



# DSD HW1

---

Speaker: Daniel

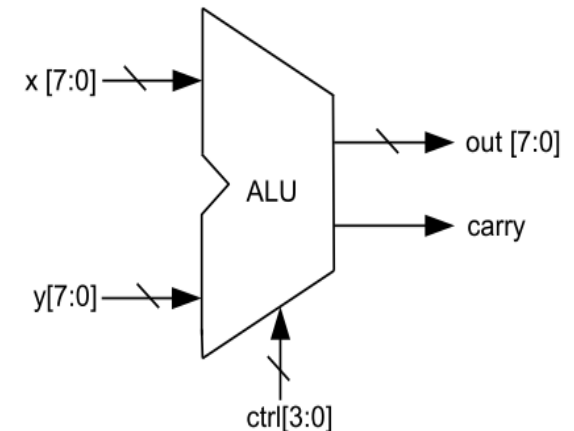
Date: 2024/03/07



# Problem 1: 8-bit Arithmetic Logic Unit

❖ Implement 8-bit ALU with following functions

Control Signal(ctrl)↵	Description↵	Function↵
0000↵	Add(signed)↵	$out = x + y$ ↵
0001↵	Sub(signed)↵	$out = x - y$ ↵
0010↵	Bitwise And↵	$out = \text{and}(x, y)$ ↵
0011↵	Bitwise Or↵	$out = \text{or}(x, y)$ ↵
0100↵	Bitwise Not↵	$out = \text{not}(x)$ ↵
0101↵	Bitwise Xor↵	$out = \text{xor}(x, y)$ ↵
0110↵	Bitwise Nor↵	$out = \text{nor}(x, y)$ ↵
0111↵	Shift left logical variable↵	$out = y \ll x[2:0]$ ↵
1000↵	Shift right logical variable↵	$out = y \gg x[2:0]$ ↵
1001↵	Shift right arithmetic↵	$out = \{x[7], x[7:1]\}$ ↵
1010↵	Rotate left↵	$out = \{x[6:0], x[7]\}$ ↵
1011↵	Rotate right↵	$out = \{x[0], x[7:1]\}$ ↵
1100↵	Equal↵	$out = (x == y) ? 1 : 0$ ↵



**Ex. Add(signed)**

$x = 8'b10010110(-106)$

$y = 8'b00101101(45)$

Ans:  $9'b1\_11000011(-61)$

→ carry = 1, out =  $8'b11000011$



# Problem 1: 8-bit Arithmetic Logic Unit

- ❖ (1) Implement RTL (*use continuous assignment, assign*) model of the ALU
- ❖ (2) Implement RTL (*use procedural assignment, always block*) model of the ALU
- ❖ (3) Modify the given testbench to verify all functions in your design are correct

