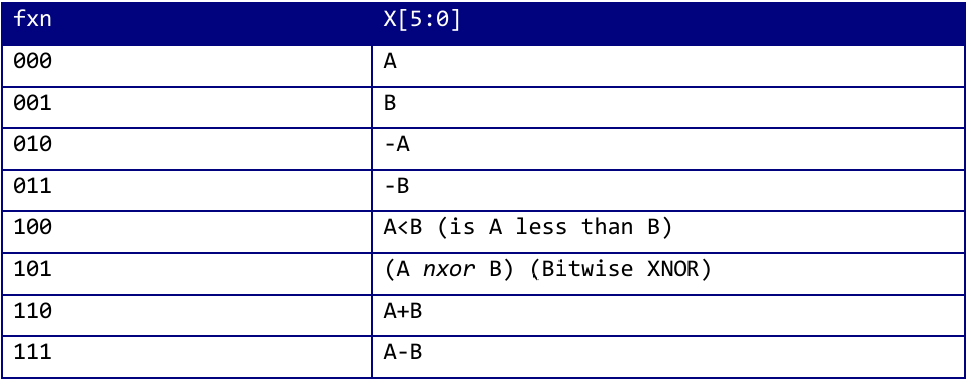
Digital System Design Assignment 1

Ruairi Mullally – 22336002

**Introduction:**

This report examines the design and implementation of multiple Verilog modules, including their architecture and testing, which is the culmination of the previous labs. Existing modules were already extensively tested and understanded and subsequently repurposed for use in the mini ALU.

**Method:**



*Figure 1. Mini-ALU connections.*

To provide the ability to select a certain function of the mini-ALU, an 8x1 multiplexer was used. The return value (X) of the mini-ALU is selected by an 8x1 multiplexer, controlled by ‘fxn’.

*Table1. Mini-ALU module re-utilization.*

|  |  |
| --- | --- |
| **fxn** | **Description of computation using existing modules.** |
| 000 | Select A from MUX (MUX is the top-level module). |
| 001 | Select B from MUX. |
| 010 | Subtract A from 0 in the ripple adder. |
| 011 | Subtract B from 0 in the ripple adder. |
| 100 | Extend A and B to 8 bits. Compute A >= B with the ‘greater than or equal’ module. Invert the output with the ‘full adder’ module to get A < B. |
| 101 | Write a **new** module to do a bitwise XOR and negate the output. |
| 110 | Add B to A with ripple adder. |
| 111 | Subtract B from A with ripple adder. |

**FXN 000:** This function return the original value of A, therefore the A input is just connected as a multiplexer input and no additional processing is needed.

**FXN 001:** This function returns the original value of B, therefore the B input is just connected as a multiplexer input and no additional processing is needed.

**FXN 010:** This function returns the negative of input A. To do this without creating a new module the 6-bit ripple adder from previous labs could be used. This was achieved by subtracting A from 0 to get the negative of A.

**FXN 011:** This function returns the negative of input B. To do this without creating a new module the 6-bit ripple adder from previous labs could be used. This was achieved by subtracting B from 0 to get the negative of B.

**FXN 100:** To implement the comparison (A < B), the “greater than or equal to” module from previous labs could be used. (A < B) is logically the opposite of (A >= B) so the output from the “greater than or equal to” module could be negated to achieve this functionality. To negate the output, I used a 1-bit full adder from the ripple adder module. By computing the original output + 1 with an initial carry-in of 0, the output is effectively negated.

