ECE 316 – HW#3 - A

Simple ALU

with

Hardware Implementation

Objective

- 8-bit signed adder
  - X, Y (1 bit sign, 7 bit value)
- Multiple operations

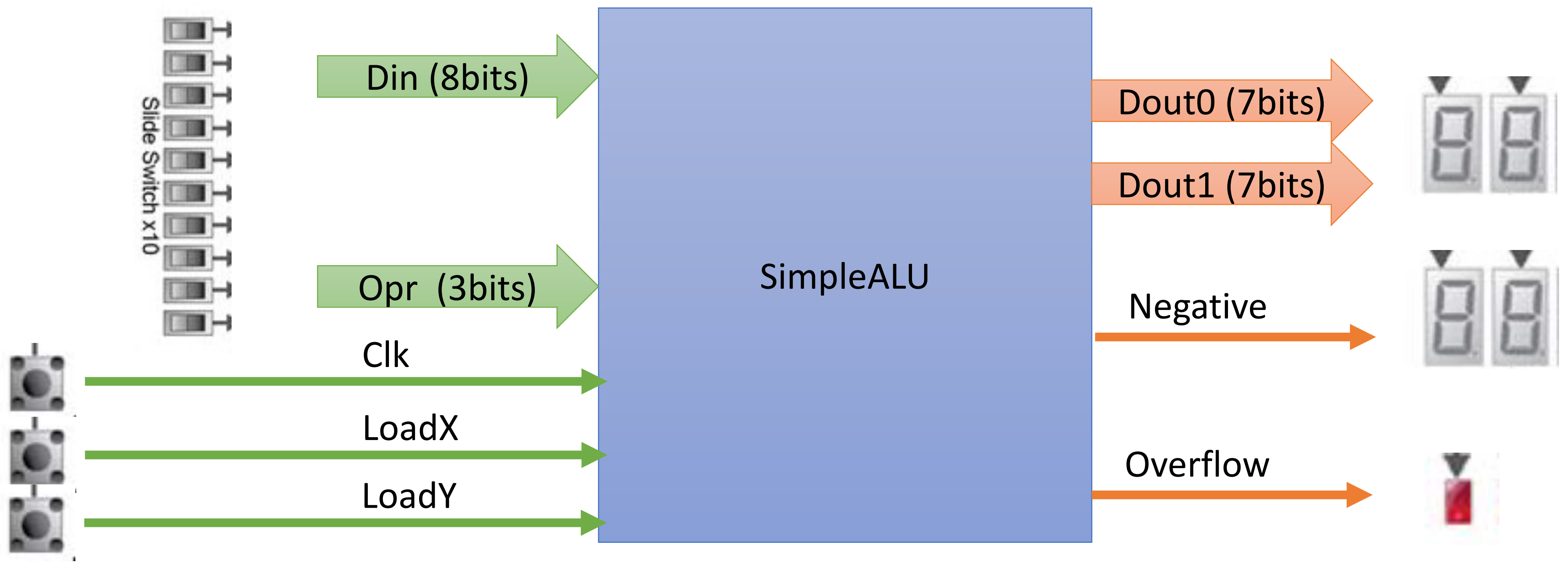
X (+0)	(0) - Y
X + 1	X + Y
X - 1	X - Y
(0) - X	Y - X

- Binary input
- Decimal Output

SimpleALU – Inputs/Outputs

Pin/Bus	I/O	Size	Name
Din	Input	8-bits	Data load inputs
LoadX	Input	1-bit	Load X Register (active LOW)
LoadY	Input	1-bit	Load Y Register (active LOW)
Opr	Input	3-bits	Operation
Clk	Input	1-bit	Register Clock (Pulse, Falling Edge)
Dout0	Output	7-bits	7-segment Low Order digit
Dout1	Output	7-bits	7-segment High Order digit
Negative	Output	1-bit	Segment 6 on 7-segment display
Overflow	Output	1-bit	Overflow flag

SimpleALU – External View



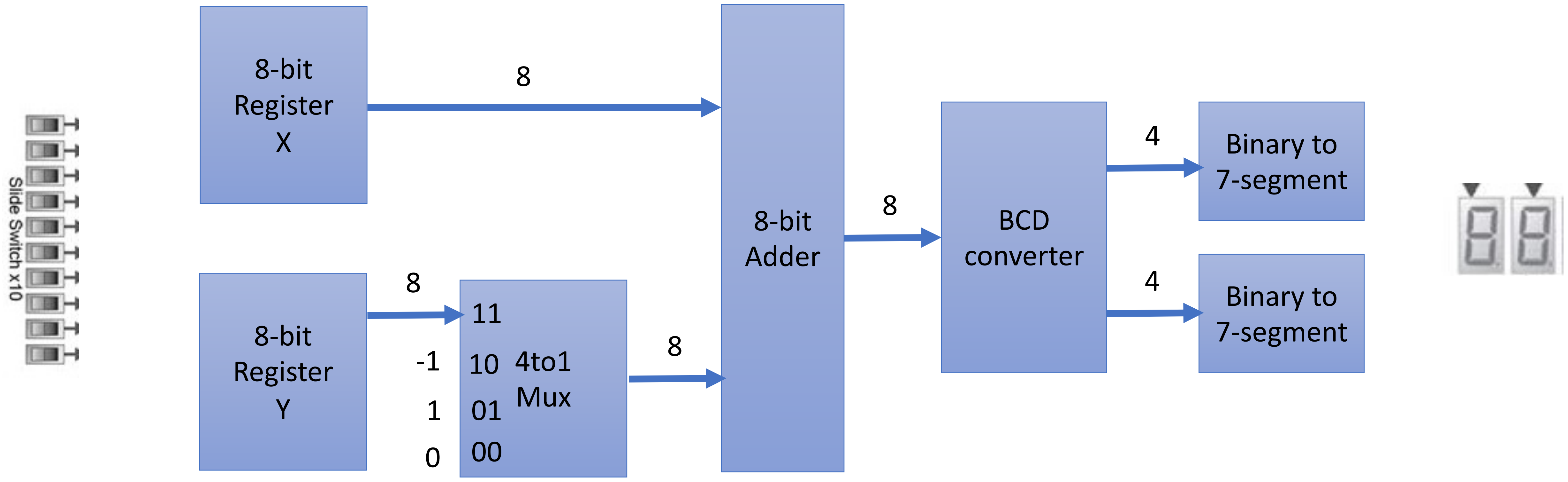
SimpleALU - Operations

LoadX LoadY	Operation
11	Disable Load (default)
01	Load Register X
10	Load Register Y
00	Load X and Y

