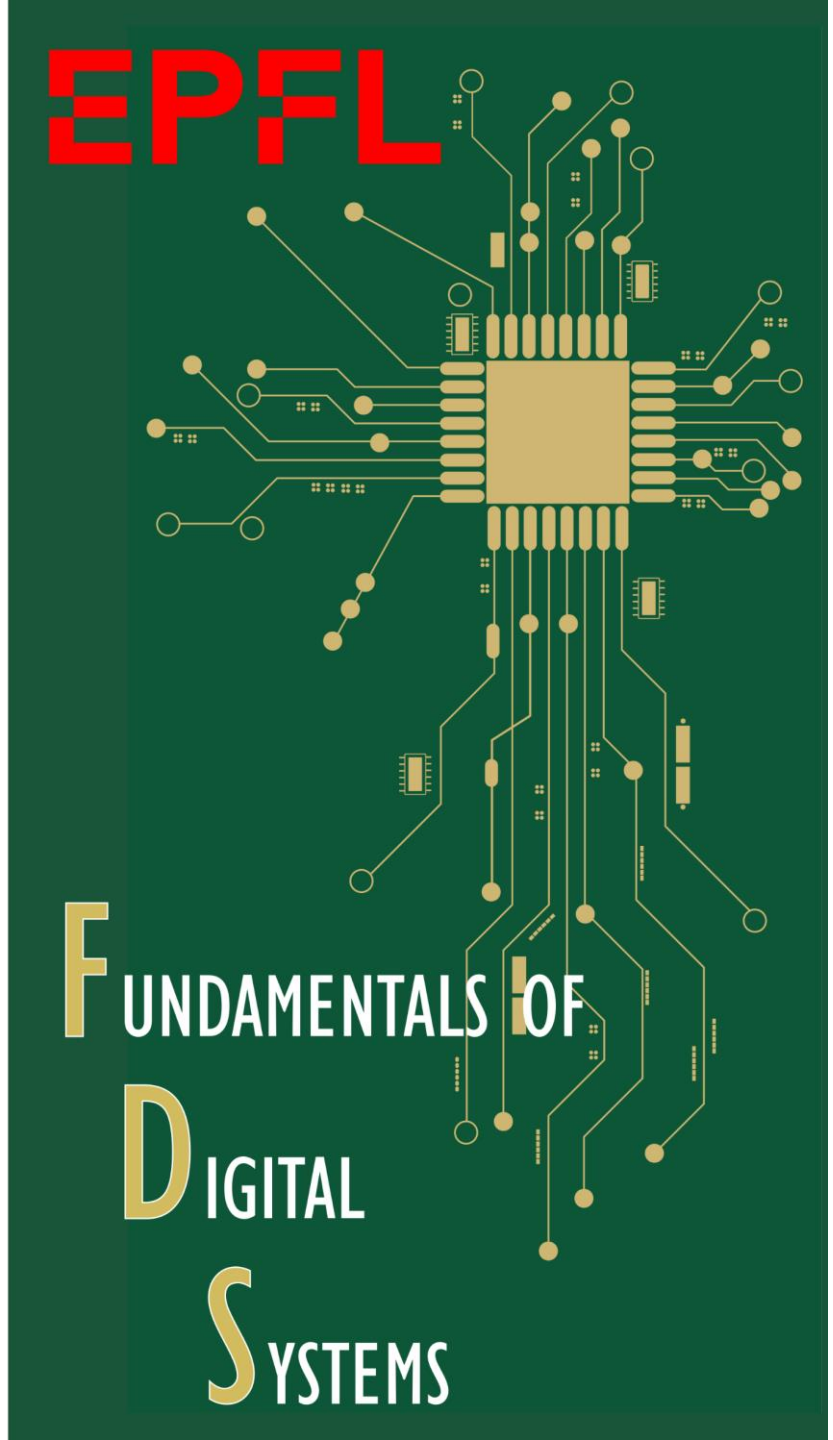


Digital Logic Circuits

Verilog for Combinational Circuits

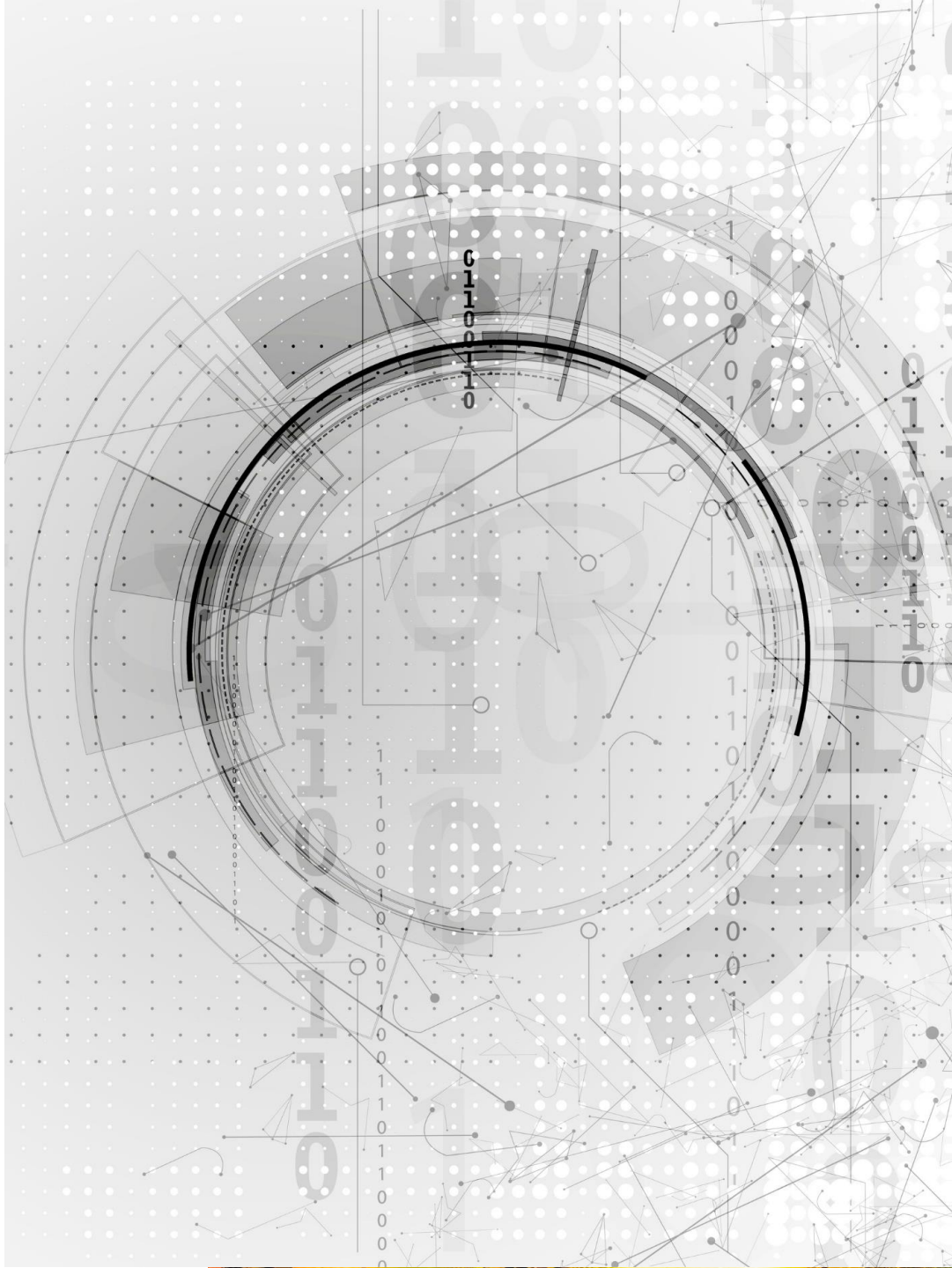
CS-173 Fundamentals of Digital Systems

Mirjana Stojilović

Spring 2025

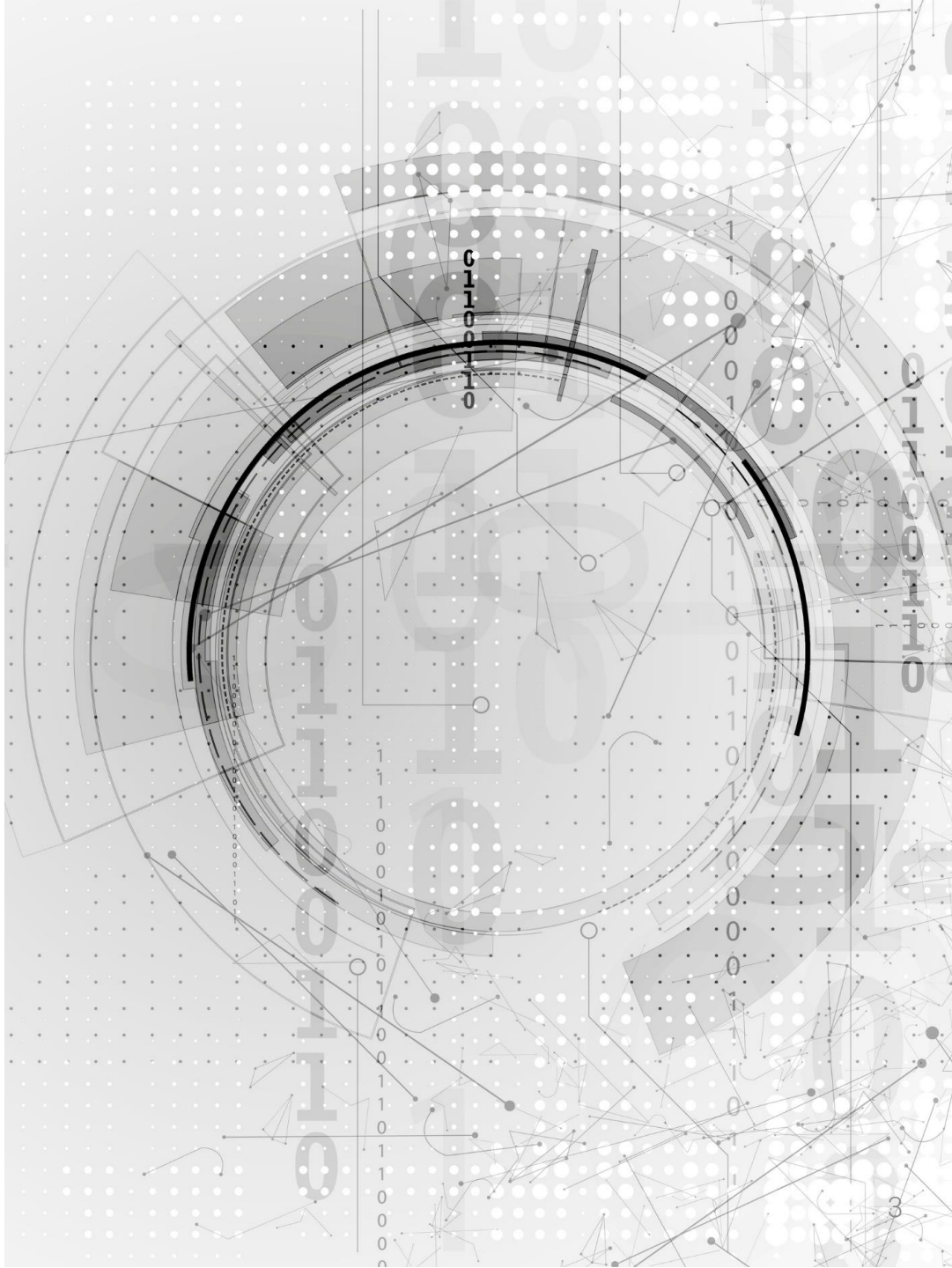
Previously on FDS

Intro to CAD and Verilog



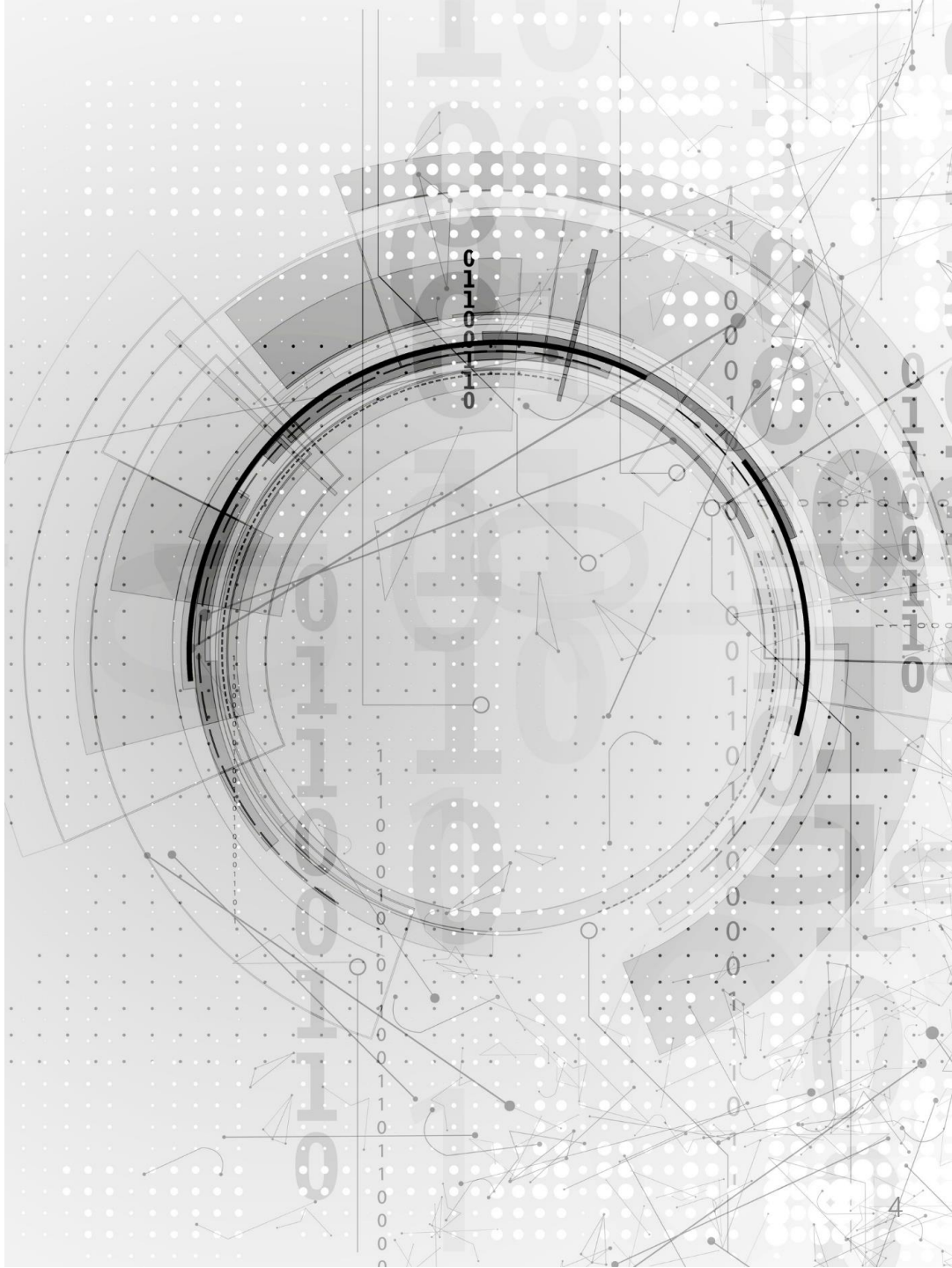
Previously

- Discovered the key phases of computer-aided design (CAD) flow
  - Hardware description language (HDL)
- Learned how to use Verilog HDL for **gate-level modeling** of logic circuits
- Verilog gates, wires, modules, ports, and subcircuits
- Gate-level modeling examples:
  - Full-adder
  - Four-bit ripple-carry adder

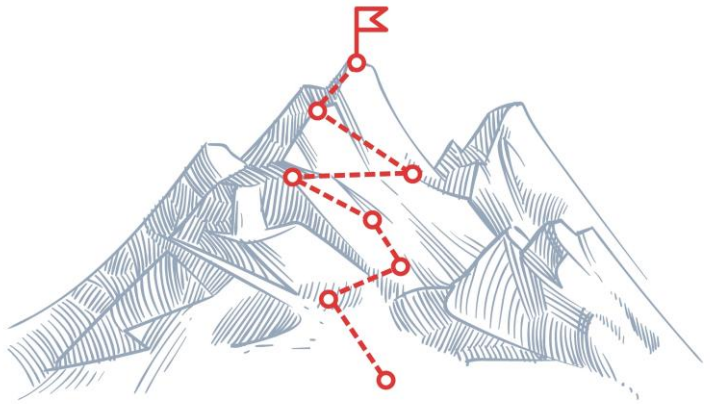


Let’s Talk About…

