Application Note PG001:

Using 36-Channel Logic Analyzer and 36-Channel Digital Pattern Generator for testing a 32-Bit ALU

Version: 1.0
Date: December 14, 2004

Designed and Developed By:

System Level Solutions, Inc. (USA)

14702 White Cloud Ct.
Morgan Hill, CA 95037

System Level Solutions (India) Pvt. Ltd

9/B, Radhakrishna Colony
MangalPura Road, Anand-380001

About this Manual

This application note provides the concepts for using 36-Channel Logic Analyzer and 36-Channel Digital Pattern Generator for testing systems with more than 36 Inputs/Outputs. It gives concepts for testing higher end systems and lower level debugging.

The following table shows Revision History of the Application Note PG001: Using 36-Channel Logic Analyzer and 36-Channel Pattern Generator for testing a 32-Bit ALU

Date	Description
December 2004	First publication of the Application Note PG001: Using 36-Channel Logic Analyzer and 36-Channel Pattern Generator for testing a 32-Bit ALU

How to find information

- The Adobe Acrobat Find feature allows you to search the contents of a PDF file. Click the binoculars toolbar icon to open the Find dialog box
- Bookmarks serve as an additional table of contents.
- Thumbnail icons, which provide miniature preview of each page, provide a link to the pages.
- Numerous links, shown in blue text, allow you to jump to related information.

Contact Information

For the most up-to-date information about SLS products, go to the SLS worldwide web site at http://www.slscorp.com.

Information type	**Contact**
Product literature services	http://www.slscorp.com
SLS literature services	http://www.slscorp.com
Non-Technical customer services	+91-02692-264661
Technical support	+91-02692-264661
	http://www.slscorp.com
FTP site	ftp.slscorp.com

CONTENTS

ABOUT THIS MANUAL I

HOW TO FIND INFORMATION I

CONTACT INFORMATION II

LIST OF FIGURES IV

1 INTRODUCTION 1

1.1 DIGITAL SYSTEM DEBUGGING 1
1.2 REQUIREMENT OF SUPPORTING SYSTEM IN DEBUGGING 2

2 SINGLE SYSTEM APPROACH 5

2.1 THE CONCEPT 5
2.2 CASE OF 32-BIT ALU TESTING 5
2.3 DESIGN OF THE DIGITAL TEST PATTERN 7
2.4 DESIGN OF THE TEST SUITE FOR DEBUGGING 10
2.5 PROS AND CONS OF THE SINGLE SYSTEM APPROACH 15

3 TWO-SYSTEM APPROACH 17

3.1 THE CONCEPT 17
3.2 CASE OF 32-BIT ALU TESTING 18
3.3 DESIGN OF THE DIGITAL TEST PATTERN 19
3.4 DESIGN OF THE TEST SUITE FOR DEBUGGING 21
3.5 PROS AND CONS OF THE TWO-SYSTEM APPROACH 26

4 CONCLUSION 27

5 REFERENCES 29

LIST OF FIGURES

FIGURE 1-1 SIMPLE 32-BIT ALU 1
FIGURE 1-2 USING SUPPORTING SYSTEMS 3
FIGURE 2-1 SINGLE SYSTEM APPROACH 6
FIGURE 2-2 SINGLE SYSTEM APPROACH: DIGITAL TEST PATTERN DESIGN WITH LSB OUTPUTS SELECTED 7
FIGURE 2-3 SINGLE SYSTEM APPROACH: DIGITAL TEST PATTERN DESIGN WITH MSB OUTPUTS SELECTED 8
FIGURE 2-4 SINGLE SYSTEM APPROACH: DIGITAL TEST PATTERN DESIGN WITH ALL OUTPUTS SELECTED 9
FIGURE 2-5 SINGLE SYSTEM APPROACH: SYSTEM DESIGNED IN QUARTUS II 11
FIGURE 2-6 SINGLE SYSTEM APPROACH: GENERATING TEST PATTERN USING SLS DIGIPAT AND CAPTURING THE TEST RESULTS USING SLS CDLOGIC FOR LSB OUTPUTS 12
FIGURE 2-7 SINGLE SYSTEM APPROACH: GENERATING TEST PATTERN USING SLS DIGIPAT AND CAPTURING THE TEST RESULTS USING SLS CDLOGIC FOR MSB OUTPUTS 13
FIGURE 2-8 SINGLE SYSTEM APPROACH: GENERATING TEST PATTERN USING SLS DIGIPAT AND CAPTURING THE TEST RESULTS USING SLS CDLOGIC FOR ALL OUTPUTS (LSB & MSB) 14
FIGURE 3-1 TWO-SYSTEM APPROACH 18
FIGURE 3-2 TWO-SYSTEM APPROACH: DIGITAL TEST PATTERN DESIGN 19
FIGURE 3-3 TWO-SYSTEM APPROACH: SYSTEM DESIGNED IN QUARTUS II FOR LSB OUTPUTS 22
FIGURE 3-4 TWO-SYSTEM APPROACH: SYSTEM DESIGNED IN QUARTUS II FOR MSB OUTPUTS 23
FIGURE 3-5 TWO-SYSTEM APPROACH: GENERATING TEST PATTERN USING SLS DIGIPAT AND CAPTURING THE TEST RESULTS USING SLS CDLOGIC FOR LSB OUTPUTS 24
FIGURE 3-6 TWO-SYSTEM APPROACH: GENERATING TEST PATTERN USING SLS DIGIPAT AND CAPTURING THE TEST RESULTS USING SLS CDLOGIC FOR MSB OUTPUTS 25

1 INTRODUCTION

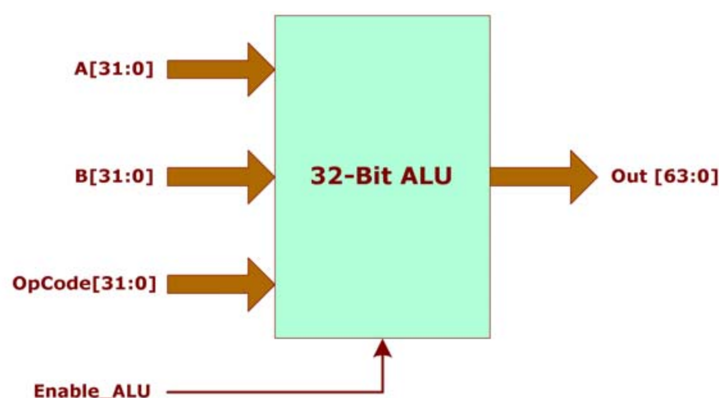
This section gives the general system test scenario and requirement of developing intermediate supporting systems for using 36-Channel Logic Analyzers and 36-Channel Digital Pattern Generators for testing systems with more than 36 Inputs/Outputs.

1.1 Digital System Debugging

When it comes to debugging a system (be it Analog or Digital), it's a headache to the design engineer. In old days, simple oscilloscope (with single channel input) was good enough to test various test points on the system and diagnose the system. In today's fast developing era, huge systems, though tiny in size but complex in nature, are designed. When it comes to test and debug such systems, even the modern tools like 36-channel Logic Analyzers and 36-Channel Pattern Generator don't cope with the test setup requirement. The usual limitation put on such test instruments is mainly the signal data width (36-Bit/Channel) and the maximum sampling frequency supported by the test instrument.

Consider a simple case of designing a simple 32-Bit ALU, which performs simple arithmetic operations like addition, subtraction, multiplication, etc. simple logical operations like AND, OR, NAND, NOR, XOR, XNOR, Compliment, etc. as shown in the Figure 1-1 below:

Figure 1-1 Simple 32-Bit ALU



This 32-Bit ALU will require 64-Bit output to support the multiplication operation. Also assume that this ALU has an additional input called Enable_ALU. When this signal is low, the output bus (Out) is tri-stated else it gives the normal ALU Output. The ALU output, Out, is purely combinatorial in nature.

Now assume that the designer wants to test the performance of this ALU using Logic Analyzer and Digital Pattern Generator. To test this ALU, the designer must have 32+32+32+1 = 97-Channel (Minimum) Digital Pattern Generator and 64-Channel (Minimum) Logic Analyzer. If the designer goes with this solution then it will increase the expenditure required to buy these test equipments, if they are available. Now imagine what will happen, when the designer wants to upgrade his design from 32-Bit ALU to 64-Bit ALU. He'll need to buy new test equipments with the specifications that match his design requirement. Obviously this is not a sensible solution.

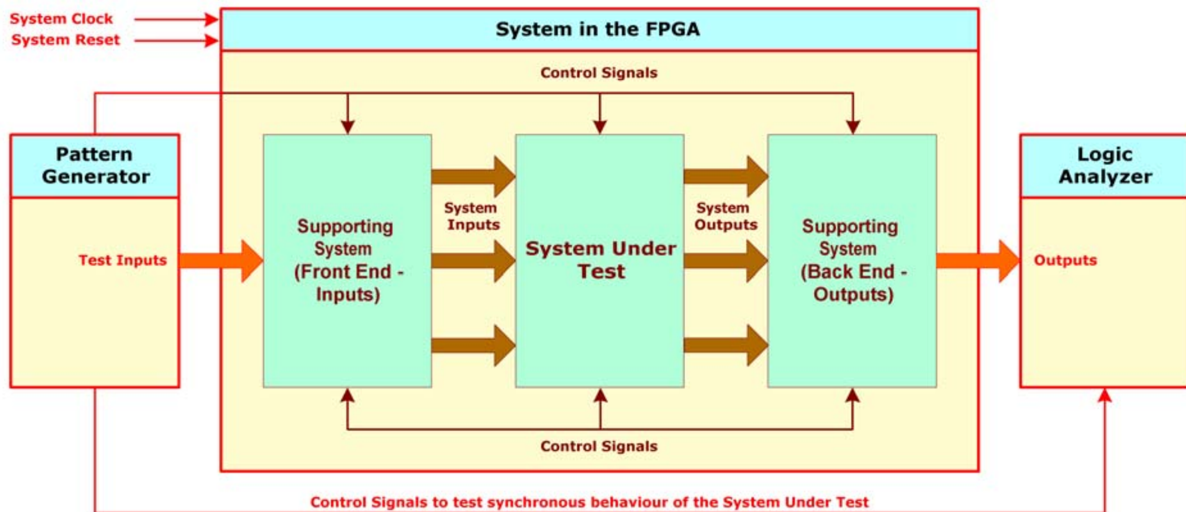
This Application Note addresses this issue and gives concept how the 36-Channel Logic Analyzer and 36-Channel Digital Pattern Generator can be used to test a 32-Bit ALU as a case study example.

