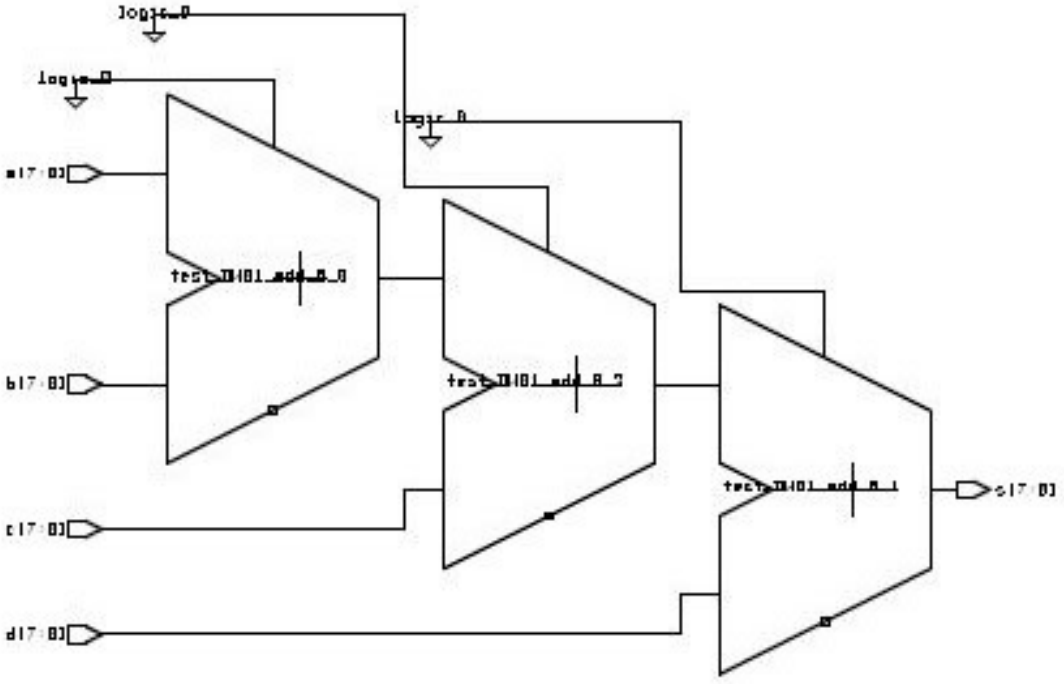
Lesson 19

Data Path and Control Path

Mohd Saufy Rohmad
Technical Trainer and Consultant

- Finite state machine is the brain of digital circuit.
- It can be viewed as a control of the circuit.
- Typically, the datapath is controlled with a FSM.
- In this hour, we will take a look at an example of an adder that can add 4-values.
- Lets assume, we want to add, a, b, c, d.
- If the output is s, the code can be written as,
- s <= ((a + b) + c) + d ;
- However, this results in a large circuit because the circuit will have 3 adders as shown in the figure,

