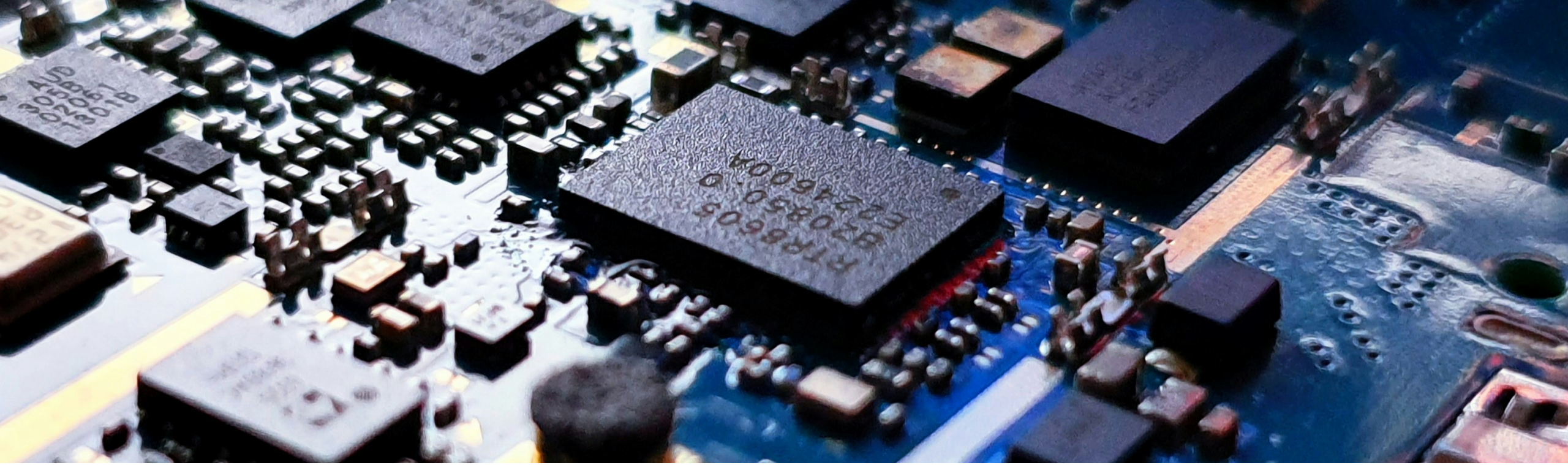


Laboratory 3 – **VHDL**

Computer Architecture

A.Y. 2023/24

Table of contents

- An introduction to VHDL
- VHDL characteristics
- Data types
- Objects
- Code examples
- The 7-segment display example and exercise

An introduction to VHDL

What is VHDL?

VHDL is a Hardware Description Language used to model a digital system, for synthesis and simulation.

When was it created?

VHDL was developed for the US military in the 1980s and has been standardized by IEEE as *IEEE Std 1076*.

What are its advantages?

Word-rich: semantics allow for self-documented, unambiguous designs.

Reusability: VHDL allows development of reusable packages for various applications.

VHDL characteristics

Hierarchical design

VHDL is hierarchical, meaning that each block (*entity*) can contain other entities which are defined separately. Multiple instances of an entity can exist in different higher-level entities.

Component reutilization

Having multiple instances of an entity allows for design reutilization and nesting structures.

Furthermore, lower-level entities are independent of each other, enabling groups of designers to work on different subsystems of the same design.

VHDL characteristics

Entities and Architectures

An ***Entity*** represents the external view of a system: in it, input/output gates and parameters are defined.

An ***Architecture*** defines how a system works by using VHDL instructions and lower-level components.

Design files

A **Design file** is the highest-level structure in a program; also known as the Top-Level Entity. It is made up of several *design units*.

