CS 207 Digital Logic - Spring 2020
Lab 11

James Yu

Apr. 29, 2020

Objective

1. Implement adders and subtractors.

Lab Exercise Submission

1. You should name all source code as instructed. Mis-named files will not be recognized.
2. You should submit all source code files with an extension "`.v`".
3. You should submit all source code directly into the sakai system below. **Do not compress them into one folder.**

   `https://sakai.sustech.edu.cn/portal/site/ebf68254-68c5-4cfe-9d42-b26758f854ee`

4. Lab exercises should be submitted before the deadline, typically one week after the lab session. No late submission policy applies to lab exercises.

1 Half-adder

Half-adder is the foundation of many arithmetic circuits. During the lecture, we introduced how to use primitive gates to implement an half-adder, which can be defined as follows: This module can be implemented as follows:

half_adder.v

```
module half_adder(A,B,S,C);
input A,B;
output S,C;

```

```
and and1(C, A, B);
xor xor1(S, A, B);
endmodule
```

Besides this gate-level design, it is also possible to use behavioral design

half_adder.v

```
module half_adder(A,B,S,C);
input A,B;
output S,C;
reg S,C;
always @(A or B)
begin
    case({A,B})
    2'b00: begin S=0;C=0;end
    2'b01: begin S=1;C=0;end
    2'b10: begin S=1;C=0;end
    2'b11: begin S=0;C=1;end
    endcase
end
endmodule
```

or data-flow design

half_adder.v

```
module half_adder(A,B,S,C);
input A,B;
output S,C;
assign S=A^B;
assign C=A&B;
endmodule
```

All above implementations are equivalent, and they can all be tested with the following code:

half_adder_tb.v

```
module half_adder_tb;
reg A, B;
wire SUM, COUT;
half_adder myadd(A,B,SUM,COUT);

initial begin
  $monitor("%3t: A is %b, B is %b, SUM is %b, COUT is %b",$time,A,B,SUM,COUT);
  # 5 A=0;B=0;
  # 5 A=0;B=1;
    # 5 A=1;B=0;
  # 5 A=1;B=1;
  # 10 $finish;
end
endmodule
```

2 Exercise

2.1 Exercise 1

Implement a full adder according to the introduction during the lecture **without using `assign`**. The module should follow the design below:

fulladder.v

```
module fulladder(S, C, A, B, X);
// Module Code
endmodule
```

and save in file `fulladder.v`. In this module, `S` is output sum, `C` is output carry, `X` is input carry, and `A` and `B` are the addend and augend, respectively.

Assignment
Save the source code in **`fulladder.v`**. Upload this file to Sakai under **Assignments $\rightarrow$ Lab Exercise 11**.

2.2 Exercise 2

Implement a full adder-subtractor with control. The counter takes a control signal X.

- When $X = 1$, the module is a full-subtractor. `A` is minuend, `B` is subtrahend $(A - B)$. `CI` is the input borrow from less significant bit. D is the resulting difference of $A - B$, and `CO` is the output borrow to the more significant bit.
- When $X = 0$, the module is a full-adder. `A` is augend, `B` is addend $(A + B)$. `CI` is the input carry from less significant bit. D is the resulting sum of $A + B$, and `CO` is the output carry to the more significant bit.

The module should follow the design below:

add_sub.v

```
module add_sub(D, CO, X, A, B, CI);
// Module Code
endmodule
```

and save in file `add_sub.v`.

⚠ **Assignment**
Save the source code in **`add_sub.v`**. Upload this file to Sakai under **Assignments** → **Lab Exercise 11**.

⚠ **Assignment**
When you finished Lab Exercise 11, there should be two `.v` files in the Sakai system: **`fulladder.v`** and **`add_sub.v`**.