NYCU-DCS-2024

HW03

Design: Huffman Coding

Data Preparation

1. Extract files from TA's directory:
   % tar xvf ~DCSTA01/HW03.tar
2. The extracted LAB directory contains:
   a. **1_SystemC**
   b. **2_Verilog**
      I. **00_TESTBED**
      II. **01_RTL**
      III. **02_SYN**
      IV. **03_GATE**
      V. **09_SUBMIT**
   c. **3_PA**

Design Description

Huffman coding is an algorithm used for lossless data compression of finding or using **Huffman code**, which is a particular type of optimal prefix code. It is developed by David A. Huffman in 1952.

In this design, a Huffman encoder receives original data and counts all source symbols. After counting each symbol, Huffman encoder encodes the Huffman code with Huffman Tree; more details will be introduced in SPEC.

Functional Description

There is a grayscale image consisting of only 100 pixels, where each pixel is stored as 8-bit grayscale data. For simplicity, only six grayscale levels (six symbols) are considered: A1, A2, A3, A4, A5, and A6, whose corresponding values (in hexadecimal) are shown in the table below. The index for symbols A1, A2, A3, A4, A5, and A6 are defined as 1, 2, 3, 4, 5, and 6, respectively.

Symbol	A1	A2	A3	A4	A5	A6
Value	0x01	0x02	0x03	0x04	0x05	0x06

Table 1. The symbols of grayscale image and corresponding values in hexadecimal

After the **reset** is ended, the Pattern makes the signal of **gray_valid** to high,

which indicating that **gray_data** in one cycle represents the input of one pixel, with a total of 100 pixels. After these 100 pixels of data input, the signal of **gray_valid** becomes low.

In the first stage, the system circuit counts the occurrence of each symbol from the input data of grayscale image.

Symbol	CNT(Ai)
A1	10
A2	40
A3	6
A4	10
A5	4
A6	30

Table 2. Counting results of each symbol, where CNT(Ai) represents the counting number of the symbol Ai

Here is an example for Huffman coding. The counting numbers and Huffman codes of each symbol in this example are shown in Table3. To explain Huffman coding, there are three stage for encoding: initialization, combination, and split.

Aid	A1	A2	A3	A4	A5	A6
CNT(Ai)	10	40	6	10	4	30
Huffman code (Ai)	011	1	01010	0100	01011	00

Table 3. The example for Huffman coding indicates the counting numbers and Huffman codes of each symbol.

