Design/Practical Experience [CSN1020]
Department of Computer Science and Engineering

Final Report

Academic Year: 2022-2023
Semester: 5
Date of Submission of Report: 03 Jan 2023
Title: Datapath Extractor from Hardware Design Descriptions

Authors: Risheek Nayak (B20AI058), Suyash Jaiswal (B20EE070), Shashank Kumar (B20EE063)

Mentor's Name:
Binod Kumar
Assistant Professor in the Electrical Engineering department at IIT Jodhpur (IITJ)

Abstract:

Given a flattened Verilog netlist that contains only primitive gates, we have written a program to extract the datapaths (arithmetic operations) from the netlist.

OUR APPROACH:

Work Done:

1. **Converting the Verilog netlist to a Graph format for simulation.**
   a. The nodes of the graph contain the following information:
      i. Name of the gate.
      ii. Type of the gate.
      iii. The in-degree of the node.
      iv. The names of the incoming gates to the node.
      v. The names of the input gates if any.

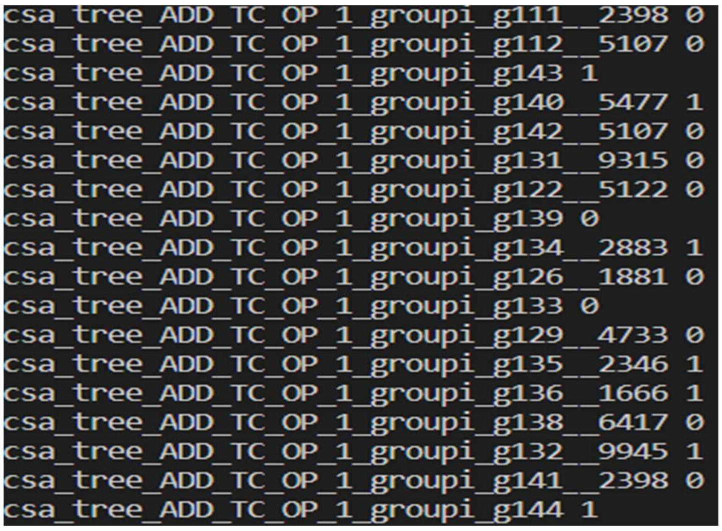
   b. Function make_wire_gate_pair:
      i. This function reads the Verilog file.
      ii. For each line in the file, it maps the gate's output wire name to the gate's name.

c. Function construct_circuit:
   i. Constructs the Verilog circuit into a graph format.
   ii. Now we can traverse the graph both from children to parent and from parent to children. This will help us to simulate the circuit.

d. Function output_value:
   i. An output boolean value is generated depending on the type of gate (XNOR, XOR, AND, NOR, OR, BUF, NOT) and the input to the gate.

e. Function simulation:
   i. Topological traversal is implemented.
   ii. Stores output of each and every gate.
   iii. Stores the name of gates per level.

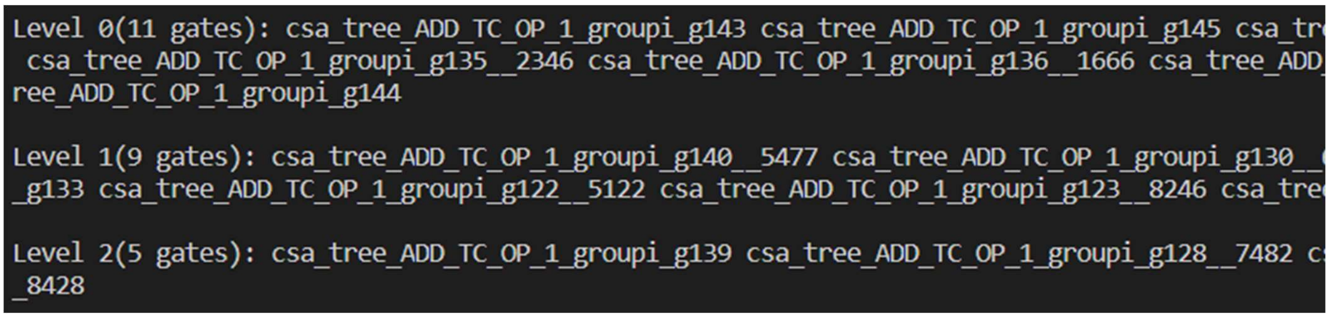
f. Function all_inputs:
   i. Generates all the possible inputs for the input wires.

