Project2 for Intro to Computer Systems 2025 spring

Yiming Cheng

12450588

Project2 for Intro to Computer Systems 2025 spring
- Result:
- Details
- Parallel Add16

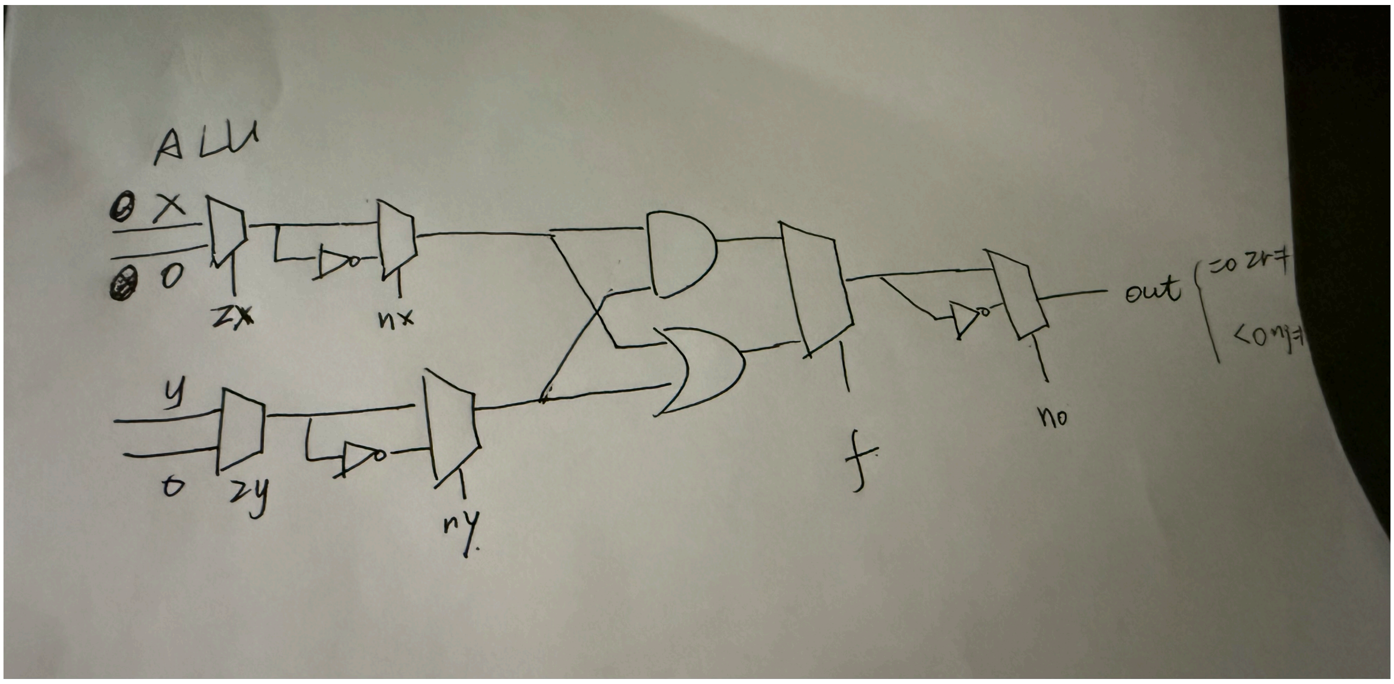
Result:

1. HalfAdder$\boxed{work}$
2. FullAdder$\boxed{work}$
3. Add16$\boxed{work}$

note：**Parallel Add16**

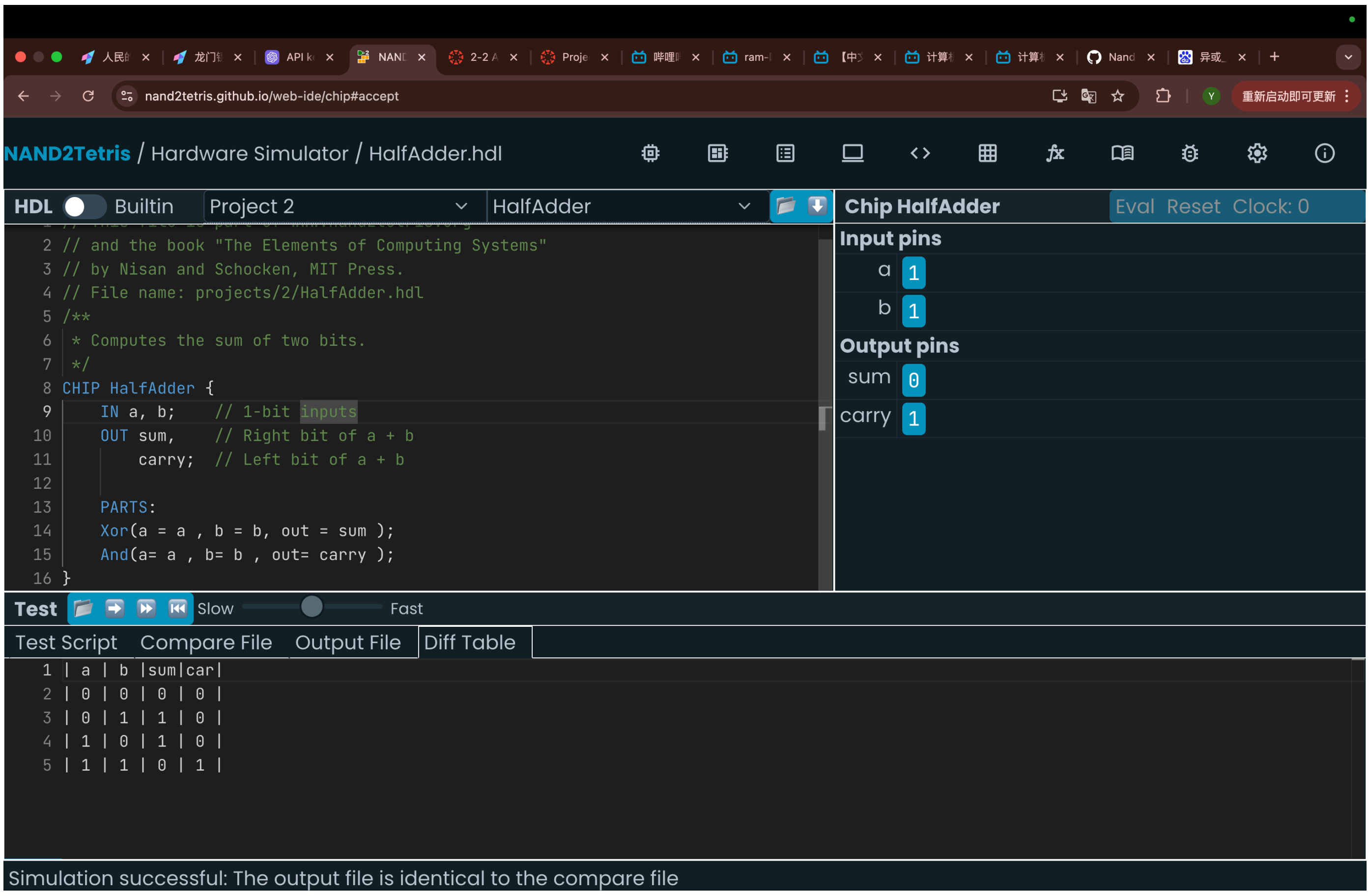
4. Inc16$\boxed{work}$

5. ALU $\boxed{work}$



Details

1. HalfAdder



```
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/2/HalfAdder.hdl
/**
 * Computes the sum of two bits.
 */
CHIP HalfAdder {
    IN a, b;    // 1-bit inputs
    OUT sum,    // Right bit of a + b
        carry;  // Left bit of a + b

    PARTS:
    Xor(a = a , b = b, out = sum );
    And(a= a , b= b , out= carry );
}
```

