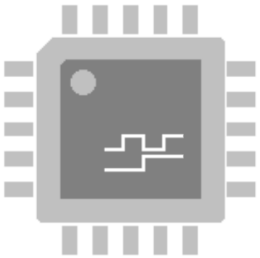
openVeriFLA manual

version 2.4



Laurentiu-Cristian Duca

openVeriFLA manual

Contents

openVeriFLA logic analyzer

1 openVeriFLA architecture

openVeriFLA is an FPGA logic analyzer. This project 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, openVeriFLA is designed and tested on and for small projects.

openVeriFLA is distributed under the GNU GPL license (the UART sources have a more generous license – written in the source code). The host computer software is written in Java, so it is platform independent. The HDL code is written in Verilog and VHDL, in both languages being fully supported.

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 USB-to-UART interface cable to the FPGA board.

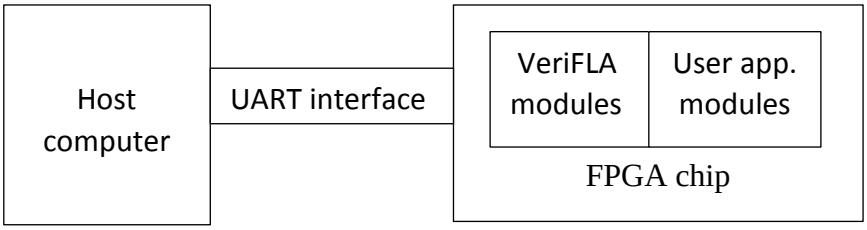


Fig. 1. Interfacing the logic analyzer

In order to use the logic analyzer, the openVeriFLA FPGA 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 behavioral (simulation) Verilog HDL file. A 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 Fig. 2. 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.

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 the host computer

- the UART modules (not shown here) particular to the host computer-FPGA interface protocol.

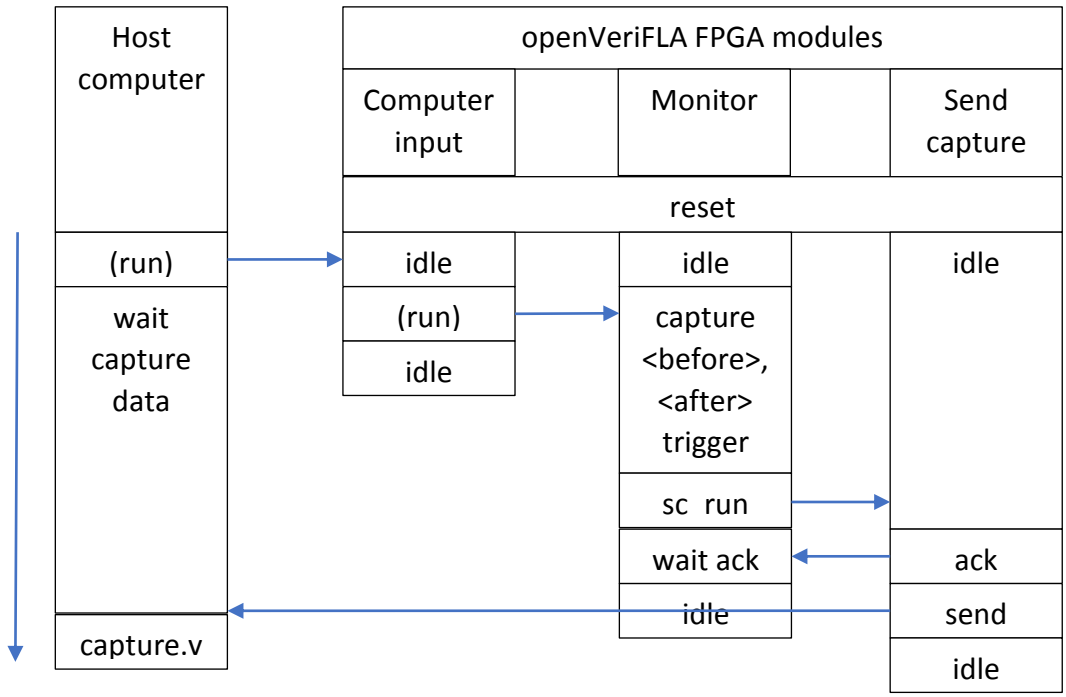


Fig. 2. openVeriFLA running flow

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 implemented in memory_of_verifla.v. The memory is kept in the mem array and addrb is used for reading and addra for writing.

```
module memory_of_verifla (clk, rst_l, addra, wea, dina, addrb, doutb);

`include "common_internal_verifla.v"

input rst_l, clk, wea;
input [LA_MEM_ADDRESS_BITS-1:0] addra, addrb;
output [LA_MEM_WORDLEN_BITS-1:0] doutb;
input [LA_MEM_WORDLEN_BITS-1:0] dina;

reg [LA_MEM_WORDLEN_BITS-1:0] mem[LA_MEM_LAST_ADDR:0];

//assign doutb = mem[addrb];
// This works too as a consequence of send_capture_of_verifla architecture.
reg [LA_MEM_WORDLEN_BITS-1:0] doutb;
always @(posedge clk or negedge rst_l)
if(~rst_l)
```

```
        doutb <= LA_MEM_EMPTY_SLOT;
else
        doutb <= mem[addrb];

always @(posedge clk)
begin
                if(wea) begin
                        mem[addra] <= dina;
                end
end
initial begin:INIT_SECTION
        integer i;
        for(i=0; i<=LA_MEM_LAST_ADDR; i=i+1)
                mem[i] <= LA_MEM_EMPTY_SLOT;
        //$readmemh("mem2018-2.mif", mem);
end
endmodule
```

Listing 1. memory_of_verifla.v

The memory buffer used for storing data is organized as in Fig. 3. 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.

