EE-421: Digital System Design

Structural vs Behavioural Verilog

Instructor: Dr. Rehan Ahmed [rehan.ahmed@seecs.edu.pk]

NUST

SCHOOL OF ELECTRICAL ENGINEERING &
COMPUTER SCIENCE (SEECS)

Structural vs. Behavioral Implementation

Two Main Styles of HDL Implementation

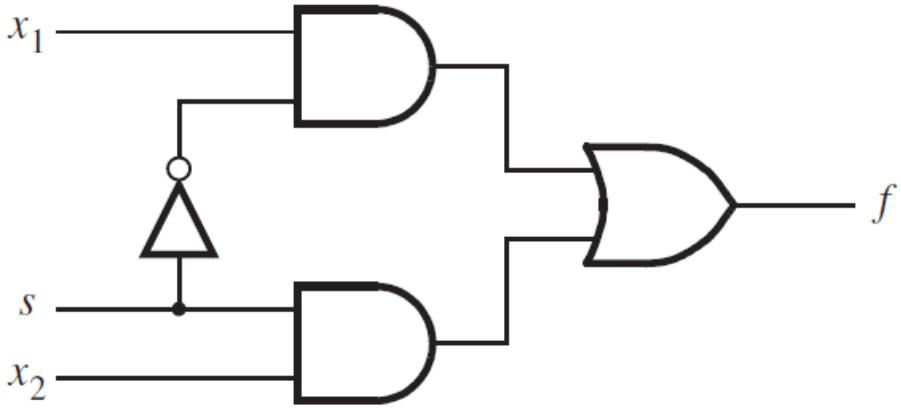
- **Structural (Gate-Level)**
  - The module body contains gate-level description of the circuit
  - Describe how modules are interconnected
  - Each module contains other modules (instances)
  - … and interconnections between these modules
  - Describes a hierarchy

- **Behavioral**
  - The module body contains functional description of the circuit
  - Contains logical and mathematical operators
  - **Level of abstraction is higher than gate-level**
    - Many possible gate-level realizations of a behavioral description

- **Practical circuits use a combination of both**

Structural HDL (Gate-Level)