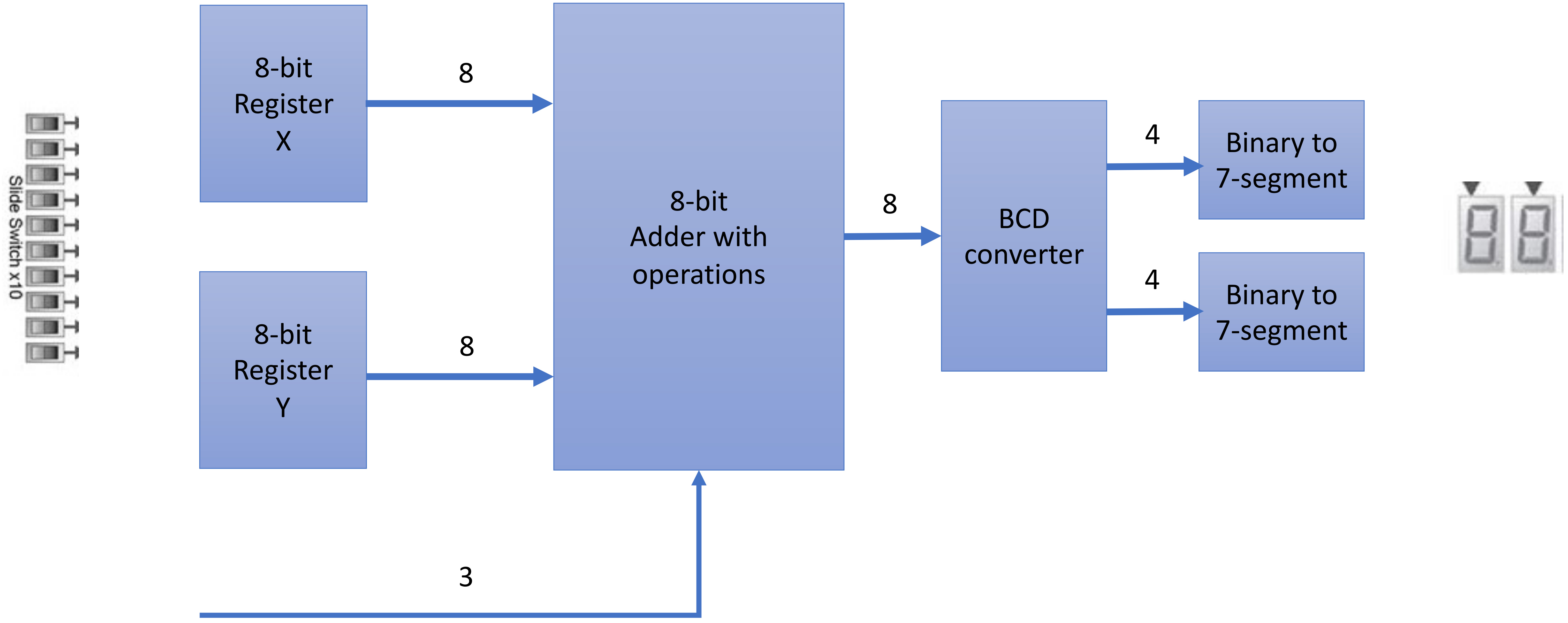
Opr	Operation/Output
0000	X (+0)
0001	X + 1
0010	X - 1
0011	-X
0100	-Y
0101	X + Y
0110	X - Y
0111	Y - X

Constraints:
- X, Y are in range -99 to 99
- Results are rage -99 to 99
- Overflow if outside range
- negative as 2’s complement

Structural Design – from 3A – NOT VALID



Structural Design – from 3B – Ver 1.



Process Statement

```
Process (sensitivity-list)
--declaration
begin
--sequential statements
end process;
```

It is not allowed to use concurrent statements inside a process

- Sensitivity list is a list of signals
- Any signal change inside sensitivity list will cause the process to execute
- Process itself is a concurrent statement

Sensitivity List Example

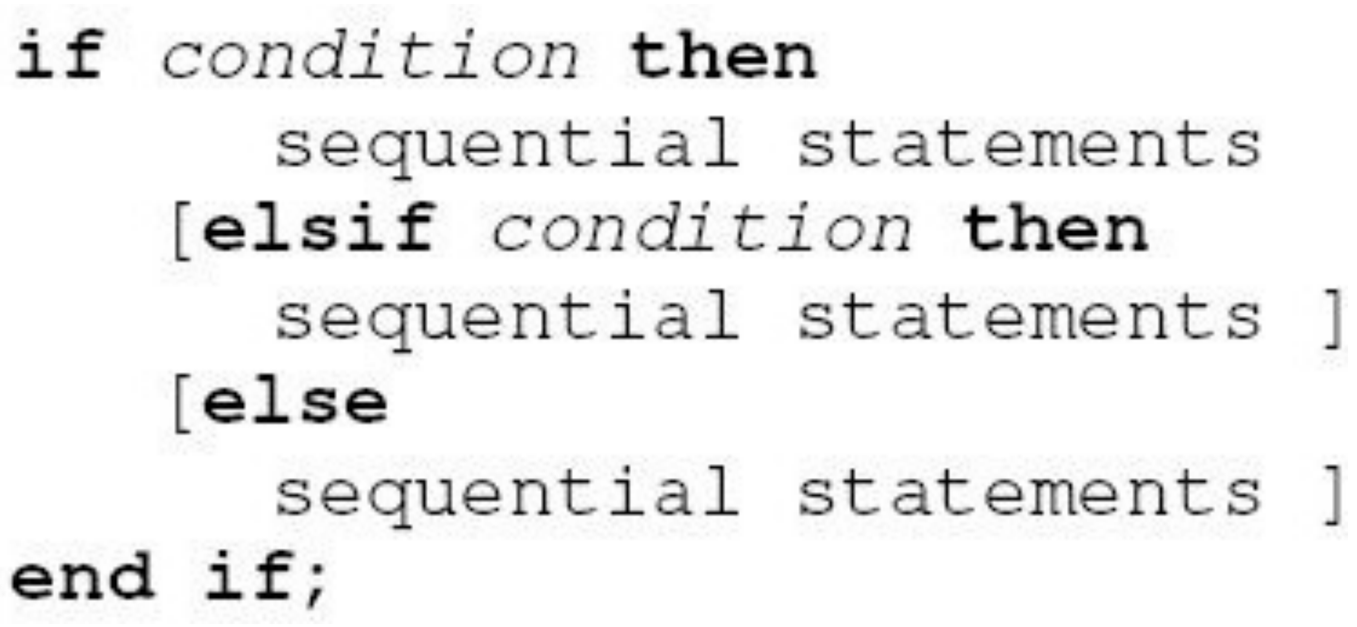
```
process (input_1, input_2)

begin
    and_gate <= input_1 and input_2;
 end
Process;
```

