Report - Analysis on for and while loop definition in Verilog

Author

- Chathura Gunaekara (200193U)

Introduction

In this report, we will discuss the differences between for and while loop in Verilog. This will be done by analyzing two implementations of a module using both for and while loop. The code for the two implementations is given below:

While Loop Implementation

```
module what_while(
 input [15:0] in,
 output reg [4:0] out);
 integer i;
 always @(*) begin: count
     out = 0;
     i = 15;
     while (i >= 0 && ~in[i]) begin
         out = out + 1;
         i = i - 1;
     end
  end
endmodule
```

For Loop Implementation

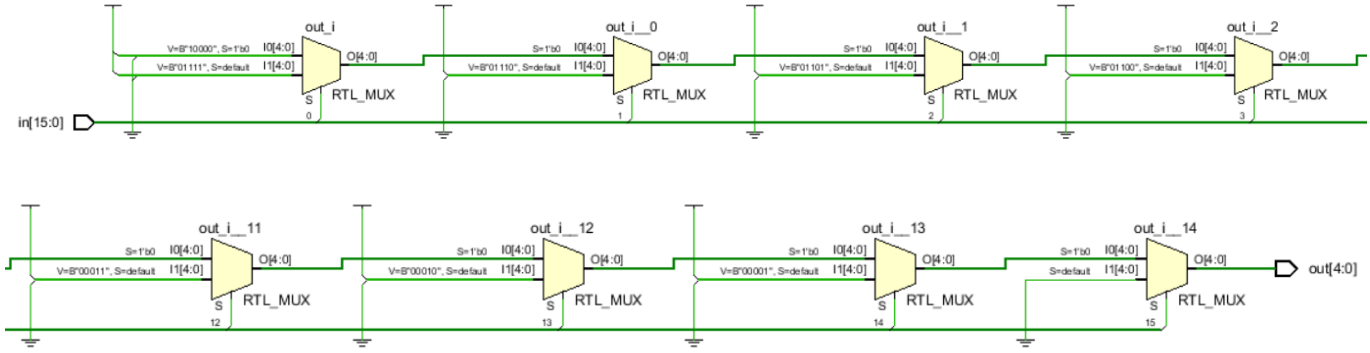
```
module what_for(
 input [15:0] in,
 output reg [4:0] out
 );
 integer i;
 always @(*) begin: count
 out = 0;
     for (i = 15; i >= 0; i = i - 1) begin
         if (~in[i]) disable count;
         out = out + 1;
     end
 end
endmodule
```

Eventhough the implementations are done using loops, they are synthesized statically. This is done by **unrolling the loop and creating a combinational circuit**. The main difference between the two implementations is the way the loop is defined.

RTL Schematic Analysis

The RTL schematic for the two implementations is shown below:

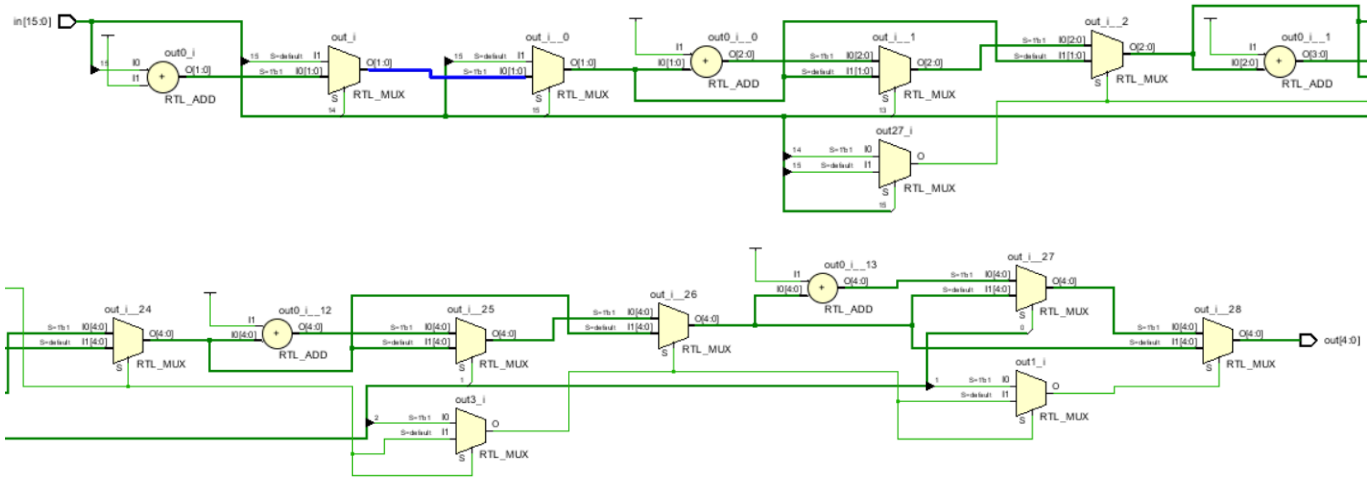
While Loop RTL Schematic



This image shows the starting and ending segments of the rtl synthesis of the while loop implementation. Here we can clearly see the loop unrolling leading to the implementation of 15 multiplexor units. The multiplexor chooses between 2 signals:

- The `0` signal corresponds to `B'10000'` (16 in decimal), which corresponds to the output for an input with all zeros. - The `1` signal in at each level corresponds to the sum upto that level. Hence, if in[10]='1' then the output will be `B'01011'` (11 in decimal).

For Loop RTL Schematic



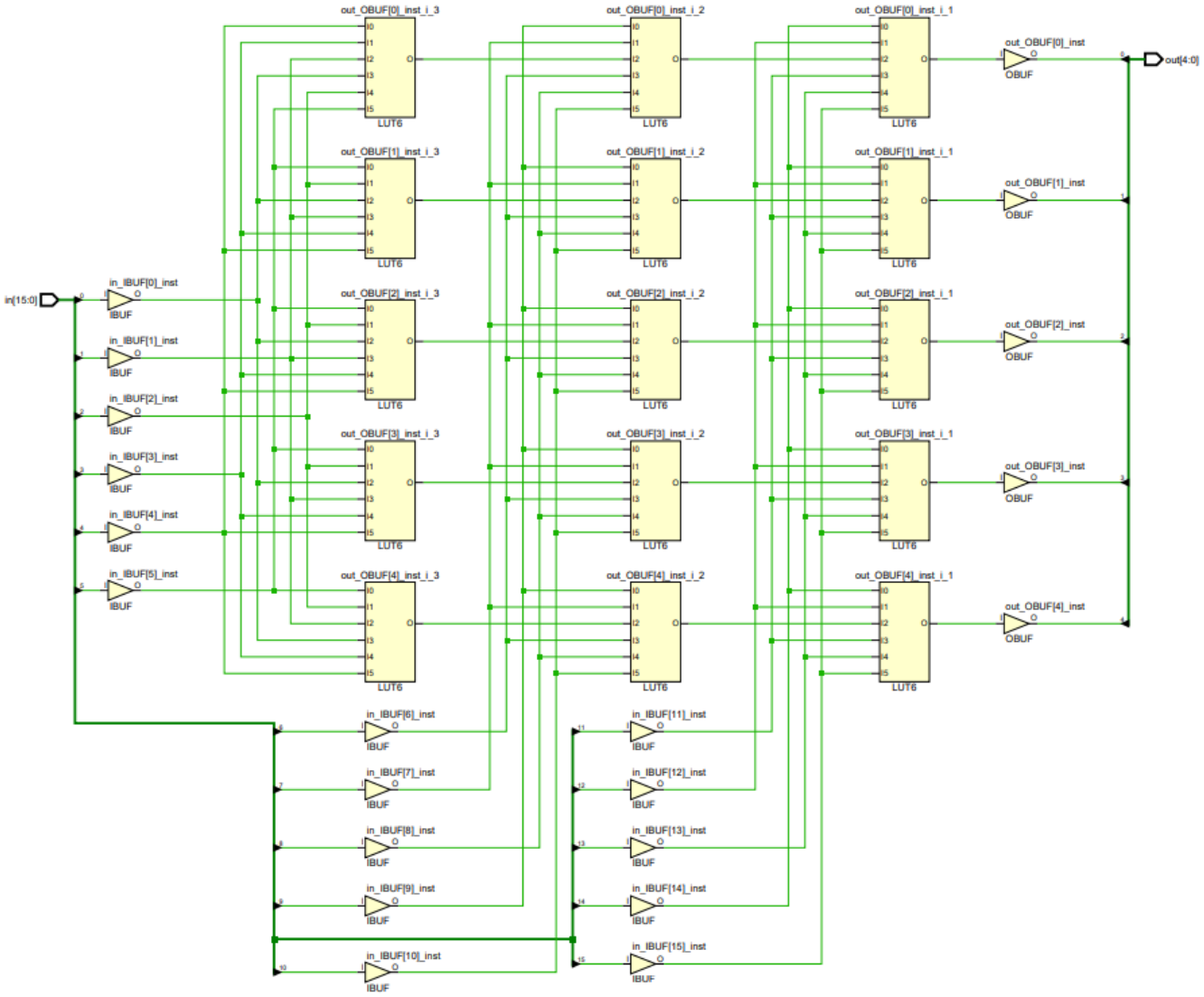
This image shows the starting and ending segments of the rtl synthesis of the for loop implementation. Unlike the while loop implementation which had a simple loop unrolling, a more complex strategy is used in the for loop implementation. The high level idea still incorperates the use of levels corresponding to the stages of the loop.
The conditional `if (~in[i]) disable count`; is implemented with MUX components that conditionally bypass the addition based on bit values. At each stage, the circuit either propagates the accumulated count