A memory word contains a value of the captured data lines and the time that these data lines are 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.

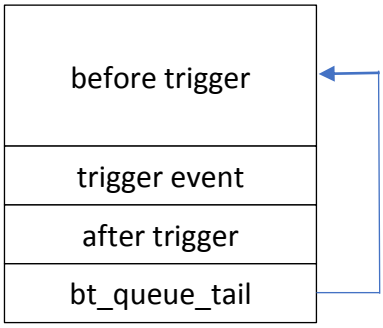


Fig. 3. Memory organization

2 Application - Simple counters capture

In order to test openVeriFLA, will need as hardware an FPGA board and a PL2303TA USB to TTL serial converter cable (connect only Rx, Tx and GND wires, without the 5V wire).

Instantiating the openVeriFLA top module in a HDL module is shown in the listing below. Please note that cntb and cnta can have arbitrary width.

```
module counters(cntb,
        clk, reset,
        //top_of_verifla transceiver
        uart_XMIT_dataH, uart_REC_dataH);

input clk, reset;
output [7:0] cntb;
//top_of_verifla transceiver
input uart_REC_dataH;
output uart_XMIT_dataH;

// Simple counters
reg [7:0] cntb, cnta;
always @(posedge clk or posedge reset)
begin
        if(reset) begin
                cntb = 0;
                cnta = 0;
        end else begin
                if((cnta & 1) && (cntb < 16'hf0))
                        cntb = cntb+1;
                cnta = cnta+1;
        end
end

// VeriFLA
top_of_verifla verifla (.clk(clk), .rst_l(!reset), .sys_run(1'b1),
                                .data_in({cntb, cnta}),
                                // Transceiver
                                .uart_XMIT_dataH(uart_XMIT_dataH),
                                .uart_REC_dataH(uart_REC_dataH));
Endmodule
```

Listing 2. Instantiating openVeriFLA in the counters 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 counters module that will be captured
- uart_XMIT_dataH which is the openVeriFLA serial transmission line (similar to txd_o from the UART chapter)
- uart_REC_dataH which is the openVeriFLA serial reception line (similar to rxd_i from the UART chapter)

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

The FPGA board clock 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 (for the counters example):

Xilinx Spartan 3E 1600	
Minimum clock period:	9.089 ns
Number of Slices:	2% (394)
Number of Slice Flip Flops:	1% (242)
Number of 4 input LUTs:	2% (677)
Number of bonded IOBs:	4% (12)
Number of GCLKs:	4% (1 of 24)

Table 1. openVeriFLA FPGA required resources

The host computer should have at least Java Runtime Environment installed (JRE) for running already compiled code or recommended the Java Development Kit (JDK) for compiling. 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_counters.txt
or
$ sudo ./run.sh VeriFLA verifla_properties_counters.txt
1
```

Fig. 4. Running openVeriFLA on the host computer

These 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.

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 figures below.

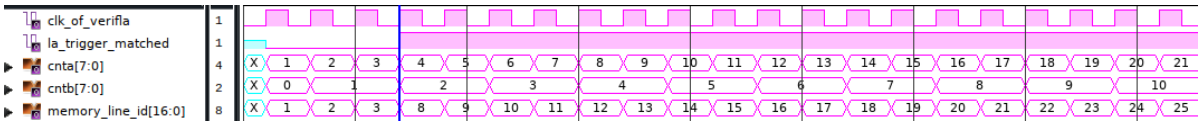


Fig. 5a. Simulating a capture.v file

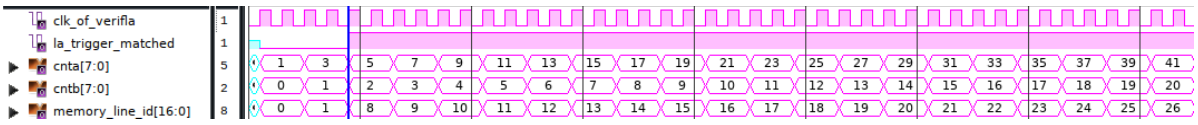


Fig. 5b. Simulating a capture.v file

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).

In the first figure, the monitor was configured to capture cntb and cnta whenever a bit of these signal changes. The trigger moment was set such as cntb=2 and cnta=4 (LA_TRIGGER_VALUE=16'h0204, LA_TRIGGER_MASK=16'hffff, LA_TRACE_MASK=16'hffff).

In the second figure, the monitor was configured (LA_TRIGGER_VALUE=16'h0200, LA_TRIGGER_MASK=16'hff00, LA_TRACE_MASK=16'hff00) to capture cntb and cnta only when cntb changes. So we can store in the same memory higher values of cntb and cnta (for example when memory_line_id is 20, in the first figure cntb is 8 and cnta is 16 and in the second figure cntb is 14 and cnta is 29). It must be mentioned the fact that in the second figure the trigger event appears when cntb=2 and, for this value, it corresponds two values of cnta (4 and 5) - the last value being kept by openverifla.

In the capture.v simulation, run command was necessary, to reach the $stop instruction of the capture.v.

3 Configuration parameters

3.1 Host computer 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_counters.txt file which is tuned for the counters 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 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.

3.2 The FPGA parameters file

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;
- 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.
- in LA_TRACE_MASK, the signals that are with 0 will be traced only when one or more signals that are with 1 change;

- 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).

4 VHDL openVeriFLA

The Verilog sources were translated line by line in VHDL. Every .v file is .vhd in the VHDL sources. Everything specified in this manual for Verilog is valid in the VHDL implementation.