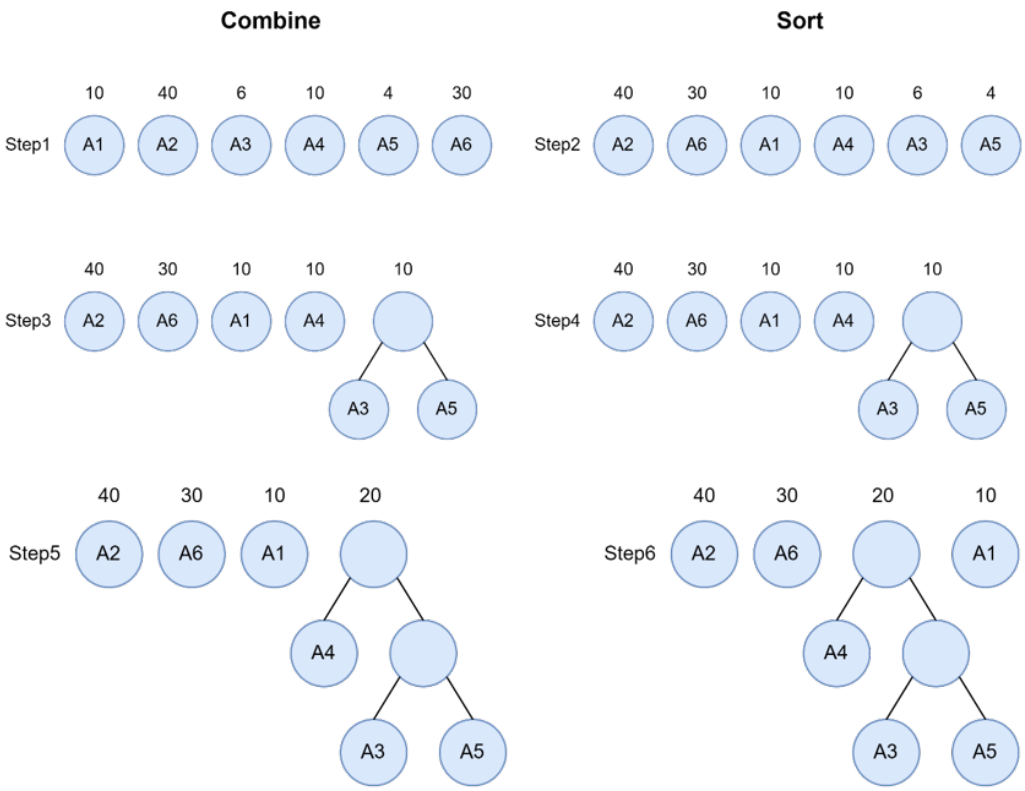
In the stage of initialization, the symbol must be sorted based on their counting number. The symbol which has largest counting number is located on the left-most side; the symbol which has smallest counting number is located on the right-most side, and so forth. Notice that the symbol with smaller index has larger weight in sorting function if there are several symbols having the same counting number. For example, A1 and A4 has the same counting number (10), but the index of A1 is 1 which has larger weight, so A1 must put on the left side of A4. The result of initialization displays on the step2 in the Fig. 1.

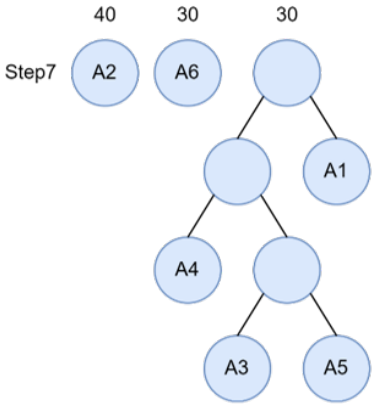
In the stage of combination, two symbols which have the smallest counting number and the second smallest one after sorting must combine together. In the first combination, A3 and A5 is combined to {A3,A5} with counting number of 10, which is depicted on the step3 in Fig. 1. After the first combination, the symbols need to re-

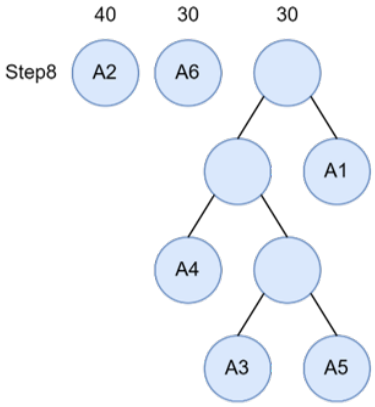
order. Notice that the combination term must have the smaller weight in sorting function if the combination term and any symbols have the same counting number. Therefore, the combination term {A3,A5} is located on the right-most side, which is shown on the step4 in Fig. 1. In the second combination, the combination term {A3,A5} and A4 is combined to {A4,A3,A5} with counting number of 20, which is illustrated on the step5 in Fig. 1. After the second combination, the symbols re-order, which is displayed on the step6 in the Fig. 1. As stated above, the combination continuing carrying out until there is only two terms left, which is indicated with the Huffman Tree as shown on step10 in the Figure. 1.

