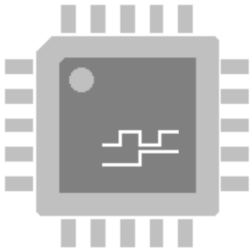
openVeriFLA

open source FPGA logic analyzer



version 2.1

Reference Manual

Developed by

Laurentiu-Cristian Duca
laurentiu.duca@gmail.com

Table of Contents

1. Introduction

openVeriFLA is an FPGA logic analyzer written in Verilog and Java.
It helps in on-board testing and debugging of the FPGA projects.
This is done by real-time capturing and then graphically displaying
the signals transitions that happen inside the FPGA chip.
Having a didactic scope, it is designed for small projects.
Please note that learning to use this logic analyzer may require more than an hour, while understanding the source code may require a few days.

openVeriFLA is distributed under the GNU GPL license (the UART sources have a more generous license – written in the source code). It can be downloaded from https://opencores.org/project/openverifla.

Keep the sources near you, as it will be referenced in this documentation.

2. Concept

The main architecture of the openVeriFLA logic analyzer is shown in the figure below. The logic analyzer has two sides, the FPGA part and the host computer one. These communicates via the host computer interface cable to the FPGA board.

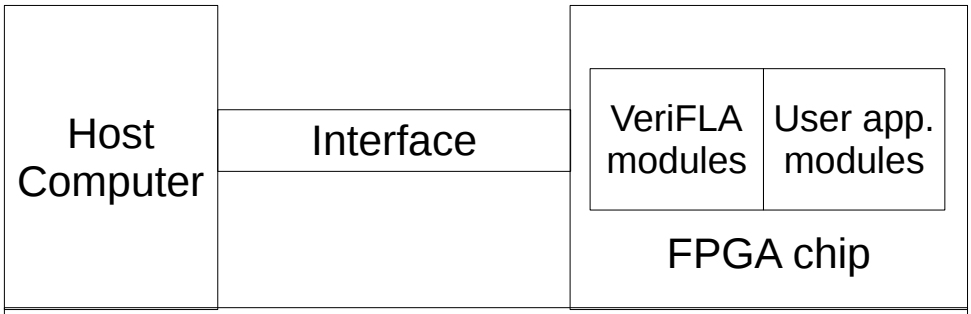


Fig. Main architecture of the logic analyzer

The openVeriFLA FPGA modules are implemented in Verilog HDL.
In order to use the logic analyzer, these modules must be implemented in the FPGA chip along with the user application. The openVeriFLA modules capture the signal transitions of the monitored lines and send the data capture to the host computer for graphical visualization and future analyze.

The host computer part of the application is implemented in the Java language. The java application receives the captured data and saves it on the disk in a file named *capture.v*. This file is a behavioural verilog HDL file. An Verilog HDL simulator with a graphical viewer for the signals is necessary in order to simulate *capture.v* and view the captured data.

The interaction between FPGA and the host computer is illustrated in the figure below. For now, important is the fact that the host may send the run command to the monitor, in order to start a new capture and send it back.

