NYCU-EE IC LAB - Fall 2021

Lab04 Exercise

Design: Artificial Neural Network

Data Preparation

1. Extract test data from TA's directory:

   % tar xvf ~iclabta01/Lab04.tar

2. The extracted LAB directory contains:

   a. **00_TESTBED**

   b. **01_RTL**

   c. **02_SYN**

   d. **03_GATE**

Design Description

Artificial Neural Networks are computing systems vaguely inspired by the biological neural networks that constitute animal brains. An ANN is based on a collection of connected units or nodes called *artificial neurons*, which loosely model the neurons in a biological brain. The connections between artificial neurons are called edges, and they contain *weights* that can be adjust during learning proceeds. Typically, artificial neurons are aggregated into layers, and different layers may perform different kinds of transformations on their inputs.

The main usage of ANN is to learn the feature from large amount of data, and perform some specific tasks, like classification or regression problems. There are many methods have been proposed for training an ANN (or making an ANN to learn), the most popular one is *back-propagation*. Following is the algorithm description of back-propagation for an ANN.

- Description

  The back-propagation algorithm contains three stages, which are *forward stage*, *backward stage*, and *update stage*. The algorithm will execute each stage sequentially. First, a data will be given to our ANN, then the forward stage will be executed, then followed by backward stage, finally is the update stage. When update stage has been done, we say that we have finished **1 iteration**. Then, we will put next data in our ANN and start to execute next iteration. Until we use up all data, we say that we have finished **1 epoch**. Then, we will continue above process for several epochs to "train" our ANN. In this lab, we are going to solve a **regression problem**, which means the **dimension of network output is 1**. The activation function, which is used to introduce non-linearity to the network, is chosen to be **ReLU function**, which show in Eq 1. The **learning rate is adjusted based on the number of epochs** in this lab. The initial learning rate is 0.000001=32'h 358637BD (IEEE-754), and each 4 epochs will be reduced to half of the original. After sending new weight to the circuit, learning rate will reset to 0.000001.

$$g(x) = \max(0, x)$$

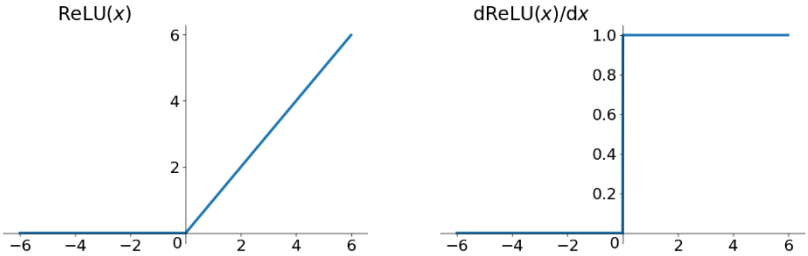


Fig 1. The ReLU activation function

- General Formula
  - Notation

    L : the number of layers in a specific network.

    l : the index of layer, range from 1 to L.

    η : the learning rate

    W_{ij}^{l}: the weight of i^{th} row and j^{th} column in layer l.

    s_j^l : j^{th} input of the neurons in layer l.

    h_i^l : the output of i^{th} neuron **before** activation function in layer l.

    y_i^l : the output of i^{th} neuron **after** activation function in layer l.

    t_i^l : the i^{th} elements of target.

    $g(\cdot)$: ReLU activation function.

    $g(\cdot)'$: the derivative of activation function.

    δ_j^l : the j^{th} error term in layer l.

  - Forward Stage

    (1) For last layer

$$y_i^L = \sum_j w_{ij}^L \cdot s_j^L = h_i^L$$

    (2) For other layers

$$y_i^l = g\left(\sum_j w_{ij}^l \cdot s_j^l\right) = g(h_i^l)$$

- Backward Stage

  (1) For last layer:

$$\delta_j^L = y_j^L - t_j \quad \text{or} \quad \delta_j^L = h_j^L - t_j$$

  Since output layer does not use activation function.