VHDL – Entities and Architectures Example

Syntax:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY __entity_name IS
    GENERIC(define parameters);
    PORT(define inputs and outputs);
END __entity_name;

ARCHITECTURE a OF __entity_name IS
    SIGNAL and COMPONENT declarations;
BEGIN
    statements;
END a;
```

Entity

Architecture

VHDL characteristics

Identifiers

An identifier represents the name of a circuit element, signal, port, variable, entity, architecture, etc.

Identifiers ***must start with a letter***, followed by a combination of letters and numbers (***no spaces***), as well as underscores ("_"), but ***not two _ in a row***.

Identifiers are not case-sensitive, and ***no other special characters are allowed***.

Valid identifiers:

```
adder_5bit
MarioLuigi
sample3
```

Invalid identifiers:

```
5bit__adder
temp?
my component
```

...why?

VHDL characteristics

Reserved words

You cannot use certain words in your identifiers, because they are reserved for the language.

This table contains some examples.

(just like you can't name a variable "if" in Python...)

abs	access	after	alias	all
and	architecture	array	assert	attribute
begin	block	body	buffer	bus
case	component	configuration	constant	disconnect
downto	else	elseif	end	entity
exit	file	for	function	generate
generic	guarded	if	in	inout
is	label	library	linkage	loop
map	mod	nand	new	next
nor	not	null	of	on
open	or	others	out	package
port	procedure	process	range	record
register	rem	report	return	select
severity	signal	subtype	then	to
transport	type	units	until	use
variable	wait	when	while	with
xor				

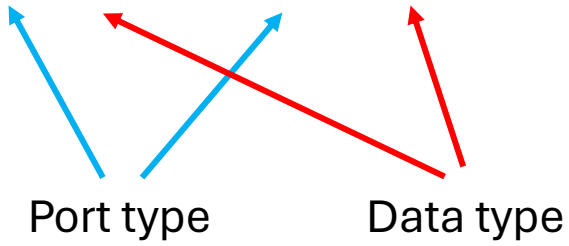
VHDL characteristics

Ports and port types

Every entity must have ports, signals that go into and out of the entity. Every signal has a type.

Port declaration syntax:

```
ENTITY my_entity IS
    PORT(a,b: IN BIT; y: OUT BIT);
END my_entity;
```



Port types:

`IN:`	read but not modified.
`OUT:`	modified but not read.
`BUFFER:`	always active. Read and modified.
`INOUT:`	bidirectional, tri-state gate. Read and modified.

VHDL characteristics

Data types

Every port has a **data type**. The most common data types in VHDL are the BIT and BIT_VECTOR types.

The BIT data type defines the logical values ‘0’ and ‘1’.

What if my ports need more than 1 bit?

Bit vectors are used to define variables using more than 1 bit.

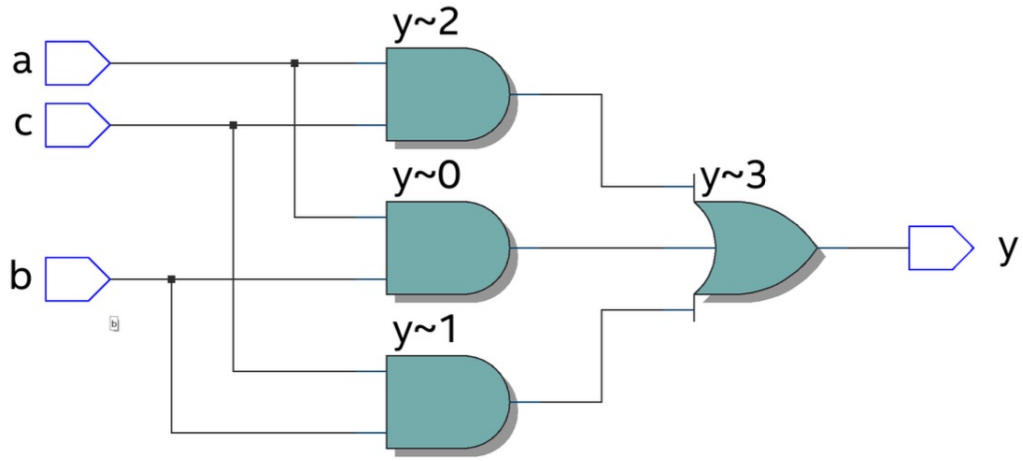
```
d: IN BIT_VECTOR (3 downto 0)   -- 4 bits total
```

Accessing a specific bit: `d(3)`

VHDL – Majority vote code example

Basic logic gates are already included in the IEEE library.