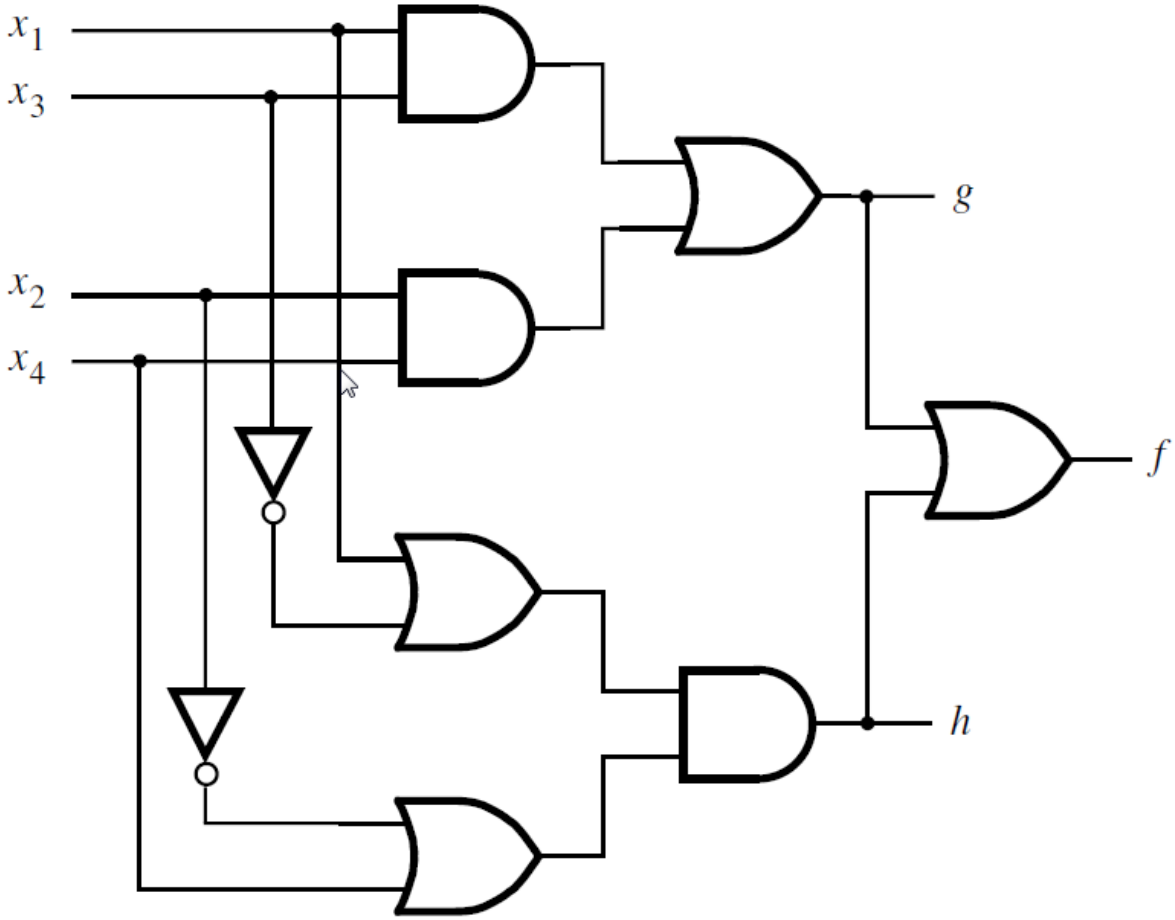
Structural Specification of Logic Ccts i.e Gate-Level Primitives




```
module example1 (x1, x2, s, f);
    input x1, x2, s;
    output f;

    not (k, s);
    and (g, k, x1);
    and (h, s, x2);
    or (f, g, h);

endmodule
```

Structural Specification of Logic Ccts i.e Gate-Level Primitives

```
module example2 (x1, x2, x3, x4, f, g, h);
        input x1, x2, x3, x4;
        output f, g, h;

        and (z1, x1, x3);
        and (z2, x2, x4);
        or (g, z1, z2);
        or (z3, x1, ~x3);
        or (z4, ~x2, x4);
        and (h, z3, z4);
        or (f, g, h);

endmodule
```

