ISSN:0975-9646

Integrating Logic Analyzer Functionality into VHDL designs

[1]S.Adilakshmi [2] K.Rajasekhar, [3] T.B.K.Manoj kumar

[1]Asst.prof,E.C.E Dept,K.L.university, AP-india.
[2&3] MTech (vlsi), K.L.university, AP-India.

Abstract A combined hardware and software system for the debugging of FPGA designs is designed. It provides a powerful logic analyzer implemented as a fully parameterized VHDL description. The system can insert the analyzer into a user design without manual labor required from the user. All processing is done on the VHDL-level, facilitating vendor-independent, source-level hardware debugging. The system also allows multiple independent FPGA-systems to be debugged in a single framework. Logic signal analyzers are very essential instrument for digital circuit or board debugging. The existing market solutions offer several features, but the cost of such instruments is very high and most of the time we don't need that much capable instruments. A low cost logic signal analyzer is implemented around the Spartan-3 FPGA. The FPGA being capable of offering high frequency data paths in them become suitable for realizing high frequency signal capturing logic. It includes development FPGA based logic signal analyzer using VHDL. The logic signal analyzer will be capable of implementing match conditions, counter based triggering, external clocking and internal clocking features. The blocks such as registers, counters, comparators, state machines will be used in realizing these blocks. The captured data will be stored in memory before transferring the data to PC. The UART core will be developed which, will be used for transferring the data to PC. Model sim Xilinx edition (MXE) tools will be used for simulation. Xilinx FPGA synthesis tools will be used for synthesizing the design for Spartan FPGAs. The developed application will be tested on Spartan 3E development board.

1. INTRODUCTION

Debugging of FPGA designs is a difficult task due to the ever-increasing complexity of the chips. Consequently, silicon and CAD system vendors offer debug tools, mostly in the form of embedded logic analyzers combined with a software analysis tool. Our own research project is on high-performance computing (HPC) for graphics and visualization using reconfigurable devices. For this purpose we have built a mini-cluster of four PCs, each equipped with two FPGA boards. Each board contains one Xilinx Virtex-II 4000FPGA and a local memory of 512MB. The boards have a PCI-X-interface and are directly connected with each other via an own-developed interconnection network. It is obvious that such distributed systems pose heightened requirements on a debug tool concerning the above criteria.

Problems can include:

• A faulty FPGA with direct access to host resources can cause the host system to crash, which renders in-system debugging impossible.

• Errors can propagate through the network and lead to system failures at different places

• Reproducing erroneous situations in systems of Independent components are a difficult task.

• Certain prerequisites, such as JTAG-chain throughall devices, might be impossible to establish. Even if the system works correctly there might be the need to examine the internal circuitry very closely: for example when the performance is lower than expected, and bottlenecks or resource contentions must be eliminated. This might require complex communication protocols to be monitored and individual data packets to be tracked through the entire system. Thus, the ability to insert own-designed analyzer functions into both the hardware and software might become indispensable for project success. As an example, the designer might want to use existing external memory as trace memory for long recording times. Essentially, this requires the hard- and software to be available as source code, or at least with proper interfaces for extensions. There are quite a number of debug tools on the market. Vendor-specific tools include *ChipScope* from Xilinx, *SignalTap* from Altera, and *Reveal* from Lattice. Vendor-neutral tools include *Identify* fromSynplicity and *FPGAView* from First Silicon Solutions; the latter, however, is for use with an external logic analyzer. Our natural choice would of course be ChipScope. This tool offers a host of powerful features and integrates well with the development environment (ISE). The hardware part includes integrated logic analyzer cores (ILAs) grouped around an integrated controller core (ICON). The latter communicates via the JTAG port of the device under test with a PC running the ChipScope analyzer software. Multiple FPGAs can be debugged as long as they are in single JTAG chain. User reports about ChipScope can be found in [5] and [9].The cores are generated by the CORE generator tool, presenting the user with dialog boxes to enter the various parameters. The cores are generated as netlists, which can be inserted into the design using Xilinx's core inserter tool. While it is convenient for the user most of the time, this "black box"-approach is difficult to adapt to the specific requirements of a user design. Complex communication protocols might be difficult to be monitored. This problem is reflected in the fact that Xilinx offers special cores for certain on-chip buses (OPB, PLB), but users of other complex buses are basically on their own. As an additional problem specific to our project, it appears to be impossible to debug our eight distributed FPGA-accelerators in one single framework. These considerations led us to the conclusion that for ultimate

