Adler 32 Accumulator

Ben Heard

April 11, 2021

Abstract

While a larger system to compute Adler32 checksums as an offload engine and conforming to input and output interface specifications is interesting, it is beyond the scope of a homework assignment. This document outlines an assignment meant to jumpstart the development of a larger offload engine by working through an implementation of just the base modulo sum accumulator that makes up the Adler32 datapath.

1 Description

In this assignment you'll produce a design and implementation of the accumulator at the heart of the Adler32 checksum engine. As always you are encouraged to draw out the design before attempting to implement it in Verilog. However, it is only your Verilog that will be assessed.

Since this is just the accumulator portion there is no control of the data coming in nor any tracking of sizes/bytes that are in the project assignment. Instead, the general function of this design is to reset to a known state (specifically, A=1 and B=0) and, out of reset, accumulate every clock cycle.

1.1 Input Interface

The input interface consists of a synchronous active low reset and the 8-bit data. Out of the reset your system should set the value of A to 1 and the value of B to 0. Then, every clock cycle that reset is not asserted your design shall accumulate the data byte as

$$A_{new} = (A_{old} + data)$$
$$B_{new} = (B_{old} + A_{new})$$

1.2 Output Interface

There is no formal output interface. The intent is that your accumulator concatenate {B, A} and always present that to the output. So, after reset your output would be $32'h00000001$ and each clock it would update to a new value based on the input data.

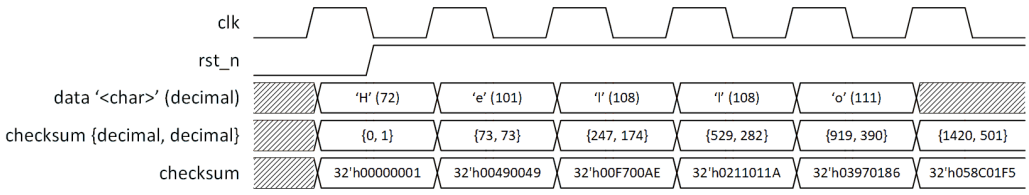
1.3 Requirements

Your implementation must conform to the following requirements.

1. The module shall be named **adler32_acc**.
2. The module shall have the following ports
   - input **clk**
   - input **rst_n**
   - input **data**[7:0]
   - output **checksum**[31:0]
3. An active low synchronous assertion of **rst_n** shall set the output to $32'h00000001$ in accordance with the Adler32 specification.
4. At each rising clock edge the accumulator will sum the input data with the value in the 16-bit register A, and then sum that result with the value in the 16-bit register B.
5. There is no end to the data; the accumulator shall continue to sum the input data as long as rising clock edges appear.

1.4 Example

The following timing diagram shows an example operation with the "Hello" string that is part of the project.



2 What to Turn In

2.1 Moodle Assignment

Submit a schematic drawing of your design. You needn't draw to the gate-level detail; structures that are easily implemented in Verilog are sufficient. E.g. you can use a circle with a + in it to represent an adder since that will be a dataflow assignment or procedural block. E.g. you can draw a single 2:1 mux even when the buses in/out are multi-bits wide.

2.2 Git

Push your implementation to GitHub for grading. Provide the SHA that you would like graded with the Moodle submission above. If none is provided, we will grade the latest commit before the due date for the assignment.

3 Grading

Grading will be based on the following criteria.

- Was anything submitted?
- Did it compile with vlog *.v without errors or warnings?
- Did it load into the simulator with the provided example testbench?
- Did it run with the provided example testbench?
- Did it produce correct results with the provided example testbench?
- Did it run to completion with the hidden testbench(es)?
- Did it produce correct results for the hidden testbench(es)?

3.1 Grading Notes

There are a few things to keep in mind when submitting your files for grading. Follow these rules to ensure that your assignment is graded.

1. All Verilog files necessary for the assignment shall be in the homework 7 directory, not a subdirectory.

2. All files needed for your implementation shall match the glob *.v. This just means that if there are Verilog files needed for your design you will name them with the .v extension.

3. The module shall be named **adler32_acc**. The filename may be whatever you choose but the module name has been specified.

4. The module ports shall be as above. The order doesn't matter but the names and directions must match.

4 Suggestions/Notes

Note that the assignment does not mention the modulus part of the accumulator that will be a requirement in the project. It is not necessary to implement it for this assignment. However, you will be required to implement it for the project. There is no extra credit for implementing the modulus in this assignment but I would recommend it to get an ever better head start on the project.

If you choose to implement the modulus capability ensure that it is done using the suggestions in the project and not with the modulus (%) operator.

To help with this assignment and the project you may generate Adler32 checksums using an online calculator such as the one found at

`https://hash.online-convert.com/adler32-generator`