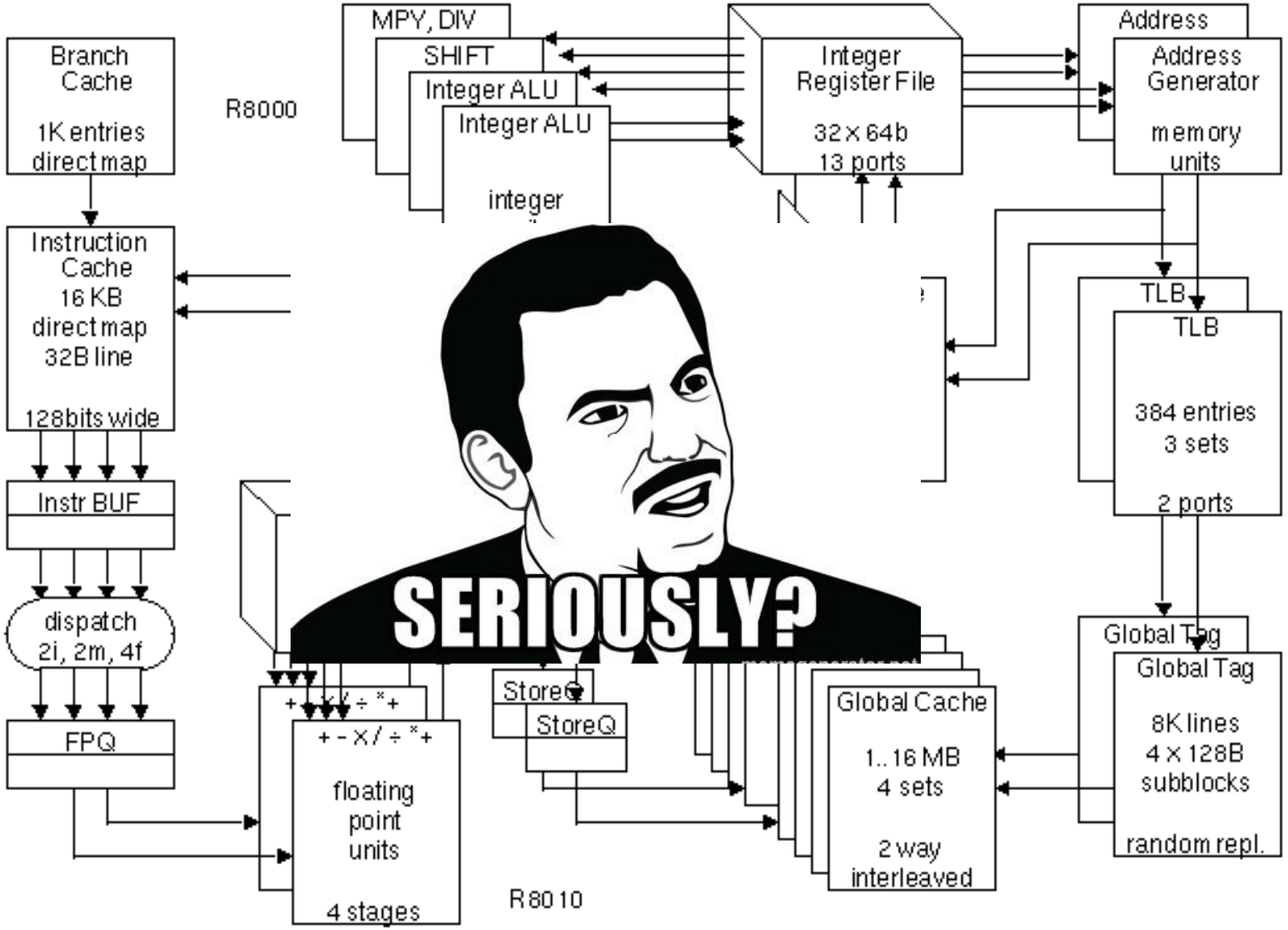


Gate Primitives: For Reference

Name	Description	Usage
and	$f = (a \cdot b \cdots)$	**and** (f, a, b,…)
nand	$f = \overline{(a \cdot b \cdots)}$	**nand** (f, a, b, …)
or	$f = (a + b + \cdots)$	**or** (f, a, b, …)
nor	$f = \overline{(a + b + \cdots)}$	**nor** (f, a, b, …)
xor	$f = (a \oplus b \oplus \cdots)$	**xor** (f, a, b, …)
xnor	$f = (a \square b \square \cdots)$	**xnor** (f, a, b, …)
not	$f = \overline{a}$	**not** (f, a)
buf	$f = a$	**buf** (f, a)

Your Turn

Express the following using Structural HDL



Behavioral HDL

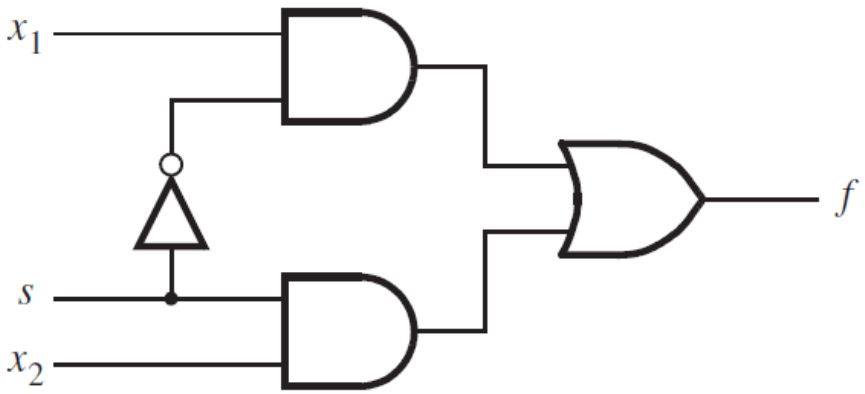
- **Behavioral HDL to Rescue!!!**

- *Behavioral* constructs specify an operation on bits
  - High-level, more abstract
    - Specified as behavior of a logic cct, e.g., out = (a & b) | c

