Faculty of Engineering and Technology
Electrical and Computer Engineering Department

CHIP DESIGN VERIFICATION

ENCS5337

Course Project

Design Verification of a Hardware Compression Decompression Chip

Prepared by

Abdallah Ayoub – 1190994

Sondos Shahin – 1200166

Instructor

Dr. Ayman Hroub

Section: 1

BIRZEIT

June – 2024

Contents

Table of Figures

Introduction

This report outlines the design and verification of a custom-designed chip that performs data compression and decompression using a dictionary-based approach. The chip's primary functionality revolves around two core algorithms: compression and decompression. The compression algorithm stores unique data entries in an internal dictionary and utilizes their indexes for compressed output. Conversely, the decompression algorithm retrieves original data from the dictionary based on the provided compressed indexes.

Compression Algorithm

Compression algorithms are designed to reduce the size of data by eliminating redundancies and exploiting patterns within the data. These algorithms are essential for optimizing storage and enhancing data transmission efficiency. One common approach is dictionary-based compression, where the chip maintains an internal dictionary of frequently occurring data patterns. As new data is received, the chip compares it to entries in the dictionary. If a match is found, the chip outputs the index of the stored data, which serves as a compressed representation. If no match is found, the chip stores the new data in the dictionary and outputs the index of the new entry. This process significantly reduces the amount of data that needs to be stored or transmitted, utilizing only the least significant bits of a 32-bit index, resulting in an 8-bit compressed data size. By efficiently managing the dictionary entries, the compression algorithm ensures that storage and bandwidth are utilized effectively. [1]

Decompression Algorithm

Decompression algorithms work to reconstruct the original data from its compressed form, reversing the compression process. This is crucial for ensuring that the data can be accurately retrieved and used after being compressed for storage or transmission. The decompression algorithm's efficiency and accuracy are vital for applications requiring reliable data recovery from its compressed state. [2]

Clock

The clock is a critical component that synchronizes the operations of the entire circuit. It generates a continuous sequence of pulses, which provide a timing reference for all sequential elements, such as flip-flops and registers, ensuring that data is correctly transferred and processed at precise intervals. The clock's frequency determines the speed at which the system operates, directly influencing the performance and efficiency of the chip. [3]

Reference Model

A reference model is a high-level, abstract representation of a system's intended functionality. It serves as a standard against which the actual implementation is compared during the verification process. The reference model is typically written in a high-level programming language, making it easier to understand and modify. It is used to ensure that the design behaves as expected by providing

a known correct output for given inputs. By comparing the outputs of the reference model and the actual design, verification engineers can identify discrepancies and potential errors, thereby improving the reliability and correctness of the final product. [4]

Verification Plan

A verification plan is a comprehensive document that outlines the strategy and methodologies for ensuring the correctness and functionality of a digital design or system. It details the objectives, scope, resources, and schedule of the verification process. The plan includes descriptions of the verification levels, functions to be verified, required resources and tools, specific tests and methods, coverage requirements, completion criteria, test scenarios, risks, and dependencies. The verification plan serves as a roadmap for verification engineers, guiding them through the process of verifying and validating the design to ensure it meets the specified requirements and standards. [5]

Coverage Plan

A Coverage Plan is an integral part of the verification process in digital design. It outlines the metrics and criteria used to measure the completeness and effectiveness of the verification efforts. Coverage metrics include code coverage, functional coverage, assertion coverage, and other specialized metrics depending on the design and verification goals. The Coverage Plan specifies the targeted coverage goals, the methodology for collecting coverage data, and the tools and resources needed to analyze and report coverage results. It helps ensure that the verification process thoroughly exercises the design, identifies areas of improvement, and ultimately increases confidence in the correctness and robustness of the design implementation.

DUT Overview

The Design Under Test (DUT) in this project is a chip capable of performing data compression and decompression using a dictionary-based compression algorithm. It consists of several key components, including an internal memory (dictionary memory) to store data and an index register to manage memory access. The DUT has a clock input for synchronization and a reset input to initialize its internal state.

Functionality

- The DUT performs both data compression and decompression operations.
- It utilizes an internal dictionary memory to store frequently encountered data entries.
- During compression, the DUT compares the input data with entries in the dictionary. If a match is found, the index of the matching entry is written as the compressed data. If no match is found, the input data is added to the dictionary, and its corresponding index becomes the compressed data.
- Decompression involves retrieving the original data from the dictionary using the received compressed data (index).

Key Components

- **Dictionary Memory:** Internal memory for storing frequently used data entries. The size of this memory determines the maximum number of unique entries the chip can handle.
- **Comparator:** Compares the input data with entries in the dictionary during compression.
- **Index Register:** Maintains the current index pointing to the last used slot in the dictionary memory.
- **Control Unit:** Decodes the "command" signal to control the chip's operation (compression, decompression, reset).
- **Data Paths:** Facilitate data flow between various components (data in, compressed out, decompressed out, ...etc.).