1.2 Requirement of Supporting System in Debugging

The main issue addressed over here is how to handle large number of Inputs/Outputs on the system under test with small number of Inputs/Outputs available on the test instrument (Logic Analyzers and Digital Pattern Generator in this case).

To solve this issue, a Supporting System can be used as explained below. A Supporting System is nothing but a wrapping module or an auxiliary system that bridges the System under test with the test instrument.

Figure 1-2 Using Supporting Systems



Looking at the problem statement, we can find that the easiest way to do this is by serializing the inputs and outputs as shown in the Figure 1-2 above. That is sending all the data serially to the system (from Digital Pattern Generator) and then converting the serial data stream into the parallel data using a supporting system (Front End Supporting System). This data, then, can be input to the system under test. The same scheme can be applied to the outputs also. Use a supporting system (Back End Supporting System) that serializes the parallel data output of the system under test. This serialized data is then fed to the Logic Analyzer for debugging. Using this method the designer can utilize whatever channels (Data width) are available on the test instruments. This is the most generic approach to solve this issue. Two schemes are discussed in the subsequent sections of this Application Note for solving this issue.

Note:

This Application Note assumes that the designer is familiar with the synthesis tools like Quartus II and the debugging tools like Logic Analyzers and Digital Pattern Generators. No specific information regarding these tools is explained in this Application Note. It is solely assumed that the designer is familiar with the tools used in Digital Design and Debugging when using this Application Note.

2 Single System Approach

In this section, the first scheme of the data serialization is discussed. This method uses single system approach, i.e., a single system is required to debug the system under test using test instruments. This section explains the scheme considering the case of testing 32-Bit ALU using 36-Channel Logic Analyzer and 36-Channel Digital Pattern Generator.

2.1 The Concept

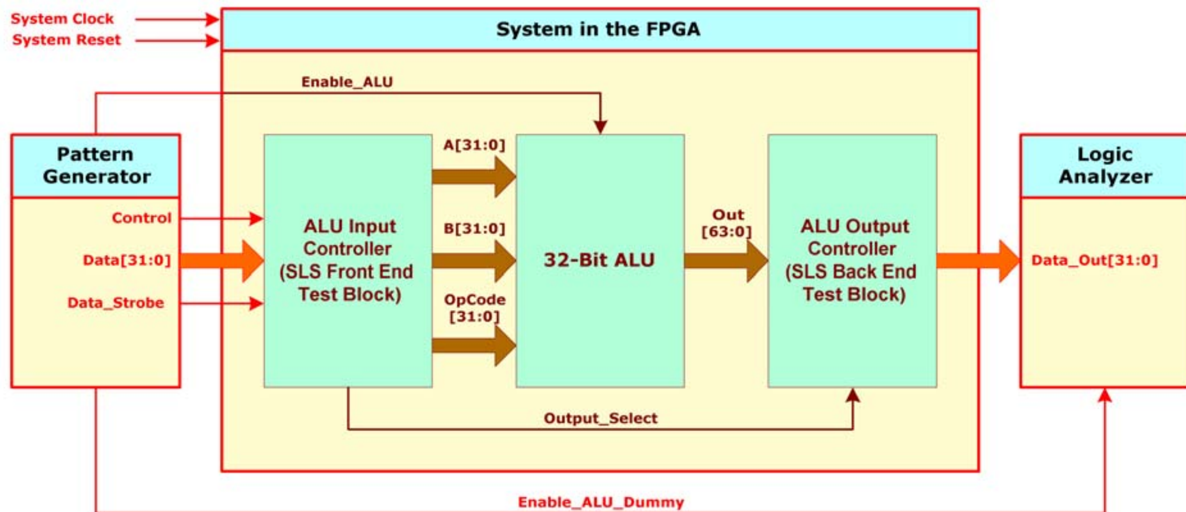
This is the very fundamental way to serialize the input data stream. In this method all the inputs are divided into a number of bundles of signals. Each bundle contains a collection of the signals. Now these bundles are sent in a fixed predefined manner from the Digital Pattern Generator, which are decoded by the Front End Supporting System, which manages the inputs. The Front End Supporting System de-serializes the input bundles from the Digital Pattern Generator in the exact predefined manner giving the exact expected parallel inputs to the system under test. The same scheme can be applied to the outputs also. A Back End Supporting System is used to serialize the parallel outputs of the system under test. The output of the Back End Supporting System is again the bundles of signal streams. This output can be decoded and utilized for debugging the system under test using Logic Analyzer.

To make this concept clearer, consider the case of testing a 32-Bit ALU using a 36-Channel Logic Analyzer and a 36-channel Digital Pattern Generator.

2.2 Case of 32-Bit ALU Testing

Assume that the designer wants to test a 32-Bit ALU using a 36-Channel Logic Analyzer and a 36-Channel Digital Pattern Generator. To assist this testing the following test set up can be done as shown in the Figure 2-1.

Figure 2-1 Single System Approach



Assume that the ALU has three 32-Bit inputs A, B and OpCode, one 1-Bit Enable input and 64-Bit output called Out to support multiplication operation. To support the testing of this ALU two supporting systems can be used as shown in the Figure 2-1 above. Thus the main system residing in the FPGA will contain the 32-Bit ALU, the ALU Input Controller (Front End Supporting System) and the ALU Output Controller (Back End Supporting System).

The Front End Supporting System, called ALU Input Controller, takes its inputs from the Digital Pattern Generator, de-serializes the input data stream and gives the parallel output to the 32-Bit ALU. The inputs to the ALU Input Controller system are 32-bit Data, 1-Bit Control and 1-Bit Data Strobe. The outputs of the ALU Input Controller system are the three 32-Bit inputs to the ALU: A, B and OpCode. This system also has a 1-Bit output, called Output Select, which controls the Back End Supporting System.

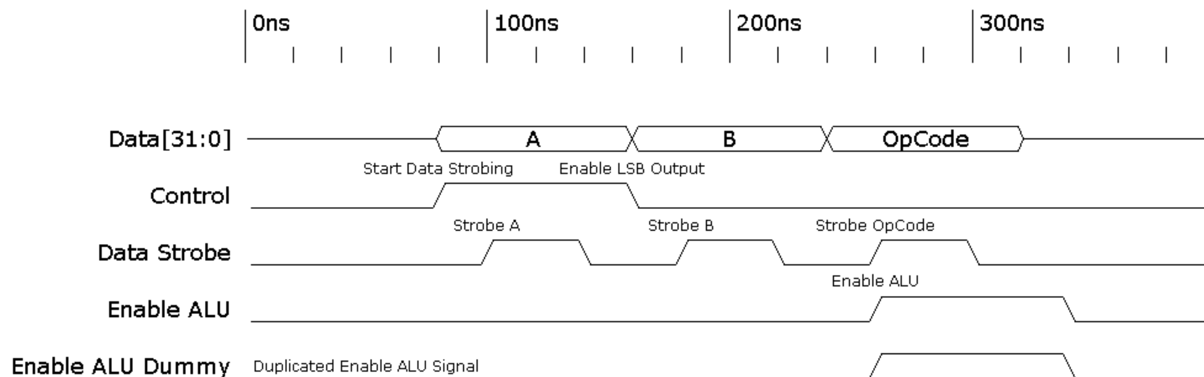
The Back End Supporting System, called ALU Output Controller, takes its inputs from the ALU, serializes the parallel inputs and gives output to the Logic Analyzer. The input to the ALU Output Controller system is the 64-Bit output of the ALU. The output of the ALU Output Controller system is a 32-Bit output part selected from the 64-Bit ALU Output as per the Output Select signal.

The functioning of the supporting systems is explained in the subsequent section.