Chip HalfAdder

Input pins

a	1
b	1

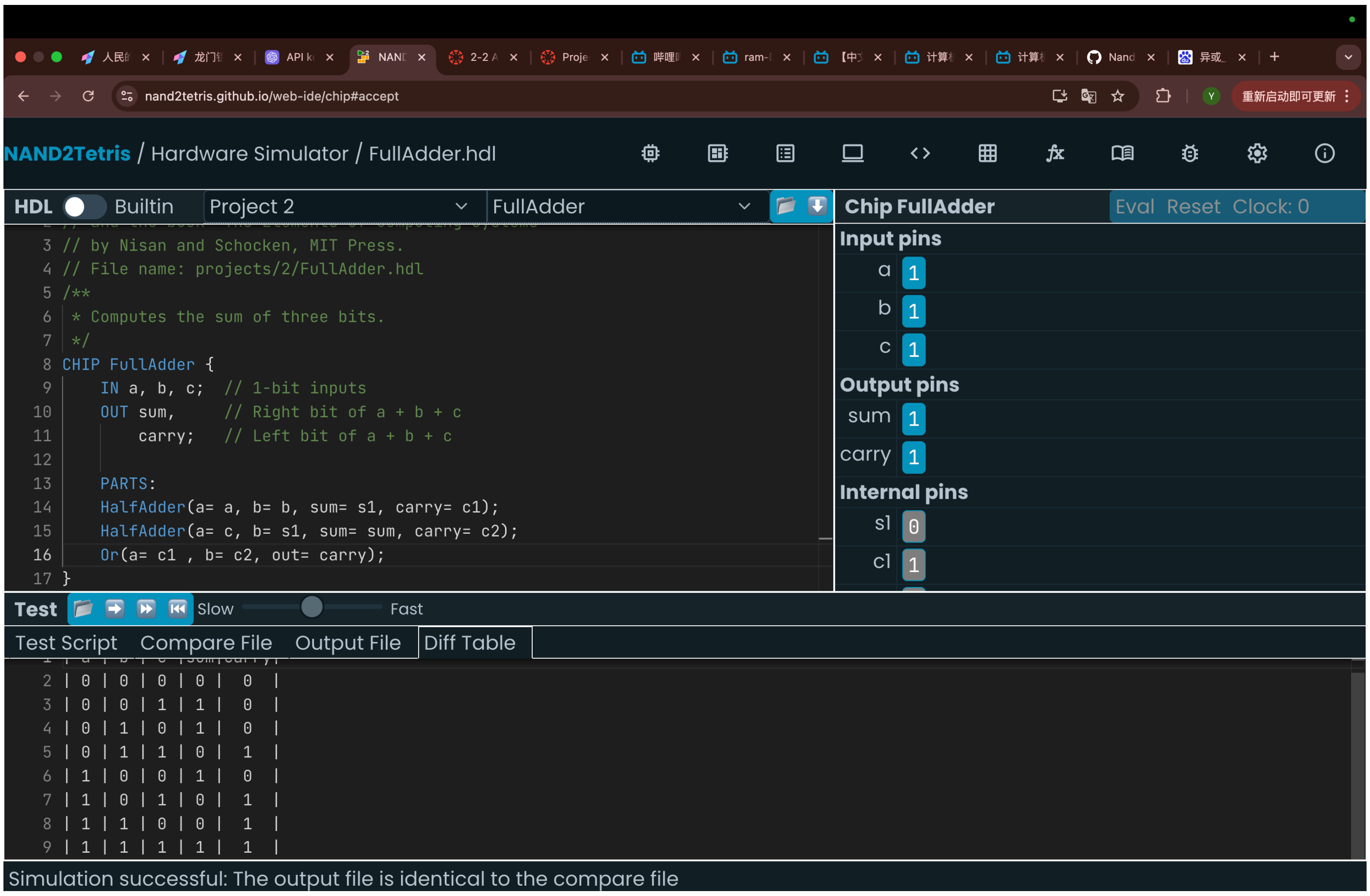
Output pins

sum	0
carry	1

```
| a | b |sum|car|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
```

Simulation successful: The output file is identical to the compare file

2. FullAdder



nand2tetris.github.io/web-ide/chip#accept

NAND2Tetris / Hardware Simulator / FullAdder.hdl

HDL Builtin Project 2 FullAdder

```
// by Nisan and Schocken, MIT Press.
// File name: projects/2/FullAdder.hdl
/**
 * Computes the sum of three bits.
 */
CHIP FullAdder {
    IN a, b, c;  // 1-bit inputs
    OUT sum,     // Right bit of a + b + c
        carry;   // Left bit of a + b + c

    PARTS:
    HalfAdder(a= a, b= b, sum= s1, carry= c1);
    HalfAdder(a= c, b= s1, sum= sum, carry= c2);
    Or(a= c1 , b= c2, out= carry);
}
```