Only test  
Boolean not

```
initial begin
    ctrl = 4'b1101;
    x     = 8'd0;
    y     = 8'd0;

    #(`CYCLE);
    // 0100 boolean not
    ctrl = 4'b0100;

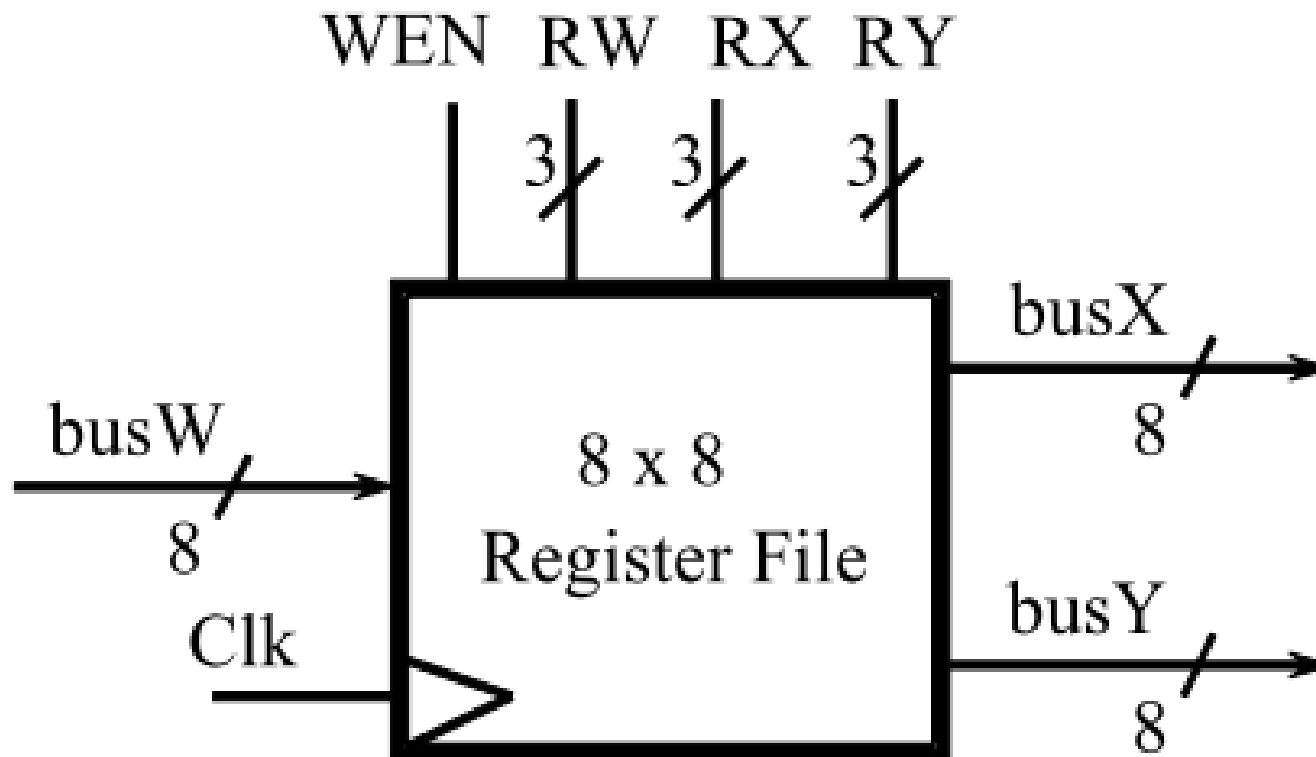
    #(`HCYCLE);
    if( out == 8'b1111_1111 ) $display( "PASS --- 0100 boolean not" );
    else $display( "FAIL --- 0100 boolean not" );

    // finish tb
    #(`CYCLE) $finish;
end
```



## Problem 2. 8x8 Register File

- ❖ Implement a 8 x 8 register file which can support read and write operation





# Flip-flop

## ❖ Combinational circuit

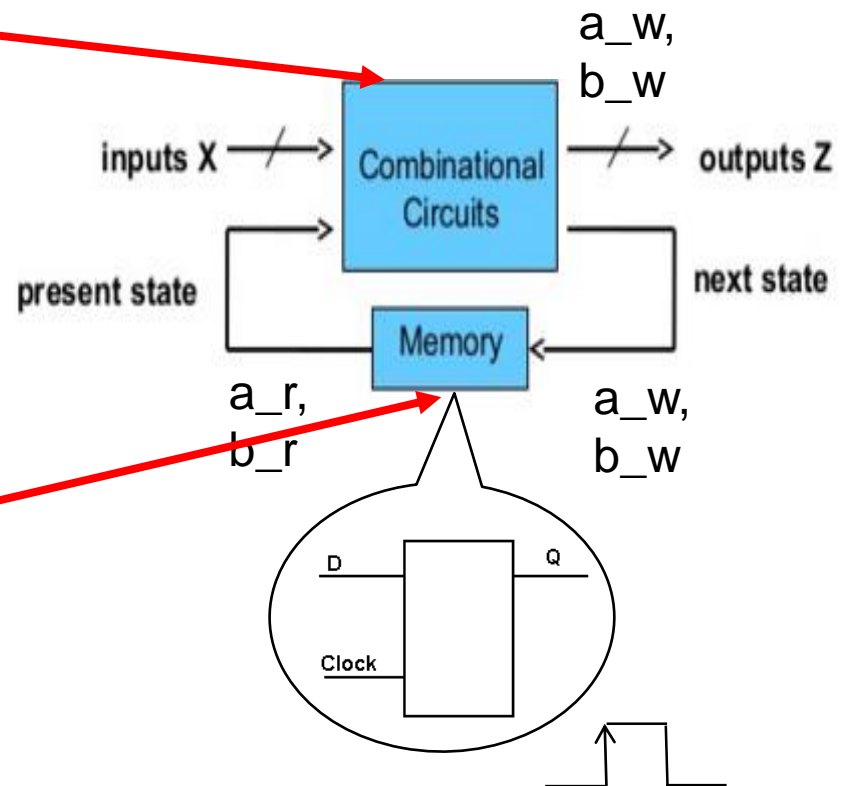
```
always@(*) begin
    a_w = a_r;
    b_w = a_r + 1;
end
```