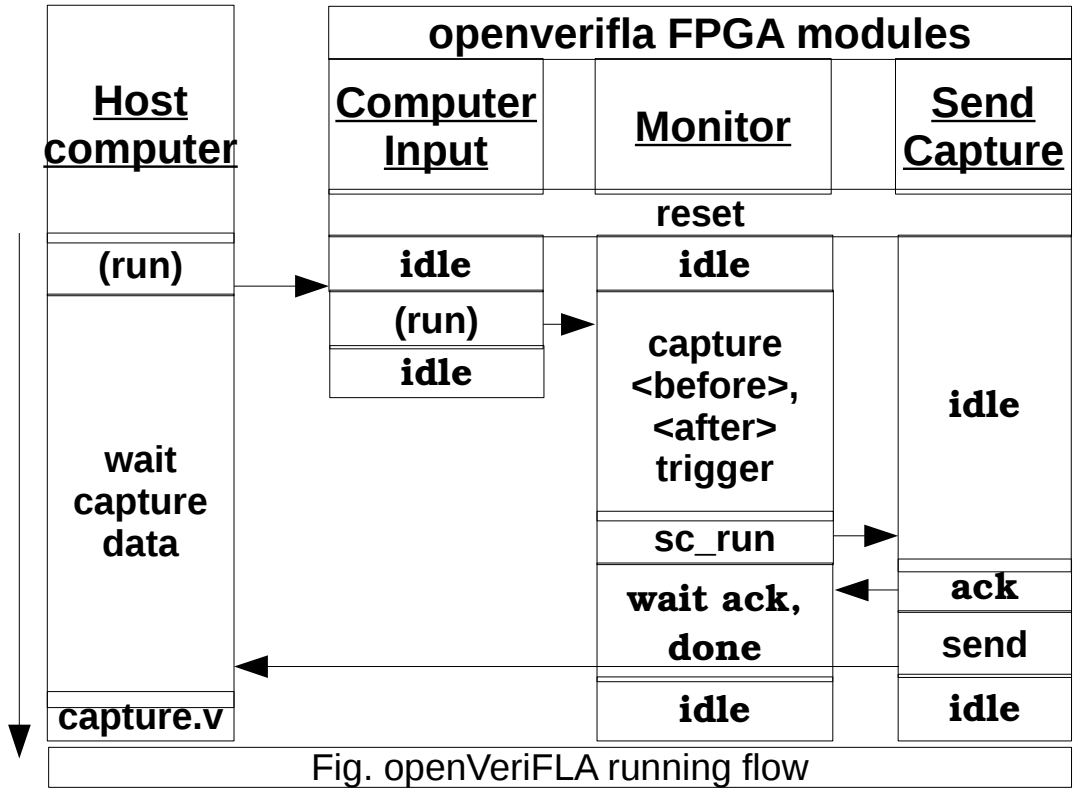


Fig. openVeriFLA running flow

As shown in this figure, the FPGA side of the logic analyzer is made by three components. These are:
- the monitor module which handles the data capturing process
- the computer-input and send-capture modules which handle the high level part of the communication between FPGA and host computer
- the UART modules (not shown here) particular to the host computer-FPGA interface protocol.

The data captured from the monitorized lines is kept in a memory buffer that must be available for the openVeriFLA modules. The memory buffer that comes with openVeriFLA by default is allocated from the FPGA configurable part and is implemented in *memory_of_verifla.v*. It can be changed if one wants to, with a memory buffer that may be allocated from the FPGA block memory. Block memory may be reserved from the FPGA by using specialized tools like Xilinx IP Core Generator from Xilinx ISE WebPack.

The memory buffer used for storing data is organized as in the figure below. A special moment in the process of data capturing is the trigger event. This is the moment when signals of the monitored lines match a user defined value. Before the trigger event, the data is stored in a circular FIFO queue named “before trigger queue”. At the end of the memory buffer it is stored the pointer to the tail of the circular queue. After the trigger event, the data is stored in a standard FIFO. When the

“after trigger queue” is full, the data capture is sent to the host computer, where the user will analyze it.

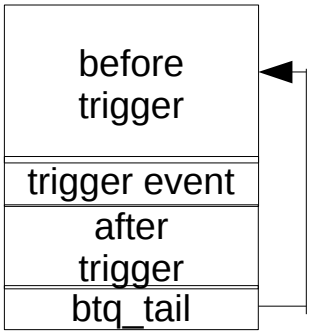


Fig. Memory organization

A memory word contains a captured data line and the time that this data line is constant. There are also reserved words which may specify:
- an empty and not used memory cell (LA_MEM_EMPTY_SLOT)
- the pointer to the tail of the before trigger queue which is stored in the last memory word.

The memory size and memory word length are parameterizable. The control-panel of the logic analyzer is the *common_internal_verifla.v* file. The other parameters of this file will be explained later.

3. Prerequisites

In order to test openVeriFLA, one will need as hardware a FPGA board and a PL2303TA USB to TTL serial converter.