  (2) For other layers:

$$\delta_j^l = g(h_j^l)' \cdot \sum_i w_{ij}^{l+1} \delta_i^{l+1}$$

- Update Stage

$$w_{ij} = w_{ij} - \eta \delta_i^l s_j^l$$

- Example Formula for the architecture in this lab
  - **Forward (first layer)**

    Input dim: 4*1 Weight dim: 3*4 Output dim 3*1

$$y_i^1 = g\left(\sum_{j=0}^{j=3} w_{ij}^1 \cdot s_j^1\right), \quad i = 0 \sim 2$$

  - **Forward (second layer)**

    Input dim: 3*1 Weight dim: 1*3 Output dim: 1*1

$$y_0^2 = \sum_{j=0}^{j=2} w_{0j}^2 \cdot s_j^2$$

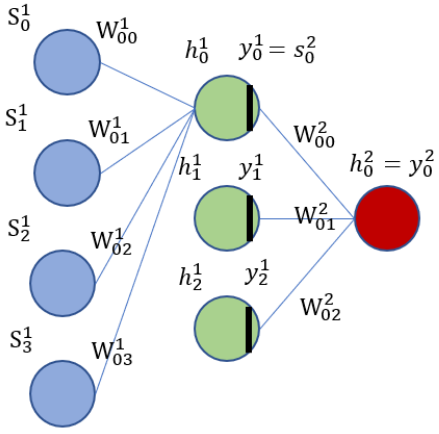


Fig 2. Example of forward stage

- **backward (second layer)**

$$\delta_0^2 = y_0^2 - t_0$$

- **backward (first layer)**

$$\delta_i^1 = g(\mathrm{h}_i^1)' \cdot w_{0i}^2 \delta_0^2, \quad i = 0{\sim}2$$

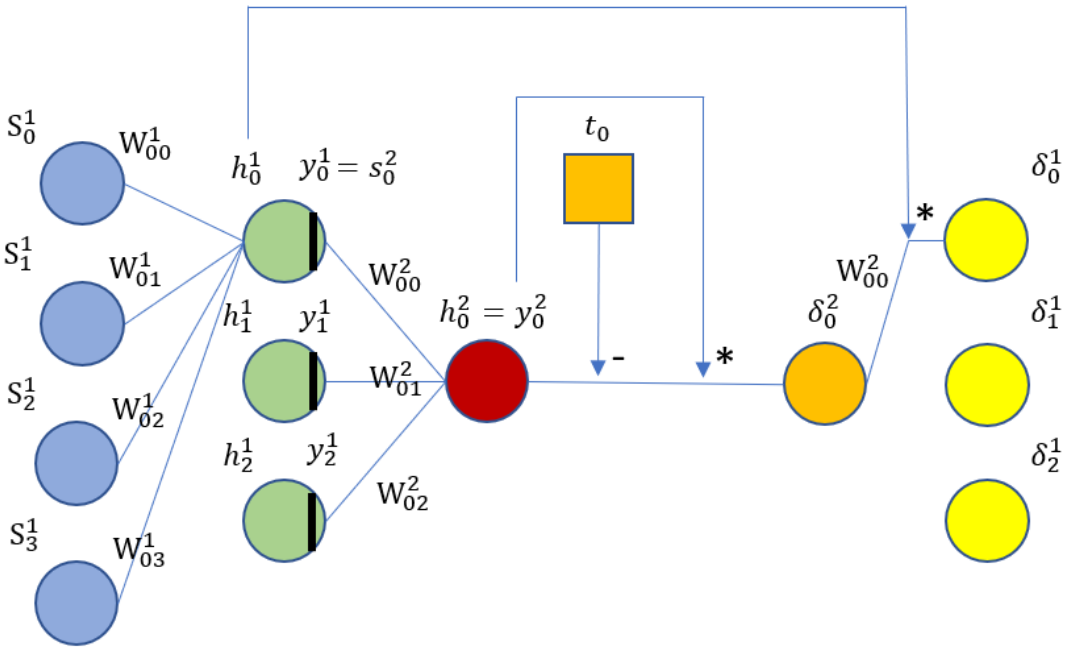


Fig 3. Example of backward stage

- **update (second layers)**

$$w_{0j}^2 = w_{0j}^2 - \eta \delta_0^2 s_j^2, \quad j = 0{\sim}2$$

- **update (first layers)**

$$w_{ij}^1 = w_{ij}^1 - \eta \delta_i^1 s_j^1, \quad i = 0{\sim}2, \quad j = 0{\sim}3$$

