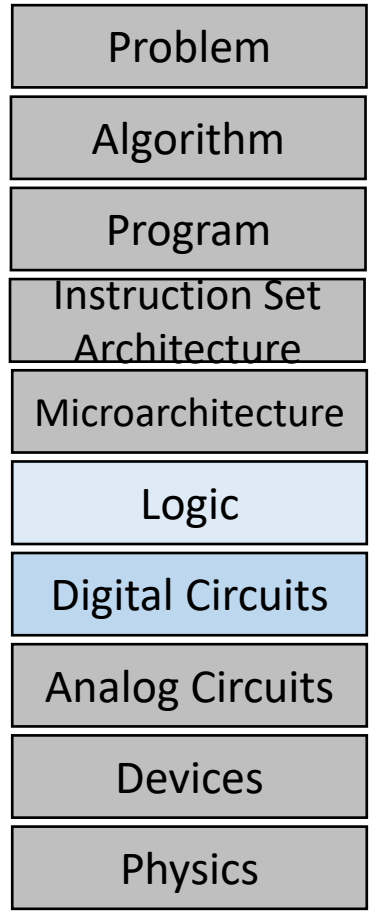
Universität Bremen

Logic Design
Combinational Circuits

Prof. Dr. Rolf Drechsler
Dr. Muhammad Hassan
M.Sc. Jan Zielasko
M.Sc. Milan Funck



From the Last lecture …

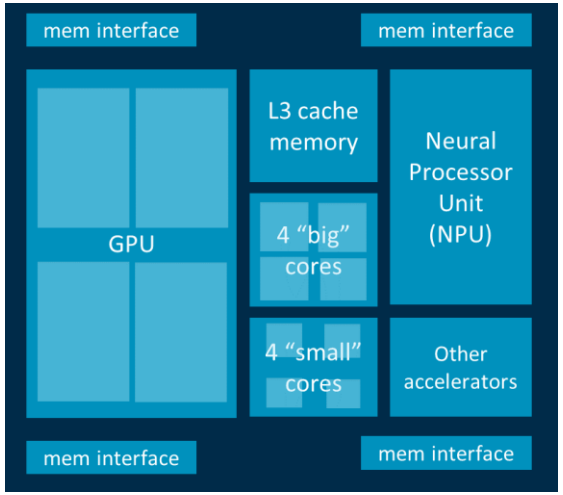
Abstractions

- Problem
- Algorithm
- Program
- Instruction Set Architecture
- Microarchitecture
- Logic
- Digital Circuits
- Analog Circuits
- Devices
- Physics

Architecture

RISC-V

Modern System on Chip



Universität Bremen

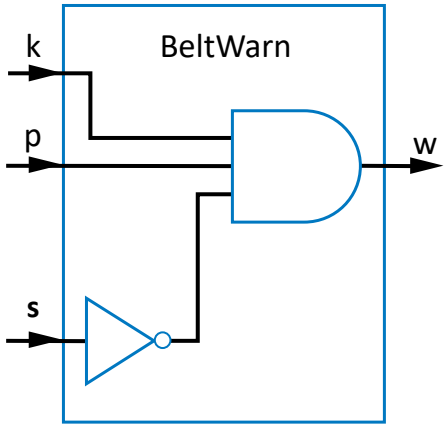
Outline

- Introduction
- Boolean Equations
- Truth Tables
- Boolean Algebra
- Gates
- Combinational Logic
  - Decoders
  - Multiplexors
  - Two-level Logic
- Hardware Description Languages
  - Verilog
  - VHDL

Introduction

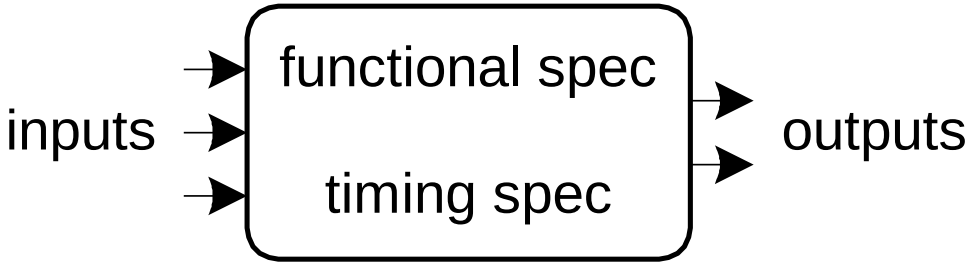
- Seat Belt Warning Light System
- Design circuit for warning light
  - Sensors
    - s=1: seat belt fastened
    - k=1: key inserted
    - p=1: person in seat
- Capture Boolean equation
  - person in seat, and seat belt not fastened, and key inserted