…Behavioral modeling in Verilog and
some new gates and combinational circuits



Learning Outcomes



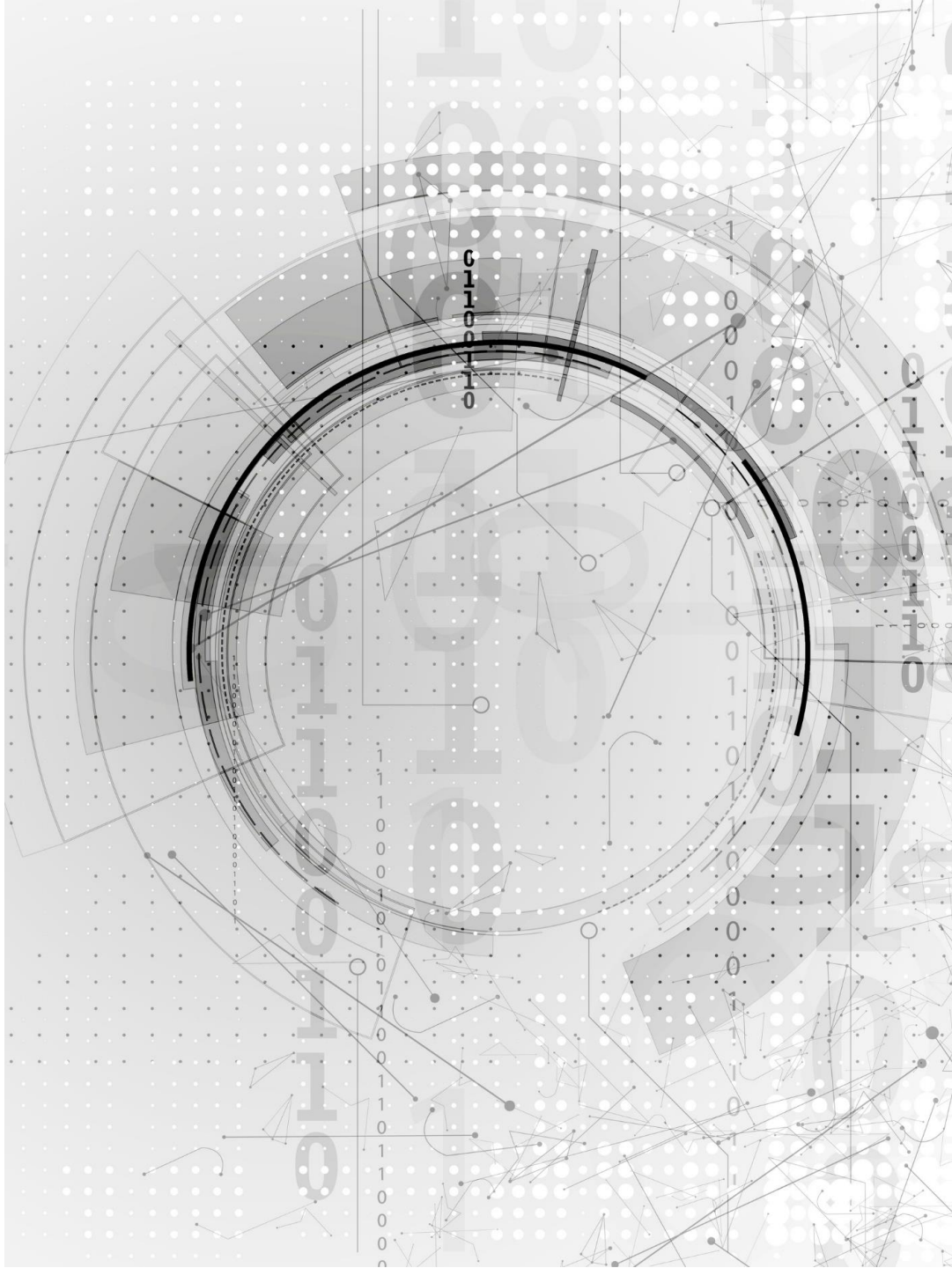
- Discover **tri-state drivers**
- Learn about **high-impedance Z**
- Model shared interconnect (a **bus**) using MUXes or tri-state drivers
- Use Verilog **behavioral** modeling to describe logic circuits
  - Understand the difference between **continuous** assignments and **procedural** statements
  - Use **always** blocks and **if-else** and **switch** statements

Quick Outline

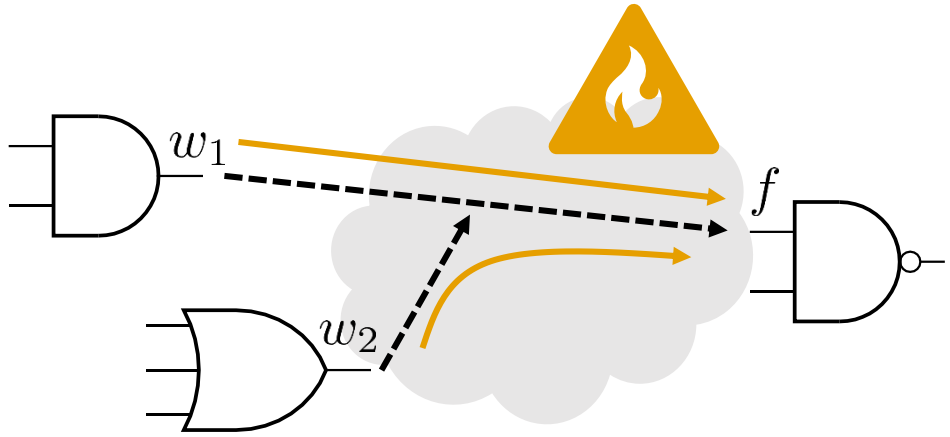
- Tri-state drivers
- Bus
  - With multiplexers
  - With tri-state drivers
- Verilog HDL
  - Logic gates
  - Values: scalar and vector
  - Constants, Concatenation
  - Parameters, Nets

