Project 6 Delays and Glitches

Signal propagation delays through combinational circuits

Introduction

So far, we have assumed that the circuit nodes in a digital circuit can switch state instantaneously, transitioning **from 0 -> 1 or 1 -> 0** with no elapsed time. But this is not really the case – **some amount of time is always required for any circuit node to change state.** When a digital signal passes through a logic gate, it switches transistors on or off to drive the output node to a given state. The output circuit node (like all circuit nodes) has some amount of capacitance, and that capacitance must be charged or discharged for the node to change state. When the output changes state, the charge trapped on the output node must flow to one of the power rails, and this takes time. The time required to **charge or discharge output nodes** can be lumped into a single "**gate delay**". This project examines the sources of such delays, their possible detrimental effects, and what actions can be taken to minimize their impact.

Before you begin, you should:

- Know Vivado and how to use the Boolean Board;
- Know how to design and implement logic circuits based on truth tables;
- Be able to describe a digital circuit in Verilog;
- Know how to write and run Verilog test benches.

After you're done, you should:

- Understand how hazards and glitches are formed;
- Know how to simulate a circuit with delays to illustrate glitches and other timing issues;
- Know how to modify a circuit to remove glitches.

Requirements

☑ 1. Illustrate the formation of a glitch in the simulator.

Follow the tutorial to simulate and illustrate the formation of a glitch:

Step 1: Implement the Circuit in Verilog

In this project, you are going to simulate a circuit in Verilog, taking delay into consideration. The circuit schematic is shown in Fig. 1, and the delay of each gate is marked in red.

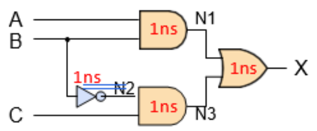


Figure 1. Circuit Diagram

Declare the Combinational Circuit Module

The circuit shown in Figure 1 has three inputs `A`, `B`, and `C`, and one output `X`. So, the declaration of the module goes as follows:

```
module CombCirc(
   input A,
   input B,
   input C,
   output X
   );

// Circuit Description

endmodule
```

Implement Circuit with Delay

The goal of this project is to simulate the delay of logic gates and analyze its effect on the circuit behavior. So, you need to describe each logic gate using an `assign` statement with a delay command as shown in the code block below.

```
wire N1, N2, N3;

// AND gate with 1ns delay
assign #1 N1 = A & B;
// Not Gate with 1ns delay
assign #1 N2 = ~B;
// And Gate with 1ns delay
assign #1 N3 = N2 & C;
// Or Gate with 1ns delay
assign #1 X = N1 | N3;
```

In the codes above, `N1`, `N2` and `N3` are three internal signals as labeled in Figure 1. Each gate has a delay of 1 reference time unit. In order to make the delay 1ns for each gate, you need to define the reference time unit as 1ns using the command `'timescale` at the beginning of the file.

```
`timescale 1ns / 1ps
module CombCirc(
...
```

Check Your Source File

So the Verilog file that describes the circuit with delay information of each gate looks as follows:

```
`timescale 1ns / 1ps
module CombCirc(
    input A,
    input B,
    input C,
    output X
    );
// Circuit Description
wire N1, N2, N3;

// AND gate with 1ns delay
assign #1 N1 = A & B;
// Not Gate with 1ns delay
assign #1 N2 = ~B;
// And Gate with 1ns delay
assign #1 N3 = N2 & C;
// Or Gate with 1ns delay
assign #1 X = N1 | N3;
endmodule
```

Step 2: Create the Test Bench and Simulate the Circuit

In order to simulate a **glitch**, you will need to generate an input sequence in your testbench that can cause the glitch to appear at the output of the circuit. By observing the circuit, there is an unbalanced path between input `B` and output `X` (i.e., there are two paths to propagate the changes of `B` to the output with different delays). So, the glitch will happen when `A` and `C` are held constant and `B` is toggled. The codes below generate a sequence where `B` changes from 1 to 0 and back to 1 when `A` and `C` are held constant. Do not forget to instantiate the `CombCirc` module in your test bench and connect the inputs and outputs to internal signals!

```
integer k = 0;

initial begin
    // Initialize Inputs
    A = 0;
    B = 0;
    C = 0;

    // Wait for 100 ns for the global reset to finish
    // Add stimulus here

    for(k = 0; k < 4; k=k+1)
    begin
        {A,C} = k;
        #5 B = 1;
        #5 B = 0;
        #5 ;
    end
end
```

Simulate the test bench

Now, you can simulate your test bench in the Vivado simulator, and you will get the waveform as shown in Figure 2 below. The red circle on the waveform specifies the glitch. It happens when `A` is 1, `C` is 1 and `B` changes from 1 to 0. The duration of the glitch is 1ns.