Behavioral Specification of Logic Circuits

- **Using logic expressions:** [f = s'x1 + sx2]

```
module example3 (x1, x2, s, f);
    input x1, x2, s;
    output f;

    assign f = (~s & x1) | (s & x2);

endmodule
```

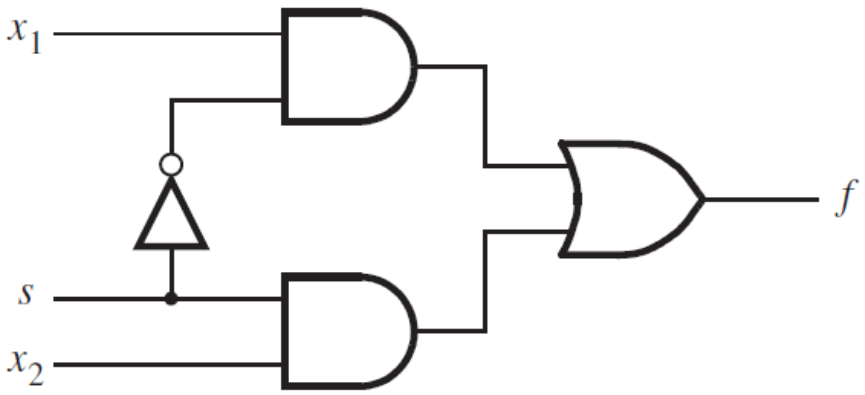


- The ***assign*** keyword provides a ***continuous assignment*** for the signal f:
  - whenever any signal on the right-hand side changes its state, the value of f will be re-evaluated.

Behavioral Specification of Logic Circuits

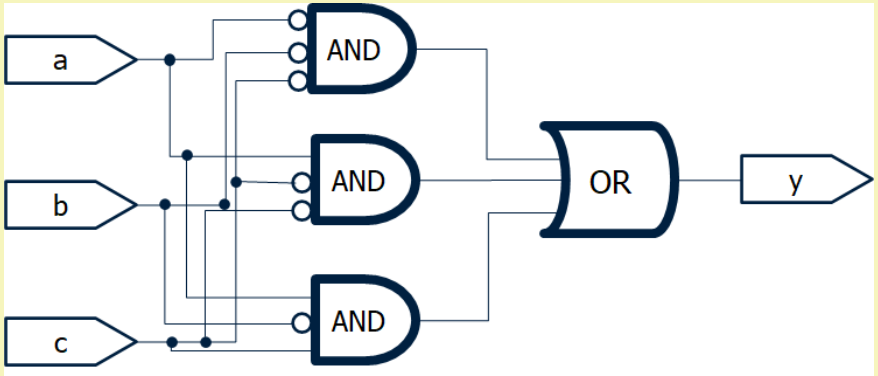
- **Using behavioral specification.**

```
// Behavioral specification
module  example5 (x1, x2, s, f);
    input  x1, x2, s;
    output f;
    reg f;

    always @(x1 or x2 or s)
        if (s == 0)
            f = x1;
        else
            f = x2;

endmodule
```

