Integrated Circuits Design by FPGA

م.م. أحمد مؤيد عبدالحسين
جامعة الفرات الأوسط التقنية / الكلية التقنية الهندسية / نجف

Lecture 16

System Design

Neural Networks

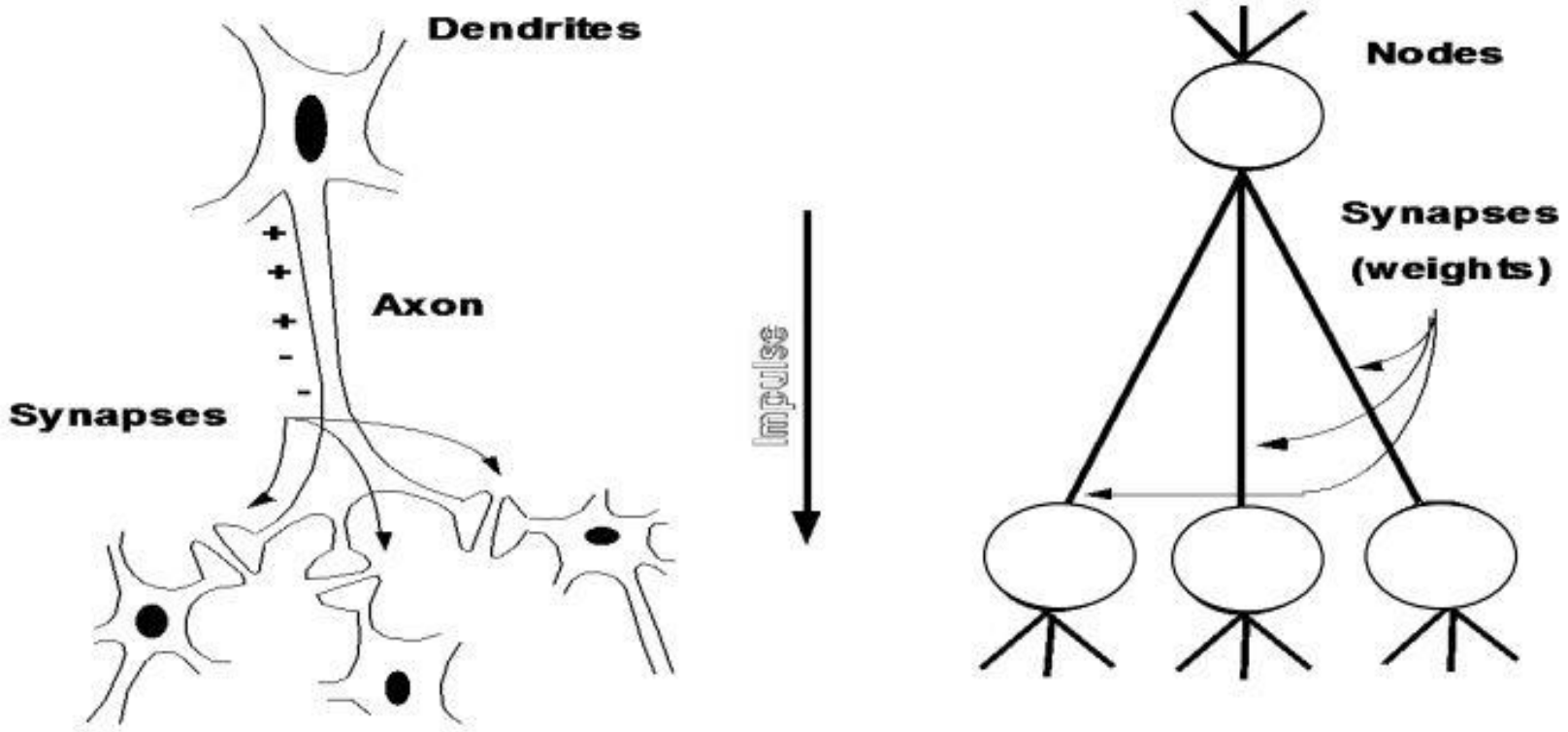
Objectives of this Lecture

- To define **Neural Networks**
- To implement **Neural Networks** in an example design.

Contents of this Lecture

- Introduction
- **Neural Networks** Example

Introduction



Introduction

- **Neural Networks** (**NN**) are highly parallel, highly interconnected systems. Such characteristics make their implementation very challenging, and also very costly, due to the large amount of hardware required.