Chip FullAdder — Eval Reset Clock: 0

Input pins
a 1
b 1
c 1

Output pins
sum 1
carry 1

Internal pins
s1 0
c1 1

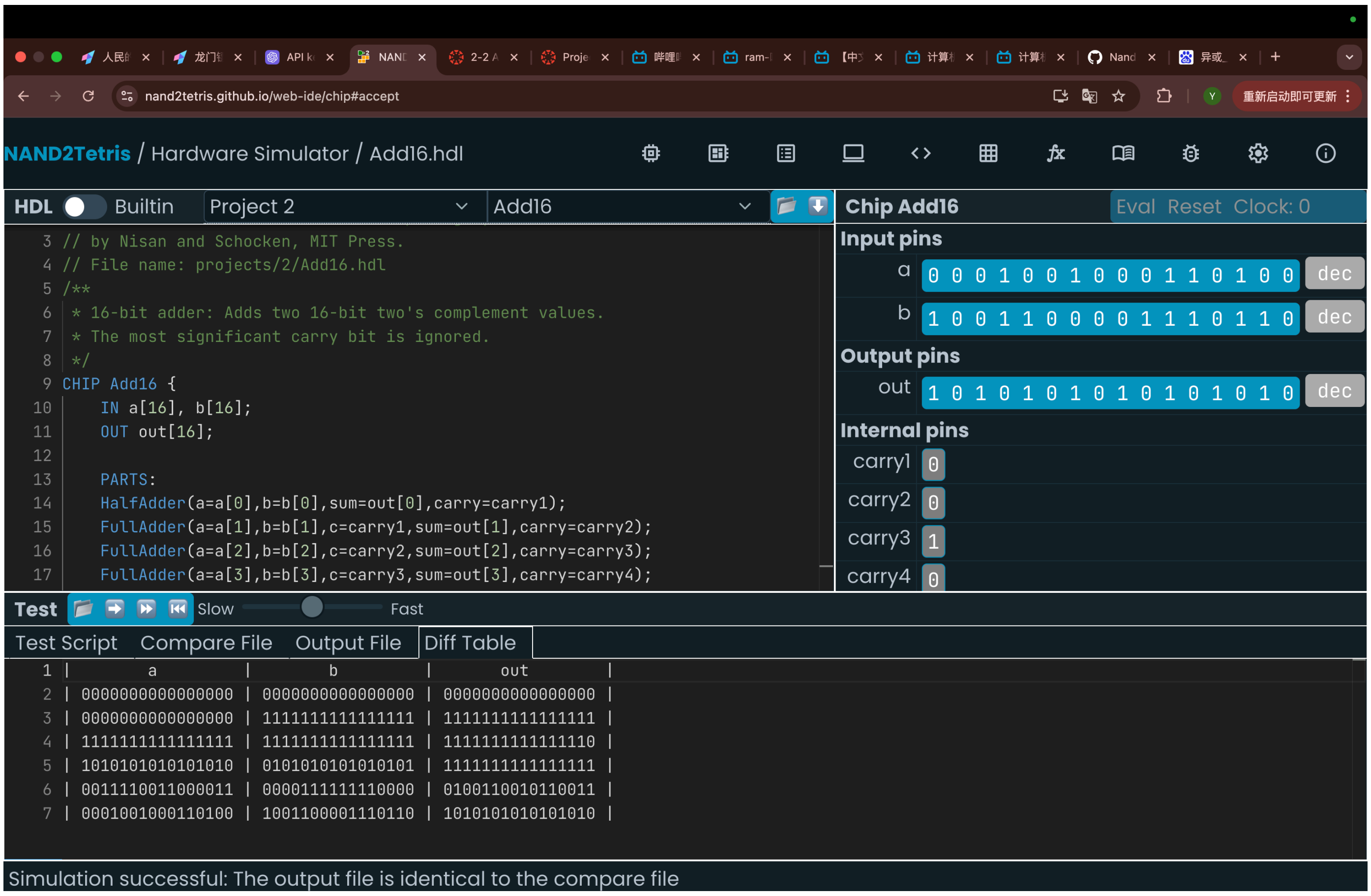
Test — Slow … Fast

Test Script | Compare File | Output File | Diff Table

```
| 0 | 0 | 0 | 0 |  0  |
| 0 | 0 | 1 | 1 |  0  |
| 0 | 1 | 0 | 1 |  0  |
| 0 | 1 | 1 | 0 |  1  |
| 1 | 0 | 0 | 1 |  0  |
| 1 | 0 | 1 | 0 |  1  |
| 1 | 1 | 0 | 0 |  1  |
| 1 | 1 | 1 | 1 |  1  |
```

Simulation successful: The output file is identical to the compare file

3. Add16



nand2tetris.github.io/web-ide/chip#accept

NAND2Tetris / Hardware Simulator / Add16.hdl

HDL Builtin Project 2 Add16

```
// by Nisan and Schocken, MIT Press.
// File name: projects/2/Add16.hdl
/**
 * 16-bit adder: Adds two 16-bit two's complement values.
 * The most significant carry bit is ignored.
 */
CHIP Add16 {
    IN a[16], b[16];
    OUT out[16];

    PARTS:
    HalfAdder(a=a[0],b=b[0],sum=out[0],carry=carry1);
    FullAdder(a=a[1],b=b[1],c=carry1,sum=out[1],carry=carry2);
    FullAdder(a=a[2],b=b[2],c=carry2,sum=out[2],carry=carry3);
    FullAdder(a=a[3],b=b[3],c=carry3,sum=out[3],carry=carry4);
```

Chip Add16 — Eval Reset Clock: 0

Input pins
a 0 0 0 1 0 0 1 0 0 0 1 1 0 1 0 0 dec
b 1 0 0 1 1 0 0 0 0 1 1 1 0 1 1 0 dec

