# 2022 Digital IC Design

# Homework 3: LZ77 Encoder and Decoder

## 1. Introduction

**LZ77** is a lossless data compression algorithm. In this homework, you need to design a LZ77 image compression encoder and a LZ77 decompression decoder. The following section will describe the details of this homework. Three different image test data are provided to verify the correctness of your design. **The cycle in the testbench is set as 30 ns, which can't be modified. You have to pass all three test data to get the score of each part.**

## 2. Design Specifications

### 2.1 Block overview

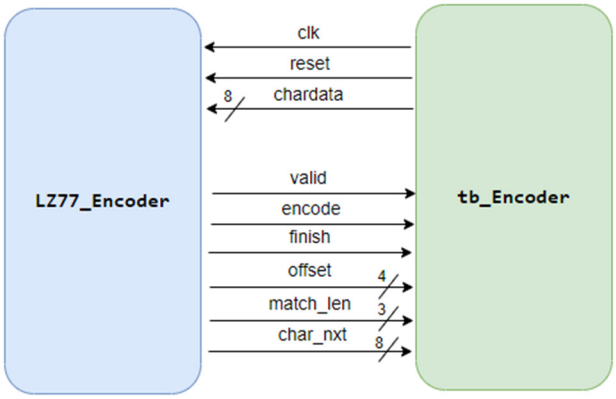



Fig.1 Encoder block overview

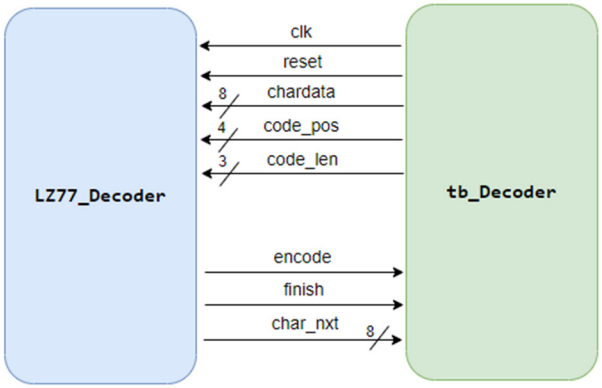



Fig.2 Decoder block overview

## 2.2 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

### 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

## 2.3 File Description

| File Name | Description |
|---|---|
| LZ77_Encoder.v | The module of LZ77_Encoder, which is the top module in this design |
| LZ77_Decoder.v | The module of LZ77_Decoder, which is the top module in this design |
| tb_Encoder.sv | Encoder testbench. The content is **not allowed** to be modified. |
| tb_Decoder.sv | Decoder testbench. The content is **not allowed** to be modified. |

Table3. file description

## 2.4 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 |
|---|---|---|---|---|---|
| **Round1** | | 112a112a | 2112112112a21$ | | [0, 0, 1] |
| **Round2** | 1 | 12a112a2 | 112112112a21$ | 1 | [0, 1, 2] |
| **Round3** | 112 | a112a211 | 2112112a21$ | | [0, 0, a] |
| **Round4** | 112a | 112a2112 | 112112a21$ | 112a | [3, 4, 2] |
| **Round5** | 112a112a2 | 11211211 | 2a21$ | 112 | [8, 3, 1] |
| **Round6** | 112a21121 | 12112a21 | $ | 12112 | [2, 5, a] |
| **Round7** | 12112112a | 21$ | | 21 | [7, 2, $] |

Table4. encoding steps

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 method is the opposite of the encoding process :

| | decode input | decode result | search buffer | look-ahead buffer |
|---|---|---|---|---|
| **Round1** | [0, 0, 1] | 1 | 1 | |
| **Round2** | [0, 1, 2] | 12 | 112 | |
| **Round3** | [0, 0, a] | a | 112a | |
| **Round4** | [3, 4, 2] | 112a2 | 112a112a2 | |
| **Round5** | [8, 3, 1] | 1121 | 112a21121 | ?? |
| **Round6** | [2, 5, a] | 12112a | 12112112a | |
| **Round7** | [7, 2, $] | 21 | 112112a21 | |

Table5. decoding steps

## 2.5 Function Description

In this homework, you need to design LZ77 encoder and decoder. The size of search buffer is 9 characters long, look-ahead buffer is 8 characters long. The input sequence to be encoded is a grayscale image, which length is 32x32x2+1 (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). And there will be a symbol "$" in the end of input sequence.

