

NYCU-EE IC LAB - Spring 2023

Lab06 Exercise

Design: Huffman Code Operation

Data Preparation

1. Extract test data from TA's directory:
`% tar xvf ~iclabTA01/Lab06.tar`
2. The extracted LAB directory contains:
 - a. Practice/
 - b. Exercise_SoftIP/
 - c. Exercise/

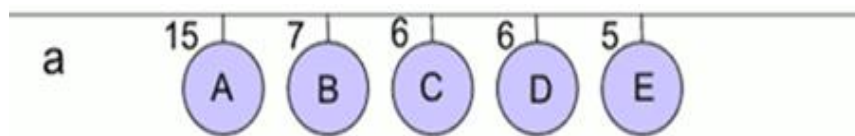
Design Introduction

■ Huffman coding

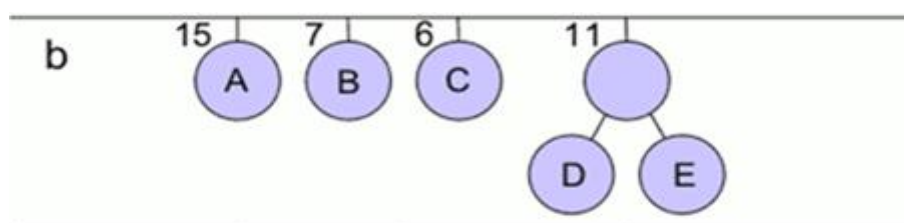
In computer science and information theory, a Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression. The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source symbol (such as a character in a file). The algorithm derives a table from the estimated probability or frequency of occurrence (weight) for each possible value of the source symbol. As in other entropy encoding methods, more common symbols are generally represented using fewer bits than less common symbols. Huffman's method can be efficiently implemented, finding a code in time linear to the number of input weights if these weights are sorted.

Example

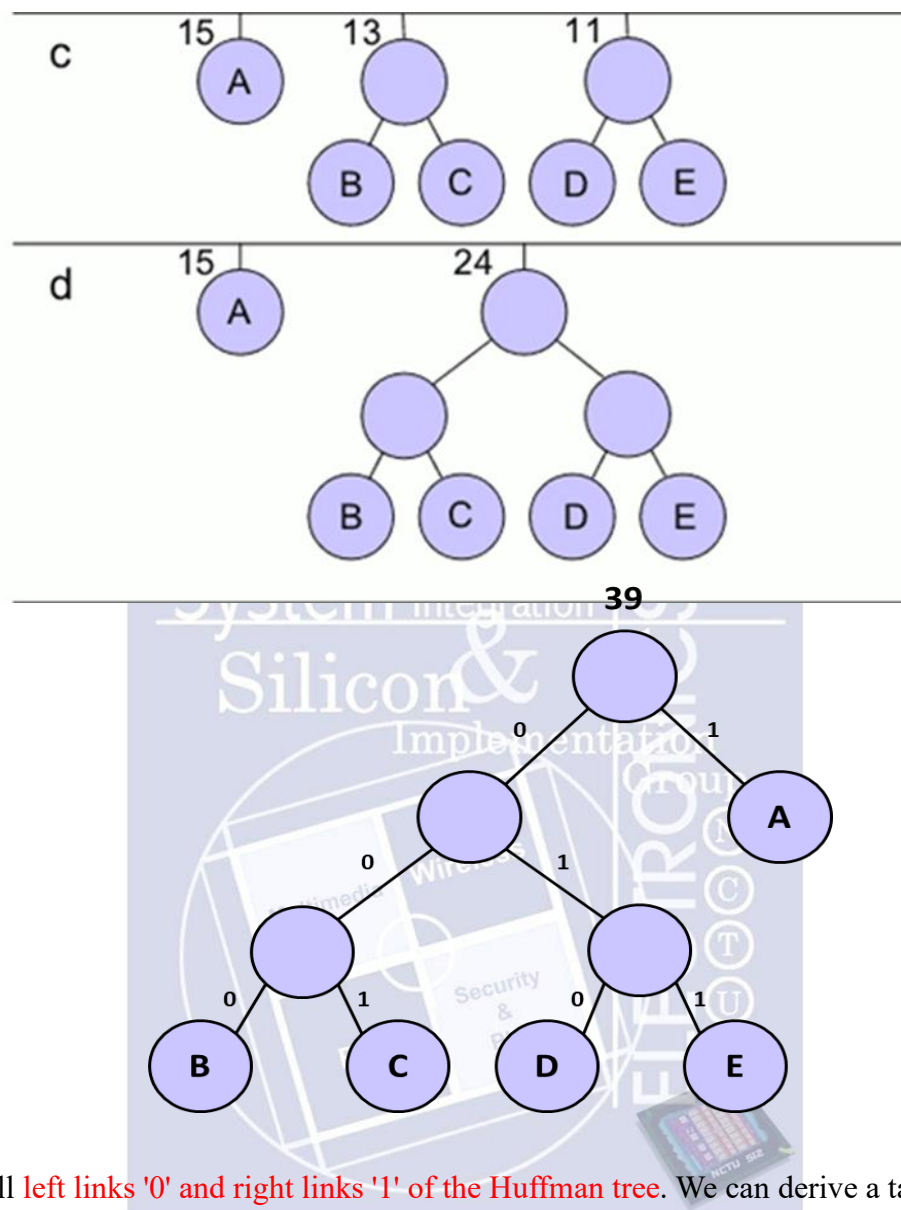
Consider five character A,B,C,D and E. Each have their own weight number represented their probability or frequency. First, **sort them by their weight**. If encountering the same weight, **sort them by their character number order** ($A > B > C > E > D > I > L > O > V > \text{Old subtree} > \text{New subtree}$).