Fig. PL2303TA USB to TTL serial converter (image from ebay.com)

Only three pins from the FPGA board, named GND, TX and RX will be

connected to (in order) black, white and green (because is cross-over connection). So, the TX from the FPGA board is connected to the RX cable wire and vice versa. This way, the FPGA board is connected to a host computer. The red cable wire (+5V) remains unconnected.

At the software level, Windows or Linux with Java JDK (or at least Java JRE) and the FPGA development program (like Xilinx ISE) is needed.

4. Case study: using openVeriFLA in a keyboard driver

4.1. Keyboard simplified protocol

In the following example, a PS2 keyboard driver implementation is verified.
The keyboard protocol is presented in the figure below.
When no event occurs, the clock and data lines are idle – being hold as high. When a key is pressed or released, the key-code is sent on the data line, bit by bit. It is preceded by the start bit and followed by the parity and stop bits. This is done synchronously with the clock signal:
- when the CLOCK signal is low (0), DATA has a stable value
- when the CLOCK signal is high (1), on the DATA line it may be a signal transition.

The key-codes are different by the ASCII codes. For example, 'a' has 0x1C
and its bits will be sent in reverse order (with least significant bit first): 0 0 1 1 1 0 0 0.

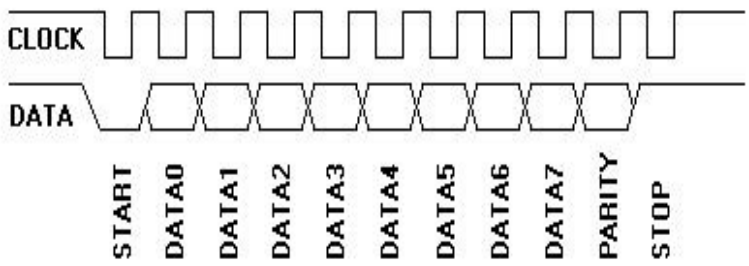


Fig. The keyboard protocol

4.2. Using openVeriFLA in a HDL module

A keyboard driver implementation is shown in the figure below. The module *top_of_verifla* is instantiated in the *keyboard* module. This way, it will be implemented in the FPGA chip along with the user application.
The signals to be captured are *kbd_data_line*, *kbd_clk*, and *kbd_key*. When a key is pressed, its keyboard-code bits are sent on the data line, conforming to the keyboard protocol shown above. The implementation of the keyboard module deserializes the data bits and then places the keyboard-code in the *kbd_key* octet register. Instantiating the openVeriFLA top module in the keyboard driver is shown in the figure below.

```
module keyboard(kbd_data_line, kbd_clk, kbd_key,
        clk, reset,
        //top_of_verifla transceiver
        uart_XMIT_dataH, uart_REC_dataH
);

input clk, reset;
//top_of_verifla transceiver
input uart_REC_dataH;
output uart_XMIT_dataH;

// App. specific
input kbd_data_line, kbd_clk;
output [7:0] kbd_key;     // register for storing keyboard data

// … Code truncated for simplicity. For details please check keyboard.v

// VeriFLA
top_of_verifla verifla (.clk(clk), .rst_l(!reset), .sys_run(1'b1),
                        .data_in({cnt, kbd_data_line, kbd_clk, kbd_key}),
                        // Transceiver
                        .uart_XMIT_dataH(uart_XMIT_dataH),
                        .uart_REC_dataH(uart_REC_dataH));

endmodule
```

Fig. Instantiating openVeriFLA in the keyboard driver module

One must instantiate *top_of_verifla* module and pass the following signals to openVeriFLA:
- *clk*, which is the board clock
- *rst_l*, which is the *top_of_verifla* reset signal and is active low
- *sys_run*, which instructs openVeriFLA whether to immediately start a data capture or wait for the user *run* command
- *data_in* which contains the signals from the keyboard module that will be captured (here these are *cnt*, *kbd_data_line*, *kbd_clk* and *kbd_key*)
- *uart_XMIT_dataH* which is the openVeriFLA serial transmission line (TX)
- *uart_REC_dataH* which is the openVeriFLA serial reception line (RX)

The signal transitions are captured on-the-fly by the openVeriFLA modules and then will be sent to the host computer, where will be prepaired to be graphically displayed.

