

Hardware LZ4 Decompressor Project Report

COE405 Project report

For Dr aimane

Faris Hijazi s201578750

Mohammed Bejadi s201582510

2018

Contents

[Introduction 3](#_Toc532727191)

[1 Design 3](#_Toc532727192)

[1.1 Modified LZ4 Convention 4](#_Toc532727193)

[1.2 Datapath 4](#_Toc532727194)

[1.2.1 Input Buffer 5](#_Toc532727195)

[1.2.2 Output Buffer 5](#_Toc532727196)

[1.2.3 Offset Register 5](#_Toc532727197)

[1.2.4 Match Length Register 6](#_Toc532727198)

[1.2.5 Literal Length Register 7](#_Toc532727199)

[1.2.6 Match Pointer Register 7](#_Toc532727200)

[1.2.7 Write Pointer 8](#_Toc532727201)

[1.2.8 Algorithmic State Machine (ASM) 9](#_Toc532727202)

[1.3 Control Unit 9](#_Toc532727203)

[1.3.1 Waiting for data 11](#_Toc532727204)

[1.3.2 Accumulating literal length 11](#_Toc532727205)

[1.3.3 Writing literals 12](#_Toc532727206)

[1.3.4 Checking for end of block 12](#_Toc532727207)

[1.3.5 Accumulating match length 13](#_Toc532727208)

[1.3.6 Copying matches 14](#_Toc532727209)

[1.3.7 Signals 14](#_Toc532727210)

[1.4 Testing process 15](#_Toc532727211)

[2 Issues and Design Decisions 18](#_Toc532727212)

[2.1 Design Decisions 18](#_Toc532727213)

[2.1.1 Synchronous/asynchronous reset 18](#_Toc532727214)

[2.1.2 Synchronous/asynchronous memory read 18](#_Toc532727215)

[2.1.3 Bonus functionality 18](#_Toc532727216)

[2.2 Issues faced 18](#_Toc532727217)

[3 Conclusion 19](#_Toc532727218)

[4 References 19](#_Toc532727219)

[5 Appendix A 19](#_Toc532727220)

[6 Appendix B 19](#_Toc532727221)

[6.1 Input buffer 19](#_Toc532727222)

[6.2 Output buffer 21](#_Toc532727223)

[6.3 Control Unit 21](#_Toc532727224)

[6.4 LZ4Decompressor (main module) 29](#_Toc532727225)

[6.5 Test bench 1 29](#_Toc532727226)

[6.6 Test bench 2 34](#_Toc532727227)

[6.7 Test bench 3 37](#_Toc532727228)

[6.8 Test bench 4 40](#_Toc532727229)

[Figure 1 Decompressor Datapath schematic 4](#_Toc532727230)

[Figure 2 Input buffer 5](#_Toc532727231)

[Figure 3 Output buffer 5](#_Toc532727232)

[Figure 4 Offset register 6](#_Toc532727233)

[Figure 5 Match Length register 6](#_Toc532727234)

[Figure 6 Literal Length register 7](#_Toc532727235)

[Figure 7 Match Pointer register 7](#_Toc532727236)

[Figure 8 Write Pointer register 8](#_Toc532727237)

[Figure 9 Datapath Algorithmic State Machine 9](#_Toc532727238)

[Figure 10 Control Unit ASMD 10](#_Toc532727239)

[Figure 11 Waiting-for-data stage 11](#_Toc532727240)

[Figure 12 Accumulating-Literal-Length stage 11](#_Toc532727241)

[Figure 13 Writing-literals stage 12](#_Toc532727242)

[Figure 14 Check-for-end-of-block stage 12](#_Toc532727243)

[Figure 15 Accumulating-match-length stage 13](#_Toc532727244)

[Figure 16 Copying-matches stage 14](#_Toc532727245)

[Figure 17 Output buffer content for test case 1 15](#_Toc532727246)

[Figure 18 Output buffer content for test case 2 16](#_Toc532727247)

[Figure 19 Output buffer content for test case 3 17](#_Toc532727248)

[Figure 20 Output buffer content for test case 4 18](#_Toc532727249)

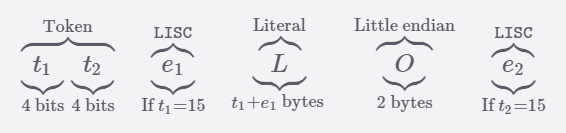
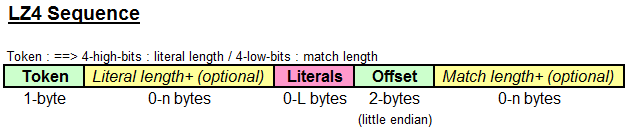
# Introduction

This report discusses designing and implementing LZ4 decompression algorithm in hardware using Verilog hardware description language.

LZ4 is lossless compression algorithm, providing compression speed at 400 MB/s. It features an extremely fast decoder, with speed in multiple GB/s.

**LZ4 Block Format Description**

LZ4 is an LZ77-type compressor with a fixed, byte-oriented encoding. An LZ4 compressed block is composed of sequences. A sequence is a suite of literals (not-compressed bytes), followed by a match copy. Each sequence starts with a token. The token is a one-byte value, separated into two 4-bits fields. Therefore, each field ranges from 0 to 15. [1]

A block looks like this:  

# Design

We’ve focused on designing the Datapath and control unit. We first drew schematics and then focused on implementing it using Verilog. For the Datapath, we first looked at the requirements and mapped them to specifications, this allowed us to choose what components are needed.

In the algorithm, we need to read, store and keep track of some values, those are depicted below:

* **Offset**: Indicates how far behind the MatchPointer should be relative to the WritePointer
* **MatchLength**: Keeps track of how many literals in the match were written to the output buffer. It is decremented every write cycle
* **LiteralLength**: Keeps track of the length of the literal
* **WritePointer**: Indicates which address to write to in the output buffer
* **MatchPointer**: A pointer to the output buffer, used to indicate which literal to copy

For each of the values (*Offset, MatchLength, LiteralLength, WritePointer, MatchPointer*) there is a special register added with functionality (such as *add*, *load, increment*) depending on what is needed.

We have gone over and reviewed the LZ4 algorithm and how it works. We used software tools to help, by reviewing some of the code, testing many examples, and consulting [professor Aiman H. El-Maleh](http://faculty.kfupm.edu.sa/COE/aimane/).

## Modified LZ4 Convention

The LZ4 algorithm is made to be used by high-level software, in the basic implementation, the compressed data would include a length field in the beginning, this helps the decompressor by informing it with the size of the data. In hardware however, this is not synthesizable, so we have added our own convention to LZ4 so the decompressor could identify when each block is over.  
The convention is as follows: having *Offset=0, and MatchLength=0*, indicates that the block has ended.