Then, **combine the two smallest elements** together. The bigger one is on the left node and the lower one is on the right node.



By repeating actions of sorting and selecting, you can yield a complete tree-like structure called Huffman tree.



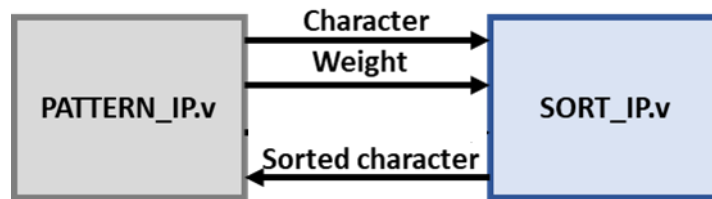
Finally, give all **left links '0' and right links '1' of the Huffman tree**. We can derive a table of Huffman code.

| Character | A | B | C | D | E |
|--------------|----|-----|-----|-----|-----|
| Weight | 15 | 7 | 6 | 6 | 5 |
| Huffman Code | 1 | 000 | 001 | 010 | 011 |

Design Description

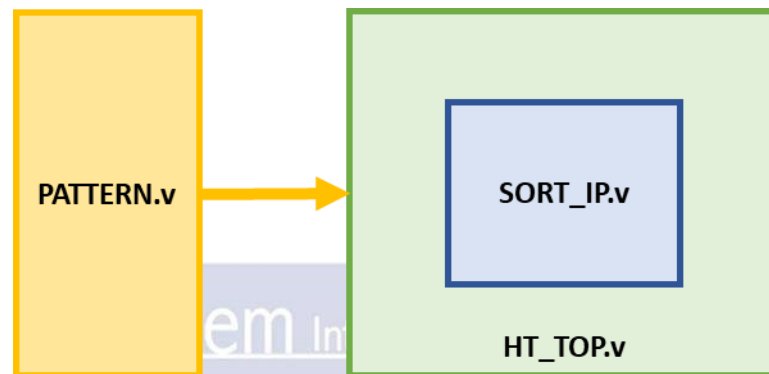
| Design | Definition |
|-------------------|---|
| Soft IP | Given input characters, weights. The soft IP need to sort weights. |
| Top Design | Given input weights, and output mode. The top design needs to build Huffman tree and derive Huffman code of each character. |

