Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 09, September-2023

Design and Implementation of a Protocol-Agnostic Serial Bus Analyzer for Real-Time Waveform Debugging and Verification

Harshal Subhash Advane
Corporate Trainer

***Abstract*— Serial bus analyzer presented here is generic serial bus debugger which can basically monitor single input line for any serial bus protocol and displays the frames/state and values both big and little endian in a hex format on the waveform like gtkwaves, VCS etc. The design is protocol agnostic, synthesizable and configurable for multiple instances.**

Keywords-*Serial Bus protocol, debugger, logic analyzers protocol, I2C protocol, CAN protocol, JTAG.*

I. INTRODUCTION

Debugging serial bus protocols can be a daunting task, especially when dealing with complex communication systems. A single error can cause a domino effect, resulting in cascading failures that are difficult to diagnose and fix. In such cases, a generic debugger can be a lifesaver.

A generic debugger is a tool that allows you to monitor and analyze the data traffic on a serial bus. It can help you pinpoint errors and track down their root causes quickly and efficiently. Additionally, a generic debugger can help you validate that your code or hardware design is functioning correctly.

When debugging serial bus protocols, it is essential to capture the data traffic on the bus in real-time. This is where a tool like a logic analyzer or an oscilloscope comes in handy. These tools can capture the data traffic on the bus and display it in a waveform or a table format. However, they only provide raw data, and interpreting this data can be a time-consuming task.

A generic debugger solves this problem by analyzing the captured data traffic and presenting it in a more user-friendly format. It can decode the data traffic and display it in a human-readable format, allowing you to quickly identify errors and anomalies. Additionally, a generic debugger can provide real-time notifications of errors, making it easier to identify and troubleshoot issues.

II. PROBLEM STATEMENT

We have lots of serial bus protocols in our Chips, we take lots of 3rd party serial bus protocols like CAN, SPI, I2C, eMMC, JTAG etc.

We need to run the simulation for verification purpose and need to make sure protocol is working correctly. Each serial protocol uses different frame format as per there need and requirements.
Also, while debugging we need to note down each bit looking at the waveform and cross check it with protocol to understand if its correct or incorrect.

III. SOLUTIONS

The solution that I have developed is a generic serial bus protocol analyzer which can be used across any serial bus protocol and can have parameterized frame format and can have multiple instances of the debugger.

It will basically give a visual display of frames in hex format which would be easier to debug.

This way we don't have to deal with each bit on serial bus and can save the debug time resulting in the many man hours of saving.

One of the significant advantages of a generic debugger is that it can be used to debug a wide range of serial bus protocols. It is not tied to a specific protocol or vendor and can be adapted to different protocols and configurations. This makes it a versatile tool that can be used in various debugging scenarios.

IV. ADVANTAGE

- It saves a lot man hour of debugging time.
- It can be used agnostic to any serial bus protocol.
- The output can be capture on the text format for text-based debugging.
- It's written inside interface hence can be used with any tools.

V. STEPS TO CONFGIURE

- Instantiate the serial bus debugger interface inside your testbench top.
- Declare the set the parameter no_of_state as how many no of states are there your serial bus protocol
- Declare the wait_table in the following format which basically signifies the number of bit in each states
- Assign the signal start = 1 from the time period you want the debugger to analyses the input signals

VI. SAMPLE EXAMPLE CODE

a) Declare the wait_table as mention in figure 1 and

figure 2 here the left-hand side number denotes the frame / state and right-hand side denote the number of bits in the frame / number of wait state inside the state.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 09, September-2023

For e.g., In an imaginary “My” serial bus protocol there are 7 data frames, and each data frame can contain 8 bits of data the wait_table can be declared as follows.

```
parameter [7:0] wait_table[6:0] = '{
        {6,8},
        {5,8},
        {4,8},
        {3,8},
        {2,8},
        {1,8},
        {0,8}
      };
```

Fig. 1. Example of a wait_state for “My” serial bus protocol..

In “Me” serial bus protocol if there are 2 data frames and 1st frame is of 1 bit and 2nd frame is of 64 bits it will be declare as follows

```
parameter [7:0] wait_table1 [1:0] = '{
        {1,63},
        {0,1}
      };
```

Fig. 2. Example of wait_state for “Me” serial bus protocol.

b) Instatitiating the serial debugger

In the second steps we need to do the installation of the serial_debugger interface inside your testbench top

```
 serial_debugger #
(.no_of_state(7),       // No of total states in serial protocol
 .wait_table(wait_table))// Wait_table declare in step 1.

        la0 (.clk(clk),          // Input clk signal
             .rst(rst),          // Input rst signal
             .in (in),           //  Input serial bus signal
             .start_la(start));  // Input signal to indicate start of
analyser
```

c) Define the start signal

Define the signal start which can be use by serial_debugger from the time where to start analysing the incoming signals.

```
    initial begin
      int total_delay = wait_table.sum();
      $display (" total_delay %0d",total_delay);
      @(negedge in)
        start = 1;
      repeat(total_delay) @(posedge clk);
        start = 0;
      #1300 $finish();
    end
```

VII. SAMPLE OUTPUT

The sample output waveform is shown in the

Fig 3 and Fig 4

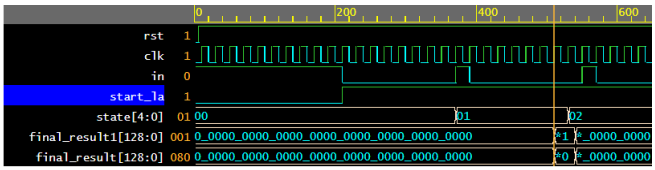

Fig 3. Output of state 01 is value ‘h01 (‘h80)

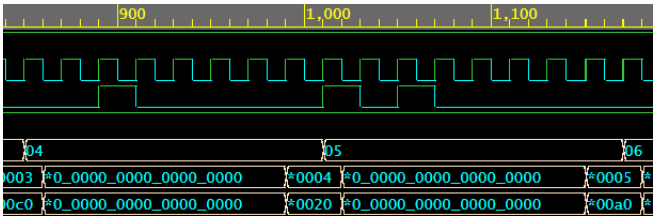

Fig 4. Output of state 04 is value ‘h04(‘h20) and output of state 05 is ‘h05 (‘ha0)

VIII. FEATURE OF INTERFACE

- It can be used with any serial bus protocol
- It been developed inside interface which can be ported in design and testbench
- The frame format is customizable hence you can increase and decrease the size as needed
- The output is shown in big and little endian format
- It shows the bits captured against the each frame
- The interface is synthesizable which can help in debugging the actual silicon bugs.

ACKNOWLEDGMENT

The work reported in this paper is developed on free open source website www.edaplayground.com.