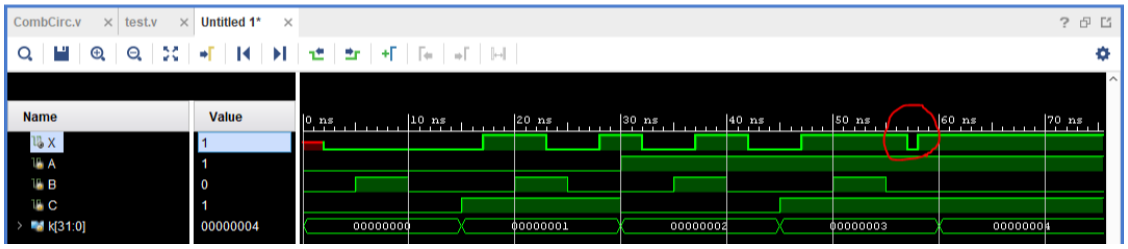


Figure 2. Simulation Waveform of the Circuit

☑ 2. Change the OR Gate Delay and resimulate

Assume the OR gate in the previous circuit has a propagation delay of 2 ns. Simulate the circuit again and try to find the glitch. If there is a glitch, when does it occur and what is the duration of the glitch? If there is no glitch, can you explain why? Think about the differences in path delay for signal B. Also, think about how the Vivado Simulator works.

☑ 3. Change the Delay of All gates and resimulate

Assume the delay of all gates is 5 ns. Modify the `CombCirc` module and the test bench properly and try to find the glitch. If there is a glitch, when does it occur and what is the duration of the glitch? If there is no glitch, can you explain why?

Latches and Flip-Flops

The basic memory elements used in sequential circuits

Introduction

This project introduces the concept of electronic memory in digital circuits. Digital circuits need memory in order to create ordered sequences of events, like playing one note after another to reproduce a digitized song or to track and respond to sequences of events, like ensuring a particular sequence of button presses is used to open a digital combination lock.

Most digital circuits use electronic memory. Electronic memory circuits store the state of an input signal at some particular time, and their memorized outputs (or data) can be used at a later time. Digital circuits that use electronic memory are called **sequential** circuits because they can use memorized data to create or sense ordered sequences of events. In contrast, **combinational** circuits do not use memory, so they can only change their outputs to reflect the current state of their inputs. Creating sequences is fundamental to all of computing, and many 100's of billions of computing circuits are in use today.

This project presents a common memory device used in digital circuit design - the D-latch. It memorizes a single input when triggered by a timing signal, and both have a single output that always shows the state of the memorized signal. The response of the D-latch to the timing signal allows an input signal to flow into and through it whenever the timing signal is asserted.

Before you begin, you should:

- Have a good amount of experience using Vivado and the Boolean Board;
- Be able to write and execute a Verilog testbench;
- Know how to model and simulate circuit delays.

After you're done, you should:

- Understand the operation of an S-R latch;
- Understand the cause of metastability;
- Understand the operation of latches;
  Be able to describe memory circuits in behavioral Verilog.

Requirements

☑ **1. Implement and simulate a NAND basic cell**

Complete the tutorial:

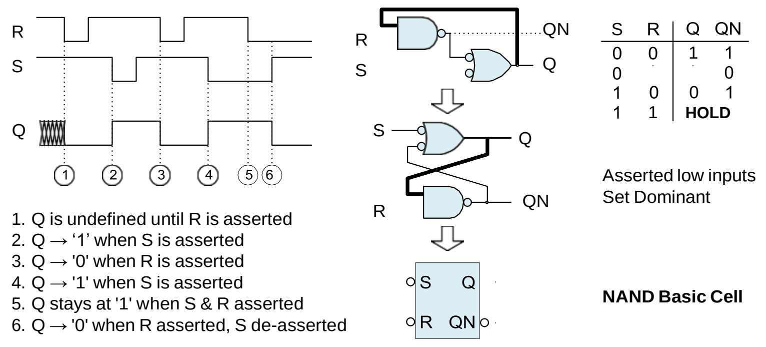
SR-Latch Tutorial

Latches and Flip-Flops

Introduction

Latches are the fundamental bi-stable memory circuit in digital systems to store data and indicate the state of the system. In this project, you are going to simulate the basic NAND cell of an SR-Latch and see how it functions.