2.3 Design of the Digital Test Pattern

This section mainly describes the functioning of the supporting systems. Consider the design of the Digital Test Pattern shown in the Figure 2-2 below:

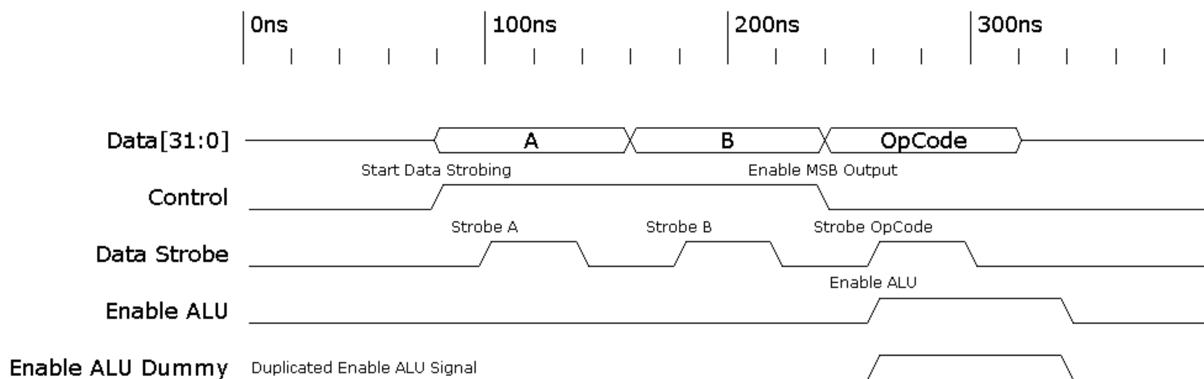
Figure 2-2 Single System Approach: Digital Test Pattern Design with LSB Outputs selected



Here Data[31:0], Control and Data Strobe are the inputs to the ALU Input Controller, Enable ALU is the input to the ALU and Enable ALU Dummy is the input to the Logic Analyzer to check the synchronous behavior of the 32-Bit ALU (i.e., to see what timing delay exists for the ALU outputs to change with the change in the ALU inputs). The above Pattern is designed for the SLS DigiPat – Digital Pattern Generator. The Digital Pattern Generator generates this pattern, which is decoded by the ALU Input Controller to generate the parallel inputs for the ALU.

The test pattern is designed as follows: When a new set of instruction starts, Control signal is asserted HIGH. On the Positive Edge of the Control Signal, the ALU Input Controller gets ready to accept new data stream in the following order: A, B and OpCode. Then as soon as a (first) pulse arrives on the Data Strobe Line, whatever data is on the Data bus, is strobed into the A register of the ALU Input Controller that feeds the A input of the ALU. On the next (second) pulse on the Data Strobe Line, whatever data is on the data bus is strobed into the B register of the ALU Input Controller that feeds the B input of the ALU. At this point if the Control Line is HIGH, then the MSB output of the ALU is enabled through the ALU Output Controller (as shown in the Figure 2-3 below) else if the Control Line is LOW, then the LSB output of the ALU is enabled through the ALU Output Controller (as shown in the Figure 2-2 above).

Figure 2-3 Single System Approach: Digital Test Pattern Design with MSB Outputs selected



On the next (third) pulse on the Data Strobe Line, whatever data is on the data bus is strobed into the OpCode register of the ALU Input Controller that feeds the OpCode input of the ALU. At this point the job of the ALU Input Controller is done. The ALU Input Controller now returns to its IDLE state and is triggered only when a positive edge is detected on the Control Line.

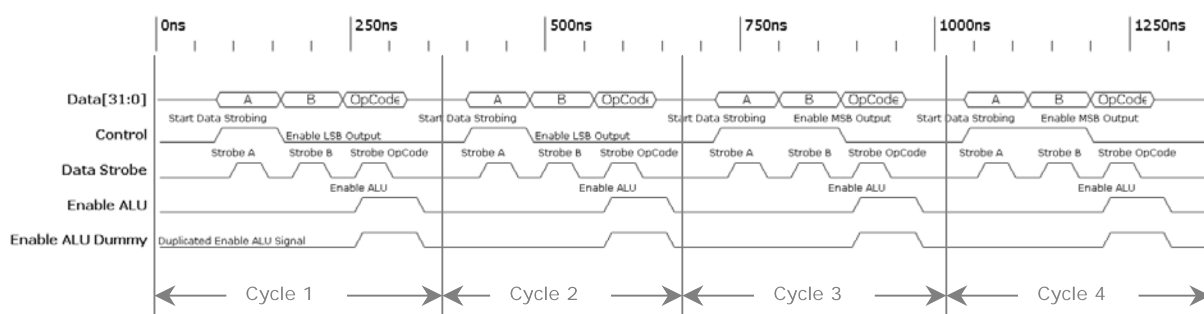
The ALU is assumed to be a purely combinatorial module. It operates as per the data on the A, B and OpCode lines. The Enable ALU Line merely controls the output of the ALU. If the Enable ALU line is HIGH, then the output of the ALU is routed to the output bus. If the Enable ALU line is LOW, then the output bus is tri-stated. (This is merely the design considered for this example. The designer may implement a different architecture and a different test suite for the same to suite a particular requirement.) The Enable ALU line can be asserted HIGH any time just to see the change in the output of the 32-Bit ALU. Similarly it can be asserted LOW anytime to disable the ALU outputs. By asserting Enable ALU line HIGH in the early stage, the designer can verify the time required for the ALU to perform/execute various operational functionalities (opcodes) implemented. The Enable ALU Dummy signal is designed for the Logic Analyzer interface. This signal gives the synchronous timing details of the output variation of the 32-Bit ALU, i.e., how much time is required by the ALU to perform the desired operation (specified by the OpCode) after the Enable ALU signal is asserted. All of this information can be verified using a Logic Analyzer.

The ALU Output Controller is merely a multiplexer. Depending upon the Output Select signal, it selects the lower or upper 32-Bit part of the 64-Bit ALU Output. In this design, if the Output Select is HIGH then the upper 32-Bit part of the 64-Bit

ALU Output is selected, else if the Output Select is LOW then the lower 32-Bit part of the 64-Bit ALU Output is selected.

The above two figures (Figure 2-2 and Figure 2-3) show how the performance of the ALU can be verified at discrete levels as only a single 32-Bit part of the 64-Bit ALU output can be observed at a time using this scheme. Now the ALU performance for the entire 64-Bit can be verified using the following method using the same scheme:

Figure 2-4 Single System Approach: Digital Test Pattern Design with All Outputs selected



As shown in the Figure 2-4 above, the digital test pattern comprises of four cycles. Each cycle is a complete opcode instruction to the ALU. Let's assume the case of multiplication instruction.

In the first cycle, the data (A and B) and the OpCode for the multiplication instruction is given on the bus with LSB output enabled. Now as soon as the Enable ALU signal is asserted HIGH, the ALU outputs the result of the multiplication operation. The result is obtained on the output bus as long as Enable ALU signal is kept HIGH. As soon as the Enable ALU signal is asserted LOW, the output bus is tri-stated. The designer can verify the timing required for the ALU outputs to change with the changes in the inputs. Now to observe the MSB output for the same instruction the following scheme is used:

The second cycle is a dummy cycle, in which 00 data is floated onto the A, B and OpCode buses. The purpose of this dummy cycle is mentioned below.

In the third cycle, the data (A and B) and the OpCode for the multiplication instruction is given on the bus with MSB output enabled. The data is the same as in cycle 1. Now as soon as the Enable ALU signal is asserted HIGH, the ALU outputs

the result of the multiplication operation. The result is obtained on the output bus as long as Enable ALU signal is kept HIGH. As soon as the Enable ALU signal is asserted LOW, the output bus is tri-stated. The designer can verify the timing required for the ALU outputs to change with the changes in the inputs.

The fourth cycle is again a dummy cycle, in which 00 data is floated onto the A, B and OpCode buses. The purpose of this dummy cycle is mentioned below.

The cycle 1 and cycle 3 can be generated subsequently. But the purpose of introducing an intermediate dummy cycle is to change the data and opcodes so that the timings for the LSB outputs and the MSB outputs can be observed correctly. If only cycle 1 and cycle 3 are used subsequently, then the data and opcodes being the same for cycle 1, there will be no change in the output of cycle 3. Hence the designer won't be able to verify how much time is taken by the ALU to perform the multiplication operation. In this case the time, required for the MSB to change, won't be verified accurately.

Thus by using this scheme, the designer can verify the MSB and LSB outputs using a single test system comprising of the system under test and the supporting systems.

2.4 Design of the Test Suite for debugging

Now we are ready for our system design for testing. As shown in the Figure 2-1 above, integrate all the modules – 32-Bit ALU, ALU Input Controller and ALU Output Controller into a single system and synthesize the design. We are using Quartus II v4.1 for generating this system. Figure 2-5 below shows the generated system. In this design all the Inputs/Outputs are diverted on the headers for the Logic Analyzer and Digital Pattern Generator interface except for the clock and reset, which are obtained from the SLS ESDK Board.

