FPGA Based Logic Signal Analyzer

Project team:

Abdallah Hussein
Ahmed El-Gayar
Aya Attia
Hasnaa Ahmed

Table of Contents

Table of Figures

Table of Tables

Abstract

In this Report FPGA based Logic signal analyzer designed in VHDL and implemented using ISE Design Suite 14.7, Xilinx Platform Studio and Xilinx SDK. The design for Spartan FPGA, Spartan 6 kit is used in this project.

Introduction

Logic analyzers are used for testing complex digital or logic circuits. It enable tracing of logic signals in a way that the operation of several lines in a digital circuit can be monitored. The first logic analyzers were developed to debug and undertake fault finding on microprocessor based systems. Logic analyzer have many characteristics that separate it from multichannel oscilloscope and other test instruments it provides a time display of logic states ,multiple channels it can monitor a large number of digital lines up to 200+ channels ,displays logic states and doesn’t display analogue information . The difference between logic analyzer and oscilloscope that logic analyzer used to verify and debug operation of digital designs using a logic states and timings while oscilloscope is used for measuring analogue waveforms: amplitude, phase values, edge measurements as rise times,etc.

Design

VHDL is used to model Logic analyzer design it has 4 inputs clock, reset, Data [7:0], UART_RX and one output UART_TX as it shown in figure one.

Block diagram

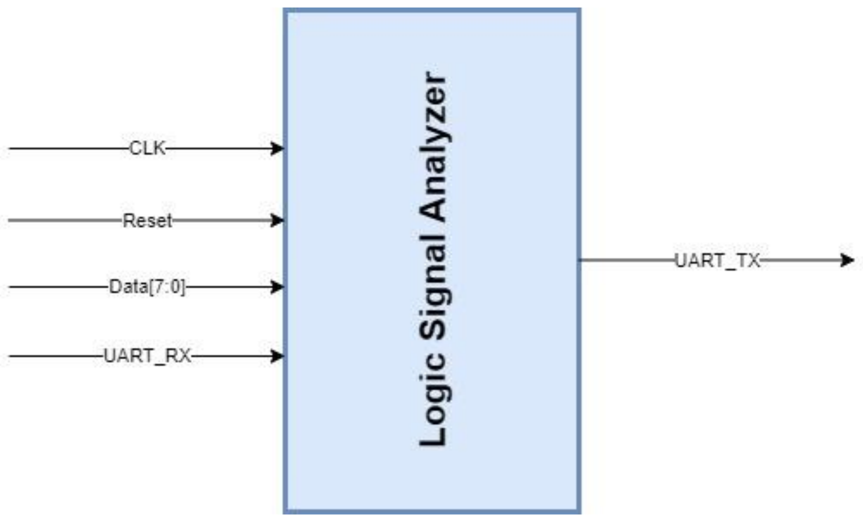
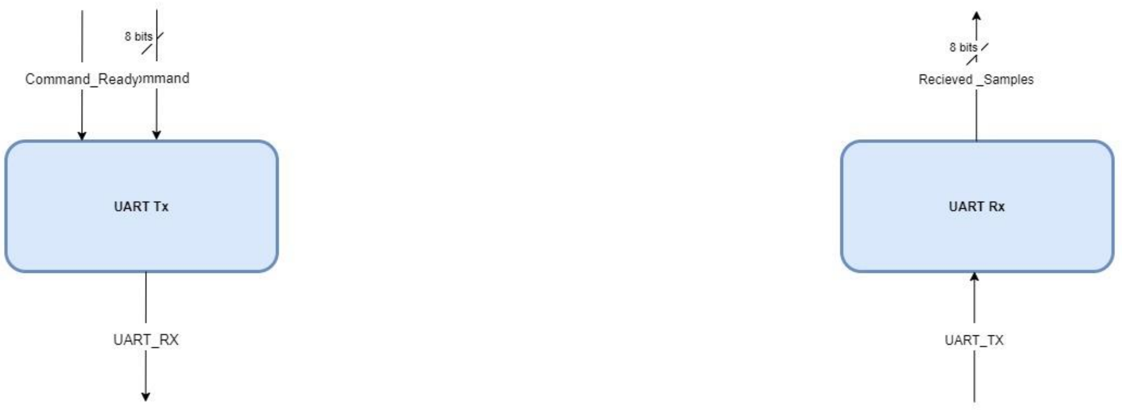