    w = p AND NOT(s) AND k
- Convert equation to circuit

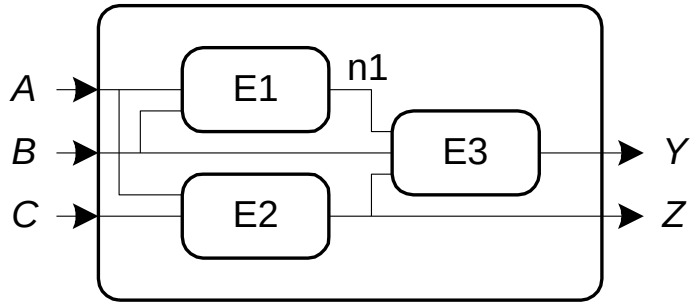


Logic Circuit

- A logic circuit is composed of:
  - Inputs
  - Outputs
  - Functional specification
  - Timing specification

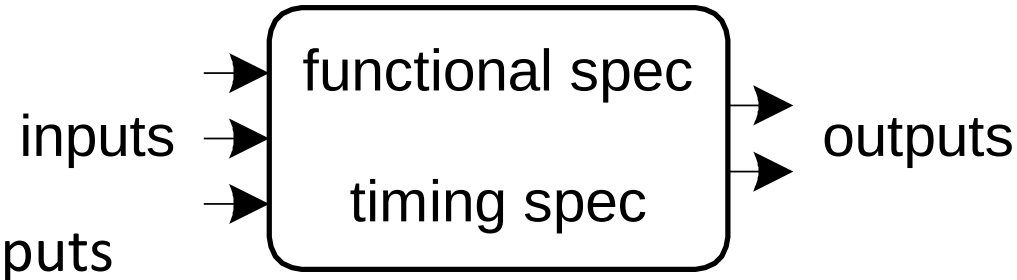


- Nodes
  - Inputs: A, B, C
  - Outputs: Y, Z
  - Internal: n1
- Circuit elements
  - E1, E2, E3 (Each a circuit)



Types of Logic Circuits

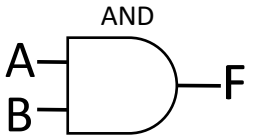
- Combinational Logic
  - Memoryless
  - Outputs determined by current values of inputs
- Sequential Logic
  - Has memory
  - Outputs determined by previous and current values of inputs



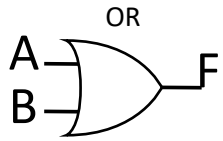
Boolean Algebra and its Relation to Digital Circuits

- Boolean Algebra
  - Variables represent 0 or 1 only
  - Operators return 0 or 1 only
  - Basic operators
    - AND: A AND B returns 1 only when both A=1 and B=1
    - OR: A OR B returns 1 if either (or both) A=1 or B=1
    - NOT: NOT a returns the opposite of a (1 if A=0, 0 if A=1)

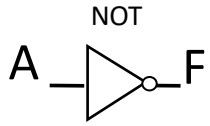
A	B	F
0	0	0
0	1	0
1	0	0
1	1	1



A	B	F
0	0	0
0	1	1
1	0	1
1	1	1



A	F
0	1
1	0

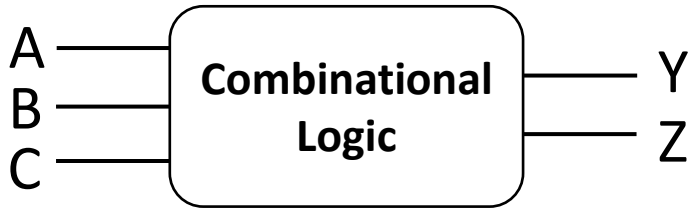


Boolean Equations

- Functional specification of outputs in terms of inputs
- Example:

$$Y = F(A, B, C)$$

$$Z = F(A, B, C)$$



Boolean Algebra Terminology

- Example equation: **F(A,B,C) = A'BC + ABC' + AB + C**
- Variable
  - Represents a value (0 or 1)
  - Three variables: A, B, and B
- Literal
  - Appearance of a variable, in true or complemented form
  - Nine literals: A', B, C, A, B, C', A, B, and C
- Product term
  - Product of literals
  - Four product terms: A'BC, ABC', AB, C
- Sum-of-products
  - Equation written as OR of product terms only
  - Above equation is in sum-of-products form

Universität Bremen

Boolean Algebra Properties

- Commutative
  - a + b = b + a
  - a * b = b * a
- Distributive
  - a * (b + c) = a * b + a * c
  - a + (b * c) = (a + b) * (a + c)
- Associative
  - (a + b) + c = a + (b + c)
  - (a * b) * c = a * (b * c)
- Identity
  - 0 + a = a + 0 = a
  - 1 * a = a * 1 = a
- Complement
  - a + a’ = 1
  - a * a’ = 0
- To prove, just evaluate all possibilities

Boolean Algebra Properties – Example

- Show **abc + abc’ = ab.**
- Use first distributive property
  - abc + abc’ = ab(c+c’).
- Complement property
  - Replace c+c’ by 1: ab(c+c’) = ab(1).
- Identity property
  - ab(1) = ab*1 = ab.