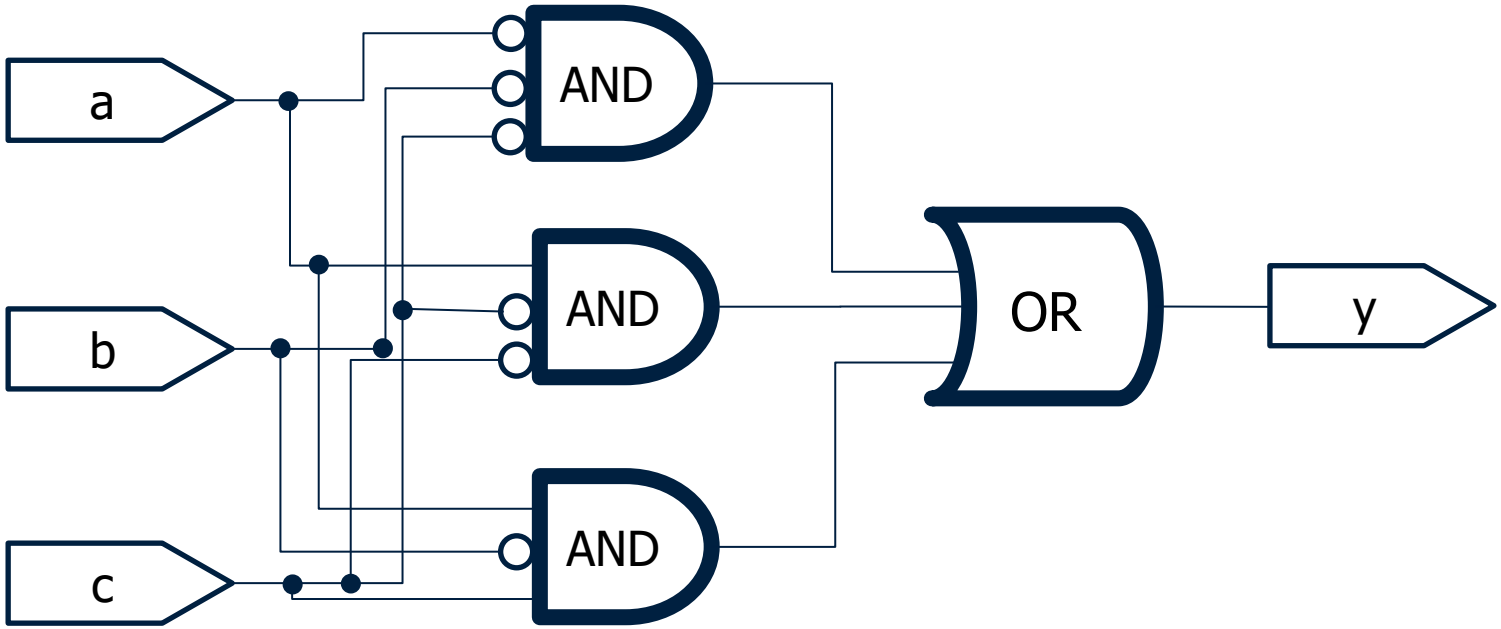


Behavioral HDL: Defining Functionality

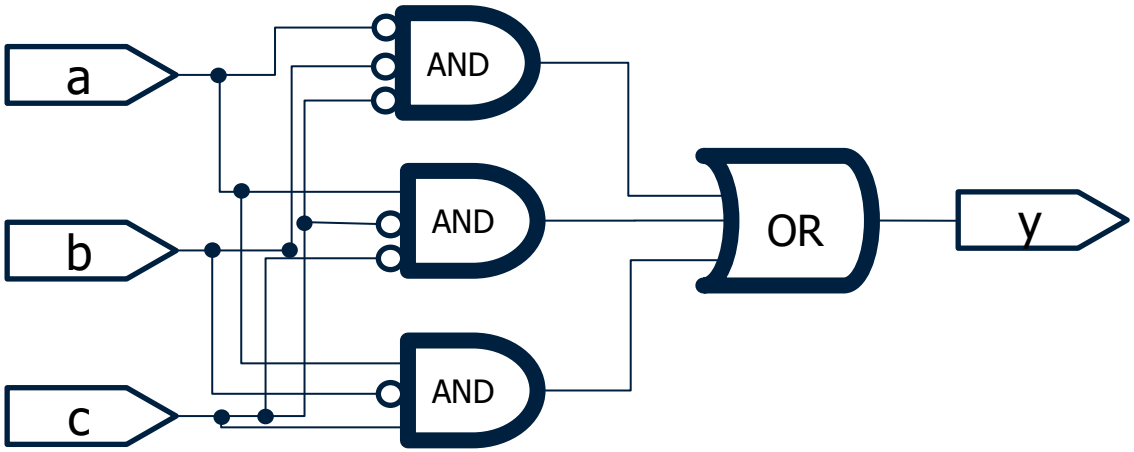
```
module example (a, b, c, y);
       input a;
       input b;
       input c;
       output y;

// here comes the circuit description
assign y = ~a & ~b & ~c |
            a & ~b & ~c |
            a & ~b &  c;

endmodule
```

