2022 Digital IC Design

Final Exam

Meeting Room Number：1

Please check if the last digit of your student id is equal to the meeting room number. Each meeting room has different exam questions. TAs will verify your code according to your student id, the mismatch between student id and meeting room number may cause you to lose part of the score.

Question 1：(50%)

In Question 1, you are requested to design a Basic Operation Engine (BOE). The BOE module will take a series of data as input, and finding the summation, minimum and decreasing sequence of them. The specification and function of BOE circuit will be described in detail in the following section.

1. Design Specifications:

1.1 Block Overview

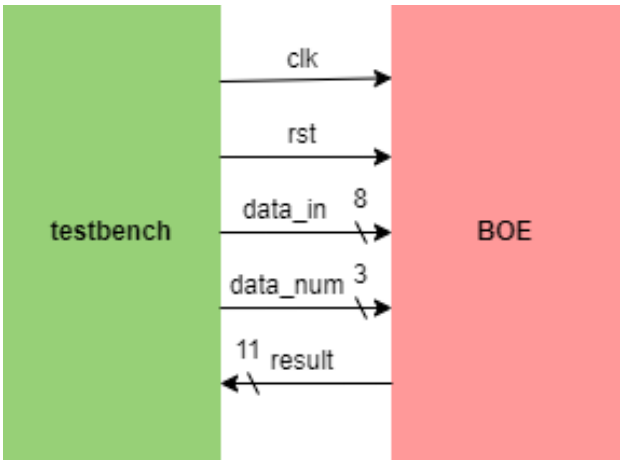

Fig. 1. The block overview.

1.2 I/O Interface

Signal Name	I/O	width	Description
clk	I	1	This circuit is a synchronous design triggered at the positive edge of clk.
rst	I	1	Active-high asynchronous reset signal.
data_in	I	8	A series of 8-bit data input port.
data_num	I	3	The number of the input data in a series. The range of *data_num* is from 2 to 6.
result	O	11	The result of basic operations of a series of data.

1.3 File Description

File Name	Description
BOE.v	The module of basic operation engine, which is the top module in this design.
BOE_tb.sv	The testbench file. The content in this file is **not allowed** to be modified.

2. Timing Specifications

After the system reset (T1), the number of the input data in a series will be input by *data_num* port. Meanwhile, the data in a series will be input by *data_in* port within *data_num* cycles. At T2, all the data is input. Then you must output the operation result from *result* port. At T3, the sum of the series should be output. The minimum of the series should be output at T4. From T5, the sequence of the data will be output from big to small for *data_num* cycles. At T6, a new series of data will be input, and the BOE will start a new round of operations.

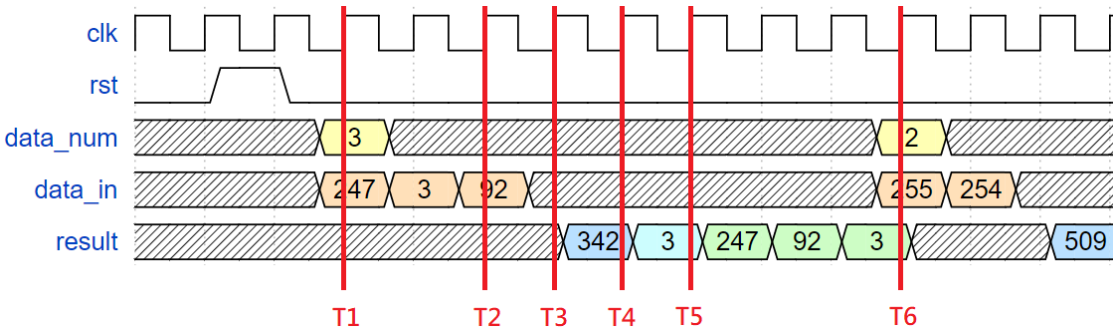

Fig. 2. Timing diagram of data input and result output.

