Digital Circuits

Cezar Ionescu
cezar.ionescu@th-deg.de

WS 2023-24

TECHNISCHE HOCHSCHULE DEGGENDORF

Outline

1. Admin
2. Review
3. Digital gates

Admin

Questions?

Review

Binary arithmetic

- Addition (carry is 0 or 1):

```
1111     (carry)
 10110
+01110
------
100100
```

- Multiplication

```
  1110
x 1001
------
  1110
 0000
0000
1110
-------
1111110
```

Digital gates

Decomposition

- **Fundamental insight:** arithmetical operations on many large numbers can be decomposed into a (large) number of standard operations on few small numbers.
- Example:
  - $a_1 + a_2 + \ldots + a_n = a_1 + (a_2 + (a_3 + (\ldots + a_n)\ldots))$

Decomposing addition in base b

$$
\begin{array}{ccccccc}
 & & [x_1 & x_2 & \dots & x_n] & \\
 & + & [y_1 & y_2 & \dots & y_n] & = \\
 & [z_0 & z_1 & z_2 & \dots & z_n] &
\end{array}
$$

where

$$
\begin{aligned}
c_k &= \text{if } k == n \text{ then } 0 \text{ else } x_{k+1} + y_{k+1} + c_{k+1} \text{ div } b \\
z_k &= \text{if } k == 0 \text{ then } c_0 \text{ else } x_k + y_k + c_k \text{ mod } b
\end{aligned}
$$

Decomposing addition in base 2

For base 2, obtaining the results

$$
\begin{aligned}
c_k &= \text{if } k == n \text{ then } 0 \text{ else } x_{k+1} + y_{k+1} + c_{k+1} \text{ div } b \\
z_k &= \text{if } k == 0 \text{ then } c_0 \text{ else } x_k + y_k + c_k \text{ mod } b
\end{aligned}
$$

involves examining only one or two *bits* at a time.

One-bit operations

Let $\mathbb{B} = \{0, 1\}$.

How many non-constant functions of type $\mathbb{B} \to \mathbb{B}$ are there?

One-bit operations

Let $\mathbb{B} = \{0, 1\}$.

How many non-constant functions of type $\mathbb{B} \to \mathbb{B}$ are there?

Two:

```
id 0 = 0
id 1 = 1

neg 0 = 1
neg 1 = 0
```

Identity

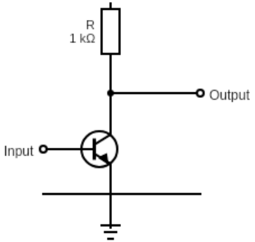
Implementing the identity function is trivial:

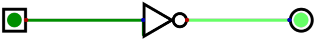
Negation

The *negate* function (also known as flip, invert, change, reverse, etc.) is usually denoted by $-$, $\neg$, or $\bar{\cdot}$ and has the truth table

x	$\overline{x}$
0	1
1	0

and is implemented by a NOT gate:





Two-bit operations

How many functions of type $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$ are there?

```
const1 (b_1, b_2) = 1

min (b_1, b_2)    = if b_1 == 0 or b_2 == 0 then 0 else 1

max (b_1, b_2)    = if b_1 == 1 or b_2 == 1 then 1 else 0

leq (b_1, b_2)    = if b_1 == 0 then 1 else b_2
```

Two-bit operations

How many functions of type $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$ are there?

$2^4 = 16$

Examples:

```
const0 (b_1, b_2) = 0

const1 (b_1, b_2) = 1

min (b_1, b_2)    = if b_1 == 0 or b_2 == 0 then 0 else 1

max (b_1, b_2)    = if b_1 == 1 or b_2 == 1 then 1 else 0

leq (b_1, b_2)    = if b_1 == 0 then 1 else b_2
```

Two-bit operations

How many functions of type $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$ are there?

$2^4 = 16$

Examples:

```
const0 (b_1, b_2) = 0

const1 (b_1, b_2) = 1

min (b_1, b_2)    = if b_1 == 0 or b_2 == 0 then 0 else 1

max (b_1, b_2)    = if b_1 == 1 or b_2 == 1 then 1 else 0

leq (b_1, b_2)    = if b_1 == 0 then 1 else b_2
```

Two-bit operations

How many functions of type $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$ are there?

$2^4 = 16$

Examples:

```
const0 (b_1, b_2) = 0

const1 (b_1, b_2) = 1

min (b_1, b_2)    = if b_1 == 0 or b_2 == 0 then 0 else 1

max (b_1, b_2)    = if b_1 == 1 or b_2 == 1 then 1 else 0

leq (b_1, b_2)    = if b_1 == 0 then 1 else b_2
```

Two-bit operations

How many functions of type $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$ are there?