Output pins
out 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 dec

Internal pins
carry1 0
carry2 0
carry3 1
carry4 0

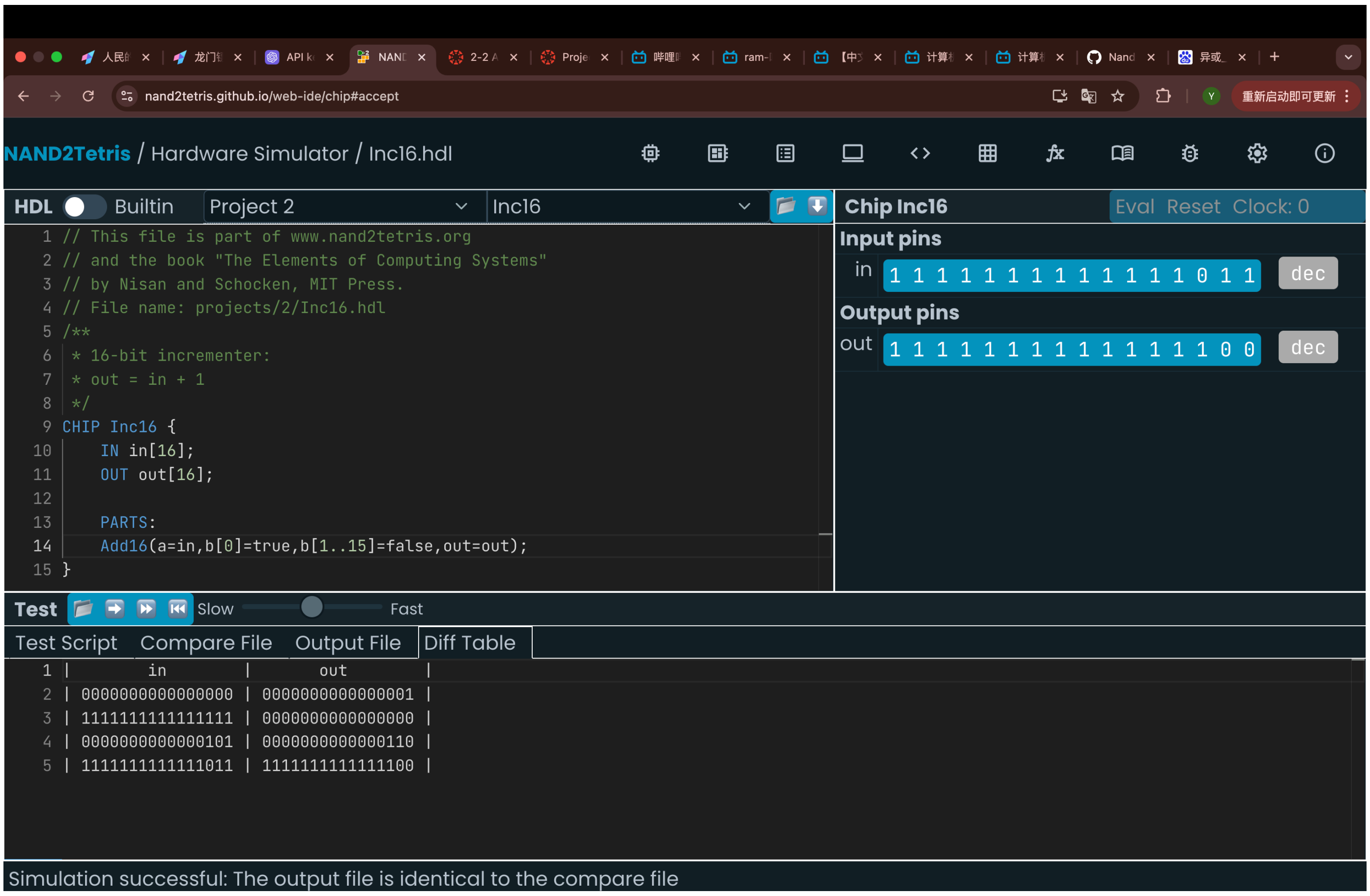
Test — Slow … Fast

Test Script | Compare File | Output File | Diff Table

```
|        a         |        b         |       out        |
| 0000000000000000 | 0000000000000000 | 0000000000000000 |
| 0000000000000000 | 1111111111111111 | 1111111111111111 |
| 1111111111111111 | 1111111111111111 | 1111111111111110 |
| 1010101010101010 | 0101010101010101 | 1111111111111111 |
| 0011110011000011 | 0000111111110000 | 0100110010110011 |
| 0001001000110100 | 1001100001110110 | 1010101010101010 |
```