Fig. 3 shows the finite state machine of the BOE. You can complete the BOE module according to Fig. 3, or design your own state machine.

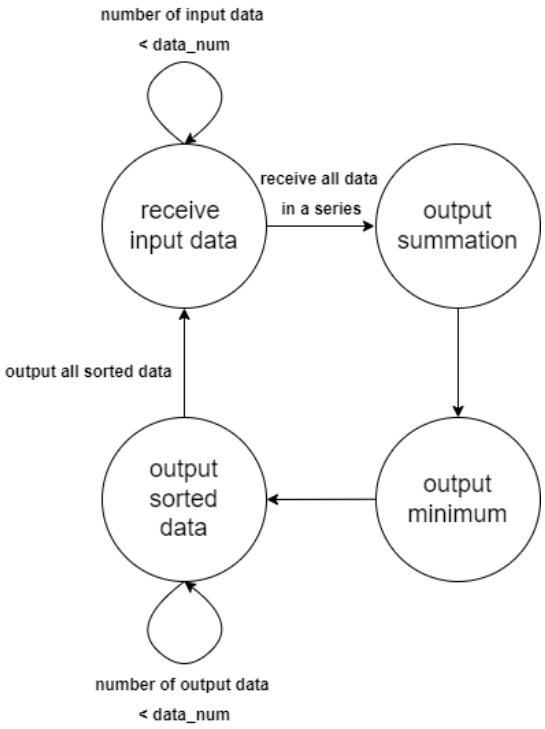

Fig. 3. Finite state machine of the BOE.

Question 2：(30%)

- **LZ77 Algorithm :**

LZ77 is a lossless data compression algorithm. It is a dictionary coder.

LZ77 has two main characteristics:

1. It makes use of encoded input sequence as dictionary.
2. Using a sliding window to encode data. Sliding window is composed of search buffer and look-ahead buffer. Search buffer saved encoded input sequence, and look-ahead buffer store sequence to be encoded.

The following are the steps of LZ77 encoding algorithm:

1. Moving the pointer to data in search buffer until equal to the first character in look-ahead buffer. Then check the next character both in search buffer and look-ahead buffer. If they are equal, check the next character again until they are not equal. The total number of equal characters is called length of match.
2. Repeating step 1 until determining the longest one in all matched sequences.
3. Output the encoding result [*offset*, *match_len*, *char_nxt*].

i. *offset* : The offset between the position of the first character in the matched string to the head of the search buffer.

ii. *match_len*: The length of the longest matched sequence.

iii. *char_nxt*: The character behind the longest matched sequence in look-ahead buffer.

4. Move *match_len* + 1 characters from look-ahead buffer to search buffer.
5. Repeating step 1 to step 4 until all of the data are encoded.

✧ The decoding algorithm is the opposite of encoding algorithm.

We provide a simple example of LZ77 algorithm: There is a sequence, " 112a112a2112112112a21$ ", assumed that the size of search buffer is 9 characters long, look-ahead buffer is 8 characters long. "$" is the end of input sequence. The following are encoding process:

	search buffer	look-ahead buffer	input sequence	matched string	encode result
step1		112a112a	2112112112a21$		[0, 0, 1]
step2	1	12a112a2	112112112a21$	1	[0, 1, 2]
step3	112	a112a211	2112112a21$		[0, 0, a]
step4	112a	112a2112	112112a21$	112a	[3, 4, 2]
step5	112a112a2	11211211	2a21$	112	[8, 3, 1]
step6	112a21121	12112a21	$	12112	[2, 5, a]
step7	12112112a	21$		21	[7, 2, $]

The result of encoding:

{ [0, 0, 1] 、[0, 1, 2] 、[0, 0, a] 、[3, 4, 2] 、[8, 3, 1] 、[2, 5, a] 、[7, 2, $] }

Notice that, in step6, because the sliding window of LZ77 algorithm is composed of search buffer and look-ahead buffer, the character to be encoded can be searched not only in the search buffer but also in the look-ahead buffer. The decoding process is the opposite of encoding process:

	decode input	decode result	search buffer	look-ahead buffer
step1	[0, 0, 1]	1	1	
step2	[0, 1, 2]	12	112	
step3	[0, 0, a]	a	112a	
step4	[3, 4, 2]	112a2	112a112a2	
step5	[8, 3, 1]	1121	112a21121	??
step6	[2, 5, a]	12112a	12112112a	
step7	[7, 2, $]	21	112112a21	

In Question2, you need to design a LZ77 algorithm encoder (Question 2-1) and

decoder (Question 2-2). The search buffer is 7 characters long, and look-ahead buffer is 3 characters long. The algorithm you need to design is the same as homework3. You can complete Question 2-1 and 2-2 by modifying your homework file or the example provided on Moodle.

Question 2-1：(encoder : 15%)

1. Design Specifications:

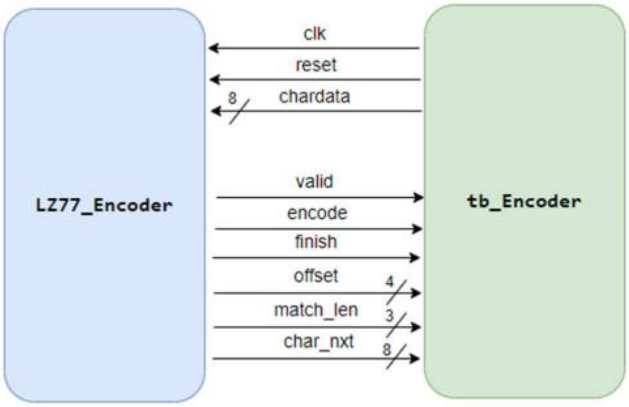

Fig. 4. Encoder block overview.

2. I/O Interface:

Encoder signal table

Signal Name	I/O	Width	Description
clk	I	1	This circuit is a synchronous design triggered at the positive edge of *clk*.
reset	I	1	Active-high asynchronous reset signal.
chardata	I	8	Character to be encoded.
valid	O	1	When output encoding result, set *valid* signal to high. The testbench will check the output result when *valid* signal is high.
encode	O	1	When encoding, set *encode* signal to high.
offset	O	4	The offset between the position of the first character in the matched string to the head of the search buffer.
match_len	O	3	The length of matched string.
char_nxt	O	8	The next character behind last matched character in look-ahead buffer.
finish	O	1	When the encoding process finishes, set *finish* signal to high.

Table1. Encoder I/O

3. File Description:

File Name	Description
Q2_Encoder.v	The module of LZ77_Encoder, which is the top module in this design
tb_Encoder_Q2.sv	Encoder testbench. The content is not allowed to be modified.

4. Function Description:

The input and output rule are the same as homework3, the input is a grayscale image sequence which is 32x32x2+1 long (each pixel value will be sent in two continuously cycle, e.g. pixel value=8'h1a, then the testbench will send 8'h01 and 8'h0a in continuously cycle). And there will be a symbol "$" in the end of input sequence.

In encoder module, you need to save all input characters before encoding them. When *chardata* is "$" (8'h24), it is the end of input sequence. The *encode*, *valid* signal should be set to high before outputting the encode result (*offset*, *match_len*, *char_nxt*). Otherwise, verification might fail. After finishing encoding, set *finish* signal to high. The testbench will terminate the simulation after the *finish* signal is pulled up. Don't pull up *finish* signal at the same time that the last set of encoding results is output.

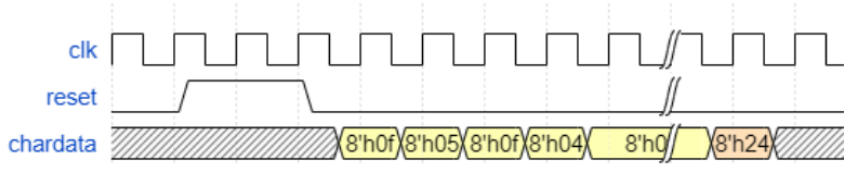

Fig. 5. Input image data.