- A change on either input_1 or input_2 will cause the Process to execute

Sequential Statements

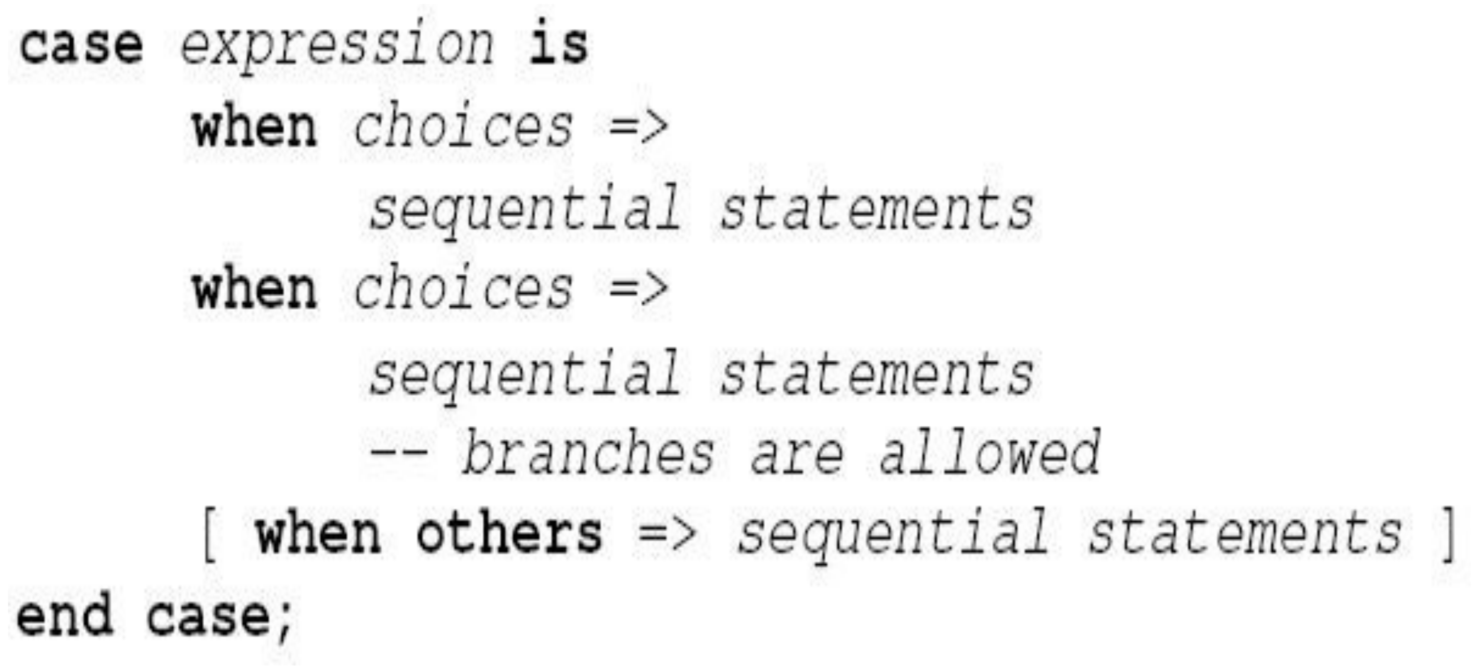
Concurrent Statements	Sequential Statement
Assignment (arithmetic and logic)	Assignment (arithmetic and logic)
When … else	If … else
With	Case
Port map	While … loop
	For … loop

Sequential Statements:
if … else



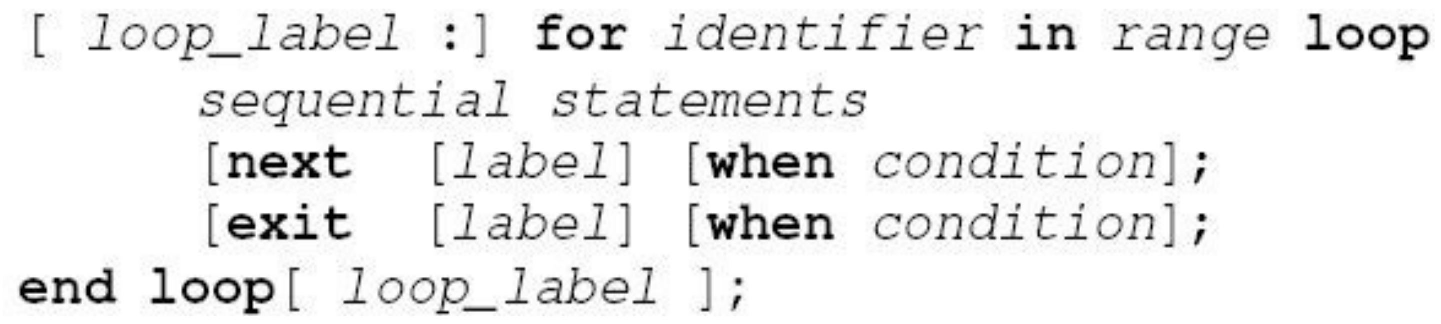
```
if condition then
      sequential statements
    [elsif condition then
      sequential statements ]
    [else
      sequential statements ]
end if;
```

Sequential Statements: case



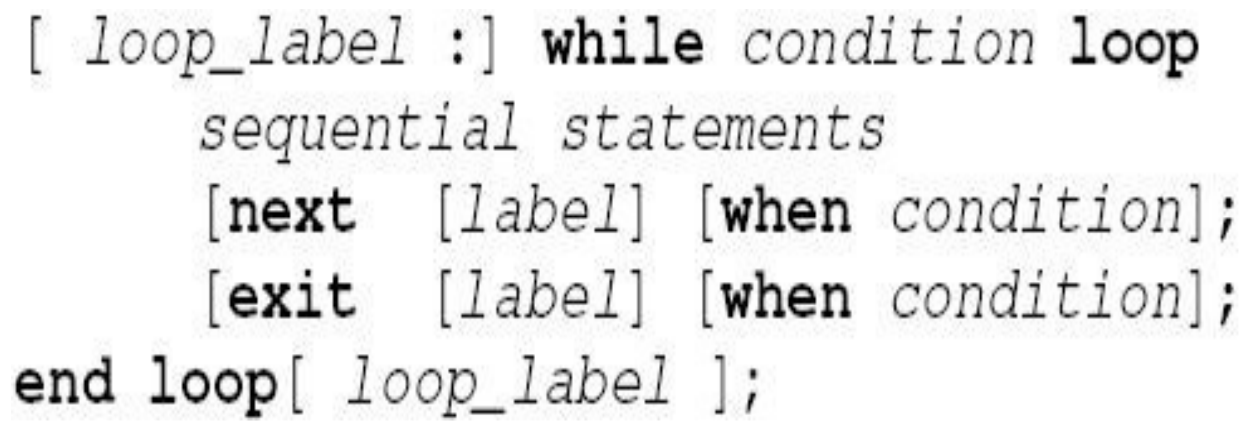
```
case expression is
     when choices =>
          sequential statements
     when choices =>
          sequential statements
          -- branches are allowed
     [ when others => sequential statements ]
end case;
```

Sequential Statements: for … loop



```
[ loop_label :] for identifier in range loop
      sequential statements
      [next   [label] [when condition];
      [exit   [label] [when condition];
end loop[ loop_label ];
```