Figure 1 Logic Signal Analyzer signals



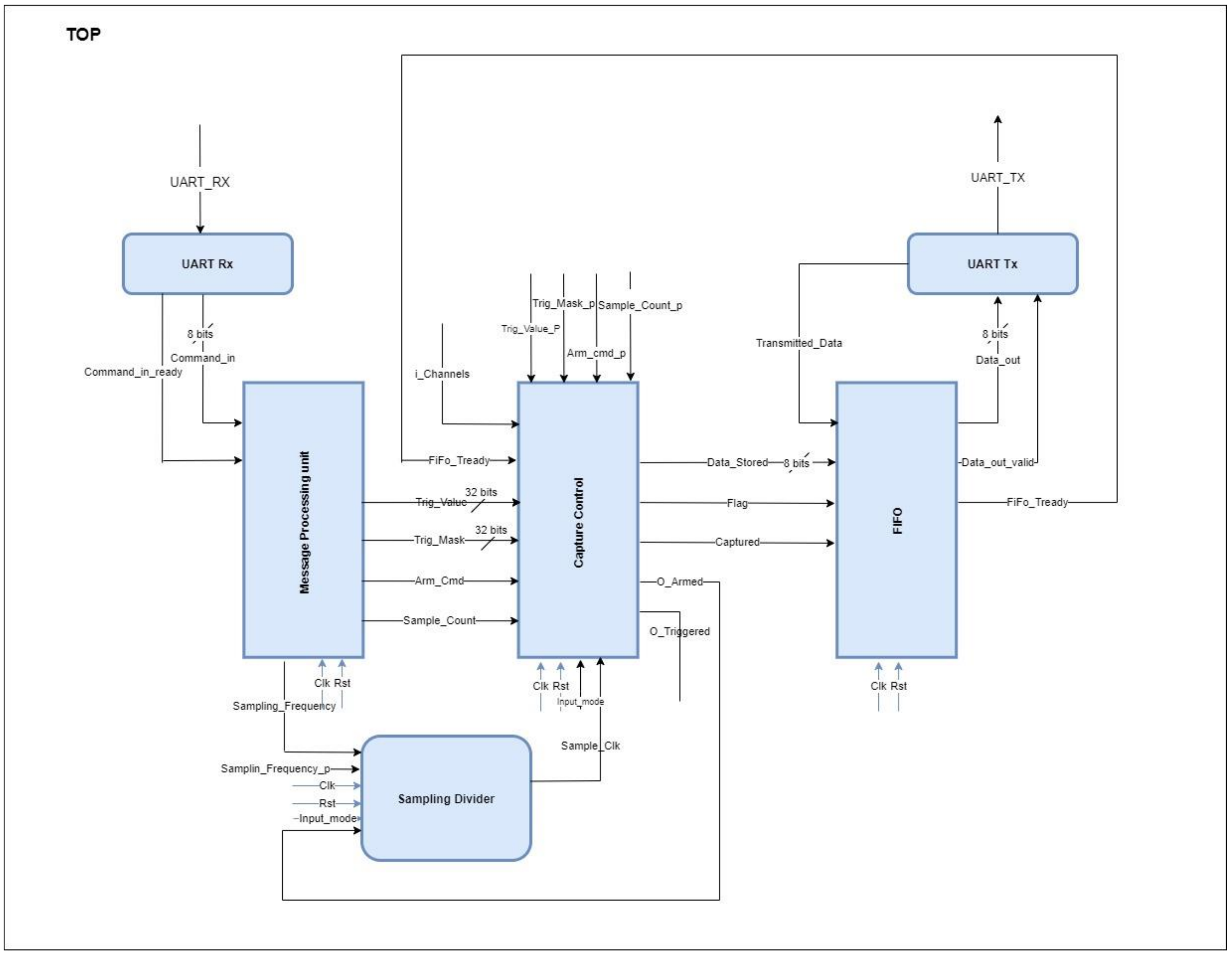


Figure 2 LSA Block diagram

Figure 2 shows LSA block diagram in details, it consists of message processing unit, capture control unit, Sample divider FIFO and UART. In the system there is a short command like ARM command and a long command like Trig mask,Trig value, Sample count ,Sample divider each command have opcode so message processing unit do some processing on the input (command in) to choose which type the system work on then the capture control took the data and apply one of the command chosen from message processing according to the opcode also the capture control unit took a signal from the sample divider to control the rate of samples in captured data . In this design there is two option for the capture control to take the commands either from micro-blaze processor or from the message processing unit so mux is added to choose between those signals the after the data have been captured it is stored in FIFO , after this we can read from the FIFO and transmit it to the UART.