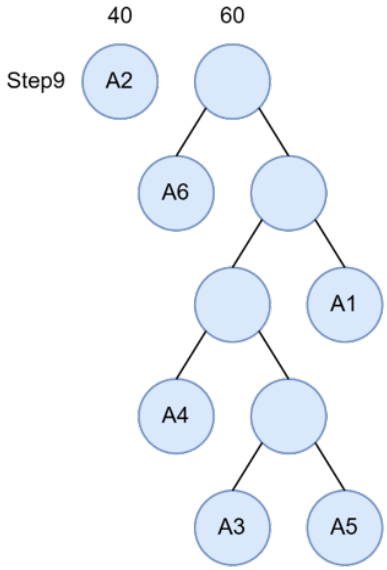
In the stage of split, the Huffman code is given for each symbol. The term on the left of node is given '0'; the term on the right of node is give '1' as illustrated on the step11 in the Figure.1. Huffman code is read from top to bottom. For example, Huffman code of A4 is 0100. In this design, the signal of **HC** is 8 bits, so the signal of **HC**(A4) is 0000_0100. Because Huffman code is variable-length code, it is need to introduce Mask, which indicated how many bits is valid. In the signal of **M**, 1 is for valid; 0 is for invalid. In the example, Huffman code of A5 is 01011, which represents the code is 5-bit length, so the signal of **M**(A5) is 0001_1111 and the signal of HC(A5) is 0000_1011.

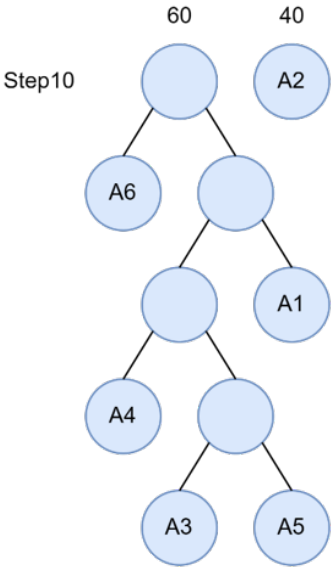
Example




Step7
40
30
30
A2
A6
A1
A4
A3
A5


Step8
40
30
30
A2
A6
A1
A4
A3
A5


Step9
40
60
A2
A6
A1
A4
A3
A5


Step10
60
40
A2
A6
A1
A4
A3
A5

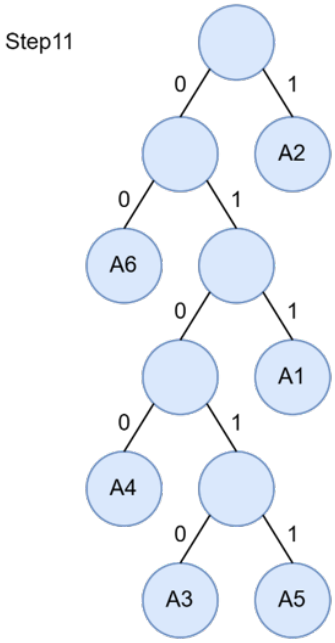

Step11
0
1
A2
A6
A1
A4
A3
A5

Figure 1. The processing of Huffman encoding, including the initialization, combination, split, which is explained with Huffman Tree.

Input

The input signals for SystemC are as follows:

Input Signal	Bit Width	Description
in_Aid_all	48	Concatenation signal of indexes of six symbols from Verilog.
in_CNT_all	48	Concatenation signal of counting result of six symbols from Verilog.

1. Both **in_Aid_all** and **in_CNT_all** are totally 48 bits. Every eight bits for one symbol.
2. The signal of **in_Aid_all** is for indexes; the signal of **in_CNT_all** is for counting result.

The input signals for Verilog are as follows:

Input Signal	Bit Width	Description
clk	1	Clock signal
reset	1	Asynchronous active-high reset
gray_data	8	Grayscale image data bus.
gray_valid	1	Be high when gray-data are valid.
out_Aid_all	48	Indexes of six symbols output bus from SystemC.
out_CNT_all	48	Counting result of six symbols output bus from SystemC.

1. When **gray_valid** is high, the 8-bits **gray_data** is delivered with **100 cycles continuously**.
2. When **gray_valid** is low after 100 cycles, the **gray_data** is tied to unknown state.
3. There are **only six symbols** for **gray_data**: A1, A2, A3, A4, A5, and A6.
4. All input signals are synchronized at **negative edge** of the clock.