Now follow the steps:

1. Download the generated *.sof or *.pof file into the SLS ESDK Board to configure the FPGA with the designed system
2. Connect the SLS Digital Pattern Generator's channels outputs to the inputs of the designed system

3. Connect the SLS Logic Analyzer's channels inputs to the outputs of the designed system
4. Design a Digital test pattern in the SLS Digital Pattern Generator GUI as shown in the Figure 2-6, Figure 2-7, Figure 2-8 below
5. Generate the test pattern from the Digital Pattern Generator and capture the output data using the SLS Logic Analyzer
6. Analyze the captured data as shown in the Figure 2-6, Figure 2-7, Figure 2-8 below and verify the performance of the 32-Bit ALU

Figure 2-5 Single System Approach: System Designed in Quartus II

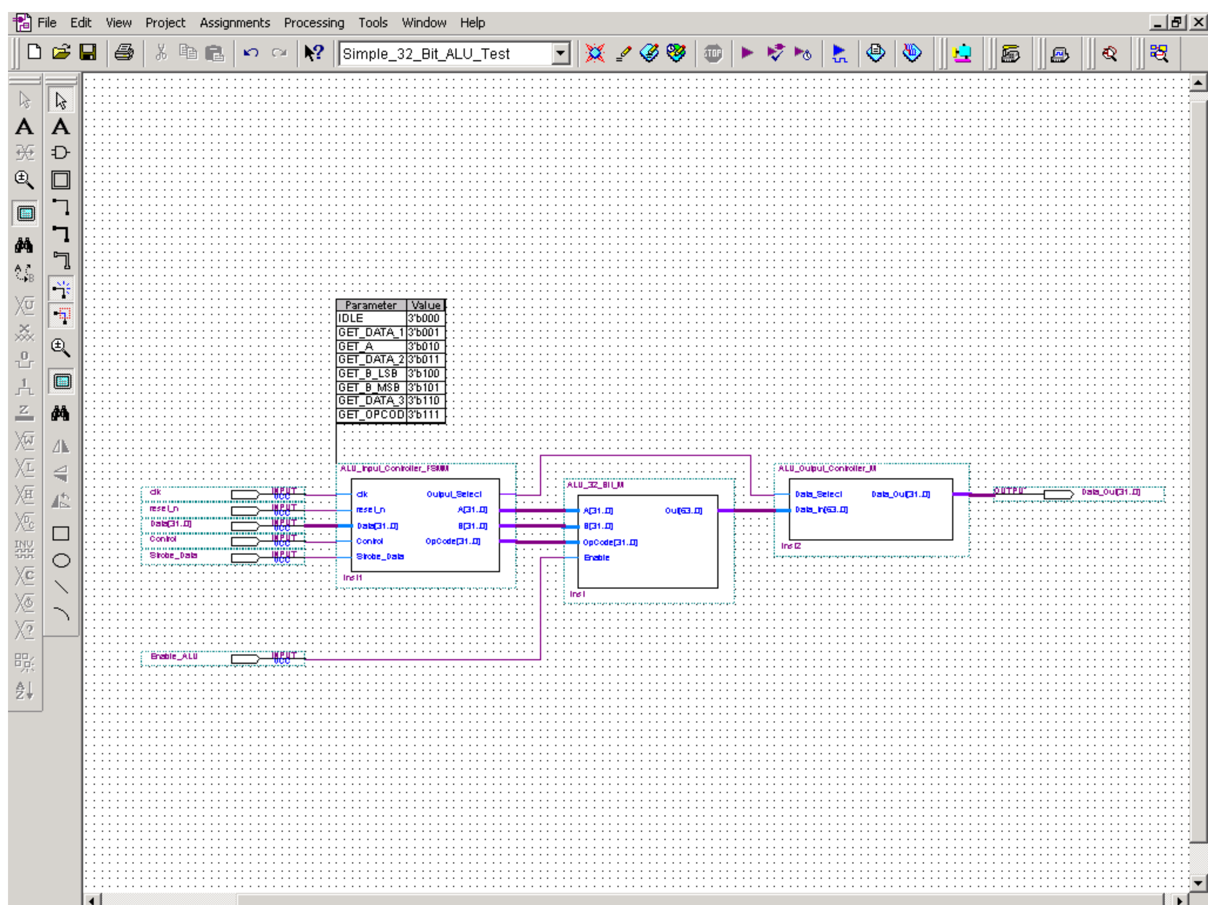


Figure 2-5 above shows the block diagram file of the top system module generated in the Quartus II software. The top system module contains the system under test – the 32-Bit ALU and the two supporting systems – ALU Input Controller and the ALU Output Controller. This design is synthesized in the Quartus II software.

Figure 2-6 Single System Approach: Generating Test Pattern using SLS DigiPAT and capturing the test results using SLS CDLogic for LSB outputs

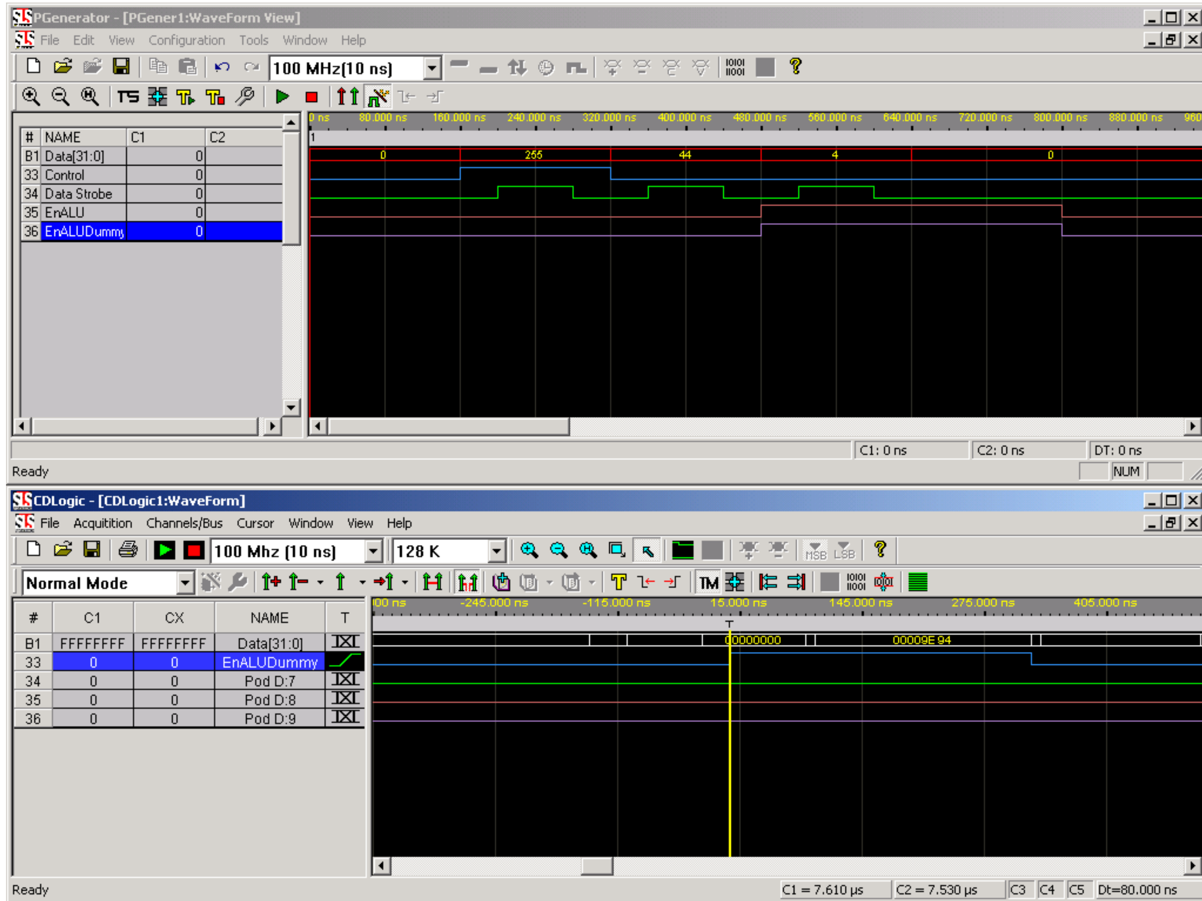


Figure 2-6 above shows the test pattern generated for LSB outputs for the 32-Bit ALU using SLS Digital Pattern Generator (DigiPAT) and the captured output using SLS Logic Analyzer (CDLogic).

Figure 2-7 Single System Approach: Generating Test Pattern using SLS DigiPAT and capturing the test results using SLS CDLogic for MSB outputs