■ *Soft IP*

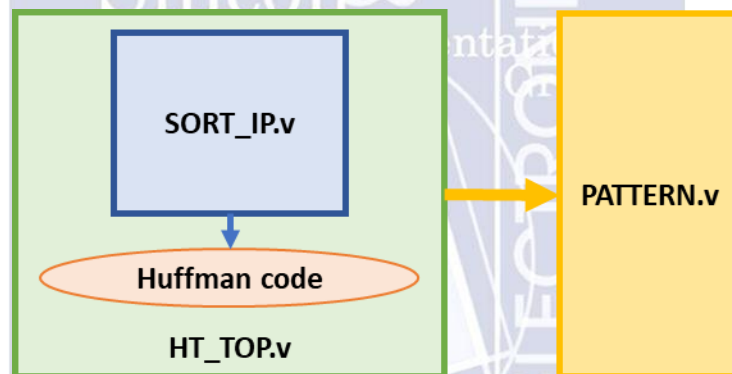


■ *Top Design*

Step-1: Given input weights, and mode. Use your own designed soft IP to sort the characters.



Step-2: Use the sorted characters to build Huffman tree and derive the Huffman code.



Inputs and Outputs (Top Design)

■ The following are the definitions of input signals:

| Input signal | Bit width | Definition |
|--------------|-----------|--|
| clk | 1 | Clock. |
| rst_n | 1 | Asynchronous active-low reset. |
| in_valid | 1 | High when input signals are valid. |
| in_weight | 3 | The weight of characters. |
| out_mode | 1 | If out_mode is 0, output the Huffman code of "ILOVE" If out_mode is 1, output the Huffman code of "ICLAB" |

■ The input of in_weight correspond sequentially to the characters A, B, C, E, I, L, O, V.

- The following are the definitions of output signals:

| Output signal | Bit width | Definition |
|---------------|-----------|---------------------------------------|
| out_valid | 1 | High when output signals are valid. |
| out_code | 1 | Output the corresponded Huffman code. |

- Example of out_code : If Huffman code of A, B is 101, 010, then the out_code of AB is sequentially output 101010.

Inputs and Outputs (Soft IP)

- The following are the definitions of input signals:

| Input signal | Bit width | Definition |
|--------------|-------------|--|
| IN_character | IP_WIDTH *4 | The characters that need to be sorted. |
| IN_weight | IP_WIDTH *5 | The weight of input characters. |

- In soft ip demo, IN_character is a fix sequence of consecutive numbers from large to small. For example, if IP_WIDTH is five, IN_character will be {4'd4,4'd3,4'd2,4'd1,4'd0}.

- The following are the definitions of two parameters:

| Parameter | Bit width | Definition |
|-----------|-----------|---------------------------------|
| IP_WIDTH | - | The number of input characters. |

Example: #(4)

1. 4 means that there are four input characters.
2. **There won't be illegal case #(0).**

- The following are the definitions of output signals:

| Output signal | Bit width | Definition |
|---------------|-------------|----------------------------------|
| OUT_character | IP_WIDTH *4 | Output of the sorted characters. |

- Example : If the sorted result is $A > C > B$, then corresponding OUT_character is {A, C, B}.

Specifications (Top Design)

Top module

1. Top module name: **HT_TOP** (design file name: **HT_TOP.v**).
2. You can adjust your clock period by yourself, but the maximum period is **20ns**. The precision of clock period is 0.1, for example, 40.5 is allowed, but 40.55 is not allowed.
3. The execution latency is limited to **2000 cycles**. The latency is the clock cycles between the falling edge of the last cycle of **in_valid** and the rising edge of the **out_valid**.
4. The total cell area should not be larger than **2,000,000 um²**.
5. **The look-up table method is forbidden**, and you need to **use your own-designed soft IP (SORT_IP) in the top module**. (TA will check your design)

Reset

6. 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.
7. 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.**

in valid

8. **in_valid** will come after reset.
9. All input signals are synchronized at **negative edge** of the clock.
10. When **in_valid** is low, input is tied to unknown state.

out valid

11. **out_valid** should not be raised when **in_valid** is high.
12. The **out_valid** is limited to be high only when the output value is valid.
13. All output signals should be synchronized at clock positive edge.
14. The TA's pattern will capture your output for checking at **clock negative edge**.
15. The **out_code** should be correct when **out_valid** is high.
16. The next input pattern will come in **2~4 negative edge of clock** after your **out_valid** falls.