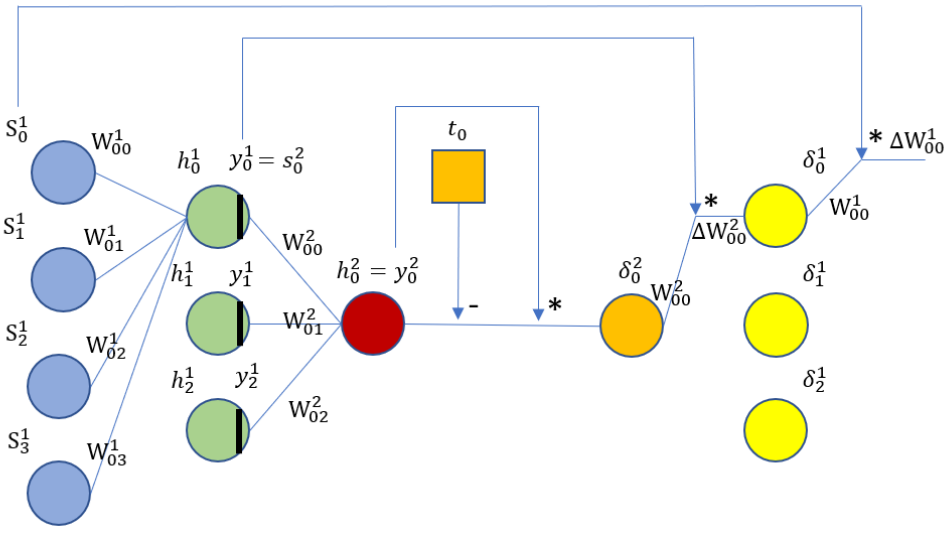


Fig 4. Example of update stage

In this lab, you need to build an ANN accelerator. The ANN will contain **three layers**. First layer is the input layer, and the **dimension is 4** (the dimension of the input data is 4). Second layer is a hidden layer, and the **dimension is 3**. Last layer is the output layer, the **dimension is 1**.

We will send **15 weights** of the network at beginning (12 for W^1 and 3 for W^2). Then, we will send **4 data points** to your ANN. After receiving the data, you need to perform back-propagation algorithm, and update the current weights. Once you finish the update, you need to output the result for the forward stage you just get. **Next data will be given to your ANN after you output the result**, the operation flow shows in Fig 5. for each data.

There are total **100 data**, we will give one data to your ANN each time, and you need to perform back-propagation algorithm including three stages for the data. For a set of weights, we will test your ANN for **25 epochs.** This means you need to perform 2500 times back-propagation algorithm for updating this set of weights. After that, we will send new weights to your ANN, and you need to perform back-propagation to train the network again, and so on.

Each time you output the result of forward stage, our pattern file will check the correctness of it. Basically, if you follow the formula and use IEEE floating point number IP, you should get same result as our answer. However, **we release the constraint; you may have an error under 0.0001 for the result after converting to float number. This means that we will convert your output from binary format into real float number, and compare with our answer. Error will be calculated by '(golden-ans)/golden'. If the error is higher than the value, you will fail this lab.**

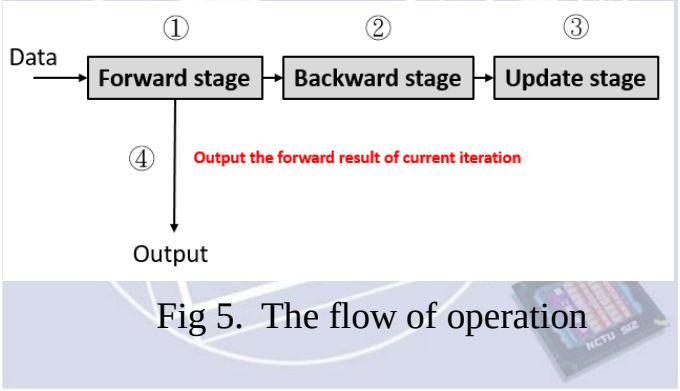
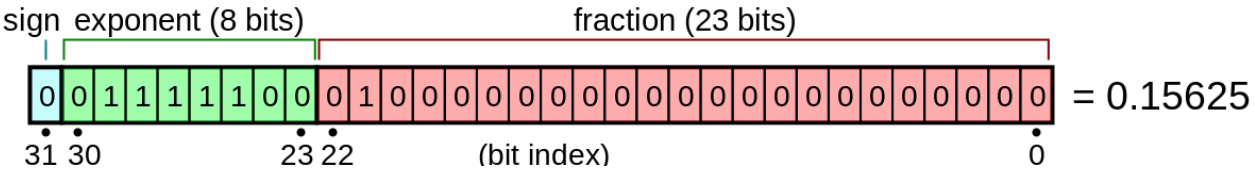


Fig 5. The flow of operation



$$\text{sign} = +1$$
$$\text{exponent} = (-127) + 124 = -3$$
$$\text{fraction} = 1 + 2^{-2} = 1.25$$
$$\text{value} = (+1) \times 1.25 \times 2^{-3} = +0.15625$$