$a_w \rightarrow a_{nxt}$

$a_r \rightarrow a$

## ❖ Sequential circuit

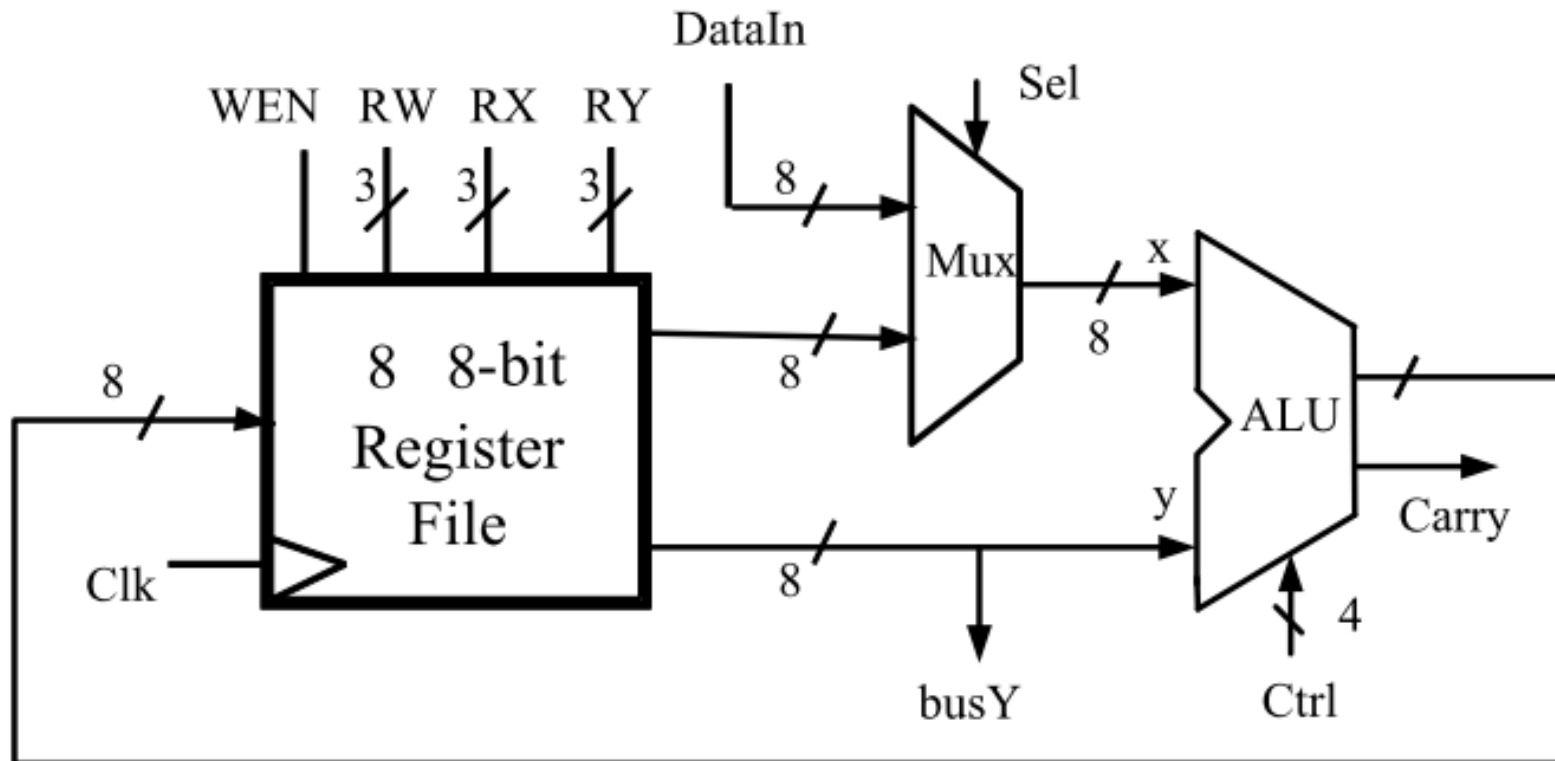
```
always@(posedge Clk) begin
    a_r <= a_w;
    b_r <= b_w;
end
```





