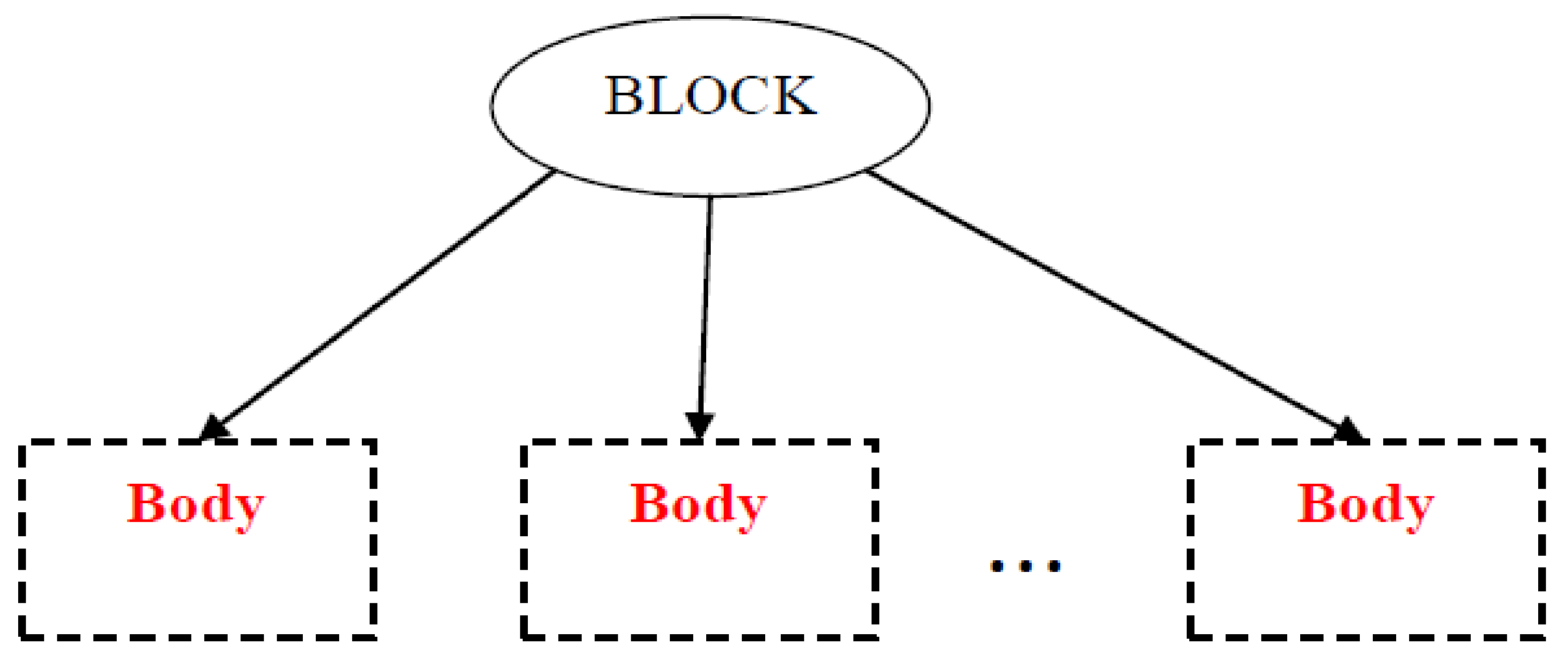

Figure2. AST of For Loop after modification

Figure 1 and 2 show the AST of a simple and standard for loop, and more complicated syntax statements will be considered in the project in order to allow Odin II to support all for loops no matter how they expresses.

2. AST optimization

The current work of optimizing AST is to reduce some branches that can be pruned. Figure 3 shows an example of assignment expression, which can be calculated.