Fig 6. Example of IEEE-754

Inputs and Outputs

The following are the definitions of input signals

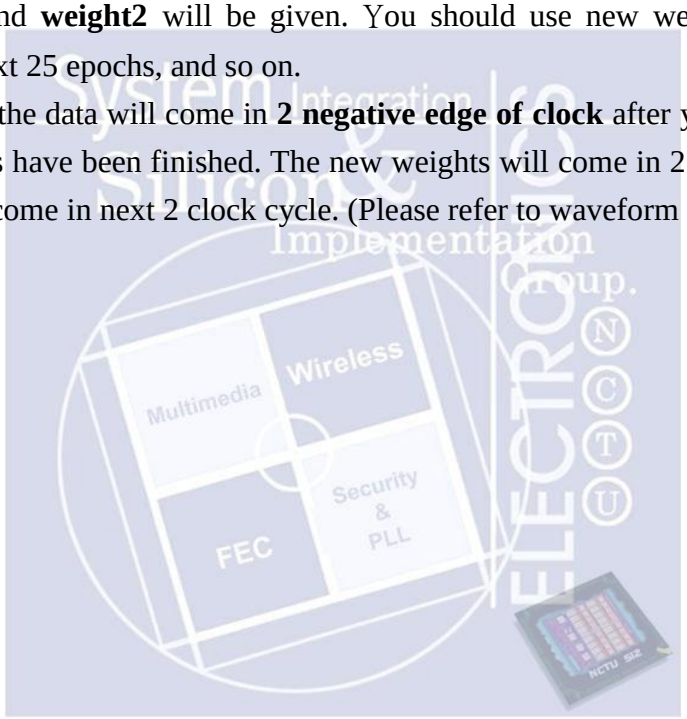
Input Signals	Bit Width	Definition
clk	1	Clock.
rst_n	1	Asynchronous active-low reset.
in_valid_d	1	High when data_point is valid.
in_valid_t	1	High when target is valid.
in_valid_w1	1	High when weight1 is valid
in_valid_w2	1	High when weight2 is valid
data_point	32	The input data point, 4 data points represent a data. The arithmetic representation follows the IEEE-754 floating number format.
target	32	The target for current data. The arithmetic representation follows the IEEE-754 floating number format.
weight1	32	The weight for first layer. The arithmetic representation follows the IEEE-754 floating number format.
weight2	32	The weight for second layer. The arithmetic representation follows the IEEE-754 floating number format.

The following are the definitions of output signals

Output Signals	Bit Width	Definition
out_valid	1	High when out is valid.
out	32	The forward result for current data. The arithmetic representation follows the IEEE-754 floating number format.

1. The input signal **data_point** is delivered for **4 cycles** continuously. The order is d_0, d_1, d_2, d_3. When **in_valid_d** is low, input is tied to unknown state.
2. The input signal **target** is delivered for **1 cycle**. When **in_valid_t** is low, input is tied to unknown state.
3. The input signal **weight1** is delivered for **12 cycles** continuously. The order is $W_{00}^1, W_{01}^1 \ldots, W_{20}^1, W_{23}^1$. When **in_valid_w1** is low, input is tied to unknown state.

4. The input signal **weight2** is delivered for **3 cycles** continuously. The order is $W_{00}^2, W_{01}^2, W_{02}^2$. When **in_valid_w2** is low, input is tied to unknown state.
5. All input signals are synchronized at negative edge of the clock.
6. The output signal **out** must be delivered for **1 cycle**, and **out_valid** should be high simultaneously.
7. The input signal **data_point** and **target** will start to be given at same time.
8. The input signal **weight1** and **weight2** will start to be given at same time.
9. The **in_valid_d** and **in_valid_t** will come in **2 cycles** after **in_valid_w1** is pulled down.
10. At the beginning, we will send **weight1** and **weight2** to your ANN. Then, we will send you first data. After receiving the output from your design, we will send second data to you. When **100 data** have been given (one epoch), we will again give from first data, and second data, and so on. We will test your ANN for **25 epochs** in this set of weights. After 25 epochs, the new input signals **weight1** and **weight2** will be given. You should use new weights to perform back-propagation for next 25 epochs, and so on.
11. The next round of the data will come in **2 negative edge of clock** after your **out_valid** is pulled down. If 25 epochs have been finished. The new weights will come in 2 negative edge of clock, and first data will come in next 2 clock cycle. (Please refer to waveform example)