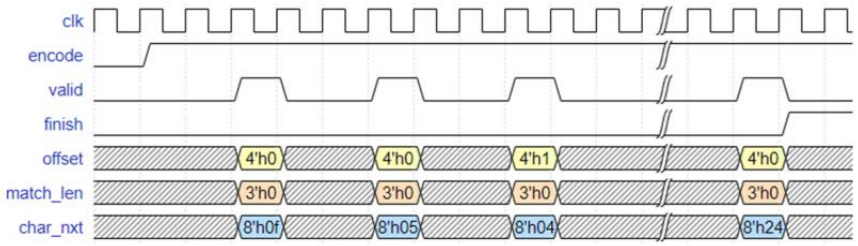

Fig. 6. Encoding output.

Question 2-2：(decoder : 15%)

1. Design Specifications:

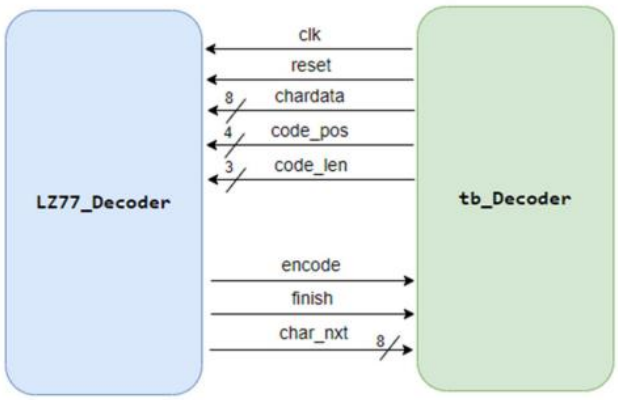

Fig. 7. Decoder block overview.

2. I/O Interface:

Decoder signal table

Signal Name	I/O	Width	Description
clk	I	1	This circuit is a synchronous design triggered at the positive edge of *clk*.
reset	I	1	Active-high asynchronous reset signal.
code_pos	I	4	The begining position of the matched string in the search buffer.
code_len	I	3	The length of matched string.
chardata	I	8	The next character behind matched string.
encode	O	1	When decoding, set *encode* signal to low.
char_nxt	O	8	Decoded character.
finish	O	1	When the decoding process finishes, set *finish* signal to high.

Table2. Decoder I/O

3. File Description:

File Name	Description
Q2_Decoder.v	The module of LZ77_Decoder, which is the top module in this design
tb_Decoder_Q2.sv	Decoder testbench. The content is not allowed to be modified.

4. Function Description:

In the decoder module, the testbench will send decoding data (*code_pos*, *code_len*, *chardata*) immediately when reset finished, and you should output the decoded result(*char_nxt*). The output timing is as below. The *encode* signal should be low and set *finish* signal to high after finishing decoding. Don't pull up *finish* signal at the same time that the last decoded character is output.

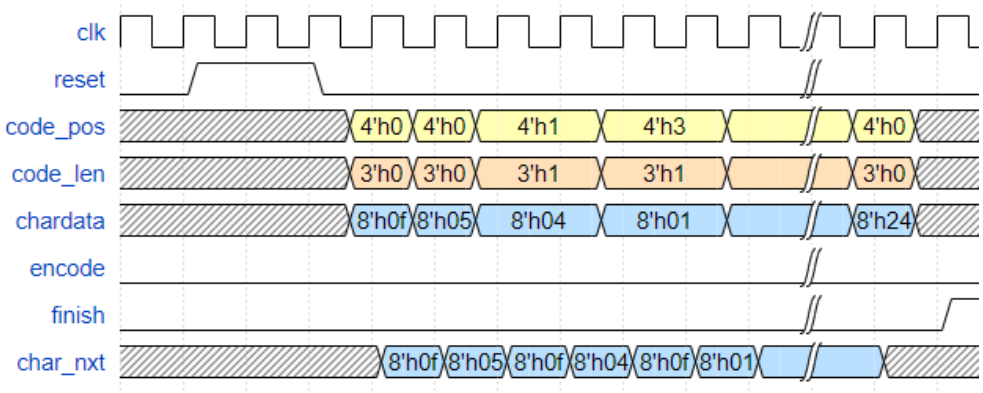

Fig. 8. Decoding output.