$2^4 = 16$

Examples:

```
const0 (b_1, b_2) = 0

const1 (b_1, b_2) = 1

min (b_1, b_2)    = if b_1 == 0 or b_2 == 0 then 0 else 1

max (b_1, b_2)    = if b_1 == 1 or b_2 == 1 then 1 else 0

leq (b_1, b_2)    = if b_1 == 0 then 1 else b_2
```

Two-bit operations

How many functions of type $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$ are there?

$2^4 = 16$

Examples:

```
const0 (b_1, b_2) = 0

const1 (b_1, b_2) = 1

min (b_1, b_2)    = if b_1 == 0 or b_2 == 0 then 0 else 1

max (b_1, b_2)    = if b_1 == 1 or b_2 == 1 then 1 else 0

leq (b_1, b_2)    = if b_1 == 0 then 1 else b_2
```

Interpretation of bit operations

Function of type $\mathbb{B}^n \to \mathbb{B}$ have several interpretations:

- as restrictions of functions of type $\mathbb{N}^n \to \mathbb{N}$
  - in particular, we have that $1 + 1 = 2$, therefore $+$ is *not* such a function
- as functions on numbers *modulo* 2 ($\mathbb{B} = \mathbb{Z}_2$)
  - in particular, we have that $1 + 1 = 0$
- as functions on *truth values* ($\mathbb{B} =$ `{False, True}`)
  - example, $+$ on $\mathbb{Z}_2$ above corresponds to the logical function `xor`

The standard terminology favours the "logical" interpretation.

Interpretation of bit operations

Function of type $\mathbb{B}^n \to \mathbb{B}$ have several interpretations:

- as restrictions of functions of type $\mathbb{N}^n \to \mathbb{N}$
  - in particular, we have that $1 + 1 = 2$, therefore $+$ is *not* such a function
- as functions on numbers *modulo* 2 ($\mathbb{B} = \mathbb{Z}_2$)
  - in particular, we have that $1 + 1 = 0$
- as functions on *truth values* ($\mathbb{B} =$ `{False, True}`)
  - example, $+$ on $\mathbb{Z}_2$ above corresponds to the logical function `xor`

The standard terminology favours the "logical" interpretation.

Interpretation of bit operations

Function of type $\mathbb{B}^n \to \mathbb{B}$ have several interpretations:

- as restrictions of functions of type $\mathbb{N}^n \to \mathbb{N}$
  - in particular, we have that $1 + 1 = 2$, therefore $+$ is *not* such a function
- as functions on numbers *modulo* 2 ($\mathbb{B} = \mathbb{Z}_2$)
  - in particular, we have that $1 + 1 = 0$
- as functions on *truth values* ($\mathbb{B} =$ `{False, True}`)
  - example, $+$ on $\mathbb{Z}_2$ above corresponds to the logical function `xor`

The standard terminology favours the "logical" interpretation.

Interpretation of bit operations

Function of type $\mathbb{B}^n \to \mathbb{B}$ have several interpretations:

- as restrictions of functions of type $\mathbb{N}^n \to \mathbb{N}$
  - in particular, we have that $1 + 1 = 2$, therefore $+$ is *not* such a function
- as functions on numbers *modulo* 2 ($\mathbb{B} = \mathbb{Z}_2$)
  - in particular, we have that $1 + 1 = 0$
- as functions on *truth values* ($\mathbb{B} =$ `{False, True}`)
  - example, $+$ on $\mathbb{Z}_2$ above corresponds to the logical function `xor`

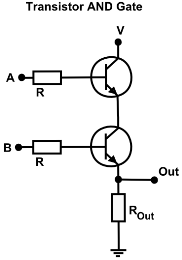
The standard terminology favours the "logical" interpretation.