project success, we would need design sovereignty over the debug tools. There are only few HPC systems on the market which employ reconfigurable devices for application acceleration. Cray's XD1 system provides six Xilinx Vir-tex-4 FPGAs and 12 AMD Opteron CPUs per chassis, of which up to 12 can be combined in one cabinet [3].CPUs and FPGAs are connected directly with each other through an interconnect fabric called Rapid Array. However, this system has apparently been discontinued, and so only few reports about its usage can be found. In [12] we can find that ChipScope is used for debugging via special cables; whether this applies to multiple FPGAs is not reported. The RASC RC100 Blade from SGI provides two Xilinx Virtex-4 FPGAs, and a globally shared memory of 80MB QDR SRAM [10]. The interconnection is provided by the SGI NUMAlink technology. For debugging SGI offers an extended version of the GNU debugger GDB called *gdbfpga*. It allows the FPGA design to be executed in single-step mode, and internal values which have been mapped to special debug ports can be read by the debugger after each step.In this paper we present our own approach to multi-host parallel FPGA debugging: the *InSight* system. InSight is combined hardware and software system which is able to debug benchmark and control up to eight independent FPGA-systems in a single framework. The hardware part is an on-chip logic analyzer implemented as a parameterized VHDL description. The software part is a graphical analysis tool running on a remote PC. All processing and user interaction is done on the VHDL level that means, the users can select signals to be monitored from their VHDL source code, and the InSight system will create a combined VHDL description which includes the user design and the analyzer functionality. No manual labor is required; processing includes passing the signals through all entity ports up the hierarchy, inserting all modified component declarations and instantiations, as well as inserting, wiring and configuring the logic analyzer modules. Debugging within a subset of multiple instances of the same module is supported, even if they have been created in a generate loop. The combined VHDL description is then ready for synthesis and place & route, and after running the user design on the FPGA the user can study the waveforms with the original signal names preserved. In the following we will explain the hardware design, and the methods and principles used for VHDL processing. An example for hardware consumption and Performance will also be given.

2. LOGIC ANALYZER ARCHITECTURE

The architecture of the logic analyzer circuitry and the host interface unit, further on collectively called *InSightCore*, is shown in Figure 1b. A number of independent Logic Analyzer Units (*LAUs*) is grouped around the host interface unit, which has a dedicated connection to the analyzer PC running the InSight software. In our case, this is a standard RS232 interface. The design of a LAU is detailed in Figure 2. Most of the circuitry is clocked by the user design clock; we use pipelining to achieve high clock rates. On the first stage, a collection of *match units* can be placed. Currently we have implemented three kinds of match units: a numerical match unit NMU, a Boolean expression unit BEU, and a transition detection unit TDU. Their design is sketched out in Figure 3.The second stage provides trigger level units and a storage qualifier unit. The third stage finally contains the trigger position counter and the trace memory. Besides that, circuitry to latch a global time stamp counters, if existent, at trigger time is provided. The signals entering a LAU are all recorded in the trace memory; a subset or all of them are also used as trigger signals. A numerical match unit (Figure 3a) can compare any slice of the input signals to two loadable constants A and B using =, > and < operators. The result flags are combined in any desired way using an 8x1 bit loadable memory. Likewise, Boolean expressions of up to 6 variables can be evaluated using a 64x1 bit memory per Boolean expression unit (Figure 3b). The same Design principle applies to the transition detection units (Figure 3c). The system currently supports 8 match units of each kind, for a total of 24 match units, per LAU.A level unit (Figure 3d) can combine the outputs of up to five match units using a 32!1 bit memory. In case a match has occurred, an event counter is decremented. If the value has reached 0, the level unit on the next level is enabled, or, if level 0 has been satisfied, the trigger position counter is enabled. All input signals have been continuously recorded in the trace memory; if the trigger position counters expires, recording stops, thus implementing variable trigger offset. Then the host interface unit is notified. This in turn sends the data to the analyzer PC, where the user can graphically inspect the data using the analyzer functions of the InSight software. All of the above memories are controlled by the InSight software, so that the exact operation of these units can be changed on the fly without affecting the user logic. The presented hardware units are very compact and fast, but still allow versatile match functions, and complex trigger sequences on up to four levels to be used. It should be emphasized again, however, that any kind of specialized match unit can be implemented if the need arises, as opposed to commercially available (closed source) tools. As can be seen in Figure 2, the user design signals are loaded with just one register at the input of the corresponding LAU, thus minimizing the effects on the user logic.