Synthesis

17. **In this lab, you should write your own syn.tcl file.**
18. Use **top** wire load mode and **compile_ultra**.
19. Use **analyze + elaborate** to read your design.
20. The input delay is set to $0.5 * (\text{clock period})$.
21. The output delay is set to $0.5 * (\text{clock period})$, and the output loading is set to 0.05.
22. The input delay of clk and rst_n should be zero.
23. **The synthesis time should be less than 2 hours.**
24. The synthesis result (syn.log) of data type **cannot** include any **latches and error**.
25. After synthesis, you can check **HT_TOP.area** and **HT_TOP.timing**. The area report is valid when the slack in the end of timing report should be **non-negative** and the result should be **MET**.

Gate level simulation

26. The gate level simulation cannot include any timing violations without the notimingcheck command.

Supplement

27. In this lab, you are **NOT** allowed to use Designware IP.
28. Don't use any wire/reg/submodule/parameter name called ***error***, ***congratulation***, ***pass***, ***latch*** or ***fail*** otherwise you will fail the lab. Note: ***** means any char in front of or behind the word. e.g: error_note is forbidden.
29. Don't write chinese comments or other language comments in the file you turned in.

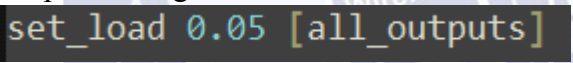
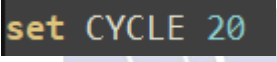
30. Verilog commands `//synopsys dc_script_begin, //synopsys dc_script_end`
`//synopsys translate_off, //synopsys translate_on` are only allowed during the usage of including and setting designware IPs, other design compiler optimizations are forbidden. Using the above commands are allowed, however any error messages during synthesize and simulation, regardless of the result will lead to failure in this lab.
- Any form of display or printing information in verilog design is forbidden. You may use this methodology during debugging, but the file you turn in should not contain any coding that is not synthesizable.

Specifications (Soft IP)

Top module

1. Top module name: **SORT_IP** (design file name: **SORT_IP.v**)
Input signals: **IN_character, IN_weight**
Output signals: **OUT_character**
One parameters : **IP_WIDTH**
2. The clock period is **20ns**. Finish calculating within **one** clock cycle.
3. You need to use **generate** to design this soft IP.
4. **The look-up table method is forbidden.** (TA will check your design)