Inputs and Outputs

This table show all inputs and outputs of each block and their description .

Block	Signal Type	Signal Name	Signal Description
Message Processing	Input	Clk	Global clock
		Rst	Synchronous reset
		Data_in (7 downto 0)	The command bits received from the UART
		Data_in_Ready	when Flag=0, the Message processing unit is not able to receive another Data_in. When Flag=1, the Message processing unit is able to receive another Data_in.
	Output	Trig_Val	A reference data to check if it matches the i_channels. If they match, trigger=1.
		Trig_Mask	Define which trigger values must match.
		Arm_Cmd	First stage triggering.
		Sample_Count (31 downto 0)	Number of samples to be captured.
		Sample_Divider (23 downto 0)	The frequency at which the data will be sampled
Capture Control	Input	Sample_clk	New clock for the sampling

			frequency
		Rst	Synchronous reset
		Trig_Val (31 downto 0)	A reference data to check if it matches the i_channels. If they match, trigger=1.
		Trig_Mask (31 downto 0)	Define which trigger values must match.
		Arm_Cmd	First stage triggering.
		Sample_Count	Number of samples to be captured.
		i_channels (7 downto 0)	The data to be captured by the capturing control unit
		Fifo_Tready	A flag to ensure that the fifo is ready to store new data. The capture control unit will not be able to capture new data unless the fifo is ready to store that captured data.
	Output	Data_Stored	The data to be stored in the fifo
		Flag	It is 1 whenever I capture a new data telling the fifo that data is ready to be stored.
		Captured	To indicate that the last capture has been finished for the fifo to start sending data to UART.
		O_Armed	To indicate that the capture control unit is working.
		O_Triggered	To indicate that the capture control unit is working.
Fifo	Input	Clk	Global clock
		Reset	Synchronous reset
		Data_Stored	The data to be stored in the fifo.
		Flag	It is 1 whenever I capture a new data telling the fifo that data is ready to be stored.
		Captured	To indicate that the last capture has been finished for the fifo to start sending data to UART.
		Transmitted_Data	A flag from UART transmitter ensuring that it is ready to send other data
	Output	Data_out	Data to be sent through UART to GUI.

		valid	A flag from fifo telling the UART transmitter that there is new data to be transmitted
		Fifo_tready	A flag to ensure that the fifo is ready to store new data. The capture control unit will not be able to capture new data unless the fifo is ready to store that captured data.
Sample Rate Control	Input	Clk	Global clock
		Rst	Synchronous reset
		Sample_Divider	The frequency at which the data will be sampled
	Output	Sample_clk	New clock for the sampling frequency

Table 1 Inputs /Outputs description

Message Processing

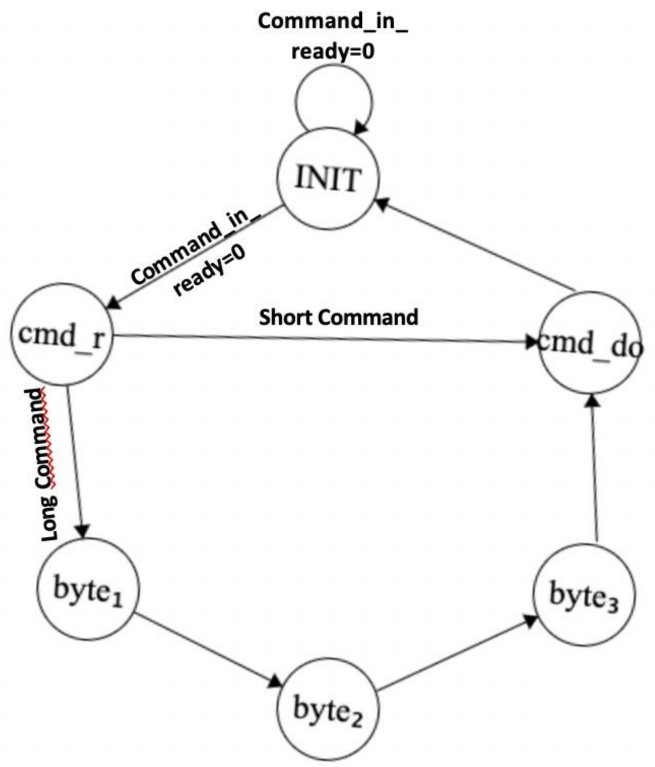
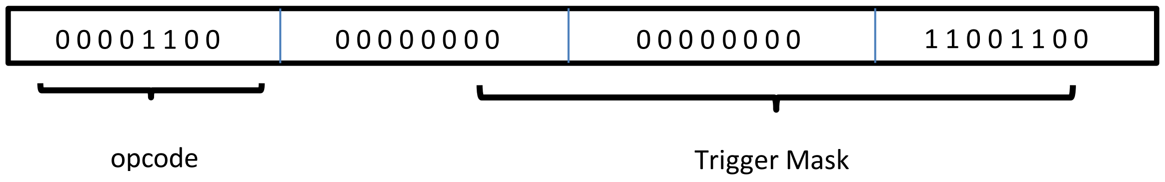