3. INTEGRATING THE INSIGHT CORE INTO A USER DESIGN

A debug session typically includes the following steps:

- selection of the user design signals to be monitored,
- definition of the trigger conditions and sequences,
- creation of a combined VHDL description for both the user design and the InSight core, ready for synthesis and place route
- configuration of the FPGAs,
- setup of all LAUs with the proper parameters.

Typically a complete VHDL design includes a top level module which instantiates several components. These might be divided further, resulting in a potentially very deep hierarchy of modules. The InSight core is to be placed in the top-level module as one additional component.Thus, if signals of a module deep down in

the hierarchy need to be connected to a LAU, they must be propagated up the entire hierarchy through all entity ports of the traversed modules. Then, potentially large numberof modules have to be modified. Multiple instances of a given module to be debugged represent a further problem, since in this case several different versions must be created. Moreover, many of those instantiations are the result of potentially nested generate- loops, which must be handled correctly. An additional issue is specific to the design environment. Timing constraints, e.g. for multi-cycle paths (so called FROM-TO-constraints), are specified in a separate *synthesis constraints file* (.xcf-file in the Xilinx framework) In a modular design there might be several such constraints files, which must be consolidated into one single file for the combined design, and all net and instance labels must be extended according to their path through the hierarchy.

3.1 The Instantiation Tree

Starting from the top-level module, a tree structure is constructed which includes one node for each component *instantiation*. For this purpose, all design files (VHDL modules, VHDL packages, and synthesis constraints files) are parsed.. In this way, each instance is represented by its own node even if it was replicated in a generate loop. At the same time the parsed constraints files are inserted into the tree. For every instantiation of an entity having constraints file, the net or instance labels along with their constraints are copied into the node data. Once the tree is built, the user can select the signals to be monitored from each individual instance (whether replicated or not) and assign them to one of the logic analyzer units. This is done via the GUI of the InSight software. Each LAU can only receive signals from a single clock domain, and that clock must be connected to its clock input. When the user has finished signal selection and assignment, the instantiation tree is modified bottom- up.If an instance at a leaf node does not have signals to be monitored, the process steps to the parent node and examines the other children. If there are selected signals, these signals are assigned to a special debug bus, such as INS_XX(0) <= user_signal_a; INS_XX(4 downto 1) <= user_bus_b; which is written into the node data. At the same time, the parent node is updated to define ports 0 through 4 of the selected LAU as being connected. Let's assume there are two selected signals in the next child, and then its node would be modified as follows:

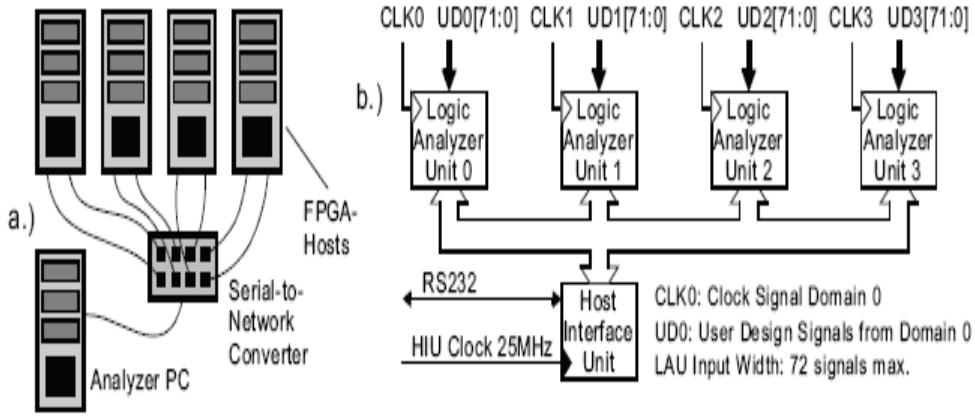

Figure 1: a.) Overall system, b.) InSight Core, block diagram.