Sequential Statements: while … loop

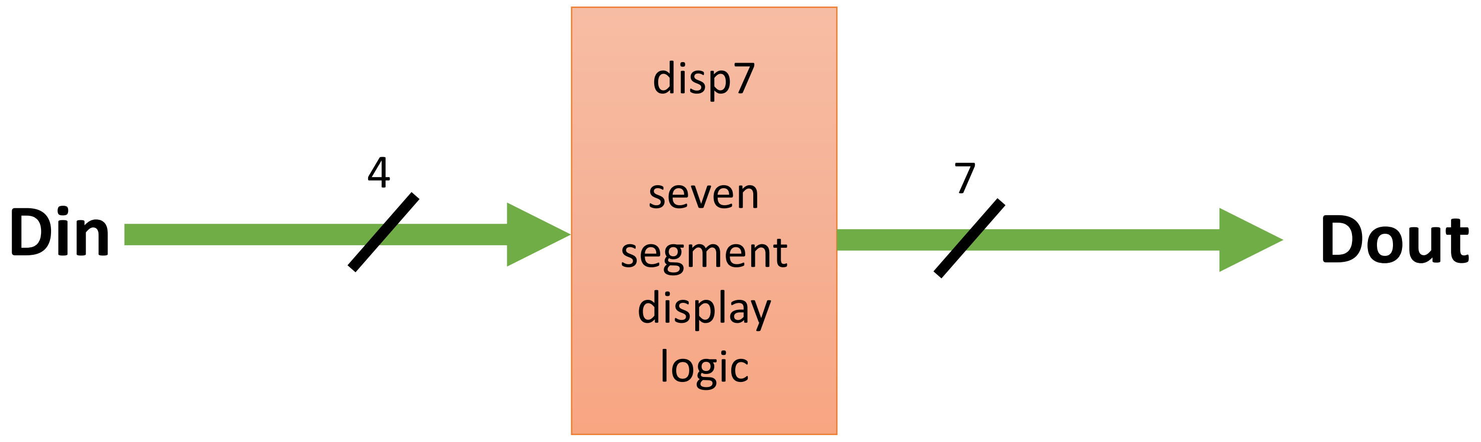


```
[ loop_label :] while condition loop
      sequential statements
      [next  [label] [when condition];
      [exit  [label] [when condition];
end loop[ loop_label ];
```

```
Library IEEE;
Use IEEE.std_logic_1164.all;

Entity xor_sig is

Port (
A, B,C: in STD_LOGIC;
X,Y: out STD_LOGIC
);
End xor_sig

Architecture SIG_ARCH of xor_sig is
Signal D: STD_LOGIC;
Signal E:  STD_LOGIC;
Signal F:  STD_LOGIC;

Begin
Process(A,B,C)
        Begin
                D <=A;          -- ignored
                E <= D xor F; --OLD value of D
                F <= B xor D; --OLD value of D
                X <= C xor E; --OLD value of E
                D <= B;         --D's value gets overwritten
        by B after process ends
                Y <= D xor F; --OLD value of D and F
End process
End SIG_ARCH
```

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

Entity xor_var is
Port (
A, B,C: in STD_LOGIC;
X,Y: out STD_LOGIC
);
End xor_var

Architecture VAR_ARCH of xor_var is
variable D:  STD_LOGIC;
variable  E:  STD_LOGIC;
variable  F:  STD_LOGIC;
variable  P:  STD_LOGIC;
variable Q:  STD_LOGIC;

Begin
        Process(A,B,C)
                Begin
                D :=A;          --D is assigned with value of A
                E := D xor F;--New value of D (A)
                F := B xor D;--New value of D (A)
                X := C xor E;--New value of E
                D := B;         --D value is changed to B
                Y := D xor F;--New value of bothe D and F

