EE332 Lab2: Simulation of Full Adder on Nexys 4 DDR

1st Qiu Kunyuan
EEE. Southern University of Science and Technology
Shenzhen, PRC
11913019@mail.sustech.edu.cn

***Abstract*—This report focuses on the phenomenons encountered in the simulation of a simple full adder, to reveal some critical features of FPGA programming comparing to ordinary programmable devices. Among these phenomenons, jitters and the race-hazard conditions are of the most concerns.**

Code for this experiment can be found at https://github.com/KagaJiankui/EE332-2024S/tree/master/lab2/lab2.srcs.

***Index Terms*—FPGA, programmable logic, full adder, race-hazard condition, jitter, delay**

I. Introduction

Half adder and full adder are basic elements in digital circuits used to perform addition operations on binary numbers.

A *half adder*

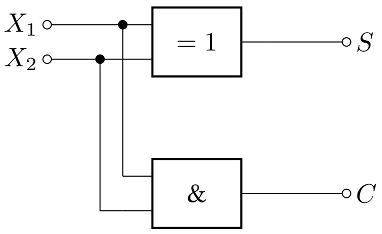


Fig. 1: Gate Level Description of Half Adder

is a combinational logic circuit shown in Figure.(1) that takes two binary inputs and produces two binary outputs: sum and carry.

$$\text{HA}:(x_1,x_2)\mapsto(Q,C):=\begin{cases}Q &= \text{xor}(x_1,x_2)\\ C &= \text{and}(x_1,x_2)\end{cases} \tag{1}$$

The main limitation of the half adder is that it cannot handle carry inputs and therefore can only be used for 1-bit addition.

A *full adder* is an extension of the half adder that accepts three binary inputs: two additions and a rounding input, and produces two binary outputs: sum and carry. Full adders can be connected in series to achieve binary addition of any number of bits.

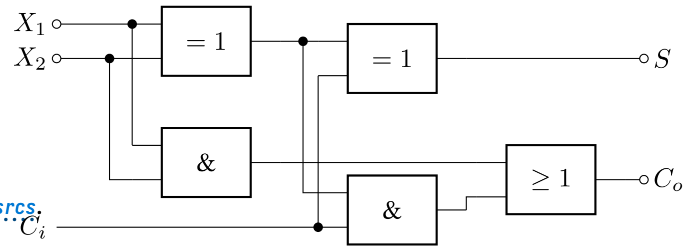


Fig. 2: Gate Level Description of Full Adder

Implementation of full adder can be derived directly from 3-digit addition, where the final carry output is toggled if any of X_1+X_2 or C_i+Q_1 produces a carry:

$$\begin{aligned}(q_1,c_1) &= \text{HA}(x_1,x_2) &(2)\\ (q_2,c_2) &= \text{HA}(q_1,c_i) &(3)\\ c_o &= \text{or}(c_1,c_2) &(4)\end{aligned}$$

Therefore the gate-level circuit of the full adder can be easily captured, as shown of Figure.(2)

II. Verilog Modeling

There are two levels of HDL modeling for full adder, one for gate level description and one for behavioral level description.

A. Gate-Level Implementation

```verilog
`timescale 1ns / 1ps



module full_adder_g (
    input  wire c0,
    input  wire a,
    input  wire b,
    output wire s,
    output wire c1
);

  assign s  = ((~c0) & (~a) & b) | ((~c0) & a & (~b
      )) | (c0 & (~a) & (~b)) | (c0 & a & b);
  assign c1 = (c0 & a) | (a & b) | (c0 & b);

endmodule
```

Listing 1: Gate Level Modeling of Full Adder in Verilog

For the gate level modeling, the DNF (OR-AND) of the output ports are required:

$$\begin{aligned} Q_2 &= HA(HA(x_1, x_2)[0], c_i)[0] \\ &= \text{xor}(\text{xor}(x_1, x_2), c_i) \\ &= \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC \\ C_{\text{out}} &= AB + BC + AC \end{aligned} \tag{5}$$

Therefore, the Verilog code is pretty straight forward, as the Code.(3). For the Vivado elaboration and synthesis process, the logic functions are expanded to its DNF, meaning only AND/OR/NOT gates remains in the elaborated schematic:

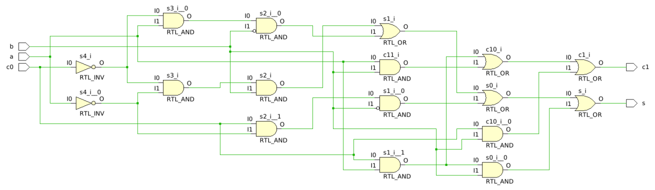


Fig. 3: Elaborated Gate Level Description

B. LUT Implementation

```
`timescale 1ns / 1ps



module full_adder_lut (
  input  wire [2:0] in_1,
  output wire [1:0] out_1
);

  assign out_1 = in_1[2] + in_1[1] + in_1[0];

endmodule
```

Listing 2: Gate Level Modeling of Full Adder in Verilog

The combine logic functions are implemented by LUT(Look-Up Table)s inside the FPGA components in real world, while the half adder is a common LUT element inside the CLBs. By simply configuring the switchable connections between the CLBs and the controlling MUXs inside the CLBs, a cascaded design can be easily obtained.

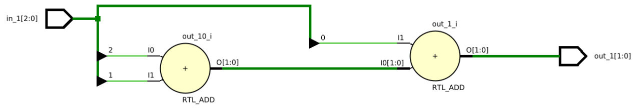


Fig. 4: Elaborated RTL Level Description

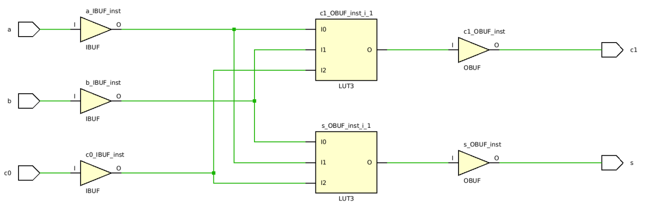


Fig. 5: Synthesized Design

X_1	X_0	C_i	dec($in[2:0]$)	C_o	S	dec($out[1:0]$)
0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	0	2	0	1	1
0	1	1	3	1	0	2
1	0	0	4	0	1	1
1	0	1	5	1	0	2
1	1	0	6	1	0	2
1	1	1	7	1	1	3

TABLE I: Truth Table of Full Adder

C. Synthesized Implementation

Either the gate level design (3) or the RTL level design (4) are synthesized into the same schematic (5) after the synthesis and implementation process, since the FPGA device only provide configurable LUT as the measure to combinational logic inside the CLB.

In the synthesized schematic (5), the two LUT cells are combinational logic functions of $S = \text{LUT}_1(X_0, X_1, C_i)$ and $C_o = \text{LUT}_2(X_0, X_1, C_i)$ respectively.

III. Simulation

The truth table of the full adder is Table.(I). For the design using gate-level description, simulation result is shown in the waveform figure corresponding to each stage:

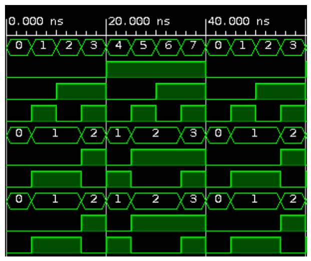
- Figure.6(a): **behavioral** simulation after **elaboration**
- Figure.6(b): **timing** simulation after **synthesis**
- Figure.6(c): **timing** simulation after **implementation**

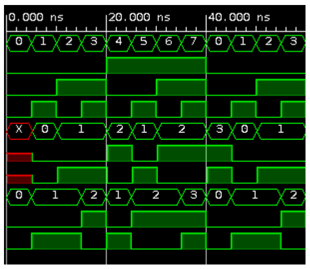
Additionally, the timing simulation result of the design using RTL block-level description after implementation is also provided in Figure.(7), since the only difference between the block-level version and the gate-level version is the routing after implementation.

Vivado only accepts one single top file in one synthesis/implementation batch, resulting that only one module can be accurately simulated at implemented level, while the others remains behavioral level. In the following discussions, the behavioral level trail are considered as a marking for comparison.

A. Consistency

Comparing result Figure.6(a) from both gate-level and RTL-level designs with the truth table Table.I, it is easy to identify that the behavioral simulation result is consistent with the truth table. Therefore the consistency is verified.