Ports

- **Clock (clk):** Provides the clock signal for synchronization.
- **Reset (reset):** Clears the dictionary memory and index register, initializing the chip.
- **Command (command):** Specifies the desired operation (compression, decompression, no operation, invalid command).
- **Data Input (data_in):** Input port for data to be compressed (80 bits).
- **Compressed Data Input (compressed_in):** Input port for compressed data (index) during decompression (8 bits).
- **Compressed Data Output (compressed_out):** Output port for compressed data (index) after compression (8 bits).
- **Decompressed Data Output (decompressed_out):** Output port for decompressed data (original data) after decompression (80 bits).
- **Response (response):** Output port indicating the operation status (valid output, error).

Testbench Structure

The testbench for the chip verification comprises several essential elements to ensure comprehensive testing of the design functionality. It includes stimulus generation, component instantiation, clock generation, and result monitoring.

- **Stimulus Generation:** The testbench generates input stimuli to simulate various scenarios and test the functionality of the chip under different conditions. This includes providing test vectors for data in, compressed in, and command inputs to cover all possible command operations.
- **Component Instantiation:** The testbench instantiates the Design Under Test (DUT), connecting its input and output ports to the corresponding signals in the testbench. This aims to communicate the testbench and the DUT during simulation.
- **Clock Generation:** Generates a periodic clock signal (clk) for the DUT. It ensures that the DUT operates synchronously with the simulated environment. The clock frequency can be adjusted as needed to test timing-sensitive operations.
- **Result Monitoring:** The testbench includes monitoring processes to observe and verify the output responses of the DUT. This involves checking the values of compressed out, decompressed out, and response signals to ensure they match the expected results based on the input stimuli.
- **Error Handler:** Detects any mismatches between expected and actual outputs and flags them as errors.

Testbench Workflow

The testbench operation follows a well-defined sequence to thoroughly evaluate the DUT (Device Under Test).

1. **Initialization:** The testbench starts by initializing all internal variables and signals. Then, it generates a reset pulse (reset) to clear the DUT's internal state (dictionary memory and index register). The initial command might be set to "No Operation" to ensure the DUT starts in a stable state.
2. **Stimulus Generation:** Test vectors are generated to provide input data for compression and decompression operations. Each test vector focuses on verifying a specific aspect of the DUT's functionality (e.g., compressing known data, handling full dictionary, etc.). During each test case, the testbench generates appropriate stimuli for the DUT's input ports:
    - command: Sets the desired operation (compression, decompression).
    - data_in: Provides data to be compressed (can be random or specific patterns).
    - compressed_in: Injects pre-defined compressed data (index) for decompression testing (if applicable).

    The testbench waits for a specific amount of time to allow the DUT to process the input and generate outputs.
3. **Output Monitoring and Comparison:** The testbench continuously monitors the DUT's output ports: compressed out, decompressed out, and response. The testbench compares the captured outputs with expected values based on the provided input stimuli and the DUT's specifications. The expected outputs are pre-defined for each test case, considering the operation performed (compression/decompression) and the input data provided.
4. **Error Handling:** If any mismatch is detected between the captured outputs and expected values, the testbench flags an error.
5. **Coverage Analysis:** The testbench performs coverage analysis to track the extent of verification achieved during simulation. It measures code coverage metrics, functional coverage, and assertion coverage to ensure that all parts of the design have been adequately exercised.
6. **Logging and Reporting:** Throughout the simulation, the testbench logs relevant information such as simulation time, input stimuli, DUT responses, and any detected errors. At the end of the simulation, a comprehensive verification report is generated, summarizing the test results, coverage metrics, and any issues encountered during testing.

This structured approach ensures a systematic verification process, covering various functionalities and potential scenarios. By following a defined workflow, any design bugs or unexpected behavior in the DUT can be efficiently identified and addressed.

Tests

Many tests were generated and driven to the design in order to test its functionality, including:

1. A random input test, including a random command, random input data and random index number.
2. No operation test, having a 00 command, other inputs are not important to generate since they are not used. The response output is checked to test the accuracy of the design, but other outputs are not necessary to check since the operation does not affect them.
3. Compression test, having a 01 command and an input data to be stored in the dictionary. Outputs related to the compression operation were monitored, such as the response and the compressed_out ports.
4. Decompression test, having a 10 command and an input index of the dictionary. Outputs related to the decompression operation were monitored, such as the response and the decompressed_out ports.
5. Invalid operation, having a 11 command, other inputs are not important to generate since they are not used. The response output is checked to test the accuracy of the design, but other outputs are not necessary to check since the operation does not affect them.

