

Chapter 2 continued...

- Combinational Logic (should be a review of 120A)
- Beginning of state machine design

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Valdig/Givargis 2

Combinational logic design

A) Problem description

y is 1 if a is 1, or b and c are 1.
z is 1 if b or c is 1, but not both, or if all are 1.

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Valdig/Givargis 3

Combinational logic design

A) Problem description

y is 1 if a is 1, or b and c are 1.
z is 1 if b or c is 1, but not both, or if all are 1.

B) Truth table

Inputs			Outputs	
a	b	c	y	z
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Valdig/Givargis 4

Combinational logic design

A) Problem description

y is 1 if a is 1, or b and c are 1.
z is 1 if b or c is 1, but not both, or if all are 1.

C) Output equations

$$y = a'bc + ab'c' + ab'c + abc' + abc$$

$$z = a'bc' + a'bc + ab'c + abc' + abc$$

B) Truth table

Inputs			Outputs	
a	b	c	y	z
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Combinational logic design

A) Problem description

y is 1 if a is 1, or b and c are 1.
z is 1 if b or c is 1, but not both, or if all are 1.

B) Truth table

Inputs			Outputs	
a	b	c	y	z
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

C) Output equations

$$y = a'bc + ab'c' + ab'c + abc' + abc$$

$$z = a'bc' + a'bc + ab'c + abc' + abc$$

D) Minimized output equations

$$y = a + bc$$

$$z = ab + b'c + bc'$$

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Valdig/Givargis 6

Combinational logic design

A) Problem description

y is 1 if a is 1, or b and c are 1.
z is 1 if b and c is 1, but not both, or if all are 1.

B) Truth table

Inputs	Outputs
a b c	y z
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	1 0
1 0 1	1 1
1 1 0	1 1
1 1 1	1 1

C) Output equations

$$y = a'bc + ab'c' + ab'c + abc' + abc$$

$$z = ab'c + a'bc' + ab'c + abc' + abc$$

D) Minimized output equations

$y = a + bc$

$z = ab + b'c + bc'$

E) Logic Gates

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis 7

Example to Work out...

A) Problem description

y is 1 if a is 0, or they are all 1.
z is 1 if b and c is 1, or if all are 0.

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis 8

Example to Work out...

A) Problem description

y is 1 if a is 0, or they are all 1.
z is 1 if b and c is 1, or if all are 0.

B) Truth table

Inputs	Outputs
a b c	y z
0 0 0	1 1
0 0 1	1 0
0 1 0	1 0
0 1 1	1 1
1 0 0	0 0
1 0 1	0 0
1 1 0	1 0
1 1 1	0 0

C) Output equations

$$y = a'b'c' + a'b'c + a'bc' + abc + abc$$

$$z = a'b'c' + a'bc + abc$$

D) Minimized output equations

$y = a' + bc$

$z = a'b'c' + bc$

E) Logic Gates

Do you see room for further improvement (minimizing space)?

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis 9

Optimization

Logic Gates

→

after removing redundancy

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis 10

ICE!

Logic Gates

→

after removing redundancy

How many transistors did we save?

Hint: For a 3-input AND, just cascade two 2-input AND's.
For a 3-input NOR cascade an OR and a NOR

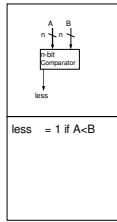
Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis 11

Common Combinational Components

<p>Item-1 If 0 Item-0 If 1 S₀ = 0 S₁ = 1 S_{log} = m O</p>	<p>A B carry sum</p>	<p>A B less equal greater</p>	<p>A B n-bit n-function ALU S₀ S_{log} O</p>
<p>O = 0 if S=0..00 I1 if S=0..01 ... I(m-1) if S=1..11</p>	<p>sum = A+B (first n bits) carry = (n+1)th bit of A+B</p>	<p>less = 1 if A<B equal = 1 if A=B greater=1 if A>B</p>	<p>O = A op B op determined by S. Can be add, subtract, multiply, lots of things...</p>
	<p>With carry-in input Ci → sum = A + B + Ci</p>		<p>May have status outputs carry, zero, etc.</p>

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis 12

Combinational components (VHDL)



Creating the VHDL for combinational components does not require a process, but you can have one if you want to:

```
architecture RTL of COMPARATOR is
begin
  LESS <= '1' when A < B else '0';
end RTL;

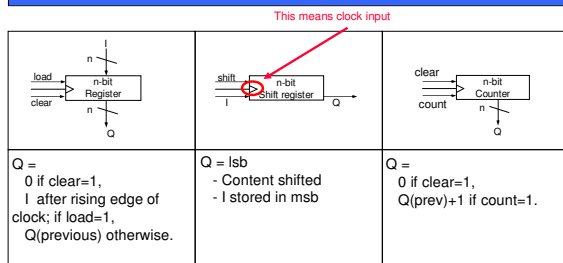
--or--

architecture RTL of COMPARATOR is
begin
  process (A, B) is
  if A < B
    LESS <= '1';
  else
    LESS <= '0';
  end if;
end process;
end RTL;
```

Sequential components

Sequential Components, like the name implies, happen in time, and therefore require a clock--unlike Combinational, which have no clock.