Output

The output signals for SystemC are as follows:

Input Signal	Bit Width	Description
out_Aid_all	48	Indexes of six symbols output bus for Verilog input.

out_CNT_all	48	Counting result of six symbols output bus for Verilog input.

1. Both **out_Aid_all** and **out_CNT_all** are totally 48 bits. Every eight bits for one symbol.
2. The signal of **out_Aid_all** is for indexes; the signal of **out_CNT_all** is for counting number.
3. The MSB of **out_Aid_all** is for index of symbol whose counting number is the largest one; the LSB of **out_Aid_all** is for index of symbol whose counting number is the smallest one.
4. The MSB of **out_CNT_all** is for counting number of symbol whose counting number is the largest one; the LSB of **out_CNT_all** is for counting number of symbol whose counting number is the smallest one.

The output signals for Verilog are as follows:

Output Signal	**Bit Width**	**Description**
CNT	48	The counting number of gray image data. The first eight bits are for symbol A1; the second eight bits are for symbol A2, and so forth.
CNT_valid	1	Be high when output signals of CNT are valid.
HC	48	The Huffman code for every symbol. The first eight bits are for symbol A1, the second eight bits are for symbol A2, and so forth.
M	48	The Huffman coding mask for every symbol. The first eight bits are for symbol A1, the second eight bits are for symbol A2, and so forth.
code_valid	1	Be high when output signals of HC and M are valid.
in_Aid_all	48	Concatenation signal of indexes of six symbols for SystemC input.
in_CNT_all	48	Concatenation signal of counting result of six symbols for SystemC input.

1. The **CNT_valid** and **code_valid** must be **low** with the **asynchronous reset** when reset is **high**.
2. The **CNT_valid** and **code_valid** must be high for **only 1 cycle**.
3. The signals of **CNT** must be delivered for **1 cycle at the same time** and **CNT_valid** should be **high** simultaneously.

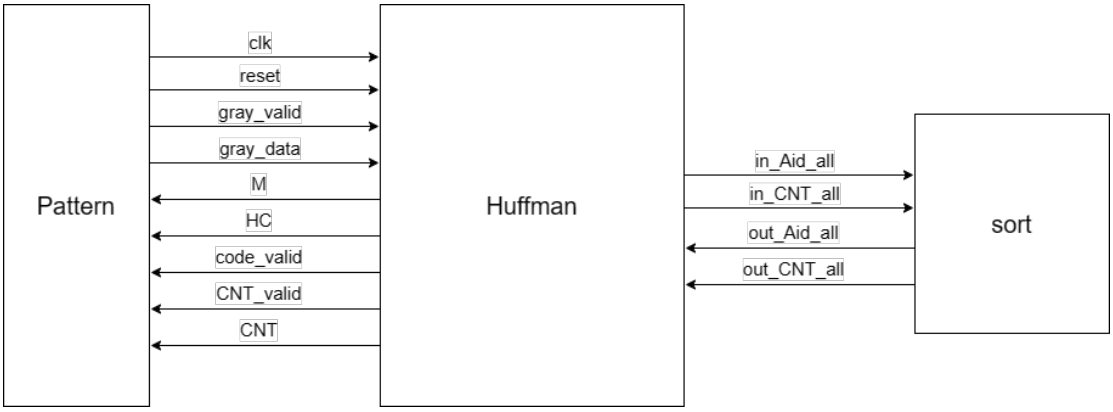
4. The signals of **HC and M** must be delivered for **1 cycle at the same time** and **code_valid** should be **high** simultaneously.
5. The signal of **CNT** recommends being **zero** when **CNT_valid** is **low**.
6. The signals of **HC and M** recommend being **zero** when **code_valid** is **low**.
7. The **CNT_valid and code_valid** cannot overlap with **gray_valid** at any time.

SPEC

1. SystemC
   i. Top module name: sort (File name: sort.cpp and sort.h)
   ii. Your design of sorting must be stable.
   iii. This sorting function is for six symbols sorting; don't change the bits of I/O for different cycle.