Figure 3 Message Processing FSM

The message processing unit is the block used to process the commands received from the UART. There are different commands sent to the capture control unit as well as the sample divider unit. Those commands might be sent as a short command like the arm command which is responsible for running the capture control unit or a long command such as trigger mask, trigger value, sample count and sample divider. Short commands are exactly one bit long while long ones are four bytes long in which the first byte represents the opcode of the command while the other three are the data related to the command. Each command input/output signal is descripted in detail in table 1.



As shown in fig(3), this block was implemented using a finite state machine of six states. The first one is the initial state. In this state, the message processing unit waits for a complete command from the UART. Once this command is received, it goes to the second state which is the read_CMD state. Read_CMD is used to detect the type of the command sent, whether it is a long command, short command, or neither. Dependent on the analysis done in this state, the machine might go to capture the three bytes if it was a long command or goes directly to the CMD_do state which is the final state. In this one, the output value is assigned to the output of the message processing unit. Finally, the machine goes back to the initial state waiting for a new command from the UART.

Capture Unit

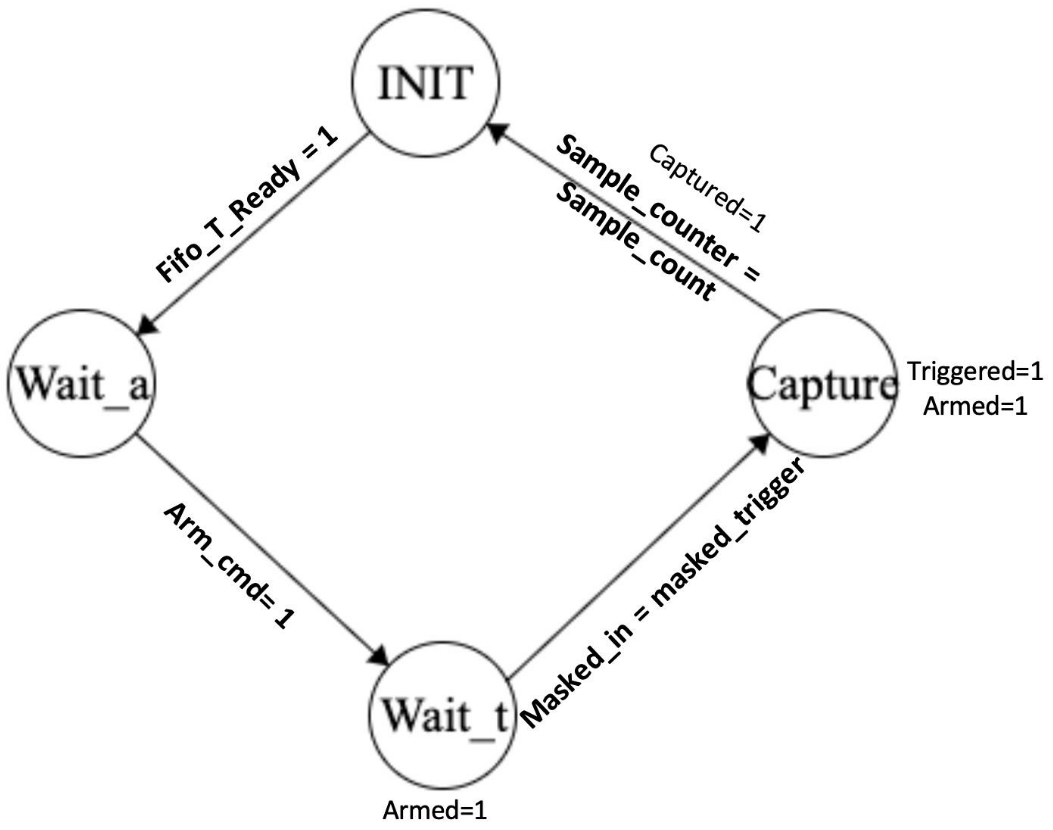


Figure 4 Capture Unit FSM

This entity is the primary capture controller of the logic analyzer. It is responsible for matching the trigger mask to the sampled data and passing the required amount of data out. As illustrated in fig(4), this block is implemented using a finite state machine of four state. When the system is reset, the control capture unit will stay at INIT state waiting for the fifo to be ready for data storing. Once ready, the capture unit will go to the next state which is wait for arm. Once the arm command is sent, it will read the trigger mask, trigger value, and sample count and waits for the data to be triggered in the wait for trigger state. Whenever the data is triggered, the capture unit will move to the capture state in which it will start capturing data. It will keep capturing the data samples until reaching the number of samples requested by the message processing unit. At that moment, the capture control unit is done capturing the data and will move back to the INIT state to wait for another cycle.

Sample divider

This block is used to receive the sampling frequency from the message processing unit and divides the clock to control the rate at which the capture control unit captures the data.

FIFO

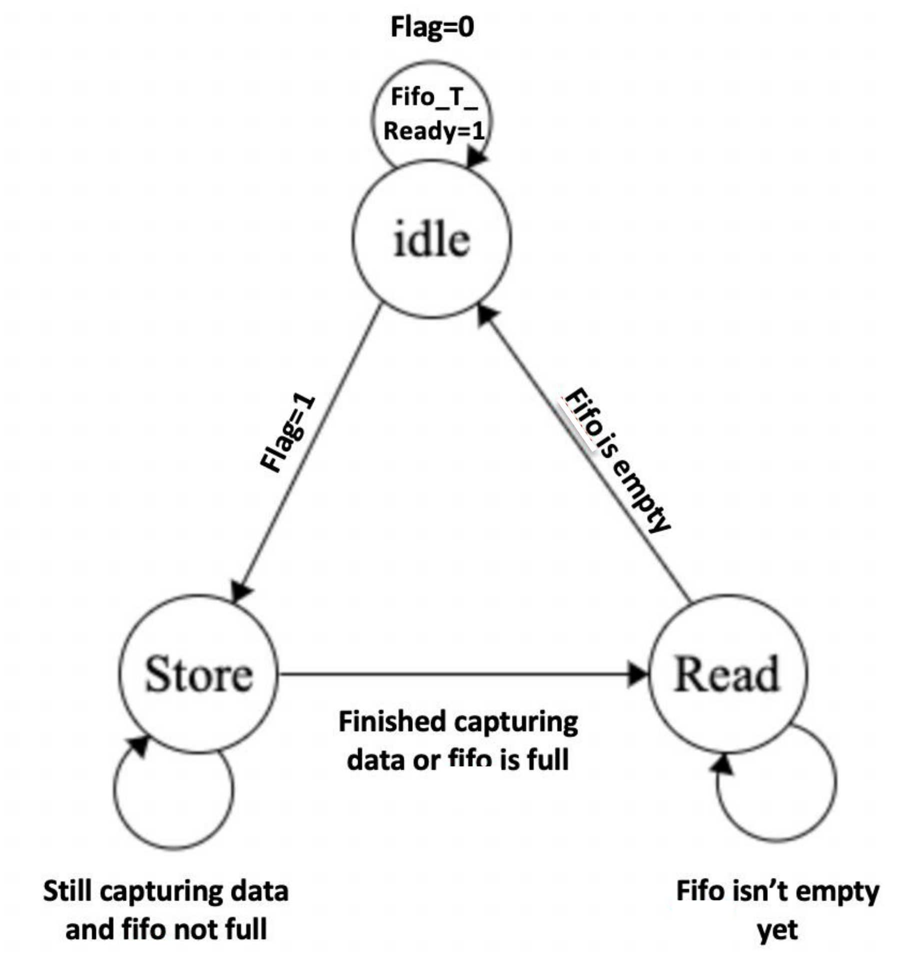

Flag=0
Fifo_T_ Ready=1
idle
Flag=1
Fifo is empty
Store
Read
Finished capturing data or fifo is full
Still capturing data and fifo not full
Fifo isn't empty yet