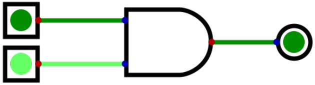
The AND gate

The and function should probably be better called `min`, and is usually denoted by $\wedge$, $\cdot$, or just juxtaposition (as we normally do in the case of multiplication). The truth table is

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

and is implemented by an AND gate:





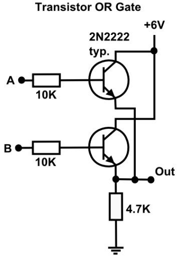
(https://commons.wikimedia.org/wiki/File:TransistorANDgate.png)

The OR gate

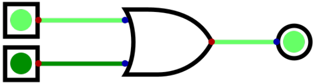
The or function should probably be better called max, and is usually denoted by ∨, |, or +. The truth table is

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

and is implemented by an OR gate:



(https://commons.wikimedia.org/wiki/File:Transistor_OR_Gate.png)

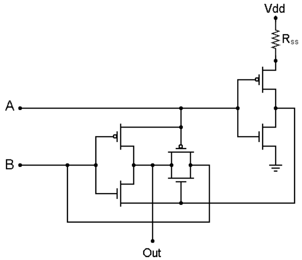


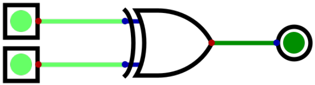
The XOR gate

The `xor` function should probably be better called `not equal`, and is usually denoted by ⊕. The truth table is

x	y	x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0

and is implemented by an XOR gate:





(`https://commons.wikimedia.org/wiki/File:TransmissionCmosXORGate.png`)

Digital: digital logic designer and simulator

Digital

Implementing XOR

Exercise: implement XOR in Digital, using NOT, AND, and OR gates.

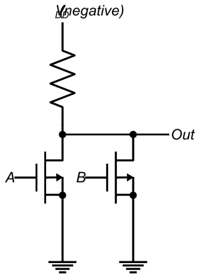
Implementing OR

Exercise: implement OR in Digital, using NOT and AND gates.

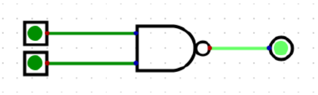
The truth table of the nand function is:

x	y	x nand y
0	0	1
0	1	1
1	0	1
1	1	0

and is implemented by a NAND gate:



PMOS NAND gate



Implementing NAND

Exercise: implement NAND using NOT and AND gates.

Implementing using NAND

Exercise: implement the NOT, AND, and OR gates using only NAND gates.

Sum-of-terms expressions for boolean functions

Consider the truth table of a boolean function `f` of `n` variables:

x_1	x_2	...	x_n	$f(x_1, x_2, ..., x_n)$
0	0	...	0	f(0, 0, ..., 0)
0	0	...	1	f(0, 0, ..., 1)
...	...	...	...	...
1	1	1	1	f(1, 1, ..., 1)

Each row of the table can be described by a *minterm*, a conjunction of the variables in which the variables corresponding to 0s are negated. Thus, the minterm correspoding to row `r` is

```
rₜₕ minterm = b₁b₂...bₙ where
  bᵢ = xᵢ,  if xᵢ = 1 in the rₜₕ row
     = ¬xᵢ, otherwise
```

A formula for `f` is then given by the *disjunction* of minterms corresponding to rows in which `f` has the value 1. A formula in this form is called a *sum-of-terms* expression.

Sum-of-terms expressions for boolean functions

Consider the truth table of a boolean function f of n variables:

x_1	x_2	...	x_n	$f(x_1, x_2, ..., x_n)$
0	0	...	0	f(0, 0, ..., 0)
0	0	...	1	f(0, 0, ..., 1)
...	...	...	...	...
1	1	1	1	f(1, 1, ..., 1)

Each row of the table can be described by a *minterm*, a conjunction of the variables in which the variables corresponding to 0s are negated. Thus, the minterm correspoding to row r is

```
rₜₕ minterm = b₁b₂...bₙ where
  bᵢ = xᵢ,  if xᵢ = 1 in the rₜₕ row
     = ¬xᵢ, otherwise
```

A formula for f is then given by the *disjunction* of minterms corresponding to rows in which f has the value 1. A formula in this form is called a *sum-of-terms* expression.

Sum-of-terms expressions for boolean functions

Consider the truth table of a boolean function `f` of `n` variables:

x_1	x_2	...	x_n	$f(x_1, x_2, ..., x_n)$
0	0	...	0	f(0, 0, ..., 0)
0	0	...	1	f(0, 0, ..., 1)
...	...	...	...	...
1	1	1	1	f(1, 1, ..., 1)

Each row of the table can be described by a *minterm*, a conjunction of the variables in which the variables corresponding to 0s are negated. Thus, the minterm correspoding to row `r` is

```
rₜₕ minterm = b₁b₂...bₙ where
  bᵢ = xᵢ,  if xᵢ = 1 in the rₜₕ row
     = ¬xᵢ, otherwise
```

A formula for `f` is then given by the *disjunction* of minterms corresponding to rows in which `f` has the value 1. A formula in this form is called a *sum-of-terms* expression.