Sequential components



Q =
0 if clear=1,
1 after rising edge of
clock; if load=1,
Q(previous) otherwise.

Q = lsb
- Content shifted
- l stored in msb

Q =
0 if clear=1,
Q(prev)+1 if count=1.

Sequential components

How do you create the VHDL for sequential components?

Make a register starting with this entity:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity REG is
port (CLK, CLEAR, LOAD : in std_logic;
      INPUT : in std_logic_vector (7 downto 0);
      OUTPUT : out std_logic_vector (7 downto 0)
);
end REG;
```

VHDL for a Register

```
library IEEE;
use IEEE.std_logic_1164.all;
entity REG is
port (
  CLK, CLEAR, LOAD : in std_logic;
  INPUT : in std_logic_vector (7 downto 0);
  OUTPUT : out std_logic_vector (7 downto 0)
);
end REG;
```

```
architecture RTL of REG is
signal OUT_REGISTER : std_logic_vector (7 downto 0);
begin
  process (CLK, CLEAR, LOAD, INPUT)
  begin
    if (CLK'event and CLK='1') then
      if CLEAR = '1' then
        OUT_REGISTER <= "00000000";
      elsif LOAD = '1' then
        OUT_REGISTER <= INPUT;
      else
        OUT_REGISTER <= OUT_REGISTER;
      end if;
    end if;
  end process;
  OUTPUT <= OUT_REGISTER;
end RTL;
```

Sequential logic design

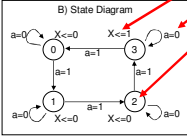
A) Problem Description

Slow down your pre-existing clock so that you output a 1 for every time four clock pulses as long as 'a' is 1.

- Given this implementation model
 - Sequential logic design quickly reduces to combinational logic design

Sequential logic design

A) Problem Description
Slow down your pre-existing clock so that you output a 1 for every time four clock pulses as long as 'a' is 1.

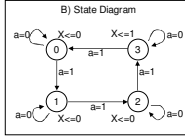


X is the output
a is the input
The numbers inside the circles are the state number (or label)

- Given this implementation model
 - Sequential logic design quickly reduces to combinational logic design

Sequential logic design

A) Problem Description
Slow down your pre-existing clock so that you output a 1 for every time four clock pulses as long as 'a' is 1.



A state machine is sequential.

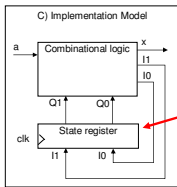
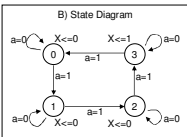
On every rising clock edge, the next state is determined by TWO things:

The **present state**, and the **inputs**.

To implement a state machine, we need some way to store the current state. To do that, we use a *register*.

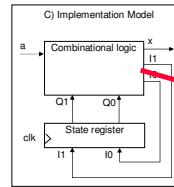
Sequential logic design

A) Problem Description
Slow down your pre-existing clock so that you output a 1 for every time four clock pulses as long as 'a' is 1.

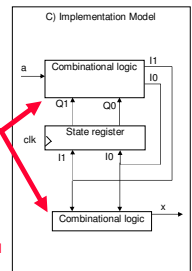


I1 is one bit, and I0 is one bit, so this register is only two bits wide!

Sequential logic design

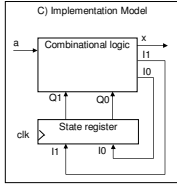
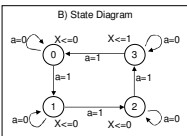


NOTE: We are using a Moore State machine
So, the output is **ONLY** dependent on the state and **NOT** the input. Therefore the combinational logic can be split up if you want to.



Sequential logic design

A) Problem Description
Slow down your pre-existing clock so that you output a 1 for every time four clock pulses as long as 'a' is 1.



D) State Table (Moore-type)

Inputs		Outputs			
Q1	Q0	a	I1	I0	x
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	1
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	0	0	1

Remember how to turn this into logic components?

ICE!

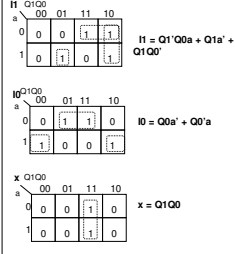
Why can we draw out output X with boxes spanning two rows?

D) State Table (Moore-type)

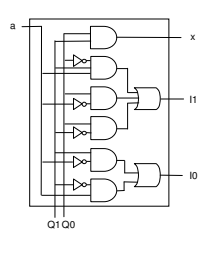
Inputs		Outputs			
Q1	Q0	a	I1	I0	x
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	1
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	0	0	1

Sequential logic design (cont.)

E) Minimized Output Equations

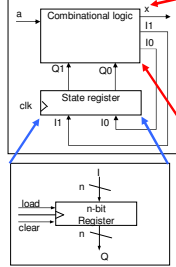


F) Combinational Logic

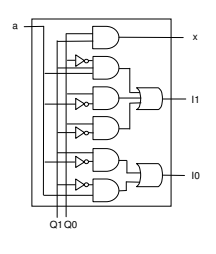


Sequential logic design

C) Implementation Model



F) Combinational Logic



To Do before Tuesday...

- Finish Homework!
- Read through Chapter 2.