g. Function print_gate_with_output_value:
   i. Prints all the gates with their respective output values.
   ii. 

```
csa_tree_ADD_TC_OP_1_groupi_g111__2398 0
csa_tree_ADD_TC_OP_1_groupi_g112__5107 0
csa_tree_ADD_TC_OP_1_groupi_g143 1
csa_tree_ADD_TC_OP_1_groupi_g140__5477 1
csa_tree_ADD_TC_OP_1_groupi_g142__5107 0
csa_tree_ADD_TC_OP_1_groupi_g131__9315 0
csa_tree_ADD_TC_OP_1_groupi_g122__5122 0
csa_tree_ADD_TC_OP_1_groupi_g139 0
csa_tree_ADD_TC_OP_1_groupi_g134__2883 1
csa_tree_ADD_TC_OP_1_groupi_g126__1881 0
csa_tree_ADD_TC_OP_1_groupi_g133 0
csa_tree_ADD_TC_OP_1_groupi_g129__4733 0
csa_tree_ADD_TC_OP_1_groupi_g135__2346 1
csa_tree_ADD_TC_OP_1_groupi_g136__1666 1
csa_tree_ADD_TC_OP_1_groupi_g138__6417 0
csa_tree_ADD_TC_OP_1_groupi_g132__9945 1
csa_tree_ADD_TC_OP_1_groupi_g141__2398 0
csa_tree_ADD_TC_OP_1_groupi_g144 1
```

   Name of the gate Output_Value

h. Function print_gate_with_level:
   i. Prints all the gates level-wise.



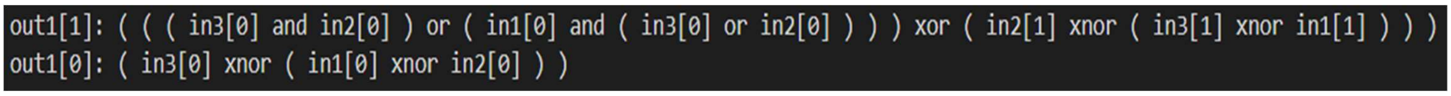
```
Level 0(11 gates): csa_tree_ADD_TC_OP_1_groupi_g143 csa_tree_ADD_TC_OP_1_groupi_g145 csa_tr
 csa_tree_ADD_TC_OP_1_groupi_g135__2346 csa_tree_ADD_TC_OP_1_groupi_g136__1666 csa_tree_ADD
ree_ADD_TC_OP_1_groupi_g144

Level 1(9 gates): csa_tree_ADD_TC_OP_1_groupi_g140__5477 csa_tree_ADD_TC_OP_1_groupi_g130__
_g133 csa_tree_ADD_TC_OP_1_groupi_g122__5122 csa_tree_ADD_TC_OP_1_groupi_g123__8246 csa_tre

Level 2(5 gates): csa_tree_ADD_TC_OP_1_groupi_g139 csa_tree_ADD_TC_OP_1_groupi_g128__7482 c
_8428
```

2. **Generating the longest possible RTL:**
    a. Function reverse:
        i. It will reverse the key with values of the Wire_gate map and will create a new map.

    b. Function relation_wire:
        i. It will iterate the netlist level-wise.
        ii. It takes a string as a parameter and generates the output on the basis of the gates and wires of the previous level.
        iii. Returns the string which is the expression of level-wise outputs.
    c. Function logic:
        i. It will create a map that has keys as wires and their corresponding value as expressions from the wires and gates through which that output was generated.

    d. Function form:
        i. It will iterate on the wires level-wise and go on storing the expressions and replacing the terms which had already been stored in the map. For each level, we need outputs in the form of the first level in this way we will get the output of the last level in the form of the first.
        ii. For example, for level 2 we have wires in form of the first level and for level 3 we have wires in the form of level 2. Now we have an expression for level 2 so we will substitute that here so that we get the output of level 3 in the form of inputs of level 1. In this way, we get output at level n in the form of inputs of level 1.

e. Example:



```
out1[1]: ( ( ( in3[0] and in2[0] ) or ( in1[0] and ( in3[0] or in2[0] ) ) ) xor ( in2[1] xnor ( in3[1] xnor in1[1] ) ) )
out1[0]: ( in3[0] xnor ( in1[0] xnor in2[0] ) )
```

3. **Cross Checking whether the longest expression is correct or not:**
    a. Function gates:
        i. It will take the input of two numbers and a string.
        ii. According to the string, it will check the gate and return the output.

    b. Function evalueate_string:
        i. This function uses stack data structure to evaluate the outputs as there are opening and closing brackets it cannot be done by simple iteration.
        ii. We will iterate on a string (expression) and will go on solving subproblems and store their result in the stack and go on solving that to generate the final output and verify this result from the simulator.

Analysis:

Generating the datapath for different test cases and noting down the time it takes to run:

TestCase Number	Size of Testcase (kb)	Time Taken (sec)
1.	6	0.002497
2.	9	0.004916
3.	47	0.011221
4.	198	0.029502
5.	788	1.18814
6.	640	2.71408
7.	194	0.141729
8.	390	5.40488

9.	156	0.29865
10.	57	0.025475
11.	10	0.015913
12.	453	13.303
13.	32	0.007914
14.	40	0.023264
15.	4	0.004028
16.	88	0.018961
17.	226	1.24794

Declaration:
We declare that no part of this report is copied from other sources.