Synthesis

5. Output loading is set to **0.05**.

6. Use **set CYCLE** to adjust your clock period.

7. Using **top** wire load mode and **compile_ultra**.
8. Use **analyze + elaborate** to read your design.

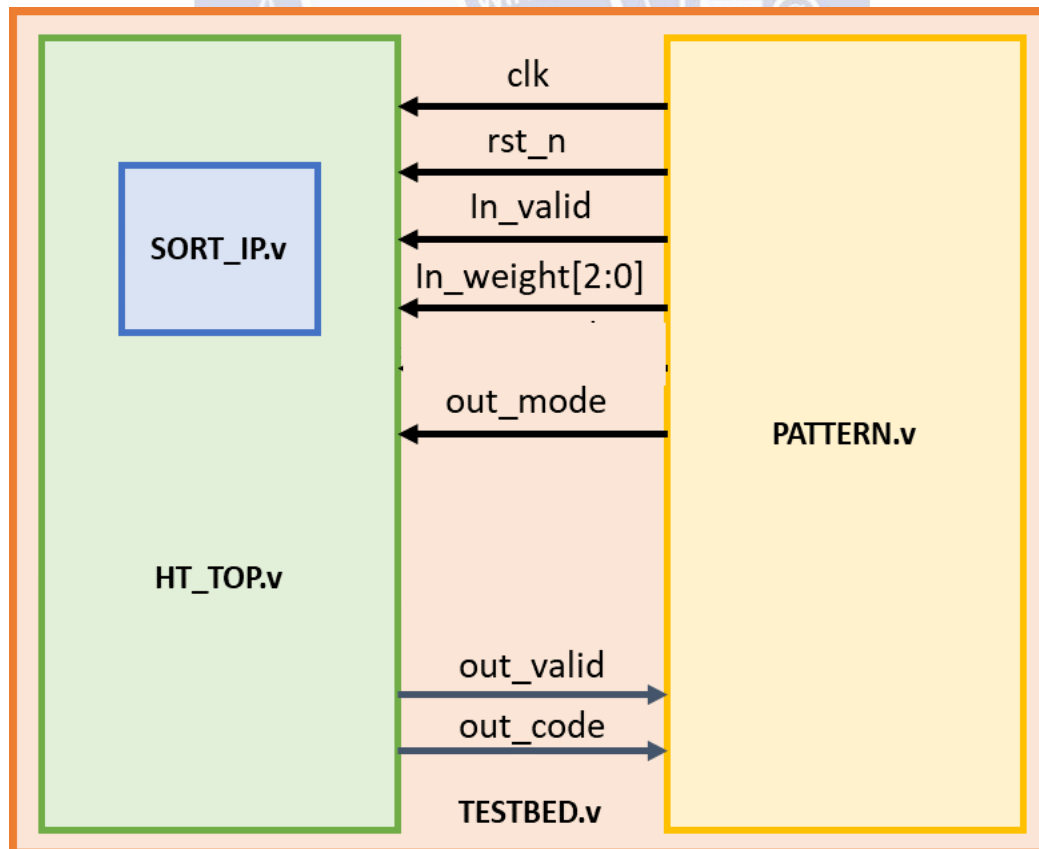
Supplement

9. Don't use any wire/reg/submodule/parameter name called ***error***, ***congratulation***, ***pass***, ***latch*** or ***fail*** otherwise you will fail the lab. Note: ***** means any char in front of or behind the word. e.g: **error_note** is forbidden.
10. Don't write chinese or other language comments in the file you turned in.
11. Verilog commands `//synopsys dc_script_begin, //synopsys dc_script_end`
`//synopsys translate_off, //synopsys translate_on` are only allowed during the usage of including and setting designware IPs, other design compiler optimizations are forbidden.
Using the above commands are allowed, however any error messages during synthesize and simulation, regardless of the result will lead to failure in this lab.
Any form of display or printing information in verilog design is forbidden. You may use this methodology during debugging, but the file you turn in should not contain any coding that is not synthesizable.

Soft IP Testing environment

```
//#####  
// *****  
// File Name : SORT_IP_demo.v  
// Module Name : SORT_IP_demo  
// *****  
//#####  
  
//synopsys translate_off  
`include "SORT_IP.v"  
//synopsys translate_on  
  
module SORT_IP_demo #(parameter IP_WIDTH = 3)(  
    //Input signals  
    IN_character, IN_weight,  
    //Output signals  
    OUT_character  
);  
  
// =====  
// Input & Output Declaration  
// =====  
input [IP_WIDTH*4-1:0] IN_character;  
input [IP_WIDTH*5-1:0] IN_weight;  
  
output [IP_WIDTH*4-1:0] OUT_character;  
  
// =====  
// Soft IP  
// =====  
SORT_IP #(.IP_WIDTH(IP_WIDTH)) I_SORT_IP(.IN_character(IN_character), .IN_weight(IN_weight), .OUT_character(OUT_character));  
  
endmodule
```

Block diagram



Note

1. Grading policy:

- Function Validity: 50% (RTL and Gate-level simulation correctness)
 - Top design
- Performance: 30% $Area * (Latency\ Time + 1) * Cycle\ time$
 - Top design
- Soft IP function correctness: 20% (**No second demo**)
 - IP_WIDTH = 8-bits: 8%
 - IP_WIDTH = 7-bits: 5%
 - IP_WIDTH = 6-bits: 3%
 - IP_WIDTH = 5-bits: 2%
 - IP_WIDTH = 4-bits: 1%
 - IP_WIDTH = 3-bits: 1%
- The performance is determined by area and latency of your design. The less cost your design has, the higher grade you get.
- The grade of 2nd demo would be 30% off.

2. Please submit your files under 09_SUBMIT. (09_SUBMIT is under Exercise/.)

1st demo : before 12:00 p.m. on 10/30(Mon.)