- Behavioral Modeling in Verilog
  - Continuous assignments
  - **assign** keyword
  - Procedural statements
  - **always** block
  - **reg** type for variables
  - **if-else** statement
    - Example: 2-to-1 multiplexer
  - **case** statement
    - Example: full-adder

Tri-State Drivers



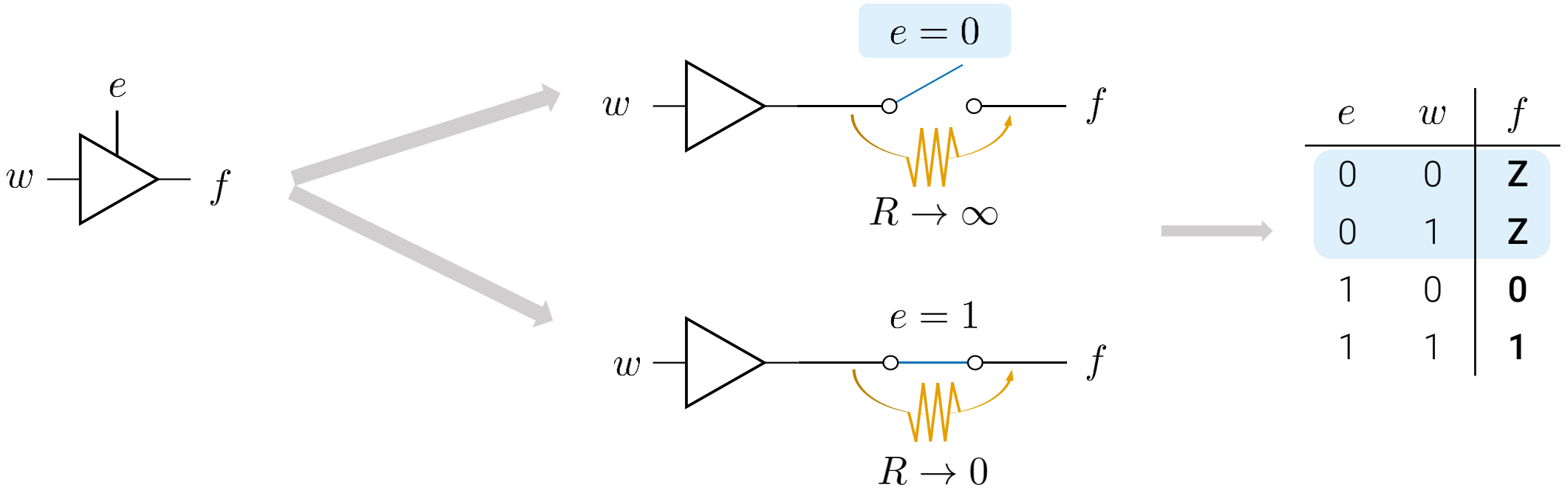
Multiple Gates Driving Same Inputs



- Example: Two logic gates wanting to drive an input of a third logic gate
- **Issue:** The logic gates outputs should not be directly connected
  - If one gate forces '1' while the other forces '0', a low resistance path between the power supply and the ground would be created, and the resulting current would be high. We call that situation a **short-circuit**
- **Solution**: Insert MUXes or tri-state drivers on the conflicting signal paths

Tri-State Drivers

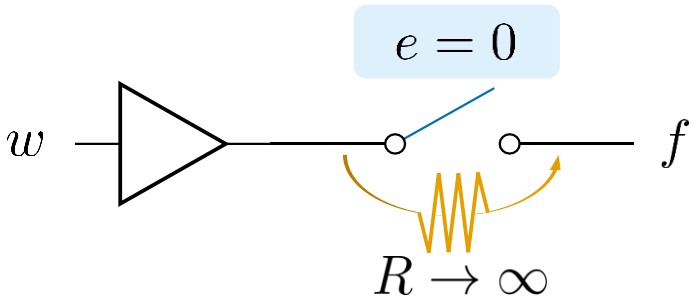
- A tri-state driver has a data input, an output, and an **enable** input



e	w	f
0	0	**Z**
0	1	**Z**
1	0	**0**
1	1	**1**

** In electrical engineering, R stands for resistance; resistance is the real part of the impedance.*

Tri-State Drivers



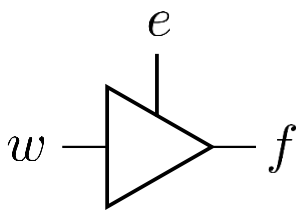
e	w	f
0	0	**Z**
0	1	**Z**
1	0	**0**
1	1	**1**

- When the enable input is inactive, the output is electrically disconnected from the data input; disconnected state is referred to as **high-impedance** state and usually denoted as **Z** (or **z**)
  - Three states of a tri-state driver are logical 0, logical 1, and Z

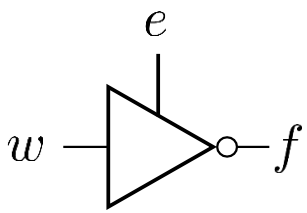
- *In electrical engineering, **impedance** is the opposition to alternating current presented by the combined effect of resistance and reactance in a circuit.*
- *https://en.wikipedia.org/wiki/Electrical_impedance*