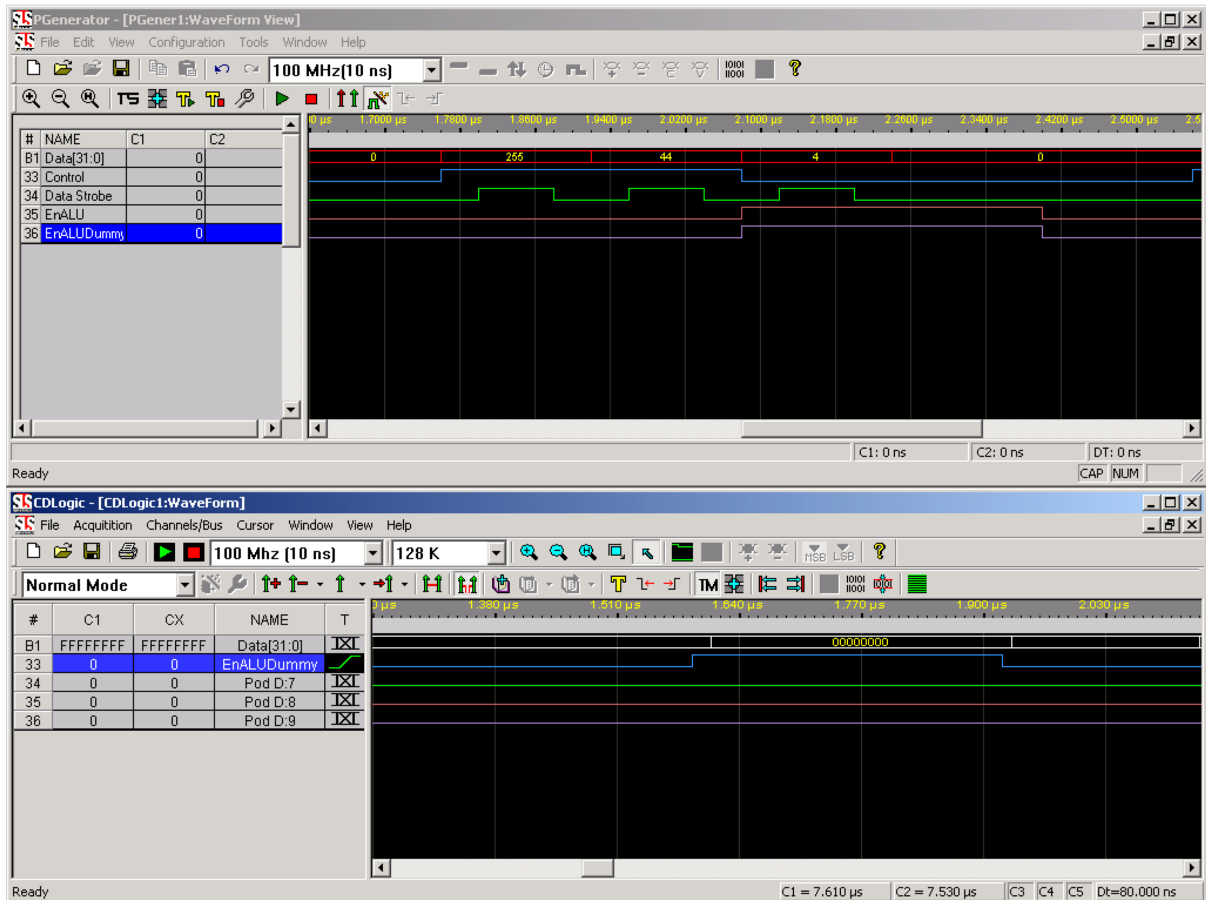


Figure 2-7 above shows the combined test pattern generated for MSB outputs for the 32-Bit ALU using SLS Digital Pattern Generator (DigiPAT) and the captured output using SLS Logic Analyzer (CDLogic).

Figure 2-8 Single System Approach: Generating Test Pattern using SLS DigiPAT and capturing the test results using SLS CDLogic for All outputs (LSB & MSB)

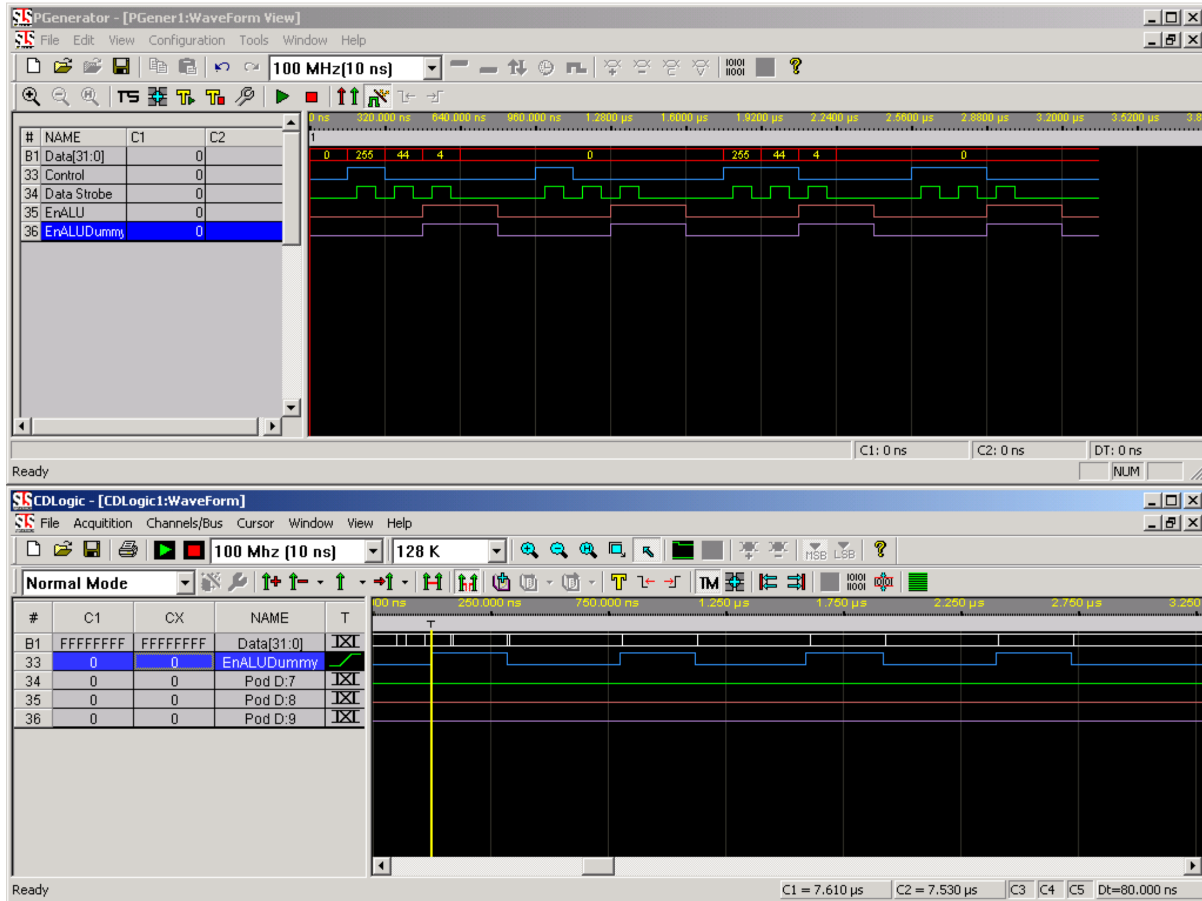


Figure 2-8 above shows the combined test pattern generated (for LSB outputs and MSB outputs) for the 32-Bit ALU using SLS Digital Pattern Generator (DigiPAT) and the captured output using SLS Logic Analyzer (CDLogic). The above Figure 2-8 shows the use of single system approach for the verification of the functionality of entire system under test using single test system.

2.5 Pros and Cons of the Single System Approach

As mentioned in the concept of the single system approach, it is the easiest way to verify the functional performance of the system under test. The main drawback of using this scheme is the output delay introduced due to the Back End Supporting System. The Back End Supporting System itself introduces some delay of its own to the output of the system under test. Hence the data, captured by the Logic Analyzer, is not purely delayed by the output of the system under test but the output is delayed by the output delay of the system under test plus the output delay of the Back End Supporting System. So user must account for the delay introduced by the Back End Supporting System even if it is a simple multiplexer design. But the delay introduced by the Back End Supporting System is fixed for all the testing carried out hence it can be viewed as a fixed output delay offset introduced to the output of the system under test. That is, considering the case of 32-Bit ALU, the output delay introduced due to the ALU Output Controller (Back End Supporting System) is fixed for all the combinations of all inputs and OpCodes. The output will be delayed by the fixed delay introduced by the ALU Output Controller plus the variable delay introduced due to various operational functions performed by the 32-Bit ALU. Only with this negative aspect, this scheme gives a very useful mean to serialize all the parallel outputs of the system under test and to verify the performance of the system under test.

3 Two-System Approach

In this section, the second scheme of the data serialization is discussed. This method uses two-system approach, i.e., generically speaking more than one system is required to debug the system under test using test instruments. The number of systems required depends upon the number of outputs supported by system under test (actual outputs) and the number of outputs desired to be tested/debugged (limited by the number of channels supported by the test instruments being used). This section explains the scheme considering the case of testing 32-Bit ALU using 36-Channel Logic Analyzer and 36-Channel Digital Pattern Generator. In this example, the ALU output is 64-Bit and the maximum output desired is 36-Bit (the limitation put by the test instruments being used). Hence two-systems are used to split the 64-Bit output Bus into two buses of 32-Bit outputs each.

3.1 The Concept

This is another method of data serialization. In this method the input is serialized in the same manner as in Single System Approach. The only difference is that there is no Back End Supporting System used for the output serialization. Only that many number of outputs are obtained as many are supported by the debugging instruments, thus giving rise to the requirement of having more than one systems for testing - each having distinctive set of outputs to be tested. Hence the name is given as two-system approach.