                X <= P
                Y <= Q
End process
End VAR_ARCH
```

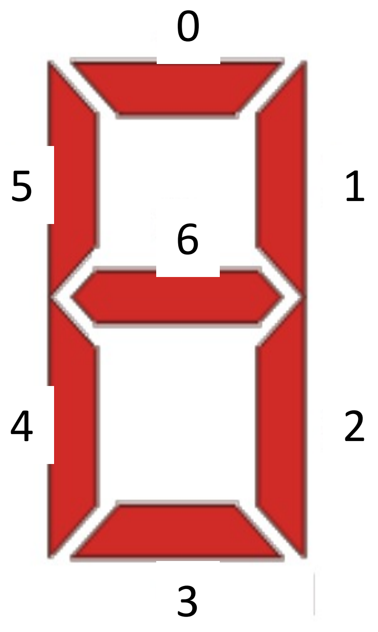
SimpleALU – Internals – disp7

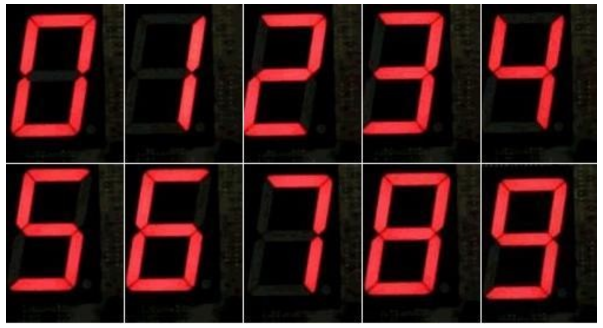


Operation:
Seven segment display logic
Converts 4-bit binary input to decimal value:
0,1,2,3,4,5,6,7,8,9
7-bits refer to segments 0-6 on display

7 Segment Display Patterns and Truth Table (ECE315)

	Input X3..X0	Display D6 … D0
0	0 0 0 0	1 0 0 0 0 0 0
1	0 0 0 1	
2	0 0 1 0	
3	0 0 1 1	
4	0 1 0 0	
5	0 1 0 1	
6	0 1 1 0	
7	0 1 1 1	
8	1 0 0 0	
9	1 0 0 1	
10	1 0 1 0	X X X X X X X X
11	1 0 1 1	X X X X X X X X
12	1 1 0 0	X X X X X X X X
13	1 1 0 1	X X X X X X X X
14	1 1 1 0	X X X X X X X X
15	1 1 1 1	X X X X X X X X





The 7 segment displays on DE1 board are designed to light up when applying a low level voltage.
Applying a Vcc to display will turn off the LEDs.

K-Map (ECE315)

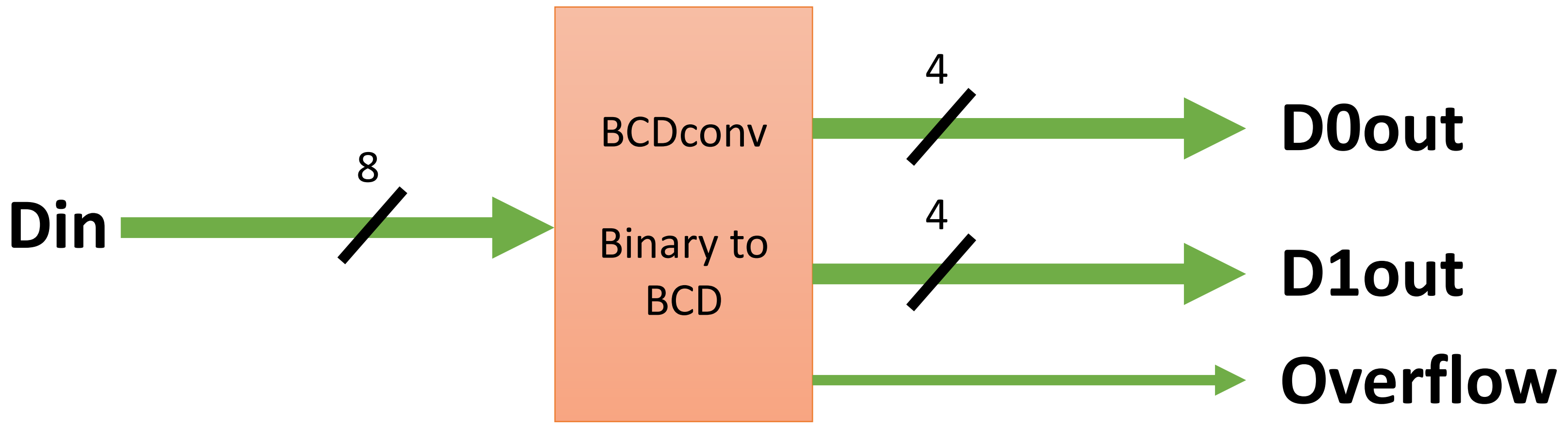
- We have 4 inputs and 7outputs.
- Implement in VHDL as equations

	Input X3..X0	Display D6…D0
0	0 0 0 0	1 0 0 0 0 0 0
1	0 0 0 1	
2	0 0 1 0	
3	0 0 1 1	
4	0 1 0 0	
5	0 1 0 1	
6	0 1 1 0	
7	0 1 1 1	
8	1 0 0 0	
9	1 0 0 1	
10	1 0 1 0	X X X X X X X X
11	1 0 1 1	X X X X X X X X
12	1 1 0 0	X X X X X X X X
13	1 1 0 1	X X X X X X X X
14	1 1 1 0	X X X X X X X X
15	1 1 1 1	X X X X X X X X

7-Segment Display Circuit - VHDL

- 1. Do K-maps and get minimal equations, implement with equations
- 2. Implement table directly in VHDL using:
  - Concurrent
    - When … else
    - With … select when
  - Process
    - If … else
    - Case

SimpleALU – Internals – BCDconv



Operation:
Converts a unsigned 8-bit binary number in the
Range: 0 to +99
to two decimal numbers with overflow

Shift Add 3 Algorithm