Simulation successful: The output file is identical to the compare file

4. Inc16

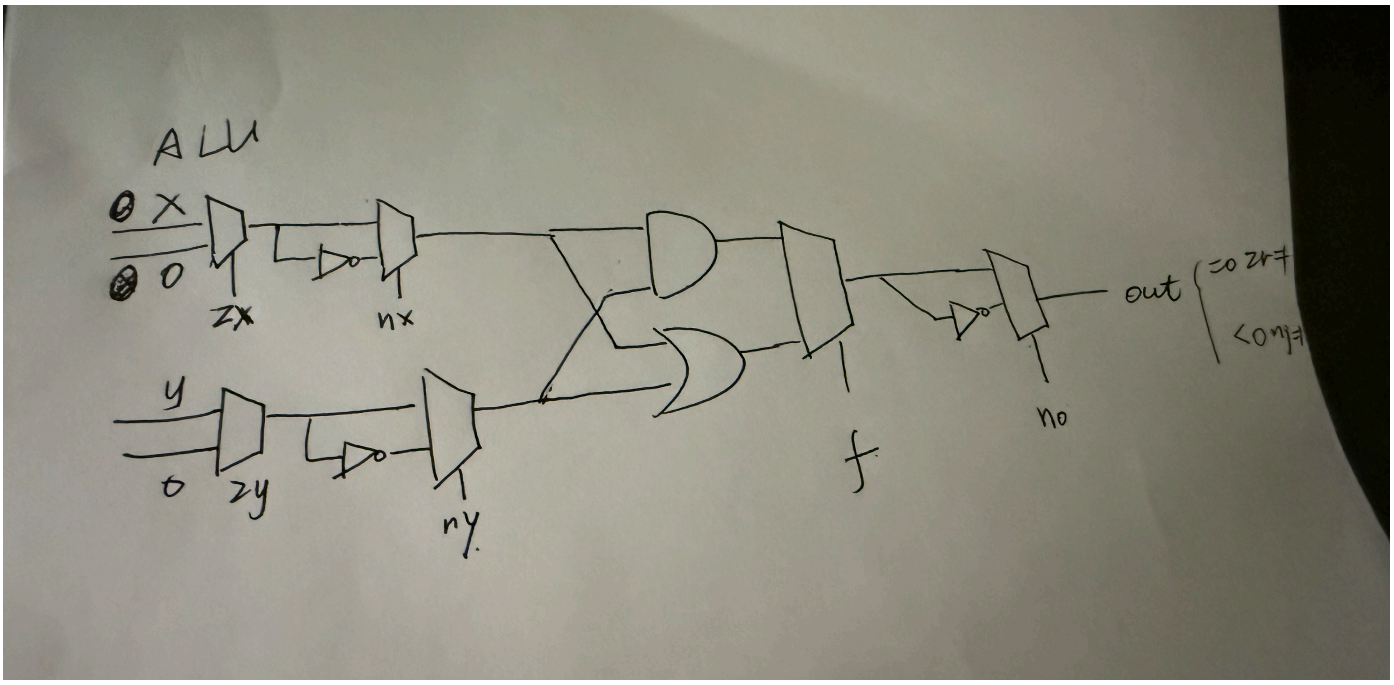

nand2tetris.github.io/web-ide/chip#accept
NAND2Tetris / Hardware Simulator / Inc16.hdl
HDL Builtin Project 2 Inc16
Chip Inc16 Eval Reset Clock: 0
Input pins
in 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 dec
Output pins
out 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 dec
Test Slow Fast
Test Script Compare File Output File Diff Table

