Project One — Nand2Tetris: Logic Gates

Status: In progress / Implemented

Overview

This repository contains HDL implementations for Project 1 of the Nand2Tetris curriculum. The work focuses on constructing correct combinational logic components (basic gates, multiplexers, demultiplexers) and extending them to 16-bit buses and multi-way variants using HDL. Emphasis is placed on modular design, test-driven verification, and composing complex behavior from simpler building blocks.

Files Included

- `And.hdl` — 1-bit AND gate
- `And16.hdl` — 16-bit AND (bitwise application of `And`)
- `Not.hdl` — 1-bit NOT gate
- `Not16.hdl` — 16-bit NOT
- `Or.hdl` — 1-bit OR gate
- `Or16.hdl` — 16-bit OR
- `Or8Way.hdl` — reduction OR across 8 inputs
- `Xor.hdl` — 1-bit XOR gate
- `Mux.hdl` — 1-bit selector (two-way)
- `Mux16.hdl` — 16-bit two-way multiplexer
- `Mux4Way16.hdl` — 4-way 16-bit multiplexer
- `Mux8Way16.hdl` — 8-way 16-bit multiplexer
- `DMux.hdl` — 1-bit demultiplexer (single input to two outputs)
- `DMux4Way.hdl` — 4-way demultiplexer
- `DMux8Way.hdl` — 8-way demultiplexer

PROFI

Component reference: purpose and truth tables

This section lists each component, a short description of what it is used for, and the truth table(s) you can use to verify it.

1-bit primitives

And — Purpose: logical conjunction; used where two signals must both be true.
Typical uses in Nand2Tetris: bitwise AND operation in the ALU, carry-generation in adders, masking bits, and combining control conditions.

A	B	Out
0	0	0
0	1	0

A	B	Out
1	0	0
1	1	1

`Or` — Purpose: logical disjunction; used where at least one signal should be true.
Typical uses in Nand2Tetris: bitwise OR operation in the ALU, combining conditions (e.g., to detect if any input bit is set), and forming logic for control signals.

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

`Not` — Purpose: logical inversion; used to negate control or data signals.
Typical uses in Nand2Tetris: implementing logical negation in ALU control, inverting signals for two's-complement subtraction, and deriving complementary control lines.

A	Out
0	1
1	0

`Xor` — Purpose: true when inputs differ; used in parity and arithmetic logic.
Typical uses in Nand2Tetris: computing the sum bit in adders (half- and full-adder logic), parity checks, and equality/inequality tests.

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

PROFI

Multi-bit and reduction components

`And16`, `Or16`, `Not16` — Purpose: apply the corresponding 1-bit operation independently to each bit of a 16-bit bus. Tests should verify representative bit patterns (all zeros, all ones, alternating bits, single-bit set).

Behavior (per-bit): For each bit i, Out[i] = Gate(A[i], B[i]) (or Out[i] = Not(A[i]) for `Not16`).

`Or8Way` — Purpose: reduction OR of eight inputs; outputs 1 if any input is 1.

i0 i1 i2 i3 i4 i5 i6 i7	Out
all 0	0
any 1	1

Multiplexers

A digital logic circuit that accepts several data inputs and allows only one of them at a time to flow through the output

As already mentioned, a **multiplexer**, also referred to as **MUX**, is a combination logic circuit that is designed to accept multiple input signals and transfer only one of them through the output line. In simple words, a multiplexer is a digital logic device that selects one-out-of-N (N = 2^n^) input data sources and transmits the selected data to a single output line.

The multiplexer is also called **data selector** as it selects one from several. The block diagram of a typical **2^n^:1** multiplexer is shown below :

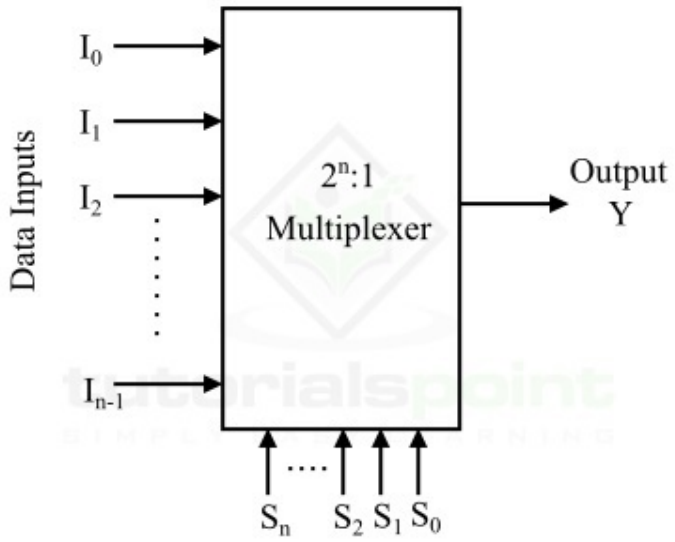


Figure 1 - Digital Multiplexer

PROFI

sel	a	b	Out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

`Mux16` — Purpose: 16-bit two-way multiplexer; applies `Mux` independently to each bit of the buses.

`Mux4Way16` / **`Mux8Way16`** — Purpose: multi-way 16-bit multiplexers; select lines encode which input bus is forwarded to the output.