Figure 5 FIFO FSM

The fifo is the storage element used in this system to store the captured data until all samples are captured. This block is implemented using the finite state machine illustrated in fig(5). The machine started at the idle state in which the fifo is empty and waiting for data to be stored in it. As soon as there is no data ready from the capture control unit, the machine will stay at this state. At the moment when there is a data ready, the fifo starts the storing phase in which it stores data until it is full or until there is no more data ready to be stored. When at least one of those two conditions is violated, the fifo goes to the read state in which it starts to send data to be transmitted through the UART. Whenever the fifo is empty, it goes from the read state back to the idle state awaiting for data. Fig(6) illustrates how fifo stores and empties data.

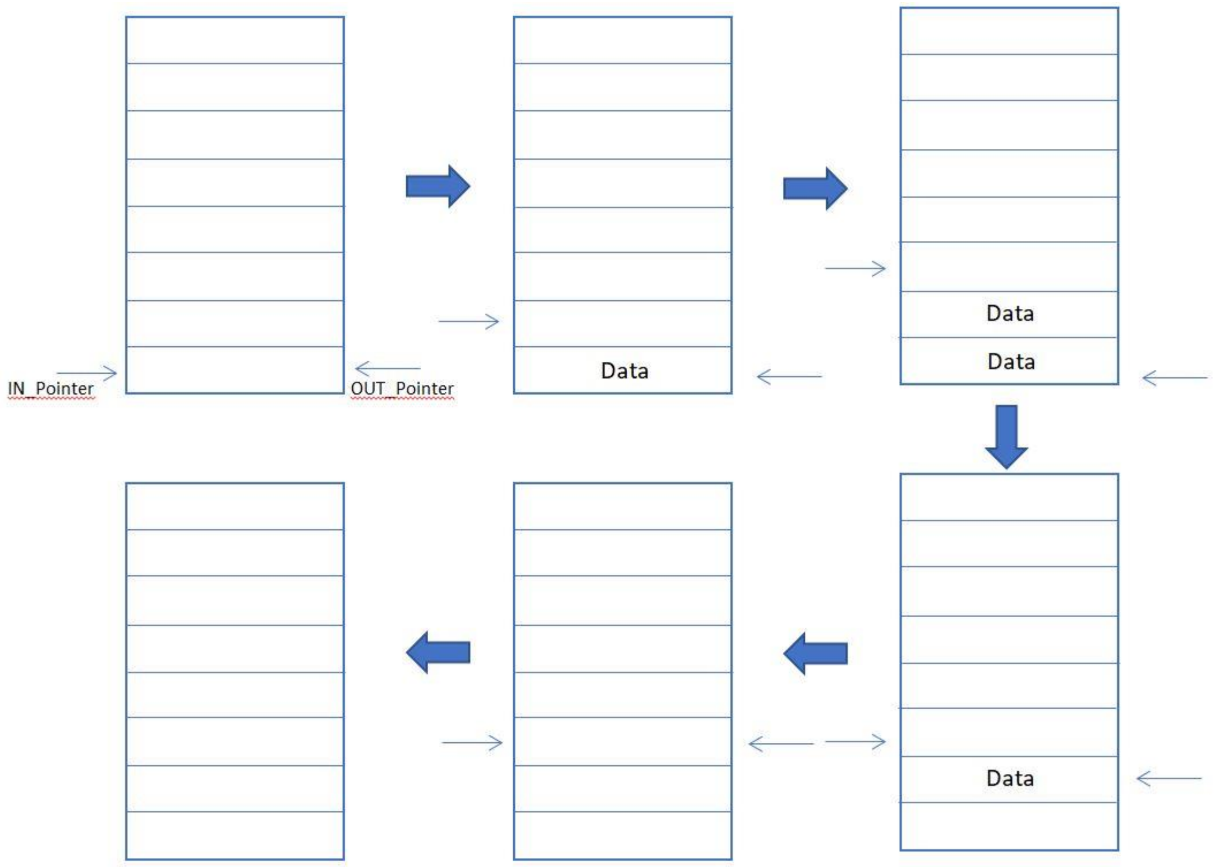


Figure 6 FIFO Demonstration

LSA interface with FPGA

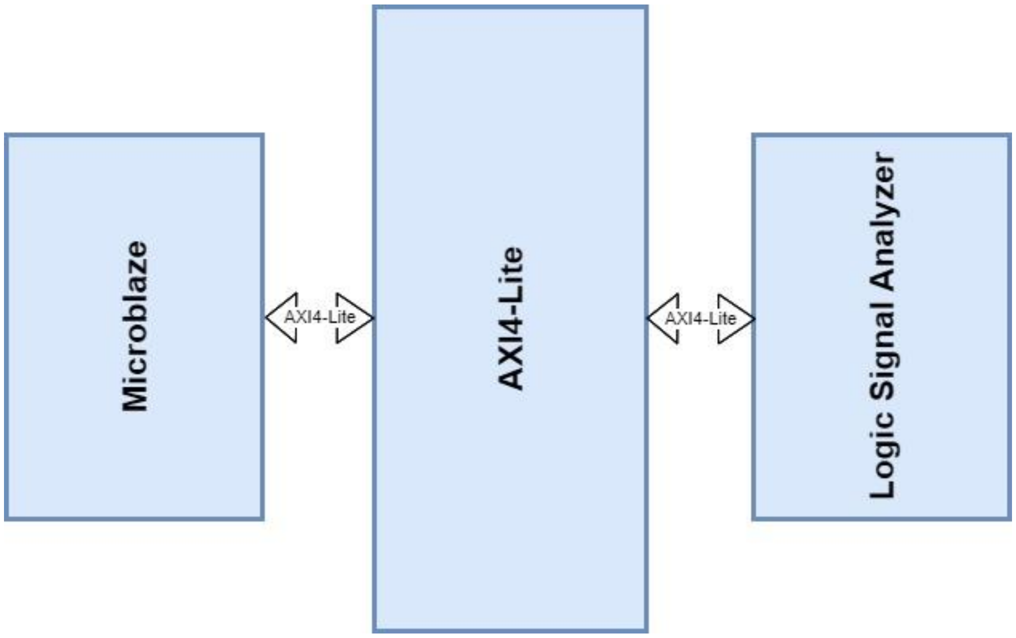


Figure 7 LSA with FPGA

Register Map

1-Input Register

Address x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								Input data							
								rw	rw	rw	rw	rw	rw	rw	rw

Table 2 Input Register register map

2-Trigger Mask Register

