Technische
Hochschule
Augsburg

VLSI Design

Concurrent Statements for Combinational Logic

J Färber

SS2026

Contents

Concurrent Statements for Combinational Logic **2**
- Learning Objectives . . . 2
- Introductory Example: 4-Bit Up-Counter . . . 3
  - Schematic Symbol and RTL Conceptional Diagram . . . 3
  - Function . . . 3
- WITH … SELECT — Selected Signal Assignment . . . 4
  - Syntax . . . 4
  - Completeness: WHEN OTHERS . . . 4
  - Example: 2-to-1 Multiplexer . . . 4
  - Example: 7-Segment Decoder (bin2seg) . . . 5
- Modeling Truth Tables with WITH … SELECT . . . 6
  - Concatenating Inputs into a Select Vector . . . 6
  - Example: 1-Bit Full Adder as Truth Table . . . 7
- WHEN … ELSE — Conditional Signal Assignment . . . 8
  - Syntax . . . 8
  - Priority of Conditions . . . 9
  - Example: 2-to-1 Multiplexer . . . 9
  - Example: Priority Encoder . . . 10
- WITH … SELECT vs. WHEN … ELSE . . . 11
- Outlook: Synchronous Logic . . . 12
- Summary . . . 12
- Reference . . . 12

Concurrent Statements for Combinational Logic

Learning Objectives

After completing this unit, you will be able to:

- Use `WITH … SELECT` to map discrete input values to output values
- Model any combinational truth table by concatenating inputs into a vector
- Use `WHEN … ELSE` to describe conditional signal assignments with priority
- Select the appropriate construct for a given design situation
- Apply both constructs as the exclusive modelling tool for combinational and synchronous logic

Introductory Example: 4-Bit Up-Counter

Schematic Symbol and RTL Conceptional Diagram

Function

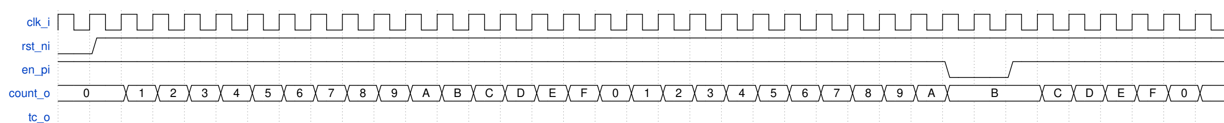


Figure 1: Behaviour - Timing Diagram

WITH … SELECT — Selected Signal Assignment

Syntax

Referring to [Mea25], chap. 5.2

```
WITH select_expression SELECT
  target_signal <= value_a WHEN choice_1,
                   value_b WHEN choice_2,
                   value_c WHEN choice_3,
                   value_default WHEN OTHERS;
```

Figure 2: Selected Signal Assignment - Conceptional Diagram

- All choices are evaluated **simultaneously** — there is no priority
- `WHEN OTHERS` is mandatory: it covers every value not listed explicitly
- The select expression must be of a discrete type (`std_ulogic_vector`, `unsigned`, `signed`, enumeration)

Completeness: WHEN OTHERS

For an N-bit `std_ulogic_vector` there are 9^N possible values because each bit can be `'0'`, `'1'`, `'U'`, `'X'`, `'Z'`, `'W'`, `'L'`, `'H'`, or `'-'`. Even a 1-bit signal has 9 states — `WHEN OTHERS` is therefore always required to make the assignment complete.

In synthesis, only `'0'` and `'1'` matter. In simulation, `WHEN OTHERS` covers uninitialised (`'U'`) and unknown (`'X'`) states, making errors visible immediately.

Example: 2-to-1 Multiplexer

Referring to [Mea25], chap. 5.1

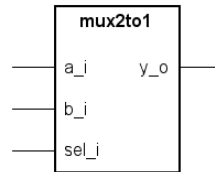


Figure 3: mux2to1 — Schematic Symbol

The function table of `mux2to1`:

```
sel_i | y_o
------+-----
  0   |  b_i
  1   |  a_i
```

```
ARCHITECTURE truthtable OF mux2to1 IS

BEGIN

  WITH sel_i SELECT
    y_o <= a_i WHEN '1',
           b_i WHEN OTHERS;

END truthtable;
```

Example: 7-Segment Decoder (bin2seg)