## Datapath

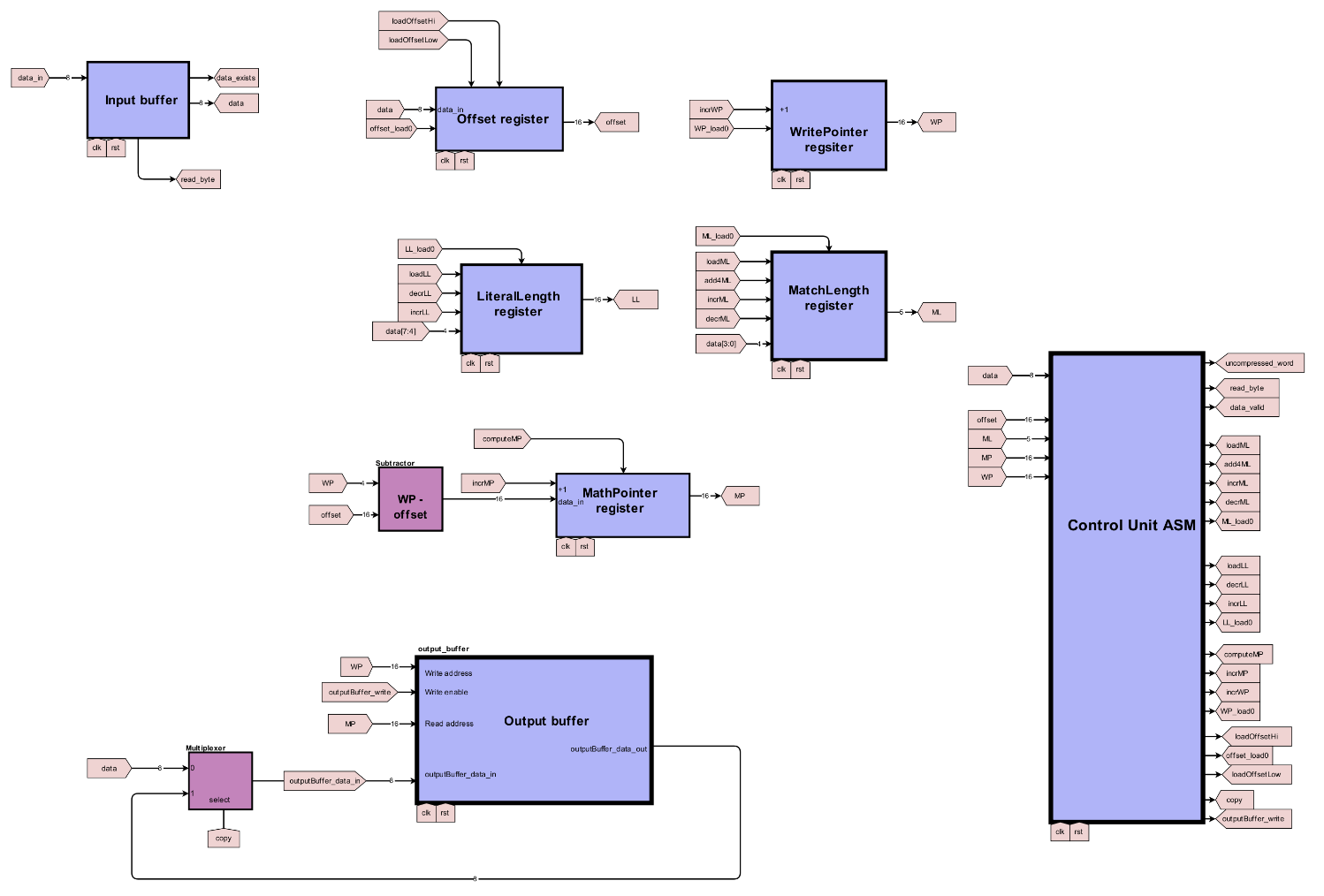


Figure 1 Decompressor Datapath schematic

This schematic was created using an online tool: digikey.com/schemeit.

The following subsections will explain and breakdown the Datapath components.

### Input Buffer

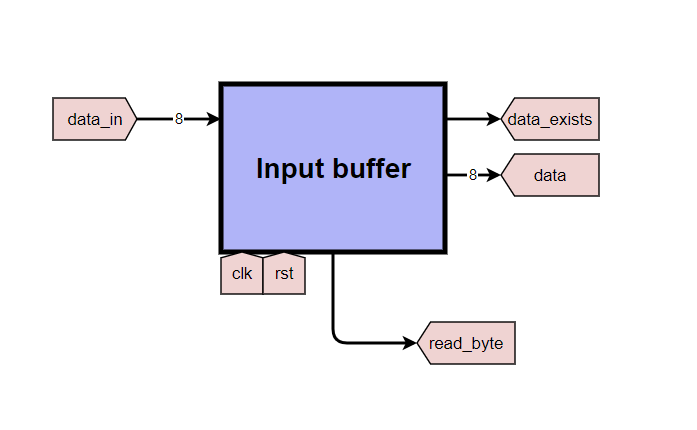


Figure 2 Input buffer

The input buffer mimics the queue data structure, it is used to buffer the incoming bytes until the decompressor decodes them. This is needed because the decompressor doesn’t read the data as fast the data is being provided. The signal ***data\_in*** is an external signal, the data is fed serially as bytes over clock cycles from the external user. The input buffer module contains pointers inside that keep track of the data that has been read and the data that has been written, from these pointers, it provides the ***data\_exists*** signal, indicating that a full block is ready to be decompressed. The data can be requested by setting ***read\_byte***=1.

### Output Buffer

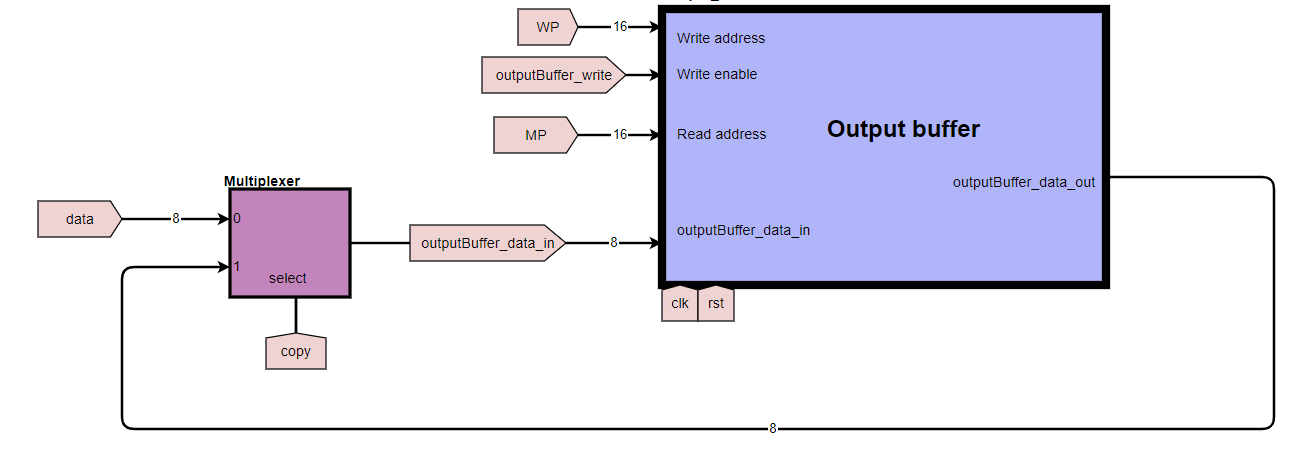


Figure 3 Output buffer

The output buffer is used to temperarely store the literals so that they can be copied. A multiplexer is used to choose between new writing literals from data, or to copy existing literals. Either way, literals are always written in the address of the write pionter (**WP**);

The match pointer (**MP**) is used read the literal to copy and written to the address of the write pointer (**WP**).

### Offset Register

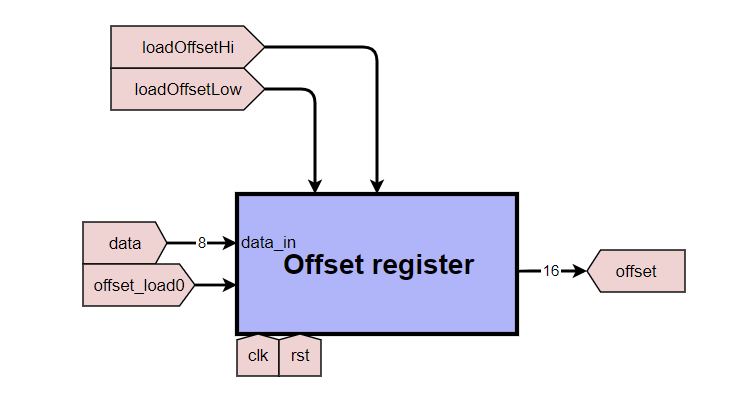


Figure 4 Offset register

The offset value indicates how far back the target match is behind the write pointer in the output buffer.

**Signals and their affects:**

* incrWP: WP = WP + 1
* loadOffsetHi: offset[15:8] = data
* loadOffsetLow: offset[7:0] = data
* offset\_load0: offset = 0

### Match Length Register

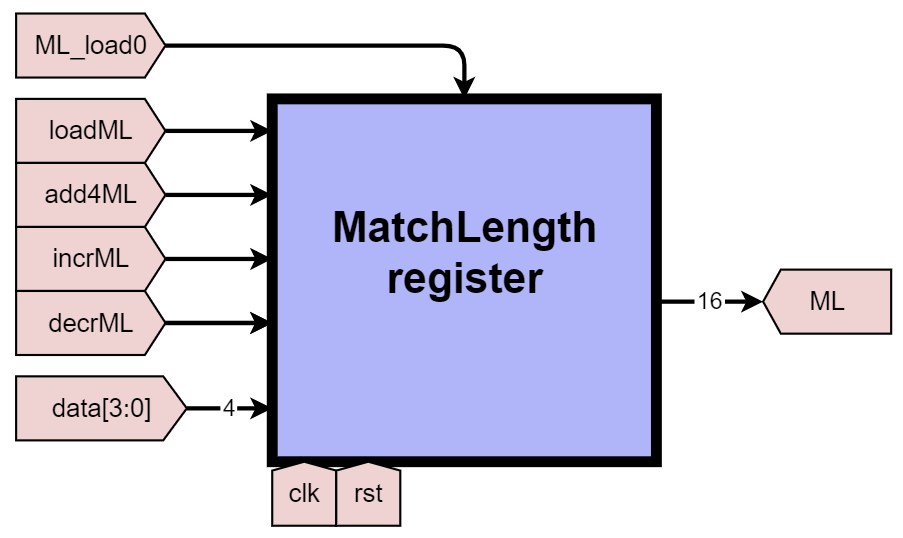


Figure 5 Match Length register

**Signals and their affects**

* ML\_load0: ML = 0
* loadML: ML = data[3:0]
* add4ML: ML = ML + 4
* incrML: ML = ML + 1
* decrML: ML = ML -1