forward or halts counting by passing a fixed count if a `in[i] == 1` is encountered. This is done by the combination of MUX and ADD components, structured similarly to a pipeline. The final count accumulation corresponds to the sequential adder outputs in the RTL schematic, creating a hardware-based approach to counting.

These are the major differences between the two implementations.

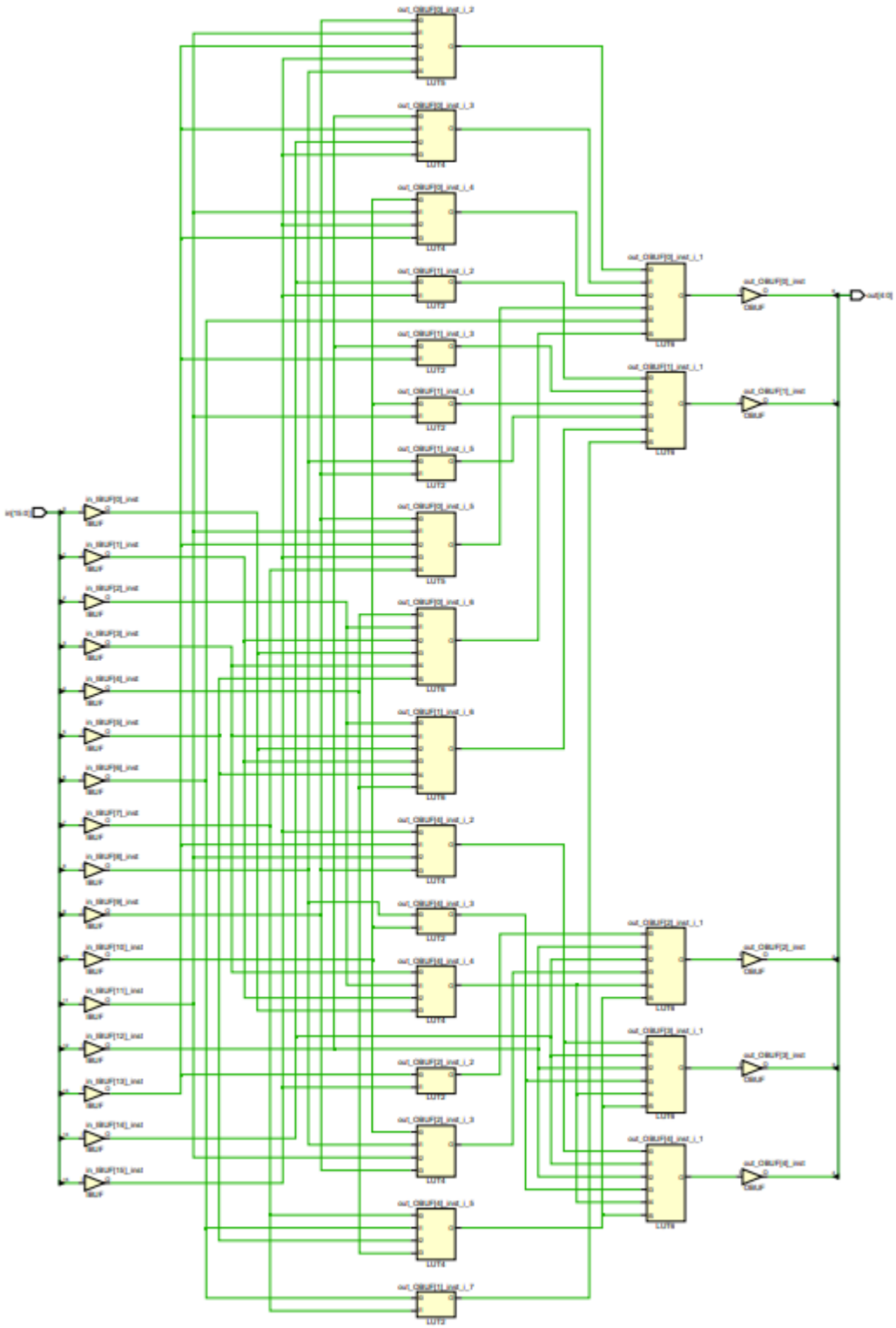
Implementation Analysis

While Loop Implementation Schematic



This schematic shows that the while loop implementation is constructed in RTL in a layered approach with 3 layers of 5 LUTs each.