Example

Find a sum-of-terms expression for the boolean function

```
f(x₁, x₂, x₃) = 1, if x₁ < x₃
              = 0, otherwise
```

Solution:

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$f(x_1, x_2, x_3) = \overline{x_1}\,\overline{x_2}x_3 + \overline{x_1}x_2x_3$

Example

Find a sum-of-terms expression for the boolean function

```
f(x₁, x₂, x₃) = 1, if x₁ < x₃
              = 0, otherwise
```

Solution:

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$f(x_1, x_2, x_3) = \overline{x_1}\,\overline{x_2}x_3 + \overline{x_1}x_2x_3$

Properties of boolean operations

Boolean operations have a number of properties, similar (but not identical) to the familiar arithmetic ones:

- *commutativity*: $xy = yx$, $x + y = y + x$
- *associativity*: $x(yz) = (xy)z$, $x + (y + z) = (x + y) + z$
- *distributivity*: $x(y + z) = xy + xz$, $x + yz = (x + y)(x + z)$ (!)
- *idempotence*: $xx = x$, $x + x = x$
- *unit elements*: $x1 = x$, $x + 0 = x$
- *duality* (*de Morgan*): $\overline{(x + y)} = \bar{x}\,\bar{y}$, $\overline{xy} = \bar{x} + \bar{y}$
- *inverses*: $x + \bar{x} = 1$ (*law of excluded middle*), $x\bar{x} = 0$ (*law of contradiction*), $\bar{\bar{x}} = x$ (*law of double negation*)

Simplifying boolean expressions

The properties of boolean operations can be used to simplify expressions, which can translate in more efficient implementations.

Example: simplify the expression obtained for f above. Solution:

$$
\begin{aligned}
& \overline{x_1}\,\overline{x_2}x_3 + \overline{x_1}x_2x_3 \\
= \quad & \{\textit{commutativity, associativity}\} \\
& \overline{x_1}x_3\overline{x_2} + \overline{x_1}x_3x_2 \\
= \quad & \{\textit{distributivity}\} \\
& \overline{x_1}x_3(\overline{x_2} + x_2) \\
= \quad & \{\textit{excluded middle}\} \\
& \overline{x_1}x_3 1 \\
= \quad & \{\textit{unit}\} \\
& \overline{x_1}x_3
\end{aligned}
$$

Simplifying boolean expressions

The properties of boolean operations can be used to simplify expressions, which can translate in more efficient implementations.

Example: simplify the expression obtained for f above. Solution:

$$
\begin{array}{cl}
 & \overline{x_1}\,\overline{x_2}x_3 + \overline{x_1}x_2x_3 \\
= & \{\textit{commutativity, associativity}\} \\
 & \overline{x_1}x_3\overline{x_2} + \overline{x_1}x_3x_2 \\
= & \{\textit{distributivity}\} \\
 & \overline{x_1}x_3(\overline{x_2} + x_2) \\
= & \{\textit{excluded middle}\} \\
 & \overline{x_1}x_3 1 \\
= & \{\textit{unit}\} \\
 & \overline{x_1}x_3
\end{array}
$$

Karnaugh maps

- *Karnaugh maps* are an alternative tabular representation of logical functions.
- In a Karnaugh map minterms that differ in only one variable are adjacent.
- The goal of using a Karnaugh map is to obtain the simplest sum of terms expression.
  - Karnaugh maps are an alternative to the equational form of simplification.

Karnaugh maps

- Let $f\colon \mathbb{B}^N \to \mathbb{B}$.
- Let R and C be such that
  - $R = C = N/2$, if N is even
  - $R + C = N$, $\mid R - C \mid = 1$, if N is odd
- We associate the variables x_1, ..., x_R to the rows and x_{R+1}, ..., x_N to the columns of the Karnaugh map.
- The Karnaugh map will have 2^R rows and 2^C columns. Each row is labelled with R bits and each column is labelled with C bits in *Gray code*.

