Lecture 4 — Combinational Logic

Table of contents

Outline . . . 1
Recap — binary adder . . . 2
Binary subtractors . . . 2
Combinational circuits in general . . . 4
Multiplexers . . . 4
Worked: 2-to-1 multiplexer truth table . . . 4
Worked: choosing mux inputs to realise $X = \bar{S_0} + S_1$ . . . 5
Decoders . . . 6
Timing diagram representations . . . 6
Worked: timing diagram for an XOR gate . . . 7
Reminders . . . 8

See combinational-logic-blocks[1] for the reference definitions/tables of the adder-subtractor, multiplexer and decoder introduced here, and timing-diagrams[2] for the timing-diagram representation and real-gate timing parameters.

Outline

- Admin
- Binary subtractors
- More combinational logic circuits
  - Multiplexers
  - Decoders
  - Timing diagram representations

[1] courses/csse2010/combinational-logic-blocks.pdf
[2] courses/csse2010/timing-diagrams.pdf

Recap — binary adder

Picking up from 2026-08-04-binary-arithmetic[3]: full adders can be **cascaded** to make a multi-bit binary adder. For 4 bits, four full adders take $A_3B_3 \ldots A_0B_0$ and produce $S_3 \ldots S_0$ plus C_4, with each stage's carry-out C_{i+1} wired to the next stage's carry-in. For addition the initial carry-in C_0 is **0**. This arrangement is a **ripple-carry adder**.

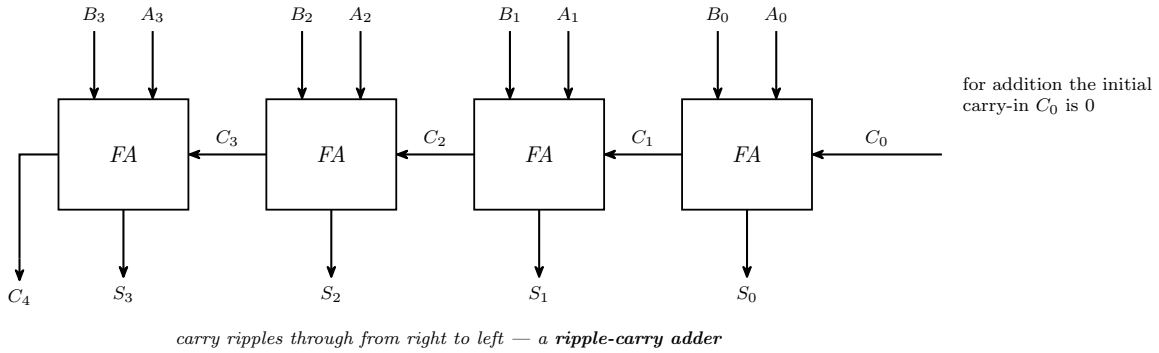


Figure 1: The 4-bit ripple-carry adder. Each full adder handles one bit position: it takes A and B for that position plus the carry coming in from the position to its right, and produces that position's sum bit plus a carry out to the left. Nothing but the carry crosses between stages.

Binary subtractors

$A - B$ is implemented as $A + (-B)$, with $-B$ the two's complement of B. Three things to keep straight:

- A and B are **multi-bit** quantities.
- The "+" means **addition**, not OR.
- $-B$ is the two's complement of B: **flip the bits and add 1**, i.e. $-B = \bar{B} + 1$.

Take $5 - 3$, first as an ordinary subtraction and then as the addition the hardware actually performs.

The two pieces needed on top of a plain adder are a **controlled inverter** (an XOR gate per bit, with the mode select M as the second input) and the extra $+1$, which is supplied via the adder's carry-in. Full derivation and the adder-subtractor circuit are in combinational-logic-blocks[4].

Two questions settle the design:

[3] courses/csse2010/2026-08-04-binary-arithmetic.pdf
[4] courses/csse2010/combinational-logic-blocks.pdf

$$-3 \longrightarrow 3 = 0011 \xrightarrow{\text{invert}} 1100 \xrightarrow{+1} 1101 = -3$$

in general $-B = \bar{B} + 1$

what we want			**what the hardware does**	
5	0101		5	0101
$-$ 3	0011	$\xrightarrow{A-B}$ $= A + (-B)$	$+$ (-3)	1101
2	0010		2	0010

the carry out of the top bit is discarded — with four bits the answer $0010_2 = 2$ is read straight off, exactly as the subtraction gives

Figure 2: 5 - 3 worked as a subtraction and as the addition the hardware performs. Negating 3 takes two steps — invert 0011 to 1100, then add 1 to get 1101 — and 1101 is the four-bit two's complement pattern for -3. Adding it to 0101 gives the same 0010 the subtraction gives, so a machine that can only add can subtract.