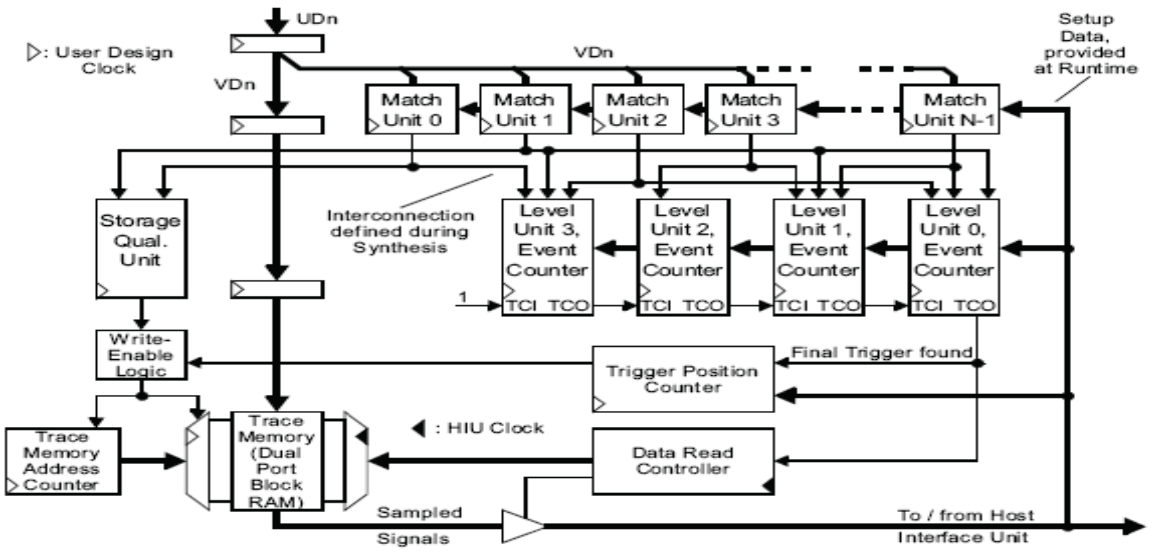

Figure 2: Block diagram of a Logic Analyzer Unit (LAU).

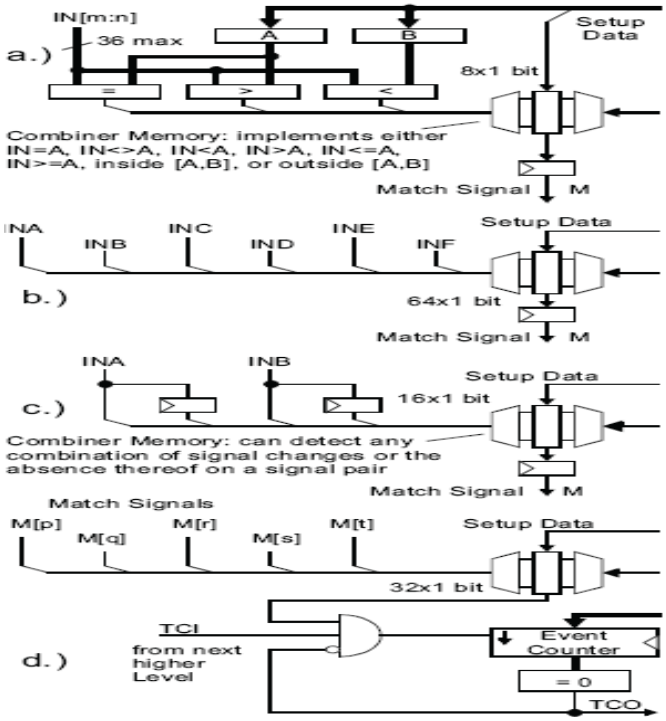

Figure 3: Match and Level Units.