Gray code

- The *Gray code* is a method of enumerating the 2^N possible binary numbers representable with N bits such that successive representations differ in *only one bit*.
- The enumeration can be seen as a table with N columns and 2^N rows.
- For $N = 1$, the table is just

$$\begin{array}{c} 0 \\ 1 \end{array}$$

- For $N > 1$, the table is obtained by taking the table for $N - 1$ and *reflecting* it across the last line. The result will then have $N - 1$ columns and 2^N rows. We extend this with a first column consisting of 2^{N-1} zeros, followed by 2^{N-1} ones.
- Example: $N = 2$

$$\begin{array}{c} 0 \\ 1 \\ \\ \\ \end{array} \Rightarrow \begin{array}{c} 0 \\ 1 \\ 1 \\ 0 \end{array} \Rightarrow \begin{array}{cc} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{array}$$

Gray code

- The *Gray code* is a method of enumerating the 2^N possible binary numbers representable with N bits such that successive representations differ in *only one bit*.
- The enumeration can be seen as a table with N columns and 2^N rows.
- For $N = 1$, the table is just

$$\begin{array}{c} 0 \\ 1 \end{array}$$

- For $N > 1$, the table is obtained by taking the table for $N - 1$ and *reflecting* it across the last line. The result will then have $N - 1$ columns and 2^N rows. We extend this with a first column consisting of 2^{N-1} zeros, followed by 2^{N-1} ones.
- Example: $N = 2$

$$\begin{array}{c} 0 \\ 1 \\ \\ \\ \end{array} \Rightarrow \begin{array}{c} 0 \\ 1 \\ 1 \\ 0 \end{array} \Rightarrow \begin{array}{cc} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{array}$$

Gray code

- The *Gray code* is a method of enumerating the 2^N possible binary numbers representable with N bits such that successive representations differ in *only one bit*.
- The enumeration can be seen as a table with N columns and 2^N rows.
- For $N = 1$, the table is just

$$\begin{array}{c} 0 \\ 1 \end{array}$$

- For $N > 1$, the table is obtained by taking the table for $N - 1$ and *reflecting* it across the last line. The result will then have $N - 1$ columns and 2^N rows. We extend this with a first column consisting of 2^{N-1} zeros, followed by 2^{N-1} ones.
- Example: $N = 2$

$$\begin{array}{c} 0 \\ 1 \\ \\ \\ \end{array} \Rightarrow \begin{array}{c} 0 \\ 1 \\ 1 \\ 0 \end{array} \Rightarrow \begin{array}{cc} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{array}$$

Gray code

- The *Gray code* is a method of enumerating the 2^N possible binary numbers representable with N bits such that successive representations differ in *only one bit*.
- The enumeration can be seen as a table with N columns and 2^N rows.
- For $N = 1$, the table is just

$$\begin{array}{c} 0 \\ 1 \end{array}$$

- For $N > 1$, the table is obtained by taking the table for $N - 1$ and *reflecting* it across the last line. The result will then have $N - 1$ columns and 2^N rows. We extend this with a first column consisting of 2^{N-1} zeros, followed by 2^{N-1} ones.
- Example: $N = 2$

$$\begin{array}{c} 0 \\ 1 \\ \\ \\ \end{array} \Rightarrow \begin{array}{c} 0 \\ 1 \\ 1 \\ 0 \end{array} \Rightarrow \begin{array}{cc} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{array}$$

Gray code

- The *Gray code* is a method of enumerating the 2^N possible binary numbers representable with N bits such that successive representations differ in *only one bit*.
- The enumeration can be seen as a table with N columns and 2^N rows.
- For $N = 1$, the table is just

$$\begin{array}{c} 0 \\ 1 \end{array}$$

- For $N > 1$, the table is obtained by taking the table for $N - 1$ and *reflecting* it across the last line. The result will then have $N - 1$ columns and 2^N rows. We extend this with a first column consisting of 2^{N-1} zeros, followed by 2^{N-1} ones.
- Example: $N = 2$

$$\begin{array}{c} 0 \\ 1 \\ \\ \\ \end{array} \Rightarrow \begin{array}{c} 0 \\ 1 \\ 1 \\ 0 \end{array} \Rightarrow \begin{array}{cc} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{array}$$

Gray code example

0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

Karnaugh maps

- We associate the variables x_1, ..., x_R to the rows and x_{R+1}, ..., x_N to the columns of the Karnaugh map.
- The Karnaugh map will have 2^R rows and 2^C columns. Each row is labelled with R bits and each row is labelled with C bits in *Gray code*.
- We find *maximal* blocks of *adjacent 1s of size power of two*. The blocks may overlap, as long as they differ in at least one cell.
- The minterm associated to each block is given by the variables that remain constant across the block.
- The sum-of-terms expression is then the sum of the minterms.

Example

(source Wikipedia)

A	B	C	D	$f(A, B, C, D)$
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

1. Derive the sum of terms expression from the truth table.
2. Derive a simplified sum-of-terms expression using the Karnaugh map representation of f.