In the encoder module, tb will send *chardata* immediately in every cycle when reset finished. Each *chardata* is 8 bits, you need to save all input characters

**in the testbench is set as 30 ns, which can't be modified. In each part, you have to pass all these three test data. Otherwise, you won't get any score.**

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

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

## 3.1 Functional Part [60%, each of encoder/decoder is 30%]

All of the result should be generated correctly, and you will get the following message in ModelSim simulation.

```
cycle 0454e, expect(3,2,f) , get(3,2,f) >> Pass
cycle 04559, expect(0,0,6) , get(0,0,6) >> Pass
cycle 04562, expect(0,0,$) , get(0,0,$) >> Pass
-----------------------------------------------
--------- Encoding finished, ALL PASS ---------
-----------------------------------------------
** Note: $finish    : E:/DIC2022/HW3_test/tb_Encoder.sv(250)
   Time: 888150 ns  Iteration: 1  Instance: /testfixture_encoder
```

```
# cycle 00801, expect f, get f >> Pass
# cycle 00802, expect 4, get 4 >> Pass
# cycle 00803, expect f, get f >> Pass
#   == Decoding string "6"
# cycle 00804, expect 6, get 6 >> Pass
# -----------------------------------------------
# --------- Decoding finished, ALL PASS ---------
# -----------------------------------------------
# ** Note: $finish    : E:/DIC2022/HW3_test/tb_Decoder.sv(222)
#    Time: 102700 ns  Iteration: 1  Instance: /testfixture_decoder
```

## 3.2 Gate Level Part [20%, each of encoder/decoder is 10%]

### 3.2.1 Synthesis

Your code should be synthesizable. After synthesizing in Quartus, the file named *LZ77_Encoder.vo(LZ77_Decoder.vo)* and LZ77_Encoder_v.sdo (LZ77_Decoder_v.sdo) will be obtained.

**Device : Cyclone II EP2C70F896C8**

### 3.2.2 Simulation

All of the result should be generated correctly using **LZ77_Encoder.vo (LZ77_Decoder.vo)** and **LZ77_Encoder_v.sdo(LZ77_Decoder_v.sdo)**, and you will get the following message in ModleSim simulation. (There should be no setup or hold time violations.)

```
# cycle 045c8, expect(7,7,8) , get(7,7,8) >> Pass
# cycle 045d8, expect(7,7,8) , get(7,7,8) >> Pass
# cycle 045ef, expect(7,6,$) , get(7,6,$) >> Pass
# -----------------------------------------------
# --------- Encoding finished, ALL PASS ---------
# -----------------------------------------------
# ** Note: $finish    : E:/DIC2022/HW3_postsim/tb_Encoder.sv(250)
#    Time: 537120 ns  Iteration: 1  Instance: /testfixture_encoder
```

```
# cycle 00802, expect 8, get 8 >> Pass
# cycle 00803, expect 0, get 0 >> Pass
# cycle 00804, expect 8, get 8 >> Pass
# ------------------------------------------------
# --------- Decoding finished, ALL PASS ---------
# ------------------------------------------------
# ** Note: $finish    : E:/DIC2022/HW3_postsim/tb_Decoder.sv(222)
#    Time: 61620 ns  Iteration: 1  Instance: /testfixture_decoder
```

### 3.3 Performance Part [20%]

The performance is scored by the number of logic elements, memory bits, and embedded multiplier 9-bit elements your design used in gate-level simulation. The score will be decided by your ranking in all received homework.

The scoring standard: (The smaller, the better)

Total score = encoder score + decoder score

Score = Total logic elements + total memory bits
+ 9*embedded multiplier 9-bit elements

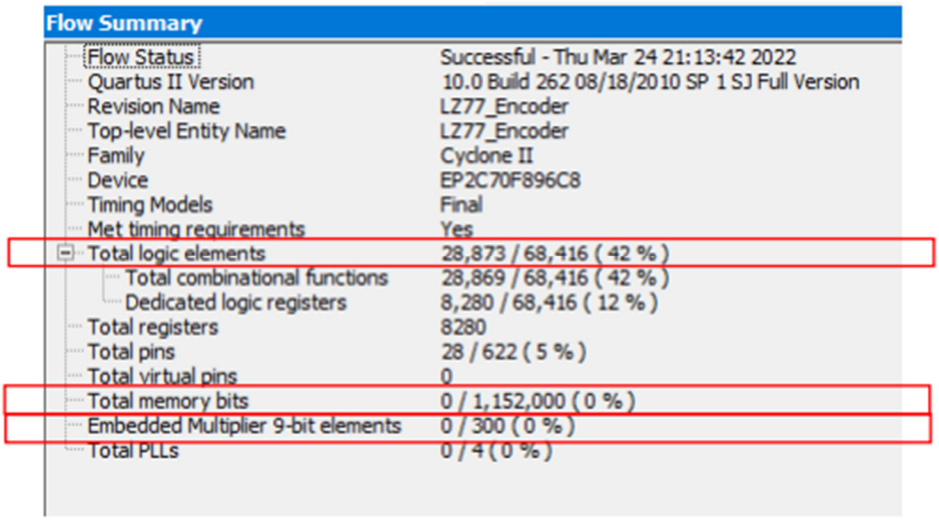
## 4. Submission

You should classify your files into three directories and compress them to .zip format. The naming rule is HW3_studentID_name.zip. If your file is not named according to the naming rule, you will lose five points.