### Literal Length Register

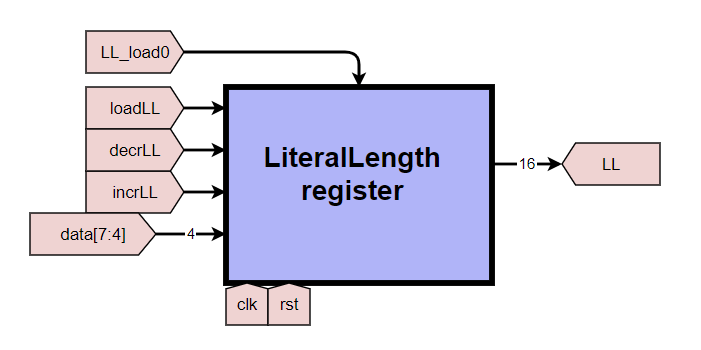


Figure 6 Literal Length register

**Signals and their affects:**

• LL\_load0: LL = 0  
• loadLL: LL = data[7:4]  
• decrLL: LL = LL - 1  
• incrLL: LL = LL + 1

### Match Pointer Register

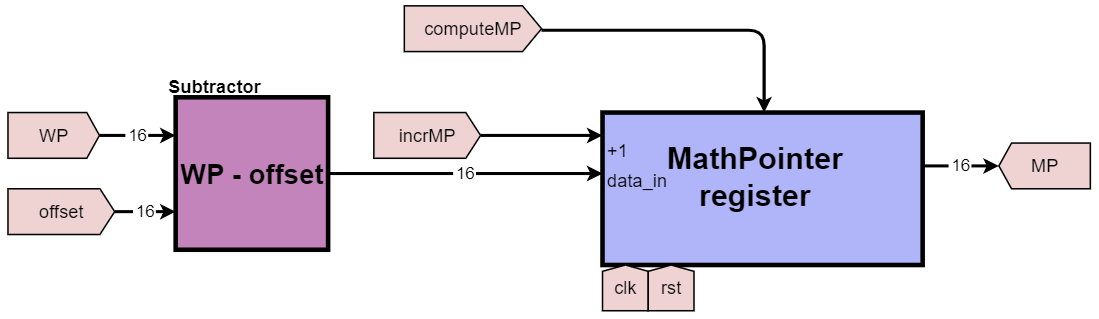


Figure 7 Match Pointer register

The match pointer (**MP**) is used to address the literals to read from the output buffer, this is done when copying literals. The value should be the write pointer (**WP**) subtracted by the offset.

In the phase of Copying matches, MP is computed, then WP and MP are incremented to go to copy the next literal.

**Signals and their affects:**

* incrMP: MP = MP + 1
* computeMP: MP = WP - offset

### Write Pointer

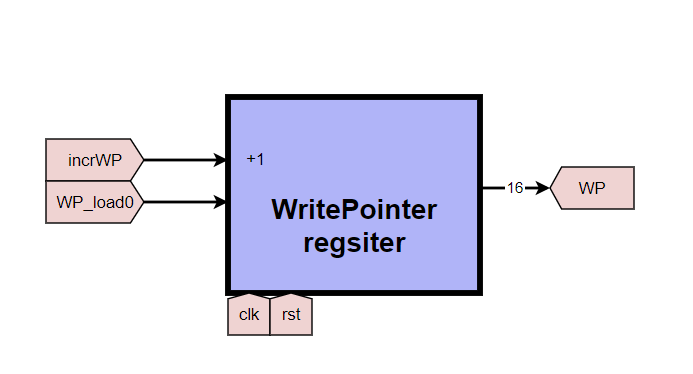


Figure 8 Write Pointer register

The write pointer (**WP**) is used to address the byte that will be written in the output buffer. It is first initialized by zero from the control unit (at the start of every block) and is incremented when writing or copying literals.

**Signals and their affects:**

* incrWP: WP = WP + 1
* WP\_load0: WP = 0

### Algorithmic State Machine (ASM)

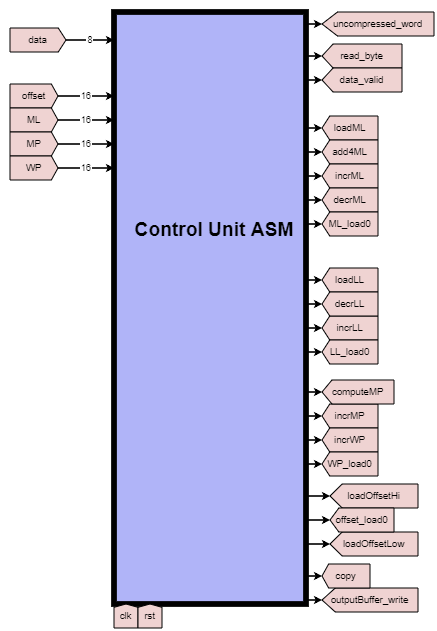


Figure 9 Datapath Algorithmic State Machine

The ASM synchronizes and manages all the other elements in the Datapath. It contains all the control signals and has the register values as inputs, those are needed to make decisions. Refer to figure// for a full diagram of the ASM.

## Control Unit

Figure10 shows the Algorithmic State Machine Diagram (ASMD) of the Control Unit of the decompressor.