- Do 8 times (for 8 bit binary input).
  - 1. Shift left one to the binary input.
  - 2. Add 3 to BCD if it’s greater than 4.
  - 3. Go to 1.

Shift Add 3 Algorithm (ECE315)

- Input = HGFE DCBA
- Take the first 3 MSB of inputs and add to output.
    - Output = 0000 0HGF
- If 00HGF > 4
    - I H1 G1 F1 = 0 H G F + 0011.
- Else
    - I H1G1F1 = 0 H G F
- Shift left one.
    - Output = 000I H1 G1 F1 E
- If H1 G1 F1 E > 4
    - H2 G2 F2 E1 = H1 G1 F1 E + 0011.
- Else
    - H2 G2 F2 E1 = H1 G1 F1 E
- Shift left one.
    - Output = 00I H2 G2 F2 E1 D
- If G2 F2 E1 D > 4
    - G3 F3 E2 D1 = G2 F2 E1 D + 0011.
- Else
    - G3 F3 E2 D1 = G2 F2 E1 D
- Shift left one.
    - Output = 0I H2 G3 F3 E2 D1 C
- If 0I H2 G3 > 4
    - J I1 H3 G4 = 0 I H2 G3 + 0011.
- Else
    - J I1 H3 G4 = 0 I H2 G3
- If F3 E2 D1 C > 4
    - F4 E3 D2 C1 = F3 E2 D1 C + 0011.
- Else
    - F4 E3 D2 C1 = F3 E2 D1 C
- Shift left one.
    - Output = I1 H3 G4 F4 E3 D2 C1 B
- If I1 H3 G4 F4 > 4
    - I2 H4 G5 F5 = I1 H3 G4 F4 + 0011.
- Else
    - I2 H4 G5 F5 = I1 H3 G4 F4
- If E3 D2 C1 B > 4
    - E4 D3 C2 B1 = E3 D2 C1 B+ 0011.
- Else
    - E4 D3 C2 B1 = E3 D2 C1 B
- Shift left one.
    - Output= **H4 G5 F5 E4 D2 C2 B1 A**

Add 3 Circuit (ECE315)

- If Input > 4
- Output = Input +3;
- Else
- Output = Input;
- End

Input = 0101 (which is greater than 4)
Output = 0101 + 0011 = 1000

Input X3X2X1X0	Output P3P2P1P0
0000	0000
0001	0001
0010	0010
0011	0011
0100	0100
0101	1000
0110	1001
0111	
1000	
1001	
1010	xxxx
.	.
.	.
1111	xxxx

When $input \leq 4.$

Fill these outputs

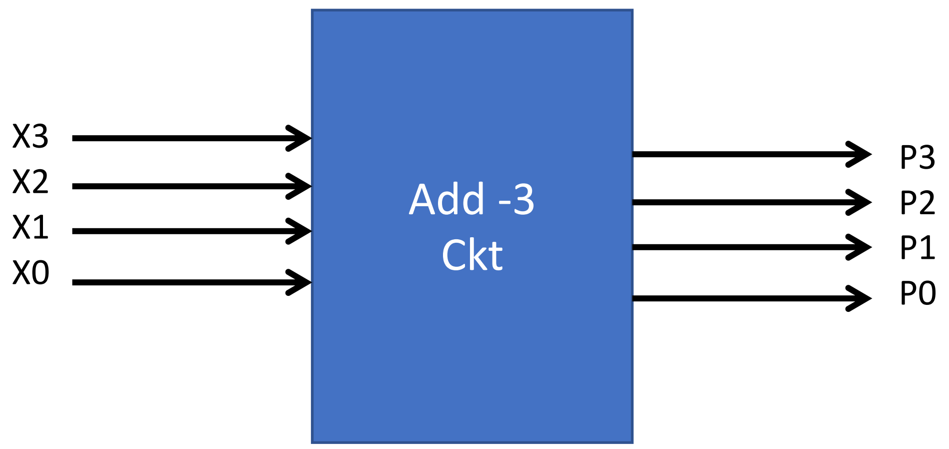
When input > 9, don't care.

Add-3 Circuit

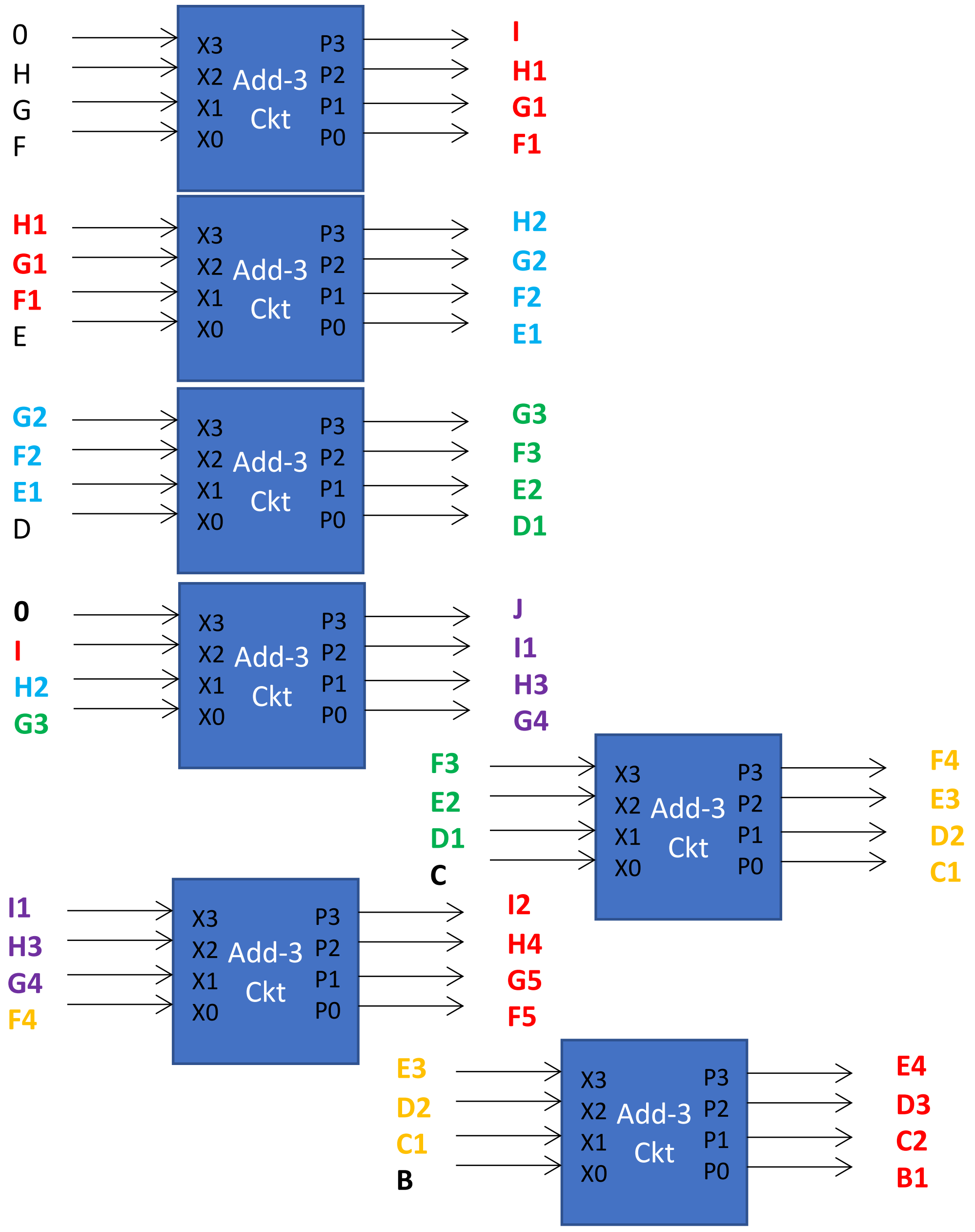