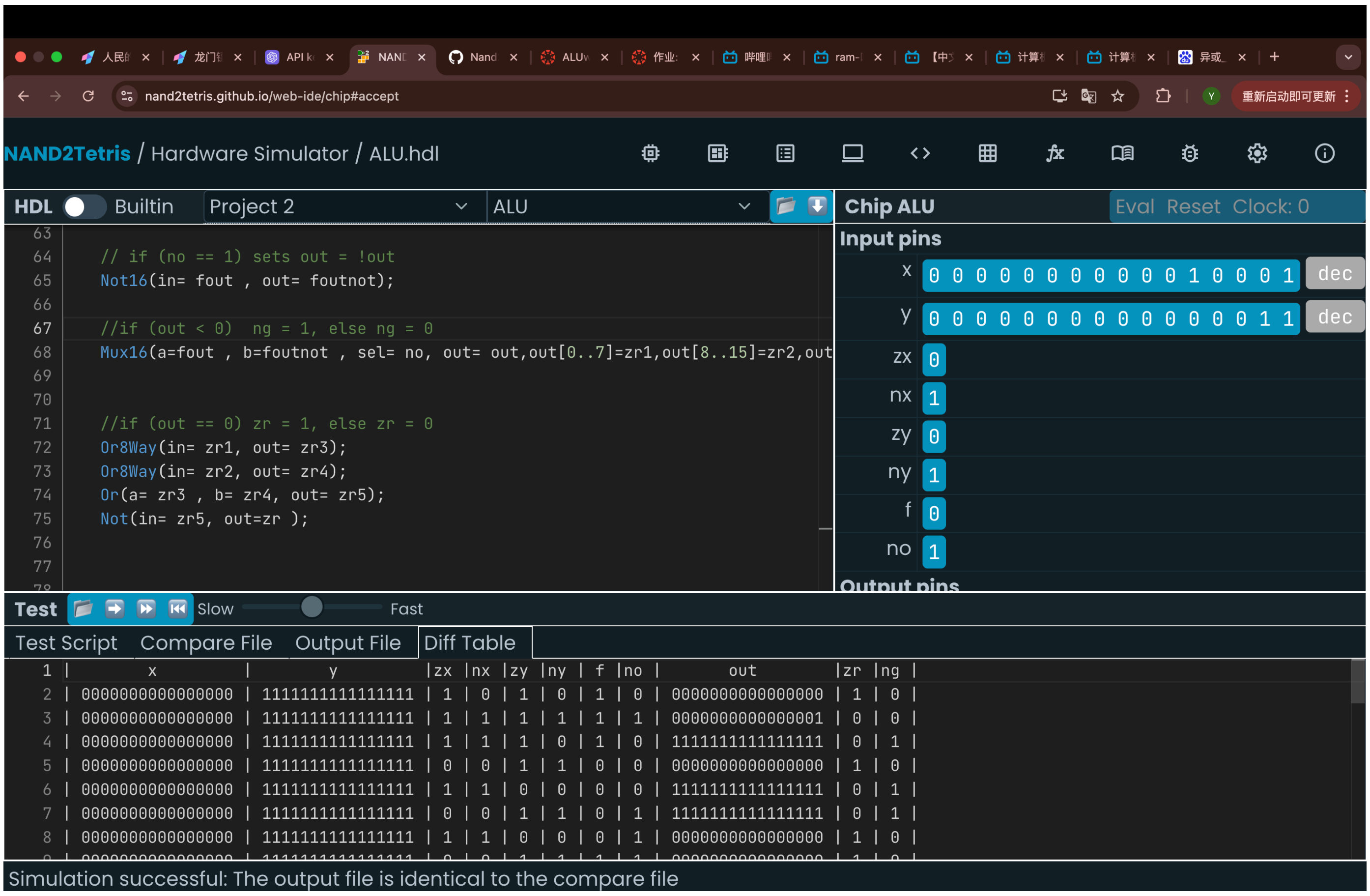
```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/2/Inc16.hdl
/**
 * 16-bit incrementer:
 * out = in + 1
 */
CHIP Inc16 {
    IN in[16];
    OUT out[16];

    PARTS:
    Add16(a=in,b[0]=true,b[1..15]=false,out=out);
}
```

```
|       in         |       out        |
| 0000000000000000 | 0000000000000001 |
| 1111111111111111 | 0000000000000000 |
| 0000000000000101 | 0000000000000110 |
| 1111111111111011 | 1111111111111100 |
```

Simulation successful: The output file is identical to the compare file

5. ALU




nand2tetris.github.io/web-ide/chip#accept
NAND2Tetris / Hardware Simulator / ALU.hdl
HDL Builtin Project 2 ALU
Chip ALU
Eval Reset Clock: 0
Input pins
Output pins
Test Slow Fast
Test Script Compare File Output File Diff Table

```
    // if (no == 1) sets out = !out
    Not16(in= fout , out= foutnot);

    //if (out < 0)  ng = 1, else ng = 0
    Mux16(a=fout , b=foutnot , sel= no, out= out,out[0..7]=zr1,out[8..15]=zr2,out


    //if (out == 0) zr = 1, else zr = 0
    Or8Way(in= zr1, out= zr3);
    Or8Way(in= zr2, out= zr4);
    Or(a= zr3 , b= zr4, out= zr5);
    Not(in= zr5, out=zr );
```

```
x  0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1  dec
y  0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1  dec
zx 0
nx 1
zy 0
ny 1
f  0
no 1
```

```
1 |        x         |        y         |zx |nx |zy |ny | f |no |       out        |zr |ng |
2 | 0000000000000000 | 1111111111111111 | 1 | 0 | 1 | 0 | 1 | 0 | 0000000000000000 | 1 | 0 |
3 | 0000000000000000 | 1111111111111111 | 1 | 1 | 1 | 1 | 1 | 1 | 0000000000000001 | 0 | 0 |
4 | 0000000000000000 | 1111111111111111 | 1 | 1 | 1 | 0 | 1 | 0 | 1111111111111111 | 0 | 1 |
5 | 0000000000000000 | 1111111111111111 | 0 | 0 | 1 | 1 | 0 | 0 | 0000000000000000 | 1 | 0 |
6 | 0000000000000000 | 1111111111111111 | 1 | 1 | 0 | 0 | 0 | 0 | 1111111111111111 | 0 | 1 |
7 | 0000000000000000 | 1111111111111111 | 0 | 0 | 1 | 1 | 0 | 1 | 1111111111111111 | 0 | 1 |
8 | 0000000000000000 | 1111111111111111 | 1 | 1 | 0 | 0 | 0 | 1 | 0000000000000000 | 1 | 0 |
```