Specifications

1. Top module name: NN (design file name: NN.v)
2. **It is asynchronous reset and active-low architecture. If you use synchronous reset (considering reset after clock starting) in your design, you may fail to reset signals.**
3. **The reset signal (rst_n) would be given only once at the beginning of simulation. All output signals should be reset after the reset signal is asserted.**
4. The **out** should be reset after your **out_valid** is pulled down.
5. The **out_valid** is limited to high only **one** cycle when you want to output the result.
6. The execution latency is limited in **300 cycles**. The latency is the clock cycles between the falling edge of the last **in_valid_d** and the rising edge of the first **out_valid**.
7. We will convert your forward result to float number and compare to our answer, if you follow the formula correctly and use the IEEE floating point related Designware, the result should be same as our pattern produced. **However, we release the constraint and allow the error under 0.0001. If the error exceeds this value, you will fail this lab**. It should be noted that the error can be propagated during the computation of algorithm, thus it is strongly recommended to use IEEE floating number IP to finish the lab.
8. You can adjust your clock period by yourself, but the maximum period is **20 ns**. The precision of clock period is 0.1, for example, 4.5 is allowed, 4.55 is not allowed.
9. The input delay is set to **0.5*(clock period)**.
10. The output delay is set to **0.5*(clock period)**, and the output loading is set to **0.05**.
11. The synthesis result of data type **cannot** include any **latches**.
12. The gate level simulation cannot include any timing violations without the *notimingcheck* command.
13. After synthesis, you can check NN.area and NN.timing. The area report is valid when the slack in the end of timing report should be **non-negative (MET)**.
14. **In this lab, you must use at least one IEEE floating point number IP from Designware. We will check it at NN.resource in 02_SYN/Report/. The example shows in following figure.**

```
=======
| Cell        | Module       | Parameters | Contained Operations      |
=============================================================================
=======
| lt_x_45     | DW_cmp       | width=10   | lt_836 (NN.v:836)         |
| add_x_46    | DW01_inc     | width=10   | add_837 (NN.v:837)        |
| add_x_48    | DW01_inc     | width=10   | add_851 (NN.v:851)        |
| add_x_50    | DW01_inc     | width=10   | add_867 (NN.v:867)        |
| M1          | DW_fp_mult   | sig_width=23 | M1 (NN.v:103)           |
|             |              | exp_width=8 |                          |
|             |              | ieee_compliance=0 |                     |
| S0          | DW_fp_sub    | sig_width=23 | S0 (NN.v:104)           |
|             |              | exp_width=8 |                          |
|             |              | ieee_compliance=0 |                     |
```

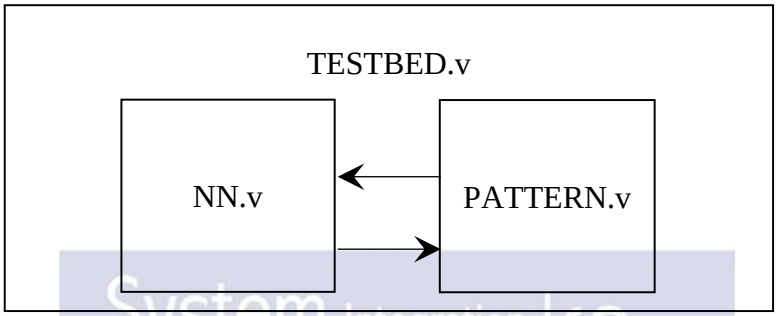
Fig 1. The information of floating point related IP in NN.resource file

15. The performance is determined by **area, latency** and **clock cycle time**. The lower, the better.
16. Don't use any wire / reg / submodule name called "error" or "congratulation", or you will fail the lab.

Grading Policy

1. Function Validity: 70%
2. Performance:30 %
   - Area * Computation time: 30%
   - Computation time = (Pattern number+ Latency) * clock cycle time

Block diagram



Note

1. **Please upload the following files on e3 platform before 12:00 at noon on Oct. 18:**
   - NN_iclab???.v and clock_cycle_iclab???.txt (ex. NN_iclab099.v & 15.5_iclab099.txt), the .txt file contents can be empty, you only need to specify the clock cycle in the file name.
   - The 2nd demo deadline is 12:00 **at noon** on Oct. 20.
   - Check whether there is any wire / reg / submodule name called “error” or “congratulation”, if you used, you will fail the lab.
   - If your file violates the naming rule, your will lose 5 point.
2. **Template folders and reference commands:**

   01_RTL/ (RTL simulation) **./01_run**

   02_SYN/ (Synthesis) **./01_run_dc**

   (Check if there is any **latch** in your design in **syn.log)**

   (Check the timing of design in /Report/NN**.timing**)