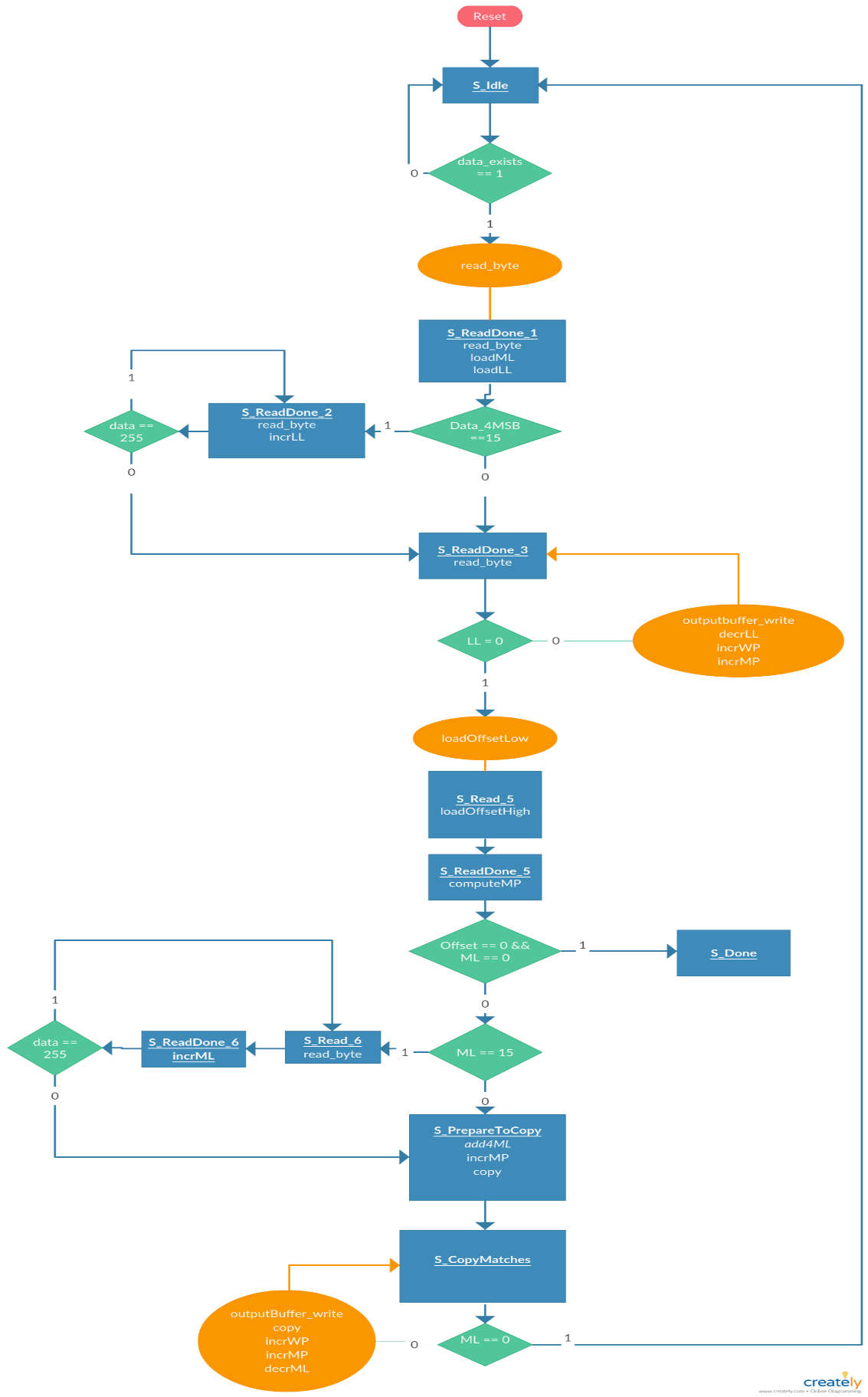
****

Figure 10 Control Unit ASMD

### Waiting for data

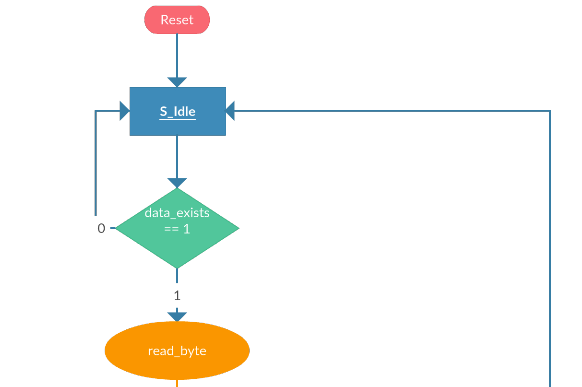


Figure 11 Waiting-for-data stage

Figure 11 shows the beginning of our ASM, as long as there’s no data, stay in S\_idle, when data exist, the decompression process starts with reading the first byte.

### Accumulating literal length

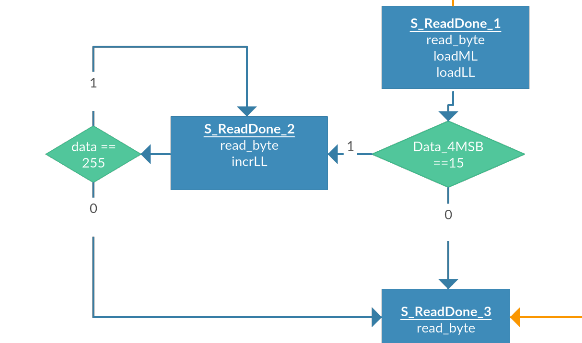


Figure 12 Accumulating-Literal-Length stage

After reading the first byte. Load signals are sent to load the data into its corresponding registers, at the same cycle, we check the 4 significant bits of the data, if it is equal to 15, we will have more bytes for the literal length. Therefore, we will accumulate it in a loop while checking the extra bytes if equal to 255 or not. When accumulating all the literal length bytes, we go to the next stage.

### Writing literals

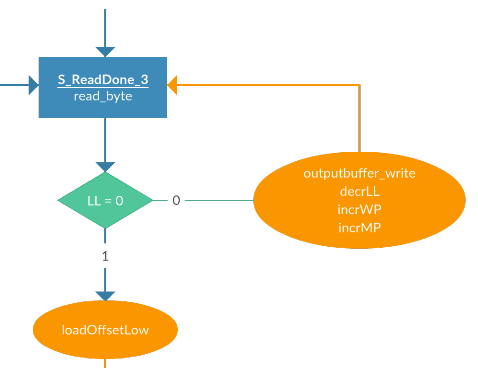


Figure 13 Writing-literals stage

The next stage is writing the literals, we read the byte and check whether the literal length is 0 or not, as long as it is not 0, we will write the byte as it is one of the literals and decrement the literal length by 1 and increment both the write and match pointers by 1. When the literal length is 0, we send a signal to load the low 8 bits of the offset with the data.

### Checking for end of block

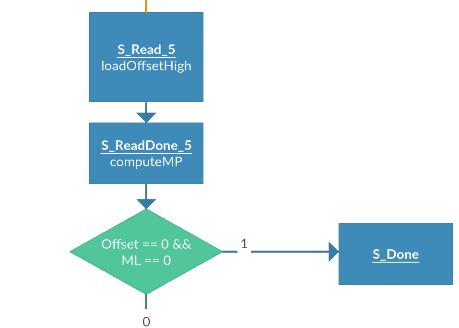


Figure 14 Check-for-end-of-block stage

Next, we must load the high part of the offset with the data. After going to the next stage, we will have the value of the offset, so we can send a signal to compute the match pointer. However, we must check whether both the offset and match length are 0, which is our indication for the end of the block.

### Accumulating match length

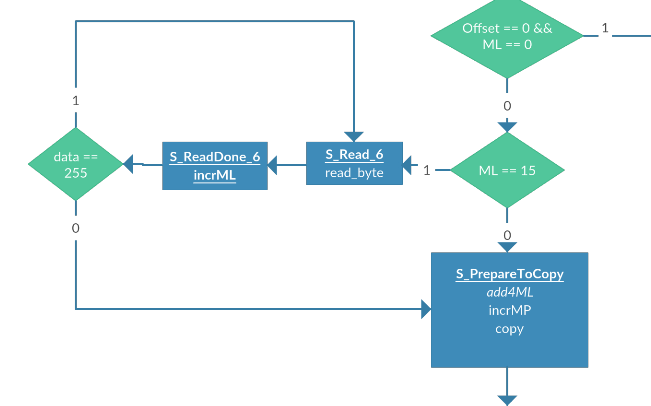


Figure 15 Accumulating-match-length stage

In case the match length is equal to 15, this indicates extra bytes exist for the length, so we will loop again to accumulate the extra bytes as long as the read byte (The extra one) is 255, when it is not. We go to the next stage to write the matched literals found.

### Copying matches

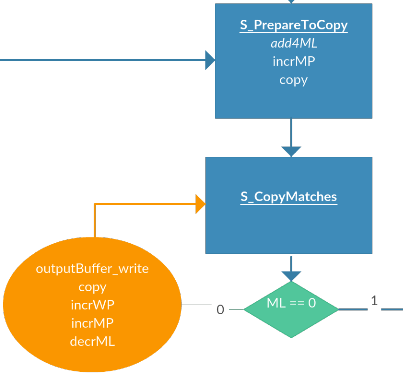


Figure 16 Copying-matches stage

The last stage is to copy the literals. We add the extra 4 to the match length and start copying and decrementing the match length as long as the match length is not 0. When it reaches 0, we go back to the idle state.

### Signals

|  |  |
| --- | --- |
| **Signal** | **Effect** |
| incrWP | WP = WP + 1 |
| incrMP | MP = MP + 1 |
| loadOffsetHigh | offset[15:8] = data |
| loadOffsetLow | offset[7:0] = data |
| decrLL | LL = LL – 1 |
| loadLL | LL = data[word\_size-1:4] |
| loadML | ML = data[3:0] |
| incrML | ML = ML + data |
| add4ML | ML = ML + 4 |
| computeMP | MP = WP - offset |
| decrML | ML = ML - 1 |
| incrLL | LL = LL + data |

## Testing process

For testing, we tried many test cases and simulated the output to ensure the correctness of the design. It is key to make sure that your test cases are sufficient and will pass through all states of the state machine. There are test cases where the literals are only copied, some where there are no matches (this helped fix the design by exposing a fault), cases where there are extra literal length bytes and match length bytes.