Question 3：(10%)

In Question3, you need to fill the blank blocks in the given file to make the encoder function correct. The given file is similar to the homework 3 example provided on Moodle, only the design of the state machine is different. The search buffer is 9 characters long, and look-ahead buffer is 8 characters long.

The blank blocks are in states **ENCODE_NOT_MATCH**, **ENCODE _MATCH**, and the combinational block. You can complete the codes with conditional operator (? :) or rewrite the codes with if-else statements. However, only the values of the given signals can be changed. In **ENCODE_NOT_MATCH** state, only the value of *search_index* can be changed. In **ENCODE _MATCH** state, only the values of *char_nxt*, *match_len*, *offset*, and *lookahead_index* can be changed. In the combinational block, only the value of *next_state* can be changed.

You have to complete the conditional judgments with already declared signals. Don't declare any new signal or modify the content of other states containing no blank blocks. Otherwise, you will lose the score of this part. The encoder block, I/O interface and timing diagram are the same as question2-1 and homework 3.

```
ENCODE_NOT_MATCH:
begin
    search_index     <= /* fill in here */;
end
ENCODE_MATCH:
begin
    char_nxt         <= /* fill in here */;
    match_len        <= /* fill in here */;
    offset           <= /* fill in here */;
    lookahead_index <= /* fill in here */;
end
```

Fig. 9. Blank blocks in states **ENCODE_NOT_MATCH** and **ENCODE_MATCH**.

```
always @(*)
begin
    case(current_state)
        IN:
        begin
            next_state = /* fill in here */;
        end
        ENCODE_NOT_MATCH:
        begin
            next_state = /* fill in here */;
        end
        ENCODE_MATCH:
        begin
            next_state = /* fill in here */;
        end
        ENCODE_OUT:
        begin
            next_state = /* fill in here */;
        end
        SHIFT_ENCODE:
        begin
            next_state = /* fill in here */;
        end
        default:
        begin
            next_state = /* fill in here */;
        end
    endcase
end
```

Fig. 10. Blank blocks in the combinational block.

1. File Description:

File Name	Description
Q3_Encoder.v	The module of LZ77_Encoder, which is the top module in this design
tb_Encoder_Q3.sv	Encoder testbench. The content is not allowed to be modified.

Question 4：(10%)

In Question 4, you need to design a LZ77 circuit including encode and decode functions at the same time. Moreover, your circuit has to be able to process input data continuously. After testbench send over whole input sequence, you should encode them and output the encoding results. Testbench will record your encoding results simultaneously, which will be the input in the decode stage. The decode stage starts after the last set of encoding result is output. When the decode stage finishes, the next input sequence will be sent by testbench. Three sets of input image data are used in this Question. The search buffer is 9 characters long, and look-ahead buffer is 8 characters long.

1. Design Specifications:

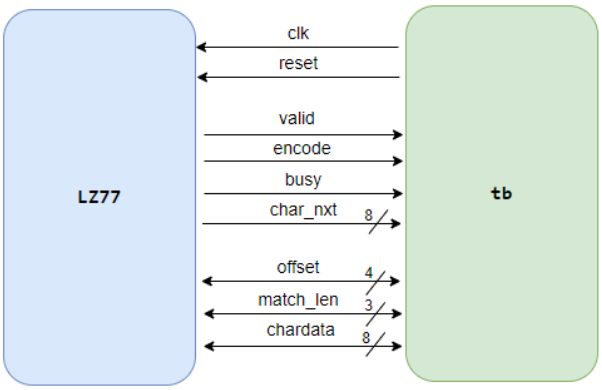

Fig. 11. LZ77 block overview.