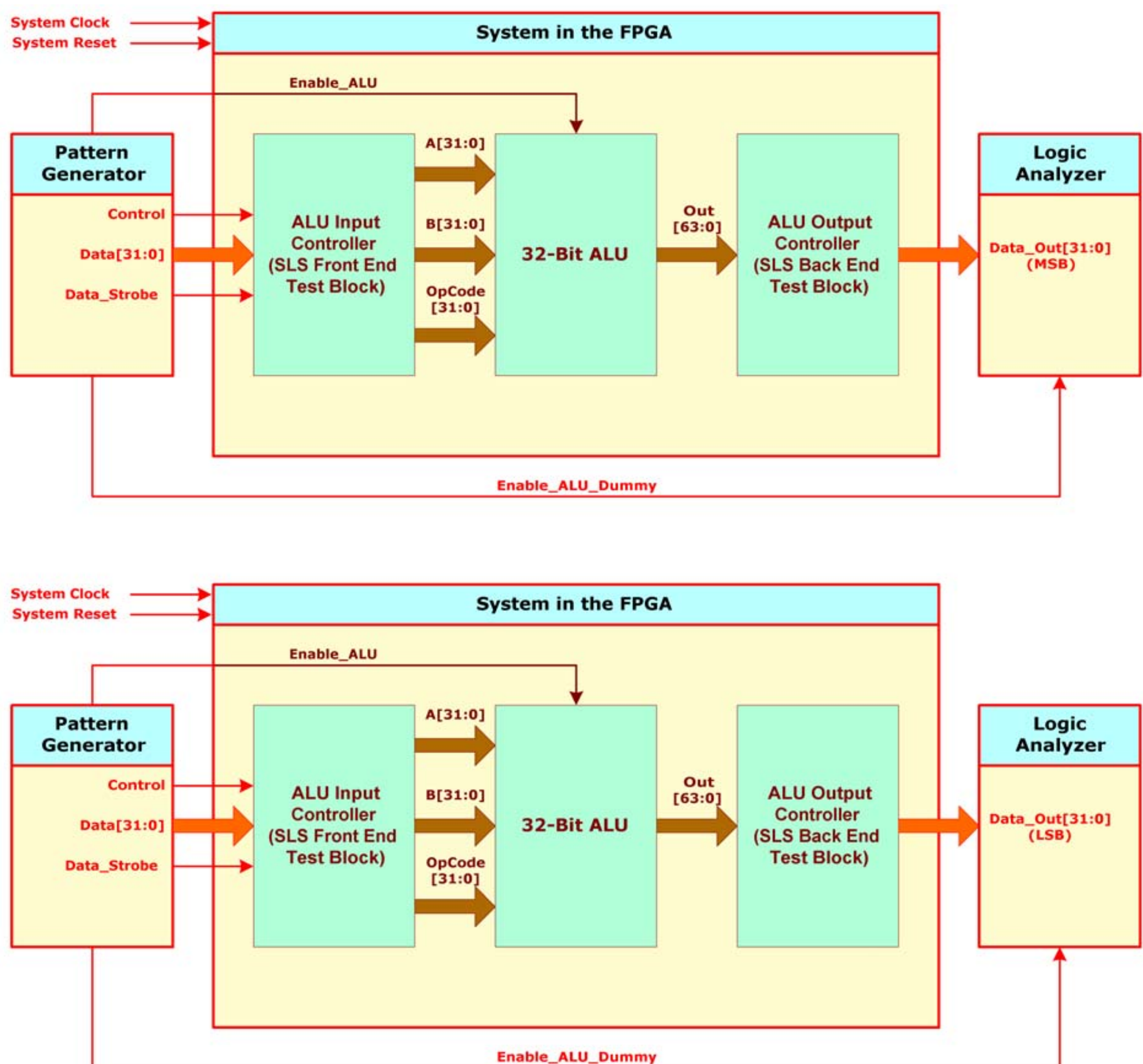
In this method all the inputs are divided into a number of bundles of signals as in the single system approach. Each bundle contains a collection of the signals. Now these bundles are sent in a fixed predefined manner from the Digital Pattern Generator, which are decoded by the Front End Supporting System, which manages the inputs. The Front End Supporting System de-serializes the input bundles from the Digital Pattern Generator in the exact predefined manner giving the exact expected parallel inputs to the system under test. Only the required number of outputs is obtained from the system under test (by making use of Back End Supporting System conceptually).

To make this concept clearer, consider the case of testing a 32-Bit ALU using a 36-Channel Logic Analyzer and a 36-channel Digital Pattern Generator.

3.2 Case of 32-Bit ALU Testing

Assume that the designer wants to test a 32-Bit ALU using a 36-Channel Logic Analyzer and a 36-Channel Digital Pattern Generator. To assist this testing the following test set up can be done as shown in the Figure 3-1.

Figure 3-1 Two-System Approach



As in the previous case, assume that the ALU has three 32-Bit inputs A, B and OpCode, one 1-Bit Enable input and 64-Bit output called Out to support multiplication operation. To support the testing of this ALU two supporting systems can be used as shown in the Figure 3-1 above. Thus the main system residing in the FPGA will contain the 32-Bit ALU, the ALU Input Controller (Front End Supporting System) and the conceptual ALU Output Controller (Back End Supporting System).

The Front End Supporting System, called ALU Input Controller, takes its inputs from the Digital Pattern Generator, de-serializes the input data stream and gives the parallel output to the 32-Bit ALU. The inputs to the ALU Input Controller system are 32-bit Data, 1-Bit Control and 1-Bit Data Strobe. The outputs of the ALU Input Controller system are the three 32-Bit inputs to the ALU: A, B and OpCode.

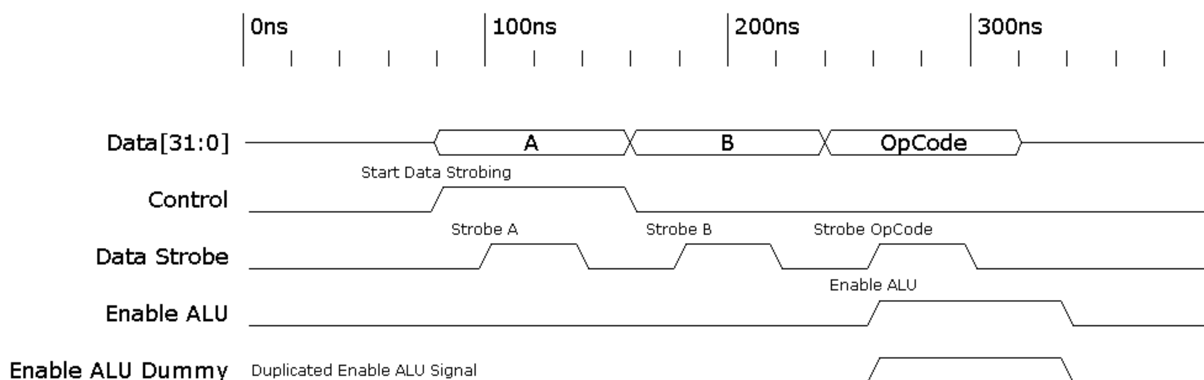
The roll of the Back End Supporting System, called ALU Output Controller, over here is merely to extract the selected 32-Bit Part of the 64-Bit ALU output conceptually, i.e., only the selected part of the ALU output is wired to the FPGA pins and routed as outputs removing the multiplexer from the design as in the single system approach. This will lead to the design of the two systems as shown in the Figure 3-1 above: one giving LSB outputs and the other giving MSB outputs.

The functioning of the supporting systems is explained in the subsequent section.

3.3 Design of the Digital Test Pattern

This section mainly describes the functioning of the supporting systems. Consider the design of the Digital Test Pattern shown in the Figure 3-2 below:

Figure 3-2 Two-System Approach: Digital Test Pattern Design



Here Data[31:0], Control and Data Strobe are the inputs to the ALU Input Controller, Enable ALU is the input to the ALU and Enable ALU Dummy is the input to the Logic Analyzer to check the synchronous behaviour of the 32-Bit ALU (i.e., to see what timing delay exists for the ALU outputs to change with the change in the ALU inputs). The above Pattern is designed for the SLS DigiPat – Digital Pattern Generator. The Digital Pattern Generator generates this pattern, which is decoded by the ALU Input Controller to generate the parallel inputs for the ALU.

The test pattern designed here is the same as in single system approach but with slight difference as mentioned below:

When a new set of instruction starts, Control signal is asserted HIGH. On the Positive Edge of the Control Signal, the ALU Input Controller gets ready to accept new data stream in the following order: A, B and OpCode. Then as soon as a (first) pulse arrives on the Data Strobe Line, whatever data is on the Data bus, is strobed into the A register of the ALU Input Controller that feeds the A input of the ALU. On the next (second) pulse on the Data Strobe Line, whatever data is on the data bus is strobed into the B register of the ALU Input Controller that feeds the B input of the ALU. Here no output selection is performed (for multiplexer) as in the case of the single system approach since only the selected part of the output is directly wired at the outputs of the system. On the next (third) pulse on the Data Strobe Line, whatever data is on the data bus is strobed into the OpCode register of the ALU Input Controller that feeds the OpCode input of the ALU. At this point the job of the ALU Input Controller is done. The ALU Input Controller now returns to its IDLE state and is triggered only when a positive edge is detected on the Control Line.

Again the ALU is assumed to be a purely combinatorial module. It operates as per the data on the A, B and OpCode lines. The Enable ALU Line merely controls the output of the ALU. If the Enable ALU line is HIGH, then the output of the ALU is routed to the output bus. If the Enable ALU line is LOW, then the output bus is tri-stated. (This is merely the design considered for this example. The designer may implement a different architecture and a different test suite for the same to suite a particular requirement.) The Enable ALU line can be asserted HIGH any time just to see the change in the output of the 32-Bit ALU. Similarly it can be asserted LOW anytime to disable the ALU outputs. By asserting Enable ALU line HIGH in the early stage, the designer can verify the time required for the ALU to perform/execute

various operational functionalities (opcodes) implemented. The Enable ALU Dummy signal is designed for the Logic Analyzer interface. This signal gives the synchronous timing details of the output variation of the 32-Bit ALU, i.e., how much time is required by the ALU to perform the desired operation (specified by the OpCode) after the Enable ALU signal is asserted. All of this information can be verified using a Logic Analyzer.

Thus by using this scheme, the designer can verify the MSB and LSB outputs using two test systems (one for LSB and the other for MSB) each comprising of the system under test and the supporting systems.

3.4 Design of the Test Suite for debugging

Now we are ready for our system design for testing. As shown in the Figure 3-1 above, integrate all the modules – 32-Bit ALU, ALU Input Controller and ALU Output Controller into a single system and synthesize two designs – one for LSB and the other for the MSB. We are using Quartus II v4.1 for generating these systems. Figure 3-3 and Figure 3-4 below shows the generated systems for LSB outputs and MSB outputs respectively. In this design all the Inputs/Outputs are diverted on the headers for the Logic Analyzer and Digital Pattern Generator interface except for the clock and reset, which are obtained from the SLS ESDK Board.