Types of Tri-State Drivers

- Enable **active high**

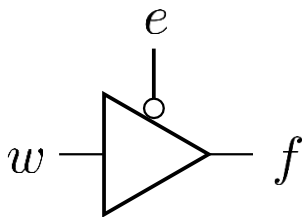


e	w	f
0	0	**Z**
0	1	**Z**
1	0	**0**
1	1	**1**

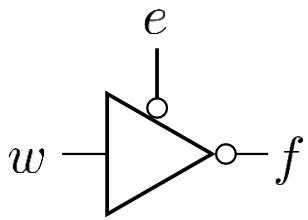


e	w	f
0	0	**Z**
0	1	**Z**
1	1	**0**
1	0	**1**

- Enable **active low**

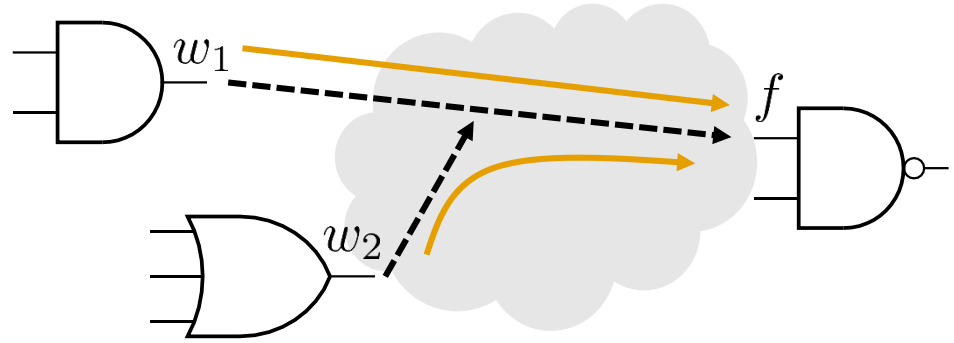


e	w	f
0	0	**0**
0	1	**1**
1	0	**Z**
1	1	**Z**

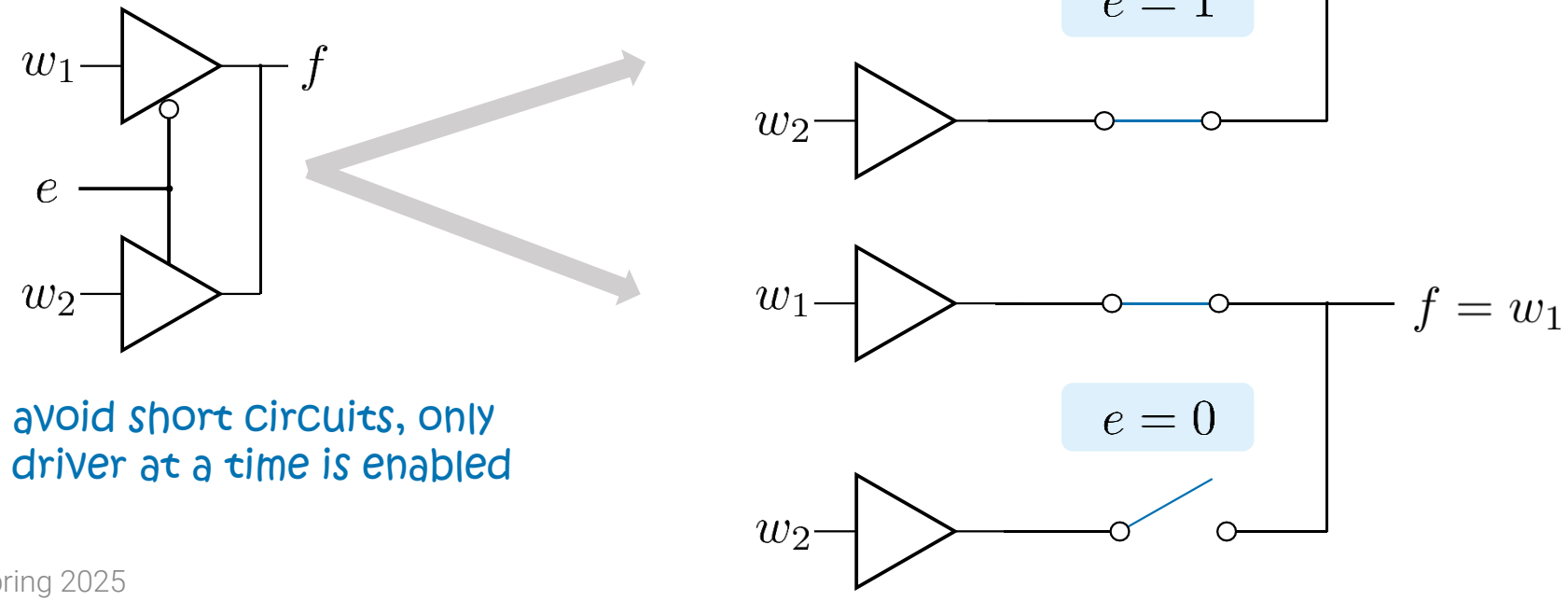


e	w	f
0	0	**1**
0	1	**0**
1	0	**Z**
1	1	**Z**

Example: Tri-State Drivers



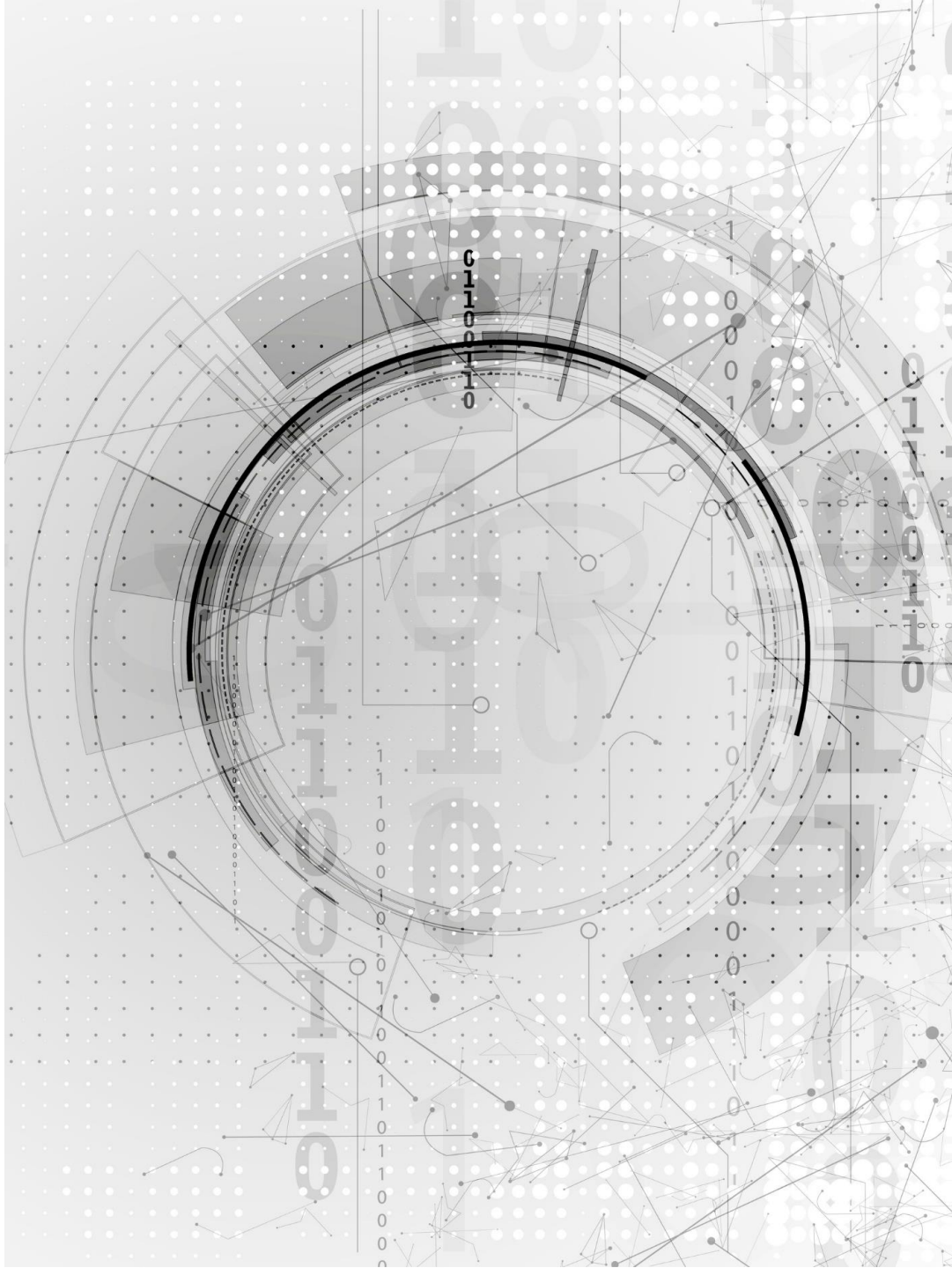
- Two gate outputs driving a wire



To avoid short circuits, only one driver at a time is enabled

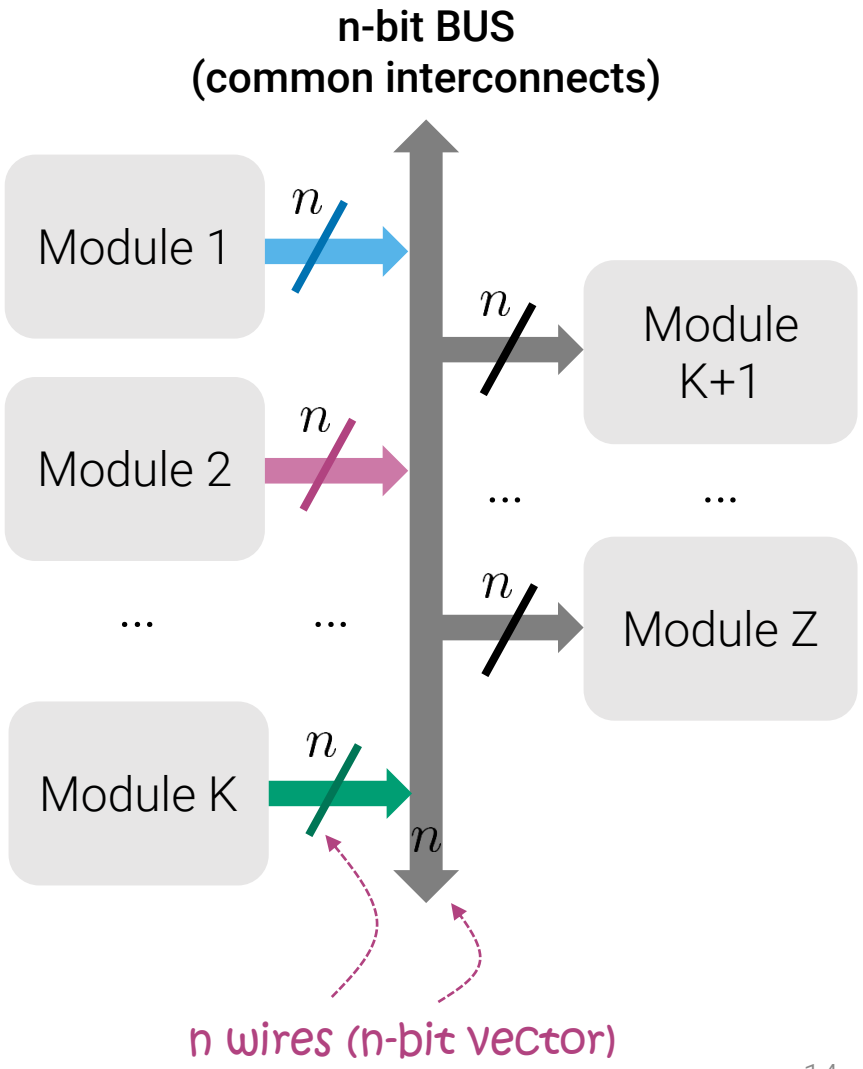
Bus

- With MUXes
- With tri-state drivers



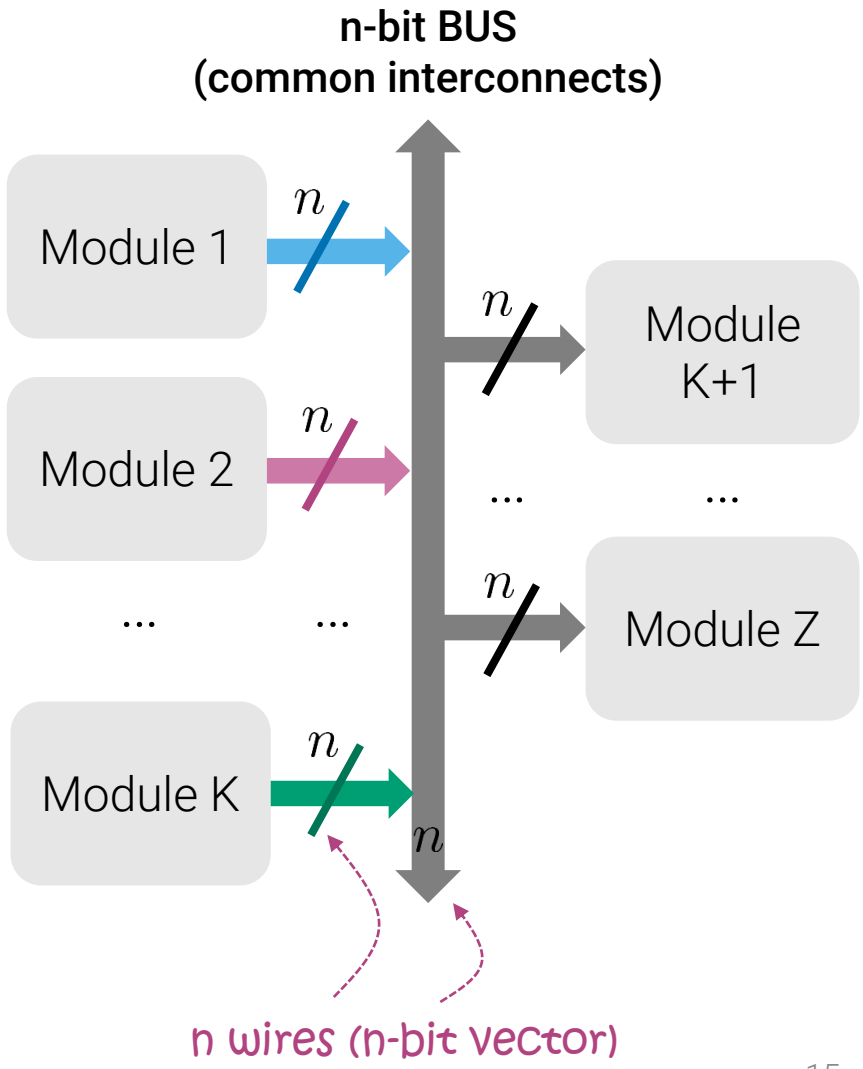
Bus

- Digital systems are commonly composed of several modules exchanging data by means of a **common** set of interconnects (wires)
- The set of wires grouped under a common name is referred to as a **bus**