- A feedforward **NN** is shown in figure 12.12(a). In this example, the circuit has three layers, with three 3-input neurons in each layer. Internal details of each layer are depicted in figure 12.12(b).

Introduction

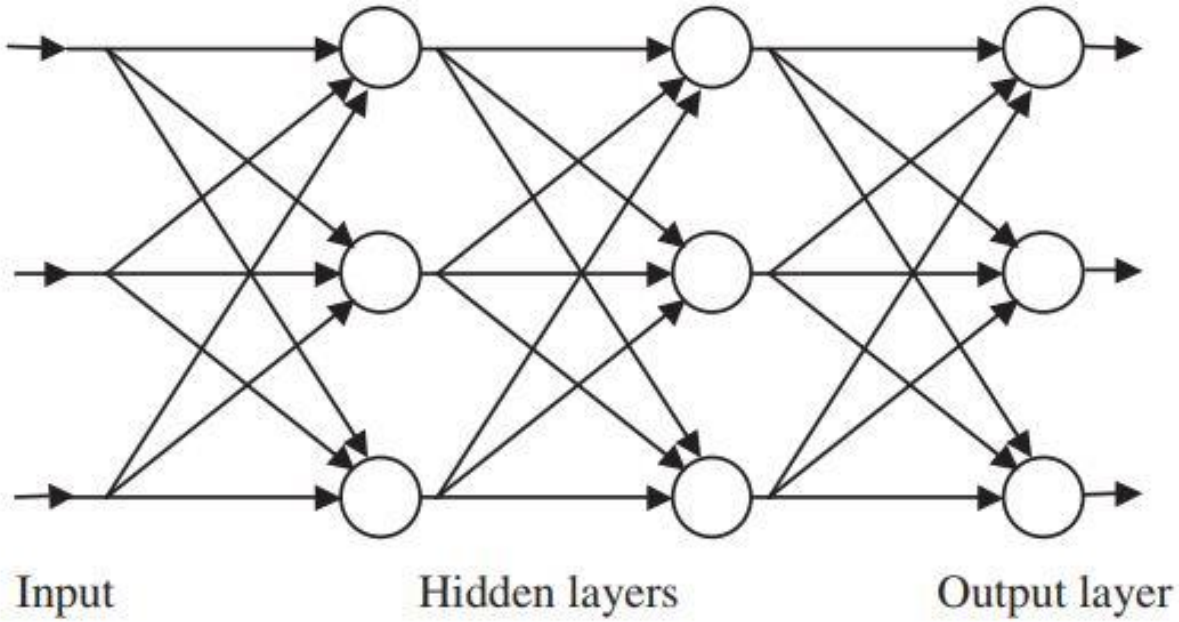


Figure : 12.12 a

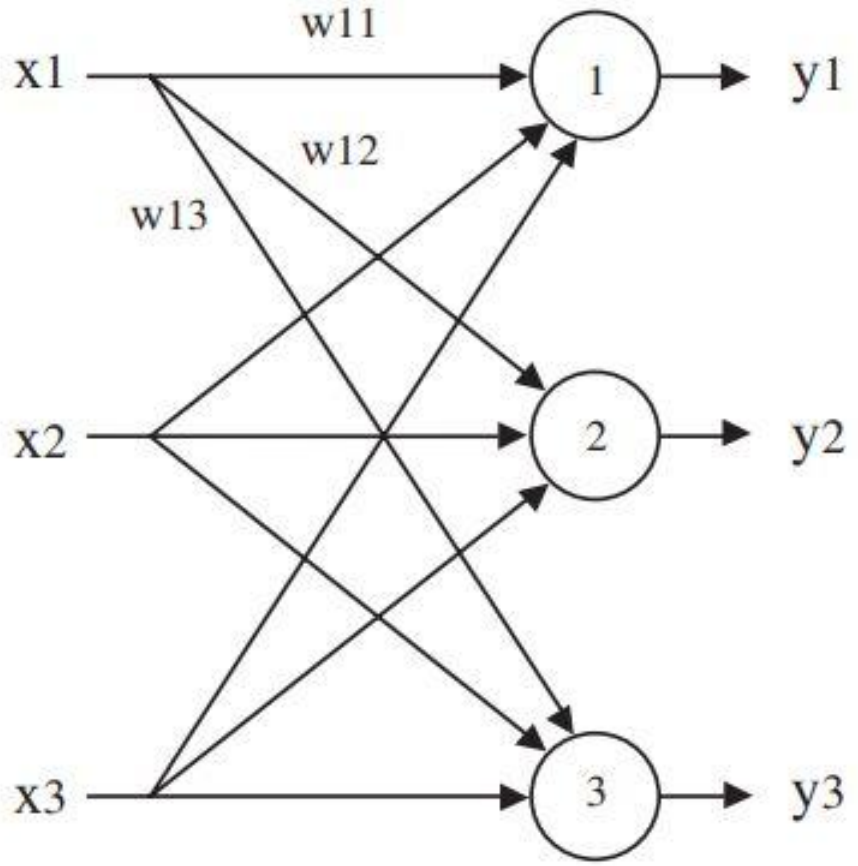
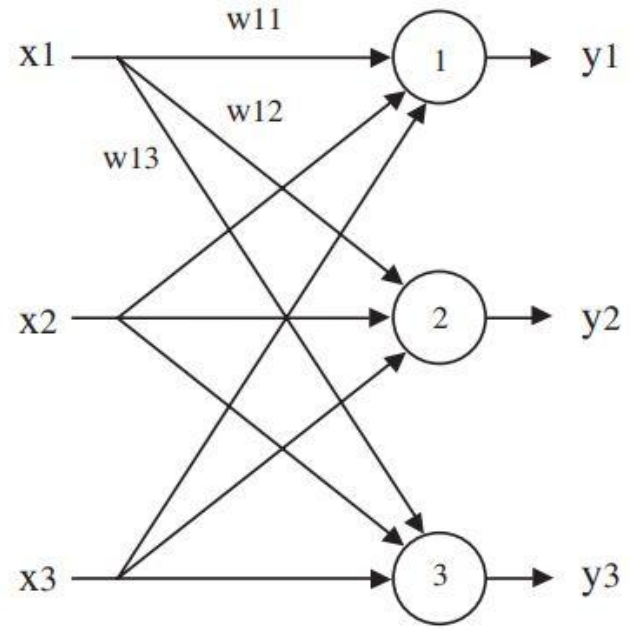


Figure : 12.12 b

Introduction

- $\mathbf{x_i}$ represents the **i**th input.
- $\mathbf{w_{ij}}$ is the weight between input **i** and neuron **j**,
- $\mathbf{y_j}$ is the **j**th output.
- Therefore, y1 = f(x1.w11 + x2.w21 + x3.w31),
- y2 = f(x1.w12 + x2.w22 + x3.w32),
- y3 = f(x1.w13 + x2.w23 + x3.w33),
- where f( ) is the **activation function** (linear threshold, sigmoid, etc.).



Introduction

Watch This Video !

Introduction

- A "ring" architecture for the **NN** of figure 12.12 is presented in figure 12.13, which implements **one layer** of the **NN**.