- Distributive
  - a * (b + c) = a * b + a * c
  - a + (b * c) = (a + b) * (a + c)
- Complement
  - a + a’ = 1
  - a * a’ = 0
- Identity
  - 0 + a = a + 0 = a
  - 1 * a = a * 1 = a

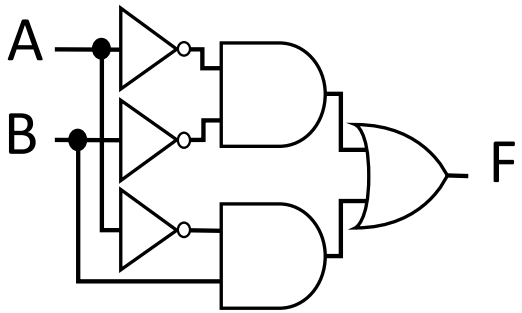
Boolean Algebra: Additional Properties

- Null elements
  - a + 1 = 1
  - a * 0 = 0
- Idempotent Law
  - a + a = a
  - a * a = a
- Involution Law
  - (a')' = a
- DeMorgan's Law
  - (a + b)' = a'b'
  - (ab)' = a' + b'
  - Very useful!
- To prove, just evaluate all possibilities

Representations of Boolean Functions

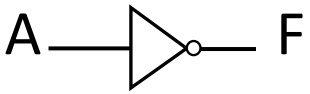
- A function can be represented in different ways
  - Natural language, Equation, Circuit, and Truth Table …
- Natural language: F outputs 1 when A is 0 and B is 0, or when A is 0 and B is 1.
- **F(A,B) = A'B' + A'B**

Is their output same?



A	B	F
0	0	1
0	1	1
1	0	0
1	1	0

- Natural language: F outputs 1 when A is 0, regardless of B's value
- **F(A,B) = A'**



Truth Table Representation of Boolean Functions

- Define value of F for each possible combination of input values
  - 2-input function: 4 rows
    - 2^n

A	B	F
0	0	1
0	1	1
1	0	0
1	1	0

  - 3-input function: 8 rows

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Truth Table Representation of Boolean Functions

- Use truth table to define function F(A,B,C) that is 1 when ABC is 5 or greater in binary

A	B	C	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	**0**	**1**	
1	**1**	**0**	
1	**1**	**1**	

Truth Table Representation of Boolean Functions

- Use truth table to define function F(A,B,C) that is 1 when ABC is 5 or greater in binary

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	**0**	**1**	**1**
1	**1**	**0**	**1**
1	**1**	**1**	**1**

Standard Representation: Truth Table

- How can we determine if two functions are the same?
  - Use algebraic methods
  - But if we failed, does that prove not equal? No.
- Solution: Convert to truth tables
  - Only ONE truth table representation of a given function
- Determine if **F1=AB+A'** is same function as **F2=A'B'+A'B+AB**, by converting each to truth table first

A	B	F1
0	0	1
0	1	1
1	0	0
1	1	1

F1=AB+A'

Same!