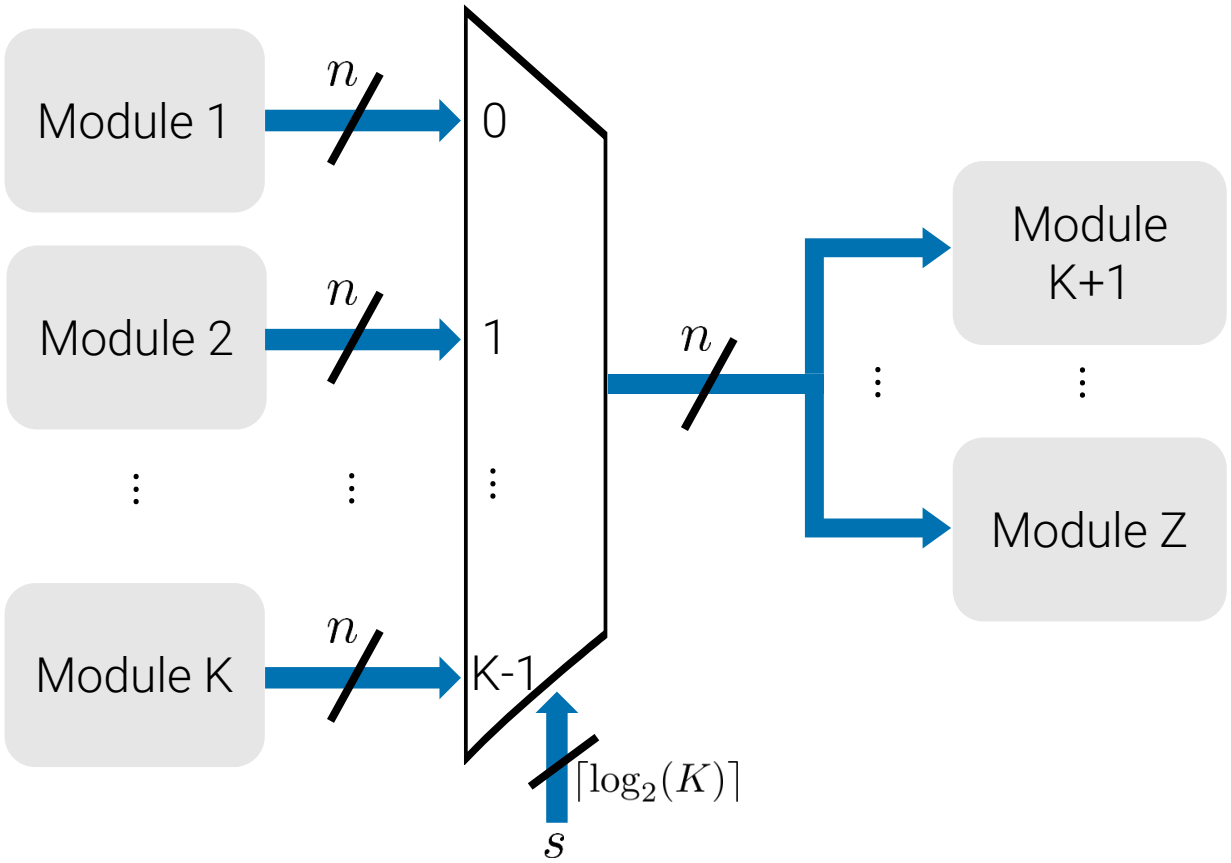


Bus

- Bus receives data from several modules (one at a time) and brings that data to the inputs of other modules
- Buses are typically n-bit wide, where $n > 1$
- Example: An n-bit bus **DATA** groups n wires, each carrying one signal
  - DATA[0], …, DATA[n-1]
- In Verilog, n-bit and 1-bit signals are called **vectors** and **scalars**, respectively

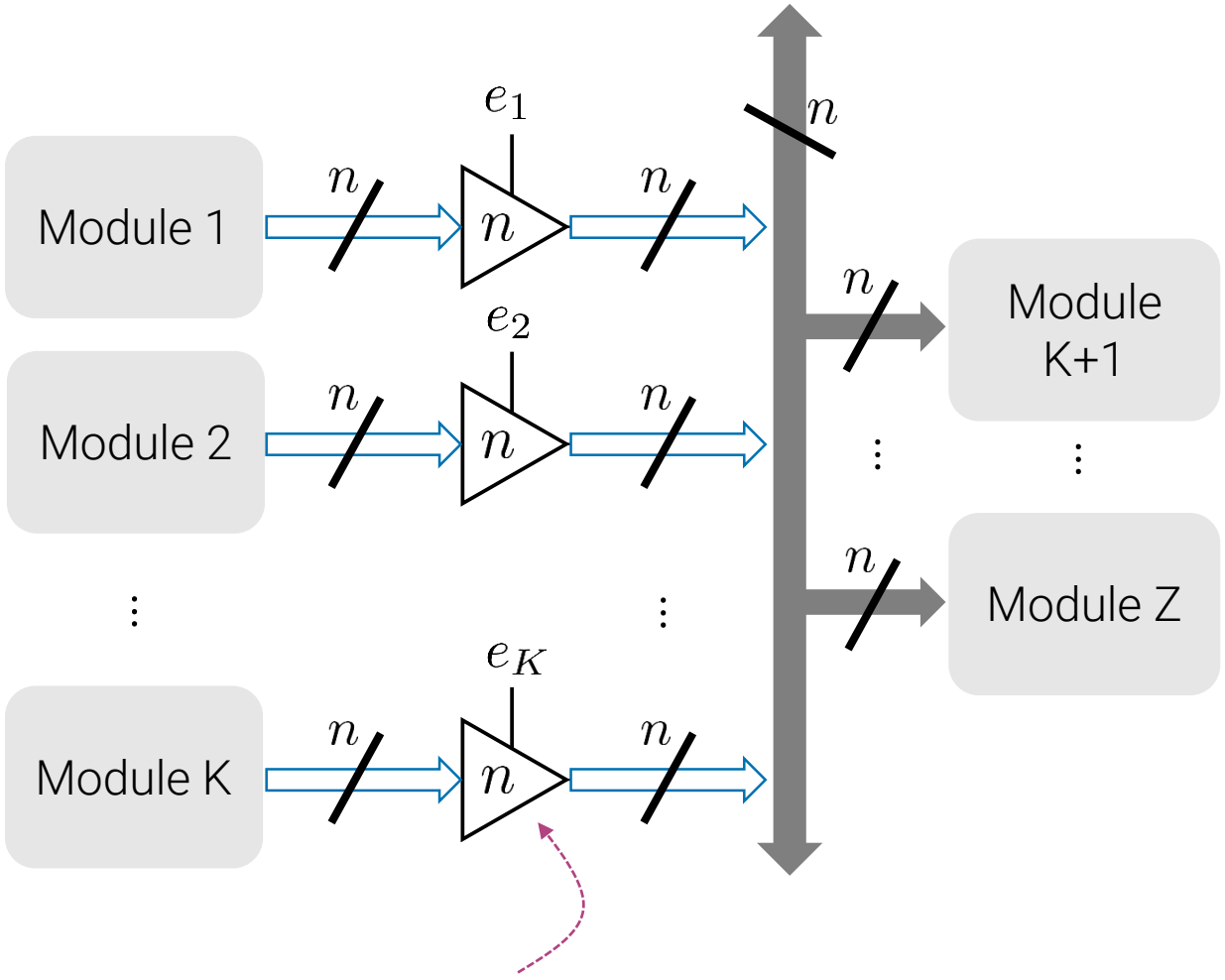


Implementing a Bus With MUXes



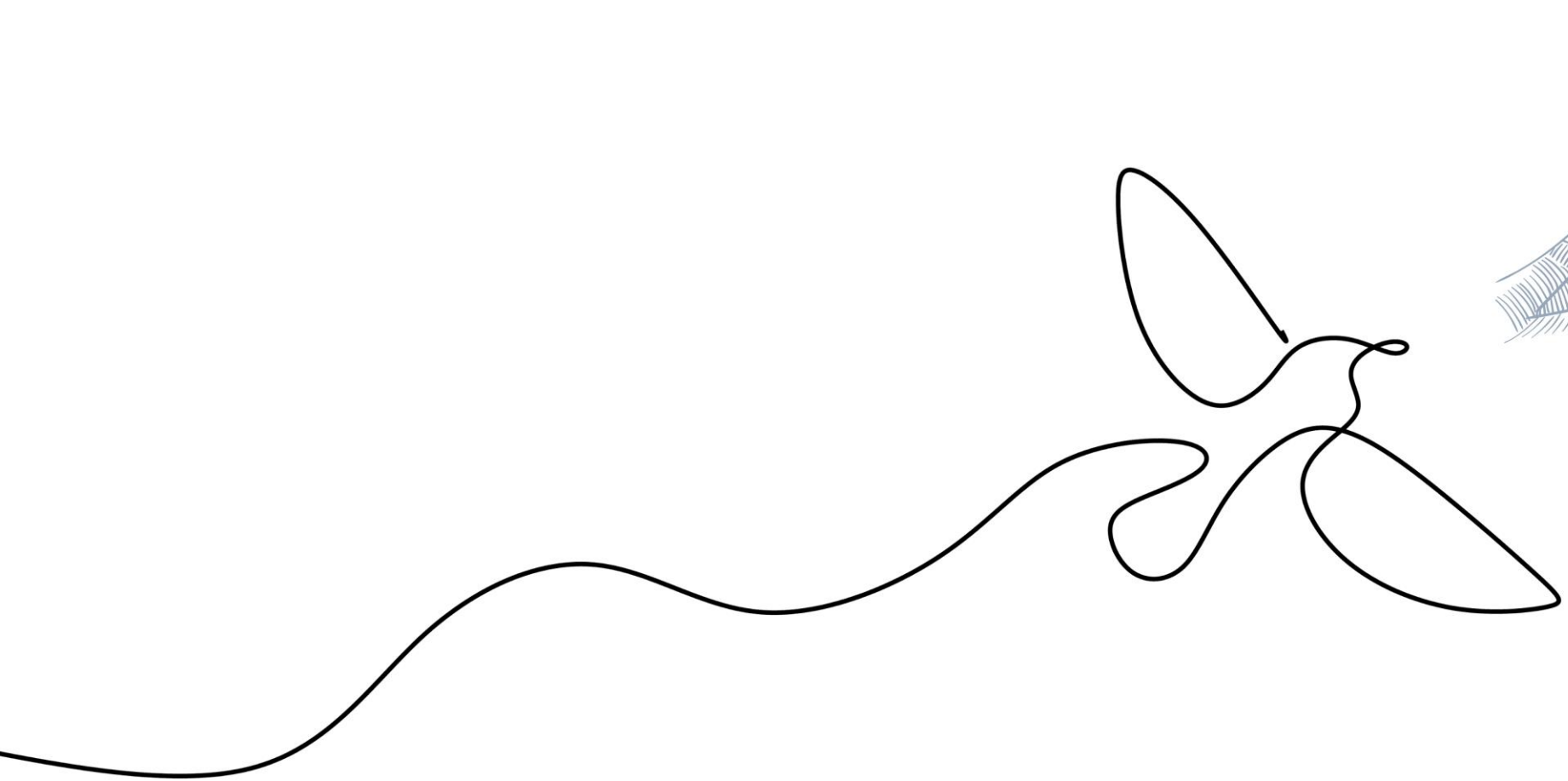
- The multiplexer takes K (K ≥ 2) n-bit data inputs and an $\lceil \log_2(K) \rceil$-bit select signal s to select which of the inputs to pass to the output
- An additional circuit that controls the activation of the **select** signals is typically present