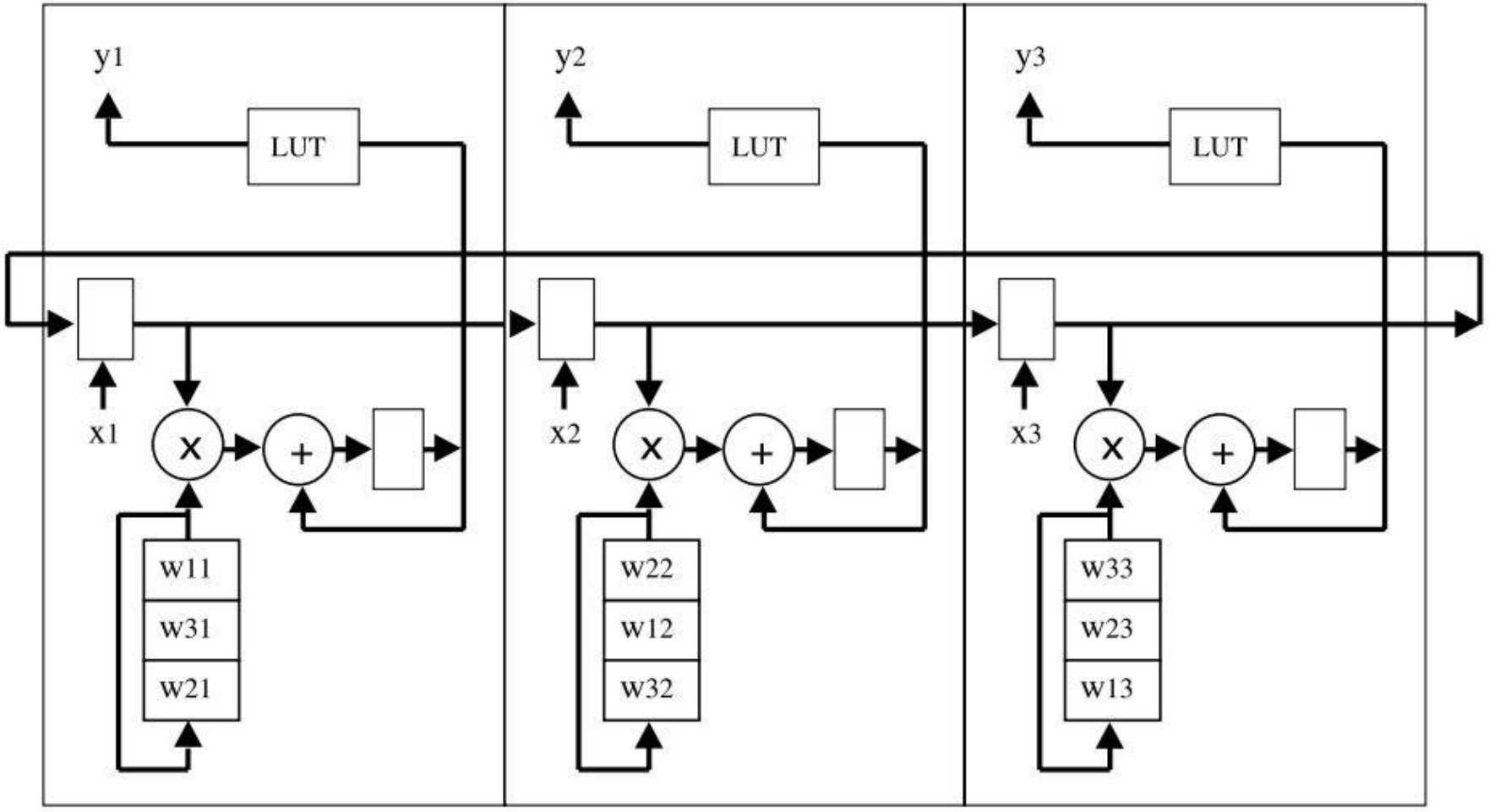


Figure 12.13

Introduction

- Each box represents one neuron.
- As shown, there are several circular shift registers, one for each neuron (vertical shifters) plus one for the whole set (horizontal shifter). The vertical shifters hold the weights, while the horizontal one holds the inputs.
- Notice that the relative position of the weights in their respective registers must match that of the input values.

  - At the output of a vertical shifter there is a MAC circuit, which accumulates the product between the weights and the inputs.