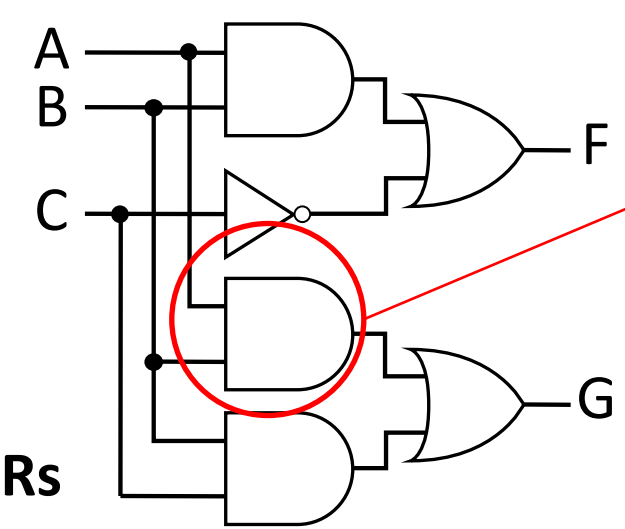
A	B	F2
0	0	1
0	1	1
1	0	0
1	1	1

F2=A'B'+A'B+AB

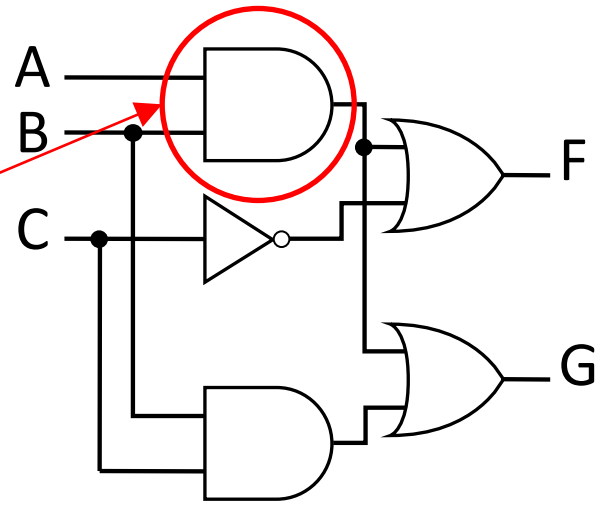
Multiple-Output Circuits

- Many circuits have more than one output
- Can give each a separate circuit, or can share gates
- Example: F = AB + C', G = AB + BC

Two-level logic:
ANDs followed by ORs



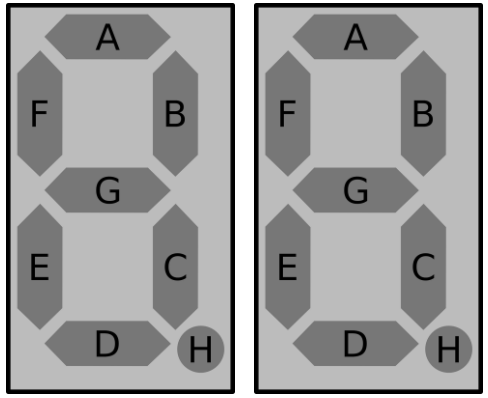
Separate circuits

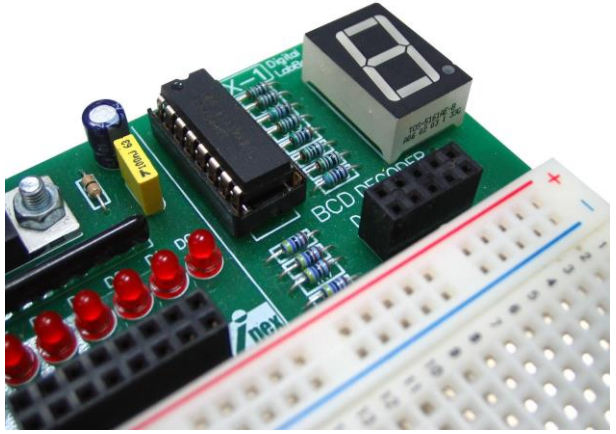


Shared gates

Example – BCD to 7-segment converter

w	x	y	z	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0





A = w’x’y’z’ + w’x’yz’ + w’x’yz +
w’xy’z + w’xyz’ + w’xyz + wx’y’z’
+ wx’y’z

B = w’x’y’z’ + w’x’y’z + w’x’yz’ +
w’x’yz + w’xy’z’ + w’xyz +
wx’y’z’ + wx’y’z

Universität Bremen

Combinational Logic Design Process

- Capture the function
  - Create a truth table or equations, whichever is most natural for the given problem, to describe the desired behavior of the combinational logic.
- Convert to equations
  - This step is only necessary if you captured the function using a truth table instead of equations. Create an equation for each output by ORing all the minterms for that output. Simplify the equations if desired.
- Implement as a gate-based circuit
  - For each output, create a circuit corresponding to the output's equation. (Sharing gates among multiple outputs is OK optionally.)