The FPGA board clk frequency (in Hz) must be written in the *inc_of_verifla.v* file before synthesis; this is required by the UART modules.

Note that openVeriFLA samples data @(posedge clk).

Part of the openVeriFLA synthesis report of the Xilinx ISE tools is shown in the table below.

Xilinx Spartan 3E 1600	
Minimum clock period:	13.826 ns
Number of Slices:	13% (1924)
Number of Slice Flip Flops:	7% (2290)
Number of 4 input LUTs:	5% (1540)
Number of bonded IOBs:	5% (14)
Number of GCLKs:	4% (1 of 24)

Table. The FPGA occupied resources

4.3 The host computer Java application

First, the *Verifla.java* source must be compiled by running *compile.sh* on Linux (with bash) or *compile.bat* on Windows; this will generate the *VeriFLA.class*. In order to receive the grabbed data from the FPGA chip, the *VeriFLA.class* is run on the host computer. The communication with the openVeriFLA modules is made via the usb-to-serial interface between the host computer and the FPGA development board; the *VeriFLA* class uses the *jssc.jar* UART library. This way, the signals capture will be sent to the host computer and saved in a form which can be displayed graphically.

On Linux, the *VeriFLA.class* is run with the following command (on Windows, one must replace *sudo ./run.sh* with *run.bat*):

```
$ sudo ./run.sh VeriFLA verifla_properties_keyboard.txt
or
$ sudo ./run.sh VeriFLA verifla_properties_keyboard.txt 1
```

Fig. How to run the *VeriFLA.class*

This scripts include in CLASSPATH the path to *jssc.jar*.

In the first case, *VeriFLA* waits for data captured to arrive on the UART serial line, while in the second case, it first sends to the FPGA the command *run* which instructs it to start a new capture and send it on the UART serial line.
After the class is run as shown, the openVeriFLA modules are instructed to start a new capture and after the capture is finished, to send the capture to the host computer.

Now, these modules wait for signal events on the monitorized lines.
If a key is pressed - for example 'a', the keyboard module implementation does its job and the openVeriFLA modules make the capture on-the-fly and send it to the host computer.

The java application gets the capture and builds the *capture.v* verilog file. After this, the *capture.v* can be added and simulated in a verilog simulator (e.g. Xilinx ISE or Icarus) project. The result is shown in the figure below.

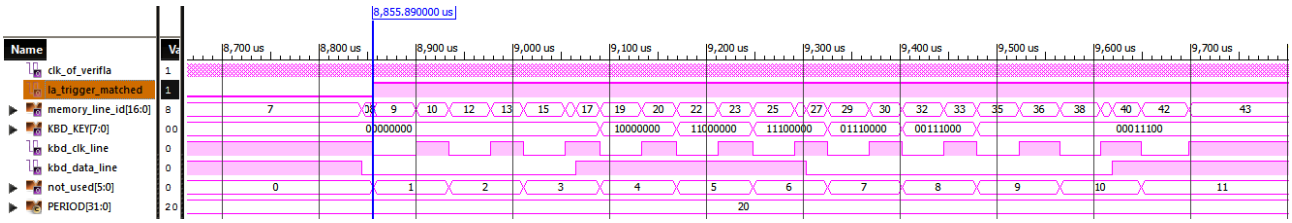


Fig. Simulation of *capture.v*

Here, the *clk_of_verifla* has the PERIOD of 20 ns. *KBD_KEY*, *kbd_clk_line*, *kbd_data_line* and *not_used* are the captured signals. It can be seen, that, at the end of capture, *KBD_KEY=8'h1c*, which represents the 'a' key. The *la_trigger_matched* shows the moment when the trigger event appeared and *memory_line_id* is the index in the captured data memory (used as debug).