`bin2seg` maps a 4-bit binary input to the active-low 7-segment encoding of the hexadecimal digits `0` to `F`. This is a truth table with 16 rows — exactly the use case `WITH … SELECT` is designed for.

```
Segment labelling:       Encoding: seg_o = "gfedcba"
     0                   active-low: '0' = segment ON
    ---
 5 |   | 1
    ---   ← 6
 4 |   | 2
    ---
     3
```

```
ENTITY bin2seg IS
  PORT (
    bin_i : IN  std_ulogic_vector(3 DOWNTO 0);
    seg_o : OUT std_ulogic_vector(6 DOWNTO 0)    -- gfedcba, active-low
  );
END bin2seg;

ARCHITECTURE rtl OF bin2seg IS
```

```
BEGIN

  WITH bin_i SELECT
    seg_o <=
      "1000000" WHEN "0000",  -- 0
      "1111001" WHEN "0001",  -- 1
      "0100100" WHEN "0010",  -- 2
      "0110000" WHEN "0011",  -- 3
      "0011001" WHEN "0100",  -- 4
      "0010010" WHEN "0101",  -- 5
      "0000010" WHEN "0110",  -- 6
      "1111000" WHEN "0111",  -- 7
      "0000000" WHEN "1000",  -- 8
      "0010000" WHEN "1001",  -- 9
      "1111111" WHEN OTHERS;  -- all segments off

END rtl;
```

Each row of the truth table maps directly to one WHEN clause. The synthesiser infers a combinational decoder — no intermediate signals, no boolean equations required.

Modeling Truth Tables with WITH … SELECT

Concatenating Inputs into a Select Vector

Example: 2-input arbitrary function

```
-- Inputs: a, b  (single bits)
-- Concatenation: a & b results in a 2-Bit-Vector

WITH (a & b) SELECT
    y <= '1' WHEN "10",    -- a=1, b=0
         '0' WHEN "00",    -- a=0, b=0
         '0' WHEN "01",    -- a=0, b=1
         '0' WHEN "11",    -- a=1, b=1
         '0' WHEN OTHERS;
```

Example: 3-input Majoriy Gate

```
-- 3-input Majoriy Gate
-- provides a y = '1' if and only if at least two of its three inputs are '1'.
-- Inputs: a, b, c  (single bits)
-- Concatenation: a & b results in a 2-Bit-Vektor

WITH (
```

When a circuit has multiple single-bit inputs, they can be **concatenated** into one `std_ulogic_vector` and used as the select expression. This directly maps a truth table to VHDL.

```
SIGNAL inputs_s : std_ulogic_vector(N-1 DOWNTO 0);

inputs_s <= in_a & in_b & in_c;   -- concatenate N inputs

WITH inputs_s SELECT
  output_s <= value_0 WHEN "000",
              value_1 WHEN "001",
              ...
              value_default WHEN OTHERS;
```

Similarly, multiple output bits can be concatenated into one vector signal and split apart after the assignment:

```
SIGNAL outputs_s : std_ulogic_vector(1 DOWNTO 0);

WITH inputs_s SELECT
  outputs_s <= "00" WHEN "000",
               "01" WHEN "001",
               ...
               "00" WHEN OTHERS;

out_a <= outputs_s(1);
out_b <= outputs_s(0);
```

Example: 1-Bit Full Adder as Truth Table

A 1-bit full adder has three inputs (`a_i`, `b_i`, `ci_i`) and two outputs (`sum_o`, `co_o`). The complete truth table has $2^3 = 8$ rows.

```
  ci_i  b_i  a_i  |  co_o  sum_o
  ----------------+--------------
    0    0    0   |
    0    0    1   |
    0    1    0   |
    0    1    1   |
    1    0    0   |
    1    0    1   |
    1    1    0   |
    1    1    1   |
```

```
ENTITY add1 IS
  PORT (
    a_i   : IN  std_ulogic;
    b_i   : IN  std_ulogic;
    ci_i  : IN  std_ulogic;
    sum_o : OUT std_ulogic;
    co_o  : OUT std_ulogic
  );
END add1;

ARCHITECTURE truthtable OF add1 IS

  SIGNAL inputs_s  : std_ulogic_vector(2 DOWNTO 0);
  SIGNAL outputs_s : std_ulogic_vector(1 DOWNTO 0);  -- (co_o, sum_o)

BEGIN

  inputs_s <= ci_i & b_i & a_i;

  WITH inputs_s SELECT
    outputs_s <=
      "    " WHEN "000",   -- 0+0+0 =    , co=
      "    " WHEN "001",   -- 1+0+0 =    , co=
      "    " WHEN "010",   -- 0+1+0 =    , co=
      "    " WHEN "011",   -- 1+1+0 =    , co=
      "    " WHEN "100",   -- 0+0+1 =    , co=
      "    " WHEN "101",   -- 1+0+1 =    , co=
      "    " WHEN "110",   -- 0+1+1 =    , co=
      "    " WHEN "111",   -- 1+1+1 =    , co=
      "    " WHEN OTHERS;

  co_o  <= outputs_s(1);
  sum_o <= outputs_s(0);

END truthtable;
```

> This approach scales to any combinational circuit: write the truth table, concatenate inputs, enumerate outputs. No boolean minimisation required — the synthesiser optimises automatically.