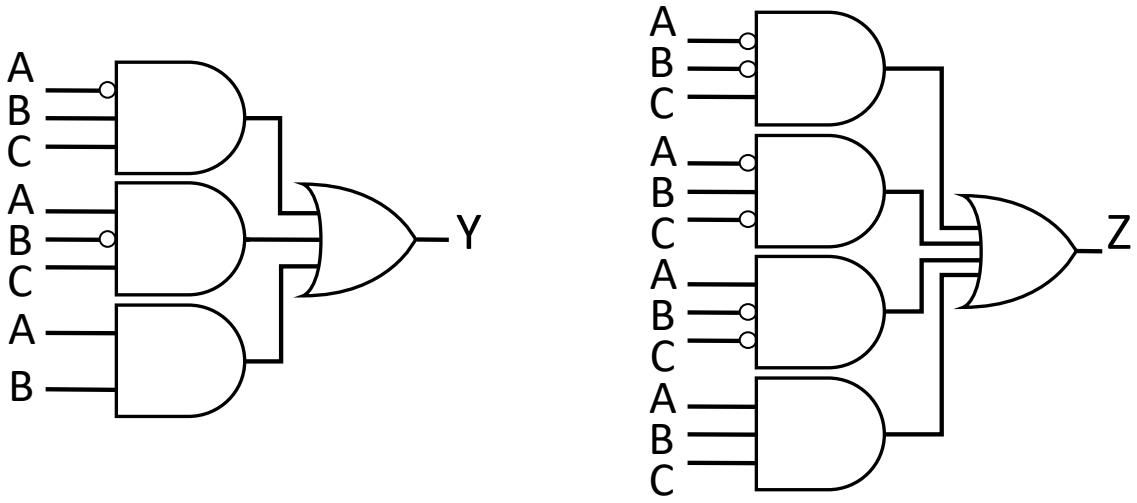
Example – Number of 1s count

- Problem: Output in binary on two outputs YZ (2 bits) the number of 1s on three inputs
  - 010 -> 01 (1)
  - 101 -> 10 (2)
  - 000 -> 00 (0)
- **Step 1:** Capture the function
  - Truth table or equation?
  - Truth table is straightforward
- **Step 2:** Convert to equation
  - Y = A'BC + AB'C + ABC' + ABC
  - Z = A'B'C + A'BC' + AB'C' + ABC

A	B	C	#1s	Y	Z
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	2	1	0
1	0	0	1	0	1
1	0	1	2	1	0
1	1	0	2	1	0
1	1	1	3	1	1

Example cont…

- **Step 3:** Implement as a gate-based circuit

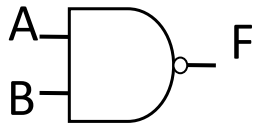


Two-level logic:
ANDs followed by ORs

A	B	C	#1s	Y	Z
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	2	1	0
1	0	0	1	0	1
1	0	1	2	1	0
1	1	0	2	1	0
1	1	1	3	1	1

More Gates

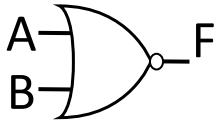
- NAND: Opposite of AND (“NOT AND”)
  - Use-case – Alarm circuits



A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

Any boolean function can be implemented by using a combination of NAND (or NOR) gates. This property is called functional completeness.

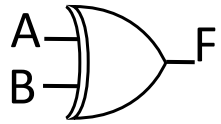
- NOR: Opposite of OR (“NOT OR”)
  - Use-case – Detect all zeros!



A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

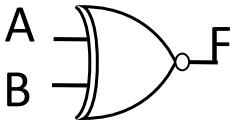
More Gates cont…

- XOR: Exactly 1 input is 1, for 2-input XOR
  - Detecting odd # of 1s
  - Useful for generating “parity” bit common for detecting errors



A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

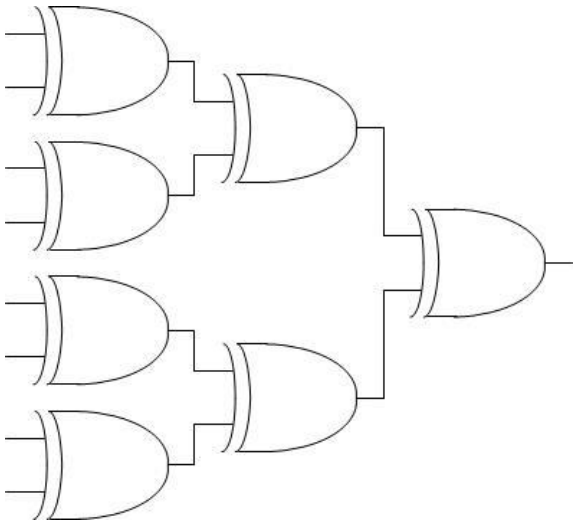
- XNOR: Opposite of XOR (“NOT XOR”)
  - Detecting equality

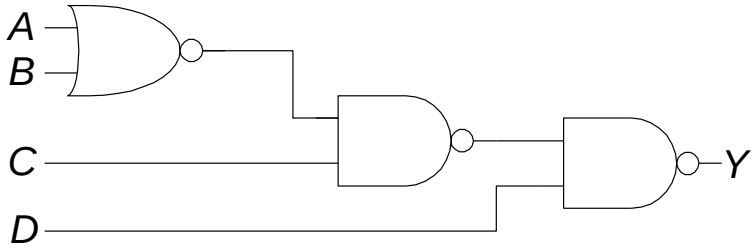


A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

Multilevel Logic

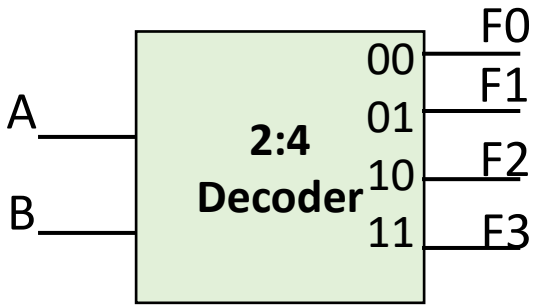
- Complex logic is often built from many stages of simpler gates