```
2  ENTITY majority_vote IS
3      PORT(
4              a,b,c: IN BIT;
5              y: OUT BIT);
6  END majority_vote;
7  ARCHITECTURE maj_vote of majority_vote IS
8  BEGIN
9      y <= (a and b) or (b and c) or (a and c);
10 END maj_vote;
```



VHDL – More code examples

Dropbox: Laboratories/lab3examples/half_add.vhd

```
 3  ENTITY half_add IS
 4      PORT(
 5              a,b: IN BIT;
 6              sum, carry: OUT BIT);
 7  END half_add;
 8  ARCHITECTURE adder of half_add IS
 9  BEGIN
10      sum <= a xor b;
11      carry <= a and b;
12  END adder;
```

Exercise 1: import the contents of *lab3examples* into a new Quartus project.

VHDL – More code examples

bitwise_and_4.vhd

```
 3  -- 4-bit bitwise and function
 4  -- y0 = a0 and b0; y1 = a1 and b1; etc.
 5
 6  ENTITY bitwise_and_4 IS
 7      PORT(
 8              a0, a1, a2, a3: IN BIT;
 9              b0, b1, b2, b3: IN BIT;
10              y0, y1, y2, y3: OUT BIT);
11  END bitwise_and_4;
12  ARCHITECTURE and_gate of bitwise_and_4 IS
13  BEGIN
14      y0 <= a0 and b0;
15      y1 <= a1 and b1;
16      y2 <= a2 and b2;
17      y3 <= a3 and b3;
18  END and_gate;
```

bitwise_and_vec_4.vhd

```
 3  -- 4-bit bitwise and function
 4  -- Y = A and B;
 5
 6  ENTITY bitwise_and_vec_4 IS
 7      PORT(
 8              a, b: IN BIT_VECTOR(3 downto 0);
 9              y: OUT BIT_VECTOR (3 downto 0));
10  END bitwise_and_vec_4;
11  ARCHITECTURE and_gate of bitwise_and_vec_4 IS
12  BEGIN
13      y <= a and b;
14  END and_gate;
```

Which one is better?

VHDL – Bit vector data type

Assignments for vectors are made from left to right, and the contents can be chosen according to the numbering of the elements.

Constant values: assignment using double quotes (like strings)

Example:

```
my_vector: OUT BIT_VECTOR (7 downto 0)
...
my_vector <= "11010011"
```

VHDL – Other data types

- **Integers**

  Positive and negative; 3 -143 15 8 0 e.g. `y <= 5;`

  Max range: depends on compiler (assume 32 bits)

- **Characters**

  Simple letters, use single quotes: ‘a’, ‘Z’…

- **Strings**

  are made up of a set of characters. “hello”, “World”…

VHDL – Other data types

- **STD_LOGIC and STD_LOGIC_VECTOR**

  The STD_LOGIC type is like the BIT type, but with additional possible values:

‘U’: uninitialized. The signal hasn’t been set yet.

‘X’: unknown. Impossible to determine this value.

‘0’: logic 0

‘1’: logic 1

‘Z’: high impedence

‘W’: weak signal, can’t tell if it should be 0 or 1

‘L’: low signal, should go to 0

‘H’: high signal, should go to 1

‘-’: don’t care

You must use the following instruction at the start of your design file to import this type:

```
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
```

Use STD_LOGIC instead of BIT!

VHDL – Objects

Variables

Variables in VHDL are only declared in sequential structures (processes), before the BEGIN keyword. They represent data stored in computer memory (not the hardware) used during synthesis to generate the circuit.

```
VARIABLE bitsize : INTEGER := 16;
```

Constants

Objects that maintain their initial value.

```
CONSTANT bitsize : INTEGER := 16;
```

Assignment

The operator <= assigns values to signals.