2. Verilog
   i. Top module name: Huffman (File name: Huffman.v)
   ii. It is **positive edge trigger** architecture.
   iii. It is **asynchronous reset** and **active-high** architecture. If you use synchronous reset (considering reset after clock starting), you may fail to reset signals.
   iv. The reset signal (reset) would be given only once at the beginning of simulation. output signals for valid should be reset after the reset signal is asserted.
   v. The execution latency is limited in **1000 cycles**. The latency is the clock cycles between the falling edge of the **gray_valid** and the rising edge of the **code_valid**.
   vi. The **code_valid** **cannot** be high before **CNT_valid** becomes high once. However, both of them can be high at the same time.
   vii. The synthesis result of data type **cannot** include any **latches** (using ctrl+F to find the term of "Latch" or using the command **./08_check** in 02_SYN/**).**
   viii. After synthesis, you can check Huffman.area and Huffman.timing. The timing report should be **non-negative (MET)**.
   ix. After gate-level simulation, the file of vcs.log cannot include any **timing violation**.

Block Diagram



Homework Upload

1. Please use the commands we provided to tar whole file under 09_SUBMIT/ and to submit your homework **before 23:59 on 5/21 (Tue.)**.
   A. If you don't do it before 23:59 on 5/21 (Tue.), you still can hand in your homework **before 23:59 on 5/28 (Tue.)**, but your **final score is 50% off of your original score (Final = 50% x Original)**.
      i. There is no partial submission. One of your homework is submitted after 5/21 but before 5/28, your **final score is 50% off of your original score (Final = 50% x Original)**.
      ii. The grade of Verilog and SystemC is based on original, but the grade of PA is 50% off.
   B. If homework isn't handed in **after 23:59 on 5/28 (Tue.)**, the score is **zero point**.
2. Please submit the screenshot of Platform Architecture (PA) to the new E3. The name of your file is **HW03_DCSXXX_PA.jpg (XXX is your account ID)**. If the file violates the naming rule, **10 points will be deducted.**
3. Please check the homework file again after you upload it. If you upload the wrong file, you will **fail** this homework.

Grading Policy

1. SystemC Function Validity: total 20%
   - Sorting function: 20%
2. Verilog Function Validity: total 60%
   - RTL, SYN and GATE: 60%
     - The score **is only based on** the demo result you submit with 09_SUBMIT. Please ensure that you successfully upload the latest version.
     - Please check whether there is any wire/reg/submodule being named as "error", "fail", "pass", "congratulations", "latch". All letters used in the

formats of uppercase or lowercase is prohibited. If there is, you will **fail** this homework.

3. Platform Architect Function Validity: total 20%
   - Co-simulation - interconnection between Verilog and SystemC: 20%
     - The screenshot of result with the simulation of Platform Architect (PA) and your account information must be submitted to the new E3, or you will **fail** 20% in this homework.

Command List

- Environment preparation for SystemC, Cell-based EDA tool, Platform Architect:
  - **yes**
  - **source /RAID2/cad/synopsys/CIC/pa_virtualizer.cshrc**
- SystemC: Compile and Run (1_SystemC/):
  - **make**
- SystemC: Clear (1_SystemC/):
  - **make clear**
- Verilog RTL simulation (01_RTL/):
  - **./01_run_vcs_rtl**
- Synthesis (02_SYN/):
  - **./01_run_dc_shell**
  - **./08_check**
- Gate level simulation (03_GATE/):
  - **./01_run_vcs_gate**
- Submit your files (09_SUBMIT/):
  - **./00_tar 20**
    - Cycle time is 20 in this design. Don't modify it in this homework.
  - **./01_submit**
  - **y**
    - You must type '**y**' when you ensure you are ready to hand in your homework.
  - **./02_check**
- Waveform for debug:
  - **nWave &**
  - find *.fsdb
  - shift+L for reloading the *.fsdb file after you simulate your design again.
- PA: Preparation and Perform (3_PA/):
  - **./01_copy**
    - Copy your designs into the 3_PA/.

