

Lecture and Practice: Multi-Bit Gates (Not16 and And16) Using NAND Gates

Objective: Teach multi-bit gates (Not16 and And16) using only NAND gates, derive Not16.hdl, and have students implement And16.hdl in the Nand2Tetris IDE.

Time Breakdown (50 minutes):

- 5 min: Introduction to multi-bit gates and NAND-only rule
 - 15 min: Derive Not16.hdl using NAND gates (whiteboard + demo)
 - 25 min: Student practice: Build And16.hdl using NAND gates
 - 5 min: Wrap-up and Q&A
-

Whiteboard Notes

1. Multi-Bit Gates Overview

- **What?** Not16, And16 extend 1-bit gates to 16-bit buses.
- **Why?** Hack computer processes 16-bit words in parallel.
- **Key Rule:** Nand2Tetris uses *NAND only* as the primitive gate.
 - All gates (Not, And, Or, etc.) must be built from NAND.
 - NAND is universal: can create any Boolean function.
- **Notation:**
 - Inputs: $\text{in}[16]$, $\text{a}[16]$, $\text{b}[16]$ (16-bit buses, [0]=LSB, [15]=MSB).
 - Output: $\text{out}[16]$ (bitwise operation).
- **Example:**
 - Not16: $\text{out}[i] = \text{NOT } \text{in}[i]$ (using NAND).
 - And16: $\text{out}[i] = \text{a}[i] \text{ AND } \text{b}[i]$ (using NAND).

2. NAND as Universal Gate

- **NOT from NAND:** $\text{NAND}(\text{a}, \text{a}) = \text{NOT}(\text{a AND a}) = \text{NOT}(\text{a})$.
- **AND from NAND:** $\text{NOT}(\text{NAND}(\text{a}, \text{b})) = \text{NOT}(\text{NOT}(\text{a AND b})) = \text{a AND b}$.
- **Why NAND?** Simplest universal gate; cheap in hardware.

3. Not16 Truth Table (Simplified)

text	
in[16]	out[16]
0000...0000	1111...1111
1111...1111	0000...0000
1010...1010	0101...0101

- Each bit: $\text{out}[i] = \text{NAND}(\text{in}[i], \text{in}[i])$.

4. And16 Truth Table (Simplified)

text

a[16]	b[16]	out[16]
0000...0000	0000...0000	0000...0000
1111...1111	1111...1111	1111...1111
10101010101010	01010101010101	0000000000000000

- Each bit: $\text{out}[i] = \text{NOT}(\text{NAND}(a[i], b[i]))$.

5. HDL Structure (NAND-Based)

- **CHIP**: Defines name, inputs, outputs.
- **PARTS**: Use only NAND gates (no And, Not, etc.).
- **Syntax**: $\text{Nand}(a=x, b=y, \text{out}=z)$; with intermediate wires if needed.

6. Common Errors

- Using And/Not directly (forbidden in Project 1).
- Incorrect bus indexing: [0] is LSB (rightmost).
- Missing intermediate wires for AND's second NAND.

7. Why No Xor16?

- Not needed for Hack ALU; 1-bit Xor used in adders.
 - Nand2Tetris prioritizes minimal gates (And16, Or16, Not16, Mux16).
 - Xor16 would be 16 Xor gates, redundant since ALU builds XOR logic.
-

Lecture (20 Minutes)

1. Introduction to Multi-Bit Gates and NAND-Only Rule (5 min)

- Multi-bit gates (Not16, And16) scale 1-bit logic to 16-bit buses for Hack's 16-bit architecture.
- Nand2Tetris uses *NAND only* because it's universal and cheap.
- Whiteboard:
 - Draw 1-bit NAND: $a, b \rightarrow [\text{NAND}] \rightarrow \text{out} = \text{NOT}(a \text{ AND } b)$.
 - NOT: $\text{NAND}(a, a) = \text{NOT}(a)$.
 - AND: $\text{NAND}(a, b) \rightarrow [\text{NAND}] \rightarrow \text{NOT}(\text{NAND}(a, b)) = a \text{ AND } b$.
 - Not16: $\text{in}[16] \rightarrow [16 \times \text{NAND}] \rightarrow \text{out}[16]$.
- And16.tst tests 6 cases for efficiency (not 2^{32} combinations).