```
SIGNAL my_signal : STD_LOGIC_VECTOR (7 downto 0);
my_signal <= "10101010"; -- assigning multiple values
my_signal (5) <= '1'; -- assigning to a specific element
my_signal (7 downto 5) <= "101"; -- assigning to certain elements of the array
my_signal <= (others => '1' ); -- assigning the same value to all array elements
```

VHDL – More code examples

signal_ex.vhd

```
3  LIBRARY ieee;
4  USE ieee.std_logic_1164.ALL;
5  ENTITY signal_ex IS
6      PORT(
7              a,b,c: IN STD_LOGIC;
8              w,x,y,z: OUT STD_LOGIC);
9  END signal_ex;
10
11 ARCHITECTURE sig of signal_ex IS
12 -- Declaration area
13 -- Define signal here
14     SIGNAL inputs: STD_LOGIC_VECTOR (2 downto 0);
15     SIGNAL outputs: STD_LOGIC_VECTOR (3 downto 0);
16 BEGIN
17 -- Concatenate input ports into 3-bit signal inputs
18     inputs <= a & b & c;
19     WITH inputs SELECT
20     outputs <=  "1000" WHEN "000",
21                 "0100" WHEN "001",
22                 "0110" WHEN "010",
23                 "1001" WHEN "011",
24                 "0110" WHEN "100",
25                 "0001" WHEN "101",
26                 "1001" WHEN "110",
27                 "0010" WHEN "111",
28                 "0000" WHEN others;
29 -- Separate signal into individual ports
30     w <= outputs(3);
31     x <= outputs(2);
32     y <= outputs(1);
33     z <= outputs(0);
34 END sig;
```

signal_ex_2.vhd

```
3  LIBRARY ieee;
4  USE ieee.std_logic_1164.ALL;
5  ENTITY signal_ex2 IS
6      PORT(
7              a,b,c,d :IN STD_LOGIC;
8              y: OUT STD_LOGIC);
9  END signal_ex2;
10 ARCHITECTURE cct of signal_ex2 IS
11 --DECLARE SIGNAL
12     SIGNAL a_xor_b: STD_LOGIC;
13 BEGIN
14 -- Define signal in terms of ports a and b
15     a_xor_b <= ((not a) and b) or (a and (not b));
16 -- Combine signal with ports c and d
17     y <= a_xor_b or ((not c) and d);
18 END cct;
19
```

Which is better?
What does the timing simulation say?

VHDL – WHEN/ELSE construct

Concurrent conditional assignment.

(Similar to if/else in programming)

It enables the assignment of different values to a specific signal based on the validity of Boolean expressions. In cases where two or more Boolean expressions evaluate to true, only the one corresponding to the first valid expression is assigned.

This design pattern is translated to multiplexers when generating the circuit.