- **./09_clean**
  - Clear your designs and all PA files from the 3_PA/.
- **pct &**

Sample Waveform

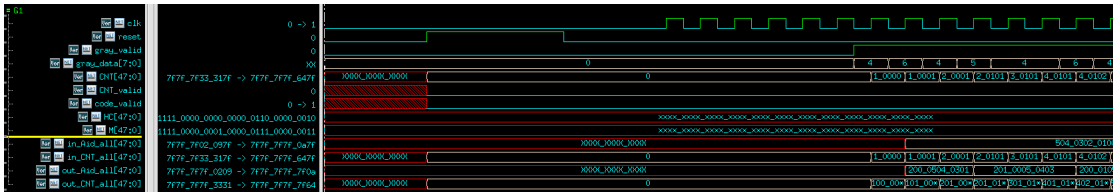


Figure 2. The waveform of Huffman coding at the start of input.

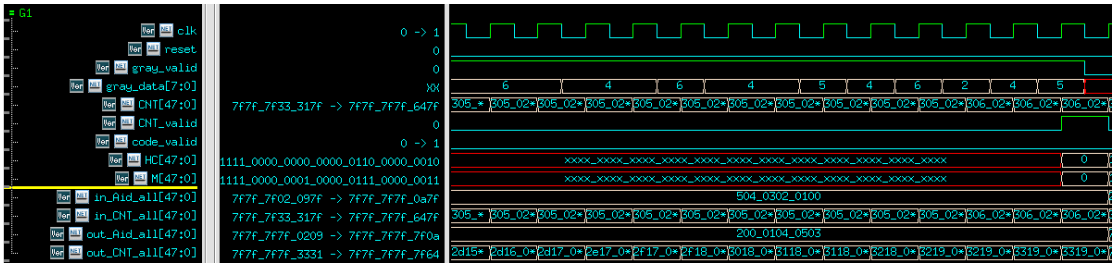


Figure 3. The waveform of Huffman coding at the end of input.

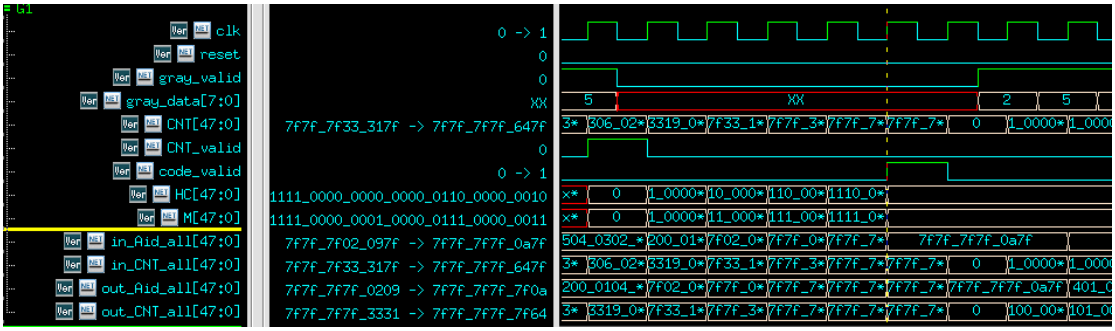


Figure 4. The waveform of Huffman coding for output.

Note

1. The sorting method isn't limited in any way, but we recommend **using bubble sort**.
   I. Notice that the MSB of output signal is for the symbol whose counting number is the largest one; in other words, the symbol with the largest value of CNT must put on the left-most of output signals.
   II. Notice that your sorting function must **be stable**.
2. **We strongly recommend that you encode the Huffman code for every symbol when you combine any two terms.**
   I. In our previous example, A3 and A5 is combined firstly, and CNT of A3 is larger than one of A5, so the last bit of HC(A3) is given 0; the last bit of HC(A5) is given 1. In the second combination, A4 and {A3,A5} is

combined, and the weight of A4 is larger than one of {A3,A5}, so the last bit of HC(A4) is 0; the last but one bits of HC(A3) and HC(A5) is 1.