2. Derive Not16.hdl Using NAND Gates (10 min)

- **Step-by-Step** (whiteboard + Nand2Tetris IDE demo):

Stub:

```
hdl
// File name: Not16.hdl
// Inverts a 16-bit input: out[i] = NOT in[i]
CHIP Not16 {
    IN in[16];
    OUT out[16];
    PARTS:
        // Use NAND gates
    1. }
```

2. **Logic:** $\text{out}[i] = \text{NOT } \text{in}[i] = \text{NAND}(\text{in}[i], \text{in}[i])$.
 - 16 NAND gates, each with same input for both pins.
3. **PARTS:**
 - Bit 0: $\text{Nand}(a=\text{in}[0], b=\text{in}[0], \text{out}=\text{out}[0])$;
 - Bit 1: $\text{Nand}(a=\text{in}[1], b=\text{in}[1], \text{out}=\text{out}[1])$;
 - ... up to bit 15.

Instructor Solution:

```
hdl
// File name: Not16.hdl
// Inverts a 16-bit input: out[i] = NOT in[i] using NAND
CHIP Not16 {
    IN in[16];
    OUT out[16];
    PARTS:
        Nand(a=in[0], b=in[0], out=out[0]);
        Nand(a=in[1], b=in[1], out=out[1]);
        Nand(a=in[2], b=in[2], out=out[2]);
        Nand(a=in[3], b=in[3], out=out[3]);
        Nand(a=in[4], b=in[4], out=out[4]);
        Nand(a=in[5], b=in[5], out=out[5]);
        Nand(a=in[6], b=in[6], out=out[6]);
        Nand(a=in[7], b=in[7], out=out[7]);
        Nand(a=in[8], b=in[8], out=out[8]);
        Nand(a=in[9], b=in[9], out=out[9]);
        Nand(a=in[10], b=in[10], out=out[10]);
        Nand(a=in[11], b=in[11], out=out[11]);
        Nand(a=in[12], b=in[12], out=out[12]);
```

```
Nand(a=in[13], b=in[13], out=out[13]);  
Nand(a=in[14], b=in[14], out=out[14]);  
Nand(a=in[15], b=in[15], out=out[15]);
```

4. }

5. Test in IDE:

- Load Not16.hdl in Nand2Tetris Hardware Simulator.
- Load Not16.tst (from projects/01).

Run script, compare to Not16.cmp:

```
text  
| in | out |  
|0000000000000000|1111111111111111|  
|1111111111111111|0000000000000000|  
|10101010101010|01010101010101|  
|0001001000110100|1110110111001011|
```

- Confirm output matches .cmp.

6. **Explain:** Each NAND gate inverts one bit using $\text{in}[i]$ twice. This adheres to Nand2Tetris' NAND-only rule.

3. Transition to And16 Practice (5 min)

- And16: $\text{out}[i] = \text{a}[i] \text{ AND } \text{b}[i]$, built with two NAND gates per bit:
 - First NAND: $\text{NAND}(\text{a}[i], \text{b}[i]) \rightarrow \text{temp}[i]$.
 - Second NAND: $\text{NAND}(\text{temp}[i], \text{temp}[i]) = \text{NOT}(\text{temp}[i]) = \text{a}[i] \text{ AND } \text{b}[i]$.
- Whiteboard:
 - Draw: $\text{a}[i], \text{b}[i] \rightarrow [\text{NAND}] \rightarrow \text{temp}[i] \rightarrow [\text{NAND}(\text{temp}[i], \text{temp}[i])] \rightarrow \text{out}[i]$.
 - Show: $\text{out}[i] = \text{NOT}(\text{NAND}(\text{a}[i], \text{b}[i]))$.
- Task: Students write And16.hdl using NAND gates, mirroring Not16 but with two NANDs per bit.
- Setup: Students open Nand2Tetris IDE, load projects/01/And16.hdl stub, and have And16.tst/And16.cmp ready.

Practice: Build And16.hdl with NAND Gates (25 Minutes)

Student Instructions:

1. Open Nand2Tetris Hardware Simulator.

Navigate to projects/01, open And16.hdl stub:

hdl

```
// File name: projects/1/And16.hdl
```

```

// Bitwise And of two 16-bit inputs: out[i] = a[i] AND b[i]
CHIP And16 {
    IN a[16], b[16];
    OUT out[16];
    PARTS:
        // Use only NAND gates

```

2. }

3. Implement PARTS:

- For each bit i:
 - First NAND: Nand(a=a[i], b=b[i], out=temp[i]);
 - Second NAND: Nand(a=temp[i], b=temp[i], out=out[i]);
 - Repeat for i=0 to 15.
 - Use unique wire names (e.g., temp0, temp1, ..., temp15).
4. Save And16.hdl in a personal folder (e.g., projects(mb/And16.hdl)).
5. Test:
- Load And16.hdl in the simulator.