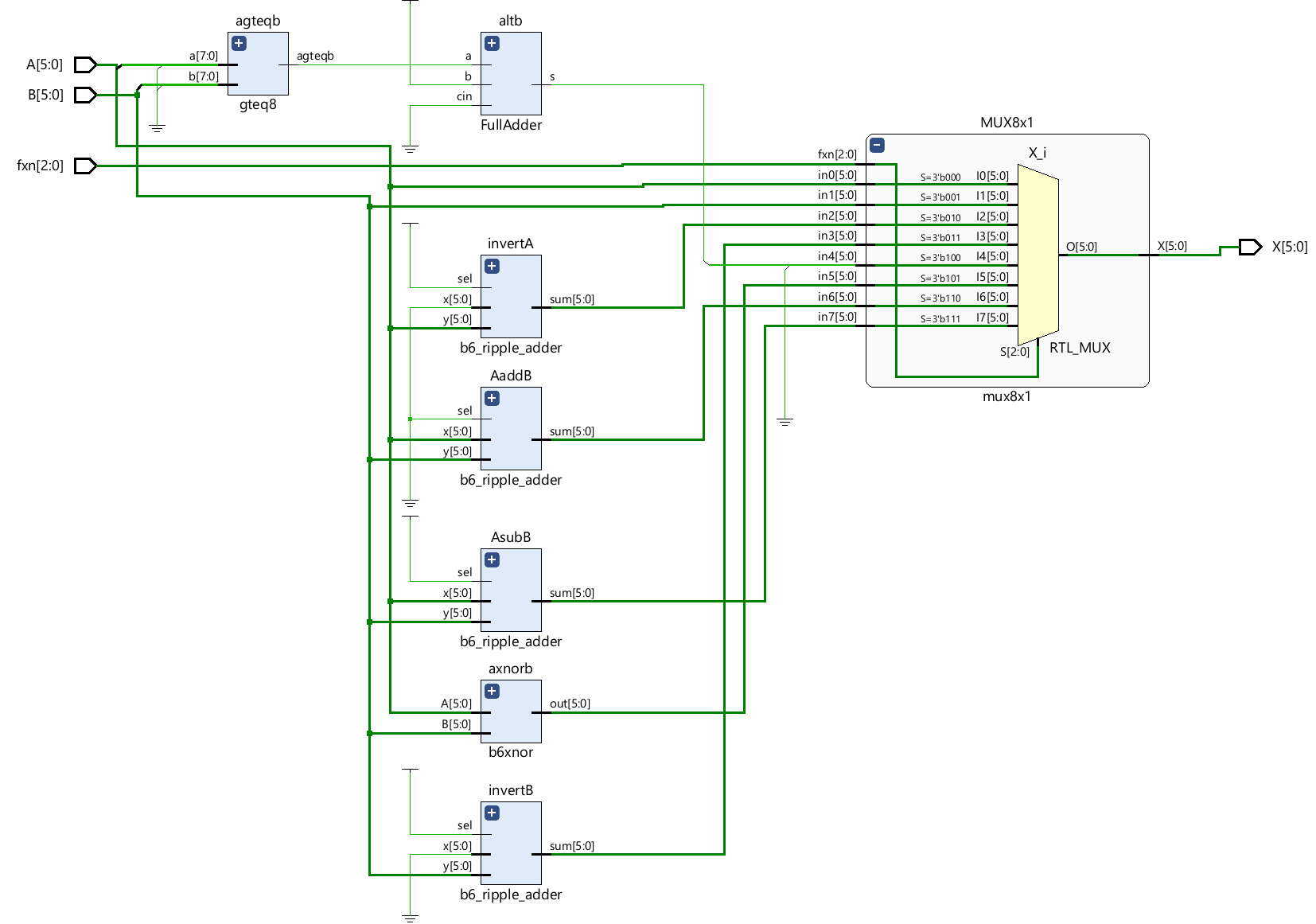
Additionally, the width of the “greater than or equal to” module was designed for 8 bits while the ALU operates on 6 bits. To adapt the “greater than or equal to” module to work, two trailing zeros were appended to the 6-bit numbers. This ensures that the sign-checking, which operates on the most significant bit, for the twos complement numbers still operates correctly.

**FXN 101:** This was the only function of the ALU that predicated a completely new module. Using the built-in Verilog functions, this module computes the bitwise XNOR of A and B.

**FXN 110:** This function computes A + B. To do this without creating a new module the 6-bit ripple adder from previous labs could be used.

**FXN 111:** This function computes A + B. To do this without creating a new module the 6-bit ripple adder from previous labs could be used.

Leveraging the use of existing modules greatly simplifies the testing of the design, as there are few modules, which have already been extensively tested. The downside, however, is that there may be inefficient resource usage. For example, in the case of inverting A, a whole ripple adder is used for something that could be done with a purpose-built inverter using less resources. For the purposes of this ALU, which aimed to re-use as much hardware as possible, this is an acceptable trade-off.