Address x0001

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								Trigger Mask Register							
								rw	rw	rw	rw	rw	rw	rw	Rw

Table 3 Trigger Mask Register map

3- Trigger Value Register

Address x0002

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								Trigger Value Register							
								rw	rw	rw	rw	rw	rw	rw	Rw

Table 4 Trigger Value Register map

4- Sample Count Register

Address x0003

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								Sample count							
								rw	rw	rw	rw	rw	rw	rw	rw

Table 5 Sample Count Register map

5- Control Register

Address x0004

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								UART-RX						Input mode	ARM CMD
								rw						rw	rw

Table 6 Control Register map

6- Sample Divider Register

Address x0005

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								Sample divider							
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sample divider															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Table 7 Sample Divider Register map

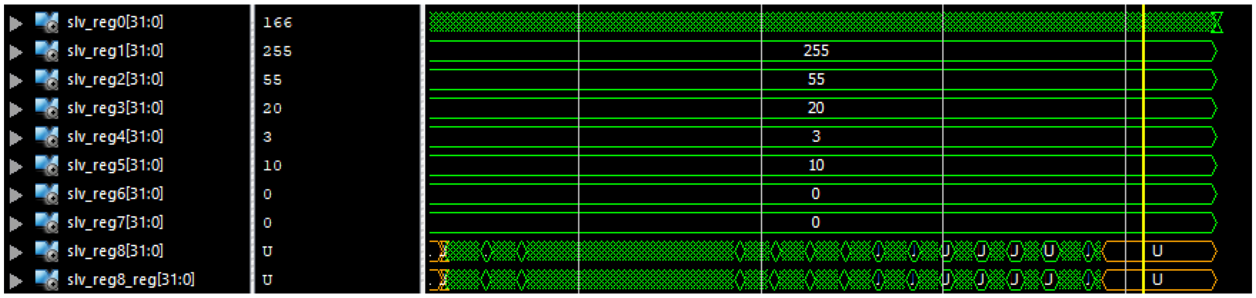
7- Output Register

Address x0008

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
							UART-TX	Output Register							
							rw	rw	rw	rw	rw	rw	rw	rw	rw