Simulation Results

No Operation Test

```
dut command: 00
reference response: 00
dut response: 00
success: dut respone = 00, ref response = 00
success: data in  = 0000000000000000000
success: index  = 00
success: data out  = 0000000000000000000
Driver: Received command = 01
 Coverage = 19.90 %
```

Figure 1: No Operation Test

Invalid Command Test

```
dut command: 11
reference response: 11
dut response: 11
success: dut respone = 11, ref response = 11
success: data in  = 0000000000000000000
success: index  = 00
success: data out  = 0000000000000000000
 Coverage = 19.90 %
```

Figure 2: Invalid Command Test

Driving Data for Compression

```
Driver: Driving data_in = 5d9816f458379c0057e5
```

Figure 3: Driving Data for Compression

Compression Command

data_in = data from the driver, at index 0.

```
dut command: 01
reference response: 01
dut response: 01
success: dut respone = 01, ref response = 01
success: data in  = 5d9816f458379c0057e5
success: index  = 00
success: data out  = 0000000000000000000
Driver: Received command = 01
 Coverage = 28.23 %
```

Figure 4: Data Compression at index zero.

Then, a new data was driven and compressed at index 1.

```
Driver: Driving data_in = c3bc1790b69db22aa64a
```

Figure 5: Driving new Data for a new Compression

As shown in Figure 6, the data were compressed successfully.

```
dut command: 01
reference response: 01
dut response: 01
success: dut respone = 01, ref response = 01
success: data in  = c3bc1790b69db22aa64a
success: index  = 01
success: data out  = 00000000000000000000
```

Figure 6: Data Compression at index one.

Decompression Command

By driving index = 0, and a decompression command to the design, the data was decompressed successfully as shown in figure 7.

```
dut command: 10
reference response: 10
dut response: 10
success: dut respone = 10, ref response = 10
success: data in  = 00000000000000000000
success: index  = 01
success: data out  = 5d9816f458379c0057e5
```

Figure 7 - data decompression at index 0

When driving index = 1, and a decompression command to the design, the data was decompressed successfully as shown in figure 8.

```
dut command: 10
reference response: 10
dut response: 10
success: dut respone = 10, ref response = 10
success: data in  = 00000000000000000000
success: index  = 01
success: data out  = c3bc1790b69db22aa64a
Driver: Received command = 00
 Coverage = 38.79 %
```

Figure 8- data decompression at index 1

When trying to decompress non-existing data at a certain index, an error message would appear, in addition to a response = 11 indicating an error. This case is shown in figure 9.

```
dut command: 10
data not found at this index
reference response: 11
dut response: 11
```

Figure 9- data not found

Reset test

When the reset input was set to 1, all outputs were cleared, in addition to clearing the internal dictionary and index register. The result is shown in figure 10.

```
reset = 1   resetting the dictionary and index
dut command: 00
reference response: 00
dut response: 00
success: dut respone = 00, ref response = 00
success: data in  = 00000000000000000000
success: index  = 00
success: data out  = 00000000000000000000
```

Figure 10- reset result

Coverage Plan and Reports

A lot of tests may be repetitive or not interesting, and applying them all might be exhaustive and not beneficial. Our coverage plan revolves around testing some interesting scenarios, in addition to some randomly generated tests, to verify the accuracy of our design and how it behaves under specific circumstances.

Our test cases include all possible values of the command input, which are 4 options. However, not all index input cases need to be tested. The index port (compressed_in) is an 8-bit port, so it has 255 possible value that are not all interesting to test. We chose in our coverage plan to include the tests where the index values are 0, which means that the dictionary is empty, and value 1 such that the dictionary has one element only. Other included values are when the index has a value of 255, indicating that the dictionary is full. Some random range of index values was also included in the tests just to make sure that operations are working correctly.

Same thing was done with the input data values. The data_in port is 80-bits wide, making a huge number of possible input values that are not necessary for testing. As a result, some random ranges were chosen to include in our coverage plan to reduce the number of repetitive and not interesting tests.

Our coverage plan resulted in a 38.79% coverage percentage.

Conclusion

By the end of this project, our verification environment was able to test all functionalities and corner cases of the provided compression decompression design. Our coverage model also was comprehensive of all interesting test cases. The design under testing appeared to be bug-free and all its functionalities were operating correctly with no errors.

References

[1] ScienceDirect, "Compression Algorithm". Available: https://www.sciencedirect.com/topics/computer-science/compression-algorithm. Accessed: June 5, 2024.

[2] ScienceDirect, "Decompression Algorithm". Available: https://www.sciencedirect.com/topics/computer-science/decompression-algorithm. Accessed: June 5, 2024.

[3] Mostafa, "What are SDA or Serial Data and CLK or Serial Clock Signal in a Digital Circuit," Medium. Available: https://medium.com/@mostafa_3262/what-are-sda-or-serial-data-and-clk-scl-or-serial-clock-signal-in-a-digital-circuit-83b7dca6a2b4. Accessed: June 5, 2024.

[4] S2C Inc., "Chip Design Verification," Medium. Available: https://medium.com/@s2cinc2004/chip-design-verification-93c51d705f3d. Accessed: June 5, 2024.

[5] Quality-One, "Design Verification and Process Review (DVPR)". Available: https://quality-one.com/dvpr/. Accessed: June 5, 2024.

[6] EEWeb, "Chip Design Verification: It's All About the Coverage". Available: https://www.eeweb.com/chip-design-verification-its-all-about-the-coverage/. Accessed: June 5, 2024.