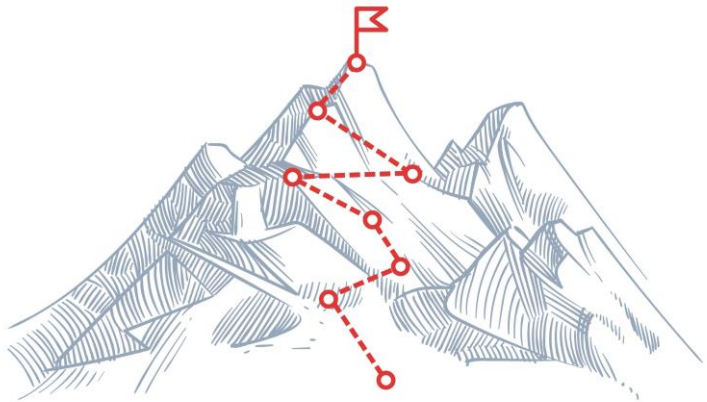
Implementing a Bus With Tri-State Drivers



n tri-state drivers, all controlled by the same enable signal e_K

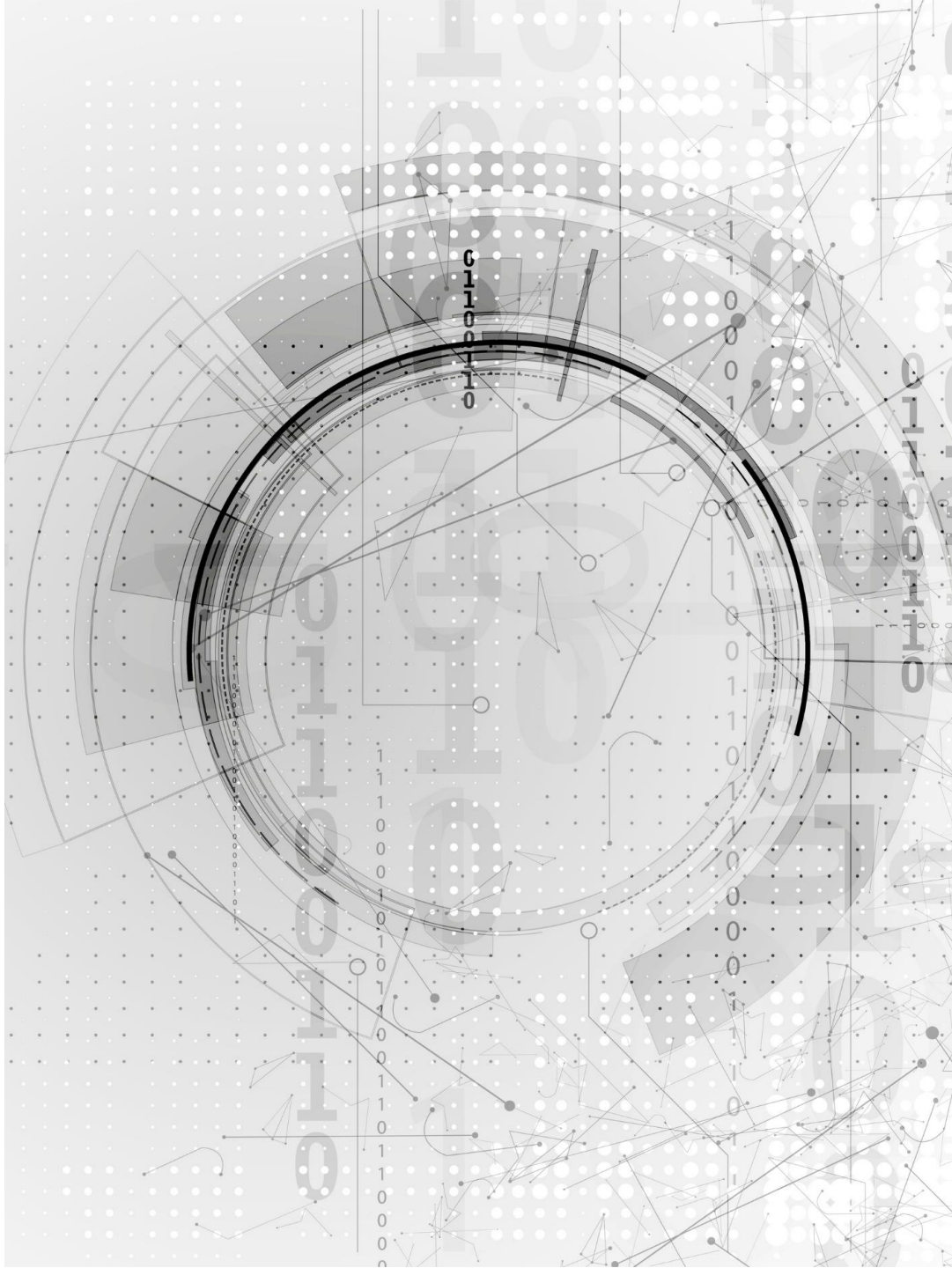
- **Only one** of the enable signals is **active** at a time so that short circuits are avoided
- An additional circuit that controls the activation of the **enable** signals is typically present





Verilog, Contd.

- Signal values, numbers, and parameters



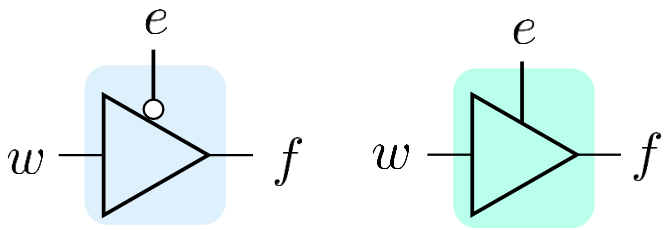
Verilog Built-In Gates

Complete List

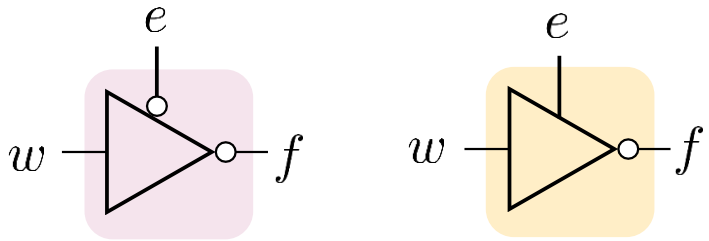
- Complete list of built-in gates
  - **bufif** is a tri-state buffer
  - **notif** is a tri-state inverter

and	xor	bufif0
nand	xnor	bufif1
or	buf	notif0
nor	not	notif1

- Tri-state driver usage
  - `bufif0(f, w, e)`
  - `bufif1(f, w, e)`



  - `notif0(f, w, e)`
  - `notif1(f, w, e)`



Scalar Signal Values

In Verilog

- Verilog supports one-bit signals (scalars)
- Each individual signal can have one of the four values:

Value	Meaning
0	logic value 0
1	logic value 1
z	or Z, tri-state (high-impedance)
x	or X, unknown value or don't care

Vector Signal Values

In Verilog

- Verilog supports multi-bit signals (vectors)
- The value of a vector variable is specified by giving a **constant** of the form

[size][‘radix]constant

where **size** is the number of bits in the constant

Radix	Meaning
d	decimal *(default if no radix is specified)*
b	binary
h	hexadecimal
o	octal

Constants

In Verilog

- *Recall:* **[size][‘radix]constant**
- If size specifies more bits than are needed to represent the given constant, then in most cases, the constant is padded with zeros
  - The exceptions to this rule are when the first character of the constant is either **x** or **z**, in which case the padding is done using that value

- Examples of constants

Constant	Meaning
0	number 0
10	decimal number 10
-8'd10	-10 as an 8-bit two's complement
‘b10	binary number $(10)_2 = (2)_{10}$
‘h10	hex number $(10)_{16} = (16)_{10}$
4’b100	binary number $(\mathbf{0}100)_2 = (4)_{10}$
4’bx	unknown 4-bit value xxxx
8’b1000_0011	_ can be inserted for readability
8’hfx	hex number $(f\mathbf{x})_{16}$, equivalent to 8’b1111_**xxxx**

Concatenation Operator

In Verilog

- Verilog concatenation operator { , } allows vectors to be combined to produce a wider resulting vector
- Example:

```
wire [3:0] upper = 4'b1100;
wire [3:0] lower = 4'b0011;
wire [7:0] combined;

assign combined = {upper, lower};  // Result: 8'b11000011
```

Parameters

In Verilog

- Parameters associate an identifier name with a constant
- Examples:
  - `parameter n = 4;`
    The identifier **`n`** can be used in place of the number 4
  - `parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;`
    The name **`S0`** can be substituted for the binary vector $(00)_2$, etc.
- We will use them when working with parameterized subcircuits

Nets
In Verilog

- Nets represent connections between circuit components
  - Do not store values, but transmit signals
- Most common net types are **wire** and **tri**

- The **wire** type is used to connect an output of one logic element in a circuit to an input of another logic element
  - Examples:
```
wire x;
wire [3:0] s;
```

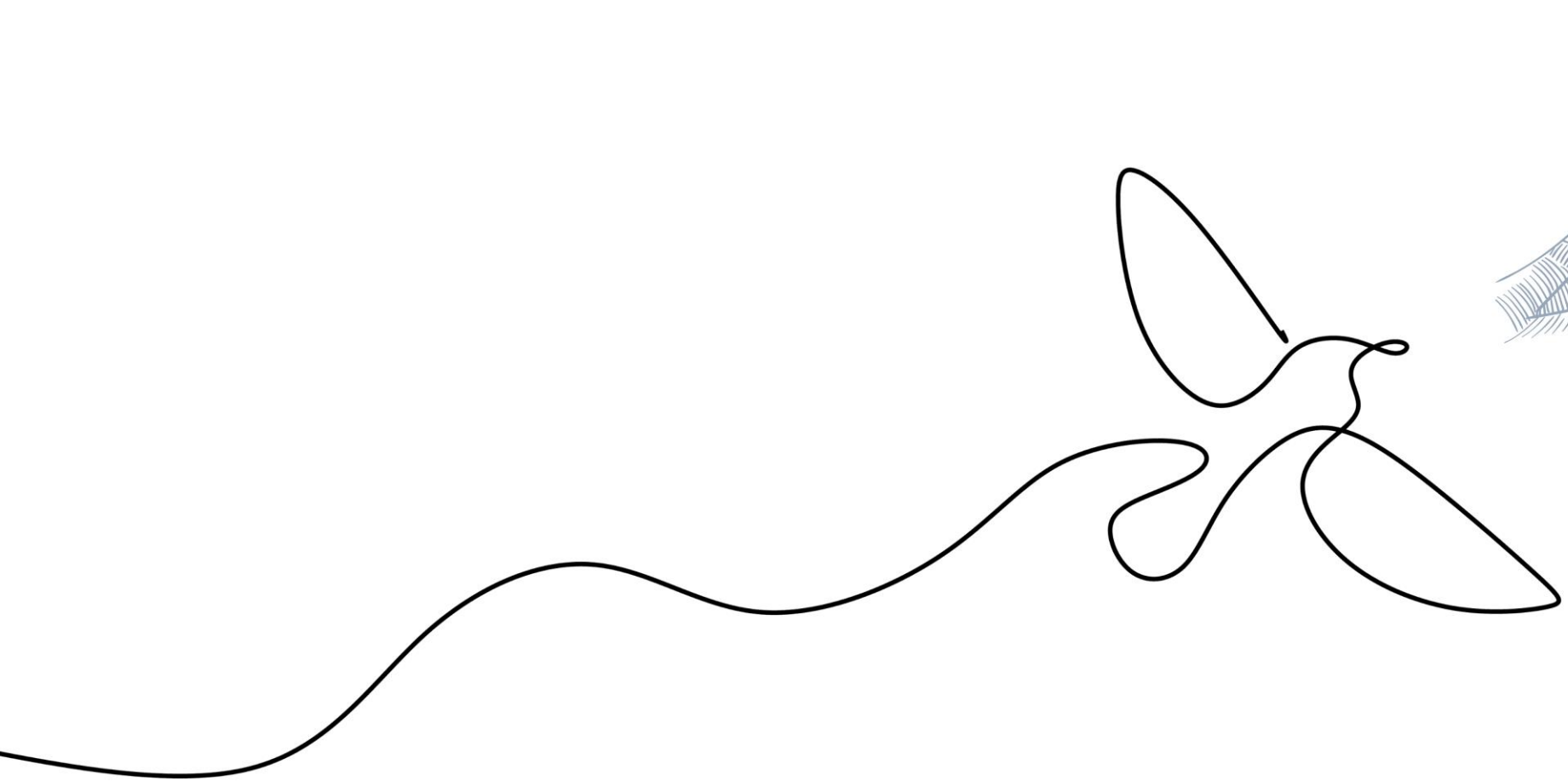
Note: [MSB:LSB] specify the vector range
MSB (most-significant bit): the leftmost bit (highest index);
LSB (least-significant bit): the rightmost bit (lowest index)