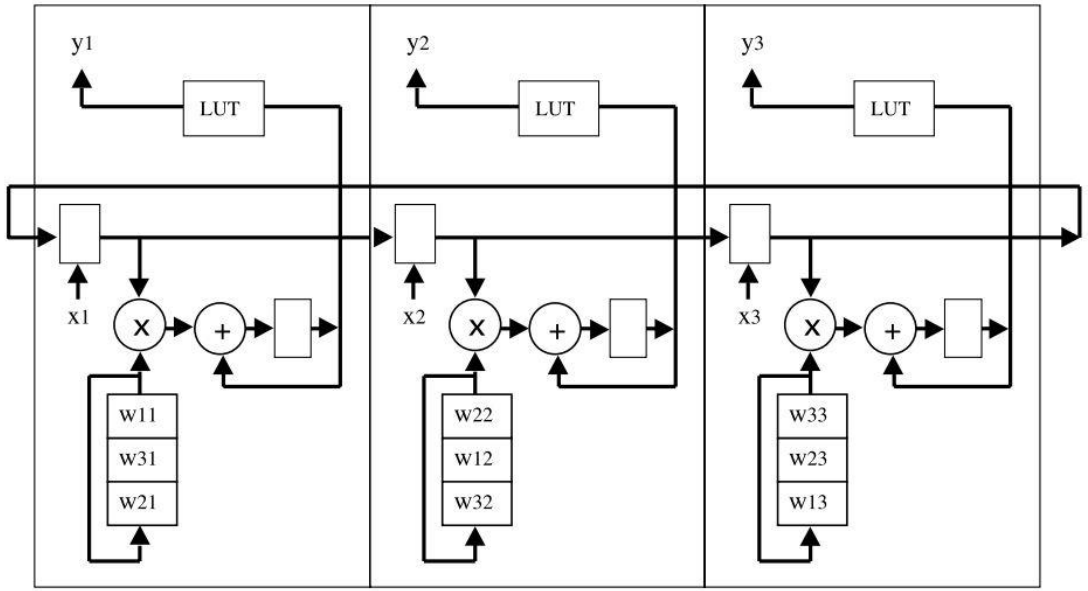


Figure 12.13

Introduction

- All shifters use the same clock signal. Therefore, after one complete circulation, the following values will be available at the output of the MAC circuits:

  $x1.w11 + x2.w21 + x3.w31$,

  $x1.w12 + x2.w22 + x3.w32$,

  $x1.w13 + x2.w23 + x3.w33$

- These values are then applied to a LUT (lookup table), which implements the activation **f**unction (sigmoid, for example), thus producing the actual outputs, yi, of the **NN.**

Introduction

- In this kind of circuit, truncation must be considered. Say that the inputs and weights are **16 bits** long. Then at the output of the MAC cells **32-bit** numbers would be the natural choice. However, since the actual outputs (after the LUT) might be connected to another layer of neurons, **truncation to 16 bits** is required. This can be done in the LUT or in the MAC circuit.

- Another approach is presented in figure 12.14, which is appropriate for general purpose **NN**s (that is, with programmable weights). It employs only one input to load all weights.