The following test cases follow the form:

Test blocks:

|  |  |
| --- | --- |
| Input (HEX) | 10 31 0100 10 32 0100 10 33 0100 00 0e00 10 31 0e00 10 32 0e00 02 0f00 03 0200 50 31 3131 3131 00 00 |
| Input (DEC) | 16 49 1 0 16 50 1 0 16 51 1 0 0 14 0 16 49 14 0 16 50 14 0 2 15 0 3 2 0 80 49 49 49 49 49 0 0 |
| Ouput(ASCII) | 11111 22222 33333 1111 12222 23333 311111 1111111 11111 |

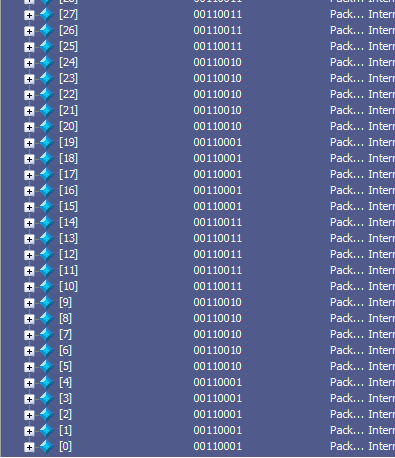


Figure Output buffer content for test case 1

|  |  |
| --- | --- |
| Input (HEX) | 10 31 0100 10 32 0100 10 33 0100 00 0e00 b0 31 3232 3232 3233 3333 3333 0000 |
| Input (DEC) | 16 49 1 0 16 50 1 0 16 51 1 0 0 14 0 176 49 50 50 50 50 50 51 51 51 51 51 0 0 |
| Ouput(ASCII) | 11111 22222 33333 1111 12222233333 |

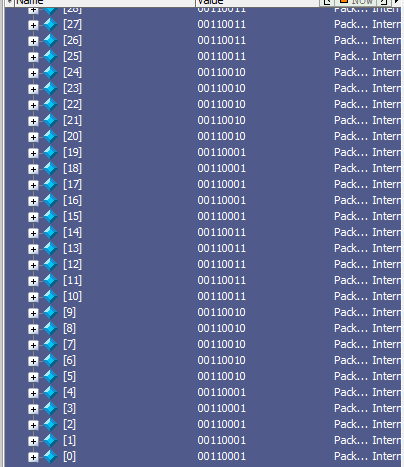


Figure Output buffer content for test case 2

|  |  |
| --- | --- |
| Input (HEX) | 1f 31 0100 01 50 3131 3131 31 |
| Input (DEC) | 31 49 1 0 1 80 49 49 49 49 49 0 0 |
| Ouput(ASCII) | 1 11111 11111 11111 11111 11111 |

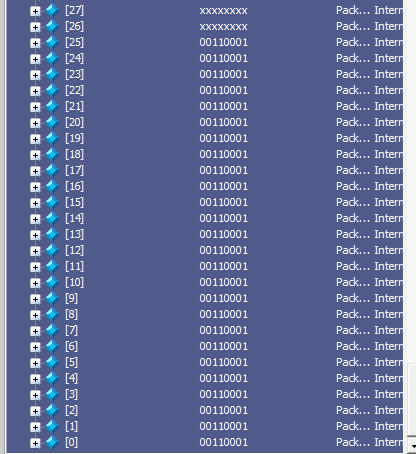


Figure Output buffer content for test case 3

|  |  |
| --- | --- |
| Input (HEX) | f0 02 3031 3233 3435 3637 3839 6162 6364 6566 67 00 |
| Input (DEC) | 240 2 4849 5051 5253 5455 5657 9798 99100 101102 103 00 |
| Ouput(ASCII) | 0123456789abcdefg |

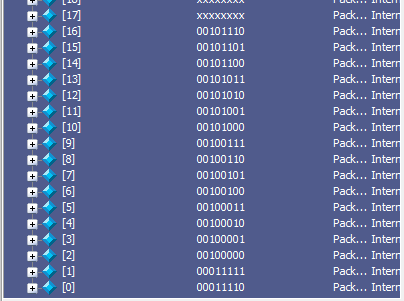


Figure Output buffer content for test case 4

# Issues and Design Decisions

## Design Decisions

### Synchronous/asynchronous reset

Making the reset synchronous or asynchronous didn’t impact the design much, so it was decided that synchronous reset would be simpler to synthesize and takes less resources.

### Synchronous/asynchronous memory read

Having asynchronous memory read would have made the design much easier, however due to the limitations of the FPGA, we had to use synchronous memory reading for both the input and output buffers. The reason is the it would be much cheaper to synthesize synchronous reading, as there are already block RAMs in the FPGA, however using asynchronous memory reading would consume many of the FPGA lookup-tables, the resources simply aren’t available to use such an approach.

### Bonus functionality

Some concepts that didn’t make it to the design, or would make it in the next version

* Having the input buffer be a circular buffer, this would allow for the system to continuously work despite the size of the buffer (assuming that on average, the *input data rate* <= *decompression data rate*)

## Issues faced

Most of the issues faced were due to incorrect timing between reading and writing the data to the input and output buffers. It was tricky since requesting the data from the input buffer on one clock cycle means that the data will come in the next cycle. We had to carefully time the signals so that the data would arrive at the correct time.

We had an issue with implementing the control unit, as we had mixed both synchronous and asynchronous components (combinational and sequential) in the *always* block. We then went over it and distinguished between the two types of signals and separated them.

# Conclusion

We have designed and implemented the control unit and Datapath of a LZ4 decompressor. We’ve also simulated and verified the results to ensure correctness. Issues, design ideas, and ideas that didn’t make it to the design were also addressed.

# References

1. <https://ieeexplore.ieee.org/document/7440278>
2. Datapath online schematic: <https://www.digikey.com/schemeit/project/lz4-decompressor-LIJ5K984006G/>
3. Control Unit diagram: <https://creately.com/diagram/jp2n42602/vD8r2aKbcIJxpNeomUlPHe6p1U%3D>

# Appendix A

Team contribution

|  |  |  |
| --- | --- | --- |
|  | **Member contribution percentage (%)** | |
| **Activity** | **Faris Hijazi** | **Mohammed Bejadi** |
| Control Unit design | 30 | 70 |
| Datapath design | 70 | 30 |
| Report writing | 60 | 40 |
| Writing Verilog code | 60 | 40 |
| Writing buffers | 100 | 0 |
| Writing Control Unit | 20 | 80 |
| Debugging | 40 | 60 |
|  |  |  |

# Appendix B

This appendix is dedicated to the Verilog code used to implement and test the design.

## Input buffer

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 1 | `timescale 1ns / 1ps | | 2 |  | | 3 | module input\_buffer #(parameter word\_size=8, address\_size=16)( | | 4 | output reg [word\_size-1:0] data\_out, | | 5 | output data\_exists, // true when there is data that hasn't been read yet | | 6 | input [word\_size-1:0] data\_in, | | 7 | input read\_byte, // outputs the next byte (if valid) and increments the read pointer | | 8 | input clk, write // write enable | | 9 | ); | | 10 |  | | 11 | parameter memory\_size=2\*\*address\_size; // memory size is computed from the address size | | 12 |  | | 13 | reg [address\_size-1:0] read\_pointer; // the address of the next word to be read | | 14 | reg [address\_size-1:0] write\_pointer; // | | 15 | reg [word\_size-1:0] memory [memory\_size-1:0]; // we have as many as memory\_size bytes | | 16 |  | | 17 | reg carry; // signal is never used, this is to avoid compiler warnings | | 18 |  | | 19 | initial begin | | 20 | write\_pointer = 0; | | 21 | read\_pointer = 0; | | 22 | end | | 23 |  | | 24 | assign data\_exists = (read\_pointer < write\_pointer); | | 25 |  | | 26 | always @ (posedge clk) begin | | 27 | if (write) begin | | 28 | memory[write\_pointer] = data\_in; | | 29 | {carry, write\_pointer} = write\_pointer + 1; // carry is never used | | 30 | end | | 31 | if (read\_byte && data\_exists) begin | | 32 | data\_out = memory[read\_pointer]; | | 33 | read\_pointer = read\_pointer + 1; | | 34 | end | | 35 | end | | 36 |  | | 37 | endmodule | |

## Output buffer

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 1 | `timescale 1ns / 1ps | | 2 |  | | 3 | module output\_buffer #(parameter word\_size=8, address\_size=4)( | | 4 | output reg [word\_size-1:0] data\_out, | | 5 | input [word\_size-1:0] data\_in, | | 6 | input [address\_size-1:0] address\_r, // for reading | | 7 | input [address\_size-1:0] address\_w, // for writing | | 8 | input clk, write); | | 9 |  | | 10 | parameter memory\_size=2\*\*address\_size; | | 11 |  | | 12 | reg [ word\_size-1:0] memory[memory\_size-1:0]; | | 13 |  | | 14 |  | | 15 | always @ (posedge clk) begin | | 16 | if (write) memory[address\_w] = data\_in; | | 17 | data\_out = memory[address\_r]; | | 18 | end | | 19 |  | | 20 | endmodule | |
|  |