- **Which gate flips a bit, but only sometimes?** The gate needs $Z = \bar{B}$ when $M = 1$ and $Z = B$ when $M = 0$. The three-column M, B, Z truth table gives a 2-input XOR, $Z = B \oplus M$. The general identities behind it are $X \oplus 0 = X$ and $X \oplus 1 = \bar{X}$ **for any signal** X.
- **What should carry-in be?** $C_0 = M$ — the same mode-select line that drives the XOR gates, so subtract mode automatically contributes the +1 that two's complement needs. $M = 0$ gives $S = A + B$, and $M = 1$ gives $S = A + (-B) = A + \bar{B} + 1$.

Combinational circuits in general

A combinational circuit is a combination of logic gates with n inputs and m outputs; each output is a function of the n inputs, the **output depends on current inputs only**, and it can be written as a truth table with n input columns, m output columns and 2^n rows. See combinational-logic-blocks[5].

Multiplexers

2^n data inputs, 1 output, n select inputs that steer one data input to the output — a **data selector**. The two sizes used here are the 4-to-1 mux (logic symbol, function table, and the AND-OR sum-of-products implementation with inverted select literals) and the 2-to-1 mux. Tables and circuit descriptions: combinational-logic-blocks[6]. Off-the-shelf part: the 74HCT157 quad 2-input mux, see device-pinouts[7].

The 4-to-1 function table has four rows, one per select combination, with F holding a *data input name* rather than a 0 or 1. That is deliberate: the full truth table over all six inputs would need $2^6 = 64$ rows.

Worked: 2-to-1 multiplexer truth table

A 2-to-1 multiplexer has data inputs D_0 and D_1, control input S_0, and output F, with function table $F = D_0$ when $S_0 = 0$ and $F = D_1$ when $S_0 = 1$. Which of three candidate truth tables (columns S_0, D_0, D_1, F) goes with this circuit?

Three inputs means $2^3 = 8$ rows, which all three candidates have. Reasoning from the function table, row by row in the order $S_0 D_0 D_1 = 000, 001, 010, 011, 100, 101, 110, 111$:

- The first four rows have $S_0 = 0$, so F must copy D_0 and ignore D_1: $F = 0, 0, 1, 1$.
- The last four rows have $S_0 = 1$, so F must copy D_1 and ignore D_0: $F = 0, 1, 0, 1$.

[5] courses/csse2010/combinational-logic-blocks.pdf
[6] courses/csse2010/combinational-logic-blocks.pdf
[7] courses/csse2010/device-pinouts.pdf

$$F = 0,\ 0,\ 1,\ 1,\ 0,\ 1,\ 0,\ 1$$

That is **option 2**. Option 1 has F following D_1 when $S_0 = 0$ and D_0 when $S_0 = 1$ — the two data inputs swapped. Option 3 is $F = 0, 0, 0, 0, 1, 1, 1, 1$, i.e. $F = S_0$: it ignores the data inputs completely.

Equivalently: $F = D_0\bar{S}_0 + D_1 S_0$.

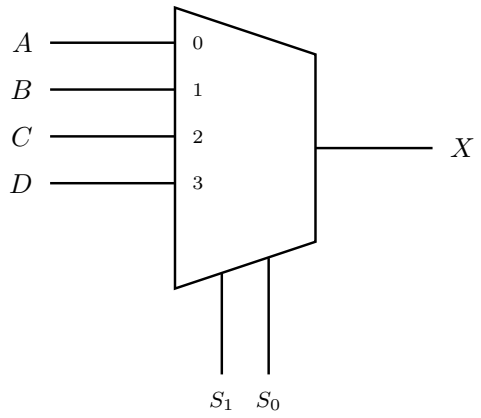
Worked: choosing mux inputs to realise $X = \bar{S}_0 + S_1$

A 4-to-1 multiplexer has A on data input 0, B on 1, C on 2, D on 3, selects S_1S_0, and output X. What must A, B, C, D be so that

$$X = \bar{S}_0 + S_1$$

Options: (1) A=0, B=1, C=0, D=0; (2) A=0, B=1, C=1, D=1; (3) A=1, B=0, C=1, D=1; (4) A=1, B=1, C=0, D=1.

Reasoning: the mux steers data input i to the output when S_1S_0 equals i in binary, so each data input must simply be **the value the desired function takes for that select combination**. Evaluate $\bar{S}_0 + S_1$ for all four:



$X = \bar{S}_0 + S_1$

S_1	S_0	$X = \bar{S}_0 + S_1$	data input selected
0	0	$1 + 0 = 1$	$A = 1$
0	1	$0 + 0 = 0$	$B = 0$
1	0	$1 + 1 = 1$	$C = 1$
1	1	$0 + 1 = 1$	$D = 1$

Figure 3: The 4-to-1 mux and the function it has to realise. Each select combination picks one data input, so the value that input must carry is just the function evaluated at that combination — the four evaluations are 1+0, 0+0, 1+1, 0+1.

S_1	S_0	$\bar{S}_0$	$X = \bar{S}_0 + S_1$	Selected input
0	0	1	1	A (input 0)
0	1	0	0	B (input 1)
1	0	1	1	C (input 2)
1	1	0	1	D (input 3)

So $A = 1, B = 0, C = 1, D = 1$ — **option 3**.

This is the general trick: tying a mux's data inputs to constants turns it into a lookup table for any function of the select variables.