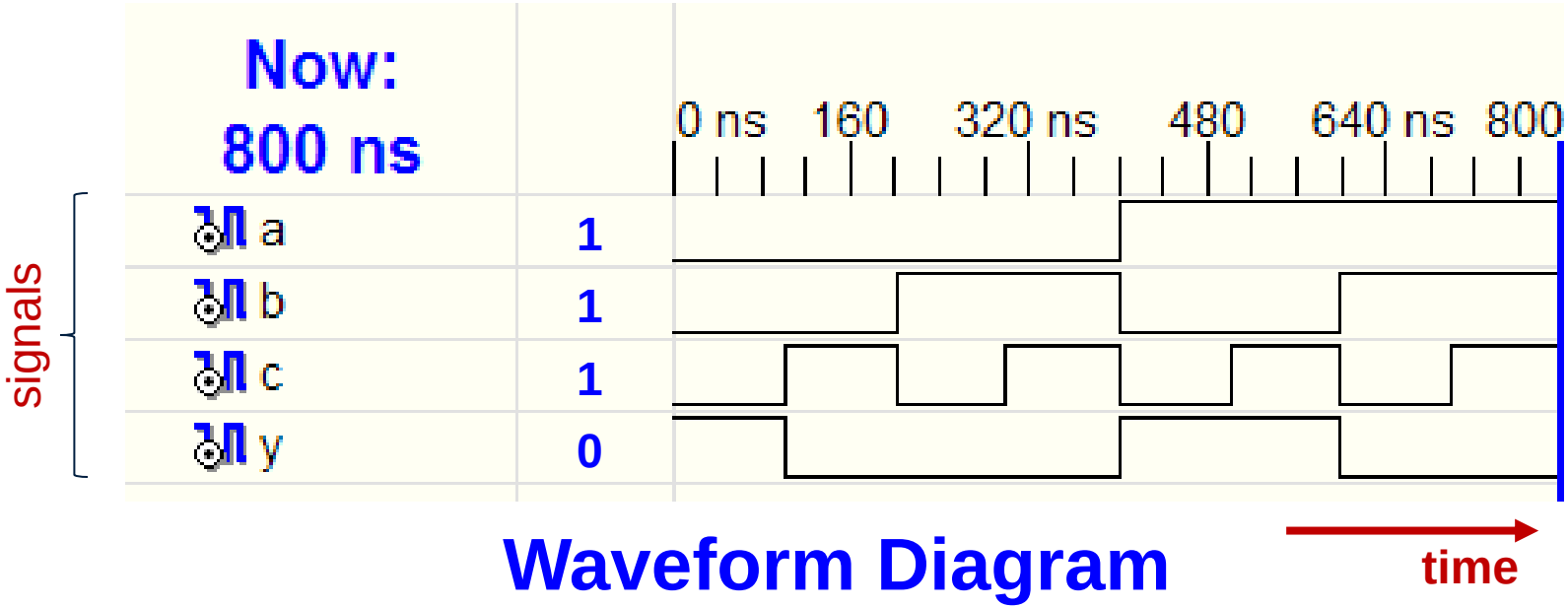


Remember: Synthesizing the “example”



Remember: Simulating the "example"





Recommended Reading

- S&Z, 2.10 and Appendix-A