## Control Unit

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 1 | `timescale 1ns / 1ps | | 2 | module ControlUnit #(parameter word\_size = 8) ( | | 3 | output  [word\_size-1:0] uncompressed\_word, | | 4 | output reg read\_byte, // for the input\_buffer | | 5 | output data\_valid, // indicates if uncompressed\_word is valid | | 6 | input [word\_size-1:0] data, // a compressed word coming from the input buffer | | 7 | input clk, | | 8 | input data\_exists, // signal from the input buffer indicating if the data is valid | | 9 | input reset // active high | | 10 | ); | | 11 |  | | 12 | parameter S\_idle =          4'b0000; // 0 | | 13 | parameter S\_ReadDone\_1 =    4'b0001; // 2 | | 14 | parameter S\_ReadDone\_2 =    4'b0010; // 3 | | 15 | parameter S\_ReadDone\_3 =    4'b0011; // 4 | | 16 | parameter S\_Read\_5 =        4'b0100; // 5 | | 17 | parameter S\_ReadDone\_5 =    4'b0101; // 6 | | 18 | parameter S\_Done =          4'b0110; // 7 | | 19 | parameter S\_Read\_6 =        4'b0111; // 8 | | 20 | parameter S\_ReadDone\_6 =    4'b1000; // 9 | | 21 | parameter S\_PrepareToCopy = 4'b1001; // a | | 22 | parameter S\_CopyMatches =   4'b1010; // c | | 23 |  | | 24 | parameter address\_size = 16; | | 25 |  | | 26 |  | | 27 | reg [3:0] next\_state, current\_state; | | 28 |  | | 29 |  | | 30 | reg [address\_size-1:0] MP, WP; | | 31 | reg [address\_size-1:0] LL, ML; | | 32 | reg outputBuffer\_write; | | 33 | reg copy; | | 34 |  | | 35 | // offset reg | | 36 | reg [15:0] offset; | | 37 |  | | 38 | wire [word\_size-1:0] outputBuffer\_data\_in; | | 39 | wire [word\_size-1:0] outputBuffer\_data\_out; | | 40 |  | | 41 |  | | 42 | reg incrWP; | | 43 | reg incrMP; | | 44 | reg loadOffsetHigh; | | 45 | reg loadOffsetLow; | | 46 | reg decrLL; | | 47 | reg loadLL, loadML; | | 48 | reg incrML; | | 49 | reg add4ML; | | 50 | reg computeMP; | | 51 | reg decrML; | | 52 | reg incrLL; | | 53 | assign uncompressed\_word = outputBuffer\_data\_in; | | 54 | assign data\_valid = outputBuffer\_write; | | 55 |  | | 56 | // output\_buffer | | 57 | assign outputBuffer\_data\_in = copy? outputBuffer\_data\_out: data; | | 58 |  | | 59 | output\_buffer #(.word\_size(word\_size), .address\_size(address\_size)) output\_buffer( | | 60 | .data\_out(outputBuffer\_data\_out), | | 61 | .data\_in(outputBuffer\_data\_in), | | 62 | .address\_r(MP), | | 63 | .address\_w(WP), | | 64 | .clk(clk), | | 65 | .write(outputBuffer\_write) | | 66 | ); | | 67 |  | | 68 | always @(posedge clk, posedge reset) begin | | 69 | if (reset) begin | | 70 | current\_state = S\_idle; | | 71 | ML = 0; | | 72 | LL = 0; | | 73 | WP = 0; | | 74 | MP = 0; | | 75 | incrWP = 0; | | 76 | incrMP = 0; | | 77 | offset = 0; | | 78 |  | | 79 | copy = 0; | | 80 | decrLL = 0; | | 81 | loadOffsetLow= 0; | | 82 | loadOffsetHigh= 0; | | 83 | loadLL= 0; | | 84 | loadML= 0; | | 85 | incrLL= 0; | | 86 | incrML= 0; | | 87 | add4ML= 0; | | 88 | computeMP= 0; | | 89 | decrML= 0; | | 90 | end | | 91 | else begin | | 92 | current\_state = next\_state; | | 93 | end | | 94 |  | | 95 | if(incrMP) begin | | 96 | MP = MP + 1; | | 97 | end | | 98 | if(incrWP) begin | | 99 | WP = WP + 1; | | 100 | end | | 101 | if (decrLL) begin | | 102 | LL = LL - 1; | | 103 | end | | 104 | if (loadOffsetLow) begin | | 105 | offset[7:0] = data; | | 106 | end | | 107 | if (loadOffsetHigh) begin | | 108 | offset[15:8] = data; | | 109 | end | | 110 | if (loadLL) begin | | 111 | LL = data[word\_size-1:4]; | | 112 | end | | 113 | if (loadML) begin | | 114 | ML = data[3:0]; | | 115 | end | | 116 | if (incrLL) begin | | 117 | LL = LL + data; | | 118 | end | | 119 | if (incrML) begin | | 120 | ML = ML + data; | | 121 | end | | 122 | if (add4ML) begin | | 123 | ML = ML + 4; | | 124 | end | | 125 | if (computeMP) begin | | 126 | MP = WP - offset; | | 127 | end | | 128 | if (decrML) begin | | 129 | ML = ML - 1; | | 130 | end | | 131 |  | | 132 | end | | 133 |  | | 134 |  | | 135 | always @(\*) begin | | 136 | read\_byte = 0; | | 137 | outputBuffer\_write = 0; | | 138 | copy = 0; | | 139 | decrLL = 0; | | 140 | loadOffsetLow= 0; | | 141 | loadOffsetHigh= 0; | | 142 | loadLL= 0; | | 143 | loadML= 0; | | 144 | incrLL= 0; | | 145 | incrML= 0; | | 146 | add4ML= 0; | | 147 | computeMP= 0; | | 148 | decrML= 0; | | 149 | incrWP = 0; | | 150 | incrMP = 0; | | 151 |  | | 152 | case (current\_state) | | 153 | S\_idle: | | 154 | if (data\_exists)begin | | 155 | read\_byte = 1; | | 156 | next\_state = S\_ReadDone\_1; | | 157 | end | | 158 | else | | 159 | next\_state = S\_idle; | | 160 |  | | 161 | S\_ReadDone\_1: begin | | 162 |  | | 163 | loadML = 1; | | 164 | loadLL = 1; | | 165 | read\_byte = 1; | | 166 | if (data[word\_size-1:4] == 15) begin// if the LL is 15 | | 167 | next\_state = S\_ReadDone\_2; | | 168 | end | | 169 | else begin | | 170 | next\_state = S\_ReadDone\_3; | | 171 | end | | 172 | end | | 173 | S\_ReadDone\_2: begin | | 174 | read\_byte = 1; | | 175 | incrLL = 1; | | 176 | if (data == 255) begin | | 177 | next\_state = S\_ReadDone\_2; // loop (go to same state) | | 178 | end | | 179 | else begin | | 180 | next\_state= S\_ReadDone\_3; | | 181 | end | | 182 | end | | 183 |  | | 184 | S\_ReadDone\_3: begin// Read Byte | | 185 | read\_byte = 1; | | 186 |  | | 187 | if (LL == 0) begin | | 188 | next\_state = S\_Read\_5; | | 189 | loadOffsetLow = 1; | | 190 | end | | 191 | else begin// Loop and Write in buffer | | 192 | outputBuffer\_write = 1; | | 193 | decrLL = 1; | | 194 | incrWP = 1; | | 195 | incrMP = 1; | | 196 |  | | 197 | next\_state = S\_ReadDone\_3; | | 198 | end | | 199 | end | | 200 |  | | 201 | S\_Read\_5: begin// Read Byte | | 202 | loadOffsetHigh = 1; | | 203 | next\_state = S\_ReadDone\_5; | | 204 | end | | 205 |  | | 206 | S\_ReadDone\_5: begin | | 207 | computeMP = 1; | | 208 | if (offset == 0 && ML == 0) | | 209 | next\_state = S\_Done; | | 210 | else if (ML == 15) | | 211 | next\_state = S\_Read\_6; | | 212 | else | | 213 | next\_state = S\_PrepareToCopy; | | 214 | end | | 215 |  | | 216 | S\_Done: begin | | 217 | end | | 218 |  | | 219 | S\_Read\_6: begin | | 220 | read\_byte = 1; | | 221 | next\_state = S\_ReadDone\_6; | | 222 | end | | 223 |  | | 224 | S\_ReadDone\_6: begin | | 225 | incrML = 1; | | 226 |  | | 227 | if (data == 255) begin | | 228 | next\_state = S\_Read\_6; | | 229 | end | | 230 | else begin | | 231 | next\_state = S\_PrepareToCopy; | | 232 | end | | 233 | end | | 234 |  | | 235 | S\_PrepareToCopy: begin | | 236 | add4ML = 1; | | 237 | incrMP = 1; | | 238 | copy = 1; | | 239 | next\_state = S\_CopyMatches; | | 240 | end | | 241 |  | | 242 | S\_CopyMatches: | | 243 | if (ML == 0) begin | | 244 | next\_state = S\_idle; | | 245 | end | | 246 | else begin | | 247 | outputBuffer\_write = 1; | | 248 | copy = 1; | | 249 | incrWP = 1; | | 250 | incrMP = 1; | | 251 | decrML = 1; | | 252 | next\_state = S\_CopyMatches; | | 253 | end | | 254 | default: | | 255 | next\_state = S\_idle; | | 256 | endcase | | 257 | end | | 258 | endmodule | |