(a) Behavioral Simulation


(b) Timing after Synthesis

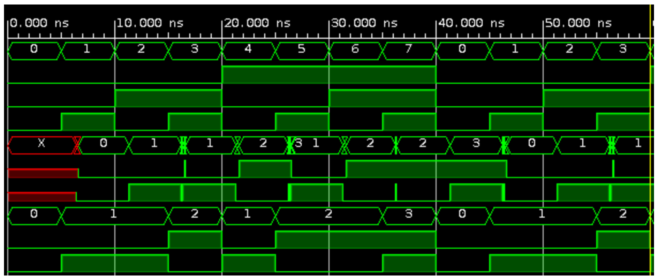

(c) Timing after Implementation

Fig. 6: Simulation Results

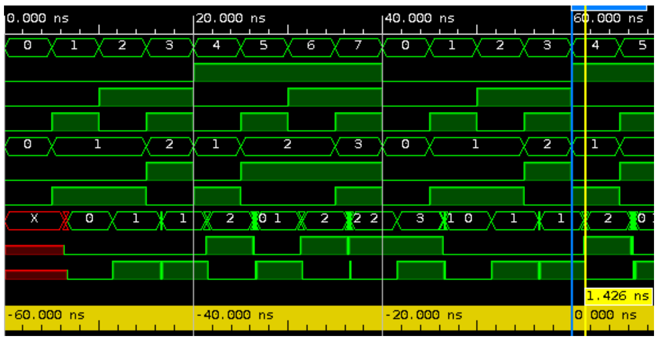


Fig. 7: Timing Simulation of RTL Block-level Design

However, there exists one difference between the behavioral simulation result and the waveforms shown in lecture notes, that the result from Vivado simlulation returns its output immediately after the initial Hi-Z undeterministric state, while the result in lecture notes shows that there is additional delay from the end of Hi-Z state to the first sensible output. The reason to this phenomenon can be the optimizations done by Vivado elaborator.

B. Delay

Compare the other 2 simulation results with the behavioral simulation result, the initial undetermined state is conspicuous since it is marked in eye-catching red highlights, while there is no undetermined regions or time domain offsets in behavioral simulation result suggesting the existence of delays.

Additionally, the delay of gate-level design after implementation is slightly higher than that of an RTL-level design, by comparing the time the system remains undetermined state. The reason to this phenomenon can be the different path length between CLBs used in the different design, and the RTL-level design using half adder suggests the elaborator and synthesizer to route a path with lower distance.

C. Jitters

Contrast the position of the jitter in the output waveform with the edge of the input waveform on the time axis, the moments when the output waveform has a jitter are easily found to correspond to the moments when both the rising and falling edges of the multiple input waveforms are present plus a gate delay.

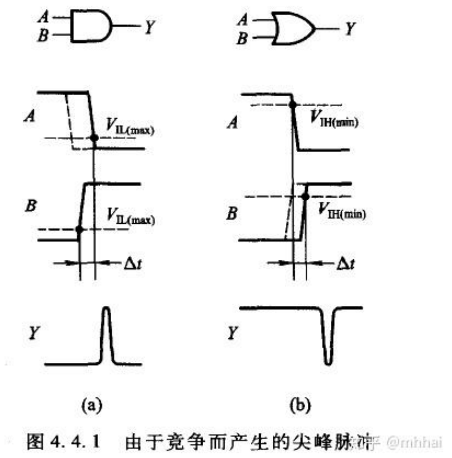


Fig. 8: Explanation of Race Conditions

By referencing the schematic of the synthesized design, there only exists one single element between the multiple inputs with race conditions and both of the output port, thus the delay between the emergence of race condition 8 and the spikes in output is the unit delay of one single LUT element.

Appendix. Testbench

```
`timescale 1ns / 10ps



module FA_tb;

  reg  [2:0] data;
  wire [1:0] out_g;
  wire [1:0] out_lut;

  full_adder_g fag (
      .c0(data[0]),
      .a (data[1]),
      .b (data[2]),
      .s (out_g[0]),
      .c1(out_g[1])
  );

  full_adder_lut fa_lut (
      .in_1 (data),
      .out_1(out_lut)
  );

  initial begin
    data=3'b000;
  end

  always begin
    #5;
    if(data>7) data=0;
    data=data+1;
  end

endmodule
```

Listing 3: Testbench