Load And16.tst:

```

text
load And16.hdl,
compare-to And16.cmp,
output-list a%B1.16.1 b%B1.16.1 out%B1.16.1;
set a %B0000000000000000, set b %B0000000000000000, eval, output;
set a %B0000000000000000, set b %B1111111111111111, eval, output;
set a %B1111111111111111, set b %B1111111111111111, eval, output;
set a %B10101010101010, set b %B01010101010101, eval, output;
set a %B0011110011000011, set b %B000011111110000, eval, output;
set a %B0001001000110100, set b %B1001100001110110, eval, output;

```

Compare to And16.cmp:

```

text
|   a   |   b   |   out   |
|0000000000000000|0000000000000000|0000000000000000|
|0000000000000000|1111111111111111|0000000000000000|
|1111111111111111|1111111111111111|1111111111111111|
|10101010101010|01010101010101|0000000000000000|
|0011110011000011|00001111110000|0000110011000000|
|0001001000110100|1001100001110110|0001000000110100|

```

- Debug failures (check indices, wiring, NAND usage).

6. Submit: Zip And16.hdl, And16.tst, And16.cmp, upload to Canvas.

Instructor Solution for And16.hdl:

```

hdI
// File name: And16.hdI
// Bitwise And of two 16-bit inputs: out[i] = a[i] AND b[i] using NAND
CHIP And16 {
    IN a[16], b[16];
    OUT out[16];
    PARTS:
        Nand(a=a[0], b=b[0], out=temp0);
        Nand(a=temp0, b=temp0, out=out[0]);
        Nand(a=a[1], b=b[1], out=temp1);
        Nand(a=temp1, b=temp1, out=out[1]);
        Nand(a=a[2], b=b[2], out=temp2);
        Nand(a=temp2, b=temp2, out=out[2]);
        Nand(a=a[3], b=b[3], out=temp3);
        Nand(a=temp3, b=temp3, out=out[3]);
        Nand(a=a[4], b=b[4], out=temp4);
        Nand(a=temp4, b=temp4, out=out[4]);
        Nand(a=a[5], b=b[5], out=temp5);
        Nand(a=temp5, b=temp5, out=out[5]);
        Nand(a=a[6], b=b[6], out=temp6);
        Nand(a=temp6, b=temp6, out=out[6]);
        Nand(a=a[7], b=b[7], out=temp7);
        Nand(a=temp7, b=temp7, out=out[7]);
        Nand(a=a[8], b=b[8], out=temp8);
        Nand(a=temp8, b=temp8, out=out[8]);
        Nand(a=a[9], b=b[9], out=temp9);
        Nand(a=temp9, b=temp9, out=out[9]);
        Nand(a=a[10], b=b[10], out=temp10);
        Nand(a=temp10, b=temp10, out=out[10]);
        Nand(a=a[11], b=b[11], out=temp11);
        Nand(a=temp11, b=temp11, out=out[11]);
        Nand(a=a[12], b=b[12], out=temp12);
        Nand(a=temp12, b=temp12, out=out[12]);
        Nand(a=a[13], b=b[13], out=temp13);
        Nand(a=temp13, b=temp13, out=out[13]);
        Nand(a=a[14], b=b[14], out=temp14);
        Nand(a=temp14, b=temp14, out=out[14]);
        Nand(a=a[15], b=b[15], out=temp15);
        Nand(a=temp15, b=temp15, out=out[15]);
}

```

Student Tips:

- Use unique wire names (temp0 to temp15) for intermediate outputs.
- Copy Not16.hdl structure, add second NAND per bit.
- If test fails, check simulator output for mismatched bits.
- Save frequently.

Instructor Support:

- Help with syntax errors (e.g., missing temp wires).
 - Common mistakes:
 - Using And instead of Nand (forbidden).
 - Wrong indices (e.g., a[1] for LSB).
 - Forgetting second NAND for AND logic.
 - Show LogiSim diagram (from 3_boolean.org) if needed:
 - And16: 16 pairs of NAND gates, each pair computing $a[i] \text{ AND } b[i]$.
-

Wrap-Up and Q&A (5 Minutes)

- **Recap:**
 - Multi-bit gates scale 1-bit logic to 16-bit buses using NAND only.
 - Not16: 16 NAND gates ($\text{NAND}(\text{in}[i], \text{in}[i])$).
 - And16: 32 NAND gates (two per bit for AND).
 - NAND-only rule reflects Nand2Tetris' minimalism.
- **Why No Xor16?**
 - Not needed for ALU; 1-bit Xor used in adders.
 - Minimal gate set: And16, Or16, Not16, Mux16 suffice.
- **Next Steps:**
 - Finish And16.hdl.
 - Preview: Multi-way gates (e.g., Mux4Way16).
- **Q&A:**
 - Why NAND only? (Universal, cheap, simplifies design.)
 - Why 16 bits? (Hack's architecture.)
 - Debug tips? (Check .out vs .cmp bit-by-bit.)