*Figure 2. Schematic diagram of mini-ALU.*

>Make MUX a module **DONE**

>Functional diagram **DONE**

>Note modifications and new modules **DONE**

**Testing:**

>Note changes to testbenches: maybe a bit on each submodule testing.

>design top level ALU testbench **The first test vector should be the last 6 bits of your board number.**

>Wave form of each function showing correct operation.

>Comment on how well-tested the ALU is and shortcomings

>How robust to unexpected inputs?

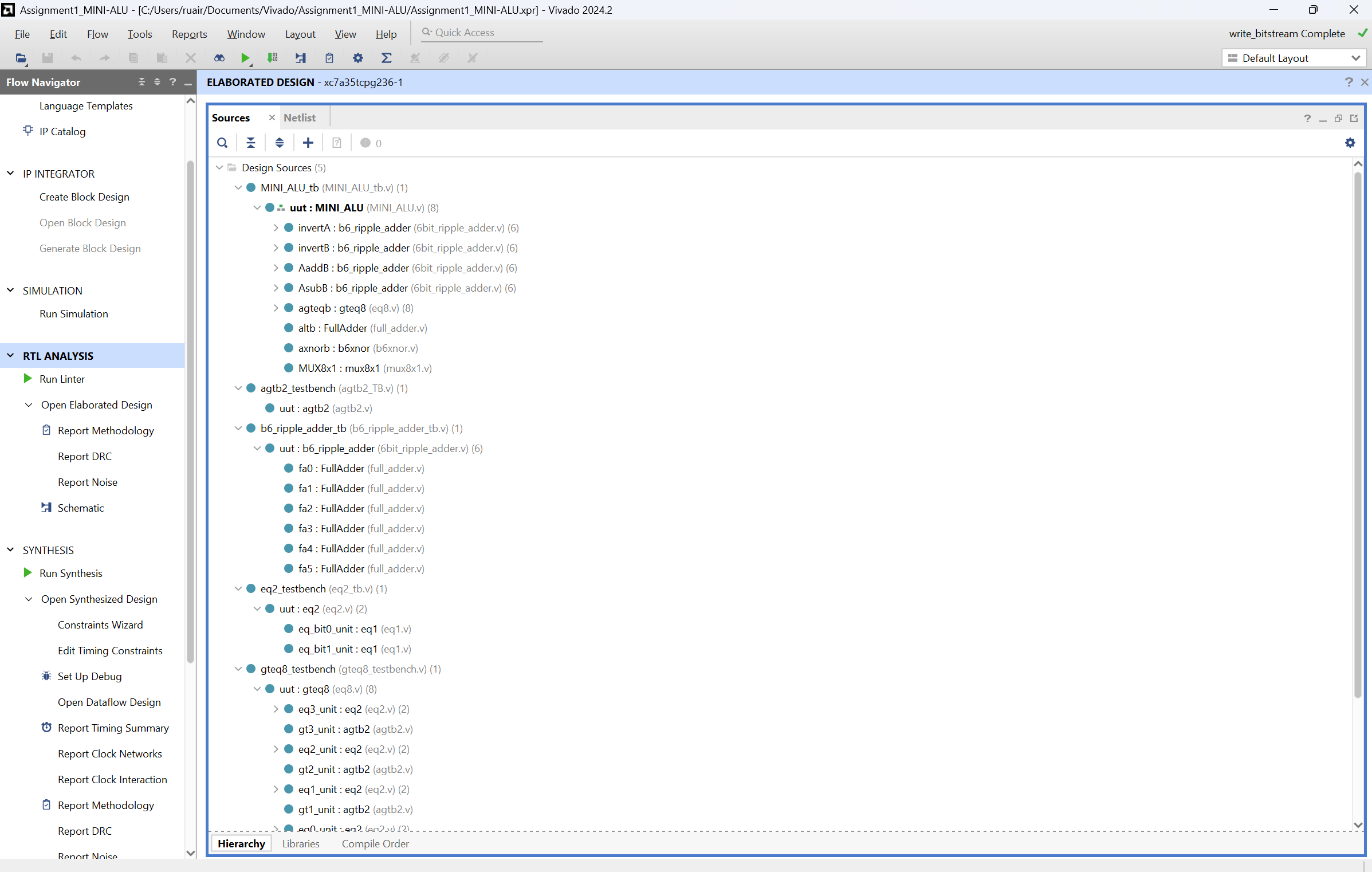
**Results:**>Show function working on board using peripherals