Introduction

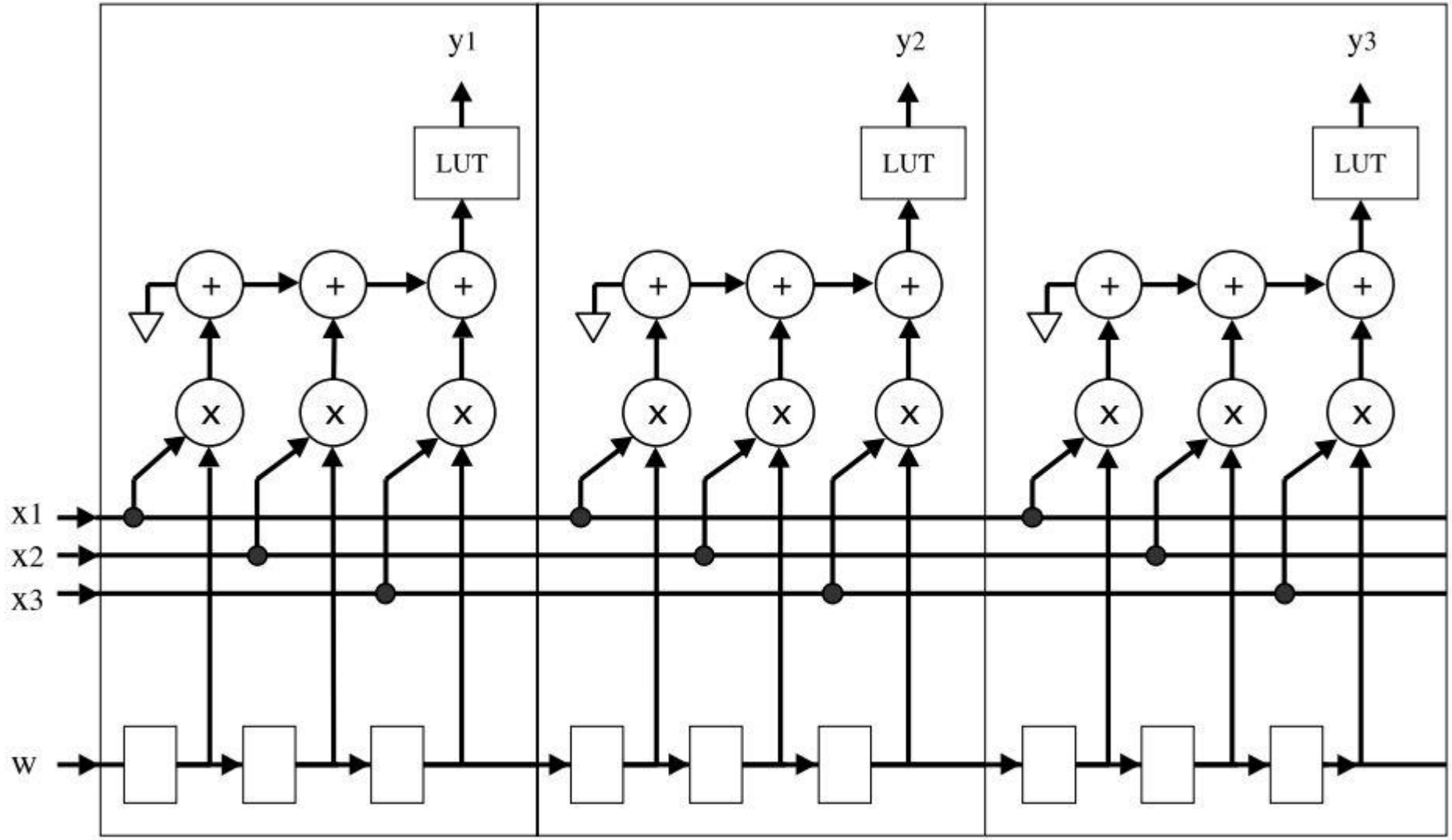


Figure 12.14
NN implementation with only one input for the weights.

Introduction

- In figure 12.14, the weights are shifted in sequentially until each register is loaded with its respective weight.
- The weights are then multiplied by the inputs and accumulated to produce the desired outputs.