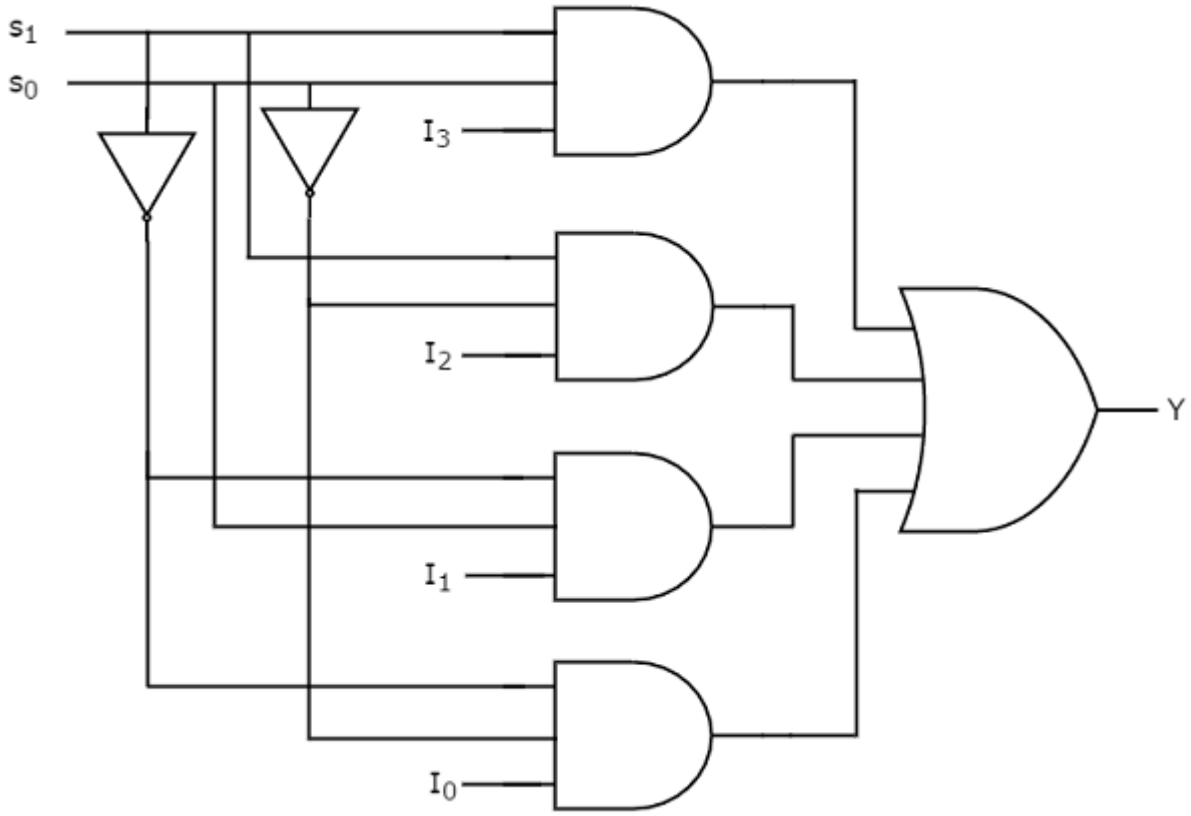
Selection mapping examples:

- `Mux4Way16` (select is 2 bits `s1 s0`): `00` → input0, `01` → input1, `10` → input2, `11` → input3.
- `Mux8Way16` (select is 3 bits `s2 s1 s0`): `000` → input0, ..., `111` → input7.

From Truth table, we can directly write the Boolean function for output, Y as

$$
Y=S1'S0'I0+S1'S0I1+S1S0'I2+S1S0I3
$$

We can implement this Boolean function using Inverters, AND gates & OR gate. The circuit diagram of 4×1 multiplexer is shown in the following figure:



PROFI

why MUX ?

- Share hardware → one ALU/wire used by many inputs instead of duplicating circuits.
- Select operations → choose which data goes where based on control signals.
- Reduce complexity & cost → fewer gates, fewer wires.
- Enable CPU control logic → instruction bits decide data paths.

In short: MUX = controlled data routing with minimal hardware.

Demultiplexers

A **Demultiplexer** is a combinational logic circuit that accepts a single input and distributes it over several output lines. Demultiplexer is also termed as **DEMUX** in short. As Demultiplexer is used to transmit the same data to different destinations, hence it is also known as **data distributor**.

There is another combinational logic circuit named multiplexer which performs opposite operation of the Demultiplexer, i.e. accepts several inputs and transmits one of them at time to the output line

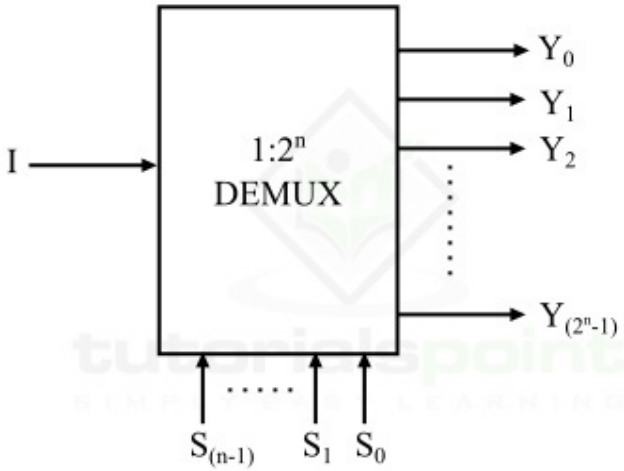


Figure 1 - Demultiplexer

table

`DMux4Way` / `DMux8Way` — Purpose: route a single input to one of 4 or 8 outputs based on multi-bit select lines. Only the selected output should carry the input value; all others must be 0.

Selection mapping (example for `DMux4Way` with two-bit select): `00` → out0, `01` → out1, `10` → out2, `11` → out3.

Resources

- https://nand2tetris-hdl.github.io/?utm_source=chatgpt.com#not16