```
input_signal = '1';
output_signal <= '1' WHEN input_signal = '0' ELSE
                 '0' WHEN input_signal = '1' ELSE
                 (others => '0')
```

VHDL – WITH/SELECT construct

Concurrent conditional assignment.

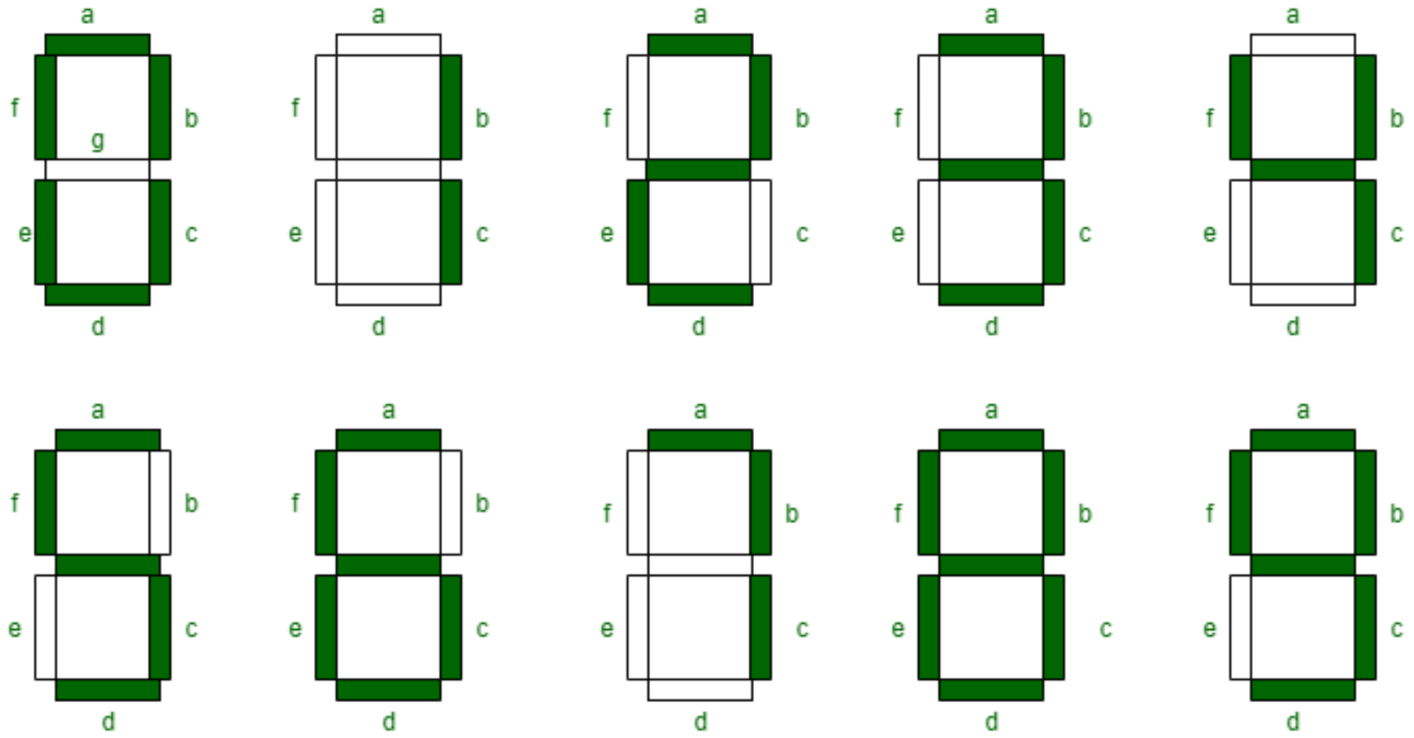
(Similar to switch case in programming)

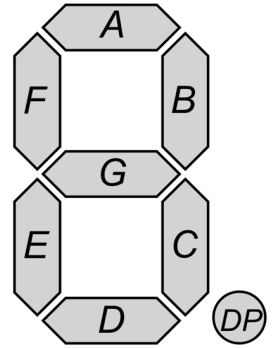
It is used when assigning a value based on constant expression.

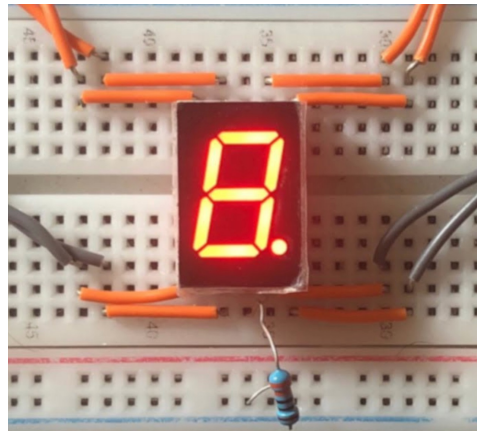
The design is synthetized with multiplexers.