- 1. Do K-maps and get mimimal equations, implement with equations
- 2. Implement table directly in VHDL using:
  - Concurrent
    - When … else
    - With … select when
  - Process
    - If … else
    - Case
- 3. Implement as arithmetic
  - variable bcddigit : std_logic_vector(3 downto 0);
  - bcddigit := std_logic_vector(unsigned(bcddigit) + “0011”);
    - use ieee.numeric_std.all;

Add 3 Circuit (ECE315)

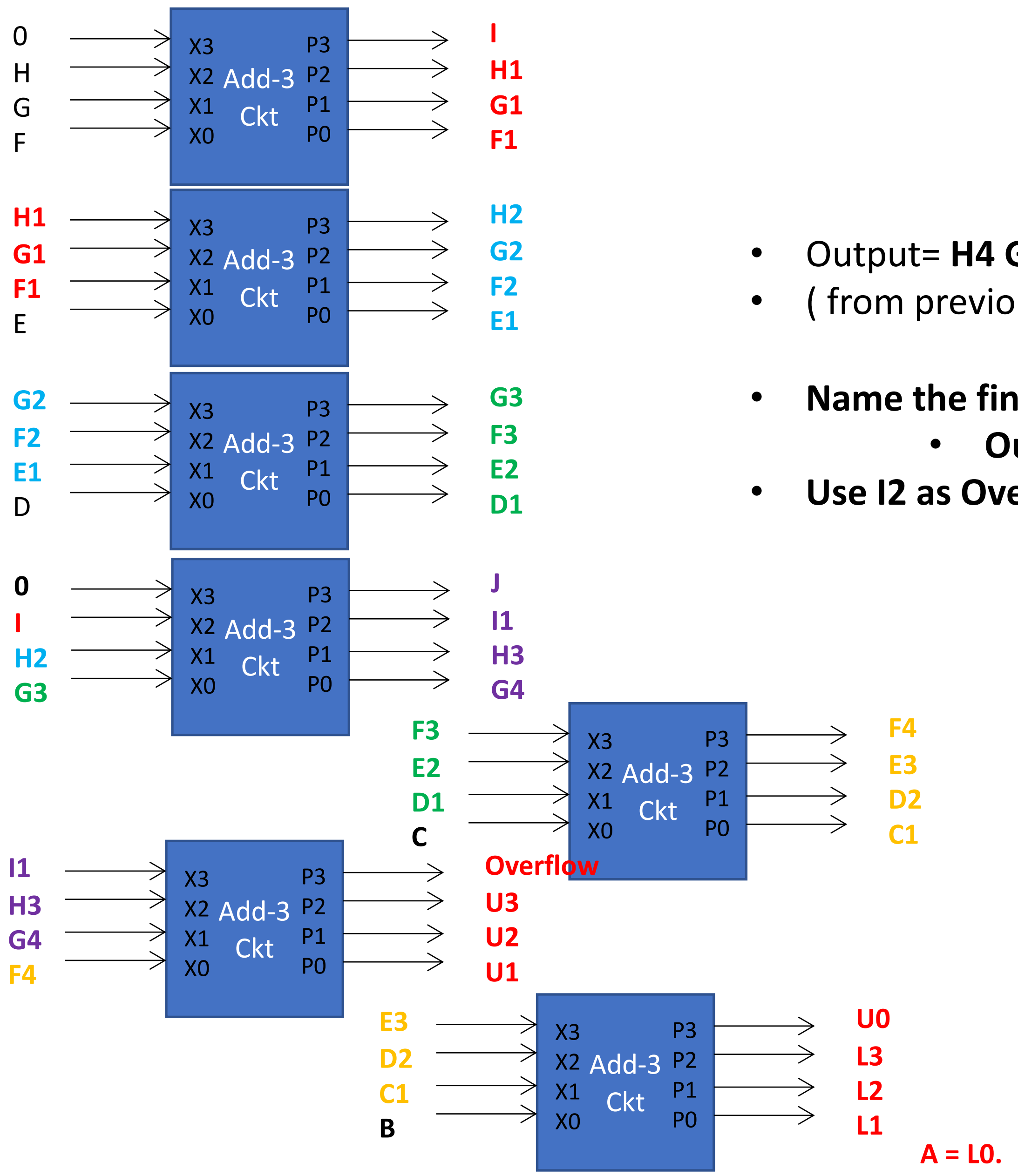
- Complete the Truth Table from prev slide
- Find the Boolean expression for P3, P2, P1 and P0.
- Implement in VHDL



- If 0HGF > 4
    - I H1 G1 F1 = 0 H G F + 0011.
- Else
    - I H1G1F1 = 0 H G F
- Shift left one.
    - Output = 000I H1 G1 F1 E
- If H1 G1 F1 E > 4
    - H2 G2 F2 E1 = H1 G1 F1 E + 0011.
- Else
    - H2 G2 F2 E1 = H1 G1 F1 E
- Shift left one.
    - Output = 00I H2 G2 F2 E1 D
- If G2 F2 E1 D > 4
    - G3 F3 E2 D1 = G2 F2 E1 D + 0011.
- Else
    - G3 F3 E2 D1 = G2 F2 E1 D
- Shift left one.
    - Output = 0I H2 G3 F3 E2 D1 C
- If 0I H2 G3 > 4
    - J I1 H3 G4 = 0 I H2 G3 + 0011.
- Else
    - J I1 H3 G4 = 0 I H2 G3
- If F3 E2 D1 C > 4
    - F4 E3 D2 C1 = F3 E2 D1 C + 0011.
- Else
    - F4 E3 D2 C1 = F3 E2 D1 C
- Shift left one.
    - Output = I1 H3 G4 F4 E3 D2 C1 B
- If I1 H3 G4 F4 > 4
    - I2 H4 G5 F5 = I1 H3 G4 F4 + 0011.
- Else
    - I2 H4 G5 F5 = I1 H3 G4 F4
- If E3 D2 C1 B > 4
    - E4 D3 C2 B1 = E3 D2 C1 B+ 0011.
- Else
    - E4 D3 C2 B1 = E3 D2 C1 B
- Shift left one.
    - Output= **H4 G5 F5 E4 D2 C2 B1 A**
- **Let's the final output of BCD = U3U2U1U0 L3L2L1L0**



BCD



- Output= **H4 G5 F5 E4 D3 C2 B1 A**
- (from previous slide)

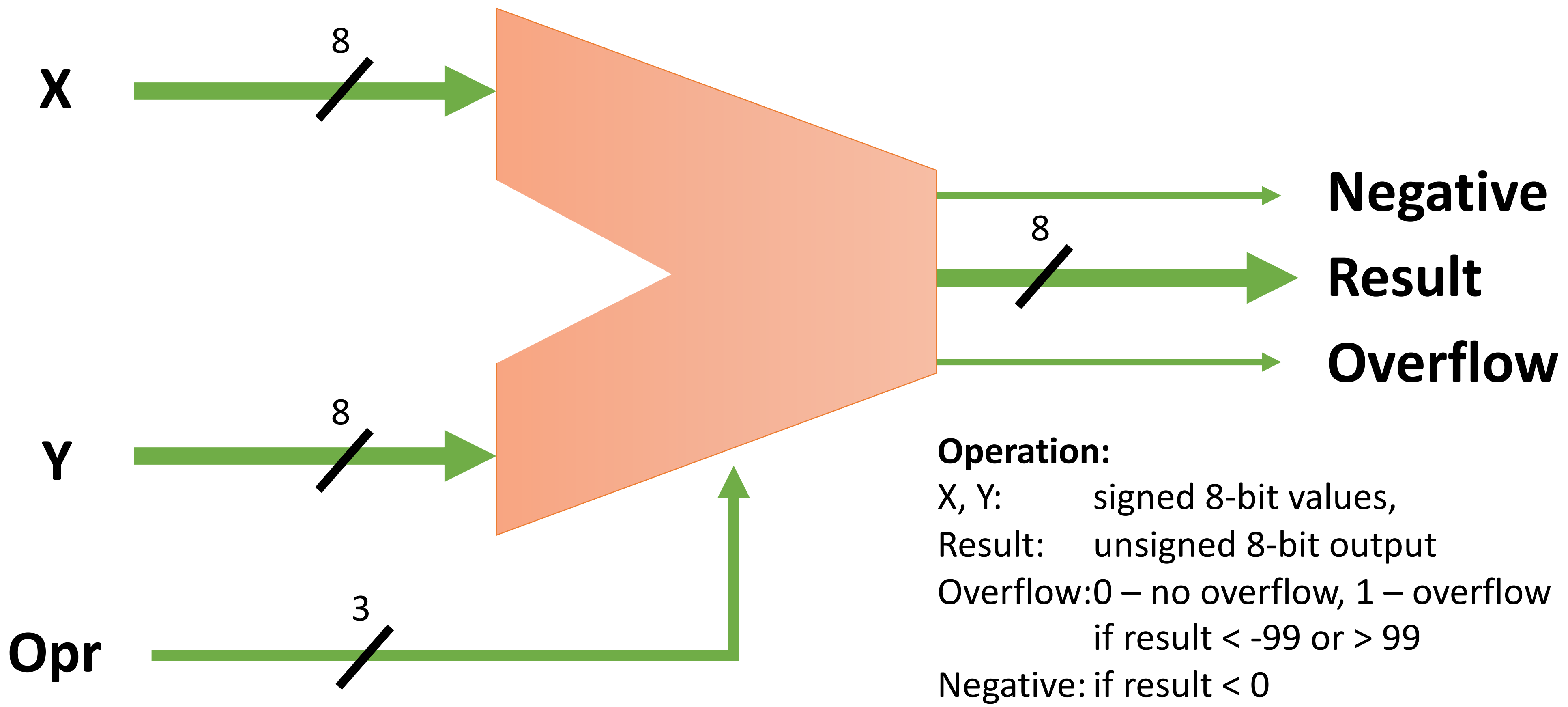
- **Name the final output of BCD as**
  - **Output = U3U2U1U0 L3L2L1L0**