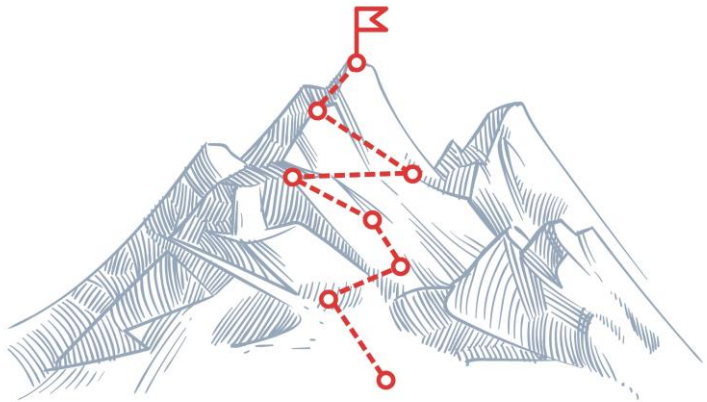
Nets
In Verilog

- Nets represent connections between circuit components
  - Do not store values, but transmit signals
- Most common net types are **wire** and **tri**

- The **tri** type denotes tri-state connections;
  tri nets are treated the **same way**
  as the **wires;** serve to enhance **readability**
  - Examples:
```
tri z1;
tri [7:0] data_out;
```

Note: [MSB:LSB] specify the vector range
MSB (most-significant bit): the leftmost bit (highest index);
LSB (least-significant bit): the rightmost bit (lowest index)

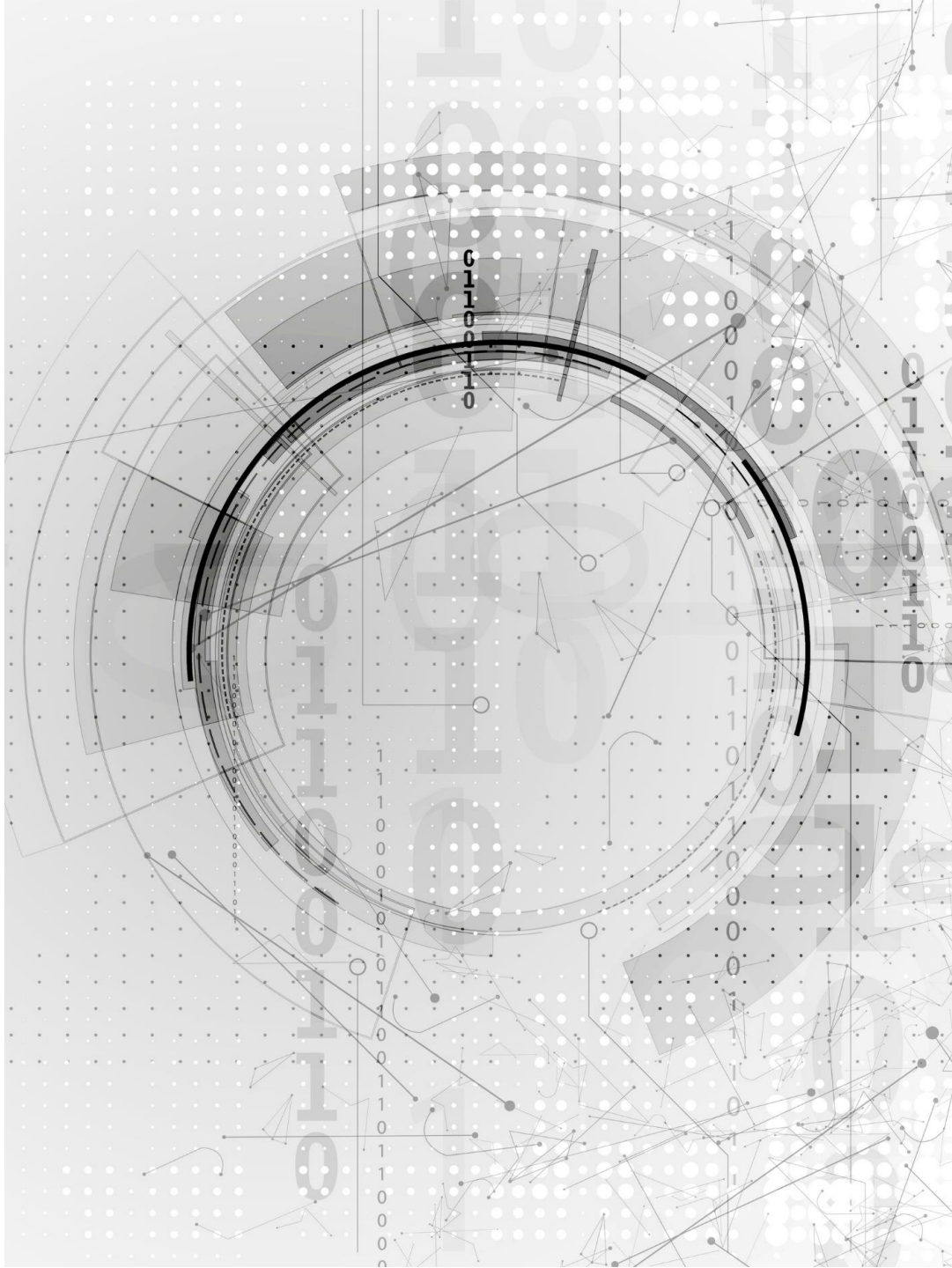




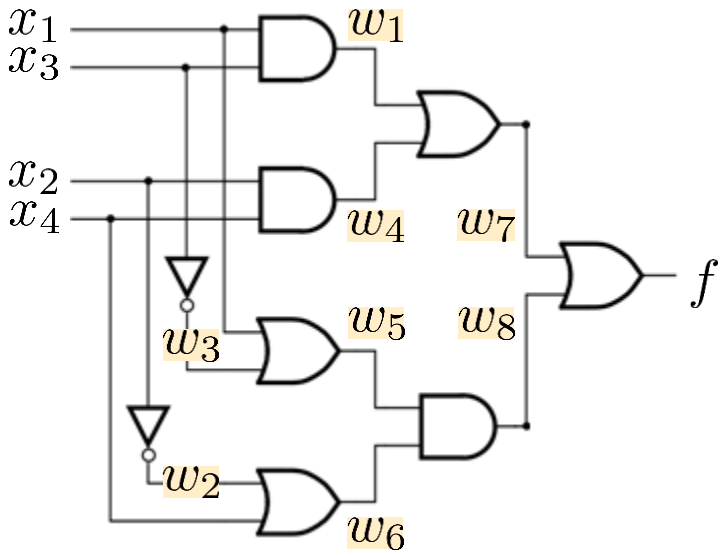
Behavioral Modeling

In Verilog

- Continuous assignments with **assign**
- Procedural assignments with **always@**



Recall Gate-Level Modeling



- Structural (gate level) modeling

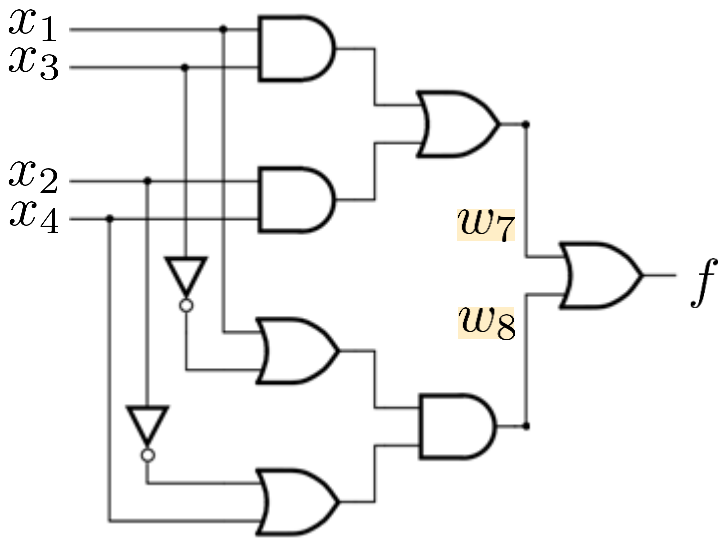
```
module my_circuit_structural (
  input x1, x2, x3, x4,
  output f
  );

  wire w1, w2, w3, w4, w5, w6, w7, w8;

  and (w1, x1, x3);
  not (w2, x2);
  not (w3, x3);
  and (w4, x2, x4);
  or  (w5, x1, w3);
  or  (w6, w2, x4);
  or  (w7, w1, w4);
  and (w8, w5, w6);
  or  (f, w7, w8);
endmodule
```

Inputs and outputs can be defined between the parentheses, for readability

Behavioral Modeling

With Continuous Assignments

- Gate-level modeling becomes tedious for large circuits
- Alternative: use more abstract expressions and programming constructs to **describe** **the behavior** of a logic circuit




```
module my_circuit_behavioral (
  input x1, x2, x3, x4,
  output f
  );

  wire w7, w8;

  assign w7 = (x1 &  x3) | (x2  & x4);
  assign w8 = (x1 | ~x3) & (~x2 | x4);
  assign f  = w7 | w8;
endmodule
```

Verilog operators:
& - bitwise AND
| - bitwise OR
~ - bitwise NOT
***Bitwise**: applied on every bit in isolation*

Continuous Assignments

Assign keyword

- The **assign** keyword provides a **continuous assignment** for a signal

- The term continuous stems from the use of Verilog in the simulation of logic circuits
  - Whenever any signal on the right-hand side of the assignment changes its value, the signal on the left-hand side will be re-evaluated
  - **Continuous assignments are executed in parallel:** therefore, the order in which they appear in the code is irrelevant

Verilog Assign Statement

- General form:

assign net_assignment {, net_assignment} ;

- Examples:
  - ```
    assign f = w7 | w8;
    // Whenever w7 or w8 change,
    // f will be re-evaluated
    ```
  - ```
    wire [3:1] A, B, C;
    assign C = A & B;
    // Whenever vectors A or B change,
    // C will be reevaluated
    // C[1] = A[1] & B[1], C[2] = A[2] & B[2], C[3] = A[3] & B[3]
    ```