   03_GATE / (Gate-level simulation) **./01_run**

Sample Waveform

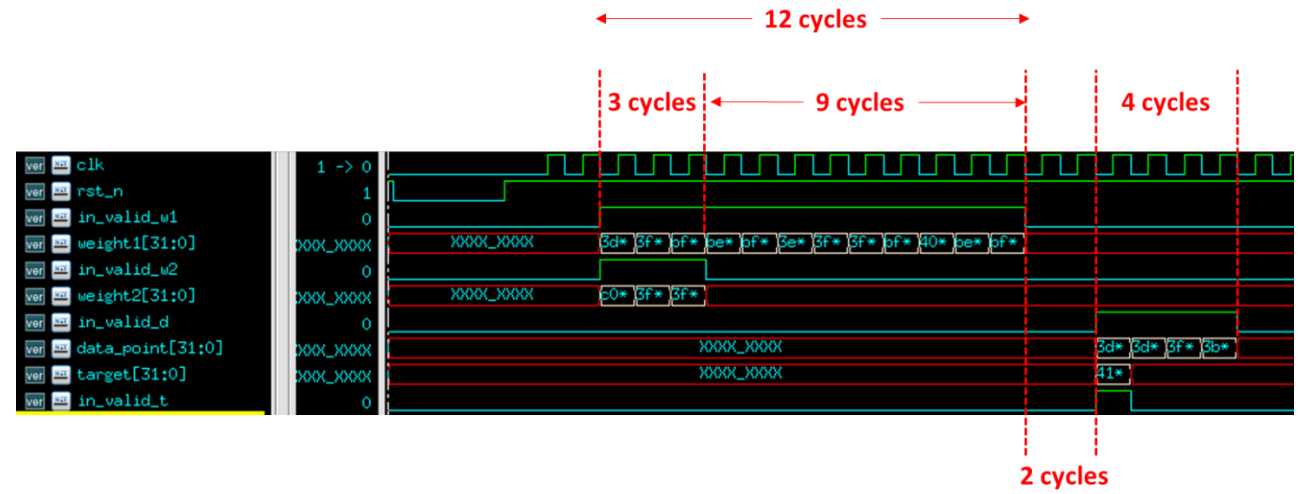


Fig 2. Input waveform (at beginning)

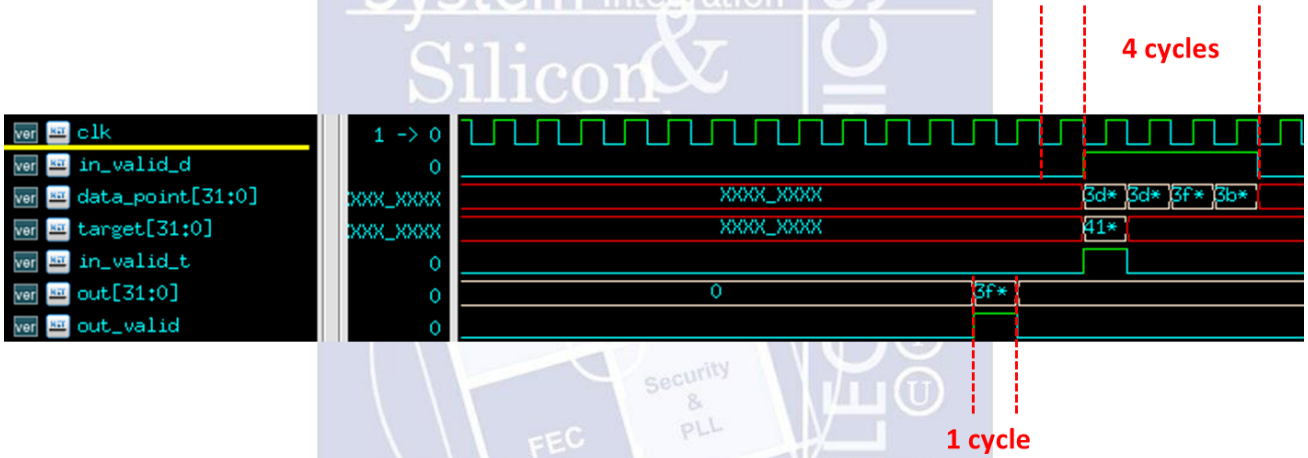


Fig 3. Output waveform

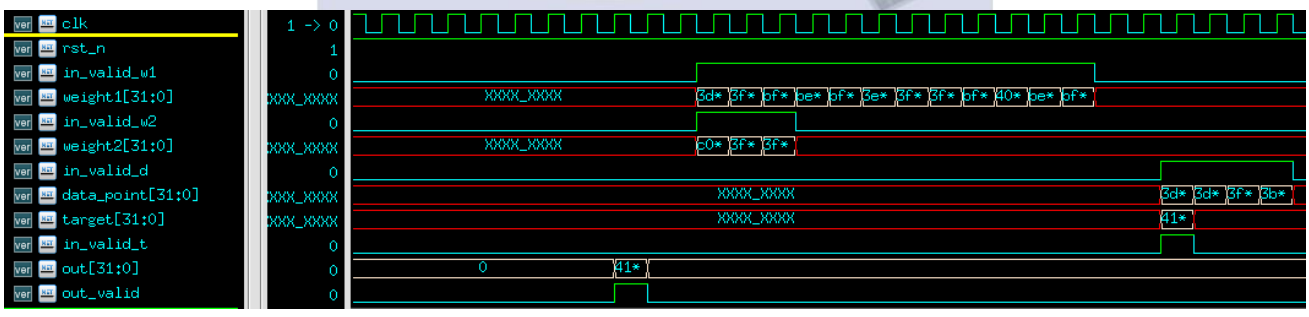


Fig 4. Input waveform (using new weights)