Figure 1 shows an implementation for SR-Latch with NAND implementation. According to the truth table on the right, S and R are active low. When only S is asserted (S is '0'), the output Q is SET to '1'. When only R is asserted (R is '0'), the output Q is RESET to '0'. When neither S nor R is asserted, the output holds its previous value.



1. Q is undefined until R is asserted
2. Q → '1' when S is asserted
3. Q → '0' when R is asserted
4. Q → '1' when S is asserted
5. Q stays at '1' when S & R asserted
6. Q → '0' when R asserted, S de-asserted

S	R	Q	QN
0	0	1	1
0			0
1	0	0	1
1	1	**HOLD**	

Figure 1. SR-Latch NAND cell

SR-Latch is a kind of bi-stable circuit. However, due to the propagation delay of the NAND gate, it is possible to drive the circuit intoametastable state, where the output is oscillating between 0 and 1. The metastable state will be triggered when neither the set operation nor the reset operation propagates through the whole cell before the cell changes to the hold state.

Step 1: Implement the Circuit in Verilog

Assume the NAND gates in SR-Latch have a delay of 1 ns. You may want to try to create the Verilog file without looking at the example code, you can write the code based on figure 1 NAND cell and then check your work by checking the code.

The Verilog file for the SR-Latch looks like follows:

```
`timescale 1ns / 1ps
module sr_latch(
    input S,
    input R,
    output Q,
    output Qn
    );

wire Q_int, Qn_int;

assign #1 Q_int = ~(S & Qn_int);
assign #1 Qn_int = ~(R & Q_int);
assign Q = Q_int;
assign Qn = Qn_int;

endmodule
```

Step 2: Create a Test Bench for the AND Cell SR-Latch

For the purpose of demonstrating the functionality of SR-Latch, we consider the following input stimulus:

Time	**Description**
0ns	De-assert both inputs
100ns	Assert `S`
200ns	De-assert `S`
300ns	Assert `R`
400ns	De-assert `R`
500ns	Assert both inputs
600ns	De-assert both inputs
700ns	Assert both inputs

```
initial begin
    // Initialize Inputs
    S = 1;
    R = 1;

    // Add stimulus here
    #100 S = 0;
    #100 S = 1;
    #100 R = 0;
    #100 R = 1;
    #100 S = 0;
         R = 0;
    #100 S = 1;
         R = 1;
    #100 S = 0;
         R = 0;
    #100 ;
end
```

Examine the output of your simulation (and Figure 2 below shows a similar simulation). In the simulation below, the reset signal is asserted at 300ns (the zoomed-in graph on the left shows a more detailed view). Note that it takes 2 ns for the reset signal R to propagate through the NAND cell (this is due to the 1ns gate delays you added into your simulation). Then at 600 ns, both set (S) and reset toggle from 0 to 1 at exactly the same time. When S and R are deasserted exactly simultaneously, the basic cell tries to enter the hold condition from the ambiguous condition of being directed to both set and reset at the same time. This sets up an oscillation, based on the gate delays we chose. In an actual, physical circuit, this high-speed oscillation may drive the basic cell output from rail to rail at the same oscillation frequency, or due to pin loading, the output voltage may only change by a few hundred millivolts In any case, this is an unstable condition that is commonly referred to as "meta-stability".

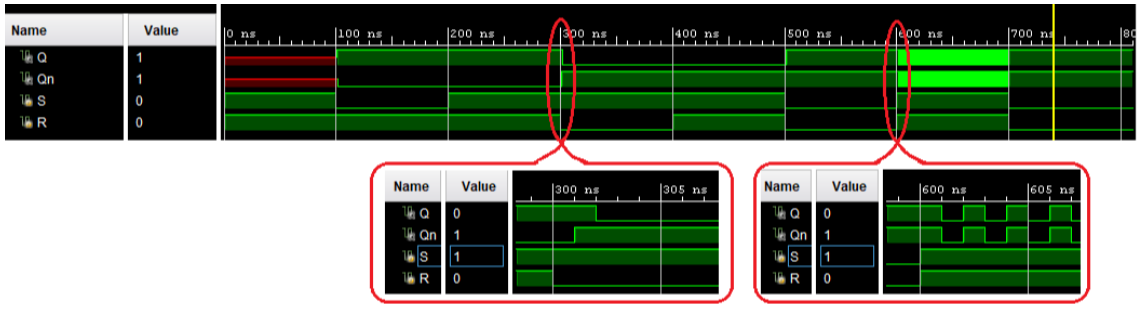


Figure 2. SR-Latch simulation waveform. (Screenshot above is from the Vivado Simulator running on Microsoft Windows 10. Altered to enhance visual understanding.)