Now follow the steps:

1. Download the generated *.sof or *.pof file for the LSB outputs into the SLS ESDK Board to configure the FPGA with the designed system
2. Connect the SLS Digital Pattern Generator's channels outputs to the inputs of the designed system
3. Connect the SLS Logic Analyzer's channels inputs to the outputs of the designed system
4. Design a Digital test pattern in the SLS Digital Pattern Generator GUI as shown in the Figure 3-5 below
5. Generate the test pattern from the Digital Pattern Generator and capture the output data using the SLS Logic Analyzer
6. Analyze the captured data for LSB outputs as shown in the Figure 3-5 below and verify the performance of the 32-Bit ALU
7. Save the sample file for the LSB outputs

8. Now download the generated *.sof or *.pof file for the MSB outputs into the SLS ESDK Board to configure the FPGA with the designed system keeping the connections of the Logic Analyzer and Digital Pattern Generator as they are
9. Generate the previously designed test pattern (in step 4) from the Digital Pattern Generator and capture the output data using the SLS Logic Analyzer
10. Analyze the captured data for MSB outputs as shown in the Figure 3-6 below and verify the performance of the 32-Bit ALU
11. Save the sample file for the MSB outputs
12. Now open both the sample files – LSB outputs and MSB outputs and tile the wave windows to see the actual 64-Bit variation in the 32-Bit ALU output

Figure 3-3 Two-System Approach: System Designed in Quartus II for LSB outputs

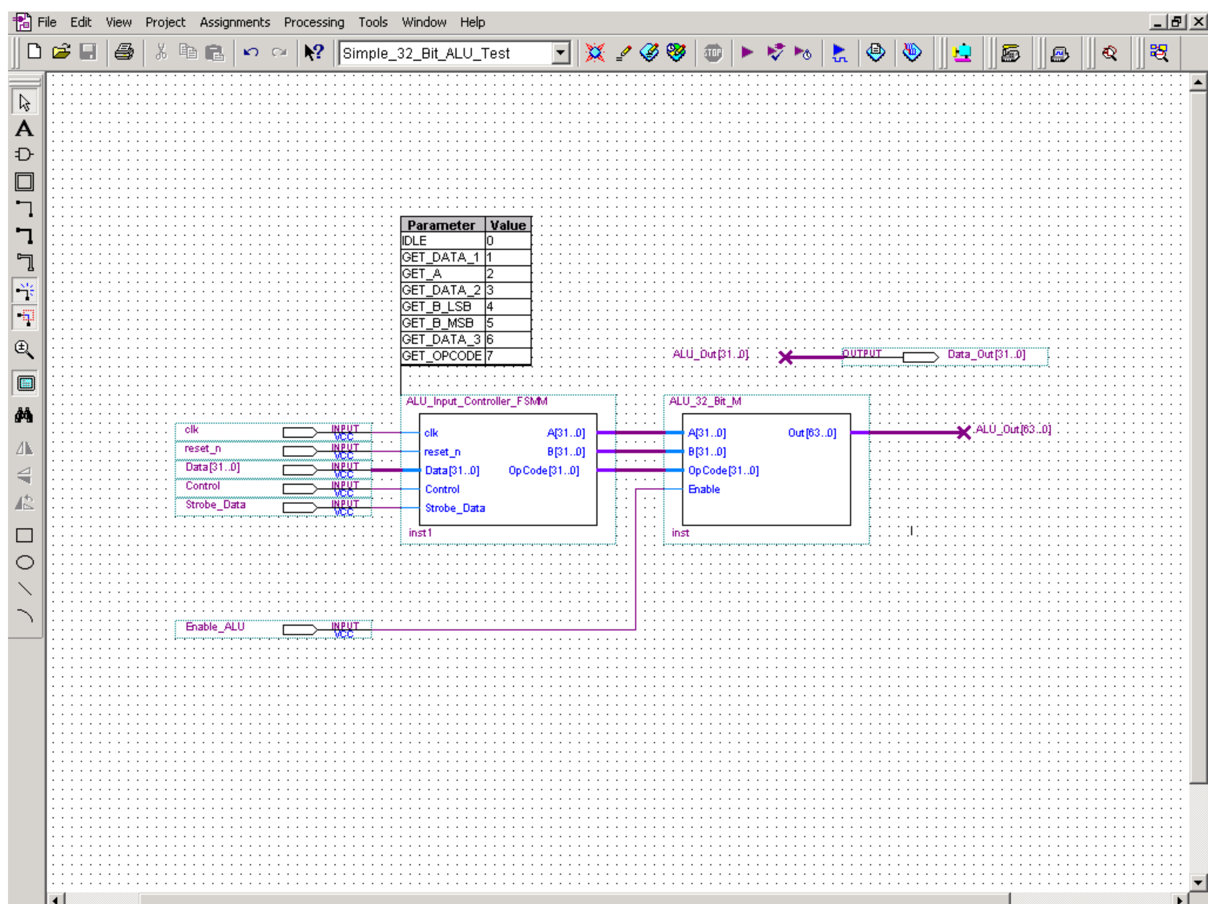


Figure 3-3 above shows the block diagram file of the top system module generated to debug the LSB outputs in the Quartus II software. The top system module contains the system under test – the 32-Bit ALU and the Front End supporting systems – ALU Input Controller. This design is synthesized in the Quartus II software.

Figure 3-4 Two-System Approach: System Designed in Quartus II for MSB outputs

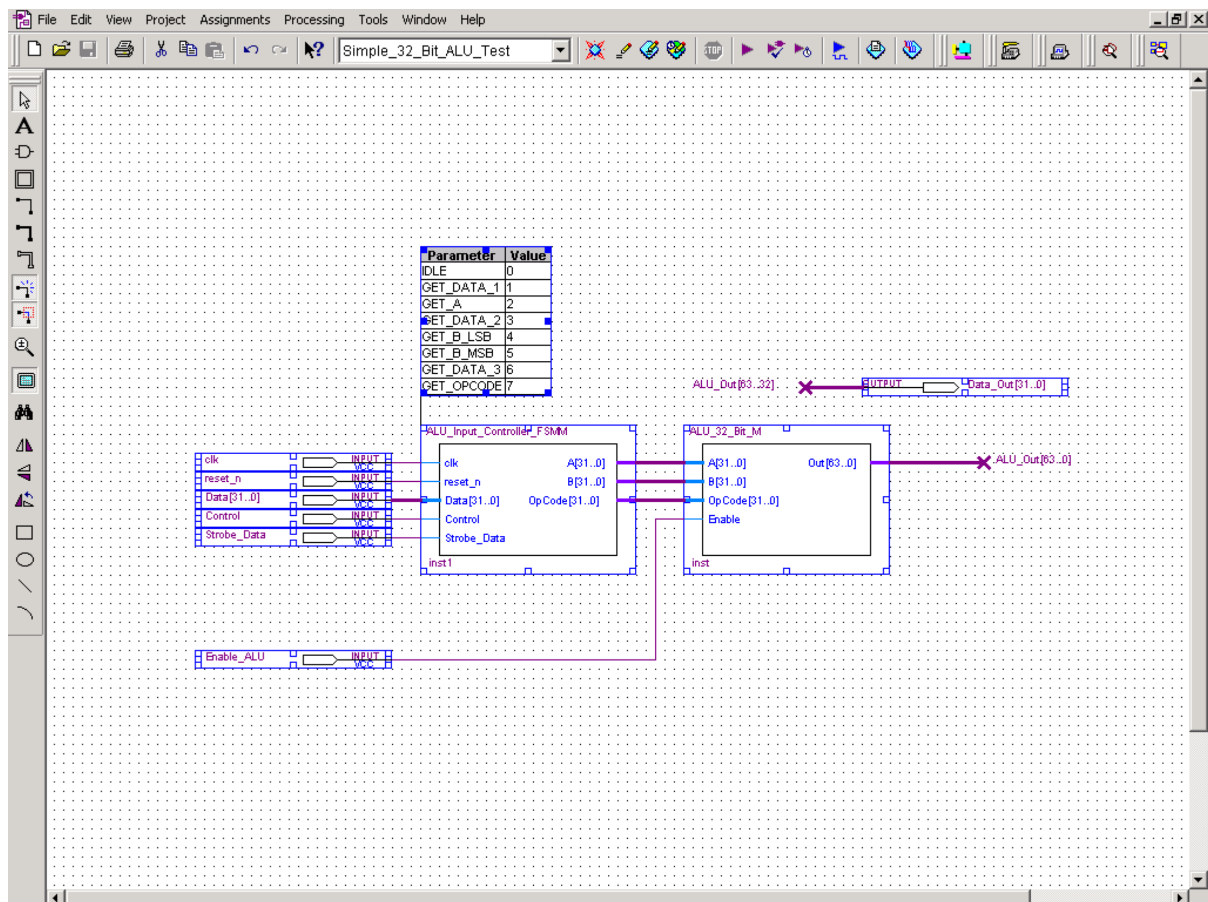


Figure 3-4 above shows the block diagram file of the top system module generated to debug the MSB outputs in the Quartus II software. The top system module contains the system under test – the 32-Bit ALU and the Front End supporting systems – ALU Input Controller. This design is synthesized in the Quartus II software.