Lab04 Note

- **About the adjustment of clock period**

  You should make sure the three clock period values identical:

  1. /00_TESTBED/Pattern.v

```
`ifdef RTL
    `timescale 1ns/10ps
    `include "NN.v"
    `define CYCLE_TIME 20.0
`endif
`ifdef GATE
    `timescale 1ns/1ps
    `include "NN_SYN.v"
    `define CYCLE_TIME 20.0
`endif
```

  2. /02_SYN/syn.tcl

```
#=========================
#  Global Parameters
#=========================
set DESIGN "NN"
set CLK_period 20.0
```

- **About the testing of design**

Do not over thinking the terminology **iteration** and **epoch**. The purpose of these definitions is just for convenience. The things you need to know is that we will send a set of weights to your design, then give you the first data. Next, you need to perform back-propagation to update the weights. Then, you need to pull-up the out_valid signal and output the result of forward stage. For now, you just finished 1 iteration.

Next, we will give you second data, and you need to perform back-propagation again, then output the forward result, and so on. After you finish the algorithm for 100th data, we say that you finish 1 epoch, then we just send the first data to you again, nothing special, and you need to perform same thing as before. Finally, when you finish 25 epochs, we will first send new weights to your ANN, then send new data to your ANN, just like the beginning of simulation. You need to use new weights to perform back-propagation for 25 epochs, and so on.

So, the things you need to do for each data is the same. When we say that you need to perform the algorithm for 25 epochs for the weights, this also means you need to perform 2500 times back-propagation for the weights.

- **Numerical example of back-propagation algorithm**

For your convenience, I provide an example for the back-propagation algorithm. The numbers are from the .txt file in 00_TESTBED. It shows the algorithm for first data. **This example is just for reference, if you find any problem, you can ask a question on FB. If you fail this lab, we do not accept any debate on this example.**

The numbers in this example is from the display of nWave, which may be a little different with true value, for example:

Binary form: 00111101101100101010000001000101
IEEE floating number: 0.08721975
nWave:

nWave will round the number for display, but the computation will not be affected. The following numbers are all from nWave, thus the computations are performed by IPs in Verilog. Here, e-02 = 10^{-2}。

The first data after converting:
8.72198×10^{-2}
3.69334×10^{-2}
9.95498×10^{-1}
3.40924×10^{-3}

The target after converting:
1.80000×10^{1}
The weights for first layer after converting:
8.45988×10^{-2}
1.29359×10^{0}
-6.82691×10^{-1}
-2.55916×10^{-1}
-1.16913×10^{0}
4.43927×10^{-1}
1.34207×10^{0}
1.97281×10^{0}
-6.81856×10^{-1}
2.81375×10^{0}
-2.58320×10^{-1}
-1.20062×10^{0}

The weights for first layer after converting:

-2.29034×10^{0}

1.33381×10^{0}

7.53121×10^{-1}

Forward (first layer)

$$h_0^1 = W_{00}^1 * d_0 + W_{01}^1 * d_1 + W_{02}^1 * d_2 + W_{03}^1 * d_3 = 8.45988 \times 10^{-2} * 8.72198 \times 10^{-2} + 1.29359 \times 10^{0} * 3.69334 \times 10^{-2} + (-6.82691 \times 10^{-1}) * 9.95498 \times 10^{-1} + (-2.55916 \times 10^{-1}) * 3.40924 \times 10^{-3} = -6.25335 \times 10^{-1}$$

$h_1^1 = 1.25718 \times 10^{0}$

$h_2^1 = -2.16800 \times 10^{-1}$

$y_0^1 = g(h_0^1) = 0$

$y_1^1 = g(h_1^1) = 1.25718 \times 10^{0}$

$y_2^1 = g(h_2^1) = 0$

Forward (second layer)

$$y_0^2 = h_0^2 = W_{00}^2 * y_0^1 + W_{01}^2 * y_1^1 + W_{02}^1 * y_2^1 = -2.29034 \times 10^{0} * 0 + 1.33381 \times 10^{0} * 1.25718 \times 10^{0} + 7.53121 \times 10^{-1} * 0 = 1.67684 \times 10^{0}$$