Simulation successful: The output file is identical to the compare file

Parallel Add16

Parallel can compute carry in advance and compute parallel making it more complex but faster

```
    // Step 1: Compute Generate (G) and Propagate (P) 
signals
    And(a=a[0], b=b[0], out=G0);
    Xor(a=a[0], b=b[0], out=P0);

    And(a=a[1], b=b[1], out=G1);
    Xor(a=a[1], b=b[1], out=P1);

    And(a=a[2], b=b[2], out=G2);
    Xor(a=a[2], b=b[2], out=P2);

    And(a=a[3], b=b[3], out=G3);
    Xor(a=a[3], b=b[3], out=P3);

    And(a=a[4], b=b[4], out=G4);
    Xor(a=a[4], b=b[4], out=P4);

    And(a=a[5], b=b[5], out=G5);
    Xor(a=a[5], b=b[5], out=P5);

    And(a=a[6], b=b[6], out=G6);
    Xor(a=a[6], b=b[6], out=P6);

    And(a=a[7], b=b[7], out=G7);
    Xor(a=a[7], b=b[7], out=P7);

    And(a=a[8], b=b[8], out=G8);
    Xor(a=a[8], b=b[8], out=P8);

    And(a=a[9], b=b[9], out=G9);
    Xor(a=a[9], b=b[9], out=P9);

    And(a=a[10], b=b[10], out=G10);
    Xor(a=a[10], b=b[10], out=P10);

    And(a=a[11], b=b[11], out=G11);
    Xor(a=a[11], b=b[11], out=P11);

    And(a=a[12], b=b[12], out=G12);
    Xor(a=a[12], b=b[12], out=P12);

    And(a=a[13], b=b[13], out=G13);
    Xor(a=a[13], b=b[13], out=P13);

    And(a=a[14], b=b[14], out=G14);
    Xor(a=a[14], b=b[14], out=P14);

    And(a=a[15], b=b[15], out=G15);
    Xor(a=a[15], b=b[15], out=P15);

    // Step 2: Compute Carry signals (C) in parallel
    Or(a=G0, b=false, out=C1); // C1 = G0

    And(a=P1, b=C1, out=P1C1);
    Or(a=G1, b=P1C1, out=C2);

    And(a=P2, b=C2, out=P2C2);
    Or(a=G2, b=P2C2, out=C3);

    And(a=P3, b=C3, out=P3C3);
    Or(a=G3, b=P3C3, out=C4);

    And(a=P4, b=C4, out=P4C4);
    Or(a=G4, b=P4C4, out=C5);

    And(a=P5, b=C5, out=P5C5);
    Or(a=G5, b=P5C5, out=C6);

    And(a=P6, b=C6, out=P6C6);
    Or(a=G6, b=P6C6, out=C7);

    And(a=P7, b=C7, out=P7C7);
    Or(a=G7, b=P7C7, out=C8);

    And(a=P8, b=C8, out=P8C8);
    Or(a=G8, b=P8C8, out=C9);

    And(a=P9, b=C9, out=P9C9);
    Or(a=G9, b=P9C9, out=C10);

    And(a=P10, b=C10, out=P10C10);
    Or(a=G10, b=P10C10, out=C11);

    And(a=P11, b=C11, out=P11C11);
    Or(a=G11, b=P11C11, out=C12);

    And(a=P12, b=C12, out=P12C12);
    Or(a=G12, b=P12C12, out=C13);

    And(a=P13, b=C13, out=P13C13);
    Or(a=G13, b=P13C13, out=C14);

    And(a=P14, b=C14, out=P14C14);
    Or(a=G14, b=P14C14, out=C15);

    And(a=P15, b=C15, out=P15C15);
    Or(a=G15, b=P15C15, out=C16); // Final carry is
ignored

    // Step 3: Compute Sum bits (S)
    Xor(a=P0, b=false, out=out[0]); // out[0] = P0 ⊕ C0
(C0 = 0)
    Xor(a=P1, b=C1, out=out[1]);
    Xor(a=P2, b=C2, out=out[2]);
    Xor(a=P3, b=C3, out=out[3]);
    Xor(a=P4, b=C4, out=out[4]);
    Xor(a=P5, b=C5, out=out[5]);
    Xor(a=P6, b=C6, out=out[6]);
    Xor(a=P7, b=C7, out=out[7]);
    Xor(a=P8, b=C8, out=out[8]);
    Xor(a=P9, b=C9, out=out[9]);
    Xor(a=P10, b=C10, out=out[10]);
    Xor(a=P11, b=C11, out=out[11]);
    Xor(a=P12, b=C12, out=out[12]);
    Xor(a=P13, b=C13, out=out[13]);
    Xor(a=P14, b=C14, out=out[14]);
    Xor(a=P15, b=C15, out=out[15]);
```