INS_XX(5) <= user_signal_c;
INS_XX(6) <= user_signal_d;
After all children have been processed, the parent's own selected signals, if any, are processed:
INS_XX(7) <= user_signal_e;
INS_XX(11 downto 8) <= user_bus_f;
The parent's node would then define INS_XX(11 downto 0) as being connected. This process is recursively repeated for all nodes in the tree.

3.2 Creating a combined VHDL Description

Once the node data items have been modified, a new VHDL description is created. It has the form of one single, large VHDL file containing all modules of the user design and the InSight Core. The original user design is left unchanged. Processing is done again bottom-up in the Instantiation Tree.
If an instance at a leaf node does not have selected signals, its VHDL description is appended unmodified to the output file. The system keeps track of all modules written into the output file so that multiple instances of the same module are written only once. If a leaf instance has selected signals, its entity name is modified to be unique, the assignments in its node data (see above) are inserted into its architecture definition and its entity port (in the above example) is extended by INS_XX : out std_logic_vector(4 downto 0); In this form the VHDL description of the leaf instance is appended to the output file. Once all child nodes have been visited, the parent node is processed. If it has modified children, their extended component declarations and module instantiations are inserted. The new debug ports at their port maps are connected to the debug bus as specified in their respective node data. If the parent has selected signals itself, the assignments are inserted, e.g.,
INS_XX(7) <= user_signal_e;
INS_XX(11 downto 8) <= user_bus_f;
and its entity port is extended by
INS_XX: out std_logic_vector(11 downto 0);
In this form its new VHDL description is appended to the output file. The whole process is recursively repeated for all modules. For the top-level module, however, instead of extending the entity port list, the debug buses are declared as internal signals. Additionally, component declaration and module instantiation of the InSight Core must be done. In this modified form, the top-level VHDL description is appended to the output file. Next, the VHDL-sources of the InSight Core are written. They are appended unmodified from their original form. All configurations of the Logic Analyzer Units is done via a separate VHDL package file (see next section).Also during tree traversal, a new combined synthesis constraints file is created. Whenever the process visits a node with a constraints file attached, it appends those constraints to the combined constraints file with properly adapted label names

4. CONFIGURATION INSIGHT CORE

During synthesis, the InSight Core has to be configured according to the specifications the user has supplied. This includes:
• LAUs may be present or not,
• match units may be present or not,
• the width of each match unit,
• the members of the input bus to which each match unit is connected,
• number of level units,
• the set of match unit outputs to which each level unit is connected.
Generating VHDL descriptions from scratch for a given configuration was found to be too complicated. Instead,

the InSight Core was implemented as a fully parameterized
design, which can be copied into the combined
VHDL file in unmodified form. All configuration information is instead written into a VHDL package file containing a large number of parameters. This shall be explained with the example of the Boolean expression unit (BEU) in Figure 3b. Its up to six inputs need to be connected to any set of signals of the VDn-bus (cf. Figure 2). Assumed the system supports a maximum of four LAUs with up to eight BEUs, each with up to six inputs, a maximum of 192 inputs must be connected. Thus, the package file contains an array of 192 integers, defining for each input the element of the corresponding VD-bus to which it shall be connected. Within the VHDL-module
of a LAU, the BEUs are instantiated as shown below (simplified). In the code snippet below, the aforementioned integer array is called BEU_INPUT_INDEX and defines the connectivity. Further parameters written into the package file are MAX_BEU and USE_BEU, which define the LAU architecture. This principle is applied in the same way to the other match units as well as the level units, and so the LAUs can be comprehensively configured
by the InSight software.

5. SYNTHESIS AND PLACE & ROUTE

With the generation of the combined VHDL file, the InSight package file, and the combined constraints file, the debug version of the design is ready for synthesis and place & route. As we have outlined, required user intervention in order to get from a (modular) VHDL design to the debug version has been kept to a minimum, with comprehensive GUI support. In our project, we use FPGA designs of fairly high frequencies (given the relatively old FPGA chips), ranging from 100MHz (PCI-X-interface) to 160MHz (onchip bus). So far, insertion of InSight Core did not cause problems for PAR to meet timing requirements. Hardware resources consumed by the InSight core depend of course on the number of LAUs and their configuration. An example is given in Table 1. A total of four LAUs have been connected to four different parts (clock domains) of our design. Once again, maximum clock frequencies of the original design have not been affected by the inclusion of the InSight Core.In this example, the InSight Core consumed 1637 slice flip-flops (out of 46,080, or 3.6%), 1162 LUTs (2.5%), one Digital Clock Manager (DCM), and 7 BlockRAMs (out of 120) for a trace depth of 512.