Decoders

A decoder converts an n-bit input to a logic 1 on exactly one of 2^n outputs. The 3-to-8 decoder (inputs A, B, C; outputs $D_0 \ldots D_7$) is implemented with eight AND gates. Each row of its truth table corresponds to one minterm — $\bar{A}\bar{B}\bar{C}$ at the top down to ABC at the bottom.

Because a decoder puts every minterm on its own wire, an arbitrary function is just an OR of the right outputs. For example

$$F = \bar{A}\bar{B}\bar{C} + ABC = D_0 + D_7$$

one OR gate on two decoder outputs. Circuit and diagram: combinational-logic-blocks[8].

Timing diagram representations

The **timing diagram** is another logic representation — like a truth table but graphical. timing-diagrams[9] has the inverter as the worked example, the convention for drawing 2-input waveforms so all input combinations are covered, and the reality that gates aren't perfect (propagation delay, fall time, rise time).

[8] courses/csse2010/combinational-logic-blocks.pdf
[9] courses/csse2010/timing-diagrams.pdf

Worked: timing diagram for an XOR gate

Given input waveforms where A is low for the first half of the diagram then high for the second half, and B toggles at twice that rate (low, high, low, high) — so the four intervals are $AB = 00, 01, 10, 11$ — which of three candidate output waveforms represents an **XOR** gate?

Reasoning from the XOR truth table ($X = 1$ iff exactly one input is 1):

Interval	A	B	$A \oplus B$
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

So the output must be **low, high, high, low** — a single pulse spanning the middle two intervals.

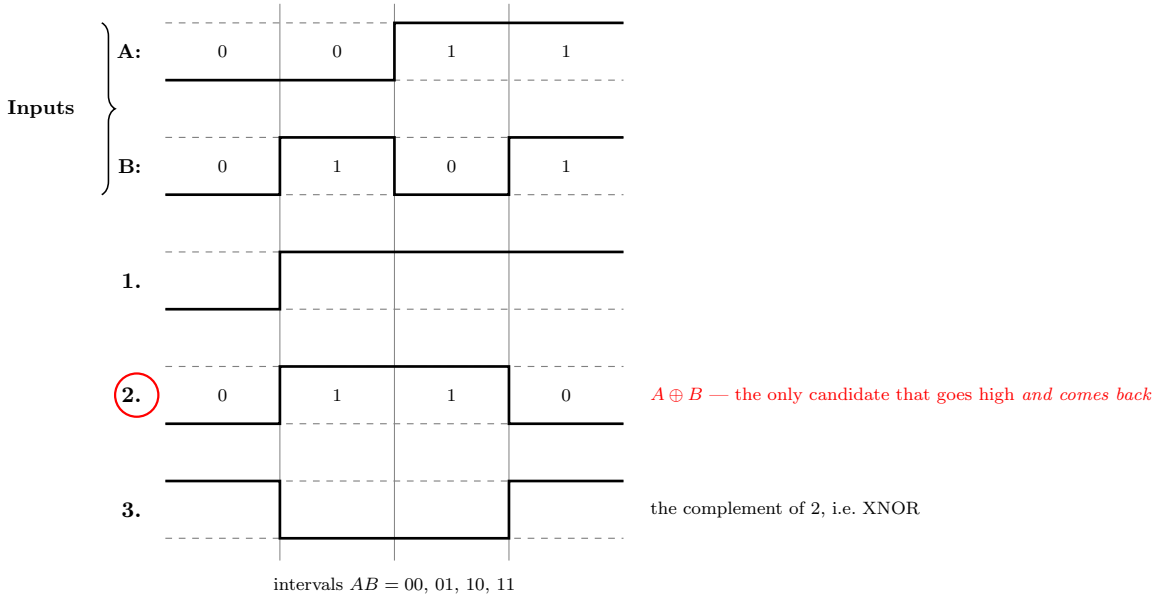


Figure 4: The inputs A and B and the three candidate outputs. Candidate 2 is the only one that goes high and comes back down, which is what a pulse over the middle two intervals requires. Candidate 1 rises once and stays high, so it can never be XOR; candidate 3 is candidate 2 inverted, i.e. XNOR.

That is **option 2** — the only candidate with a pulse that goes high and comes back low again. Option 1 has a single rising edge and then stays high to the end, so it can never be XOR. Option 3 is the exact complement of option 2 (high, low, low, high), i.e. XNOR.

Reminders

- **Lab 4 preparation task** — next week (week 3). It will be available on Blackboard under *Learning Resources*; it involves drawing some circuit schematic diagrams (see circuit-schematics[10]), and you must bring it to Lab 4 (P2 Mon-Tue sessions in week 3). Complete it before Lab 4. Worked answers: week3-lab-lab-4-pre-work[11].
- Supporting material for the labs is now on Blackboard.
- Attempt the posted exercises and quiz (not assessed).
- Any doubts: use course consultation, ask after the lecture, or use the Ed forum.

[10] courses/csse2010/circuit-schematics.pdf
[11] courses/csse2010/week3-lab-lab-4-pre-work.pdf