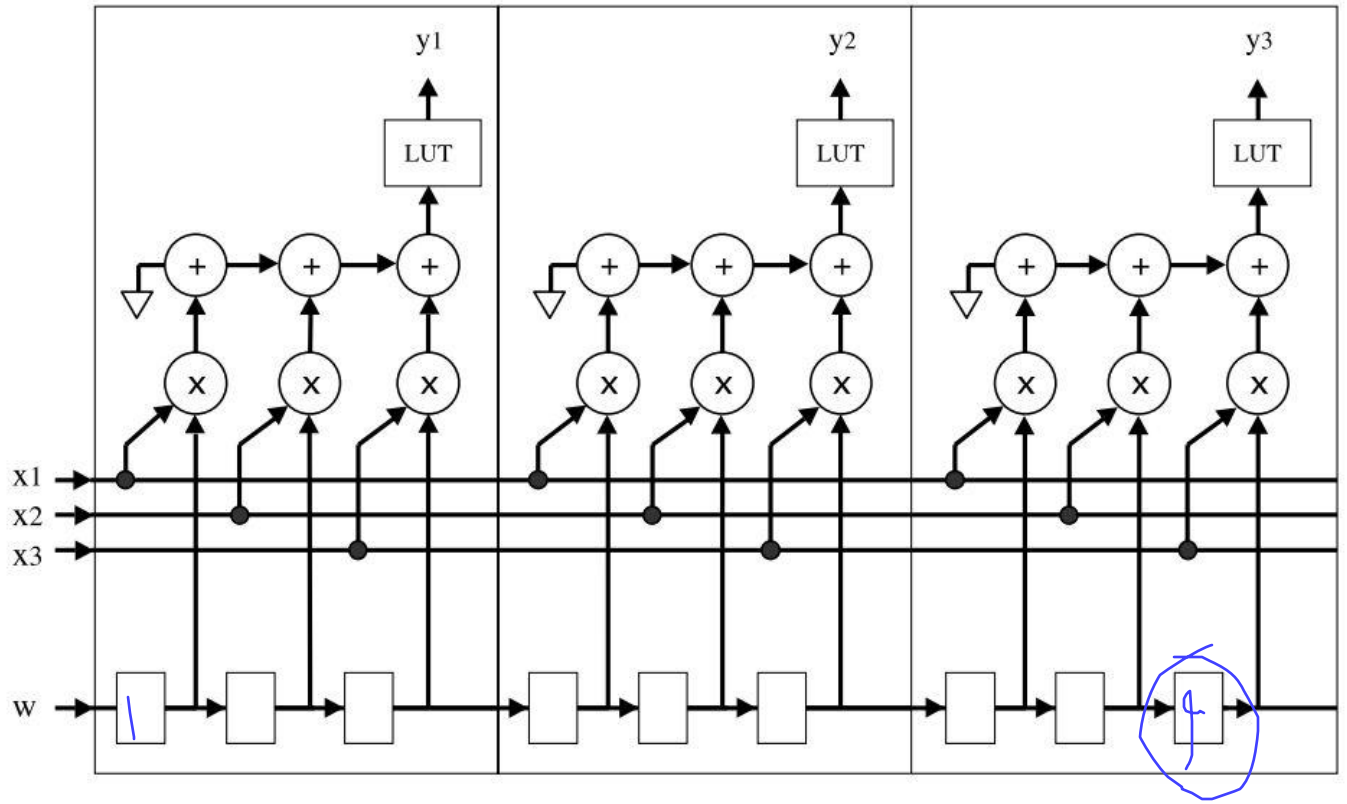


Figure 12.14
NN implementation with only one input for the weights.

Introduction

- **NN** example is in the next slides, which implementing the architecture of figure 12.14. However, the solution does not contain LUT(**activation function**).

- The example on the next slides has the advantage of being simple, easily understandable, and self-contained in the main code. Its only limitation is that the inputs (x) and outputs (y) are specified rather than generic input and output, thus making it inappropriate for large NNs. Everything else is generic.

NN Example

```
1  ----------------------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE ieee.std_logic_arith.all;  -- package needed for SIGNED
5  ----------------------------------------------------------
6  ENTITY nn IS
7  GENERIC ( n: INTEGER := 3;  -- # of neurons
8            m: INTEGER := 3; -- # of inputs or weights per neuron
9            b: INTEGER := 4); -- # of bits per input or weight
10 PORT ( x1: IN SIGNED(b-1 DOWNTO 0);
11        x2: IN SIGNED(b-1 DOWNTO 0);
12        x3: IN SIGNED(b-1 DOWNTO 0);
13        w: IN SIGNED(b-1 DOWNTO 0);
14        clk: IN STD_LOGIC;
15        test: OUT SIGNED(b-1 DOWNTO 0);  -- register test output
16        y1: OUT SIGNED(2*b-1 DOWNTO 0);
17        y2: OUT SIGNED(2*b-1 DOWNTO 0);
18        y3: OUT SIGNED(2*b-1 DOWNTO 0));
19 END nn;
20 ----------------------------------------------------------
```

NN Example

```
21 ARCHITECTURE neural OF nn IS
22    TYPE weights IS ARRAY (1 TO n*m) OF SIGNED(b-1 DOWNTO 0);
23    TYPE inputs IS ARRAY (1 TO m) OF SIGNED(b-1 DOWNTO 0);
24    TYPE outputs IS ARRAY (1 TO m) OF SIGNED(2*b-1 DOWNTO 0);
25 BEGIN
26    PROCESS (clk, w, x1, x2, x3)
27       VARIABLE weight: weights:= ("0000","0000","0000","0000","0000","0000","0000","0000","0000");
28       VARIABLE input: inputs;
29       VARIABLE output: outputs;
30       VARIABLE prod, acc: SIGNED(2*b-1 DOWNTO 0);
31       VARIABLE sign: STD_LOGIC;
32    BEGIN
33       ----- shift register inference: -------------
34       IF (clk'EVENT AND clk='1') THEN
35          weight := w & weight(1 TO n*m-1);
36       END IF;
37       --------- initialization: ------------------
38       input(1) := x1;
39       input(2) := x2;
40       input(3) := x3;
```

NN Example

```
41        ------ multiply-accumulate: -----------------
42        L1: FOR i IN 1 TO n LOOP
43            acc := (OTHERS => '0');
44           L2: FOR j IN 1 TO m LOOP
45              prod := input(j)*weigth(m*(i-1)+j);
46              sign := acc(acc'LEFT);
47              acc := acc + prod;
48              ---- overflow check: ----------------
49              IF (sign=prod(prod'left)) AND
50                    (acc(acc'left) /= sign) THEN
51                 acc := (acc'LEFT => sign, OTHERS => NOT sign);
52              END IF;
53           END LOOP L2;
54           output(i) := acc;
55        END LOOP L1;
56        --------- outputs: -------------------------
57        test <= weight(n*m);
58        y1 <= output(1);
59        y2 <= output(2);
60        y3 <= output(3);
61     END PROCESS;
62  END neural;
```