Table 1: Hardware Configuration

	LAU0	LAU1	LAU2	LAU3
Clock	100MHz	160MHz	125MHz	125MHz
Width	43bits	69bits	66bits	35bits
NMUs	1x32bits 1x8bits	2x32bits	2x32bits	32bits
BEUs	1x3bits	1x5bits	-	1x3bits
TDUs	-	-	1x2bits	
RAMs	2	2	2	1

6. THE INSIGHT SOFTWARE SYSTEM

The InSight software is a multithreaded application with a comprehensive graphical user interface. Communication with the FPGAs is handled in separate threads to assure responsiveness and stability of the system. For a trace memory of 512!72 bits, transmission of all debug data from a LAU to the analyzer PC takes roughly 2.4s at38.4kbaud. Much design effort was spent to provide high-performance graphics output. Since trigger events can be widely separated in time on the different FPGAs, waveforms tend to be very large along the time axis. Likewise,the high number of signals extends the height of the display as well. Thus, our analyzer PC drives a total of four displays; still smooth scrolling of very large bitmaps is possible without noticeable flicker. A screenshot of the waveform window is shown in Figure 4, displaying a collection of signals from two FPGAs. Means for controlling the FPGAs from the InSight software have also been implemented. All parts of the InSight Core can be reset by means of a software command (a working RS232 interface provided), and the FPGAs can be forced into a complete reconfiguration.

7. CONCLUSIONS

We have presented the InSight system, a powerful tool for debugging parallel, independent FPGA-systems. It offers comprehensive trigger functions and can process elaborate trigger sequences. Its most important feature, however, is the autonomous processing of VHDL designs without work required from the user. This has been achieved by means of an instantiation tree structure, which facilitates signal propagation through a VHDL design hierarchy, and a fully parameterized design of the logic analyzer hardware.

8. REFERENCES

[1] M. A. Aguirre, J. Tombs, A. Torralba, L. G. Franquelo, "*UNSHADES-1: An advanced tool for In-System Run- Time Hardware Debugging*", LNCS, Proc. FPL 2003

[2] Altera Corporation, "*SignalTap II Embedded Logic Analyzer Documentation*", May 2008, www.altera.com/literature/hb/qts/qts_qii53009.pdf

[3] CRAY Inc., "*Cray XD1 Datasheet*", www.cray.com/downloads/Cray_XD1_Datasheet.pdf

[4] First Silicon Solutions, "*FPGAView Software*", http:// www.fs2.com/fpgaview.html

[5] P. S. Graham, "*Logical Hardware Debuggers for FPGA-Based Systems*", PhD Dissertation, Bringham Young University, 2001

[6] K. Klues, K. Gyang, J. Helmes, "*Networked On-chipLogic Analyzer (NOLA) for Real-Time Debugging ofHardware Circuits on the FPX Platform*", 2004,userfs.cec.wustl.edu

[7] S. Lass, "*Improving Time to Design Closure with ISESoftware*", Xcell Journal, Fourth Quarter 2005, page 6

[8] Lattice Semiconductor Corporation, "*Reveal: A NewSolution to an Old Problem*", http://www.latticesemi.com/corporate/newscenter/newsletters/newsjune2007/revealdesigntool.cfm

[9] O. Oltu, P. L. Milea, A. Simion, "*Testing of digital circuitry using Xilinx chipscope logic analyzer*", Proceedings International Semiconductor Conference CAS2005, pages 471 - 474

[10] SGI, "*SGI RASC RC100 Blade Datasheet*", 2008,www.sgi.com/pdfs/3920.pdf

[11] Synplicity, "*Identify*", www.synplicity.com/products/identify/index.html