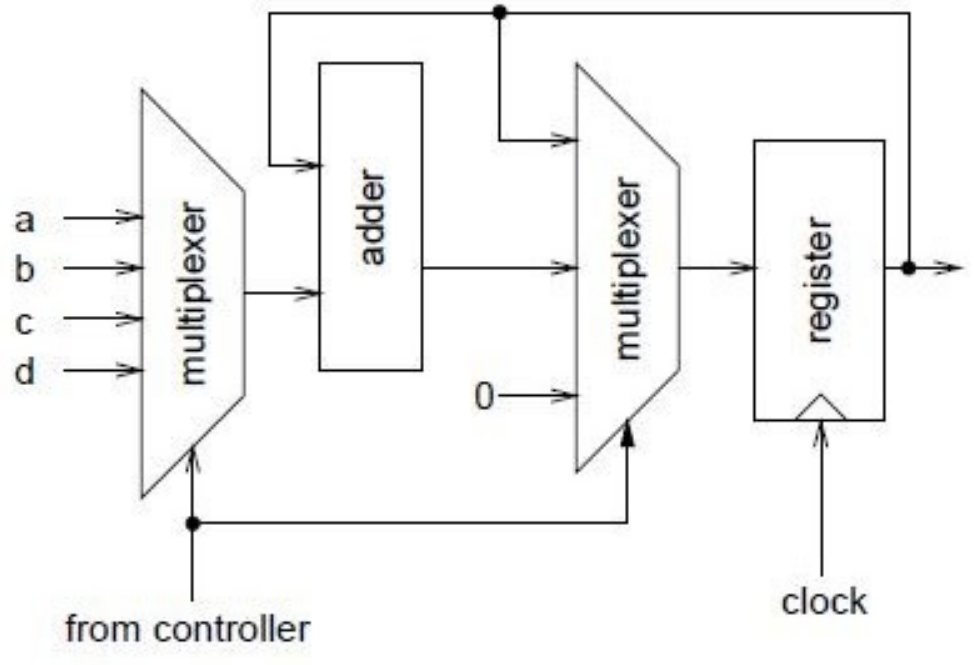
Data Path Control Unit



Data Path Control Unit

- We want to use one adder and use it sequentially to add up all the four values.
- This is when we will need FSM/a control path.
- Lets take a look how the circuit might look like,

Data Path Control Unit



- First a package is used to define the states just like the last FSM that we learned in lesson 18.

```
-- subtype used in design
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
package averager_types is
    subtype num is unsigned (7 downto 0) ;
    type states is (clr, add_a, add_b, add_c,
    add_d, hold) ;
end averager_types ;
```

Datapath

- Next we have the datapath which does all the necessary works based on the 2-bit input sel and 1-bit inputs load and clear from the control path.

```
-- datapath
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use work.averager_types.all ;
entity datapath is
port (
    a, b, c, d : in num ;
    sum : out num ;
    sel : in std_logic_vector (1 downto 0) ;
    load, clear, clk : in std_logic
) ;
end datapath ;
architecture rtl of datapath is
    signal mux_out, sum_reg, next_sum_reg : num ;
    constant sum_zero : num :=0;
    conv_unsigned(0,next_sum_reg'length) ;
```

datapath

```
begin
    -- mux to select input to add
    with sel select mux_out <=
    a when "00",
    b when "01",
    c when "10",
    d when others ;
    -- mux to select register input
    next_sum_reg <=
    sum_reg + mux_out when load = '1' else
    sum_zero when clear = '1' else
    sum_reg ;
    -- register sum
    process(clk)
    begin
        if clk'event and clk = '1' then
        sum_reg <= next_sum_reg ;
        end if ;
    end process ;
    -- entity output is register output
    sum <= sum_reg ;
end rtl ;
```

datapath

```
-- controller
library ieee ;
use ieee.std_logic_1164.all ;
use work.averager_types.all ;
entity controller is
port (
    update : in std_logic ;
    sel : out std_logic_vector (1 downto 0) ;
    load, clear : out std_logic ;
    clk : in std_logic
) ;
end controller ;
architecture rtl of controller is
    signal s, holdns, ns : states ;
    signal tmp : std_logic_vector (3 downto 0) ;
begin
```

controller

```
    -- select next state
    with s select ns <=
    add_a when clr,
    add_b when add_a,
    add_c when add_b,
    add_d when add_c,
    hold when add_d,
    holdns when others ;
-- hold
    -- next state if in hold state
    holdns <=
    clr when update = '1' else
    hold ;
    -- state register
    process(clk)
    begin
        if clk'event and clk = '1' then
        s <= ns ;
        end if ;
    end process ;
     -- controller outputs
    with s select sel <=
        "00" when add_a,
        "01" when add_b,
        "10" when add_c,
        "11" when others ;
    load <= '0' when s = clr or s = hold else '1' ;
    clear <= '1' when s = clr else '0' ;
end rtl ;
```

Control unit

- The controller and the datapath are placed in a package. We will discuss about the package more in the next hour.

```
-- package for datapath and controller
library ieee ;
use ieee.std_logic_1164.all ;
use work.averager_types.all ;
package averager_components is
    component datapath
    port (
    a, b, c, d : in num ;
    sum : out num ;
    sel : in std_logic_vector (1 downto 0) ;
    load, clear, clk : in std_logic
    ) ;
    end component ;
    component controller
    port (
    update : in std_logic ;
    sel : out std_logic_vector (1 downto 0) ;
    load, clear : out std_logic ;
    clk : in std_logic
    ) ;
    end component ;
end averager_components ;
```

package

```
-- averager
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use work.averager_types.all ;
use work.averager_components.all ;
entity averager is port (
a, b, c, d : in num ;
sum : out num ;
update, clk : in std_logic ) ;
end averager ;
architecture rtl of averager is
    signal sel : std_logic_vector (1 downto 0) ;
    signal load, clear : std_logic ;
    -- other declarations (e.g. components) here
begin
    d1: datapath port map ( a, b, c, d, sum, sel, load,
    clear, clk ) ;
    c1: controller port map ( update, sel, load,
    clear, clk ) ;
end rtl ;
```

Top module

Thank You