2nd demo : before 12:00 p.m. on 11/1(Wed.)

You should check the following files under 09_SUBMIT/Lab06_iclabXXX/

- Top : HT_TOP_iclabxxx.v
- Soft IP : SORT_IP_iclabxxx.v
- Cycle Time : CYCLE_iclabxxx.txt
- Syn.tcl : syn_iclabxxx.tcl
- xxx is your account number, i.e. HT_TOP_iclab999.v \ SORT_IP_iclab999.v
- If you miss any files on the list, you will fail this lab.
- If the uploaded file violating the naming rule, you will get **5 deduct points**.

Then use the command like the figure below to check the files are uploaded or not.

```
[Exercise/09_SUBMIT]% ./02_check 1st_demo
```

3. Template folders and reference commands:

01_RTL/ (RTL simulation) ./01_run_vcs_rtl

02_SYN/ (Synthesis) ./01_run_dc_shell

(Check latch by searching the keyword "Latch" in syn.log)

(Check the design's timing in /Report/HT_TOP.timing)

(Check the design's area in /Report/ HT_TOP.area)

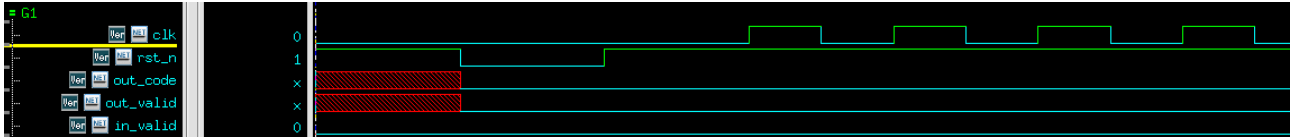
03_GATE / (Gate-level simulation) ./01_run_vcs_gate

- You can key in ./09_clean to clear all log files and dump files in each folder.

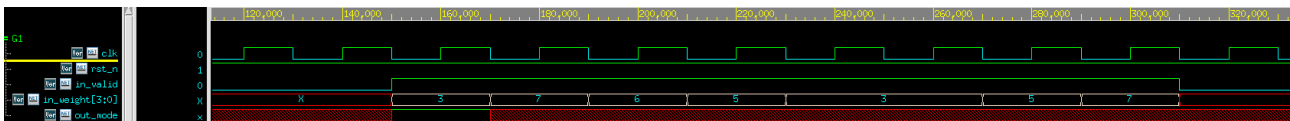
- You should make sure the **three clock period values identical** in 00_TESTBED/PATTERN.v and 02_SYN/syn.tcl

Sample Waveform

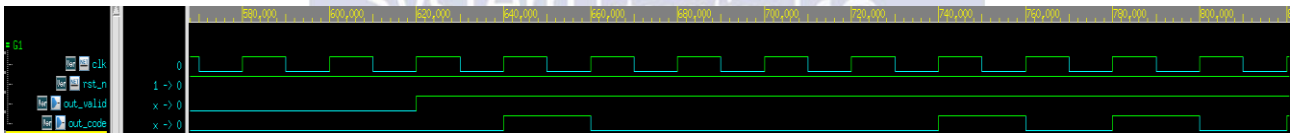
- Asynchronous reset and active-low and reset all output.



- 8 cycle for input the information of weights and 1 cycle for input the information of mode in each round/pattern.