Note: Braces indicate that additional entries are permitted

Behavioral Modeling

With Procedural Statements

- Verilog allows us to use an even higher level of abstraction with **procedural statements**
  - Also called sequential statements
  - Examples: **if-else, case**, loops
- Procedural statements must be contained inside an **always**-block
  - **Evaluated in the order** given in the code
- To describe circuit behavior, **variables** are used instead of wires
  - For circuit modeling, **variables of type reg** are used

Always Block

Behavioral Modeling with Procedural Statements

- General format

  (for combinational circuits)

always @*
[**begin**]
  [procedural assignments]
  [**if-else** statements]
  [**case** statements]
  [**while**, **repeat**, and **for** loops]
  [**task** and **function** calls]
[**end**]

Note: The square brackets indicate an optional field

- When multiple statements are included in the block, the **begin** and **end** keywords are needed; otherwise, they can be omitted*

  [*] Not recommended because it is error-prone

Always Block

Behavioral Modeling with Procedural Statements

- General format
  (for combinational circuits)

always @*
[**begin**]
  [procedural assignments]
  [**if-else** statements]
  [**case** statements]
  [**while**, **repeat**, and **for** loops]
  [**task** and **function** calls]
[**end**]

Note: The square brackets indicate an optional field

- **always @*** treats **all input** signals used in the block as relevant
  - If **any** input signal mentioned in the block changes its value, all statements using that signal are evaluated **in the order** presented. A signal **assigned multiple** values inside the block **retains the last one**

If-Else Statement

- If the expression is **True**, the statements inside the **begin-end** block are evaluated
  - When **multiple** statements are involved, they have to be included inside a begin-end block
- **else if** and **else** clauses are optional
  - When included, they are paired with the most recent unfinished **if** or **else if**

- General format

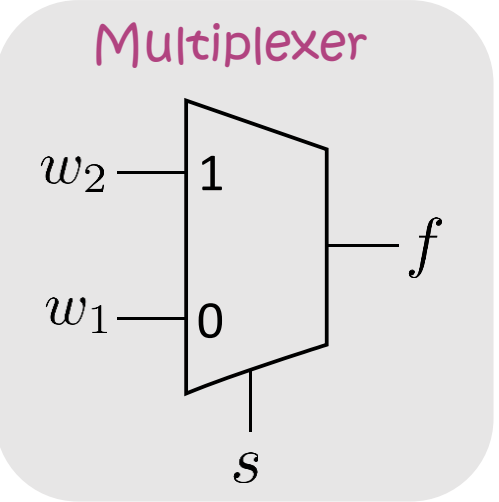
```
if (expression1)
begin
   statement;
end
else if  (expression2)
begin
   statement;
end
else
begin
   statement;
end
```

2-to-1 MUX

Always Block and an If-Else Statement

- Write a behavioral model of a 2-to-1 MUX using procedural statements in Verilog

```
module my_2to1mux (
  input w1, w2, s,
  output reg f);

  always @* begin
    if (s == 0) begin
      f = w1;
    end
    else begin
      f = w2;
    end
  end
endmodule
```



*Note the output is of the type **reg**, because this is a behavioral model (in an always block)*

*Verilog relational operators **op** return 1 (True) or 0 (False) based on the result of the specified comparison **A op B**:*

*A **==** B: equality check **op**erator*
*A **!=** B: inequality check*
*A **>** B, A **>=** B, A **<** B, A **<=** B: comparisons*

Case Statement

- The bits in the expression, called the **controlling expression**, are checked for a match with each alternative
  - **Each digit in each alternative** is compared for an exact match of the four values **0**, **1**, **x**, **z**
  - A special case is the **default** clause, which takes effect if **no other** alternative matches
- The **first successful** match causes the associated statements to be evaluated

- The form of the case statement

```
case (expression)
alternative1: begin
                  statements;
              end
alternative2: begin
                  statements;
              end
[default:     begin
                  statements;
              end]
endcase
```

Note: The square brackets indicate an optional field

Full-Adder

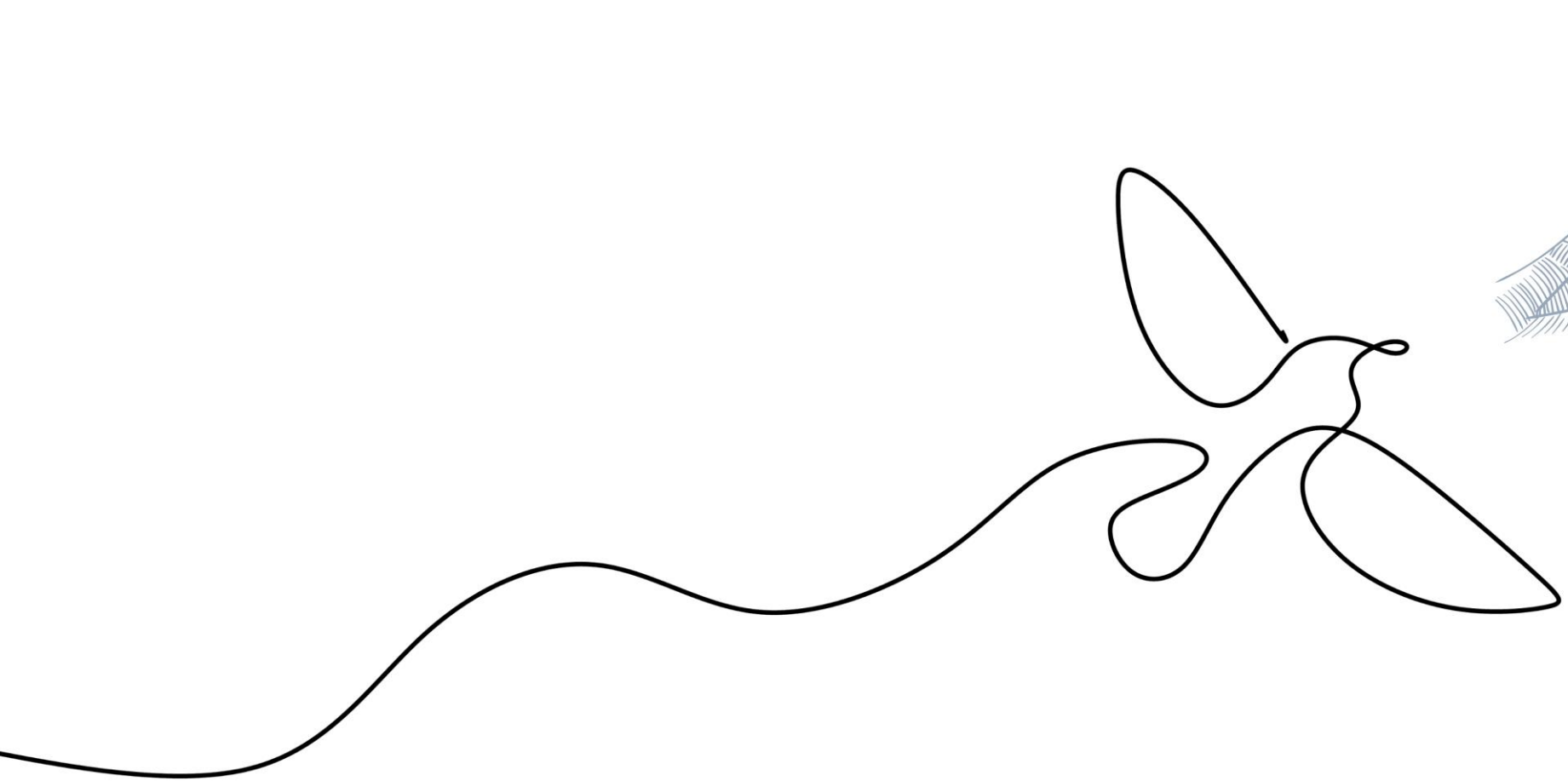
Always Block and a Case Statement

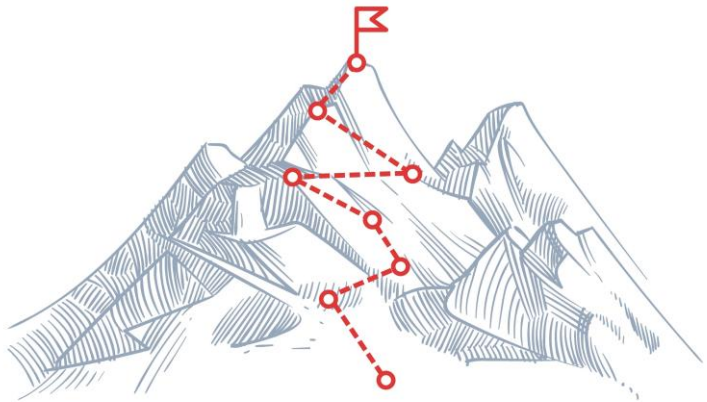
- **case** statements can be used to describe truth tables
  - model a full adder using a case statement

x	y	Cin	s	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

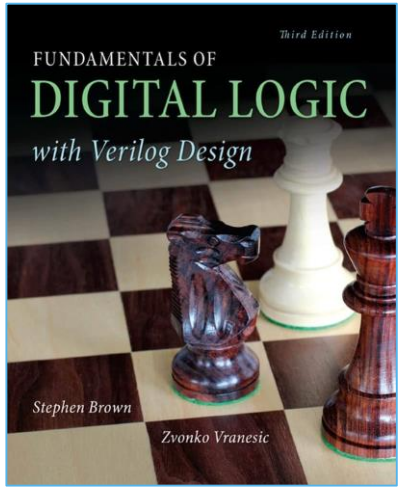
```
module fulladd (
  input x, y, Cin,
  output reg s, Cout);

  always @* begin
    case ({x, y, Cin})
      3'b000: {s, Cout} = 'b00;
      3'b001: {s, Cout} = 'b10;
      3'b010: {s, Cout} = 'b10;
      3'b011: {s, Cout} = 'b01;
      3'b100: {s, Cout} = 'b10;
      3'b101: {s, Cout} = 'b01;
      3'b110: {s, Cout} = 'b01;
      3'b111: {s, Cout} = 'b11;
    endcase
  end
endmodule
```

*Verilog **concatenation operator {,}** allows vectors to be combined to produce a wider resulting vector.*

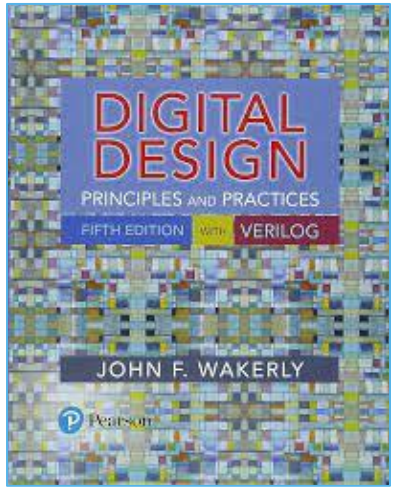




Literature



- Chapter 2: Introduction to Logic Circuits
  - 2.10.2
- Appendix A: Verilog Reference
  - A.1 – A.11.4



- Chapter 5: Verilog Hardware Description Language
  - 5.9