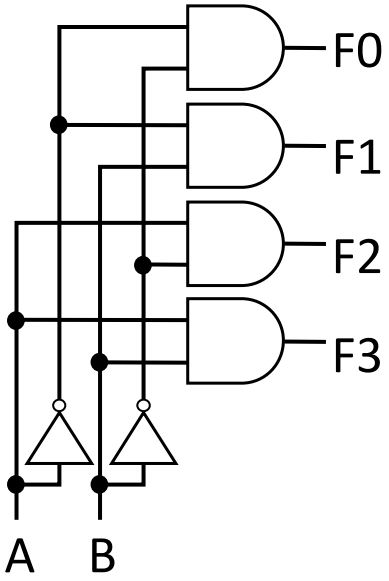


Decoders

- Popular combinational logic building block, in addition to logic gates
  - Converts input binary number to one high output
  - N inputs, 2^N outputs
- Internal design
  - AND gate for each output to detect input combination
- One-hot outputs: only one output **HIGH** at once



A	B	F0	F1	F2	F3
0	0	**1**	0	0	0
0	1	0	**1**	0	0
1	0	0	0	**1**	0
1	1	0	0	0	**1**

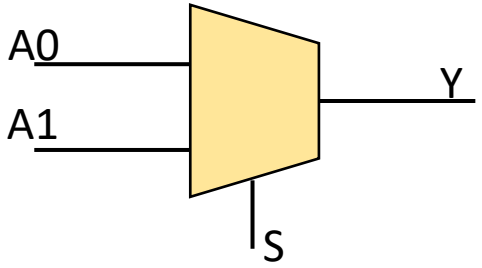


Multiplexer (MUX)

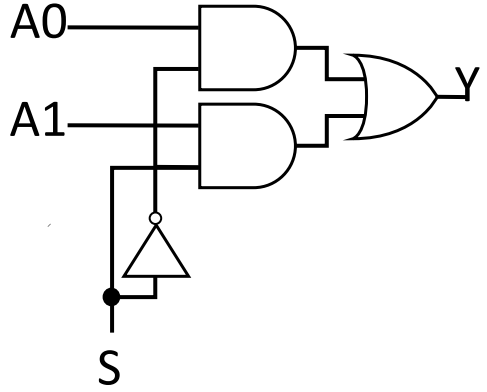
- Mux: Another popular combinational building block
  - Routes one of its N data inputs to its one output, based on binary value of select inputs
    - 4 input mux → needs 2 select inputs to indicate which input to route through
    - 8 input mux → 3 select inputs
    - N inputs → log2(N) selects
  - Like a railyard switch

Multiplexer (MUX) cont …

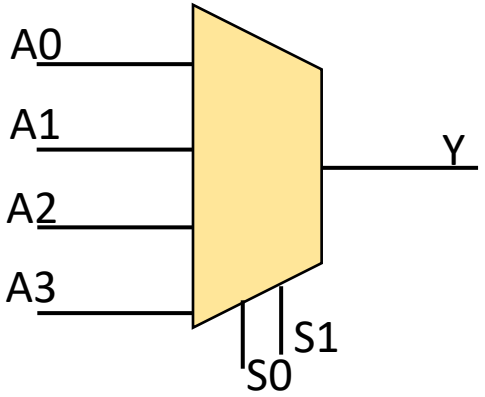
- Select input is $\log_2 N$ bits – control input
- Example
  - 2:1 MUX