WHEN … ELSE — Conditional Signal Assignment

Syntax

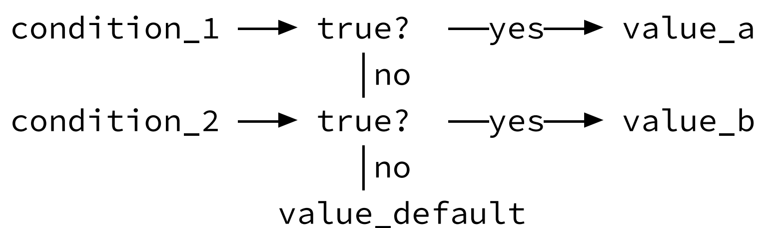
```
target_signal <= value_a WHEN condition_1 ELSE
                 value_b WHEN condition_2 ELSE
                 value_default;
```

Figure 4: Conditional Signal Assignment - Conceptional Diagram

- Each branch is evaluated in order — **the first true condition wins**
- The final `ELSE` clause is mandatory: it covers all remaining cases
- The statement executes **concurrently** — it models a combinational network, not a sequential program

Priority of Conditions

Because conditions are checked from top to bottom, `WHEN … ELSE` naturally models **priority logic**: a condition listed earlier takes precedence over all conditions listed below it.



> This priority behaviour maps directly to a **priority multiplexer** in hardware — useful whenever one input must override others.

Example: 2-to-1 Multiplexer

The `mux2to1` from the previous section can also be written with `WHEN … ELSE`. Comparing both architectures makes the difference in readability concrete:

```
-- WITH … SELECT (previous section) — reads like a lookup table
WITH sel_i SELECT
  y_o <= a_i WHEN '1',
         b_i WHEN OTHERS;

-- WHEN … ELSE — reads like a condition chain, identical hardware
y_o <= a_i WHEN sel_i = '1' ELSE
       b_i;
```

Both architectures produce exactly the same synthesised circuit. WHEN … ELSE allows arbitrary boolean expressions as conditions, not just equality with a single signal.

Compare also with the boolean-equation architecture from Lecture 2:

```
-- boolean equation (Lecture 2)
y_o <= (sel_i AND a_i) OR (NOT sel_i AND b_i);
```

All three architectures are equivalent — only readability differs.

Example: Priority Encoder

A 4-to-2 priority encoder asserts the index of the **highest-priority** active input. Input x_i(3) has highest priority.

```
  x_i       | y_o
  ---------+------
  1 x x x  |  11       x_i(3) overrides all
  0 1 x x  |  10
  0 0 1 x  |  01
  0 0 0 1  |  00
```

The overlapping x entries make this table impossible to model with WITH … SELECT (which requires mutually exclusive, fully enumerated values). WHEN … ELSE encodes priority directly through the order of conditions:

```
ENTITY priority42 IS
  PORT (
    x_i     : IN  std_ulogic_vector(3 DOWNTO 0);
    y_o     : OUT std_ulogic_vector(1 DOWNTO 0);



  );
END priority42;

ARCHITECTURE rtl OF priority42 IS

BEGIN
```

```
  y_o <=









END rtl;
```

> The order of conditions encodes priority — swapping two lines changes which input wins. This is the key capability that `WITH … SELECT` lacks.

WITH … SELECT vs. WHEN … ELSE

	`WITH … SELECT`	`WHEN … ELSE`
Conditions	Discrete values of one expression	Arbitrary boolean expressions
Priority	No priority — all choices equal	First matching condition wins
Completeness	`WHEN OTHERS` required	Final `ELSE` required
Best for	Truth tables, decoders, multiplexers	Priority logic, range checks, complex conditions
Typical use	bin2seg, mux, full adder	Priority encoder, address decoder

Rule of thumb:

- Use `WITH … SELECT` when the output depends on the **value** of a single expression — reads like a lookup table
- Use `WHEN … ELSE` when the output depends on **conditions** that may overlap or carry priority

Both constructs produce equivalent hardware when conditions are mutually exclusive.

Outlook: Synchronous Logic

The same two constructs — `WITH … SELECT` and `WHEN … ELSE` — are used in the next Lecture to model **synchronous (clocked) circuits** such as registers and counters.

Summary

- `WITH … SELECT` assigns a value based on the **enumerated cases** of one expression — no priority, reads like a truth table
- `WHEN … ELSE` assigns a value based on **ordered conditions** — the first true condition wins (priority behaviour)
- Both constructs are **concurrent** — they model hardware, not software
- Any combinational truth table can be modelled by **concatenating inputs** into a select vector and listing all output combinations
- `WHEN OTHERS` / final `ELSE` are mandatory for completeness
- The same constructs extend to synchronous logic in Lecture 4

> **Next Lecture:** Synchronous Logic: registers, counters, and clocked circuits with `WHEN … ELSE`

Reference

- [Mea25] Mealy, B., Tappero, F.: *Free Range VHDL*, 2025 — Chapter 5