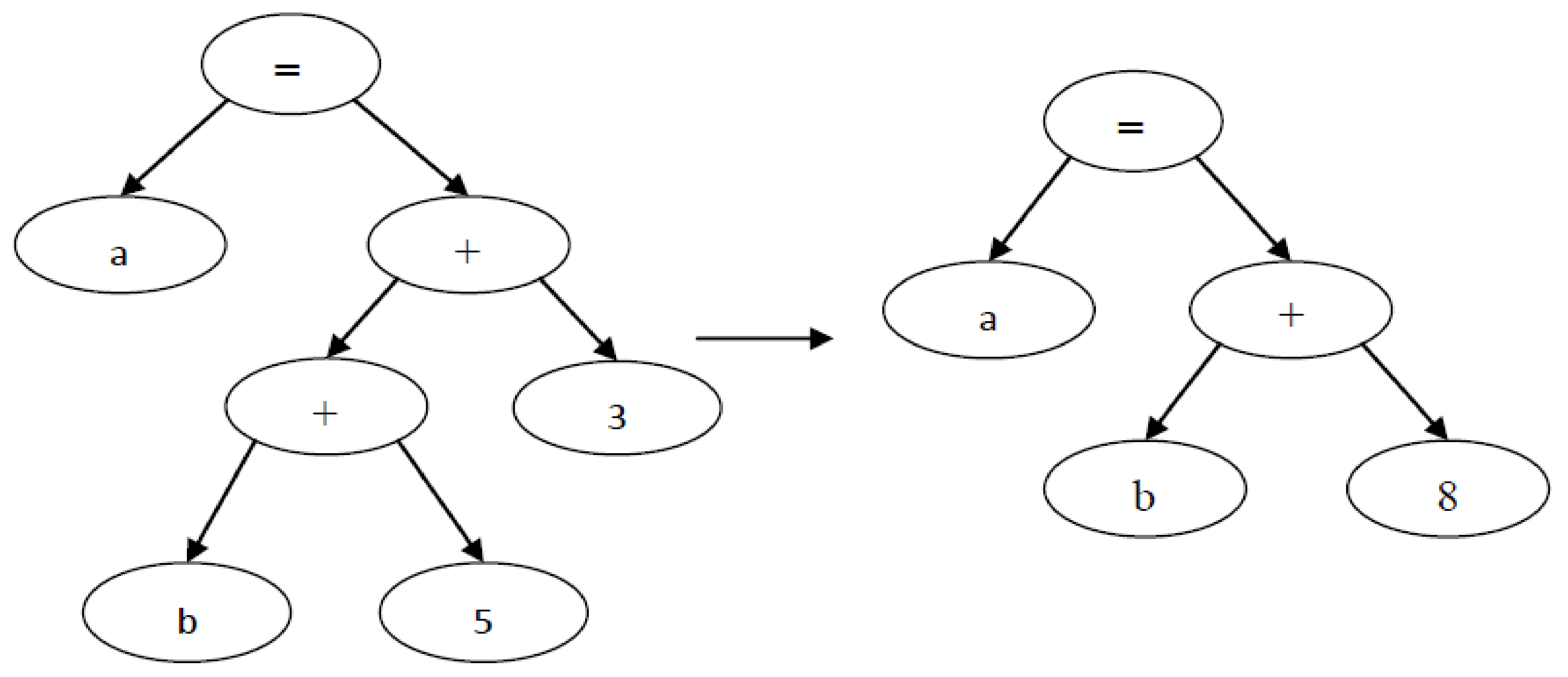

Figure3. An Example of AST Reduction

The example above shows the basic idea of how to reduce an AST. If there are some parts of branches representing expressions, from which we can calculate the results, such parts will be pruned and replaced by the result node.

Result

Currently, Odin II can support simple for loops, and the work is continued to allow all for loops, no matter how complicated. When this work is finished, the functionality of Odin II is more complete. AST optimizations can reduce the structure of the AST and simplify the netlist and BLIF afterward. The BLIF will be sent to the next work flow tool as input, and because of the optimization the whole work flow will be more efficient.