## LZ4Decompressor (main module)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 1 | `timescale 1ns / 1ps | | 2 | module LZ4Decompressor #(parameter word\_size=8)( | | 3 | *output* [word\_size-1:0] uncompressed\_word,   // | | 4 | *output* data\_valid, | | 5 | *input* [word\_size-1:0] compressed\_word, | | 6 | *input* write, clk, reset | | 7 | ); | | 8 |  | | 9 | *wire* [word\_size-1:0] data; | | 10 | *wire* read\_word; | | 11 |  | | 12 | *input\_buffer* #(.word\_size(word\_size), .address\_size(16)) input\_buffer ( | | 13 | .data\_out(data), | | 14 | .data\_exists(data\_exists),// true when there is data that hasn't been read yet | | 15 | .data\_in(compressed\_word), | | 16 | .read\_byte(read\_word), // outputs the next byte (if valid) and increments the read pointer | | 17 | .clk(clk), | | 18 | .write(write) | | 19 | ); | | 20 |  | | 21 | *ControlUnit* #(.word\_size(word\_size)) ControlUnit( | | 22 | .uncompressed\_word(uncompressed\_word), | | 23 | .read\_byte(read\_word), | | 24 | .data\_valid(data\_valid), | | 25 | .data(data), | | 26 | .clk(clk), | | 27 | .data\_exists(data\_exists), | | 28 | .reset(reset) | | 29 | ); | | 30 | endmodule | |

## Test bench 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 1 | `timescale 1ns / 1ps | | 2 | module LZ4DecompressorTB (); | | 3 | parameter word\_size = 8; | | 4 | // parameter address\_size = 16; | | 5 |  | | 6 |  | | 7 | // Compressed data =    10 31 0100  10 32 0100  10 33 0100  00 0e00     10 31 0e00      10 32 0e00      02 0f00     03 0200         50 31 3131 3131 00 00 | | 8 | // Compressed input:    16 49 1 0   16 50 1 0   16 51 1 0   0 14 0      16 49 14 0      16 50 14 0      2 15 0      3 2 0           80 49 49 49 49 49 0 0 | | 9 | // Compressed Decimal = 11111       22222       33333       1111        12222           23333           311111      1111111         11111 | | 10 |  | | 11 | // 11111 22222 33333 1111 12222 23333 311111 1111111 11111 | | 12 |  | | 13 | reg clk, reset, data\_exists; | | 14 | reg [word\_size-1:0] compressed\_word; | | 15 | reg write\_en; | | 16 | wire [word\_size-1:0] uncompressed\_word; | | 17 | wire data\_valid; | | 18 |  | | 19 | LZ4Decompressor #(.word\_size(word\_size))  decompressor( | | 20 | .uncompressed\_word(uncompressed\_word),  // | | 21 | .data\_valid(data\_valid), | | 22 | .compressed\_word(compressed\_word), | | 23 | .write(write\_en), | | 24 | .clk(clk), | | 25 | .reset(reset) | | 26 | ); | | 27 |  | | 28 | initial begin | | 29 | clk = 0 ; forever #10 clk = ~clk ; | | 30 | end | | 31 |  | | 32 | initial begin | | 33 | reset = 1; | | 34 | @(negedge clk) | | 35 | reset = 0; | | 36 |  | | 37 | @(negedge clk) // readbyte s1 | | 38 | write\_en = 1; | | 39 | compressed\_word = 16; | | 40 | @(negedge clk) // Read\_Done\_3 | | 41 | compressed\_word = 49; | | 42 | @(negedge clk) | | 43 | compressed\_word = 1; | | 44 | @(negedge clk) | | 45 | compressed\_word = 0; | | 46 | @(negedge clk) | | 47 |  | | 48 | compressed\_word = 16; | | 49 | @(negedge clk) | | 50 | compressed\_word = 50; | | 51 | @(negedge clk) | | 52 | compressed\_word = 1; | | 53 | @(negedge clk) | | 54 | compressed\_word = 0; | | 55 | @(negedge clk) | | 56 |  | | 57 |  | | 58 | compressed\_word = 16; | | 59 | @(negedge clk) | | 60 | compressed\_word = 51; | | 61 | @(negedge clk) | | 62 | compressed\_word = 1; | | 63 | @(negedge clk) | | 64 | compressed\_word = 0; | | 65 | @(negedge clk) | | 66 |  | | 67 |  | | 68 |  | | 69 |  | | 70 | compressed\_word = 0; | | 71 | @(negedge clk) | | 72 | compressed\_word = 14; | | 73 | @(negedge clk) | | 74 | compressed\_word = 0; | | 75 | @(negedge clk) | | 76 |  | | 77 |  | | 78 | compressed\_word = 16; | | 79 | @(negedge clk) | | 80 | compressed\_word = 49; | | 81 | @(negedge clk) | | 82 | compressed\_word = 14; | | 83 | @(negedge clk) | | 84 | compressed\_word = 0; | | 85 | @(negedge clk) | | 86 |  | | 87 |  | | 88 | // 10 32 0e00 | | 89 | // 23333 | | 90 | // 16 50 14 0 | | 91 | compressed\_word = 16; | | 92 | @(negedge clk) | | 93 | compressed\_word = 50; | | 94 | @(negedge clk) | | 95 | compressed\_word = 14; | | 96 | @(negedge clk) | | 97 | compressed\_word = 0; | | 98 | @(negedge clk) | | 99 |  | | 100 | // 02 0f00 | | 101 | // 311111 | | 102 | // 2 15 0 | | 103 | compressed\_word = 02; | | 104 | @(negedge clk) | | 105 | compressed\_word = 15; | | 106 | @(negedge clk) | | 107 | compressed\_word = 0; | | 108 | @(negedge clk) | | 109 |  | | 110 |  | | 111 |  | | 112 | // 03 0200 | | 113 | // 1111111 | | 114 | // 3 2 0 | | 115 |  | | 116 | compressed\_word = 03; | | 117 | @(negedge clk) | | 118 | compressed\_word = 02; | | 119 | @(negedge clk) | | 120 | compressed\_word = 00; | | 121 | @(negedge clk) | | 122 |  | | 123 | compressed\_word = 80; | | 124 | @(negedge clk) | | 125 | compressed\_word = 49; | | 126 | @(negedge clk) | | 127 | compressed\_word = 49; | | 128 | @(negedge clk) | | 129 | compressed\_word = 49; | | 130 | @(negedge clk) | | 131 | compressed\_word = 49; | | 132 | @(negedge clk) | | 133 | compressed\_word = 49; | | 134 | @(negedge clk) | | 135 |  | | 136 | compressed\_word = 00; | | 137 | @(negedge clk) | | 138 |  | | 139 | compressed\_word = 00; | | 140 |  | | 141 | end | | 142 | endmodule | |