Figure 3-5 Two-System Approach: Generating Test Pattern using SLS DigiPAT and capturing the test results using SLS CDLogic for LSB outputs

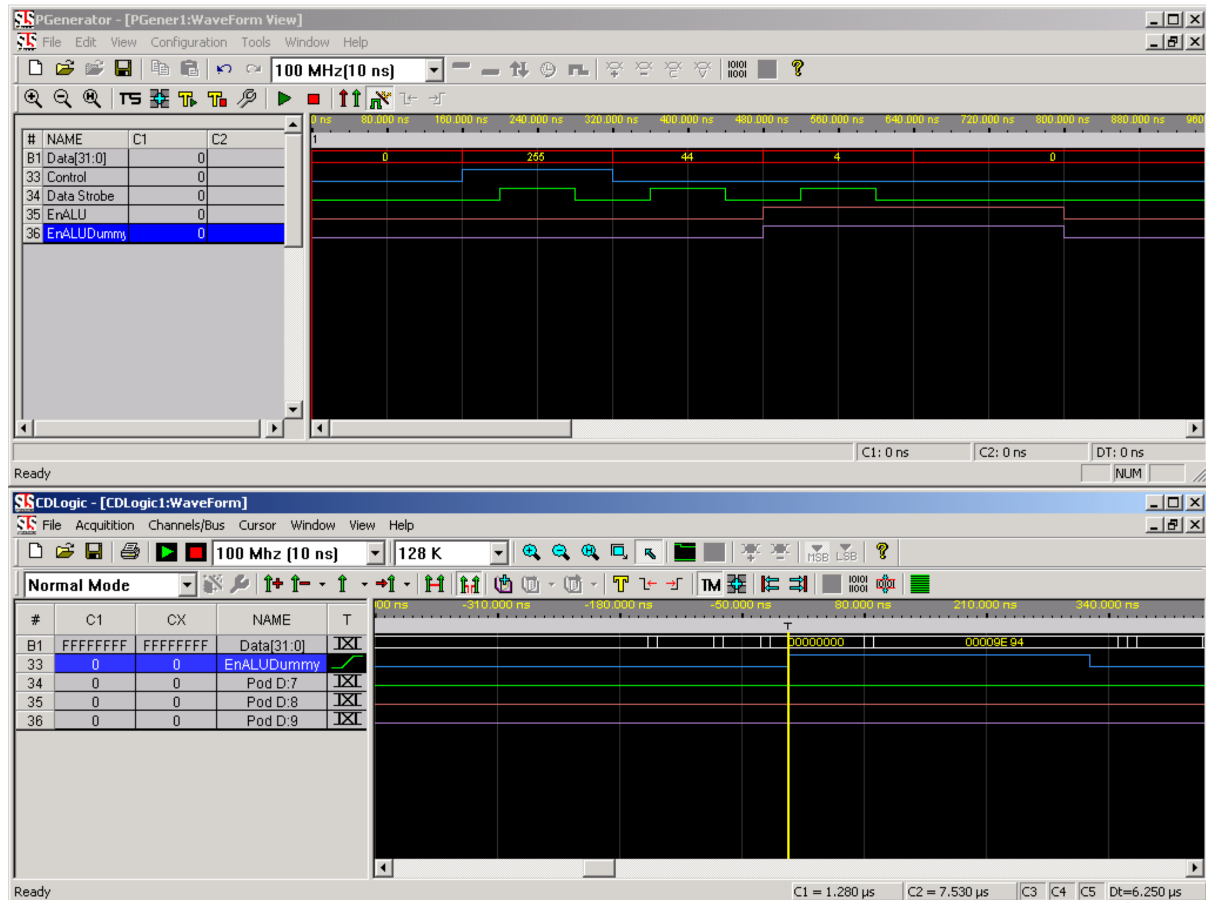


Figure 3-5 above shows the test pattern generated for LSB outputs for the 32-Bit ALU using SLS Digital Pattern Generator (DigiPAT) and the captured output using SLS Logic Analyzer (CDLogic).

Figure 3-6 Two-System Approach: Generating Test Pattern using SLS DigiPAT and capturing the test results using SLS CDLogic for MSB outputs

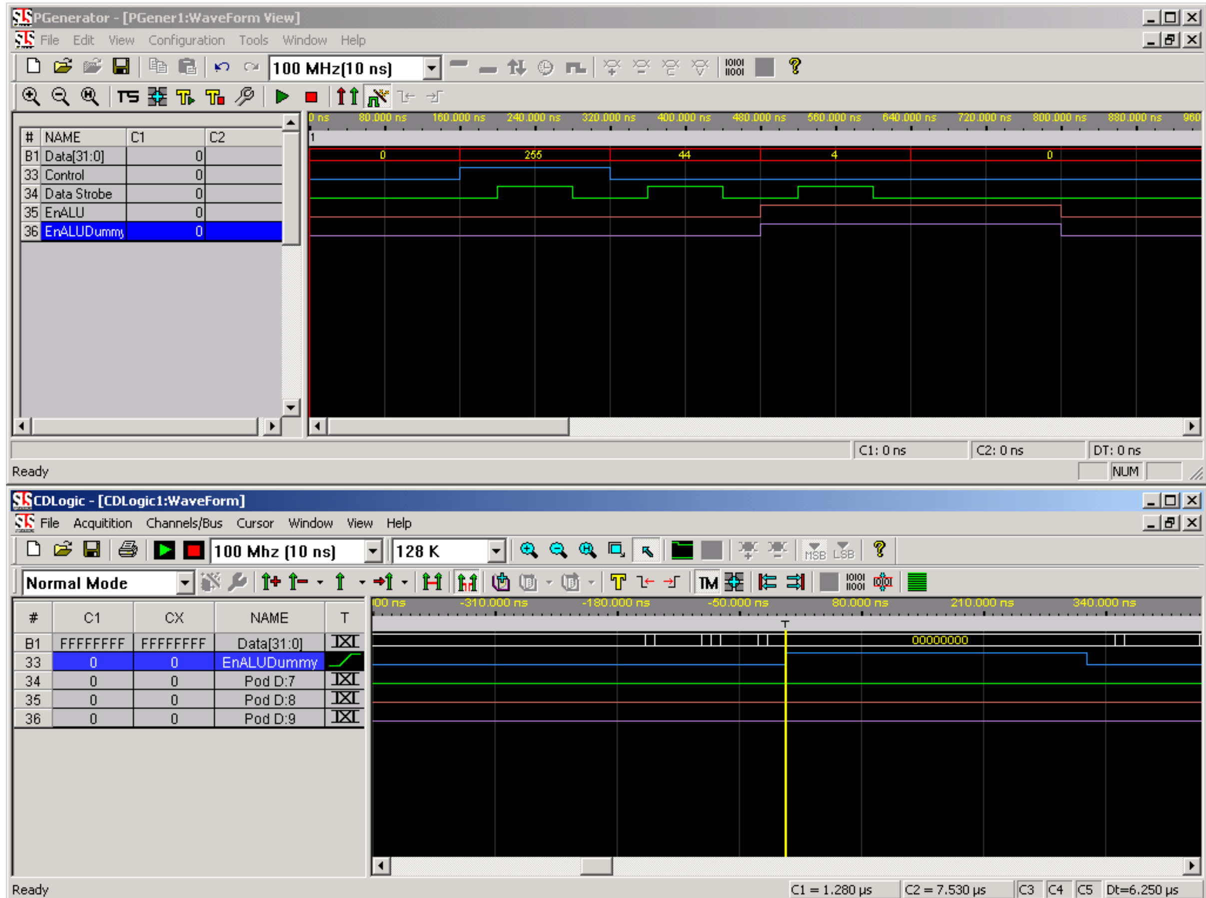


Figure 3-6 above shows the test pattern generated for MSB outputs for the 32-Bit ALU using SLS Digital Pattern Generator (DigiPAT) and the captured output using SLS Logic Analyzer (CDLogic).

3.5 Pros and Cons of the Two-System Approach

As mentioned in the concept of the two-system approach, it gives the best way to check the output performance of the system under test, as there is no Back End Supporting System introducing any delay of its own as in the single system approach. But the main drawback of using this scheme is the requirement of more than one testing systems. The designer is required to design as many systems as are required to cope with the requirement of the test suite design, i.e., in the case of testing 32-Bit ALU using 36-Channel Logic Analyzer and 36-Channel Digital Pattern Generator, the designer needs to generate two systems each having 32-Bit output part selected out of the actual 64-Bit output of the 32-Bit ALU. With the increase in the data-width of the system under test, the number of the test systems generated increases. This leads the user to download as many systems into the FPGA as are designed, capture each design's output data in different sample file of the Logic Analyzer and then compare the output results to verify the design performance. Only with this negative aspect, this scheme gives a very useful mean to verify the timing performance of the system under test as there is no additional delay is introduced in the test set up.

4 CONCLUSION

This application note gives the concept of designing a test suite for the digital design debugging and verification using tools like Logic Analyzers and Digital Pattern Generators.

The single system approach can be selected when timing is not a critical issue or the designer can find out what is the exact fixed delay offset introduced by the Back End Supporting System. In this case the single system approach will give the best results helping the designer verifying the entire system functionality using single test system design.

The two-system approach can be selected when the timing performance is of the main interest and the designer cannot afford to have any delay introduced by the supporting system. In this case the two-system approach will give the best results helping the designer verifying the actual system timing using more than one test system design.

5 REFERENCES

ESDK Reference Manual, System Level Solutions (INDIA) Pvt. Ltd.

CDLogic User Guide, System Level Solutions (INDIA) Pvt. Ltd.

DigiPat User Guide, System Level Solutions (INDIA) Pvt. Ltd.

Copyright©2004 System Level Solutions, Inc. (SLS) All rights reserved. SLS, An Embedded systems company, the stylized SLS logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of SLS in India and other countries. All other products or service names are the property of their respective holders. SLS products are protected under numerous U.S. and foreign patents and pending applications, mask working rights, and copyrights. SLS reserves the right to make changes to any products and services at any time without notice. SLS assumes no responsibility or liability arising out of the application or use of any information, products, or service described herein except as expressly agreed to in writing by SLS. SLS customers are advised to obtain the latest version of specifications before relying on any published information and before orders for products or services.

ANPG001.1