2. I/O Interface:

Signal Name	I/O	Width	Description
clk	I	1	This circuit is a synchronous design triggered at the positive edge of clk.
reset	I	1	Active-high asynchronous reset signal.
chardata	IO	8	This signal is bidirectional. When it acts as an input signal, you should not output data from it. Otherwise, it might cause unknown result. Input sequence stage: It should be used as an input port for inputting sequence characters from testbench. Encode stage: It should be used as an output port for outputting the next character behind

			the last matched character in the look-ahead buffer. Decode stage: it should be used as an input port for inputting the next character behind matched string from testbench.
valid	O	1	When output encoding or decoding result, set valid signal to high. Testbench will check the output result when this signal is pulled up.
encode	O	1	When encoding, set encode signal to high. When decoding, set encode signal to low.
offset	IO	4	This signal is a bidirectional design. In encode stage, it should be used as an output signal, which is the position of the first character in the matched string to the head of the search buffer. In decode stage, it should be used as an input signal, which is the position of first character in matched string.
match_len	IO	3	This signal is a bidirectional design. In encode stage, it should be used as an output signal, which is the length of matched string. In decode stage, it should be used as an input signal, which is the length of matched string.
char_nxt	O	8	The decoded character in decode stage.
busy	O	1	When finished reading input sequence from testbench, set *busy* signal to high. It means your circuit is in encoding or decoding stage. After finished decoding, set busy to low, and then testbench will send next input sequence.

Table.3. LZ77 I/O

3. File Description:

File Name	Description
LZ77.v	The module of LZ77, which is the top module in this design
tb.sv	LZ77 testbench. The content is not allowed to be modified.

4. Function Description:

In this module, you need to deal with three input sequence continuously by controlling *busy* signal. When an input sequence is sent over from testbench, the *busy* signal should be set to high. Testbench will stop sending next input sequence. After finishing decoding stage, the *busy* signal should be set to low. Then testbench will send next input sequence. The timing specifications is as below.

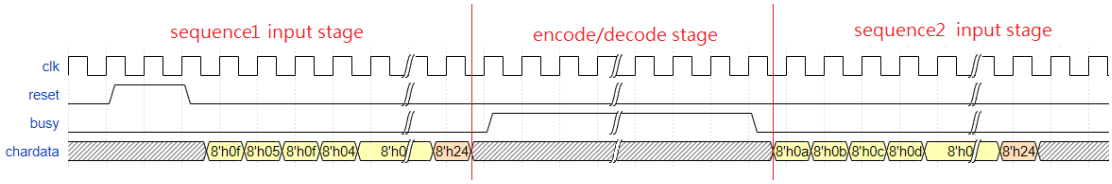

Fig. 12. Timing diagram of *busy* signal.

In input sequence stage, the specification is the same as the encoder side of homework3. The input sequence is 32x32x2+1 long (each pixel value will be sent in two continuously cycle, e.g. pixel value=8'h1a, then tb will send 8'h01 and 8'h0a in continuously cycle). Before encode stage, you need to save all input characters. When *chardata* is "$" (8'h24), the input sequence is end.

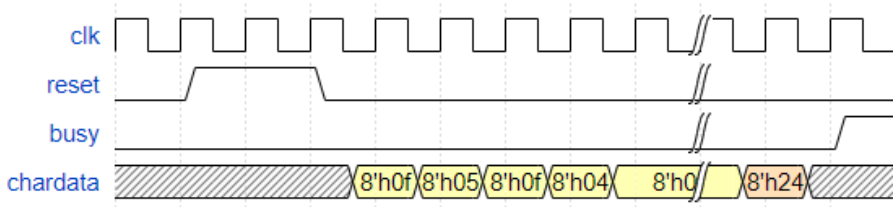

Fig. 13. Timing diagram for inputting image data.