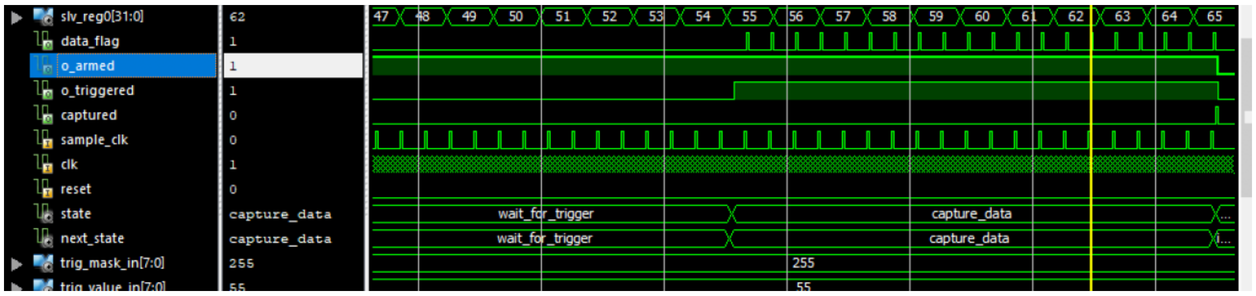
Table 8 Output Register map

Simulation Results



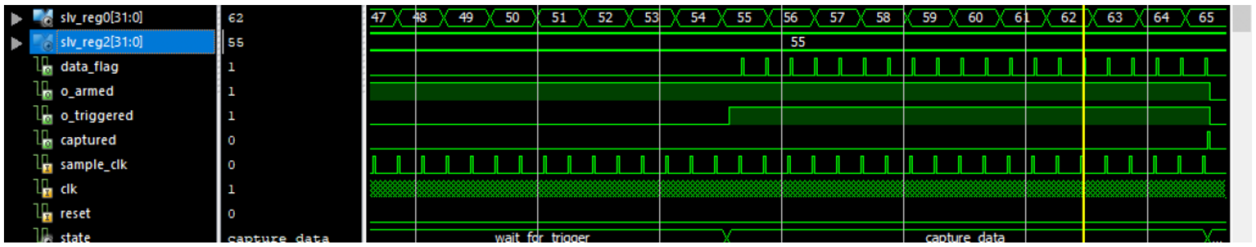
slv_reg0[31:0]	166
slv_reg1[31:0]	255
slv_reg2[31:0]	55
slv_reg3[31:0]	20
slv_reg4[31:0]	3
slv_reg5[31:0]	10
slv_reg6[31:0]	0
slv_reg7[31:0]	0
slv_reg8[31:0]	U
slv_reg8_reg[31:0]	U

Figure 8 9 Registers



slv_reg0[31:0]	62
data_flag	1
o_armed	1
o_triggered	1
captured	0
sample_clk	0
clk	1
reset	0
state	capture_data
next_state	capture_data
trig_mask_in[7:0]	255
trig_value_in[7:0]	55

Figure 9 States



slv_reg0[31:0]	62
slv_reg2[31:0]	55
data_flag	1
o_armed	1
o_triggered	1
captured	0
sample_clk	0
clk	1
reset	0
state	capture_data

Conclusion

In this Report FPGA based Logic signal analyzer designed in VHDL and implemented using ISE Design Suite 14.7, Xilinx Platform Studio and Xilinx SDK. The design for Spartan FPGA, Spartan 6 kit is used in this project

References

1- Implementation of FPGA based low cost logic signal analyzer

2- https://www.electronics-notes.com/articles/test-methods/logic-analyzer/basics-tutorial.php

3- https://github.com/ashtonchase/logic_analyzer