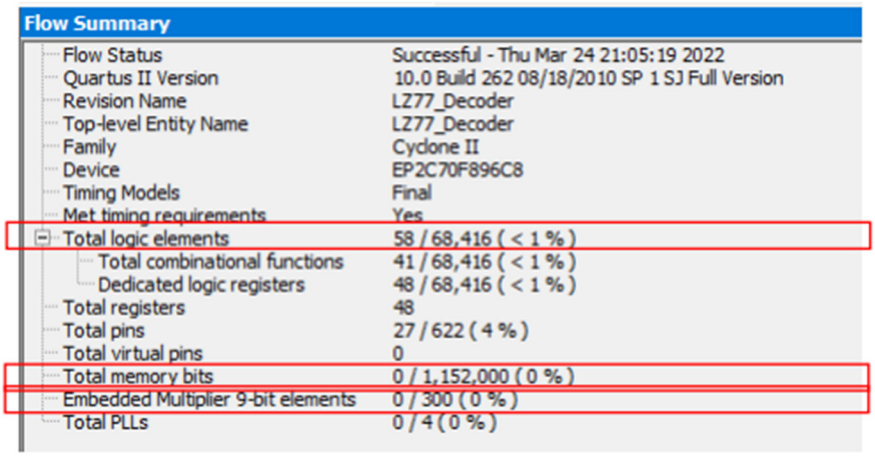
| | RTL category |
|---|---|
| *.v | All of your Verilog RTL code |
| | Gate-Level category |
| *.vo | Gate-Level netlist generated by Quartus |
| *.sdo | SDF timing information generated by Quartus |
| | Documentary category |
| *.pdf | The report file of your design (in pdf). |

### 4.1 Report file

Please follow the spec of report. If your report does not meet the spec, you may lose up to ten points. You are asked to describe how the circuit is designed as detailed as possible, and the flow summary results are necessary. Note that, you need to record all of the three pattern of gate-level simulation time in report, we will test your design with them.



| Flow Summary | |
|---|---|
| Flow Status | Successful - Thu Mar 24 21:13:42 2022 |
| Quartus II Version | 10.0 Build 262 08/18/2010 SP 1 SJ Full Version |
| Revision Name | LZ77_Encoder |
| Top-level Entity Name | LZ77_Encoder |
| Family | Cyclone II |
| Device | EP2C70F896C8 |
| Timing Models | Final |
| Met timing requirements | Yes |
| Total logic elements | 28,873 / 68,416 ( 42 % ) |
| Total combinational functions | 28,869 / 68,416 ( 42 % ) |
| Dedicated logic registers | 8,280 / 68,416 ( 12 % ) |
| Total registers | 8280 |
| Total pins | 28 / 622 ( 5 % ) |
| Total virtual pins | 0 |
| Total memory bits | 0 / 1,152,000 ( 0 % ) |
| Embedded Multiplier 9-bit elements | 0 / 300 ( 0 % ) |
| Total PLLs | 0 / 4 ( 0 % ) |



| Flow Summary | |
|---|---|
| Flow Status | Successful - Thu Mar 24 21:05:19 2022 |
| Quartus II Version | 10.0 Build 262 08/18/2010 SP 1 SJ Full Version |
| Revision Name | LZ77_Decoder |
| Top-level Entity Name | LZ77_Decoder |
| Family | Cyclone II |
| Device | EP2C70F896C8 |
| Timing Models | Final |
| Met timing requirements | Yes |
| Total logic elements | 58 / 68,416 ( < 1 % ) |
| Total combinational functions | 41 / 68,416 ( < 1 % ) |
| Dedicated logic registers | 48 / 68,416 ( < 1 % ) |
| Total registers | 48 |
| Total pins | 27 / 622 ( 4 % ) |
| Total virtual pins | 0 |
| Total memory bits | 0 / 1,152,000 ( 0 % ) |
| Embedded Multiplier 9-bit elements | 0 / 300 ( 0 % ) |
| Total PLLs | 0 / 4 ( 0 % ) |

## 4.2 Note

This homework required your circuit to operate in given clock frequency, which is 30ns clock width. You could not modify the defined CYCLE in testbench file. The only part you can modify is End_CYCLE, which decide the maximum cycles your circuit took to complete simulation.

Please submit your .zip file to folder HW3 in the moodle.

Deadline: 2022/04/18 23:55

If you have any problem, please contact by the TA by email

tankwu@gmail.com