S	Y
0	A0
1	A1

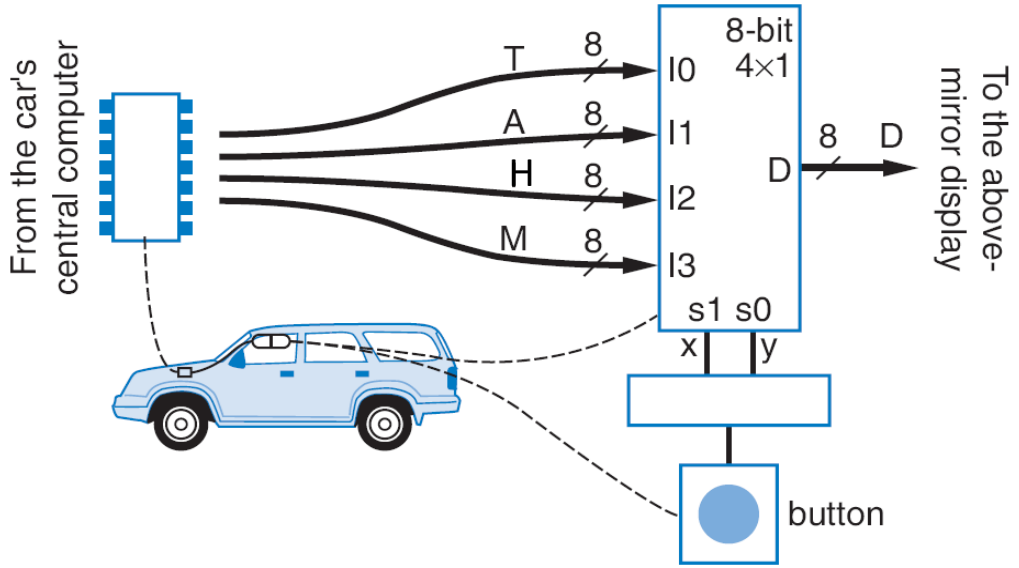


  - 4:1 MUX



S0	S1	Y
0	0	A0
0	1	A1
1	0	A2
1	1	A3

N-bit MUX Example



- Four possible display items
  - Temperature (T), Average KMs-per-litre (A), Humidity (H), and Mileage (M) -- each is 8-bits wide
  - Choose which to display using two inputs x and y
  - Use 8-bit 4x1 mux

Universität Bremen

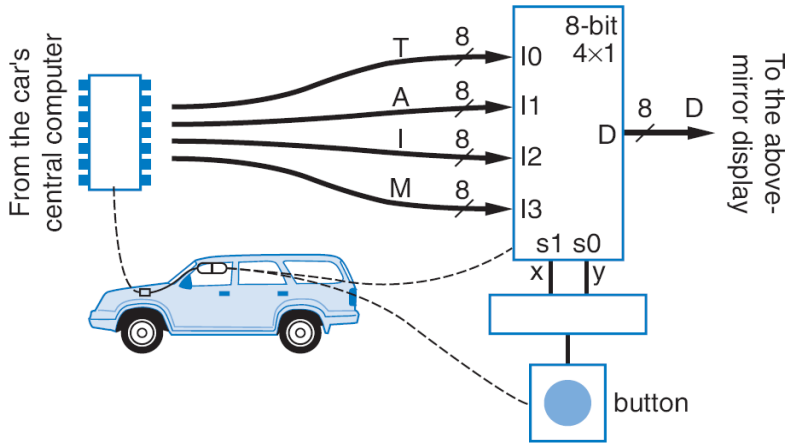
Hardware Description Languages

- Describe hardware at varying levels of abstraction
- Structural description
  - Textual replacement for schematic
  - Hierarchical composition of modules from primitives
- Behavioral/functional description
  - Describe what module does, not how
  - Synthesis generates circuit for module
- Simulation semantics

Hardware Description Languages cont…

- Verilog (circa 1985) - developed by Gateway (absorbed by Cadence)
  - Sim
  - Dela
  - Fair
  - IEEE standard

- **Modellierung digitaler Systeme auf Register-Transfer-Ebene (RTL) mit Verilog HDL (G) 03-IMGS-RTL**
- **SoSe 23**

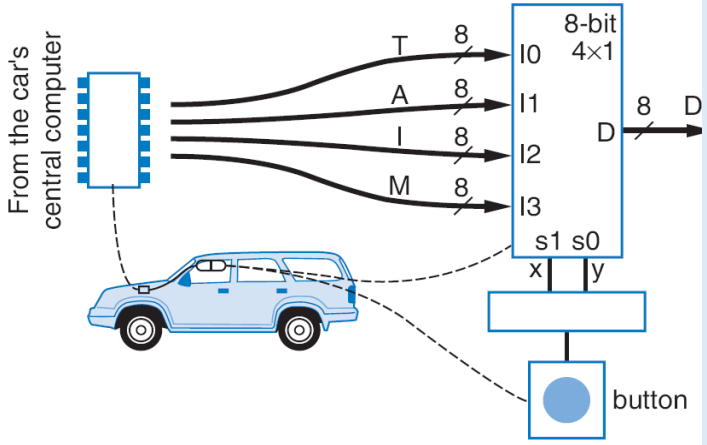



```
module m41 ( I0, I1, I2, I3, s0, s1, D);

input wire [7:0] I0;
input wire [7:0] I1;
input wire [7:0] I2;
input wire [7:0] I3;

input wire [7:0] s0;
input wire [7:0] s1;

output reg [7:0] D;
```

```
always @ (I0 or I1 or I2 or I3 or s0, s1)
begin
        case (s0 | s1)
        2'b00 : D <= I0;
        2'b01 : D <= I1;
        2'b10 : D <= I2;
        2'b11 : D <= I3;
        endcase
end

endmodule
```