```
16  MUX4: WITH s SELECT
17                y <=    d(0) WHEN "00",
18                        d(1) WHEN "01",
19                        d(2) WHEN "10",
20                        d(3) WHEN "11";
```

VHDL – The 7-segment display



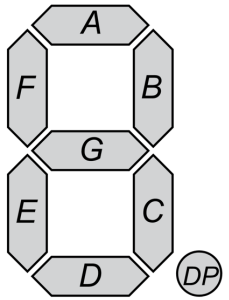




VHDL – 7-segment display example

bcd_7seg_dec.vhd

```
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4  ENTITY bcd_7seg_dec IS
5      PORT(
6              inputs: IN STD_LOGIC_VECTOR (3 downto 0);
7              a,b,c,d,e,f,g: OUT STD_LOGIC);
8  END bcd_7seg_dec;
9
10 ARCHITECTURE decoder of bcd_7seg_dec IS
11     SIGNAL output: STD_LOGIC_VECTOR (6 downto 0);
12 BEGIN
13     WITH inputs SELECT
14     output <=   "1111110" WHEN "0000",
15                 "0110000" WHEN "0001",
16                 "1101101" WHEN "0010",
17                 "1111001" WHEN "0011",
18                 "0110011" WHEN "0100",
19                 "1011011" WHEN "0101",
20                 "1011111" WHEN "0110",
21                 "1110000" WHEN "0111",
22                 "1111111" WHEN "1000",
23                 "1111011" WHEN "1001",
24                 "0000000" WHEN others;
25 -- Separate the output vector to make individual pin outputs
26     a <= output(6);
27     b <= output(5);
28     c <= output(4);
29     d <= output(3);
30     e <= output(2);
31     f <= output(1);
32     g <= output(0);
33 END decoder;
```

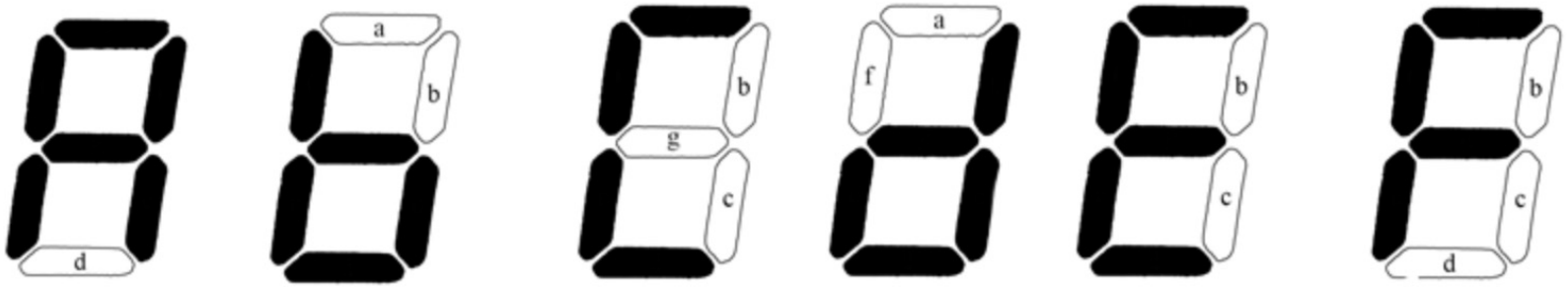


A	B	C	D		F_a	F_b	F_c	F_d	F_e	F_f	F_g
0	0	0	0								
0	0	0	1								
0	0	1	0								
0	0	1	1								
0	1	0	0								
0	1	0	1								
0	1	1	0								
0	1	1	1								
1	0	0	0								
1	0	0	1								
1	0	1	0								
1	0	1	1								
1	1	0	0								
1	1	0	1								
1	1	1	0								
1	1	1	1								

VHDL – 7-segment display exercise

Exercise 2. Extend the *bcd_7seg_dec.vhd* VHDL file to decode hexadecimal values (a, b, c, d, e, f).

Test the digital design with a functional testbed (as in Lab2)



References

- Tokheim, R., *Digital Electronics: Principles And Applications*.
- Floyd, Thomas L., *Digital Fundamentals*.
- Mano, Morris. *Digital Design*.
- Intel QUARTUS Help Manuals
- Tocci, Ronald. *Digital Systems Principles and Applications*.