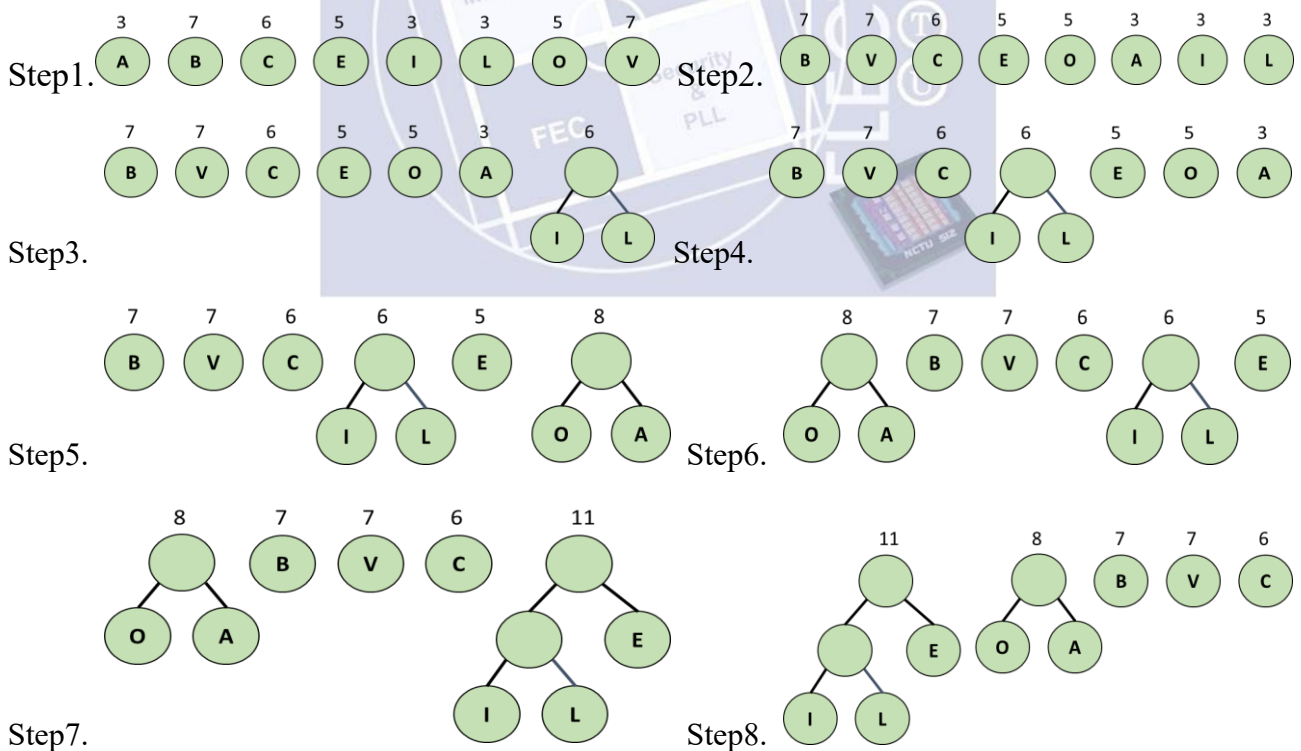


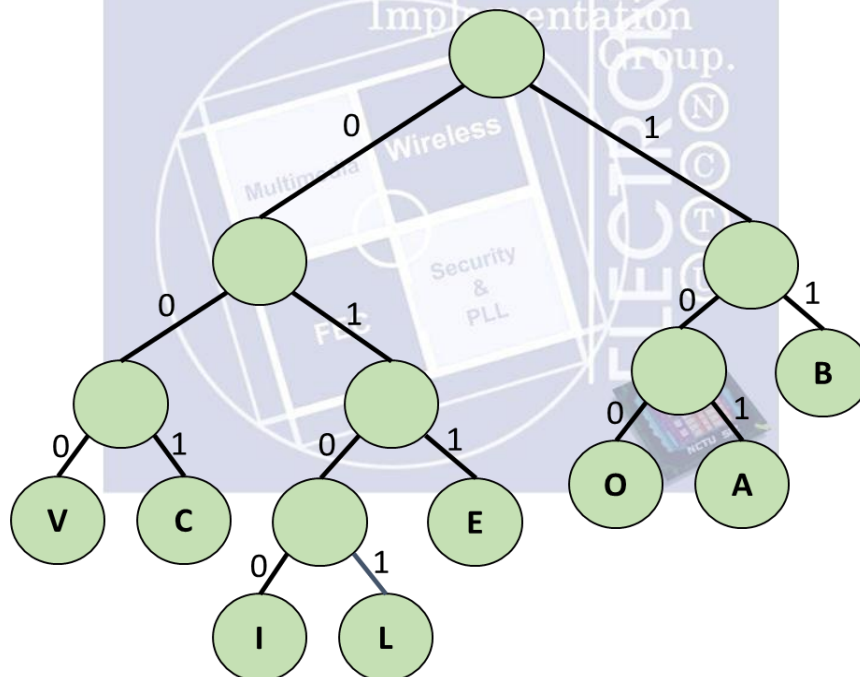
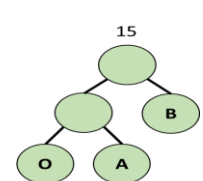
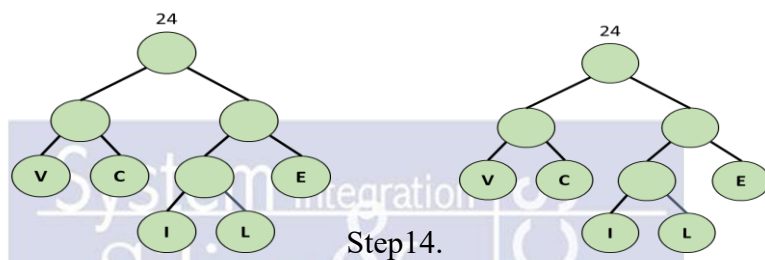
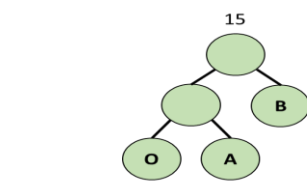
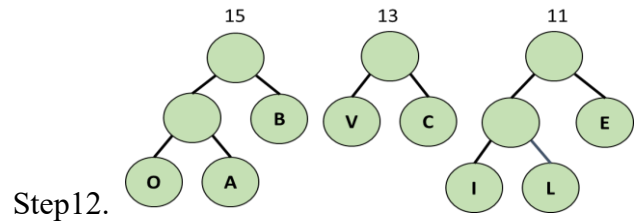
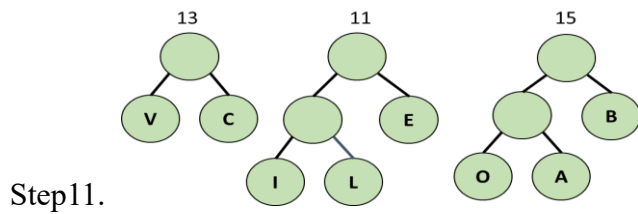
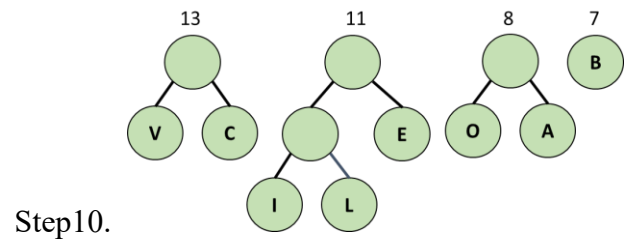
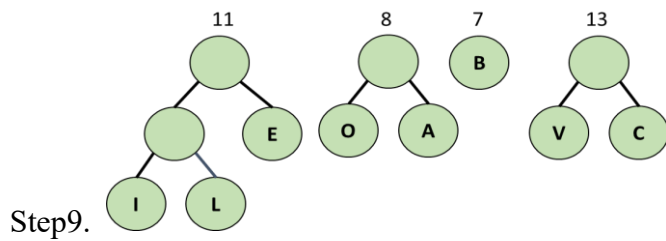
- Continuously output the result of deferent mode in each round/pattern.



- The **out_code** should be correct when **out_valid** is high, that is, **both signals will be checked at every clock negative edge.**

Example





| Character | A | B | C | E | I | L | O | V |
|--------------|-----|----|-----|-----|------|------|-----|-----|
| Weight | 3 | 7 | 6 | 5 | 3 | 3 | 5 | 7 |
| Huffman Code | 101 | 11 | 001 | 011 | 0100 | 0101 | 100 | 000 |

1. out_mode = 0 : out_code is “ILOVE” -> **01000101100000011**
2. out_mode = 1 : out_code is “ICLAB” -> **0100001010110111**

Reference

■ *Huffman coding*

https://en.wikipedia.org/wiki/Huffman_coding