In the encode stage, *valid* signal should be set to high when outputting the encode result (*offset*, *match_len*, *chardata*). Otherwise, verification might fail.

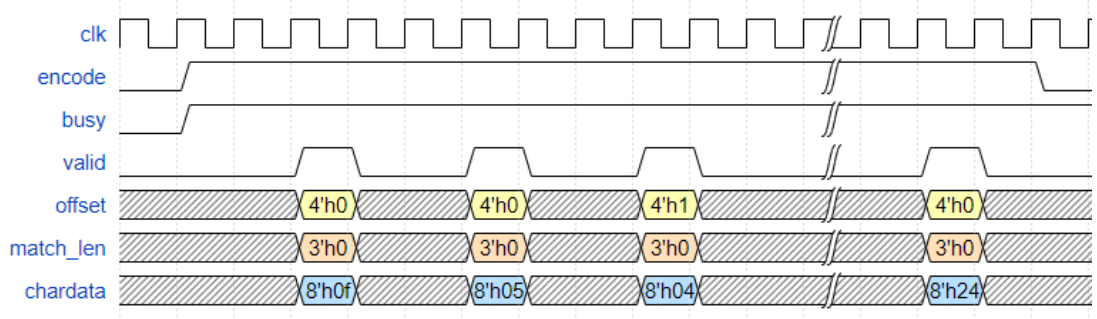

Fig. 14. Timing diagram of the encode stage.

In the decode stage, testbench will send decoding data (*offset*, *match_len*, *chardata*), and you should output the decoded result (*char_nxt*). The output timing diagram is as below. The *encode* signal should be low and *valid* signal should be set to high when outputting the decoded result. After a valid decoded result is output, testbench will send the next decoding data.

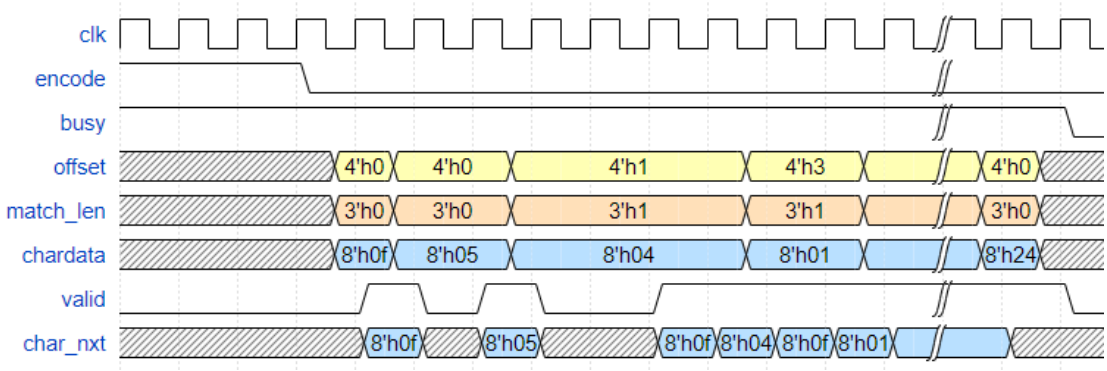

Fig. 15. Timing diagram of the decode stage.

After finished encode and decode three input sequence, testbench will display the final result and terminate the simulation.

- Inout port (Bidirectional signal): In Question 4, there are three bidirectional signals (*chardata*, *offset*, *match_len*). You might use the below verilog statement to control the behavior of the port.

  reg $output\_reg$;
  wire $enable$;
  assign bidirectional_signal = ($enable$) ? $output\_reg$: $'bz$;

  When the *enable* signal is high, the bidirectional signal acts as output port. The value of *output_reg* will be outputted to testbench. When the *enable* signal is low, the bidirectional signal acts as input port. It will be set as high impedance (*'bz*) to receive the data from testbench.