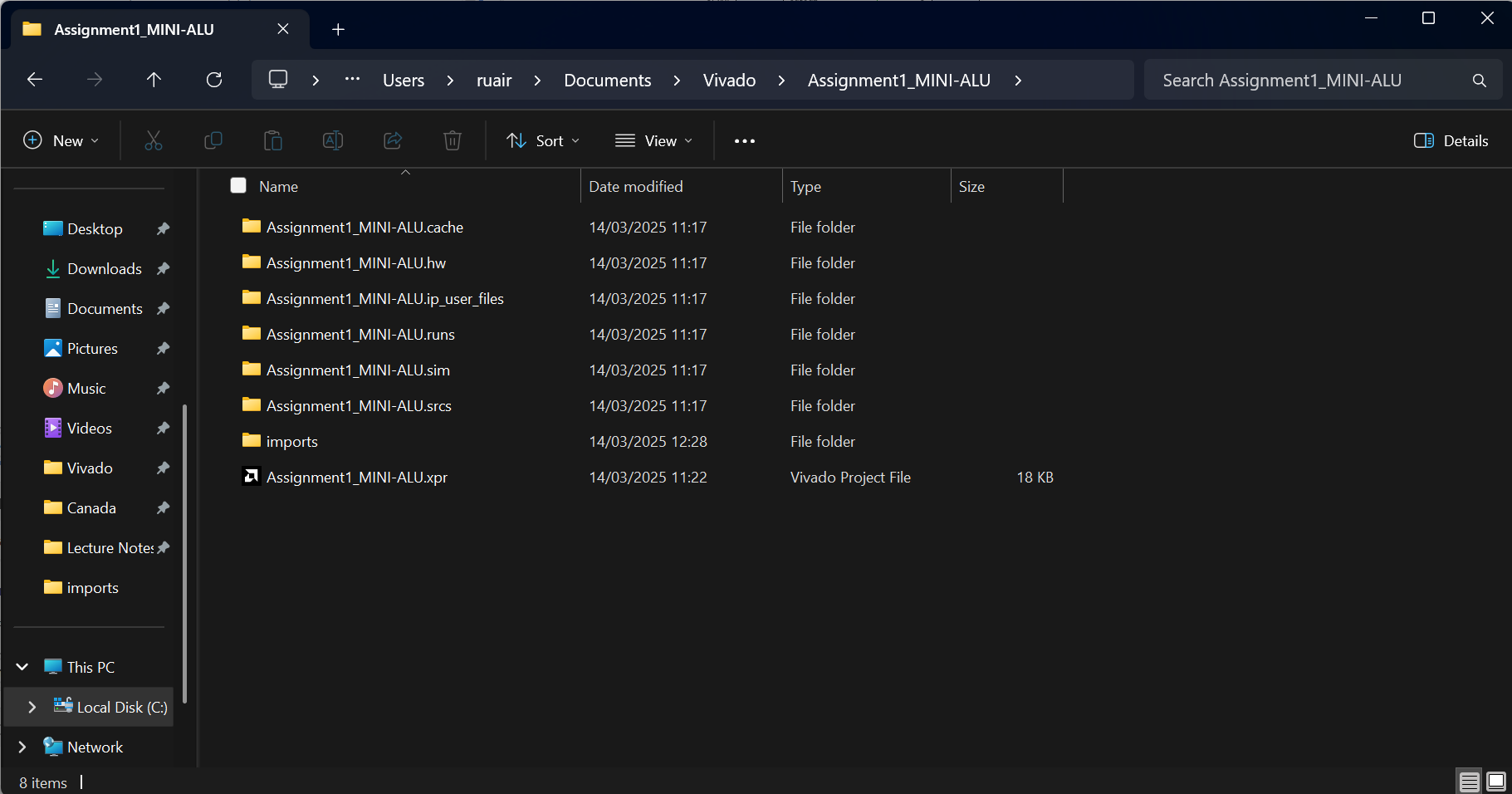
>usage

>How to demo/use

**Other:**

>Screenshots of file hierarchy in vivado and file explorer





>Zip file containing modules and bitfile