L1 : For i IN 1 to 3 Loop

i=1

L2 : For j IN 1 to 3 Loop

j=1

prod := input(j) * weight(m*(i-1)+j)

↳ prod := input(1) * weight(1)

= 3 * 1

= 3

j=2 prod := input(2) * weight(2)

= 4 * 2

= 8

...

NN Example

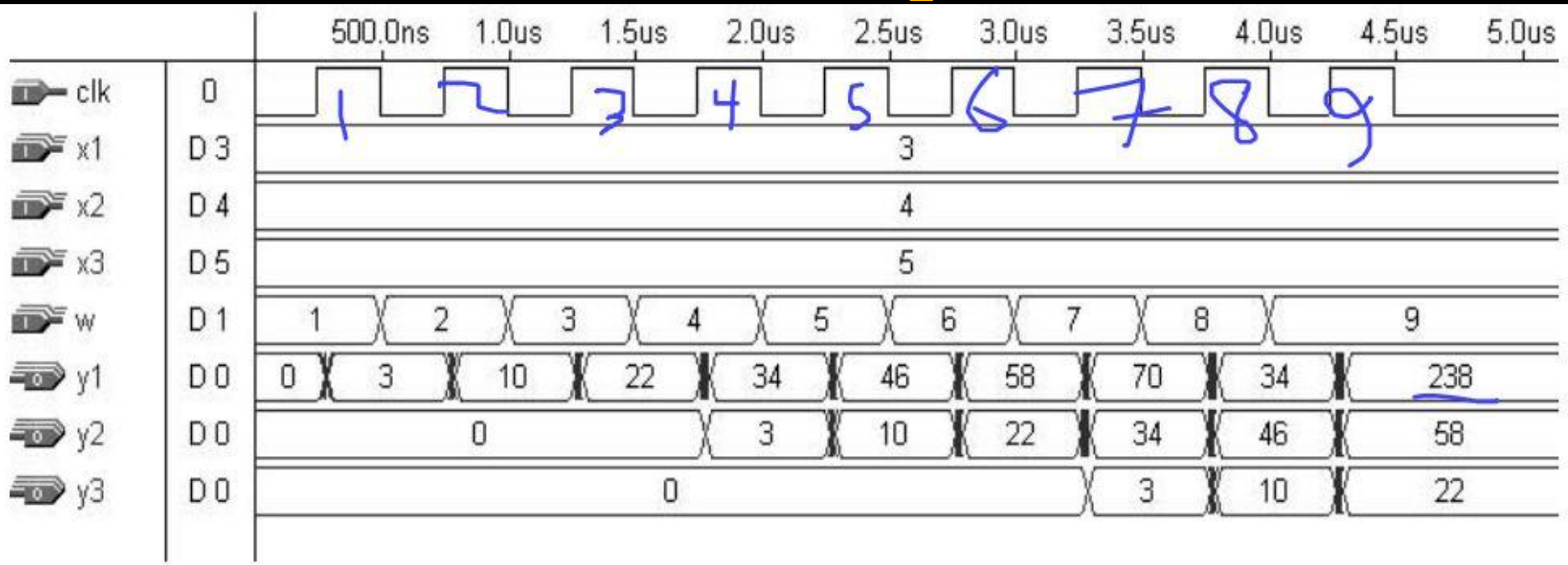


Figure 12.15
Simulation results of NN implemented in solution 1.

y1 = x1.w1 + x2.w2 + x3.w3 = (3)(−7) + (4)(−8) + (5)(7) = −18 (represented as 256 − 18 = 238); y2 = x1.w4 + x2.w5 + x3.w6 = (3)(6) + (4)(5) + (5)(4) = 58; and y3 = x1.w7 + x2.w8 + x3.w9 = (3)(3) + (4)(2) + (5)(1) = 22. These values (238, 58, and 22) can be seen at the right end of figure 12.15.

Assignments

The assignments will be attached to your class room

End of lecture 16
Any Questions ?