Scoring:

In this exam, you only need to pass the functional simulation. The scoring of each part is as follows:

1. Question 1: [50%]

1.1 Summation function testing [20%]

In stage 1, testbench will verify the summation function of your design. If the

summation of each series of data are all generated correctly, you will get the following message in ModelSim simulation.

```
------------------------------------------------
--          Stage 1 Simulation finish          --
--               stage 1  : PASS!              --
------------------------------------------------
```

Fig. 16. Simulation result of Question 1 in stage 1.

1.2 Minimum function testing **[20%]**

In stage 2, testbench will verify the minimum function of your design. If the minimum of each series of data are all generated correctly, you will get the following message in ModelSim simulation.

```
------------------------------------------------
--          Stage 2 Simulation finish          --
--               stage 2  : PASS!              --
------------------------------------------------
```

Fig. 17. Simulation result of Question 1 in stage 2.

1.3 Sorting function testing **[10%]**

In stage 3, testbench will verify the sorting function of your design. If the sorted result of each series of data are all generated correctly, you will get the following message in ModelSim simulation.

```
------------------------------------------------
--          Stage 3 Simulation finish          --
--               stage 3  : PASS!              --
------------------------------------------------
```

Fig. 18. Simulation result of Question 1 in stage 3.

For Question 2-1, Question 2-2, and Question 3, you should test all three patterns in testbench. You need to pass all of three patterns, or you will get no score.

```
`define PAT         "./img0/testdata_encoder.dat"
// `define PAT         "./img1/testdata_encoder.dat"
// `define PAT         "./img2/testdata_encoder.dat"
```

2. Question 2-1: [15%]

In this question, testbench will verify the LZ77 encoder with 7 characters long search buffer and 3 characters long look-ahead buffer. If the simulation is correct, you will get the following message in ModelSim simulation.

```
------------------------------------------------
--------- Q2-1 Encoding finished, ALL PASS ----------
------------------------------------------------
```

Fig. 19. Simulation result of Q2-1

3. Question 2-2: [15%]

In this question, testbench will verify the LZ77 decoder with 7 characters long search buffer and 3 characters long look-ahead buffer. If the simulation is correct, you will get the following message in ModelSim simulation.

```
------------------------------------------------
--------- Q2-2 Decoding finished, ALL PASS ----------
------------------------------------------------
```

Fig. 20. Simulation result of Q2-2

4. Question 3: [10%]

In this question, testbench will verify the LZ77 encoder with 9 characters long search buffer and 8 characters long look-ahead buffer. If the simulation is correct, you will get the following message in ModelSim simulation.

```
------------------------------------------------
------ Q3 Encoding finished, ALL PASS ---------
------------------------------------------------
```

Fig. 21. Simulation result of Q3

5. Question 4: [10%]

In this question, testbench will verify the LZ77 module with 9 characters long search buffer and 8 characters long look-ahead buffer. If the simulation is correct, you will get the following message in ModelSim simulation.

```
------------------------------------------------
--              Simulation finish             --
--              IMAGES 1 All PASS             --
--              IMAGES 2 All PASS             --
--              IMAGES 3 All PASS             --
------------------------------------------------
```

Fig. 22. Simulation result of Q4

Submission:

1. Submitted files

You should classify your files into four directories and compress them to .zip format. The naming rule is final_studentID_name.zip. If your file is not named according to the naming rule, you will lose five points. Please submit the compressed file to folder Final in moodle.

final_studentID_name.zip	
Q1	
*.v	All of your Verilog RTL code for Q1
Q2-1	
*.v	All of your Verilog RTL code for Q2-1
Q2-2	

*.v	All of your Verilog RTL code for Q2-2
Q3	
*.v	All of your Verilog RTL code for Q3
Q4	
*.v	All of your Verilog RTL code for Q4

2. Note

Please do not modify any content of testfixture, excluding the patterns of Question 2-1, Question 2-2, and Question 3.

Deadline: 2022/5/24 12:00.

No late submissions will be accepted, please upload your code in time. You can upload your answer file multiple times, so don’t upload your file at a very close time to the deadline.