## Test bench 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 1 | `timescale 1ns / 1ps | | 2 |  | | 3 |  | | 4 |  | | 5 | module LZ4DecompressorTB2 (); | | 6 | parameter word\_size = 8; | | 7 | // parameter address\_size = 16; | | 8 |  | | 9 |  | | 10 | // Compressed data =    10 31 0100          10  32 0100     10 33 0100      00 0e00         b0 31 3232 3232 3233 3333 3333 0000 | | 11 | // Decompressed data =  11111               22222           33333           1111            12222233333 | | 12 | // decimal input:       16  49  1   0       16  50  1   0   16  51  1   0   0   14  0       176 49  50  50  50  50  50  51  51  51  51  51  0   0 | | 13 |  | | 14 | // 11111 22222 33333 1111 12222 23333 311111 1111111 11111 | | 15 |  | | 16 | reg clk, reset, data\_exists; | | 17 | reg [word\_size-1:0] compressed\_word; | | 18 | reg write\_en; | | 19 | wire [word\_size-1:0] uncompressed\_word; | | 20 | wire data\_valid; | | 21 |  | | 22 | LZ4Decompressor #(.word\_size(word\_size))  decompressor( | | 23 | .uncompressed\_word(uncompressed\_word),  // | | 24 | .data\_valid(data\_valid), | | 25 | .compressed\_word(compressed\_word), | | 26 | .write(write\_en), | | 27 | .clk(clk), | | 28 | .reset(reset) | | 29 | ); | | 30 |  | | 31 | initial begin | | 32 | clk = 0 ; forever #10 clk = ~clk ; | | 33 | end | | 34 |  | | 35 | initial begin | | 36 | reset = 1; | | 37 | @(negedge clk) | | 38 | reset = 0; | | 39 |  | | 40 | @(negedge clk) // readbyte s1 | | 41 | write\_en = 1; | | 42 | compressed\_word = 16; | | 43 | @(negedge clk) // Read\_Done\_3 | | 44 | compressed\_word = 49; | | 45 | @(negedge clk) | | 46 | compressed\_word = 1; | | 47 | @(negedge clk) | | 48 | compressed\_word = 0; | | 49 | @(negedge clk) | | 50 |  | | 51 | compressed\_word = 16; | | 52 | @(negedge clk) | | 53 | compressed\_word = 50; | | 54 | @(negedge clk) | | 55 | compressed\_word = 1; | | 56 | @(negedge clk) | | 57 | compressed\_word = 0; | | 58 | @(negedge clk) | | 59 |  | | 60 |  | | 61 | compressed\_word = 16; | | 62 | @(negedge clk) | | 63 | compressed\_word = 51; | | 64 | @(negedge clk) | | 65 | compressed\_word = 1; | | 66 | @(negedge clk) | | 67 | compressed\_word = 0; | | 68 | @(negedge clk) | | 69 |  | | 70 |  | | 71 |  | | 72 | compressed\_word = 0; | | 73 | @(negedge clk) | | 74 | compressed\_word = 14; | | 75 | @(negedge clk) | | 76 | compressed\_word = 0; | | 77 | @(negedge clk) | | 78 |  | | 79 |  | | 80 | compressed\_word = 176; | | 81 | @(negedge clk) | | 82 | compressed\_word = 49; | | 83 | @(negedge clk) | | 84 | compressed\_word = 50; | | 85 | @(negedge clk) | | 86 | compressed\_word = 50; | | 87 | @(negedge clk) | | 88 |  | | 89 |  | | 90 | compressed\_word = 50; | | 91 | @(negedge clk) | | 92 | compressed\_word = 50; | | 93 | @(negedge clk) | | 94 | compressed\_word = 50; | | 95 | @(negedge clk) | | 96 | compressed\_word = 51; | | 97 | @(negedge clk) | | 98 | compressed\_word = 51; | | 99 | @(negedge clk) | | 100 | compressed\_word = 51; | | 101 | @(negedge clk) | | 102 | compressed\_word = 51; | | 103 | @(negedge clk) | | 104 | compressed\_word = 51; | | 105 | @(negedge clk) | | 106 |  | | 107 | compressed\_word = 00; | | 108 | @(negedge clk) | | 109 |  | | 110 | compressed\_word = 00; | | 111 |  | | 112 | end | | 113 | endmodule | |

## Test bench 3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 1 | `timescale 1ns / 1ps | | 2 |  | | 3 |  | | 4 |  | | 5 | module LZ4DecompressorTB3 (); | | 6 | parameter word\_size = 8; | | 7 | // parameter address\_size = 16; | | 8 |  | | 9 |  | | 10 | // Compressed data =    1f 31 0100 01                               50  3131 3131 31 | | 11 | // Decompressed data =  1   11111   11111   11111   11111           11111 | | 12 | // decimal input:       31  49  1   0   1                           80  49  49  49  49  49  0   0 | | 13 |  | | 14 | // 11111 22222 33333 1111 12222 23333 311111 1111111 11111 | | 15 |  | | 16 | reg clk, reset, data\_exists; | | 17 | reg [word\_size-1:0] compressed\_word; | | 18 | reg write\_en; | | 19 | wire [word\_size-1:0] uncompressed\_word; | | 20 | wire data\_valid; | | 21 |  | | 22 | LZ4Decompressor #(.word\_size(word\_size))  decompressor( | | 23 | .uncompressed\_word(uncompressed\_word),  // | | 24 | .data\_valid(data\_valid), | | 25 | .compressed\_word(compressed\_word), | | 26 | .write(write\_en), | | 27 | .clk(clk), | | 28 | .reset(reset) | | 29 | ); | | 30 |  | | 31 | initial begin | | 32 | clk = 0 ; forever #10 clk = ~clk ; | | 33 | end | | 34 |  | | 35 | initial begin | | 36 | reset = 1; | | 37 | @(negedge clk) | | 38 | reset = 0; | | 39 | @(negedge clk) // readbyte s1 | | 40 | write\_en = 1; | | 41 | compressed\_word = 31; | | 42 | @(negedge clk) // Read\_Done\_3 | | 43 | compressed\_word = 49; | | 44 | @(negedge clk) | | 45 | compressed\_word = 1; | | 46 | @(negedge clk) | | 47 | compressed\_word = 0; | | 48 | @(negedge clk) | | 49 |  | | 50 | compressed\_word = 1; | | 51 | @(negedge clk) | | 52 | compressed\_word = 80; | | 53 | @(negedge clk) | | 54 | compressed\_word = 49; | | 55 | @(negedge clk) | | 56 | compressed\_word = 49; | | 57 | @(negedge clk) | | 58 |  | | 59 |  | | 60 | compressed\_word = 49; | | 61 | @(negedge clk) | | 62 | compressed\_word = 49; | | 63 | @(negedge clk) | | 64 | compressed\_word = 49; | | 65 | @(negedge clk) | | 66 | compressed\_word = 0; | | 67 | @(negedge clk) | | 68 |  | | 69 |  | | 70 |  | | 71 | compressed\_word = 0; | | 72 |  | | 73 | end | | 74 | endmodule | |

## Test bench 4

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 1 | `timescale 1ns / 1ps | | 2 |  | | 3 |  | | 4 |  | | 5 | module LZ4DecompressorTB4 (); | | 6 | parameter word\_size = 8; | | 7 | // parameter address\_size = 16; | | 8 |  | | 9 | // 0123456789abcdefg 00 | | 10 | // Compressed data =    f0  02  3031    3233    3435    3637    3839    6162    6364    6566    67  00 | | 11 | // Decompressed data =  0123456789abcdefg | | 12 | // decimal input:       240  2  4849    5051    5253    5455    5657    9798    99100   101102  103 00 | | 13 |  | | 14 | // 11111 22222 33333 1111 12222 23333 311111 1111111 11111 | | 15 |  | | 16 | reg clk, reset, data\_exists; | | 17 | reg [word\_size-1:0] compressed\_word; | | 18 | reg write\_en; | | 19 | wire [word\_size-1:0] uncompressed\_word; | | 20 | wire data\_valid; | | 21 |  | | 22 | LZ4Decompressor #(.word\_size(word\_size))  decompressor( | | 23 | .uncompressed\_word(uncompressed\_word),  // | | 24 | .data\_valid(data\_valid), | | 25 | .compressed\_word(compressed\_word), | | 26 | .write(write\_en), | | 27 | .clk(clk), | | 28 | .reset(reset) | | 29 | ); | | 30 |  | | 31 | initial begin | | 32 | clk = 0 ; forever #10 clk = ~clk ; | | 33 | end | | 34 |  | | 35 | initial begin | | 36 | reset = 1; | | 37 | @(negedge clk) | | 38 | reset = 0; | | 39 | @(negedge clk) // readbyte s1 | | 40 | write\_en = 1; | | 41 |  | | 42 | // f0   02  0123456789abcdefg 00 | | 43 | compressed\_word = 240; | | 44 | @(negedge clk) // Read\_Done\_3 | | 45 | compressed\_word = 2; | | 46 | @(negedge clk) | | 47 | compressed\_word = 48; | | 48 | @(negedge clk) | | 49 | compressed\_word = 49; | | 50 | @(negedge clk) | | 51 |  | | 52 | compressed\_word = 50; | | 53 | @(negedge clk) | | 54 | compressed\_word = 51; | | 55 | @(negedge clk) | | 56 | compressed\_word = 52; | | 57 | @(negedge clk) | | 58 | compressed\_word = 53; | | 59 | @(negedge clk) | | 60 | compressed\_word = 54; | | 61 | @(negedge clk) | | 62 | compressed\_word = 55; | | 63 | @(negedge clk) | | 64 | compressed\_word = 56; | | 65 | @(negedge clk) | | 66 | compressed\_word = 57; | | 67 | @(negedge clk) | | 68 |  | | 69 |  | | 70 |  | | 71 | compressed\_word = 97; //a | | 72 | @(negedge clk) | | 73 | compressed\_word = 98; | | 74 | @(negedge clk) | | 75 | compressed\_word = 99; | | 76 | @(negedge clk) | | 77 | compressed\_word = 100; | | 78 | @(negedge clk) | | 79 | compressed\_word = 101; | | 80 | @(negedge clk) | | 81 | compressed\_word = 102; | | 82 | @(negedge clk) | | 83 | compressed\_word = 103; | | 84 | @(negedge clk) | | 85 | compressed\_word = 104; | | 86 | @(negedge clk) | | 87 | compressed\_word = 105; | | 88 | end | | 89 | endmodule | |