Hardware Description Languages cont…

- VHDL (circa 1987) - DoD sponsored standard
  - Similar to Ada (emphasis on re-use and maintainability)
  - Simulation semantics visible
  - Very general but verbose
  - IEEE standard



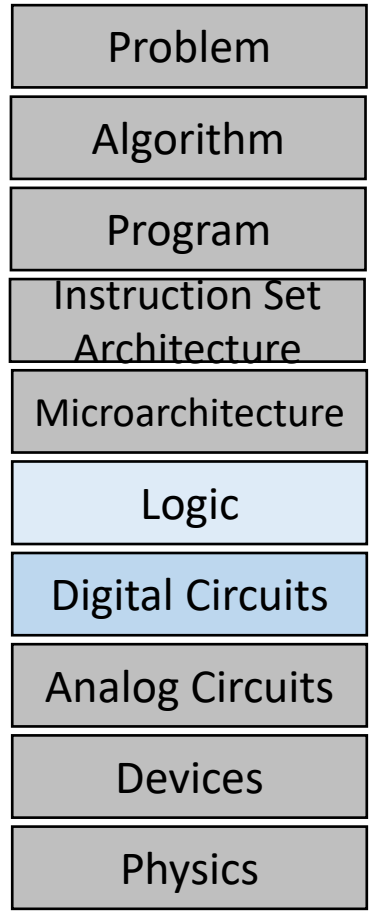

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity mux_4to1 is
 port(

    I0,I1,I2,I3 : in std_logic_vector (7 downto 0);
    S0,S1: in STD_LOGIC;
    D: out std_logic_vector (7 downto 0)
  );
end mux_4to1;
```

```
architecture bhv of mux_4to1 is
begin
process (I0,I1,I2,I3,S0,S1) is
begin
  if (S0 ='0' and S1 = '0') then
     D <= I0;
  elsif (S0 ='1' and S1 = '0') then
     D <= I1;
  elsif (S0 ='0' and S1 = '1') then
     D <= I2;
  else
     D <= I3;
  end if;

end process;
end bhv;
```

Universität Bremen

Logic Design
Combinational Circuits

Prof. Dr. Rolf Drechsler
Dr. Muhammad Hassan
M.Sc. Jan Zielasko
M.Sc. Milan Funck



Universität Bremen

Literature

- D. Patterson, J. Hennessy: Computer Organization and Design RISC-V Edition – The Hardware Software Interface, Elsevier, 2020.
- S. L. Harris, D. Harris: Digital Design and Computer Architecture, RISC-V Edition, Elsevier, 2021.
- ARM University program: Introduction-to-Computer-Architecture-Education.
- J. Teich, C. Haubelt: Digitale Hardware/Software-Systeme – Synthese und Optimierung, Springer Verlag, 2. Auflage, 2007.
- G. De Micheli: Synthesis and Optimization of Digital Circuits, McGraw-Hill, 1994.
- G. D. Hachtel, F. Somenzi: Logic Synthesis and Verification Algorithms, Kluwer, 1996.

Universität Bremen

Literature

- H. Bähring, J. Dunkel, G. Rademacher: Mikrorechner-Systeme: Mikroprozessoren, Speicher, Peripherie, Springer-Verlag, 2. Auflage, 1994.
- J. P. Hayes: Computer Architecture and Organization, McGraw-Hill, 1998.
- M. G. Arnold: Verilog Digital Computer Design: Algorithms to Hardware, Prentice Hall, 1998.
- A. Tanenbaum: Structured Computer Organization, Prentice Hall, 5th Edition, 2006.
- D. A. Patterson, J. L. Hennessy: Computer Organization and Design - The Hardware/Software-Interface, Morgan Kaufmann, 3. Auflage, 2007.
- J. L. Hennessy, D. A. Patterson: Computer Architecture - A Quantitative Approach, Morgan Kaufmann, 4. Auflage, 2007.
- H. Kaeslin: Digital Integrated Circuit Design, From VLSI to CMOS Fabrication, Cambridge University Press, 2008.
- A. Biere, D. Kroening, G. Weissenbacher, C. M. Wintersteiger: Digitaltechnik – Eine praxisnahe Einführung, Springer-Verlag, 2008.