The development-board clock is used as the openVeriFLA modules clock; here, it has a frequency of 50 MHz. The frequency of the keyboard clock is about few Khz. So, in the *capture.v* simulation, *run 1000000 ns* command was necessary, to reach the *$stop* instruction of the *capture.v*.

5. Configuration parameters

5.1 The host computer application parameters

The java application takes its parameters from a properties file. This file contains general parameters and application-specific parameters, like the names of the signals to be captured. An example is the *verifla_properties_keyboard.txt* file which is tuned for the keyboard driver example. Each parameter name starts with "*LA.*" (here this prefix is trimmed). The important parameters are:

- the UART serial *portName* and *baudRate;*
- *memWords* represents the size of the memory used to store the capture
- data input width and indentical samples bits (clones) must be multiples of 8 and are stored in *dataWordLenBits* and *clonesWordLenBits*
- the index in memory where the trigger event appeared is stored in *triggerMatchMemAddr*; it also delimits the before and after trigger queues
- the verilog signals passed to *top_of_verifla* module are grouped. Each group of signals is defined by a number of group parameters. First is *groupName* which should be the same as the verilog signal name. The *groupSize* represents the number of the signal lines in the group and is the same with the size of the verilog signal. Sum of the *groupSize* parameter from all groups must be equal to *totalSignals.* The *groupEndian* specifies if the data represented by the group is in big-endian or low-endian format. Each group has an unique id specified after at the end of the each parameter.
- *timescaleUnit*, *timescalePrecision* used for the verilog timescale line in *capture.v* and *clockPeriod* is the period of the development board clock

5.2. The FPGA parameters files

The clock frequency of openVeriFLA and the UART baudrate must be set in the *inc_of_verifla.v* file. This is used by the UART modules to compute the *uart_clk*. If the clock frequency of openVeriFLA is lower than 50 Mhz, then the baudrate must be lower than 115200 (for example 9600).

The control-panel of the logic analyzer is the *common_internal_verifla.v* file. It contains the configurable parameters of the logic analyzer.

- LA_MEM_WORDLEN_BITS represents the length in bits of a memory word; it is made of LA_DATA_INPUT_WORDLEN_BITS (the length in bits of data input) and LA_IDENTICAL_SAMPLES_BITS (the length in bits of the identical samples number) which means the number of clock periods that the data remains constant;
- considering that a new capture could be requested after an old one, the parameter LA_MEM_CLEAN_BEFORE_RUN specifies wheter the memory should be cleaned out between different captures
- if it is necessary to start capture at boot and the capture would not fill up the whole memory, the parameter LA_INIT_MEM_AT_RESET can be enabled – and in this situation the memory will be cleaned at reset, but the compile time increases significantly;
- LA_MEM_EMPTY_SLOT is the value that sets every memory line when cleaning the memory
- LA_TRIGGER_MASK specifies the bits to be considered when checking for the trigger value; it is used to mask the LA_TRIGGER_VALUE and the capture data when these two are compared.
- LA_TRIGGER_MATCH_MEM_ADDR is the index in memory where the trigger event appeared;
- when the memory is full or it were captured LA_MAX_SAMPLES_AFTER_TRIGGER samples, the data capture is sent to the host computer.
- in order to represent an interval of time slots when the monitored lines are constant, the parameter LA_MAX_IDENTICAL_SAMPLES is the maximum identical samples number allowed to be stored in a memory word (it is built on LA_IDENTICAL_SAMPLES_BITS).

6. Change log

2.1.g

- instructions regarding the size of BAUDRATE param of UART
- add parameter *baudRate* to openVeriFLA java properties file
- rename of IDDLE in IDLE states in UART and 3’d07 in 4’d07 for u_rec

2.1.e

- user readable form for single_pulse_of_verifla.v
- LA_MEM_CLEAN_BEFORE_RUN is a parameter now (not a `define)

2.1d
- all openVeriFLA modules and HDL files end with “_of_verifla”

2.1
- LA_INIT_MEM_AT_RESET
- single clock shared by VeriFLA and UART

2.0
- new memory layout
- new UART drivers