## Problem 3. Simple Calculator

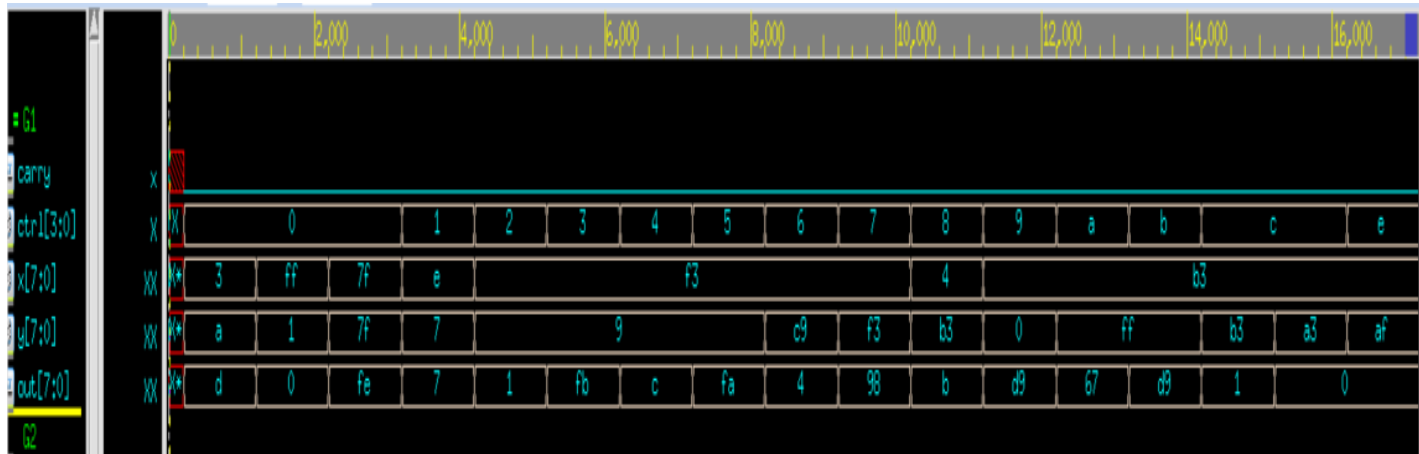
- ❖ Combine the previous two designs (ALU, Register File) into a simple calculator unit





# Report

1. **ALU:** Screenshot the waveform result in nWave of alu\_assign & alu\_always, describe how you verify the correctness
2. **Register file:** Screenshot the waveform result in nWave of register\_file, describe how you verify the correctness.
3. **Bonus (5%):** Write down what you found. Feel free to share your experience. (Ex: some mistakes you spend a lot of time, your environment, naming method)





# Submission

- ❖ Compress all the files into one **ZIP** file, and Upload to **NTUCOOL**
  - ❖ File name: DSD\_HW1\_學號.zip (DSD\_HW1\_b10901001.zip)
- ❖ DSD\_HW1\_學號/
  - 1-ALU/
    - 1\_assign/
      - alu\_assign.v, alu\_assign\_tb2.v
    - 2\_always/
      - alu\_always.v, alu\_always\_tb2.v
  - 2-RegisterFile/
    - register\_file.v, register\_file\_tb.v
  - 3-SimpleCalculator/
    - simple\_calculator.v
  - report.pdf
- ❖ Deadline: **2024/03/27 23:59**
- ❖ Late submission penalty: **20% per day**





# Note

- ❖ Continuous assignment
  - ❖ LHS has to be wire type
- ❖ always statement
  - ❖ LHS has to be reg type
- ❖ Don't use non-synthesizable syntax
- ❖ Port
  - ❖ input
    - within the module, input must be declared as type wire(net), externally, they can connect to a type reg or wire
  - ❖ output
    - within the module, output can be declared as type reg or wire(default wire), externally they must always connect to net(wire)
    - if output port used in an always statement, the port must be redeclared as type reg, e.g. output reg carry;