- **Use I2 as Overflow.**

Shift-Add-3 Algorithm - VHDL

- Implement as a process using:
  - for loop
  - and if .. then statements

- Shift can be implemented as a concatenate
  - variable x : std_logic_vector(3 downto 0);
  - variable y : std_logic_vector(3 downto 0);
  - To left shift high-bit from x into y:
  - y := y(2 downto 0) & x(3);

SimpleALU – Internals – addopr



Operation:

X, Y:	signed 8-bit values,
Result:	unsigned 8-bit output
Overflow:	0 – no overflow, 1 – overflow if result < -99 or > 99
Negative:	if result < 0

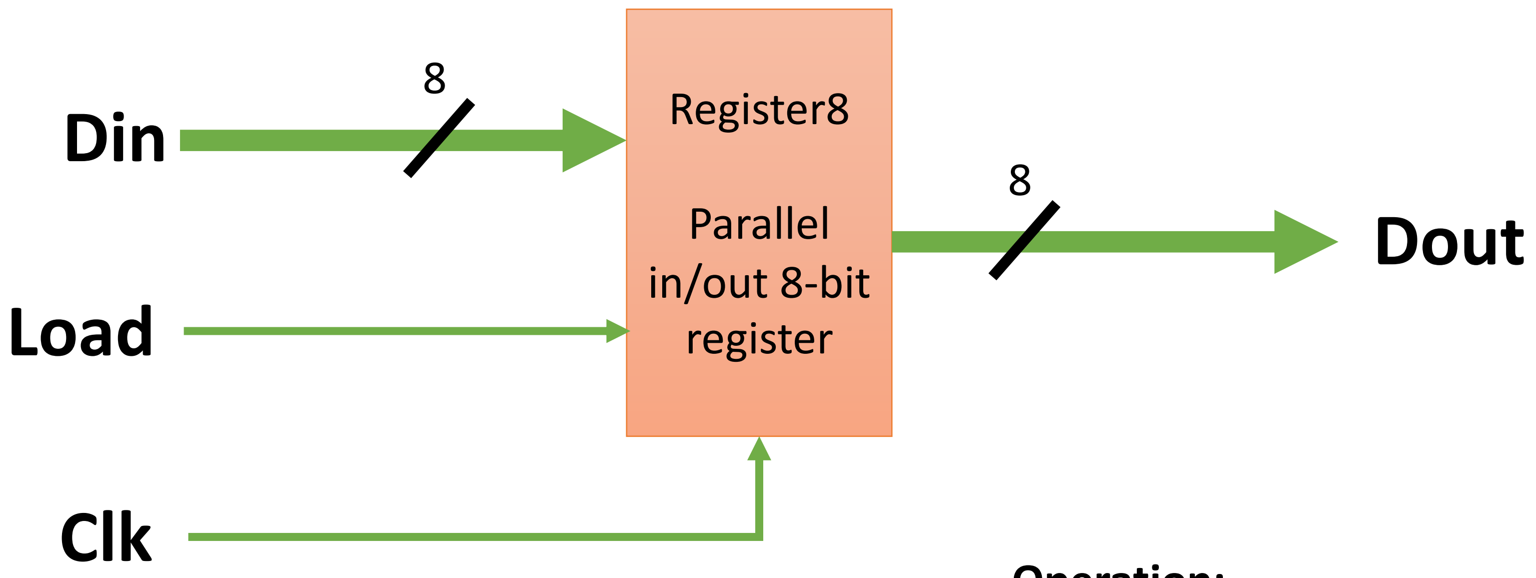
Data Types

- bit {‘0’,’1’}
- bit_vector
- ieee.std_logic:
  - std_logic {‘0’, ‘1’, ‘Z’, ‘X’}
  - std_logic_vector
    - Range Declaration (e.g. 8 bits)
    - E.g. 7 downto 0, 8 downto 1
    - E.g. 0 to 7, 1 to 8

Data Types

- ieee.logic_signed:
  - Signed integers
  - Operators (+, -, x, /)
    - Using signed arithmetic
- ieee.numeric_std:
  - Logic and arithmetic
  - Conversion functions

SimpleALU – Internals – register8



Operation:

Load = 0:

Load Din to Internal on Clk falling edge

Load internal to Dout on Clk rising edge

Load = 1:

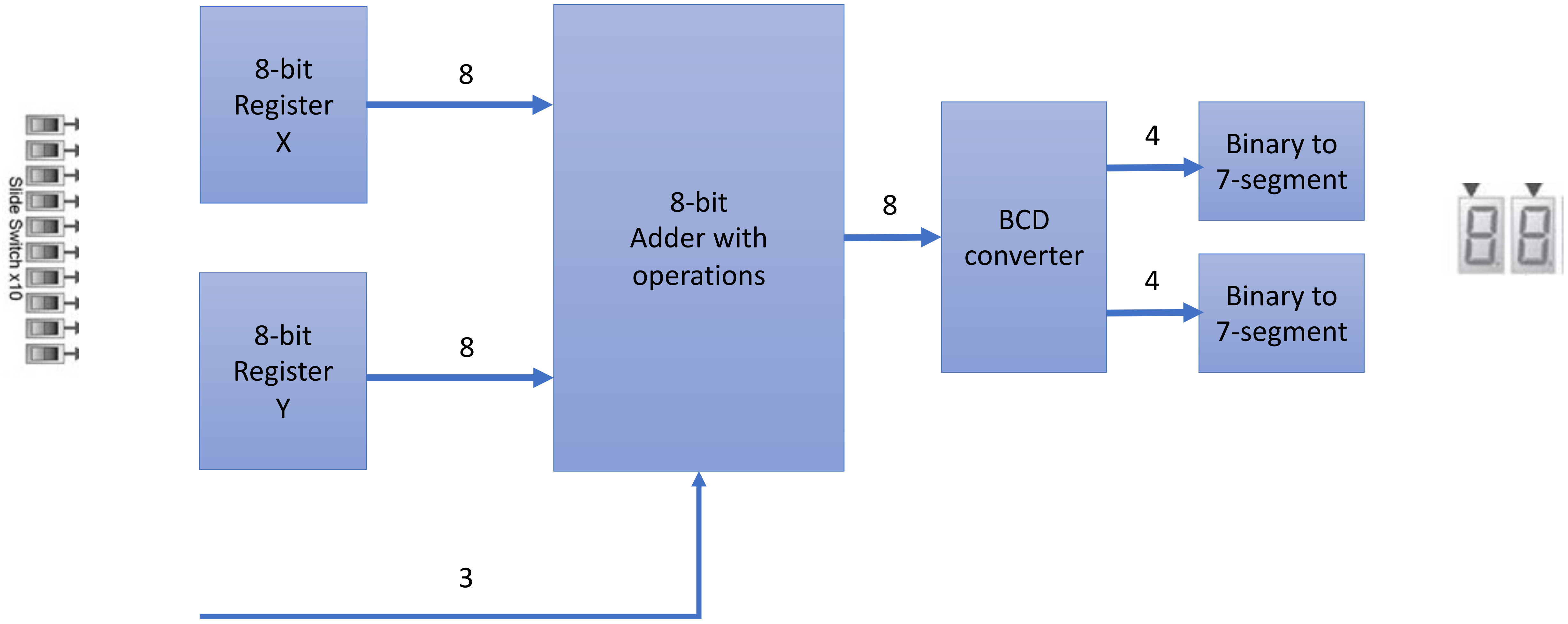
Hold Dout, ignore Din

Adjustments

- Load is active LOW
- Clock is active LOW

Din	Load	*Store*	Clk	Dout	Note
X	1	Store	X	Dout	Hold last value
X	0	Din	↓	Dout	Din to internal storage
X	0	Store	↑	Store	Internal storage to Dout

Structural Design – from 3B – Ver 1.



To Demo – 3b

- Implement all components as VHDL
- Create a top-level design with these components
- Simulate in Modelsim (for report)
- Demo on DE1 board
  - Show all operations for at least 3 values of X and Y
  - Show the overflow
  - Show negatives