Backward (second layer)

$$\delta_0^2 = (1.67684 \times 10^{0} - 1.80000 \times 10^{1}) = -1.63232 \times 10^{1}$$

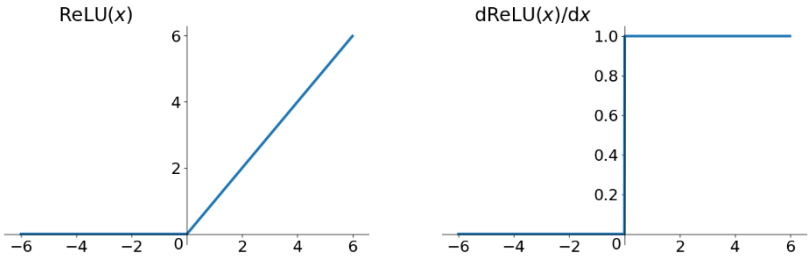
Backward (first layer):

$g(\cdot)'$: the derivative of activation function.

$g(h_0^1)' = 0$

$g(h_1^1)' = 1$

$g(h_2^1)' = 0$



$$\delta_0^1 = g(h_0^1)' * W_{00}^2 \delta_0^2 = 0 * (-2.29034 \times 10^{0} * (-1.63232 \times 10^{1})) = 0$$

$$\delta_1^1 = g(h_0^1)' * W_{01}^2 \delta_0^2 = 1 * (1.33381 \times 10^{0} * (-1.63232 \times 10^{1})) = -2.17720 \times 10^{1}$$

$\delta_2^1 = 0$

Update (second layers)

$$w_{00}^2 = w_{00}^2 - \eta\delta_0^2 s_0^2$$
$$= -2.29034 \times 10^0 - (0.000001 * (-1.63232 \times 10^1) * 0)$$
$$= -2.29034 \times 10^0$$

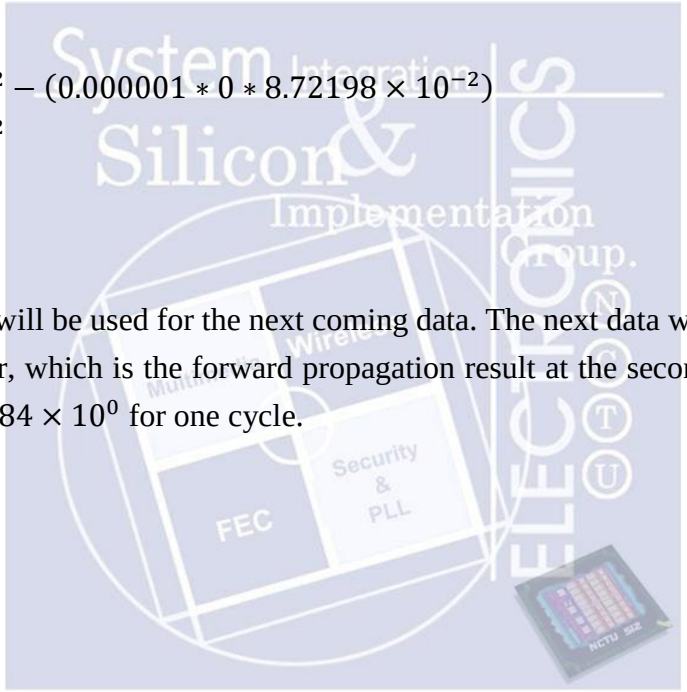
$$w_{01}^2 = w_{01}^2 - \eta\delta_0^2 s_1^2$$
$$= 1.33381 \times 10^0 - (0.000001 * (-1.63232 \times 10^1) * 1.25718 \times 10^0)$$
$$= 1.33383 \times 10^0$$

$$w_{02}^2 = w_{02}^2 - \eta\delta_0^2 s_2^2$$
$$= 7.53121 \times 10^{-1} - (0.000001 * (-1.63232 \times 10^1) * 0)$$
$$= 7.53121 \times 10^{-1}$$

Update (first layers)

$$w_{00}^1 = w_{00}^1 - \eta\delta_0^1 s_0^1$$
$$= 8.45988 \times 10^{-2} - (0.000001 * 0 * 8.72198 \times 10^{-2})$$
$$= 8.45988 \times 10^{-2}$$

And so on …



These updated weights will be used for the next coming data. The next data will come after you gives an output signal number, which is the forward propagation result at the second layer output you just calculated: $y_0^2 = 1.67684 \times 10^0$ for one cycle.