For Loop Implementation Schematic



It can be observed in the code that the `disable` term is used. This is used to break out of the loop when the condition is met. This is a feature of the for loop in Verilog [Reference]. This is not part of the Synthesizable code. (i.e. a register cannot be removed when you are not using it).

Timing Analysis

Critical Path Analysis

Before the logic optimization, it is required to identify the critical paths in the design. This can be done by performing a static timing analysis. This can be done by running the synthesis tool with the timing constraints

relaxed and considering the consider the **setup time** and **hold time**. This will give a clear picture of the worst case scenario and the total delay.

For Loop Implementation

- Setup total delay - `in[4] -> out[2] : 8.635 ns`
- Hold total delay - `in[12] -> out[3] : 2.230 ns`

While Loop Implementation:

- Setup total delay - `in[5] -> out[1] : 9.217 ns`
- Hold total delay - `in[12] -> out[3] : 2.060 ns`

Timing Constraints Tightening

After identifying the critical paths, the timing constraints can be tightened to improve the performance of the design. This can be done by reducing the slack in the design. This can be done by setting 2 constraints:

- Set the **maximum delay** of all inputs to the setup total delay
- Set the **minimum delay** of all inputs to the hold total delay

For Loop Implementation:

Setup

Worst Negative Slack (WNS):	0.000 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	5

Hold

Worst Hold Slack (WHS):	0.000 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	5

All user specified timing constraints are met.

- Setup

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	0.000	4	3	3	in[4]	out[2]	8.635	4.024	4.610	8.6	input port clock
Path 2	0.023	4	3	3	in[4]	out[4]	8.612	4.017	4.595	8.6	input port clock
Path 3	0.051	4	3	7	in[15]	out[0]	8.584	3.985	4.599	8.6	input port clock
Path 4	0.147	4	3	3	in[4]	out[3]	8.488	4.025	4.463	8.6	input port clock
Path 5	0.728	4	3	6	in[11]	out[1]	7.907	4.013	3.894	8.6	input port clock

- Hold

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 6	0.000	3	2	5	in[12]	out[3]	2.230	1.339	0.891	2.2	input port clock
Path 7	0.046	3	2	5	in[12]	out[4]	2.276	1.331	0.944	2.2	input port clock
Path 8	0.054	3	2	3	in[6]	out[0]	2.284	1.334	0.950	2.2	input port clock
Path 9	0.069	3	2	5	in[14]	out[2]	2.299	1.330	0.969	2.2	input port clock
Path 10	0.133	4	3	3	in[5]	out[1]	2.363	1.370	0.993	2.2	input port clock

While Loop Implementation:

Setup		Hold	
Worst Negative Slack (WNS):	0.000 ns	Worst Hold Slack (WHS):	0.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	5	Total Number of Endpoints:	5

All user specified timing constraints are met.

- Setup

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	0.000	5	4	5	in[5]	out[1]	9.217	3.906	5.311	9.2	input port clock
Path 2	0.062	5	4	5	in[0]	out[4]	9.155	3.902	5.253	9.2	input port clock
Path 3	0.127	5	4	5	in[0]	out[3]	9.090	3.910	5.180	9.2	input port clock
Path 4	0.167	5	4	5	in[1]	out[0]	9.050	3.902	5.148	9.2	input port clock
Path 5	0.417	5	4	5	in[2]	out[2]	8.800	3.908	4.892	9.2	input port clock

- Hold

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 6	0.000	3	2	5	in[13]	out[0]	2.060	1.317	0.744	2.1	input port clock
Path 7	0.035	3	2	5	in[12]	out[1]	2.095	1.337	0.759	2.1	input port clock
Path 8	0.062	3	2	5	in[13]	out[4]	2.122	1.312	0.810	2.1	input port clock
Path 9	0.064	3	2	5	in[13]	out[2]	2.124	1.319	0.805	2.1	input port clock
Path 10	0.066	3	2	5	in[13]	out[3]	2.126	1.320	0.807	2.1	input port clock

Tightening the timing constraints beyond the critical paths would result in a violation of the timing constraints as it causes **negative slack**